Deep Image and Video Compression

André Nortje

Report submitted in partial fulfilment of the requirements of the module
Project (E) 879 for the degree Master of Engineering (MEng) in the Department of Electrical
and Electronic Engineering at Stellenbosch University.

Supervisors: Dr H. Kamper & Prof. H.A. Engelbrecht

March 2020

Acknowledgements

I would sincerely like to thank,

- My family for their support and encouragement.
- Prof. H.A. Engelbrecht and Dr H. Kamper for their mentorship and guidance.

"There can be no doubt that the dream working has resulted in an extraordinary compression."

Sigmund Freud, Dream Psychology

UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Plagiaatverklaring / Plagiarism *Declaration*

1 Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.
Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2 Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.
I agree that plagiarism is a punishable offence because it constitutes theft.

3 Ek verstaan ook dat direkte vertalings plagiaat is.
I also understand that direct translations are plagiarism.

4 Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.
Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.

5 Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aangedui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.
I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Abstract

Forecasts indicate that video will make up 82% of all Internet traffic by 2022. Advancing video compression efficiency will play a crucial role in curbing high bitrates and mitigating excessive bandwidth consumption. To this end, recent deep learning models are emerging as likely successors to hand-tuned standard video codecs.

Our goal is to further refine the compression quality of existing video codecs by improving their ability to predict video content. We subdivide video compression into two focus areas:

1. Still image compression of video frames, for which we propose the Binary Inpainting Network (BINet).

2. Motion compression in video, for which we learn binary motion codes (P-FrameNet and B-FrameNet).

With BINet we learn to inpaint an image patch from the binary codes of its nearest neighbours to better compress a still image or single video frame (intra-frame compression). We adapt BINet to perform inter-frame prediction with P-FrameNet and B-FrameNet by learning binary motion codes that compensate for the relative displacement undergone by objects in a video sequence across time. Within the context of video compression our prediction methods are, to the best of our knowledge, the first fully parallelisable means of video intra-frame and inter-frame prediction.

We show how inclusion of the BINet framework improves the intra-frame compression of a competitive deep image codec across a range of bitrates such that it outperforms the standard image codec JPEG. Experiments also highlight that its full-context patch inpaitings are of a higher quality than those sequentially predicted by the standard image codec WebP. In terms of inter-frame video prediction, we show that our learned binary motion codes describe more complex motion than the block-based optical flow algorithms employed by the standard video codecs: H.264 and H.265. This indicates that the BINet and our learned binary motion codes could be valuable extensions to existing video codecs, specifically in improving their intra-frame and inter-frame compression capabilities.

Opsomming

Voorspellings dui daarop dat video teen 2022, 82% van alle internetverkeer sal uitmaak. Die bevordering van videokompressie doeltreffenheid sal 'n belangrike rol speel in die bekamping van hoë bitrates en die vermindering van buitensporige bandwydte verbruik. Met die oog hierop verskyn die onlangse diepleermodelle as waarskynlike opvolgers vir die standaard handgestemde videokodekse.

Ons doel is om die kompressiekwaliteit van bestaande videokodekse verder te verfyn deur hul vermoë om video-inhoud te voorspel, te verbeter. Ons verdeel videokompressie in twee fokusareas:

1. Stilbeeldkompressie van videorame, waarvoor ons die 'Binary Inpainting Network' (BINet) voorstel.
2. Bewegingskompressie in video, waarvoor ons binêre bewegingskodes leer (P-FrameNet and B-FrameNet).

Deur die gebruik van BINet, leer ons om 'n beeldpatroon uit die binêre kodes van sy naaste bure te 'inpaint' om 'n enkele videoraam (kompressie binne raam) beter saam te druk. Ons pas BINet aan om interraamvoorspellings uit te voer met P-FrameNet en B-FrameNet deur binêre bewegings kodes te leer wat kompenseer vir die relatiewe verplasing wat deur voorwerpe in 'n videosekwensie oor tyd heen ondergaan word. BINet is binne die konteks van videokompressie, na die beste van ons wete, die eerste volledige parallelle middle van voorspelling van videorame.

Ons bewys hoe die insluiting van die BINet-raamwerk die kompressie binne die raam van 'n mededingende diepbeeldkodek oor 'n reeks bitrates verbeter sodat dit die standaard-beeldkodek JPEG oortref. Eksperimente beklemtoon ook dat die volledige konteks van kol 'inpaintings' van hoër gehalte is as dié wat opeenvolgend voorspel word deur die standaard-beeldkodek WebP. In terme van voorspelling tussen raamwerke, toon ons aan dat ons aangeleerde binêre bewegingskodes meer ingewikkelde beweging beskryf as die blokgebaseerde optiese vloei-algoritmes wat gebruik word deur die standaard-videokodekse: H.264 en H.265. Dit dui daarop dat die BINet en ons aangeleerde binêre bewegingskodes waardevolle uitbreidings vir bestaande videokodekse kan wees, veral om hul binne-raam en interraam kompressievermoë te verbeter.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

$\boldsymbol{x}$	Matrix, tensor or pixel array filled with scalar element $x_{c,h,w}$ at position (c, h, w).
$\boldsymbol{x}^\top$	Transpose of matrix $\boldsymbol{x}$.
$\frac{dy}{dx}$	The derivative of y with respect to x.
$\frac{\partial y}{\partial x}$	The partial derivative of y with respect to x.
$\nabla \boldsymbol{x}$	Multivariate derivative of $\boldsymbol{x}$.
$\vec{V}$	Directional or optical flow vector.
$E(\cdot)$	Encoder function.
$D(\cdot)$	Decoder function.
$\mathrm{Auto}(\cdot)$	Autoencoder function with encoding and decoding steps.
Δt	Difference in t.
$\sum$	Summation.
μ	Statistical mean.
$\mathrm{E}[\boldsymbol{x}]$	Expectation of the random variable $\boldsymbol{x}$.
σ	Statistical variance.
$\sigma(\cdot)$	Sigmoid activation function.
$\tanh(\cdot)$	Hyperbolic tangent activation function.
$\otimes$	Deep learning convolution operator.
$\star$	Cross correlation operator.
∞	Infinity.
$\simeq$	Approximately equal to.
$\sim$	Approximated by.
$\lvert x \rvert$	Absolute value of x.
$\boldsymbol{x} \cdot \boldsymbol{y}$	Dot product of vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.
$\lvert\lvert \boldsymbol{x} \rvert\rvert$	Magnitude of vector $\boldsymbol{x}$.
$\lfloor \cdot \rfloor$	Floor operator.
$\lceil \cdot \rceil$	Ceiling operator.

L_1	Sum of all the absolute differences between the target and predicted values.
L_2	Sum of all the squared differences between the target and predicted values.
$\hat{x}$	An approximate reconstruction of x.

Acronyms and abbreviations

AC	Alternating Current
Adam	Adaptive Moment
AI	Artificial Intelligence
a.k.a	also known as
ANN	Artificial Neural Network
AR	Additive Reconstruction
ARPS	Adaptive Rood Pattern Search
AVC	Advanced Video Coding a.k.a H.264
AUC	Area Under Curve
bits	binary units
B-Frame	Bi-directional Frame
BINet	Binary Inpainting Network
bpp	bits-per-pixel
CLIC	Challenge on Learned Image Compression
CNN	Convolutional Neural Network
conv-layer	convolution-layer
CODEC	enCOder DECoder
CPU	Central Processing Unit
CRF	Constant Rate Factor
CUDA	Compute Unified Device Architecture
DBA	Dynamic Bit Assignment
DC	Direct Current
DCG	Dynamic Computation Graph
DCT	Discrete Cosine Transform
DLM	Detail Loss Metric
DNN	Deep Neural Network
docs.	Documentation
DS	Diamond Search
DWT	Discrete Wavelet Transform
EPE	End Point Error
ES	Exhaustive Search

ESPCN	Efficient Sub-Pixel Convolutional Network
FFmpeg	Fast Forward MPEG
FSS	Four Step Search
GB	Gigabyte
GCP	Google Cloud Platform
GAN	Generative Adversarial Network
GOP	Group Of Pictures
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HD	High Definition
HEVC	High Efficiency Video Coding a.k.a H.265
HR	High Resolution
HSV	Hue Saturation Value
HVS	Human Visual System
IDCT	Inverse Discrete Cosine Transform
I-Frame	Intra-Frame
I/O	Input/Output
JPEG	Joint Photographic Experts Group
LR	Low Resolution
LSTM	Long Short Term Memory
MJPEG	Motion JPEG
MPEG	Motion Picture Experts Group
MSE	Mean Square Error
MV	Motion Vector
MVD	Motion Vector Difference
MVP	Motion Vector Predictor
NTSS	New Three Step Search
NVVL	NVIDIA Video Loader
OSR	One-Shot Reconstruction
PCA	Principal Component Analysis
P-Frame	Predicted Frame
PIL	Python Imaging Library

PNG	Portable Network Graphics
PSNR	Peak Signal to Noise Ratio
PSNR-HVS	Peak Signal to Noise Ratio - Human Visual System
ReLU	Rectified Linear Unit
RAM	Random Access Memory
RGB	Red Green Blue
RLE	Run Length Encoding
RNN	Recurrent Neural Network
SES	Simple and Efficient Search
SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SR	Super Resolution
SSIM	Structural SIMilarity
STE	Straight Through Estimation
SVD	Singular Value Decomposition
SVM	Support Vector Matrix
TanH	Hyperbolic Tangent
TSS	Three Step Search
VCS	Version Control System
VIF	Visual Information Fidelity
VLC	Variable Length Coding
VMAF	Video Mutli-method Assessment Fusion
VTL	Video Trace Library
YCrCb	Luminance Chroma-red Chroma-blue a.k.a YUV
2D	Two Dimensional
3D	Three Dimensional

Chapter 1

Introduction

Video compression attempts to reduce the number of bits needed to represent a video file without compromising its perceived quality. Without compression, HD video files can easily run over 300 GB, making scalable streaming and storage next to impossible. Tweaking the tradeoff between bitrate and video quality is an ever present problem in video codec design.

1.1. Related Work

Standard video codecs, that have dominated compression for the past decade, are meticulously hand-engineered and lack end-to-end optimisation [1, 2]. Lately, deep learning has spearheaded the development of state-of-the-art video compression systems [3–5]. These systems typically consist of an image codec for compressing reference frames and an inter-frame prediction model that predicts a set of target frames from the reference frame content [6].

Deep neural networks trained through loss-driven end-to-end optimisation have been shown to outperform standard image codecs such as JPEG and WebP [7–18]. These deep image codecs are trained to compress full-sized images and applying them on a patch-by-patch basis requires that each image patch be encoded and decoded independently. The structural influence imposed by pixels from adjacent patches is therefore lost, which can cause block artefacts at low bitrates. Patch-based encoding schemes are, however, preferred to their full-resolution counterparts, as they are more memory efficient [6]. Intra-frame prediction improves patch-based compression by reinstating the structural ties between independently encoded patch regions. Currently, intra-frame prediction is dominated by sequential inpainting techniques [19, 20], where previous patch decodings within an image (single video frame) are used to predict a basis for a target patch region.

Up to now optical flow has enabled inter-frame prediction in standard video codecs such as H.264 [1] and H.265 [2] as well as deep video compression systems [3–5]. Optical flow vectors describe how pixels in a video frame should be translated spatially over time to best estimate true object and camera motion [21]. After the transmission of a reference video frame, which has been compressed independently by an image codec, only highly compressible optical flow vectors need be transmitted to motion-compensate pixels in the reference frame to form predictions of the subsequent frames within a video sequence.

1.2. Problem Statement

Despite these recent advances in deep image and video compression, we recognise shortcomings in the way prediction is carried out to aid compression. In intra-frame prediction, image patches are sequentially inpainted from previous patch decodings. This ignores context from future patches and prevents parallelised prediction. Existing video codecs rely on optical flow based inter-frame prediction, where motion vectors spatially shift pixels in past frames to predict future frames. Although effective at capturing translational motion, optical flow fails to express complex motion transforms such as warping, occlusion, rotation and colour shift. Optical flow based prediction also requires sequential motion estimation and compensation steps, which slows down the compression process.

1.3. Research Objectives

- Research both standard and deep learning based techniques that enable image and video compression in modern codecs, while noting common trends, strengths and weaknesses.
- Implement and experiment with existing deep image codecs to gain experience in the field of deep learning based compression and establish best practices.
- Design and implement innovative intra-frame and inter-frame prediction strategies for fully parallelised video prediction using deep neural networks.
- Demonstrate how these learned intra-frame and inter-frame prediction strategies can be included into existing image and video compression systems to improve their compression efficiency.

1.4. Contributions

- For intra-frame prediction we propose the Binary Inpainting Network (BINet) in Chapter 4. BINet is an autoencoder framework which learns to inpaint a still image patch from the binary encodings of its nearest neighbours. As opposed to sequential inpainting methods where patches are decoded sequentially and previous reconstructions are used to predict subsequent patches, BINet operates directly on the binary codes of surrounding patches without access to the original or reconstructed image data. Both encoding and decoding can therefore be performed in parallel. We show that binary inpainting improves the compression quality of a competitive deep image codec across a range of compression levels, outperforming JPEG. BINet's intra-frame predictions are also shown experimentally to be of a higher quality than the sequential intra-frame inpainting performed by the standard image codec WebP.

- Inspired by BINet, in Chapter 5 we propose P-FrameNet and B-FrameNet to learn binary motion codes for video inter-frame prediction instead of relying on block-based optical flow vectors. Our learned binary motion codes are shown to model more complex motion than flow-based methods and enable our decoder module to perform parallel video frame prediction. Replacing the optical flow based block-motion algorithms in existing video codecs with our learned inter-frame prediction model, we are able to outperform the standard video codecs H.264 and H.265 at low-bitrates.

- Building on recent work in deep image compression we propose learning 3D dynamic bit assignment in Chapter 5 as a way of adapting our deep video codec's binary motion codes across both space and time. This improves compression by allocating more bits to complicated video regions with moving objects and less bits to still video scenes, resulting in a lower bitrate on average.

Our work "BINet: a binary inpainting network for deep patch-based image compression" in Chapter 4 and "Deep motion estimation for parallel inter-frame prediction in video compression" in Chapter 5 is submitted as two separate articles to the *Journal of Visual Communication and Image Representation*, and is currently under review.

1.5. Thesis Outline

This thesis proceeds as follows: First, in Chapter 2 we define various video compression and deep learning concepts that are key to one's comprehension of this work. We then implement and experiment with existing deep image codecs in Chapter 3, to gain insights into the world of deep video compression. The BINet, a novel means of deep image inpainting, is then introduced in Chapter 4 for improved intra-frame video prediction. Building on BINet, in Chapter 5 we learn binary motion encodings with P-FrameNet and B-FrameNet that enhance H.264 and H.265's ability to predict consecutive video frames. Finally, we conclude with notes on future research applications.

Chapter 2

Key Concepts

In this chapter we touch upon various video coding and deep learning concepts to assist with the reader's understanding of this thesis. First off, we introduce the key components found in a typical lossy video codec and explain the associated video coding terminology and concepts. Next, we detail the inner workings of the deep neural network layers that are at the heart of our deep compression systems. Finally, we discuss our software development process, highlighting tools that may prove useful for continued research in this field.

2.1. Video Compression

Digital video is the discrete representation of a 'natural visual scene' that has been sampled both spatially and temporally [22]. Capturing a grid of discrete colour intensity values (pixels) at a specific point in time produces a still image or video frame. A video consists of several images sampled at a time interval known as frame rate. Videos vary spatially in terms of texture, colour and brightness; and temporally due to object or camera motion and changes in lighting [6]. Video quality is proportional to the number of pixels and frame rate used during sampling. A higher number of pixels or resolution improves video frame quality and a higher frame rate depicts smoother motion. As such, High Definition (HD) video contains a large amount of information. Compression attempts to reduce the number of bits needed to represent a video without severely compromising its perceived quality. Compression allows videos to be transmitted without exceeding bandwidth and storage requirements [22]. Typical compression systems make use of an enCOder and DECoder pair termed a CODEC.

Compression can either be lossless or lossy. Lossless compression involves the removal of statistical redundancy from video or image data [6]. This allows for perfect reconstruction at the cost of shallow compression ratios. Lossy compression is irreversible as codecs reconstruct an approximation of the input data based on a low-dimensional encoding [23]. Lossy codec development works towards reducing the tradeoff between the degree of compression afforded by a codec and the quality of its reconstruction. We focus on lossy compression, without which the level of compression required for modern video transmission would be impossible.[1]

[1] This work focusses on video compression for digital storage and transmission; research into sub-sampling techniques used for compressed sensing [24] during video capture is left to future work.

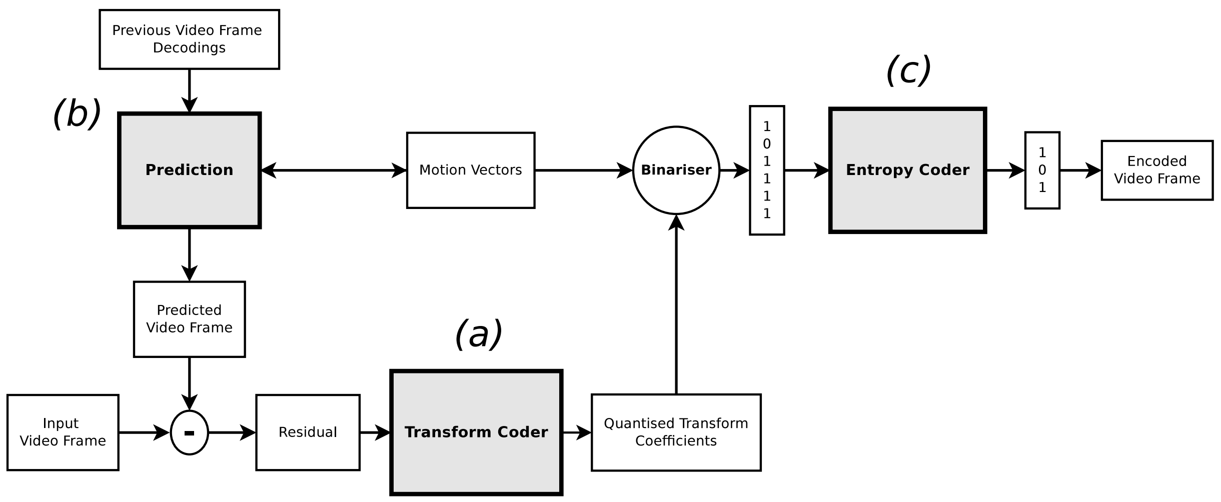


Figure 2.1: The key components that make-up a typical video compression system. Here we show the interplay between transform coding *(a)*, prediction *(b)* and entropy coding *(c)* sub-systems that together form a video codec.

2.1.1. Video Codec

A lossy video codec typically consists of prediction, transform and entropy coding sub-systems as depicted in Figure 2.1 [6].

Transform Coding

Transform coding, indicated by *(a)* in the figure, involves transforming image or image residual data into a transform space that separates the signal into its uncorrelated base frequencies. The transform must ensure that the bulk of the signal's energy is concentrated over as few fundamental components as possible, so that higher frequencies can be quantised or removed without significantly affecting the image signal's quality [25]. Prominent block-based transforms include the Discrete Cosine Transform (DCT) and Singular Value Decomposition (SVD) [25]. The Discrete Wavelet Transform (DWT), which takes a whole image as input, has been shown to outperform the DCT as it is less prone to block artefacts. However, it requires significantly more memory for processing and does not allow for block-based motion estimation/compensation apparent in most standard video codecs [6]. We implement existing deep image codecs for transform coding in Chapter 3.

Prediction

The prediction system shown at *(b)* is used to infer the pixel values of specific patch regions called macroblocks based on that of their surroundings. The error between the predicted macroblock and the original is termed a residual. Only this residual error need be transmitted to the decoder. It is important to note that in most compression systems the decoder is used during the encoding process to assist with the prediction and formation of residuals.

Video codecs employ two forms of block-based prediction: intra-frame (spatial prediction) and inter-frame (temporal prediction) [22]. Intra-frame prediction aims to predict each macroblock's content from neighbouring patches that have already been decoded within a single video frame. In inter-frame prediction motion vectors guide the movement of macroblocks across time to predict future frames from those that have previously been decoded [1]. Improved prediction quality results in the transmission of lower energy residuals that are more easily compressible. In Chapter 4 we submit learned binary inpainting: a novel means of deep intra-frame prediction. We then build on this to perform motion guided inter-frame prediction in Chapter 5.

Entropy Coding

Entropy coding at *(c)* involves the lossless conversion of the compression values that need to be transmitted to the decoder (transform coefficients, motion vectors, etc.) into a serial bitstream. This conversion is accomplished using Variable Length Coding (VLC) strategies, like Huffman Encoding [26], to exploit the statistical redundancy that exists between the compressed values. The basic premise of VLC is to assign longer bit codes to values that occur less often and shorter codewords to the most prominent compression values [6]. Optimising the entropy of our models' codes is left to future research.

2.1.2. Optical Flow

Optical flow estimates true object and camera motion by describing the displacement of pixels in video frames over time [21]. Brightness constancy is assumed [21], i.e. the pixel intensity $\boldsymbol{I}(x, y, t)$ at the point (x, y) in a video frame sampled at time t does not change when it moves,

$$\boldsymbol{I}(x, y, t) = \boldsymbol{I}(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.1}$$

Taylor expansion of the r.h.s of equation (2.1) yields the optical flow equation,

$$\frac{\partial \boldsymbol{I}}{\partial x}\vec{V}_x + \frac{\partial \boldsymbol{I}}{\partial y}\vec{V}_y + \frac{\partial \boldsymbol{I}}{\partial t} = 0, \tag{2.2}$$

where $\vec{V}_x = \frac{dx}{dt}$ and $\vec{V}_y = \frac{dy}{dt}$ are the x and y motion vector components of $\boldsymbol{I}(x, y, t)$ that indicate the movement of the pixel between two video frames sampled Δt seconds apart.

We can calculate the image gradients, $\frac{\partial \boldsymbol{I}}{\partial x}$, $\frac{\partial \boldsymbol{I}}{\partial y}$ and $\frac{\partial \boldsymbol{I}}{\partial t}$, as the entire video surface is assumed smooth and differentiable [21]. This assumption ignores occlusion, where overlapping object edges introduce discontinuity. To solve for the two unknowns, $\vec{V}_x$ and $\vec{V}_y$, we require additional constraints.

Block-based optical flow methods assume that neighbouring pixels undergo similar motion [27, 28]. Applying this assumption, on say a 2×2 pixel region, we get four variants of equation (2.2) that can be solved iteratively for $\vec{V}_x$ and $\vec{V}_y$ [27].

Farneback approximates a group of adjacent pixels, $f_t(\boldsymbol{x})$, as a quadratic polynomial [28],

$$f_t(\boldsymbol{x}) \approx \boldsymbol{x}^\top \boldsymbol{A}_t \boldsymbol{x} + \boldsymbol{b}_t^\top \boldsymbol{x} + c_t. \tag{2.3}$$

The symmetric matrix $\boldsymbol{A}_t$, vector $\boldsymbol{b}_t$ and scalar c_t are obtained via a least squares fit to the input pixel data [29]. The ideal translation or displacement, $\boldsymbol{d}$, of this pixel region after Δt seconds is

$$\begin{aligned} f_{t+\Delta t} &= f_t(\boldsymbol{x} - \boldsymbol{d}) \\ &= (\boldsymbol{x} - \boldsymbol{d})^\top \boldsymbol{A}_t (\boldsymbol{x} - \boldsymbol{d}) + \boldsymbol{b}_t^\top (\boldsymbol{x} - \boldsymbol{d}) + c_1 \\ &= \boldsymbol{x}^\top \boldsymbol{A}_{t+\Delta t} \boldsymbol{x} + \boldsymbol{b}_{t+\Delta t}^\top \boldsymbol{x} + c_2. \end{aligned} \tag{2.4}$$

By equating the coefficients in equations (2.3) and (2.4) (brightness constancy assumption [21]), we can solve for the displacement vector:

$$\boldsymbol{d} = -\frac{1}{2}\boldsymbol{A}_t^{-1}(\boldsymbol{b}_{t+\Delta t} - \boldsymbol{b}_t), \tag{2.5}$$

assuming $\boldsymbol{A}_t$ is non-singular [28].

In Chapter 5 we compute the dense (per-pixel) optical flow between our compressed video frames with OpenCV's default implementation of Farneback's polynomial expansion method [28] and LiteFlowNet—a pre-trained deep flow estimator [30].

2.1.3. Colour Spaces

A colour space is a mathematical model that represents 'true colour' with a range of numerical values [22]. Monochrome (black and white) images require a single value to express the brightness of each sampled pixel, while colour images require at least three per pixel position [6]. Colour spaces used in the field of video compression include RGB and YCrCb. The RGB colour space uses three integer values in the range of 0-255 to indicate the relative strengths of Red, Green and Blue (RGB) light that constitute a colour sample. Digital displays illuminate each RGB component separately, with an intensity proportional to its given RGB value to create the impression of 'true colour'. RGB values can be linearly transformed to fall within the YCbCr colour space by separating each pixel's luminescence (Y) and colour components (CbCr):

$$\begin{aligned} \text{Y} &= K_R R' + K_G G' + K_B B' \\ \text{Cb} &= \frac{B' - Y}{2(1 - K_B)} \\ \text{Cr} &= \frac{R' - Y}{2(1 - K_R)}, \end{aligned} \tag{2.6}$$

where R', G', B' are RGB values normalised to fall in the range $[0, 1]$ and K_R, K_G and K_B are constants tuned for different video applications [31, 32].

The Human Visual System (HVS) is more sensitive to changes in luminescence than it is to changes in colour. Subsampling the YCrCb colour space's chroma components can be used as a simple means to store images more efficiently without noticeable loss in perceptual quality (4:2:2 and 4:2:0 subsampling strategies are typical) [22]. The deep compression algorithms presented in this thesis are trained to operate in the RGB colour space. This design choice is arbitrary as output images can easily be converted between colour spaces.

As is common practice [30, 33], optical flow fields are displayed in the Hue Saturation Value (HSV) colour space [34], according to

$$\begin{aligned} \mathrm{H} &= \cos^{-1}\left(\frac{\vec{\boldsymbol{V}}_x \cdot \vec{\boldsymbol{V}}_y}{\|\vec{\boldsymbol{V}}_x\|\,\|\vec{\boldsymbol{V}}_y\|}\right) \\ \mathrm{S} &= \mathrm{norm}(\vec{\boldsymbol{V}}_x \cdot \vec{\boldsymbol{V}}_y) \\ \mathrm{V} &= 100\%, \end{aligned} \tag{2.7}$$

where $\vec{\boldsymbol{V}}_x$ and $\vec{\boldsymbol{V}}_y$ are the x and y components of the optical flow vector field $\vec{\boldsymbol{V}}$. The angular direction of the optical flow vectors is indicated by Hue (H), so vectors pointing in the same direction are coloured the same. The magnitude of the vectors is normalised to fall in the range $[0, 100]$, and is indicated by the Saturation (S) or colour intensity, so vectors with lower magnitudes are more transparent and vice versa. Value (V) is set to 100%, so vectors with zero magnitude are white (zero colour intensity).

2.1.4. Video Containers and Formats

It is important at this point to clarify the distinction between video codecs and video file containers. A codec refers to the type of encoder and decoder used to compress a video file, whereas a container specifies the file format in which the compressed bits are stored. Certain formats are synonymous with specific codecs, e.g. MPEG4 ('`.mp4`') files typically use, but are not limited to, the H.264 codec [6]. RAW ('`.raw`') or YUV ('`.yuv`') files are used for uncompressed video storage. PNG ('`.png`') files store images losslessly. The image compression models in Chapters 3 and 4 are trained on lossless PNG images. As video datasets are typically encoded with lossy codecs and not in uncompressed format, we resize video frames prior to training to avoid learning unwanted compression artefacts in Chapter 5.

2.1.5. Standards

Video and image compression standards are a set of guidelines that govern the interoperability between encoders and decoders from different developers. These guidelines define I/O (Input/Output) formats as well as settings and features that need to be implemented in order to make a codec valid under the standard. Adherence to standards is crucial as it allows for ease of integration between different applications, and guarantees a certain level of performance.

Currently, the most widely adopted image codecs are JPEG [25, 35] and WebP [19]. Eminent video codecs include the Advanced Video Codec (AVC or H.264) [1] and the High Efficiency Video Coding standard (HEVC or H.265) [2]. We use these codecs to gauge the effectiveness of our deep compression models.

2.2. Evaluation Metrics

Quantifying video quality is non-trivial due to the inherently subjective nature of the Human Visual System (HVS) [6]. Video quality can be assessed subjectively through the survey-like procedures outlined in [36]. These procedures are unfortunately painstakingly slow, susceptible to viewer bias and garner results that are not easily reproducible or directly comparable. Objective algorithms have therefore become the norm in video codec performance assessment.

2.2.1. PSNR and SSIM

This thesis adopts two objective image evaluation metrics, namely Peak Signal-to-Noise Ratio (PSNR) and Structural SIMilarity index (SSIM), to guide model development. More sophisticated measures, like PSNR-HVS [37], that better model the HVS do exist but the aforementioned metrics are used due to their prominence in related research [4, 7, 9]. Both algorithms are termed full-reference as they are used to measure the degree of semblance between an $m \times n$ image, $\boldsymbol{X}$, and its distorted reconstruction, $\boldsymbol{Y}$, having undergone lossy compression. The power of the distortion noise introduced by compression is given by the mean squared error (MSE) between the original image and its compression,

$$\text{MSE} = \frac{1}{mn} \sum_{m} \sum_{n} (\boldsymbol{X}_{mn} - \boldsymbol{Y}_{mn})^2. \tag{2.8}$$

PSNR can then be expressed as:

$$\text{PSNR} = 10 \log_{10} \left(\frac{\boldsymbol{X}_{max}^2}{\text{MSE}} \right), \tag{2.9}$$

where $\boldsymbol{X}_{max}$ is the maximum pixel value in the original image $\boldsymbol{X}$.

The HVS is highly sensitive to changes in structural information [38]. As such structural equivalence is an important factor when determining the quality of a compressed image. Unlike PSNR, the SSIM algorithm models changes in the interdependencies that exist amongst adjacent pixels. This allows it to penalise structural and edge distortions caused by blurring [38]. The SSIM between a reference, $\boldsymbol{x}$, and compressed, $\boldsymbol{y}$, pixel region is given by:

$$\text{SSIM}(\boldsymbol{x}, \boldsymbol{y}) = \frac{(2\mu_{\boldsymbol{x}}\mu_{\boldsymbol{y}} + C_1)(2\sigma_{\boldsymbol{xy}} + C_2)}{(\mu_{\boldsymbol{x}}^2 + \mu_{\boldsymbol{y}}^2 + C_1)(\sigma_{\boldsymbol{x}}^2 + \sigma_{\boldsymbol{y}}^2 + C_2)}. \tag{2.10}$$

Following the procedure recommended in [38], the final SSIM score of a compressed image is obtained by applying the SSIM index in equation (2.10) over 11×11 pixel regions in a convolutional manner on a per channel basis and averaging the result. Equation 2.10 is parameterised as stipulated in [38] with $K_1 = 0.01$, $K_2 = 0.03$ and means and variances scaled by application of a Gaussian weighting process ($\sigma = 1.5$). The SSIM and PSNR image metrics are applied to video sequences by averaging scores across multiple video frames.

2.2.2. EPE

End-Point-Error (EPE) is used in Chapter 5 to score the quality of translational motion in reconstructed video frames. EPE is given as:

$$\text{EPE} = \sqrt{(\vec{\boldsymbol{V}}_g - \vec{\boldsymbol{V}}_p)^2}. \tag{2.11}$$

We use EPE to measure the Euclidean distance between the optical flow vectors of the predicted, $\vec{\boldsymbol{V}}_p$, and ground-truth, $\vec{\boldsymbol{V}}_g$, video frames.

