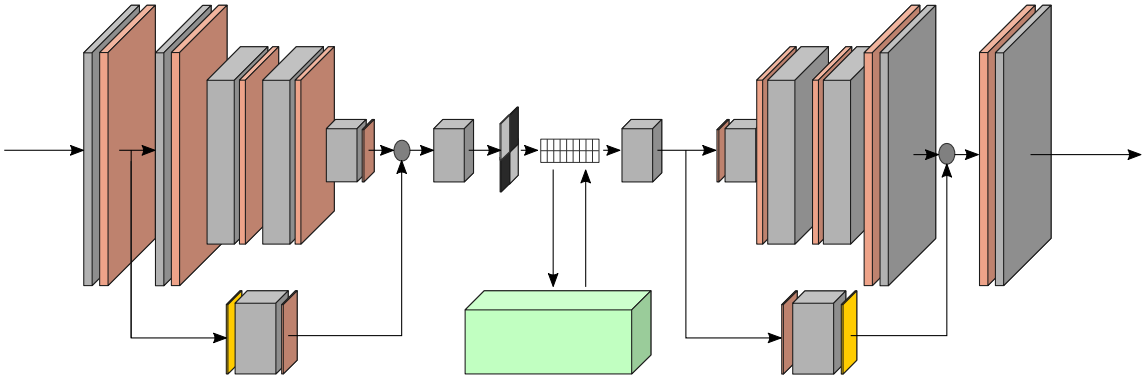
TÉCNICO LISBOA



Video Compression Using (End-to-End) Deep Learning

Jorge Guilherme Pessoa dos Santos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Doutor Pedro Filipe Zeferino Aidos Tomás,
Doutora Helena Isabel Aidos Lopes Tomás

Examination Committee

Chairperson: Doutor José Luís Brinquete Borbinha
Supervisor: Doutor Pedro Filipe Zeferino Aidos Tomás
Members of the Committee: Doutor Arlindo Manuel Limede de Oliveira

October 2018

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

Acknowledgments

Before embarking on this journey, I always thought of the acknowledgments as a formality which had become an academic tradition. But, nearing the end of the thesis, it has become clear for me that I could not finish it without some words to the people who played an essential role in it. Thus, I would like to express my sincere gratitude to everyone who helped me in this process. In no particular order, I would like to thank:

My supervisors, Prof. Pedro Tomás and Prof. Helena Aidos. From the first day, they guaranteed that they would do everything in their reach to challenge me, and ensure I would work to the best of my abilities. But they went way beyond anything I expected: the dedication, the support, the availability, their knowledge, the constructive criticism. They did everything, and much more, that I could ever ask for.

Prof. Mário Figueiredo. For accepting the challenge of helping in this thesis with so many things going on. For the invaluable knowledge and guidance brought into the work; for the availability and the help. I honestly doubt any other professor could have been a better addition.

My parents for fully supporting and providing me with the opportunity to be here. And above all, because everything I am and accomplish would not be possible without them. My uncles, for supporting me as if I was their own. My girlfriend, for the unrelenting support, and for lifting me up in the more difficult times. For not letting me delay for tomorrow what I was supposed to do today.

My close friends. For always being present; for being a source of distraction, conversation, advice and support. For upholding the necessary balance of work and leisure during this journey.

Finally, I would like to thank the funding provided by the projects PTDC/EEI-HAC/30485/2017 and UID/CEC/50021/2013 which helped in the development of the thesis.

Abstract

Deep learning (DL) is having a revolutionary impact in image processing, with DL-based approaches now holding the state of the art in many tasks, including image compression. However, video compression has so far resisted the DL revolution, with scarce published research. An initial investigation in DL-based image compression was done, due to the lack of available research in video compression, and four landmark architectures were implemented. By building upon this preliminary work, this dissertation proposes what the author believes to be the first approach to end-to-end learning of a single network for video compression. The problem is tackled in a novel way, avoiding explicit motion estimation/prediction, by formalizing it as the rate-distortion optimization of a single spatio-temporal autoencoder, *i.e.*, by jointly learning a latent-space projection transform and a synthesis transform for low-bitrate video compression. The quantizer uses a rounding scheme, which is relaxed during training, and an entropy estimation technique to enforce an information bottleneck, inspired by recent advances in image compression. The obtained video compression network is compared with standard widely-used codecs, and a naïve frame-by-frame compression baseline. It shows better performance than the baseline and the MPEG-4 Part 2 codec, being competitive with H.264/AVC and H.265/HEVC for low bitrates.

Keywords

Deep learning, image compression, video compression, rate-distortion optimization, convolutional autoencoders, end-to-end learning

Resumo

A aprendizagem profunda (AP, *deep learning* na literatura de língua inglesa) está a ter um impacto revolucionário no processamento de imagem, com as abordagens baseadas em AP sendo hoje consideradas o estado da arte em muitas tarefas, incluindo a compressão de imagens. No entanto, a compressão de vídeo tem resistido, até agora, à revolução da AP, com muito poucos resultados publicados. Foi levada a cabo uma investigação inicial sobre compressão de imagens baseada em AP, devido à falta de resultados de investigação disponíveis em compressão de vídeo, e quatro arquiteturas de referência foram implementadas. Usando este trabalho como ponto de partida, esta tese propõe o que o autor acredita ser a primeira abordagem para aprendizagem extremo-a-extremo (*end-to-end*) de uma rede única para compressão de vídeo. O problema é abordado de uma nova forma, evitando estimativas/previsões explícitas de movimento, formalizando-o como problema de otimização ritmo-distorção (*rate-distortion*) de um auto-codificador (*autoencoder*) espacio-temporal, isto é, aprendendo em conjunto uma transformação de projeção sobre um espaço latente e uma transformação de síntese para compressão de vídeo de baixo débito binário. O quantizador usa um esquema de arredondamento, que é relaxado durante o treino da arquitetura, e uma técnica de estimação de entropia para impor um limite na informação utilizada para compressão, inspirado por avanços recentes em compressão de imagens. A rede proposta para compressão de vídeo é comparada com os codecs padrão amplamente utilizados e uma base de referência usando compressão trama a trama. Os resultados da rede mostram melhor desempenho do que a base de referência e o codec *MPEG-4 Part 2*, sendo competitivo com *H.264/AVC* e *H.265/HEVC* para débitos binários baixos.

Palavras Chave

Aprendizagem profunda, compressão de imagem, compressão de video, otimizão ritmo-distorção, auto-codificadores convolucionais, aprendizagem extremo-a-extremo

Content

List of Figures

List of Tables

Abbreviations

DL Deep Learning

DNN Deep Neural Network

ANN Artificial Neural Network

GAN Generative Adversarial Network

VAE Variational Autoencoder

RCNN Recurrent Convolutional Neural Network

RNN Recurrent Neural Network

CNN Convolutional Neural Network

3DCNN Three Dimensional Convolutional Neural Network

MS-SSIM Multi Scale Structural Similarity

PSNR Peak Signal-to-Noise Ratio

AUC Area under Curve

GRU Gated Recurrent Unit

LSTM Long Short-Term Memory Unit

3D three-dimensional

1

Introduction

Contents

1.1 Motivation

The advent of digital technology spawned new ways of sharing information, knowledge, and art on a global scale, *e.g.*, through the Internet. Image and video are two of the most prolific types of media. However, their storage and transfer in raw format were soon understood to be unsustainable due to the high cost of digital storage and bandwidth. The fast technological improvements in imaging devices lead to a higher generation rate of images and videos, as well as an increase in the resolution of such media. Consequently, the growth rate in the volume of data, from media, that needs to be handled, outpaces the advancements in the capacity to store and transfer this data. Solving this problem is particularly difficult to achieve, especially for the increasing share of mobile internet users with restricted bandwidth available. The effects of the aforementioned problem seem unavoidable considering that video traffic will make up an estimated 82% of all consumer IP traffic by 2021 [6], with prospects to continue increasing.

As a consequence, there is a growing need for more efficient (preferably patent-free) image and video compression schemes to expedite the process of sharing visual media over limited bandwidth channels. Compression techniques can be used to counteract the increasing sizes by exploiting redundancies in the stored media, to create a smaller representation from which the original can be retrieved. Due to their visual nature, images have recurring patterns with repeated information that make them **spatially redundant**. This means that an area of the image can be correlated with a similar but different area so that both can be encoded together. Only the significant differences need to be stored, decreasing the size of the final representation. Similarly, the bits used to encode the image in digital format exhibit regular patterns due to **coding redundancy**, which can be exploited by studying the underlying repetitions and their frequency in the encoded data. Both spatial and coding redundancy can be used to create compact representations, without affecting the original quality of the image, referred to as **lossless compression**. To further shrink the size of the data, it is possible to discard information while preserving most of the features important for human perception. The size of an image can be reduced to almost any size, with the consequence of increasing the impact on the visual fidelity of the reconstruction. This technique known as **lossy compression** can be understood as a trade-off between the compression ratio and the reconstruction quality of the image, usually referred to as the rate-distortion optimization problem in compression research.

Video compression is tightly coupled to image compression due to the nature of video: a sequence of images (frames) temporally aligned and played in succession to resemble motion. Most types of video compression techniques require an image compression strategy or are based on one (*e.g.*, compression of key frames in H.264/AVC [7]). In addition to exploiting spatial redundancy within each frame in a sequence, video compression techniques use the fact that those sequential frames exhibit **temporal redundancy**, as there are temporal dependencies between pixel values. If an area of one frame is correlated to a similar area in the previous or next frames of the video, the frame does not need to be fully stored for later reproduction; the redundancy can be used to create smaller representations from which the frame can be fully retrieved, similarly to what is done for spatial redundancy.

Figure 1.1: Results of the compressive autoencoder proposed by Rippel and Bourdev [1] (2017).

Video compression algorithms can also employ both lossless compression and lossy compression when a higher reduction in size is necessary.

Traditionally, image and video compression [8] efforts were focused on deterministic algorithms that use heuristics to exploit spatial and temporal redundancy, while discarding information until the desired rate-distortion ratio is achieved. For video compression, most algorithms use block-based predictive schemes along with residual compression. They make use of the advances in image coding to compress both key frames and the residuals between predicted and actual blocks. However, this type of investigation is heavily guided by hand-crafted improvements and techniques. It is thus limited by the extent of the human understanding regarding the properties and statistical dependencies in both image and video.

Deep Learning (DL) is having a revolutionary impact in the areas of image and video processing, with DL-based approaches now holding the state of the art in many related tasks [9–11]. Image compression has not been an exception as new architectures are proposed with the capability of competing with the best performing image compression algorithms [1, 4]. The tools provided by Deep Learning differ from traditional algorithms as they are not explicitly programmed for their task, and instead, they are carefully architectured and trained to do so. These developments still rely on our understanding of the problem to devise learning strategies which can optimize neural architectures to solve the problem in hand, which then have the potential to learn beyond our understanding, by continuously adjusting their parameters to best approximate the desired model.

The applications of Deep Learning for image compression have been known since the 90's [12], but the computational capacity of computing systems significantly limited its application. In the meanwhile, research in novel computers architectures has produced new processors with significantly

higher computational power (particularly massively parallel processors, such as GPUs). This allowed the adoption of more complex models and drove the application of deep learning in a variety of fields [13]. In a few years, researchers have gone from compressing small fixed size images [2] struggling to beat JPEG, to achieve results that compete and surpass the current best heuristics-based algorithms [1, 4, 14] such as JPEG2000, WebP [15] and BPG (`bellard.org/bpg/`) as exemplified in Figure 1.1.

There are multiple architectures proposed for image compression, which are generally based on the same principle: the original image representation is transformed into a latent representation using a projection transform. The latent representation is quantized and entropy coded, to be sent as a stream of bits over a transmission channel, and then reconstructed using a synthesis transform. There are two common ways to ensure the compression of data: significantly restricting the size of the latent representation at the bottleneck [1, 3, 14], or approximating the entropy model of the quantized representation, allowing the rate-distortion optimization of the model [4, 16, 17]. DL-based image compression relies on the capacity of Deep Neural Networks (DNNs) to extract meaningful representations from two-dimensional data, since the latent space in which the network represents the image must contain information about the most important features and structure of the image. The network must also generalize this transformation so that it can then be inferred for images that have not been previously seen. Convolutional auto-encoders are particularly suited for image due to their ability to exploit spatial redundancy in images.

Even though image and video compression are two closely related areas, the study of video compression using DL-based techniques has taken a while to gain momentum, with scarce published research so far. As is detailed in Section 2.5.1, the quality of the solutions and our understanding on image compression has greatly advanced in the previous three years, but the same has not happened in the analogous field of video compression. As there is a close link between the video compression and the image compression problems, it would be expected that those two fields would evolve simultaneously. However, video compression is a more complex task due to the additional temporal scale of the data, the higher amount of data to process, and the inherent need to enforce higher compression ratios to perform up to par with currently used compression techniques.

This thesis focuses on the parallelism between image compression and video compression, as well as the lessons that can be learned from traditional image and video coding, in order to explore new approaches that can push forward the field of video compression using DL.

1.2 Objectives

The main objective of this dissertation is the advancement of the area of video compression using DL, which was an unexplored field at the beginning of this thesis. Consequently, the starting point of the research was instead focused on the application of DL-based methods to image compression.

For image compression, the objectives include both theoretical and practical work. Namely, a broad study and review of the field of image compression using DL and a practical implementation

and evaluation of a select choice of the studied techniques, both aimed at providing a solid base for the work in video compression.

For video compression itself, the main objective is to propose, implement and evaluate the first video compression architecture using DL, demonstrating that DL can be used to solve the problem. To do so, it is also necessary to present a theoretical formalization of the problem, in a framework that can be approached with DL-based methods.

This thesis is an exploratory work in the field of video compression. Thus, the proposed scheme does not intend to surpass the best performing codecs (*e.g.*, H.265/HEVC) for video compression, which are the result of decades of research. Instead, similarly to what happened for DL research in image compression, the focus is on opening a new path in the application of DL to an unexplored field.

1.3 Original Contributions

The original contributions presented in this MSc. thesis can be summarized as:

- In image compression:
  - Broad open-source implementations of image compression landmark works in PyTorch, and a thorough evaluation of the implemented methods attesting to the expected performance of the source articles (see Appendix B).
- In video compression:
  - A completely new formalization of the video compression problem as a latent-space learning task, for which a rate-distortion optimization framework is proposed, considering: reconstruction quality of the video, expected size of the compressed representation and temporal continuity between sequential frames;
  - A new spatio-temporal architecture using stacked autoencoders, with multi-scale connections and using three-dimensional convolutions that includes an extension of the scale hyperprior (as initially proposed by Ballé et al. [4]) concept to take into account existing temporal dependencies in a quantized latent representation;
  - An extensive analysis of the performance of the newly proposed architecture in terms of video quality assessment and runtime evaluation, using multiple metrics common to video evaluation, compared against widely deployed traditional video codecs and a naïve baseline.

The contributions of this thesis have been partially submitted for publication in:

1. J. Pessoa, H. Aidos, P. Tomás, and M.A.T. Figueiredo. ”End-to-End Learning of Video Compression Using Spatio-Temporal Autoencoders”, 7th International Conference on Learning Representations (ICLR), 2019 *(under review)*.

1.4 Thesis Outline

The remaining of the dissertation is organized in six chapters. Chapter 2 details the background work to contextualize the research in the deep learning. It includes a brief review of traditional image and video compression, and a thorough analysis of the research in DL-based image and video compression. Chapter 3 presents a discussion, implementation and the comparison of four landmark works in image compression using DL. The discussion includes a motivation to why each architecture was chosen, and the potential applications to video compression. Chapter 4 focuses on the video compression problem using DL. An initial motivation and overview of the research process are done. Subsequently, the video compression is formalized, followed by the proposal of a novel optimization framework and a corresponding spatio-temporal architecture for video compression. Chapter 5 explains and presents the evaluation process of the proposed video compression scheme. The comparison is done against a baseline (using image compression), and three video codecs: H.265/HEVC [18], H.264/AVC [7] and MPEG4 Part 2. The last chapter concludes the dissertation by assessing the work done given the initial proposition for the thesis, and finalizes with a proposal of the future work in the introduced field of latent-space video compression.

2

Background and Related Work

Contents

2.1 Motivation

To ensure a self-contained discussion of the deep learning applications in image and video compression, this chapter presents the necessary DL concepts used in the remainder of the document. These concepts were researched during an initial stage of the investigation process, although successively updated as new works appeared, given the high rate of progresses and discoveries in the DL field. Specifically, the initial sections are focused on a quick introduction to deep learning, followed by the techniques and architectures suitable for unsupervised learning tasks in image and video processing. Furthermore, to contextualize the image and video compression landscape, the chapter overviews recent traditional algorithms and techniques for both tasks, followed by a thorough review of the recent history in research of DL-based image compression, as well as a shorter (due to the scarce available research) look at DL-based video compression.

2.2 Background in Deep Learning

Deep Learning is a sub-field of Machine Learning that studies the process of learning estimated models from data representations using DNNs. In recent years, the field of Machine Learning has greatly benefited from the advances in this area, pushing the state-of-the-art forward in some areas such as computer vision and natural language processing. They are the result of several decades of research and a boost by breakthroughs in GPU computation efficiency [13] leading to a surge of investigation in DL. When supported with enough computing power and training data, research has shown that it is possible to approximate complex models for tasks such as voice recognition [19], text-to-speech synthesis [20], language translation [21], real-time object detection [22], image classification [23], *etc...*

Instead of task-specific algorithms, DL focuses on techniques that can be used in a broad range of problems. The 'Deep' connotation refers to the layered approach that is used to create DNNs, where layers are stacked in depth to improve the representation capacity of a network. Neural networks are particularly suited to learn complex models with a high number of hidden variables and dependencies while tolerating noisy data during training and execution. It is a recurring scenario in compression tasks, especially image and video compression, further motivating the research on using deep learning in both tasks.

DL studies both the process of creating DNN architectures and optimizing them to solve supervised (where there is a ground truth label in the training data) and unsupervised (where the network learns without labels) learning problems. In the following sections, a brief explanation of neural networks and the optimization process will be presented, followed by an overview of common techniques applied in Deep Learning, focusing on methods that are potentially useful for image and video compression.

2.2.1 Neural Networks

Neural Networks are structures loosely based on a possible interpretation of information processing and communication patterns in a biological neural system, where information is relayed through specialized groups of neurons responsible for specific operations. Conceptually, an Artificial Neural Network (ANN) is composed of neurons organized in layers, with a varying degree of connections to the previous and the next layer. On their own, neurons are simple processing units that apply a transformation function to a certain input to produce the output value (the activation of the neuron). The operations usually include trainable weight variables, and can also include a bias. As other methods of Machine Learning, neural networks are a flexible technique to automate model learning from training data, which can then be used for both supervised and unsupervised tasks.

In practice, neural networks are implemented as a set of ordered matrix operations that are applied over multi-dimensional matrices of data (*e.g.*, images, videos, feature vectors). In turn, layers are constructed using mathematically differentiable operations over matrix inputs. If the operation uses trainable weights or biases, they are positions of a weight matrix and bias matrix respectively, which are used in the applied operation, and where all the coefficients co-exist for that layer. This simplified approach allows the processing of neural networks to be optimized with highly parallel computations, contributing to the aforementioned advances in efficiency. The matrix representation also presents a simpler method of describing ANN-based architectures. Recently published articles adopted this simpler formulation to describe neural networks, as long as the focus of the concern is the whole architecture of the network. Following the same convention, this document will refer to networks with this adaptation in order to offer a discussion in-line with what is used currently in research.

The flexibility of ANNs, and why they are used in so many fields, stems from the fact that a particular architecture can have an arbitrary amount of layers, with different operations and varying configurations for each operation, all while maintaining a cohesive trainable unit. The different parameters that modify an architecture but are fixed during training are known as **hyper-parameters**. Almost all non-trainable properties of a network can be considered hyper-parameters, from the amount and type of layers, to training configurations.

Since the structure of the networks is complex and varied, to simplify the discussion it is usual for networks to be classified based on their properties. This is frequently done according to the operations applied at the majority of the layers, as in the case of Convolutional Networks. However, they may also be distinguished by their structure or the connections between layers as in the case of Recurrent Networks and Autoencoders. Notwithstanding, most of the concepts can be used simultaneously in the same network (e.g., there are formulations in which a network is both convolutional and recurrent) and it is easier to consider a neural network as a structured set of layers that apply operations, while the architecture may or may not fall within a well-defined class of networks.

2.2.2 Optimization

The optimization of an ANN requires two elements: an optimization procedure and an optimization target or loss function. The optimization procedure describes the steps and mathematical operations used to modify the weights of a neural network in each training operation, in order to minimize the loss function used for that network. In turn, the loss function guides the network towards the intended model to approximate, penalizing deviations.

Most optimization procedures that are commonly used in research are based on the gradient descent technique. Gradient descent optimization is divided into two phases: the forward and the backward propagation. In the forward phase, the training data is propagated through the network in the forward direction, and the corresponding transformations are applied, producing an output at the last layer. At the end of the forward pass, the loss function is calculated, evaluating the performance of the network. Subsequently, the gradient of the loss function is calculated with relation to each weight of the network, by propagating the error signal backward, in what is called the backpropagation phase (as the network can be considered a composition of functions applied at each layer, the chain rule of gradients is used). The gradients of each weight are used to adjust the weight in the direction of minimizing the loss function.

In theory, standard gradient descent optimization requires the training of the network with the whole dataset for each iteration, so as to minimize the global loss of the network. This process is prohibitively expensive for large datasets that may be impossible to fit in memory and is slow to converge when each step requires processing the whole dataset.

To circumvent this, the currently used techniques for training networks are based on a more efficient version of the gradient descent technique applied over one or a few training examples (a *batch*) at a time: the Stochastic Gradient Descent [24]. It is a probabilistic and randomized approximation of the gradient descent technique which approximates the true gradient of the loss function by considering only a few training examples in each iteration. This method has shown to converge fast enough and support arbitrarily large datasets [24], while having better chances escaping from local minima due to the randomness introduced when considering a small portion of the dataset at each iteration.

There are several variants of the Stochastic Gradient Descent, with the objective of improving both the results and the efficiency of the training procedure. One of the most used variants, which will be relevant for the task in hand is the Adam Optimizer [25]. Adam uses the concept of momentum: a residual gradient from the previous iterations, that is used to accelerate the approximation to the solution when the gradient changes in the same direction during several sequential iterations, speeding up the initial convergence during training. It also employs a selective learning rate for different parameters of the network to avoid the norm of the gradients of negatively impacting weights with smaller updates.

To implement backpropagation, the computation graph of a neural network (a record of which operations were applied to the input to obtain the output) is stored. Later, during the optimization stage, the gradients for each parameter are calculated and stored in a matrix of the same size as the weight matrix, which is then used as previously described for optimization.

2.2.3 Loss Functions

The loss function is a method that must be differentiable, which receives some information from the network's execution (*e.g.*, input, output, the structure of the data at a certain layer, *etc*...) and penalizes deviations from the intended model to learn.

For the unsupervised learning of image compression, the minimum that the loss function includes is a reconstruction loss, as the objective is to reproduce the original image as closely as possible after compression. The reconstruction loss used in training compressive networks calculates a distance between two images depending either on simple techniques such as pixel-wise distance or on more complex ones, such as convolutional application between images (*e.g.*, SSIM [26], MS-SSIM [27]).

The naïve approach using the absolute distance between every pixel in the image is based on the per-pixel norms of the residual vector between the original and the compressed image. In a generalized way, the L_n **loss** function determines the distance between images using the n-dimensional norm function so that $\mathsf{L}_n(x, y) = \sum_i^w \sum_j^h ||x_{i,j} - y_{i,j}||_n$, where w and h are the width and height of the image. The most common formulations are the L_1 and L_2 loss, penalizing deviations from the input linearly and quadratically, respectively. The L_2 **loss** is usually known as *Mean Squared Error* (MSE).

The reconstruction loss can be more complex as long as the operations used are completely differentiable; an example used in recent research is the optimization for the MS-SSIM [27] metric. MS-SSIM uses several applications of the SSIM [26] metric at different scales using downsampling.

Video compression and other tasks that require video reconstruction are usually optimized using a per-frame quality metric. The per-frame metrics have been shown to translate adequately to the final quality of the video, while being simpler than metrics that take inter-frame dependencies into account.

The overall loss function for both image and video compression can include more than just the reconstruction loss between two observations, as demonstrated by Ballé et al. [28]. In this case, the network is trained as a rate-distortion optimization problem, taking into account both the reconstruction quality of the image and the size of the compressed output, to balance both during training.

2.2.4 Deep Learning Layers

Layers are the essential building block of recent research in DL. As explained, they are constituted by differentiable operations over matrix inputs and can be stacked in depth to create expressive DNNs. In this section, the most important layers for image and video compression are presented.

A Fully Connected Layer

A fully connected layer of size n learns that same amount of linear combinations from the outputs of the previous layer, outputting n activations. This allows the layer to learn how to adjust, during training, the importance of the activation of the output of the previous layer to its own. With this operation, it can learn feature maps from the features extracted in the network up until this layer. In the fully connected layer, a position of the output usually specializes in different combinations of features that identify a particular higher-level feature, such as the existence of a square or circle in an

image.

Since this layer applies a linear function, it can only learn linear feature maps. Thus, if a network only uses fully connected layers like this, it can only act as an approximator for linear models, as there is no component in it that can mimic the behavior of a non-linear function.

B Non-Linear Activation

A non-linear activation layer is an essential component of Deep Learning which applies an element-wise non-linear function to the output of the previous layer. By augmenting the network with non-linear activation functions, it is possible to create intermediate non-linear feature maps, which greatly expands the capability of the network in terms of practical results, as almost all the complex problems in which neural networks are applied require non-linearity.

There are a lot of non-linear functions used in research, with the only necessary property being differentiability. The way a non-linear function reacts to its output varies; some are more sensitive to a certain zone of the input, others ignore a range of values, *etc*... The behavior of each non-linear function makes it more or less suitable for different problems. The most commonly used non-linear functions are the sigmoid (or logistic) function σ, where $\sigma(x) = \frac{1}{1+e^{-x}}$, the tanh (or hyperbolic tangent) function where tanh$(x) = \frac{2}{1+e^{-2x}} + 1$ and the ReLU (Rectified Linear Unit) calculated by the branch function

$$\text{ReLU}(x) = \begin{cases} 0 & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$$

There are also several variations for the ReLU method which slightly change the formula, and show improvements over the regular formulation such as the Leaky ReLU or the PReLU, proposed by He et al. [29]. While the original ReLU negates all negative inputs, both these variations allow a small residual response to negative numbers: the leak parameter in the Leaky ReLU specifies the magnitude of the response, while the PReLU has a learnable parameter that is updated during training controlling the response to negative numbers.

C Convolution

The convolution layer is one of the most used important for tasks related to image and video processing since it can capture local patterns within an n-dimensional signal, by applying convolution operations to the input. For the particular case of image and video, convolutions are especially useful in extracting structural redundancy within their input. Deep Convolutional Networks can capture simple patterns at the first layers, which can be made increasingly complex as multiple layers are stacked. For example, in images, particular features such as edges, surfaces, and curves are typically captured in the first layers, and general structural patterns such as squares, rectangles, circles, complex shapes and so on in the deeper layers.

A convolution layer has a set of kernels with weights that are tuned during training. The layer applies convolution operations per kernel; the number of kernels in a layer is referred to as the number of channels of the convolution layer, where each channel calculates the output based on its kernel.

