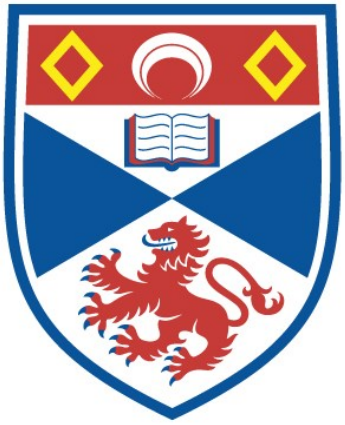
USING MACHINE LEARNING TO SELECT AND OPTIMISE MULTIPLE OBJECTIVES IN MEDIA COMPRESSION

Oleksandr Murashko

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



2018

Using Machine Learning to Select and Optimise Multiple Objectives in Media Compression

Oleksandr Murashko

University of
St Andrews

This thesis is submitted in partial fulfilment for the degree of PhD at the University of St Andrews

April 25, 2018

Abstract

The growing complexity of emerging image and video compression standards means additional demands on computational time and energy resources in a variety of environments. Additionally, the steady increase in sensor resolution, display resolution, and the demand for increasingly high-quality media in consumer and professional applications also mean that there is an increasing quantity of media being compressed.

This work focuses on a methodology for improving and understanding the quality of media compression algorithms using an empirical approach. Consequently, the outcomes of this research can be deployed on existing standard compression algorithms, but are also likely to be applicable to future standards without substantial redevelopment, increasing productivity and decreasing time-to-market.

Using machine learning techniques, this thesis proposes a means of using past information about how images and videos are compressed in terms of content, and leveraging this information to guide and improve industry standard media compressors in order to achieve the desired outcome in a time and energy efficient way.

The methodology is implemented and evaluated on JPEG, WebP and x265 codecs, allowing the system to automatically target multiple performance characteristics like file size, image quality, compression time and efficiency, based on user preferences. Compared to previous work, this system is able to achieve a prediction error three times smaller for quality and size for JPEG, and a speed up of compression of four times for WebP, targeting the same objectives. For x265 video compression, the system allows multiple objectives to be considered simultaneously, allowing speedier encoding for similar levels of quality.

Declaration

Candidate's declarations:

I, Oleksandr Murashko, hereby certify that this thesis, which is approximately 47 000 words in length, has been written by me, and that it is the record of work carried out by me, or principally by myself in collaboration with others as acknowledged, and that it has not been submitted in any previous application for a higher degree.

I was admitted as a research student in October 2013 and as a candidate for the degree of PhD in October 2013; the higher study for which this is a record was carried out in the University of St Andrews between 2013 and 2017.

Date April 25, 2018 signature of candidate .....

Supervisor's declaration:

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate for the degree of PhD in the University of St Andrews and that the candidate is qualified to submit this thesis in application for that degree.

Date April 25, 2018 signature of supervisor .....

Permission for Publication

In submitting this thesis to the University of St Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and the abstract will be published, and that a copy of the work may be made and supplied to any bona fide library or research worker, that my thesis will be electronically accessible for personal or research use unless exempt by award of an embargo as requested below, and that the library has the right to migrate my thesis into new electronic forms as required to ensure continued access to the thesis. I have obtained any third-party copyright permissions that may be required in order to allow such access and migration, or have requested the appropriate embargo below.

The following is an agreed request by candidate and supervisor regarding the publication of this thesis:

- PRINTED COPY: No embargo on print copy;
- ELECTRONIC COPY: No embargo on electronic copy.

Date April 25, 2018 signature of candidate .....

signature of supervisor .....

Acknowledgements

First of all, I would like to thank my supervisor Dr John Thomson, who gave me an opportunity to experience and participate in the world of cutting edge science and research. John did an excellent job guiding me through the course of my study from the first day till submission over the last four years. I greatly appreciate his effort and exceptional personal contributions to the research project. In particular, I acknowledge his assistance in expressing complex technical concepts and ideas in simple yet understandable language.

I also would like to thank all my advisers from the School of Computer Science in the University of St Andrews: Dr Graham Kirby, Dr Alexander Konovalov, Dr Mark-Jan Nederhof, Dr Kasim Terzic, Dr Susmit Sarkar, as well as Dr Hugh Leather from the School of Informatics in the University of Edinburgh.

I also want to express my gratitude to all my previous supervisors, whom I worked with over the course of my academic career: Mr John Davies, Prof Volodymyr Kazymyr, Dr Olga Sherba and especially Dr Oleksandr Narovlyansky, who instilled my interest to math and research.

With great warmth I want to thank my mother, who had been monitoring my each and every step.

Acronyms

ACACIA	–	Advanced content-adaptive compressor of images
API	–	Application programming interface
AVX	–	Advanced vector extensions (set of CPU instructions)
BPG	–	Better portable graphics (name of a relatively new image format)
CNN	–	Convolutional neural network
CRF	–	Constant rate factor (parameter in x264 and x265 video codecs)
DCT	–	Discrete cosine transform
DWT	–	Discrete wavelet transform
GA	–	Genetic algorithm
HC	–	Hill climbing algorithm
JPEG	–	Joint Photographic Experts Group (also name of a popular image format)
JPEG 2000	–	Name of an image format that was designed as a JPEG alternative
LMSSIM	–	Logarithmised mean structure similarity index
LSTM	–	Long short-term memory
MLP	–	Multilayer perceptron (type of a feed-forward artificial neural network)
MB	–	Megabyte; 10^6 bytes
MP	–	Megapixel; 10^6 pixels
MSSIM	–	Mean SSIM; mean structure similarity index
PNG	–	Portable network graphics (name of a popular image format)
PSNR	–	Peak signal-to-noise ratio
PSNR-HVS	–	Peak signal-to-noise ratio based on the human visual system
RMSE	–	Root-mean-squared error
RMSRCoP	–	Root-mean-squared relative correction of prediction
SIMD	–	Single instruction, multiple data
SSIM	–	Structure similarity index
VP9	–	Name of a popular video format developed by Google
WebP	–	Name of an image format developed by Google
WHT	–	Walsh-Hadamard transform
YUV	–	A colour space consisting of one luminance (Y) and two chrominance (UV) components

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

Improving the efficacy of image and video compression algorithms has been an ongoing area of research for many decades. The challenges associated with image and video compression remain important today due to the steady increase in sensor resolution, display resolution, and the demand for increasingly high quality media in consumer and professional applications. However, introducing a new compression standard to meet these demands is difficult in practice, even with an improved quality to size ratio, because modern infrastructure has been optimised specifically for current standards. As a result, employing image and video codecs currently in use is often thought preferable to making radical changes.

This work focuses on a methodology for improving and understanding the quality of media compression algorithms using an empirical approach. Consequently, the outcomes of this research can be deployed on existing standard compression algorithms, but are also likely to be applicable to future standards.

Processing modern large photographs and high-quality videos requires increased storage capacities, network bandwidth and a considerable amount of time and energy. This means that the compression time associated with the use of traditional codecs with modern images and videos has become a major concern. These challenges are compounded when we consider not just the increased size, quality and resolution of modern images and videos, but the ever increasing volume and sheer bulk of image data in existence.

The smartphone revolution of the last decade that has led to their now near ubiquity in society and the development of video streaming services have substantially changed not only how multimedia content is produced but how it is consumed. According to Cisco, Internet video traffic makes up the majority of all IP traffic today and is expected to grow dramatically in the near future [1, figure 13]. Moreover, Cisco expect live video traffic to increase by 15 times in the next few years.

This rapidly growing global volume of multimedia data requires new strategies for image and video compression that balance quality and file size, minimise compression time and have the flexibility to allow optimisation for specific use cases. For example, live steaming video can suffer from inconsistent quality because of strict limits for compression time, as highly detailed scenes generally take longer to process. The offline compression can be demanding as well. If a hypothetical medium-sized multimedia company needs to compress thousands of images or videos per day, which occupy few terabytes of disk space, that company will experience some non-trivial requirements for server equipment and energy consumption.

However, there is an alternative to increasing the number of servers for data compression. Large quantities of images and videos can actually be considered as an advantage for applying methods of statistical optimisation such as machine learning. This means that it may not be possible to decrease compression time for every singe image, but at the scale of 1000 images the difference in time will be substantial in comparison with using just built-in compressor options.

In reality, optimisation may not be possible in some cases. For example, compressing images into smaller file sizes with the same quality using the same codec is contradictory, unless of course, the codec has specific options that allow such scenario at the cost of increased compression time. Overcoming limitations of a particular codec will require changes in the compression algorithm.

But in many practical situations, there are alternative compression strategies which can give better results in terms of speed. It is relevant for applications that require a specific outcome of compression such as a given level of quality or fixed file size; this thesis is focused on such use cases. This research proposes a new methodology for predicting the outcome of compression allowing various preliminary optimisations of encoding strategy in order to save compression time and even improve resulting quality or decrease size.

In view of the above, it is clear that compression time optimisations are important from a practical point of view, and it becomes necessary to explain why they are possible and indeed necessary. The main reason is unpredictability of modern codecs.

1.1 Problem Examples

The result of compression is determined by a particular codec, which is assumed to be a fixed choice, and the following factors:

– encoding parameters;

– input resolution (and length for videos);
– actual content of the image or video.

Upon compressing various images or videos with the same options, the results are different in terms of quality and size (figure 1.1). The problem is that there is no easy way to tell anything about the outcome of compression beforehand because it depends on the three above-listed factors, which influence result simultaneously in a complex manner. This creates an obstacle for use cases aiming for specific quality or file size as it is often unclear how to choose appropriate compression parameters. The simplest way to deal with this problem is to perform several recompressions with different parameters until the desired constraints are satisfied. Such approach could be very expensive in terms of compression time.

(a) Original images

(b) Compressed images

Figure 1.1: Two ordinary photographs resized to 600×400 pixels (a) and then compressed into WebP format with the same minimum possible quality (b). The road image had size 3 kB and quality 0.7; the sky image – 1 kB and 0.9 in the MSSIM quality metric (mean structure similarity index [2]).

Example on the figure 1.1 demonstrates how compression results can vary in the case of the same lowest quality option “-q 0”, same resolution, but different image

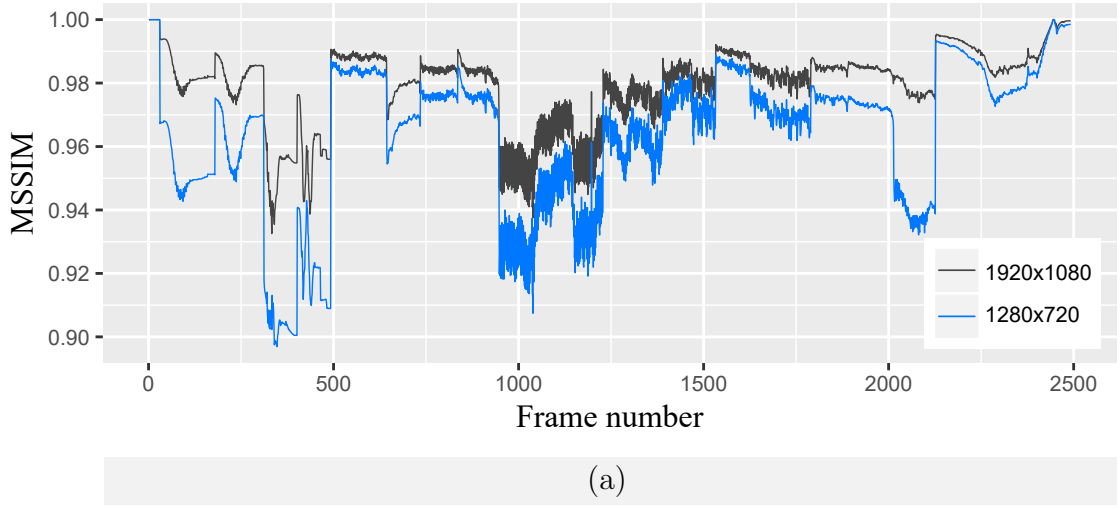
content. This example case uses WebP codec to show explicit quality degradation because WebP does not make an image unrecognisable at very low quality compression. Image codecs are not very good at understanding content and connection between its complexity and outcome of compression. Identical encoding parameters do not imply either same quality or same size.

The need for compression time optimisations arises because modern image and video codecs do not implement functionality for estimating compression result characteristics such as quality and size. There are a couple of exceptions like fixed bitrate video encoding that produces constrained size by definition and JPEG 2000 lossy compression, which aims for a given file size by design. However, in a majority of probable scenarios image/video codecs are still hard to predict. Although there are no theoretical obstacles to perform estimations of quality and size inside a codec based on the content of a particular image or video, it is pertinent to mention that estimating compression time does not exist in any modern encoder due to an objective reason – it may be possible only for a fixed hardware configuration.

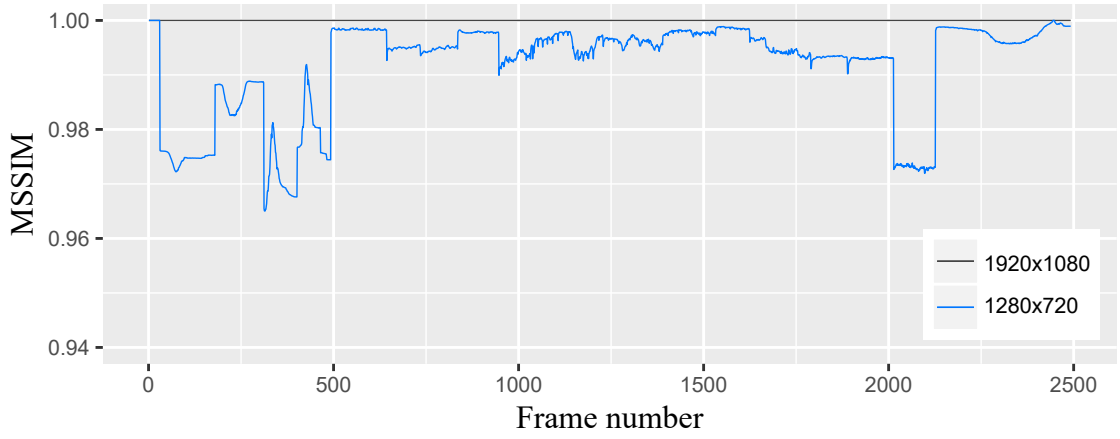
In the case of video compression choosing optimal parameters could be even more complicated because video content can vary significantly between the first and the last frames of a single video, which makes any set of options more suitable for some parts of the video and less appropriate for the others. Although many video compressors can perform quality balancing across different segments of the video, it is often not perfect and there are scenarios, where such balancing cannot work properly.

For example, downsampling an entire video into a different resolution is a widely used method of quality control in many popular video streaming services including YouTube and Twitch. Figure 1.2a shows distribution of quality in a particular video compressed in two resolutions: 1920×1080 and 1280×720 assuming that the display resolution is Full HD, so 720p version is upscaled to 1080p upon playback.

Analysing the frame quality graph at figure 1.2a it is clear to the naked eye that the quality level does not remain the same across the entire video in case of either resolution. Moreover, there are disproportional quality drops in different parts of the 720p video. The problem is that when the video codec tries to balance quality for a lower resolution video, it is not aware that this video has already been compressed through resampling. This can cause uneven quality drops. The resampling procedure can be considered as a compression operation that blurs frames with high detail. Figure 1.2b shows inconsistent quality degradation upon just resizing the video from 1080p to 720p without actual compression. So, video compression with resampling can be unpredictable and very challenging in terms of getting a desired result.



(a)



(b)

Figure 1.2: Frame quality in the mean SSIM metric for an ordinary 1080p video and its 720p version encoded with x265 (a) and without any compression (b) at the 1920×1080 display resolution.

Considering the described examples of different image compression outcomes and uneven video quality distribution it becomes clear that in order to obtain a desired result with predefined characteristics a single compression is insufficient. Instead, there is a need for multiple encoding runs with different parameters and a subsequent selection of the best result. So, following the argumentation about usefulness of the compression time optimisations, it is possible to conclude that there are various practical use cases where obtaining a specific result is not straightforward and can be improved.

1.2 General Idea

This research proposes a universal method that helps to deal with the problem of unpredictable compression. It is based on a simple idea – knowing the outcome of compression without actual compression allows a saving in total computational time. Considering the fact that conducting several recompressions will eventually lead to a desired solution, it is possible to notice that such a procedure is redundant in terms of the amount of produced information. Actual compressed files are discarded, only their characteristics like quality and size are recorded for comparison. This fact inspired the idea that the compression result can be estimated without performing any compression but using statistical models.

The main factor that brings uncertainty into the problem of estimating a compression outcome is how the content of various images and videos interacts with different codec parameters to produce various compression results. However, it is reasonable to assume that gathering enough statistics will help to identify some trends and regularities, which in turn can facilitate prediction of the outcome characteristics.

This concept should be applicable to a range of existing image and video compressors due to the fact that they operate by the same basic principle – eliminating redundancy in the low entropy data, which enforces correlations between the outcomes of different codecs.

Based on the possibility of improvements with the statistical approach, it is possible to put forward the *Predictable Compression Hypothesis*:

> Regardless of a particular codec the desired compression result can be obtained in a single encoding operation thus avoiding the need for several recompressions and extra processing time.

The main problem on a path towards obtaining a desired compression result is lack of functionality for estimating the compression outcome characteristics much faster than doing actual compression. One of the key ideas of this research is to build machine learning models that can predict such objectives as compressed file size, resulting quality and encoding time for images and videos.

The problem of predicting image compression is present in the literature mostly in the form of the JPEG transcoding – a concept of the controlled recompression of already encoded JPEG images into lower quality. An example of such strategy can be found in the work of Pigeon and Coulombe [3]. Their approach is conceptually simpler than the methodology presented in this research but it has important drawbacks – for example, its application is limited to a single image format. This thesis

investigates a general scenario without any format constraints and demonstrates superior accuracy in comparison with the transcoding method.

As for the video compression, the existing research does not address the problem of predicting compression results and barely considers optimisations, which are external to the codec. Despite some indication of the growing interest in this area, in particular by Netflix, no comparable research projects have been found. In this research we do not just build a predictor for the x265 video codec, but also propose a novel concept of the dynamic resolution video, which permits to surpass standard codec capabilities and decrease total compression time in some use cases. The main idea of the dynamic resolution is to use variable size for different segments of the video according to their complexity.

So, the main goals of this work are:

– to investigate the connection between image/video content and the outcome of compression with different parameters;

– to propose optimisation strategies for bypassing multiple recompressions.

The methodological basis of this research is an experiment-driven investigation, which uses the following methods: gathering statistical data, modelling through regression analysis, statistical comparison and heuristic-based decisions.

The scope of the research consists of investigating lossy compression for image and video formats, which are suitable for photographical (natural) scenes. The considered problems are related to processing only graphical information (pixel data) in the images and videos. The metadata and audio streams were not studied. This work does not propose new compression algorithms or any modifications to the existing ones. Instead it introduces methods for obtaining a desired compression result faster and in a predictable way using only real image and video codecs.

The investigation commences with the case of image compression and proceeds into scenarios for video compression. Such approach allows a gradual increase the problem complexity because images represent a fundamentally simpler data type in comparison with videos. The problem of predicting image compression was tested on a wide range of resolutions up to 24 megapixels.

An important aspect of this research is image and video quality because it plays a significant role in practice by affecting viewers' perception. Due to the lack of an established consensus among the objective quality metrics for images and videos, using real people to mark compressed graphics is often considered a reliable way to estimate the quality loss upon compression. However, this research operates with relatively large datasets consisting of thousands of images and videos, which makes it impractical to involve actual human viewers in the process. Therefore, only

objective quality metrics are used in this thesis with the main focus on the mean structure similarity index (MSSIM) by Wang et al. [2]. It is a popular and flexible quality metric independent from picture size and video length. Although it is not a perfect metric, this research considers any quality estimation algorithm as a "black box" proposing a universal methodology applicable to various quality metrics.

1.3 Contributions

The research contains the following contributions:

• *A concept of content features for representing entropy of an image or video.* The content features presented in this work were designed specifically with an intention to provide a computationally cheap metric for image and video complexity. They were engineered manually based on the factors influencing compression result as well as on the author's own experience in image compression. This research uses the most adequate set of features for the prediction problem in comparison with related work, where the choices were barely explained. An extra effort was made to tweak the features to be suitable for implementation with Intel AVX (advanced vector extensions) instructions.

• *A universal method for predicting outcome of compression.* The method is based on employing the machine learning models that use content features and values of compression parameters as input vectors for explicit predictions of quality, size or computational time for images and videos before compression.

• *The problem of predicting image compression was solved with reasonable accuracy.* The image compression with size and quality constraints was tested on a range of resolutions including large photographs and has shown a relatively small average error of 3% for the file size and 0.01 for quality in the MSSIM metric. Such accuracy is sufficient for applying the proposed methodology in various practical use cases. The necessity to analyse image content before compression increases total computational time for approximately 20–30% using the JPEG codec and 1–2% for WebP in comparison with a single default compression.

• *The problem of video compression with a given level of quality was solved with a sufficient accuracy for practical use.* Although the errors of the prediction models for video compression were relatively high, in particular the average error of the file size prediction model was about 30%, it was still possible to use such models for compressing long videos with a given level of quality.

• *A concept of the dynamic resolution video, interacting with compression parameters.* The dynamic resolution is an idea for optimising video compression without modifying the codec. Different parts of the video are allowed to be encoded

in various resolutions according to their content complexity and the desired quality. The video is then resampled to the display resolution upon playback. This approach results in a decreased compression time and sometimes even smaller file size, while the quality is maintained at approximately the same level. This thesis investigates the efficacy of dynamic resolution video and its interaction with other compression parameters.

1.4 Thesis Structure

The thesis includes a literature review (chapter 2) and two technical chapters dedicated to image and video compression respectively:

– *Predicting and optimising image compression* (chapter 3), which describes the methodology and demonstrates its universality by applying it to JPEG and WebP image formats;

– *Dynamic resolution and video compression with target quality* (chapter 4) introduces a concept of video compression with variable resolution across different segments using x265 codec.

The main results from the third chapter "Predicting and Optimizing Image Compression" were presented in the paper of the same title [4] at the ACM Multimedia conference in 2016.

This work ends with a conclusion and three appendices.

Chapter 2

Related Work

This chapter provides a summary of existing image and video compression standards, explains how classic compression algorithms require user-generated input parameters, like the quantizer factor, and gives a brief overview of the related research that attempts to deal with this problem.

2.1 Modern Image and Video Codecs

There are few image and video formats widely used in practice, in particular, on the Web. Existing implementations of media compression standards tend to demonstrate a similar efficiency in terms of quality-to-size ratio, and it is difficult to identify an universally best compressor.

2.1.1 Image compression standards

JPEG is one of the oldest image compression formats. The JPEG File Interchange Format (JFIF) specification was proposed in 1991. It is the most widely used still image format, which became the de-facto standard for photographic images and Internet graphics.

Today, JPEG is defined by the reference implementation from the Independent JPEG Group [5], although a number of other implementations are more popularly used, like *libjpeg-turbo* [6], which is optimised with CPU vector instructions.

JPEG has a relatively simple algorithm that consists of a discrete cosine transform (DCT), quantization and Huffman encoding [7, chapter 11]. The main compression parameter is quality factor in the range $[0; 100]$. The quality factor is the main determiner of the output image quality.

Google's **WebP** format [8] was proposed in 2010 as an alternative to JPEG and PNG (Portable Network Graphics) formats for Web applications. It relies mostly on a discrete cosine transform, similar to JPEG, but also uses prediction techniques and arithmetic compression.

WebP codec has more compression parameters than a typical JPEG encoder. However, the main parameter is still quality factor in the same range [0; 100]. There are a couple of options, which are of particular interest because they allow a user to compress images into given file size or quality in PSNR (peak signal-to-noise ratio) metric using multiple recompressions and a binary search. The availability of these options has allowed a more direct comparison between the method proposed in this thesis and the built-in functionality of the WebP codec.

JPEG 2000 was proposed as an alternative standard for replacing JPEG. It is based on the discrete wavelet transform (DWT) instead of DCT. It was originally designed to compress images into a given file size without multiple recompressions, which is related to the problem investigated in this research. Such functionality was implemented through progressive encoding from low- to high-frequency components of the discrete wavelet spectrum, so that the redundant information, which does not fit into specified file budget, is discarded. The details of the standard implementation can be found in [9].

The JPEG 2000 codec was not considered in this thesis as the format is not widely used today, and is generally not considered successful in comparison with JPEG. It has been criticised, in particular, for blurring the smooth colour areas in the image [10], which affects its applicability in practice.

2.1.2 Video compression standards

H.264 is a widespread video compression standard used by many streaming services and supported by all popular browsers. It was first proposed in 2003 and resulted in many software and hardware implementations. One of the well known and efficient implementations is the x264 video codec.

H.265 is a more complex and improved version of the H.264 standard oriented mainly at high resolution video [11]. Playing the videos encoded in this format is not yet extensively supported, but is becoming more popular. Among the open source software implementations, two should be noted: the highly efficient x265 codec that offers many compression parameters and the Kvazaar codec, which has recently been recognised as the best open source project at the ACM Multimedia conference [12]. Although Kvazaar is positioned as an H.265 implementation for academic purposes, it does not target high performance, which is crucial for the evaluation of the work

in this thesis. Currently, it has considerably fewer options and general functionality than x265. The latter is used as the reference implementation for research in this thesis due to its real-world use and emphasis on performance. Multiple comparisons demonstrate the efficiency of the H.265 standard and x265 video codec in particular [13, 14, 15].

VP9 video compression format was developed by Google as an alternative to H.265 (supposedly due to the licensing issues). This standard is used by the YouTube online video service along with H.264.

Thor video codec by Cisco is positioned as a free alternative to the licensed standards. It is defined with its reference implementation [16] and openly published description [17], which is relatively simple and easy to understand. For example, it explicitly shows the role of the discrete cosine transform in regulating the quality by compressing individual frames [17, fig. 1].

Daala is a relatively new codec [18], which is still under development sponsored by Mozilla. It is presented as a potentially more efficient alternative to the existing VP9 and H.265 formats based on more sophisticated techniques than a standard DCT. The codec is in the experimental stage, not properly optimised and does not use parallel processing.

2.2 DCT Quantizer as a Main Codec Parameter

The discrete cosine transform (DCT) is the most widely used method for audio-visual data compression. Many popular image and video codecs employ it in order to change the representation of the pixel data into a spectral form more suitable for subsequent entropy encoding.

The DCT is a reversible orthonormal discrete spectral transformation, which is usually implemented in practice as a set of 2-dimensional convolutions applied to the levels of brightness in the pixels of an image. One of the most important properties of the DCT is that removing the high-frequency components from its spectrum keeps the reconstructed signal close to the original one in terms of the mean squared error. This property was utilised in the process of spectrum quantization, which decreases the amplitude of the spectrum components in specific proportions defined by the value of quantizer. Changing the quantizer directly affects the amount of entropy in the spectrum and its subsequent compressibility with arithmetic encoding or similar algorithm. Large quantizer values, for example, smooth the signal and consequently blur the image. More information about DCT specifics and quantization process can be found in [7, chapter 11] or [19, chapter 3].

Due to the fact that all popular image and video codecs are based on spectral transformations, the quantizer plays a major role in regulating the level of compression and resulting quality of the image or video.

Most image and video codecs ask user to specify the value of quantizer or conceptually similar parameter before compression. The problem is that although such options can be considered and used as a measure of quality degradation, it does not correspond to the subjectively perceived level of quality or objective quality metrics like PSNR and SSIM. The situation with the compressed size is even worse – it cannot be estimated based on the quantizer value.

As image and video codecs become more complicated, other compression options are added in order to further tweak the compression process. This only complicated the problem of getting a desired outcome without trying multiple combinations. Modern video codecs operate with dozens of parameters, and it is very difficult for a user to understand the implications of each of these parameters on a compression process. This is why this research investigates the connection between compression parameters and the resulting characteristics using statistical methods.

2.3 Predicting Image Compression

Chandra and Ellis (1999)

The problem of predicting image compression outcome and optimal codec parameters has already been discussed in few papers. Various authors consider only the JPEG transcoding scenario – i.e. recompressing JPEG image into lower quality.

Probably the oldest paper that addresses the problem is the work of Chandra et al. in [20], which explicitly states that predicting compressed JPEG file size is impossible without taking into account the image content. The paper proposed an approach to classify a transcoding operation according to the potential improvement in file size. The idea of predicting explicit image characteristics was not developed.

The research of Chandra et al. is based on a simple fact that the value of JPEG quality factor, which is used to scale quantization tables, correlates with the compression ratio of the image. Therefore, having enough statistics it is possible to estimate the compression ratio for another quantizer value. A considerable attention was paid to the problem when the original JPEG image was compressed using different software with custom DCT quantization tables, and consequently its original compression ratio may not correspond to the quantizer used in the test codec.

Although Chandra et al. do not perform explicit file size or quality estimation,

they used some image characteristics as parameters for classification. They obtain a couple of values from manipulating DCT coefficients of the compressed image, for example, by calculating the percentage of high-amplitude DCT components. The idea of using coefficients of image spectral transformations as a measure of entropy has influenced design of the few image content features in this thesis.

Pigeon and Coulombe (2008 – 2014)

Another approach for transcoding JPEG images, which was extensively used by Steven Pigeon and Stéphane Coulombe in several papers [3, 21, 22, 23, 24, 25, 26], consists of constructing a classifier for estimating only a relative change in the compressed file size and quality in comparison with the input JPEG image. Such an approach is similar to the work of Chandra and Ellis [20] but does not use any features describing the image content.

For example, in [3] Pigeon et al. aim to predict the JPEG file size for transcoding operations prior to recompression. They consider two compression options: quality factor and scale. The main parameter controlling the aggressiveness of JPEG compression is the quality factor, which is used in the JPEG algorithm to scale quantization tables. In addition, an external parameter is introduced – the scaling factor for changing the output image resolution. Both quality factor and scale are taken into account when predicting transcoding outcome.

Pigeon et al. use a dataset of 70 300 images for gathering statistics. Firstly, they cluster original images with K-means algorithms considering original resolution (width and height) and original quality factor as independent variables. Then for each cluster they create a statistical table of 100 cells corresponding to all combinations of the quality factors (10, 20, ..., 100) and ten scales (10%, 20%, ..., 100%). Each cell represents an average relative change in the file size upon recompressing with particular quality factor and resolution.

A new image that needs to be recompressed is mapped to one of the clusters and subsequently to one of the cells in the corresponding table that represents a desired change in the compressed file size. This allows to identify optimal quality factor and target resolution for recompressing the image.

This approach has several areas with scope for improvement. Firstly, it requires original image to be already encoded in a specific format. Secondly, the classifier does not take into account the image content. The only features used are image resolution and quality factor, which can be instantly extracted from a JPEG image. One of the the advantages of this method is the lack of computational time overhead.

Although the JPEG transcoding method does not explicitly rely on the image content, it has such advantages as instant feature extraction and a possibility of a partial recompression. These benefits may not be applicable to other image formats like WebP. Moreover, Pigeon et al. did not consider cases if the original JPEG image was compressed using different options (e.g. Huffman code optimisation) or quantization tables (despite the fact it was previously done by Chandra et al. [20]), which is likely to decrease accuracy of the classifier.

In [21] Pigeon and Coulombe added the MSSIM quality metric as a second objective to be predicted for JPEG transcoding. Using the same approach as in the previous work [3] they show that it is possible to transcode images with a given quality relative to the original image. They also propose to apply multiple recompressions in cases where a single transcoding did not fit the target objective. However, they do not compare computational expenses between the multiple recompressions with and without prediction as an extra a priori information. In addition, calculating quality metric relative to an already compressed JPEG image is not suitable for scenarios where original image is not in the JPEG format – e.g. a raw data from the camera sensor or an edited photo.

In their last paper [26] Pigeon et al. adopted the problem to transcoding multiple images and positioned it as an optimisation for the Multimedia Messaging Service (MMS). Considering a set of images, the aim is to recompress them into the same level of quality. A reasonable objective function is proposed – the product of quality metric values for all images in the group, which encourages the uniform quality distribution for multiple images. It is based on the fact that a product value is maximum when its factors are identical. The paper is focused mostly on solving this optimisation problem by maximising the product with dynamic programming. Although the proposed method can be used in practice, the role of the MMS service in particular looks secondary and artificially attached to the topic.

The work of Pigeon and Coulombe can be summarised into two different use cases: recompressing (transcoding) one JPEG image with constraints and recompressing a group of images with constraints. They use up to three main target objectives: desired quality, compression ratio and resolution.

These papers discuss similar practical cases with methodologies which only differ slightly, or iteratively. Although the work by Pigeon et al. only considers compressed images and not uncompressed images, it is the most closely related to the problems considered in this thesis. For this reason their core methods and experimental techniques were repeated for comparison purposes.

Tichonov et al. (2017)

The recent work by Tichonov et al. [27] is dedicated to the problem of predicting image quality when compressing images into JPEG format. In the paper a classification approach is proposed, which is conceptually similar to the works of Pigeon and Coulombe. The major difference is in the fact that Tichonov et al. use a feature vector of several parameters, which are the colour depth and eight colour features – statistical characteristics of an image after applying an edge filter to the red, green and blue colour channels. Unfortunately, this choice has not been explained, but it can still be considered as the only explicitly formulated idea of the image content features in the research literature.

Due to the fact that this thesis also proposes a different set of content features specifically designed to predict compression outcome, it would be interesting to compare them with those proposed by Tichonov et al. However, it was impractical due to the fact that when Tichonov et al. published their paper after the work on image compression for this thesis had been completed and published in 2016 [4]. This circumstance made it inefficient to return to image compression experiments to replicate and test another method. The intention behind this was to save time and disk space in order to get more interesting results from video compression experiments, because the latter was assumed the more perspective research direction.

Nevertheless, it is possible to make some general conclusions. For example, considering how the features in [27] are calculated, it is clear that they were not designed to capture local spacial complexity of the image relief. Instead, the statistical calculations are applied to an image as a whole. Presumably, such an approach is not an efficient mean of representing the diversity of image content especially for large images. As a consequence, it should reduce the accuracy of a machine learning model for predicting quality.

Tichonov et al. did not experiment with predicting compressed file size. The reason given for this is that they considered file size more difficult to predict than quality. This question of relative difficulty is discussed in the thesis in Chapter 3.

2.4 Video Compression Optimisation

Choi et al. (2016)

Recently published research by Choi et al. [28] presents a method for estimating an appropriate target bitrate for each video frame that should result in the reduced blockiness in the compressed video. Choi et al. rely on the fact that different frames

in the video encoded with a fixed bitrate have variable amount of block artefacts. Due to the fact that increasing the bitrate for the whole video may be too expensive, they propose to deal specifically with the problematic frames.

For the experiments in [28] the reference implementation of the H.264 standard was chosen. Six standard short test videos from the xiph.org database were used as a dataset. The optimisation starts with recompressing a test video into multiple target bitrates and recording the amount of blocking artefacts for all frames in every scenario with a specially designed method.

Then using an optimisation procedure, based on the collected data, Choi et al. choose which bitrate should be targeted for each particular frame so that the amount of blockiness is minimal. Finally, they claim that the test video can be compressed with the optimal bitrate for each frame. The problem is that they do not explain how to perform this compression in practice. It appears to be impossible to achieve this through standard encoding procedure without modifying the codec. The reference codec manual [29] shows that the user can specify bitrate for the whole video, but not for individual frames.

Although the research of Choi et al. discusses some similar issues, the context in which the work is done is very different – it demonstrates what result can be obtained theoretically using their proposed balancing technique, but not in a way which can be put into practice in under current standards. Consequently, this research is not directly comparable with the methods presented in the current thesis, which are bound by the real world constraints of real formats.

Nevertheless, Choi et al. [28] proposed an interesting and original idea, which demonstrates an example of how multiple recompression can be useful for quality balancing inside a video. This thesis, conversely, investigates practical scenarios that help to avoid compressing a video more than once in order to save time and energy.

YouTube machine learning for video transcoding (2016)

Probably the most related investigation in comparison with the methods presented in this thesis was done by Covell et al. [30] – researchers from Google. They considered the problem of multiple recompressions to achieve a desired quality level for different video chunks upon transcoding videos uploaded to Youtube into the H.264 format. The need to compress videos several times dramatically increases energy consumption and computational time.

Covell et al. proposed to predict an optimal compression parameter value that leads to the desired quality level. The main idea was to extract general video features like resolution, frame rate, original bitrate, as well as hundreds of more specific ones like various motion compensation vectors. In order to get this information they perform a preliminary compression pass with a relatively low quality to get a reference compression ratio. Then they calculate several hundred features (depending on the approach modification) and use a neural network trained with the data from 9250 5-second video segments to predict the optimal CRF (constant rate factor) parameter values in the x264 video codec.

Covell et al. did not estimate the computational complexity or potential time savings upon using their method [30]. It would be an interesting question to investigate because using a video precompression stage requires a considerable amount of time as well as calculating lots of features from the compressed file. It is also unclear if the hundreds of features used, which depend on a particular video codec, might be transferable to other codecs, making this approach much less general. Due to the fact that the proposed system for predicting a single compression parameter is quite complicated, it should add a substantial time overhead, thus reducing the potential compression time benefits in comparison with multipass encoding.

Unfortunately, the original paper is relatively short and does not provide enough information to replicate the proposed method to a reasonable extent for comparison purposes. The general description of the method was also published in the YouTube developers blog [31] but it also does not contain technical details.

Netflix Dynamic Optimizer (2017)

The video streaming company *Netflix* recently announced the development of a tool called "Dynamic Optimizer" [32], which is expected to help with the problem of video bufferization when using slow connections. It should improve the compressed video quality or reduce bitrate in comparison with a typical scenario of the constant bitrate encoding.

Although no specific technical details have been reported except the fact that they use machine learning to balance the quality of all frames in the video, this topic is still related to the thesis.

Three years ago a post in the Netflix Technology Blog [33] demonstrated some results of an investigation into choosing optimal compression options according to the content of different videos. A year later another post on the same site [34] was dedicated to the problem of a reliable video quality metric. It presented the VMAF video quality measurement tool based on a method that combines several

objective quality metrics in order to approximate the mean subjective score of the frame quality degradation.

Although the news about Dynamic Optimizer look more like marketing claims to advertise the company services, it is possible to put forward an educated guess about how it might be implemented based on the preceding research published in the Netflix blog.

Firstly, it is reasonable to assume that one of the target objectives is the same level of quality for all frames in the video. The visual quality perception for each frame should be verified using VMAF tool considering the amount of work they invested into it. Another aim of the Dynamic Optimizer should be the quality to size ratio for each particular video. Considering the fact that compressed video is viewed many times, the encoding time factor is less important than minimising file size or improving the quality. Therefore, they can afford multiple recompressions of every frame to obtain a specific level of quality.

Consequently, it is likely that the developers modified a video codec by replacing the default heuristic for choosing quantizer at every frame with an exhaustive search for an optimal quantizer value that yields a chosen level of quality for each particular frame depending on its complexity. Despite the fact that there could be problems with taking into account the motion compensation part of the algorithm, such idea of per-frame quality balancing makes sense in terms of practically constant perceived quality across the entire video. This is speculation, but there is no scientific publication to refer to.

This thesis also considers the problem of targeting constant quality, although at a different level. Instead of dealing with individual frames, the optimisation is performed for video segments corresponding to separate scenes in a long video.

Ejembi and Bhatti (2014)

Video compression optimisations may be related not just to the obvious characteristics like resulting quality, file size and compression time, but also to the decoding complexity and its energy demands.

Ejembi and Bhatti [35] investigated the energy consumption when playing videos of different resolutions using software and hardware decoders. Their results indicate that with increasing resolution the raw energy requirements grow exponentially as well as the processor load and memory allocation. Moreover, decoding modern video formats like H.264, H265 and VP9 needs considerably more energy than older, less complex formats.

2.5 Objective Quality Metrics

The purpose of using an objective quality metric is to have a quick, deterministic and reliable way to measure quality degradation of the compressed image or video in comparison with the original.

Depending on the practical purpose of compressing an image or video, it can be targeted for a subsequent algorithmic processing or for the human vision. Consequently, the quality metrics used to estimate the amount of introduced noise may be different. In the case if the image/video is a subject to further machine analysis, the standard mean squared error is often considered sufficient. However, if the compressed material will be viewed by people, there is no solid consensus on which metric should be used as a replacement for subjective quality perception. Many quality metric investigations like [36] suggest that it is still an open question.

There are two kinds of objective quality metrics. One is calculated by comparing the compressed image with the original. Another – referenceless metrics – estimate the image quality without reference to any other source. Referenceless metrics are usually aimed at detecting explicit compression artefacts like blur or blockiness. For example, Tong et al. [37] propose to measure the amount of blur using spectrum coefficients of the discrete Haar wavelet transform, while Chen et al. [38] use conceptually similar method based on calculating the gradient at different image resolutions. To detect blockinness Gunawan et al. [39] propose to use a Sobel operator while taking into account the entropy of the image area. There seem to be no established standards among the referenceless metrics.

This research uses only metrics calculated with respect to original image or video (full-reference metrics). Probably the most popular quality estimation methods today are PSNR (peak signal-to-noise ratio) and SSIM (structure similarity index).

The **PSNR** metric is a simplest one based on the mean squared error. It uses a logarithmic scale in decibels to facilitate interpretation of the values. For two images (discrete signals) x and y the PSNR metric is calculated as:

$$PSNR = 10 \cdot \log_{10} \left(\frac{L^2}{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - y_i)^2} \right),$$

where N is a number of discrete samples (pixels in the image); $L = 255$ is a dynamic range of the brightness levels or maximum possible pixel value; x_i and y_i are values of the corresponding image pixels.

Although PSNR is still widely used today, it has been criticised for insufficient correlation with the human vision system. For example, Huynh-Thu et al. [40] empirically established that PSNR can reliably indicate quality levels only for a single image or video. In case of different images and videos, the same values of this metric do not correspond to the equal amount of the visible quality degradation.

The **SSIM** metric and, more importantly, its window version – MSSIM (mean structure similarity) were designed from scratch by Wang et al. [2] based on several basic assumptions about specifics and sensitivity of human vision. The MSSIM was proposed as a more sophisticated alternative to a widely used PSNR. The authors' aim was to improve the approximation of the human quality perception.

Between two images MSSIM is calculated as an average of the SSIM metric values in all positions of 11×11 sliding window. According to Wang et al., this approach is usually more reliable than simply calculating SSIM for the entire image because of higher attention to the details in each window.

The SSIM in 11×11 pixels window is calculated as follows:

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

$$\mu_x = \sum_{i=1}^{N} w_i x_i$$

$$\sigma_x = \sqrt{\sum_{i=1}^{N} w_i (x_i - y_i)^2}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y),$$

where $N = 121$ is the number of pixels in the window; $C_1 = (0.01 \cdot L)^2$; $C_2 = (0.03 \cdot L)^2$; $L = 255$ is a maximum luminance; x_i and y_i are corresponding pixel values; w_i represents a value from the 11×11 matrix of samples from the 2D Gaussian with $\sigma = 1.5$ in the range [–5; +5].

The range of theoretically possible values of the SSIM metric is [–1; +1], where 1.0 means identical images. However, in practice the metric values do not drop below zero.

Various comparisons between PSNR and SSIM metrics including the one made by Wang et al. [2] in the original paper indicate that SSIM better correlates with the subjective quality perception. For example, Hore et al. [41] compared relative sensitivity of these two metrics on images compressed into JPEG and JPEG 2000 formats. The metrics are closely correlated in case of blurred images, but SSIM is more sensitive to JPEG blockiness than PSNR. Kotevski et al. [42] compared

PSNR and SSIM metrics for compressed videos and experimentally established that SSIM is considerably more adequate than PSNR for measuring quality degradation in video sequences. Kotevski et al. also note that SSIM is not a perfect metric and has some issues too, mainly due to reduced sensitivity to changes in brightness and contrast.

Butteraugli is a new image quality metric proposed by Google. It is defined by its reference implementation [43] and unfortunately lacks a comprehensive explanation of its mechanisms and basic principles. According to the information available, the Butteraugli project aims to reach a sufficient approximation of the way the human visual system reacts to minor quality differences. It is a full-reference metric that uses information from all colour channels to calculate a difference value. The PSNR and SSIM metrics are typically calculated only for image luminance. The reference implementation of Butteraugli is relatively slow. It takes approximately 10 seconds for a Full HD frame in comparison with about 1 second for MSSIM metric and only few milliseconds for PSNR.

The most reliable method to estimate quality degradation in images and videos is to ask real people to rank them and calculate the **mean opinion score** (MOS). However, this approach is not feasible in many research projects, including this thesis. In the case of calculating quality of the compressed material as one of the final stages of an experiment, obtaining MOS is a demonstration of a solid result. However, if multiple quality measurements need to be conducted in real time to make certain decisions, using an objective quality metric is the only practical choice.

Despite the fact that new alternative quality metrics are introduced, only PSNR and SSIM/MSSIM remain widely used by multimedia research community. The problem seems to be in the balance between the metric complexity and its correlation with the human perception. When another more complex quality metric is introduced, it becomes necessary to understand how reliable it is in the different use cases. Even assuming that the new metric demonstrates better correlation with mean subjective opinion than SSIM, it may not be the case in the alternative scenario. For example, considering Butteraugli, its authors admit that it may not be a reliable measure of significant image distortions.