2.2.3. VMAF

Video compression models are also assessed using the Video Multi-method Assessment Fusion (VMAF) framework developed and deployed by Netflix [39]. VMAF is a machine-learning based video quality metric trained to combine the results of various perceptual models such that its scores are more closely aligned with the human visual system than stand-alone objective algorithms such as PSNR-HVS and SSIM [39].

More specifically, the scores obtained through application of Visual Information Fidelity (VIF), Detail Loss Metric (DLM) and mean optical flow measures are weighted using a learnt Support Vector Machine (SVM) regressor to produce a final VMAF score that falls within the range of 0-100 [40]. A higher score is again indicative of greater reconstruction quality.

The FFmpeg commands used to calculate the quality between reference and compressed video frames are listed in Appendix B.

2.3. Deep Learning Techniques

Machine learning algorithms give computers the ability to model patterns in data, and apply these observations to process unseen data accordingly, without being explicitly programmed to do so [41]. Deep learning is the branch of machine learning algorithms implemented with Artificial Neural Network (ANN) architectures, inspired by the hierarchical structure of the biological nervous system [42].

Applying Deep Neural Networks (DNNs) to video compression requires unsupervised learning, i.e. the model must learn to only extract perceptually relevant features and these features have to be determined by the model from the input video itself. Here we give an overview of the deep learning techniques at the foundation of our deep compression networks.

2.3.1. Architectural Building Blocks

Neurons

Neurons (also called perceptrons) form the most basic structural and functional elements of deep neural networks. Figure 2.2 illustrates the structure of a neuron. Its output y in terms of i input values $x_{0,\ldots,i-1}$ can be expressed as:

$$y = a(\theta_0 x_0 + \ldots + \theta_{i-1} x_{i-1} + b), \tag{2.12}$$

where $\theta_{0,\ldots,i-1}$ denotes multiplicative weight values, b a biasing term and $a(\cdot)$ a non-linear activation function that is either differentiable or assumed differentiable via approximation [43]. Activation functions allow Deep Neural Networks (DNNs) to model complex non-linear functions and restrict neuron outputs. We adopt the commonly used ReLU [43], $a(x) = \max(0, x)$, activation on most hidden layers and TanH, $a(x) = \tanh(x)$, whenever we want to squash values into the range $(-1, 1)$.

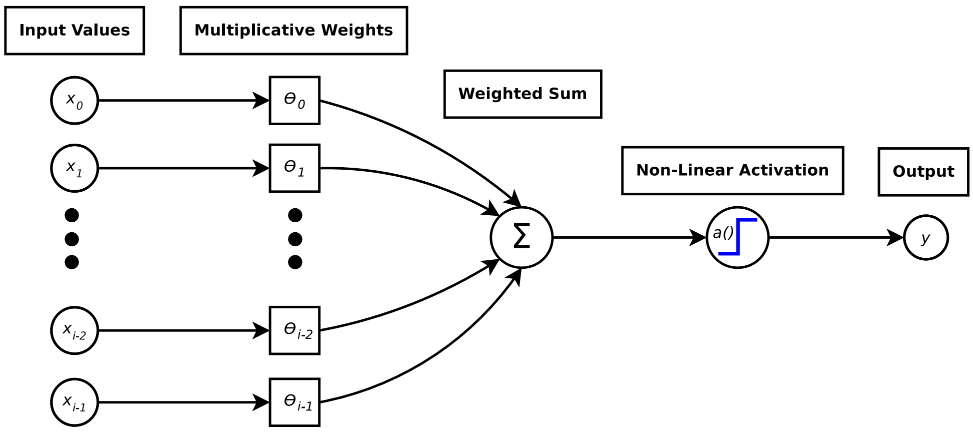


Figure 2.2: Structure of a neuron, the functional element of deep neural networks.

Feed-Forward Neural Networks

A feed-forward layer (multi-layer perceptron) consists of several neurons arranged such that its output can be described as,

$$\boldsymbol{y} = a(\boldsymbol{x}\boldsymbol{\theta}^\top + \boldsymbol{b}), \tag{2.13}$$

where $\boldsymbol{\theta}$ is the layer's weight vector matrix and $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{b}$ denote the input, output and bias term vectors, respectively [44]. Given that the input vector $\boldsymbol{x}$ consists of i values (equation (2.12)), $\boldsymbol{\theta}$ contains a vector of i weights per perceptron in the feed-forward layer. Cascading multiple feed-forward layers by connecting each layer's output to the next layer's input gives rise to a feed-forward network.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the go-to architectural components in modern deep computer vision applications, having pioneered breakthroughs in image classification [45], semantic segmentation [46], real-time object detection [47].

Present state-of-the-art deep image and video codecs are all CNN based [4, 15, 18]. The connectivity of neurons in CNNs resembles the structure of the animal visual cortex [48]. A 2D convolutional layer consists of several learnable filters where each filter is made up of neurons arranged along three dimensions, namely width, height and channel depth [49, 50]. Each filter has a small spatial extent (receptive field), but reaches through the full channel depth C of the input tensor. Filters move across the input's width and height calculating dot products between their weights and the pixels that fall within their respective receptive fields [50]. The convolution of an input, $\boldsymbol{x}$, by f filters whose weights are grouped in $\boldsymbol{\theta}$ can be written as [51],

$$\begin{aligned}\boldsymbol{y} &= \boldsymbol{b} + \boldsymbol{\theta} \otimes \boldsymbol{x} \\ &= \boldsymbol{b}[f] + \sum_{c=0}^{C} \boldsymbol{\theta}[f][c] \star \boldsymbol{x}[c],\end{aligned} \tag{2.14}$$

where $\star$ represents the channel-wise valid cross correlation between two 2D signals, defined as [23]:

$$(\boldsymbol{h} \star \boldsymbol{g})[i][j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \boldsymbol{h}[m-i][n-j]\boldsymbol{g}[i][j]. \tag{2.15}$$

The output, $\boldsymbol{y}$, is a set of 2D feature maps—one per filter, f. Intuitively, each filter learns to detect a distinct aspect of the input image (e.g edges, shapes, colour patterns) whose presence or absence at different spatial locations is noted in the outputted feature map. Adding consecutive conv-layers therefore allows for more complex feature extraction. Equation (2.14) can be extended to perform 3D convolutions by shifting four dimensional filters across a video's width, height and time axes.

So what makes conv-nets so popular?

- **Weight-sharing:** using the same filter weights at all spatial pixel locations allows CNN's to scale more efficiently to high-resolution image content than fully-connected layers [50].
- **Local connectivity:** makes CNN's translation invariant, i.e. a filter's ability to correctly identify a feature is impervious to shifts in its spatial position [48].

Certain hyperparameters have to be hand-chosen for each conv-layer prior to training. These include kernel-size, stride, dilation and padding [50]. Our parameter selections are always included in detailed network diagrams, and result from informal grid-search tuning or are based on best-practices from previous works [9, 20, 52]. As an aside, we found that zero-padding is necessary to preserve edge features and including bias terms significantly improved model performance.

Contemporary work argues for fully convolutional networks, devoid of pooling and fully-connected layers that hinder performance [53] . Following suit, we strive towards developing fully convolutional codecs as it would allow us to compress a diverse range of input image or video sizes.

Multi-Scale Convolutions

Video compression often demands that we encode motion occurring at different scales (see Chapter 5). Regrettably, standard convolutional layers are confined to capturing relationships between pixels that fall within their allocated kernel-size [54]. This prevents the learning of large scale movements.

Multi-scale convolutions address this issue by combining filters with different dilation factors [55]. Figure 2.3 shows how dilation, changing the spacing between filter elements, allows for richer sampling at different scales. This approach is more lightweight than using normal convolutional filters with different kernel-sizes as dilations do not introduce any new weights. The context derived from different scales is aggregated by a post-processing convolutional layer, ergo the entire multi-scale convolutional unit implements a type of deep Scale Invariant Feature Transform (SIFT) [56].

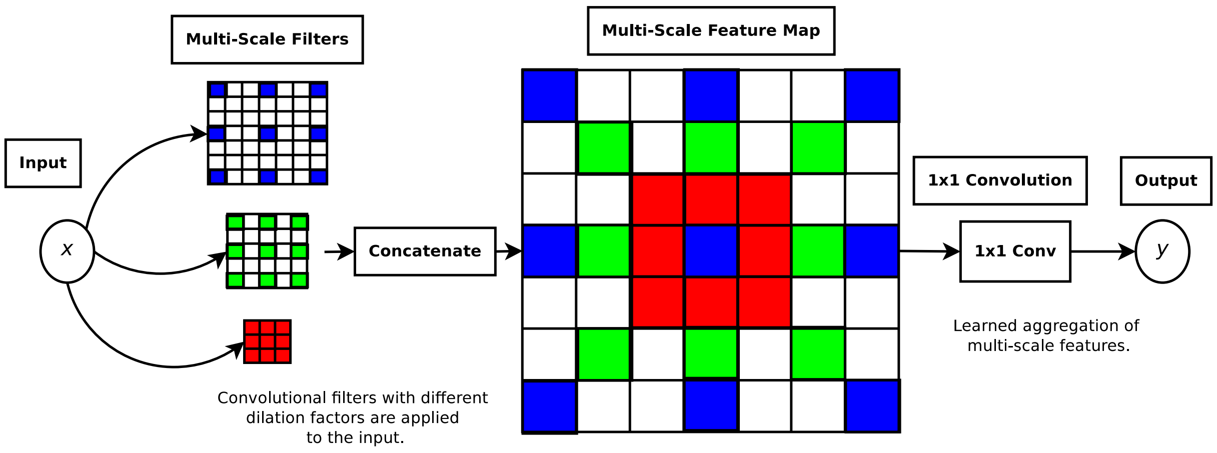

Multi-Scale Filters
Multi-Scale Feature Map
Input
x
Concatenate
1x1 Convolution
Output
1x1 Conv
y
Learned aggregation of multi-scale features.
Convolutional filters with different dilation factors are applied to the input.

Figure 2.3: Structure of a multi-scale convolution layer.

Skip Connections

Adding links or ‘skip connections’ between convolutional layers makes the training of very deep neural networks less prone to vanishing gradients, when gradients become so small that learning stagnates [57, 58]. Figure 2.4 shows how the network’s output can be back-propagated to earlier network stages directly via the link without being attenuated by intermediate layers. Linking convolutional layers involves either a summation or concatenation of the output from previous layers to the input of the current layer. Later layers can then learn to incorporate features extracted by past layers for improved image reconstruction [59].

We tend to incorporate a U-Net [60] inspired linkage structure into our compression networks. U-Net concatenates the outputs of multiple down-sampling layers and links these features to downstream upsampling layers. We use 1×1 convolutions [61] to reduce the channel dimensionality of the concatenated features whilst maintaining their original resolution. We do not show 1×1 convolutions on our compression model diagrams for brevity.

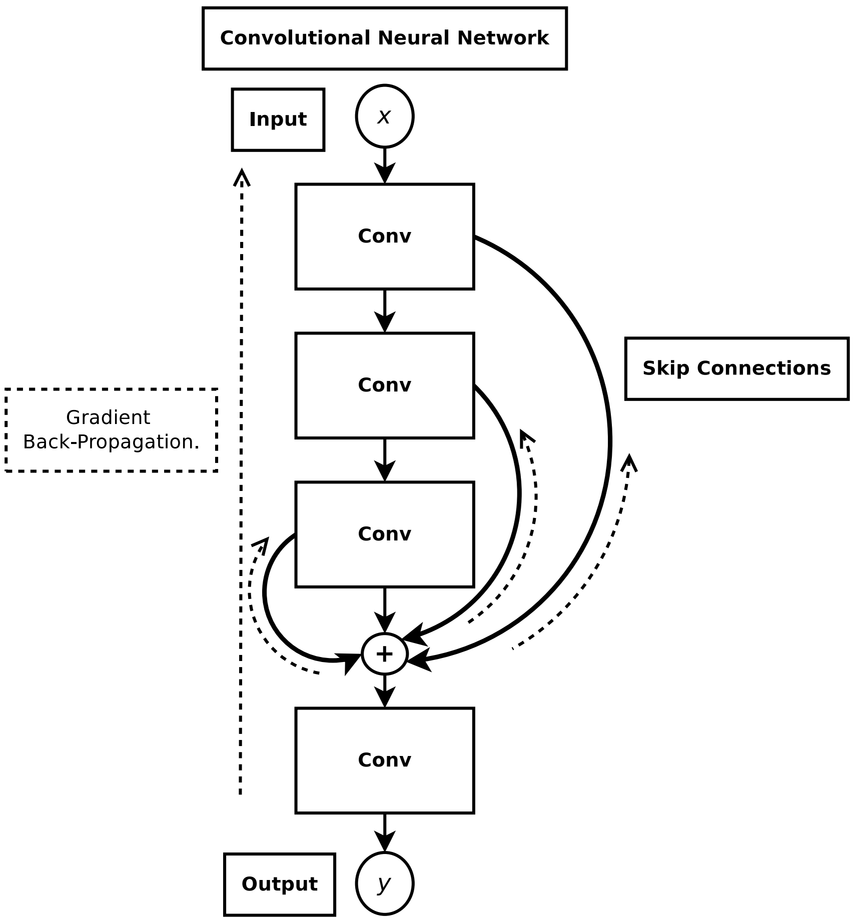


Figure 2.4: Skip connections for deeper Convolutional Neural Networks (CNNs).

Convolutional Gated Recurrent Units

Recurrent Neural Networks (RNNs) are particularly well suited towards modelling sequential data [62]. This is because they have persistence, i.e. they maintain state memory that is passed to subsequent model iterations. Non-gated RNNs struggle to model long term dependencies and are prone to vanishing gradients during training [63]. Gated Recurrent Units (GRUs) [64] and Long Short Term Memory (LSTM) cells [63] introduce memory gates to address these shortcomings.

In particular use cases GRUs lead to faster parameter updates and computation times than LSTM cells [62]. Convolutional GRUs are also shown in [9] to be better suited towards progressive image compression.

Consequently, we use convolutional GRUs to implement the recurrent structures presented in Chapters 3 and 4. In connection to Figure 2.5, Convolutional GRU logic at an arbitrary time-step t can be expressed as [64]:

$$\begin{aligned}
\boldsymbol{r}_t &= \sigma(\boldsymbol{\theta}_{ir} \otimes \boldsymbol{x}_t + \boldsymbol{b}_{ir} + \boldsymbol{\theta}_{hr} \otimes \boldsymbol{h}_{(t-1)} + \boldsymbol{b}_{hr}), \\
\boldsymbol{z}_t &= \sigma(\boldsymbol{\theta}_{iz} \otimes \boldsymbol{x}_t + \boldsymbol{b}_{iz} + \boldsymbol{\theta}_{hz} \otimes \boldsymbol{h}_{(t-1)} + \boldsymbol{b}_{hz}), \\
\boldsymbol{n}_t &= tanh(\boldsymbol{\theta}_{in} \otimes \boldsymbol{x}_t + \boldsymbol{b}_{in} + \boldsymbol{r}_t(\boldsymbol{\theta}_{hn} \otimes \boldsymbol{h}_{(t-1)} + \boldsymbol{b}_{hn})), \\
\boldsymbol{h}_t &= (1 - \boldsymbol{z}_t)\boldsymbol{n}_t + \boldsymbol{z}_t\boldsymbol{h}_{t-1}.
\end{aligned} \tag{2.16}$$

Where $\otimes$ denotes the convolution operation detailed earlier in this section. Intuitively, the 'update gate' $\boldsymbol{z}_t$ selects information from the previous state $\boldsymbol{h}_{(t-1)}$ that is pertinent and needs to be passed on to the next model iteration, whereas $\boldsymbol{r}_t$, the 'reset gate', decides what is to be forgotten. The 'new gate' $\boldsymbol{n}_t$ ensures only relevant information from $\boldsymbol{h}_{t-1}$ is combined with the current input $\boldsymbol{x}_t$ to produce new state information. The current state $\boldsymbol{h}_t$ is then updated with new state, $\boldsymbol{n}_t$, and previous state information in accordance with $\boldsymbol{z}_t$. Weights and bias terms in Equation (2.16) are labeled such that $\boldsymbol{\theta}_{ir}$ and $\boldsymbol{\theta}_{hr}$ denote the weights applied to the input and hidden state by the 'reset gate' $\boldsymbol{r}$.

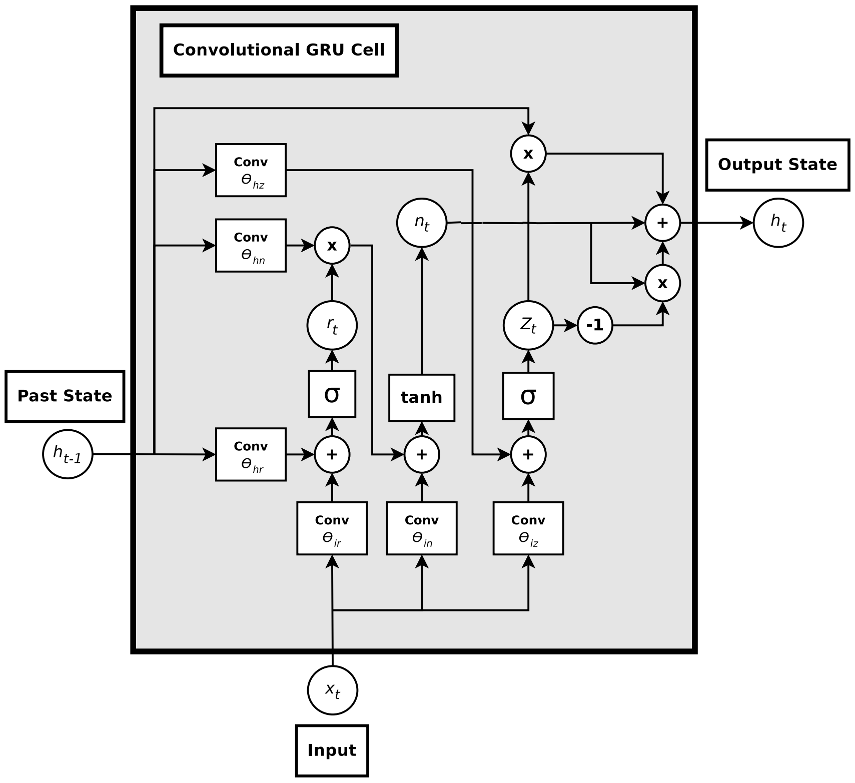


Figure 2.5: Structure of a convolutional Gated Recurrent Unit (GRU) layer.

Autoencoder Networks

Autoencoders provide a means of learned dimensionality reduction that has been shown to outperform previously popular statistical methods, e.g. Principal Component Analysis (PCA) [65]. An autoencoder consists of a multi-layer encoder and decoder network linked by a low dimensional layer or bottleneck. Equation (2.17) summarises the autoencoding process [65]. The encoder network $E(\cdot)$ reduces high dimensional input data $\boldsymbol{x}$ to a lower-dimensional space. The decoder $D(\cdot)$ then aims to create a faithful reconstruction of the input, $\hat{\boldsymbol{x}}$, from this compact representation.

$$\hat{\boldsymbol{x}} = D(E(\boldsymbol{x})) \tag{2.17}$$

Take note that $E(\cdot)$ and $D(\cdot)$ are not limited to any specific neural network architecture. Autoencoders are ever-present in related deep compression research [5, 18], and form the architectural basis of all the deep image and video codecs presented in this work.

Pixel Shuffling

The decoders in our compression networks are tasked with upsampling low-resolution feature (LR) encodings to re-synthesize high-resolution (HR) images and videos. This process is analogous to image super resolution (SR) (increasing the size of small-scale images) [66]. Upscaling an image by a factor r can be achieved through the use of nearest-neighbour [67] or bi-cubic pixel interpolation algorithms [68]. Deep deconvolutional methods such as transposed convolutions [69] and sub-pixel convolutions [70], $stride = \frac{1}{r}$, are able to learn more complex pixel interpolations for improved SR quality.

Upsampling at a decoder layer increases the computational cost at subsequent layers by a factor of r^2. The Efficient Sub-Pixel Convolutional Network (ESPCN) [71] sidesteps this issue by learning pixel interpolations in LR space. Put differently, it learns to use LR feature extractions to represent HR content. The ESPCN consists of normal convolutional layers that operate on LR features followed by a Pixel-Shuffling layer that trades depth for space,

$$\underbrace{(C, H, W)}_{\text{LR}} \xrightarrow{\textit{Pixel-Shuffle}} \underbrace{\left(\frac{C}{r^2}, H \times r, W \times r\right)}_{\text{HR}}. \tag{2.18}$$

Where (C, H, W) denotes the channel depth, height and width of the LR image features. Equation (2.19) can readily be extended to include a time dimension, T, for video SR,

$$\underbrace{(C, T, H, W)}_{\text{LR}} \xrightarrow{\textit{Pixel-Shuffle}} \underbrace{\left(\frac{C}{r^3}, T \times r, H \times r, W \times r\right)}_{\text{HR}}. \tag{2.19}$$

Networks incorporating ESPCN outperform deconvolution based SR systems both qualitatively and in terms of computational efficiency [71, 72]. This prompts us to use Pixel-Shuffling for upsampling in all our decoder architectures.

2.3.2. Learning and Optimisation

In deep learning we endeavour to find a set of network weights, $\boldsymbol{\theta}$, that minimises some objective function, $F(\boldsymbol{\theta})$. If $F(\boldsymbol{\theta})$ is defined and differentiable around an initial parameterisation $\boldsymbol{\theta}_0$, it will decrease most rapidly in the direction of its negative gradient, $-\nabla F(\boldsymbol{\theta})$ [73]. Most deep learning algorithms are trained using some form of Stochastic Gradient Descent (SGD). SGD updates network weights iteratively per batch of N training examples according to equation (2.20) [74].

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \gamma \nabla F(\boldsymbol{\theta}_t) \\ &= \boldsymbol{\theta}_t - \frac{\gamma}{N} \sum_{n=0}^{N-1} \nabla F_n(\boldsymbol{\theta}_t)\end{aligned} \tag{2.20}$$

In equation (2.20) t is used to denote the current training iteration and γ the learning rate.

For a neural network with L-layers, the derivative of $F(\boldsymbol{\theta})$ with respect to an arbitrary layer's weights, $\boldsymbol{\theta}_l$, is calculated via chain rule or back-propagation of gradients [73]:

$$\frac{dF(\boldsymbol{\theta}^l)}{d\boldsymbol{\theta}^l} = \frac{\partial F(\boldsymbol{\theta}^l)}{\partial \boldsymbol{y}^L} \frac{\partial \boldsymbol{y}^L}{\partial \boldsymbol{y}^{L-1}} \cdots \frac{\partial \boldsymbol{y}^l}{\partial \boldsymbol{\theta}^l}, \tag{2.21}$$

where $\boldsymbol{y}^l$ is used to denote the output of layer l.

To train our compression networks we adopt a more advanced flavour of SGD called Adam (Adaptive Moment) Optimisation [75]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \frac{\tilde{\boldsymbol{m}}_t}{\sqrt{\tilde{\boldsymbol{v}}_t} + \epsilon}, \tag{2.22}$$

where ϵ is a very small scalar introduced to prevent division by zero. Adam actively adapts its learning rate to the magnitudes of past gradients by introducing first, $\boldsymbol{m}_{t+1}$, and second order moments, $\boldsymbol{v}_{t+1}$, that are approximated as:

$$\tilde{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_{t+1}}{1 - \beta_1^{t+1}},$$

$$\tilde{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_{t+1}}{1 - \beta_2^{t+1}},$$

$$\boldsymbol{m}_{t+1} = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \nabla F(\boldsymbol{\theta}_t),$$

$$\boldsymbol{v}_{t+1} = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \nabla F(\boldsymbol{\theta}_t)^2.$$

The variables β_1 and β_2 are hyperparamters with default values of 0.9 and 0.999, respectively. Unlike SGD, Adam introduces momentum which prevents oscillations in the training loss, quickening its convergence [75].

2.4. Software Development

The code for our deep video codecs is publicly available on GitHub, a remote code repository that promotes open source development. Links to the provided interactive IP[y] notebooks are provided at the beginning of each chapter.

We use git, a distributed Version Control System (VCS), to keep track of changes to our source code. Git also provides efficient branching, allowing us to try and test new ideas without breaking the main line of development. All code is written in Python 3.

Each of our repositories contain a Docker image that emulates our development environment.

2.4.1. Deep Neural Networks

We adopt the Pytorch deep learning framework to build our deep compression networks. Pytorch is appealing as it includes TorchVision, a tool specially geared towards manipulating images for learning computer vision applications. Moreover, it enables CUDA Graphics Processing Unit (GPU) support and the creation of Dynamic Computation Graphs (DCG). DCGs are defined at run-time and can be actively adapted to different image or video inputs [76]. This proves particularly useful when coding the iterative image compression architectures in Chapters 3 and 4.

All our DNN models are developed from scratch using the network layers available in Pytorch in addition to our own custom layers: Convolutional GRUs, multi-scale convolutions and 3D pixel shuffling, which are detailed in Section 2.3.1 and extend Pytorch's `nn.Module` base class. Pytorch's `Autograd` package automatically computes network gradients for back-propagation during execution. Compression requires a non-differentiable discretisation step, so we extend `Autograd` to compute custom gradients for the discretisation layers in Sections 3.1.3 and 5.2.3.

Loading random video segments as Pytorch tensors requires that one first decode the entire video and store its uncompressed frames in Random Access Memory (RAM). This is CPU intensive and computationally intractable for large video files. We resort to NVIDIA Video Loader (NVVL): a library that speeds up data-loading by placing video packets onto a Graphics Processing Unit (GPU) in compressed form [77]. Video packets are then decoded into tensors directly in GPU device memory as they are fetched by the DNN model. We found that setting Pytorch's data loading thread count too high resulted in Central Processing Unit (CPU) deadlocks. To avoid this Pytorch forums suggest that the number of 'workers' or dataloader threads should be set to no more than four times the GPU count.

We trained our models on a single GeForce GTX 1060 GPU sponsored by the NVIDIA GPU Grant Program. Some of the recurrent deep image codecs in Chapter 4 were trained on parallel GPU's in Google Cloud Platform's (GCP) AI and Machine Learning environment.

2.4.2. Video and Image Code Libraries

We compare our deep compression codecs to the popular JPEG [25] and WebP [19] image codecs as well as the universal H.264 [1] and H.265 [2] video coding standards, which are listed below alongside their open source implementations:

- **JPEG:** Independent JPEG Group (libjpeg)
- **WebP:** Google Developers (libwebp)
- **H.264:** VideoLAN (libx264)
- **H.265:** x265 HEVC (libx265)

Post installation, these libraries can be used by FFmpeg, OpenCV and the Python Imaging Library (PIL) to process image and video files. Apart from being able to transcode (convert between coding standards) and compress image and video files, these libraries also offer a host of helpful tools such as evaluation metrics, image transformations, optical flow estimation, block-motion prediction, etc. We encourage anyone interested by this field to read through the available online documentation.

As mentioned in Section 2.2.3 we asses the quality of our reconstructed video frames using Netflix's VMAF model. LiteFlowNet is deployed in Chapter 5 for deep optical flow estimation. OpenCV's default Farneback algorithm (see Section 2.1.2) is also used as a secondary means of dense optical flow calculation for the experiments in Chapter 5.

2.4.3. FFmpeg Commands

```bash
#!/bin/bash

ffmpeg -i "INPUT VIDEO FILE" \\
      -codec:v "VIDEO CODEC" \\
      -g "GOP SIZE" \\
      -crf "TARGET QUALITY" "OUTPUT VIDEO FILE"
```

Listing 2.1: Simplified FFmpeg bash command used to compress video files with standard video codecs. Actual full commands are available in Apendix A

FFmpeg offers a wide range of coding options for video, each suited towards different video applications (e.g. streaming, storage, etc). For reference, the FFmpeg commands used to compress video files and generate the compression curves in Chapter 5 are included in Appendix A.

Our deep image and video codecs are all single-pass systems, which means that the same encoder weights are applied irrespective of the input video file. Two-pass encoding improves compression quality by first analysing a video file and determining optimal coding options for that file specifically. To make our comparison against H.264 and H.265 fair, we limit them to perform single-pass encoding.

Listing 2.1 is a shorthand bash script example of the video compression commands in Appendix A. The key-frame interval (how often to place intra-coded frames) is set with the Group Of Pictures (GOP) `-g` flag. We vary bitrate by targeting different output qualities with the `-crf` flag. Constant Rate Factor (CRF) encoding aims to achieve a constant quality across all video frames with as few bits as possible According to the documentation, CRF mode produces the best quality at the lowest possible bitrate using a single encoding pass [78]. CRF mode allows H.264/5 to adapt its bitrate to the content of the video being compressed, similar to the learned 3D dynamic bit assignment strategy in Chapter 5. Prescribed CRF values range from 0 to 51, with higher values resulting in lower video quality and bitrate.

2.5. Chapter Summary

In this chapter we explored various video compression concepts and deep learning techniques. We found that the three main components that enable lossy compression in standard video codecs (<u>co</u>ders/<u>dec</u>oders) are transform, prediction and entropy coding [6]. Transform coding converts image data to a low-dimensional latent representation that only preserves perceptually relevant information [25]. Video compression relies on two prediction modes: intra-frame and inter-frame [22]. Intra-frame prediction mode predicts patches in a video frame from previously decoded patches within the same frame, whereas in inter-frame prediction mode optical flow vectors approximate true object motion to predict a set of unseen video frames from past reference frames [1]. Entropy coding then removes statistical redundancy from the binarised transform and prediction codes [26]. In Section 2.1.2 we observe that the derivation of optical flow used in video inter-frame prediction assumes brightness constancy, which makes it unable to model complex motion such as occlusion, warping and colour shift [21].

The deep compression networks presented in this thesis are all based on the autoencoder architecture, which consists of an encoder and a decoder network used to compress and reconstruct an input, respectively [65]. All our models are developed with Pytorch, a deep learning framework specifically geared towards computer vision. Following [53], we strive to develop fully-convolutional autoencoders that can compress a wide range of input image/video sizes. We use pixel-shuffling in place of transposed convolutions in our decoders as it provides more efficient upsampling [71]. In our video encoders we sample motion at different scales by combining convolutional filters with different dilation factors [55]. Skip-connections and GRU cells are used to propagate information to later network layers and avoid vanishing gradients during training [57, 63]. Network optimisation is performed by Adam, an advanced version of SGD that introduces momentum to speed-up the training process [75]. The PSNR, SSIM, EPE and VMAF objective quality metrics discussed in Section 2.2 are chosen to judge the visual quality of the image and video reconstructions produced by our trained deep compression codecs. VMAF is the most decisive video metric as its scores align best with the HVS [39].

Having dealt with the background information, we proceed with the development and implementation of deep image and video compression models for transform and prediction coding. Lossless entropy coding is left to future work.

Chapter 3

Deep Image Compression

An integral component of all modern video codecs is a still image transform coding system, which handles the compression of reference frame and residual content [6]. Transform coding entails encoding a raw image signal into a low dimensional representation that only retains perceptually relevant information (see Section 2.1.1). In this chapter we investigate various approaches to still image transform coding. We start off by exploring the inner workings of JPEG, one of the world's most popular image codecs [35]. We contrast it to approaches in deep learning and motivate our selection of the progressive architectures in [7, 9] for implementation. Several experiments are undertaken to help steer our design choices and gain further insights into deep image compression. Finally we evaluate our system's performance against JPEG. Our code for this chapter is made available on GitHub: DeepImage.[1]

3.1. Related Work

3.1.1. The JPEG Standard

The JPEG image compression standard was released in 1992 by the Joint Photographic Experts Group (JPEG) [35]. It is designed to provide lossy compression for still images and intra-frame compression for video codecs such as Motion JPEG (MJPEG) and H.264/5 [6]. JPEG images carry the '`.jpeg`' file extension and can be opened by most image processing applications due to the universal popularity of the codec. The standard supports progressive enhancement as well as a wide range of image sizes and formats. Lossy JPEG compression, as outlined in [25], can be summarised by the following key stages:

- **Discrete Cosine Transform (DCT):** Each image channel is partitioned into 8×8 pixel grids termed macroblocks. Each macroblock is decomposed by the DCT into 64 constituent cosine basis functions. The DCT concentrates spectral energy at a few fundamental frequencies. This aids compression as low energy components can be negated without causing drastic degradation to the original image signal. The zero or DC-coefficient contains the average pixel value of the macroblock. Other coefficients are termed AC.