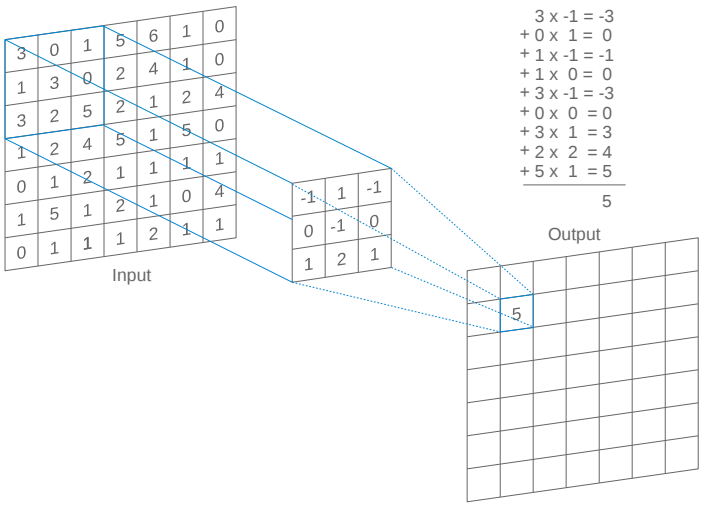


Figure 2.1: Application of a convolution using a kernel of size 3x3 to an input. The result of the convolution is represented by the sum on the upper corner of the image. The full convolution results from repeating the operation for each possible position of the input according to the kernel size, stride and padding

The size of the kernel determines the receptive field of the convolution and the granularity of features that it can capture. The kernel for an n-dimensional convolution is an n-dimensional matrix, and each application of this kernel will consider an input with the same size. It also has a set of other parameters with the same dimensionality as the kernel. Padding controls the border added to each side of the input before the application of the convolution in order to guarantee that the convolution takes all the positions of the input into account. Stride controls the distance to use between each application of the kernel, which can be used to downsample the input if the stride is higher than one. There are other parameters to further tune the convolutions such as channel grouping, which are not relevant for image or video compression.

To simplify the explanation of the convolution operation this section will focus on the case of two-dimensional convolutions, as are applied on images. The extension for higher dimensionality convolutions can be done by simply applying all the operations using the additional dimensions. For two dimensional convolutions consider a kernel with size (K_w, K_h), of padding (P_w, P_h) and stride (S_w, S_h). The commonly used convolution operation calculates, for each position of the output, the result of the valid cross-correlation operation between the convolutional kernel and a portion of the input with the same size (sum of the point-wise multiplication between each element of the kernel and the section of the input), as demonstrated in Figure 2.1. The operation is repeated by moving the kernel horizontally by S_w units until the width of the input is exhausted. At this point, the kernel is moved back to position 0 and moved vertically S_h units. The process is repeated until the input is fully processed, once for each channel of the convolution.

The size of the output of a convolution layer is important when stacking layers to obtain the desired output size. Given an input of dimensions (H_i, W_i), the width W_{out} and height H_{out} of the output are calculated as

$$
\begin{aligned}
H_{out} &= \text{floor}((H_i + 2 \times P_h - (K_h - 1) - 1)/S_h + 1), \\
W_{out} &= \text{floor}((W_i + 2 \times P_w - (K_w - 1) - 1)/S_w + 1),
\end{aligned}
\tag{2.1}
$$

where W_{out} and H_{out} are the width and height of the layer's output, respectively.

D Pooling

Pooling operations are used to sub-sample the result of the previous layer by applying a function that combines the elements of a moving window from the previous layer output (commonly used with convolutions). It reduces the dimensionality of the data while retaining important information. In turn, this layer simplifies the complexity of a network following the pooling layer. The used pooling operations vary from **max-pooling** (choosing the value of the maximum activation from the window), **average-pooling** (averaging the values from the window) to more complex functions with specific uses. The application of the pooling layer follows the same principles of the window-based application of the previously described convolution, and require similar parameters such as padding and stride.

E Transposed Convolution

A transposed convolution layer shares all the properties of a regular convolutional layer but applies the transposed convolution operation instead of the regular convolution. In this case, each application of the kernel covers only a single source position (of the padded input) for each position of the output, effectively doing the inverse of the convolution operation. This means that the application of a transposed convolution layer creates an output that is at least of equal size than the input, or larger. The size of the output of this layer (H_{out}, W_{out}) is calculated as follows:

$$
\begin{aligned}
H_{out} &= (H_i - 1) \times S_h - 2 \times P_h + K_h, \\
W_{out} &= (W_i - 1) \times S_w - 2 \times P_w + K_w.
\end{aligned}
\tag{2.2}
$$

The transposed convolutional layer is usually implemented in practice as a convolution layer where the forward and backward pass are swapped. This layer is frequently used to map inputs from lower dimensional spaces into higher dimensional spaces, or as a learnable up-sampling operation for image related tasks.

2.2.5 Deep Learning Network Classes

Most of the described operations can be interchangeably applied to create different architectures, but there are common design choices when devising networks that are shown to provide good results in practice. To facilitate the denomination of networks, the common cases are usually grouped into classes of networks that share common properties. The most important classes for the scope of this dissertation are addressed below.

A Autoencoders

Autoencoders are a network architecture first proposed by Rumelhart et al. [30] in 1986, as a way to solve an unsupervised learning problem. Autoencoders are always composed by an encoder unit

and a decoder unit, which must be trained simultaneously but can be used separately. Between the encoder and decoder, there is at least one bottleneck layer, on which a compacted representation of the input is generated. The encoder is used to extract a set of features of the input (which depend on the optimization target) representing the input in a different space than the original structure. The space in which an autoencoder represents the input is described in the literature as the latent space, and a representation in that space as a latent representation. A value of the latent representation is also called a latent attribute. The decoder is then used to project the extracted latent representation of the input to a new structure (*e.g.*, the original structure of the data). The architecture of an autoencoder allows the network to train the different layers of the network so that the learned latent space has the necessary features for the decoder to achieve the optimization target.

Autoencoders can be used to learn efficient transformations of data to lower-dimensional spaces [31] by ensuring that the latent space has a smaller size than the original inputs. Due to their obvious similarities with a compression scheme, autoencoders have a very important role in the research of different compression problems using neural networks, including for images [1, 2, 14, 16, 17].

B Convolutional Networks

Convolutional Neural Networks (CNNs) are simple networks that use sequences of convolutional layers and optionally pooling and non-linear activation layers, to create a hierarchical decomposition of the input signal. Since the convolution is a linear operation, the non-linear activation functions allow the network to approximate non-linear feature maps, instead of being limited to linear approximations. When designing convolutional networks it is necessary to take the size of the output of each layer into account so that the final output of the network corresponds to the desired size.

Deep CNNs excel in exploiting spatial structure in images (or other two-dimensional data) to extract features, creating meaningful representations that can be used for processing tasks. One of the most notorious uses of these networks is in image classification [23]. They also have shown great results when using one-dimensional convolutions, for example in sequence-to-sequence language translation tasks [21].

C Recurrent Networks

Recurrent Neural Networks (RNNs) are a class of neural networks that present connections between units in such a way that the information also flows cyclically during inference. In other words, a recurrent neural network propagates the input signal forward as a regular network, but a portion of the output is propagated back to the same unit that produced it, the **hidden state**, to store a context of what was previously processed. As exemplified in Figure 2.2, an output h_t is produced in each iteration (marked in green) and a hidden state that is propagated to the next iteration (marked as a gray arrow), serving as a form of memory between iterations. The right side of the figure demonstrates the result of unrolling the execution of the network, to make clear how the propagation of the hidden state between iterations occurs. The hidden state is usually reset between new sequences. So, in the example of the figure, the hidden state is propagated when processing elements of the sequence

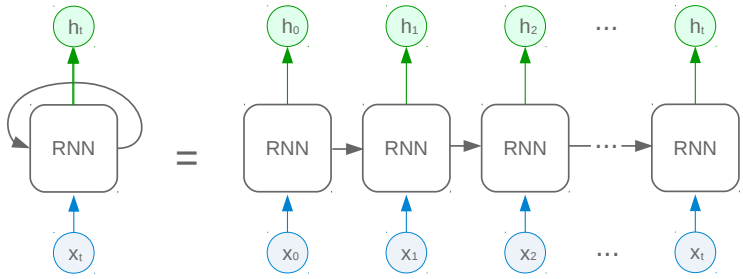


Figure 2.2: A simplified representation of the execution flow of a RNN, for several iterations . On the left of the figure, the normal execution is represented. On the right of the figure, the unrolled execution is presented.

$(x_0, x_1, ..., x_t)$, but if the network was to process a new sequence $(y_0, y_1, ..., y_t)$, then the hidden state would be reset. This extension of the normal neural network formulation provides the networks with the capability of properly processing sequential information that would be otherwise difficult.

As opposed to the previously discussed classes of networks, when training recurrent neural networks, the most common scenario is to only perform the back propagation phase and gradient descent after a whole sequence is processed. The loss function of the network may take into account the output at each iteration or only the final output of the network, but the network is still trained only after processing a whole sequence.

There are several recurrent units with varying degrees of complexity, that calculate the hidden states differently, and excel at different tasks. For example, Long Short-Term Memory Units (LSTMs) are a recurrent neural network element proposed by Zaremba et al. [32] that propagate two different states between iterations: the cell state and a hidden state. Formally, let x_t, c_t and h_t denote the input, cell state and hidden state at iteration t, respectively. Each iteration of an LSTM unit takes the input x_t and the previous cell and hidden state h_{t-1}, c_{t-1} and produces an output o_t as well as the current cell and hidden state. The operation applied by an LSTM is formulated as

$$\begin{aligned} [f, i, o, j]^T &= [\sigma, \sigma, \sigma, \text{tanh}]((Wx_t + Uh_{t-1}) + b), \\ c_t &= f \odot c_{t-1} + i \odot j, \\ h_t &= o \odot \text{tanh}(c_t), \end{aligned} \tag{2.3}$$

where the $\odot$ operation represents the Hadamard (element-wise) product. Being a more recent proposal, Gated Recurrent Units (GRUs), are simpler recurrent units first proposed by Cho et al [33] in 2014, with only update and reset trainable weights. Each GRU's hidden state is simply its previous iteration output. Given an input x_t and a hidden state output h_t, the GRU can be formulated as

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1}), \\ r_t &= \sigma(W_r x_t + U_r h_{t-1}), \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \sigma(Wx_t + U(r_t \odot h_{t-1})). \end{aligned} \tag{2.4}$$

Even if the GRU is a simpler unit, it has been shown to perform equally to the LSTM or even better in some cases, as will be discussed in section 2.5.1 and section 3.7 in relation to the image compression task.

D Recurrent Convolutional Networks

Recurrent Convolutional Neural Networks (RCNNs) are the result of extending recurrent units such as the described LSTMs and GRUs with the convolution operation, where normal element-wise multiplications are expected. The obtained units are able to contextually process inputs, while still benefiting from the convolution operation to extract structural information from spatial dependencies. This can be used, for example, to process spatial information which is sequentially ordered (within a temporal dimension) as would be the case for video (explained in more detail in section G).

The equivalent of the LSTM unit, in this case, is the convolutional LSTM (CLSTM). This unit is obtained by replacing the element-wise multiplications (Wx_t and Uh_{t-1}) by convolutions in the first expression of Equation 2.3. Similarly to the CLSTM, the Convolutional GRU (CGRU) is formulated by replacing all element-wise multiplication between any W matrix and the input x, and between any U matrix and the hidden state h in Equation 2.4.

E Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a class of networks introduced in 2014 by Goodfellow et al. [34], which are used for unsupervised learning. They work by using a system of two networks that compete against each other, in order for the generator's outputs to conform to a certain probability distribution of values (which may be difficult to learn otherwise).

The usual structure for a GAN is to have a Generator Network, a Discriminator Network and two different sets of inputs: real data and artificially generated data. The purpose of the Generator Network is to create the artificial inputs that are sent to the Discriminator, in order to fool the discriminator into thinking they exhibit the same value distribution; in turn, the purpose of the Discriminator Network is to learn how to distinguish real inputs from the artificial examples created by the generator. The training process is akin to a zero-sum game where the Von Neumann's min-max theorem applies [34]: the discriminator network calculates the loss based on the results of its predictions for arbitrary inputs, and the generator network calculates its loss based on the quality of the discriminator's predictions for its outputs. For a generator G and a discriminator feedback D (loss function of the discriminator), the initially proposed adversarial loss for the generator is calculated as

$$\mathcal{L}_a((x, y), D) = -\frac{1}{2}\log(D(x)) - \frac{1}{2}\log(1 - D(G(y))), \tag{2.5}$$

where x is a real input, and y is either a random vector of noise or a specific input (depends on the network's purpose)

During the regular training process for GANs, the generator learns how to create artificial samples from its input, which should fool the discriminator by approximating the probability distributions of real data. After training, the generator can be used to infer new samples that were not previously in training data, and so it can be considered a generative model. However, the adversarial loss can also be combined with other losses for purposes other than generative models. In this case, the network is usually provided with a particular input and target output, and the discriminator is used to ensure

that the output conforms to the desired probability distribution. Both cases are a form of **adversarial training**.

F Variational Autoencoders

A Variational Autoencoder (VAE) is a particular formulation of an autoencoder first proposed by Kingma and Welling [35], trained with recourse to variational inference, in order to provide a probabilistic manner of describing an object in the learned latent space. Instead of mapping an object to a discrete point of the latent space, as a regular autoencoder, the VAE maps each value of the latent representation to a learned probability distribution. As a consequence, the latent representation of an input for a VAE is not a point of the latent space but parameters to the probability distributions that describe the input.

The regular autoencoder requires a training sample for each point of the latent space. In comparison, VAEs allow the network to learn the overall distribution of data by considering the training data as a point sampled from the distributions that characterize each attribute of the latent space. Computing the probability distributions directly for each latent attribute may be too difficult if the corresponding solution has no closed-form, or is computationally expensive (referred to as an intractable probability distribution). Variational inference allows the network to approximate arbitrary density functions, even if they are intractable, through optimization [36].

VAEs are trained by minimizing the KL-divergence (a measure of the difference between two probability distributions) between the probability of the encoder inferring a certain latent representation of an input and the probability of the decoder inferring the same input from sampling a random point from that latent representation. As the process of random sampling is not differentiable, VAEs require the use of what is called *reparametrization trick*: the input for the decoder is sampled from a unit Gaussian distribution and then shifted and scaled by the mean and variance, which constitute the latent representation. Thus, the input is described in the latent space by parametrized probability distributions which can still be optimized due to this technique.

As the learned latent space is a set of probability distributions, it is possible to interpolate between latent representations by sampling new points for each latent attribute from the learned distributions. If the network is provided with an unseen vector, it generates a new point of data that was not part of the training data. Thus, the VAE, as a GAN, can be used as a generative model.

G Spatio-temporal Autoencoders

Spatio-temporal autoencoders are a sensible choice to model or extract dependencies in data that has both spatial and temporal dimensions, as they extend the convolutional autoencoder formulation with the ability to extract information about spatial and temporal dependencies in the data. Contrary to the previously presented classes, spatio-temporal autoencoders are a conceptual class of networks that can be approached by different formulations.

RCNNs (see D) are one possible implementation to combine the extraction of spatial information using convolutions [37], with the state propagation between iterations to exploit temporal dependen-

cies. RCNNs have been successfully used for tasks requiring spatio-temporal reasoning (*e.g.*, in the work of Wang et al. [38], Liu et al. [39], Patraucean et al. [40]). RCNNs extract a latent representation for every unit in the temporal dimension, meaning that the number of iterations of the network must equate the number of temporal steps.

The other option for spatio-temporal autoencoders is to use Three Dimensional Convolutional Neural Networks (3DCNNs), which are structurally similar to convolutional autoencoders but use three-dimensional (3D) convolutions at each layer. This type of network is typically applied to problems in which the data resides on a 3D grid structure (*e.g.*, point clouds, 3D meshes or 3D medical imaging). 3DCNNs can also be adapted to process spatio-temporal data (*e.g.*, Zhao et al. [41]) by organizing data on a 3D grid structure, allowing the joint processing of spatial and temporal information.

2.2.6 Deep Learning Frameworks

There are several leading deep learning frameworks that are widely used for research and production. Some notable examples are TensorFlow, Theano, Caffe, Torch, Keras and more recently PyTorch.

There is an important distinction between the existing frameworks, which separates them by the type of execution. Some frameworks perform **static** graph computation in order to run and train networks, which means that a neural network must be symbolically defined before execution, and then can be run inside the framework's environment, leaving no control of the execution flow to the user. In opposition, there are also frameworks that provide **dynamic** graph computation where the user has full control of the environment during the network's execution and its structure can change from one iteration to the next: the networks are dynamically defined during execution. From the enumerated frameworks this dissertation provides a brief analysis of two tested frameworks, one for each class: **Tensorflow** [42] and **PyTorch** [43].

In the case of the Tensorflow framework, the networks are defined statically, where trainable weights and bias are defined as variable Tensors. The inputs are marked with placeholder Tensors that are replaced during execution with the input data. The network is then defined as a connected graph of operators (logits) that modify the aforementioned inputs, and produce an output. These networks can be run inside a Session that initializes the necessary components of the framework to execute a neural network and registers the trainable variables that must be tracked for optimization. The user can attach an optimizer to train the network, and any other necessary procedures. The static definition of the graph allows the networks to be optimized for performance prior to execution, as their structure is previously known.

In the case of the PyTorch framework, the networks are defined as dynamic graphs. This means that the user has a set of methods that can be applied during execution to create a neural network. By extending the existing Module definition (encapsulates the network), the users can override the forward propagation of an empty network to define a custom architecture. The training of this type of networks is possible due to the AutoGrad feature (Automatic Gradient Differentiation)[43]. Each operation during the dynamic execution is registered in the history of the operation's output, meaning

that the output of the network recursively contains the history of operations starting from the input. This information is used to automatically calculate the gradients of a certain error function in relation to the weights of the network by iterating the execution graph backward and performing the differentiation for each operation. Since the user is always in control of execution, this framework requires writing more code, including the training by iterating over the dataset once per epoch, and all the necessary flow control cautions. However, the code written is more intuitive since it fits a regular imperative coding style.

There were several identified differences during experimentation with both frameworks. Dynamic graph computation allows for easier debugging of the created networks and more flexibility in implementing new techniques. As the tested dynamic computation framework, PyTorch also provides seamless integration with several Python scientific frameworks that simplify development such as SciPy, NumPy and Python Imaging Library. In comparison, Tensorflow provides a wider range of implemented operations and is established as a reference in deep learning for both research and production. This framework had more time to mature, and more research projects using it.

In the end, PyTorch was used for all the research in this thesis, as it presented a more intuitive workflow, especially in implementing custom techniques that are not part of the pre-implemented functionalities. It also has an easier learning curve, while offering competitive performance and an equally active and responsive community.

2.3 Evaluation Metrics

For a designed and implemented network architecture, it is necessary to define objective comparison metrics that can be used to evaluate the performance of the optimization process. There is a clear similarity between evaluation metrics and loss functions as the objective is the same: to objectively calculate the quality of the result of a network. However, loss functions require differentiablity in order to be used for backpropagation which is not a restriction in the case of evaluation metrics.

Evaluation metrics are domain specific, considering the data and the purpose of the task. In the case of image quality, the commonly used metrics compare two images: an input and an output. The objective of image quality metrics is to score the quality of the image in a way that corresponds as closely as possible to the attributed relative quality by the opinion of humans. In other words, image quality metrics should approximate the perceived quality of the human visual system as best as possible.

As videos are a sequence of frames, most of the times they are evaluated using an average of an image assessment metric applied to each frame, referred to as per-frame metrics. There are video evaluation metrics that take into account inter-frame dependencies but they are complex to implement and do not always correlate to better quality when scored by humans. For the purpose of this thesis, the focus will be on per-frame metrics.

Both traditional image and video compression schemes are commonly optimized for the Peak Signal-to-Noise Ratio (PSNR). The PSNR is calculated using the previously defined MSE (see Sec-

tion 2.2.3) and is measured in decibels (dB), given the original image f and the reconstructed image $\hat{f}$, as

$$\text{PSNR}(f, \hat{f}) = 10 \log_{10} \left(\frac{R^2}{\text{MSE}(f, \hat{f})} \right), \tag{2.6}$$

where R is the range of the pixel values. This metric only evaluates the pixel-wise statistical properties of two compared images. Although it has served the area of image and video compression well in the previous decades, there is enough evidence that points to PSNR being lackluster in terms of detecting particular coding artifacts and other distortions, especially compared to SSIM (Structural Similarity) [44].

SSIM [26] is a more complex metric based on convolutional applications of a search window, which tries to extract a quality index of an image based not only on pixel-wise metrics but also on local comparisons using a broader receptivity field for improved results. SSIM is a clear improvement over PSNR, but the recently proposed Multi Scale Structural Similarity (MS-SSIM) [27] achieves even better results by using several applications of the SSIM metric in increasingly lower scales of the image (through downsampling), not only capturing low-level local distortions but also higher level artifacts. Although there are other proposed metrics which achieve good results, MS-SSIM has been widely used in research to access the quality of both image and video compression schemes [4, 14, 28] particularly due to its differentiability which allows direct optimization for this metric. MS-SSIM will be used for most of the comparisons present in this document in order to facilitate comparison with other research and any other compression schemes or implementations that may be devised in the future.

2.4 Traditional Image and Video Compression

2.4.1 Image Compression

All of the image compression algorithms rely on the fact that in images, pixels are highly correlated with their neighbors. This means that these pixels contain redundant information, both due to the nature of the visual representations and the binary coding schemes used. The end goal of an image compression algorithm is to create an intermediate representation that minimizes the correlation and consequently the redundancy of the information stored, to shrink the size of that representation. More specifically, it is possible to distinguish two types of redundancy that are exploited by compression algorithms:

1. **Spatial Redundancy** The values of a pixel in a two-dimensional image can be strongly related to either its neighboring pixels or another area of the image, both within the same color channel and within different channels. The pixels create features (*e.g.*, surfaces of the same color, edges, pattern repetitions, etc) that are structurally predictable and repeat along the image. This means that the associated values can sometimes be inferred from other pixels and recurring patterns in the same image.

2. **Coding Redundancy** As in any file or data stored in a sequence of bits, there are repeating sequences of bits of varying sizes that are bound to be repeated. The common repetitions can

be abstracted to reduce the size of the final code. A coding scheme is as good as the estimation of the likelihood of the sequences [45].

Additionally, since the human visual system does not perceive all information in the same way (e.g., it is more sensitive to changes in brightness than color) algorithms can exclude part of the visual information during encoding, taking care to minimize the lost quality.

The example of JPEG, currently one of the most widely used compression algorithms, shows how this last property can be exploited in different ways: before encoding an image in *RGB* (Red-Green-Blue) space, the algorithm converts it to *YCbCr* (Luma-ColorBlue-ColorRed) space. It is then possible to downsample some components of the image reducing the size used to encode the chroma differences (Cb and Cr), which are less noticeable by the human eye. The downsampling has different modes such as 4:2:2 (meaning that both chroma parameters are sampled at half the resolution of the luma), 4:2:1, 4:1:1, and so on. The JPEG algorithm subsequently splits the images into 8x8 blocks and uses the **Discrete Cosine Transform** (DCT) to convert each block into a frequency-domain representation. The DCT has a strong energy compaction property [46] which means that most of the important information from the original signal (using a model based on the human visual system) is concentrated in a few low-frequency components. Thus the compression can exclude the higher frequency components to compress the original file with a small impact on the overall quality of the result. Both techniques remove information in a way that has almost no impact in our perception of the image, and the actual lossy compression that produces the artifacts JPEG is known for, happens during the next stage. After the DCT transform is applied, JPEG applies a quantization method, that discards some of the information in the remaining components, by reducing the possible values for each component with a rounding strategy allowing the fine-tuning of the distortion to compression ratio. The last step of the algorithm is to apply lossless entropy encoding of the resulting bitstream from the quantization, minimizing the coding redundancy.

An improvement over the JPEG algorithm, known as JPEG2000, was presented by the same authors, featuring a similar overall process but using wavelet transformations instead of the DCT, variable sized blocks and a more complicated quantization and entropy coding strategy. While JPEG2000 seems like an improvement over JPEG, it is more computationally expensive and the improvement of the results is marginal (and potentially worse at low compression ratios [47]), halting the efforts to replace JPEG by JPEG2000.

Even though the JPEG algorithm was initially proposed in 1992, it still serves as a good introduction to modern image compression, as several of its techniques are used in state-of-the-art algorithms as of today. More specifically, the WebP codec recently proposed by Google consistently produces much better results than JPEG in all compression ratios [15] but reuses some concepts of the older scheme. The major conceptual difference between both is the introduced method to explore spatial redundancy known as **block estimation and prediction** in which the subdivisions of the image are compared to other similar areas to find matches in redundant information for both areas (if successful, the two areas are matched). The image is subdivided into square blocks by selecting areas of the image with low entropy within the pixels of a square (and not necessarily fixed 8x8 blocks), denomi-

nated macroblocks. As done in JPEG, the image is converted to the YCbCr color-space before this process. After the space conversion and block estimation and prediction, the macroblocks that were not matched to a similar area need to be fully compressed, undergoing a DCT transform, followed by a novel quantization step tuned to enforce the requested compression ratio. Within each matched macroblock, when possible, only the motion (offset to the image origin) and color information that is redundant is stored as the residual differences. The final step of the algorithm is a newer entropy coding scheme, Boolean Arithmetic Coding, that shows considerable improvement over the run-length encoding strategy of JPEG [15].

To summarize, the image compression algorithms tend to focus on two properties: spatial redundancy and coding redundancy, discarding more information when necessary. In recent years, well-known techniques have been augmented with new methods such as block prediction to improve the results, achieving higher quality compression and smaller file sizes, as demonstrated by the recent WebP and BPG codecs [15].

2.4.2 Video Compression

The landscape of traditional video compression schemes is less homogeneous than the video compression one and has more diversity in the different types of algorithms tested for their applications in video compression. In fact, according to Vigliano et al. [48], this scene can be divided in four different classes that have been studied in the past couple of decades: waveform transform based [18], object-based [49], model-based [50] and fractal coding based [51]. While they all have demonstrated viability, the algorithms using waveform transforms have consistently achieved better performance for conventional use, and as such are the main compression method currently used in video coding [18][52]. These algorithms rely on motion-prediction/frame-estimation techniques for their functioning and will be referred as such during the remainder of the dissertation. Hence, this section will focus only on prediction architectures, as any video compression scheme will be compared against them.

Intuitively, the compression of video is deeply linked with the compression of images. In fact, all of the wavelet transform video compression algorithms must rely on some sort of image compression technique to explore spatial redundancy as one of the several tools to reduce the size of the media (e.g., the H.264/AVC codec utilizes a similar prediction process that inspired WebP when exploring spatial redundancy with intra-frame prediction). However, videos display another interesting property that is explored by video compression algorithms: the pixels of a frame are not only correlated with their neighbors in the same image (**spatial redundancy**) but they are also correlated with the pixels in similar positions of the previous and next frames in the sequence. This is denominated as **temporal redundancy** and provides the basis for most of the differences between the image and video compression schemes. One interesting aspect of temporal redundancy is that blocks of pixels can move along a sequence of frames so that a group of pixels is still correlated to previous frames, but within a moving reference frame. If this motion can be detected, the information about the moving blocks does not need to be repeated.

Currently, the best performing video compression algorithms use a technique denominated **block motion estimation** (also known as **motion compensated prediction**) to explore the spatial and temporal redundancy within an image. Two widely used and similar compression schemes are the H.264/AVC [52] and H.265/HEVC codecs [18] (H.26x will be used when referring to both codecs); while they have a range of small tricks to increase the performance wherever possible, the motion compensated prediction is the main technique powering the results of both algorithms.

In the case of the H.26x codecs, and similarly to WebP, each frame of the video is subdivided, and each division can either be fully stored, or encoded as the residual error against any other macroblock. In the H.264 codec, the divisions are macroblocks sized up to 16x16 pixels, but in H.265 the concept of macroblock is replaced by Coding Tree Blocks (CTBs): quadtree structures that recursively store information of areas up to 64x64 pixels. The subdivisions can go as small as 8x8 pixels and are called Coding Units (CUs). The new approach to encode predictions in different resolutions provides increased flexibility for H.265 when compared to H.264.