Consequently, the researchers often prefer slightly less efficient metrics with relatively simple and transparent implementations than more complicated ones with lower reliability for general use.

Almost all quality measurements for compressed images and videos in this thesis were done in the MSSIM metric. One of the advantages is that due to its window-based computation algorithm the metric is independent of image and frame

resolution as well as length of the video.

It is important to emphasise that MSSIM is not an ideal quality metric. However this thesis proposes methods which are independent from any particular metric by design. The logic behind this decision is that although the results may slightly differ upon using a different metric, the methodology remains the same.

2.6 Machine Learning in Image Compression

Although making compression algorithms or comparing their efficiency in terms of quality to size ratio is not the focus of this research, it is important to know how machine learning had been used for improving compression algorithms.

Hinton et al. (2006)

In recent years the idea of utilising the approximation capabilities of artificial neural networks for image compression has been gaining popularity. In particular, this relates to *autoencoders* – a type of multilayer network that replicates input signals on the output using a relatively small number of neurons in the middle of the network. The smaller transverse diameter in one of the central layers forces generalisation of the input data upon transferring it through the network.

Hinton and Salakhutdinov [44] pointed out that this property of autoencoders can be used for dimensionality reduction of the graphic and text data. Using autoencoder is conceptually similar to the principal component analysis (PCA) but in a non-linear space. Lossy image compression is a suitable use case for such methodology.

Hinton et al. only presented a concept in an explicit form but did not conduct an efficiency comparison with the existing image compression techniques. This makes the paper interesting conceptually, but does not provide an evaluation that would suggest it is better than existing implementations.

Toderici et al. (2016)

One of the recent interpretations of the way autoencoders can be used for image compression was presented by Toderici et al. [45] from a research group in Google. They proposed an efficient method to compress tiny images of 32×32 pixels (thumbnails) using a single autoencoder network. The idea is to use a deep convolutional/deconvolutional autoencoder with LSTM layers (LSTM or long short-term

memory unit is a type of a neural network component with recurrent connections that facilitates the vanishing gradient problem).

Toderici et al. use a single neural network, which works as both compressor and decompressor, so the network structure was not considered as a part of the compressed data. This means that such network had to be trained as a universal encoder for any image content. It was achieved by using a dataset of more than 200 million images. Considering a small image size the raw data occupied approximately 600 GB.

One of the key features of this work is that Toderici et al. also addressed an important problem related to autoencoders – a fixed compression ratio, which is usually impossible to adjust without retraining the model. They proposed a relatively simple idea of progressive encoding. Thus compression process consists of several iterations that reduce input pixel data to a fixed number of bits. The input for the first iteration is the original image, but each subsequent iteration compresses a difference (error) between original and decoded image. Using this approach allows to gradually reduce error over multiple encoding iterations whilst increasing the number of compressed bits.

The proposed compression method [45] was compared with the existing image formats like JPEG, JPEG 2000 and WebP. The results demonstrated a noticeable superiority in image quality over all tested codecs at various compression levels.

Although the work by Toderici et al. proposes a relatively new approach to the problem of image compression with autoencoders and demonstrates good results, it has certain limitations. Besides an obvious one related to the tiny size of images, the computational complexity of the proposed method has not been compared with the existing codecs even approximately. Moreover, training the model on such a large dataset would require some high-end equipment and a considerable amount of time, which was not described in the paper.

Nevertheless, taking into account a considerably better performance of the described autoencoder-based method over existing image formats, it is reasonable to ask how it can be adopted to larger images. This can be done in two ways: simply by increasing image resolution or by splitting a big image onto 32×32 blocks compressed separately. The first way is likely not be feasible in practice due to enormous computational complexity, but the second way was attempted by a slightly different team of authors in the work described below.

Toderici et al. (2017)

Considering a relative success of the autoencoder-based method described in [45] the research was apparently continued in pursuit of better quality to size ratio for bigger images. In their latest paper [46] Toderici, Vincent and others investigated the idea of compressing a relatively large image (1280×720 pixels in particular) block by block using similar approach.

However, the problem of compressing images block by block is not straightforward. There are couple of reasons why it is a considerably more complicated task than compressing a single block.

Firstly, Toderici et al. note that the amount of details inside a small sample taken from a large image is different from the case when it was obtained by resampling a bigger image like it was done in [45]. Small versions of large images tend to be more detailed. Indeed, consider an example of a road photo from figure 1.1. Its resized version of 32×32 pixels is shown on figure 2.1 along with a random sample of equal size copied from the same image. The content of the resampled original has noticeably higher detaili. So, the training process had to be adopted to a wider range of internal segment entropy.

(a) (b)

Figure 2.1: Level of detail comparison between an image resampled to 32×32 pixels (a) and an unmodified random block from the original image (b).

Secondly, a considerable amount of entropy in a large image is contained between the blocks, i.e. in the diversity of their content. A method that compresses them individually cannot discover possible correlations or similarities for any group of blocks. Toderici et al. deal with this problem by introducing an additional lossy entropy encoding step similar to existing image codecs. Due to using the iterative compression principle from the previous paper [45], the entropy coder keeps the context between iterations.

Toderici et al. tested the efficiency of the proposed algorithm on a small Kodac dataset using multiscale SSIM and PSNR-HVS (modified PSNR based on the human

visual system [47]) quality metrics. On average their method performs marginally better than JPEG on a range of compression ratios up to 2 bits per pixel, which can be considered a relative success. They admitted that no comparison with the WebP codec had been conducted, although stated that it is in the plans after they adopt the system to a variable block size method similar to the one used in WebP.

The results of this research were presented in Google blog [48], where engineers pointed out a noticeable blur in the compressed images. This type of compression artefacts is also present in the JPEG 2000 format, therefore, it would be natural to compare the proposed system with a JPEG 2000 codec. However, the comparison with JPEG 2000 appears to have been dropped between the two papers [45, 46].

Spectrum-based algorithms vs LSTM autoencoders

The papers from Google mentioned above [45, 46], propose a reasonable idea – to use specifically recurrent neural networks for image compression.

One of the common features of the DCT- and DWT-based image compression algorithms is that they perform lossy compression through quantizing the spectrum, i.e. deliberately throwing out a part of information from the image. In particular, the compression algorithms focus on removing relatively small details because deleting or smoothing any large enough elements on the image relief will immediately cause obvious distortions.

The problem is that there is a limited number of small details that can be removed without making image unrecognisable. In the discrete spectral representation this means that there is a finite number of high-frequency spectrum components that can be removed without destroying the image. The remaining low-frequency components are always compressed with a lossless entropy coder. Any entropy coder has a theoretical limit according to the Shannon theorem, unless a regular pattern is compressed, which is not the case for natural images. Consequently, it is possible to assume that compression algorithms based on the discrete spectral transformations have a certain limit to the compression ratio that can be achieved in practice while keeping image in a recognizable state.

However, using DCT with entropy encoder is not the only possible way to compress images. One of the alternatives was fractal image compression, in which the image compressibility is determined by the amount of similarities in its content. The definition also does not require to remove any details. Theoretically it is an excellent idea, but in practice it runs into difficulties, mainly due to the lack of similarities that can be found using any existing methods [49].

Assuming that a better image compression algorithm exists, but it is difficult to discover or design manually, it is reasonable to employ machine learning for this problem. In this light the idea of using recurrent neural networks looks promising because Siegelmann and Sontag [50] proved that such network can implement a Turing complete system. It is hard to imagine if a highly efficient image compression algorithm can be learned from scratch in practice, but it seems a worthwhile direction for future research.

Unfortunately, Toderici et al. did not discuss this advantage or give an explanation to their choice of recurrent networks, which leads to an assumption that they did not consider this fact and proposed to use LSTM for some other reason.

Romano et al. (2016)

Instead of developing an efficient encoder, it is possible to focus the efforts at improving image quality upon decompression. One of the recent research works by Romano et al. [51] proposes a new method – RAISR (Rapid and Accurate Image Super-Resolution) for increasing image resolution in 2, 3 or 4 times.

Image upscaling (or interpolation) is one of the important areas in computer graphics. The problem is that without any additional a priori information about a set of discrete samples (pixel brightness levels), in a general case it is not possible to recover the intermediate signal values. In some cases, for example, knowing frequency characteristics of the original signal it is possible to reconstruct it with certain accuracy according to the Nyquist-Shannon sampling theorem. However, when dealing with images in practice, the main assumption is a smooth colour transition between pixels in the image. Therefore, one of the widely used resampling techniques is a bicubic interpolation, which represents any 4×4 area as a superposition of cubic parabolas.

Romano et al. propose to store a priori information in the form of a table with optimal filters for different types of image fragments. It seems to be the most important feature of this work that differs it from the more complex alternatives based on convolutional networks.

When image is upscaled with an ordinary algorithm like bilinear or bicubic interpolation, the obtained result is different from the original high-resolution version (which may not exist in real world scenario, but always present in tests for comparison).

The RAISR approach is based on reconstructing not the whole image upon increasing resolution but only the difference between the ground truth and an already

resized version, which was obtained with bilinear or bicubic method. It was demonstrated how this idea can help reduce complexity and improve accuracy of the image enhancement.

In order to construct a "difference image", the proposed method uses a table with custom filters for different types of image patches. The filters were calculated through minimising a specially designed objective function, which is based on the statistical data collected upon resizing images from a training set.

In the proposed method an image is processed sequentially using overlapping patches. Finding a corresponding filter in the table for a previously unseen image patch is not a trivial task. The method employs a special hash function instead of more obvious clustering approach to reduce computational complexity.

Romano et al. also propose to add a sharpening effect and combine the upscaled result with bicubic interpolation for a visually better outcome. The emphasis of this work however is not inventing the best image resizing technique but obtaining the smallest computational complexity, which was achieved by design through time-optimising all stages of the algorithm. Nevertheless, RAISR demonstrates a level of quality compatible with the alternative methods.

Although the original paper does not deal specifically with image compression, Google advertised it as a technique suitable for image compression and fast upscaling, for example, in mobile devices [52]. The reason is that the process of resizing an image to smaller resolution and back can be considered as a compression operation. The problem of image resampling is related to the concept of dynamic resolution video discussed in this thesis.

Jiang et al. (2017)

Jiang et al. [53] proposed a new approach for image compression that combines several techniques:

– decreasing image resolution with subsequent upscaling;

– deep convolutional neural networks (CNN) for image resampling;

– the concept of autoencoder;

– using actual image codec as an intermediate compression stage.

The paper [53] describes an autoencoder system that includes three main components: the convolutional network to downsample input image into the fixed size of 64×64 pixels; an intermediate stage of compressing and decompressing the small image version with any external image codec; and the deconvolutional network that reconstructs an image in the original resolution. The compressed representation is

the result of encoding a small image, say, with JPEG.

This method can be considered simply as an ordinary compression into JPEG using smaller resolution with a subsequent advanced upscaling. This makes it closely related, for example, to the RAISR paper [51].

One of the main challenges in creating such a system was to train the networks responsible for changing image resolution. The problem is that due to using an external image compressor, the complete autoencoder structure is not differentiable between the original image at the input and the reconstructed image on the output. Jiang et al. used a specific iterative approach to train the model by processing the intermediate image compression stage externally. The non-differentiable part was replaced by a mean squared error metric calculated for 64×64 before and after JPEG compression. Training process consisted of only 50 epochs.

The most important part of the system is a deconvolutional network that upscales an image. Jiang et al. compared its efficiency with other image enhancing techniques. Unfortunately, there was no RAISR among them or any other highly efficient methods like SRCNN or A+ used for comparison with RAISR in [51].

For the role of intermediate compressor several image codecs were chosen: JPEG, JPEG 2000 and BPG (Better Portable Graphics). The BPG is a highly efficient custom implementation of the I-frame encoder from the H.265 video compression standard [54]. These codecs were also used by themselves in the quality comparison with the proposed method.

According to the presented results, the proposed system demonstrated the ability to provide a superior quality at the same compression ratio in comparison with all three tested image codecs. However, only a relatively narrow range of 0.1–0.4 bits per pixel was considered. Such compression levels typically correspond to below medium image quality. Due to involving image resampling, which is a relatively coarse compression method by itself, it is not possible to obtain a high quality result. In this light it is strange that Jiang et al. positioned the performance of their method as state of the art while being modestly silent about the lack of applicability for high quality encoding with compression ratio at least 1 bit per pixel.

In general, the work presented an interesting idea that led to excellent results at least for low quality compression. Among other limitations is that the method is suitable only for relatively small fixed resolution images. It is impractical to use for large images because even a big deconvolutional network cannot unroll a tiny intermediate representation into, say, a 24 MPix photograph.

What about video compression?

To the best of author's knowledge, none of the available popular or experimental video codecs explicitly use machine learning as a part of their algorithm specifically to improve compressibility of the video while maintaining the quality.

There is, however, some related work on video quality balancing and predictability of the video compression, which was described in the section 2.4.

2.7 Summary

Despite the fact that image and video compression have been the areas of active research for decades, there is still a considerable attention to developing new image compression algorithms from the scientific community.

As for the research related to the predictable compression, it is not numerous yet but already quite diverse in its methods and even approaches to the problem.

In particular, there is a substantial amount of effort dedicated to various transcoding methods for specific image and video formats. Table 2.1, containing a short comparative summary of the literature sources from sections 2.3 and 2.4, shows that there are papers on JPEG image and H.264 video transcoding applications.

The papers related to image compression the table 2.1 focus on building statistical models for explicit or implicit prediction of the JPEG compression. However, there is no attention to the time predicting models. Presumably, this is the case because compression time depends on a particular hardware and therefore is difficult to measure reliably.

Predictable video compression methods according to the same table 2.1 heavily rely on additional compression runs in order to obtain a certain result. This is one of the important issues addressed by this thesis.

Another essential aspect of the thesis – a concept of the content features for images and videos does not seem to be a popular approach. However, it is present in distinct forms in different research works.

In general, none of the recent research on predictable compression considers the problem from multiple perspectives taking into account both positive and negative aspects. Instead a typical situation is that a very narrow specific problem is emphasised, while other related observations are often left behind the scenes.

Table 2.1: Brief summary of the recent research related to predictable image and video compression.

Area	Research work	Authors	Year	Content features	Involves multiple recompressions	Transcoding method	Estimated objectives			Distinct characteristic
							Quality	Size	Time	
Image compression	JPEG compression metric as a quality aware image transcoding	Chandra et al.	1999	+ basic, handmade	–	+	+	+	–	Describes a basic JPEG transcoding method that considers image content and applies quality factor correction in case of a different original codec
	Computationally efficient algorithms for predicting the file size of JPEG images subject to changes of quality factor and scaling	Pigeon et al.	2008	–	–	+	–	+	–	A simple classifier-based JPEG transcoding method predicting a coarse relative change in compression ratio; also supports image scaling
	Computationally efficient algorithms for predicting the file size of JPEG images subject to changes of quality factor and scaling	Pigeon et al.	2010	–	–	+	+	+	–	The JPEG transcoding method supporting predictions of both compression ratio and MSSIM quality metric
	Quality-aware predictor-based adaptation of still images for the multimedia messaging service	Pigeon et al.	2014	–	–	+	+	+	–	Using JPEG transcoding to fit a set of images into given file budget while keeping the same quality
	Quality prediction of compressed images via classification	Tichonov et al.	2017	+ advanced, handmade	–	–	+	–	–	JPEG file size prediction using custom content features and classification
Video compression	Help save the planet: Please do adjust your picture	Ejembi et al.	2014	–	N/A	N/A	–	–	+	Estimations on energy consumption and complexity of the video playback depending on resolution
	Adaptive bitrate selection for video encoding with reduced block artifacts	Choi et al.	2016	–	+	–	+	–	–	Theoretical estimations on video quality balancing using multiple compressions
	Optimizing transcoder quality targets using a neural network with an embedded bitrate model	Covell et al.	2016	+ advanced, codec-specific	+	+	–	+	–	Machine learning for video transcoding from Google; using a very complex model for bitrate prediction for H.264 standard
	Netflix Dynamic Optimizer Tool	Netflix team	2017	–	+	–	+	–	–	New quality metric for constant quality video; optimal result is obtained through multiple compressions

Chapter 3

Predicting and Optimising Image Compression

This chapter introduces a general methodology for predicting the outcome of compression and demonstrates potential improvements in some use cases as a result of applying this methodology to practical scenarios related to JPEG and WebP image compression.

In particular, the following problems are considered:

– how images can be compressed into WebP format with given constraints faster than using built-in options;

– how to save file size when compressing into JPEG and keeping image quality above a certain threshold.

3.1 Methodology

The main problem that needs to be solved in order to perform compression with constraints is to find optimal compression parameters for a particular image, which subsequently lead to a desired outcome.

The core idea of the proposed approach is to create a statistical model to predict the characteristics of the result without performing actual compression. This allows to quickly estimate outcome of compression with different parameters for every individual image and find optimal settings faster than compressing multiple times.

In this chapter only one compression parameter is considered – the quality factor (or just "quality"). Although image codecs can have more than a single option, the quality factor is the one influencing the result to the most extent. It is present in both *libjpeg* and *libwebp* codecs and has the equal meaning and range of values (0–100).

Using this parameter, it is possible to apply the same algorithms for both image codecs, thus demonstrating universality of the methodology. Multiple compression parameters are used in the next chapter upon considering video compression.

The top square on figure 3.1 depicts a sketch of the statistical model that takes uncompressed image data with codec parameters as input and calculates file size or quality level values expected upon such compression. Raw image data in the model is represented by ten *content features*, which are used mainly to reduce the dimensionality of the input data in a relatively simple predefined way. The content features are calculated from raw pixel values and serve as a measure of entropy in the image.

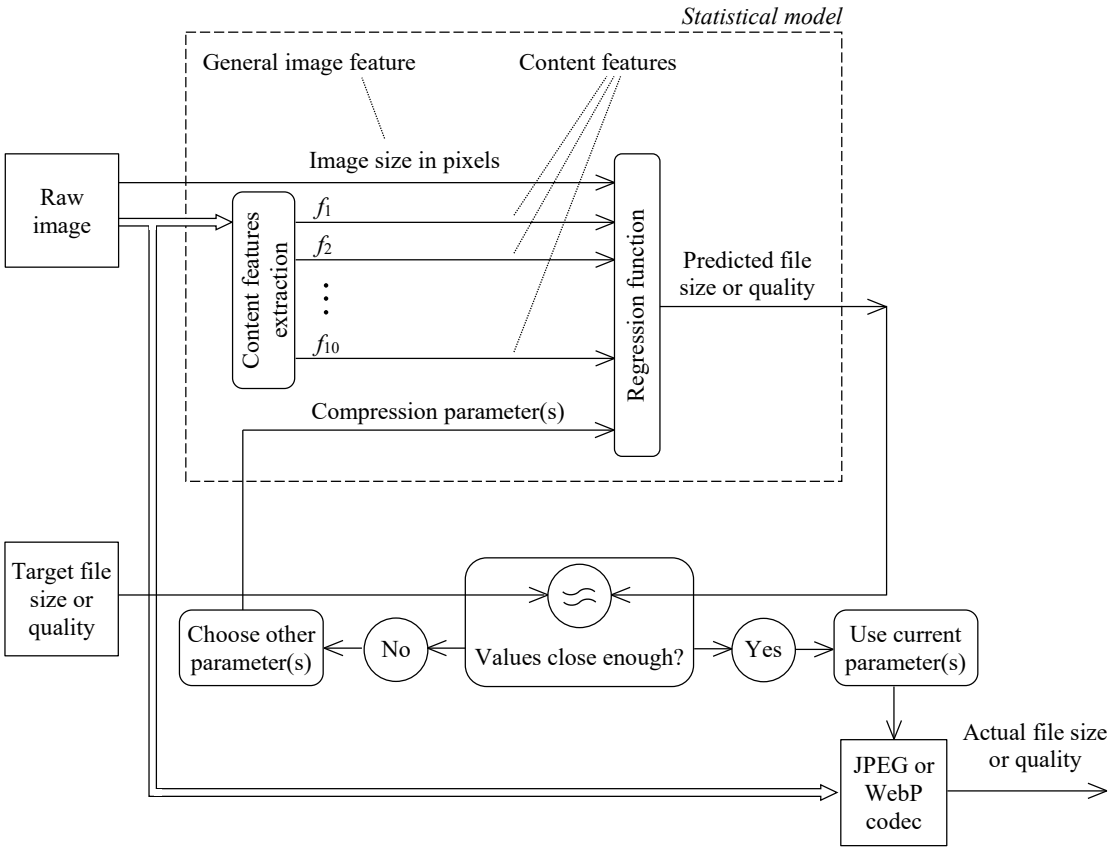


Figure 3.1: Sketch of the proposed method for compressing images with the desired outcome.

Predicted size or quality is calculated from image resolution, set of content features and compression options using regression function, which in turn is obtained through machine learning process involving statistical data for thousands of training images. Predicted value should be compared with the target objective to decide whether the chosen compression parameters are optimal in terms of satisfying given conditions. If they are not, the function is evaluated again using different parame-

ters. The choice of these parameters can be random or systematic – so it is possible to perform different kinds of search in the space of compression options, for example, binary search in case of monotonic dependencies between codec parameters and result characteristics.

The regression function is a standard feed forward neural network, which is cheap to evaluate and suitable for all problems considered in this research. It allows us to search over a space of encoding options in less than a millisecond assuming that we already have a set of content features for a particular image. The content features depend only on the image pixel data and need to be calculated once. Although this could be a relatively expensive operation, it was designed to be considerably faster than a single compression.

The main advantage of the proposed method is a smaller total encoding time in comparison with multiple recompressions. The statistical model is used as a heuristic for discovering optimal parameters, while actual compression can be done only once.

The accuracy of meeting a target objective in the described method depends on the accuracy of the statistical model. It is reasonable to expect the difference between, for example, target quality and the actual quality levels, to correspond to the difference between target and predicted values. Therefore, it is important to have content features accurately representing image content and minimise the errors during training of the regression function.

In case of a single compression parameter (quality factor), it is possible to use it directly as an output of the predictor, while having target objective as an input. As image size and quality usually grow monotonically with increasing quality factor (fig. 3.2), this can eliminate the necessity for parameter search. The drawback of this approach is in using standard error functions when training such models due to non-linearity of a quality factor scale. For an ordinary image the difference in file size between quality factors 49 and 50 is tiny in comparison with the difference between 99 and 100. The mean squared error will treat these differences as equivalent.

3.2 Content Features Description

The need for features arises from the necessity to have a relatively simple representation of the image content, which is independent from its resolution. Many image features used in computer vision can work at different resolutions, but they are oriented mostly at recognising objects rather than capturing general pixel-level information. All modern image codecs operate specifically with raw pixel values

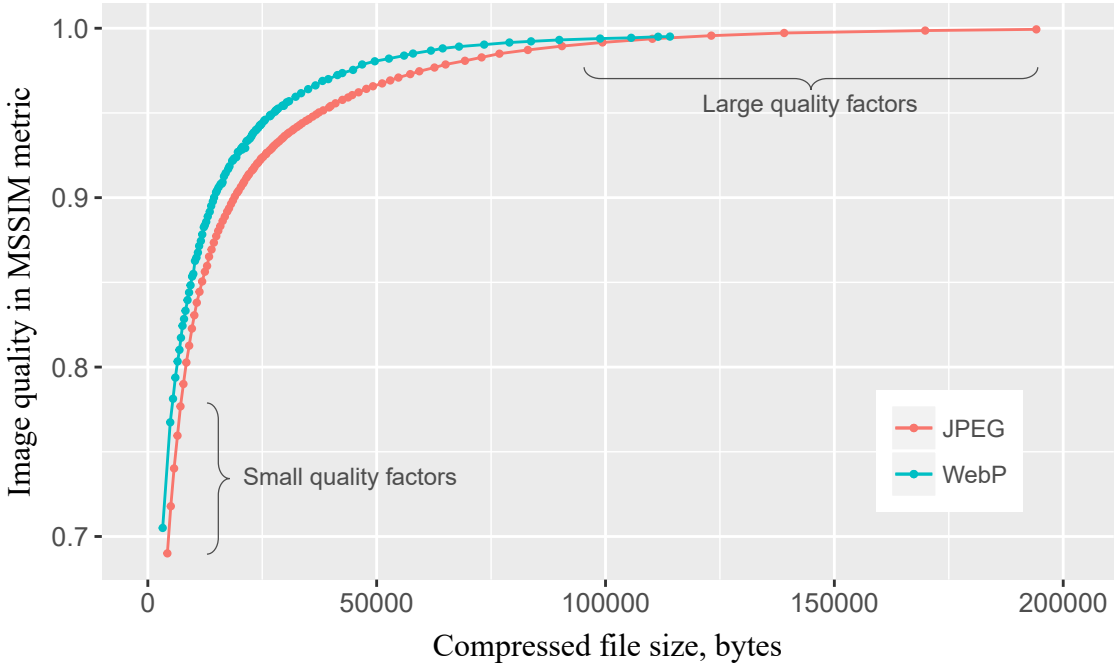


Figure 3.2: Size/quality trends for an image compressed with JPEG and WebP codecs using a range of quality factors (photo of the road from figure 1.1a was used as an example for this graph).

aiming to find regularities between them. The actual meaning of the pixels during compression is usually ignored. So, the independence from resolution in the current context means not an ability to identify some objects in the images of different sizes, but a way to extract the same amount of low-level information from an arbitrary sized image.

Therefore, specifically for the problem of predicting compression results, a set of ten image features was designed. These features were created manually based on the assumption that highly detailed and noisy images are harder to compress. The initial idea for a content feature was to use the average magnitude of the image gradient as a measure of noise. The actual gradient length is expensive to calculate but it inspired some simpler features used in this research. Another avenue explored for possible features was the assumption that the amount of detail in an image correlates with its high-frequency characteristics. In all popular spectral transformations like discrete cosine transform, discrete Haar wavelet transform and the Walsh-Hadamard transform [18, fig. 3] the high-frequency components are extracted by applying convolutions that look like a checkerboard pattern. This pattern was utilised in some of the proposed features.

The first step in calculating a set of content features for an image is to divide it into fragments of 8×8 pixels. This is not essential but a recommended operation.

Firstly, it facilitates program implementation by using vector instructions, which in turn reduces feature extraction time. Secondly, using a small constant size for sample fragments allows to skip some of them at a cost of prediction accuracy in cases when analysing entire image is too expensive, so only representative parts of the image – randomly selected fragments – can be taken into account.

At the next step the raw pixel values in RGB format are converted to YC_bC_r color space following ITU-R BT.601 standard [55], but using full range for luminance and chrominance components according to formulas:

$$\begin{aligned} Y &= 0.299\,R + 0.587\,G + 0.114\,B; \\ C_b &= -0.169\,R - 0.331\,G + 0.5\,B + 128; \\ C_r &= 0.5\,R - 0.419\,G - 0.081\,B + 128. \end{aligned}$$

The JPEG file format specification [56] uses these formulas too, so it is possible to integrate the proposed method directly into JPEG codec, avoid extra color conversion during feature extraction and consequently reduce total encoding time. However, in this work all feature extraction is done externally without any modifications to image codecs.

The last step is to calculate a vector of ten features for every 8×8 fragment and take their absolute values, which are considered *local content features*. The general content features for a particular image are obtained upon averaging these vectors of absolute values across all fragments sampled from that image. This technique ensures resolution independence.

Features $\boldsymbol{f_1}$–$\boldsymbol{f_9}$ are calculated only for luminance component Y. Feature $\boldsymbol{f_{10}}$ is calculated from both chrominance components C_b and C_r. The following is a detailed description of the local features.

Feature $\boldsymbol{f_1}$ is a mean absolute neighbour pixel difference. In simple words, if a and b are brightness levels of two adjacent pixels, then this feature is an arithmetic average of $|a-b|$. In program implementation only some of the possible differences in the fragment are considered to optimise for specific AVX instructions like horizontal subtraction. Figure 3.3a shows selected differences in the 8×8 fragment with arrows.

Feature $\boldsymbol{f_2}$ is a mean absolute difference between neighbour blocks of 2×2 pixels. It can be considered as a "lower frequency" modification of the $\boldsymbol{f_1}$. Conceptually $\boldsymbol{f_2}$ is the same as $\boldsymbol{f_1}$, but instead of pixels a and b it subtracts mean values for adjacent blocks 2×2 (fig. 3.3b). Again, not all possible differences are considered due to usage of AVX instructions.

Feature $\boldsymbol{f_3}$ is a mean absolute difference between neighbour blocks of 4×4 pixels. $\boldsymbol{f_3}$ is an analogue of $\boldsymbol{f_1}$ and $\boldsymbol{f_2}$ (fig. 3.3c).

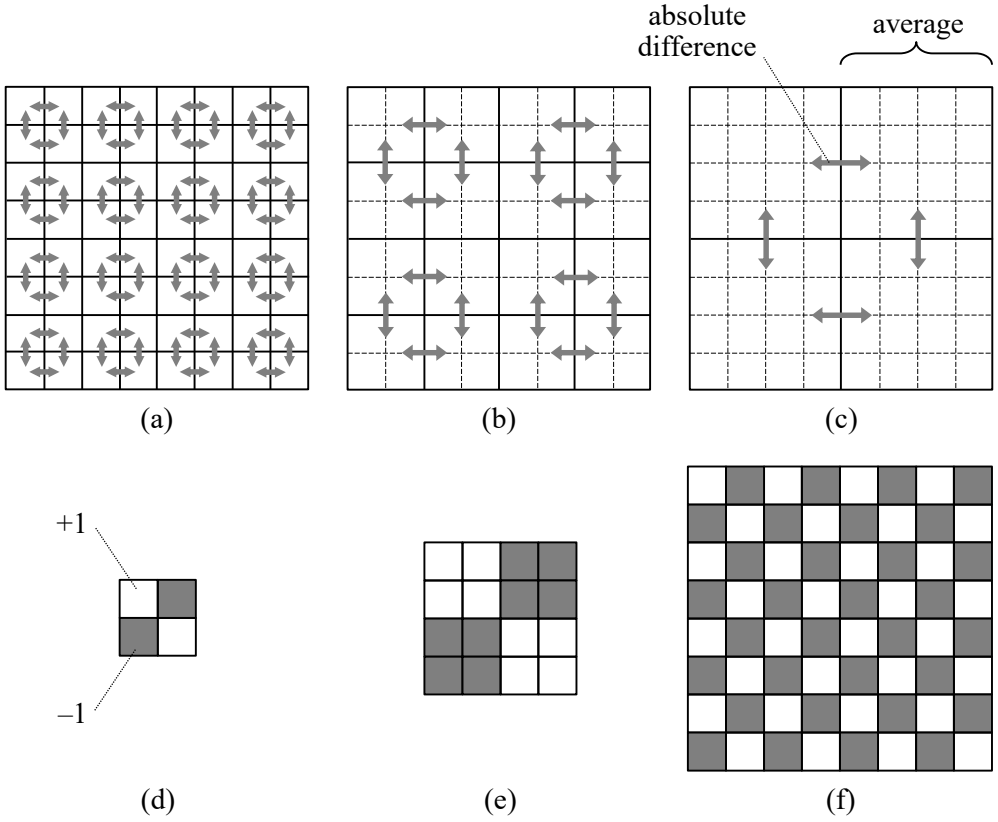


Figure 3.3: Differences between pixels and blocks for calculating features: (a) $\boldsymbol{f_1}$ and $\boldsymbol{f_4}$, (b) $\boldsymbol{f_2}$ and $\boldsymbol{f_5}$, (c) $\boldsymbol{f_3}$ and $\boldsymbol{f_6}$; and convolutions for features (d) $\boldsymbol{f_7}$, (e) $\boldsymbol{f_8}$, (f) $\boldsymbol{f_9}$.

Features $\boldsymbol{f_4}$, $\boldsymbol{f_5}$ and $\boldsymbol{f_6}$ are very similar to the described $\boldsymbol{f_1}$, $\boldsymbol{f_2}$, $\boldsymbol{f_3}$. They only use squared differences instead of absolute, for example, $\boldsymbol{f_4}$ is a mean squared neighbour pixel difference.

Features $\boldsymbol{f_7}$ and $\boldsymbol{f_9}$ are mean absolute values of "checkerboard" convolutions in all non-overlapping blocks of 2×2 and 8×8 pixels respectively (fig. 3.3d and 3.3f). By "checkerboard" convolution we mean the last high-frequency basis function in 2-dimensional Walsh-Hadamard transform (WHT) of the respective size [18, fig. 3]. Feature $\boldsymbol{f_8}$ is another WHT coefficient of lower frequency calculated in blocks of 4×4 pixels (fig. 3.3e).

$\boldsymbol{f_{10}}$ is the only feature extracted from chrominance components of the image. Basically, $\boldsymbol{f_{10}}$ is exactly the same as $\boldsymbol{f_2}$, but it is calculated and averaged for both colour components C_b and C_r instead of luminance Y.

The dynamic range of these features is quite large and the majority of values are close to zero (but always > 0). To ensure even coverage of the feature space by machine learning models all ten features are logarithmised: $\boldsymbol{f_i} \leftarrow ln(\boldsymbol{f_i} + 1)$. Incre-

ment is required to avoid $ln(0)$. Logarithmising makes feature values distribution close to normal (fig. 3.4).

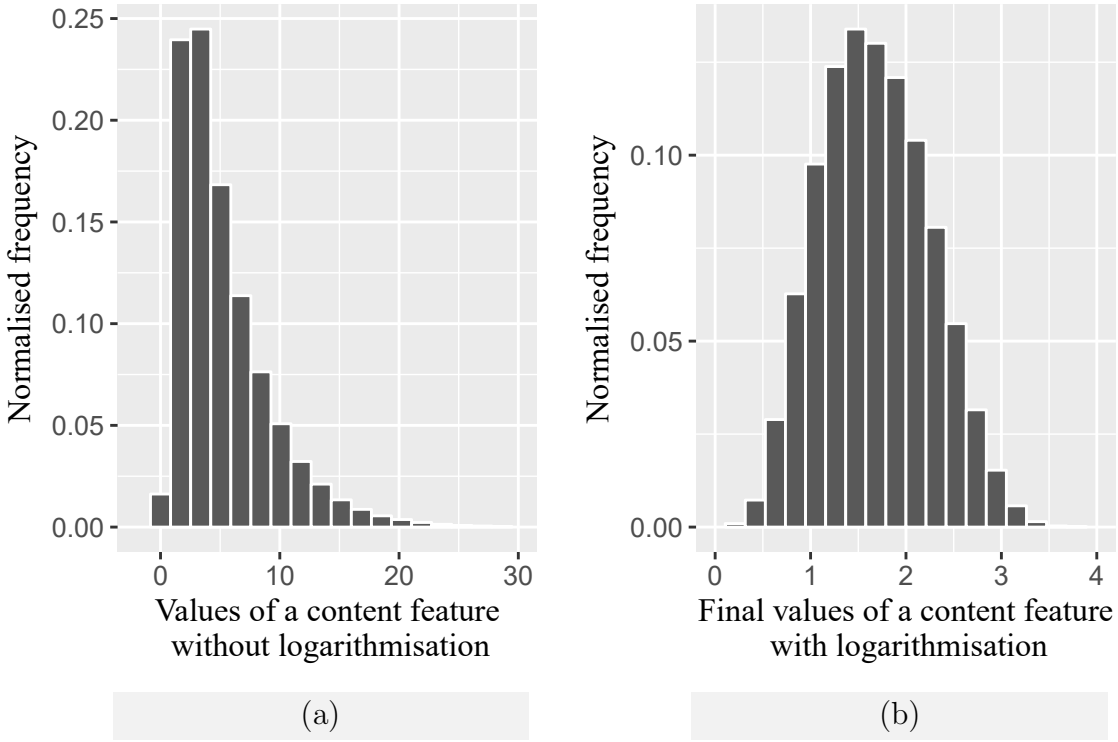


Figure 3.4: Histograms demonstrating how logarithmising influences content feature values distribution (using feature $\boldsymbol{f_1}$ as an example).

3.3 Feature Extraction Program

The process of extracting features involves calculation of various differences between neighbour pixels as well as convolutions in 2×2, 4×4 and 8×8 blocks. There were two program implementations made for calculating feature vectors for images and videos respectively. This section focuses on explaining the role of CPU vector instructions in the program implementation.

Including feature extraction stage into the experiments with JPEG images almost doubled total compression time for a single image. In order to decrease the computational expenses, the process of feature calculation was adopted for using vector instructions and the program was optimised with AVX/AVX2 intrinsics.

One of the first procedures to utilise vector instructions was RGB to YC_bC_r colour space transformation. It was a relatively straightforward operation because each pixel can be processed independently. However, calculating even simple pixel differences was not trivial.

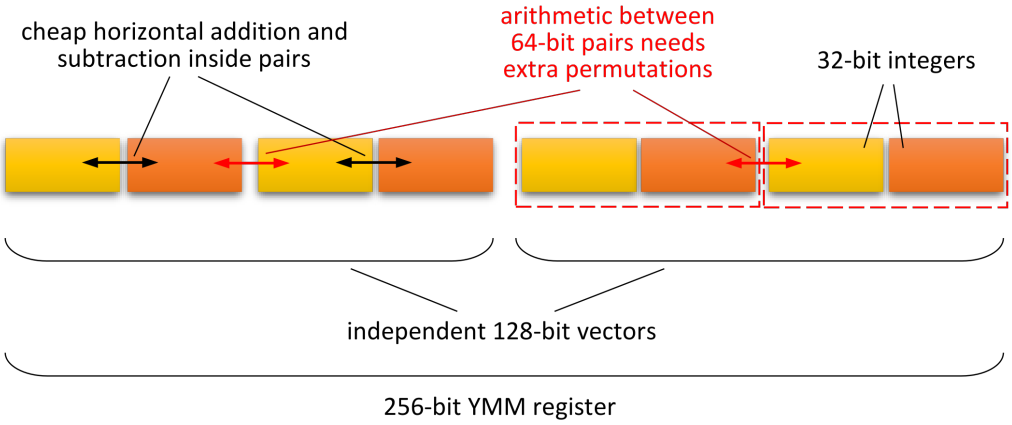


Figure 3.5: Relative cost of arithmetical operations between eight integers in the 256-bit vector.

The program implementation used integer arithmetic throughout all subroutines. Although pixel brightness is expressed with 8 bits, the vectors of feature accumulators for the entire image were made with 32-bit elements to be suitable even for large images. The largest vector size supported by AVX instructions is 256 bits, which corresponds to eight 32-bit integer components. Consequently, before extracting features the image was split into non-overlapping blocks 8×8 pixels, and most calculations were done inside a single block independently from the others.

The problem with AVX instructions is that they are very limited in terms of allowed arithmetical operations between different vector elements. Figure 3.5 shows that the cheap addition or subtraction is possible only in each of four 64-bit components. Doing arithmetical operation between them would need an extra shifting or permutation of vector elements. The two 128-bit components of an YMM register are independent from each other, so in order to add or subtract their elements they have to be copied into different locations, which increases the cost even more than for 64-bit components. Therefore, aiming to minimise usage of expensive arithmetic during feature extraction some of the pixel differences were excluded. Figure 3.6 demonstrates which pixel differences are the cheapest to calculate inside and between two 256-bit vectors. Removing some of the possible differences in this particular scenario has a negligible effect on the result but saves a considerable amount of time especially for large images that include millions of such vectors.

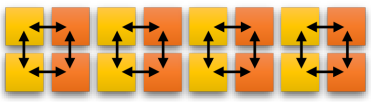


Figure 3.6: Horizontal and vertical pixel differences in two related vectors that can be quickly calculated with AVX instructions.

The AVX instructions were used only for calculating features in images and video frames, while some other video features like interframe differences described in the next chapter were implemented in C++ and not specifically optimised.

The program for feature extraction simply prints a vector of feature values with an optional in-memory processing time:

```
./image-feature-extractor-8x8 -i road_600x400_original.png -time
2.076279,2.200178,2.349404,5.291269,5.455056,5.628005,1.369269,
                                          1.412731,1.100775,0.498707
1 ms
```

3.4 Image Dataset

One of the important hypotheses this research aims to prove is that the proposed methodology works for a wide range of resolutions. The image dataset was designed to contain images from 600×400 pixels (0.24 MP) to 6000×4000 pixels (24 MP). There are two reasons to have such a wide range – firstly, to demonstrate universality of the methodology the proposed content features should be tested with both small images and large photographs, and secondly, to reflect modern real world use. The existing related research does not consider large images, but today 24 megapixels is a common photo resolution.

In total there are 100 different image resolutions in the dataset, which are logarithmically distributed by size in the range $[0.24; 24]$ megapixels. This provides an even coverage of different image sizes by design, so that accurate predictions can be made for an image of any resolution from this range.

Every resolution is represented by a dedicated set of 5000 different images of that size. This allows to calculate a reliable statistics for each resolution independently from the others.

Consequently, the experimental dataset considered in this chapter consists of $100 \cdot 5000 = 500\,000$ images. It was not a trivial task to collect so many different images of the appropriate sizes, assuming that upscaling them to bigger resolutions was not a reasonable option.

Initially only 100 000 distinct images were gathered from various image hosting websites and personal archives. These files were mainly photographs and had large resolutions from 24 MP to approximately 36 MP. All images were cropped to a constant aspect ratio of 3:2 to facilitate creation of the dataset. This was done under assumption that the aspect ratio does not have any significant effect on the outcome of compression (except for images with very small width or height). Then

every original photo was resampled into five smaller images of different resolutions using Fant method [57], which is a geometrically accurate anti-aliasing technique. The smaller versions can be considered different images without loss of generality and despite the fact that they picture the same scene. Image codecs look at the dependencies between pixels, and the pixel grid inevitably changes upon resizing. Thus 500 000 images were obtained.

The majority of the original images were in JPEG format. Resampling procedure destroyed JPEG blocking structure making the experimental images previously not compressed (or *uncompressed* for short). It is purely a side effect of the way this dataset was designed. On the one hand it is useful because uncompressed form is used as input in the proposed method anyway, but on the other hand the statistical models trained on such data may not be accurate for previously compressed images, which have specific artefacts at the pixel level.

If resized images do not have the same resolution they are technically different. Despite this fact some similarities may remain if an original image was resampled into very close sizes, for example, 2.5 MP and 2.6 MP. So, to eliminate the possibility of images, which contain the same scene, falling into both training and test sets they are grouped together and assigned to either training, validation or test set. Every twenty random original images were converted into hundred experimental images, all having different sizes. Figure 3.7 shows an example of assembling a group. These 100 images form a fixed set, which is subsequently processed as an indivisible unit. Such modular approach simplifies creation of a dataset with predefined image sizes.

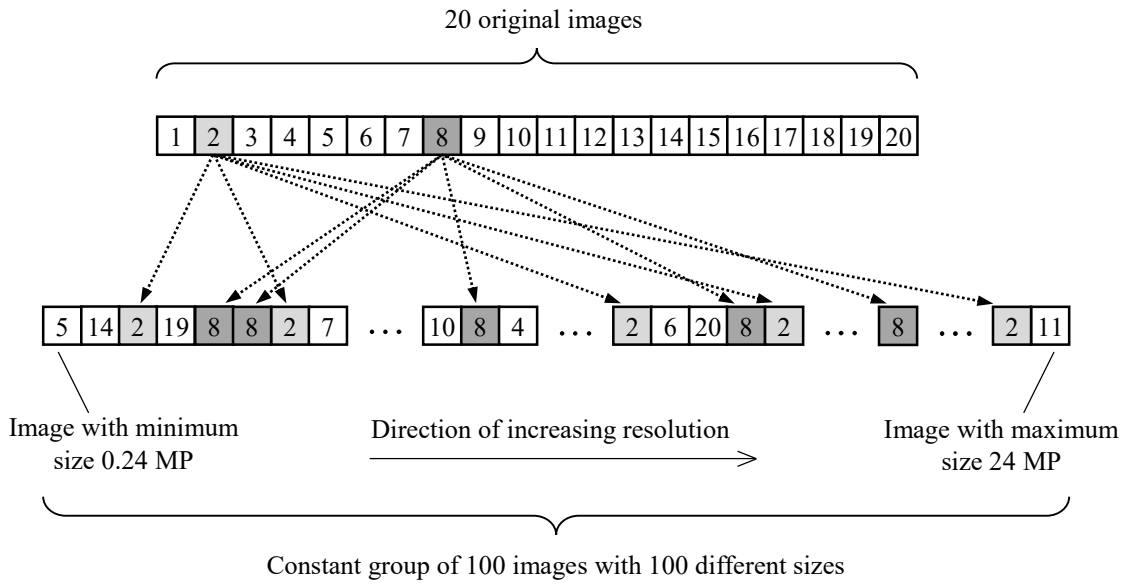


Figure 3.7: Group of 100 different images is formed by randomly resampling 20 large images into 5 smaller versions each.

Duplicates and very similar images were not included in the dataset. This was engineered by design and verified using an external tool – "Awesome Duplicate Photo Finder" [58].

There are about 4% of grayscale images in the entire set. They naturally occurred among those downloaded from the Internet.

3.5 Image Resampling Program

The dataset of images prepared for experiments consisted of the photos with resolutions from 600×400 to 6000×4000 pixels, which were obtained from the high resolution originals. In order to do so the large original images were resized to specific predefined resolutions. As this procedure was a part of the experiment preparation and not the actual tests, the only requirement for the resampling technique was to preserve the quality and details of the images.