[1] `https://github.com/adnortje/deepimage`

- **Quantisation:** The DCT coefficients are then quantised according to a prescribed quantisation table. Quantisation aims to discard basis signals that are visually insignificant. Coefficients that are quantised to zero are discarded, reducing the number of coefficients transmitted to the decoder. Quantisation tables are hand-tuned using extensive psycho-visual analysis studies in order to determine optimal normalisation parameters [6].

- **Encoding:** DC-coefficients are high-valued as they tend to contain a significant portion of the image signal's energy. JPEG exploits the strong correlation that tends to exist between successive DC-coefficients by only encoding the difference between these terms. DC and AC-coefficients are binarised and ordered using a Run Length Encoding (RLE) pattern. This aids the Huffman entropy coding of the bitstream by grouping homogeneous low-frequency components together.

- **Decoding:** The Huffman encoded bitstream is converted back into 8×8 macroblocks containing the quantised DCT coefficients. Each macroblock undergoes inverse normalisation based on the quantisation table used during the encoding process. The quantisation tables are transmitted to the decoder as header information. The Inverse Discrete Cosine Transform (IDCT) reconstructs an approximation of the originally compressed macroblock.

Apart from lacking end-to-end optimisation and relying on arduous hand-tuned parameterisation, JPEG's patch-based encoding method makes it prone to block-artefacts at low bitrates. JPEG2000 mitigates block-artefacts by one-shot encoding full-sized images and replacing the DCT with a Discrete Wavelet Transform (DWT). The high computational cost incurred by one-shot encoding has, however, prevented its widespread adoption [6].

3.1.2. Deep Image Compression

Deep neural networks have been shown to outperform standard image codecs, achieving state-of-the-art image compression in terms of bit depth and quality [7, 9, 11–18]. In [79] models are trained to assist in the parameterisation of standard image codecs by learning more effective frequency transforms, quantisation tables and predictive coding schemes.

Currently, the most effective deep image compression models adopt an end-to-end autoencoder type architecture [15]. Discriminative approaches are usually trained to minimise reconstruction loss. This does not necessarily align well with the Human Visual System (HVS) due to the attenuation of high frequency components during quantisation. Adversarial loss is used by [14] to generate crisper images as the decoder must learn to fool an increasingly frequency sensitive discriminator. Instabilities in the adversarial training process causes this approach to 'fail' at compressing high-resolution images [14]. A textural loss term based on feature maps generated by a pre-trained object recognition network (e.g. ImageNet [52]) is included by [10]. Reconstruction quality is improved as ImageNet's feature maps contain textural information significant to human perception.

Image complexity varies spatially and consequently compression schemes benefit from dynamic bit assignment, i.e. allocating different bitrates to different image regions. Semantic segmentation maps are used by [12] to selectively generate unimportant image regions. Pyramidal decomposition is used by [11] for multi-scale feature extraction and aggregation. In [16] an entropy term is estimated from the latent representation to penalise high bitrates. Content-weighting by means of a learned image importance map guides bit allocation in [17]. A complex hyperprior modeling the distribution of the input image's content is transmitted as side information by [15, 18] to exploit spatial dependencies in the latent representation for improved arithmetic coding.

The recurrent autoencoders proposed in [7, 9, 13] propagate decoded information to support progressive image encoding. Progressive compression entails encoding an image such that it can be reconstructed at various quality levels based on the number of bits transmitted to the decoder. This capability is essential in streaming applications prone to bandwidth fluctuations [23]. Progressive systems also facilitate manual dynamic bit assignment governed by a predetermined quality threshold such as PSNR. Manual bit allocation [7] is more suitable than its learned counterparts [15, 17, 18] in situations where the bitrate is governed by external factors such as channel capacity [23]. Due to the present surge in online streaming applications and image traffic, the progressive image frameworks proposed in [7, 9] are selected as an architectural basis for image compression in this research.

3.1.3. Deep Discretisation

The discretisation and subsequent binarisation of compressed image features is necessary for digital storage and transmission [23]. However, this process requires a non-differentiable quantisation step making standard backpropogation impossible [8]. Both [8] and [10] suggest substituting quantisation with additive uniform noise. A differentiable proxy distribution is then used to approximate and fine-tune the entropy of the quantised coefficients. We use the direct binarisation method put forward in [80] as it enables fine-tuned control over the number of bits used and effortless conversion of continuous encoder outputs to a serialised bitstream.

Following from [7] the binarisation procedure occurs in two stages:

1. The dimensionality of the output data from the encoder is reduced to reflect the number of bits required for a preselected compression ratio using a feed-forward or convolutional neural network layer. Encoder output values are forced to fall in the continuous range $[-1, 1]$ by means of a subsequent TanH non-linearity.

2. Each encoder value is then binarised to take on one of two distinct values in the set $\{-1, 1\}$.

Although simple thresholding is suitable for inference, training requires the addition of uniform quantisation noise in order implement the stochastic binarisation function [7]:

$$b_{train}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{\epsilon} \quad \in \{-1, 1\} \tag{3.1}$$

$$\text{with } \boldsymbol{\epsilon} \simeq \begin{cases} 1 - \boldsymbol{x}, & \text{with probability } \frac{1+\boldsymbol{x}}{2}, \\ -\boldsymbol{x} - 1, & \text{with probability } \frac{1-\boldsymbol{x}}{2} \end{cases}.$$

The gradient of $b_{train}(\boldsymbol{x})$ can then be estimated from its expectation [80]. Noting that $\boldsymbol{\epsilon}$ is zero-mean, the gradient of $b_{train}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is

$$\frac{d}{d\boldsymbol{x}}\mathrm{E}[b_{train}(\boldsymbol{x})] = \frac{d}{d\boldsymbol{x}}\boldsymbol{x} = 1. \tag{3.2}$$

Equation (3.2) indicates that during training gradients are passed unchanged through the binarisation layer. This Straight Through Estimation (STE) of gradients is biased as $\boldsymbol{\epsilon}$ is assumed independent of the input values, $\boldsymbol{x}$. However the consequences of this assumption are shown to be negligible [80]. In fact the addition of random binarisation noise to the output of a networks hidden layers improves regularisation [81], which helps prevent overfitting.

3.2. Progressive Image Compression Architectures

Deep progressive image compression systems are constructed in [7,9,13] by sequentially linking several autoencoder networks. Each autoencoder stage, $\mathrm{Auto}(\cdot)$, consists of three components, namely:

- Encoder (E): used to reduce image data dimensionality.
- Binariser (B): discretises the continuous encoder features by means of the STE binarisation approach outlined in Section 3.1.3.
- Decoder (D): tries to reconstruct the original image from its compressed binary representation.

Given these network modules the reconstruction, $\hat{\boldsymbol{r}}_0$, of an image, $\boldsymbol{r}_0$, can be formulated as:

$$\hat{\boldsymbol{r}}_0 = \mathrm{Auto}(\boldsymbol{r}_0) = D(B(E(\boldsymbol{r}_0))). \tag{3.3}$$

Successive autoencoder stages are joined using either an additive reconstruction (AR) or one-shot reconstruction (OSR) framework. Both frameworks compress and reconstruct the input image at the first iteration. Subsequent stages are then used to encode the difference or residual error between this initial reconstruction and the input image.

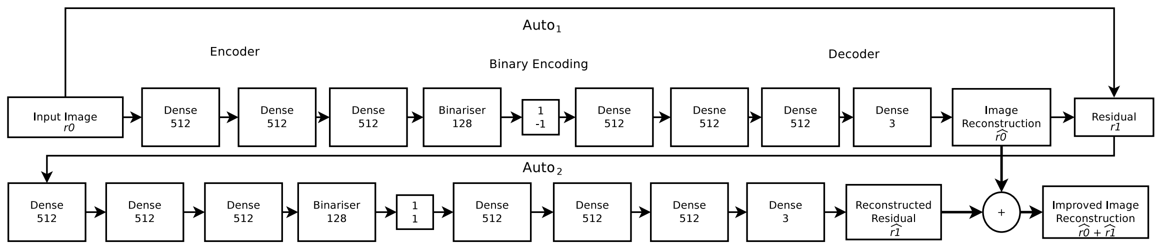


Figure 3.1: Two-iteration implementation of the Feed-Forward Additive Reconstruction network (FeedForwardAR).

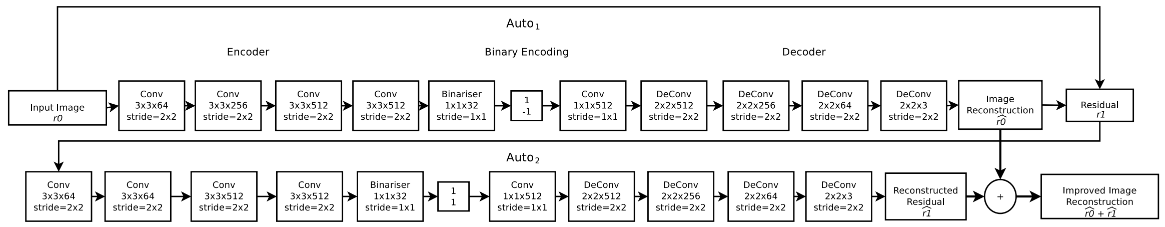


Figure 3.2: Two-iteration implementation of the Convolutional Additive Reconstruction network (ConvAR).

3.2.1. Additive Reconstruction (AR)

Additive reconstruction (AR) is widely used in traditional image codecs for variable bitrate encoding and progressive image enhancement [25]. Variable bitrate encoding entails assigning fewer bits to simpler image regions and vice versa, thereby reducing the overall bitrate on average. Progressive image compression involves encoding an image such that it can be reconstructed at various quality levels as bits are received by the decoder. Using AR, this is achieved by transmitting the difference (residual) between successive compression iterations and the original image so that the decoder can enhance its reconstruction by adding subsequently received residuals [25]. The AR process can be expressed mathematically as

$$\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \mathrm{Auto}_i(\boldsymbol{r}_{i-1}). \tag{3.4}$$

Each autoencoder stage, Auto_i, attempts to reconstruct the residual error $\boldsymbol{r}_{i-1}$ from the previous stage, with $\boldsymbol{r}_0$ representing the original image [7]. The reconstruction error $\boldsymbol{r}_i$ is then passed to the following network iteration, which attempts to reconstruct it. The final output image is progressively refined by summing over all the residuals produced across multiple network stages. Our implementation of the feed-forward (FeedForwardAR) and convolutional (ConvAR) AR models proposed in [7] are shown in Figures 3.1 and 3.2, respectively.

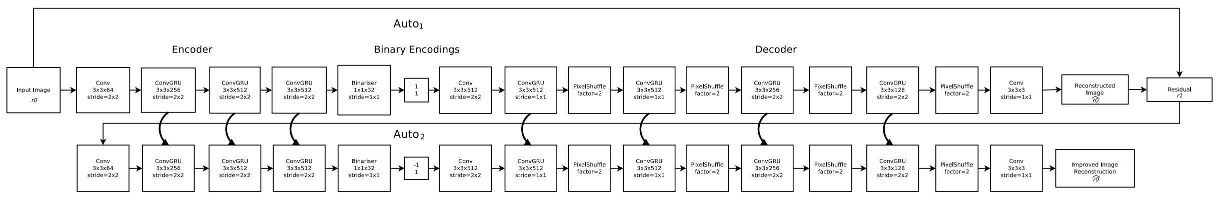

Auto$_1$
Encoder
Binary Encodings
Decoder
Input Image r0
Conv 3x3x64 stride=2x2
ConvGRU 3x3x256 stride=2x2
ConvGRU 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=2x2
Binariser 1x1x32 stride=1x1
1 1
Conv 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x256 stride=2x2
PixelShuffle factor=2
ConvGRU 3x3x128 stride=2x2
PixelShuffle factor=2
Conv 3x3x3 stride=1x1
Reconstructed Image $\widehat{r0}$
Residual r1
Auto$_2$
Conv 3x3x64 stride=2x2
ConvGRU 3x3x256 stride=2x2
ConvGRU 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=2x2
Binariser 1x1x32 stride=1x1
-1 1
Conv 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x256 stride=2x2
PixelShuffle factor=2
ConvGRU 3x3x128 stride=2x2
PixelShuffle factor=2
Conv 3x3x3 stride=1x1
Improved Image Reconstruction $\widehat{r0}$

Figure 3.3: Two-iteration implementation of the Convolutional GRU One-Shot Reconstruction network (ConvGRU-OSR).

3.2.2. One-Shot Reconstruction (OSR)

One-shot reconstruction (OSR) is defined mathematically as follows: [9]:

$$\boldsymbol{r}_i = \boldsymbol{r}_0 - \text{Auto}_i(\boldsymbol{r}_{i-1}). \tag{3.5}$$

Each iteration, i, accepts the previously incurred residual error, $\boldsymbol{r}_{i-1}$, as input and uses it to reconstruct an improved quality approximation of the original image. OSR differs from AR in that the original image is reconstructed at each network stage as opposed to the previous stage's residual. This is achieved by recurrent links that propagate encoder and decoder state information. The compression quality of the current iteration is thus influenced by relevant information from previous encodings and decodings that persist in the network's memory. Figure 3.3 illustrates the Convolutional Gated Recurrent Unit (GRU) OSR system [9], denoted as ConvGRU-OSR.

3.2.3. Iterative Optimisation

Both ConvAR and ConvGRU-OSR are optimised according to the following L_1 loss function [7]:

$$L = \frac{1}{I}\sum_{i=1}^{I} |\boldsymbol{r}_i|. \tag{3.6}$$

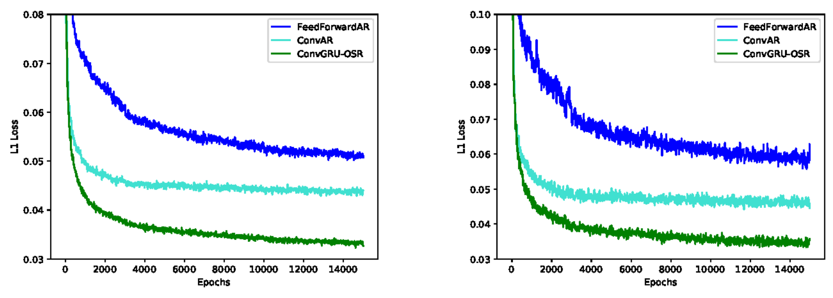
The loss in equation (3.6) is calculated by summation of the residuals, $\boldsymbol{r}_i$, incurred at each compression stage. It is normalised by the total number of stages, I, as well as the input image's width and height to obtain a mean pixel-wise loss per iteration. Each encoding stage's loss is incorporated into equation (3.6) to ensure that quality image reconstructions are produced at every iteration and not just at the terminal stage.

3.3. Experimental Setup

3.3.1. Data and Training Procedure

The networks depicted in Figures 3.1, 3.2 and 3.3 are trained on the CLIC Compression Challenge Professional Dataset [82]. The dataset is already subdivided into train, validation and test sets each containing a diverse assortment of professionally captured high-resolution natural images. The images are stored in lossless PNG format (‘`.png`’) to avoid learning the compression artefacts generated by lossy codecs.

Following [7], our implementation of the above networks is restricted to sixteen iterations, $I = 16$, with each iteration contributing 0.125 bits per pixel (bpp) to the overall compression of an image. Each network is trained to compress and reconstruct 32×32 pixel regions using the loss function expressed in equation (3.6). During training, image patches are randomly cropped from the training set at every epoch. This exposes the models to a wide variety of image content and prevents overfitting. Validation images are centre cropped to ensure that the validation losses used for early stopping are directly comparable across epochs. Input image patches are grouped into batches of 32 and their pixel values are normalised to fall in the range $[-1, 1]$. The training process spans 15 000 epochs and utilises Adam optimisation [75]. The initial learning rate is set to 0.0001 and decayed by a factor of 2 at epochs 3 000, 10 000 and 14 000. The losses drawn in the course of training are plotted in Figure 3.4. ConvGRU-OSR is shown to achieve the lowest training and validation loss.



(a) Training Loss (b) Validation Loss

Figure 3.4: Training and validation losses for 16-iteration implementations of FeedForwardAR, ConvAR and ConvGRU-OSR.

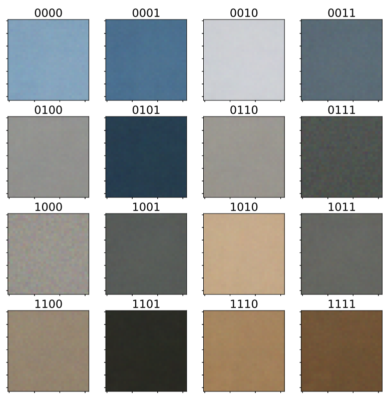
3.3.2. Evaluation Procedure

We evaluate each model’s compression efficiency based on the images in the CLIC test and validation sets. Images are resized to 320×224 pixels such that their dimensions are cleanly divisible by the 32×32 patch size used for training. During inference images are partitioned into 32×32 pixel patches that are encoded individually. PSNR and SSIM quality scores are then computed on and averaged across the reassembled images. We employ patch-based encoding as it is more memory efficient than one-shot encoding high-resolution images, and therefore more widely adopted by standard video and image codecs [6]. The performance of each network is contrasted at various bitrate allocations in order to gauge its efficacy at different work points. Additionally we perform various experiments to ascertain the pros and cons of the chosen patch-based progressive approach to deep image compression. Note that all developmental experiments are conducted on validation data.

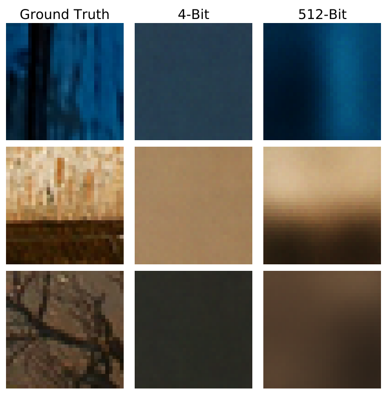
3.4. Experiments

3.4.1. Bitrate vs. Reconstruction Quality

We train 4-bit and 512-bit implementations of FeedForwardAR ($I = 1$) to determine the effect of bitrate on image reconstruction quality. The colour-mappings acquired by feeding all possible binary combinations to the 4-bit decoder are presented in Figure 3.5(a) . Each binary sequence can be seen to correspond to a unique colour pattern. The compression system essentially learns to categorise complex pixel content into one of 2^b colour combinations, where b is the number of bits used. One can assume that the model chooses which colour patterns to assign to the bits it has available based on the relative frequency with which specific colour patterns occur in the training data. Figure 3.5(b) demonstrates the trade-off between bitrate (colour range) and how closely the system's thresholded colour-mappings resemble the original image (reconstruction quality).[2] Increasing FeedForwardAR's bit allocation allows it to learn more complex colour patterns. This in turn allows the network to form a closer approximation of the ground truth image patch at the cost of a higher bitrate.



(a) 4-bit colour mappings



(b) 4-bit vs 512-bit

Figure 3.5: 32×32 patch reconstructions produced by FeedForwardAR ($I = 1$).

3.4.2. Progressive vs. Non-Progressive Encoding

Image residuals generally carry less energy and are consequently more easily compressible than raw image signals [6]. Deep image compression via progressively encoding residuals should therefore outperform compressing an image non-progressively. To substantiate this statement we implement progressive $I = 16$ and non-progressive $I = 1$ versions of FeedForwardAR where each network uses a total of 2.0 bpp. Table 3.1 indicates that the progressive approach enhances

[2] Binarisation outputs that equal -1 are flipped to 0 in this figure to improve clarity.

SSIM quality by 57% relative to the non-progressive implementation. Progressive encoding also outperforms the non-progressive approach by 36% in terms of PSNR. The scores put forward in the table are averaged across 32×32 image patches centre-cropped from the validation set.

Model	PSNR	SSIM
Non-Progressive	21.90	0.552
Progressive	**29.89**	**0.868**

Table 3.1: Averaged SSIM and PSNR scores for progressive and non-progressive implementations of FeedForwardAR (2.0 bpp).

3.4.3. Patch-Based vs. One-Shot Encoding

We deploy a one-iteration ConvGRU-OSR network to compare patch-based and one-shot encoding. One-shot encoding involves compressing a full-scale image all at once, whereas patch-based encoding subdivides an image into non-overlapping tiles that are compressed independently and reassembled. Table 3.2 indicates that patch-based encoding is more adaptive to changes in input image dimensions than its one-shot counterpart. One-shot encoding results in a mismatch between the image size selected for training and the vast array of image sizes encountered during inference, leading to suboptimal performance. Patch-based models always compress a fixed patch-size regardless of the input image's dimensions. In fact, increasing the input image size effectively lowers the complexity of the pixel content in a fixed-sized patch region. This allows the patch-based model to produce substantialy higher PSNR scores for larger input image sizes. Figure 3.6 exposes that patch-based encoding produces block-artefacts at shallow bit depths. Although the patch-based model is able to produce a less noisy rendition of the input image (higher PSNR), the smooth reconstruction produced by one-shot encoding is more structurally coherent (higher SSIM). This experiment highlights the importance of using multiple quality metrics to assess a codecs performance. Patch-based encoding is adopted for all further evaluations due to its computational efficiency and superior PSNR. We provide a means of suppressing the block-artefacts incurred by this approach in Chapter 4.

Encoding Process	PSNR		
	128×128	512×512	1280×1280
One-Shot	21.32	22.09	22.60
Patch-Based	**22.61**	**25.38**	**26.89**

Table 3.2: PSNR scores achieved across different input image dimensions by patch-based and one-shot encoding implementations of ConvGRU-OSR ($I = 1$; 0.125 bpp).

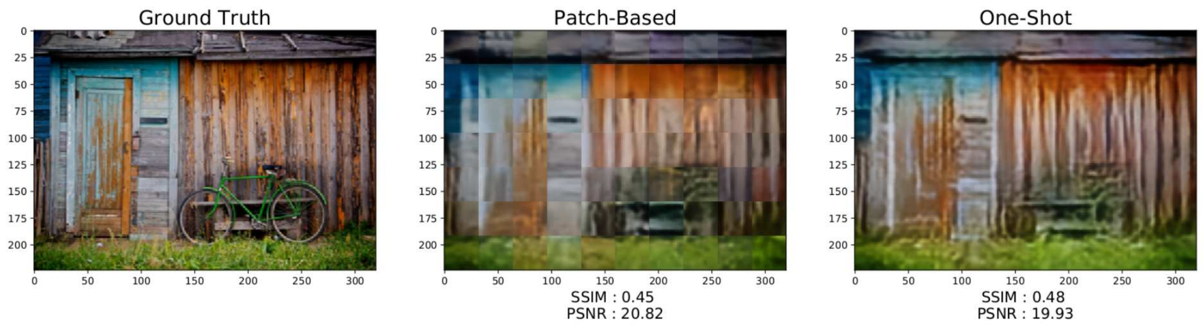


Figure 3.6: 224×320 image reconstructions produced by patch-based and one-shot encoding implementations of ConvGRU-OSR ($I = 1$; 0.125 bpp).

3.5. Model Evaluation

We train $I = 16$ iteration patch-based implementations of FeedForwardAR, ConvAR, and ConvGRU-OSR and compare their compression efficiency to JPEG. All codecs are assessed on images from the CLIC test set [82]. Figures 3.7 and 3.8 compare the SSIM and PSNR rate-distortion curves of the various codecs. The top-performing deep image model, ConvGRU-OSR, outperforms its nearest rival, Progressive JPEG, across the entire bit range. OSR surpasses AR in that it allows for learned non-linear residual combinations as opposed to linear residual summation. Figure 3.9 contrasts the low-bitrate images produced by each image codec. Progressive image reconstructions by ConvGRU-OSR (c) are both qualitatively and quantitatively (SSIM and PSNR) superior to JPEG (d) & (e).

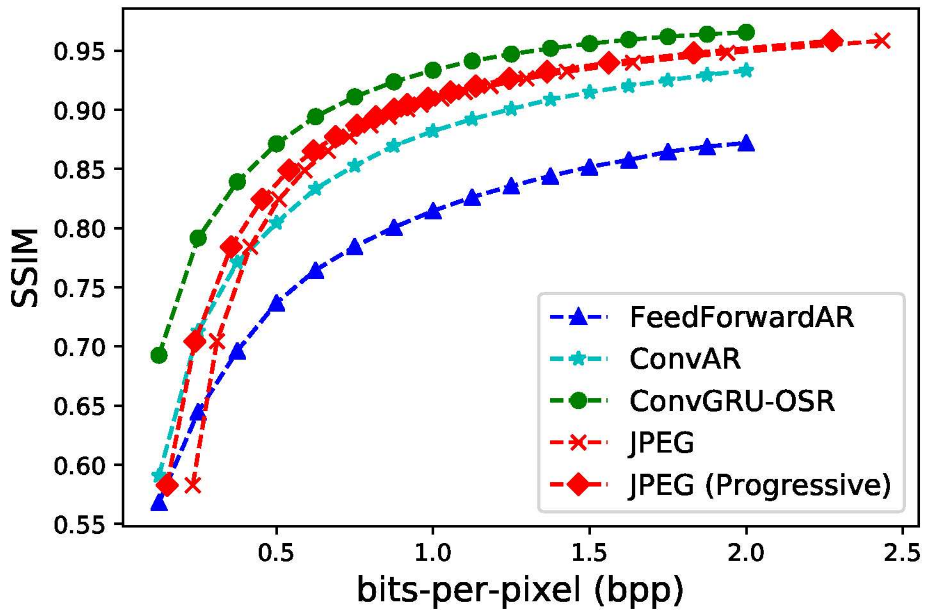


Figure 3.7: JPEG vs. deep image codecs SSIM rate-distortion curves for 224×320 images.

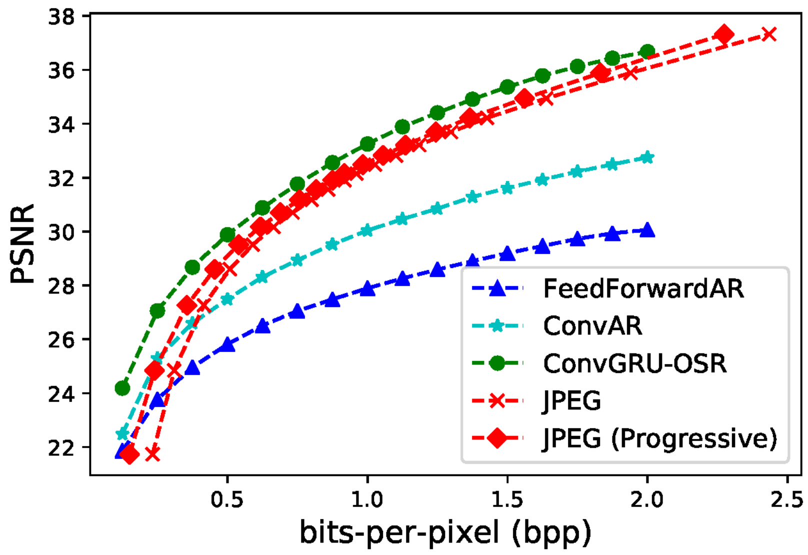


Figure 3.8: JPEG vs. deep image codecs PSNR rate-distortion curves for 224×320 images.

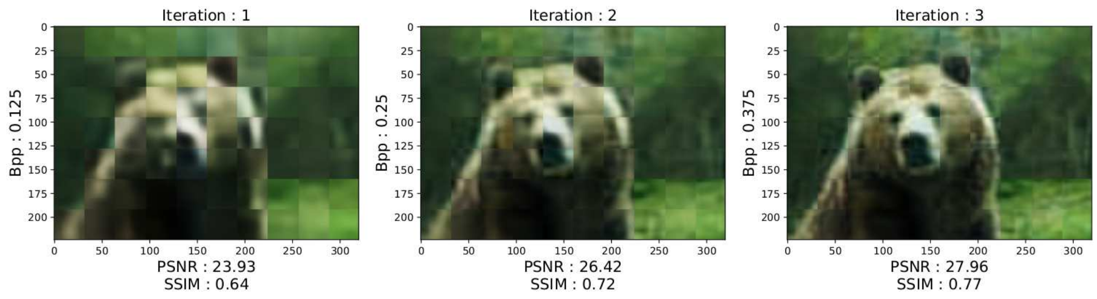
3.6. Chapter Summary

We successfully implemented several deep neural network architectures from [7, 9] for image compression. Experiments revealed that progressive encoding outperformed applying the same architecture non-progressively at a fixed bitrate. During evaluation we found that OSR produces better patch reconstructions than AR as it is able to learn a more complex superposition of residual information. Our patch-based progressive ConvGRU-OSR model triumphs over JPEG across all bit depths in terms of PSNR and SSIM. This is significant as ConvGRU-OSR is purely a transform coding model and, unlike JPEG, does not incorporate any form of entropy coding.

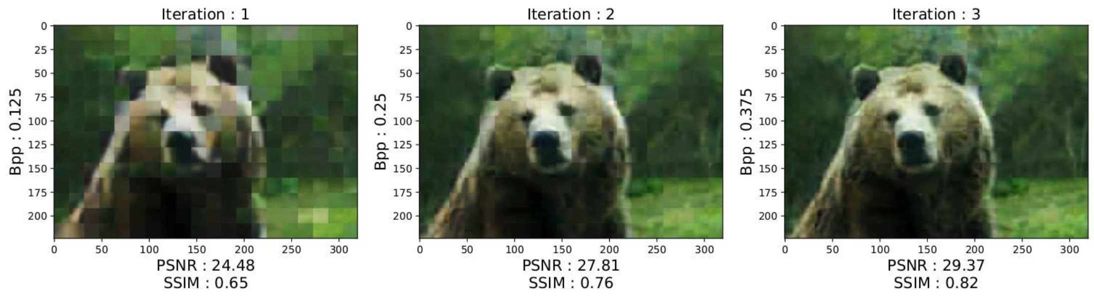
In somewhat related research, we successfully applied a variant of ConvGRU-OSR with STE binarisation to acoustic-unit discovery for unsupervised speech synthesis. Instead of compressing image patches, ConvGRU-OSR was tasked with extracting speaker independent symbolic representations from unlabeled speech data and decoding these discrete symbols in a target speaker's voice. This work formed part of our submission to *Interspeech 2019* [83] and demonstrates the relevance of deep image compression techniques to other research areas.

Future work will focus on deep methods that can be used to optimise the codes produced by our models for lossless entropy coding. We will also consider using loss functions that are more closely aligned with the HVS during training.

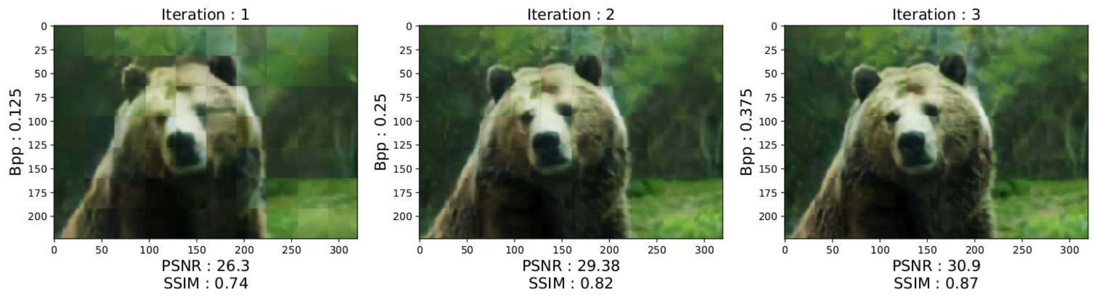
In Section 3.4.3 we showed experimentally that the PSNR achieved by patch-based encoding scaled better to high resolution image inputs than one-shot coding. This indicates that patch-based systems are more adaptable to changes in input image size. Patch-based compression is however shown to produce block artefacts at low-bitrates, which worsens its SSIM score. In the Chapter to follow we introduce a binary image inpainting strategy that reduces block artefacts by re-instating dependencies between adjacent patch regions.