The process to encode a macroblock is similar to the encoding of each CU. The algorithm tries to find a matching zone from which the new block can be predicted so that only the residual error between them must be stored and is minimized; the search length for the best matching block is determined by the available computational budget that is attributed to the compression process. However, iterative greedy algorithms are now usually favored due to the computational complexity of full search strategies. Finding the best matching block is similar to the prediction within an image; however, in the case of video, the encoding can take into account not only the same frame (**intra-prediction**, *I-frames*), but also any other frame, forward or backward (**inter-prediction**, *P-frames* and *B-frames*). The mentioned codecs only support one type of prediction per frame, thus the frames can be classified accordingly to the type of prediction used on that frame. The I-frames are encoded only using intra-prediction. There is an additional distinction for frames that use inter-prediction: P-frames are encoded using data from previous frames and B-frames are bidirectionally encoded, meaning that they can use data from previous and forwards frames.

The H.265 codec uses these two techniques alongside a quantization step (which helps regulate the bitrate): the DCT for its energy compaction property (as in WebP and JPEG) and boolean entropy coding (to tackle coding redundancy), producing impressive results when compressing video compared to previous algorithms. The intuitions used to produce these results should also be useful when devising a neural network architecture for video compression, even if not directly applicable.

2.5 Image and Video Compression using Deep Learning

2.5.1 Image Compression

Image compression is formulated as an unsupervised learning problem, as the problem only requires an uncompressed image to be applied. To this effect, deep learning was initially adapted to perform unsupervised learning by Rumelhart et al. [30] with the proposal of the **autoencoder** architecture. Similarly to the autoencoder architecture of encoder, bottleneck and decoder, the problem

of image compression can be formulated as three phases: a projection transform that produces an intermediate representation in a latent space, a sequence of bits that is produced by entropy coding the latent representation, and a synthesis transform to recreate the input from the encoded latent representation. This makes the autoencoder network architecture seem intuitively suited for the problem and constituted the initial efforts in compressing images with neural networks.

In the 90's several published surveys had already shown the interest in investigating the use of neural networks for image compression [12, 53]. Jiang [53] in 1990 notes three different ways in which neural networks were expected to help in image compression:

- direct image compression using neural networks with autoencoder-like architectures
- implementations of existing techniques using neural networks
- using neural networks to improve some parts of traditional algorithms

The potential of using networks for this task is acknowledged at this time and some promising theoretical results are registered, but the approaches using neural networks still faced three big problems as was later identified [31]: the need for huge datasets of uncompressed images to use for training, the computing power to process them in a reasonable time, and the ability to develop a flexible architecture for compression applicable to images of all sizes. From the three possible approaches, the test of time has shown that using neural networks with autoencoder architectures is the best way to approach the problem [1, 4, 14, 17], and will be the focus of this sub-section.

In 2006 Hinton and Salakhutdinov [31] demonstrate the viability of using autoencoders for end-to-end compression of data, focusing on images, and motivated by both the improvements in computation efficiency for neural networks and the increasing computation power available. The researchers used a network with a small central layer as the bottleneck and an intelligent initialization of the weights to produce lower dimensional representations of images with smaller sizes than the original, which could then be used to recreate the input with decent results (no metric for the quality of the reconstructions is used). However this research did not produce binary versions of the reduced images during the bottleneck layer, did not fix the flexibility problem and used an architecture that was not particularly suited to images, as it was not the intended purpose of the research.

By 2011, Krizhevsky and Hinton [54] demonstrate the utility of binarizing the latent representation of an image generated the bottleneck layer of an autoencoder. Although this research applied to the problem of retrieving related images instead of encoding and decoding an image for compression, it showed the possibility of quantizing a latent representation of an image. The authors demonstrate that the quantized representation was a meaningful and efficient description of the image, that could even be used to compare images accurately, solving one of the previously mentioned problems.

The research in this task moved slowly until 2014/2015. At this point, the research in image compression sees a resurgence with the advances in parallel of other image processing topics such as image super-resolution [55] (solving the inference of high-resolution images from lower resolution original samples) and image denoising [56].

In 2015 Toderici et al. [2] achieved an important milestone in image compression by presenting a novel framework that allowed for variable rate compression based on the autoencoder architecture and residual encoding. This novel approach introduces the flexibility that was missing in previous attempts by compressing the image in iterations, encoding the residuals between the original and the currently encoded image in each iteration. The researchers tested different possible networks within the framework including RNNs, CNNs and Recurrent Convolutional Neural Networks (RCNNs), specifically the recurrent LSTM introduced also in 2015 by Shi et al. [37]. The authors also discuss a known problem of quantization for training models: a quantization function such as rounding stops the propagation of the gradients backward through the quantizer, thus difficulting the training of the encoder. Their solution is based on a classical result from quantization theory [57] noting that high-rate quantization noise/error is well approximated by additive noise, with uniform distribution in the quantization interval. Thus, during training, the explicit quantization is replaced by additive noise with a uniform density on the interval $[-\frac{1}{2}, \frac{1}{2}]$. This approach, referred to as stochastic binary quantization technique, is similar to a technique proposed by Krizhevsky et al. [54], used to create binarized representations which can be optimized during training. The authors conclude that the Recurrent Convolutional Networks have the most promising results, but acknowledge the need for further research in the area (interestingly, they also mention the need to test the viability of this approach for video compression). The limiting factor of this article when comparing the proposed solution to existing compression schemes is the evaluation in 32x32 pixel images. This puts the neural network architecture in considerable advantage due to the optimizations used in the comparison algorithms (JPEG and JPEG2000) for higher resolution images.

Building upon previous research in variational autoencoders [35], Gregor et al. [58] (2016) demonstrate that a Recurrent VAE can be used to extract hierarchical latent representation with increasing levels of detail, which are focused on global information first. The authors show how the latent representations at different iterations are useful to compare images in a latent space with high significance (as in the work of Krizhevsky and E. Hinton [54]), and to compress the image by discarding lower level detail at higher compression ratios. The achieved results are similar to the previous work by Toderici et al. [2] albeit the low bitrate representations provide more faithful reconstructions of the whole image. However, the authors do not binarize their latent representations, leaving this work short of being a full compression scheme.

Toderici et al. [3] (2016) continue to improve on their previous work, by introducing significant improvements to the proposed network architecture and the entropy coding techniques. The result is a new compression scheme that can successfully tackle full resolution images using the same recurrent convolutional networks. They replace the previously used CLSTMs units by new convolutional recurrent units which perform better for this task (CGRUs). They propose a new image reconstruction method: one-shot reconstruction where the network predicts the full image in each iteration, making use of the memory properties of recurrent units, and compare it to the previously proposed additive reconstruction, which predicts a residual component (the difference between the original image and current reconstruction) in each iteration. In the last case, the residual components are added in the

end to complete the reconstruction. They also switch the transposed recurrent convolutions by a technique first introduced for super-resolution tasks: a depth to space or pixel shuffle transposition [59] followed by a convolutional recurrent unit. The result is similarly an upsampled image, which exhibits fewer checkerboard artifacts as usually introduced by transposed convolutions [60]. Finally, they replace the simpler entropy coding by a technique based on the PixelRNN architecture [61]. While their best result surpasses the widely deployed JPEG and follows closely behind the JPEG2000 codec, this solution still suffers from blurriness in low bitrates, as identified by the authors. They note that an important step of research would be to improve the awareness of the network to higher entropy areas of the image to compress.

Balle et al. [16, 28] (2016) propose a completely new approach to the image compression problem: rate-distortion optimization. Instead of using a dimensionality bottleneck in which the latent representation is smaller by forcing a reduced size, the authors learn an entropy model of the quantized latent representation during training, using probability density modeling techniques, which can then be used to control the information flowing through the bottleneck. As the final size resulting from the entropy coding is directly related to the estimated entropy model of this quantized representation, the authors can include a term penalizing that size in the optimization target of the network. Thus, this new training framework allows the joint optimization of the reconstruction quality and the output size of the compressed image. To allow this rate-distortion framework to be efficient the authors also propose a new way to quantize the compressed representation by simply rounding the latent representation, and letting the entropy coding stage deal with how to efficiently binarize the tensor. The proposed network is simpler than the previously presented recurrent networks but their results are on par, or slightly superior to the work that was done so far; however, this new architecture can only be optimized for one target bitrate at a time, deviating from the previous research in variable bitrate compression schemes. The authors also explore optimization for different metrics. As an additional remark, the article shows how optimizing an autoencoder for rate-distortion performance is formally equivalent to some formulations of VAEs.

Theis et al. [17] (2017) expand on the work of rate-distortion optimization [28] by presenting a deeper and more expressive autoencoder architecture. They explore new quantization techniques and quantitatively evaluate different techniques to avoid the quantization problem of stopping the propagation of gradients. They conclude that their proposed solution: using a pass-through gradient from the decoder to the encoder performs better considering the generated approximations during training at the bottleneck. In contrast to the previous work on autoencoders, the authors train and test their network on high-quality images, instead of training on small images. This work surpasses all the previous works, including the work by Toderici et al. [3] for a fixed bitrate, closely matching the results of the codec JPEG2000 even for high-quality images. However, it still has the disadvantage of needing one network per bitrate target, thus requiring the training and storage of a high number of models to achieve different compression ratios. The article discusses a small change to help in this problem by considering a parametrization of the loss function so that each bitrate trade-off can be trained from a pre-trained coarse model.

Johnston et al. [14] (2017) continue the research on the variable bitrate compression framework proposed by Toderici et al. [3] substantially improving the obtained results. The most important change is the introduction of a spatially adaptive bitrate based on the entropy model of the latent representation, which allocates more bits of the compression to areas of the image with higher entropy. However, albeit the entropy of the compressed representation is taken into account to do bitrate allocation, it is not directly used during optimization, and thus does not provide feedback to the network for optimization as the rate-distortion optimization frameworks do. For training, the authors adopt a new weighted **loss function** based on the SSIM metric [26], with the objective of optimizing the quality of the reconstructions for a perceptual model which better resembles the human eye. They also use a technique called **hidden state priming** which allows a trade-off between computing power and better results by doing a run of the recurrent network where the output is discarded but the hidden state is kept. The final result is obtained by starting the network with the previously calculated hidden state, which the researchers show to increase the results of compression. The introduced improvements result in an impressive increase in the compression performance, making this network competitive with the WebP codec, and even surpassing it in some cases, when evaluating with the MS-SSIM metric. As had never happened, the research using DL-based methods for image compression establishes a result comparable with state-of-the-art methods in the field.

The research in adversarial training for image compression surges in 2017 with the work of Santurkar et al. [62], which propose a hybrid GAN-VAE framework with an intercalated optimization procedure where the decoder is first trained using an adversarial loss from the discriminator. The decoder is then fixed and the encoder is trained to minimize the reconstruction loss of the whole network, providing a stable training procedure, which is usually the main difficulty in adversarial training. The resulting network, however, fails when applied to higher resolution images, and only achieves competitive results in evaluations with smaller samples.

Soon after, Rippel and Bourdev [1] (2017) present the WaveOne codec, also based on adversarial training. The authors extend a deep convolutional autoencoder by introducing several innovations: pyramidal analysis for feature extraction, adaptive coding, and code length regularization. However, their main contribution is the proposal of a new optimization framework for adversarial training that combines both reconstruction loss and adversarial loss simultaneously, but dynamically balances both gradients in order to enforce the stability of the training. Their proposed solution works well enough that they present trained models for low bitrates which can be applied to full resolution images. Furthermore, the hierarchical feature extraction means that the model creates better overall representations of the initial image when compared to sequential convolutional models. Finally, the code length regularization technique ensures a way to reduce the entropy during training so as to ensure the target bitrate of the model (still, they do not completely model the entropy of the compressed representation). This solution presents exceptional results, surpassing most traditional codecs and all deep learning methods previously proposed by a fair margin, at any compression ratio, and in both RGB and YCbCr color spaces, when measured in MS-SSIM. However, there is one clear problem of this compression method, that is especially problematic in low bitrates: the adversarial training intro-

duces hallucinatory effects in the images when too much information is to be inferred by the network, which is undesirable in image compression schemes.

Very recently, Ballé et al. [4] (2018) extended the rate-distortion optimization approach for fixed rate compression. They propose a new density model to estimate the entropy of a representation by fitting a parameterized Gaussian distribution to the latent representation, whose parameters are sent in the bitstream as side-information. The parameters are inferred by stacking an additional autoencoder to predict the variance of the Gaussian distributions (which they call the scale hyperprior). The predicted variances are equally quantized and coupled to the main representation. This proposed method is much simpler than the adversarial model previously discussed, both from an architecture standpoint, and from a training difficulty perspective, and still greatly improves on the obtained results in both MS-SSIM and PSNR metrics. In fact, their results are, for the first time, comparable to state-of-the-art methods such as Google's *WebP* and *BPG* (`bellard.org/bpg/`) measured in PSNR, and better than all codecs when using the MS-SSIM perceptual quality metric.

2.5.2 Video Compression

When researchers identified image compression as one area with interesting applications for deep learning, it was also obvious that video compression could benefit due to the clear similarities [48]. However, the investigation in video compression using DL lagged behind image compression, with scarce published research. As video and image compression are related tasks, it would be expected that they progress in parallel. Thus, an important process in the research was understanding why, given the advances of DL in both image compression and video processing, this problem still persists.

Albeit being related, video compression is a harder task compared to image compression due to the additional temporal dimension. The amount of data that needs to be handled in video compression is higher, as there is a much larger space of possibilities in which redundancies within the video can be found and exploited. Hence, efficient video compression (in compression ratio) requires more computational resources and time compared to image compression. If a proposed scheme is not properly devised and optimized, both these constraints may increase beyond what is acceptable for a compression scheme. Particularly, the step of decoding the video must be executable in real-time, as the video is usually decoded during playback. So, the difficulty of video compression is to create coding and decoding schemes that can compress videos efficiently, while limited to reasonable computational resources and runtime.

On the other hand, the recent video codecs (*e.g,* H.264/AVC [7], H.265/HEVC [18]) focus on predictive architectures to tackle this issue. This type of architecture has been improved by decades of research, and by various teams with countless hours of work [48]. By looking at the inner workings of a recent video compression architecture such as H.264/AVC, it is possible to understand how it builds incrementally upon the previous architecture (H.263) [52]. The result of this continuous research, across many years, is new compression schemes that are extremely well optimized both in runtime requirements and in the produced results.

As far as was understood during the investigation, the lack of research in video compression can

be attributed mainly to two factors: the increased difficulty of the task, and the greatly optimized traditional architectures that new schemes must be compared against.

With that said, there are some works that were identified as closely related to the task of video compression, in the area of video processing, due to the proximity of the tackled problems. Srivastava et al. [63] (2015) show that RNNs can be used as spatio-temporal networks to improve video recognition tasks by extracting video latent representations and identifying the actions in the latent space. Additionally, the authors test if these representations are accurate enough to reconstruct a video sequence from the inputs, and they do obtain moderate success (no concrete evaluation is presented). This motivates the idea that recurrent networks may be adapted to the task of video compression (as at the time already shown for images [2]).

Shi et al. [59] (2016) propose a network architecture with an adversarial structure that is used to do super-resolution (up-sampling) of videos in real time, with notable results in the improved quality of the videos obtained. This research shows that adversarial training is an important technique for video processing and, motivated by the results of Rippel and Bourdev [1], that it should also be explored for video compression.

Lotter et al. [64] (2016) present an architecture that is used to do frame predictions of small video sequences. The authors achieve considerable results in one-frame prediction windows showing that networks can adequately explore temporal redundancy in images and even capture some of the dependencies between sequential frames, as well as having potential usage in a prediction-based architecture.

In late 2018 two video compression methods were proposed using DL-based methods. Both works were made available to the public on the final stretch of the thesis after all the research work was finalized, and most of the dissertation finished. For completeness, the works are presented here, along with a brief discussion, as they are related work. However, it should be noted that none of the works were available during the execution of the thesis. In the remaining chapters of the dissertation, both works will not be considered, as the research is presented from the available perspective during the execution of the thesis.

The two presented compression schemes diverge from the current approach to image compression. Although video compression can be seen as a natural extension to image compression, the state-of-the-art DNN architectures for the latter are not directly applicable to higher-dimensional data such as video: they can only be used to process spatial information, whereas video is a form of spatio-temporal data. To circumvent this, both rely on similar prediction architectures as used for traditional video compression. Wu et al. [65] address video compression as an interpolation task using two networks: one for image compression; another for interpolation, guided by the optical flow or block motion information. The idea is to exploit temporal redundancy between consecutive frames by learning a continuity model for interpolation. Although this method yields results only slightly below H.264/AVC, it has a significant computational overhead and is very limited in the compression ratios that it can produce; additionally, frames are interpolated instead of compressed, which may result in videos that to do not completely correspond to the original source. In contrast, Chen et al. [66]

exploit temporal information through a voxel-based prediction module, also achieving performance slightly below H.264/AVC. Their prediction module uses a spatio-temporal dependency model to predict subsequent voxels, with the residuals being then stored for transmission. However, prediction must be used during both encoding and decoding, involving a significant computational burden which questions the ability of the model to run in real-time as necessary for video codecs.

2.6 Summary

This chapter provides the necessary context to understand the image and video compression problems, from both a traditional and a DL-based perspective. An initial overview of Deep Learning (DL) is provided, with emphasis on the relevant techniques for image and video compression, which are then used throughout the dissertation. It is followed by a brief explanation of the current (traditional) approaches to both image and video compression. The latter is focused on motion-estimation video coding schemes.

The chapter is finalized with a thorough review of the existing DL research in both image and video compression. For image compression, several articles are presented in chronological order and the contributions discussed until reaching the state-of-the-art. For video compression, a possible explanation for the lack of research is given (complexity of the problem and highly optimized existing solutions), followed by a review of works that are closely related to video compression.

3

Understanding Image Compression

Contents

3.1 Motivation

Chapter 2 presents both the theoretical background and the research landscape, mostly focused on the current work in image compression. As a reminder, the lack of research in video compression, lead to the investigation of image compression techniques to serve as the basis for this thesis. However, while some of these techniques are intuitively suited for image compression, they may or may not be usable in the context of video compression, which, in the end, is the desired application. Unfortunately, most of the state-of-the-art research does not include open-source versions of the employed architectures (with rare exceptions, such as Ballé et al. [16] that published partial implementations of the used techniques). Thus, one important step to prepare the research that was conducted in this thesis regarded the implementation, reproduction and evaluation of important works in image compression - the focus in this chapter. A considerable portion of the thesis is therefore dedicated to this work, namely by implementing the techniques for image processing and compression, in order to evaluate their usefulness in video compression, instead of relying in theoretical guesses. Particularly, four distinct landmark works were chosen, that contributed to the advancement of the image compression field, and in which the applied techniques had potential for video compression.

Before introducing the architectures, a formalization of the image compression problem is presented, alongside a conceptual framework which is used to simplify the discussion by detailing the functioning of DL-based image compression works and introducing the common concepts to all works. For each remaining section, a different architecture is presented, with a discussion of the article that proposed it, the importance of the techniques and why it was chosen to be implemented based on the potential applications in video compression. The works are, in order, a Convolutional Autoencoder [2] (Section 3.3), a Convolutional Recurrent Autoencoder [3] (Section 3.4), a Generative Adversarial Network [1] (Section 3.5) and a Variational Autoencoder [4] (Section 3.6). This chapter is finalized with a comparison between all the implemented architectures and some remarks on the quality of the presented implementations (Section 3.7). This chapter also serves as a comprehensive introduction and motivation for the different video compression architectures proposed in Chapter 4, through summarized explanations of the techniques included in the proposed architectures.

All the implementations were written from scratch using **PyTorch 0.4**, and are available at a public repository. In what concerns currently available open-source implementations, it constitutes the first large scale repository of image compression architectures, techniques, and evaluation metrics. All the code is hosted in GitHub [1] with a permissive MIT license. A more detailed explanation can be found in Appendix B.

3.2 Compression Framework

The presented conceptual framework is a formalization of the image compression problem using a compressive autoencoder, inspired by the work of Toderici et al. [2]. It is, in practice, a subproduct of the video compression problem (the objective of this thesis).

[1] https://github.com/jorge-pessoa/pytorch-image-compression

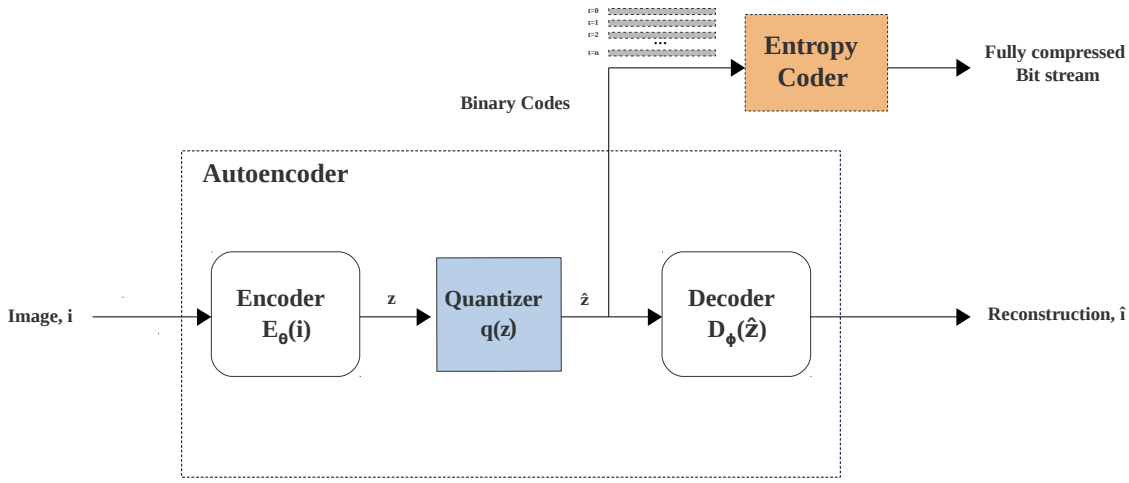


Figure 3.1: Overview of the interaction between the four components of the compression method during encoding and decoding of an image for fixed rate compression.

To elaborate, the compression framework is constituted by 4 components: a projection transform, a quantization step, an entropy coder, and a synthesis transform. The entropy coding step is always external to the network and will not be part of the definition of a network unless when made relevant. The rest of compression is translated to an autoencoder model: the encoder module serves as a projection transform and the decoder module as the synthesis transform; likewise, the quantization method is applied at the bottleneck of the autoencoder and the entropy coding to the output of the bottleneck. Each module is responsible for one step of the compression pipeline: the encoder projects the input image into a representative latent space exploring the spatial redundancy in the image. The bottleneck, which includes the quantization step, enforces the information loss in the image, necessary for lossy compression. The entropy coding is applied right after quantization and explores the existent coding redundancy in the latent representation. Finally, the decoder learns a transformation of the quantized latent representation back to the image space, reconstructing the original image with as much perceptual fidelity as possible. Note the clear similarities between each of these components and some techniques of the traditional algorithms, such as compressing the images not in their original space but in a transformed space, and enforcing information loss by quantization. A representation of this compression scheme can be seen in Figure 3.1

Formally, the encoder, decoder and the bottleneck form an autoencoder model, that creates compressed representations of an image $i \in \mathcal{I}$, where $\mathcal{I}$ is the space of all images in the expected format. For the example of RGB images with one color byte per pixel: $\mathcal{I} \in [0, 255]^{3\times W\times H}$ where W is the width H is the height of the image. The trainable encoder function $E_\theta : \mathcal{I} \to \mathcal{Z}$ transforms the original image i into a latent representation $z = E_\theta(i)$, where θ represents the trainable parameters of the encoder. The quantized sequence is created at the bottleneck layer by a quantization technique $q(z) : \mathcal{Z} \to \mathcal{S}$, where $\mathcal{S}$ is a finite length codebook, that transforms z into a quantized representation $\hat{z} = q(z)$. The decoder $D_\phi : \mathcal{S} \to \mathcal{I}$, with trainable parameters ϕ, reconstructs the image $\hat{\imath} = D_\phi(\hat{z})$, so that $\hat{\imath} \approx i$. The three components can be composed in sequence to formalize the autoencoder architecture that

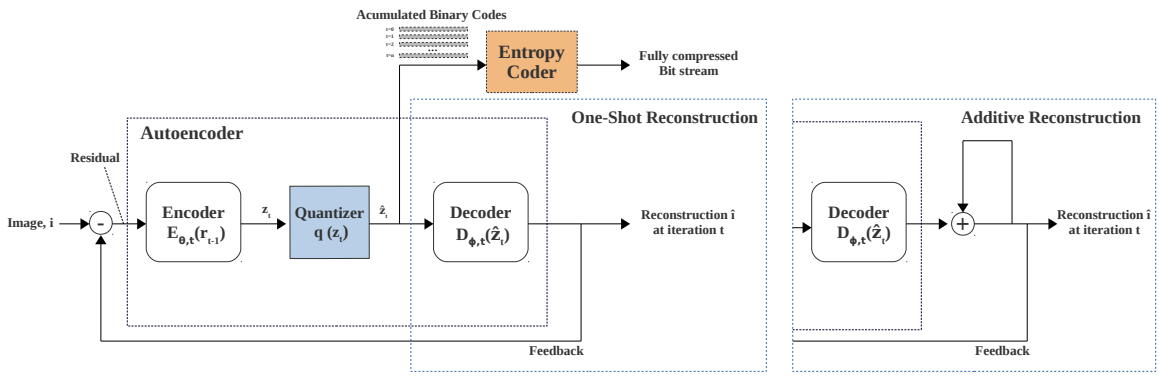


Figure 3.2: Overview of how the four components of the compression method interact during encoding of an image for variable rate compression methods. The entropy coder is only applied after the last iteration of the autoencoder. The components inside the two blue dashed rectangles are replaceable, but only one can be used at the same time and they demonstrate different reconstruction methods [3].

serves as a base for the compressive network $N_{\theta,\phi}$

$$\hat{\imath} = N_{\theta,\phi}(i) = D_\phi(q(E_\theta(i))). \tag{3.1}$$

This formulation is restricted to the case where image compression requires exactly one iteration of the network to attain the final result, *i.e.*, a fixed rate compression scheme. Hence, under this formulation, to allow for multiple compression ratios (as is necessary to plot rate-distortion curves), several models need to be trained with different parameters θ and ϕ, one for each target bitrate.

In an alternative formulation, a single network can recurrently be used for different bitrates, depending on the number of times the input passes through the network – this type of networks are referred to as variable rate compression schemes. The formalization of the variable rate compression methods is slightly different: the compressed image is obtained by applying the same components recursively until the desired compression rate is achieved and concatenating all the produced quantized representations for the final compressed image. The methods use a residual approach to the compression where r_t is the residual (difference) between the original image i and the target output at iteration t. The output at iteration t, $\hat{\imath}_t$, of the modified architecture, can be expressed as

$$\hat{\imath}_t = N_{\theta,\phi,t}(r_{t-1}) = D_{\phi,t}(q(E_{\theta,t}(r_{t-1}))), \tag{3.2}$$

where $r_0 = i$. Note that the quantizer is a stateless component that does not depend on the iteration number and the entropy coder is a separate network that is not part of the autoencoder for recursion purposes. The latter is only used to losslessly compress the final representation, resulting from the accumulation of the codes that were produced by the quantizer. The inverse transformation is therefore applied to the entropy coded representation to retrieve the quantized sequences before decoding. The network either predicts the full image at each iteration known as **one-shot** reconstruction, or the remaining residual at each iteration and the residuals are added together to obtain the reconstruction, known as **additive** reconstruction. The autoencoder must be trained accordingly to the reconstruction method, to either predict the full image or the residuals. However, one-shot reconstruction is

only possible in networks that can store context, as otherwise, the different iterations will not produce increasingly better results by doing the same operation.