A range of image scaling algorithms can be used for this purpose. The most popular are methods like bicubic and lanczos resizing based on a smooth curve interpolation. Their main advantage is a low computational complexity. However, the problem of reducing image resolution is conceptually similar to performing a multi-sample anti-aliasing, while the interpolation techniques are used mainly for enlarging images. In the multi-sampling approach approach the target pixel value is an average of several samples.

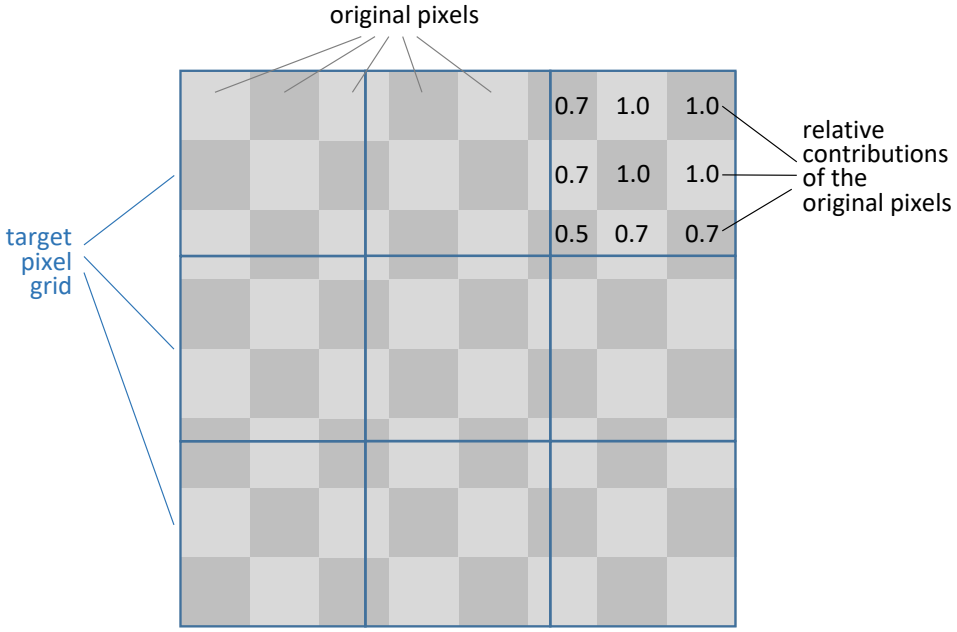


Figure 3.8: Resampling 8×8 pixels image into 3×3.

For example, the graphical editor Paint.NET by default uses Fant method [57] as a 'best quality' strategy for decreasing resolution. It is a geometrically accurate version of the multi-sample anti-aliasing technique that calculates the target pixel value as a weighted average of the original pixels that overlap with the target one. Although this method is slower than traditional widely used techniques, it was assumed more accurate and implemented for resizing images during experiment preparation. Figure 3.8 demonstrates overlapping pixel grids (assuming that pixels are square) in 8×8 original image and 3×3 target one. Each target pixel is calculated by averaging the fully contained original pixels and proportional fractions of the partially overlapping ones.

Technically, the resampling could have been done with an existing image processing tool. However, at the time of designing the experiments it was important from the exploratory point of view to look closely at each particular aspect of the investigation. It was unclear which direction could provide improvements or optimisation possibilities. Therefore, some of the existing techniques were reimplemented to see if adapting them specifically to the considered problems has any advantages. Moreover, it was originally assumed that special image resampling techniques may be an important part of the video compression experiemnts, which are described in the next chapter, in particular those related to the dynamic resolution concept.

3.6 Image Quality Measuring Program

Before getting into the experimental details it is important to describe how quality estimation was conducted in this chapter. The PNSR and MSSIM metrics were used in various quality measurements for the compressed images. Other metrics were not utilised mainly due to the issues with computational complexity and lack of implementation details.

The PSNR is de-facto a standard among objective quality metrics and the easiest one to implement. It is used in the WebP image codec as a target parameter, so in this work it was implemented for comparison purposes.

In order to compare the JPEG transcoding by Pigeon et al. [3] with the proposed machine learning techniques, an implementation of the MSSIM metric was created strictly following the original paper [2, section III.C]. The algorithm operates with the recommended 11×11 pixels sliding window and a 2-dimensional Gaussian weighting kernel with the sum normalised to 1 (table 3.1). The computations were performed according to the formulas in the section 2.5. The only difference from the standard version is that in the current implementation the sliding window moves

Table 3.1: Normalised weighting coefficients w_i
for calculating SSIM in an 11×11 pixels window.

0.000001	0.000008	0.000037	0.000112	0.000219	0.000274	0.000219	0.000112	0.000037	0.000008	0.000001
0.000008	0.000058	0.000274	0.000831	0.001619	0.002021	0.001619	0.000831	0.000274	0.000058	0.000008
0.000037	0.000274	0.001296	0.003937	0.007668	0.009577	0.007668	0.003937	0.001296	0.000274	0.000037
0.000112	0.000831	0.003937	0.011960	0.023294	0.029091	0.023294	0.011960	0.003937	0.000831	0.000112
0.000219	0.001619	0.007668	0.023294	0.045371	0.056662	0.045371	0.023294	0.007668	0.001619	0.000219
0.000274	0.002021	0.009577	0.029091	0.056662	0.070762	0.056662	0.029091	0.009577	0.002021	0.000274
0.000219	0.001619	0.007668	0.023294	0.045371	0.056662	0.045371	0.023294	0.007668	0.001619	0.000219
0.000112	0.000831	0.003937	0.011960	0.023294	0.029091	0.023294	0.011960	0.003937	0.000831	0.000112
0.000037	0.000274	0.001296	0.003937	0.007668	0.009577	0.007668	0.003937	0.001296	0.000274	0.000037
0.000008	0.000058	0.000274	0.000831	0.001619	0.002021	0.001619	0.000831	0.000274	0.000058	0.000008
0.000001	0.000008	0.000037	0.000112	0.000219	0.000274	0.000219	0.000112	0.000037	0.000008	0.000001

with 2 pixels offset instead of 1 in both horizontal and vertical directions. This has a negligible effect on the results but considerably decreases computation time. Among other optimisations, the program used Intel AVX instructions to obtain a reasonable processing time even for 24 MP images.

A standalone program implementation supporting both metrics was written in C++ to serve as a command line tool for batch processing scripts.

The PSNR and MSSIM were originally designed for grayscale images, therefore in the experiments they were calculated only for luminance component (Y) assuming that it contains most of the visual information and its quality degradation correlates with chrominance channels. The luminance quality metrics are sometimes denoted as Y-MSSIM and Y-PSNR, but the prefix was omitted for simplicity.

These metrics do not represent the best possible choice for image quality evaluation, however, they were used mainly for the purpose of comparison with the alternative compression strategies that rely on them. The subjective quality evaluation on people was not conducted due to the involvement of a large dataset in the experiments. The proposed methodology does not depend on any particular quality metric and operates with any of them in the same manner.

3.7 External Tools

In this work some external image compression libraries were used for encoding images into JPEG and WebP formats in the experiments. This section gives a brief description of these tools.

3.7.1 JPEG compressors

There are two popular program implementations of the JPEG standard. The classic reference implementation simply called *libjpeg* is maintained by the Independent JPEG Group. It defines the standard API (application programming interface) for compression and decompression procedures as well as the complete set of encoder and decoder programs written in C. An alternative unofficial implementation is called *libjpeg-turbo*. This library provides the same API with some extensions for the in-memory formats of the raw pixel data. The main feature of libjpeg-turbo is the optimisations involving CPU vector instructions, which allow a considerable decrease in the computational time for image encoding and decoding. Due to this aspect the compressed images produced by libjpeg-turbo are practically the same as in the libjpeg although not mathematically identical.

A relatively new project MozJPEG [59] aimed at improving JPEG compression has been developed by Mozilla since 2014 [60]. The main focus of this tool is to enhance the visual perception of the compressed image without increasing file size. The means for achieving such result include adaptive quantization for different image areas according to their content complexity and optimising the lossless Huffman compression. The MozJPEG is based on libjpeg-turbo, however it is considerably slower because of extra internal logic.

In 2017 Google also presented a tool for optimising JPEG compression called Guetzli [61]. Similar to MozJPEG it is aimed to improve quality to size ratio but with main focus on subjective quality perception. It uses Butteraugli image quality metric [43], which works as a psychovisual model of human vision in the Guetzli encoder. Due to the fact that Butteraugli metric is reliable only in cases of minor quality degradation, the Guetzli tool does not support JPEG quality factors below 70. One of its disadvantages is a slow compression time – encoding a single image can take few minutes. According to external testers [62] the codec attempts several recompressions and quality measurements to reach a specific level of quality metric. It is reasonable to assume that calculating Butteraugli is a main factor causing slow compression because it is very computationally intensive by design.

Using a slow compressor is not an optimal choice for experiments involving thousands of large images. Therefore, all tests in this work were based on the libjpeg-turbo as the fastest software implementation of the JPEG standard used in practice. For example, it is installed in the Ubuntu operating system as a default library for JPEG support. In addition, usage of a fast image compressor in the experiments allowed to demonstrate empirically that the time spent on the image analysis is still considerably smaller than the actual compression.

A typical JPEG library contains *cjpeg* and *djpeg* applications for encoding and decoding images. However, these programs were rarely used in the experiments. Instead, a specially designed application was written in C++ for compressing images into JPEG. It utilised the standard API from the externally linked library mainly with a purpose of precisely recording the compression time when the raw image data is already loaded into memory.

The compression result produced with a custom program is equivalent to the following example of the encoding command:

```
./cjpeg -optimize -quality 90 -outfile output.jpg input.tga
```

The *--optimize* flag corresponds to the Huffman code optimisation, which was enabled in all experiments. The quality factor was specified with *--quality <int>*.

3.7.2 WebP compressor

Google's WebP is a relatively new image format, which is still under development and supports lossy and lossless image compression as well as alpha channel (pixel transparency). The reference implementation is called *libwebp*. In the experiments it was used only for lossy compression of the photographic images without transparency.

The compression algorithm is more complicated than in JPEG. It includes two passes over the image, which predict similarities in the neighbour blocks and convert the subsequent differences with DCT or other spectral transformation. The coefficients are compressed with a modified arithmetic encoder, which is usually slower but more efficient than Huffman compression in JPEG.

Due to the complicated compression algorithm it takes approximately ten times longer to compress an image with WebP codec than using the ligjpeg-turbo. Image codecs become slower with increasing complexity of the compression algorithms. So it is reasonable to expect that the need to get a desired result without multiple recompressions becomes more apparent.

The WebP codec supports quality factor parameter similar to JPEG implementations. Therefore, adding WebP into the tests did not require any major redevelopment of the experimental strategy or scripting. It was one of the reasons why it was chosen over the JPEG 2000. The standard compression with a given quality factor can be done using the *cwebp* tool. The following command line syntax is an example of compressing image with quality factor 90:

```
./cwebp -q 90 input.png -o output.webp
```

Instead of quality factor the codec supports compression into given size or quality in the PSNR metric, which can be specified with parameters *-size <int>* or *-psnr <float>* respectively. According to the user manual [63] WebP encoder performs multiple recompression passes in order to accurately fit a given constraint. Up to ten passes can be specified with *-pass <int>* option. The accuracy increases with the number of partial compressions performed. The following command is an example of compressing an image with 100 kB target size using four passes:

```
./cwebp -size 100000 -pass 4 input.png -o output.webp
```

The cwebp tool was used for data collection in all experiments, and the compression time measurement was performed externally. The WebP library API was utilised only in the ACACIA application presented in the next chapter, which was created after completing the experiments. Due to a relatively long compression time, the milisecond precision was not necessary, but in order to minimise the overhead for input/output operations, the images were copied to virtual disk in RAM prior to compression.

Although the WebP codec supports multithreaded compression, all tests were done in a single CPU thread because the feature extraction stage was implemented without multicore parallelism using only AVX instructions.

3.8 Experimental Setup

The experiments in this chapter were conducted on an Intel Core i7 5820K @ 4.0 GHz, 16 GB RAM running Ubuntu 14.04 64-bit with GCC 4.8 compiler. Important calculations like feature extraction and training the models were performed in double precision floating point format. The external image compression tools used in the experiments were libjpeg-turbo-1.4.2 and libwebp-0.4.3.

3.9 Gathering Image Compression Statistics

Collecting quantitative data from the compressed images was an essential stage of the experiments. In particular, such characteristics as file size, relative quality and compression time were recorded. The goal was to prepare information for training and testing of the machine learning models.

The statistical data was gathered for the entire dataset. It started with calculating content features for all 500 000 images. Then every experimental image was compressed once into both JPEG and WebP formats with integer quality fac-

tors randomly selected from intervals $[5; 100]$ and $[0; 100]$ respectively. The JPEG quality factors below 5 were not used in the experiments as they are not practical because they create severe image distortions. Therefore, the focus was principally on the useful parameters.

The resulting file size and quality were recorded for every image after compression. The encoding time was noted only for some of the images compressed with the WebP codec in order to compare it with a multi-pass encoding in one of the experiments. General time predicting models were not created for image codecs. The main reason is that the image content and quality factor have a negligible influence on the compression time for JPEG codec (it is not the case for WebP, but this format was tested only as an alternative validation of the proposed concept). The time difference between images (especially small ones) is hard to measure reliably because it is only few milliseconds, which is the level of a measurement error. Another reason is related to difficulties in organising an unbiased compression time recording for such a big dataset. The images should be processed sequentially in a single CPU thread for the duration of several weeks, without any other calculations running at the time, which was impractical when other experiments had been conducted. Nevertheless, encoding time is an important objective that was taken into consideration during tests.

In the context of compression time it is reasonable to note that, for the JPEG codec, a parameter called Huffman codes optimisation was enabled in all experiments. This option was chosen upon assumption that it is often used in practice allowing a noticeable reduction in file size without affecting the quality but at the cost of increased compression time. For example, for a 24 MP image the encoding time needed is almost 200 ms in comparison with 120 ms using only default settings. Assuming that compression time depends mostly on the image resolution, the largest size was used as a reference for further comparisons because it is more reliable for time measurements than the smaller ones.

All WebP codec parameters except quality factor were kept at their defaults during experiments.

In general, the following steps were performed for every image:
– recording extracted features;
– compression with a random quality factor into JPEG or WebP;
– recording compression time and file size;
– decompression into a lossless format (*.bmp or *.png);
– measuring quality degradation in MSSIM or PSNR metric.

The process of calculating a vector of content features representing image com-

plexity needed to be done only once for every image in the dataset regardless of how many codecs were used and how many times the image was compressed. This is an essential component of the proposed universal methodology for image compression.

The entire procedure of collecting the data was organized using standard bash scripting language in the Ubuntu operating system.

The compression step was performed in a single CPU thread for all images. Due to the lack of reliability, the encoding time was not measured during JPEG compression, therefore this particular case allowed to process several images simultaneously in parallel using a special external tool – GNU Parallel [64].

Here is an example of how Parallel was used with a custom bash function that compresses and records data for a single image:

```
#!/bin/bash
function processImage {
    ...
}
export -f processImage
parallel -a list_of_images.csv --jobs 12 processImage
```

The collected statistical information was subsequently used to train regression models for predicting compression result characteristics.

3.10 Regression Models

The role of a regression model in the proposed methodology is to map image features and quality factor to compression ratio or quality loss. The models are obtained by utilising a standard machine learning approach – gathering enough data about compressed images and using it for regression analysis, which yields a model representing a functional dependency between input and output values.

After statistical data was recorded for all experimental images, they were split onto training (10%), validation (10%) and testing (80%) sets (fig. 3.9). The majority of images were dedicated to the test set for the purpose of obtaining more reliable estimations of the proposed method's accuracy. Having images already split into groups of 100 (see fig. 3.7) ensured that all resolutions were equally represented in all three subsets.

The regression analysis starts with choosing an optimal parametric model, which is an important a priori assumption for any machine learning approach that uses

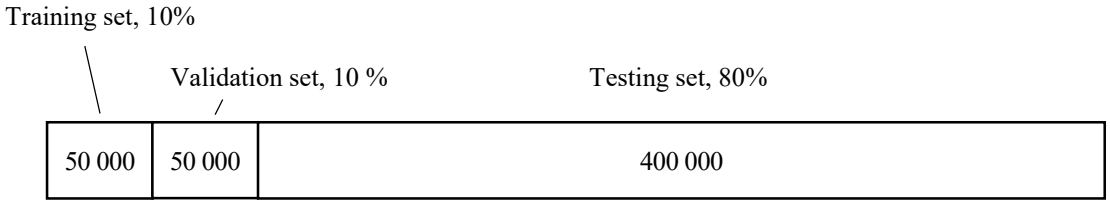


Figure 3.9: Number of images in training, validation and testing sets.

explicitly parametrised model. It should be based on the types of dependent and independent variables as well as on the basic assumptions about the geometric form this functional dependency may have – e.g. linear, non-linear, discrete, periodic.

In the problem of predicting compressed size or quality all input and output parameters are continuous numerical variables. The input vector corresponding to every experimental image consists of twelve values: ten content features, image size in megapixels and quality factor (different for JPEG and WebP).

Consider, for example, a model predicting compressed file size of the JPEG images. It is necessary to investigate the correlations between input parameters and the resulting size before putting forward an assumption about how the functional dependency should look like. Figures 3.10 and 3.11 demonstrate apparent strong correlations between one of the image features or the compression parameter and a logarithm of the JPEG file size. Consequently it is reasonable to start with a linear regression to build a predictor.

The file size values should be logarithmised for better accuracy according to trends on the graphs. All content features already have a logarithmic scale by definition. Image sizes in megapixels were logarithmised too as they have a suitable distribution by design. A standard linear regression is performed through minimising a root-mean-squared error (RMSE) [65]:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}.$$

However, because of the logarithmised file size such metric is not very informative from the user perspective. So, this chapter uses an alternative error metric to evaluate file size models – mean absolute percentage error:

$$mean\_abs\_percentage\_error = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{size_{predicted} - size_{target}}{size_{target}}\right| \cdot 100\%.$$

The percentage error allows to compensate for a wide distribution of the file size values across several orders of magnitude by normalising the absolute error.

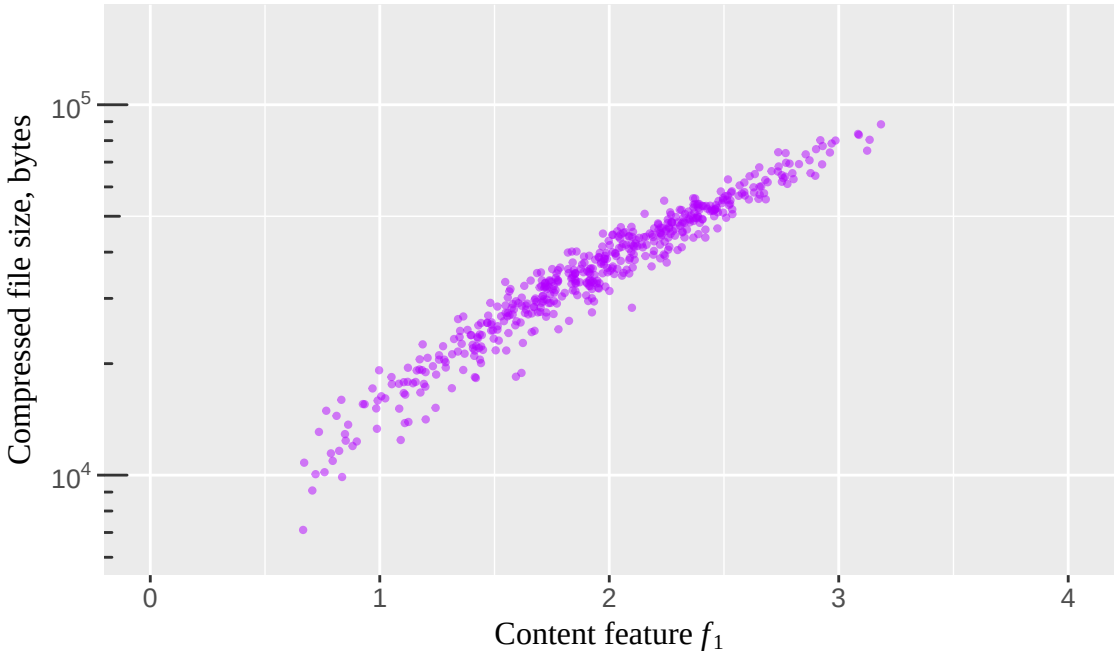


Figure 3.10: Correlation between first content feature $\boldsymbol{f_1}$ and the default JPEG size for 500 training images 600×400 pixels.

Figure 3.11: Correlation between quality factor and file size of JPEG images for 500 training images 600×400 pixels.

The mean absolute error is used for models predicting quality metrics like MSSIM and PSNR with relatively small range of values:

$$mean\_abs\_error = \frac{1}{n}\sum_{i=1}^{n} \left| quality_{predicted} - quality_{target} \right| .$$

The error of the trained linear model for predicting JPEG compressed size was 18.4% for the entire test set. This number implies the average deviation of the predicted file size from the actual compressed image size. This is a relatively large error, which may cause inaccurate results in some practical use cases. This model utilised all twelve input parameters. In order to estimate the importance of content features another model was trained based only on two inputs: image resolution and quality factor, excluding all the features. The average error for 400 000 test images was 51.8%, which means that content features are essential for the model's accuracy.

A non-linear regression model was employed to reduce the error. It was composed of a standard feed-forward neural network with a single hidden layer of ten neurons, and it was used to train an alternative regression function using all twelve input variables (fig. 3.12). After 10 000 iterations of the gradient descent with momentum the model's error on the test set was 4.5%, which is considerably lower than 18.4% with linear model. Supposedly, the neural network (or multilayer perceptron, MLP) enabled more interactions between input features allowing such a dramatic improvement. See Appendix A for alternative approaches that can be used for solving this regression problem.

Prior to actual MLP training the input values were statistically standardised to ensure that they have approximately the same influence on the error gradient in the beginning of the training process, because it is unclear which variables are more significant. A z-score standardization was used for this purpose [66, p. 524]:

$$standardised\_input\_value = \frac{input\_value - mean}{standard\_deviation}.$$

In comparison with simplicity of linear regression, the result of MLP training depends on several parameters like the network configuration, initial weights, learning algorithm, number of training iterations etc. To increase probability of obtaining a good model the network can be trained multiple times with different configurations assuming that training is sufficiently long. The best model should be chosen based on the error for the validation set, which does not participate in gradient calculation during training process.

The pilot result of 4.5% error is not the best one that an MLP model is capable of achieving. Table 3.2 contains results of a systematic approach towards discovering

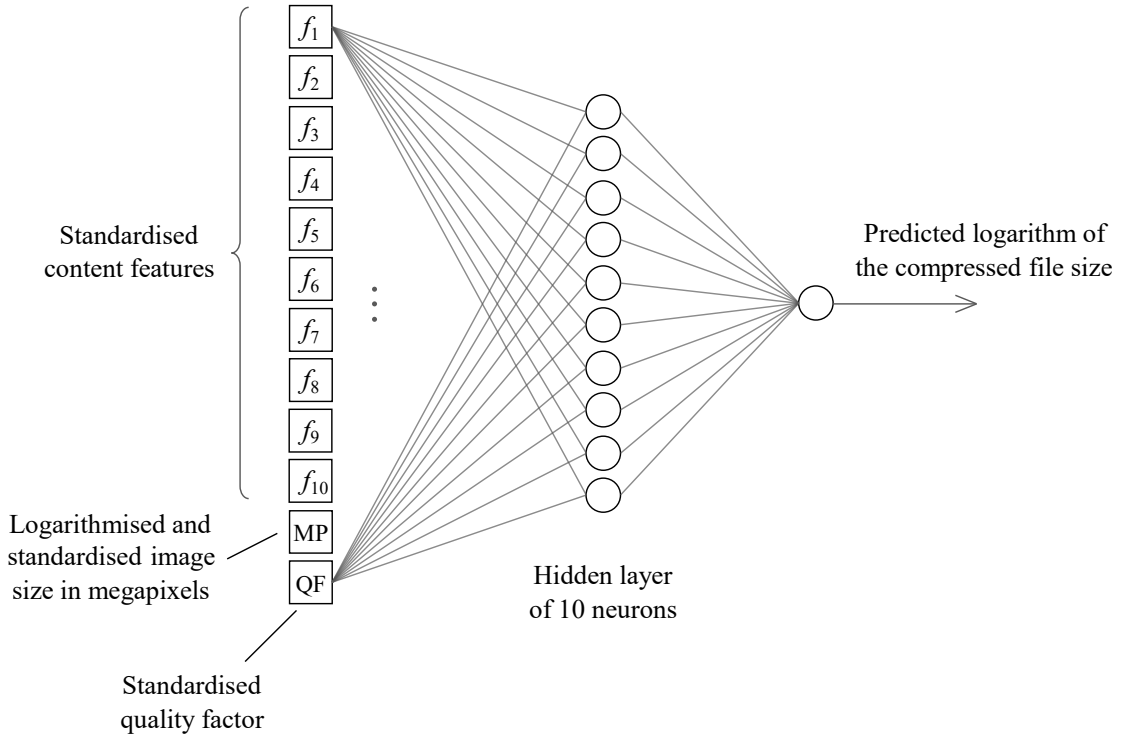


Figure 3.12: Sketch of the first trained feed-forward neural network.

a better model. There are five network configurations with ten to fifty neurons in a hidden layer and three sets of training data representing from 5% to 20% of the entire dataset, which all were tested in fifteen combinations, and each combination contained ten models trained with different random initial states. The highlighted line indicates the best model in each scenario according to the validation set error.

The last columns in each combination from table 3.2 named "relative correction of prediction" were used only as an extra sanity check during models training, in particular to indicate the worst cases. The correction of prediction is a percentage of the predicted value by which it should be adjusted to obtain exact target value. However, this data was not used anywhere later because simpler metrics were preferred mainly with intent to publish results in a conference paper.

Apparently there is not much difference between best models obtained with different number of training images. Even 5% of the dataset (25 000 images) allows to get results similar to the models trained on 20% of all images – training and validation sets merged together. Initially planned purpose of the validation set was to use it in a regularizing technique called early stopping [67, chapter 7], when the best result is chosen by the smallest validation error across all training iterations. Theoretically this allows to capture a point where overfitting starts, assuming that training and validation sets equally represent the parameter space. However, overfitting never occurred in these experiments because of the large number of train-

Table 3.2: Selecting the best model for JPEG file size prediction among several models trained with different settings.

Size of training data set (% of total)	Test run	10 hidden neurons (141 degrees of freedom)					20 hidden neurons (281 degrees of freedom)					30 hidden neurons (421 degrees of freedom)					40 hidden neurons (561 degrees of freedom)					50 hidden neurons (701 degrees of freedom)				
		Mean absolute percentage errors			Relative correction of prediction for the test set		Mean absolute percentage errors			Relative correction of prediction for the test set		Mean absolute percentage errors			Relative correction of prediction for the test set		Mean absolute percentage errors			Relative correction of prediction for the test set		Mean absolute percentage errors			Relative correction of prediction for the test set	
		Train. set	Valid. set	Test set	Range (worst cases)	99% conf. interval	Train. set	Valid. set	Test set	Range (worst cases)	99% conf. interval	Train. set	Valid. set	Test set	Range (worst cases)	99% conf. interval	Train. set	Valid. set	Test set	Range (worst cases)	99% conf. interval	Train. set	Valid. set	Test set	Range (worst cases)	99% conf. interval
5%	1	3.66%	3.71%	3.71%	[-72%; +60%]	±15.4%	3.2%	3.29%	3.28%	[-66%; +297%]	±13.9%	3.07%	3.19%	3.19%	[-64%; +204%]	±13.7%	2.99%	3.13%	3.13%	[-63%; +280%]	±13.4%	2.95%	3.12%	3.12%	[-66%; +240%]	±13.4%
	2	3.7%	3.77%	3.75%	[-73%; +63%]	±15.7%	3.2%	3.29%	3.28%	[-70%; +80%]	±14.0%	3.06%	3.18%	3.18%	[-63%; +161%]	±13.6%	3.0%	3.15%	3.15%	[-65%; +280%]	±13.5%	2.95%	3.12%	3.11%	[-66%; +167%]	±13.4%
	3	3.81%	3.87%	3.86%	[-68%; +70%]	±16.0%	3.21%	3.31%	3.29%	[-70%; +170%]	±14.0%	3.09%	3.21%	3.21%	[-67%; +234%]	±13.8%	3.01%	3.15%	3.15%	[-62%; +395%]	±13.4%	2.97%	3.13%	3.12%	[-68%; +517%]	±13.4%
	4	3.73%	3.78%	3.77%	[-74%; +80%]	±15.9%	3.22%	3.31%	3.3%	[-67%; +107%]	±14.1%	3.09%	3.21%	3.2%	[-67%; +234%]	±13.7%	3.0%	3.15%	3.14%	[-66%; +534%]	±13.5%	2.98%	3.14%	3.14%	[-70%; +321%]	±13.5%
	5	3.77%	3.84%	3.83%	[-74%; +131%]	±16.2%	3.21%	3.3%	3.3%	[-65%; +164%]	±13.9%	3.05%	3.15%	3.15%	[-66%; +103%]	±13.4%	3.0%	3.15%	3.14%	[-68%; +170%]	±13.5%	2.94%	3.11%	3.12%	[-74%; +174%]	±13.4%
	6	3.73%	3.8%	3.79%	[-73%; +63%]	±15.7%	3.22%	3.31%	3.31%	[-63%; +201%]	±14.0%	3.09%	3.2%	3.19%	[-66%; +99%]	±13.6%	3.0%	3.15%	3.15%	[-66%; +589%]	±13.5%	2.95%	3.12%	3.12%	[-62%; +140%]	±13.4%
	7	3.74%	3.8%	3.79%	[-75%; +67%]	±15.7%	3.22%	3.3%	3.29%	[-71%; +185%]	±14.1%	3.06%	3.19%	3.18%	[-67%; +530%]	±13.7%	2.99%	3.14%	3.14%	[-62%; +276%]	±13.5%	2.95%	3.12%	3.12%	[-66%; +173%]	±13.4%
	8	3.83%	3.89%	3.88%	[-73%; +259%]	±16.1%	3.16%	3.25%	3.24%	[-67%; +308%]	±13.9%	3.06%	3.17%	3.17%	[-68%; +434%]	±13.6%	3.0%	3.14%	3.13%	[-65%; +99%]	±13.5%	2.94%	3.11%	3.11%	[-64%; +114%]	±13.4%
	9	3.7%	3.77%	3.76%	[-68%; +104%]	±15.8%	3.2%	3.3%	3.3%	[-67%; +338%]	±14.1%	3.08%	3.19%	3.19%	[-65%; +310%]	±13.6%	3.0%	3.16%	3.15%	[-66%; +109%]	±13.5%	2.95%	3.13%	3.13%	[-66%; +99%]	±13.5%
	10	3.71%	3.77%	3.76%	[-73%; +90%]	±15.5%	3.26%	3.36%	3.35%	[-68%; +400%]	±14.2%	3.08%	3.2%	3.19%	[-68%; +165%]	±13.6%	3.01%	3.15%	3.14%	[-65%; +261%]	±13.5%	2.96%	3.13%	3.12%	[-61%; +204%]	±13.4%
10%	1	3.67%	3.71%	3.7%	[-74%; +70%]	±15.3%	3.25%	3.3%	3.3%	[-70%; +69%]	±14.0%	3.08%	3.16%	3.15%	[-69%; +117%]	±13.4%	3.01%	3.1%	3.1%	[-66%; +107%]	±13.2%	2.96%	3.07%	3.07%	[-67%; +189%]	±13.2%
	2	3.85%	3.89%	3.88%	[-72%; +101%]	±16.1%	3.2%	3.27%	3.26%	[-69%; +123%]	±13.8%	3.12%	3.2%	3.2%	[-71%; +154%]	±13.6%	3.01%	3.1%	3.1%	[-69%; +171%]	±13.2%	2.95%	3.06%	3.05%	[-68%; +97%]	±13.0%
	3	3.76%	3.79%	3.77%	[-72%; +57%]	±15.8%	3.22%	3.29%	3.28%	[-69%; +74%]	±14.1%	3.05%	3.13%	3.12%	[-68%; +127%]	±13.4%	3.01%	3.09%	3.08%	[-67%; +76%]	±13.2%	2.98%	3.08%	3.08%	[-68%; +132%]	±13.1%
	4	3.71%	3.75%	3.74%	[-73%; +88%]	±15.7%	3.19%	3.25%	3.25%	[-66%; +160%]	±13.7%	3.08%	3.16%	3.16%	[-70%; +149%]	±13.4%	2.98%	3.08%	3.08%	[-68%; +81%]	±13.2%	2.98%	3.08%	3.08%	[-66%; +175%]	±13.2%
	5	3.67%	3.7%	3.7%	[-73%; +80%]	±15.5%	3.2%	3.25%	3.24%	[-67%; +81%]	±13.8%	3.09%	3.17%	3.16%	[-68%; +161%]	±13.5%	2.99%	3.09%	3.08%	[-67%; +105%]	±13.3%	2.97%	3.07%	3.07%	[-65%; +81%]	±13.1%
	6	3.77%	3.81%	3.8%	[-74%; +75%]	±16.1%	3.19%	3.25%	3.25%	[-68%; +204%]	±13.9%	3.08%	3.15%	3.15%	[-70%; +169%]	±13.4%	3.02%	3.11%	3.11%	[-67%; +130%]	±13.3%	2.97%	3.07%	3.07%	[-68%; +100%]	±13.1%
	7	3.78%	3.82%	3.81%	[-74%; +102%]	±16.1%	3.21%	3.27%	3.26%	[-69%; +133%]	±13.9%	3.09%	3.15%	3.15%	[-69%; +99%]	±13.4%	3.02%	3.11%	3.11%	[-67%; +82%]	±13.2%	2.95%	3.05%	3.05%	[-68%; +87%]	±13.1%
	8	3.71%	3.75%	3.74%	[-73%; +89%]	±15.7%	3.23%	3.29%	3.28%	[-68%; +106%]	±14.0%	3.1%	3.17%	3.16%	[-70%; +222%]	±13.5%	3.01%	3.1%	3.1%	[-67%; +169%]	±13.2%	2.97%	3.08%	3.07%	[-68%; +223%]	±13.2%
	9	3.84%	3.88%	3.86%	[-72%; +76%]	±16.0%	3.24%	3.3%	3.3%	[-70%; +83%]	±14.0%	3.08%	3.15%	3.15%	[-68%; +185%]	±13.4%	3.03%	3.13%	3.12%	[-65%; +286%]	±13.3%	2.97%	3.07%	3.07%	[-67%; +104%]	±13.1%
	10	3.83%	3.87%	3.85%	[-72%; +140%]	±16.0%	3.2%	3.26%	3.26%	[-66%; +107%]	±13.7%	3.08%	3.15%	3.15%	[-68%; +156%]	±13.4%	2.99%	3.08%	3.08%	[-67%; +85%]	±13.2%	2.97%	3.08%	3.08%	[-69%; +86%]	±13.2%
20%	1	3.76%	–	3.78%	[-72%; +57%]	±15.9%	3.21%	–	3.26%	[-68%; +66%]	±13.8%	3.11%	–	3.17%	[-65%; +176%]	±13.4%	3.01%	–	3.08%	[-67%; +148%]	±13.1%	2.99%	–	3.07%	[-65%; +119%]	±13.1%
	2	3.73%	–	3.75%	[-73%; +70%]	±15.8%	3.2%	–	3.24%	[-69%; +84%]	±13.8%	3.08%	–	3.13%	[-69%; +113%]	±13.3%	3.02%	–	3.09%	[-67%; +82%]	±13.1%	3.0%	–	3.07%	[-66%; +88%]	±13.0%
	3	3.75%	–	3.78%	[-73%; +65%]	±15.8%	3.21%	–	3.26%	[-69%; +161%]	±13.7%	3.08%	–	3.13%	[-68%; +75%]	±13.4%	3.01%	–	3.08%	[-68%; +78%]	±13.1%	2.99%	–	3.06%	[-67%; +77%]	±13.1%
	4	3.89%	–	3.93%	[-71%; +92%]	±16.4%	3.21%	–	3.25%	[-69%; +124%]	±13.7%	3.11%	–	3.16%	[-69%; +69%]	±13.4%	3.0%	–	3.06%	[-67%; +71%]	±13.1%	2.97%	–	3.04%	[-65%; +98%]	±12.9%
	5	3.82%	–	3.85%	[-67%; +103%]	±16.1%	3.19%	–	3.24%	[-67%; +91%]	±13.7%	3.07%	–	3.13%	[-67%; +80%]	±13.3%	3.01%	–	3.08%	[-66%; +133%]	±13.1%	2.98%	–	3.06%	[-65%; +101%]	±13.1%
	6	3.73%	–	3.76%	[-71%; +67%]	±15.6%	3.22%	–	3.27%	[-68%; +169%]	±13.8%	3.07%	–	3.13%	[-68%; +67%]	±13.3%	3.01%	–	3.07%	[-66%; +106%]	±13.1%	2.97%	–	3.05%	[-68%; +80%]	±13.0%
	7	3.83%	–	3.86%	[-70%; +79%]	±16.0%	3.2%	–	3.26%	[-69%; +194%]	±13.7%	3.06%	–	3.11%	[-68%; +105%]	±13.3%	3.01%	–	3.07%	[-67%; +93%]	±13.1%	2.98%	–	3.06%	[-66%; +74%]	±13.0%
	8	3.74%	–	3.77%	[-73%; +69%]	±15.8%	3.22%	–	3.27%	[-67%; +72%]	±13.8%	3.07%	–	3.13%	[-65%; +77%]	±13.3%	3.02%	–	3.09%	[-66%; +151%]	±13.2%	2.98%	–	3.05%	[-66%; +83%]	±13.0%
	9	3.7%	–	3.73%	[-73%; +83%]	±15.5%	3.19%	–	3.24%	[-69%; +142%]	±13.8%	3.09%	–	3.15%	[-68%; +77%]	±13.4%	3.04%	–	3.11%	[-66%; +74%]	±13.1%	2.98%	–	3.06%	[-67%; +136%]	±13.1%
	10	3.67%	–	3.71%	[-73%; +100%]	±15.3%	3.2%	–	3.25%	[-69%; +68%]	±13.9%	3.06%	–	3.12%	[-67%; +138%]	±13.3%	2.98%	–	3.05%	[-67%; +95%]	±13.0%	2.98%	–	3.06%	[-67%; +53%]	±13.1%

ing points. So, the validation set error was used simply to choose the best model. The lowest validation set error was achieved when using 10% training set and fifty neurons in the hidden layer. It corresponds to 3.05% test set error.

Although having a large number of hidden neurons in a single network layer is generally not recommended because it provokes overfitting, but due to a large number of samples the MLP did not show signs of overfitting even after few million training iterations. Each model from table 3.2 was trained for exactly one million iterations.

A special application was written in C++ to train a large number neural networks efficiently. It used double precision floating point calculations throughout and implemented gradient calculation in parallel threads allowing efficient utilisation of multicore CPUs specifically for this problem. The learning algorithm was based on backproparation implementation as described by Bishop [68, chapter 4] and the gradient descent with momentum [69]. For example, training the best model took almost 3 hours. See Appendix B for more details of this implementation.

In order to simplify the process of obtaining regression models for other objectives and WebP codec, the same network topology of fifty hidden neurons was used under assumption that it should have more than enough flexibility to predict WebP file size and quality metrics for both codecs. Ten models were trained for every objective and the best network was selected based on the validation set error. Table 3.3 contains testing errors of the best models for file size and two quality metrics: MSSIM and PSNR.

Table 3.3: Test set errors for all JPEG and WebP regression models.

Codec	File size mean absolute percentage error	Mean absolute errors for quality metrics	
		MSSIM	PSNR
JPEG	3.05%	0.008	0.21 dB
WebP	4.75%	0.009	0.22 dB

It is worth to emphasise that in the following sections the reported errors have a different meaning from those in the table. The predicted objectives do not participate in error calculation when models are actually used for image compression with constraints according to the proposed methodology. All subsequent errors in this chapter are for target versus actual values (see fig. 3.1) – i.e. how different the real compression result was from the user request.

3.11 Results for JPEG

This section aims to estimate how useful the proposed method can be in some practical use cases involving JPEG codec. The first one is compressing images *roughly* into the given file size or quality, and the second case is compressing into a fixed *minimum* level of quality.

There are two conditions that guided the choice of these particular scenarios. Firstly, the proposed statistical approach does not always allow to fit the constraints accurately; and this fact leaves for consideration only those tasks that can tolerate some errors. Secondly, each of the scenarios has an alternative technique to compare with, which allow to demonstrate usefulness of the proposed method. Thus, the approximate fit to the given conditions can be compared with the transcoding method, and the minimal desired quality case – with employing a constant quality factor. The following two subsections are dedicated to these cases.

3.11.1 Comparison with JPEG transcoding

The accuracy of the proposed method for compressing images with constraints was evaluated using the best trained models from table 3.3 and compared with the efficiency of the existing JPEG transcoding method by Pigeon et al. [3]. The main difference between compared techniques is the algorithm for finding optimal quality factors. Authors of the transcoding method use simple classifiers while this work utilises machine learning models based on content features and regression functions.

Each method was tested for satisfying two types of constraints: file size and quality level. In the works of Pigeon and Coulombe the MSSIM metric is used to estimate loss of quality after compression. Consequently, the same metric was chosen for current comparison.

Our method consists of three main steps:
– calculating content features from the image data;
– searching for an optimal quality factor using the regression model;
– actual compression run using *libjpeg* functions.

The JPEG transcoding method was represented during comparison by our implementation of a clustered quality factor prediction algorithm described in [3]. The image scaling aspect was omitted because it is not used in this chapter. The transcoding approach requires input images to be encoded in JPEG format already, so in order to make a set of images for this method all 500 000 images in the dataset were pre-compressed with random quality factors $QF_{original}$ between 60 and 95. This procedure simulated a set of real world JPEG images of various

sizes. The dataset was subsequently split into training, validation and test sets in the same proportions that were used during training of the regression functions. Then, following the original description, every image was assigned with a vector ($width$; $height$; $1000\,QF_{original}$), where $width \times height$ is image resolution. Using K-means clustering algorithm the training set was split into 200 clusters assuming these 3-component vectors as coordinates. A standard function *kmeans* from R language was used for this classification. The training process concluded with calculating statistical tables for every cluster, which allow to choose an appropriate quality factor based on the desired compression ratio or MSSIM value. Each table contains an average expected change in compression ratio and quality for various target quality factors. More information about this method can be found in [3] and [24].

Both encoding methods were evaluated in terms of fitting given file size or quality level on the test set images using a single compression run. The target objectives were selected from valid ranges for every test image. The deviations in file size and quality of the images compressed with predicted quality factors were recorded to calculate average errors and compare error distributions.

Upon using our proposed methodology to fit given file size the mean absolute percentage error for the test set was 3.2%, which is expectedly similar to the error of the respective regression model in table 3.3. The corresponding error of the JPEG transcoding method was 10.3% (fig. 3.13), which implies considerably less accurate predictions on average. The situation with compressing into desired quality was similar – the mean absolute error for our machine learning approach was three times smaller than for JPEG transcoding: 0.008 versus 0.023 in MSSIM metric. Hereafter the graphs related to size and quality are implemented in two contrast colours – blue and orange respectively.

Although the average error is an indicative characteristic by itself, it is important to compare error distributions as well to check for possible anomalies and worst case scenarios. Figure 3.14 compares distributions for the transcoding method and the proposed machine learning approach. In the first case the distribution is considerably wider resulting in bigger average error. The proposed feature based methodology allows to reduce the majority of errors close to zero. Considering a small $\pm 5\%$ error allowance for the file size, there are three-quarters of all test images that fit into this interval if using our method and less than a half in case of transcoding.

The distributions of figure 3.14 are not exactly normal. Especially it is related to the proposed method, where the error function used during training was implying a normal distribution of the logarithmised file size errors. Moreover, the predictor rounds down the optimal quality factor by design to cause slight underestimation of

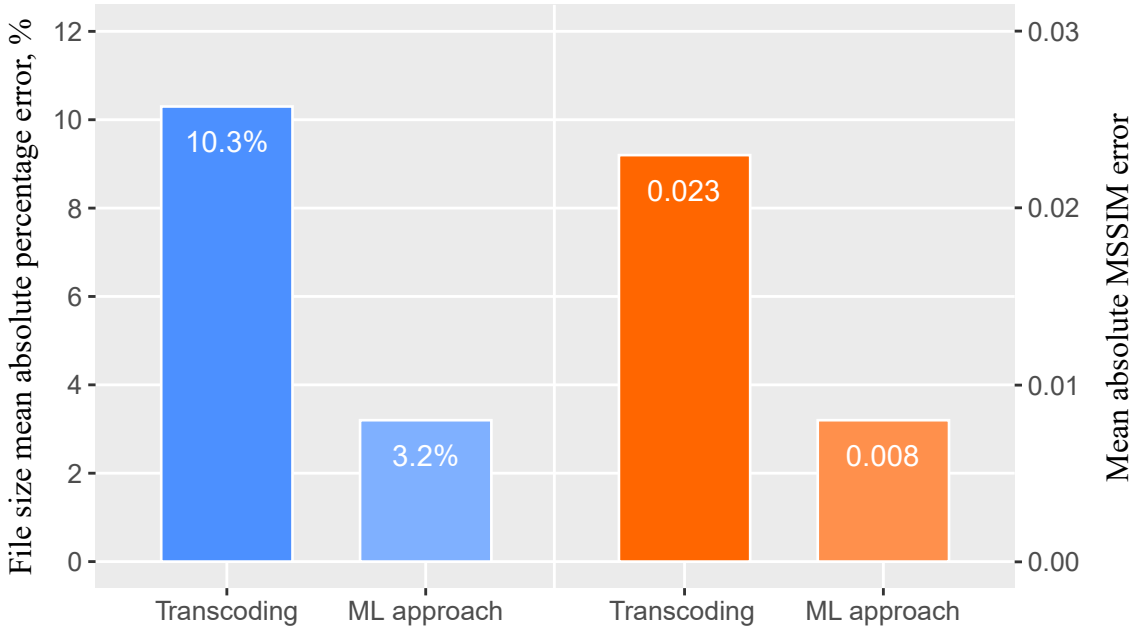


Figure 3.13: Comparison of the proposed machine learning method with JPEG transcoding through average errors between target and actual objectives.