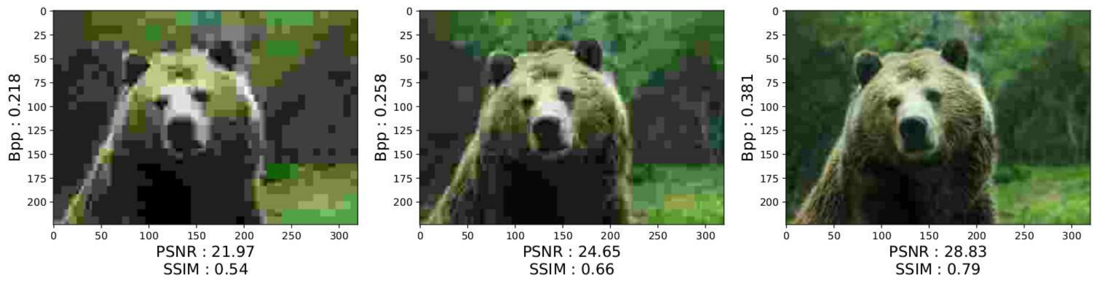
(a) FeedForward AR



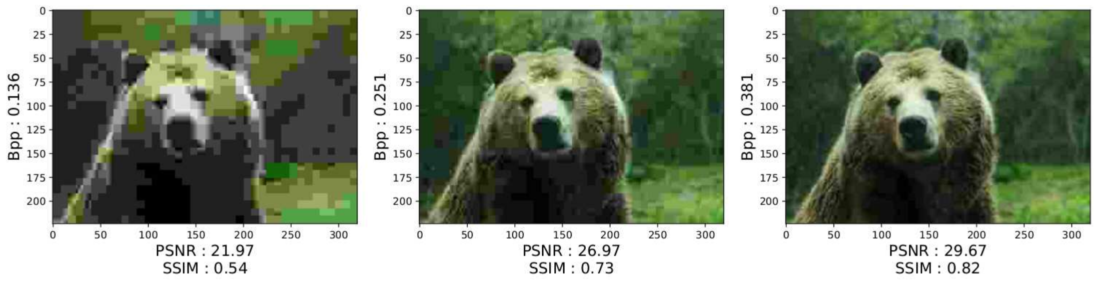
(b) ConvAR



(c) ConvGRU-OSR



(d) JPEG



(e) Progressive JPEG

Figure 3.9: Deep image codecs vs. JPEG 224 $\times$ 320 image reconstructions.

Chapter 4

BINet: a binary inpainting network for deep patch-based image compression

Standard video codecs further compress reference frame content through intra-frame prediction. The decoder predicts a premise for each patch based on prior patch decodings within the same frame. Compression is improved as only the residual between the decoder's prediction and the original patch need then be transmitted.

Recent deep learning models outperform standard lossy image codecs for full-resolution image compression. Applying these models on a patch-by-patch basis, however, requires that each image patch be encoded and decoded independently. The structural influence imposed by pixels from adjacent patches is therefore lost, often leading to block artefacts at low bitrates.

In this chapter we propose the Binary Inpainting Network (BINet), an autoencoder framework which incorporates learned binary inpainting to reinstate interdependencies between adjacent patches, for improved patch-based compression of still images. When decoding a specific patch, BINet additionally uses the binarised encodings from surrounding patches to guide its reconstruction. This is inspired by work on inpainting, where blocked-out image regions are reconstructed. In contrast to sequential inpainting methods where patches are decoded sequentially and previous reconstructions are used to predict subsequent patches, BINet operates directly on the binary codes of surrounding patches without access to the original or reconstructed image data. Both encoding and decoding can therefore be performed in parallel.

We demonstrate that binary inpainting improves the compression quality of a competitive deep image codec across a range of compression levels. Qualitatively, the inpainting learned by BINet is shown to produce smoother image reconstructions at low bitrates.

Our work on BINet is submitted for publication in the *Journal of Visual Communication and Image Representation* and is currently undergoing review.

4.1. Related Work

Previous approaches to deep image compression, although effective, are not optimised for patch-based encoding since they use the full image content to steer compression. Full image context is, unfortunately, not available for patch-based systems as each patch is encoded independently.

Patch-based encoding is therefore avoided in deep compression models [7, 13], as it may result in block artefacts at shallow bitrates. To remedy this, we propose the Binary Inpainting Network (BINet) framework, which is inspired by research in image inpainting.

Image inpainting involves reconstructing a masked-out image region by using the surrounding pixels as context. It is often used as an error-correction strategy to restore patches lost during transmission. Traditional inpainting models, such as PixelCNN [84], assume access to original pixel content; in Figure 4.1(a), the model would be asked to predict the shaded region in the middle, given the surrounding context as input. We extend this idea in order to perform patch-based image compression. When decoding a particular patch, BINet incorporates the compressed binary codes from adjacent image patches as well as the current patch to reinstate relationships between separately encoded regions. As depicted in Figure 4.1(c), BINet therefore exploits encoded binary information from a full-context region as well as the patch being inpainted in order to formulate its prediction of the inpainted region. The overall approach is illustrated in Figure 4.3: BINet encodes patches as discrete binary codes using a single encoder. The decoder then reconstructs a particular centre patch by incorporating the binary codes of surrounding patches. It therefore allows for parallel encoding and decoding of image patches aided by learned inpainting from a full binary context region.

In sequential compression techniques such as WebP [19], linear combinations of previously reconstructed outputs are used when decoding a particular patch. WebP's four main prediction modes either average (DC_PRED), directly copy (H_PRED, V_PRED), or linearily combine (TM_PRED) pixels from previously decoded patches, as shown Figure 4.2 [19]. This is similar to sequential patch-based inpainting [20], as illustrated in Figure 4.1(b), where previously decoded output from the model is treated as the context region and used to perform inpainting on the next patch. In contrast to these approaches, BINet decodes a particular patch, not based on previous patch reconstructions, but based directly on the binary encodings of the surrounding patches. Since it does not need to wait for surrounding patches to be decoded, BINet can decode all patches in parallel while still taking the full surrounding context into account.

BINet's encoder and decoder are trained jointly through end-to-end optimisation. In contrast to [20], where separate compression and inpainting networks are trained, BINet builds inpainting directly into its decoder architecture and does not require training an additional inpainting network. Our aim is to show that this approach allows spatial dependencies between patches to be re-instated from independently encoded patches, thereby advancing patch-based encoding in a neural compression model.

We proceed with a description of the BINet framework and the formulation of a loss function for learning binary encodings that exploit spatial redundancy between neighbouring image patches. BINet can be used with different types of encoder and decoder architectures, and in this chapter we specifically employ two competitive iterative decoding methods [7, 9], namely additive reconstruction (AR) and one-shot reconstruction (OSR). We describe these specific instantiations of BINet in Section 4.2. To show the benefit of incorporating inpainting, the BINet models are compared to convolutional AR and OSR models without inpainting. Compression efficiency is evaluated quantitatively using the SSIM and PSNR image quality metrics. We show that BINet performs better than the conventional AR and OSR approaches over the complete range of compression levels considered (Section 4.4). On the standard Kodak dataset [85], we show that the OSR variant of BINet consistently outperforms JPEG. Although it falls short of outperforming WebP, we show qualitatively that BINet produces smoother image reconstructions and is capable of more complex inpainting than the sequential decoding methods used by WebP. We released a full implementation of BINet online[1].

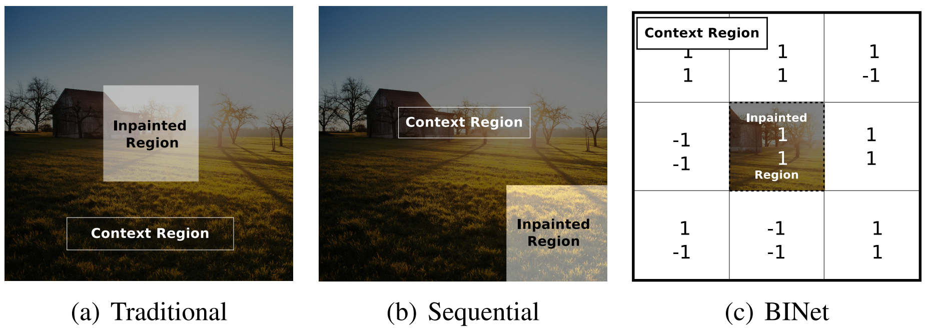


Figure 4.1: Context regions available to various inpainting models.

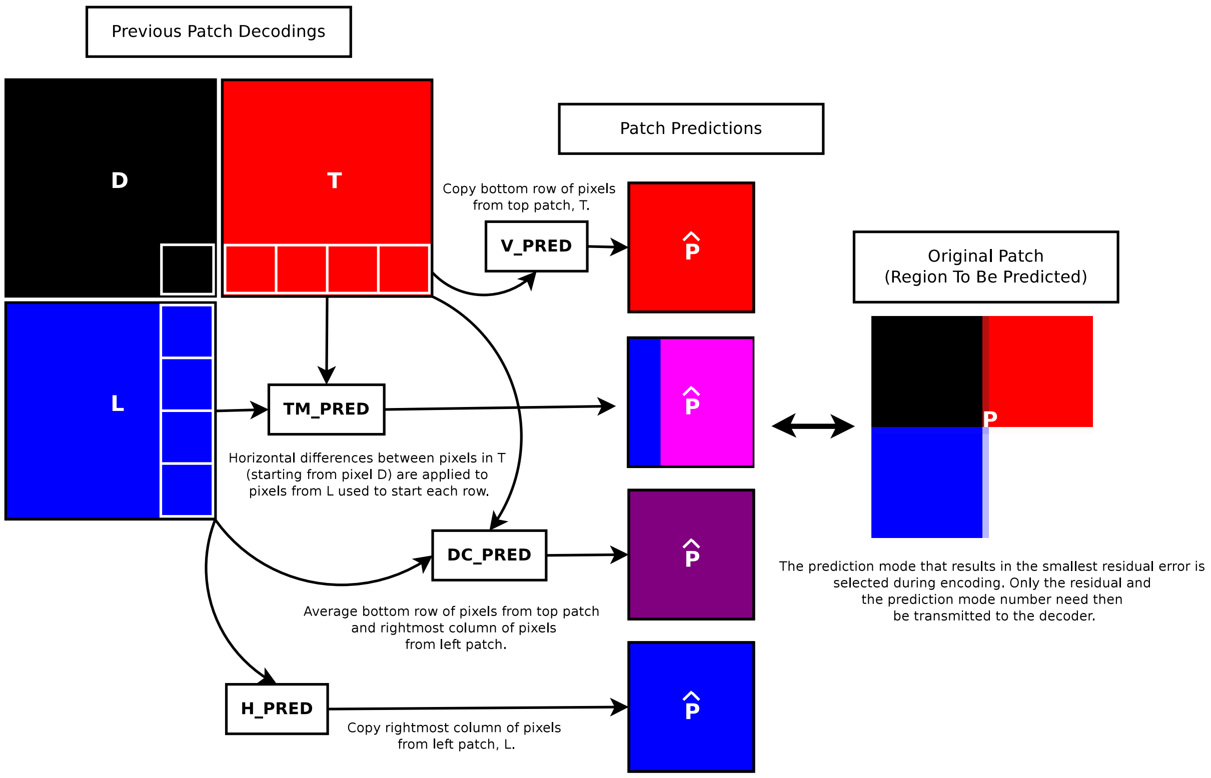

Previous Patch Decodings
D
T
L
Patch Predictions
Copy bottom row of pixels from top patch, T.
V_PRED
P̂
TM_PRED
Horizontal differences between pixels in T (starting from pixel D) are applied to pixels from L used to start each row.
P̂
DC_PRED
Average bottom row of pixels from top patch and rightmost column of pixels from left patch.
P̂
H_PRED
Copy rightmost column of pixels from left patch, L.
P̂
Original Patch (Region To Be Predicted)
P
The prediction mode that results in the smallest residual error is selected during encoding. Only the residual and the prediction mode number need then be transmitted to the decoder.

Figure 4.2: WebP's four main prediction modes (H_PRED, V_PRED, DC_PRED and TM_PRED) used for sequential patch prediction.

[1] https://github.com/adnortje/binet

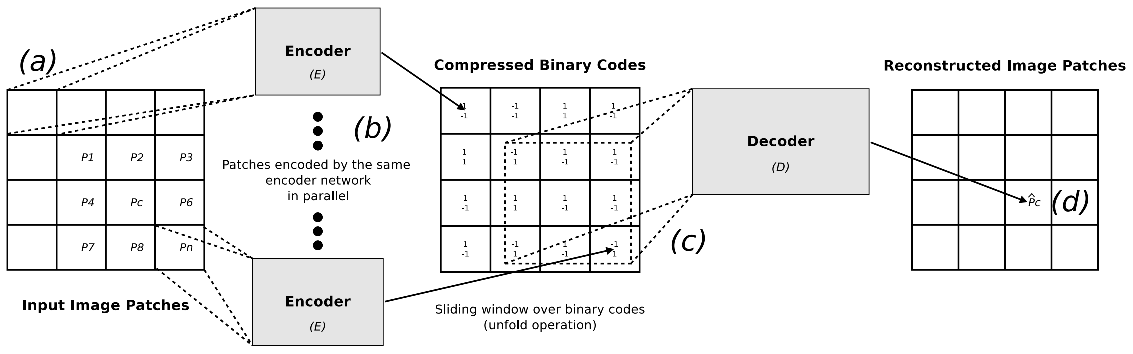

(a)
Input Image Patches
P1
P2
P3
P4
Pc
P6
P7
P8
Pn
Encoder
(E)
(b)
Patches encoded by the same encoder network in parallel
Encoder
(E)
Compressed Binary Codes
Sliding window over binary codes (unfold operation)
(c)
Decoder
(D)
Reconstructed Image Patches
P̂c
(d)

Figure 4.3: The Binary Inpainting Network (BINet) framework. For illustration, the compressed binary codes here consist of two bits per patch and each bit is indicated with a -1 or 1.

4.2. Binary Inpainting Network (BINet)

4.2.1. Architectural Overview

BINet is a variation of a basic autoencoder [65]. Figure 4.3 shows BINet's encoding and decoding process. It accepts as input a set of image patches, indicated by *(a)* in the figure, that are reduced to low dimensional representations and binarised, as shown at *(b)*. Binarisation is required for digitally storing and/or transmitting a compressed version of an image [6]. As in [7, 80] a stochastic binarisation function is used during training by adding uniform quantisation noise. This allows us to backpropagate gradients through the binarisation layer in the encoder by copying the gradients from the first decoder operation to the penultimate encoder layer. The decoder network at *(c)* is applied as a sliding window across the generated binary codes such that each image patch at *(d)* is decoded using both its own binary code and the codes of adjacent patches that fall within a specific grid region. Intuitively, because the encoder and decoder networks are trained jointly, the decoder learns to inpaint from binary codes within its context region whilst the encoder learns to produce more compact codes that promote the inpainting performed by the decoder. The same encoder network is applied to each individual image patch, meaning that encoding on multiple patches can be performed in parallel. In principle any model can be used as the encoder and decoder in Figure 4.3, which is why we refer to BINet as a framework.

As depicted in Figure 4.3, the reconstruction of a patch $\boldsymbol{P}_c$ from its compressed representation can be formulated as

$$\hat{\boldsymbol{P}}_c = D(E(\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_c, \ldots, \boldsymbol{P}_n)), \tag{4.1}$$

where $E(\cdot)$ and $D(\cdot)$ represent the encoder and decoder mappings shown at *(b)* and *(c)*, respectively. The n patches used as context for predicting the centre patch $\boldsymbol{P}_c$ are $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n$. The sliding window at the decoder can be implemented using unfold operations to maintain parallelisation, and takes the bits produced for $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n$ at *(b)* as context to make the prediction $\hat{\boldsymbol{P}}_c$. Note that the same encoder network is applied to each of the input image patches individually and in parallel. Edge regions of the binary codes are appropriately padded so that the spatial resolution of the input image is maintained. To learn how to inpaint, we use the L_1 loss:

$$L_{\text{inpaint}} = |\boldsymbol{P}_c - \hat{\boldsymbol{P}}_c| = |\boldsymbol{P}_c - \text{Auto}(\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_c, \ldots, \boldsymbol{P}_n)|, \tag{4.2}$$

where $\text{Auto}(\cdot)$ is equivalent to $D(E(\cdot))$.

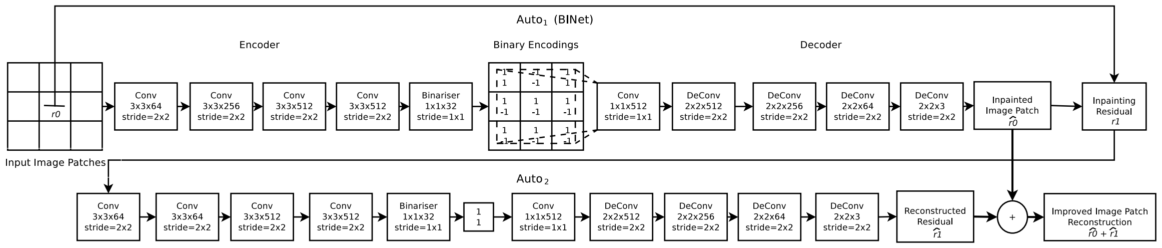


Figure 4.4: Two-iteration implementation of BINet with additive reconstruction (AR).

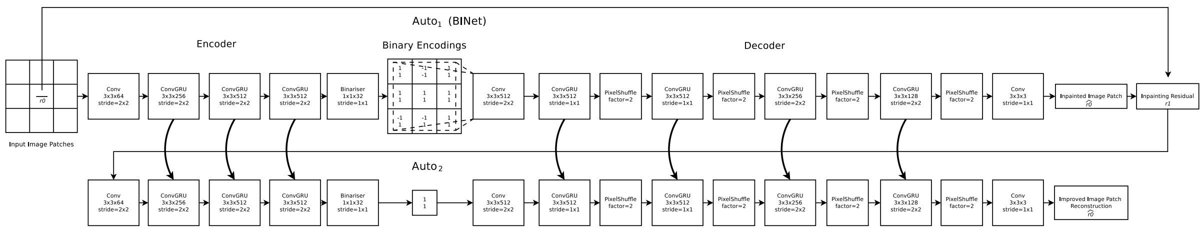

Auto1 (BINet)
Encoder
Binary Encodings
Decoder
Input Image Patches
Conv 3x3x64 stride=2x2
ConvGRU 3x3x256 stride=2x2
ConvGRU 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=2x2
Binariser 1x1x32 stride=1x1
Conv 3x3x512 stride=2x2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x512 stride=1x1
PixelShuffle factor=2
ConvGRU 3x3x256 stride=2x2
PixelShuffle factor=2
ConvGRU 3x3x128 stride=2x2
PixelShuffle factor=2
Conv 3x3x3 stride=1x1
Inpainted Image Patch r̂0
Inpainting Residual r1
Auto2
Improved Image Patch Reconstruction r̂0

Figure 4.5: Two-iteration implementation of BINet with one-shot reconstruction (OSR).

4.2.2. BINet with AR and OSR

Both AR and OSR can be used naturally with BINet. As baselines, we use the progressive ConvAR [7, 20] and ConvGRU-OSR [9] networks shown in Figures 3.2 and 3.3, respectively. The reconstruction of an image patch $\boldsymbol{P}$ for a single iteration of these models can be written as:

$$\hat{\boldsymbol{P}} = \text{Auto}_1(\boldsymbol{P}) = D_1(E_1(\boldsymbol{P})). \tag{4.3}$$

Their patch reconstruction are therefore based on the encoding of a single input patch $\boldsymbol{P}$. In other words, they do not incorporate inpainting to aid compression. The training loss for both the ConvAR and ConvGRU-OSR baselines is expressed in equation (3.6).

The BINet framework is incorporated into ConvAR and ConvGRU-OSR by including learned binary inpainting at the first iteration, as shown in Figures 4.4 and 4.5. Later iterations encode the residual error incurred by this initial inpainting prediction. We only include inpainting at the first iteration, as intuitively this stage encodes details that contain the most spatial redundancy compared to later stages whose purpose is to encode finer and less correlated patch details.[2] Our

[2] Future work may focus on ways of including binary inpainting at later network stages.

goal is to show the benefit of this binary inpainting strategy.

The encoding process of BINetAR (Figure 4.4) and BINetOSR (Figure 4.5) can be expressed as in equations (3.4) and (3.5), where $\boldsymbol{r}_0$ again represents the original input image patch while $\boldsymbol{r}_1$ is the initial iteration's inpainting loss given in equation (4.2). An I-iteration implementation of BINet with either AR or OSR is trained to optimise the loss:

$$L_{\text{BINet}} = L_{\text{inpaint}} + \sum_{i=2}^{I} |\boldsymbol{r}_i|. \tag{4.4}$$

4.3. Experimental Setup

4.3.1. Data and Training Procedure

The models discussed in Section 4.2 are trained on the CLIC Compression Challenge Professional Dataset [82], which is pre-partitioned into training, validation and test sets. Each set contains a variety of professionally captured high resolution natural images, saved in lossless PNG format to prevent the learning of compression artefacts introduced by lossy codecs.

The loss functions in equations (3.6) and (4.4) are used to train $I = 16$ iteration implementations of the baseline (ConvAR, ConvGRU-OSR) and BINet (BINetAR, BINetOSR) systems, respectively. All models are trained to encode and reconstruct randomly cropped 32×32 image patches. Following the approach in [7], the networks are constrained such that each autoencoder stage contributes 0.125 bits per pixel (bpp) to the overall compression of an input image patch. During training, BINet encodes nine directly adjacent image patches independently and reconstructs the central patch region based on the binary codes produced for the nine patches. Training patches are randomly cropped from the images in the training set at every epoch while centre cropping is used on images in the validation set to ensure that the validation losses for the BINet and baseline models are directly comparable across epochs. Image patches used during training are batched into groups of 32 and normalised such that pixel values fall in the range $[-1, 1]$. Models are trained for 15 000 epochs and early stopping is employed based on the validation loss.[3] We use Adam optimisation [75] with an initial learning rate of 0.0001. The learning rate is decayed by a factor of 2 at epochs 3 000, 10 000 and 14 000.

4.3.2. Evaluation Procedure

For evaluation, each image is resized to 320×224 pixels such that evaluation image dimensions are cleanly divisible by the chosen 32×32 patch size.[4] Images are then partitioned into 32×32 pixel patches and encoded, and quality scores are calculated on and averaged across the

[3] For the preliminary analyses in Section 4.4.1 we stop training at 5 000 epochs.

[4] We also ran tests on full unscaled images, and found that trends were exactly the same as when images are resized in this way, due to the models always compressing a fixed patch size irrespective of input image dimensions.

reassembled images. The performance of BINet is contrasted to that of the baseline systems at various bit depths in order to gauge the effectiveness of incorporating the proposed binary inpainting framework across different operating points. Additionally, we perform various preliminary analyses on validation data to further illustrate BINet's capabilities.

4.4. Experiments

We first perform a preliminary analysis on development data to better understand the properties of BINet and the benefit of binary inpainting as opposed to conventional sequential inpainting techniques. We then turn to quantitative analyses on test data where BINet is compared to the baseline neural compression models as well as standard image compression codecs.

4.4.1. Preliminary Analysis

Is Inpainting from Binary Codes Possible?

In order to assess qualitatively whether inpainting of image patches from compressed binary codes is possible, a 1-iteration implementation of BINetAR (0.125 bpp) is trained to explicitly predict the pixel content of an unknown 32×32 patch region located at the centre of a 96×96 pixel grid. This version of BINet is purposefully altered such that it masks bits pertaining to the central patch region, i.e. the context region available to the decoder matches that of Figure 4.1(a). This forces the network to become fully reliant on the binary encodings of surrounding patches when predicting the central patch's pixel content.

Figure 4.6 demonstrates the inpainting capabilities of this masked BINet, and indicates that it is able to predict a basis for an unknown patch using the compressed binary codes of its nearest neighbours. Figure 4.7 compares inpaintings from BINet (green border) and WebP (red border). The four main modes used by WebP to sequentially predict a patch region are included in the diagram and abbreviated as in Figure 4.2 [19]. The figure shows that the inpaintings produced by BINet resemble the ground truth patches (black border) more closely than those of WebP.

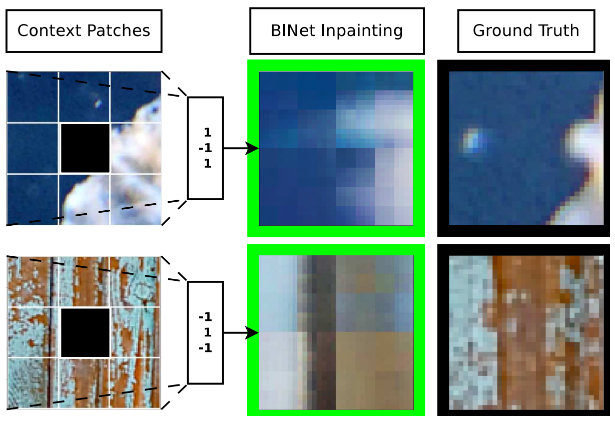


Figure 4.6: Inpaintings performed by masked BINet.

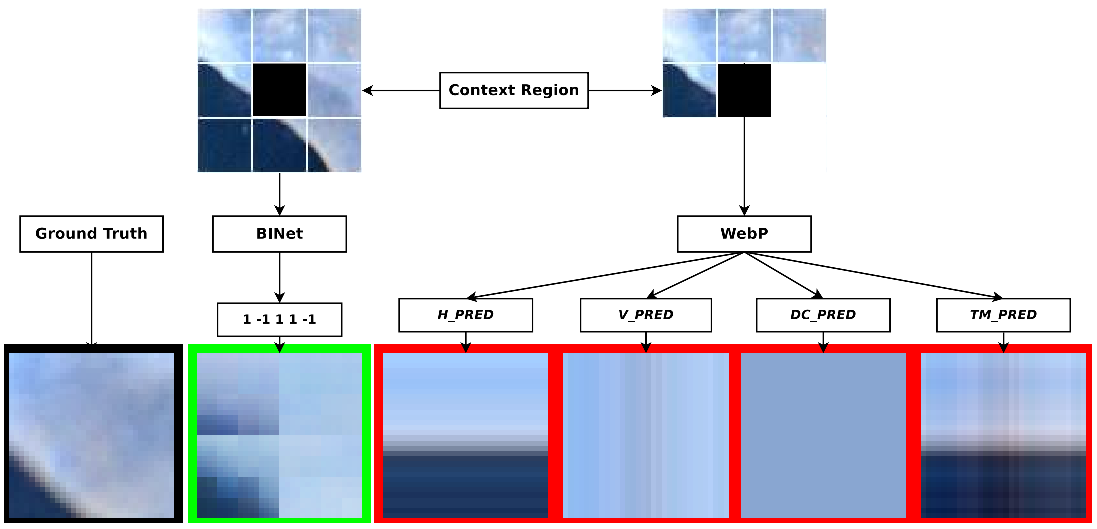


Figure 4.7: A comparison of the inpainting performed by masked BINet and WebP.

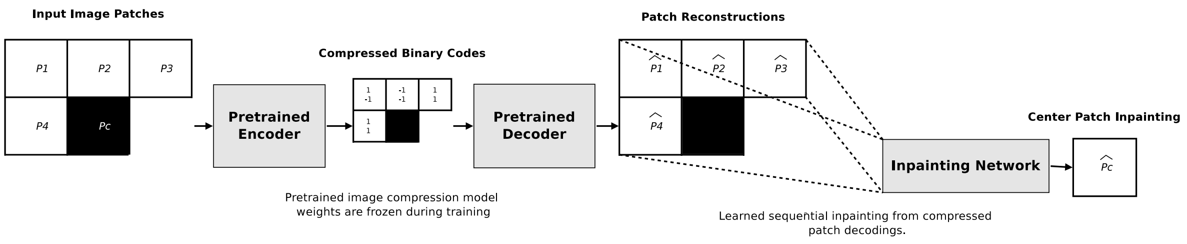


Figure 4.8: The Sequential Inpainting Network (SINet).

Is Full-Context Binary Inpainting Superior to Sequential Inpainting?

In this experiment we compare full-context binary inpainting to the sequential inpainting scheme proposed by [20]. A masked 1-iteration realisation of BINetAR is pitted against the Sequential Inpainting Network (SINet) in Figure 4.8. SINet consists of a pre-trained image compression model (ConvAR with $I = 1$, bpp $= 0.125$) coupled to an inpainting network (ConvAR decoder). SINet's inpainting network is trained to sequentially predict the central patch $\boldsymbol{P}_c$ from previously decoded patches such that its context region is like that of Figure 4.1(b). Table 4.1 compares the average SSIM and PSNR scores achieved by BINet and SINet on the validation set. BINet's full-context binary inpainting mechanism leads to a 6% improvement in SSIM and a 11% increase in PSNR relative to the partial-context sequential inpainting performed by SINet. Figure 4.9 illustrates how BINet's ability to harness pixel content from a full context region aids its inpainting ability. BINet (green border) correctly identifies that the lower right-hand corner of its inpainting should be white, whereas SINet (red border) is oblivious to this due to its limited context region. Importantly, BINet has a major additional benefit in that it can be parallelised, since reconstruction of a particular patch is not performed based on previously decoded patches but rather directly on the binary codes of all surrounding patches.

Model	PSNR	SSIM
SINet	19.85	0.47
BINet	**22.08**	**0.50**

Table 4.1: Averaged masked BINet and SINet SSIM and PSNR scores for 32×32 image patch inpaintings

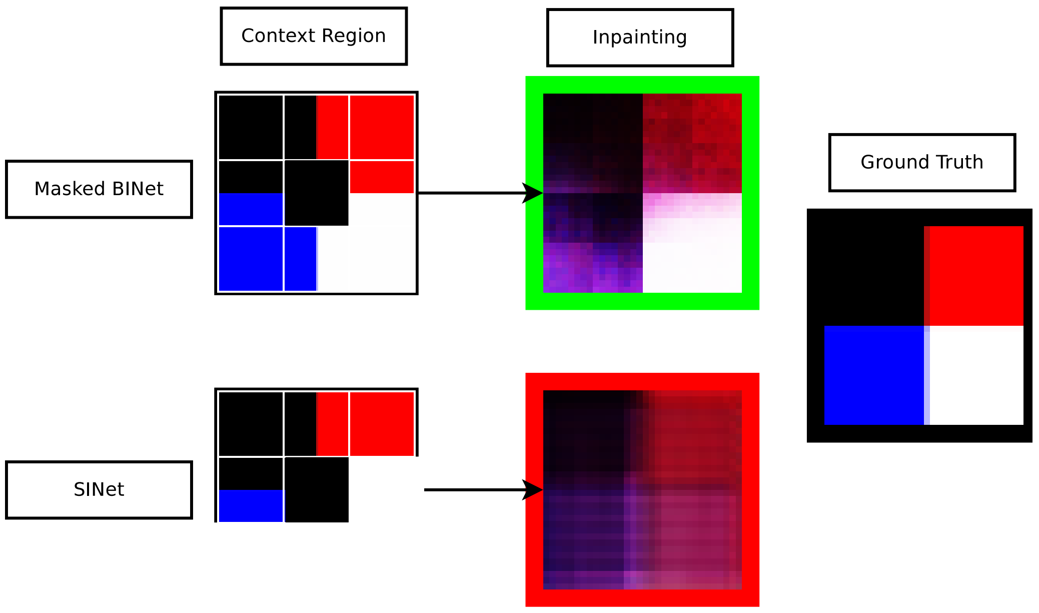
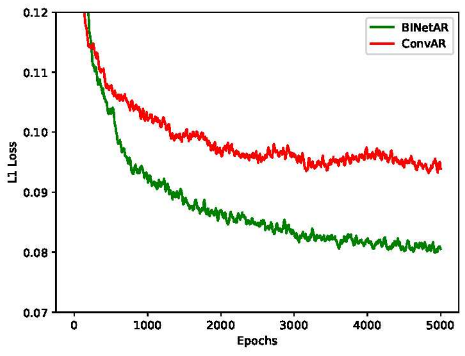


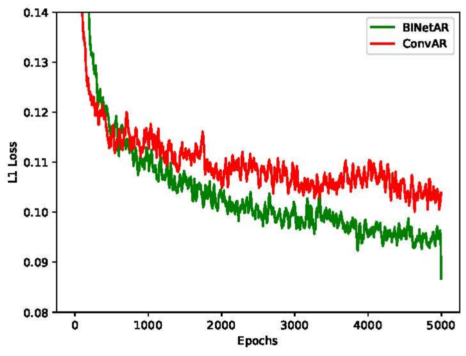
Figure 4.9: Comparison of inpainting performed by masked BINet and SINet given an artificial 32×32 image patch.