Figure 3.2 depicts how the four modules are structured and interact to encode an image in a variable rate scheme. Additionally, it showcases two different reconstruction methods that are interchangeable: one-shot reconstruction and additive reconstruction. The four modules can be used to reconstruct the image in different ways by changing the output target.

Of the four architectures that are to be presented, the first two (Sections 3.3 and 3.4) are variable rate compression methods. The last two (Section 3.5 and 3.6) are fixed rate compression methods.

3.3 Simple Convolutional Networks

3.3.1 Architecture Overview

The first studied network was introduced in the earlier work in image compression by Toderici et al. [2] which presented for the first time variable rate compression methods. This network is the simpler formulation of the compressive autoencoder, using a deep convolutional encoder and decoder, as described in Section C. The encoder $E_\theta(r_{t-1})$ uses a set of convolutional layers with downsampling strides and a non-linearity before the bottleneck to produce the bitstream. The decoder applies transposed convolutions in symmetric order with upsampling strides to recover the image.

For the compression strategy, the presented autoencoder uses a two-fold bottleneck strategy to enforce information loss: the dimensionality of the latent representation is highly reduced compared to the input image, and the latent representation is quantized using a direct binarization technique (that is proceeded by the application of a hyperbolic tangent non-linearity to shift the output of the encoder to the $[-1, 1]$ range). Any negative number is directly rounded to -1, and any positive number or zero is rounded to 1. The authors note an important problem with quantization that had to be solved in order to train the compressive autoencoder: this procedure invalidates most of the gradients of the parameters θ of the encoder during the backpropagation of the loss from the decoder, due to the quantization. To circumvent this, they propose a relaxation of the quantization technique during training by applying a stochastic quantization method based on a classical result from quantization theory [57]. They note that high-rate quantization noise/error is well approximated by additive noise, with uniform distribution in the quantization interval. Thus, the rounding is replaced by adding uniform noise in the interval $[-\frac{1}{2}, \frac{1}{2}]$ to the latent representation, approximating the effect of quantization with a step-size of 1, without zeroing the gradients going through the bottleneck. Albeit other methods of relaxation have later been proposed (e.g. gradient passthrough by Theis et al. [17]), the relaxation by adding uniform noise has shown to be the prevailing method for training [1]. The additive noise quantization shows additional advantages as demonstrated by Ballé et al. [4] which are discussed in Section 3.6.

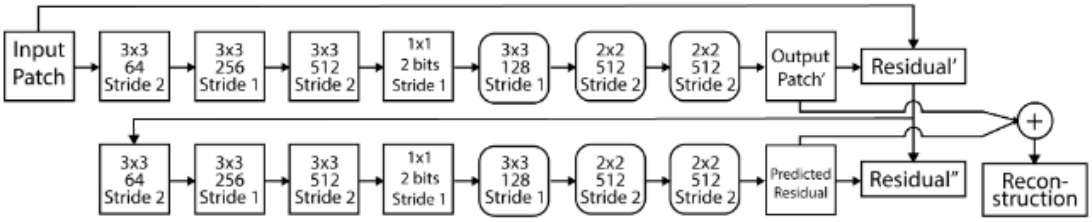


Figure 3.3: Representation of two iterations of the CNN, extracted from the work Toderici et al. [2], with all the hyper-parameters of the used architecture. Note the propagation of the residual from the output of one iteration to both the input of the next iteration and the reconstruction of the output, by using additive reconstruction.

3.3.2 Importance to image and video coding

The dimensionality bottleneck, the quantization that can be backpropagated and the variable rate frameworks were vital contributions that *kickstarted* the research in image compression, as these techniques were unheard of in DL-based solutions. The problems that they tackle are clearly situations which video compression will also have to deal with: the compression bottleneck and trainable quantization are mandatory for any full compression scheme; variable rate compression may be useful for flexible image or residual compression methods that may expend more or less bitrate to attain a target quality while only requiring a single model. The usefulness of the techniques for image compression (and the fact that they are used in most subsequent models) make the convolutional autoencoder a mandatory implementation, which also serves as the baseline for the evaluation of more complex methods.

3.3.3 Implementation details

For implementation purposes, the network's architecture is replicated from the hyper-parameters provided in the source article [2]. Figure 3.3 depicts two iterations of this network, with the corresponding kernel sizes, number of filters and stride. The implementation of the techniques is simple and straight-forward, and the source article provides enough context in order to recreate the proposed work. The network is trained to optimize the reconstruction loss between the input x and the output y of the network, by minimizing the MSE loss function (see 2.2.3). The compression rate of the model is guaranteed by the discussed compression methodology. The network is trained on $2,5 \times 10^5$ 32x32 pixel crops of the ImageNet [67] dataset.

The network is implemented as a variable bit rate compression scheme, so the inputs for each iteration are of residual nature and weights are shared between iterations. The reconstruction of the image is obtained using additive reconstruction as the network does not store context, and one-shot reconstruction would not work in this case. Due to the variable bitrate scheme, only a single model is trained for this network.

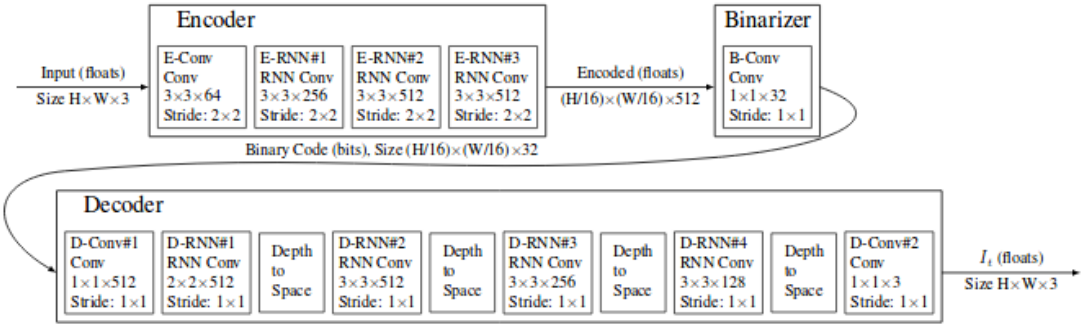


Figure 3.4: A single simplified iteration of the RCNN, extracted from the work Toderici et al. [3], with all the hyper-parameters of the used architecture. The proapagation of the hidden-state is not represented in this figure.

3.4 Recurrent Convolutional Networks

3.4.1 Architecture Overview

The second discussed architecture is the RCNN network proposed in Toderici et al. [3], which follows and improves upon the work of Toderici et al. [2]. The architecture is formulated in a similar fashion to the one of Toderici et al. [2], but makes use of the ability of RCNNs to propagate information between executions of the same network through its hidden state. The encoder $\mathrm{E}_\theta(r_{t-1})$ is composed of multiple convolutional recurrent layers. However, to avoid the implementation of recurrent transposed convolutions, the decoder uses a technique first introduced for super-resolution tasks: a depth to space or pixel shuffle transposition [59] followed by a convolutional recurrent unit. A depth to space unit with an upsampling factor of 2 reorganizes C channels of width w and height h in $C/4$ channels of $2w$ width and $2h$ height, effectively transforming channel depth in increased spatial resolution. The result of the two operations is an upscaled tensor that can be used to replace transposed convolutions with upsampling strides while having the advantage of avoiding the usual checkerboard artifacts of transposed convolutions [60]. Thus, the decoder $\mathrm{D}(z)$ is composed of alternated convolutions and depth to space units. Each recurrent layer has a hidden state that is propagated to the same layer for the next iteration of the variable rate compression strategy, with the number of iterations being determined by the bitrate.

Due to the increased ability to propagate information between states, the simpler one-shot reconstruction method for variable bitrate compression is introduced, in which the network predicts the full image in each iteration. The techniques used for quantization and binarization, as well as the relaxation of the quantization for training, are the same as presented for the previous convolutional architecture.

3.4.2 Importance to image and video coding

The work by Jonston et al. [3] is particularly important, especially when studying techniques to be applied in video compression. In the case of this architecture, the recurrent capability is used to propagate information between iterations of varying bitrate; however, this is not the only application

that can be devised for such architecture. As explained in Section G, RCNNs are one of the possible architectures for spatio-temporal autoencoders, and are commonly used for tasks that require spatial and temporal reasoning such as video semantic segmentation [68]. By propagating state between iterations, RCNNs are able to store and extract information in any dimension, including the processing of sequential data. In the case of spatio-temporal data such as video, this type of architecture combines the memory capabilities to capture temporal dependencies between frames, with convolutional operations which allow for the exploitation of the spatial redundancy existing in each frame. RCNNs are obvious candidates for any task in video processing. With this in mind, the need to implement convolutional recurrent units was a certainty in the long term, as these units had to be tested, which made the implementation of this image compression architecture convenient and important. The depth to space unit is also an interesting technique to avoid checkerboard artifacts which can be used in future video compression architectures. The remaining techniques applied, from the variable bit rate compression framework to the quantization and binarization, were already needed for the previous convolutional architecture, simplifying the implementation process.

3.4.3 Implementation details

There are two commonly used recurrent units in literature: LSTMs and GRUs. Both types of units were implemented to be compared. Additionally, RCNNs can work in both an additive reconstruction or one-shot reconstruction scheme for variable bit rate compression. For completeness, a model was trained for each combination of reconstruction method and recurrent unit in order to assess the best performing formulation of this architecture. Thus, the total number of models for this section is four: GRU one-shot reconstruction, GRU additive reconstruction, LSTM one-shot reconstruction and LSTM additive reconstruction. Each model is implemented by setting the hyper-parameters according to the source article by Toderici et al. [3], and the configurations for each layer are detailed in Figure 3.4, also extracted from the article. As in the convolutional autoencoder, the four models are trained to optimize the reconstruction loss between the input and the output of the network, by minimizing the MSE loss function. The training is done on $2,5 \times 10^5$ 32x32 pixel crops of the ImageNet [67] dataset. The compression rate of the model is guaranteed by the same dimensionality bottleneck and quantization strategy as in Section 3.3.

3.5 Generative Adversarial Networks

3.5.1 Architecture Overview

The third implemented architecture is a Generative Adversarial Network based on the work of Rippel et al.[1]. As explained in Section E, GANs use a system of two networks, a generator and a discriminator, that compete against each other. The discriminator learns to differentiate between real and fake (compressed) images, while the generator learns to approximate the probability distribution function of the reconstructed images to the originals found in the training dataset.

The most significant contribution presented in this work is the adversarial training framework,

which combines an adversarial loss with a reconstruction loss, and stabilizes the training by balancing the gradient of both losses. In this case, a reduction of compression artifacts is expected when producing lower bitrate images, since the network infers the missing information from the expected features present in uncompressed images.

However, adversarial training introduces instability in training [1, 62] due to the continuous feedback loop between the discriminator and the generator. In a training iteration, the discriminator minimizes the loss of distinguishing artificial and real examples. This change forces the generator to improve its approximation of the real data, which, in turn, causes an increase in the loss of the discriminator. Thus, minimizing the loss of the discriminator increases the loss of the generator and minimizing the loss of the generator increases the loss of the discriminator. This loop is difficult to keep stable as one of the networks can become overwhelmingly good at its task, and the other will not be able to recover, especially when the adversarial loss is not the only optimization target. To fight this, Rippel and Bourdev [1] alternate between optimizing both the networks or just the worst performing network, depending on the relative performance between them, so that the disparity between the networks never increases beyond an acceptable threshold.

The resulting architecture is considerably more complex than the previously introduced networks due to stabilization techniques and the implementation of both a discriminator and a generator.

3.5.2 Importance to image and video coding

Rippel and Bourdev [1] demonstrated for the first time that adversarial training can successfully be used for image compression. Still, the adversarial loss was not the only technique employed; from the innovations in the work, the following stand out for their possible applications, not only in image compression but potentially in video compression:

- Multi-scale architecture for the compression network and the discriminator, with pyramidal feature extraction
- Bucket quantization focused on bit-planes, with gradient pass-through
- Adaptive arithmetic entropy coder using contextual bit-plane information
- Encoder output regularization to enforce reduced entropy of the latent representation

During the research for appropriate techniques that could translate to video compression, this work was considered particularly important, mainly due to the suitability of the adversarial process to the task of compression. Notwithstanding, the other proposed techniques also have merit to be further investigated. The pyramidal extraction architecture is particularly useful to extract dependencies at different scales in the input data, which is commonly seen in temporal data (temporal redundancy can happen in sequential frames or over long-sequences of frames). The introduced bucket quantization and entropy coding scheme achieve higher entropy coding performance compared to all previously introduced methods (excluding rate-distortion methods). As a result, even though the architecture

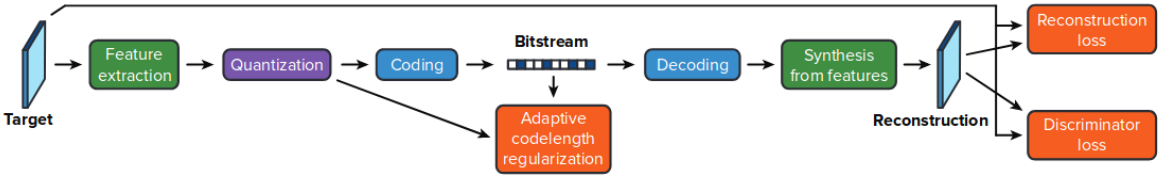


Figure 3.5: Overview of the autoencoder architecture, entropy coding and losses, used in Rippel and Bourdev [1], for adversarial training. The illustration is extracted from the source article. Note the additional adversarial loss compared to previous formulations.

was complex and used multiple new techniques, it was included in the choice of works to study and implement.

3.5.3 Implementation details

The WaveOne (as the compression scheme is named in Rippel and Bourdev [1]) is composed of a multi-scale encoder and decoder for which a general representation (extracted from the source article) is presented in Figure 3.5. Albeit not represented in the figure, the output of the network is the result of the joint processing of the output from each scale. The discriminator used for adversarial training has a similar multi-scale architecture which can be seen in Figure 3.6. The hyper-parameters of the compression network were extracted from the description in the original article [1] (the ones that were disclosed, as detailed further below). The result of the interscale encoder of the compressive network is subject to a penalty that is included in the loss function to enforce a lower entropy in the latent representation. The compressed bitstream goes through an adaptive entropy coder, which uses a trained classifier to estimate the probability of a single value given the spatial context of its filter. The source article describes a bucket quantizer with a variable number of buckets B, for which we set $B = 2$. The probabilities given for each symbol are used by an arithmetic encoder for the final coding stage.

The final loss that was used to train the network is more complex than the previously presented ones, and can be summarized as:

$$\mathcal{L}(i, \hat{\imath}, z, D) = \text{MS-SSIM}(i, \hat{\imath}) + \alpha \mathcal{L}_a(i, \hat{\imath}, D) + \mathcal{R}(z), \tag{3.3}$$

where MS-SSIM is the function discussed in Section 2.3, i is the input of the network, $\hat{\imath}$ the output of the network, and z the latent representation of the compressed image. $\mathcal{L}_a$ is the adversarial loss function for a generator as presented in Section E. $\mathcal{R}$ is the regularization term which is applied per position of the latent representation. This penalty enforces lower entropy representations when considering a position and its context, so that it can be more efficiently entropy coded and the size of the latent representation can be controlled. The network was trained on 10^6 256x256 pixel crops extracted at random from images of the YFCC100M [69] (Yahoo Flickr Creative Commons 100 Million) dataset.

As the other networks, an open-source implementation of this work was not made available by the authours of the research. The complexity of the network, aggravated by the lack of details in some

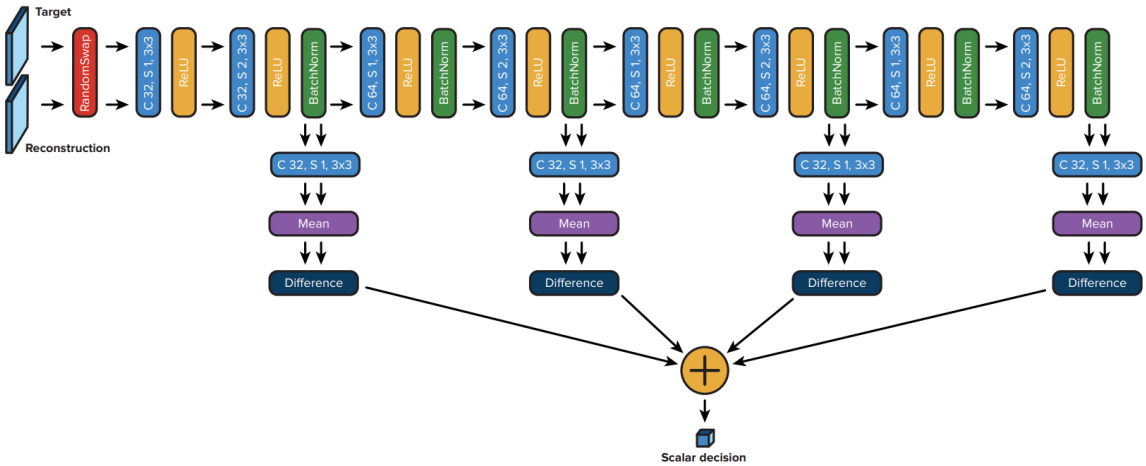


Figure 3.6: Multi-scale discriminator used in the adversarial training. The illustration is extracted from Rippel and Bourdev [1]

areas of the article, made the reproduction of this exact work and the originally obtained results very difficult. Even though the general architecture of the compression scheme is understandable from the article, the authors do not go into detail in several techniques, leaving a lot of guesswork to be done in replicating their work. The results presented for this network in Section 3.7 are considerably below the results achieved by Rippel and Bourdev [1], with a reduction of quality ranging from 25% in low bitrates to 5% in higher bitrates. The contextualize these results, the difficulties faced are explained in detail below.

Firstly, the authors propose a multi-scale composition (referred to as pyramidal feature extraction) of the encoder and decoder, and motivate its benefits; however, they never present a concrete implementation for the encoder and decoder in terms of hyper-parameter configurations (number of layers, layer configurations, how many multi-scale connections,*etc...*). Thus, the whole compressive network is guessed from the broad description presented in the article. The implementation used for evaluation uses 5 blocks of a convolution of kernel size 3 and a non-linearity (LeakyReLU with a leak of 0.2). The output of each block is sent forward and to a different scale. Each scale goes through an alignment block using downsampling. The aligned scales are added together and jointly processed to create the output of the encoder.

Furthermore, the article does not describe in detail the implementation of the stabilization method to balance the gradients of the reconstruction and adversarial loss. It states however that the losses are to be balanced according to a weighted norm of their gradient. For the presented implementation, this technique was achieved as follows: with $G_r(i, \hat{\imath})$ being the gradient of the reconstruction loss and $G_a(i, \hat{\imath}, D)$ the gradient of the adversarial loss given the input i, output $\hat{\imath}$ and the discriminator feedback D, the norm of each gradient is calculated per training batch by backpropagating the losses once and clearing the gradients. Then, a ratio between the two $R(i, \hat{\imath}) = \alpha(|G_a(i, \hat{\imath})|/|G_r(i, \hat{\imath})|)$ is calculated. The constant α is empirically determined based on the compression ratio of the network. The ratio is used to scale the weight of the adversarial loss in order to match the gradient of the reconstruction loss.

The entropy coding technique is the last element that is only briefly described, leaving the implementation details for further guessing. In the concerned implementation here, a logistic regression was used as the predictive unit. The context used to help the classifier predict each bit contains the bit values for each position surrounding the bit to be predicted. The logistic regression was trained with a set of compressed bitstreams after fully training the network, obtained by compressing 10^3 images of size 256x256 pixels.

3.6 Variational Auto Encoders

A Architecture Overview

The final implemented architecture is the work by Ballé et al. [4] using Variational Autoencoders (VAEs) with a scale hyperprior and rate-distortion optimization. It is the best performing DL-based method of image compression published so far. Surprisingly, it is much simpler than the previously presented multi-scale autoencoder with adversarial training [1], and the results are considerably better. The two-fold improvement (simpler and better performing) was a considerable landmark in image compression. Although the network is referred to as a VAE, it should be noted that the presented compressive autoencoder does not use the normal reparametrization trick of VAEs. Instead, the network is formally equivalent to a VAE due to using the additive noise approximation to quantization during optimization VAE [4, 28].

The formulation of this work is easily framed in the initial formulation of Section 4.3.1. The authors propose a simple convolutional encoder and decoder for E_θ and D_ϕ, serving as a projection transform and synthesis transform respectively, to a learned latent space. The bottleneck does not reduce the dimensionality of the latent space as all the previously presented works. Instead, an entropy model for the latent space is learned, per channel, during the training of the network. The entropy model estimation can be used to calculate an individual likelihood of each value in the latent representation. In turn, the probabilities can be used to estimate the entropy of the model, which is used to obtain the expected final size of the quantized latent representation after entropy coding. As the entropy model estimation is differentiable, it can be used to penalize the size of that representation, enforcing the size reduction at the bottleneck. This method, as the previous rate-distortion optimization methods, focuses on using an information bottleneck which controls a quantified amount of information going through the bottleneck, instead of simply reducing the dimensionality of the bottleneck before quantization. This ensures that the network can jointly optimize the reconstruction loss of the network and the size of the compressed representation, resulting in efficient representations in which the network optimizes the information that flows through the bottleneck to maximize the reconstruction quality.

To be more specific, the solution proposed by Ballé et al. [4] is presented in Figure 3.7. The encoder g_a extracts the latent representation of the image. To estimate the probability model of the quantized latent representation, $\hat{y}$, the authors propose a Gaussian distribution of mean 0 and parametrized standard deviance ($\hat{\sigma}$) which describes the distribution of each position of the latent representation. The standard deviation $\hat{\sigma}$ is inferred using the scale hyperprior: an additional stacked

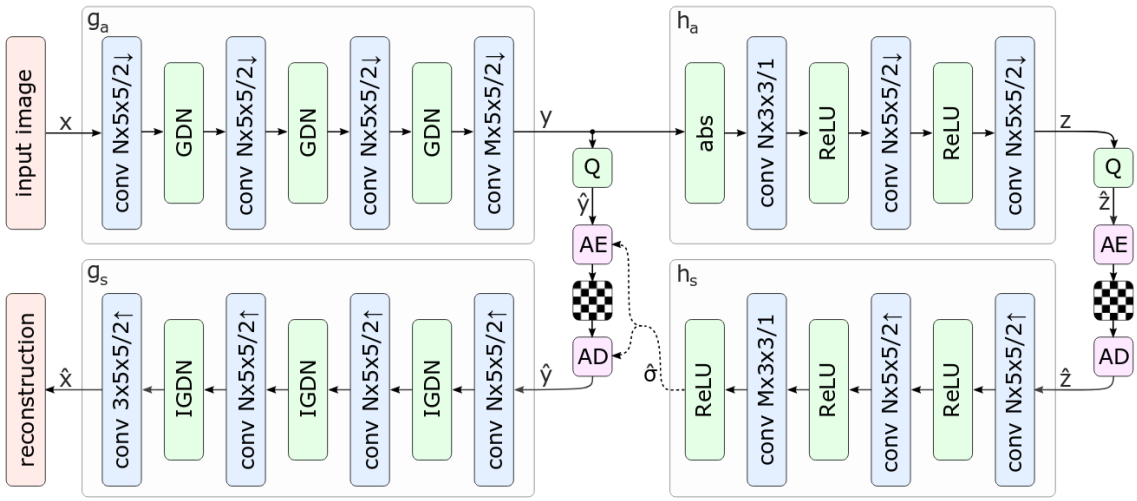


Figure 3.7: Stacked autoencoder architecture containing the main network and the hyperprior. Used by Ballé et al [4] to achieve state of the art results in image compression. Note the two bitstreams produced: one from the entropy coding of the image, and another from the output of the scale hyperprior. There is a causal dependency between the entropy coding of the output of the first autoencoder with the decoding of the hyperprior, however both due encoding and decoding the output of the hyperprior is available at that point of the process.

autoencoder (h_a and h_s) that is included in the network. Instead of simply predicting the variances, the hyperprior is also allocated a portion of the bitstream, $\hat{z}$, sent as additional side-channel information. The standard deviations are retrieved using the hyperprior decoder, h_s, and used to predict the probability model of the image latent representation. The image is reconstructed by the decoder, g_s, from $\hat{y}$. However, as the side-channel information is also sent as part of the compressed image, it is necessary to estimate it's size during training in order to jointly optimize both the compressor and the hyperprior. The entropy model for the hyperprior is obtained by using a general approximator of density functions in the form of a probability density function (pdf) described in Ballé et al. [4]. The pdf $p : \mathbb{R} \to \mathbb{R}^+$ is defined in terms of its cumulative distribution function (cdf) $P : \mathbb{R} \to [0, 1]$ ($P(x) = \int_{-\infty}^{x} p(y)dy$), as a general approximator of density functions. The two functions are defined as follows, for $0 < k \leq K$ ($K = 4$ for the purpose of the implementation)

$$
\begin{aligned}
P &= g_K \circ g_{K-1} \cdots \circ g_1, \\
p &= g_K^{'} g_{K-1}^{'} \cdots f_1^{'}, \\
g_k(x) &= h_k(H^{(k)}x + b^{(K)}), \\
g_K(x) &= \text{sigmoid}(H^{(k)}x + b^{(K)}), \\
h_k(x) &= x + a^{(k)} \odot \tanh(x).
\end{aligned}
\tag{3.4}
$$

During optimization, the quantization method is replaced by adding additive uniform noise to the latent representation, a method introduced in Section 3.3. The authors demonstrate that this approximation not only solves the problem of quantization blocking gradients but also has a positive impact in the rate-distortion optimization. More precisely, the stochastic effect of the uniform noise allows the entropy modeling of the latent representation to approximate arbitrary probability density functions, which was not possible otherwise. The improved results in density modeling with this technique justify

its usage, even after other quantization methods had been proposed.

B Importance to image and video coding

The information bottleneck is the main advantage of the rate-distortion optimization branch of research for image compression, and this work serves to explain how they work differently from the previously introduced networks. Although an earlier work could have been used for illustrative purposes, the architecture presented here serves two purposes: explaining how the rate-distortion optimization methods work in theory, and presenting the best performing method proposed for image compression using DL-based methods. The importance of this work in image compression should not be understated: it uses a simplified network (see Figure 3.7), and no additional techniques other than the rate-distortion optimization framework, to consistently beat the all the both the previously presented architectures, and the best performing traditional codecs in most of the metrics used in the original comparison.

In a similar way, the work presented here is highly relevant for video compression. First, it it supposedly the best performing architecture for image compression and does not exhibit the adversarial artifacts of GANs. Thus, if any image compression is needed in a video compression scheme, this is the network to use if the results can be replicated. Additionally, rate-distortion optimization is not only applicable to image compression. In fact, the information bottleneck is a flexible technique that can be used to formulate optimization frameworks that enforce the efficiency of a latent representation in terms of optimizing another term such as a reconstruction loss. It can potentially be used for residual compression, or even for a more direct video compression scheme that does not rely in predictive analysis (as will be proposed in Chapter 4). The estimation techniques used here provide the most accurate and flexible methods of entropy model approximation, which results in more efficient entropy coding and as such justify the additional complexity needed for their implementation.

3.6.1 Implementation details

The implementation of this architecture was easier than the previously described GAN, not only due to its simplicity but also because the structure of both the compressive autoencoder and the hyperprior are explained in detail. The entropy modeling techniques are also carefully explained, including the probability density function used for the hyperprior which is transcribed in Section 3.6. The overall structure of the stacked autoencoders (compressive network in the left and hyperprior in the right) is detailed in Fig 3.7, along with all the hyper-parameters used to implement the network. Additionally, a range encoder and range decoder were implemented from scratch in a native language in order to adapt to the format of the probabilities outputted by the model, so that the estimated sizes for the entropy coded latent representations could be confirmed using actual entropy coding.

The network was trained end-to-end with the presented rate-distortion framework on 10^6 256x256 pixel crops at random from images of the YFCC100M [69] (Yahoo Flickr Creative Commons 100 Million) dataset.