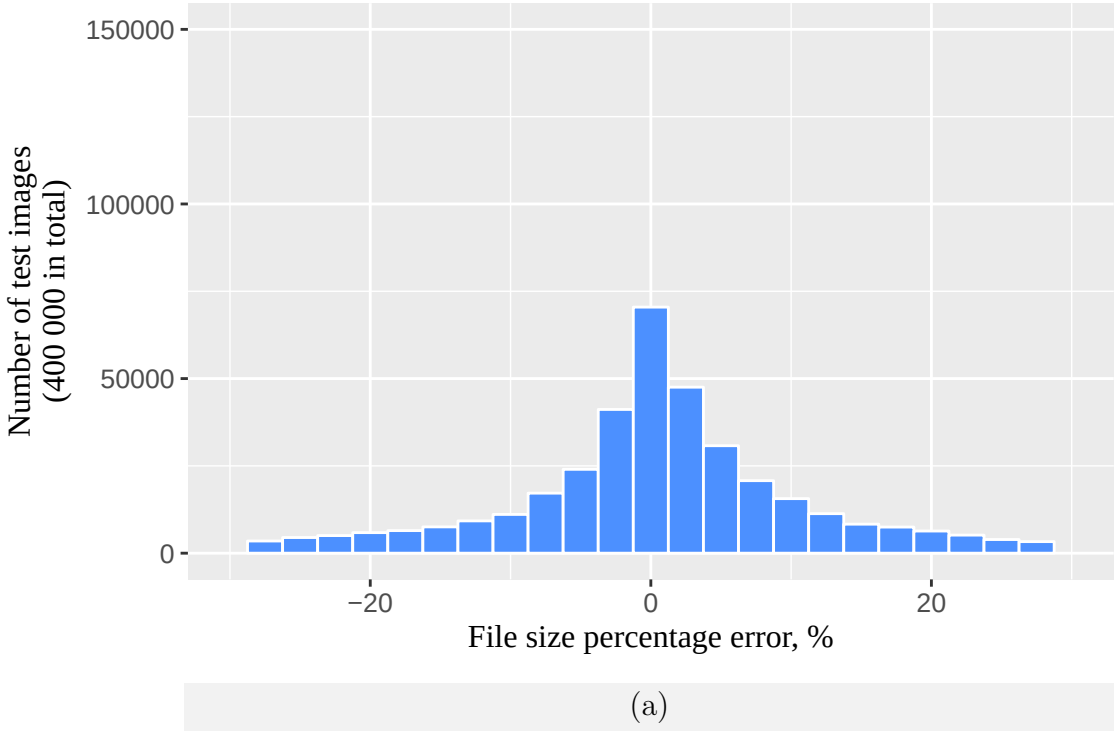
the file size, which is reflected in a skew towards negative percentage errors.

Figure 3.15 compares distributions of the errors in the expected quality. For this objective the proposed methodology also demonstrated superiority over the transcoding approach. The proportions of images with errors in a small interval of ± 0.01 are about the same as in case of the file size distributions – more than 75% of test images fall into ± 0.01 allowance in MSSIM metric using the proposed method, while for transcoding it is less than 50%.

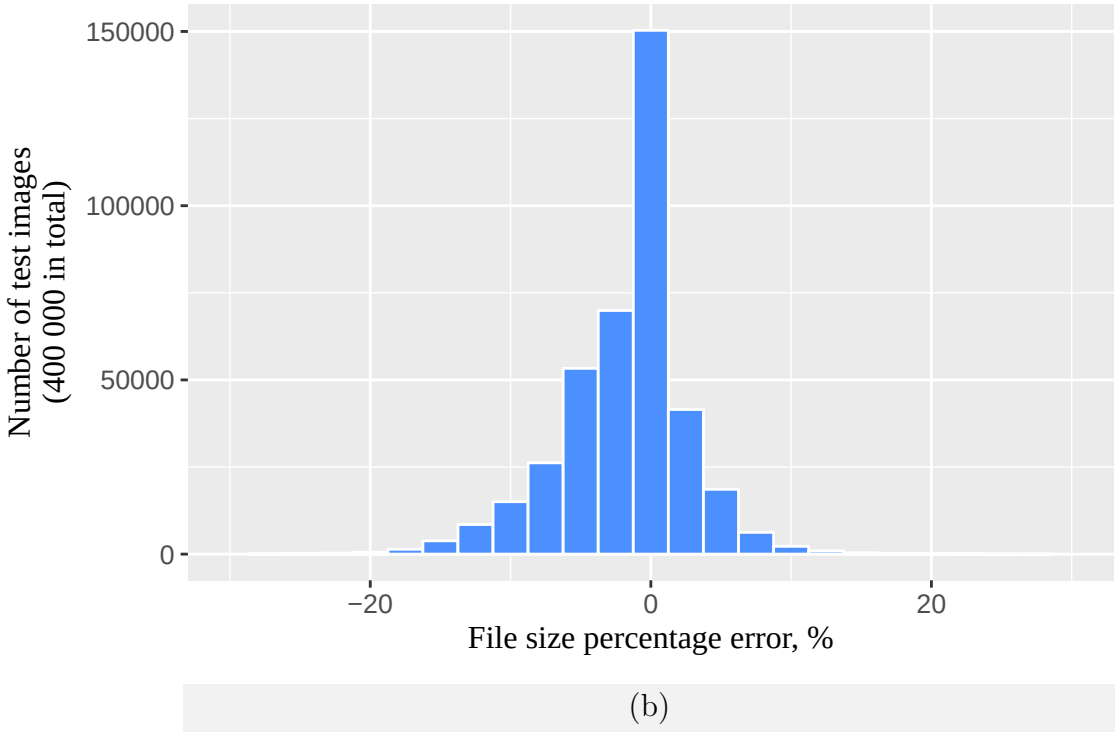
Both predictors seem to overestimate quality mainly due to the fact that for a wide range of quality factors many resulting levels in MSSIM metric have values close to 1.0 (see trends on fig. 3.2), which deviates distribution of target values from normal. However, such property may even be useful in practice.

So, the machine learning approach outperforms the JPEG transcoding method significantly without having specific requirements to the input image format. Apparently, the results show that image content is an important and reliable source of information about "compressability" of the image. Actively utilising it allows to achieve desired compression results with reasonable accuracy using just a single compression.

However, performing analysis of the image content does not come for free. The feature extraction step is faster than actual compression but still takes a consid-

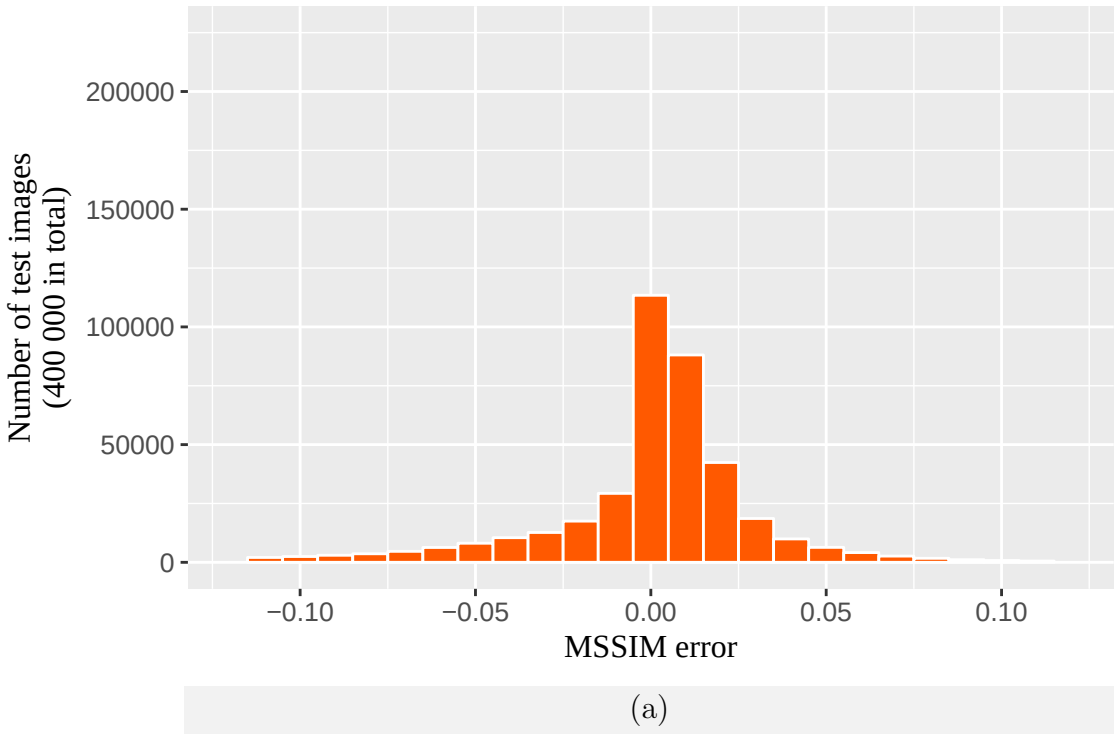


(a)

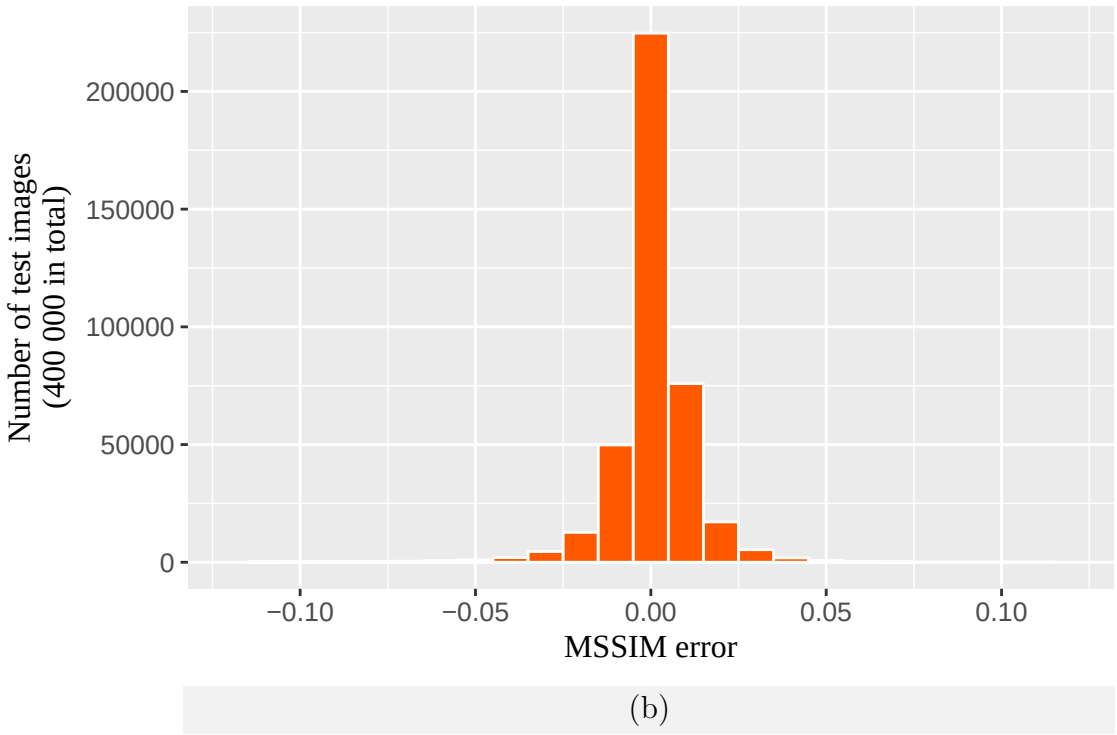


(b)

Figure 3.14: Distributions of the file size percentage errors for transcoding method (a) and proposed machine learning approach (b), where $percentage\_error = \frac{size_{actual} - size_{target}}{size_{target}} \cdot 100\%$.



(a)



(b)

Figure 3.15: Distributions of quality deviations in MSSIM metric for transcoding method (a) and proposed machine learning approach (b), where $quality\_error = quality_{actual} - quality_{target}$.

erable time. Table 3.4 shows how feature extraction is related to actual JPEG compression in terms of time. It usually takes 40 ms if implemented with CPU vector instructions. The JPEG encoding takes three or five times longer (depending on the Huffman codes optimisation) with *libjpeg-turbo* library, which also utilises SIMD (single instruction, multiple data) commands. Time was measured for in-memory operations exclusive of input/output procedures.

Table 3.4: Typical execution time for different operations on a 24 MP image using *libjpeg-turbo* and vector instructions.

Operation	Time, ms
Predicting optimal QF in the JPEG transcoding	< 1
Predicting optimal QF in the proposed method	< 1
Feature extraction in the proposed method	40
Default JPEG encoding/decoding	120
JPEG encoding with Huffman optimization	200

Evaluating a trained model in any of the compared methods takes negligible amount of time. The proposed method needs one feature extraction operation and at least one compression, so processing a 24 MP test image in the conducted experiments required $40 + 200 = 240$ (ms).

In case if input image is already encoded in some format it has to be decompressed prior to extracting content features. Consequently, the transcoding method will require less total processing time if input is in JPEG format. However, it is risky, and not just due to high algorithm errors, but also because the original codec that produced such image could have used different quantization tables making the image incompatible with statistical tables created for standard codec.

To conclude the demonstration of the JPEG models, we investigated whether our method performs equally well for an entire range of considered image sizes. A large number of images dedicated to each individual resolution allows to see how good the model is for compressing with file size or quality constraints at various resolutions. Figure 3.16 shows that the average error when compressing images into given file size gradually increases from about 2.9% for small images to approximately 3.6% for big resolutions. The trend line explicitly shows the direction of error change. This behaviour is expected because larger images can have more possibilities to vary content complexity, which subsequently is harder to predict. Image sizes of 0.24 and 24 megapixels have slightly higher errors on average because these are "edge cases" of the regression model used in this experiment.

The graph in figure 3.17 demonstrates an unexpected trend – average quality error for large images is noticeably lower than for the small ones. This happens

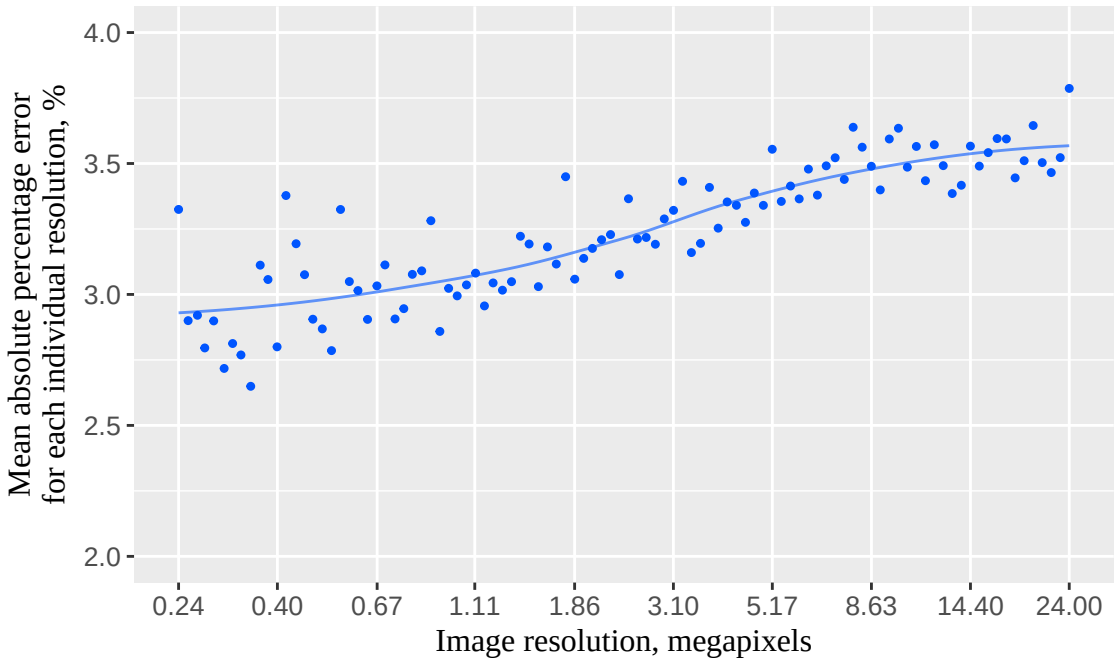


Figure 3.16: Average file size deviation upon compressing into JPEG with target size (using the proposed methodology). Each of 100 points is a mean error for 4000 images of a particular resolution.

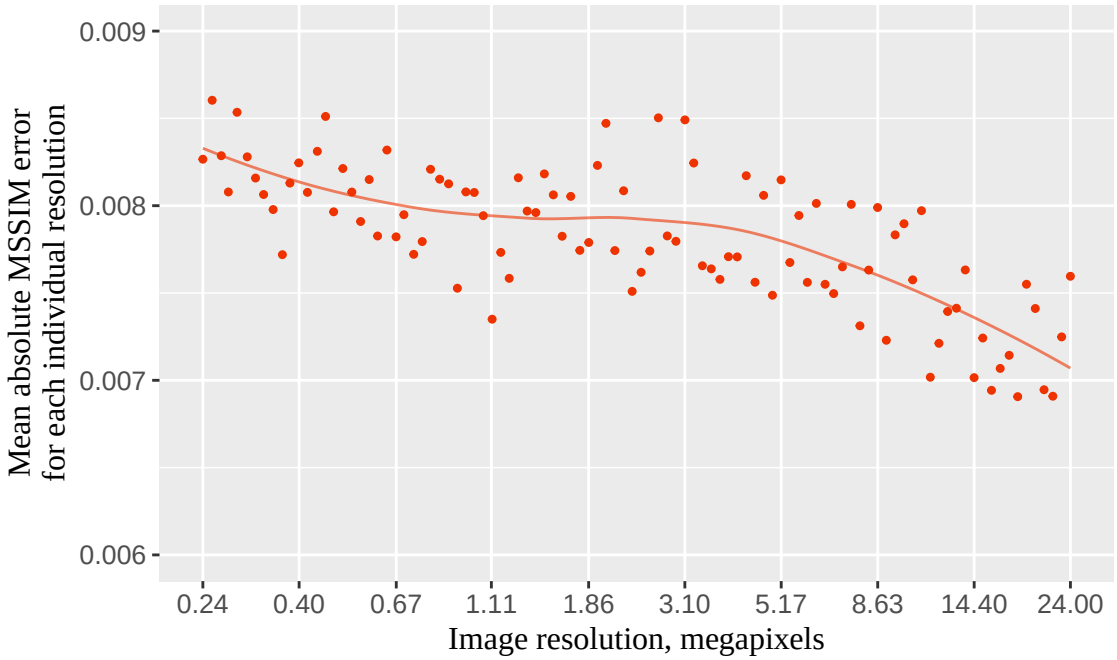


Figure 3.17: Average MSSIM deviation upon compressing into JPEG with given level of quality.

because of the specific of calculating mean SSIM metric using a fixed 11×11 pixels window. Thus, with increasing resolution the amount of significant details in each small area of the image decreases being replaced by some noise, which makes it slightly easier to predict quality degradation at the pixel level.

3.11.2 Comparison with a constant quality factor

In some practical scenarios using a fixed value of quality factor for JPEG compression may seem a reasonable choice even knowing that it does not guarantee the same resulting quality.

However, if the use case requires a particular minimal level of quality for compressed images, then using the proposed machine learning method can give better results than a constant quality factor. This can be useful, for example, in mobile devices assuming that the user wants to keep visual quality of the produced photographs at a certain level.

For comparison purpose the JPEG quality factor was set to 90 – a typical value implying a high quality image. A thousand images of 4980×3320 pixels were selected from the testing set to be used in this comparison. The size of approximately 16 megapixels is a common photo resolution today.

This experiment was conducted under assumption that 95% of compressed images should be above the same fixed level of quality in MSSIM metric. Using statistical models it is not possible to get a desired result in 100% of cases, at least with a single compression run. In order to improve accuracy of the method several recompressions may be necessary for some images. However, such strategies were not investigated in this research because they depend principally on the specifics and priorities of the practical use cases, which in addition may involve time constraints.

Firstly, the chosen subset of images was compressed with quality factor 90 resulting in the average quality 0.978 and average file size 3.12 MB per image. The 5^{th} percentile threshold was at the level of 0.942, i.e. 95% of compressed images had at least this quality (fig. 3.18).

Secondly, we calculated 90% confidence interval (-5% for each side of the distribution) for JPEG MSSIM regression model to be used for predictions. It appeared to be approximately ± 0.018. Aiming for the same threshold 0.942 the equivalent mean quality value for the new distribution should be $0.942 + 0.018 = 0.96$.

Thirdly, targeting MSSIM quality level of 0.96 the selected images were compressed using the proposed methodology. As a result, a narrower distribution was obtained with mean 0.96 and 95% of images above 0.95 threshold, which is even

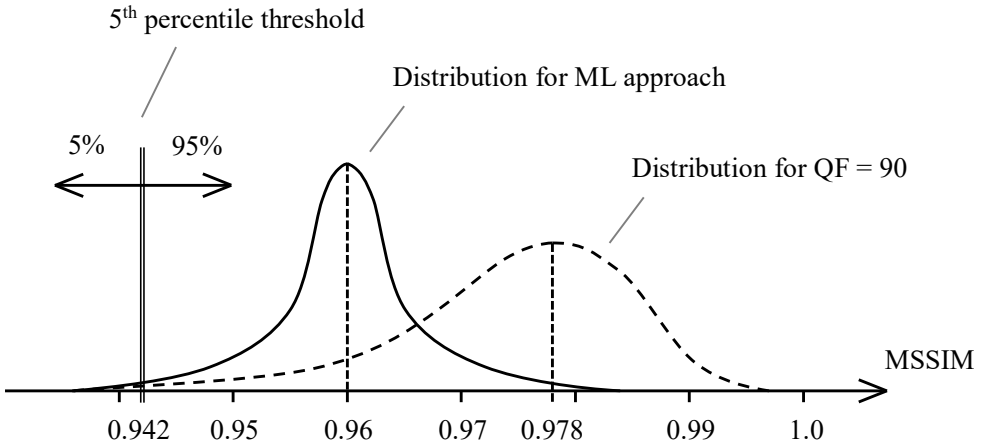


Figure 3.18: Sketch of quality distributions for 16 MP images compressed with a constant quality factor and with the proposed method while keeping the same minimal quality for 95% of images.

better than planned 0.942 because the used regression model tends to overestimate quality. A significant advantage of this approach is the smaller average file size of 1.94 MB per image, which is about 38% less than 3.12 MB in the case of $QF = 90$.

This example of minimising file size while maintaining the quality demonstrates how proposed machine learning approach allows to balance between target objectives for JPEG compression.

3.12 Results for WebP

Google's WebP is a relatively new format that can be used for compressing photographic images. It is often considered as a modern alternative to JPEG. This research uses WebP codec alongside JPEG to demonstrate universality of the proposed methodology.

The previous section presented various aspects of applying our machine learning system to the JPEG codec. It explained the mechanism of obtaining a desired result and evaluated the expected accuracy. Although none of the existing JPEG codecs employs any strategy to encode images into given file size or quality, the WebP codec contains a built-in algorithm to compress images with constraints through multiple partial recompressions – a relatively obvious and reliable choice assuming that the user has enough time to conduct such procedure. This allows to perform a direct comparison of the proposed method with an existing alternative solution. This section does not investigate all possible use cases for WebP, but considers some real world scenarios of obtaining images with particular size or quality.

The WebP codec supports option *-pass <int>* that allows to control the accuracy of fitting the target objectives by specifying up to ten partial compression runs that converge to the desired result. The file size and quality level can be set with parameters *-size <int>* and *-psnr <float>* respectively according to the WebP user manual [63]. The codec uses PSNR metric to specify target level of quality. Consequently, this section employs PSNR for comparison purposes too.

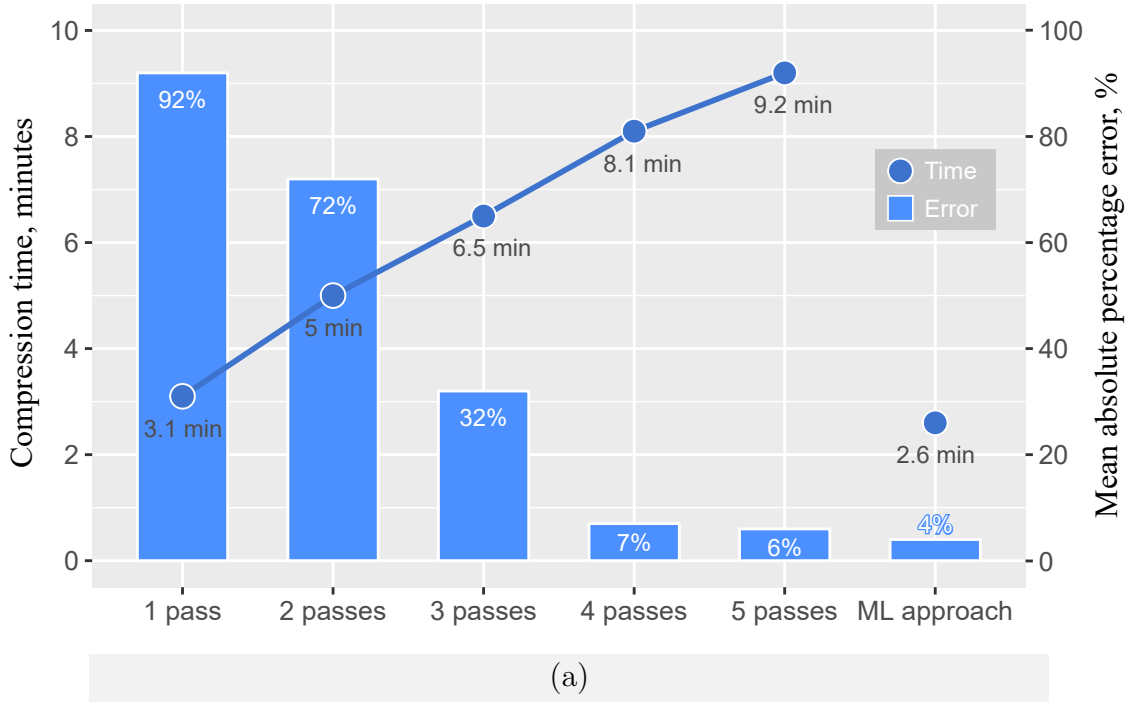
The same dataset of 500 000 uncompressed images was used for experiments with WebP codec, so the content image features also remained constant since experiments with JPEG. The regression models for predicting file size and quality were trained identically to those for JPEG.

Considering the fact that multiple recompressions in the WebP codec substantially increase total encoding time, it was taken into account during comparison. To accurately measure compression time the test images were processed sequentially and independently in a single CPU thread, while recording only in-memory compression time. As such procedure was hardly practical to perform for all 400 000 test images, only the smallest (600×400 px) and the largest (6000×4000 px) resolutions were tested under assumption that similar results can be obtained for any intermediate image resolution. So, only 8000 test images were used in the comparison – 4000 small and 4000 large.

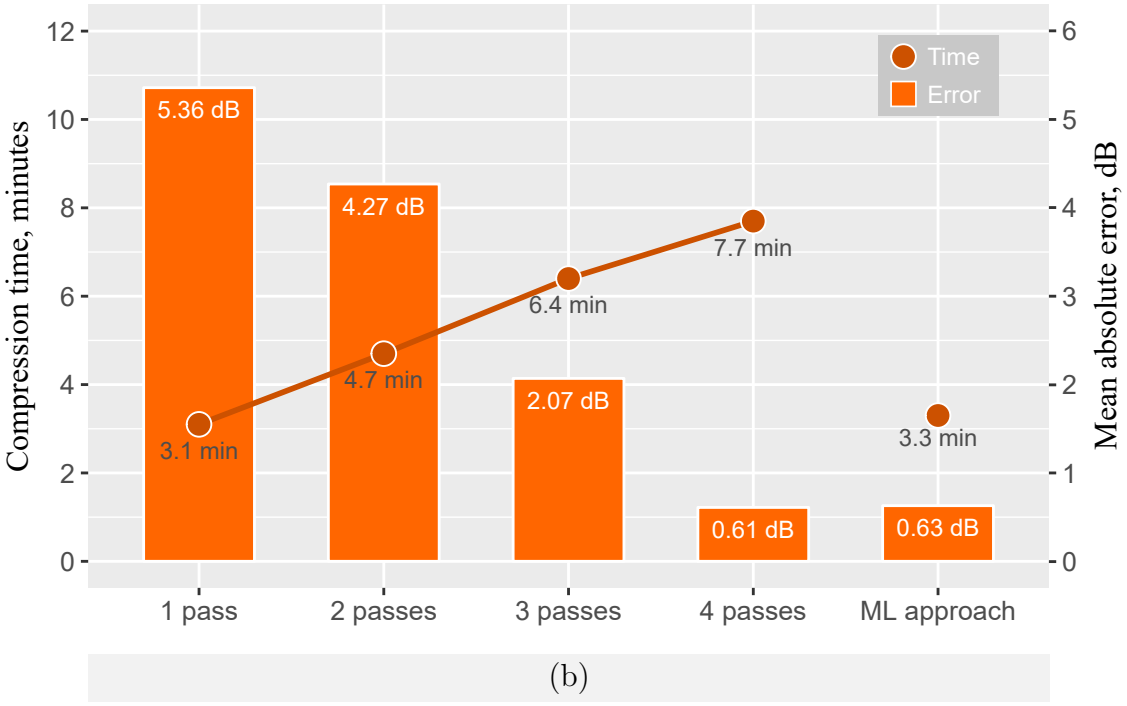
After randomly choosing reasonable target values of the resulting size and quality for all selected test images, they were compressed aiming for each of these constraints separately and using both: the proposed method and the WebP codec, in which up to five passes were specified.

Figure 3.19 compares mean errors and total compression time between the proposed approach and different numbers of WebP codec passes for 0.24 MP images only. Our method on average needs approximately the same amount of time as the single-pass WebP encoding but demonstrates much smaller errors between target and actual compression results, when aiming for either file size or quality. The average error achieved by the proposed method is approximately at the level of four or five passes in the WebP codec while the compression time is substantially smaller. This experiment proves that using the proposed methodology it is possible to find optimal compression parameters much faster, and consequently in a more energy efficient way, than by performing image recompressions.

Figure 3.20 looks into file size errors in detail. It compares size error distributions after several encoding passes as well as a single compression with our method using the same 4000 small 0.24 MP images as an example test set. One or two passes in WebP codec look more like consequences of a random guess rather than a



(a)



(b)

Figure 3.19: Comparison of the average file size percentage errors (a), PSNR quality deviations (b) and the total compression time for multipass WebP encoding versus the proposed machine learning method using 4000 **small** test images of 0.24 MP.

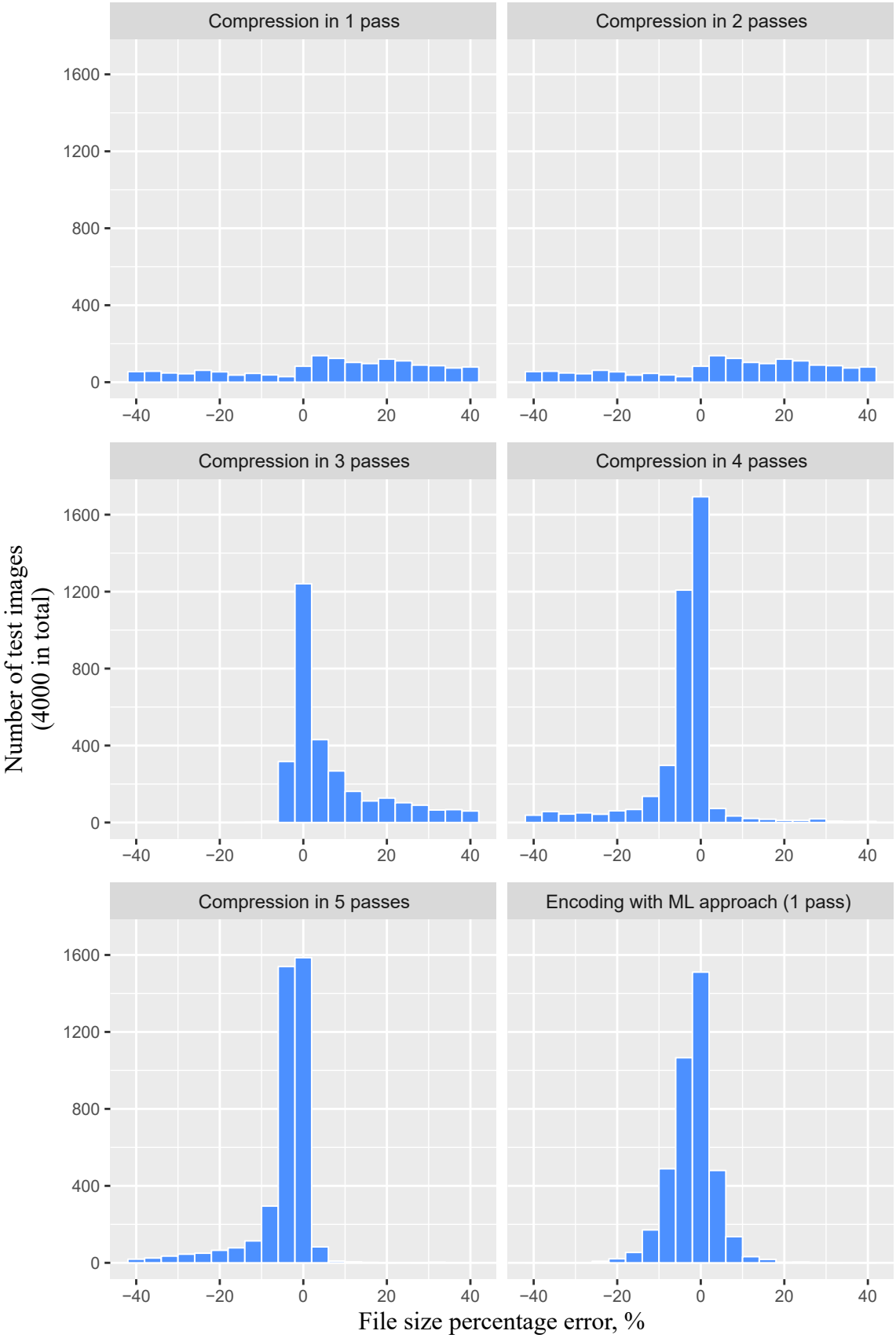


Figure 3.20: Distributions of the file size percentage errors for multipass WebP encoding and proposed machine learning approach.

reasonable choice of compression parameters. Their distributions are so wide that the majority of bars had to be clipped to keep the same range of values for the graphs. After five passes the situation with WebP codec errors improves dramatically. It is even better than error distribution from our method in a sense that there is much less cases with file size overestimation. The last two distributions show that both compared methods tend to underestimate file size. However, 5-pass WebP compression has much longer tail of large amplitude negative errors. Conversely, the proposed machine learning method has error distribution close to normal, which was intended by design.

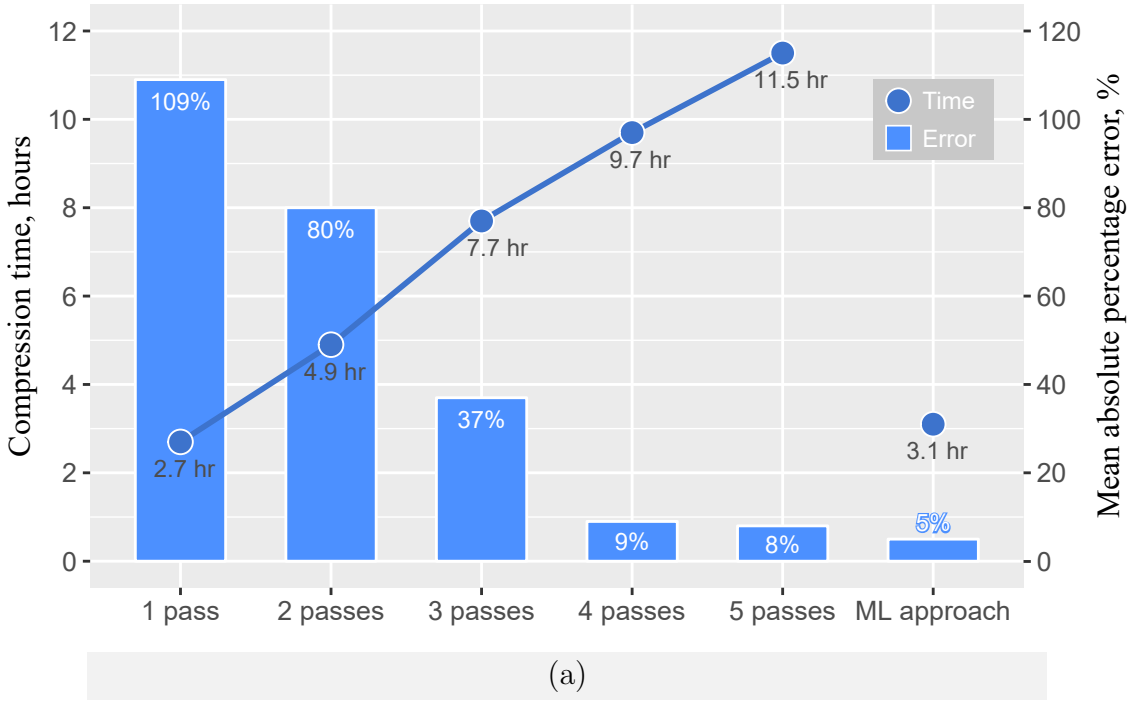
Therefore, for small images our method demonstrated superiority over existing multipass WebP compression in many aspects without any modifications to the codec. This fact supports the Predictable Compression Hypothesis formulated previously in the introduction.

Large images (24 MP) were tested separately due to much bigger compression time in order to avoid distorted time statistics. Figure 3.21 shows very similar trends in comparison of size and quality errors to those for small images. The only difference is that total compression time for 4000 images is measured in hours instead of minutes. Such closely correlated trends in average errors and compression time for images of different resolutions suggest that the same picture remains across a wide range of image resolutions when the proposed method is applied to WebP codec.

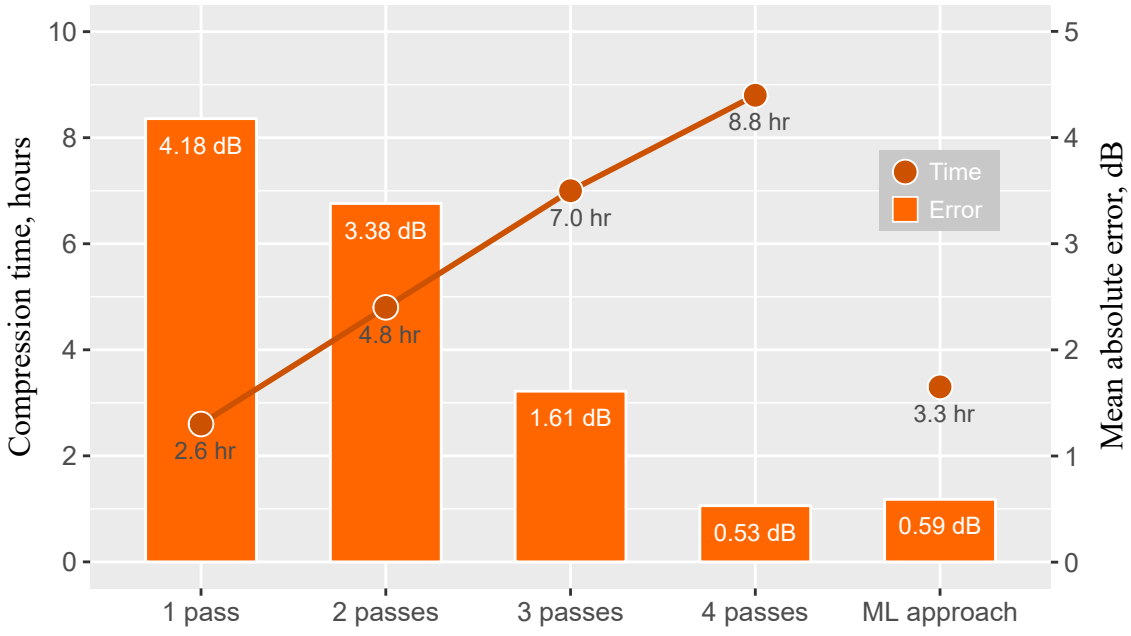
There are some interesting trends in the average errors that can be observed from the graphs of figures 3.19 and 3.21. The average file size errors for large images are bigger than for small ones in every compression scenario, while the average PSNR deviations demonstrate a completely opposite pattern. The quality level for large images is easier to predict. This phenomenon is very similar to the trends on the graphs 3.16 and 3.17 from the previous section related to the JPEG models. This cannot be due to a statistical error, which one may assume by comparing deviations of 0.63 dB and 0.59 dB from our method applied to small and large images respectively, because every multipass compression case demonstrates the same behaviour.

The reason why 24 MP images are easier to compress into given quality may be related not so much to the way the error metric is calculated (as it was assumed previously) because PSNR is computed in a different way than MSSIM, but mainly down to the actual compression algorithms.

Assuming that high resolution photographs have a considerable amount of evenly distributed noise at the pixel level implied by the physical properties of the camera, and the image compression algorithm works as a low-pass filter, then



(a)



(b)

Figure 3.21: Comparison of the average file size percentage errors (a), PSNR quality deviations (b) and the total compression time for multipass WebP encoding versus the proposed machine learning method using 4000 **large** test images of 24 MP.

it changes the distribution of such noise in a predictable way. However, small resolution images tend to have more high- and low-contrast fragments (like sharp edges and smooth colour areas) unevenly distributed across the image, which results in more diverse distributions of the introduced errors in different image fragments upon applying a low-pass filter. Therefore it is harder to estimate the average distribution of the introduced noise for small images.

There is another peculiarity about the compression time difference between the proposed method and the single-pass encoding. It is not obvious why there are both positive and negative deviations in average compression time between these scenarios. Our method can be slightly faster (fig. 3.19a) or slower (fig. 3.19b) than 1-pass compression. The reason is that WebP compression with different quality factors requires different amount of time. This is not related to overhead for the feature extraction procedure in our system because its time is tiny in comparison with actual WebP compression. A typical encoding time for 24 MP image according to figure 3.22 is about 2 s, while feature extraction requires constant 40 ms, which is fifty times smaller, so the contribution of the feature extraction stage into the total processing time can be considered negligibly small – one extra second per minute of actual compression time.

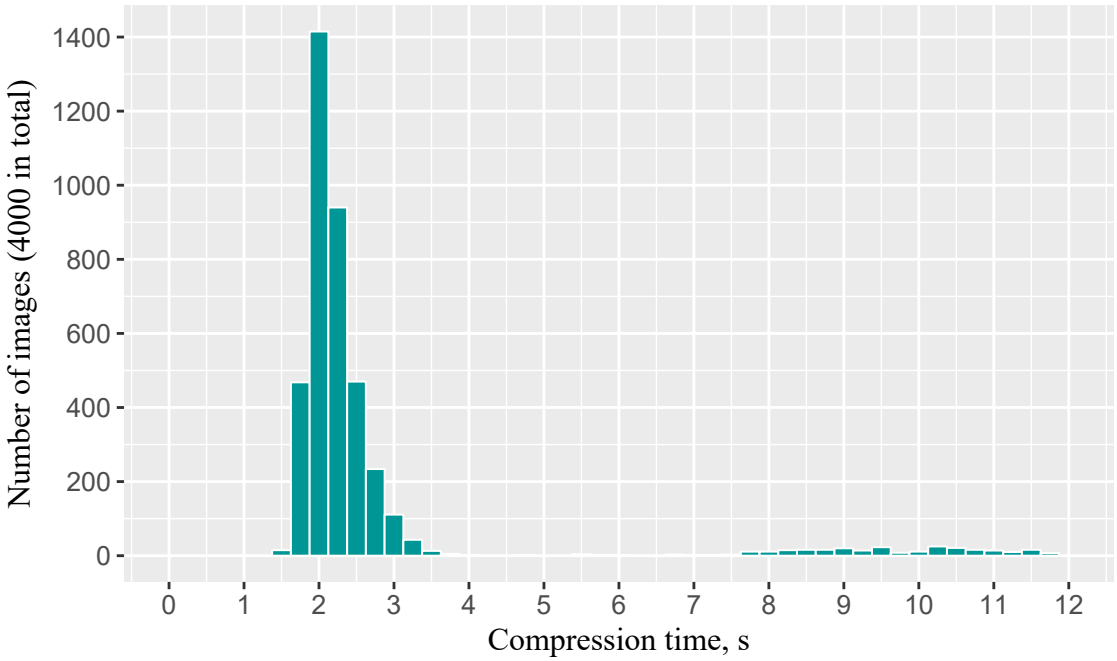


Figure 3.22: WebP compression time distribution for 24 MP images using the machine learning method (exclusive of feature extraction).