Does Inpainting Improve Compression using a Single Iteration?

To determine if teaching a model to inpaint from binary codes aids its compression capabilities, 1-iteration (0.125 bpp) implementations of BINetAR and the baseline ConvAR are pitted against each other. Figure 4.10 demonstrates how BINetAR outperforms ConvAR quantitatively in terms of training and validation loss. Losses represent the mean error between the ground truth and predicted patches and are indicative of the quality of the model's patch reconstructions. Figure 4.11 shows an assortment of images encoded by BINetAR and ConvAR. Note that in each case BINetAR produces images with a higher perceptual fidelity than ConvAR, according to the SSIM and PSNR scores achieved by its reconstructions. The images produced by BINetAR are qualitatively smoother than those of ConvAR at equally low bitrates, making BINetAR better suited for patch-based compression. The improved smoothness can be attributed to BINetAR's decoder which learns to constrain a patch to match its surroundings. All the images used here are from the CLIC validation set [82].



(a) Training Loss



(b) Validation Loss

Figure 4.10: Training and validation losses for 1-iteration implementations of BINetAR and the ConvAR baseline.

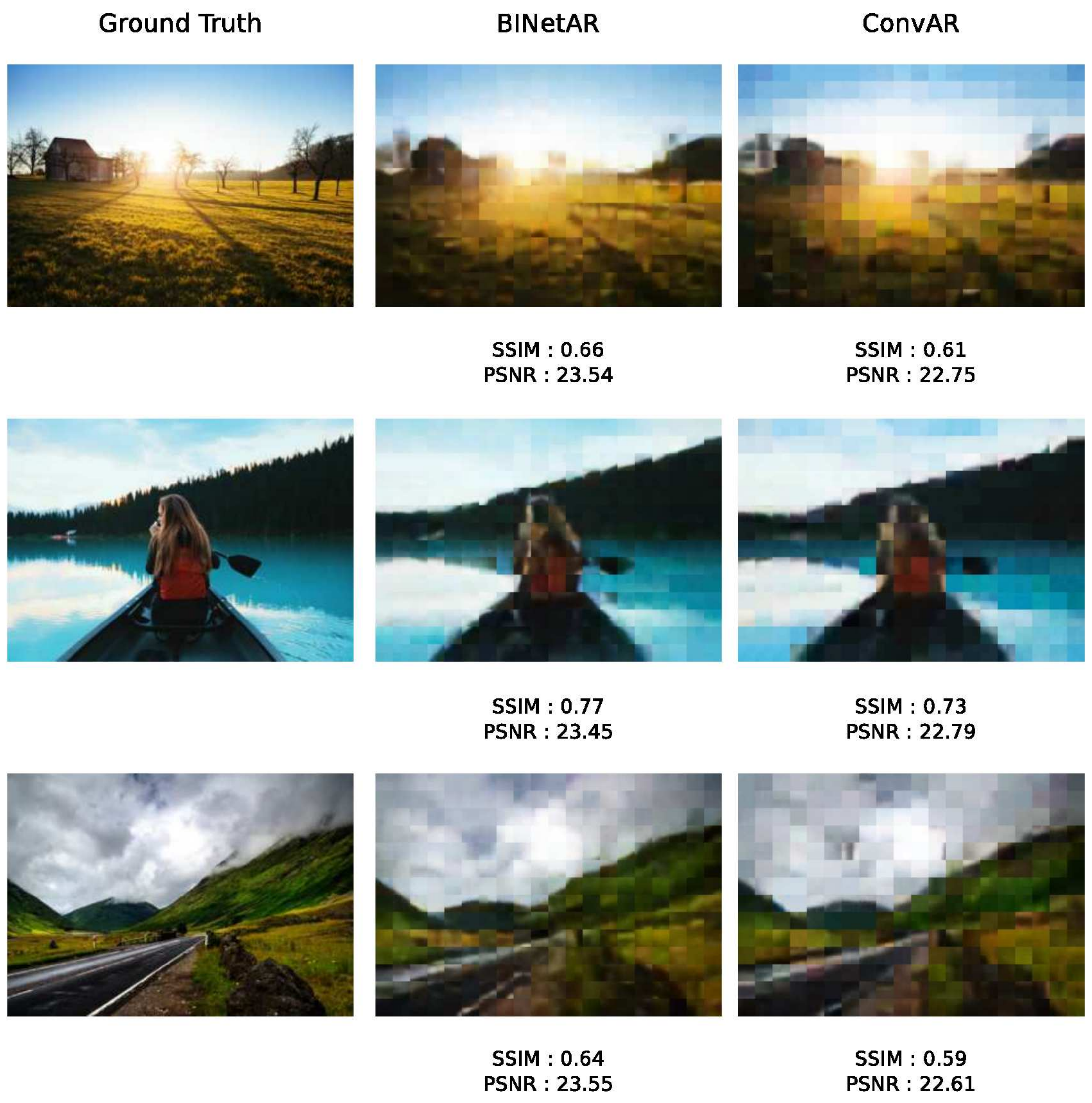


Figure 4.11: Full image reconstructions for 1-iteration implementations of BINetAR and the ConvAR baseline.

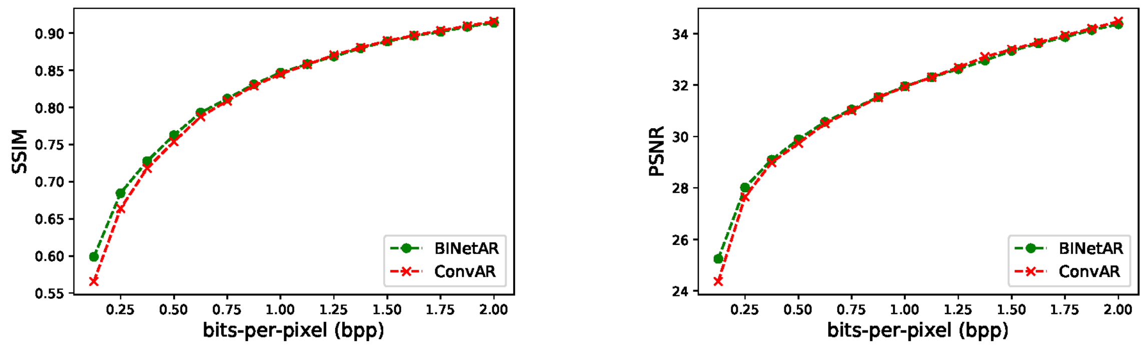
4.4.2. Quantitative Analysis: BINetAR vs. ConvAR

We now turn to quantitative analyses on the CLIC test set [82]. We train 16-iteration implementations of BINetAR and ConvAR to assess the effect of incorporating a single inpainting stage on the performance of an AR model. We first consider reconstruction of single patches (an intrinsic measure) and then consider the more realistic evaluation on full images.

Patch Reconstruction

We first assess the abilities of BINetAR and ConvAR to reconstruct single 32×32 image patches centre-cropped from the test set. Each model is trained to compress 32×32 patches, for an intrinsic evaluation of model performance. The resulting SSIM and PSNR rate-distortion curves are shown in Figures 4.12(a) and 4.12(b). The variable bit rate is achieved by varying the number of encoding iterations from $I = 1$ to $I = 16$. The figures indicate that at low bitrates close to the inpainting layer BINetAR gives a small but consistent improvement over ConvAR.

Dynamic bit assignment entails encoding different patch regions with varying bit allocations governed by a predetermined quality threshold such as PSNR. This aids compression as image regions are not necessarily equally complex. At low bitrates close to the inpainting layer BINetAR consistently produces patches of a higher quality than ConvAR. This means that if dynamic bit assignment were implemented, BINetAR would reach target quality thresholds after fewer encoding iterations (resulting in fewer bits) compared to ConvAR.



(a) SSIM (b) PSNR

Figure 4.12: BINetAR vs. ConvAR: rate-distortion curves for 32×32 image patches.

Full Image Reconstruction

The average PSNR and SSIM scores achieved by BINetAR and ConvAR on 224×320 test images are compared in Table 4.2, for various bit allocations. Although improvements are small, BINetAR consistently outperforms ConvAR across all bit depths. If one compares the first iteration of the two models, BINetAR outperforms ConvAR by 8% in terms of SSIM and results in a 3% relative improvement in PSNR. This comparison between the first iteration of the models is important as binary inpainting is only incorporated at the first stage of the BINetAR model.

Model	SSIM			PSNR		
	0.125 bpp	0.25 bpp	0.5 bpp	0.125 bpp	0.25 bpp	0.5 bpp
ConvAR	0.591	0.712	0.805	22.486	25.288	27.488
BINetAR	**0.639**	**0.732**	**0.813**	**23.222**	**25.643**	**27.623**

Table 4.2: BINetAR vs. ConvAR: SSIM and PSNR scores at various bit-per-pixel (bpp) allocations for 224×320 images.

4.4.3. Quantitative Analysis: BINetOSR vs. ConvGRU-OSR

Sixteen-iteration implementations of BINetOSR and ConvGRU-OSR are trained to assess the effect of incorporating a single inpainting stage on the performance of an OSR model. Models are again evaluated on the CLIC test set [82].

Patch Reconstruction

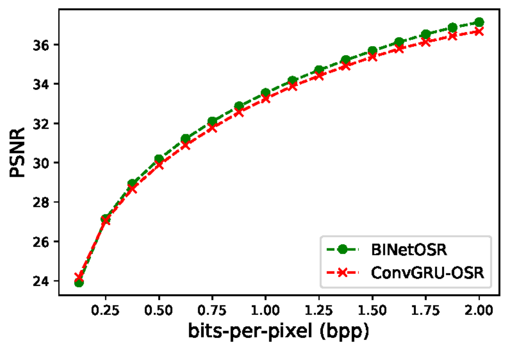
We first asses BINetOSR's and ConvGRU-OSR's intrinsic capacity to reconstruct 32×32 patches center-cropped from the test data. The resulting areas under the PSNR and SSIM rate-distortion curves are given in Table 4.3. Note that a greater area is indicative of increased perceptual quality across all sixteen allocated bitrates. Table 4.3 shows that incorporating learned inpainting into just one iteration of the ConvGRU-OSR model effectively increases its area under the PSNR and SSIM rate-distortion curves.

Model	Area under the curve	
	PSNR	SSIM
ConvGRU-OSR	1.661	64.352
BINetOSR	**1.668**	**65.10**

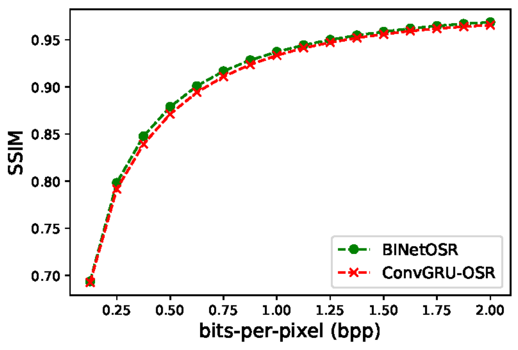
Table 4.3: BINetOSR vs. ConvGRU-OSR: area under the curve for SSIM and PSNR rate-distortion, calculated on 32×32 image patches.

Full Image Reconstruction

The PSNR and SSIM curves achieved by BINetOSR and ConvGRU-OSR on 224×320 test images are shown in Figures 4.13(a) and 4.13(b). Unlike the AR model, inpainting gains are more pronounced at stages further from the inpainting layer, as recurrence allows BINetOSR to better propagate inpainting information to later decoding stages. This forces the first stage to learn an inpainting strategy that is beneficial to the system as a whole as opposed to BINetAR where improvements are concentrated around the inpainting layer. Again, while performance gains are small, they are consistent over the bit rates considered.



(a) PSNR



(b) SSIM

Figure 4.13: BINetOSR vs. ConvGRU-OSR: rate-distortion curves for complete 224×320 images.

4.4.4. Quantitative Analysis: BINet vs. Standard Codecs

Up to now we focused on incorporating BINet into the ConvAR and ConvGRU-OSR approaches in order to investigate the effect of binary inpainting on patch-based compression in isolation. Here we compare BINet to standard image codecs. Evaluation is carried out on full images from the Kodak dataset [85] resized to 320×224 pixels. Figures 4.14 and 4.15 compare BINet's PSNR and SSIM performance to that of WebP and JPEG. BINetAR outperforms JPEG at low bitrates, but gradually worsens at bitrates produced by stages further from the inpainting layer. JPEG is patch-based, and Figure 4.16 indicates that block artefacts arising through the use of an independent patch-based encoding scheme can be suppressed by BINetAR for enhanced quality at shallow bit allocations.

We showed in Section 4.4 that BINet is capable of learning more complex inpainting predictions than WebP. Although the inclusion of a binary inpainting stage does consistently improve ConvGRU-OSR's performance (Figure 4.13), BINetOSR still falls short of outperforming WebP (Figures 4.14, 4.15 and 4.17). Using the same decoder module for inpainting and image patch reconstruction may result in a conflict between learning compression and the high quality inpainting shown in Section 4.4. One must also take into consideration that WebP and JPEG's codes are further compressed by lossless entropy coding, whereas BINet's codes are not.

Our aim here was not to achieve state-of-the-art performance, but rather to investigate whether binary inpainting improves patch-based image compression in a deep neural network model, and this was shown in Sections 4.4.2 and 4.4.3. Any model can be used as the basis in the BINet framework, and future work will consider incorporating BINet into the more powerful recurrent models of [13].

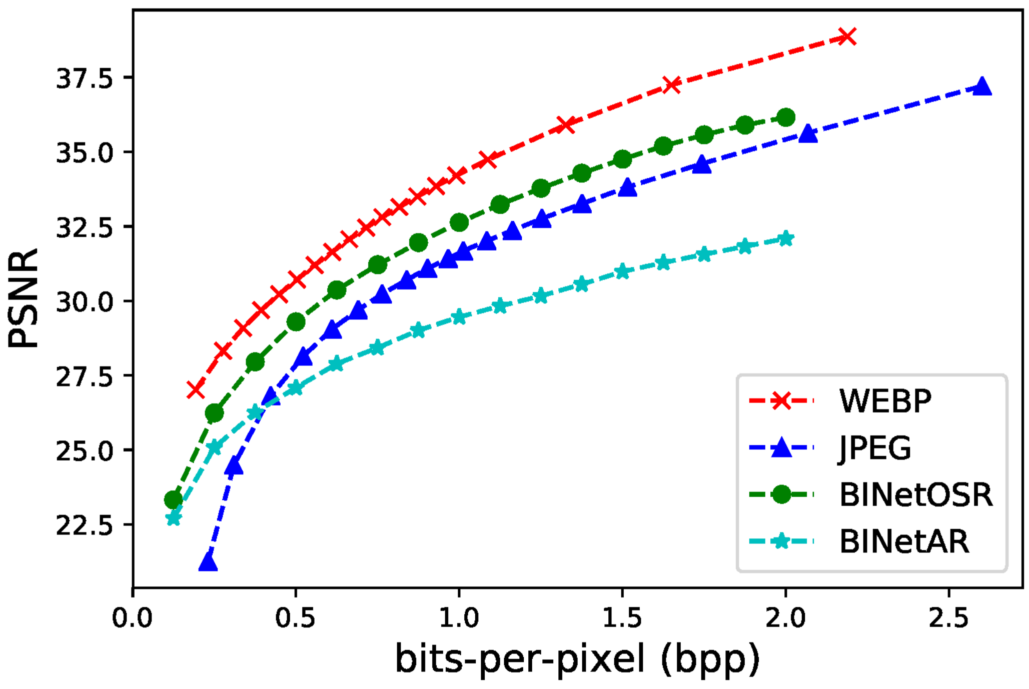


Figure 4.14: PSNR rate-distortion curves comparing BINet to standard image codecs on the Kodak dataset [85].

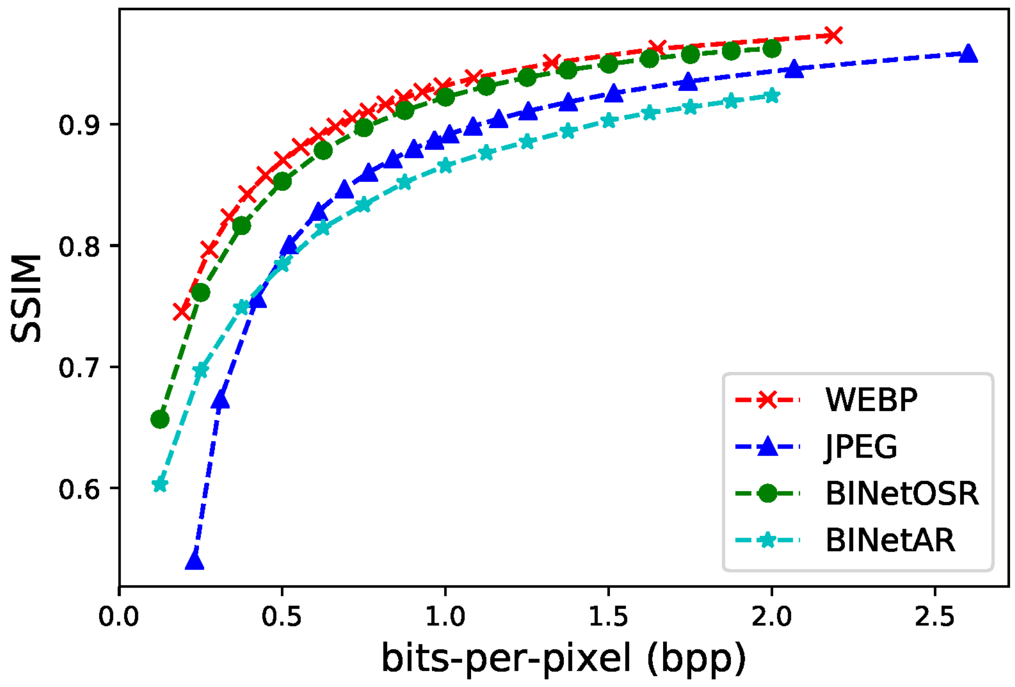


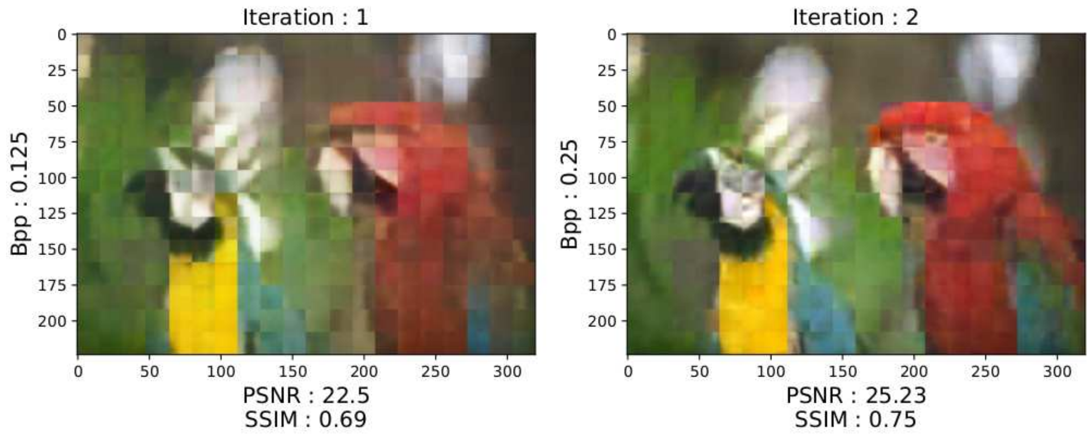
Figure 4.15: SSIM rate-distortion curves comparing BINet to standard image codecs on the Kodak dataset [85].

4.5. Chapter Summary

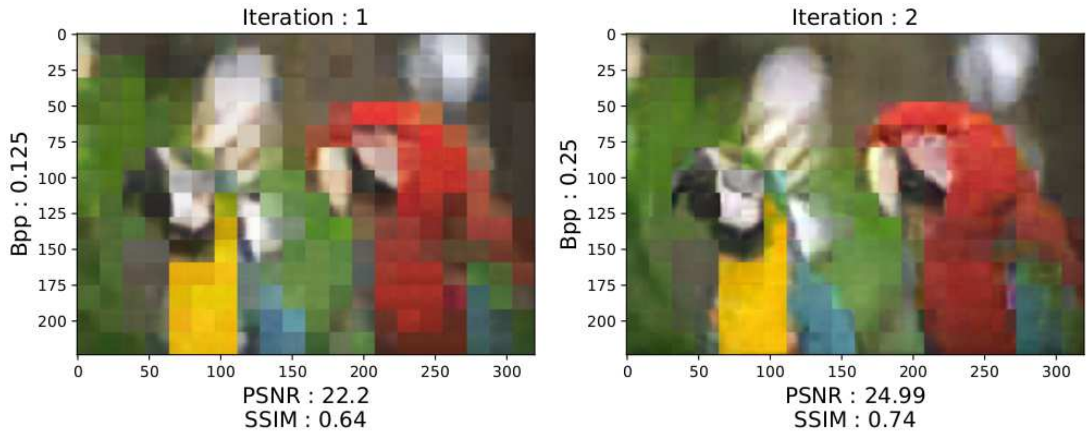
We introduced the Binary Inpainting Network (BINet), a novel framework that can be used to improve an existing system for patch-based image compression. Building on ideas from image inpainting as well as deep image compression, BINet is novel in two particular ways. Firstly, in contrast to work on inpainting, BINet incorporates explicit binarisation in an encoder module, which allows it to be used for compression. Secondly, in contrast to most deep compression models, BINet incorporates information from adjacent patches when decoding a particular patch. The result is a patch-based compression method which allows for parallelised inpainting from a full-context region without access to original image data. In quantitative evaluations, we showed that BINet yields small but consistent improvements over baselines without inpainting. Qualitatively we showed that BINet results in fewer block artefacts at shallow bitrates compared to standard image codecs, resulting in smoother image reconstructions.

Apart from incorporating BINet into more advanced neural architectures in future work, we aim to also explore alternative applications for binary inpainting such as binary error correction and patch-based video-frame interpolation.

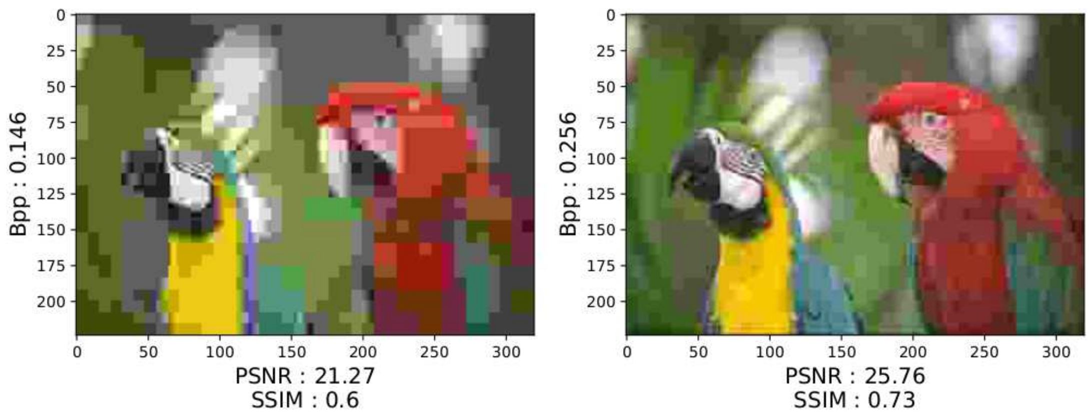
In this chapter we have shown that it is possible to predict image content from binary codes. Building on this, our goal in the following chapter is to learn binary motion codes for video inter-frame prediction.



(a) BINetAR

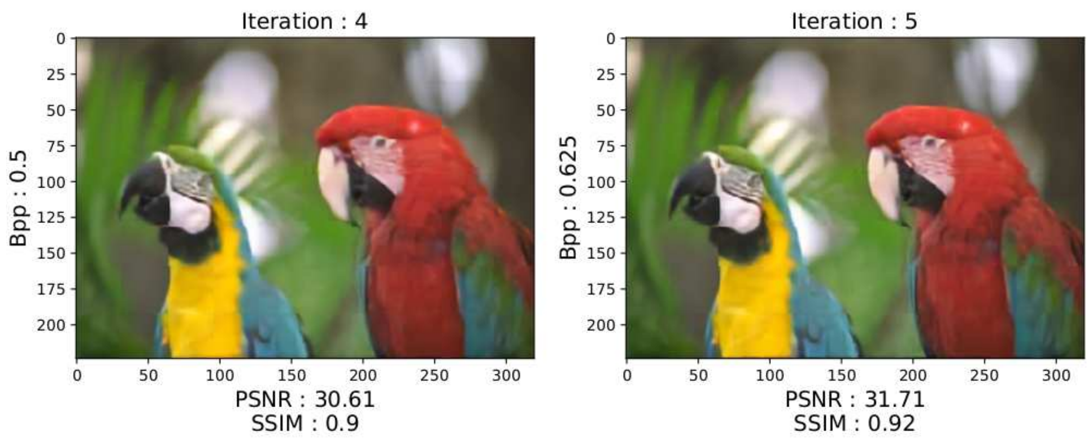


(b) ConvAR

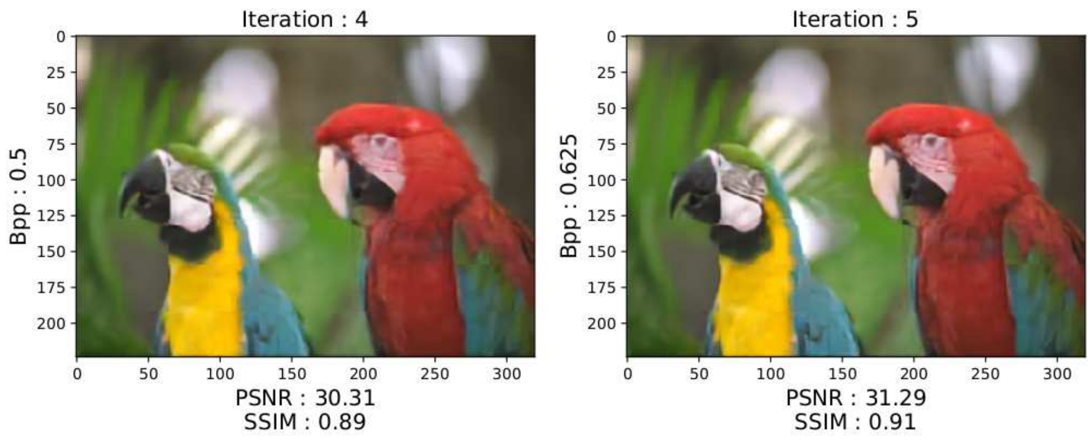


(c) JPEG

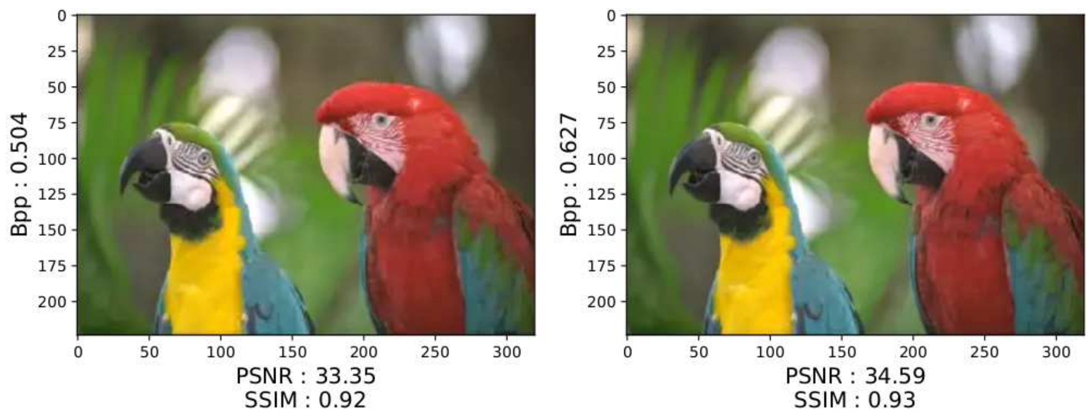
Figure 4.16: BINetAR vs. ConvAR vs. JPEG 224×320 image reconstructions.



(a) BINetOSR



(b) ConvGRU-OSR



(c) WebP

Figure 4.17: BINetOSR vs. ConvGRU (OSR) vs. WebP 224 $\times$ 320 image reconstructions.

Chapter 5

Deep motion estimation for parallel inter-frame prediction in video compression

Earlier we showed how binary inpainting improves compression of reference frames through deep intra-frame prediction. Here we tackle inter-frame compression: predicting a set of target video frames from past or future reference frames.

Standard video codecs and recent developments in deep video compression rely on optical flow to guide inter-frame prediction: pixels from past or future reference frames are moved via motion vectors to predict a set of target video frames. A video can then be encoded by compressing and transmitting only the motion vectors and reference frames.

In this chapter, we propose to *learn* binary motion codes that are encoded directly based on an input video sequence, instead of using explicitly encoded optical flow vectors. This allows us to model complex motion (e.g. warping, rotation and occlusion) that is not limited to the 2D translations rendered by standard optical flow vectors. Our binary motion codes are learned as part of a single neural network which also learns to directly compress and decode them. As an added benefit, the resulting binary motion codes support parallel video frame decoding; in contrast, flow-based methods require that both motion estimation and compensation be performed sequentially on a frame-by-frame basis.

Building on recent advances in deep image compression, we also introduce 3D dynamic bit assignment as a means of shifting spatially varying bit allocations through time to adapt to object displacements caused by motion. This results in significant bit savings without degrading video prediction quality.

By replacing the optical flow based block-motion algorithms found in an existing video codec with our learned inter-frame prediction model, we are able to outperform the standard H.264 and H.265 video codecs across a range of low bitrate operating points.