The approximated discrete entropy of the quantized latent representation can be interpreted as

Full name	Abbreviation	Section introduced
Convolutional autoencoder	CN	3.3
GRU One-shot reconstruction	GRU OS	3.4
GRU Additive reconstruction	GRU AR	3.4
LSTM One-shot reconstruction	LSTM OS	3.4
LSTM Additive reconstruction	LSTM AR	3.4
Generative Adversarial Network	GAN	3.5
Variational Autoencoder with Scale HyperPrior	VAE/H	3.6

Table 3.1: Naming convention used to refer to each presented architecture in the Results section

the information content of that representation (which in turn approximates the estimated size for an optimal encoding). It can be calculated using the estimated probability models for both the quantized image latent representation $\bar{z}$, and the quantized hyperprior latent representation $\bar{w}$

$$L(\bar{z}, \bar{w}) \simeq \sum_k -\log_2 \mathcal{N}(\bar{z}_k|0, \sigma_k^2) + \sum_j -\log_2 p(\bar{w}_j|\delta_j), \tag{3.5}$$

where $\mathcal{N}(x_i|\mu_i, \sigma_i^2)$ is the probability for element i of x, given a Normal distribution of mean μ_i and variance $sigma_i^2$, and $p(\bar{w}_i|\delta_i)$ is the probability for element i of x given the pdf described in Equation 3.4 and the estimated parameters during training δ_i.

The total loss for rate-distortion optimization combines the reconstruction loss using the MS-SSIM metric (see Section 2.3) and this entropy loss, using β as an empirically determined parameter which allows tuning the compression ratio, with the final form being

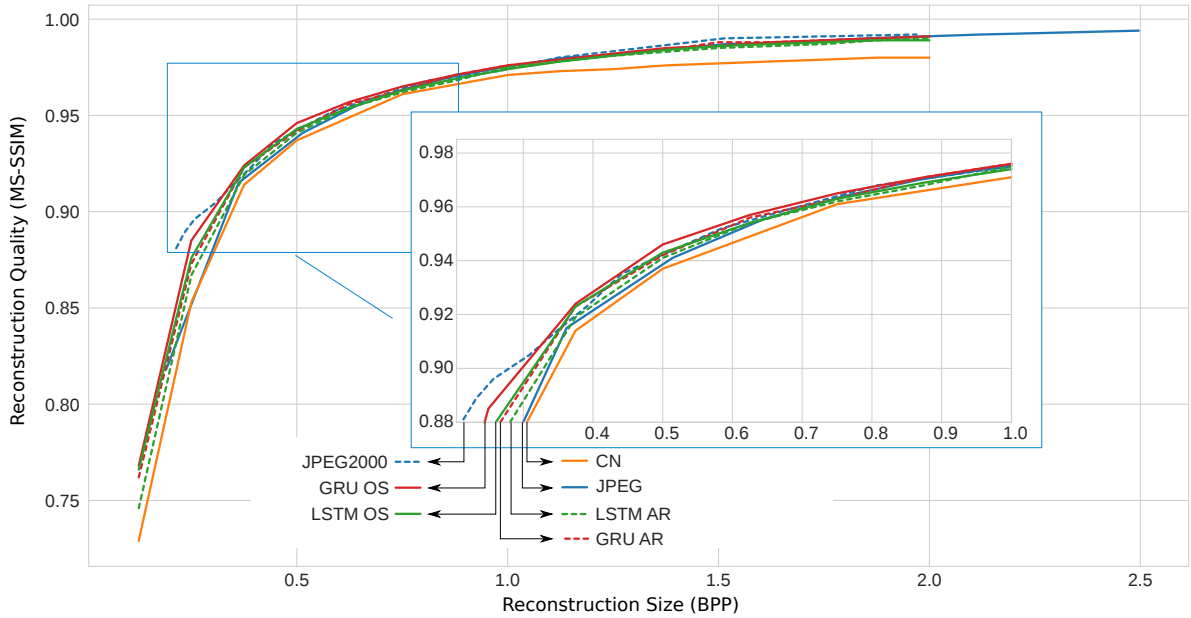
$$\mathcal{L}(i, \hat{\imath}, \bar{z}, \bar{w}) = \text{MS-SSIM}(i, \hat{\imath}) + \beta L(\bar{z}, \bar{w}). \tag{3.6}$$

Several models were trained for this implementation by varying β, in order to be able to plot the rate-distortion curve for this architecture.

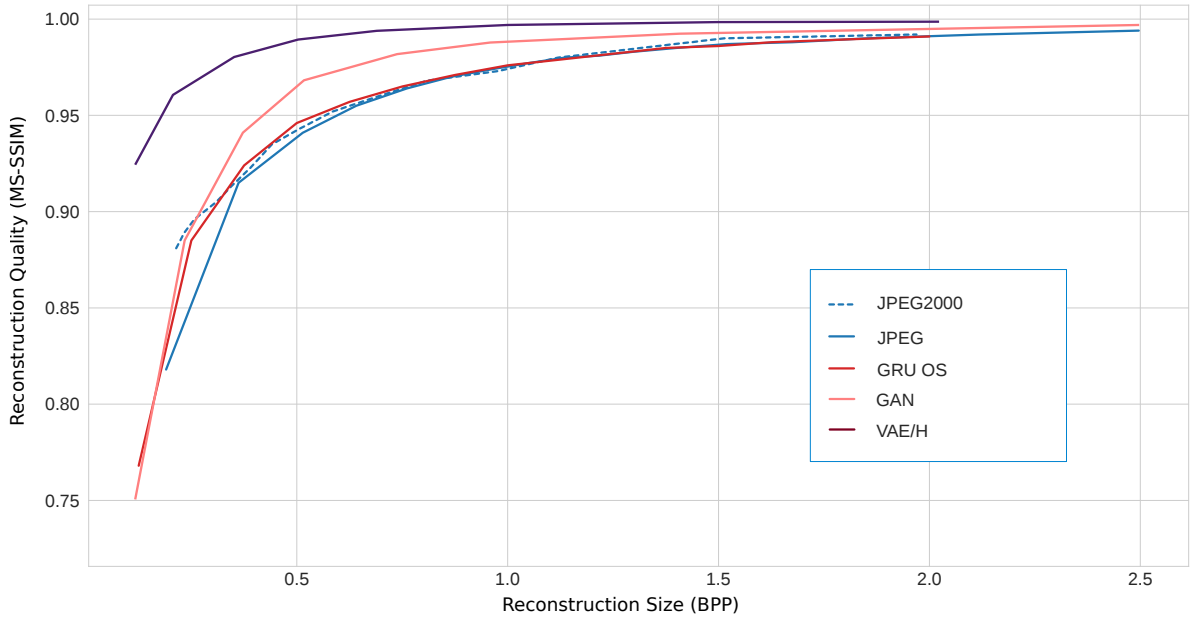
3.7 Comparison

The final step of the image compression research is a general comparison between the implemented architectures. While the main conclusion to withdraw from this comparison is to quantitatively evaluate the performance of the architectures, it will also be useful to compare the impact of the improvements between each implemented network and the usefulness of each network for specific tasks (*i.e.* one of the first two networks may be chosen for a specific task where variable bitrate compression is necessary). If any network is to be reused in a video compression architecture, the results that are relevant are the ones obtained by the implementation used here, and as such, being able to rank their performance is essential for choosing correctly. To simplify the discussion, Table 3.1 presents a naming convention used to refer to each network in both the presented graphs and the discussion.

The evaluation of the networks is done by testing the compression on the 24 images of the Kodak Lossless True Color Image Suite [70]. For each possible bitrate, all the test images are compressed



(a) Variable-rate reconstruction



(b) Fixed-rate reconstruction

Figure 3.8: Average rate-distortion curves measured using MS-SSIM for the networks presented in Table 3.1, and for JPEG and JPEG2000

and decompressed, and both the compression ratio and the obtained reconstruction quality using MS-SSIM are averaged to obtain a point in the graph for a certain network. The average rate-distortion curves are obtained by connecting the existing points for each network.

We separate the comparison presented in Fig. 3.8 in two graphs: one for variable rate compression and one for fixed rate compression. The former also contains the best performing variable bitrate network in order to provide context between both graphs. By presenting the networks in two separate analysis, the corresponding graphs are less cluttered and easier to analyze. Furthermore, we expect

	AUC (MS-SSIM)
JPEG2000	1.80298
GRU OS	1.80031
GRU AR	1.79756
LSTM OS	1.79656
JPEG	1.79407
LSTM AR	1.79287
CN	1.78031

Table 3.2: Area Under Curve (AUC) metric for the MS-SSIM curves obtained during evaluation as displayed in Fig. 3.8(a)

the fixed rate compression methods to perform much better when measured in MS-SSIM, as they use more complex architectures, and each trained model is optimized to maximize the MS-SSIM metric a certain compression ratio, instead of the MSE, making the direct comparison less meaningful.

Due to the proximity of the results attained for each method in variable rate compression and the higher number of curves, we also provide the Area under Curve (AUC) metric in Table 3.2. This metric is calculated for each network according to the average MS-SSIM curves and not on a per-image basis. The AUC is obtained by integrating the average curve over the bitrate axis. The AUC can be used in this case as all the schemes included in this comparison achieve exactly the same minimum and maximum compression rate, having the same length under the integrated axis (thus the area of the curve is directly comparable).

Both comparisons also contain the metrics for two traditional and commonly used algorithms: JPEG and JPEG2000. This serves to contextualize the quality of the results obtained by the neural networks, so that the displayed curves have meaning for the reader. The algorithms included are the most commonly used.

Within the first comparison, all the results are according to the expectation from the corresponding literature, within a margin of error of 5% relative performance. By analyzing the graph in Figure 3.8(a), it is possible to see that the worst performing network is the CN and the best performing network is the GRU OS, with both LSTM models also performing adequately. Above all, RCNNs demonstrate a clear advantage over the simpler CNN, especially as the bitrate increases and memory capabilities of RNNs become more useful. As the best performing variable compression method, the GRU OS network was used down the line in residual compression for a possible approach to video compression, in which the residuals were compressed a variable number of times until the desired target quality was reached; a deeper discussion of the usage is provided in Chapter 4.

In the second comparison, the GAN performs worse than the performance reported in the original article, especially for low bitrates, with the relative error to the reported results ranging from 25% in low bitrates to 5% in higher bitrates. The explanation for this, which was already alluded to in Section 3.5.3, is a combination of the complexity of the techniques used, the lack of detail in the source article (to allow for proper reproduction) and the challenges of adversarial training. The stabilization of this model was difficult to achieve and there is no guarantee that it resembles the actual method in the article. The effects of the instability of training are particularly evident in low bitrate scenarios, as

less information is available to recreate the output of the network. In this case, it was necessary to resort to lowering the weight of the adversarial loss on the training, which in turn explains the worse performance for the initial points. With this in mind, the adversarial training was deemed too difficult to stabilize and use properly in order to keep investigating further for video compression, but these results do not invalidate other interesting techniques proposed by Rippel and Bourdev [1] such as the multi-scale architecture.

In opposition, the VAE/H performs very closely to the originally reported performance (within a 2% margin of error), and achieves by far the best performance of all the tested networks. The network was easier to implement and the training process was straightforward, suggesting the suitability of this type of compression for more complex tasks. The overall great results from low bitrates to high bitrates achieved by the work of Ballé et al. [4] make it the obvious choice for any video compression schemes that require image compression. The rate-distortion optimization is a clever approach to the compression problem by forcing the network to learn the balance between the information flowing through the bottleneck and the optimization of a certain metric; thus, it makes it easier to approach any other compression task with simpler end-to-end integrated approaches. This image compression architecture is used in a video compression attempt based on image and residual compression with a prediction architecture. But more importantly, the rate-distortion optimization approach serves as a crucial base and inspiration to the final method presented as the solution to the problem in the thesis. In a final note, it also serves to show that complexity is not necessarily good in DL, and simpler methods may achieve better results as long as they are cleverly architectured.

3.8 Summary

This chapter focuses on theoretical and practical work in image compression. It begins with an explanation of the motivation behind researching image compression to propose a video compression scheme. The main driving factors presented are the lack of research in DL-based video coding to serve as a starting point for the thesis, and the similarity of the image and video compression problems.

In the remainder of the chapter, four distinct architectures for image compression using DL are studied and implemented. Two are variable bitrate compression methods: a CNN [2] and a RCNN [3]; and two are fixed bitrate compression methods: a GAN [1], and a VAE [4]. For each, an overview of the architecture is presented, followed by a discussion of their importance for both image and video compression, and the final implementation details.

The chapter is finalized by comparing the performance of the presented architectures in compressing the True Color Kodak Suite [70]. The RCNN achieves the best performance from the variable bitrate methods, and the VAE achieves the overall best performance, by far.

4

Learning Video Compression

Contents

4.1 Motivation

The proposition for this thesis was mainly motivated by the lack of research in DL-based schemes for video compression, and the success of such approaches in image compression. This dichotomy has been explained in more detail in Section 2.5.2. The main identified cause was the increased difficulty of video compression, when compared to image compression, due to the additional temporal dimension. As a result, it is a more computationally demanding task, with runtime requirements that further complicate the problem.

The current video codecs (*e.g.*, H.264/AVC [7], H.265/HEVC [18], see Section 2.4.2) are highly optimized for the task, posing another barrier for new compression schemes. They are based on decades of iterative research in motion-prediction/frame-estimation architectures, which has proved to be a great framework to exploit temporal redundancy in video. Thus, in order to be competitive with the existing architectures, the intuitive first approach to video compression was to adapt a proven architecture, using motion prediction, to be used with DL-based methods.

4.2 Motion Prediction Architectures

As previously referred, the initial pursued strategy for video compression takes inspiration on predictive video coding strategies. Hence, a motion prediction architecture was first investigated, by separating the video compression problem in three steps (see Figure 4.1): image compression, motion estimation, and residual compression.

One of the investigated solutions for this problem employed a motion estimation method that relied on a DNN for predicting future frames, given a small set of past information. In practice, such solution attempted to avoid storing motion vectors to increase the compression ratio. For frame prediction, a network was devised that inspired on the work of Canziani and Culurciello [71], consisting on a RNN architecture augmented with top-down feedback and lateral connections. The network used a short sequence of previously compressed frames in the initial iterations, to extract existing temporal dependencies and infer a new future frame. Even though initial results were promising, the frame prediction had two major problems: it could not generalize properly outside the training dataset, and the produced frames were generally blurry.

Another investigated solution was devised, again inspired on predictive video strategies (see Figure 4.1). Under such compression scheme, the frames are processed in smaller blocks of $n \times n$ pixels, where n was set to 32 for this implementation. In total, it uses four modules to achieve compression: a Frame Compression module for single frame compression, a Block Predictive Coding module for motion vector prediction of each block, a Block Residual Compression module to encode the residuals of motion estimation and a Block Absolute Compression module to code motion miss-predictions. Three of the modules are implemented using neural networks: the Frame Compression module is implemented using the image compression architecture presented in Section 3.6; the Block Absolute Compression module is implemented as similar architecture to the Frame Compression network, but using smaller kernels adapted to the block size; and the Block Residual Compression module was

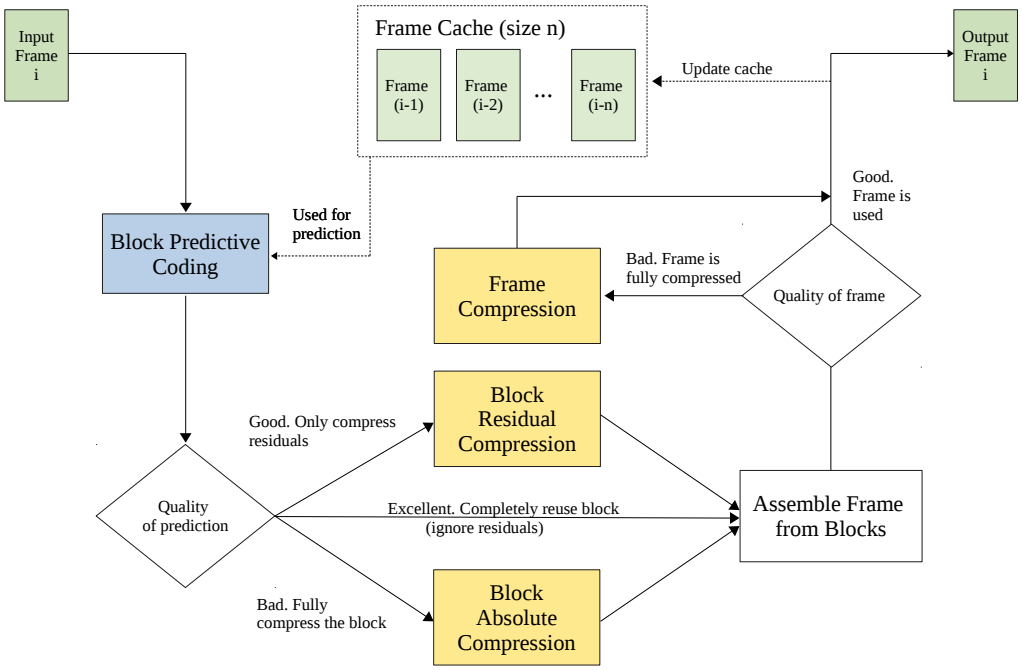


Figure 4.1: The general work-flow of the proposed architecture for video compression with DL using four modules: block prediction, block compression, frame compression and residual compression. Modules in yellow are implemented as neural networks.

implemented similarly to the Block Absolute Compression network, but using adequate normalization techniques for residual compression. The last module, for Block Predictive Coding, is implemented with a window based search with a window of size s, and using a frame cache of size q. The window size is set to $s = 27$, and the cache size $q = 4$ for this implementation.

The envisioned architecture always processes the first frame using the Frame Compression module, as the frame cache is not populated (as necessary to run Block Prediction). The remaining frames are processed in blocks. Each block is predicted by the Block Prediction module using the frame cache, with previously encoded frames as contextual information. Depending on the quality of the block prediction, one of three processes is chosen: fully compress the block (using the Block Absolute Compression module), compress the residual between the block and the prediction (using the Block Residual Compression module), or the predicted block is simply repeated as the new block and the residual ignored. The choice is made depending on a quality threshold, with the last case being the optimal case of an almost perfect prediction. For a frame, all blocks are processed in this manner, and the frame is then recreated by assembling the blocks together. The frame cache is updated with the newly compressed frame, and an older is discarded if necessary. The frame compression network is used in two cases: if the frame cache was empty (*e.g.*, the first frame), or if the resulting frame compressed by blocks is below a quality threshold, in this case justifying the compression of the whole frame. A more thorough explanation of the functioning of this compression scheme is provided

in Appendix A.

While this compression scheme achieved the objective of video compression and seemed intuitive, it faulted in several areas:

- The system was complex, as it depended on four independent networks;
- The prediction system suffered from blurriness issues when implemented with a neural network, and from optimization problems with the final window-based approach;
- The use of so many components made the video compression slow, to the point where encoding short sequences of video could take almost an hour. While decoding was fast, this put in jeopardy the previously mentioned constraints of video compression algorithms in terms of runtime;
- The compression scheme was using the already established predictive architecture, but the results did not get close to H.264/AVC or H.265/HEVC.

As a consequence of the enumerated problems, the research in predictive architectures stalled. The most direct path to improving results was to make the network more complex. In turn, this would aggravate the complexity and runtime problems of the network. The lack of potential for this architecture eventually lead to a search for new approaches. Hence, the research took a step back to explore other approaches to video compression, from which the final solution resulted.

4.3 Proposed approach

Given the difficulties faced using predictive architectures, the research was steered in a different direction with the objective of departing from any kind of motion estimation. A natural way of avoiding the hand-crafted nature, and computational burden, of frame prediction or motion estimation, is to use a single module which creates the compressed video, without any intermediate steps. In this case, it is necessary to directly design and optimize a network for efficient video representation, as is done for image compression.

Looking at the different architectures that have been proposed for image compression, they are generally based on a common principle: the original image is transformed into a latent representation by an encoder; this representation is quantized and entropy-coded; a decoder reconstructs the image from the quantized latent representation. This type of approach relies on the ability of *deep neural networks* (DNN) to extract meaningful and compact representations from 2D data (for which convolutional autoencoders are particularly suited, see Section C of 2.2.4).

This simple approach to the compression problem served as the main inspiration for the final solution. It was further motivated by previous work on spatio-temporal autoencoders (see Section G of 2.2.5) especially when implemented as 3DCNNs (*e.g.*, Hara et al. [72], Zhu et al. [73], Qiu et al. [74]). Just as image compression uses a projection transform to extract spatial information, a projection transform that analyzes the spatial-temporal redundancy within and between frames is able

to learn a meaningful latent-space representation of video. If the learned latent-space is efficient, it is possible to compress videos by quantizing this latent representation.

With this in mind, for the final solution, video compression is formulated as the problem of learning a low-entropy latent-space representation from which the original video can be reconstructed as well as possible. The compression loss results from quantizing the learned latent-space representation, in order to selectively discard information that does not significantly penalize reconstruction quality. By learning a probability model for the quantized latent representation, it is possible to quantify the amount of information that is used for the quantized representation, and impose a limit on it. Since the reconstruction quality increases if more information is allowed to flow through the bottleneck, the result is a rate-distortion trade-off, as is common in compression algorithms.

In other words, the video compression is tackled via the optimization of a single spatio-temporal autoencoder architecture which learns a low entropy latent-space for videos. The spatio-temporal autoencoder is divided in two transformations: one for projection and one for synthesis, to and from a latent space representative of the features of videos. Spatio-temporal autoencoders (see Part G of Section 2.2.5) were chosen, as they extend the convolutional autoencoder formulation used for image compression, with the ability to extract information about spatial and temporal dependencies. Of the possible formulations presented, 3DCNNs are more suitable to video compression due to the joint processing of spatial and temporal information.

To optimize the two projections, a rate-distortion optimization framework is proposed, inspired by and extending the work of Ballé et al. [4] (presented in Section 3.6). It is modified so as to also enforce temporal consistency between frames. Hence, this solution bases upon state-of-the-art DL-based image compression, extending it from the 2D realm of images to the 3D world of video data. The architecture is trained by end-to-end learning, optimizing a loss function that combines: reconstruction distortion; an estimate of the length of the entropy-coded quantized latent representation; a temporal consistency loss. Both transforms and the quantizer are jointly learned. The video is processed on a 3D pixel grid with three input channels (*e.g.*, RGB or YUV). The compressed video is the result of entropy-coding the quantized latent-space representation, and the reconstruction is obtained by applying the synthesis transform to that representation.

As far as it is possible to determine in publicly available research, this proposed architecture is the first proposed DL-based method that learns to compress videos in an end-to-end fashion, without intermediate steps.

With this solution, the problem of relying in multiple networks is averted, as the final solution departs from the traditional video codec structures, by avoiding any type of explicit motion prediction. Instead, the focus is on learning a direct correspondence between a compressed representation and the original video. In the following sections, the problem is first introduced formally, and then the proposed architecture and implementation is discussed.

4.3.1 Problem Formulation

Formally, a video is a sequence of N frames $f = (f_1, f_2, \ldots, f_N) \in \mathcal{F}^N$, where $f_i \in \mathcal{F}$, with $\mathcal{F}$ denoting the space to which each frame belongs (*e.g.*, $[0; R]^{W \times H \times 3}$, for RGB frames with $(W \times H)$ pixels, and a maximum pixel value of R). An encoder $\mathsf{E}_\phi : \mathcal{F}^N \to \mathcal{Z}$, where $\mathcal{Z} = \mathbb{R}^L$ is the latent space and L its dimensionality, extracts a representation $z = \mathsf{E}_\phi(f)$. The latent-space representation is then quantized, $\bar{z} = q(z)$, where $q : \mathcal{Z} \to \mathcal{S}$ is a quantizer, with some finite code-book $\mathcal{S}$. Finally, the quantized latent-space representation $\bar{z} \in \mathcal{S}$ is used by a decoder $\mathsf{D}_\theta : \mathcal{S} \to \mathcal{F}^N$ to reconstruct an approximation $\hat{f}$ of the original video f.

In this formulation, ϕ and θ are the parameters of the encoder (or projection transform) and the decoder (synthesis transform), respectively. The whole network $\mathsf{C}_{\phi,\theta} : \mathcal{F}^N \to \mathcal{F}^N$ is the composition of the encoder, quantizer, and decoder:

$$\hat{f} = \mathsf{C}_{\phi,\theta}(f) = \mathsf{D}_\theta\Big(q\big(\mathsf{E}_\phi(f)\big)\Big). \tag{4.1}$$

In this work the quantizer q represents quantization by rounding each component of z to the nearest integer, thus $\mathcal{S}$ is simply a finite subset of $\mathbb{Z}^L$ (vectors of integers).

The compressed representation of the video results from the entropic coding of $\bar{z}$, which is a lossless operation, thus omitted in (4.1). Optimal entropic coding depends on the probability distribution of $\bar{z}$, $p_{\bar{z}}$, which must be estimated during training to allow the rate-distortion optimization of the network. For this, the proposed method, inspires from the work of Ballé et al. [4], while noting that the components of z are not independent. Thus, each element of z, say z_i, is modeled by a zero-mean Gaussian distribution with variance σ_i^2, which should be dependent on other (in some neighborhood) components of z. These variances are estimated by an additional autoencoder, $\mathsf{H}_{\epsilon,\rho} : \mathcal{Z} \to \mathbb{R}_+^L$ (the *hyperprior* network), itself the composition of an encoder $\mathsf{X}_\epsilon : \mathcal{Z} \to \mathcal{W}$ (where $\mathcal{W}$ is the latent space of the hyperprior), the same rounding quantizer q as above, and a decoder $\mathsf{Y}_\rho : \mathcal{S} \to \mathbb{R}_+^L$. As the variances are needed to decode the entropy-coded bitstream, the quantized latent representation of the hyperprior, $\bar{w} = q(w) = q\big(\mathsf{X}_\epsilon(z)\big)$, must also be sent to the decoder as part of the compressed representation. Thus, the hyperprior can be seen as a side-information channel that improves the overall rate-distortion performance of the encoder, although contributing to the bitstream size.

To jointly optimize (in the rate-distortion sense) the main network $\mathsf{C}_{\phi,\theta}$ and the hypeprior network $\mathsf{H}_{\epsilon,\rho}$, an entropy model for the hyperprior needs to be estimated during training. The probability distribution of $\bar{w}$ is modeled as a factorized model

$$p(\bar{w}|\delta) = \prod_i p(\bar{w}_i|\delta_i), \tag{4.2}$$

where each δ_i is estimated during training. More specifically, each δ_i is obtained by approximating $p(\bar{w}_i|\delta_i)$ by a probability density function (pdf) defined in terms of its cumulative density function (cdf) $P : \mathbb{R} \to [0, 1]$. The pdf $p : \mathbb{R} \to \mathbb{R}^+$ in (4.2) is defined in terms of its cdf $P : \mathbb{R} \to [0, 1]$ ($P(x) = \int_{-\infty}^{x} p(y)dy$), as a general approximator of density functions [4]. The two functions are defined as

follows, for $0 < k \leq K$ (setting $K = 5$ in this architecture)

$$\begin{aligned} P &= g_K \circ g_{K-1} \cdots \circ g_1, \\ p &= g_K^{'} g_{K-1}^{'} \cdots f_1^{'}, \\ g_k(x) &= h_k(H^{(k)}x + b^{(K)}), \\ g_K(x) &= \text{sigmoid}(H^{(k)}x + b^{(K)}), \\ h_k(x) &= x + a^{(k)} \odot \tanh(x). \end{aligned} \tag{4.3}$$

To simplify the model, the assumption is made that $\delta_i = \delta_j$, if i and j are in the same channel of the hyperprior latent representation.