Images encoded with large quality factors (90-100) require much longer compression time (around 10 s) and result in considerably bigger files.

3.13 Discussion

This chapter proved the idea of a universal methodology that allows to make reasonably accurate estimations of the compression outcome, which in turn can be used to find optimal quality factor for different image codecs. The proposed approach is flexible not just in terms of the target image format but also can be used with different sets of features and machine learning techniques.

Looking at the error comparison graphs related to JPEG and WebP codecs it is easy to notice how closely correlated the average size and quality error trends are despite the fact that they represent distinct characteristics measured in different units. Nevertheless, the compressed file size in general seems to be more difficult to predict than the resulting quality level. Although there is no explicit evidence to support such conclusion, it looks reasonable considering the following point of view – the average deviation of 0.6 dB in target quality (according to fig. 3.21 using the proposed method) may indicate a small difference hardly noticeable even upon close manual inspection, while the "equivalent" 5% deviation in file size is trivial to see and can be too big for some applications. Also, the lack of a reliable objective quality metric makes measurements of quality degradation somewhat less decisive compared to absolute certainty of a compression ratio. Considering similar trends in other compared methods, it is possible to make a supposition that fitting a file size constraint is a fundamentally more difficult task in comparison with a quality level.

Among other general observations it is worth to mention slightly higher errors of the regression models for WebP codec in comparison with those for JPEG. It is likely to be a consequence of the WebP's more intricate and slow compression algorithm that involves a search for similar neighbour fragments and an arithmetic compression – more efficient version of entropy reduction than Huffman codes. Anyway, this work does not aim for the best possible accuracy in practice.

The application domain of the proposed concept of predicting compression result properties before actual compression is not limited to images. The next chapter investigates how it can be used for more complex type of multimedia data – videos.

3.14 ACACIA image compression tool

Using the techniques described in this chapter we created ACACIA [70] – an application that allows user to compress images into JPEG and WebP formats targeting explicitly specified level of quality or compressed file size with minimal overhead for computational time.

ACACIA (Advanced Content-Adaptive Compressor of ImAges) is a demonstrative software that supports GUI mode (fig. 3.23) as well as command line interface.

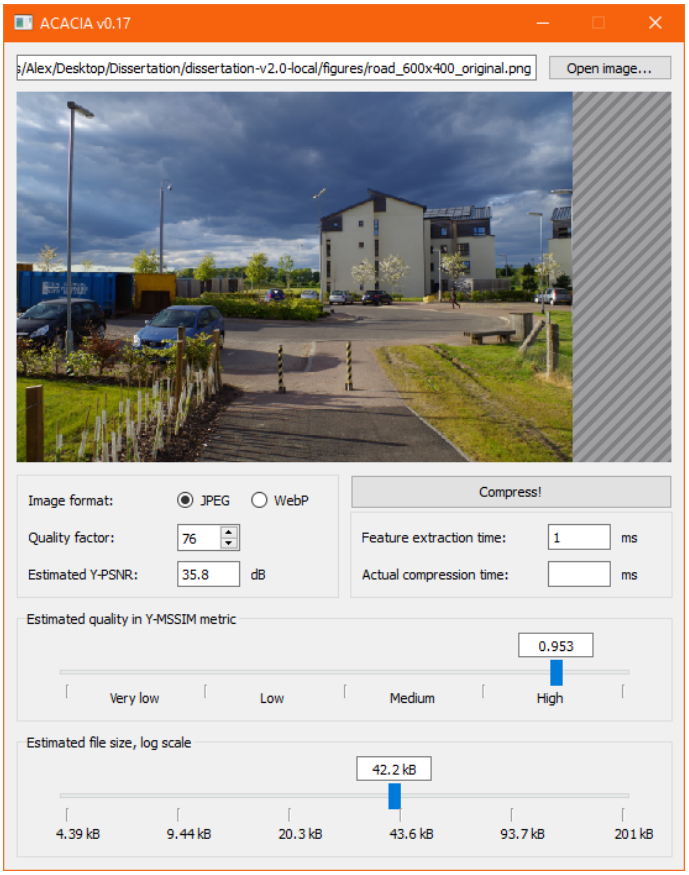


Figure 3.23: ACACIA interface in Windows.

Upon loading an image into the program, the feature extraction process is performed. The feature vector is stored in memory. There are two sliders for adjusting quality and file size. When user moves one of the sliders, the closest quality factor value is found using a regression model built into the program. Then by substituting values into another regression model for the free objective, the prediction is made for the second slider, and it moves to the new position automatically indicating the connection between the two objectives. The predictions takes less than a milisecond, so the program reacts to user input in real time.

The quality slider displays values in the MSSIM metric in the range $[0.6; 1.0]$ equally split onto four intervals approximately corresponding to the 'high', 'meduim', 'low' and 'very low' levels of quality. The file size slider has a logarithmic scale between the smallest and the largest estimated compressed file size.

The 'Compress!' button allows user to choose the location for the compressed image file. ACACIA also reports the time spent on the library function call, which performs actual compression.

Despite that fact that MSSIM metric was used only for JPEG and PSNR was tested only for WebP, the program uses statistical models for both quality metrics in either target format.

3.15 Summary

This chapter introduced a general methodology for compressing images with explicitly specified target objectives like file size in bytes or quality level in MSSIM or PSNR metrics. The methodology is based on the machine learning models that predict the outcome of compression. It allows to avoid multiple image recompressions to get a desired result, thus saving processing time and energy.

In order to represent the image entropy in a compact and descriptive form suitable for training regression models, a set of ten content features was designed from scratch. The concept is based on estimating the amount of the high-frequency image components, but in a computationally efficient way. As a result, the features correlate with the compressibility of an image and require only few milliseconds to calculate even for a large photograph. Also the content features were devised with resolution independence in mind.

The universality of the proposed methodology was demonstrated using two image codecs for popular compression standards: JPEG and WebP. It is pertinent to emphasise that the algorithms used in this chaper did not require any modifications to the considered image compressors. However, by embedding the introduced techniques into the codecs it is reasonable to expect the reduction in the total computational time mainly due to merging external and internal colour space conversions into one.

The tests for JPEG involved a comparison with an existing transcoding method that recompresses JPEG images into lower quality. The compression with both target quality and size using the proposed methodology demonstrated a considerably better accuracy of fitting the constraints in comparison with the transcoding method by Pigeon et al. [26]. This result was expected because the transcoding approach does not use information about image content, only resolution. As a consequence, the feature extraction stage is not required for JPEG transcoding, which makes it slightly faster than the proposed technique. However, the main advantage of the proposed method is that its area of application is not limited to a single image codec.

The experiments with WebP codec explicitly demonstrated better performance of the new system in comparison with the scenario involving multiple recompressions. The WebP codec has built-in options to encode images into given size or quality using

several partial compressions. This functionality allowed a direct comparison of the total compression time required to achieve a certain proximity of the desired result. The WebP codec takes approximately three time longer to get to the same level of error in comparison with the machine learning approach. Looking from the side of accuracy, the proposed method has the average error comparable with 4–5 passes of WebP codec but using only a single compression.

Due to the fact that WebP codec is considerably slower than JPEG (~2000 ms and 120-200 ms respectively), the overhead for feature extraction and prediction is negligibly small in comparison with a single WebP compressor run. Therefore, the proposed methodology can be considerably more useful for new and emerging image compression formats, which are more complex than JPEG.

3.16 List of Contributions

The following results were obtained during investigation into image compression optimisation:

- a low-complexity technique for analysing uncompressed content of the images and predicting the resulting size and quality before compression;
- an accurate method for one-shot compression with target file size or quality level based on the prediction technique;
- the proposed system considerably outperforms the existing JPEG transcoding method in accuracy for either size or quality;
- the proposed method allows to obtain a narrower quality distribution for a set of images in comparison with using a constant JPEG quantizer;
- the method of predictable single compression substantially outperforms multipass encoding in therms of processing time;
- the result of WebP compression is more difficult to predict than for the JPEG codec;
- the compressed file size becomes more difficult to predict with increasing image resolution, while the quality in MSSIM metric is on the contrary easier to estimate for large images;
- a standalone tool ACACIA designed as a proof of concept for two different image codecs.

Chapter 4

Dynamic Resolution and Video Compression with Target Quality

This chapter is dedicated to optimisations in video compression. It is a broad area, so this research focuses on a particular aspect of balancing the quality and compression ratio between different parts of the video.

The research also addresses the challenge of compressing the videos with a target quality without running multiple recompressions. This issue is considered alongside optimisation, which takes priority in an attempt to decrease file size or compression time. Shifting focus towards video optimisation builds and expands on the work of the previous chapter. Considering real-world optimisation problems is a more interesting scenario than predicting video compression results. For example, adding dynamic resolution increases optimisation possibilities in video compression. It is the combined compression outcome and dynamic resolution prediction methodology that makes this novel research.

The problem of quality balancing is investigated for x265 video codec using the machine learning methodology from the previous chapter adopted to video compression. Although the large number of encoding options provided by video codec adds complexity to the problem of predicting compression outcome, it allows the discovery of optimal parameter combinations, which are unobvious and otherwise obtainable through a time consuming search with actual compressions alone. The result is a better tradeoff between quality and file size for the whole video.

In order to increase the number of possibilities for optimisation, the segments inside a video are allowed to have variable resolution by design. Is is assumed that the display resolution remains constant, which means that some frames must be upscaled upon playback. Although modern video codecs work only with frames of constant resolution, it is possible to compress and store different parts separately as

well as play them in a sequential order like a normal video.

Besides a variable resolution, which can be considered as a modification to the video container formats, this research does not introduce any changes to the video codec itself.

4.1 Levels of Quality Balancing

Aiming for a constant quality for entire video during compression is a common use case. However, video is a complex data structure that can consist of thousands of frames, which are technically separate images. Consequently, the problem of quality balancing is not trivial, especially if the user has provided other constraints like the target file size.

Considering an uncompressed video, there are four structural levels, elements of which can be subjects to optimisation:

– entire video;

– video segments;

– separate frames;

– blocks in a frame.

The content complexity of the components at each level is different, which means that by varying compression options it is possible to keep the resulting quality at a certain level.

It is often user's responsibility to select compression parameters for the entire video based on its complexity and desired outcome characteristics. This is a difficult task, so video codecs usually provide options to control the resulting file size. The constant bitrate, for example, ensures a fixed compression ratio but causes uneven quality degradation across the video. In order to produce a given total file size some video codecs employ a strategy similar to WebP image codec – multipass compression, which requires much longer compression time. As for the desired quality, there is no explicit targeting for quality metrics implemented in any popular video codec. Modern compressors by default try to maintain the same quality level by adjusting the DCT quantizers according to the frames complexity. But in any case, it is done with relation to a reference quantizer value provided by user.

All video codecs split videos of sufficient length into segments separated by key frames (or I-frames), which are encoded independently from other frames and used to facilitate navigation in the compressed video stream and increase its error robustness. At this level the codec is primarily responsible for balancing quality across different segments. If the compressor provides options to set content specific preferences

then the user can indirectly influence this process. This chapter investigates quality balancing between video segments. In practice such a procedure can be done without modifications to the video codec, while almost any attempt to alter compression processes at lower levels will need changes in the codec algorithm (but not necessarily the video format).

Individual frames in the video may require quality balancing due to the fact that different images compressed with the same quantizer result in technically different quality levels (see example in the introduction on fig. 1.1). The same problem is related to the blocks inside each frame. Modern video codecs like VP9, x264 and x265 support adaptive quantization – i.e. independent quantizer values for each macroblock in a frame. The codecs use heuristics based on the content complexity and potential motion range of the blocks to determine optimal quantizer. Performing quality balancing at such low level requires not just deep understanding of the encoding process, but also reliable quality metrics, which can justify even minor changes.

Current research considers image and video codecs more like black boxes aiming to distant itself from the details of particular implementations. Consequently, the most suitable scale, at which quality balancing can be performed, is the level of separate video segments. The distinct feature of this work in comparison with the built-in quality balancing is the dynamic adjustment of the segment resolution during compression process. It can be considered as an extra mean of quality control. In addition, such problem was not previously investigated in the literature.

The proposed optimisation arises from the fact that different parts of the video, despite being related in terms of meaning or depicted objects, have different content at the pixel level and therefore require an individual approach from the video codec perspective. Adding variable resolution to the problem prevents the video codec from performing quality balancing between segments. However, the low-level balancing between frames and their blocks is still done by the video codec using its heuristics.

4.2 Video Quality Measuring Program

The main focus of this chapter is on the problems related to the compressed video quality. Therefore, it was important to prepare appropriate tools for measuring quality degradation in various video compression scenarios.

According to Kotevski et al. [42] objective quality metrics like PSNR and SSIM have a range of problems when applied to videos. In particular, some motion artefacts visible to a human are difficult to capture with formulas. Therefore, human

viewers are often asked to rank a dataset of compressed videos against the original versions thus allowing to calculate a mean opinion score. This method however has its own drawbacks. For example, in order to create a reliable statistical model, multiple measurements should be made for a single video using different compression options, but users often cannot detect minor quality changes between similar parameter combinations. A substantial amount of statistics is needed to detect the regularities. Such an approach may require thousands of viewers and thousands of videos to mark, which often makes it impractical.

For example, consider a dataset of 2500 video segments used in this chapter experiments, where every video was compressed with 300 parameter combinations. It required 750 000 quality evaluations in total. Such procedure was time consuming even with a relatively simple MSSIM metric implemented using vector and multi-threaded parallelism. Relying on the human judgment would make these experiments impossible and require a completely different approach.

The MSSIM metric was used to calculate quality levels in all video compression experiments. Despite the fact that this is not a perfect option, it was used mainly to demonstrate the working methodology, which is independent from any particular metric by design.

In this thesis the MSSIM between compressed and original videos is calculated as an average MSSIM of all decoded frames with respect to the corresponding original frames.

For video experiments this metric was chosen due to several reasons. Firstly, it is widely used in the literature and, in particular, in the related research. Secondly, it provides a reasonable balance between the computational (and implementation) complexity and the correlation with subjective perception. And finally, the specific of calculating mean SSIM metric using separate frames allows to instantly obtain the average quality of a long video by knowing the average quality levels of its individual segments. The experiments on optimising video compression included assembling videos from segments with separate values of MSSIM. So, this property added flexibility to the experiment planning and decreased the experimental time.

However, some tests required to calculate the frame quality distribution and investigate per-frame quality correlation between different compression scenarios. Therefore, the program for measuring video quality was extended with extra functionality – printing MSSIM for every frame in addition to the average. It is a relatively unique feature in comparison with the existing tools for video quality calculation. Here is an example of the program output:

```
./y4m-psnr-mssim-fp32 original.y4m decoded.y4m -nopsnr -notime
                                              -print-per-frame -threads 6
Frame Y-MSSIM
    1 0.9615719
    2 0.9586021
    3 0.9563300
    4 0.9555718
    5 0.9553111
[...]
average 0.9575674
```

Internally the programs for calculating MSSIM metric for images and videos used exactly the same subroutine that processes two arrays of image pixel data. Taking into account a fixed 11×11 window size and predefined weighting kernel, the SSIM subroutine was optimised with AVX/AVX2 instructions. However, in case of videos this metric was still quite expensive in therms of computational time. So, the program was improved by implementing parallel processing of the corresponding individual frames. This approach allowed to efficiently utilise multicore CPUs for video quality evaluation and considerably speed up the experiments.

It is also important to mention that technically all experimental videos were stored in the YC_bC_r colour space according to the ITU-R BT.709 standard, which has limited range for luminance and chrominance components. For example, the values in the luminance channel Y have a recommended range of brightness levels [16; 235], which is different from a standard dynamic range of [0; 255] used for images. Formally, this should affect the dynamic range $L = 255$ in the SSIM formula. However, in the decompressed videos many pixel values were out of the recommended range. Such phenomenon is called overexposure or underexposure depending on a particular value and is formally allowed by the standard. The only brightless levels that are strictly forbidden are 0 and 255. Assuming that over- and underexposure has a minimal influence on the result, the dynamic range in the MSSIM formula was left at the default value 255 for simplicity.

4.3 External Tools

Couple of external applications were employed for the experiments with video compression, such as an opensource x265 video codec and the FFmpeg tool for resizing videos and decompressing them into the raw format.

4.3.1 FFmpeg and FFprobe tools

FFmpeg is a popular program for video and audio conversion [71]. It aggregates different libraries for encoding and decoding a wide range of image, video and audio formats. FFmpeg supports various video containers like AVI, MP4, MKV and WEBM. In this research the program was used mainly as a fast tool for changing video resolution and for decoding compressed video streams in the lossless *.y4m format. Actual video compression was performed with a standalone version of the x265 video codec described in the next section.

In order to test the concept of dynamic resolution all video segments were resized to smaller resolutions before compression using a simple command like the following:

```
./ffmpeg -i input.y4m -s 960x540 output.y4m
```

By default FFmpeg uses bicubic interpolation for resampling operations. It is not explicitly stated in the documentation but can be found in the source code [72, line 1229]. Investigating the resampling methods is beyond the scope of this research, but it is reasonable to employ bicubic interpolation for videos because it is suitable for natural images, computationally cheap and widely used in practice.

So, FFmpeg was used during experiments to decrease resolution before compression and to decompress video files to the raw format with restored resolution for subsequent quality calculation. Decoding H.265 format with resampling was done using the following syntax:

```
./ffmpeg -i input.h265 -s 1920x1080 output.y4m
```

It was empirically established that the decoded video obtained with FFmpeg is identical to the output stream of the reference H.265 decoder [73]. The advantage of the FFmpeg is the multithreaded processing enabled by default, which helped to reduce the experimental time.

In the video experiments there was a need to split the long videos onto individual segments separated by I-frames. The optimal positions were automatically determined by x265 video codec during a test compression run. Then the FFprobe tool from the FFmpeg library was used to extract the key frame positions from the compressed video stream:

```
./ffprobe -i video.h265 -select_streams v -show_frames
                        -show_entries frame=pict_type -of csv
```

4.3.2 Introduction to x265

x265 is one of the best modern video codecs in terms of quality to size ratio according to multiple comparisons [13, 14, 15]. It is a relatively sophisticated software implementation of the H.265 standard for video compression. H.265 is positioned as a format for the next generation video codecs. However, in this research it was chosen for a different reason – it allows to utilise a wide range of macroblock sizes from 4×4 to 64×64 pixels [74]. Big enough blocks can be compressed with DCT as whole or subdivided into smaller ones through quad-tree partitioning in order to preserve the details. This helps to cover smooth and detailed areas in the large resolution frames more efficiently. Theoretically this should dramatically reduce the potential benefits of the dynamic resolution approach proposed in the thesis by using such built-in variable-resolution DCT. Nevertheless, this work investigates what kinds of improvements are still possible.

There are various software implementations of the H.265 standard, but x265 codec was chosen for the experiments among other implementations due to the following reasons:

– high efficiency of the encoder by default, which leaves a small room for the optimisation in general and with dynamic resolution in particular;

– a wide range of codec options for controlling quality and compression time;

– open source and vector instructions support (to speed up the experiments).

The codec provides a range of different parameters which directly influence outcome of compression. Some of them are grouped into presets, which according to user's manual [75] can be used to control thoroughness of the search for interframe motion compensation. There are ten presets in total that allow to vary compression speed in a range of several orders of magnitude for a small tradeoff in quality.

For quality control x265 by default uses so-called "constant rate factor" (or CRF, specified as *--crf <float>*), which implies a reference quantizer value for adaptive quantization. It is a major parameter determining a balance between resulting quality and compression ratio.

In order to demonstrate the influence of the main compression options of the x265 codec we used a short test video "Bosphorus" (fig. 4.1), which is available at [76]. This video contains a single 600-frame scene with a moving boat. It is a standard material used for testing video codecs.

Figure 4.2 shows a set of typical trends in the average quality, compressed size and encoding time upon compressing the test video with an entire range of CRF parameter values $[0; 51]$. The main effect of increasing the reference quantizer is the exponential decrease in the file size, which is accompanied by a considerable

Figure 4.1: First frame of the test video "Bosphorus" [1].

reduction in quality and compression time as well.

Figure 4.3 demonstrates that the compression time is a main objective affected by different x265 presets. It increases exponentially when moving towards slow option presets. It is understandable considering that changing preset generally adjusts the search radius and precision of detecting similar blocks in the related neighbour frames. The slow presets also tend to produce slightly higher quality and larger file size. This is a typical regularity, although the proportions may be different in case of another video.

Due the fact that this research investigates an idea of dynamic resolution video, it is reasonable to consider the effect of changing resolution. Although x265 does not support video resizing, it can be taken into account as an additional compression parameter – 'scale factor'. Figure 4.4 shows almost linear trends in resulting quality, size and time upon changing the scale from 1.0 to 0.5 and compressing the video with default options. Quality comparison for videos of different resolutions is made assuming that the display resolution has 1.0 scale (1920×1080 pixels).

The quality balancing experiments presented in this research targeted optimal combinations of the following parameters for each video segment:

– CRF;

– preset;

– scale factor (not x265 option).

It is important to mention that we designate quality balancing between frames in a video segment to the codec by using constant rate factor (*--crf <float>*) instead

[1]The video frame was taken from an unmodified source distributed under the Attribution-NonCommercial Creative Commons license: `https://creativecommons.org/licenses/by-nc/3.0/deed.en_US`

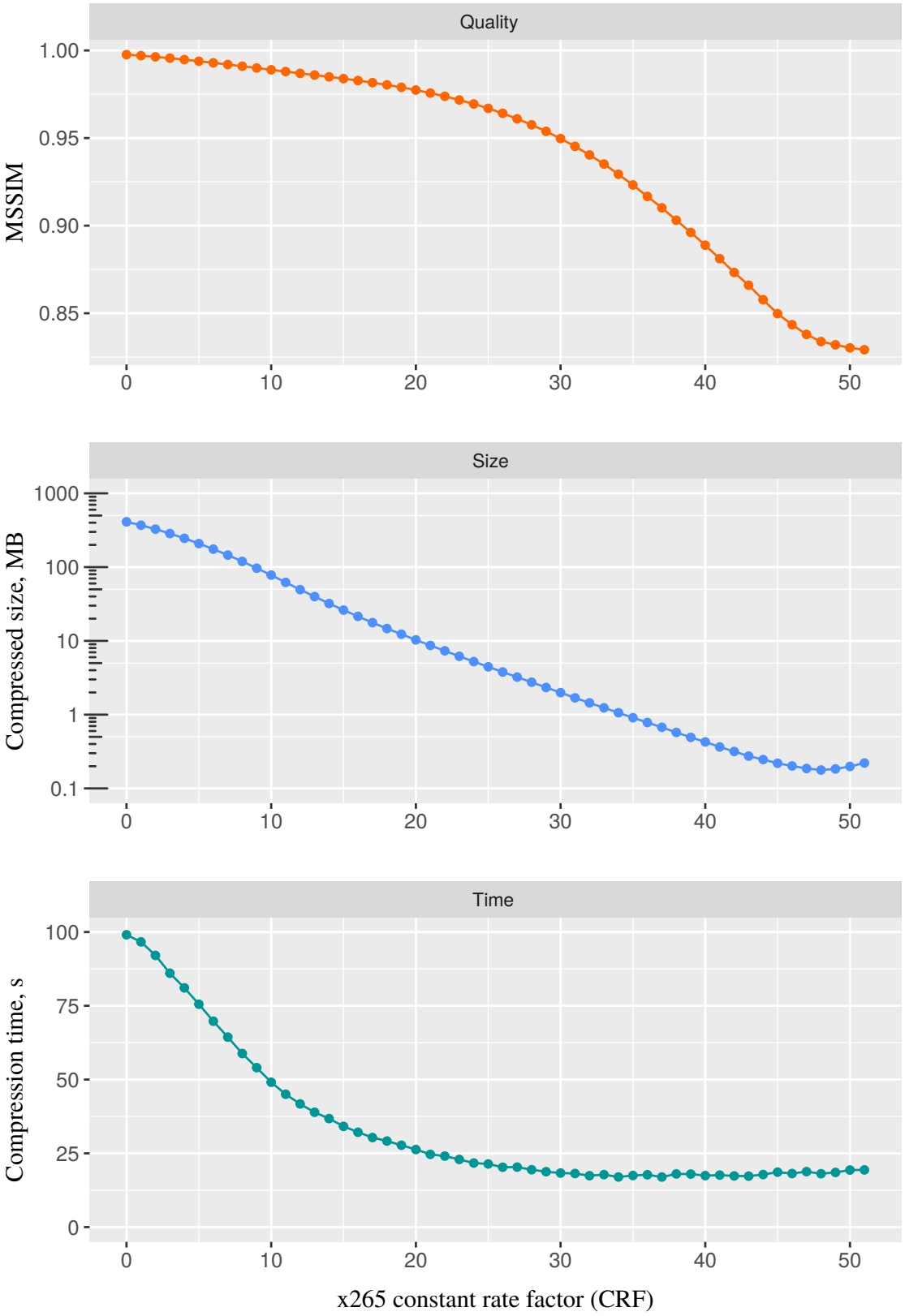


Figure 4.2: Example of the trends for quality, size and compression time when encoding a small test video with different CRF values.

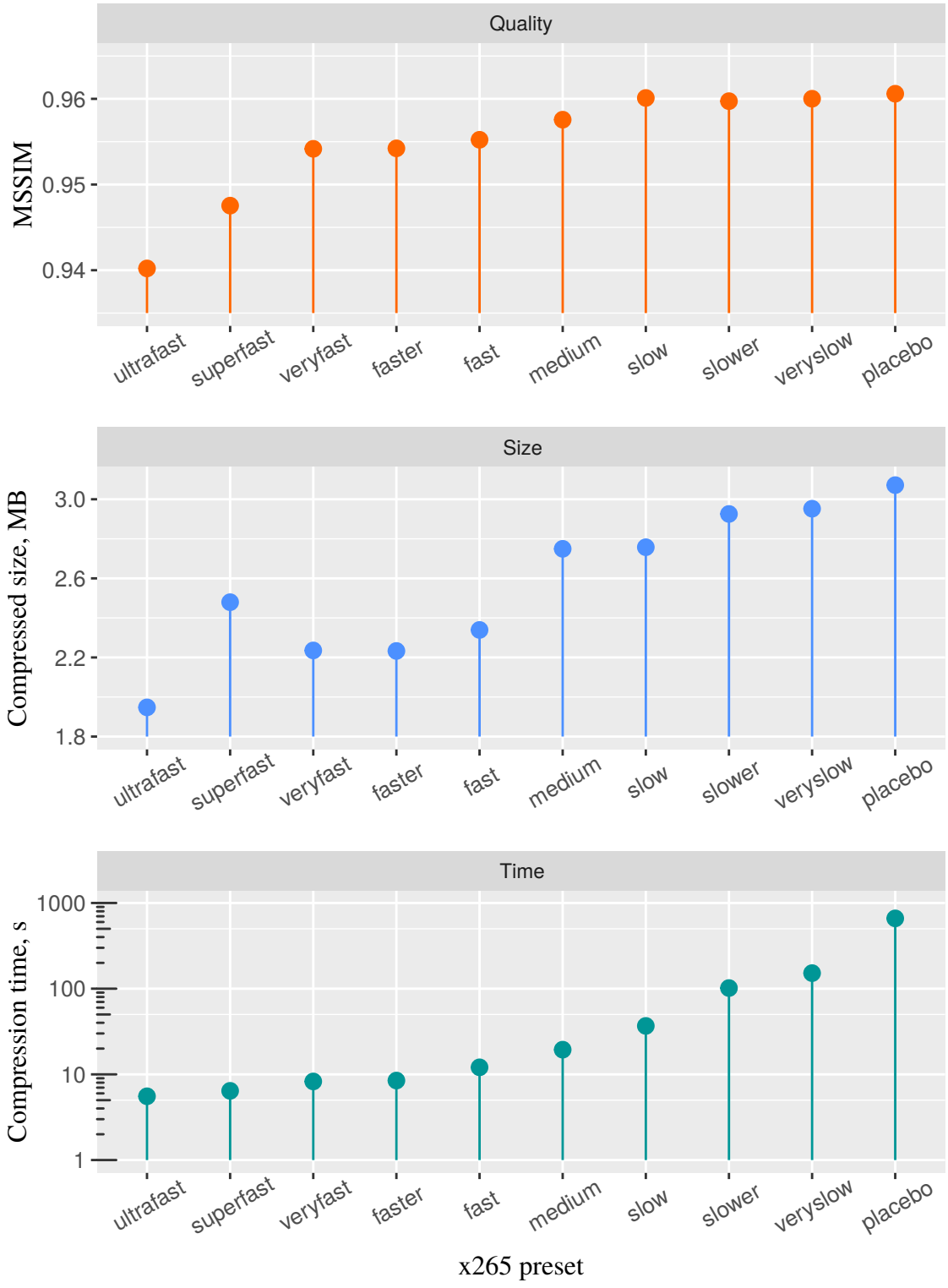


Figure 4.3: Example of the compressed video characteristics when using different presets with default CRF = 28.

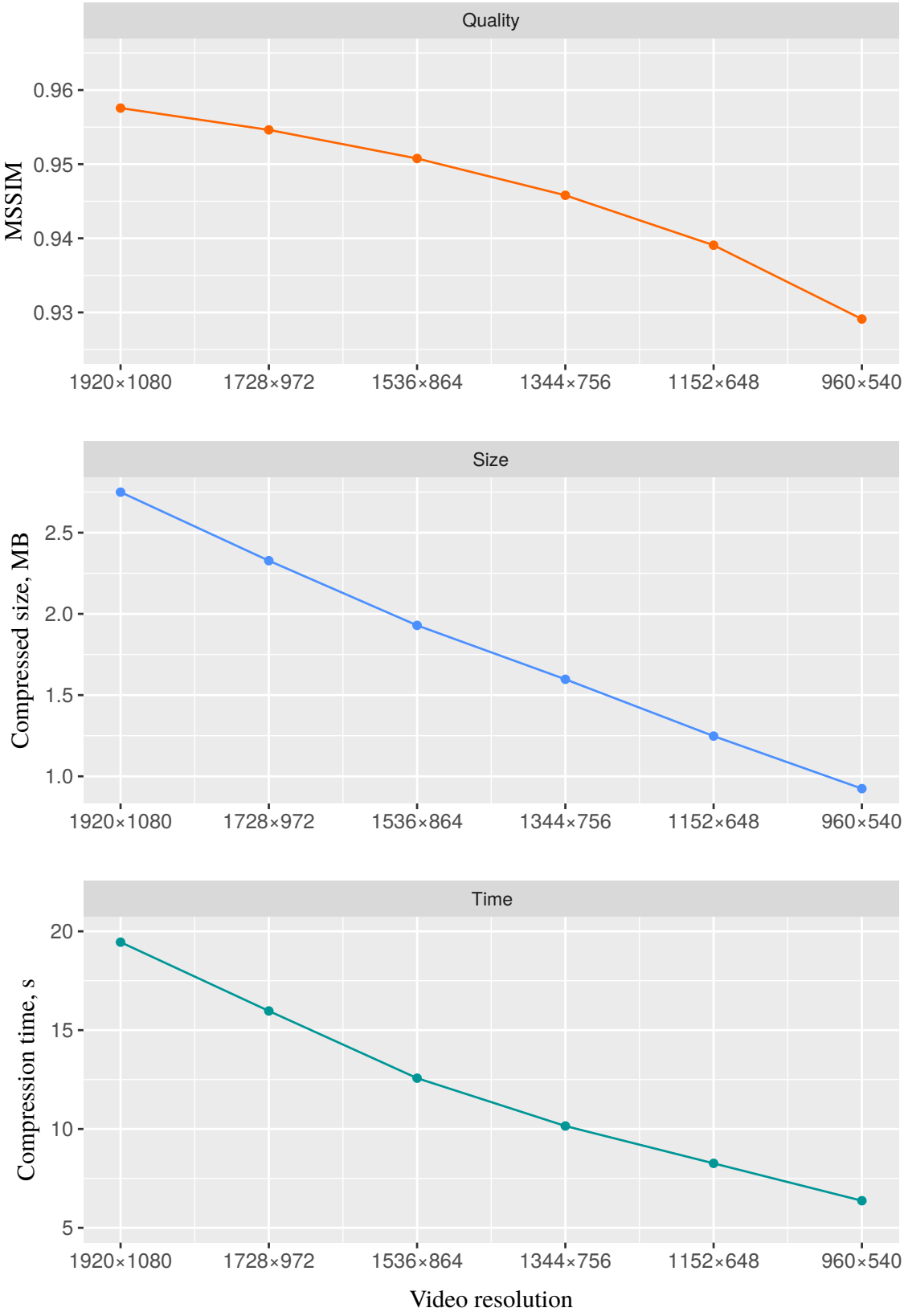


Figure 4.4: Trend examples of the default compressed video using preliminary resampling to different resolutions, and assuming that the quality is calculated with respect to the 1080p original.

of quality parameter (*--qp <int>*). This choice was made deliberately based on the fact that the "quality parameter" option means simply a constant quantizer for all frames, which is not the best scenario. We assume that using CRF allows x265 to keeps quality inside a segment at approximately the same level.

In order to compress a video the following command line syntax was used:

```
x265 --pools 8 --crf 28 --preset medium --no-progress
                        --input video.y4m --output video.h265
```

In this command *--pools 8* explicitly specifies the number of computational threads; CRF = 28 is a default reference quantizer value; *--preset medium* represents a recommended set of values for the minor codec options; *--no-progress* disables percentage counter to reduce the amount of logged information during automated batch processing.

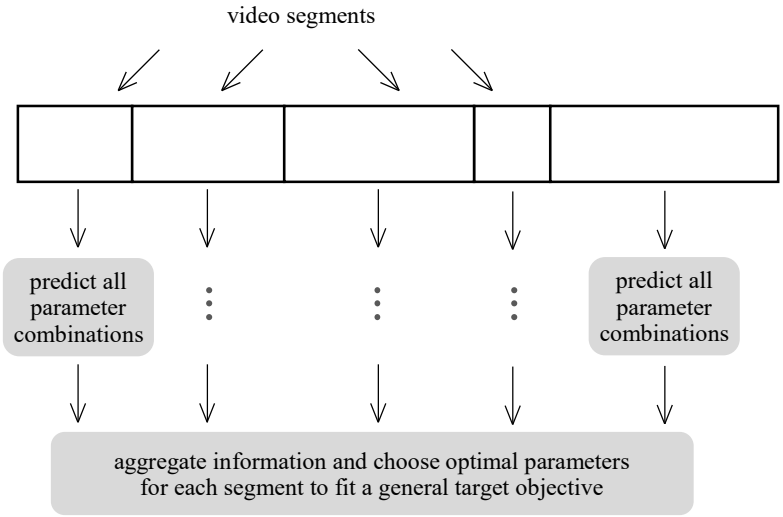
4.4 Problem Formulation

The general problem considered in this chapter is to find optimal combinations of compression parameters for all segments in a video that lead to the same quality but reduced file size or compression time in total. The result of video encoding with default options is used as a typical alternative for comparison purposes.

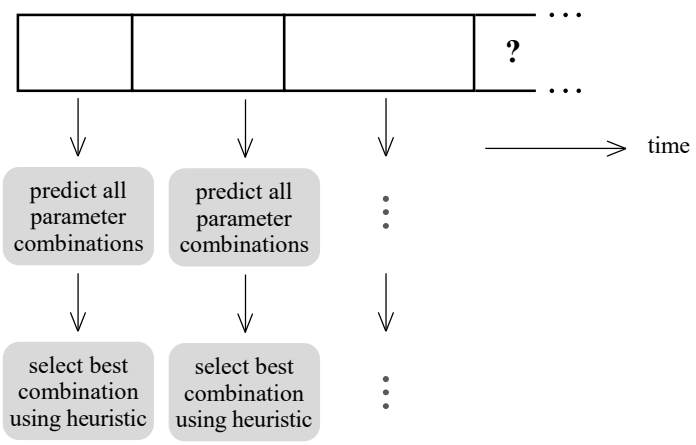
Multiple optimisation strategies are possible depending on the user priorities. This research focuses on the use cases that involve quality balancing because comparing other objectives if quality is the same is considerably easier and more reliable than comparing different values of the objective quality metric like MSSIM or PSNR.

One of the possible strategies is to match the average quality of the entire video with the default compression case while decreasing total size of the video stream. Such a method does not take into account the frame quality distribution in the compressed video. The main requirement of this approach is a preliminary analysis pass of the whole video (fig. 4.5a). It also needs to employ a search algorithm in order to find a suitable sequence of parameter sets for all segments.

An alternative method that does not require an extra pass is based on constraining the resulting quality level in a relatively small range for all segments, thus minimising frame quality distribution by definition. It does not necessarily require any preliminary samples from any part of the video. Instead this approach can rely on some heuristics guiding the choice of an optimal parameter combination among multiple alternatives with similar result characteristics. In this case video segments can be processed sequentially and independently from each other (fig. 4.5b).



(a) Search-based balancing



(b) Heuristic-based balancing

Figure 4.5: Sketches of different quality balancing strategies.

The problem is that having more than one compression option often results in several parameter combinations, which produce a given level of quality for a particular video segment. Assuming that it is possible to predict all such combinations, making an optimal choice is not a trivial task. For example, randomly selected options among those satisfying the quality constraint may not be the best alternative in terms of quality to size ratio. There could be a better parameter combination that provides a considerably smaller file size for a tiny decrease in quality – which is a more preferable choice.

So, having just a target quality level as an objective technically does not provide enough information to develop a straight and simple strategy for optimal quality adjustments in a sequence of independent segments. There should be some extra information about the balance of preferences between quality and compressed size. Creating a reasonable heuristic may help to deal with this problem.

4.5 Dataset of Video Segments

The problem of predicting compression result characteristics is an integral part of this research. The machine learning methods require a considerable amount of diverse training material to achieve a reasonable accuracy in their statistical estimations.

In order to provide training samples for the regression models a dataset of 2500 video segments was constructed. They were obtained from 47 YouTube videos, which were chosen more-or-less randomly but with deliberate intention to diversify the content of the whole dataset. The videos include amateur and professional filming material that contains scenes of urban and natural landscapes, people and animals as well as some aerial drone footage.

The resolution of all original video clips was 4K (3840×2160 pixels). However, due to relatively poor quality of the source material the frame detailization was increased by resampling all videos into smaller Full HD resolution (1920×1080 pixels). Between various processing steps the videos were stored in a lossless *.y4m file format [77], which contains separate frames as raw pixel values in YC_bC_r colour space. FFmpeg tool [71] was used to decode and resize videos at all stages of the experiments.

The optimal positions of key frames in each video were extracted from the compressed versions of each 1080p video using another program from the FFmpeg toolset called FFprobe. The x265 codec was used to perform the default compression in order to define positions of the key frames (fig. 4.6). The lossless videos were

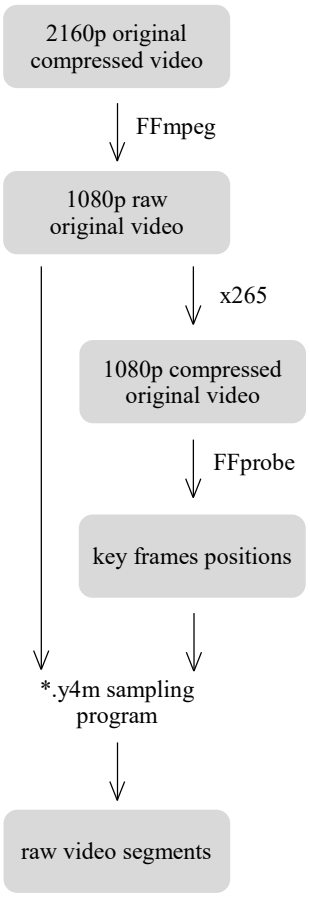


Figure 4.6: Video segments extraction procedure.

subsequently split into segments using a specially created program for frame-precise raw video sampling.

The length of obtained video segments varied between 1 and 250 frames (fig. 4.7). By default x265 limits segment size to 250 frames maximum, therefore any longer scene in the video was cut into smaller parts making 250 the most common size.

All raw video material was stored in a lossless *.y4m container format using 4:2:0 chroma subsampling (C_b and C_r colour components have two times smaller resolution 960×540 pixels). It was assumed that colour space transformation should be done according to ITU-R BT.709 standard [78], although actual conversion to RGB was not explicitly used in the experiments. The quality metric was calculated for luminance component Y under assumption that x265 balances chromatic components in a reasonable manner in accordance with luminance quality degradation. The latter assumption is based on the fact that the codec by default uses the same quantization parameter for all colour components (see x265 manual [75], section Quality Control, parameters *--cbqpoffs* <*int*> and *--crqpoffs* <*int*>).

The dataset of 2500 video segments was randomly split into two equal subsets:

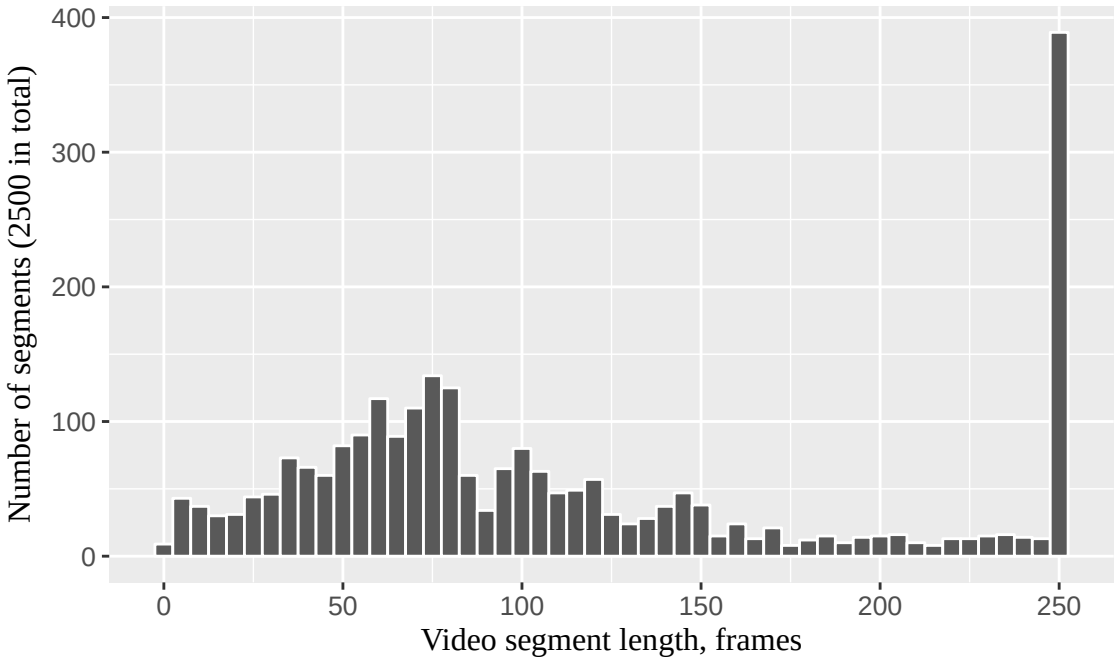


Figure 4.7: Video length distribution in the dataset of 2500 segments.

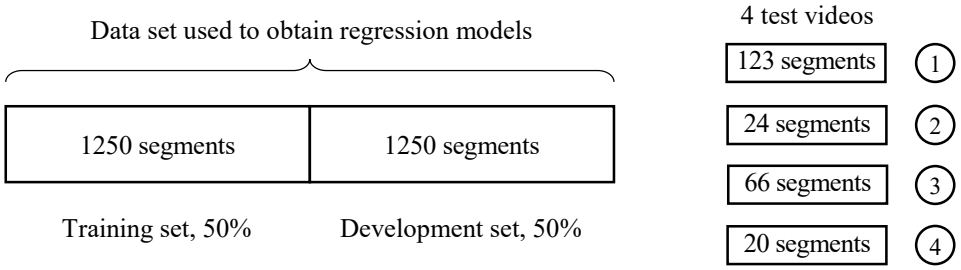


Figure 4.8: Dataset structure.

training and development (fig. 4.8). The training set was used in gradient calculations at each iteration of learning the model. The development set was employed to control overfitting during training and to choose the best model among several alternatives by the lowest error. Term "development set" is used by Andrew Ng in his recommendations to machine learning strategies [79]. This term reflects the purpose more accurately than traditional "validation set", although the meanings are similar. It is obvious that the models can overfit to the development set, however it can be considered as an intermediate test set to a certain extent when creating the statistical modes.