The work in this chapter is currently submitted for publication in the *Journal of Visual Communication and Image Representation*. A full implementation of the code developed as part of this chapter is made available online: DeepVideo.[1]

[1] https://github.com/adnortje/deepvideo

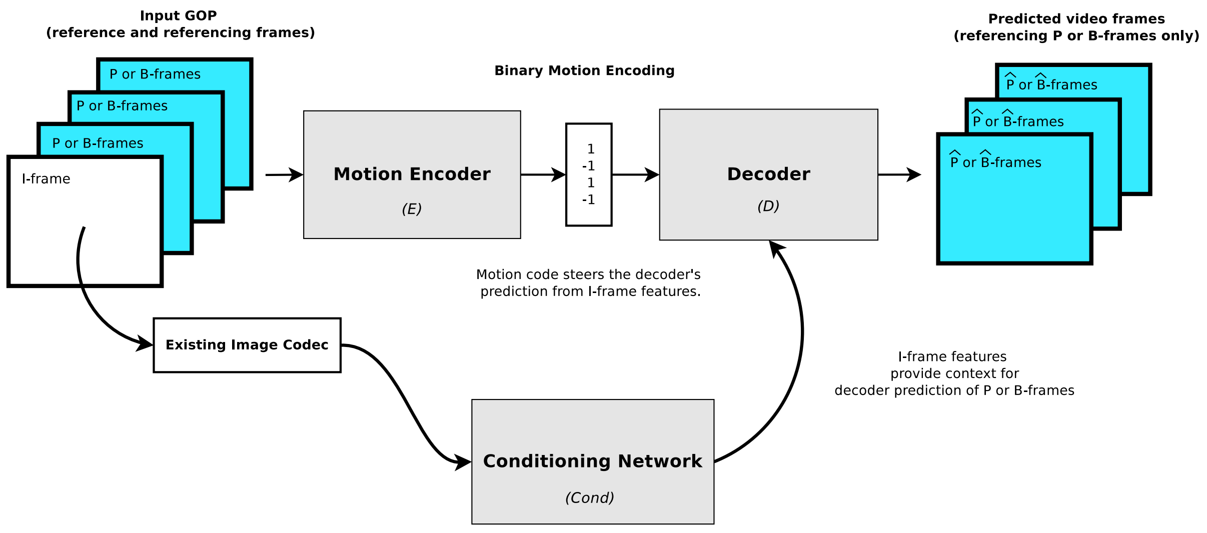

Input GOP
(reference and referencing frames)
P or B-frames
P or B-frames
P or B-frames
I-frame
Binary Motion Encoding
Motion Encoder
(E)
1
-1
1
-1
Decoder
(D)
Predicted video frames
(referencing P or B-frames only)
$\hat{P}$ or $\hat{B}$-frames
$\hat{P}$ or $\hat{B}$-frames
$\hat{P}$ or $\hat{B}$-frames
Motion code steers the decoder's prediction from I-frame features.
Existing Image Codec
I-frame features provide context for decoder prediction of P or B-frames
Conditioning Network
(Cond)

Figure 5.1: Video prediction network architectural overview. A learned binary motion encoding either guides the extrapolation of P-frames (Predicted-frames) from past I-frames (Intra-frame) or the interpolation of B-frames (Bi-directional-frame) from bounding past and future I-frames. Reference I-frames are coded and decoded independently with an existing image codec.

5.1. Related Work

5.1.1. Standard Video Codecs

Standard video codecs, such as H.264 [1] and H.265 [2], take advantage of the spatial and temporal redundancies in videos to aid compression. They assign video frames into one of three groups [86]:

- **I-frames**, or ‘intra-frames’, are compressed independently from surrounding frames by means of an image codec.
- **P-frames** are ‘predicted-frames’ extrapolated from past frames.
- **B-frames** are ‘bi-directionally’ interpolated from bounding past and future frames.

The compressed I-frames are transmitted directly, while the extrapolation and interpolation of P-frames and B-frames are achieved via the transmission of highly compressible optical flow vectors [6]. These motion vectors (MVs) convey motion by specifying the movement of pixels from one frame to another.

Dense optical flow [28] produces too many MVs for efficient compression (one per pixel location). Consequently, standard video codecs resort to block-based motion estimation and compensation techniques [1, 2, 87]. This entails partitioning video frames into patches called macroblocks. In the motion estimation step, each macroblock in the current frame is related to the location of the most similar macroblock in a past or future reference frame by means of an MV which contains its displacement in the x and y directions. Searching for representative macroblocks in the reference frame is computationally expensive and numerous search algorithms have been proposed to help speed up this process [88–92]. After the transmission of a reference I-frame, only MVs need be transmitted to motion-compensate macroblocks in the I-frame and form predictions of the subsequent frames within a video sequence. Standard image compression is used to encode the residuals (differences) between the vector-based motion predictions and the original video frames to improve reconstruction quality.

Block-based motion prediction, although effective, is prone to block artefacts and only allows for sequential decoding [22]. Furthermore, these algorithms suffer from hand-tuned parameterisation and lack the ability to undergo joint optimisation with the rest of a video compression system. We present a deep learning approach to video frame prediction that can be optimised end-to-end as part of a larger video compression system. Given I-frame context, our model is also able to decode P-frames or B-frames in parallel without the additional overhead of motion-estimation search.

Our approach is illustrated at a high level in Figure 5.1. Video *interpolation* aims to predict a set of unseen intermediate frames from a pair of bordering reference frames. Video *extrapolation*, on the other hand, forecasts unobserved video frames based on those that have occurred in the past. In our approach, an encoder E learns how to produce a binary motion encoding, shown in the middle of the figure, with binarisation performed directly within the neural network. The resulting learned binary motion code is subsequently used to guide the extrapolation of P-frames conditioned on a past I-frame, or the interpolation of B-frames conditioned on bounding past and future I-frames. Interpolation or extrapolation is performed by the decoder D and the conditioning is indicated through the 'Cond' block in the figure, which extracts features from I-frames that have been compressed and decompressed independently by an existing image codec. We therefore view the decoding of P-frames or B-frames in video compression as motion guided interpolation or extrapolation, where a low dimensional learned binary motion code helps direct prediction from I-frames.

5.1.2. Deep Learning

In work interested purely in prediction (without compressing), deep learning has been shown to produce high quality video frame interpolations [93–100] and extrapolations [54, 101–103] for small time-steps. Typically, unseen video frames are predicted solely based on the reference frame [94, 97, 98]. For predicting unseen frames over longer time-spans (as would be the case if we were interested in video compression), additional information is required.

In video compression, we do not need to rely solely on the reference frame content: we can estimate the motion from the actual unseen video frames, compress these motion encodings, and then transmit this together with the compressed versions of the reference frames. This extra motion information can enable video frame prediction over extended timespans [3]. Based on this idea, deep learning has recently been applied to video compression, producing models capable of outperforming standard video codecs (H.264, H.265) at certain bit allocations [3–5, 104–106]. These models combine state-of-the-art image compression, flow prediction and entropy coding networks to produce end-to-end optimisable video compression systems. Despite their success, whole systems are evaluated as a single unit, making it difficult to discern to what extent each individual component outperforms its more conventional standard implementation. In this chapter, we focus specifically on motion compression for video prediction—we consider P- and B-frame prediction in isolation, decoupled from all other compression components.

In [3] video frames are hierarchically interpolated by warping input reference frame features with standard block-MVs, while discrete representations of motion are learned by encoding optical flow patterns in [4,5]. Optical flow vectors describe how pixels in a video frame should be moved over time to best estimate true object and camera motion [21]. It is effective at modelling translational motion, but fails to capture more complex transformations such as rotation, warping, occlusion and changes in lighting [6]. In [3–5] this is addressed by jointly compressing the residuals produced after flow compensation. In this thesis, rather than using optical flow, we show that it is possible to learn compact encodings that are representative of complex motion directly from a video sequence. More specifically, we train an encoder network to produce learned binary motion codes which guide the prediction of P-frames and B-frames from I-frame context at the decoder. Experiments show that the complex motion contained in our binary motion codes outperforms that of conventional optical flow. The codes produced by our network could, therefore, provide an alternate means of motion conditioning for applications that are currently reliant on optical flow-based methods [3–5, 107].

Different spatial and temporal locations in a video sequence are not necessarily equally complex. In image compression, it has been shown that compression rates can be improved by varying bitrates such that less complex image regions are assigned fewer bits [13, 15, 17, 18, 108]. Varying the bitrate temporally in accordance to complexity of motion is equally important in video compression [1, 2, 87]. Most videos contain still segments interspersed with sequences depicting rapid motion. A model with access to reference frame context at its decoder should learn to encode very little information for still video segments and allocate the bulk of its bits to intervals containing a high degree of motion. In [4, 109] recurrence is used to sequentially maintain state information across time such that previously decoded information need not be re-encoded. We, on the other hand, extend the 2D content-weighted image compression technique in [17] to three dimensions and present parallelised P-frame and B-frame video compression models that learn to vary bitrate both spatially and temporally. Experiments show how our approach to 3D dynamic bit assignment substantially reduces the bitrate of a motion encoding model without adversely affecting its reconstruction quality.

5.1.3. Contributions

We proceed as follows. First we give a detailed description of our P-frame and B-frame compression architectures. We then formulate our approach to 3D content-aware bit weighting and demonstrate its applicability to bitrate optimisation. Finally, compression efficiency is evaluated in terms of various video quality metrics (PSNR, SSIM, VMAF and EPE). We demonstrate that our models' P-frame and B-frame predictions outperform those of the block-motion prediction algorithms employed by standard video codecs such as H.264 and H.265. Additional experiments are carried out to determine the impact of an optical flow-based loss term and if multi-scale convolutions result in richer motion sampling. We find that including multi-scale convolutions in our encoder architecture slightly improves the quality of our model's video frame predictions. On the other hand, limiting our model to learning pixel-wise translational motion with a flow loss term worsens its prediction quality. This indicates that we are able to learn more representative motion than conventional optical flow.

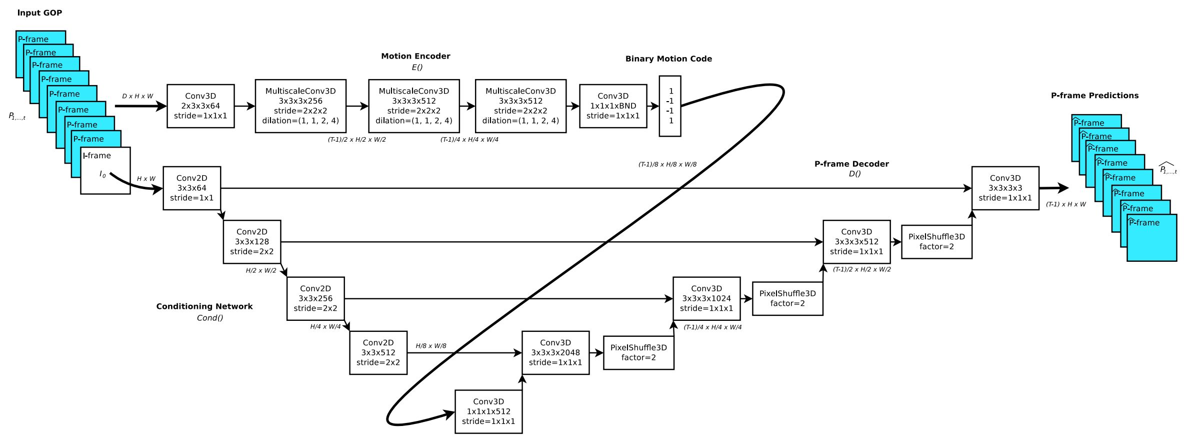


Figure 5.2: The P-frame prediction network (P-FrameNet) used to extrapolate video frames from a past I-frame.

5.2. Video Frame Prediction Architecture

5.2.1. Architectural Overview

Figure 5.1 illustrates our approach to P-frame and B-frame prediction. The neural network encoder E compresses and binarises the motion occurring in a Group Of Pictures (GOP): a video segment containing designated reference (I) and referencing (P or B) frames. Binarisation via thresholding is non-differentiable. To perform this operation directly within a neural network, we therefore resort to the stochastic binarisation function presented in Section 3.1.3 [7]: during training, each encoder output, which lies within $(-1, 1)$, is made to take one of two distinct values in the set $\{-1, 1\}$ by adding uniform quantisation noise. This allows for a straight through estimate [80] of gradients, i.e. gradients flow through the binarisation layer unchanged. At the decoder D, I-frame features extracted by the conditioning network 'Cond' are transformed based on the information held in the binarised motion encoding to predict the P or B referencing frames. Note that 'Cond' is not responsible for I-frame compression: this is done by an existing image codec.

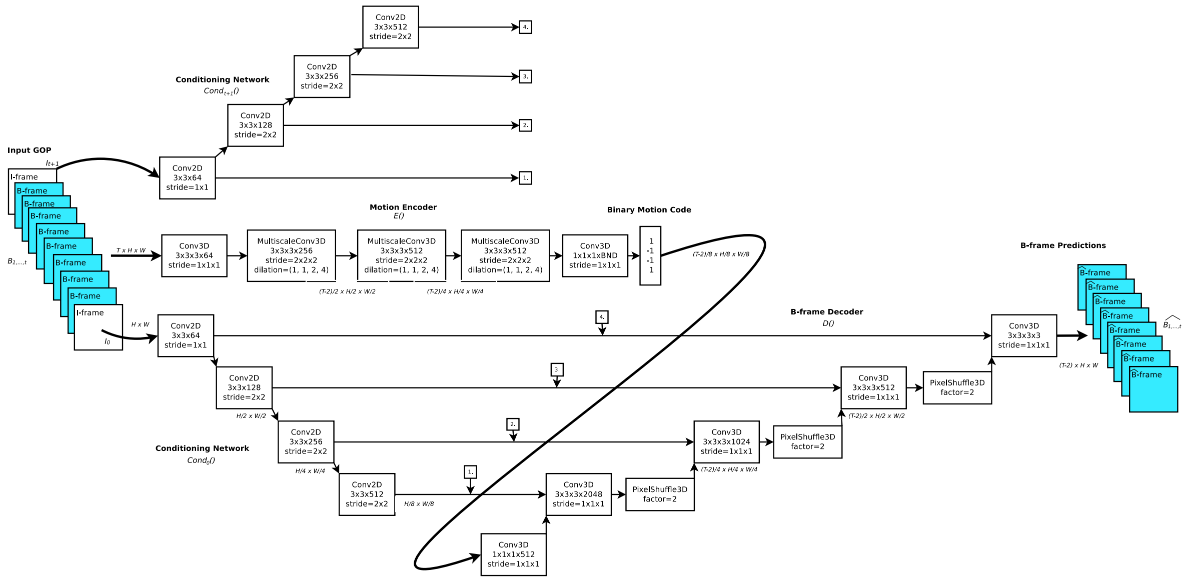

Conditioning Network
$Cond_{t+1}()$
Conv2D 3x3x64 stride=1x1
Conv2D 3x3x128 stride=2x2
Conv2D 3x3x256 stride=2x2
Conv2D 3x3x512 stride=2x2
1.
2.
3.
4.
Input GOP
I_{t+1}
I-frame
B-frame
$B_{1,...,t}$
I-frame
I_0
T x H x W
Motion Encoder
E()
Conv3D 3x3x3x64 stride=1x1x1
MultiscaleConv3D 3x3x3x256 stride=2x2x2 dilation=(1, 1, 2, 4)
MultiscaleConv3D 3x3x3x512 stride=2x2x2 dilation=(1, 1, 2, 4)
MultiscaleConv3D 3x3x3x512 stride=2x2x2 dilation=(1, 1, 2, 4)
(T-2)/2 x H/2 x W/2
(T-2)/4 x H/4 x W/4
Binary Motion Code
Conv3D 1x1x1xBND stride=1x1x1
1 -1 -1 1
(T-2)/8 x H/8 x W/8
H x W
Conv2D 3x3x64 stride=1x1
Conv2D 3x3x128 stride=2x2
H/2 x W/2
Conv2D 3x3x256 stride=2x2
H/4 x W/4
Conv2D 3x3x512 stride=2x2
H/8 x W/8
Conditioning Network
$Cond_0()$
Conv3D 1x1x1x512 stride=1x1x1
Conv3D 3x3x3x2048 stride=1x1x1
PixelShuffle3D factor=2
(T-2)/4 x H/4 x W/4
Conv3D 3x3x3x1024 stride=1x1x1
PixelShuffle3D factor=2
(T-2)/2 x H/2 x W/2
Conv3D 3x3x3x512 stride=1x1x1
PixelShuffle3D factor=2
B-frame Decoder
D()
Conv3D 3x3x3x3 stride=1x1x1
(T-2) x H x W
B-frame Predictions
$\widehat{B}$-frame
$\widehat{B_{1,...,t}}$

Figure 5.3: The B-frame prediction network (B-FrameNet) used to bi-directionally interpolate video frames from bounding past and future I-frames.

5.2.2. P-frame and B-frame Prediction Networks

Figures 5.2 and 5.3 illustrate our P-frame and B-frame prediction networks in greater detail. Equations (5.1) and (5.2) summarise the P-frame and B-frame prediction processes depicted in Figures 5.2 and 5.3, respectively.

$$\widehat{\boldsymbol{P}_{1,\ldots,t}} = D\left(E(\boldsymbol{I}_0, \boldsymbol{P}_{1,\ldots,t}), \mathrm{Cond}(\boldsymbol{I}_0)\right) \tag{5.1}$$

$$\widehat{\boldsymbol{B}_{1,\ldots,t}} = D\left(E(\boldsymbol{I}_0, \boldsymbol{B}_{1,\ldots,t}, \boldsymbol{I}_{t+1}), \mathrm{Cond}_0(\boldsymbol{I}_0), \mathrm{Cond}_t(\boldsymbol{I}_{t+1})\right) \tag{5.2}$$

The decoder $D(\cdot)$ uses context derived from reference frames $\boldsymbol{I}$ by the conditioning network $\mathrm{Cond}(\cdot)$ to predict a sequence of $t-1$ frames, $\boldsymbol{P}_{1,\ldots,t}$ or $\boldsymbol{B}_{1,\ldots,t}$. The prediction process is supervised by a binarised motion encoding, $E(\cdot)$, of the original GOP sequence. Because the encoder always compresses the input GOP's width, height and time axes by a factor of 8, P-FrameNet and B-FrameNet's bitrate is determined by the number of output channels we set in the final encoder layer, C_{bnd}. We denote predicted video-frames as either $\boldsymbol{P}$ or $\boldsymbol{B}$, depending on whether the decoder performs motion guided extrapolation or interpolation. The decoder performs extrapolation, Figure 5.2, when conditioned on a single I-frame, $\boldsymbol{I}_0$, and interpolation, Figure 5.3, when conditioned on a pair of bounding I-frames, $\boldsymbol{I}_0$ and $\boldsymbol{I}_{t+1}$. During training we use a L_2 reconstruction loss:

$$L_R = ||\boldsymbol{B} - \hat{\boldsymbol{B}}||^2 \quad \text{or} \quad ||\boldsymbol{P} - \hat{\boldsymbol{P}}||^2. \tag{5.3}$$

Throughout training we give $D(\cdot)$ access to the original I-frame content, but at test time $\boldsymbol{I}_0$ and $\boldsymbol{I}_{t+1}$ are encoded and decoded independently by an existing image codec.

Motion in video often occurs at different scales. To account for this, we implement the 3D multi-scale convolutional layers [20, 55] discussed in Section 2.3.1 in our encoder network as a lightweight substitute for deep pyramidal decomposition [11, 54]. Each multi-scale convolution combines filters with different dilation factors for more diverse motion sampling across a range of spatial and temporal scales, and can be seen as a type of learned scale invariant feature transform (SIFT) [56]. In Section 5.4.2 we demonstrate how the inclusion of multi-scale convolutions consistently improves video frame prediction quality. As shown in Figures 5.2 and 5.3, manifold layers from the conditioning network 'Cond' are joined to the decoder D in a fashion reminiscent of the U-Net [60] architecture, prevalent in previous video interpolation work [94, 97]. I-frame conditioning at the decoder enables P-FrameNet and B-FrameNet to learn motion compensation (how to transform I-frame content) instead of just compressing input P- and B-frames directly. The experiment in Section 5.4.1 shows that the binary motion codes learnt through I-frame conditioning are more easily compressible than raw video frames. Upscaling at the decoder is accomplished via pixel-shuffling [71], an efficient alternative to transposed convolutions. Scene changes and motion complexity often dictate GOP length selection in standard codecs [6]. Our designs are, therefore, fully convolutional to ensure that they are able to accommodate a diverse range of input frame-sizes and dynamic GOP lengths.

5.2.3. 3D Dynamic Bit Assignment

In order to vary the bitrate of our binary motion codes, we leverage [17]'s approach to content-weighted image compression and *learn* to vary bitrate across an extra dimension: time. Video regions that are smooth and predominantly stationary are easier to compress than those containing rich texture and rapid motion. An ideal motion compression model should, therefore, actively adapt its bitrate according to fluctuations in video complexity by assigning fewer bits to simplistic video regions and vice versa. As it stands, our encoder architecture allocates a fixed number of bits to each spatio-temporal location in its code-space, specified by C_{bnd}, the number of channels in its binarisation layer. Based on [17], we learn a 3D bit-distribution-map, $\boldsymbol{B}_{\text{map}}$, that determines how many bit channels are allocated to our binary motion encoding per point in space-time.

Figure 5.4 illustrates the key stages in our approach to 3D dynamic bit assignment. First we learn $\boldsymbol{B}_{\text{map}}$, shown at *(i)* in the figure, from the input GOP by passing features extracted by the penultimate encoder layer through a 3D convolutional network. The 3D bit-distribution-map $\boldsymbol{B}_{\text{map}}$ is a single-channel feature map whose values fall in the range $(0, 1)$ and whose spatial and temporal size is the same as the binary motion code produced by the encoder, represented by the blue cubes at *(ii)*. While in the figures thus far we have indicated the serialised motion encoding with a box of -1 and 1s, the cubes at *(ii)* in this figure indicate the individual learned MVs for each video frame over its width and height (these are serialised later, as explained below). The lighter regions in $\boldsymbol{B}_{\text{map}}$ are higher valued and identify video regions that should be allocated more bits (channels). Following [17], we portion the available C_{bnd} bits produced for each video frame by the encoder into L groups each containing $\frac{C_{\text{bnd}}}{L}$ bits. With $\lfloor\cdot\rfloor$ denoting the mathematical floor operator, each element, $b_{t,h,w}$, in $\boldsymbol{B}_{\text{map}}$ is quantised to one of L integer levels,

$$Q_L(b_{t,h,w}) = \lfloor L b_{t,h,w} \rfloor, \tag{5.4}$$

to decide how many bit levels need to be retained per point in space-time. To avoid allocating non-integer bit numbers we require that C_{bnd} be cleanly divisible by L and $L \leq C_{\text{bnd}}$. Guided by $Q_L(\boldsymbol{B}_{\text{map}})$ at *(iii)*, we populate a mask $\boldsymbol{M}$, shown at *(iv)*, that zeros-out unnecessary bit channels produced by the encoder at *(ii)*:

$$m_{c,t,h,w} = \begin{cases} 1, & \text{if } c \leq \frac{C_{\text{bnd}}}{L} Q_L(b_{t,h,w}), \\ 0, & \text{otherwise} \end{cases}. \tag{5.5}$$

The cubes at *(v)* shows how masked bits (zeros) are cropped-out prior to the transmission of the serialised motion bitstream. Zeros are reinstated at the decoder by zero-padding each channel to C_{bnd} (the maximum bit-length). After multiplication by $\boldsymbol{M}$ and zero-cropping, the number of bits transmitted per point in space-time is reduced from C_{bnd} to $\frac{C_{\text{bnd}}}{L} Q_L(b_{t,h,w})$. In order for the decoder to reshape the serial bitstream correctly, a binarised version of $Q_L(\boldsymbol{B}_{\text{map}})$ is sent separately as additional overhead at *(vi)*. The integer values in $Q_L(\boldsymbol{B}_{\text{map}})$ are binarised using base-2 expansion [23] for transmission.

Realising that our serial bit-count is proportional to the summation over $\boldsymbol{B}_{\text{map}}$, we can use an additional loss term,

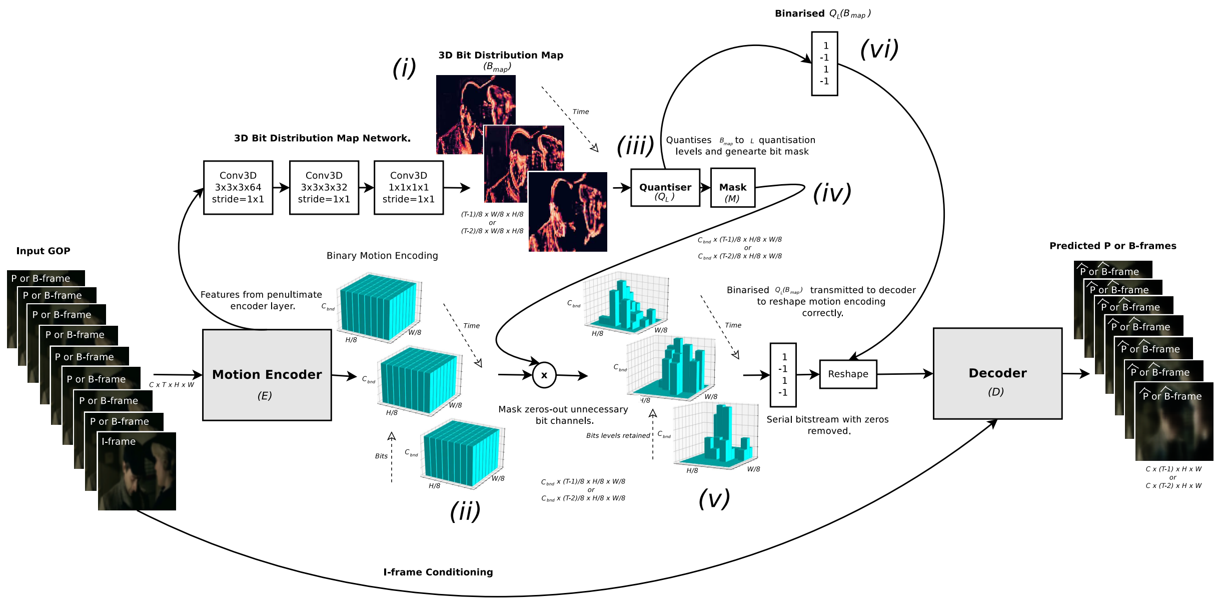
$$L_B = \sum_{t,h,w} b_{t,h,w}, \tag{5.6}$$

to drive down our model's bitrate during training [17]. L_B penalises bitrates above zero. This prevents assigning bits to stationary video regions that can be deduced from I-frame context alone.

Both the mask formation, *(iv)*, and quantisation functions, *(iii)*, in equations (5.4) and (5.5) are non-differentiable. Luckily, using straight through estimation [17, 80] again, the gradient of $\boldsymbol{M}$ with respect to $b_{t,h,w}$ can be approximated by,

$$\frac{\partial m_{c,t,h,w}}{\partial b_{t,h,w}} = \begin{cases} L, & \text{if } L b_{t,h,w} - 1 \leq \lceil \frac{cL}{C_{\text{bnd}}} \rceil \leq L b_{t,h,w} + 2, \\ 0, & \text{otherwise} \end{cases}. \tag{5.7}$$

We show the benefit of this 3D dynamic bit assignment approach experimentally in Section 5.4.5.



Binarised $Q_L(B_{map})$
1
-1
1
-1
(vi)
(i)
3D Bit Distribution Map
(B_{map})
Time
3D Bit Distribution Map Network.
(iii)
Quantises B_{map} to L quantisation levels and genearte bit mask
Conv3D 3x3x3x64 stride=1x1
Conv3D 3x3x3x32 stride=1x1
Conv3D 1x1x1x1 stride=1x1
Quantiser (Q_L)
Mask (M)
(iv)
(T-1)/8 x W/8 x H/8 or (T-2)/8 x W/8 x H/8
C_{bnd} x (T-1)/8 x H/8 x W/8 or C_{bnd} x (T-2)/8 x H/8 x W/8
Input GOP
Binary Motion Encoding
Predicted P or B-frames
Features from penultimate encoder layer.
Binarised $Q_L(B_{map})$ transmitted to decoder to reshape motion encoding correctly.
P or B-frame
I-frame
C x T x H x W
Motion Encoder
(E)
x
Reshape
Decoder
(D)
Mask zeros-out unnecessary bit channels.
Serial bitstream with zeros removed.
Bits levels retained
Bits
C_{bnd}
H/8
W/8
C_{bnd} x (T-1)/8 x H/8 x W/8 or C_{bnd} x (T-2)/8 x H/8 x W/8
(ii)
(v)
C x (T-1) x H x W or C x (T-2) x H x W
I-frame Conditioning

Figure 5.4: 3D dynamic bit assignment incorporated into a video frame prediction model to vary its bit allocations across space-time. In this figure the motion encoding bit-space is represented in its true multi-dimensional form by blue blocks. $\boldsymbol{B}_{\text{map}}$, indicated by *(i)* in the figure, is used to generate a mask $\boldsymbol{M}$ at *(iv)* that crops out unnecessary bits at *(v)*.

5.2.4. Optical Flow Loss

We deploy the setup shown in Figure 5.5 to determine if including an explicit additional optical flow based loss term leads to improved motion compression. The optical flow between two video frames is defined in Section 2.1.2 as a 2D vector field that relates the movement of pixels from the one frame to the other [6]. We denote the dense (per-pixel) optical flow for each consecutive pair of frames in the input GOP as $\vec{\boldsymbol{V}}_g$: the ground truth flow, indicated at *(i)* in the figure. As shown at *(ii)*, $\vec{\boldsymbol{V}}_p$ represents the flow vectors derived from the frames predicted by our motion compression network. A host of techniques can be used to calculate $\vec{\boldsymbol{V}}_g$ and $\vec{\boldsymbol{V}}_p$, including differential [28], phase [110] and energy [21] based methods, or more recent deep learning approaches [30, 33, 111–113]. In this work, we use LiteFlowNet [30], a state-of-the-art deep flow estimation model. LiteFlowNet's weights are pre-trained on the MPI Sintel dataset [114] and frozen when training our video compression models. We experiment with the optical flow losses defined by the Euclidean distance between the ground truth and predicted flow vectors called the end-point-error (EPE),

$$L_{\text{EPE}} = \sqrt{||\vec{\boldsymbol{V}}_g - \vec{\boldsymbol{V}}_p||^2}, \tag{5.8}$$

and cosine similarity,

$$L_{\text{cosine}} = 1 - \frac{\vec{\boldsymbol{V}}_g \cdot \vec{\boldsymbol{V}}_p}{||\vec{\boldsymbol{V}}_g||\;||\vec{\boldsymbol{V}}_p||}. \tag{5.9}$$

L_{cosine} differs from L_{EPE} in that it only penalises directional deviations between the ground-truth and predicted flow vectors as disregarding differences in magnitude may provide beneficial regularisation. We normalise the x and y components of the flow vectors in $\vec{\boldsymbol{V}}_g$ and $\vec{\boldsymbol{V}}_p$ by the width and height of the input video frames to avoid directional biasing. We investigate the consequences of adding these optical flow loss terms in Section 5.4.3.