4.3.2 Optimization Framework

The process of obtaining the compressed representation of a video and recreating the video from this representation is expressed in (4.1). However, parameter learning cannot be carried out directly on this structure, since the quantizer would block the backpropagation of the gradients through the bottleneck, a fact that has been pointed out by several researchers [3, 4, 14, 17]. One way to circumvent this difficulty exploits a classical result from quantization theory [57]: high-rate quantization noise/error is well approximated by additive noise, with uniform distribution in the quantization interval. During training, the explicit quantization is thus replaced by additive noise with a uniform density on the interval $[-\frac{1}{2}, \frac{1}{2})$, since the quantizer q simply rounds each component of its input to the nearest integer. As explained in Section 3.3, this method was first proposed by Toderici et al. [2], and successfully used in many more image compression works (*e.g.*, Toderici et al. [3], Ballé et al. [4]).

The proposed network is end-to-end optimized for both the reconstruction loss ($\mathcal{L}_r$) and the total entropy of the representation ($\mathcal{L}_h$). Additionally, a third loss term is included, encouraging temporal consistency between consecutive frames ($\mathcal{L}_t$). The final optimization loss function $\mathcal{L}$ is expressed as the weighted sum

$$\mathcal{L}(f, \hat{f}) = \mathcal{L}_r(f, \hat{f}) + \alpha\, \mathcal{L}_h(f) + \beta\, \mathcal{L}_t(f, \hat{f}), \tag{4.4}$$

with α and β empirically determined to balance training stability, proximity to a target bitrate, reconstruction quality, and temporal consistency. Each term in (4.4) is described in more detail in the next sub-sections.

A Reconstruction Loss $\mathcal{L}_r$

The adopted reconstruction loss is simply the MSE (as defined in Section 2.2.3,

$$\mathcal{L}_r(f, \hat{f}) = \text{MSE}(f, \hat{f}). \tag{4.5}$$

Optimizing the MSE is equivalent to optimizing the PSNR (*peak signal-to-noise ratio*), since $\text{PSNR}(f, \hat{f}) = 10 \log_{10}\big(R^2/\text{MSE}(f, \hat{f})\big)$ (where R is the range of the pixel values), which is the most common metric used in assessing video compressors.

It is possible and would make sense to optimize more sophisticated metrics that correlate better with (human) perceptual quality, such as the well-known MS-SSIM [27]). However, optimizing the

PSNR helps to adequately compare the proposed network with other codecs that are also optimized for PSNR, which is a far more relevant goal at this point of the work (initial exploration of this type of compression schemes for video), so as to understand the capacity of the network in summarizing information from videos.

B Entropy Loss $\mathcal{L}_h$

The approximate probability models learned during training allow estimating the length of corresponding optimal codes for the quantized variables [45]. The length of the optimal codeword for some particular $\hat{z}_i$ is approximately (*i.e.*, ignoring that it has to be integer and assuming that the probability mass function of the quantized variable is well approximated by the corresponding pdf at that value) equal to $-\log p(\hat{z}_i)$, where $p(\hat{z}_i)$ is approximated by a Gaussian distribution, as described in Section 4.3.1; similarly, the optimal codeword for some particular $\hat{w}_i$ has length $-\log p(\hat{w}_i|\delta_i)$, as given by (4.2). This reasoning uses assumes that the entropy of a discrete variable in the quantized latent representation, is well approximated by the modeled differential entropy during training. This has already been shown to be true for image compression as discussed by Ballé et al. [4]. Consequently, the total number of bits used by optimal codes for $\bar{z}$ and $\bar{w}$, under these probability models, is approximately given by

$$L(\bar{z}, \bar{w}) \simeq \sum_i -\log_2 \mathcal{N}(\bar{z}_i|0, \sigma_i^2) + \sum_j -\log_2 p(\bar{w}_i|\delta_i), \tag{4.6}$$

where $\mathcal{N}(x|\mu, \sigma^2)$ denotes a Gaussian pdf of mean μ and variance σ^2, computed at x. Finally, since both $\bar{z}$ and $\bar{w}$ are functions of f, the entropy loss is given by

$$\mathcal{L}_h(f) = L\Big(q\big(E_\phi(f)\big), q\big(X_\epsilon(E_\phi(f))\big)\Big). \tag{4.7}$$

By penalizing the entropy of the quantized representations at the bottleneck as in Ballé et al. [4], rather than simply its dimensionality (*e.g.*, Rippel and Bourdev [1], Toderici et al. [2, 3]), the amount of information to be used for reconstruction is controlled, thus forcing the network to jointly optimize the reconstruction loss and the bitrate.

C Temporal Consistency Loss $\mathcal{L}_t$

The generative nature of the spatio-temporal autoencoder may produce inconsistencies between frames, mainly due to the projection of a quantized latent representation back to the original video space. At low bitrates, the reconstructed video relies heavily on prior information embodied in the reconstruction sub-network D_θ, which may yield inconsistencies between consecutive frames, since the reconstruction loss used in training operates on a frame-by-frame fashion. Since the human visual system is very sensitive to disruptions to the temporal consistency/continuity between frames, this may cause a serious degradation of the perceptual quality of the reconstructed video sequence. To address this problem, a short-term temporal consistency loss is introduced, $\mathcal{L}_t$, inspired by the work of Lai et al. [75]. This short-term temporal loss is based on the warping error between each pair of subsequent input frames and the corresponding pair of output frames. The warping uses the

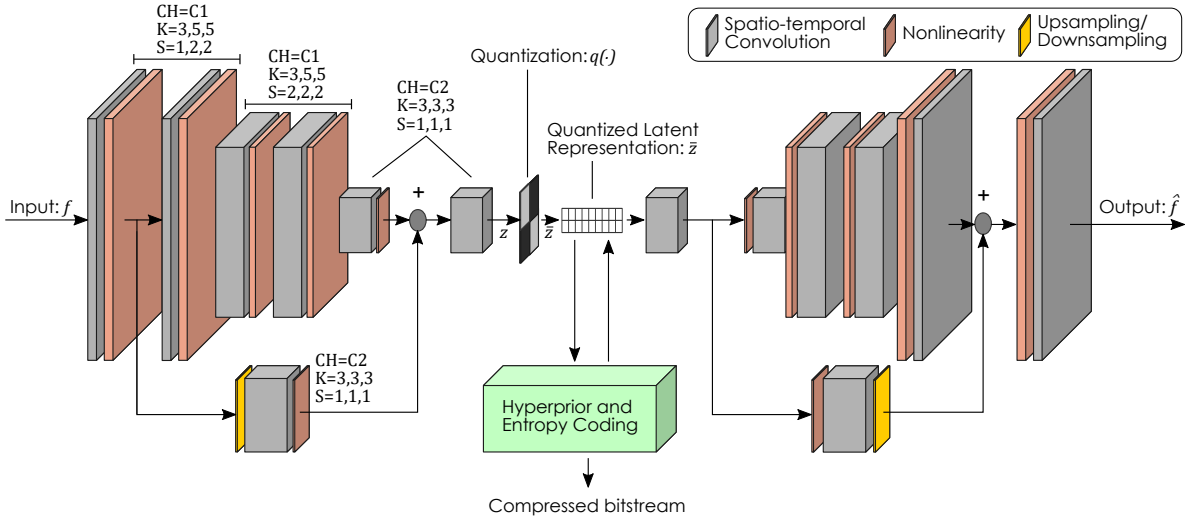


Figure 4.2: Illustration of the spatio-temporal autoencoder architecture proposed as the final solution for video compression using DL. The parameters of each block correspond to the convolution configuration (CH – convolution channels; K – kernel size; S – stride). When the parameters are presented as a set of three numbers, the first refers to the temporal dimension and the remaining two to the spatial dimensions. Blocks of different sizes represent different hyper-parameter combinations.

optical flow between the two input frames, obtained by a pre-trained version of *Flownet 2.0* [76]. More specifically, the temporal loss is defined as

$$\mathcal{L}_t = \sum_{t=2}^{N} ||M_t \odot (\hat{f}_t - \tilde{f}_{t-1})||_2^2, \tag{4.8}$$

where $\tilde{f}_{t-1}$ is the result of warping the frame $\hat{f}_t$ to time $t-1$ by using the estimated backwards optical flow from f_t to f_{t-1}; M_t is a binary occlusion mask excluding pixels that are not present in both $\hat{f}_t$ and $\tilde{f}_{t-1}$ (thus are not comparable); and $\odot$ denotes the Hadamard (pixel-wise) product,

4.3.3 Network Architecture

Both RCNNs and 3DCNNs were considered for this architecture. However, a single RCNN cannot be used to learn a latent space for video as only one frame can be processed per iteration. Instead, it learns a representation for one frame at a time, which makes use of its memory along the temporal dimension to learn what could be considered a sequential latent space. A representation at iteration n only makes sense considering all the previous iterations. RCNNs proved to be unfit for due to the lack of efficiency in the latent representations, specifically when focusing on low bitrates, and in the time spent to process a sequence of frames (one iteration per frame is required). The result is an inflexible network that cannot adapt to varying amounts of information to encode and is either insufficient or inefficient in most cases. Thus, a 3DCNN architecture is used, which does not suffer from the two previous problems and achieved decent performance on preliminary tests against the alternative, while only running once per frame sequence. With this architecture, the spatial and temporal information is processed simultaneously and all the frames in a sequence are encoded in a single iteration. The only downside identified for 3DCNNs is the higher memory requirements. The comparison between RCNNs and 3DCNNs is not presented here as no formulation of a RCNN that

was tested was able to optimize the reconstruction loss within acceptable values (as in, the training could not even be stabilized for the compression ratios used in video).

The encoder E_ϕ is composed of processing blocks on two different scales (see Figure 4.2), for which the motivation is discussed below. Each processing block is composed of a 3D convolution and a non-linearity. Leaky ReLUs [77] with a leak of 0.2 were used as the non-linearity. Other non-linearities were tested, in particular: PReLUs, ReLUs (see Section A and GDN Layers (as presented by Ballé et al. [78]). From these three, PReLUs presented the same average performance, with the disadvantage of slightly higher computation cost and ReLUs were slightly worse than the other variations of ReLUs. GDNs, albeit theoretically useful to exploit dependencies in the data through normalization, were not applicable to the temporal dimensions. Thus, even though an effort was made to extend the GDN normalization to spatio-temporal data, the layer difficulted convergence and the only obtained results were marginally worse than with the Leaky ReLU, which was chosen for the final architecture.

The latent representation results from adding the two scales and processing the result with a final inter-scale block. The shallower path in Figure 4.2 uses trilinear downsampling to align both outputs. To choose the number of processing blocks in each scale, a preliminary test was done using the common number of layers for 3DCNNs, ranging from three to six in the deeper scale; the secondary scale was always kept as a skip connection, with a single processing block. The problem with adding more layers is that the additional expressiveness that the encoder and decoder can gain due to processing the input at higher levels may not translate to increased performance if the current depth of the network is enough; in turn, the computational resources used by the network keeps increasing, meaning that the obtained performance must be balanced against the available resources, especially because there are other ways of increasing the extraction capability of the network. The number of filters for the network was chosen in a similar, empirical manner; as the channels vary depending on the target bitrate for the model, the values for the parameters in Figure 4.2 are specified alongside each trained model and the corresponding results, in Chapter 5. The decoder D_θ is built by reversing the order of the operations in the encoder, replacing the convolutions by their transposes, and downsampling operations by the corresponding upsamplings. The architecture's two initial blocks only use non-unit stride in the spatial dimensions, so as to apply a lower reduction in size on the temporal dimension than on the spatial ones.

Because dependencies exist in the data at different spatial scales, combining the results from different layers of a DNN has shown considerable improvements in many tasks. The multi-scale architecture of the image compression network studied in Section 3.5 serves as an example of using multi-scale connections within the presented network, with residual networks being another instance of this idea [29, 79]. Temporal dependencies are also present in short and long frame sequences (short or long time-scales). Motivated by the discussed architectures and this observation, a dual-scale architecture is used (see Figure 4.2) to better adapt to the variable dependency/redundancy scale of the input video. The main path of the network is structured as a regular deep encoder, while the secondary path resembles a high-level skip connection in some residual architectures. Notice

that the secondary path is fed before applying any form of temporal compression. The two paths aggregate information from different temporal and spatial scales, yielding a more expressive latent representation. The use of additional scales was evaluated, but the improvements were marginal or non-existent; the increase in computational and memory requirements of additional scales makes their inclusion impractical.

Finally, when compared to the original formulation of Ballé et al. [4] discussed in Section 3.6, the scale hyperprior network is extended to estimate the variance from both the remaining spatial and temporal redundancy in z, by structuring $\mathrm{H}_{\epsilon,\rho}$ also as a spatio-temporal autoencoder. The hyperprior network is not shown in detail in Figure 4.2, to keep it from becoming too complex. The encoder X_{ϵ} uses three sequential processing blocks with a 3D convolution ($C1$ channels, kernel size 3 and stride (1,2,2)) and a ReLU non-linearity. The decoder Y_{ρ} inverts X_{ϵ} by replacing convolutions with the corresponding transposed convolutions.

4.4 Summary

This chapter presents the work performed in video compression in the scope of the thesis. It begins by reinforcing the probable cause for the lack of DL-based video compression methods: the added complexity of video coding when compared to image. It is also noted that all the existing video codecs rely on motion estimation/frame prediction techniques to tackle this problem. An initial prediction based architecture is described, based on the established video codecs, but using neural networks for image compression, residual compression, and motion prediction. However, the achieved results are poor, and the complexity of the architecture impacts the runtime.

Thus, to avoid the complexity of motion estimation, a different approach to video compression is proposed using a more direct method. The problem is first formalized as a search for a latent-space where videos can efficiently be represented, using a single projection and a synthesis transform. Then, a multi-scale spatio-temporal autoencoder architecture is presented as the implementation for both transforms. To train the architecture, a novel rate-distortion optimization framework is proposed, based in recent research in image compression, with the additional concern of ensuring temporal consistency between frames.

The chapter is finalized with an overview of the implementation details for the spatio-temporal autoencoder, and the decision process behind some particular choices, such as the multi-scale architecture.

5

Experimental Results

Contents

Figure 5.1: The twelve sequences of the MCL-V video database, accompanied by an example frame from each. The abbreviation used throughout this chapter for each clip is in parenthesis. The figure is extracted from the original article by Lin et al. [5].

5.1 Experimental Setup

To validate the proposed video compression scheme, it was implemented in **PyTorch 0.4**. The entropy coding module is implemented as a single-threaded range coder and decoder, in native *C++*, which interfaces with PyTorch data. The source code was also made publicly available in a GitHub repository (see Appendix B for more instructions). The network was optimized for five target bitrates, each using a different parameter setting (see Table 5.1). Two filter configurations are used (parameters C1 and C2), one concentrated on very low bitrates (networks A-C), the other allowing additional information to improve the video quality (networks D-E). The training set is composed of 10^4 *high-definition* (HD) videos with at least 1080p resolution (1920×1080), randomly chosen from the YouTube-8M dataset [80]. Each video was downscaled to half of its original size in both spatial dimensions (to minimize any existing compression artifacts) and sliced into 32-frame sequences, from each of which a random 128×128 spatial crop was extracted. Each training iteration uses a batch of 4 of these cropped sequences. The learning rate was fixed at 10^{-4} until the 5000-th iteration, beyond which a 0.2 decay was applied (once). Each network was trained until it reached stability in com-

Network	C1	C2	α	β
A	128	256	18	2.5
B	128	256	38	3.5
C	128	256	59	5.5
D	256	384	78	8.5
E	256	384	108	11.0

Table 5.1: Parameters used for the configuration of the five trained models (named A to E) of the proposed video compression scheme.

pressing a validation set of 10 videos also extracted from the YouTube-8M collection (disjoint from the training set), downscaled to 640×360.

Evaluation was carried out on a collection of 10 uncompressed 1080p videos from the MCL-V database [5]. The MCL-V database was created to assess streaming codecs, thus it includes a variety of scenes making it well-suited to test the general-purpose nature of the proposed framework. A preview of all the twelve videos pertaining to the dataset is provided in Figure 5.1.

This collection was chosen in particular as it showcases videos with different balances between spatial and temporal redundancy, which can be used to assess in what conditions the presented architecture performs adequately. The original submission of the dataset by the authors Lin et al. [5] provides a concrete analysis on that balance, which was a significant factor when choosing this dataset in particular. The plot of Temporal Information (TI) against Spatial Information (SI) is provided in Figure 5.2 along with an explanation. Because the training set has very few synthetic videos, two of the videos from MCL-V have been excluded from the test set (BB and FB), to avoid skewing the results. Each evaluation clip was downscaled to a width 640 pixels, maintaining its aspect ratio. Our architecture is evaluated against the MPEG-4 Part 2 [81] (referred to as MPEG-4 in plots) implemen-

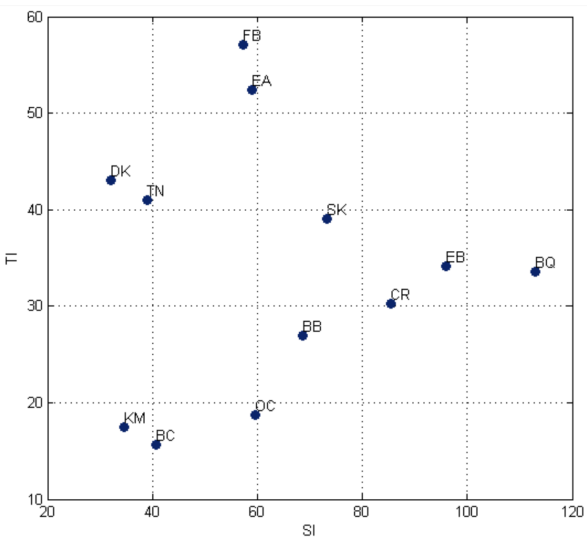


Figure 5.2: Plot of the balance between Temporal and Spatial information present in each sequence of the MCL-V video database. The abbreviations in the graph correspond to the introduced ones in Figure 5.1. The Spatial Information is calcualted using a spatial entropy measure and the Temporal Information using an optical flow-based displacement measure. The figure is extracted from Lin et al. [5], which contains a deeper explanation of the calculation method.

tation in libxvid (version 1.3.4), the H.264/AVC [82] implementation in libx264 (version 142 r2495a, using profile "High Level-2.2") and the H.265/HEVC [18] implementation in libx265 (version 1.9, using profile "Main Level-2.1"). In the three cases, the *ffmpeg* (`www.ffmpeg.org`) (version 2.6.8) software-frontend was used for compression. The comparison also includes a baseline video compression achieved by compressing each frame individually with a DL-based image compression scheme. More concretely, the image compression architecture used is the one introduced in Section 3.6, with four trained models for the lower achievable bitrates. The results of the baseline are useful to assess if the spatio-temporal nature of the proposed solution does, in fact, capture both spatial and temporal redundancies in the video, or if the achieved performance is solely due to the superiority of the rate-distortion framework and the DL approach. The proposed video compression scheme is always referred to as "Proposal" in the evaluation plots and figures.

The video quality evaluation is presented in two different evaluation metrics: the classical PSNR, as well as MS-SSIM, which is often used to estimate the perceived quality of images and videos (see Section 2.3 for a detailed discussion of these metrics).

5.2 Results

5.2.1 Video Quality Assessment

The average rate-distortion curve for each network (spatio-temporal autoencoder and baseline) and each profile level (MPEG-4 and H.264/AVC) is shown in Figure 5.3. Individual curves for each of the test sequences are shown in Figures 5.4 and 5.5 for MS-SSIM and PSNR respectively. For the baseline and the proposed architecture, the curves are obtained by compressing all the evaluation videos with each trained model. For each model, the obtained bitrates and obtained metrics are averaged, determining one point of the curve. The obtained points are connected in order to obtain the final average curve. For the video codecs, the curves are calculated in a similar way, except each point is associated to one compression profile choice from the available in the used implementation.

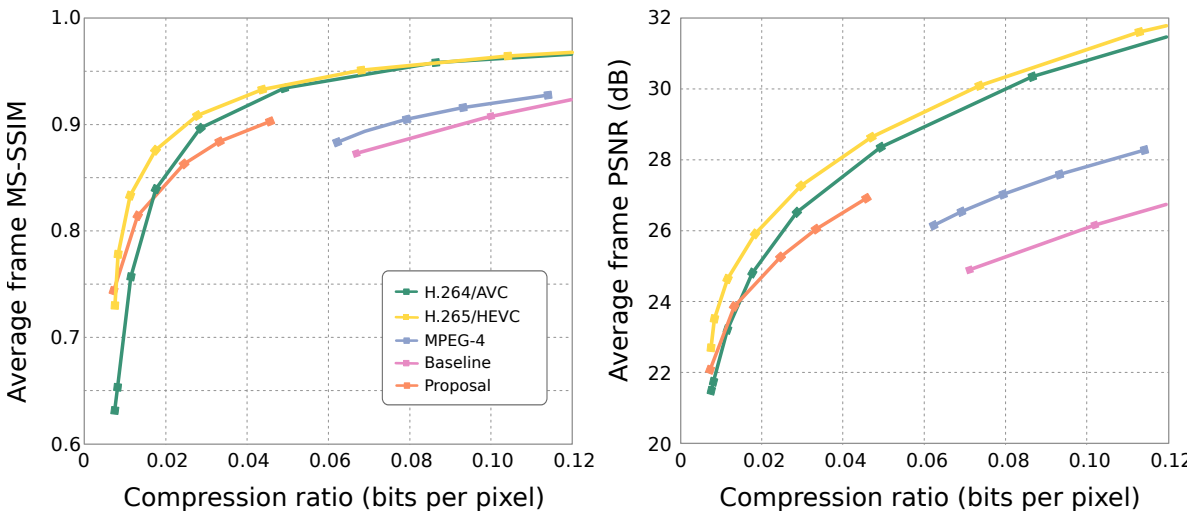


Figure 5.3: Summarized results for the MCL-V dataset for both MS-SSIM and PSNR.

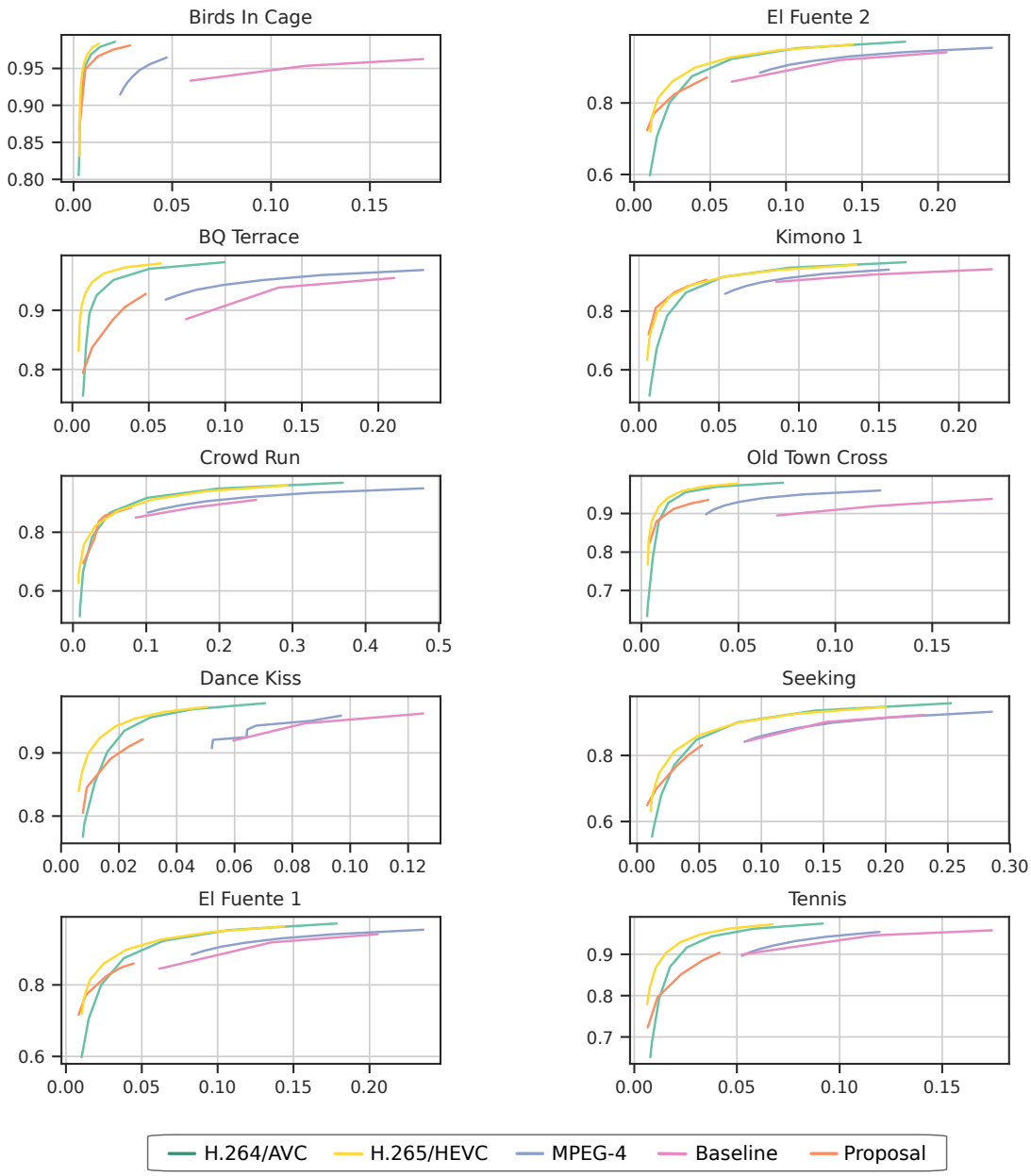


Figure 5.4: A grid of the individual MS-SSIM results for each one of the 10 considered video clips of the MCL-V dataset (names in Figure 5.1). The legend of the last graph applies to all the graphs in the grid.

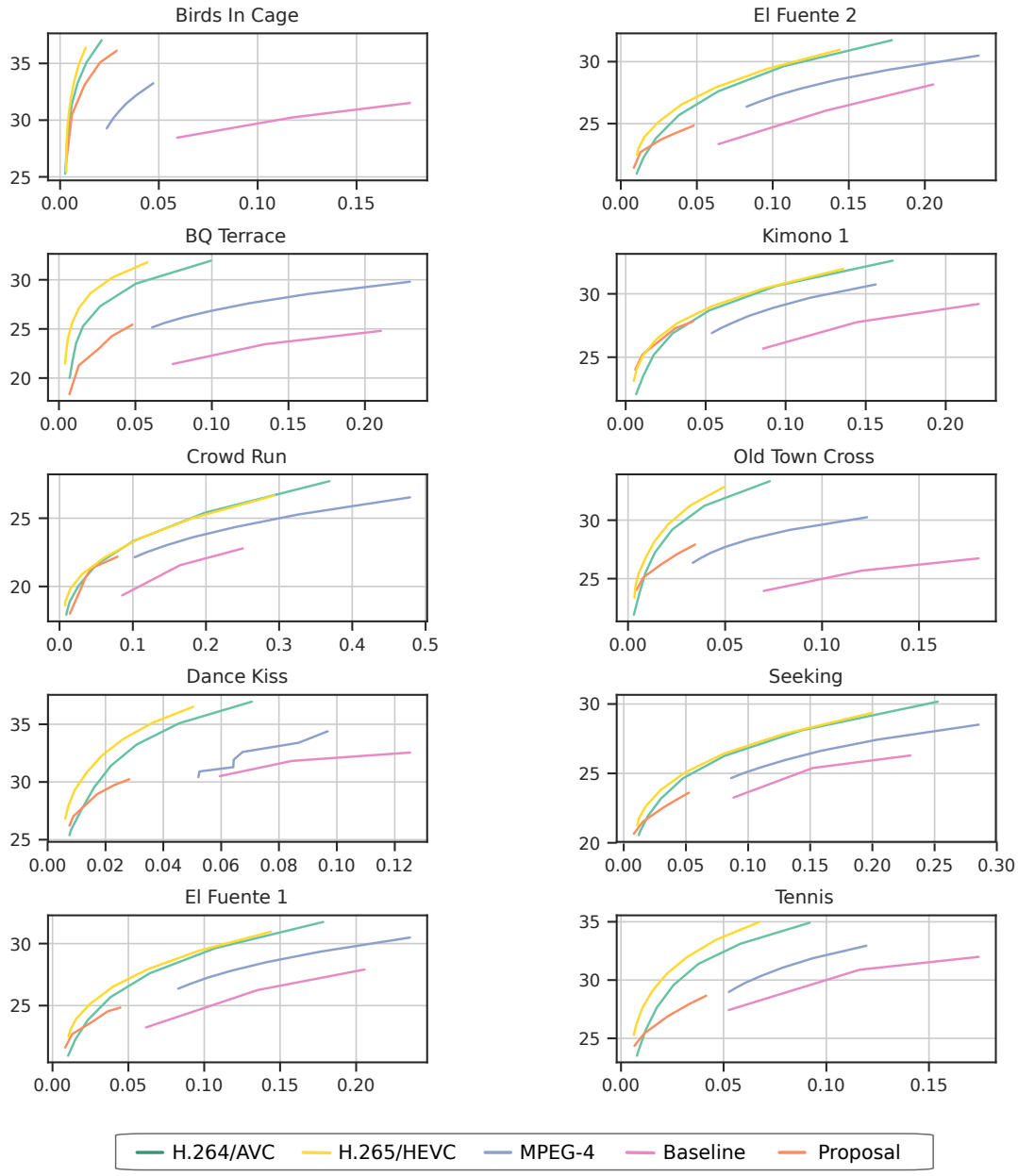


Figure 5.5: A grid of the individual PSNR (measured in dB) results for each one of the 10 considered video clips of the MCL-V dataset (with the same name as in the article for the dataset). The legend of the last graph applies to all the graphs in the grid.