There was no explicit test set allocated among the 2500 segments. Instead, testing of the models was conducted on a separate group of four videos. The reason is that this research aims to optimise some real world use cases that involve complete videos rather than individual segments. Table 4.1 provides more detailed informa-

Table 4.1: Short summary of four test videos.

#	Video name	Content description	Frames	Segments	Source address
1	GoPro HERO5 + Karma: The Launch in 4K	Action scenes	7255	123	https://youtu.be/vlDzYIIOYmM
2	Horizon Zero Dawn PS4 Pro 4K Showcase	Video game recording	5257*	24	https://www.digitalfoundry.net/2017-02-23-free-download-horizon-zero-dawn-ps4-pro-4k-showcase
3	New York in 4K	City landscapes, complex motion	8421	66	https://youtu.be/TmDKbUrSYxQ
4	Sony Glass Blowing Demo	Colourful scenes on a dark background	2492	20	https://youtu.be/74SZXCQb44s

* the original frame rate was reduced by half to remove frame duplicates

tion about these videos. The first video contains a large number of various short action scenes. The second test video contains long seamless scenes of the gameplay from a modern 3D computer game. The third video mostly consists of highly detailed scenes with complex motion patterns. The last video contains images with smooth colour transitions and has relatively small amount of motion.

4.6 Error Metrics

An error metric defines an objective function, which is minimised during training of the regression models. Balancing video segments without actual compression requires reasonably accurate predictions of the following values:

– compressed file size;

– quality level in the MSSIM metric;

– compression time.

The range of possible values for quality and time is relatively narrow in comparison with compressed size. So, for different objectives this chapter adopts two error metrics: a widely used root-mean-squared error (RMSE) and specially designed root-mean-squared relative correction of prediction (RMSRCoP). The explanation below considers the differences in calculating partial derivatives for these two metrics. The problem is that the output of the regression models is not a directly predicted objective but its statistically standardised value (and also logarithmised for the file size model). This adds some extra complexity to the error function but facilitates training process by providing error metrics in the same units as target objectives (see also Appendix B for details).

4.6.1 RMSE

Assume that we have S training samples indexed with $<s>$ from 0 to $S-1$ (in superscript notation for convenience). The root-mean-squared error is defined as

$$\mathrm{RMSE} = \sqrt{\frac{1}{S} \cdot \sum_{s=0}^{S-1} \left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right)^2},$$

where $Q_{predicted}$ and Q_{target} are estimated and actual quality values for a training sample. Predicted quality value is calculated by reversing the standardization:

$$Q_{predicted}^{<s>} = \Theta^{<s>} \cdot sdevQ + meanQ,$$

where $\Theta^{<s>}$ is a neural network output, $meanQ$ and $sdevQ$ are the average and standard deviation values for target quality distribution.

Suppose that w_0 is one of the regression model parameters (a network connection weight). Then a partial derivative of the error function RMSE with respect to w_0 is calculated as follows:

$$\begin{aligned}
\frac{\partial\,\mathrm{RMSE}}{\partial w_0} &= \frac{1}{2} \cdot \frac{1}{\mathrm{RMSE}} \cdot \frac{\partial}{\partial w_0} \left(\frac{1}{S} \sum_{s=0}^{S-1} \left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right)^2 \right) = \\
&= \frac{1}{2} \cdot \frac{1}{\mathrm{RMSE}} \cdot \frac{1}{S} \cdot \sum_{s=0}^{S-1} \frac{\partial}{\partial w_0} \left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right)^2 = \\
&= \frac{1}{2} \cdot \frac{1}{\mathrm{RMSE}} \cdot \frac{1}{S} \cdot \sum_{s=0}^{S-1} \left(2 \cdot \left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right) \cdot \frac{\partial Q_{predicted}^{<s>}}{\partial w_0} \right) = \\
&= \frac{1}{\mathrm{RMSE} \cdot S} \cdot \sum_{s=0}^{S-1} \left(\left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right) \cdot \frac{\partial \left(\Theta^{<s>} \cdot sdevQ + meanQ\right)}{\partial w_0} \right) = \\
&= \frac{sdevQ}{\mathrm{RMSE} \cdot S} \cdot \sum_{s=0}^{S-1} \left(\left(Q_{predicted}^{<s>} - Q_{target}^{<s>}\right) \cdot \underbrace{\frac{\partial \Theta^{<s>}}{\partial w_0}}_{\substack{\text{calculated with} \\ \text{backpropagation}}} \right)
\end{aligned}$$

The network output derivatives with respect to any internal parameter were obtained using backpropagation algorithm according to Bishop [68, chapter 4].

One of the advantages of using RMSE is the fact that it is equal to the standard deviation of the prediction error distribution (fig. 4.9), of course under assumption that the expected value of the error is zero. Since the distributions of prediction errors and target objective values are connected, the latter needs to be transformed into normal form prior to the training process.

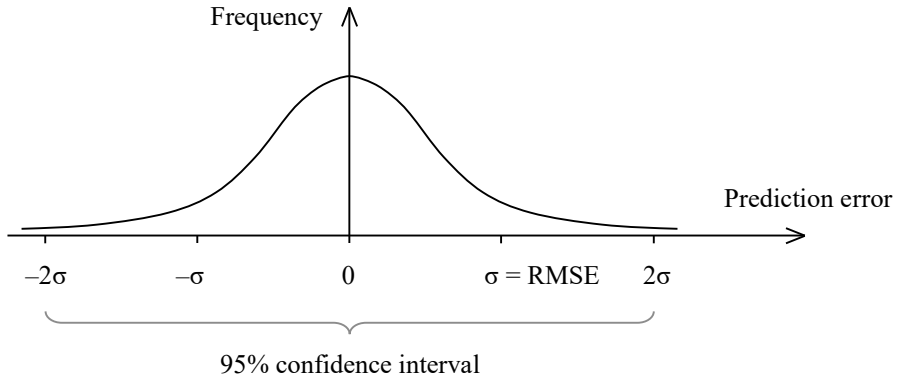


Figure 4.9: Relation between RMSE and error distribution.

4.6.2 Relative correction of prediction

RMSE is an optimal choice for the regression models predicting quality or time. It is relatively easy to use as an error function because it operates in the units of a target objective. However, it is not the best option for representing compressed file size errors due to a wide range of possible values.

Relative correction of prediction is a percentage error metric defined as

$$\mathrm{RCoP} = \frac{size_{target} - size_{predicted}}{size_{predicted}} \cdot 100\%.$$

The main difference from a simple percent error is that it is expressed in the fractions of a predicted value instead of target one (fig. 4.10).

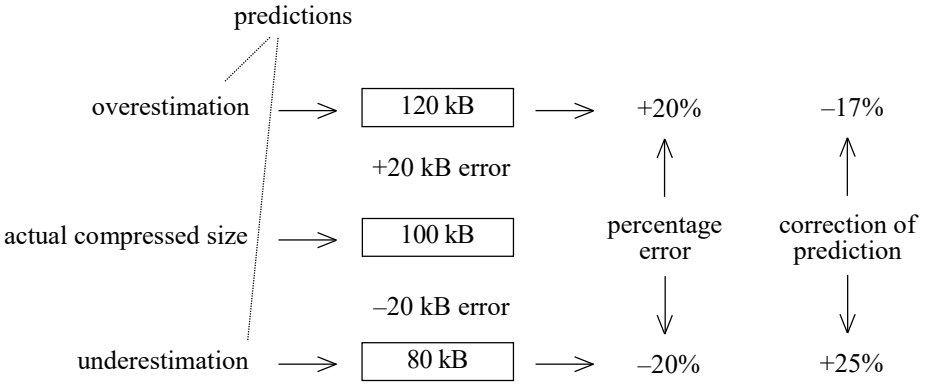


Figure 4.10: Example comparison between percent error and relative correction of prediction.

In order to connect this metric with a confidence interval of the prediction it

was further developed into root-mean-squared relative correction of prediction:

$$\begin{aligned}\mathrm{RMSRCoP} &= \sqrt{\frac{1}{S}\cdot\sum_{s=0}^{S-1}\left(\frac{size_{target}^{<s>}-size_{predicted}^{<s>}}{size_{predicted}^{<s>}}\right)^2}\cdot 100\% = \\ &= \sqrt{\frac{1}{S}\cdot\sum_{s=0}^{S-1}\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)^2}\cdot 100\%.\end{aligned}$$

Using RMSRCoP as an error function for the regression model enforces normal distribution of the percentage correction values and allows to express target file size as a predicted value with confidence interval:

$$size_{target} = size_{predicted} \pm (2\sigma)\,\%,$$

where σ = RMSRCoP. Confidence interval calculated as a fraction of a predicted value helps to estimate a range of possible file size values in bytes.

The target file size values were logarithmised and statistically standardised to facilitate training of the MLP. Predicted file size value is calculated from the network output through reversed standardization and subsequent exponentiation:

$$size_{predicted}^{<s>} = e^{\Theta^{<s>}\cdot sdevLnSize + meanLnSize},$$

where $\Theta^{<s>}$ is a neural network output, $meanLnSize$ and $sdevLnSize$ are the average and standard deviation values of the logarithmised file size distribution.

A partial derivative of the RMSRCoP with respect to any regression model parameter w_0 is calculated as follows:

$$\begin{aligned}\frac{\partial\,\mathrm{RMSRCoP}}{\partial w_0} &= \frac{1}{2}\cdot\frac{1}{\mathrm{RMSRCoP}}\cdot\frac{\partial}{\partial w_0}\left(\frac{1}{S}\cdot\sum_{s=0}^{S-1}\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)^2\right) = \\ &= \frac{1}{2}\cdot\frac{1}{\mathrm{RMSRCoP}}\cdot\frac{1}{S}\cdot\sum_{s=0}^{S-1}\frac{\partial}{\partial w_0}\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)^2 = \\ &= \frac{1}{2}\cdot\frac{1}{\mathrm{RMSRCoP}}\cdot\frac{1}{S}\cdot\sum_{s=0}^{S-1}\left(2\cdot\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)\cdot\frac{\partial}{\partial w_0}\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}\right) = \\ &= \frac{1}{\mathrm{RMSRCoP}\cdot S}\cdot\sum_{s=0}^{S-1}\left[\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)\cdot\frac{-size_{target}^{<s>}}{(size_{predicted}^{<s>})^2}\cdot\frac{\partial size_{predicted}^{<s>}}{\partial w_0}\right] = \\ &= \frac{sdevLnSize}{\mathrm{RMSRCoP}\cdot S}\cdot\sum_{s=0}^{S-1}\left[\left(\frac{size_{target}^{<s>}}{size_{predicted}^{<s>}}-1\right)\cdot\frac{-size_{target}^{<s>}}{size_{predicted}^{<s>}}\cdot\underbrace{\frac{\partial\Theta^{<s>}}{\partial w_0}}_{\substack{\text{calculated with}\\ \text{backpropagation}}}\right]\end{aligned}$$

4.7 Gradient Descent Techniques

Training the regression models quickly and efficiently is an important part of the investigation. Two gradient descent optimisation techniques were considered in this chapter for training multilayer feed-forward neural networks: gradient descent with momentum [69] and Adam (adaptive moment estimation) [80]. The latter is a more complex algorithm, however typically it demonstrates better performance in terms of number of iterations to reach the same level of error. Figure 4.11 shows example of the error trends, where Adam also converges to a slightly better result.

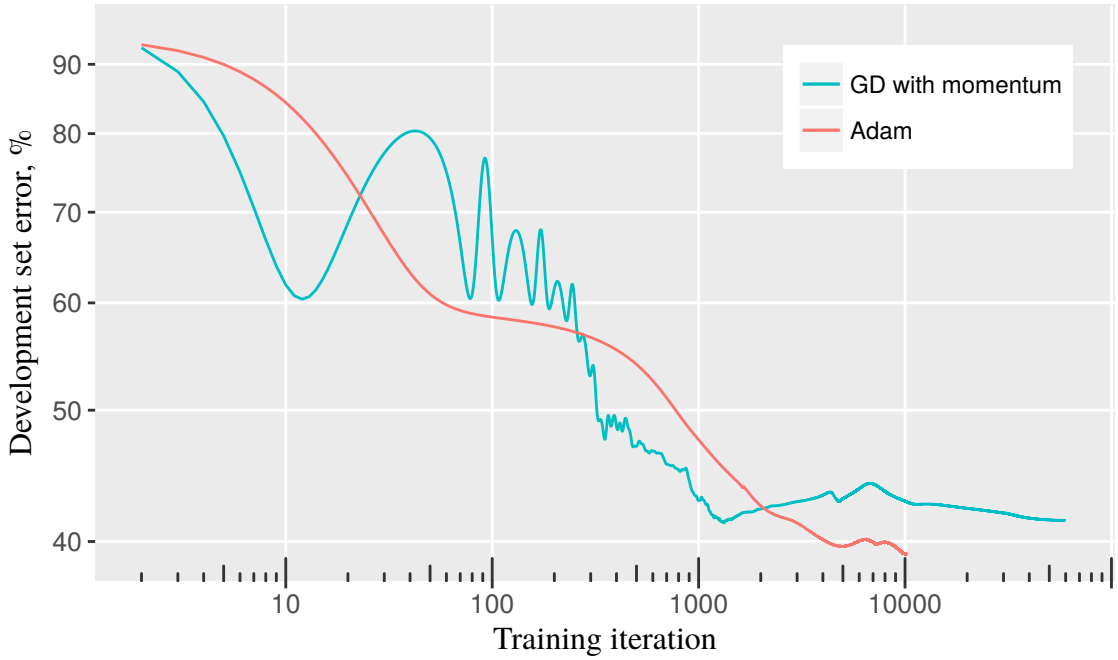


Figure 4.11: Example of the error trends comparison using different gradient descent modifications.

When using gradient descent with momentum at every iteration i the network parameters $\overrightarrow{w_i}$ are updated according to formulas:

$$\overrightarrow{m}_{i+1} \leftarrow \beta \cdot \overrightarrow{m}_i + (1-\beta) \cdot \nabla E\left(\overrightarrow{w_i}\right);$$
$$\overrightarrow{w}_{i+1} \leftarrow \overrightarrow{w_i} - \overrightarrow{m}_{i+1},$$

where $\beta = 0.99$ and ∇E is a gradient of the error function.

Adam works similar to previous formulas, however there are two separate mo-

mentum vectors (first and second moment estimates):

$$\begin{aligned}
&\overrightarrow{m}_{i+1} \leftarrow \beta_1 \cdot \overrightarrow{m}_i + (1-\beta_1) \cdot \nabla E\left(\overrightarrow{w_i}\right); \\
&\overrightarrow{v}_{i+1} \leftarrow \beta_2 \cdot \overrightarrow{v}_i + (1-\beta_2) \cdot \nabla E\left(\overrightarrow{w_i}\right) \odot \nabla E\left(\overrightarrow{w_i}\right); \\
&\overrightarrow{w}_{i+1} \leftarrow \overrightarrow{w_i} - \alpha \cdot \frac{\overrightarrow{m}_{i+1}}{\sqrt{\overrightarrow{v}_{i+1}} + \epsilon},
\end{aligned}$$

where $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ according to the original algorithm description [80, algorithm 1]. The only difference is that bias correction for moment initialization, which increases step size at first iterations, was not used to simplify program implementation.

4.8 Experimental Setup

Encoding videos and video segments with a purpose of recording size, quality and compression time was conducted on a cluster of 12 computers with Intel Xeon E3-1240 v2 @ 3.4 GHz, 16 GB RAM running Scientific Linux 7.3 64-bit with GCC 4.8 compiler. The x265 codec was compiled with AVX instructions enabled (but not AVX2 due to lack of support). The software versions used: x265-2.4, ffmpeg-3.3. Compression of all video material was performed in 8 threads.

Training of the regression models was done on Intel Core i7 5820K @ 4.0 GHz with 16 GB RAM running Ubuntu 17.04 64-bit and GCC 6.3 compiler.

4.9 Video Segment Features

In this chapter fifteen content features were employed as a measure of video segment complexity for the machine learning models. The feature set consists of ten image features used in the previous chapter and five interframe differences, which estimate the amount of motion in different directions.

The process of choosing appropriate content features was guided by the error of the regression models for predicting compressed size. It was done considering an observation in the previous chapter that file size is the hardest objective to predict accurately.

In order to train a basic model for predicting compressed size, all 2500 video segments from the dataset were encoded with the default compression options using x265 codec. The obtained file sizes were normalised by the video segment length to represent the compressed size per frame. However, the length parameter was still

used as an input in the models because it is related to the distribution of P- and B-frames (forward-predicted and bi-directionally predicted frames) in the segment. Different frame types use different quantizers, which in turn affect total file size, compression time and average quality. For example, short segments often contain a small number of B-frames, while long segments consist mainly of B-frames. The difference in compression ratio between P- and B-frames can be greater than an order of magnitude.

The idea behind designing a set of content features is to predict the default compressed file size using different groups of features and choose the best ones. For this purpose the preliminary regression models were trained. They were based on a simple feed-forward neural network (fig. 4.12) with two hidden layers containing 3 and 2 neurons. This network configuration yields the smallest errors in the development set for the current problem. It was empirically established that using more than one hidden layer allows to delay the moment of overfitting. During training the early stopping regularization technique was extensively used [67, chapter 7]. So, the reported results relate to the point of minimal development set error.

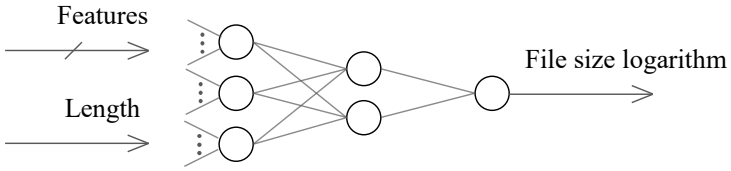


Figure 4.12: Sketch of an MLP used to evaluate content features.

The process of evaluating features efficiency started with a baseline model that had no content features as input, only length. The network on fig. 4.12 was trained on 1250 segments to predict the default compressed size per frame. Minimal root-mean-squared correction of prediction was used as a training objective and an error metric. The stopping criterion was the smallest development set error, which was 59.5% (fig. 4.13). It is important to mention that the used percentage error metric is independent from the video segment length because it is based on the ratio between actual and predicted file size.

Considering the fact that video compression aims to reduce both spacial (inside frames) and temporal redundancy (interframe similarities) features representing both these types of content entropy are required.

In the previous chapter a set of ten image features was proposed and proven useful for predicting compression result characteristics. It was reasonable idea to try them for videos too. Image features were modified to be calculated as an average for all frames in the video segment – i.e. the frames and their non-overlapping

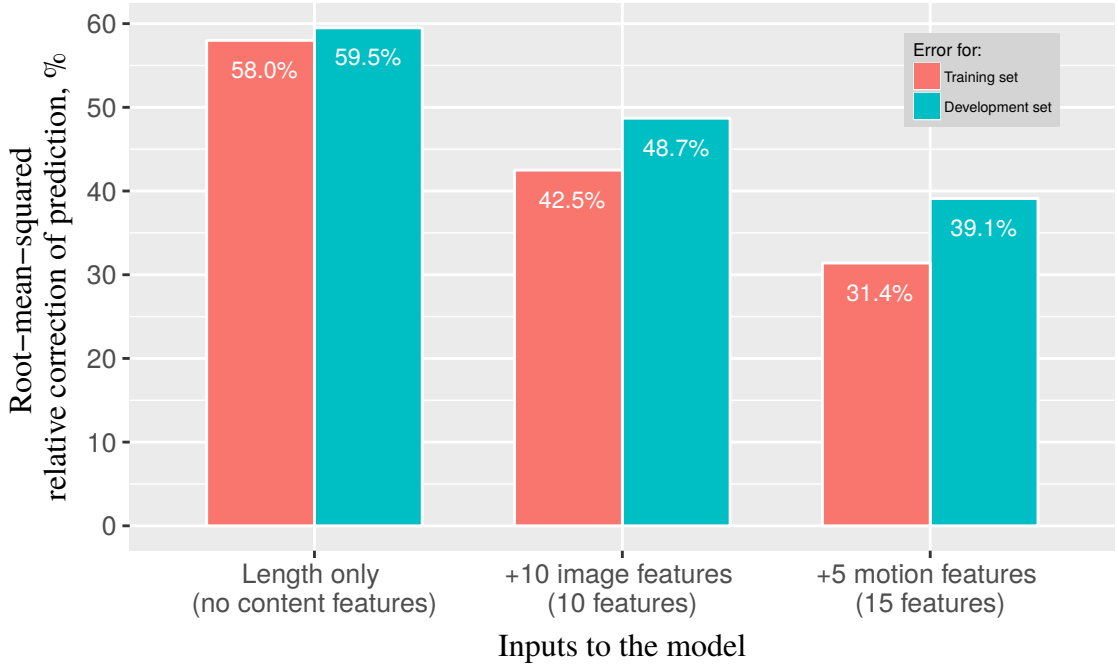
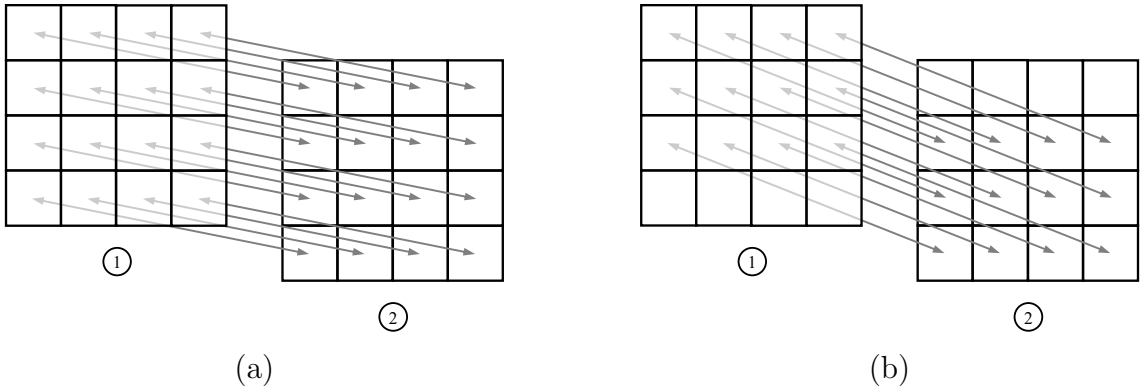


Figure 4.13: Adding content features decreases average file size prediction errors.

blocks are considered as separate parts of a single huge image. The length of the video segment does not matter because content features are resolution independent by design. These adopted video features were denoted $\boldsymbol{f_1}$–$\boldsymbol{f_{10}}$ in exactly the same order as in section 3.2. Upon adding this feature set as an input to the regression model, the error in the development set reduced to 48.7% in comparison with no features case (fig. 4.13). It means that content features play a significant role in the problem of predicting compressed size of the video, which is expectable considering how important they are for images.

In order to estimate the amount of change in the content of neighbour frames a set of five "motion features" was introduced. The first feature ($\boldsymbol{f_{11}}$ for convenience) is an average absolute pixel difference between consecutive frames. Figure 4.14a illustrates all differences between 4×4 pixels block and its subsequent projection from the next frame. This feature measures lack of direct similarity between frames.

The other four motion features $\boldsymbol{f_{12}}$–$\boldsymbol{f_{15}}$ are somewhat similar to $\boldsymbol{f_{11}}$. They are calculated as mean absolute differences between pixels in neighbour frames that are shifted by one pixel in each of four orthogonal directions: left, right, up and down. For example, figure 4.14b shows how subtracting pixels with offset allows to estimate motion downwards. All four features are calculated in a similar manner. Although, exactly speaking, these features measure "lack of movement" in the corresponding offset directions because the feature value decreases if it correlates with the line of



(a) (b)

Figure 4.14: Differences used in video features to detect static regions (a) and movement down (b).

actual content motion.

The design of the motion features was based on an assumption from author's subjective observation that the most common motion radius of a pixel in an ordinary video is zero pixels or one pixel (static background or slightly shaking camera). In general it is less likely to find bigger radius.

The motion features were averaged across all frames in the video segment and logarithmised similar to image features: $\boldsymbol{f_i} \leftarrow log_{10}(\boldsymbol{f_i} + 1)$. A different logarithm basis 10 was used due to large range of values, although it is not fundamentally important. These features are also independent from video length and resolution.

Adding five extra features to the regression model reduced average error further to 39.1% (fig. 4.13), which means that the set of fifteen described features so far is the most efficient for predicting compressed file size. The number of training iterations to obtain this model was approximately 10 000 (and 15 s of training time).

Figure 4.15 demonstrates how correction of prediction is distributed for 1250 segments from the development set using the proposed features. Although the range of deviations is quite wide, its distribution is strictly controlled by the error metric. For example, more than 95% of predictions (1194 out of 1250) are in ±78.2% interval ($2\sigma = 2 \cdot 39.1\% = 78.2\%$).

Unfortunately, any subsequent attempt to add extra content features like various other interframe differences or 3-dimensional convolutions did not noticeably reduce the average error. This may be due to several reasons.

Firstly, increasing number of input variables leads to overparametrizing the model, which in turn provokes overfitting. The growing error gap between training and development set on fig. 4.13 suggests that the risk of overfitting is raising. Using a considerably larger training set can be a rational course of action, however in this

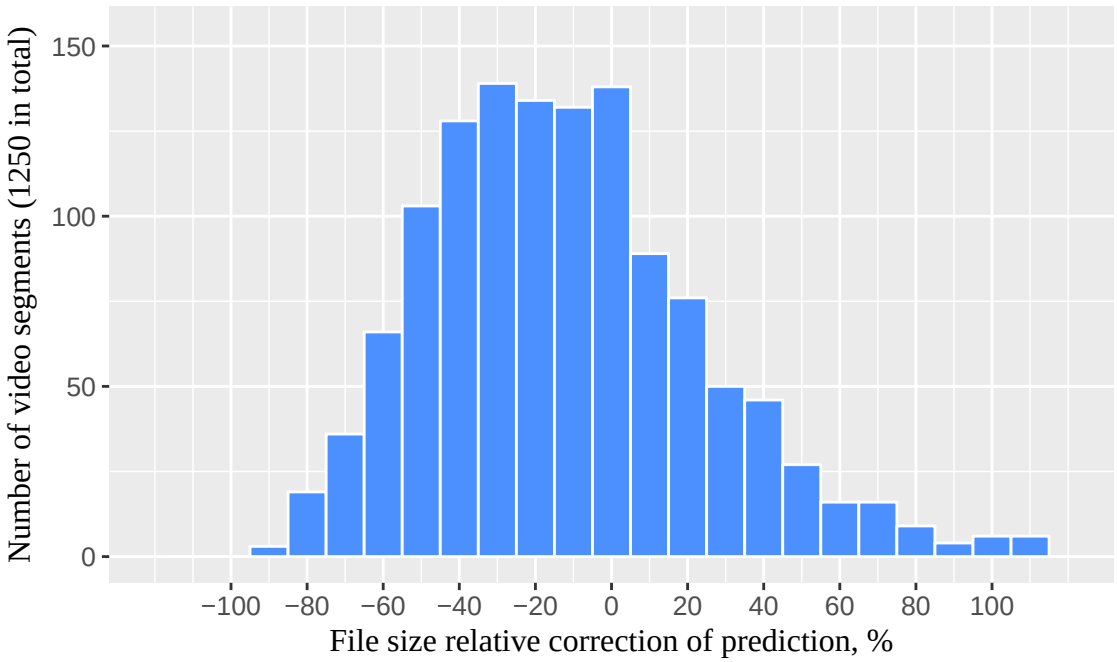


Figure 4.15: Example distribution of the file size errors.

case the original quality and diversity of the training material becomes a problem.

Secondly, the low informativeness of the features themselves could be an issue. It is obvious that the proposed motion features cannot possibly capture all kinds of temporal complexity in the video. Supposedly, creating more complex and computationally expensive features will help to increase accuracy of model predictions.

In an attempt to perform a feature selection procedure, content features $\boldsymbol{f_2}$, $\boldsymbol{f_5}$, $\boldsymbol{f_9}$ and $\boldsymbol{f_{10}}$ were excluded from the model inputs. This reduced development set error from 39.1% to 38.3%. However, as this tendency did not remain for complex regression models, described in the next section, the decision was made to leave unchanged the proposed set of fifteen features. Although some features can be almost harmlessly removed from the models predicting compressed size, they may still be useful in the models for quality or time. Nevertheless, reducing number of required features in practice will decrease time for their computation.

The set of video content features was designed manually, and despite the extra attempts it could not be improved with other handmade features. However, this does not mean that it is the best possible solution.

Calculating a set of 15 features for 2500 video segments in a sequential order took about 2.5 hours. So, processing 1000 random segments requires approximately an hour including time for reading from HDD. For a single Full HD frame feature extraction takes 30 ms including disk reading time and only 5 ms using in-memory

processing. These are average values obtained from a long video. It is reasonable to assume that with some extra programming effort the time can be reduced.

4.10 Regression Models

This section describes how the models used in the experiments for actual predictions were obtained from the video segments dataset. The regression models in this chapter were employed for the analogous purpose as in the previous – to predict compressed video characteristics without performing actual compression. Besides an obvious advantage of saving processing time this allows to optimise other objectives like quality and compression ratio for an entire video.

So, using the statistical models it is possible to discover optimal compression parameters relatively quickly for each segment inside a video depending on the goals of a specific practical scenario. The space of compression parameters was designed to include three options that have a major impact on the compression result:

– *scale factor* – a coarse way of quality control by changing resolution;
– *constant rate factor* for more precise manipulation with quality to size ratio;
– *preset*, which is mainly used to adjust compression speed.

Due to the fact that this work aims to surpass the default compression in terms of mainly file size or encoding time but keep quality the same, there is no need to consider all possible values of the parameters – only those that make the objectives smaller. For example, increasing frame resolution is not useful because it raises compression ratio and encoding time. For the same reason small CRF values (target quantizers), which produce files considerably larger than the default, were not used.

The default encoding options are CRF = 28, medium preset and, of course, no resolution scaling. The regression models were trained using six values of scale factor in the range [0.5; 1.0] (with corresponding resolutions):

– 1.0 (1920×1080);
– 0.9 (1728×972);
– 0.8 (1536×864);
– 0.7 (1344×756);
– 0.6 (1152×648);
– 0.5 (960×540).

Twenty five integer vales of CRF from the range [24; 48] were considered allowing smaller quantizers than the default 28 under assumption that, for example, a combination of CRF = 24 and scale = 0.9 can be better than the default options, say, for a low-detailed scene.

As for the preset parameter, it was included into the models because it can be used for comparison purposes as a simple alternative to default compression. From ten preset values provided by x265 only two were considered: *medium* (used by default) and *fast*. Using fast preset usually decreases compression time at a small penalty in quality or file size. In order to give numerical representation to the chosen presets, they were assigned codes 4 (fast) and 5 (medium).

Other presets were not considered in the experiments because of the difficulties in gathering statistical data for training. Thus, very fast presets complicate compression time measurement due to a considerable relative cost of reading each video segment from the disk, while using slow presets requires a lot of time to simply compress video segments with all combinations of scale factor and CRF. Also adding slow presets does not help to decrease compression time for any segments.

After defining the parameter space (fig. 4.16) each of 2500 video segments designated for model training purposes was compressed with 300 combinations of options ($6\ scales \cdot 25\ CRFs \cdot 2\ presets = 300\ combinations$). The framerate in all compression runs was set to 25 FPS.

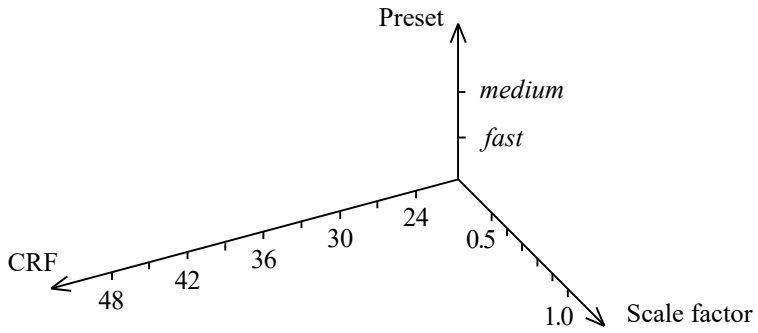


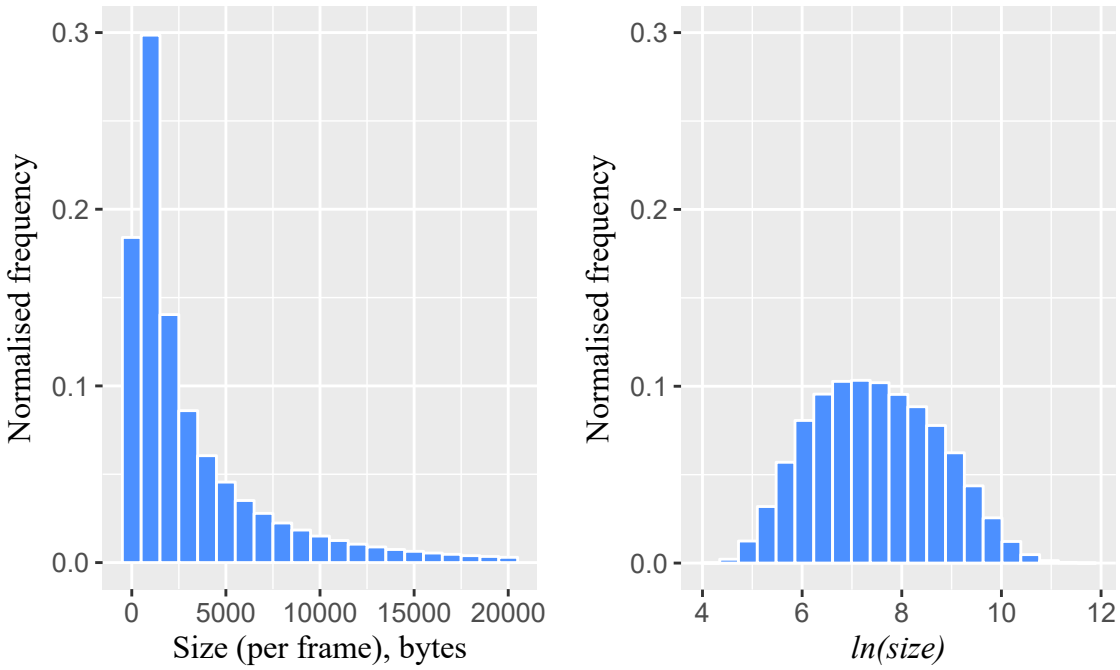
Figure 4.16: Range of scales and x265 parameters used in the experiments.

In each case compressed size, quality and time were recorded. This data together with the previously calculated content features was used to train regression models according to the scheme:

$$(features;\ length;\ scale;\ CRF;\ preset) \rightarrow (size \mid quality \mid time).$$

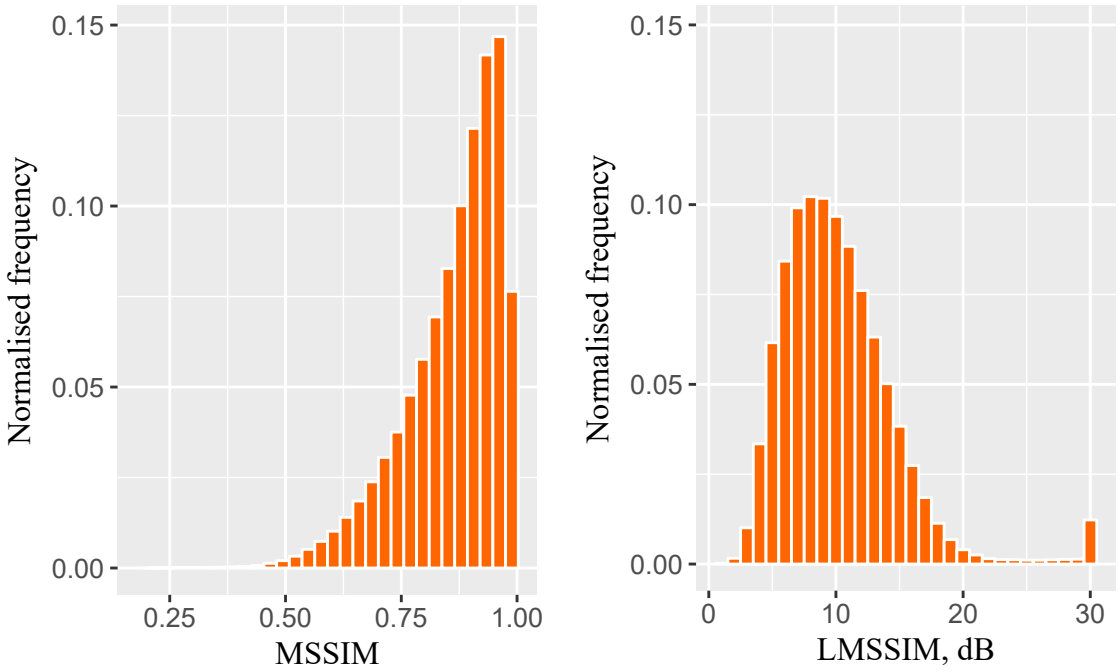
Quality measurement was performed between the raw video segment and its compressed version that had been decoded into raw *.y4m format. The MSSIM metric was calculated for luminance components using a specially designed program. Compression time was recorded according to the measurements by x265 codec itself.

Before training a regression model for predicting compressed file size the target values were normalised by length of the corresponding segments to represent size per



(a) (b)

Figure 4.17: Compressed frame size distribution before and after logarithmisation.



(a) (b)

Figure 4.18: Distribution of mean SSIM quality metric values: standard (a) and measured in decibels (b).

frame, which narrowed the distribution of possible outputs. Then the values were logarithmised to normalise the distribution (fig. 4.17). Finally the target values of the logarithmised size were statistically standardised to increase the speed of converging to a solution.

The target quality levels in MSSIM metric were also preprocessed prior to training the model. In order to normalise the distribution of quality values they were transformed using a special function for representing SSIM metric in decibels, which had been introduced into x264 video codec under the name x264_ssim [81]. In this chapter this metric was called "logarithmised mean SSIM" (LMSSIM) to avoid confusion related to a different codec. It was calculated according to formula:

$$\mathrm{LMSSIM} = -10 \cdot \log_{10}(1 - \mathrm{MSSIM}).$$

Figure 4.18 shows how this transformation affects distribution of quality values. Considering the facts that the MSSIM numbers in this research are expressed using three digits after decimal point, they were limited by 0.999 maximum to prevent infinitely large values of LMSSIM. As 0.999 corresponds to 30 dB, some video segments with quality close to 1.0 created a spike at 30 dB on figure 4.18b.

The training process commenced with statistical standardization of all input and output variables, has been reported to facilitate the gradient descent. This process was repeated for all input features and compression parameters, even preset codes.

Due to the fact that the preset option is not a continuous variable but a set of categories, and with intention to decrease the load on a predictor, it was assumed that excluding preset from the model inputs will improve accuracy. Alternatively, training a separate model for each preset value seemed a reasonable option. However, after comparing the best models obtained with and without preset as a regressor the outcome was the opposite.

Table 4.2 compares different network topologies by the correction of prediction in the development set. Each configuration was checked once and took between 2 and 8 hours to train (for the smallest and the biggest network respectively). The left part of the table contains training results for models without preset, while the right one reflects tests with this option. The models excluding preset parameter were learned using half of the training points corresponding only to medium preset. It would appear that adding the preset as input parameter led to better result. The MLP configuration with the lowest error (30.3%) is shown on figure 4.19. This model was used in further experiments.

Comparing the 30.3% error with 39.1% obtained during feature set construction

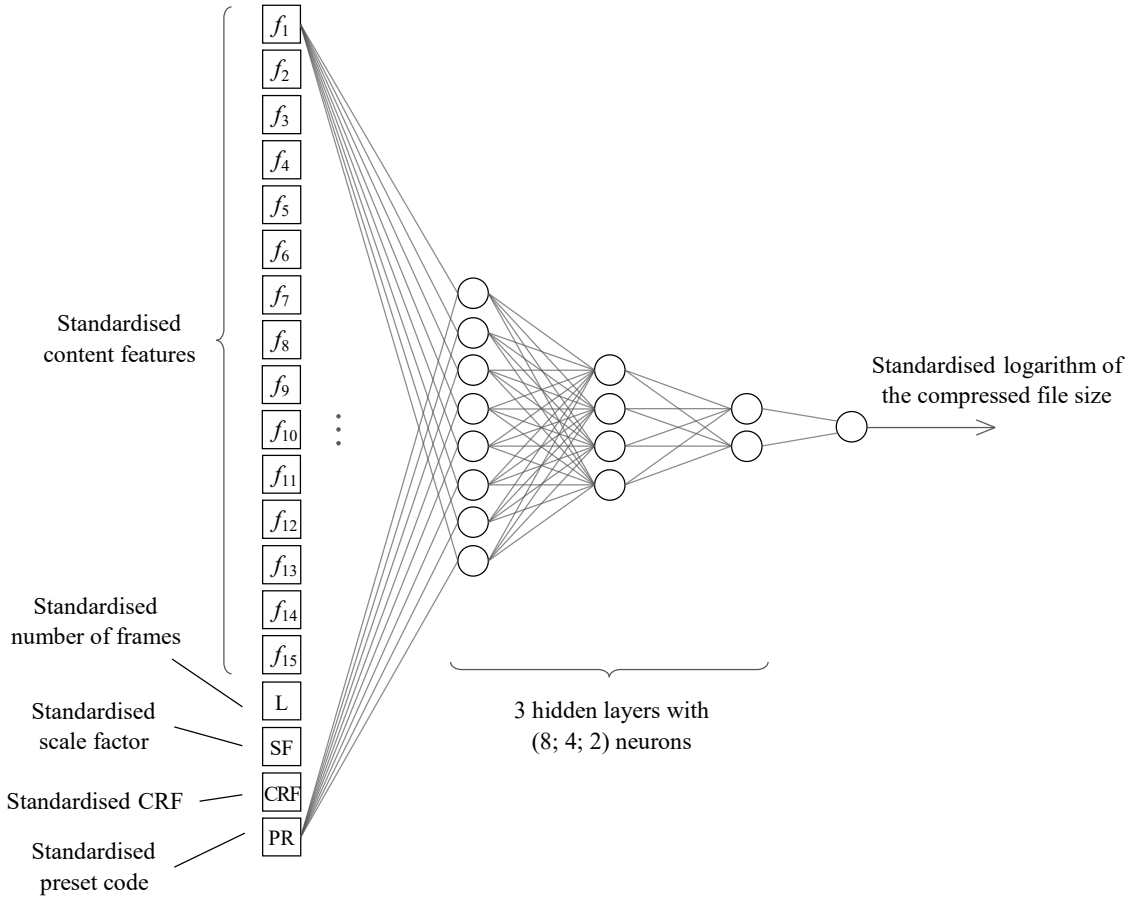


Figure 4.19: Sketch of the best network for file size predictions.

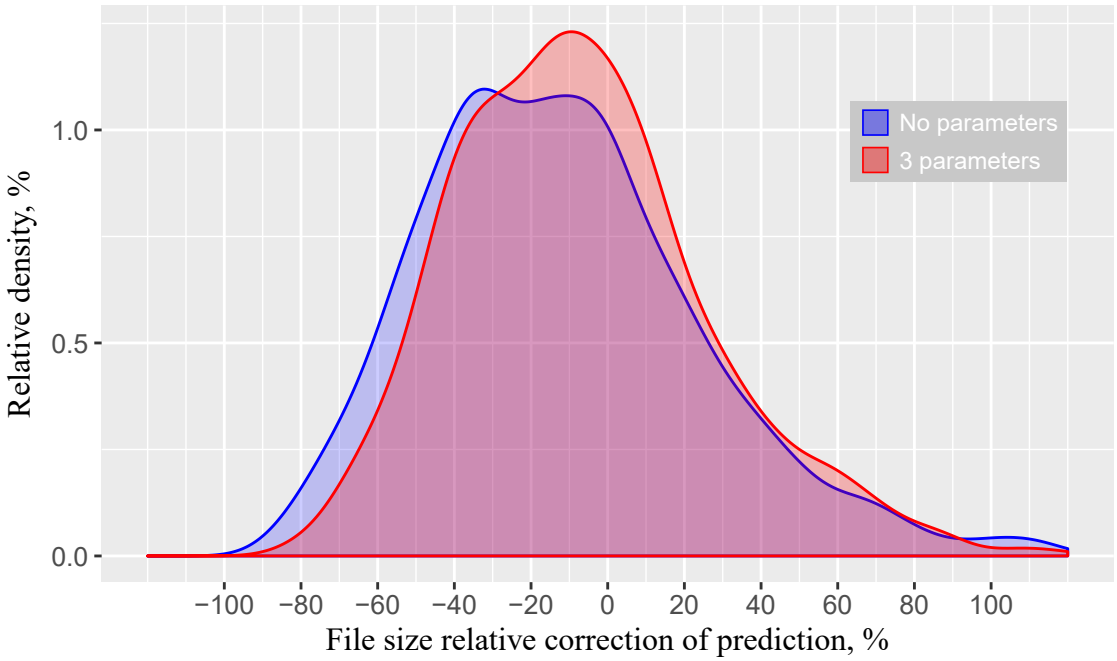


Figure 4.20: Difference between file size error distributions for models with and without compression options (calculated for development set).