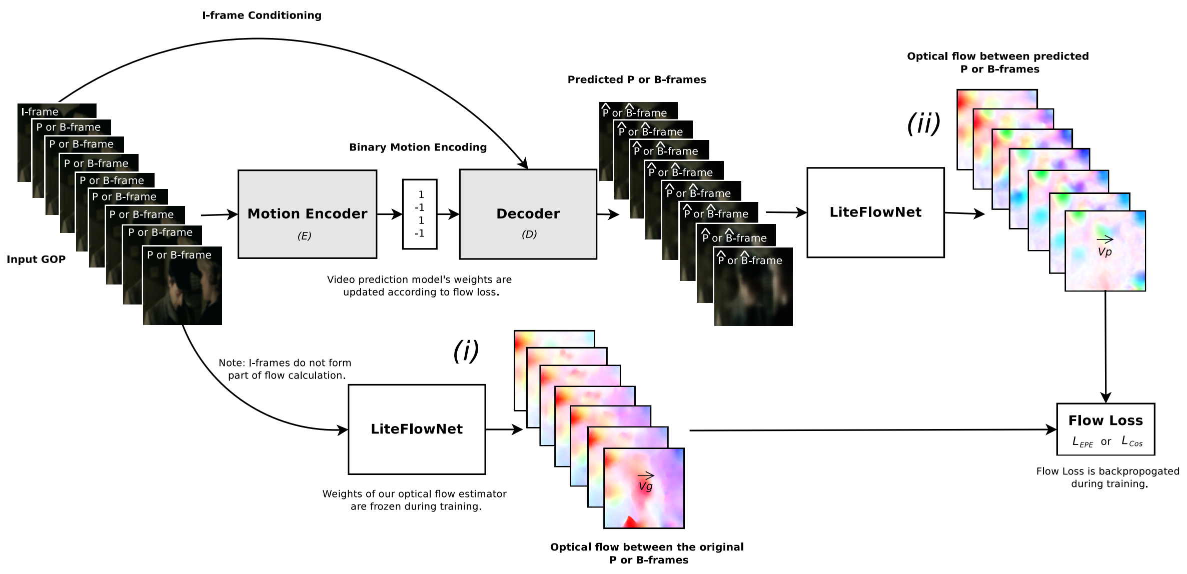
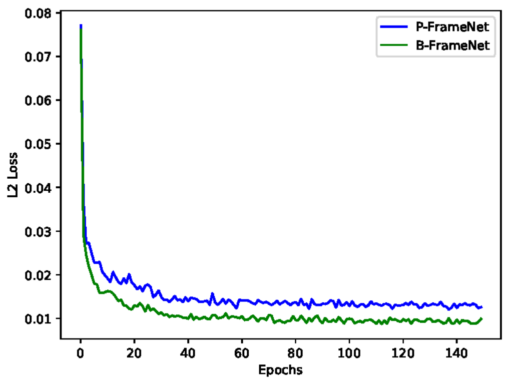

I-frame Conditioning
I-frame
P or B-frame
Input GOP
Binary Motion Encoding
Motion Encoder
(E)
1
-1
1
-1
Decoder
(D)
Video prediction model's weights are updated according to flow loss.
Predicted P or B-frames
P̂ or B̂-frame
Optical flow between predicted P or B-frames
(ii)
LiteFlowNet
Vp
Note: I-frames do not form part of flow calculation.
(i)
LiteFlowNet
Weights of our optical flow estimator are frozen during training.
Vg
Optical flow between the original P or B-frames
Flow Loss
L_EPE or L_Cos
Flow Loss is backpropogated during training.

Figure 5.5: Setup used to train a video frame prediction network with an optical flow based loss term. We use LiteFlowNet [30] to calculate and compare the optical flow of the input and predicted video frames. LiteFlowNet's weights are fixed when optimising our video frame prediction models.

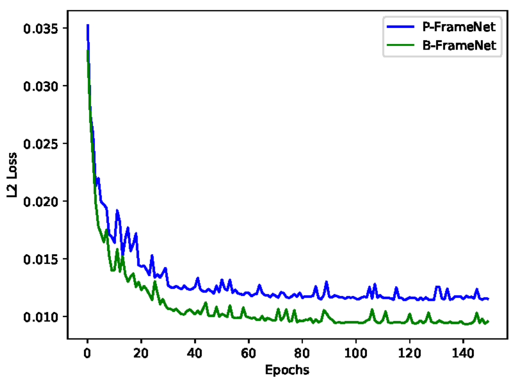
5.3. Experimental Setup

5.3.1. Data and Training Procedure

The P-frame and B-frame prediction networks in Figures 5.2 and 5.3 are trained on the Hollywood dataset [115]. This dataset contains 475 AVI movie clips from a wide range of classic films. The average clip length in our training corpus is around 5 seconds. Prior to training we transcode each clip with the H.264 [1] codec to ensure NVIDIA Video Loader (NVVL) data loader compatibility [77, 116]. We split this dataset into training and validation sets containing 435 and 40 clips, respectively. To avoid learning compression artefacts introduced by H.264, we train our models on resized 64×64 pixel video frames. During training we randomly crop a GOP from each clip, so although our dataset only contains 435 videos, our models are exposed to substantially more data. The GOPs used for validation are cropped from the start of each video in the validation set to ensure that the validation losses used for early stopping are directly comparable across epochs. GOP length is set to 18 for B-FrameNet and 17 for P-FrameNet. In both cases our models are trained to predict 16 video frames using the reconstruction loss in equation (5.3). Input video clips are grouped into batches of 3 and their pixel values are normalised to fall in the range $(-1, 1)$. The training process spans 150 epochs and utilises Adam optimisation [75]. The initial learning rate is set to 0.0001 and decayed by a factor of 2 at epochs 30, 100 and 140. We train models with a range of bottleneck-depths C_{bnd} to gauge performance across a range of bitrates. P-FrameNet and B-FrameNet's training and validation losses are plotted in Figure 5.6 for $C_{\text{bnd}} = 8$.



(a) Training Loss



(b) Validation Loss

Figure 5.6: P-FrameNet and B-FrameNet training and validation losses.

Bitrate Optimisation

Pre-trained P-FrameNet and B-FrameNet models are trained to perform 3D dynamic bit assignment (Section 5.2.3) by undergoing another 150 epochs of training with the optimisation function defined by

$$L_{RB} = L_R + \lambda L_B, \tag{5.10}$$

where L_{RB} combines the reconstruction loss L_R in equation (5.3) with the loss L_B in equation (5.6) that encourages low bitrates.

As done in [17], we introduce the hyperparameter λ to control the trade-off between reconstruction quality and compression rate. Setting $\lambda = 0$ zeros out L_B, which prevents a model from learning to vary its bitrate.

Flow Loss

When training our models to minimise the difference between the optical flow of the input and that of the predicted frames (Section 5.2.4) we use the loss defined by,

$$L_{RF} = L_R + \alpha L_F, \tag{5.11}$$

where L_F is one of the the optical flow losses in equations (5.8) or (5.9) and L_R is the distortion loss in equation (5.3). A weighting term α is used to normalise L_F by the total number of flow vectors during training.

5.3.2. Evaluation

To quantify the quality of our predicted video frames we use three objective evaluation metrics: PSNR, SSIM [38] and VMAF [39]. Higher scores signify greater prediction quality.

To understand and probe different aspects of our approach, the experiments in Section 5.4 are carried out on videos from our validation set: the 40 videos from the Hollywood dataset [115]. In Section 5.5 final versions of P-FrameNet and B-FrameNet are pitted against the block-motion algorithms used in standard video codecs. This evaluation is carried out on 235 raw (‘`.yuv`’) video clips sampled from the Video Trace Library (VTL) [117].[2] Each clip is partitioned into 17-frame or 18-frame sequences depending on whether we are predicting P-frames or B-frames, such that our models always predict 16-frames. VMAF, SSIM, PSNR and EPE scores are then calculated and averaged across the reconstructed video-frames. We denote EPE as EPE (FlowNet) or EPE (Farneback) depending on whether we calculate optical flow using LiteFlowNet [30] or Farneback’s polynomial method [28]; in contrast to the other metrics, lower EPE is better.

5.4. Results: Ablation Experiments and Analysis

In order to probe and better understand our approach through ablation studies and developmental experiments, we guide our analysis using the following questions.

5.4.1. P-frame vs. B-frame Decoder Conditioning?

We explore the benefits of conditioning our video decoder on learned features extracted from reference I-frames (Section 5.2.2). Table 5.1 compares a video autoencoder (P-FrameNet without I-frame conditioning) to P-FrameNet (single reference frame conditioning) and B-FrameNet (dual reference frame conditioning) in Figures 5.2 and 5.3, respectively. In Table 5.1 conditioning is shown to consistently improve the quality of the predicted frames as it allows the encoder to focus primarily on motion extraction—we learn to transform available pixel-content rather than compressing it directly.

Table 5.1 reaffirms that motion transforms are easier to compress than raw video content [6]. B-frame conditioning is shown to outperform its P-frame counterpart, as context from bounding reference frames allows it to learn both forward and reverse motion transformations. Unlike standard video codecs, B-FrameNet is able to predict B-frames in parallel without the extra overhead of having to transmit the order in which frames are to be decoded [2].

[2] Additionally, we provide YouTube links to videos compressed by P-FrameNet/B-FrameNet that have been taken from the wild.

Conditioning	PSNR	SSIM	VMAF
None	23.71	0.64	41.21
P-frame	28.25	0.80	62.31
B-frame	**29.83**	**0.84**	**70.45**

Table 5.1: Quality scores for various decoder conditioning schemes.

5.4.2. Do Multiscale Convolutions Learn More Representative Motion?

We experiment with incorporating the multi-scale convolutions [55] discussed in Section 5.2.2 in our motion encoder architecture. This provides us with a lightweight means of sampling motion at a variety of spatial and temporal scales. Table 5.2 compares two implementations of P-FrameNet, one with multi-scale convolutions and the other with normal convolutions in its motion encoder. Using multi-scale convolutions leads to modest but consistent improvements in the quality of the predicted frames, as indicated by higher PSNR, SSIM and VMAF scores. It also allows P-FrameNet to perform more representative motion encoding (lower EPE). Multi-scale convolutional layers are, therefore, used in our motion encoders throughout the rest of this work.

Convolution	PSNR	SSIM	VMAF	EPE	
				FlowNet	Farneback
Standard	28.25	0.80	62.31	0.477	$9.28 \cdot 10^{-7}$
Multiscale	**28.48**	**0.81**	**63.90**	**0.471**	$\mathbf{9.24 \cdot 10^{-7}}$

Table 5.2: Multis-scale vs. standard convolutional implementations of P-FrameNet.

5.4.3. Is an Optical Flow Based Loss Beneficial?

We experiment with the EPE and cosine similarity flow losses in equations (5.8) and (5.9) in Section 5.2.4 to discover if an optical flow based penalty helps B-FrameNet to learn improved motion. As stated in equation (5.11) in Section 5.3.1, we add our chosen flow loss, L_F, to the reconstruction loss, L_R, during training. The hyperparameter α in equation (5.11) is used to weight L_F such that the mean loss per flow vector is added to L_R.

Table 5.3 reveals that an additional optical flow loss term worsens the quality of B-FrameNet's reconstructions (lower PSNR, SSIM and VMAF scores). The added loss term does, however, cause the optical flow of the predicted frames to match that of the input more closely (lower EPE). At first glance, this result seems contradictory. How can learning better motion lead to a depreciation in quality? Realising that optical flow is essentially only 2D pixel shuffling, the results in Table 5.3 imply that B-FrameNet is able to learn more advanced motion transforms (rotation, warping, occlusion and colour shift) when it is not limited to translational motion by a flow loss term. Informal experiments showed that increasing the weight α of the flow loss term in equation (5.11) improved the EPE but further degraded the quality of the predicted video frames.

Flow Loss	PSNR	SSIM	VMAF	EPE	
				FlowNet	Farneback
None	**29.81**	**0.842**	**71.03**	0.477	$9.21 \cdot 10^{-7}$
EPE	29.59	0.836	70.19	**0.461**	$\mathbf{9.17 \cdot 10^{-7}}$
Cosine	24.04	0.652	59.44	0.514	$9.69 \cdot 10^{-7}$

Table 5.3: Influence of an additional EPE (equation (5.8)) or cosine (equation (5.9)) optical flow loss term on B-FrameNet's performance.

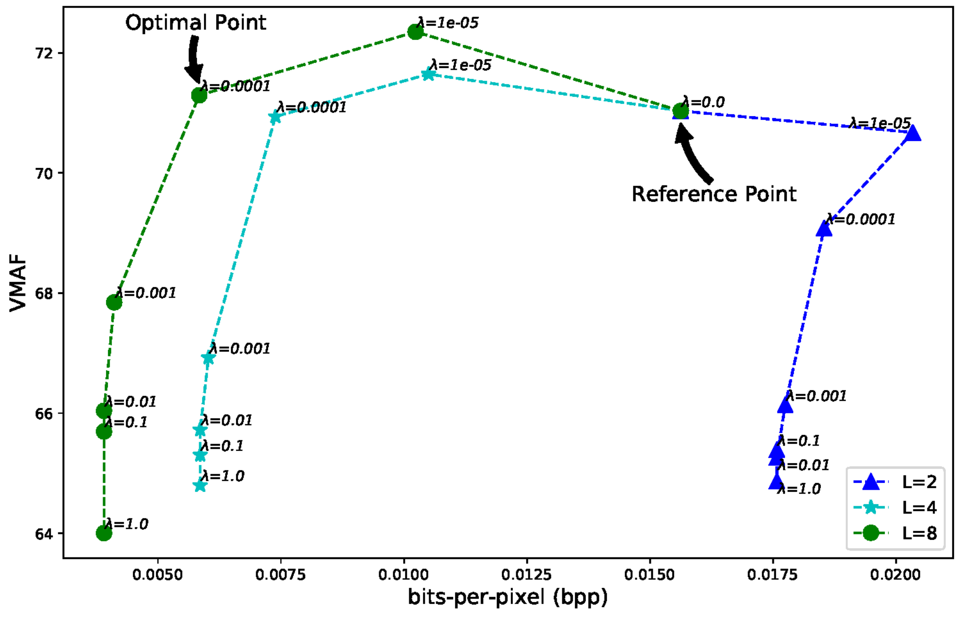
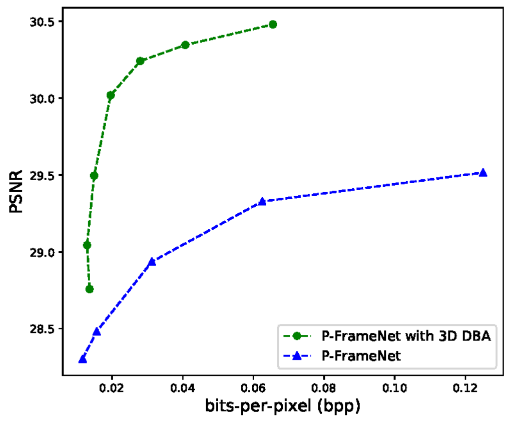


Figure 5.7: VMAF scores and learned bitrates produced on development data by B-FrameNet trained with different parameterisations of L_{RB}.

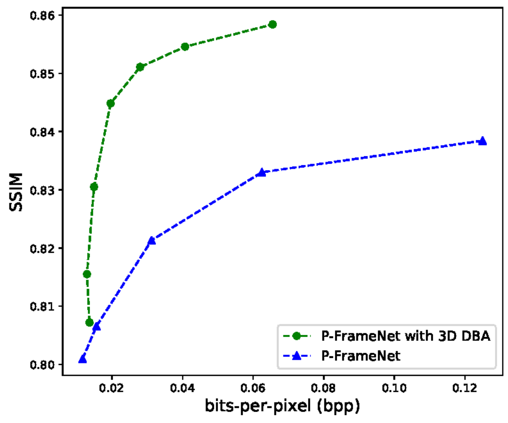
5.4.4. What are Optimal Parameters for Learning 3D Dynamic Bit Assignment?

With this experiment we try to find an optimal parameterisation of the 3D dynamic bit assignment loss L_{RB} in Section 5.3.1. We tune B-FrameNet ($C_{\text{bnd}} = 8$) with different quantisation levels L in equation (5.4) and different values of λ for the weight of the binary penalisation term L_B in the overall loss L_{RB} in equation (5.10). The learned bitrates and corresponding VMAF scores are shown in Figure 5.7.

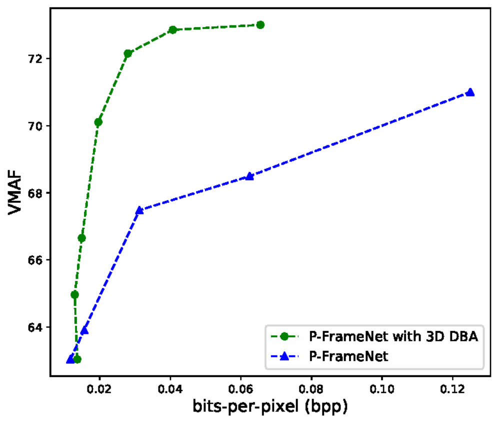
Using $\lambda = 0$ (no 3D dynamic bit assignment) as a point of reference, we find that setting L equal to its maximum possible value, in this case the channel depth 8, yields the best compression efficiency. L controls the number of bit levels the model can choose from per point in space-time. Higher settings of L gives B-FrameNet more options in deciding how many bits to allocate to different video regions, bringing about a lower bitrate on average. Setting L too low, $L = 2$, worsens compression as the bit-savings are not substantial enough to outweigh the cost of sending the quantised bit distribution map at *(vi)* in Figure 5.4. We see that for $L = 8$ and $L = 4$, setting $\lambda = 0.0001$ is optimal, as it results in the greatest bit reduction without decreasing the quality of the predicted frames below that of the reference point, $\lambda = 0$. Based on these results we set $\lambda = 0.0001$ and L equal to our encoder's bottleneck-depth C_{bnd} whenever optimising a system for 3D dynamic bit assignment.



(a) PSNR



(b) SSIM

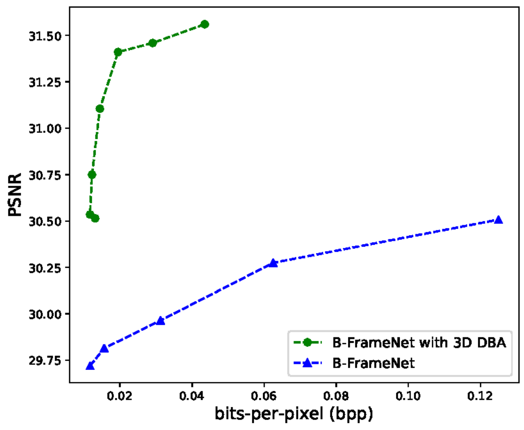


(c) VMAF

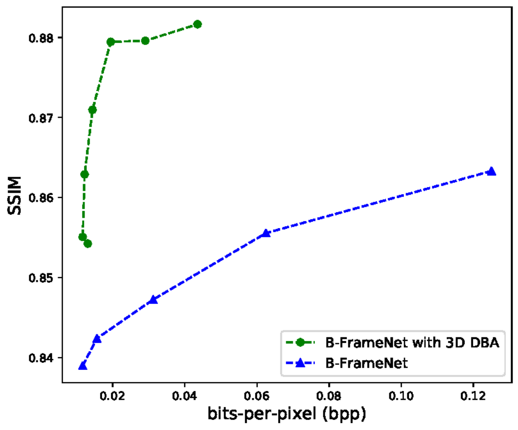
Figure 5.8: P-FrameNet with and without 3D dynamic bit assignment (3D DBA).

5.4.5. Does 3D Dynamic Bit Assignment Aid Motion Compression?

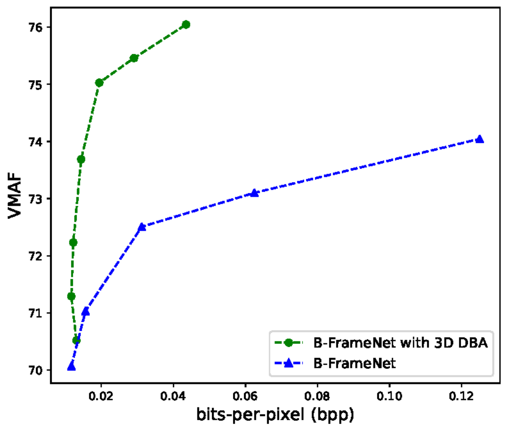
To establish whether 3D dynamic bit assignment aids compression efficiency, we train P-FrameNet and B-FrameNet to vary their bitrate across space-time (Sections 5.2.3 and 5.3.1). We train several instantiations of P-FrameNet and B-FrameNet with different bit channel depths, $C_{\text{bnd}} = \{6, 8, 16, 32, 64\}$, to assess the impact of learned dynamic bit assignment at different operating points. The compression curves in Figure 5.9 show that optimising a model for 3D dynamic bit assignment substantially improves its PSNR, SSIM and VMAF scores across all bitrates, even with the additional overhead of transmitting the binarised importance map, $Q_L(\boldsymbol{B}_{\text{map}})$.



(a) PSNR



(b) SSIM



(c) VMAF

Figure 5.9: B-FrameNet with and without 3D dynamic bit assignment (3D DBA).

5.4.6. Do Spatial Bit Allocations Change Over Time?

Figure 5.10 plots B-FrameNet's bit-distribution map $\boldsymbol{B}_{\text{map}}$ for the given 26-frame input GOP. Because B-FrameNet compresses both space and time by a factor of around 8, $\boldsymbol{B}_{\text{map}}$ consists of three distinct bit distributions—one per 8 frame interval. Higher valued regions in $\boldsymbol{B}_{\text{map}}$ are brighter and correspond to areas encoded with more bits.

The optical flow charts in Figure 5.10 are plotted in the hue saturation value (HSV) colour space. As discussed in Section 2.1.3 the angular direction of the optical flow vectors is indicated by hue, so that vectors pointing in the same direction are coloured the same. Saturation indicates the magnitude of the vectors, so vectors with higher magnitudes (moving objects) are less transparent and are represented with more intense colours. Comparing $\boldsymbol{B}_{\text{map}}$ to the optical flow charts in Figure 5.10, we notice qualitatively that more bits are assigned to regions containing moving objects (brightly coloured regions in the optical flow chart).

$\boldsymbol{B}_{\text{map}}$'s spatial bit distribution also changes across time to compensate for object displacements caused by motion. As an aside, for video scenes containing rapid motion or multiple scene changes it may help to lessen B-FrameNet's compression factor across time as this would yield more frequent updates to $\boldsymbol{B}_{\text{map}}$.

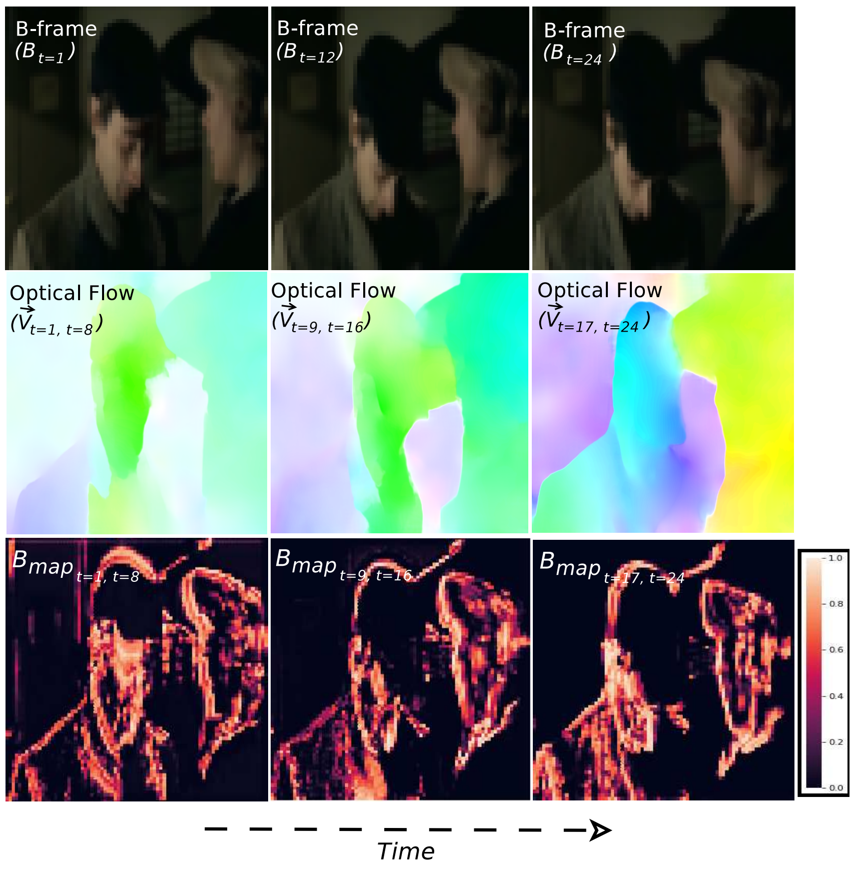


Figure 5.10: B-FrameNet's bit-distribution map, $\boldsymbol{B}_{\text{map}}$, compared to optical flow (FlowNet) and input video frames. Brighter regions in $\boldsymbol{B}_{\text{map}}$ are allocated higher bitrates and correspond to moving objects.

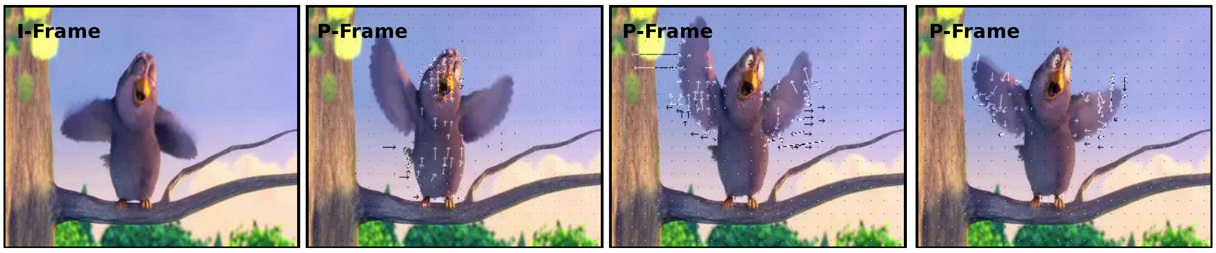


Figure 5.11: P-frames overlayed with the block-MVs that guided their prediction. MVs are estimated using the Diamond Search (DS) algorithm [91].

5.5. Results: Comparing to Conventional Video Compression

We next compare our learned video compression approach to standard codecs.

5.5.1. Deep Motion Estimation vs. Standard Block Motion Algorithms

Block-based motion estimation involves finding motion vectors (MVs) that model the movement of macroblocks between consecutive video frames. We compare P-FrameNet and B-FrameNet, $C_{\text{bnd}} = 8$, optimised for 3D dynamic bit assignment to several block-based motion estimation algorithms employed by standard video codecs, namely:

- Exhaustive Search (ES) [6]
- Three Step Search (TSS) [118]
- New Three Step Search (NTSS) [88]
- Simple and Efficient Search (SES) [90]
- Four Step Search (FSS) [89]
- Diamond Search (DS) [91]
- Adaptive Rood Pattern Search (ARPS) [92]

This evaluation is carried out on videos from the VTL dataset [117]. Our models are intended for video prediction only, so here we strip down the standard video codecs so that only the mechanisms used for inter-frame prediction are compared. As shown in Figure 5.11, we apply the standard block-motion algorithms to *IPPP* GOP sequences, so that each macroblock in the currently decoded P-frame is linked to the closest matching macroblock region from the preceding frame by way of a MV that indicates its relative spatial displacement.

For transmission, we binarise each MV's x and y components as well as the centre coordinates of the reference macroblock to which it points. Zero-vectors and overhead bits needed for reshaping are discounted; here we only consider bits that effect motion transformation. Searching all possible pixel locations in the reference frame for each predicted macroblock's closest match is computationally expensive, especially for high resolution videos. Hence, the search area is typically limited to $p = 7$ pixels around the predicted macroblock's location [86]. We experiment with mb $= 8 \times 8$ and mb $= 16 \times 16$ macroblock size parameterisations of the different algorithms in Tables 5.4 and 5.5, respectively. Smaller macroblocks produce denser MVs resulting in finer motion prediction at the cost of a higher bitrate and longer execution time. In this evaluation all models and algorithms are used to predict sixteen 224×320 video frames. Note that for this evaluation we assume uncompressed I-frame context is available at the decoder.

Model	bpp	PSNR	SSIM	VMAF	Time (sec)
ES	0.0108	16.35	0.850	44.64	11.35
TSS	0.0108	16.37	0.851	44.81	1.53
NTSS	0.0107	16.34	0.851	44.76	1.18
SES	0.0068	15.80	0.846	40.94	0.96
FSS	0.0077	16.29	0.852	44.77	1.01
DS	0.0100	15.70	0.816	37.90	0.77
ARPS	0.0097	15.66	0.816	37.89	0.63
P-FrameNet	0.0052	28.89	0.829	65.66	0.28
B-FrameNet	**0.0038**	**30.36**	**0.859**	**71.19**	**0.28**

Table 5.4: Motion compensation scores for 16 frame video prediction $(\text{mb} = 16 \times 16, p = 7)$.

Model	bpp	PSNR	SSIM	VMAF	Time (sec)
ES	0.0582	19.45	0.901	62.97	46.32
TSS	0.0578	19.47	0.901	63.15	5.67
NTSS	0.0568	19.44	**0.902**	63.17	4.29
SES	0.0396	18.80	0.893	57.18	3.71
FSS	0.0437	19.38	0.901	62.19	3.86
DS	0.0552	18.58	0.860	53.09	2.82
ARPS	0.0539	18.54	0.861	53.16	2.15
P-FrameNet	0.0052	28.89	0.829	65.66	0.28
B-FrameNet	**0.0038**	**30.36**	0.859	**71.19**	**0.28**

Table 5.5: Motion compensation scores for 16 frame video prediction $(\text{mb} = 8 \times 8, p = 7)$.

Tables 5.4 and 5.5 show that for fewer bits-per-pixel (bpp), P-FrameNet and B-FrameNet's predictions score higher in terms of PSNR and VMAF than those produced by the block-matching algorithms. SSIM scores are comparable, but our models use at least 23% and 88% fewer bits-per-pixel (bpp) than the block-matching algorithms in Tables 5.4 and 5.5, respectively. P-FrameNet and B-FrameNet's encoding and decoding time is faster than than that of all the block-based motion estimation algorithms. This speedup stems from their ability to predict frames in parallel without the need for a search-step during encoding.

5.5.2. Deep Motion Compression vs. Standard Video Codecs

In Section 5.5.1 we demonstrated that with fewer bits our learned binary motion codes are able to express richer motion than several block-based motion estimation algorithms. Standard video codecs improve MV compression via techniques not included in the standalone implementation of the algorithms used above. To reduce bitrate, similar MVs are grouped together and only the Motion Vector Difference (MVD) between each vector and a Motion Vector Predictor (MVP) is transmitted [119]. MVD values are normally lower than those of the original MVs, especially for a good choice in MVP,[3] and can be represented with fewer bits. Standard video codecs also actively adapt their block-motion algorithm's macroblock-size and search distance to better suit the content of different video regions.

For these reasons, we now compare P-FrameNet and B-FrameNet to the standard video codecs H.264 [1] and H.265 [2]. FFmpeg is used to compress videos with H.264/5. We varied each codec's constant rate factor (CRF); this aims to achieve a constant quality across all video frames using as few bits as possible. The full FFmpeg commands used are included in Appendix A and explained in greater detail in Section 2.4.3.

Both P-FrameNet and B-FrameNet are intended to provide inter-frame prediction of P/B-frames as part of a larger video compression system. To compress I-frames we adopt H.264's intra-frame codec, which is similar to that of H.265 [1, 2]. Any image codec can be used for I-frame compression, and more powerful deep image codecs do exist [13, 15, 18], but to keep this evaluation fair we include P-FrameNet and B-FrameNet into an existing video codec in order to investigate the effects of including learned motion prediction in isolation. Since all of the video codecs being evaluated share H.264's I-frame codec, any compressive gains stem mainly from improved inter-frame coding. We vary the quality (bit allocation) of the I-frames to gauge P-FrameNet and B-FrameNet's performance across different operating points. For this evaluation, we train $C_{\text{bnd}} = 8$ versions of P-FrameNet and B-FrameNet that have been optimised for 3D dynamic bit assignment. Bear in mind that unlike H.264 and H.265 our learned binary motion codes and overhead bits do not undergo any form of entropy coding.

P-FrameNet vs. Standard Video Codecs

We plot rate-distortion curves for P-FrameNet, H.264, and H.265, based on their respective compression of 17-frame 64×64 videos clips sampled from the VTL dataset [117]. Each clip consists of a single I-frame followed by sixteen referencing frames. We allow the standard codecs to decide on their own whether to assign referencing frames as P or B (or a mixture of

[3] In H.264, a MVP is selected as the mean of a MV group [1].

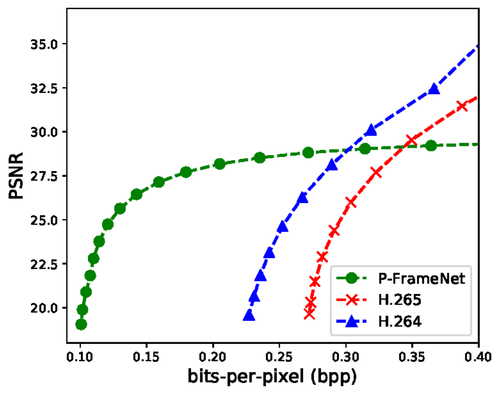
both), so that they can achieve their best possible compression. Quality scores are calculated on and averaged across the 17-frame video reconstructions.