The proposed architecture achieves far better performance than the comparison baseline, clarifying that it does use both temporal and spatial information in the data. The results also show that the video compression network attains similar quality levels than MPEG-4 Part 2 in both metrics for significantly lower bitrates. Both the curves for the baseline and MPEG-4 Part 2 start in a higher bitrate than the other curves due to the inability to reach higher compression ratios. Moreover, it is possible to see that the network has competitive performance to H.264/AVC, slightly surpassing it at the low bitrates. When compared to H.265/HEVC the proposed architecture is still competitive at the lowest bitrate, but clearly falls behind as the bitrate increases. One of the most noticeable improvements of the proposed scheme is that the artifacts produced by the network at low bitrates can be considered as perceptually more acceptable, compared to all the traditional video codecs (H.265/HEVC, H.264/AVC and MPEG-4), even at similar PSNR and MS-SSIM values. Figure 5.6 shows a comparison between crops of frames from two different sequences of the MCL-V database, at similar compression ratios, for both the proposed network, H.264/AVC, and H.265/HEVC, illustrating the aforementioned property. A more detailed analysis is provided after the results.

5.2.2 Execution Time Evaluation

Additionally, a simple assessment of the runtime is provided for the different compression methods, including a breakdown of the runtime for the proposed compression method. All the runtimes were measured on an Intel Core i7-4770K CPU at 3.50GHz, and a NVIDIA Titan Xp (GP102) GPU. In the proposed architecture, there are two main components in terms of runtime: the spatio-temporal autoencoder which runs on the GPU and the entropy coder which runs on the CPU. All the other compression schemes run solely on the CPU. The measurements for the video codecs were obtained using the *ffmpeg*'s benchmarking utility.

Figure 5.7 shows a comparison between the time taken per frame to encode the same clip (BBB) for each video compression scheme, as a function of the bitrate. All the methods are within a reasonable time range; the proposed architecture is slightly slower compared to H.264/AVC but reasonably quicker than H.265/HEVC. MPEG-4 Part 2 stands out for the faster runtime, which can be mostly attributed to the simplicity of the codec.

Figure 5.8 provides a closer analysis to the runtime of the proposed architecture by separating the time spent in the video compression network (in red), and in the entropy coding stage (in blue). Both encoding (Sub-figure 5.8(a)) and decoding (Sub-figure 5.8(b)) are analyzed. Even without optimization concerns, both runtimes for encoding and decoding are quite below the necessary for real-time execution, which is particularly important in decoding.

No concern was taken at this point in optimizing the presented scheme for runtime, as can be seen for the entropy coding stage. It was created from zero (due to method constraints) in a single-threaded manner. As a result, although it is computationally simpler than spatio-temporal autoencoder, entropy coding slows down the execution of the proposed compression scheme due to lack of optimization. On the contrary, the other tested video codecs use widely distributed implementations which are thoroughly optimized. And even then, the runtimes achieved by the proposed architecture are com-

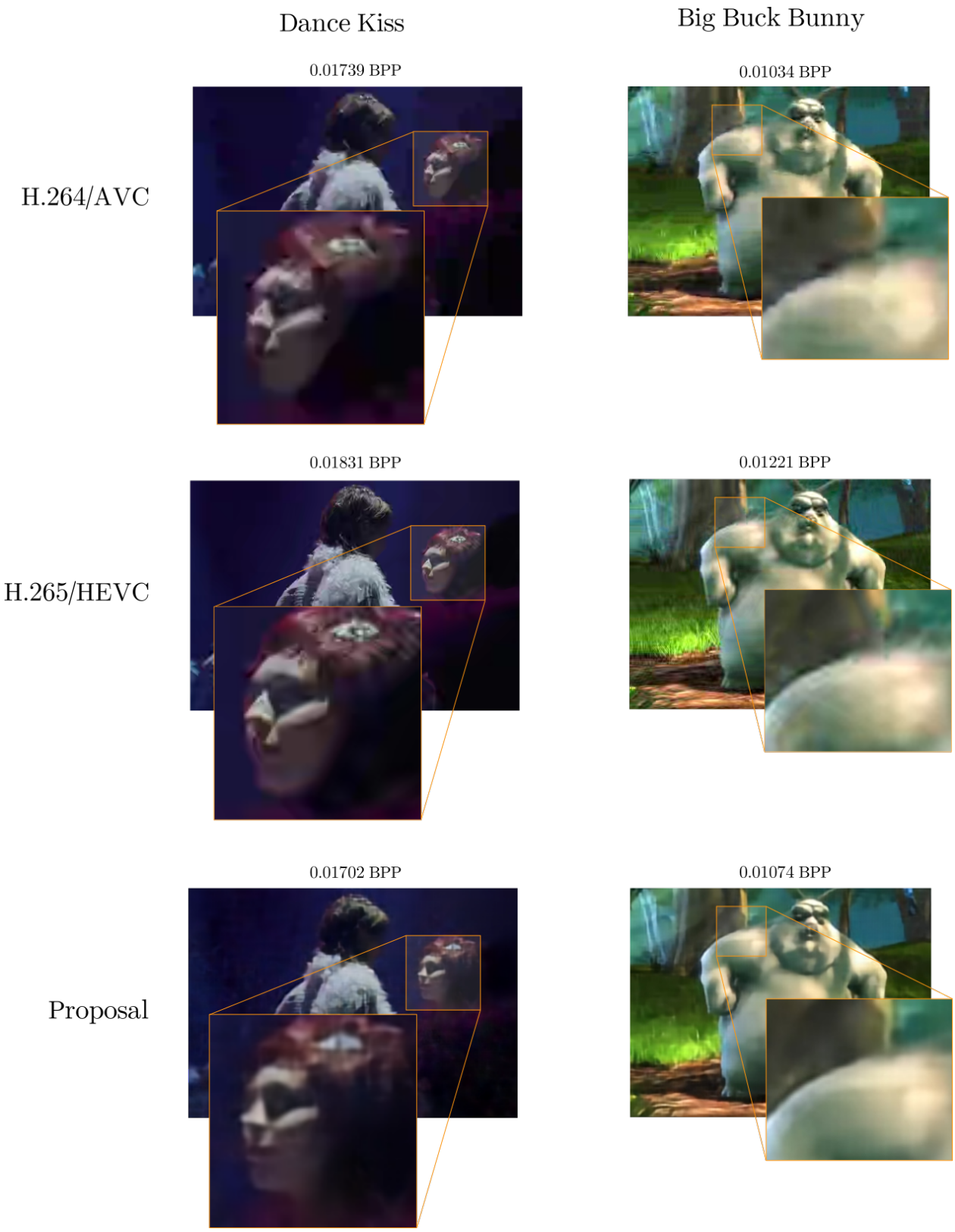


Figure 5.6: An example frame comparison between compressed sequences (at similar compression ratios) from the Dance Kiss clip and Big Buck Bunny clip of the MCL-V database. Results are shown for the proposed network, H.265/HEVC and H.264/AVC. A detailed zoom shows that distortions produced by the proposed approach are more natural. The image can be zoomed to analyze other details.

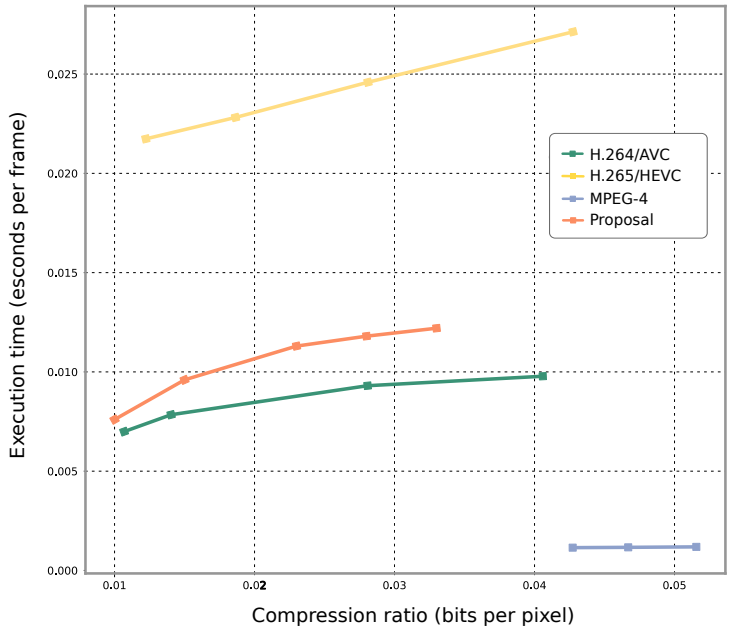


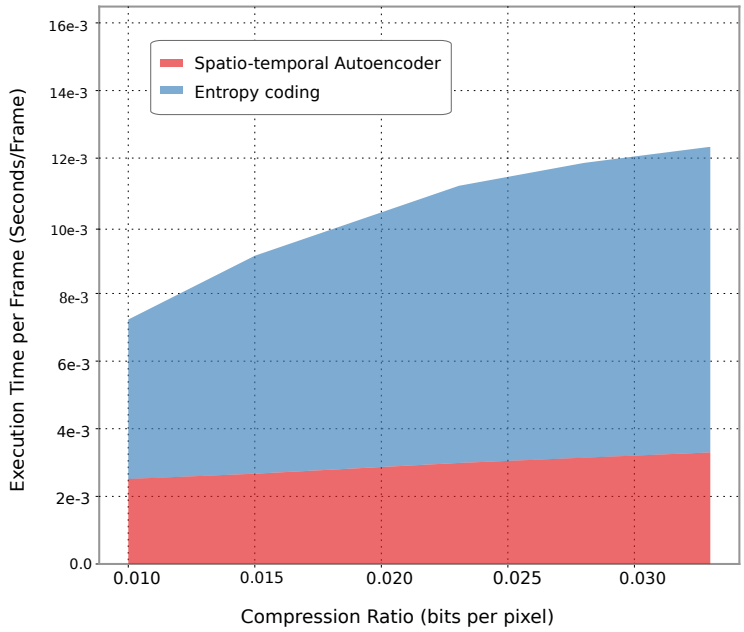
Figure 5.7: Comparison of the runtime during encoding, for each considered video compression scheme, except the baseline.

parable, demonstrating that fits well within the constraints of computational resources and runtime for video compression, discussed in the motivation of Chapter 4.

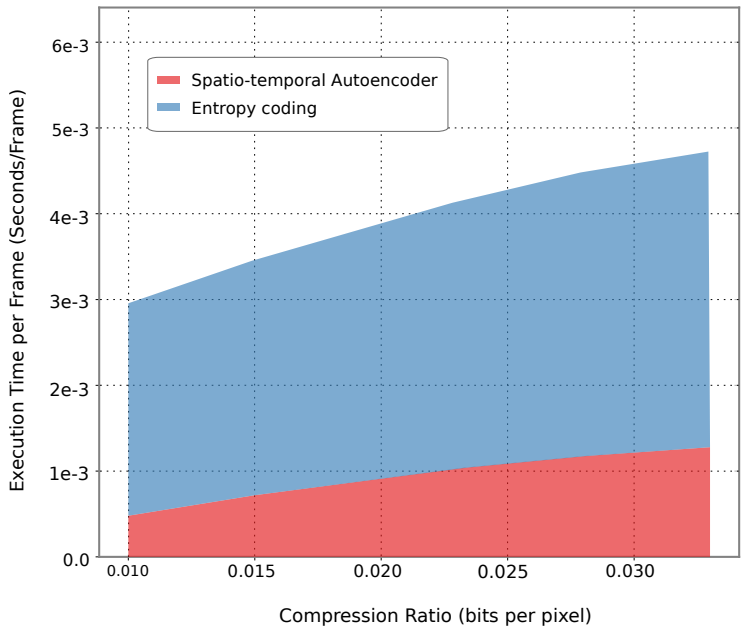
5.2.3 Discussion

The proposed network performs exceedingly well for lower bitrates as can be seen in the average curves for each compression scheme (see Figure 5.3). The produced results at these bitrates are arguably more pleasant from a quality degradation perspective, demonstrating the advantages of avoiding motion-estimation techniques and block-based compression (see Figure 5.6). This is because the video compression scheme relies on the learned latent space to perform video compression. Accordingly, even at lower bitrates, the generated video conforms to the learned distribution of features and structure of videos in training dataset. Additionally, frames are jointly generated, and not the result of block-based processing. As a result, the generated artifacts look more natural, guaranteeing a graceful degradation of quality even for very low bitrates. Under similar conditions, traditional codecs suffer from highly unnatural blocking artifacts which are detrimental to the perceptual quality of the videos.

By cross-checking the detailed results of Figures 5.4 and 5.5, with the balance between spatial and temporal information (see Figure 5.2), it is possible to see a tendency of the proposed scheme to have worse performance for videos with higher imbalances of information. For example, the worst performing sequence is clearly the BQ, which demonstrates the highest imbalance towards spatial information from all clips. On the other end, DK and TN are also problematic for the network, and EA has a particularly bad tendency for the higher bitrates. These three are the clips with where the



(a) Encode time



(b) Decode time

Figure 5.8: Detailed runtime for the proposed video compression architecture, for encoding and decoding the BB video sequence, in seconds per frame. The graphs separate the time spent on the spatio-temporal autoencoder (red), and on the entropy coder (blue). Note that, to achieve real-time decoding of a video sequence with 30 frames per second, a decode time of $33e-3$ or lesser is necessary.

balance between temporal and spatial information is more skewed to the former. It should be noted that the units in which temporal and spatial information are measured are not normalized, so the comparison of spatial and temporal information is based on the relative differences between each clip, with the identity line in Figure 5.2 being considered an almost equally balanced video [5]. As a generalization, the network seems to suffer in performance for videos with a higher imbalance between spatial and temporal information. This is most likely the case due to the incapacity of the network to completely prioritize one type of information over the other, as it uses a fixed structure.

There is also a visible trend of the network to perform worse for videos with more temporal information, compared to spatial information. As for this problem, there are two probable causes: global motion (discussed in detail below) and the need to use smaller kernels and a smaller downsampling factor on the temporal dimension. This is done in order to avoid higher temporal compression but reduces the number of weights related to the temporal dimension in the network, and in turn the expressiveness in the temporal dimension.

In the particular case of global motion (*e.g.*, panning or zooming) the DL-based compression struggles to achieve efficient bitrates for any compression rate. Global motion refers to sequences in which consecutive frames are identical after a simple geometric transform. Capturing these global transformations is a trivial task for conventional video codecs with simple hand-crafted heuristics and almost no cost in bitrate. However, it is a particularly difficult case for the network as there is no easy way to generalize this information in an end-to-end architecture. This is one of the most important roadblocks that need to be resolved to further improve the performance of this type of compression schemes, and is further discussed in Section 6.1).

The proposed scheme also shows some difficulties when scaling for higher bitrate, which can already be seen in the presented results as the diminishing returns for this architecture are noticeable as bitrate increases. The architecture is limited by the complexity of the latent space that the encoder and decoder can jointly learn; in turn, they are limited by the expressiveness of both transformations. Increasing the number of filters allows, in principle, more complex models to be learned, which to some extent works, but also leads to an increase in computation cost and training time, to a point where it's not sustainable. Thus, one of the main areas of concern for the future in this type of compression schemes should be to either increase expressiveness of the network with low impact in computational resources used, or to decrease the overall use of computational resources without losing expressiveness.

Still, this works shows that through an end-to-end learning framework, with a single-network spatio-temporal auto-encoder, it is possible to achieve video compression performances for low bitrates that is competitive with classical codecs, designed with the contribution of several research groups over many years. Hence, this initial investigation should open the path for a new research line, focused on building efficient video compression networks while avoiding computational complex motion compensation and residual compression strategies.

5.3 Summary

This chapter focuses on the evaluation of the proposed video compression architecture in Chapter 4. Before presenting the results, the evaluation method is described. Namely, the metrics used to assess the results, and the compression schemes used for comparison: a baseline (based on image compression), H.265/HEVC, H.264/AVC, and MPEG-4 Part 2.

The presented results show that the proposed scheme performs visibly better than the baseline and MPEG-4 Part2, and is competitive with H.265/HEVC and H.264/AVC, especially in low bitrates. The generated artifacts by the network are also shown to look more natural, guaranteeing the graceful degradation of quality for lower bitrates. In contrast, the network struggles in videos with high imbalance towards either spatial or temporal information, particularly in the case of global motion.

An additional runtime assessment for the compression scheme is also presented. It demonstrates that the architecture runs within reasonable time constraints, and that both the encoding and decoding happen faster than real-time.

6

Conclusions and Future Work

Contents

While *Deep Learning* (DL) has raised the bar in many problems of image and video processing, including image compression, the same effect has not yet propagated to the area of video compression. Inspired by the parallelism between image and video compression, this thesis focuses on demonstrating that DL solutions can be made competitive with traditional video codecs. However, video is a complex type of data, for which the existing codecs are extremely well optimized, placing a very high barrier of entry into this field. The first step in devising a scheme that could surpass this barrier was to investigate the existing landscape of image and video.

However, due to the lack of research in video compression using DL, the initial focus was on the analogous field of image compression, which occupied a significant part of the research process. In the scope of this preparatory work, four distinct landmark works in image compression with DL were implemented and evaluated. The comparison served not only to understand the best performing image compression scheme but also to understand what techniques, based on their merits, were worthy of being pursued and adapted for video compression. As a conclusion to the preparatory step of research, the rate-distortion optimized VAE introduced in Section 3.6 is highlighted as the best performing compression scheme. The implementation of the different architectures represented a significant contribution to the field, by providing the first open-source repository of several image compression architectures, most of which were not provided by the original authors of the articles they are based on.

The review of existing traditional video codecs showed the hegemony of frame prediction/motion estimation architectures in the current landscape of video compression. Both the widely deployed codecs (H.264/AVC, MPEG-4 Part2) [7] and the best performing codecs (H.265/HEVC) [18] conform to this architecture. The natural conclusion to start the research in video compression was to apply the techniques studied in DL-based image compression, but relying on predictive coding schemes similar to conventional codecs. However, the results obtained were sub-par: being constrained to frame prediction and residual compression resulted in a dilemma between reusing the complete structure of existing codecs or trying to implement an architecture from scratch. In the former case, the resulting method would be far from a DL-based video compression method; in the latter, the resulting architecture used multiple networks, was slow, and performed poorly.

Thus, to push the research on DL-based video compression forward, a new direction is pursued by taking a step back and abandoning the established prediction-based approaches. Instead, the video compression problem is formulated as a latent space search with an entropy constraint. The new formulation can be approached with only two transformations: one for projection to the learned latent space, and one for synthesizing the reconstructed video. The bitstream representing the compressed video results from entropy-coding a quantized version of its latent representation. To adapt an architecture to the problem formulation, a rate-distortion optimization framework is proposed to train a single spatio-temporal autoencoder for both its reconstruction loss and entropy model. To combat temporal inconsistencies in the decoded videos, the optimization framework is augmented with a short-term temporal loss, encouraging the network to ensure temporal continuity between consecutive frames. The resulting work is, as far as it is possible to confirm in known publications, the

first end-to-end learned video compression architecture using DL.

The DL-based video compression has a superior performance than the image compression baseline and achieves similar video quality for much lower bitrates when compared to MPEG-4 Part 2. It is also competitive with both H.264/AVC and H.265/HEVC, especially in lower bitrates. Additionally, the obtained results show that the artifacts introduced by the proposed network are more natural, and arguably more visually pleasing, than the unnatural blocking artifacts of standard video codecs, guaranteeing the graceful degradation of video as bitrate decreases.

The provided runtime assessment and the comparison for the proposed architecture also show its viability in terms of computational resources and runtime, when compared with the other video codecs. It runs in a comparable time to H.264/AVC, and slightly faster than H.265/HEVC, even without any optimization concerns for the presented implementation.

The nature of this thesis dissertation is exploratory, and there is still much to understand before fully exploiting the DL potentials in this application. Hence, the expectation was not to surpass the performance of the carefully built standard video compression codecs, but to show that the presented DL-based methods are viable and represent an alternative solution. Even then, the proposed video compression architecture proved competitive against these codes both in terms of video quality and execution time.

As an overall assessment, the work done during the thesis has fulfilled all the initially defined objectives. The image compression research provided the intended comparison and tools. Then, the presented approach to video compression demonstrated the feasibility of tackling video compression with DL, and serves as an important base for all future work on video compression solutions.

6.1 Future Work

The proposed approach to video compression opens a new path in research which shows potential to be competitive with the traditional video codecs. However, further investigation is needed in this path to ensure robustness to different video properties and performance in all bitrates, with special attention to the issues that were discussed in Section 5.2.3.

In particular, the proposed architecture struggled in efficiently compressing videos as the bitrate increases. This problem occurs due to the difficulty in balancing the expressiveness needed to represent more complex models with the increase of computational resources needed for execution and time for training. The multi-scale model provided an adequate way to increase the expressiveness with a small increase in both, but additional scales had a reduced impact. There are several possible approaches to take here based on other research with 3DCNNs, but one of the most promising is the use of octrees to represent data which is usually structured in three-dimensional grids (such as video). For example, Wang et al. [83] use this approach to represent three-dimensional meshes in an octree structure. By using a similar approach, it should be possible to hierarchically represent video based on spatial and temporal complexity and process one layer of the hierarchy at a time. The three-dimensional convolutions used in Wang et al. [83] have an adapted formulation that can

be applied to an octree representation and which allow successive iterations of the network to focus in different areas of detail. By approaching the problem in this manner, several additional possibilities arise, such as having multiple bitrates with the same model by producing a bitstream at each iteration. But the most important advantage is the ability to process one video with much lower computational resources, which can then increase the available *budget* to increase the expressiveness of the network.

Other important problem to tackle, is the difficulty demonstrated by the network to compress videos with high temporal redundancy and lower spatial redundancy. This problem is mainly attributed to global motion effects which are responsible for a large amount of temporal information that can be easily captured by simple heuristics, as done in traditional video codecs. However, as the proposed scheme is end-to-end learned, there is no intermediate step in which the network can learn the simple geometric transformations applied. One possible approach to this is to use a pre-processing step in which a similar heuristic is used to detect the geometric transformations; then the video can be aligned in a different three-dimensional space so that the same areas of the video are represented in the same coordinates to be jointly processed. However, this approach may be inflexible for more complex transformations. Thus, another proposed approach is to include a learned, per-frame, spatial transformer unit in the network (in the fashion of STN networks as proposed by Jaderberg et al. [84]), which can distort each frame to provide spatial invariance between sequential frames that can then be exploited to reduce temporal redundancy.

Finally, the last important problem that should be understood is the decrease in performance for videos with a higher imbalance between spatial and temporal information. The most probable cause for this problem is the fixed structure of the network, making it difficult to disregard one type of information in benefit of the other, for highly imbalanced cases. To combat this issue, the approach with more potential is to rely on the normalization of the input video, before and during the network's execution. The initial objective should be to understand if it is possible to have any impact in reducing the imbalance between spatial and temporal information. In particular, learned normalization transformations such as the Generalized Divisive Normalization (GDN) proposed by Ballé et al. [78] seem a good fit for the problem in hand. The GDN was tested in the context of the network, however, the used formulation was a simple extension of the two-dimensional unit, applied per video frame. In this case, only the spatial dependencies are normalized which further aggravates the problem of the previous paragraph. The suggestion is to expand the formulation of such a learned normalization and apply it in three-dimensional grids, so as to account for both spatial and temporal statistical dependencies. With this, it should be possible to combat the sequences with more extreme inequalities in the type of information, and overall increase the performance of the network.

Bibliography

[1] O. Rippel and L. Bourdev, “Real-Time Adaptive Image Compression,” *ArXiv e-prints*, May 2017.

[2] G. Toderici, S. M. O’Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, “Variable rate image compression with recurrent neural networks,” *CoRR*, vol. abs/1511.06085, 2015. [Online]. Available: http://arxiv.org/abs/1511.06085

[3] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, “Full resolution image compression with recurrent neural networks,” in *CVPR*, 2017, pp. 5435–5443.

[4] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational image compression with a scale hyperprior,” *ArXiv e-prints*, Jan. 2018.

[5] J. Y. Lin, R. Song, C. Wu, T. Liu, H. Wang, and C.-C. J. Kuo, “Mcl-v,” *J. Vis. Comun. Image Represent.*, vol. 30, no. C, pp. 1–9, Jul. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jvcir.2015.02.012

[6] W. Cisco, “Visual Networking Index: Forecast and Methodology, 2016–2021,” Tech. Rep., 01 2016. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[7] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h.264/avc video coding standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[8] R. J. Clarke, “Image and video compression: A survey,” in *International Journal of Imaging Systems and Technology - INT J IMAGING SYST TECHNOL*, vol. 10, 01 1999, pp. 20–32.

[9] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” in *Neurocomputing*, 12 2016.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, pp. 354 EP –, Oct 2017, article. [Online]. Available: http://dx.doi.org/10.1038/nature24270

[11] K. Ahmed, N. S. Keskar, and R. Socher, “Weighted transformer network for machine translation,” *CoRR*, vol. abs/1711.02132, 2017. [Online]. Available: http://arxiv.org/abs/1711.02132

[12] R. D. Dony and S. Haykin, “Neural network approaches to image compression,” *Proceedings of the IEEE*, vol. 83, no. 2, pp. 288–303, Feb 1995.

[13] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML ’09. New York, NY, USA: ACM, 2009, pp. 873–880. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553486

[14] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. T. Chinen, S. J. Hwang, J. Shor, and G. Toderici, “Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks,” *CoRR*, vol. abs/1703.10114, 2017. [Online]. Available: http://arxiv.org/abs/1703.10114

[15] Z. Si and K. Shen, *Research on the WebP Image Format*. Singapore: Springer Singapore, 2016, pp. 271–277. [Online]. Available: https://doi.org/10.1007/978-981-10-0072-0_35

[16] J. Ballé, V. Laparra, and E. P. Simoncelli, “End-to-end optimization of nonlinear transform codes for perceptual quality,” in *Picture Coding Symposium (PCS), 2016*. IEEE, 2016, pp. 1–5.