Table 4.2: Selecting optimal network structure for the regression model predicting file size.

Neurons in hidden layers	Scale factor and CRF as input variables			Scale factor, CRF and preset as input variables		
	Root-mean-squared correction of prediction, %		Training iterations	Root-mean-squared correction of prediction, %		Training iterations
	Training set	Development set		Training set	Development set	
(3; 2)	29.1	40.5	20 000	27.5	32.8	9 622
(4; 3)	26.9	32.8	10 410	26.6	32.3	10 711
(6; 4)	24.6	31.3	12 170	26.0	31.3	10 417
(6; 4; 2)	24.3	31.1	19 885	34.1	31.0	16 584
(8; 4; 2)	25.0	32.2	8 206	23.5	30.3	17 276

in section 4.9, it is possible to assume that the first value is lower due to including some parameter combinations, outcome of which is easier to predict than for the default, that consequently lowered the average error. However, calculating the correction of prediction for the file size on the development set, which was performed specifically for default compression options (like during feature selection), resulted in 34.8%. This value is still less than original 39.1%, and comparing the error distributions corresponding to these values on figure 4.20 shows that the more complicated model involving three compression options and no extra content features corresponds to a noticeably narrower distribution in comparison with a simpler model (from fig. 4.12), which involved only segment features. The underlying blue graph on figure 4.20 corresponds to the histogram on figure 4.15.

This observation means that adding compression parameters as model inputs delays the moment of overfitting allowing to train more complex network configuration and leads to slightly better prediction accuracy.

The regression models for quality and time were trained similar to the those for file size using all the same input variables. Table 4.3 presents the results of selecting the optimal network configurations for these objectives. The RMSE of the best model for predicting quality was 0.91 dB in LMSSIM metric using MLP with two hidden layers containing 6 and 4 neurons. The same network structure was discovered to be optimal for the model predicting compression time. It resulted in average error of 0.80 s per video segment (not per frame).

The fact that optimal MLP for predicting file size has more hidden neurons and, consequently, degrees of freedom in comparison with models for other objectives suggests that file size may be more difficult to predict. This correlates with a similar observation made for images in the previous chapter.

Table 4.3: Selecting optimal network structure for the quality and compression time models.

Neurons in hidden layers	Models predicting quality (LMSSIM)			Models predicting compression time		
	Root-mean-squared error, dB		Training iterations	Root-mean-squared error, s		Training iterations
	Training set	Development set		Training set	Development set	
(3; 2)	0.69	0.98	19 913	0.70	0.87	9 158
(6; 4)	0.62	0.91	12 692	0.66	0.80	4 853
(8; 4; 2)	0.57	0.93	16 646	0.65	0.88	3 136

4.11 Experimental Results

This section presents two principally different approaches for balancing quality and compressed size between segments in a video.

The first approach is focused on obtaining better quality to size ratio on average for the whole video. It is based on the assumption that high- and low-detailed scenes have different rate of quality degradation upon lowering or increasing their bitrate. For example, decreasing the quality of a high complexity segment reduces file size by a considerable amount equivalent to that obtainable by decreasing quality of several segments with small amount of motion and smooth image. So, the size budget can be distributed across the video in a way that makes low complexity parts having better quality in comparison with the highly detailed ones. This allows us to obtain a compressed file size which is smaller than in case of having the same quality for all segments. Consequently, an obvious downside of this method is uneven quality distribution across different video segments.

The second approach considers the problem of unbalanced segment quality instead of just targeting an average value for the whole video. The priority is dedicated to compressing each segment with a given level of quality.

4.11.1 Targeting average frame quality

In this scenario we investigated the potential of the proposed dynamic resolution concept in terms of improving quality to size ratio. It was done under assumption that an average frame quality is the main objective. The distribution of quality was not taken into account.

Knowing the outcome of compression with different parameters for all video segments creates many optimisation possibilities. When each segment can be encoded with individual options the result could potentially be better than, say, a default

compression. At the scale of the whole video these options form a set of parameter combinations that corresponds to a group of video segments.

Testing of the dynamic resolution started with predicting compressed size, time and quality for video segments using the previously trained regression models. The predictions were obtained for all segments from the four test videos and using 150 parameter combinations from the parameter space on figure 4.16 excluding fast preset. In addition, all test segments were actually compressed with all those different options, and the results were recorded alongside the predictions.

As x265 codec produces deterministic result by default, having both actual and predicted outcomes of compression for each segment facilitated organization of the experiments. The necessary information could be subsequently retrieved faster and multiple times when necessary according to the logical flow of the experiment. This approach reduced time for the experiments allowing to conduct more tests and propose reasonable optimisation strategies.

In practice the optimisation procedure would consist of the following steps:

– making a preliminary analysis pass of the video splitting it onto segments and calculating features;

– predicting compression results for 150 parameter combinations in all segments;

– searching for an optimal set of parameter combinations that leads to the desired outcome;

– compressing segments with the corresponding options;

– recording total result characteristics for comparison.

Need for a search strategy

It is infeasible to perform an exhaustive search over all possible ways to compress several video segments with different options. Even the shortest test video "Sony Glass Blowing Demo" contains 20 segments. Each one of them can be encoded with 150 parameter combinations (or even more if necessary), which leads to 150^{20} possible sets for the whole video.

Therefore, a search algorithm should be employed. Two different techniques were used for this purpose: the genetic algorithm and the hill climbing search. They were guided by an error function (or inverted fitness) defines as follows:

$$error = \begin{cases} S_{predicted}, & \text{if } Q_{predicted} \geq Q_{target}; \\ S_{predicted} + 10^7 \cdot e^{-1000\left(Q_{target} - Q_{predicted}\right)}, & \text{otherwise;} \end{cases}$$

where S and Q are total video size and average quality respectively.

This metric uses compressed file size as a basis of the function that should be minimised. It applies a penalty of 10 MB even for 0.001 drop in predicted quality, which also exponentially increases. As a result, the search process brings quality to the required level first and then tries to minimise the file size. Figure 4.21 demonstrates an example of the hill climbing trends for size and quality.

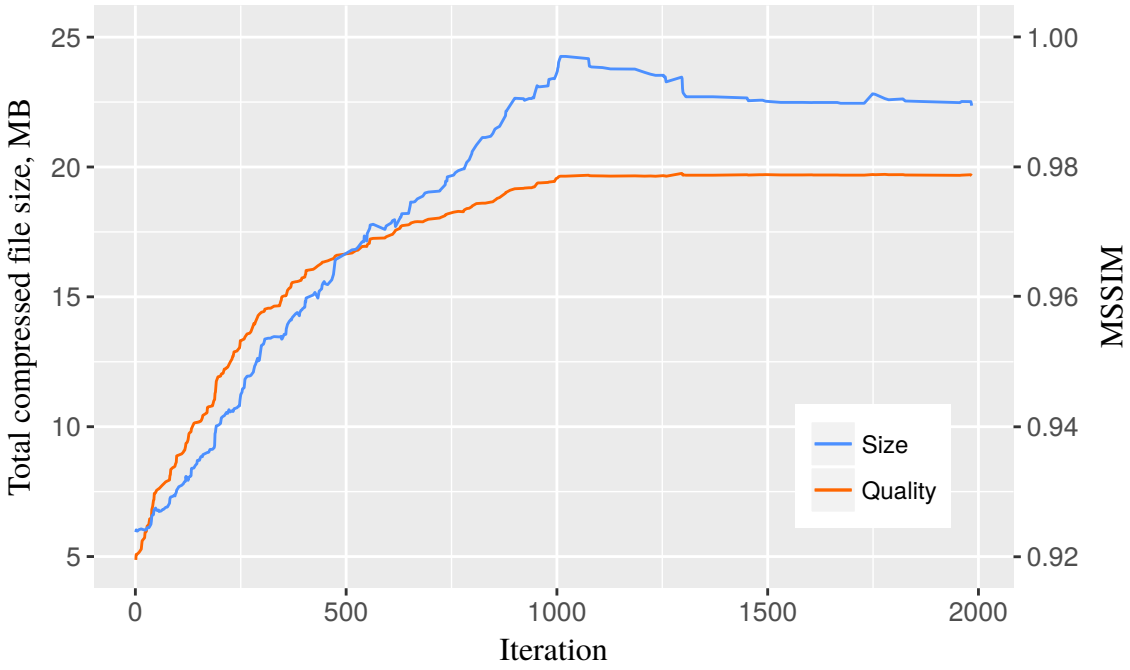


Figure 4.21: Hill climbing search trends for the compressed file size and average quality in MSSIM metric using random start and 4th test video as an example.

Genetic algorithm and hill climbing search

Considering a high efficiency of the x265 video codec "out of the box", it is reasonable to estimate how much room for improvement can be offered by the dynamic resolution concept and if there is any possibility for optimisation at all.

For this purpose a classic genetic algorithm (GA) was employed. It used all the standard attributes: a population of solutions, crossingover, mutation and ranking solutions by the smallest error, so that better ones were mixed more frequently. A solution represented a set of combinations of three parameters describing compression options for each video segment. For example, a crossover operator was randomly mixing segment options from two parent solutions to obtain a new one. The mutation procedure was changing one of three parameters in a random segment with 5% probability. The population size was 2000 and the maximum number of

generations – 200.

Due to the fact that assigning different selection probabilities to potential solutions in a population may be not an easy task, a custom approach was used in the experiments. See Appendix C for more details about selecting random solutions for crossingover.

In this search problem any change in the compression options of a single video segment has a relatively small influence on the total file size or quality. Due to this specific even the simple search strategies demonstrate good performance in terms of the number of iterations required to converge to a minimum file size value.

So, for the real world comparison the hill climbing algorithm (HC) was used instead of GA. It was empirically established that the GA performs marginally better than HC. However, because GA is a more complex and computationally expensive method, the HC was considered a more practical choice as it need less processing time. It was used to search for the optimal parameter combinations based on the predictions made with regression models.

The hill climbing algorithm changes a random compression parameter by one unit in a single or two randomly chosen segments at every iteration. This minor difference from the traditional HC decreases probability of the search process getting stuck in the local minima beforehand. It required between 1000-3000 iterations to converge depending on the video and the starting point (see example on fig. 4.21).

Comparing results with typical x265 compression

Table 4.4 presents characteristics of both the best possible and the real compression results for the dynamic resolution, which are compared with three ordinary compression options: default (CRF = 28), CRF = 36 and CRF = 44. The latter options correspond to the lower quality and size.

The row "standard x265 encoding" contains results of the normal compression for every test video without any modifications. The "best possible" scenario of the dynamic resolution shows theoretically obtained outcome found by the genetic algorithm under assumption that all video segments have been compressed beforehand with different options. It is quite expensive to do in practice, but such procedure allowed to see that all videos except "New York in 4K" demonstrate a potential for improvement in total file size and compression time. With increasing compression at higher values of CRF the possible dynamic resolution alternatives look even better. However, the time values shown in the "best results" do not include the exhaustive enumeration of parameter combinations for all segments or time for any video anal-

Table 4.4: Comparison summary of applying a search-based dynamic resolution method to four test videos.

#	Video name	Tested resolution	Compression scenario		Compression results with additional x265 quality control option (CRF)								
					--crf 28			--crf 36			--crf 44		
					MSSIM	Size, MB	Time, s	MSSIM	Size, MB	Time, s	MSSIM	Size, MB	Time, s
1	GoPro HERO5 + Karma: The Launch in 4K	1080p	Standard x265 encoding		0.967	100.0	714	0.926	33.6	578	0.866	10.5	477
			Dynamic resolution	Best possible	0.967	93.3	588	0.926	28.1	335	0.866	6.7	197
				Real result	0.967	97.7	720	0.926	46.5	522	0.866	11.5	242
		2160p	Standard x265 encoding		0.983	280.4	3004	0.961	104.2	2414	0.923	36.2	1949
			Ideal dynamic resolution case		0.983	266.0	2171	0.961	80.4	995	0.923	20.8	616
2	Horizon Zero Dawn PS4 Pro 4K Showcase	1080p	Standard x265 encoding		0.956	58.2	532	0.908	18.5	427	0.843	5.4	354
			Dynamic resolution	Best possible	0.956	57.0	422	0.908	16.1	216	0.843	3.4	117
				Real result	0.956	59.6	522	0.908	17.5	336	0.843	3.9	153
		2160p	Standard x265 encoding		0.972	176.6	2157	0.946	58.9	1710	0.907	18.7	1395
			Ideal dynamic resolution case		0.972	157.9	1376	0.946	44.3	645	0.907	10.7	389
3	New York in 4K	1080p	Standard x265 encoding		0.957	134.2	789	0.894	41.3	639	0.790	11.8	535
			Dynamic resolution	Best possible	0.957	135.2	793	0.894	41.5	507	0.790	9.5	263
				Real result	0.957	142.1	841	0.895	52.3	449	0.790	13.7	328
		2160p	Standard x265 encoding		0.974	387.0	3178	0.936	131.2	2570	0.865	41.0	2154
			Ideal dynamic resolution case		0.974	385.4	2900	0.936	123.3	1580	0.865	31.3	885
4	Sony Glass Blowing Demo	1080p	Standard x265 encoding		0.979	22.4	424	0.953	8.4	360	0.908	3.3	308
			Dynamic resolution	Best possible	0.979	20.3	321	0.953	6.3	189	0.908	1.7	114
				Real result	0.979	22.4	409	0.953	7.6	290	0.908	2.3	142
		2160p	Standard x265 encoding		0.988	64.4	1703	0.973	25.8	1420	0.946	10.7	1207
			Ideal dynamic resolution case		0.988	60.1	1211	0.973	17.3	575	0.946	5.0	361

ysis, or the GA search itself. This time is just a raw sum required to compress the segments with the discovered options.

Despite a considerable theoretical potential of the dynamic resolution, the actual tests based on the predicted values without any preliminarily obtained information about compression outcome for the segments are not very encouraging. The "real result" row shows that using dynamic resolution in practice does not allow to decrease the file size in comparison with the default scenario (see two highlighted rows for each video). It is possible only at lower quality levels corresponding to higher CRF values, although not for all videos.

The real world tests used hill climbing instead of GA for searching compression options. The C++ implementation needs roughly 1 second per 1000 HC iterations per 1000 video frames. A typical search needed 2000 iterations, so the time overhead was accounted as 2 s per 1000 frames and included in the total time. The feature extraction stage required about 5 s per 1000 Full HD frames. So, the total processing time combines compression of all segments and 7 s per 1000 frames overhead for feature extraction and HC search.

In the context of time measurement it is important to note that the time needed for splitting video into segments was considered negligibly small. The reason is that in order to identify an optimal position for a key frame the x265 codec must calculate a frame difference metric between all pairs of neighbour frames. This procedure is performed regardless of the number of frames. So the same time overhead is implicitly present when compressing video either as a whole or by individual segments.

The average quality in the table 4.4 was deliberately made the same in all scenarios to make an accurate file size comparison. The accuracy of fitting quality in particular was investigated in the next use case aiming for equal segment quality.

The difference between theoretical and practical cases, in particular considering worse file size, can be explained only by inaccurate predictions made by the regression model for estimating compressed size. However, there is no 30% error like in the predictor itself because the errors for multiple segments compensated each other resulting in a reasonable but not accurate enough estimations.

It is worth mentioning that the ideal case comparison was done for the original 4K videos as well (see two rows with 2160p test resolution for each video in the table 4.4). The potential for optimisations in terms of quality to size ratio as well as compression time look noticeably better than for the resampled 1080p. However, due to the low quality of these source videos it was assumed that the results will be biased by the fact that such material is highly compressible. If training video data also limited to low quality, it is easier to predict outcome of compression. This

circumstance prevented a proper investigation of the 4K resolution case.

In addition, the difficulties in gathering training statistics such as the need for large amounts of disk space and processing time complicated obtaining the regression models for the 4K resolution, so the decision was made to set aside this problem at least until the reasonable quality source material can be obtained.

Detailed video scenes and dynamic resolution

The 3rd test video in the table 4.4 – "New York in 4K" demonstrated the lack of even a possibility for optimisation in comparison with the default compression. This video consists mostly of the scenes with high detailization like buildings, trees and other objects with many edge elements. Considering the fact that resizing is a relatively coarse compression operation that eliminates the small details, it may not be possible to obtain a high quality after resampling the frames into smaller resolution and back.

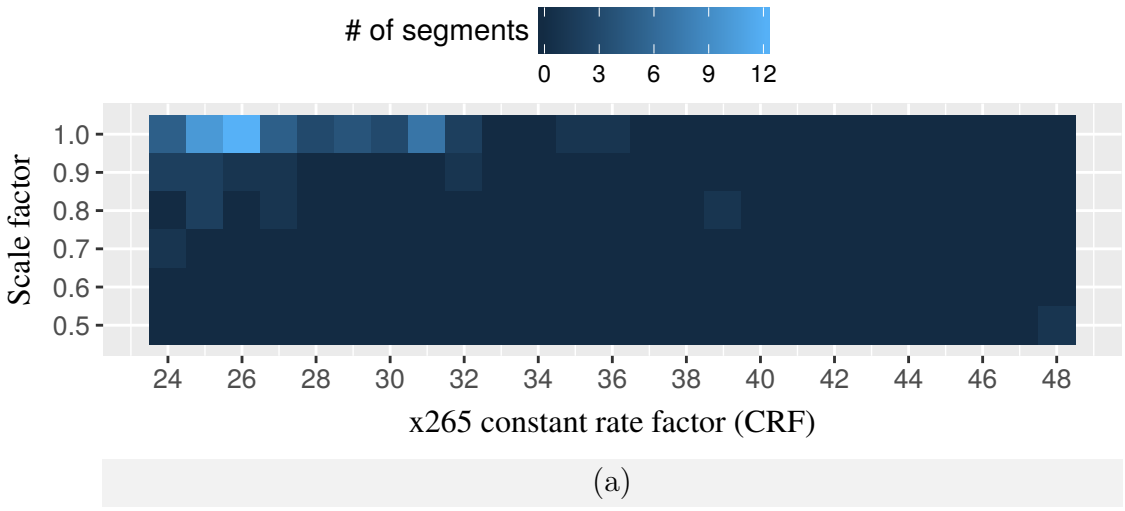
It is quite easy to conclude how suitable a particular video is for applying the dynamic resolution by looking at the optimal set of compression parameters discovered with the GA, which was aimed for a relatively high quality. Figure 4.22a illustrates this with a heatmap of the parameter combinations (scale; CRF) for all segments from the 3rd test video, which were obtained with the GA targeting the default average quality 0.957. Apparently, the optimal strategy for the vast majority of segments from this video is to use the original resolution for compression. Otherwise it will be impossible to meet the target quality 0.957.

If the resolution cannot be changed then the proposed strategy for balancing the segments has to operate only with the quantization control parameter CRF. But due to the fact that the internal x265 adaptive quantization is usually better than the external, on average the dynamic resolution performs slightly worse in terms of quality to size ratio.

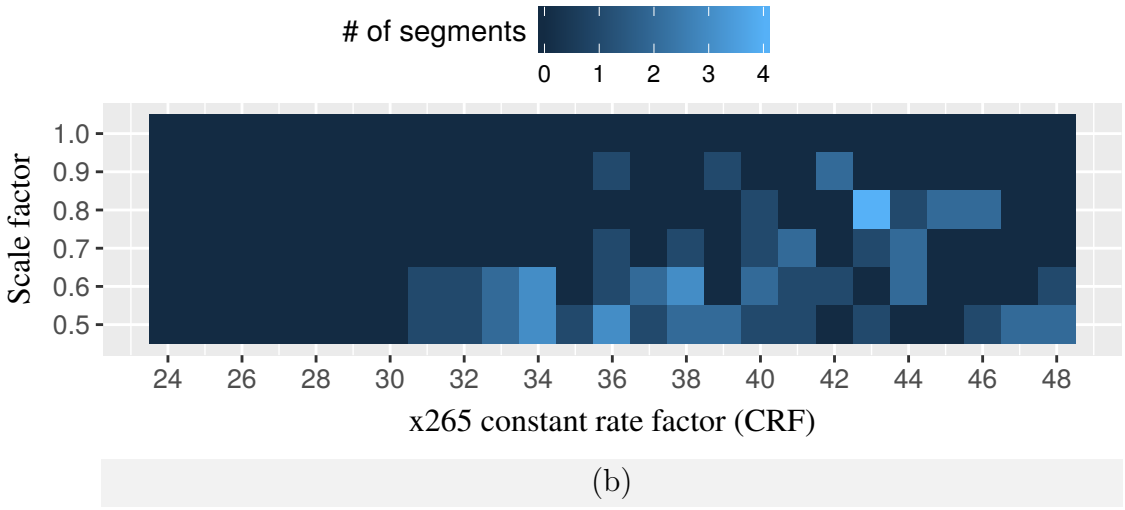
However, in case if target quality is lower than the default (say 0.790 corresponding to CRF = 44), the dynamic resolution demonstrates a possibility for optimisation even for complex scenes. Figure 4.22b shows that in such scenario the situation with the scale factors is the opposite – using the original resolution is not the optimal strategy for any of the segments. A wide range of scale and CRF values was used to balance average quality and file size for the low quality compression.

So, adding variable resolution as an additional compression parameter increased the optimisation space and allowed to discover better compression strategies for long videos. However, a considerable limitation of the dynamic resolution is the fact that

it is more suitable for compressing into low quality rather than high.



(a)



(b)

Figure 4.22: Heatmaps of the best parameter combinations found by the GA for 3rd test video with target quality 0.957 (a) and 0.790 (b).

Frame quality distribution

The distribution of the frame quality across the test videos was not used as a standalone objective in the experiments. It was done under assumption that the deviations of the segment quality from the average will stay in a reasonable range. Otherwise, a significant quality drop even in one of the segments will be reflected in the average quality due to a relatively small number of segments in the test videos.

Nevertheless, it is important to consider how the frame quality behaves in this

scenario. The graph on figure 4.23 plots the quality for individual frames from the shortest video "Sony Glass Blowing Demo". It compares the trends between default compression and the real world dynamic resolution test, when both resulted in the same size and average quality.

However, the standard deviation of the frame quality distribution was 0.014 for the dynamic resolution versus 0.013 for the default compression. The proposed method performed slightly worse in this objective, although it is an expectable result because this parameter was not constrained. In this light it is reasonable to conclude that aiming for a minimal size does not allow to obtain a constant quality.

Yet the more important fact is a very close correlation between the trends on figure 4.23. It was unexpected because according to the purpose of the default $\mathrm{CRF} = 28$ it should result in a more evenly distributed quality. Instead, the video compressed as a whole with the default settings has practically the same quality trend as the dynamic resolution version assembled from segments of various resolutions, which were encoded with different options, with a sole purpose to minimise file size at the cost of increasing quality distribution.

The x265 codec does not explicitly rely on MSSIM metric and does not use variable resolution. Despite these facts, there could be a couple of reasons explaining how such close correlation could happen.

Firstly, the codec may use some internal heuristic, which allocates quality levels among different parts of the video in a way that minimises total file size but with the intention to keep the average quality at a particular statistically determined level. It is unclear whether this is an intended behaviour (which would contradict the user manual) or simply a side effect of adjusting the quantizer based on a frame complexity. The latter is likely to be the case because x265 uses so-called psychovisual optimisations. In order to adjust the video to the human perception the codec applies higher quality degradation in the more complex scenes in favour of improving total compression ratio.

Secondly, the x265 intensively uses quad-tree partitioning of the frame blocks, which technically could do the same job as dynamic resolution. This technique partially compensates the diversity of smooth and highly detailed frames by using blocks of different sized for each frame type. It can be considered as an alternative method to the dynamic resolution causing unexpected similarities.

This interesting observation about the frame quality distribution and minimising file size was not found in the literature. Supposedly, the reason could be that the researchers usually test their algorithms on the specially selected individual video scenes (approximately 250-500 frames) instead of long videos.

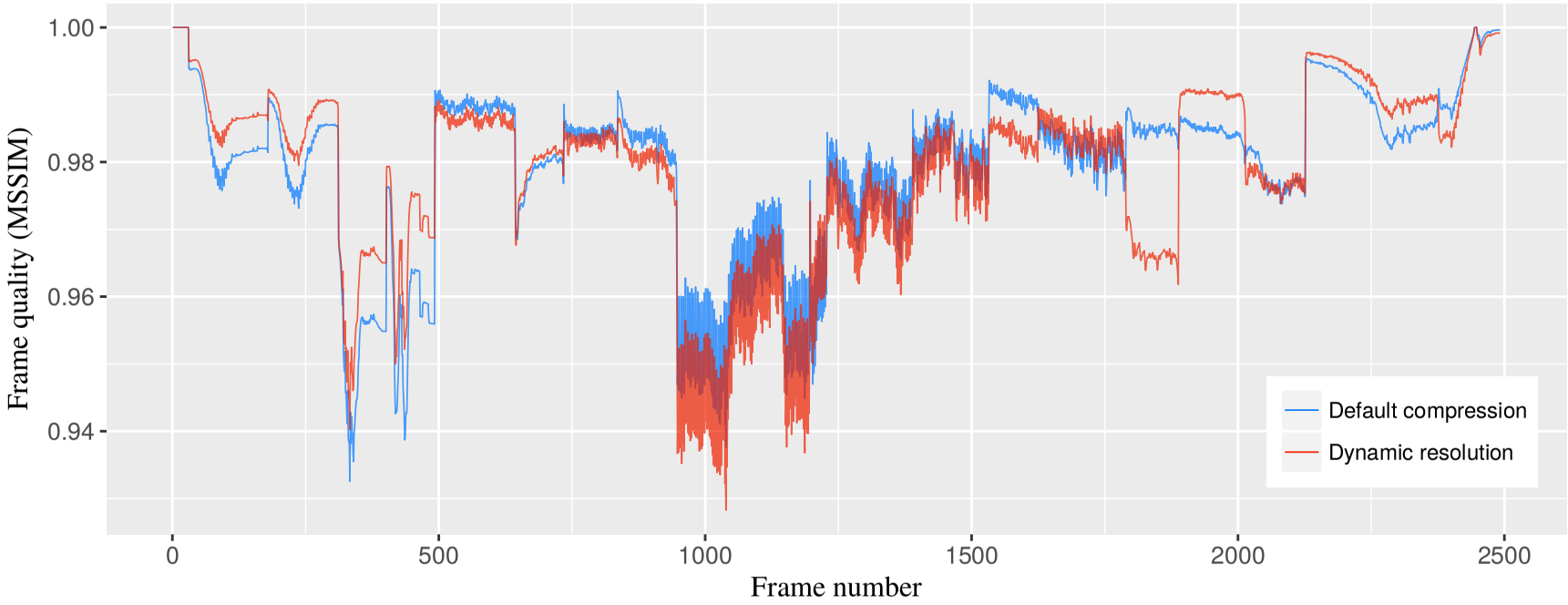


Figure 4.23: Similarity in the frame quality trends between the default x265 compression and per-segment encoding with the search-based dynamic resolution targeting minimal file size (4^{th} test video was used as an example).

Beyond medium preset

Besides the quality distribution, which was intentionally set aside in the considered use case, the compression time also was not properly taken into account as either a target objective or a part of the fitness function. The reason was the difficulty of including it into the fitness function. There is no obvious and universal way to add time into the formula as it would require focusing on a specifically made up narrow practical scenario, which was not a goal of this work.

Due to this specific, the fast preset was not used in these experiments. The preset parameter purpose is mostly the compression time management rather than quality to size ratio. Any possible decrease in the compression time from the table 4.4 is a side effect caused by smaller resolution in some segments.

However, in the next use case involving dynamic resolution the proposed method considers all four objectives during optimisation.

4.11.2 Targeting constant frame quality

This section proposes an approach for sequential segment optimisation without performing a search. Such technique can be useful for live video streaming or other time constrained scenarios. In addition we propose a concept of an alternative mechanism for controlling the balance between resulting quality, size and compression time by specifying user priorities.

Heuristic-based parameter selection

Consider a use case, in which conducting a preliminary analysis pass of the whole video is infeasible or too expensive and each video segment should be processed sequentially and independently from the forthcoming ones. In this situation only previous parts of the video can be a source of extra information. However, it may not be useful if the video consists of substantially different scenes. So, in this investigation it is assumed that each video segment must be analysed and compressed separately based only on its content and user preferences.

The complexity of this problem is in the fact that video compression is a process involving multiple objectives influencing each other. If user specifies a desired level of quality, it is still unclear how to obtain a reasonable balance between file size and encoding time because both of these characteristics should be minimal for each segment. Among the positive sides of such approach is the fact that the decision process guided mainly by target quality aims to reduce quality jumps between the segments.

Many live streaming services include a few seconds delay for video processing, so it is reasonable to assume that in practice a video segment can be analysed as a whole with intention to calculate the content features (at a cost of approximately 5 ms per 1080p frame as measured previously).

After extracting content features from a video segment, the results of compressing it with several parameter combinations can be predicted in a fraction of a second using pre-trained regression models. At this point it is possible to identify multiple sets of options, which are expected to compress the segment close enough to the desired level of quality. This situation usually takes place when using more than one compression option in the video codec.

The problem of choosing an optimal combination is not trivial. The obvious cases with the highest quality or lowest file size are often the worst in terms of quality to size ratio. One of the ways to deal with this task is to use a problem-specific heuristic to rank all parameter combinations for a particular video segment. Such heuristic basically includes some a priori information about preferences between resulting size, quality and time. For example, quality to size ratio is a reasonable candidate for a metric that allows to rank different compression options:

$$rank = \frac{Q}{S},$$

where Q and S are predicted quality and compressed size per frame respectively.

However, using this simple ratio still leads mainly to the edge cases with maximum quality and consequently very large size or, alternatively, minimal file size and very low quality. There is no typical default value for this ratio either because it can vary in a wide range for different segments.

Nevertheless, it is possible to use such ratio as a fitness metric assuming that the user has provided a desired quality level. This allows to limit the choice of possible quality values. For example, consider a Gaussian function that can be used to measure a deviation from the given quality:

$$G(Q, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(Q-\mu)^2}{2\sigma^2}},$$

where μ is a target average quality and σ defines a range of acceptable deviations (in the experiments it was set to $\sigma = 0.005$). This function can be used as a coefficient for eliminating any edge cases when ranking different compression options:

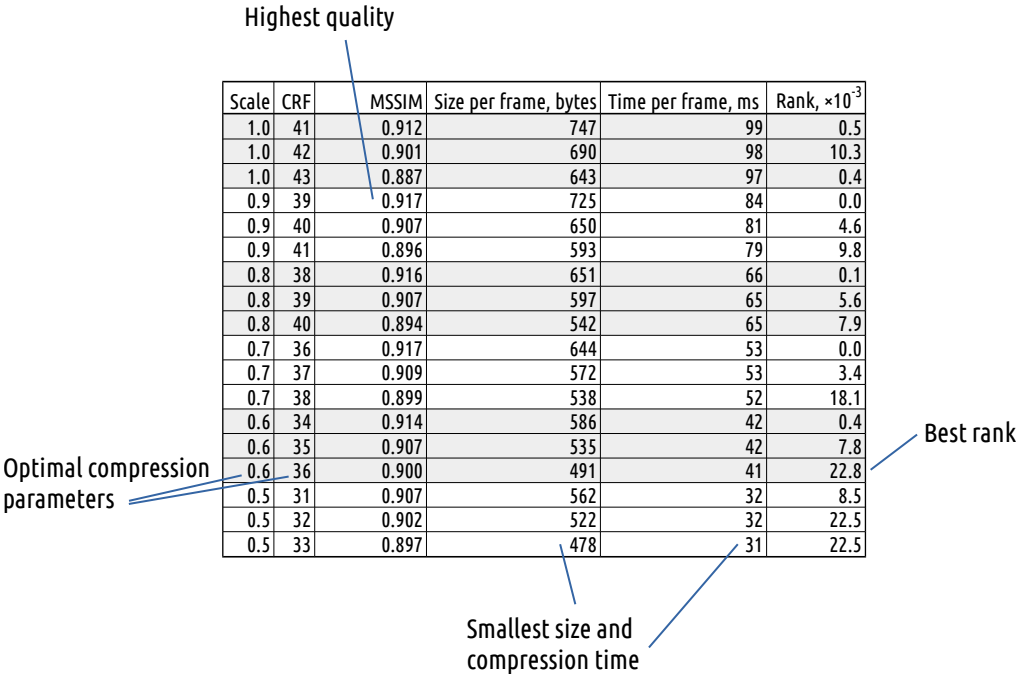
$$rank = \frac{Q}{S} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(Q-\mu)^2}{2\sigma^2}}.$$

In order to take into account the compression time T, it was also included into the formula. Time should be inversely proportional to the rank of a parameter

combination, so it was placed in the denominator although raised to the power 0.5 to slightly decrease its influence in comparison with quality to size ratio:

$$rank = \frac{Q}{S \cdot \sqrt{T}} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(Q-\mu)^2}{2\sigma^2}}.$$

This formula was used to find optimal parameter combinations for all segments in the test videos. Figure 4.24 shows a typical example of ranking several combinations of two compression options: scale factor and CRF, which correspond to the encoding results with approximately the same average quality. Assuming that the user provided 0.9 as the target quality level, it is hard to select the optimal case without any extra information. However, calculating the rank values for all scenarios helps to make a definitive choice of the optimal compression parameters. So, the proposed ranking heuristic represents some reasonably constructed predefined priorities, which facilitate the problem.



Scale	CRF	MSSIM	Size per frame, bytes	Time per frame, ms	Rank, ×10^{-3}
1.0	41	0.912	747	99	0.5
1.0	42	0.901	690	98	10.3
1.0	43	0.887	643	97	0.4
0.9	39	0.917	725	84	0.0
0.9	40	0.907	650	81	4.6
0.9	41	0.896	593	79	9.8
0.8	38	0.916	651	66	0.1
0.8	39	0.907	597	65	5.6
0.8	40	0.894	542	65	7.9
0.7	36	0.917	644	53	0.0
0.7	37	0.909	572	53	3.4
0.7	38	0.899	538	52	18.1
0.6	34	0.914	586	42	0.4
0.6	35	0.907	535	42	7.8
0.6	36	0.900	491	41	22.8
0.5	31	0.907	562	32	8.5
0.5	32	0.902	522	32	22.5
0.5	33	0.897	478	31	22.5

Figure 4.24: Example of choosing optimal compression parameters among several alternatives by using heuristic-based ranking (assuming $\mu = 0.9$ and $\sigma = 0.005$).

Dynamic resolution video with heuristic optimisation

Using the proposed heuristic it is possible to compress each video segment separately from others. In order to show that such approach allows to obtain a decent result at the scale of a whole video, it was compared with the x265 default

compression as well as the fast preset. The latter option was included for time comparison purposes because it was initially considered as a compression parameter for all test segments.

The following procedure was employed for compressing test videos into dynamic resolution with heuristic optimisation:

– split a test video onto segments;

– predict compression outcome for 300 parameter combinations in all segments;

– choose the best combination for each segment using heuristic;

– compress each segment with the corresponding options;

– record total average quality, file size and compression time.

Table 4.5 compares two ordinary compression scenarios (medium and fast presets) with the results of applying the proposed balancing algorithm to the dynamic resolution video. The target quality for each test video was set to the corresponding average quality of the default compressed video. The total time reported in the table includes the feature extraction time of 5 ms per frame.

Table 4.5: Comparison summary of applying dynamic resolution method with heuristic-based quality balancing to four test videos.

#	Video	Compression scenario	Comp. time, s	Comp. size, MB	Frame quality (MSSIM)	
					Average	Stand. deviation
1	GoPro HERO5 + Karma: The Launch in 4K	Default	714	100.0	0.967	0.018
		Fast preset	503	94.3	0.963	0.021
		Dynamic resolution	488	102.9	0.962	0.014
2	Horizon Zero Dawn PS4 Pro 4K Showcase	Default	532	58.2	0.956	0.013
		Fast preset	364	53.1	0.951	0.014
		Dynamic resolution	316	47.4	0.946	0.014
3	New York in 4K	Default	789	134.2	0.957	0.019
		Fast preset	580	123.9	0.951	0.022
		Dynamic resolution	635	163.3	0.957	0.010
4	Sony Glass Blowing Demo	Default	424	22.4	0.979	0.013
		Fast preset	295	21.5	0.977	0.014
		Dynamic resolution	234	22.0	0.971	0.009

In general, the dynamic resolution video requires slightly less processing time than the default scenario and reduces the spread of the frame quality values. This fact is an important evidence supporting the Predictable Compression Hypothesis. However, using the proposed approach results in a larger file size and slightly lower average quality.

There are some exceptions from these observations. For example, the second test video has the largest deviation from the target level of quality and does not improve frame quality distribution. This occurred because the game footage was not present in the dataset used for training the models, which lowered the prediction accuracy and did not allow to properly match the target quality.

The third test video with the most complex and detailed content was the easiest in terms of predicting the outcome of segment compression, which is indicated by exact match of the average quality with the default value and a substantially lower frame quality distribution. However, it resulted in a considerably larger file size. This correlates with a similar result in the previous search-based segment balancing (table 4.4).

It seems that narrowing the frame quality range is the most important advantage of the proposed heuristic balancing. This was engineered by design through including a Gaussian function into the ranking metric, and was adjusted deliberately by limiting possible quality values for each segment with $\sigma = 0.005$.

Figure 4.25 compares the quality per frame for the longest "New York in 4K" video between dynamic resolution and default compression approach. A noticeably smaller amplitude of the quality oscillations in case of the heuristic-based balancing indicates that the algorithm regulates the quality of all segments to make it more-or-less even. The differences in frame quality inside the segments are due to x265 internal balancing, which was not modified in the proposed method.

Figure 4.26 allows to see quality variations in more details for the smallest tested video "Sony Glass Blowing Demo". At the end of the graph there is an example of incorrectly predicted segment quality, represented as a noticeable drop. It is a reminder that the proposed technique is based on a statistical methodology and does not always produce an ideal result.

User priorities for multiobjective optimisation

Due to the fact that the proposed heuristic is just a type of a priori assumption, it can be modified to adjust the priorities between quality, file size and time. It was empirically established that using the ratio $\frac{Q}{S\sqrt{T}}$ in ranking compression parameters provides a reasonable tradeoff between quality, size and time, at least in author's opinion. However, some practical scenarios may need a different balance between target objectives. For example, compression for online publishing should be less strict about time but more demanding towards improving compression ratio because video will be viewed multiple times.

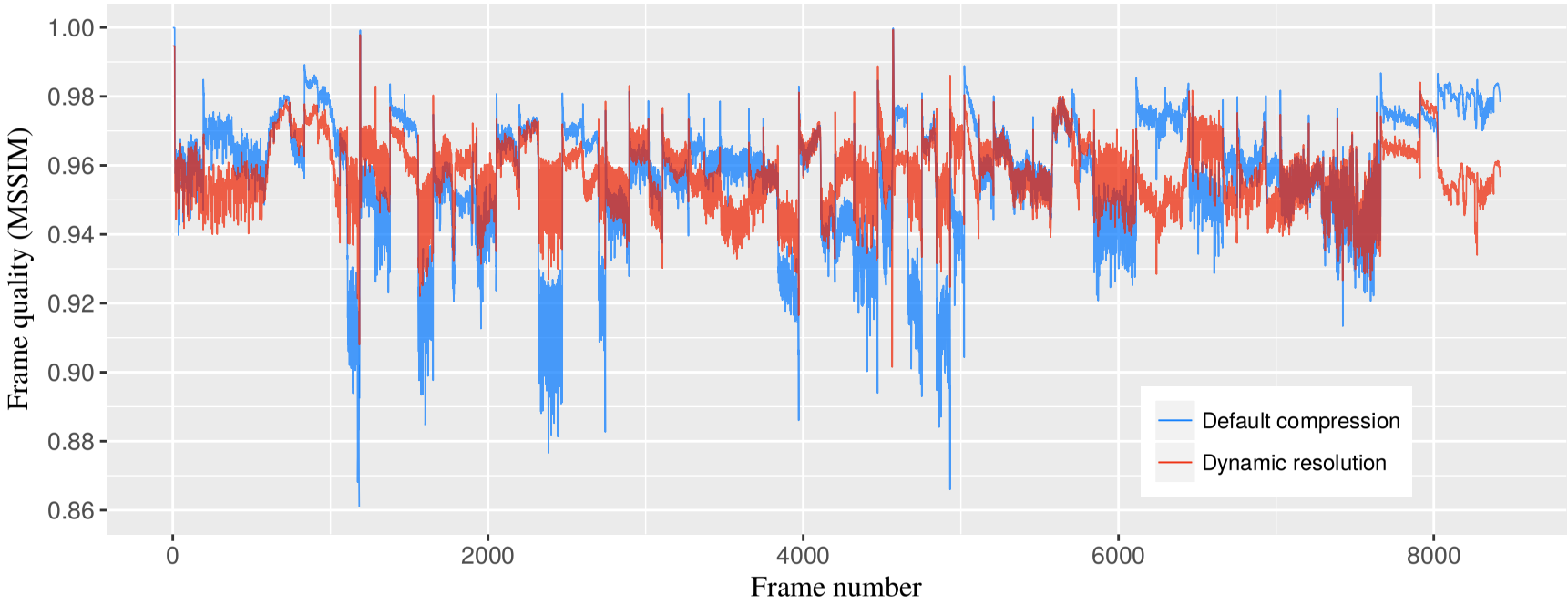


Figure 4.25: Comparison of the frame quality trends between default x265 compression and per-segment encoding with dynamic resolution using 3rd test video as an example.

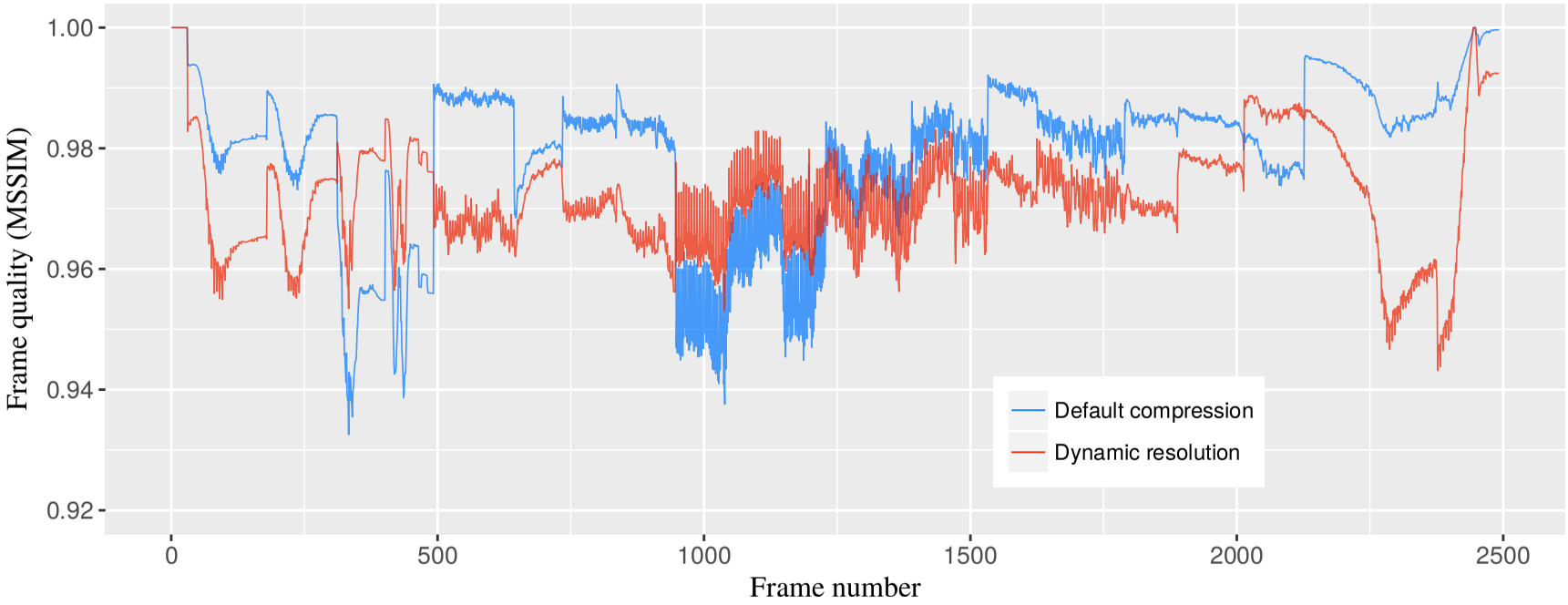


Figure 4.26: Comparison of the frame quality trends between default x265 compression and per-segment encoding with dynamic resolution using 4^{th} test video as an example.

So, in order to make balancing more flexible from the user perspective, a concept of relative priorities was introduced. The priorities are specified in a form of three independent variables α, β, $\gamma \in [0; 1]$, which are used as powers for the objectives Q, S, T in the rank formula:

$$rank = \frac{Q^{\alpha}}{S^{\beta} \cdot T^{\gamma}} \cdot \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(Q-\mu)^2}{2\sigma^2}}.$$

Decreasing any of the power values will reduce the influence of the corresponding characteristic on the final score, and setting any of them to zero eliminates the objective from the equation. Assigning the same value (except zero) to all variables, say $\alpha = \beta = \gamma = 0.5$ is equivalent to $\alpha = \beta = \gamma = 1$.