Figure 5.12 reveals that P-FrameNet outperforms both H.264 and H.265 in terms of PSNR, SSIM and VMAF at low-bitrates. The standard video codecs fare better at higher bit-depths, but this is only because P-FrameNet and B-FrameNet do not, as of yet, compress residual information to improve the quality of their predictions. This research concentrated on improving motion estimation and compensation in video compression and this is shown by P-FrameNet performance gain at low-bitrates where video reconstruction is mostly the result of inter-frame prediction and not residual coding.

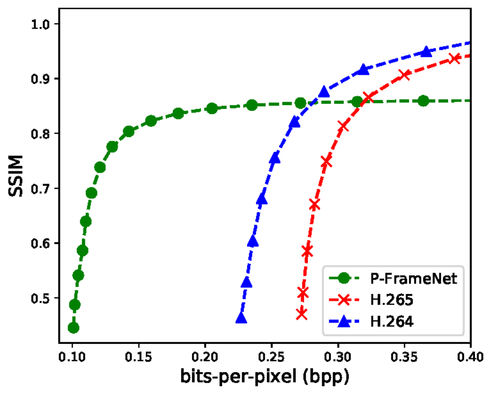
The link below gives a side-by-side example of P-FrameNet's compression compared to that of H.264 and H.265 at a low bitrate. The ground truth video is taken from the wild and split into 17-frame GOP groups. Each GOP is then encoded and decoded individually. The decoded GOPs are then concatenated to re-create the full-length clip.

https://youtu.be/LiHLVyIJg_I

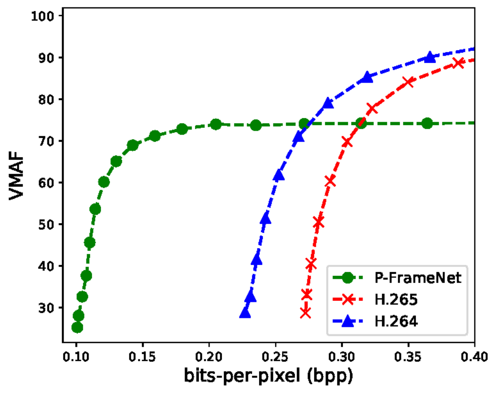
https://youtu.be/ICyL5m0L51U



(a) PSNR



(b) SSIM



(c) VMAF

Figure 5.12: P-FrameNet vs. standard video codecs rate-distortion curves.

B-FrameNet vs. Standard Video Codecs

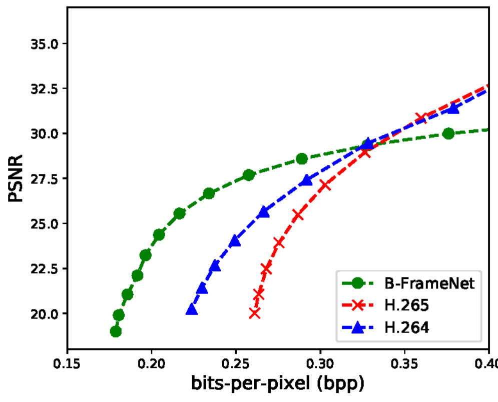
We compare B-FrameNet's compression efficiency to that of the standard H.264 and H.265 video codecs. Here we compress 18-frame 64×64 videos, each containing two bounding I-frames and sixteen P/B-frames.

Figure 5.13 compares the PSNR, SSIM and VMAF rate-distortion curves of the various codecs. B-FrameNet outperforms all of the standard video codecs at low-bitrates, which indicates that it is able to produce higher quality inter-frame predictions. This claim is further supported by Figure 5.15, which shows the difference in quality between H.264, H.265 and B-FrameNet's inter-frame predictions. We only show the middle five predicted video frames for each codec—those furthest from the I-frames and most reliant on motion compensation. It can be seen that B-FrameNet's predictions are qualitatively and quantitatively (PSNR, SSIM, VMAF) preferable.

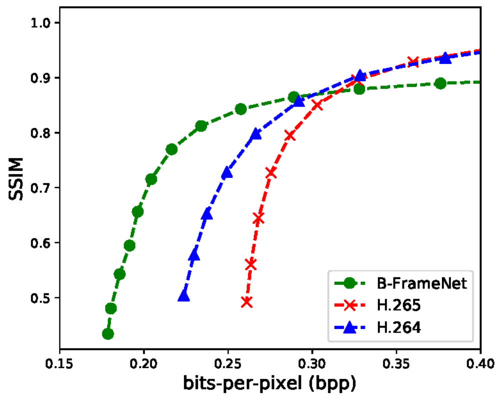
The link below gives a side-by-side example of B-FrameNet's compression compared to that of H.264 and H.265 at a low bitrate.

https://youtu.be/313hqyf0tBg

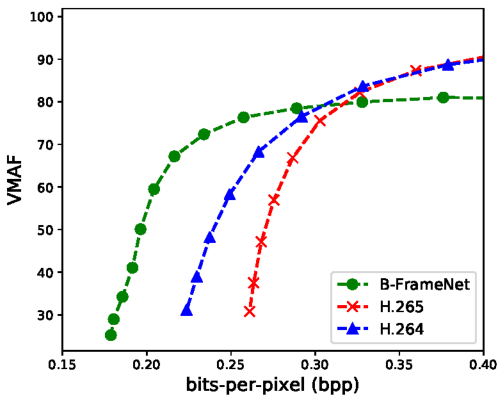
https://youtu.be/nV7mLPwOXTI



(a) PSNR



(b) SSIM



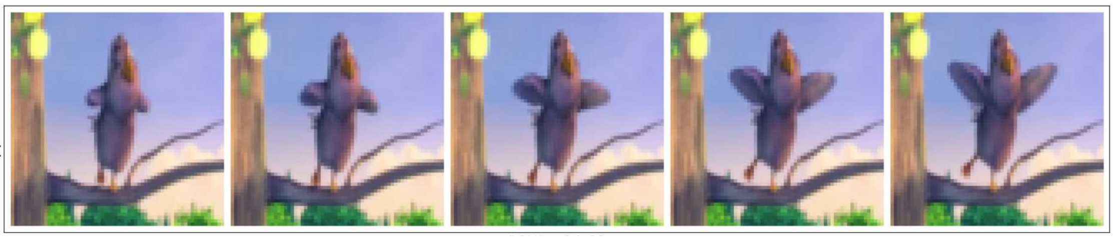
(c) VMAF

Figure 5.13: B-FrameNet vs. standard video codecs rate-distortion curves.

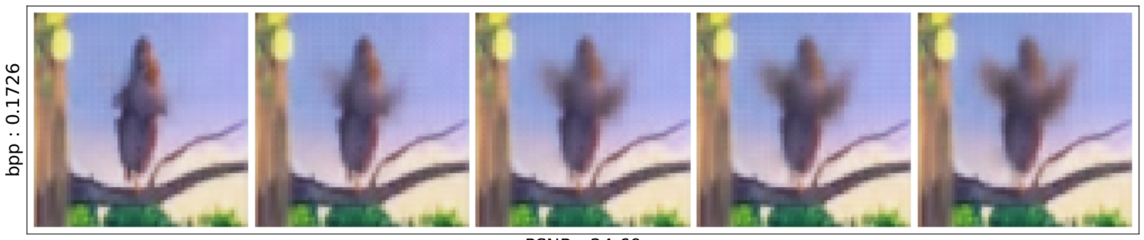
5.6. Chapter Summary

We introduced P-FrameNet and B-FrameNet, deep motion estimation and compensation networks that can replace block-motion algorithms in existing video codecs for improved inter-frame prediction. In contrast to previously developed video codecs, we do not transmit optical flow vectors to guide our video frame predictions. Instead, our encoder network learns to identify and compress the motion present in a video sequence directly. The ensuing binary motion code is used to direct P-FrameNet and B-FrameNet's decoder in transforming reference frame content. This allows for parallel motion compensation that predicts more complex motion than flow-based methods. Leveraging recent work in deep image compression, we also train P-FrameNet and B-FrameNet to perform 3D dynamic bit assignment, i.e. vary their bit allocations through space-time. We show that this improves compression by focusing bits on complicated video regions. Experiments show that at a lower bitrate, both P-FrameNet and B-FrameNet's inter-frame predictions are of a higher quality than those of the standard video codecs, H.264 and H.265.

Apart from porting our inter-frame prediction networks into existing deep video codecs, future work will explore replacing flow-based motion estimation in alternative video applications (e.g. slow-motion) with conditioning on our learned binary motion encodings.

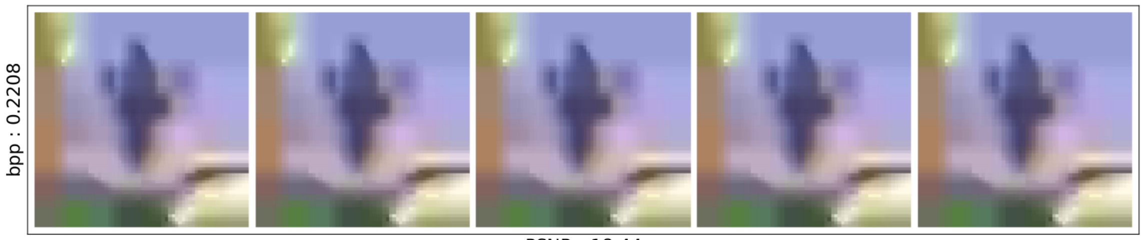


(a) Ground Truth



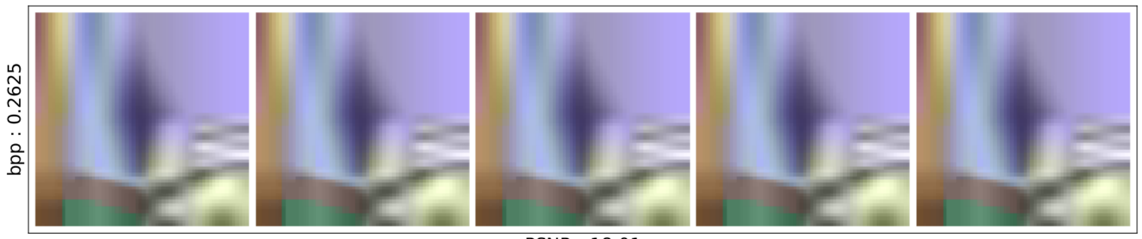
PSNR : 24.69
SSIM : 0.807
VMAF : 52.83

(b) P-FrameNet



PSNR : 18.44
SSIM : 0.494
VMAF : 25.87

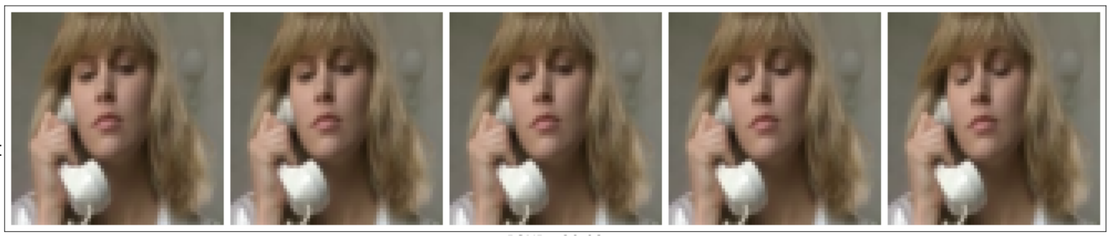
(c) H.264



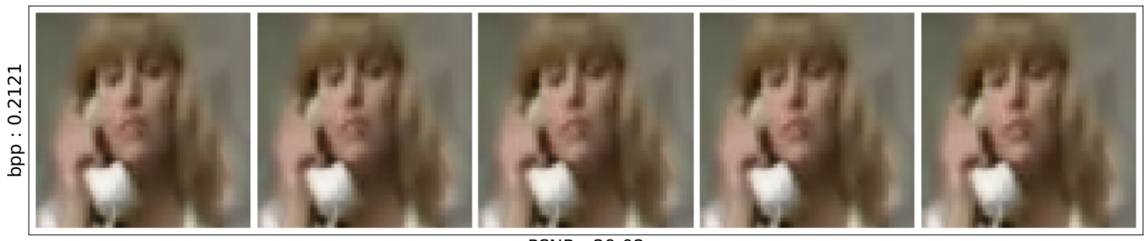
PSNR : 18.01
SSIM : 0.482
VMAF : 19.26

(d) H.265

Figure 5.14: P-FrameNet vs. standard video codecs inter-frame predictions. We only show the last five predicted frames furthest from the I-Frame.

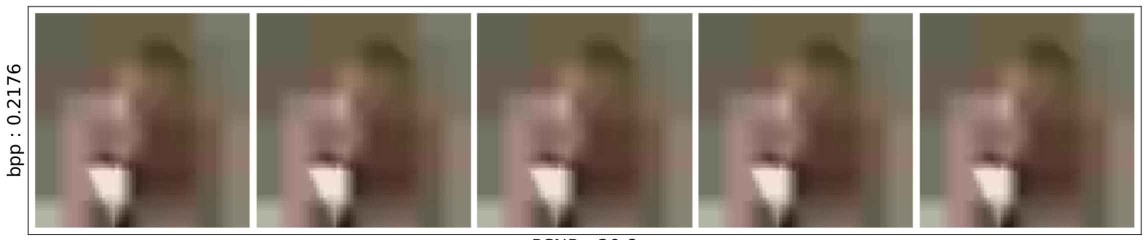


(a) Ground Truth



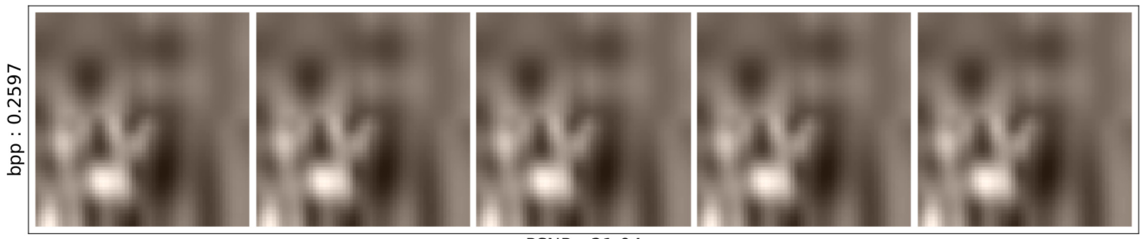
PSNR : 29.02
SSIM : 0.875
VMAF : 77.26

(b) B-FrameNet



PSNR : 20.8
SSIM : 0.465
VMAF : 24.69

(c) H.264



PSNR : 21.04
SSIM : 0.551
VMAF : 27.54

(d) H.265

Figure 5.15: B-FrameNet vs. standard video codecs inter-frame predictions. We only show the five predicted frames midway between the bounding I-Frames.

Chapter 6

Summary and Conclusion

Our main contribution is the Binary Inpainting Network (BINet), a framework that allows deep neural networks to predict video frame content directly from learned binary codes.

When compressing a single video frame BINet allows for parallelised inpainting of its patches from a full-context region without access to the original pixel data. Experiments in Chapter 4 showed that BINet's intra-frame predictions are of a higher quality than the sequential inpainting performed by the existing intra-frame codecs: WebP [19] and the deep inpainting in [20]. In quantitative evaluations BINet produces small but consistent compressive gains when integrated into a progressive image codec without inpainting. Qualitatively BINet's reconstructions suppress block-artefacts at low bitrates making it better suited towards memory efficient patch-based compression.

Inspired by BINet we developed P-FrameNet and B-FrameNet: inter-frame video prediction models that transform reference frame content according to a learned binary motion code that moves objects through time to compensate for relative motion in a video sequence. In Chapter 5 we demonstrate that ground truth motion is better modelled by P-FrameNet and B-FrameNet than the optical flow techniques prevalent in current video codecs [1, 2, 4]. Moreover, at a lower bitrate our parallel video frame predictions surpass the sequential reconstructions produced by the popular video codecs H.264 and H.265.

This research warrants the future inclusion of binary inpainting for improved parallel prediction in modern video codecs.

Recommendations for Future Work

BINet only includes binary inpainting at the first iteration of its progressive image coding process (see Chapter 4) . Recall that the inpainting stage predicts a centre 32×32 patch from a nine patch 96×96 pixel input. Inpainting at all sixteen iterations would require a 1056×1056 input size, which does not fit onto a single GPU's memory. We tried learning multi-stage inpainting by reflection or zero padding the encoded bits during training, but informal experiments showed that this hack turned out negligible improvements. Given the necessary GPU memory, we suggest training BINet with larger inputs for multi-stage inpainting.

We encourage replacing the optical flow based motion prediction in existing deep video codecs [4, 5] with our inter-frame prediction models (P-FrameNet and B-FrameNet), which are shown capable of learning higher quality motion transforms in Chapter 5. Apart from training P-FrameNet and B-FrameNet on HD video, learning to shuffle video frames so that similar video scenes are clustered and encoded together, in a way that aids compression, is also a potential area of interest.

Bibliography

[1] "H.264 advanced video coding for generic audiovisual services," International Telecommunication Union, Standard, 2003.

[2] "H.265 high efficiency video coding," International Telecommunication Union, Standard, 2018.

[3] C.-Y. Wu, N. Singhal, and P. Krahenbuhl, "Video compression through image interpolation," *European Conference on Computer Vision (ECCV)*, 2018.

[4] O. Rippel, S. Nair, C. Lew, S. Branson, A. G. Anderson, and L. Bourdev, "Learned video compression," *arXiv preprint arXiv:1811.06981*, 2018.

[5] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, "DVC: an end-to-end deep video compression framework," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] I. E. G. Richardson and Vcodex, *The H.264 advanced video compression standard*, 2nd ed. Wiley, 2010.

[7] 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," *International Conference On Learning Representations (ICLR)*, 2015.

[8] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-To-End Optimized Image Compression," *International Conference On Learning Representations (ICLR)*, 2016.

[9] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv preprint arXiv:1703.00395*, 2017.

[11] O. Rippel and L. Bourdev, "Real-time adaptive image compression," *arXiv preprint arXiv:1705.05823*, 2017.

[12] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. Van Gool, "Generative adversarial networks for extreme learned image compression," *arXiv preprint arXiv:1804.02958*, 2018.

[13] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. Jin Hwang, J. Shor, and G. Toderici, "Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[14] S. Santurkar, D. Budden, and N. Shavit, "Generative compression," *Picture Coding Symposium (PCS)*, 2018.

[15] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," *arXiv preprint arXiv:1802.01436*, 2018.

[16] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Conditional probability models for deep image compression," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[17] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang, "Learning convolutional networks for content-weighted image compression," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] S. C. Jooyoung Lee and S.-K. Beack, "Context-adaptive entropy model for end-to-end optimized image compression," *International Conference On Learning Representations (ICLR)*, 2019.

[19] Google Developers, "Compression Techniques — WebP — Google Developers," available at https://developers.google.com/speed/webp/docs/compression, 2016.

[20] M. H. Baig, V. Koltun, and L. Torresani, "Learning to inpaint for image compression," *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[21] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence (AI)*, 1981.

[22] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital video : an introduction to MPEG-2*. Chapman and Hall, 1997.

[23] B. P. Lathi and Z. Ding, *Modern digital and analog communication systems*, 4th ed. Oxford University Press, 2018.

[24] Y. Wu, M. Rosca, and T. Lillicrap, "Deep Compressed Sensing," *International Conference on Machine Learning (ICML)*, 2019.

[25] G. K. Wallace, "The JPEG still picture compession standard," *Communications of the Association for Computing Machinery (ACM)*, 1991.

[26] M. Sharma, "Compression using Huffman coding," *International Journal of Computer Science and Network Security (IJCSNS)*, 2010.

[27] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," *International Joint Conference on Artificial Intelligence (IJCAI)*, 1981.

[28] G. Färneback, "Two-frame motion estimation based on polynomial expansion," *Scandinavian Conference on Image Analysis (SCIA)*, 2003.

[29] ——, "Polynomial expansion for orientation and motion estimation," Ph.D. dissertation, Department of Electrical Engineering Linköpings Universitet, Sweden, 2002.

[30] T.-W. Hui, X. Tang, and C. C. Loy, "LiteFlowNet: a lightweight convolutional neural network for optical flow estimation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[31] "Basic parameter values for the HDTV standard for the studio and for international programme exchange," International Telecommunication Union, Recommendation, 1994.

[32] "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios," International Telecommunication Union, Recommendation, 2011.

[33] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "FlowNet: learning optical flow with convolutional networks," *International Conference on Computer Vision (ICCV)*, 2015.

[34] G. Valensi, "Improvements relating to colour television," Patent 21 082/46, 1946.

[35] JPEG, "Overview of JPEG," available at https://jpeg.org/jpeg/index.html, 2014.

[36] "Methodology for the subjective assessment of the quality of television pictures," International Telecommunication Union, Standard, 2012.

[37] K. Egiazarian, J. Astola, N. Ponomarenk, V. Lukin, F. Battisti, and M. Carli, "A new full-reference quality metrics based on hvs," *International Workshop on Video Processing and Quality Metrics (VPQM)*, 2006.

[38] Z. Wang, A. C. Bovik, H. Rahim Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing (TIP)*, 2004.

[39] Netflix, "VMAF: the journey continues," available at https://medium.com/netflix-techblog/vmaf-the-journey-continues-44b51ee9ed12, 2018.

[40] ——, "VMAF - Video Multi-Method Assessment Fusion," available at https://github.com/Netflix/vmaf, 2018.

[41] O. Simeone, “A brief introduction to machine learning for engineers,” *arXiv preprint arXiv:1709.02840*, 2018.

[42] A. Marblestone, G. Wayne, and K. Kording, “Toward an integration of deep learning and neuroscience,” *Frontiers in Computational Neuroscience (FCN)*, 2016.

[43] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” *International Conference on Machine Learning (ICML)*, 2010.

[44] M. Sazli, “A brief review of feed-forward neural networks,” *Communications, Faculty of Science, University of Ankara*, 2006.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Conference on Neural Information Processing Systems (NIPS)*, 2017.

[46] R. Girshick, J. Donahue, T. Darrell, and J. i. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: unified, real-time object detection,” *Conference on Neural Information Processing Systems (NIPS)*, 2015.

[48] K. Fukushima, “Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics (BC)*, 1980.

[49] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition.” *IEEE*, 1998.

[50] F.-F. Li, J. Johnson, and S. Yeung, “CS231n: convolutional neural networks for visual recognition,” available at http://cs231n.stanford.edu/, 2019.

[51] Facebook, “PyTorch1.2.0 documentation,” available at https://pytorch.org/docs/stable/index.html/, 2019.

[52] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[53] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: the all convolutional net,” *International Conference On Learning Representations (ICLR) Workshop*, 2015.

[54] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *International Conference On Learning Representations (ICLR)*, 2016.

[55] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *International Conference On Learning Representations (ICLR)*, 2016.

[56] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision (IJCV)*, 2004.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[58] K. R. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *International Conference on Machine Learning (ICML) Workshop*, 2015.

[59] X.-J. Mao, C. Shen, and Y.-B. Yang, "Image restoration using convolutional auto-encoders with symmetric skip connections," *arXiv preprint arXiv:1606.08921*, 2016.

[60] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: convolutional networks for biomedical image segmentation," *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[61] M. Lin, Q. Chen, and S. Yan, "Network in network," *International Conference On Learning Representations (ICLR)*, 2014.

[62] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of Gated Recurrent Neural Networks on sequence modeling," *Conference on Neural Information Processing Systems (NIPS) Deep Learning and Representation Workshop*, 2014.

[63] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation (NC)*, 1997.

[64] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[65] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, 2006.

[66] Z. Wang, J. Chen, and S. C. Hoi, "Deep learning for image super-resolution: a survey," *arXiv preprint arXiv:1902.06068*, 2019.

[67] R. Olivier and C. Hanqiang, "Nearest neighbor value interpolation," *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2012.

[68] F. Yeganli, M. Nazzal, and H. Ozkaramanli, "Super-resolution using multiple structured dictionaries based on the gradient operator and bicubic interpolation," *Signal Processing and Communication Application Conference (SIU)*, 2016.

[69] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[70] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[71] 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," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[72] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, A. Tejani, J. Totz, C. Ledig, and Z. Wang, "Is the deconvolution layer the same as a convolutional layer," *arXiv preprint arXiv:1609.07009*, 2016.

[73] D. G. Zill and W. S. Wright, *Advanced Engineering Mathematics*, 5th ed. Jones & Bartlett Publishers, 2014.

[74] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *arXiv preprint arXiv:1606.04838*, 2018.

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

[76] M. Looks, M. Herreshoff, D. Hutchins, and P. Norvig, "Deep learning with dynamic computation graphs," *International Conference On Learning Representations (ICLR)*, 2017.

[77] NVIDIA, "NVIDIA open sources NVVL: library for machine learning training," available at https://hub.packtpub.com/nvidia-open-sources-nvvl-library-for-machine-learning-training/, 2018.

[78] FFmpeg, "”FFmpeg Encode H.264”," available at https://trac.ffmpeg.org/wiki/Encode/H.264, 2018.

[79] J. Jiang, "Image compression with neural networks - a survey," *Signal Processing: Image Communication*, 1999.

[80] T. Raiko, M. Berglund, G. Alain, and L. Dinh, "Techniques for learning binary stochastic feedforward neural networks," *arXiv preprint arXiv:1406.2989*, 2014.

[81] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research (JMLR)*, 2014.

[82] W. T. Freeman, G. Toderici, and M. Covell, "CLIC Compression Challenge," available at http://www.compression.cc/, 2018.

[83] R. Eloff, A. Nortje, B. van Niekerk, A. Govender, L. Nortje, A. Pretorius, E. van Biljon, E. van der Westhuizen, L. van Staden, and H. Kamper, "Unsupervised acoustic unit discovery for speech synthesis using discrete latent-variable neural networks," *International Speech Communication Association (Interspeech)*, 2019.

[84] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *International Conference on Machine Learning (ICML)*, 2016.

[85] Kodak, "Kodak lossless true color image suite (PhotoCD PCD0992)," available at http://r0k.us/graphics/kodak/, 1999.

[86] D. Le Gall, "MPEG: a video compression standard for multimedia applications," *Communications of the Association for Computing Machinery (ACM)*, vol. 34, 1991.

[87] "VP9 Bitstream & Decoding Process Specification," Google, Specification, 2016.

[88] R. Li, B. Zeng, and M. L. Lio, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 1994.

[89] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 1996.

[90] J. Lu and M. L. Lio, "A simple and efficient search algorithm for block-matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 1997.

[91] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing (TIP)*, 2000.

[92] Y. Nie and K.-K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *IEEE Transactions on Image Processing (TIP)*, 2002.

[93] B. De Brabandere, X. Jia, T. Tuytelaars, and L. Van Gool, "Dynamic filter networks," *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[94] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala, "Video frame synthesis using deep voxel flow," *International Conference on Computer Vision (ICCV)*, 2017.

[95] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive convolution," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[96] S. Niklaus and F. Liu, "Context-aware synthesis for video frame interpolation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[97] H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz, "Super SloMo: high quality estimation of multiple intermediate frames for video interpolation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[98] S. Meyer, A. Djelouah, B. McWilliams, A. Sorkine-Hornung, M. Gross, and C. Schroers, "PhaseNet for video frame interpolation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[99] W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang, "Depth-aware video frame interpolation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[100] Y.-L. Liu, Y.-T. Liao, Y.-Y. L. Lin, and Y.-Y. Chuang, "Deep video frame interpolation using cyclic frame generation," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2019.

[101] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[102] T. Xue, J. Wu, K. L. Bouman, and W. T. Freeman, "Visual dynamics: probabilistic future frame synthesis via cross convolutional networks," *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[103] C. Finn, I. Goodfellow, and S. Levine, "Unsupervised learning for physical interaction through video prediction," *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[104] Z. Chen, T. He, X. Jin, and F. Wu, "Learning for video compression," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 2018.

[105] A. Habibian, T. van Rozendaal, J. M. Tomczak, and T. S. Cohen, "Video compression with rate-distortion autoencoders," *International Conference on Computer Vision (ICCV)*, 2019.

[106] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learning image and video compression through spatial-temporal energy compaction," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[107] Y.-H. Ho, C.-Y. Cho, W.-H. Peng, and G.-L. Jin, "SME-Net: Sparse Motion Estimation for parametric video prediction through reinforcement learning," *International Conference on Computer Vision (ICCV)*, 2019.

[108] D. Minnen, J. Ballé, and G. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," *Conference on Neural Information Processing Systems (NIPS)*, 2018.

[109] J. Han, S. Lombardo, C. Schroers, and S. Mandt, "Deep probabilistic video compression," *arXiv preprint arXiv:1810.02845*, 2018.

[110] T. Gautama and M. M. Van Hulle, "A phase-based approach to the estimation of the optical flow field using spatial filtering," *IEEE Transactions on Neural Networks (TNN)*, 2002.

[111] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: evolution of optical flow estimation with deep networks," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[112] S. Zweig and L. Wolf, "InterpoNet, A brain inspired neural network for optical flow dense interpolation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[113] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network." *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[114] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," *European Conference on Computer Vision (ECCV)*, 2012.

[115] M. "Marszałek, I. Laptev, and C. Schmid, "Actions in context," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[116] NVIDIA, "NVVL," available at https://github.com/NVIDIA/nvvl, 2018.

[117] A. S. University, "Video Trace Library YUV video sequences," available at http://trace.kom.aau.dk/yuv/index.html, 2000.

[118] S. D. Kamble, N. V. Thakur, and P. R. Bajaj, “Modified three-step search block matching motion estimation and weighted finite automata based fractal video compression,” *International Journal of Interactive Multimedia and Artificial Intelligence (IJIMAI)*, 2017.

[119] W. Yang, O. C. Au, C. Pang, J. Dai, and F. Zou, “An efficient motion vector coding algorithm based on adaptive motion vector prediction,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010.

Appendix A

FFmpeg Compression Commands

```bash
#!/bin/bash

ffmpeg -y \\
      -i "input.mp4" \\
      -codec:v "libx264" \\
      -g "17" \\
      -keyint_min "17" \\
      -crf "51" \\
      -frames:v "18" \\
      -an \\
      -f "mp4" \\
      "output.mp4"
```

Listing A.1: FFmpeg bash command to compress a video file with the H.264 video codec. Output video file has two bounding intra-coded I-Frames with 16 interpolated P/B-Frames in-between.

```bash
#!/bin/bash

ffmpeg -y \\
      -i "input.mp4" \\
      -codec:v "libx265" \\
      "-x265-params" "'keyint=17:min-keyint=17'" \\
      -crf "51" \\
      -frames:v "17" \\
      -an \\
      -f "mp4" \\
      "output.mp4"
```

Listing A.2: FFmpeg bash command to compress a video file with the H.265 video codec. Output video file has one intra-coded I-Frame followed by 16 extrapolated P/B-Frames.

Appendix B

FFmpeg Quality Metric Commands

```bash
#!/bin/bash

ffmpeg -i "ref.mp4" \\
      -i "comp.mp4" \\
      -lavfi "libvmaf" \\
      -f "null" \\
      "-"
```

Listing B.1: FFmpeg bash command to assess the VMAF quality between a reference and compressed video file.

```bash
#!/bin/bash

ffmpeg -i "ref.mp4" \\
      -i "comp.mp4" \\
      -lavfi "[0:v][1:v]psnr" \\
      -f "null" \\
      "-"
```

Listing B.2: FFmpeg bash command to assess the PSNR quality between a reference and compressed video file.

```bash
#!/bin/bash

ffmpeg -i "ref.mp4" \\
      -i "comp.mp4" \\
      -lavfi "[0:v][1:v]ssim" \\
      -f "null" \\
      "-"
```

Listing B.3: FFmpeg bash command to assess the SSIM quality between a reference and compressed video file.