[17] L. Theis, W. Shi, A. Cunningham, and F. Huszár, “Lossy image compression with compressive autoencoders,” *CoRR*, vol. abs/1703.00395, 2017. [Online]. Available: http://arxiv.org/abs/1703.00395

[18] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, “Overview of the high efficiency video coding (hevc) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec 2012.

[19] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, K. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, “State-of-the-art Speech Recognition With Sequence-to-Sequence Models,” *ArXiv e-prints*, Dec. 2017.

[20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” in *SSW*, 2016, p. 125.

[21] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional sequence to sequence learning,” *CoRR*, vol. abs/1705.03122, 2017. [Online]. Available: http://arxiv.org/abs/1705.03122

[22] D. Anisimov and T. Khanova, “Towards lightweight convolutional neural networks for object detection,” *CoRR*, vol. abs/1707.01395, 2017. [Online]. Available: http://arxiv.org/abs/1707.01395

[23] Y. Sun, B. Xue, and M. Zhang, “Evolving deep convolutional neural networks for image classification,” *CoRR*, vol. abs/1710.10741, 2017. [Online]. Available: http://arxiv.org/abs/1710.10741

[24] L. Bottou, *Large-Scale Machine Learning with Stochastic Gradient Descent*. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186. [Online]. Available: https://doi.org/10.1007/978-3-7908-2604-3_16

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, vol. 5, 2015.

[26] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[27] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, Nov 2003, pp. 1398–1402 Vol.2.

[28] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *CoRR*, vol. abs/1611.01704, 2016. [Online]. Available: http://arxiv.org/abs/1611.01704

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318–362. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104293

[31] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: http://science.sciencemag.org/content/313/5786/504

[32] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[33] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[34] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[35] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *ArXiv e-prints*, Dec. 2013.

[36] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational Inference: A Review for Statisticians," *ArXiv e-prints*, Jan. 2016.

[37] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, “Convolutional LSTM network: A machine learning approach for precipitation nowcasting,” in *Advances in neural information processing systems*, 2015, pp. 802–810.

[38] Y. Wang, M. Long, J. Wang, Z. Gao, and P. S. Yu, “PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017, pp. 879–888.

[39] J. Liu, A. Shahroudy, D. Xu, and G. Wang, “Spatio-temporal LSTM with trust gates for 3d human action recognition,” in *European Conference on Computer Vision*. Springer, 2016, pp. 816–833.

[40] V. Patraucean, A. Handa, and R. Cipolla, “Spatio-temporal video autoencoder with differentiable memory,” *CoRR*, vol. abs/1511.06309, 2015. [Online]. Available: http://arxiv.org/abs/1511.06309

[41] Y. Zhao, B. Deng, C. Shen, Y. Liu, H. Lu, and X.-S. Hua, “Spatio-temporal autoencoder for video anomaly detection,” in *Proceedings of the 2017 ACM on Multimedia Conference*, ser. MM ’17. New York, NY, USA: ACM, 2017, pp. 1933–1941. [Online]. Available: http://doi.acm.org/10.1145/3123266.3123451

[42] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.

[44] A. Hore and D. Ziou, “Image quality metrics: Psnr vs. ssim,” in *2010 20th International Conference on Pattern Recognition*, Aug 2010, pp. 2366–2369.

[45] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley, 2006.

[46] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. San Diego, CA, USA: Academic Press Professional, Inc., 1990.

[47] F. Ebrahimi, M. Chamik, and S. Winkler, “Jpeg vs. jpeg2000: An objective comparison of image encoding quality,” 08 2004.

[48] D. Vigliano, R. Parisi, and A. Uncini, *Video Compression by Neural Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 205–234. [Online]. Available: https://doi.org/10.1007/3-540-32367-8_10

[49] R. Talluri, K. Oehler, T. Barmon, J. D. Courtney, A. Das, and J. Liao, “A robust, scalable, object-based video compression technique for very low bit-rate coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 221–233, Feb 1997.

[50] C. Feller, J. Wuenschmann, R. Wagner, and A. Rothermel, “Model-based video compression for real world data,” in *2013 IEEE Third International Conference on Consumer Electronics ¿¿ Berlin (ICCE-Berlin)*, Sept 2013, pp. 210–214.

[51] M. Wang and C.-H. Lai, “A hybrid fractal video compression method,” *Computers & Mathematics with Applications*, vol. 50, no. 3, pp. 611 – 621, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0898122105002683

[52] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, “Video coding with h.264/avc: tools, performance, and complexity,” *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, First 2004.

[53] J. Jiang, “Image compression with neural networks – a survey,” *Signal Processing: Image Communication*, vol. 14, no. 9, pp. 737 – 760, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0923596598000411

[54] A. Krizhevsky and G. E. Hinton, “Using very deep autoencoders for content-based image retrieval,” in *Using Very Deep Autoencoders for Content-Based Image Retrieval*, 01 2011.

[55] C. Dong, C. C. Loy, K. He, and X. Tang, “Image super-resolution using deep convolutional networks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[56] J. Xie, L. Xu, and E. Chen, “Image denoising and inpainting with deep neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 341–349. [Online]. Available: http://papers.nips.cc/paper/4686-image-denoising-and-inpainting-with-deep-neural-networks.pdf

[57] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*. Springer, 1992.

[58] K. Gregor, F. Besse, D. Jimenez Rezende, I. Danihelka, and D. Wierstra, “Towards Conceptual Compression,” in *Advances In Neural Information Processing Systems*, 2016, pp. 3549–3557.

[59] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[60] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and checkerboard artifacts,” *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[61] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *CoRR*, vol. abs/1601.06759, 2016. [Online]. Available: http://arxiv.org/abs/1601.06759

[62] S. Santurkar, D. M. Budden, and N. Shavit, “Generative compression,” *CoRR*, vol. abs/1703.01467, 2017. [Online]. Available: http://arxiv.org/abs/1703.01467

[63] N. Srivastava, E. Mansimov, and R. Salakhutdinov, “Unsupervised learning of video representations using lstms,” in *International conference on machine learning*, 2015, pp. 843–852.

[64] W. Lotter, G. Kreiman, and D. D. Cox, “Deep predictive coding networks for video prediction and unsupervised learning,” *CoRR*, vol. abs/1605.08104, 2016. [Online]. Available: http://arxiv.org/abs/1605.08104

[65] C. Wu, N. Singhal, and P. Krähenbühl, “Video compression through image interpolation,” *CoRR*, vol. abs/1804.06919, 2018. [Online]. Available: http://arxiv.org/abs/1804.06919

[66] Z. Chen, T. He, X. Jin, and F. Wu, “Learning for video compression,” *CoRR*, vol. abs/1804.09869, 2018. [Online]. Available: http://arxiv.org/abs/1804.09869

[67] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[68] M. Siam, S. Valipour, M. Jägersand, and N. Ray, “Convolutional gated recurrent networks for video segmentation,” in *Image Processing (ICIP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3090–3094.

[69] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li, “Yfcc100m: The new data in multimedia research,” *Commun. ACM*, vol. 59, no. 2, pp. 64–73, Jan. 2016. [Online]. Available: http://doi.acm.org/10.1145/2812802

[70] E. K. Company, “Kodak Lossless True Color Image Suite,” http://r0k.us/graphics/kodak/, 1999, [Online; accessed 09-October-2018].

[71] A. Canziani and E. Culurciello, “Cortexnet: a generic network family for robust visual temporal representations,” *CoRR*, vol. abs/1706.02735, 2017. [Online]. Available: http://arxiv.org/abs/1706.02735

[72] K. Hara, H. Kataoka, and Y. Satoh, “Can spatiotemporal 3D CNNs retrace the history of 2D CNNs and ImageNet?” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA*, 2018, pp. 18–22.

[73] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai, “Deep learning representation using autoencoder for 3D shape retrieval,” *Neurocomputing*, vol. 204, pp. 41–50, 2016.

[74] Z. Qiu, T. Yao, and T. Mei, “Learning spatio-temporal representation with pseudo-3D residual networks,” in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5534–5542.

[75] W.-S. Lai, J.-B. Huang, O. Wang, E. Shechtman, E. Yumer, and M.-H. Yang, “Learning blind video temporal consistency,” in *The European Conference on Computer Vision (ECCV)*, September 2018.

[76] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “Flownet 2.0: Evolution of optical flow estimation with deep networks,” in *IEEE conference on computer vision and pattern recognition (CVPR)*, vol. 2, 2017, p. 6.

[77] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical evaluation of rectified activations in convolutional network,” *CoRR*, vol. abs/1505.00853, 2015. [Online]. Available: http://arxiv.org/abs/1505.00853

[78] J. Ballé, V. Laparra, and E. P. Simoncelli, “Density modeling of images using a generalized normalization transformation,” *CoRR*, vol. abs/1511.06281, 2015. [Online]. Available: http://arxiv.org/abs/1511.06281

[79] C. Feichtenhofer, A. Pinz, and R. P. Wildes, “Spatiotemporal residual networks for video action recognition,” in *Advances in neural information processing systems*, 2016, pp. 3468–3476.

[80] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, “Youtube-8m: A large-scale video classification benchmark,” *CoRR*, vol. abs/1609.08675, 2016. [Online]. Available: http://arxiv.org/abs/1609.08675

[81] A. Puri and A. Eleftheriadis, “Mpeg-4: An object-based multimedia coding standard supporting mobile applications,” *Mob. Netw. Appl.*, vol. 3, no. 1, pp. 5–32, Jun. 1998. [Online]. Available: http://dx.doi.org/10.1023/A:1019160312366

[82] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the h.264/avc video coding standard,” *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003. [Online]. Available: https://doi.org/10.1109/TCSVT.2003.815165

[83] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong, “O-CNN: octree-based convolutional neural networks for 3d shape analysis,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 72, 2017.

[84] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial transformer networks,” in *Advances in neural information processing systems*, 2015, pp. 2017–2025.

A

Prediction based Video Compression

A.1 Problem Formulation

The compression scheme described in detail in this section is based on the continued success of the frame prediction/motion estimation architectures in the past decades.

The reasoning behind these architectures is that there is no need to encode each frame separately, as a lot of repeated information would be stored in that case. To minimize the information needed to reconstruct each frame, only the information that has not been previously stored needs to be encoded. Thus, the most important step of these architectures is the ability to represent a frame based in previously encoded information.

This is usually achieved by estimating the motion in video sequences, which, althought uncommon, can be done at a frame level. The more common approach is to process frames in smaller blocks (to simplify the calculations). For each block, a correspondence is found, to an already encoded block, which maximizes the contained redundant information: the block estimation. It is commonly referred to as block motion estimation since the matching block is usually obtained by isolating the movement of the original block throughout the frames.

A similar scheme can be framed/formalized in the context of DL. Consider the case of one frame being encoded in the middle of a sequence of frames $f = (f_1, f_2, \ldots, f_N) \in \mathcal{F}^N$, where $f_i \in \mathcal{F}$, with $\mathcal{F}$ denoting the space to which each frame belongs (*e.g.*, $[0; R]^{W \times H \times 3}$, for RGB frames with $(W \times H)$ pixels, and a maximum pixel value of R). First, it is necessary to construct an initial frame estimation or prediction, which will be referred as $\tilde{f}_i$ that best approximates the frame f_i, based on the information that is already encoded in the previous frames $f_{i-1}, f_{i-2}, ..., f_1$ so as to minimize the residual error $\mathrm{E}_r(\tilde{f}_i, f_i)$. As predicting the whole frame is computationally difficult, it may be easier to divide the task in sub-problems. In that case, the frame f_i is split in k blocks, so that $f_i = (b_{i1}, b_{i2}, ..., b_{ik})$. The

sub-divided objective is to find a set of blocks $(\tilde{b}_{i1}, \tilde{b}_{i2}, ..., \tilde{b}_{ik})$, so as to minimize the sum of errors $\sum_j \mathsf{E}(b_{ij}, \tilde{b}_{ij})$.

The second step, after obtaining the block predictions, is to use the least amount of information possible to reconstruct the block. Depending on the error for each predicted block j of frame i, $\mathsf{E}(b_{ij}, \tilde{b}_{ij})$ (the residual), one of three actions are needed:

- If the error is almost non-existent, the predicted block can be used as replacement for the original block
- If the error is small, the original block can be reconstructed from the predicted block plus a residual. Only the residual needs to be compressed
- If the error is substantial, the original block must be fully compressed. In this case, compressing the residual would cost as much, or more, than compressing the whole block from zero.

.

The only case that is not contemplated in the previous description is the compression of a frame when there is no context available (*e.g.*, the first frame). In this case the simple approach is to encode the frame as a single image, disregarding any potential temporal redundancy.

The described procedures presents the necessary steps for a video compression scheme, by using a prediction architecture. Two possibilities are formulated which are conceptually similar: frame prediction or block prediction (motion estimation). From an high-level perspective, it uses the same reasoning behind the traditional video codecs, albeit a simplified version. At the same time, the whole problem of video compression is separated in convenient tasks that can be implemented using DL.

A.2 Architecture

One initial studied approach for this problem employed a motion estimation method based on a DNN for frame prediction. That network used past information so that, given a sequence of previously compressed frames, one or a few frames into the future were predicted (using a frame prediction architecture as discussed in Section 2.5.2). Even though initial results were promising, the frame prediction had two major problems: it could not generalize properly outside the training dataset, and the produced frames were generally blurry.

Thus, from the two presented formulations in the previous section, the block motion estimation is chosen instead of the frame prediction approach. A comparison between the results obtained using frame prediction and block prediction is presented in the Section A.3.2 to provide some backing to the choice made here.

To tackle the chosen problem formulation, a compression scheme is proposed that processes each frame in blocks of size $n \times n$ pixels, with $n = 32$. In total, it uses four modules to achieve compression: a Frame Compression module for single frame compression, a Block Predictive Coding module for motion vector prediction of each block, a Block Residual Compression module to encode the residuals of motion estimation and a Block Absolute Compression module to code motion miss-predictions.

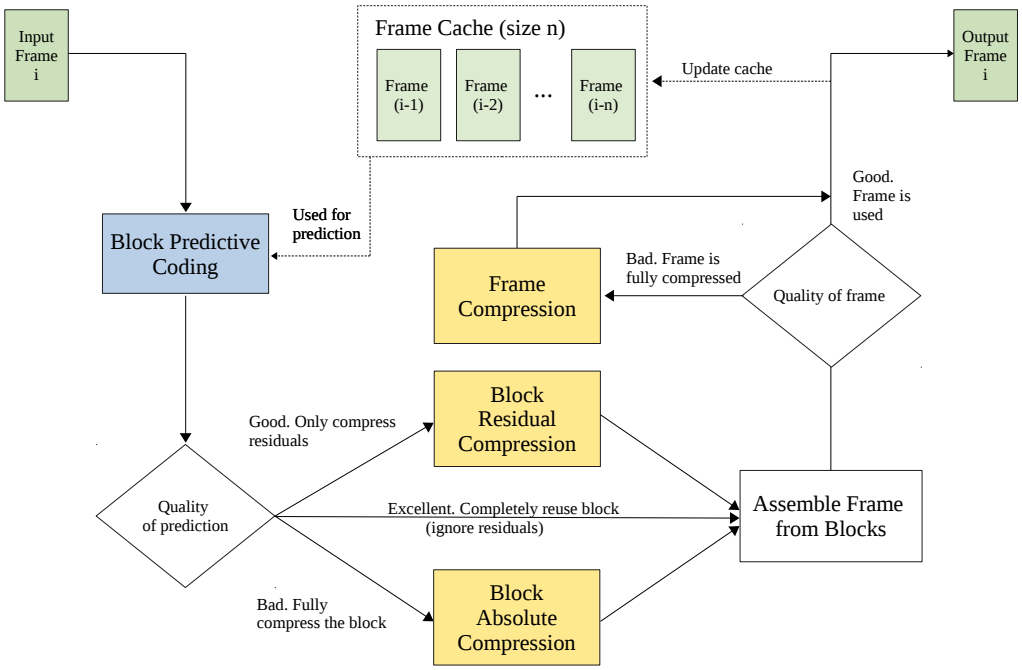


Figure A.1: The general work-flow of the proposed architecture for video compression with DL using four modules: block prediction, block compression, frame compression and residual compression. Modules in yellow are implemented as neural networks.

The four modules fit within the previously presented formulation. The Frame Compression module is used to encode full frames when no previous information is available, or when block processing is not worth it; the Block Prediction module is used to produce a prediction for each block given a context; the Block Absolute Compression module is used when there is a need to fully compress a block; and the Residual Block Compression module is used to encode the residuals between two blocks. The implementation chosen for each module and the decision process behind each choice is explained in the next section of this appendix. An illustration of how the proposed architecture works is provided in Figure A.1.

The envisioned architecture always processes the first frame using the Frame Compression module, as the frame cache is not populated (as necessary to run Block Prediction). The remaining frames are processed in blocks. Each block is predicted by the Block Prediction module using the frame cache, with previously encoded frames as contextual information. Depending on the quality of the block prediction, one of three processes is chosen: fully compress the block (using the Block Absolute Compression module), compress the residual between the block and the prediction (using the Block Residual Compression module), or the predicted block is simply repeated as the new block and the residual ignored.

The choice is made depending on a quality threshold, with the last case being the optimal case of an almost perfect prediction. It is partially responsible for controlling the bitrate of the compression

scheme, as each option has different bitrate implications. For each frame, all blocks are processed in this manner, and the frame is then recreated by assembling the blocks together. A second threshold is used to decide if the produced frame has a good quality/size ratio; in the negative case, the per-block encoding is discarded and the frame is compressed again using the frame compression network.

The frame cache is updated with the newest compressed frame, and an older is discarded if necessary. The length of the frame cache impacts both the quality of the reconstruction and the runtime of the encoding process. By increasing the number of available frames in the cache, the probability of finding a better match increases, but the search space also increases, which impacts the number of iterations of the prediction algorithm. For this implementation, the maximum length used for the cache is 4 frames. At higher cache sizes, during experimentation, a soft threshold was reached, where increasing this size any further had an increasing impact in runtime, without improving the performance accordingly.

This process of per-block processing is repeated for all frames within the sequence to encode, and the output is created by concatenating all the processed frames back into a sequence. Both the thresholds used in this compression scheme must be determined empirically, in order to obtain a good balance between an acceptable quality of the produced block/frame, and fitting the constraints of the desired bitrate.

A.3 Modules

A.3.1 Frame Compression Encoder

The frame encoder works independently and is based on the best performing method during the research in image compression (in Chapter 3), particularly the image compression network described in Section 3.6. The details of the hyper-parameters of this architecture are stated on that same section, in Figure 3.7, and are derived from the work of Ballé et al. [4]. The reasoning behind choosing this architecture for this module was a simple consequence of the results obtained in Section 3.7, in which this architecture is, by far, the one with best performance for image compression. As it is suitable for both low and high bitrates, it is flexible enough for the task of frame compression. It also has the advantage of generating natural-looking artifacts compared to other image compression schemes, which is a necessity for low bitrate video compression.

A.3.2 Prediction

The objective of the frame predictor is to create a frame prediction $\tilde{f}_i$ that best approximates the frame f_i, based on the information that is already encoded in the previous frames $f_{i-1}, f_{i-2}, ..., f_1$ so as to minimize the residual error $\mathrm{E}_r(\tilde{f}_i, f_i)$. Another possible formulation is to split the frame f_i in k blocks, so that $f_i = (b_{i1}, b_{i2}, ..., b_{ik})$. The new prediction task is finding a set of blocks $(\tilde{b}_{i1}, \tilde{b}_{i2}, ..., \tilde{b}_{ik})$, so as to minimize the sum of errors $\sum_j \mathrm{E}(b_{ij}, \tilde{b}_{ij})$.

There were two considered possibilities, one for block motion prediction and one for full frame prediction: a window-based algorithmic search as in the traditional codecs (although a simpler version),

or a prediction scheme based on a neural network, respectively. In the remainder of this sub-section the two approaches will be described and a brief comparison is presented to justify the advantages and disadvantages of each of the two methods. Both considered schemes rely on the frame cache, a concept that was already explained in the previous section. Note that the decision of this section influeced the overall architecture of the compression scheme (frame prediction vs block prediction), and the implementantion for the network responsible for encoding residuals.

A Window-based search

The window-based approach searches the cache of previously encoded frames for the closest matching block given the $L1$ distance between each prediction and the input block. The search is done within a window of width and height s, centered on the position of the original block, in each one of the cached frames. The search is done with a one pixel stride, and all the 27×27 possibilities are compared to the original block until the search space is exhausted. The best prediction is the block that minimizes the used criteria. For this approach, $s = 27$ was chosen so as to balance the quality of the results with the needed runtime.

B Neural Network Based

The problem of frame prediction in video compression might seem exactly the same as the video frame prediction problem, described in Section 2.5.2. However there is an important difference from the formulation in the research: the predictions in regular frame prediction are always made from the previous ground truth frames of the video; in comparison, for video compression, these predictions can only take into account the previously encoded frames, which have a significant associated error. Due to the nature of how neural networks work, this error is prone to being propagated from the previous frames to the new generated frame, which has has an aggravated impact of the final prediction.

Still, an attempt was made to solve this task using DL-based methods as well. This task is based on processing sequences of frames and so, is intuitively suited for recurrent networks, particularly the ones that can model spatio-temporal dependencies (see Part G of Section 2.2.5). The recent research has been mostly focused on the use of RCNNs for this task, by processing the first frames of the sequence so as to extract temporal patterns, and then predicting one or a few frames into the future. In this case, RCNNs are prefered to 3DCNNs due to stronger capabilities of detecting patterns in regular time intervals, when there is no need for efficiency. This is the scheme followed in the research by Lotter et al. [64], which propose the PredNet architecture. It tries to improve on the usual recurrent network structure with additional connections between iterations that send back the predictive error of the local frame prediction, following the principles of predictive coding in neuroscience literature. The result is a recurrent network that can, to some degree, predict the movement of the camera and objects in the scene. However, the produced frames are notably blurry even in zones of the video that are unchanged, which is unwanted in video compression as the overall quality of the image decreases. A more recent approach by Canziani and Culurciello [71], referred to

as CortexNet, uses a similar recurrent network architecture augmented with top-down feedback and lateral connections added to the regular bottom-up feed-forward connections, inspired by the human visual cortex system. This network achieves better results in the task of frame prediction, but still suffers, to a lesser extent of the problems noted for PredNet. As the CortexNet displayed the best results from the investigated architectures it was chosen as the DL-based attempt to frame prediction. It was trained on a set of sequential frames extracted from a collection of 10^3 videos of the Youtube 8M dataset [80], using a collection of compressed frames mimicking the frame cache as contextual information.

C Comparison

A simplified evaluation is done on a disjoint subset of the Youtube 8M dataset using twelve videos, by using each network to predict complete frames of the selected sequences. In the case of the frame prediction, this means one iteration of the network; in the case of the block motion prediction this means predicting all blocks of each evaluated frame. In this comparison, the DL-based approach reaches an average of 0.8913993 MS-SSIM, compared to the 0.9317033 of the brute-force search.

From this comparison and by looking at the results individually it is possible to conclude that the actual level of research for using neural networks in frame prediction, while great for tasks such as predicting the movement of an object, is still not good enough to justify using these networks in the context of video compression. The quality deteriorates quickly, especially when there is already an error on the frames used for prediction. Additionally, the two networks tested were very sensitive to the training data, and had a difficult time in generalizing to new data. Thus, while they usually achieve good results within a specific objective and a predictable dataset (*e.g.*, vehicle motion), as they are usually tested, they are unsuited to the task of generalized frame prediction for video compression.

The results leave no question that, for this particular use, the window-based search is much better as the blocks from previous frames are re-used without introducing or propagating more error than the one already present after encoding. As it is a brute-force approach, this window-based search has a significant downside: the runtime for a single block is almost as high as the whole frame prediction by the network. Considering the number of blocks in a high resolution video, the runtime of the whole architecture is significantly increased by the prediction stage.

A.3.3 Block Absolute Compression

The block encoder, being an image compression network, is also based on the best performing image compression network described in Section 3.6. However, the blocks to be encoded by this network are particularly small, and require some changes to the network's hyper-parameters to best suit the block size of 32x32 pixels used in this video compression architecture. Particularly, the kernel sizes were all reduced to a dimension of 3×3, and the number of channels was adjusted accordingly to the pretended bitrate for each model. This network is trained on 10^6 32x32 pixel crops extracted at random from frames of videos of the Youtube 8M dataset [80].

A.3.4 Block Residual Compression

The residual block encoder as a module has the objective of encoding in a compressed representation the residual $\tilde{b_{ij}} - b_{ij}$ for each block b_{ij} of a particular frame f_i, so as to minimize an error function $E(\tilde{b_{ij}}, b_{ij})$, where $\tilde{b}_{ij}$ is the predicted block by the block prediction module.

Since the residual is the difference, or L1 norm, between the predicted and the actual block, by extension it is assumed to have some sort of spatial structure, similarly to images. Residual structure and analysis is a complex topic on itself that would require a lot of effort to properly break-down; in the context that this topic is presented, which is not even the final solution of the thesis, the intricate discussion of theoretical properties of residuals is skipped. Instead, a summary of what should be expected is presented.

Residuals have two main properties besides spatial structure that apply and are relevant for the task in hand: most of the information contained in the residual is concentrated in a small number of positions, and the remaining positions are overwhelmingly close to 0. Thus the mean and standard deviations of residuals do not fit well into the expected zero mean and standard deviation of one that networks can learn efficiently.

Given the spatial structure of the residuals, it is still reasonable that a convolutional network is suitable to explore the redundancy in residuals and to extract meaningful information in the same way as an image compression network. However some adjustments have to be made; namely, the residuals are normalized to an approximate mean of zero and standard deviation of one using a transformation based on the average values of the training dataset. The transformation is inverted at the output of the residual compression network. The normalization parameters are fixed throughout training and evaluation.

The compression architecture for this module is based on the RCNN network presented in D due to the objective of the network. The quality of the predicted block is quite variable; thus, by using a variable compression scheme it is possible to adjust the number of iterations needed, with a single model, to attain a consistent target quality for all blocks, without wasting bitrate in residuals that are compressed more easily.

This network is trained by extracting residuals from the combination of the image compression network and the block predictor for arbitrary two-frame sequences within 10^3 videos of the Youtube 8M dataset. The bottleneck size was adjusted to use less bitrate in the initial iterations, ranging from 0.03125 to 0.5 BPP.

B

Open-sourced Implementations

All the open-sourced implementations developed in the context of this thesis use the PyTorch deep learning framework. Due to the broad range of implementations, they are divided in several repositories, all of which are presented below. The detailed instructions and structure for each repository in detail is available by accessing the README in the respective root directory. This information is not included here due to the immutability of the thesis, and the possibility of changes that may invalidate it.

B.1 Image Compression

The four implemented image compression architectures are available at the GitHub repository `https://github.com/jorge-pessoa/pytorch-image-compression`. In no particular order, the available implementations are:

- A Convolutional Autoencoder by Toderici et al. [2]
- A Recurrent Convolutional Autoencoder by Toderici et al. [3]
- An adversarially trained Convolutional Autoencoder by Rippel and Bourdev [1]
- A rate-distortion optimized Variational Autoencoder by Ballé et al. [4]

The repository includes the necessary scripts for training and testing each architecture.

B.2 Video Compression

The implementation for the proposed spatio-temporal autoencoder for video compression is available at the GitHub repository `https://github.com/jorge-pessoa/video-compression-dl`.

The repository contains the implementation of the presented architecture and all related techniques (not guaranteed to correspond exactly to the formulation in the thesis), and the necessary scripts for training and testing.

B.3 Metrics

For the optimization and evaluation using the MS-SSIM metric, a differentiable version was implemented in PyTorch. Due to the importance of the metric, it was included in an individual GitHub repository, available at `https://github.com/jorge-pessoa/pytorch-msssim`.