The variables can be represented as three independent sliders (fig. 4.27), which can be regulated manually. Like many video codec options these values are still abstract parameters, but in some cases they can be an informative mean of controlling compression. For example, if an entire video can be analysed, it is possible to calculate the resulting video characteristics based on per-segment predictions in real time upon moving the sliders. Such hint should make compression process more transparent to the user.

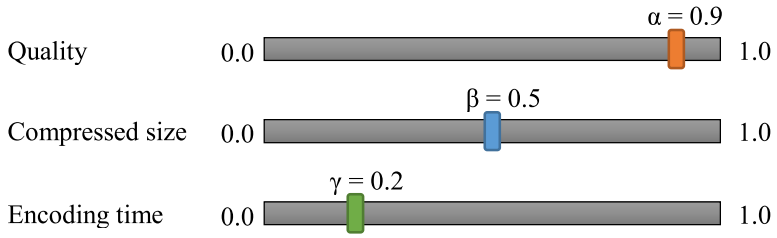


Figure 4.27: Concept of the three sliders to specify priorities for each target objective.

Several priority combinations were tested with dynamic resolution video compression. Table 4.6 contains examples of the compressed video characteristics for the short test video "Sony Glass Blowing Demo" using different priorities. The target quality was set to 0.979 as in the default encoding. Consider, for example, the last two rows in the table – decreasing size priority β from 1.0 to 0.2 resulted in larger file size but smaller time and better average quality.

It is important to emphasise that the main objective in this case is still a target level of quality provided by user. Varying the relative priorities only modifies the heuristic for ranking compression option for each video segment. This alters the process of choosing between similar alternative outcomes, so the actual compression result can sustain only minor changes.

Table 4.6: Comparison of the compression results obtained with different priority combinations using 4^{th} test video as an example.

Priorities	Comp. time, s	Comp. size, MB	MSSIM
$\alpha = 1; \beta = 1; \gamma = 0$	302	21.9	0.972
$\alpha = 1; \beta = 1; \gamma = 1$	213	22.0	0.971
$\alpha = 1; \beta = 1; \gamma = 0.5$	234	22.0	0.971
$\alpha = 1; \beta = 1; \gamma = 0.2$	253	21.8	0.972
$\alpha = 1; \beta = 0.2; \gamma = 0.2$	233	24.2	0.974

4.12 Discussion

The goal of this chapter was to investigate how predicting video compression can be used for improving its result, and to develop methods for controlling the outcome of compression in a more explicit way than through the quantization parameters.

The concept of dynamic resolution allowed us to see some of the optimisation possibilities when using one of the fastest and efficient video codecs in the world. However, obtaining a solid improvement in quality to size ratio in actual tests is difficult because it relies on the accuracy of approximate statistical estimations.

Predicting the outcome of video compression, especially file size, appears to be a complicated problem. Due to the low accuracy of the regression models the optimisation potential of the dynamic resolution in some scenarios remains unreachable in practice without multiple compressions. However, for low quality encoding the dynamic resolution concept is useful even in the proposed implementation, at least in terms of the reduced processing time.

It may seem that a major limitation of the dynamic resolution concept is that it is not suitable for high quality compression. Technically, it is not a fundamental problem but the result of a particular experimental design. For example, it is possible to upscale some high quality segments before compression to preserve their details even better than at the 100% scale and then resize them back to the display resolution, thus removing limitations for the highest quality achievable with dynamic resolution. However, such scenario has not been investigated in this research.

An important factor in controlling the outcome of video compression in addition to size, time and average quality is the frame quality distribution, which is not a commonly considered characteristic. In this work investigating the problem of constant quality for all parts of the compressed video allowed the introduction of the concept of relative user priorities to balance the free objectives like file size and compression time.

The focus on using a fixed given quality as a main reference target was due to the fact that quality metric values use non-linear scale and are difficult to compare unless they are the same. Aiming for a given file size or time requires some additional constraints for the quality distribution, otherwise it has tendency to increase like in one of the considered use cases.

Although such experiments were not conducted as part of this research it would also be possible to target a specific total file size or even a limited time budget. This, however, would require a preliminary analysis of the whole video. Moreover, the accuracy of the statistical models complicates a reliable investigation of such scenarios.

4.13 Summary

This chapter investigated video compression with a target level of quality using the x265 video codec. There are two principally different compression scenarios that are possible when aiming for a specific average quality. The difference is in considering the frame quality distribution. If the quality level of different segments in the video is not constrained, this leads to a similar result as produced by the x265 default strategy. The total compressed file size is minimised at the cost of increasing frame quality distribution. In case if the priority is a fixed quality across the entire video, the compression strategy requires a heuristic to choose compression options for each segment among several alternatives with similar quality but different size and encoding time.

Considering the fact that the first optimisation strategy aiming for a simple average quality requires a search to minimise the free objectives, it usually needs more computational time than the default compression. The experimental results show that it becomes useful only when dealing with low quality compression.

The alternative optimisation strategy that processes each video segment individually looks more promising. By considering all four characteristics – time, compressed size, frame quality average and distribution, it is possible to decrease quality fluctuations inside a video. This can be done in a comparable time or even faster than using either default or fast codec options (table 4.5). The price is a slightly reduced quality and, in some cases, increased file size. However, the error between actual and a target quality in the real world experiments is at a reasonable level of approximately -0.01 in MSSIM metric, which means it is practically impossible to see the difference.

So, it was experimentally established that video compression with given quality

can be done not just in a single compression run but sometimes even faster. Although among the factors that influenced compression time was a dynamic resolution, which allowed smaller size and compression time for some segments, this fact does not detract from the significance of the obtained results. On the contrary, it shows how the proposed dynamic resolution concept can be useful for reducing time and energy required for video compression.

4.14 List of Contributions

The following results were obtained in this chapter:

- a new technique for predicting the outcome of video compression by using a set of low-complexity features calculated directly from the uncompressed content;
- a novel concept of the dynamic resolution video, allowing the achievement of a smaller compression time in comparison with the default options;
- a new method for one-shot video compression with a target quality level based on the prediction models and dynamic resolution;
- the discovery that predicting compressed video characteristics is a substantially more difficult problem in comparison with the same problem for images;
- that the search-based method proposed and investigated in this chapter is unable to consistently outperform the default x265 compression in terms of the file size and compression time when maintaining the same average quality;
- that dynamic resolution concept is more suitable for low quality compression than medium or high;
- that using heuristic-based quality balancing minimises time overheads for the video analysis and allows to obtain a compression result with smaller quality fluctuations even quicker than using either default or fast x265 options, although at a cost of small decrease in average quality and compression level;
- that heuristics can be adjusted to control the balance between the free objectives like file size and compression time while maintaining the target quality.

Chapter 5

Conclusions

The thesis proposes a novel methodology for image and video compression with explicitly specified target objectives, without requiring multiple compression passes – i.e. in a time and energy efficient way.

The majority of the existing research into the problem of image and video compression optimisation aims to investigate the potential improvements in the quality to size ratio rather than predictability of the compression process. However, the latter problem is gaining popularity because it allows to reduce compression time in various practical scenarios using only existing compression standards.

This work is dedicated to the methods of time-optimised application of existing image and video standards and investigates possible video compression improvements using the dynamic resolution concept, and how it interacts with compression.

The growing interest particularly to video compression optimisations in the companies like Google and Netflix indicates that the related problems considered in this thesis are actual and relevant to the present day challenges.

5.1 Summary of Results

There are two types of approaches for improving image and video compression considered in the related literature: creating new methods and improving existing ones. The first is dedicated to developing new image compression algorithms that can outperform the existing standards mainly in terms of quality-to-size ratio. The second approach includes investigation into the possibilities of compression optimisation for already existing and widely used image and video formats. There are also two variations of the second approach – it may involve certain modifications of the compressor like in MozJPEG, or it can be implemented as a completely external

procedure like the JPEG transcoding.

The research methods that do not rely on any compressor modifications typically use machine learning techniques or other statistical analysis in order to model some regularities related to the compression process. However, in the previous research such methods are connected to the properties and functionality of a particular image or video codec like JPEG (transcoding) or x264 (YouTube optimiser). This thesis presented a novel universal methodology that is not attached to any specific compression standards but considers their algorithms as "black boxes".

One of the main features in this research was application of the simple machine learning models for predicting compression outcome characteristics without performing any actual compression. Another important aspect was designing a unique set of computationally cheap image and video content features, which are extracted from the raw pixel data without any substantial preprocessing. This approach allowed to minimise complexity and execution time of the proposed methods.

All techniques proposed in this thesis rely on using a single compression pass over the encoded data following the Predictable Compression Hypothesis. The assumption for targeting specifically a one-shot compression was based on the fact that even two codec passes may be too expensive in an energy sensitive environment or real time compression scenario. The unique feature of this work is a combination of a single compression pass with a deliberate minimisation of the time overhead for analysing raw data and making decisions about optimal compression parameters.

Current research is dedicated mainly to the problem of compressing images and videos with the desired objective characteristics. The methods used to obtain a target compression result are based primarily on image or video complexity, or in other words – entropy. The complexity is estimated from the uncompressed data without taking into account the details of a particular codec implementation except, of course, the specific compressor parameters and some general assumptions about the compression process like using the YC_bC_r colour space and the concept of I-frames in video compression. This fact leads to the universality of the proposed methods and eliminates the necessity to introduce any modifications in the compression algorithms. This property substantially differs current research from the related work.

This research presented the low-complexity techniques for analysing image and video content and explicitly predicting such compression characteristics as file size, quality and encoding time with respect to different compression parameters. The predicting mechanism uses the numerical regression models based on shallow feed-forward neural networks, which are relatively simple to train and computationally

cheap when evaluating predictions in practice. The numerical regression is used for predicting various compression characteristics in a unified manner, i.e. with the same or similar network configurations. This property makes the proposed methods suitable for using different quality metrics and a wide range of compression options.

The ability to quickly predict the outcome of compression creates a "bridge" through the complexity of a compression algorithm between the input image or video and the expected result characteristics. This property allowed to introduce methods for searching optimal values for either a single compression parameter or a parameter combination, which satisfy a target objective.

The proposed methods were tested on a range of randomly selected images and videos. Two popular lossy image formats – JPEG and WebP were used for the image compression experiments. A distinctive property that JPEG and WebP codecs have in common is a quality factor parameter between 0 and 100. Each value of the quality factor determines a one-to-one correspondence between compressed size and quality. This property considerably simplified design of image compression experiments in comparison with the videos.

The experiments on predictable video compression were based on the state-of-the-art x265 video codec. This allowed to investigate non-trivial multiobjective optimisation scenarios, which were made possible due to using several compression options that affect compression outcome in different ways. It is a complex task to obtain a desired video compression result based on a single target objective like quality because it is unclear how to balance the remaining characteristics like file size and compression time.

It was experimentally established that for the image compression scenarios the proposed machine learning approach demonstrates superiority over the closely related JPEG transcoding method in terms of accuracy of satisfying target file size or quality. In addition, it is reasonable to recommend the proposed system instead of using a constant JPEG quality factor, which is still a common practice, because the machine learning method accurately fits a given quality level while fixed quantization does not imply constant quality.

In terms of comparing the proposed method with multipass compression using a particularly suitable WebP codec implementation as an example, the experimental evidence confirmed the expected substantial improvement in compression time in case of one-shot compression, even with a minor time overhead for content analysis. This can be seen as an essential fact for the confirmation of the Predictable Compression Hypothesis at least in relation to image compression.

It was empirically demonstrated that the proposed method is equally well ap-

plicable to a wide range of image resolutions from small 0.24 MPix images to very big 24 MPix photographs. It is an important fact considering lack of attention in the related work to the compression optimisation in very large images. In order to prove applicability of the designed system in practice to different image codecs, an ACACIA image compression tool was created based on the prediction models utilised in the experiments.

The idea of statistical optimisation methods is particularly suitable for video compression. An ordinary video consists of multiple segments often depicting different content, which creates appropriate conditions for various balancing strategies. Although statistical methods are not always accurate, the errors tend to compensate each other in the long run. In order to increase the number of optimisation possibilities, the concept of dynamic resolution video was implemented in all video experiments.

This thesis considers only the problem of video compression with target quality level, which is the most applicable practical scenario. It proposes a new method for single pass video compression, which is based on the conceptually similar techniques of predicting video characteristics without actual compression. This problem, however, appeared to be substantially more complicated in comparison with the previous image case. It resulted in high errors of predictions, although the main concept remained functional.

Using several video compression options leads to a multiobjective optimisation problem. In order to deal with it the different strategies can be implemented. This research considers a search-based and a heuristic-based quality balancing methods. The first approach used a hill climbing search to find a set of parameter combinations for all video segments that minimise total compressed size while meeting a given level of average quality. The experimental results show that the proposed system is unable to outperform default x265 compression on a constant basis.

Applying the alternative strategy, which uses a specially designed heuristic to constrain quality level of the video segments, demonstrates a more predictable behaviour by making the frame quality more evenly balanced across the video. According to this concept, the video is processed sequentially segment by segment, which reduces time overhead in comparison with a search. This property allowed to minimise the decision time in the heuristic-based strategy. The experimental results show that this method outperformed the default x265 compression on all test videos in terms of total compression time while meeting a target quality with reasonable accuracy. A considerable downside of this approach in comparison with the default encoding is increased file size, which is caused by segment quality rebalance and low prediction accuracy.

The possibility to compress a video into the specified quality at a time comparable with ordinary compression operation provides extra evidence in support of the Predictable Compression Hypothesis. Of course, there are other aspects related to this hypothesis that were not fully investigated, like higher accuracy and compression within a given time budget, but it is reasonable to conclude that there is a solid approach that allows to confirm the hypothesis through implementing the necessary functionality using the proposed methodology.

In general, the heuristic-based method can be recommended for practical use even for real time video compression because of its reliability in terms of the frame quality distribution as well as the flexibility in balancing between file size and compression time using the concept of user-defined priorities.

Speaking of the video compression optimisations in general, there does not seem to be a possibility for "free" improvements – i.e. a strategy that performs better than the default compression in one of the objectives without worsening the others.

5.2 Relation to Other Work

Despite many distinct features this thesis remains related to the existing research. In particular, the most related works are the JPEG transcoder by Pigeon et al. [24] for image compression and the YouTube optimiser by Covell et al. [30] in case of video compression. While the image transcoding method was investigated and thoroughly compared with the proposed methodology in Chapter 3, the video optimisation approach by Covell et al. was impossible to replicate due to lack of details. Nevertheless, it is possible to make an approximate comparison based on the reported results.

Thus, according Covell et al. [30], the obtained model for predicting compressed video bitrate has 20% error in 80% of cases. Assuming normal distribution of errors, it is possible to calculate (of course, with some degree of uncertainty) that the standard deviation of such error distribution is 15%. This error is twice less than the 30% correction of prediction (which is also σ) for the file size regression model obtained in this thesis. Despite the possibility that predicting file size for the x264 and x265 video codecs may slightly differ, the work by Covell et al. demonstrated a relatively good result. However, the main difference from this paper is that the methods presented in the thesis do not use video precompressions. Moreover, the feature extraction stage in [30] should be relatively expensive, while in the current research the features are calculated relatively fast directly from the raw video content.

As for the features, their design was influenced by a range of literature. The existing idea of using convolutions as a proxy for the image complexity was further developed into a set of computationally cheap content features that hold useful information for the statistical models.

The intention to reduce computational expenses even for relatively small problems remains very important today. This can be demonstrated with an example of a new time-optimised image upscaling method by Romano et al. [51] from Google that was discussed in the section 2.6. This problem, no matter how narrowly specialized, was worth the effort to develop a solution that works a fraction of a second faster than analogues. Following this principle, a reasonable amount of effort was invested into optimising experiments and proposed solutions in this thesis.

Targeting time and energy efficient solutions is a noticeable trend in the modern research literature. For example, Ejembi and Bhatti [35] recommended to increase energy awareness of the producers and consumers of the video content. This partially inspired the idea of the dynamic resolution video presented in Chapter 4. The variable resolution video should not just be faster to compress but it is also expected to consume less energy for decompression upon playback, which can, for example, increase the battery life of the mobile devices without affecting user experience.

5.3 Theoretical and Practical Significance

The most important outcomes of this research are as follows:

- a single-pass compression with the target objectives;
- a universal methodology for predicting image and video compression.

This thesis presents a comprehensive investigation of the time-oriented compression optimisations for some of the popular image and video codecs. One of the key features of this work is a systematic analysis of the possible practical implications of the proposed methods. In particular, we consider a full spectrum of the compression process characteristics, such as quality, file size, compression time and even frame quality distribution for video codecs. As the optimisation process involves balancing these objectives, the research draws attention to the way they influence each other.

By comparing methods used to deal with the problems of image and video compression targeting a specific level of quality, which were presented in this research, it is possible to conclude that using a single compression parameter (in case of images) leads to a substantially simpler method of a single-objective optimisation that relies directly on the statistical models without involving any additional heuristics.

However, using more than one encoding parameter, as shown in the the video

compression investigation, lead to the development of a distinct and considerably more complicated strategy for multiobjective optimisation that required extra a priori information in the form of a handmade heuristic formula. The problem is that different parameter combinations can produce the desired outcome. This phenomenon is caused by intricate interactions between several compression parameters, which is difficult to describe or put under control, especially taking into account an additional factor of different video content subject to compression with these parameters.

The relation between the number of compression parameters and the potential complexity of the optimisation strategies is an important theoretical outcome, which has not been pointed out in the related work. This fact should increase the awareness of the complex interactions between compression parameters and subsequent difficulties with predictability of the compression process.

Considered properties of the optimisation problems should improve understanding of the challenges arising when compressing images and videos with specific target objectives. An example of such challenge can be the necessity to specify additional priorities for multiobjective optimisation.

The proposed methods for image encoding with a desired file size or quality and for the sequential video compression with the target quality have a direct practical application, the main benefit of which is a reduced processing time due to using prediction techniques instead of several compression passes to meet the specified objective. An example of the program implementation is the ACACIA tool that was specifically designed as a proof of concept. Although the analogous application for video compression was not created, it would be possible to achieve with some extra engineering effort.

Technically, the proposed methods can also be integrated directly into the respective compressors, which will allow to implement new compression scenarios with explicit control of the compression outcome. Supposedly, the idea of predictable compression can also facilitate optimisation of the experimental and emerging multimedia compression standards.

5.4 Limitations of the Considered Techniques

The first obvious limitation of the techniques proposed in this research is the fact that they were build on top of the the statistical methods. This means that they cannot be 100% accurate and reliable in current implementation. This however was an intentional tradeoff between accuracy and potential practical efficiency.

Another fundamental limitation is derived from one of the main advantages of the proposed methods – unmodified compressor functionality. This means that all optimisation strategies are bounded by the efficiency and capabilities of a particular image or video codec.

The quality evaluation procedures used in the thesis did not employ human judgment, which would likely be a more reliable approach than conventional quality metrics. Real people were not involved in the research process due to the specifics of the experiments and large volumes of data suitable only for automated processing. However, technically the proposed methodology can be used with various quality metrics including the mean opinion score.

Probably the most significant unsolved problem is a relatively low accuracy of the regression models predicting outcome of video compression.

The problem of image compression with resizing was not investigated, although such functionality was considered in the related work. However, the scaling factor was used within the concept of dynamic resolution video. The image compression experiments instead utilised a wide range of original resolutions. The video experiments on the contrary were conducted based on a single original Full HD resolution. The 4K size was superficially considered in one of the experiments but not properly investigated due to lack of high quality 4K sources.

The concept of dynamic resolution video in the described implementation, which allows only decreasing resolution with relation to original, is not suitable for high quality compression as the x265 codec is already extremely good in this area.

The fact that the existing video containers support concatenation of video streams specifically with the same frame resolution creates a problem with applying the dynamic resolution method in practice. Consequently, its implementation may require saving video segments as separate files or designing a simple container format to represent video as a single file. Playing such video however will not cause major technical difficulties as all video players support frame resizing to any display resolution.

5.5 Future Work

Due to the fact that predicting image and video compression has not been extensively researched, it is possible to propose a range of potential directions for further investigation.

Learning image and video content features automatically

A considerable obstacle towards obtaining reliable improvements in comparison with the default x265 compression is a relatively high error level of the regression models for predicting video compression characteristics. It is reasonable to suggest that using machine learning in feature design will improve the performance of the predictors and optimisation strategies in general. Also, for example, having accurate enough models will allow to consider a new video compression scenario with predefined time budget.

The most popular approach to automated feature design and extraction from image and video content is using deep learning techniques, in particular, convolutional neural networks. However, the problem is that standard convolutional networks are not suitable for such task because of the variable image/frame resolution and video length. A typical image rescaling approach to normalise the input size would destroy the pixel structure and alter the content entropy that need to be estimated. Consequently, a specially designed technique is required for this problem.

Moreover, applying deep learning would require a substantial amount of training data and computational resources in comparison with a shallow network that uses handmade features. In this light it seems inevitable that such an approach will need the power of GPU cards or co-processors in order to train models from scratch in a reasonable time.

Energy efficiency of the dynamic resolution video

The energy consumption issues were not investigated in this thesis, although the compression time was considered as an important optimisation objective. Due to the fact that the video compression results using dynamic resolution are approximately the same as those produced by the default compression, while the frame resolution in some parts is smaller, it is reasonable to assume that the energy consumption for decoding and playing the dynamic resolution video is lower than of the default one because Ejembi and Bhatti [35] demonstrated that playing smaller resolution video uses substantially less energy.

Using alternative image upscaling techniques for dynamic resolution video

The image resampling techniques play an important role in the concept of the dynamic resolution. Although the experiments in this thesis relied solely on the bicubic interpolation – it is not the most efficient method for upscaling image in terms of the resulting quality. Considering the methods proposed in the related

research, it is possible to suggest that implementing other upscaling techniques will improve the efficiency of the proposed optimisation methods in terms of quality to size ratio. However, a potential downside of this approach will be the increased time for video decompression.

Investigating dynamic resolution efficiency for 4K video

According to the experimental results on the search-based method for video compression optimisation, there is an indirect evidence that 4K video has more potential for using dynamic resolution in comparison with the Full HD resolution. However, there is also some uncertainty based on a fact that that the source videos had relatively low original quality with a considerable amount of blur that could have facilitated the role of dynamic resolution and consequently resulted in a formally better outcome. Therefore, a lack of the high quality 4K videos is a limiting factor for constructing reliable and unbiased machine learning models.

Nevertheless, it is reasonable to expect that the dymanic resolution is indeed more suitable for 4K videos because 3840×2160 pixels size can be resampled into a wider range of smaller resolution than 1920×1080 frame, thus increasing the optimisation space and potential benefits. This could be an interesting research direction.

5.6 Closing Thoughts

Predicting the effects of compression empirically has the tremendous advantage that it can be applied quickly to both existing and new compression techniques alike, without large scale engineering effort. It is reasonable to expect that the future image and video codecs despite the growing algorithm complexity will have fewer compression parameters because a considerable amount of minor adjustments will be performed automatically based on statistical analysis.

The author also hopes that eventually the concept of predictable compression will play its role in a discovery of the vastly more efficient compression algorithms than the ones existing today.

Appendix A

Alternative approaches to numerical regression

A part of the early experimental work consisted of numerous attempts to improve accuracy of the regression models for predicting compressed file size and quality. The main focus in this direction was on using various approaches to non-linear regression that can be alternative to artificial neural networks. In particular, the tests were conducted using polynomial and Gaussian process regression.

Polynomial regression involves a conceptually simple parametric model of a non-linear function of multiple variables. According to Bishop [68, section 1.7], a typical polynomial function of n^{th} degree represents a weighted sum of all possible products of the variables up to length n. For example, a 3^{rd} degree polynomial from two variables $(x_1;\, x_2)$ has 10 parameters:

$$F(x_1;\, x_2) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2 + \\ + w_6 x_1^2 x_2 + w_7 x_1 x_2^2 + w_8 x_1^3 + w_9 x_2^3.$$

Increasing the number of variables or the degree of polynomial function causes the number of its free parameters to grow exponentially, which can quickly lead to overparametrisation and consequently overfitting.

The Gaussian process regression [82] is a kernel based method that has only several metaparameters, which, for example, adjust curvature of the solution at a certain scale. The distinct feature of the Gaussian process regression is a more explicit control over the regression result in comparison with traditional weight based functions. However, the kernel based approach is very computationally expensive especially for problems involving multidimensional parameter spaces.

Several regression methods were compared using a problem of predicting the file size of JPEG compressed images. A relatively small dataset of 24 000 training

Table A.1: Brief comparison of the regression problem solutions obtained with Gaussian processes, polynomial function and multilayer perceptron.

Regression type	**# training points**	**Prediction error, %**		**Training time, s**
		Training set	**Test set**	
Gaussian process	1000	9.12	9.09	< 1
	2000	8.74	8.71	1
	4000	8.39	8.37	9
	8000	8.12	8.11	69
	16000	7.76	7.76	511
	20000	7.65	7.66	997
	24000	7.06	7.13	1900
Polynomial	24000	7.03	7.07	≈ 60
MLP	24000	7.16	7.19	≈ 300

points obtained from a basic set of 1000 images up to 4 MP in size was used for training purposes. The regression problem was formulated as follows:

$$compressed\_size = F(\boldsymbol{f_1}; quality\_factor; megapixels),$$

where $\boldsymbol{f_1}$ is a content feature value as described in section 3.2.

The problem was intentionally simplified by reducing the number of content features from ten to one. This allowed to perform a simple comparison of prediction accuracy for the models obtained with Gaussian process, polynomial and MLP regression in a reasonable amount of time. As a consequence, the error values are at about 7% instead of about 3% obtained in Chapter 3.

Table A.1 shows that for the above described regression problem all three methods are capable of producing models with approximately the same average error.

Gaussian process regression was performed using Python Scikit-learn library [83]. Due to a large computational complexity of this method, it was first tested using smaller subsets of training points. Moreover, there was a problem using all 24 000 training samples – the memory consumption of the script was over 16 GB available on the working machine, and consequently required space in the swap file on an SSD and longer training time. The problem is that Gaussian process regression requires solving a very large system of equations, proportional to the size of the training set, which consumes a substantial amount of memory.

For polynomial regression a custom application was written in C++ that allowed some customization of the fitting function. This method also requires solving a system of linear equations, but its size is relatively small and equals to the number of parameters in the model function. For example, the polynomial used to obtain results in the table A.1 had 7^{th} degree and 220 free parameters. It was selected as the best of several alternatives, results of which unfortunately were not recorded for comparison. Although polynomial functions with the degree higher than 3 are generally not very useful due to the high probability of overfitting, it worked well in this particular case.

Among the three tested alternatives the polynomial regression appeared to be the most straightforward and computationally cheap method. However, upon adding more features even 3^{rd} degree polynomial becomes overparametrised and quickly overfits the data. One of the possible ways to deal with this issue would be to remove components from the polynomial, but it makes overall problem too complicated and not worth the effort.

As for the Gaussian process regression, it is impractical to train a model on large datasets with multidimensional input vectors mainly due to the high memory consumption, which makes it impossible to run even on server machines without extra problem-specific optimisation.

A multilayer perceptron model requires a considerable training time but instead it has excellent scalability potential regarding multidimensional inputs and size of the training dataset. Therefore, this type of a feed-forward neural network was chosen for all experiments in the thesis as a reasonable compromise between accuracy, flexibility and computational complexity.

Appendix B

Tool for training feed-forward neural networks

The machine learning models used in this research were created in a form of the regression functions based on a standard MLP template (multilayer perceptron, or feed-forward neural network).

The models were trained to predict compressed file size, quality and encoding time for images and videos. The original intention however was making statistical estimations – i.e. a prediction with a confidence interval to make more confident compression decisions. But in the end a confidence interval was not playing a significant role in the experiments because a single predicted value is enough for comparison purposes in most of the real world scenarios. However, in the case if a target objective has a strict limit, the confidence interval boundaries can be used as target values instead of simply the predicted value, which by design is an average of the confidence interval.

All regression models in this work were trained with a specially designed tool written in C++ with the multi-thread parallelism. Its pilot version was implemented in order to perform quick tests of different training set sizes and network configurations specifically for image compression problems. It was optimised to use a single hidden layer because the gradient formulas in this case are considerably simpler in terms of computational resources than for networks with more than one hidden layer. The first version used the standard mean squared error metric as a minimisation objective.

The problem was in the error units, which were not decibels, bytes or percent but some abstract units like squared logarithm differences. This required to do extra calculations to present the meaningful error values in the report. Another issue was that the training metric (mean squared error) was different from the mean abso-

lute error used during evaluation. So, for the investigation into video compression problems these metric were merged into one – the root-mean-squared error, which corresponded to the units of the predicted objective.

Another important factor that influenced the custom implementation was a relatively low accuracy of the trained models for predicting video compression characteristics. In order to deal with this problem the network output was assumed to be statistically standardised ($\mu = 0, \sigma = 1$), which facilitated convergence but increased complexity of the formulas for partial derivatives (see section 4.6) under the requirement to keep errors in certain units.

Special attention has been dedicated to the parallel execution of the program. The most computationally intensive operation during the network training is calculating the gradient of the error function – i.e. partial derivatives of the network output with respect to all free parameters (connection weights). The complexity of each partial derivative depends on the number of training samples. Consequently, there are two possibilities for parallel implementations: by gradient components and by training samples. Due to the fact that in this research the number of training samples was considerably larger than network weights, the computational threads were designed to deal with training samples for the sake of smaller granularity.

The parallel implementation used a pool of processing threads running on a CPU without involving any GPU capabilities. This allowed to train several models simultaneously on different university servers. The program contained a pool of training samples – input vector and output value pairs. Each computational thread had been taking one training sample at a time and calculating respective parts of the gradient. The validation samples were also evaluated together with the training ones but did not participate in the gradient calculation. After processing the pool, the gradient was formed and the network weights adjusted according to the predefined gradient descent technique. This operation concludes a single training iteration.

The backpropagation algorithm was implemented as a standard learning strategy according to the recommendations by Bishop [68, chapter 4]. One if the important properties of this method is that the partial derivatives are calculated for each layer simultaneously with a forward network pass, which saves processing time but increases memory consumption.

Appendix C

Fitness bias in the genetic algorithm

A genetic algorithm (GA) was employed in one of the experimental scenarios to find an optimal combination of compression parameters for compressing test videos in parts. One of the steps in a classical GA is the crossover operation between two solutions that results in a new combination. Although the parent solutions are chosen by random, the preference should be dedicated to the solutions with better fitness (value of the objective function).

The problem is that selecting a random solution in a population taking into account its relative value is not a trivial task. It was empirically established that in some search scenarios the majority of population can have a very similar fitness value, for example, [2.01; 2.03; 2.07; 2.11]. Using a standard probability wheel will lead to allocating approximately the same chances to all solutions, which slows down the evolution process. In addition, recalculating probabilities based on the fitness of new solutions at each iteration is a relatively time consuming procedure. Therefore, in this work a set of probabilities was assumed constant for all iterations and prepared beforehand.

The vector of probabilities can be assumed constant at all iterations because its size depends only on the number of solutions in the population. It is calculated starting with a set of samples from a Gaussian curve between 0 and 3σ (fig. C.1), which is then normalised with intention to have a total sum equal to 1.

This approach subsequently allows to use any ordinary random number generator capable of producing floating point numbers that are uniformly distributed in the interval [0; 1).

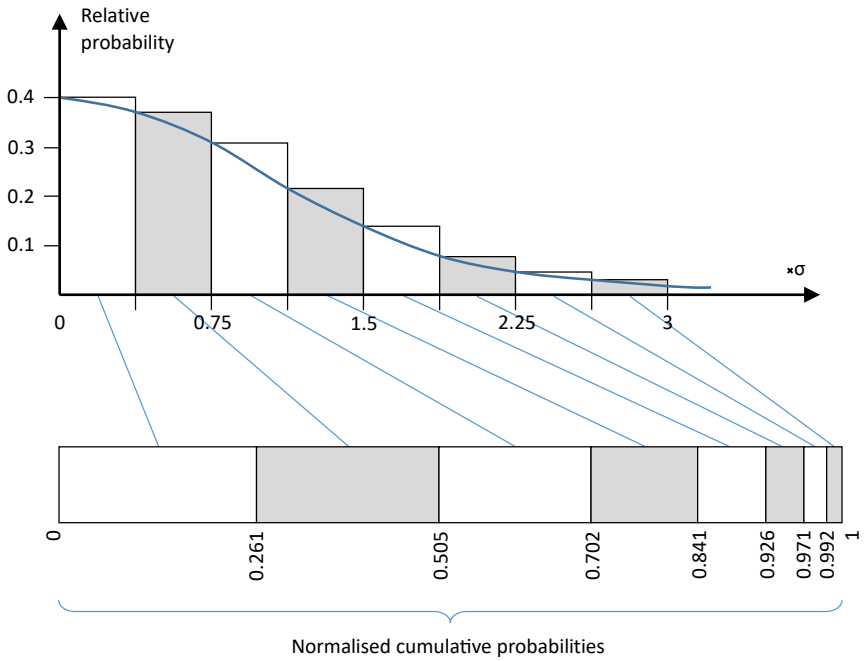


Figure C.1: Relative probabilities sampled from a Gaussian function normalised to cumulative probabilities with a sum of 1 to make a table for selecting a solution from the population of 8 elements.

These selection probabilities are assigned element-wise to the list of solutions in a population, which is sorted by fitness value at every iteration. Then choosing a random solution with respect to its fitness consists of generating a random number from $[0; 1)$ and matching it with a list of cumulative probabilities (fig. C.1).

References

[1] Cisco, "The zettabyte era: Trends and analysis." http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html, June 2017.

[2] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, pp. 600–612, April 2004.

[3] S. Pigeon and S. Coulombe, "Computationally efficient algorithms for predicting the file size of JPEG images subject to changes of quality factor and scaling," in *Communications, 2008 24th Biennial Symposium on*, pp. 378–382, June 2008.

[4] O. Murashko, J. Thomson, and H. Leather, "Predicting and optimizing image compression," in *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, (New York, NY, USA), pp. 665–669, ACM, 2016.

[5] "JPEG reference sources." http://jpegclub.org/reference/reference-sources.

[6] "libjpeg-turbo home page." https://libjpeg-turbo.org.

[7] M. Nelson and J. Gailly, *The Data Compression Book*. Wiley, 1995.

[8] Google Developers, "A new image format for the Web." https://developers.google.com/speed/webp, March 2016.

[9] D. S. Taubman and M. W. Marcellin, *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Norwell, MA, USA: Kluwer Academic Publishers, 2001.

[10] J. Garrett-Glaser, "Diary of an x264 developer: The problems with wavelets." http://archive.is/K7jKA, February 2010.

[11] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1649–1668, Dec 2012.

[12] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: Open-source HEVC/H.265 encoder," in *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, (New York, NY, USA), pp. 1179–1182, ACM, 2016.

[13] J. D. Cock, A. Mavlankar, A. Moorthy, and A. Aaron, "A large-scale video codec comparison of x264, x265 and libvpx for practical VOD applications," in *Proceedings Volume 9971, Applications of Digital Image Processing XXXIX*, vol. 9971, pp. 9971 – 9971 – 17, 2016.

[14] D. Vatolin, D. Kulikov, M. Erofeev, S. Dolganov, and S. Zvezdakov, "Twelfth MSU video codecs comparison." `http://compression.ru/video/codec_comparison/hevc_2017`, August 2017.

[15] N. Barman and M. G. Martini, "H.264/MPEG-AVC, H.265/MPEG-HEVC and VP9 codec comparison for live gaming video streaming," in *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 1–6, May 2017.

[16] Cisco, "Thor video codec implementation." `https://github.com/cisco/thor`.

[17] Cisco, "Thor video codec Internet draft." `https://tools.ietf.org/html/draft-fuldseth-netvc-thor-03`, October 2016.

[18] M. Montgomery, "Next generation video: Introducing daala part 3." `https://people.xiph.org/~xiphmont/demo/daala/demo3.shtml`, August 2013.

[19] D. Salomon, *A Guide to Data Compression Methods*. Springer, 2002.

[20] S. Chandra and C. S. Ellis, "JPEG compression metric as a quality aware image transcoding," in *2nd USENIX Symposium on Internet Technologies & Systems (USITS'99)*, 1999.

[21] S. Coulombe and S. Pigeon, "Low-complexity transcoding of JPEG images with near-optimal quality using a predictive quality factor and scaling parameters," *Image Processing, IEEE Transactions on*, vol. 19, no. 3, pp. 712–721, 2010.

[22] S. Pigeon and S. Coulombe, "Efficient clustering-based algorithm for predicting file size and structural similarity of transcoded JPEG images," in *Multimedia (ISM), 2011 IEEE International Symposium on*, pp. 137–142, Dec 2011.

[23] S. Pigeon and S. Coulombe, "Optimal quality-aware predictor-based adaptation of multimedia messages," in *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, vol. 1, pp. 496–499, Sept 2011.

[24] S. Pigeon and S. Coulombe, "K-means based prediction of transcoded JPEG file size and structural similarity," *International Journal of Multimedia Data Engineering and Management (IJMDEM)*, vol. 3, no. 2, pp. 41–57, 2012.

[25] H. Louafi, S. Coulombe, and U. Chandra, "Efficient near-optimal dynamic content adaptation applied to JPEG slides presentations in mobile web conferencing," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pp. 724–731, March 2013.

[26] S. Pigeon and S. Coulombe, "Quality-aware predictor-based adaptation of still images for the multimedia messaging service," *Multimedia Tools and Applications*, vol. 72, pp. 1841–1865, Sep 2014.

[27] J. Tichonov, O. Kurasova, and E. Filatovas, *Quality Prediction of Compressed Images via Classification*, pp. 35–42. Cham: Springer International Publishing, 2017.

[28] M.-K. Choi, H.-G. Lee, M. Song, and S.-C. Lee, "Adaptive bitrate selection for video encoding with reduced block artifacts," in *Proceedings of the 2016 ACM on Multimedia Conference*, MM '16, (New York, NY, USA), pp. 282–286, ACM, 2016.

[29] JVT, "H.264/14496-10 AVC reference software manual." `http://iphome.hhi.de/suehring/tml/JM%20Reference%20Software%20Manual%20(JVT-AE010).pdf`, January 2009.

[30] M. Covell, M. Arjovsky, Y.-c. Lin, and A. Kokaram, "Optimizing transcoder quality targets using a neural network with an embedded bitrate model," *Electronic Imaging*, vol. 2016, no. 2, pp. 1–7, 2016.

[31] YouTube Engineering and Developers Blog, "Machine learning for video transcoding." `https://youtube-eng.googleblog.com/2016/05/machine-learning-for-video-transcoding.html`, May 2016.

[32] J. I. Wong, "Netflix's new AI tweaks each scene individually to make video look good even on slow internet." `https://qz.com/920857/netflix-nflx-uses-ai-in-its-new-codec-to-compress-video-scene-by-scene`, February 2017.

[33] Netflix Technology Blog, "Per-title encode optimization." http://techblog.netflix.com/2015/12/per-title-encode-optimization.html, December 2015.

[34] Netflix Technology Blog, "Toward a practical perceptual video quality metric." http://techblog.netflix.com/2016/06/toward-practical-perceptual-video.html, June 2016.

[35] O. Ejembi and S. N. Bhatti, "Help save the planet: Please do adjust your picture," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 427–436, ACM, 2014.

[36] S. Chikkerur, V. Sundaram, M. Reisslein, and L. J. Karam, "Objective video quality assessment methods: A classification, review, and performance comparison," *IEEE Transactions on Broadcasting*, vol. 57, pp. 165–182, June 2011.

[37] H. Tong, M. Li, H. Zhang, and C. Zhang, "Blur detection for digital images using wavelet transform," in *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, vol. 1, pp. 17–20 Vol.1, June 2004.

[38] M.-J. Chen and A. C. Bovik, "No-reference image blur assessment using multiscale gradient," *EURASIP Journal on Image and Video Processing*, vol. 2011, p. 3, Jul 2011.

[39] I. P. Gunawan and A. Halim, "Local blur and blocking artifact detection in digital images," *Ultima Computing*, vol. 1, no. 1, 2010.

[40] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electronics Letters*, vol. 44, pp. 800–801, June 2008.

[41] A. Hore and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, August 2010.

[42] Z. Kotevski and P. Mitrevski, "Experimental comparison of PSNR and SSIM metrics for video quality estimation," *ICT Innovations 2009*, pp. 357–366, 2010.

[43] Google, "butteraugli." https://github.com/google/butteraugli.

[44] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[45] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, "Variable rate image compression with recurrent neural networks," *CoRR*, vol. abs/1511.06085, 2015.

[46] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," *CoRR*, vol. abs/1608.05148, 2016.

[47] P. Gupta, P. Srivastava, S. Bhardwaj, and V. Bhateja, "A modified psnr metric based on hvs for quality assessment of color images," in *2011 International Conference on Communication and Industrial Application*, pp. 1–4, Dec 2011.

[48] N. Johnston and D. Minnen, "Image compression with neural networks." `https://research.googleblog.com/2016/09/image-compression-with-neural-networks.html`, September 2016.

[49] P. Schlessinger, "Problems with self-similarity as a basis for image compression." `https://paulschlessinger.wordpress.com/2014/09/17/problems-with-self-similarity-as-a-basis-for-image-compression`, February 2017.

[50] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[51] Y. Romano, J. Isidoro, and P. Milanfar, "RAISR: rapid and accurate image super resolution," *CoRR*, vol. abs/1606.01299, 2016.

[52] J. Nack, "Saving you bandwidth through machine learning." `https://www.blog.google/products/google-plus/saving-you-bandwidth-through-machine-learning`, January 2017.

[53] F. Jiang, W. Tao, S. Liu, J. Ren, X. Guo, and D. Zhao, "An end-to-end compression framework based on convolutional neural networks," *CoRR*, vol. abs/1708.00838, 2017.

[54] F. Bellard, "BPG image format." `https://bellard.org/bpg/`, 2015.

[55] ITU, "Recommendation ITU-R BT.601-7." `https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf`, March 2011.

[56] E. Hamilton, "JPEG file interchange format." `https://www.w3.org/Graphics/JPEG/jfif3.pdf`, September 1992.

[57] K. Fant, "A nonaliasing, real-time spatial transform technique," *Computer Graphics and Applications, IEEE*, vol. 6, pp. 71–80, January 1986.

[58] U. Javaid, "Find & remove similar photos instantly." http://www.addictivetips.com/windows-tips/find-remove-similar-photos-instantly, June 2010.

[59] "Mozilla jpeg encoder project." https://github.com/mozilla/mozjpeg, 2017.

[60] Mozilla Research, "Introducing the 'mozjpeg' project." https://research.mozilla.org/2014/03/05/introducing-the-mozjpeg-project, March 2014.

[61] R. Obryk and J. Alakuijala, "Announcing Guetzli: A new open source JPEG encoder." https://research.googleblog.com/2017/03/announcing-guetzli-new-open-source-jpeg.html, March 2017.

[62] R. Obryk and J. Alakuijala, "A closer look at Guetzli, Googles new JPEG-encoder." https://cloudinary.com/blog/a_closer_look_at_guetzli, April 2017.

[63] Google Developers, "cwebp." https://developers.google.com/speed/webp/docs/cwebp, June 2017.

[64] O. Tange, "GNU Parallel - the command-line power tool," *The USENIX Magazine*, February 2011.

[65] Kaggle, "Root mean squared error." https://www.kaggle.com/wiki/RootMeanSquaredError, May 2017.

[66] T. Little, *The Oxford Handbook of Quantitative Methods, Vol. 2: Statistical Analysis*. Oxford Library of Psychology, Oxford University Press, 2013.

[67] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[68] C. Bishop, *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics, Clarendon Press, 1995.

[69] S. Ruder, "An overview of gradient descent optimization algorithms." http://ruder.io/optimizing-gradient-descent, June 2017.

[70] O. Murashko and J. Thomson, "ACACIA." https://github.com/johndthomson/acacia, 2016.

[71] "FFmpeg." https://www.ffmpeg.org.

[72] "FFmpeg source code mirror." https://github.com/FFmpeg/FFmpeg/blob/master/libswscale/utils.c#L1229.

[73] Fraunhofer Heinrich Hertz Institute, "High efficiency video coding (HEVC)." https://hevc.hhi.fraunhofer.de, 2016.

[74] MulticoreWare Inc., "HEVC/H.265 explained." http://x265.org/hevc-h265.

[75] MulticoreWare Inc., "x265 documentation." http://x265.readthedocs.io.

[76] Ultra video group, "Test sequences." http://ultravideo.cs.tut.fi/#testsequences.

[77] M. J. Marjanovic, "yuv4mpeg - Linux man page." https://linux.die.net/man/5/yuv4mpeg, 2004.

[78] ITU, "Recommendation ITU-R BT.709-6." https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf, June 2015.

[79] Andrew Ng, "Machine learning yearning: Technical strategy for AI engineers in the era of deep learning. Draft version 0.5." https://gallery.mailchimp.com/dc3a7ef4d750c0abfc19202a3/files/Machine_Learning_Yearning_V0.5_01.pdf, December 2016.

[80] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[81] J. Garrett-Glaser, "[x264-devel] commit: Display SSIM measurement in dB." https://mailman.videolan.org/pipermail/x264-devel/2010-June/007391.html, June 2010.

[82] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[83] Scikit-learn, "Gaussian processes." http://scikit-learn.org/stable/modules/gaussian_process.html#gaussian-process-regression-gpr.