Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

FINAL DEGREE PROJECT

Application of different Machine Learning models for the H.264/AVC to HEVC video transcoding problem

UNIVERSITY OF OVIEDO

School of Computer Science

Bachelor of Software Engineering

Miguel García García

Oviedo, July 2018

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

FINAL DEGREE PROJECT

Application of different Machine Learning models for the H.264/AVC to HEVC video transcoding problem

UNIVERSITY OF OVIEDO

School of Computer Science

Bachelor of Software Engineering

Miguel García García

Supervisor: Antonio Jesús Díaz-Honrubia

Co-supervisor: Susana Irene Díaz Rodríguez

Oviedo, July 2018

Abstract

Nowadays video is everywhere. Services like Netflix, YouTube or Facebook are hosting enormous amounts of video content, consumed by thousands of millions of users, which can now access those contents from any place, due to the popularization of mobile devices, able to reproduce multimedia content, and the emergence of wireless networks. Also, new and heavier video formats like 4K or 3D video are being more and more demanded.

However, telecommunication networks cannot stand such a huge amount of information, so the data to be transmitted has to be compressed. Thus, new coding standards able to provide higher compression than the existing ones without important reductions in quality nor huge increments on encoding times are required.

The development of video coding standards is done by two expert groups: the International Telecommunication Union (ITU) and the Organization for Standardization and International Electrotechnical Commission (ISO/IEC). A collaboration between both groups led to the release of the H.264/AVC standard in 2003. This standard became the state of the art in video coding, and reached an enormous popularity, that still retains today, given the good compromise between quality and size. The amount of sequences encoded in H.264/AVC is huge, since it is the format used for HD television, Blu-ray discs and most video cameras, dominating both the domestic and professional markets for over ten years.

In 2013, a new standard, HEVC, was proposed by the same joint group, providing double the compression while maintaining the same quality. As expected, this originated a progressive migration process from the previous formats. Nevertheless, the transformation of a sequence to the HEVC standard is not an easy task, given the complexity of the encoder, which requires more computational resources than its predecessor.

This project aims to provide a solution to this problem, incorporating Machine Learning techniques to the transcoding process from H.264/AVC to HEVC in order to reduce the time needed to encode a sequence in the new format.

The results obtained are promising for situations in which the encoding speed is of foremost importance, being more appropriate for the most complex and high-resolution sequences, where this proposal is able to perform the encoding 3.20 times faster than the reference encoder with a bitrate increase of only 3.4%.

Acknowledgements

Four years ago, a new stage of my life started: I was finally studying what I had been dreaming for so long. Four years seemed to be a long time at first, so I can't believe that this journey is already finishing with this last project.

With its good and bad moments, I can definitely say that it was an incredible ride, full of learnings not only in the academic field, but also in the personal ambit. I could meet fantastic friends, who I hope to be able to preserve for the rest of my life. Special thanks to Jorge, since, if it weren't for him, I wouldn't be where I'm today. He was always keen to help in whatever I needed and was an invaluable support these last two years. He was an example of effort and dedication which helped me keep motivated and excel myself in a way I couldn't have otherwise. Thank you.

Also, I have to thank those childhood friends, that are still there and which, even though our lives are most likely going to follow different courses, I'm completely sure I'll keep frequent contact with.

Of course, immense thanks to my family. To my godmother for helping me and my parents in everything she could since I was a child. To my grandfather who always thought that I was going to get very far and who I'm sure is genuinely happy now with my most recent achievements. And how could I forget my grandmothers and, of course, my other grandfather, who I'm sure would also be proud.

I have to dedicate a separate paragraph to my parents, Jose Miguel and Mª Gloria, although I would need the length of the whole document to thank everything they had made for me. They have given me all I needed during all my life, prioritizing my education and my future over everything else, even when the situation was not favorable. They never imposed me what to study or what to do with my life but taught me the values and advices I needed for being able to succeed taking my own decisions.

I don't want to forget to thank some of my teachers at school, specially my tutors in Bachillerato, who supported me when I was going through a complicated phase. Of course, thanks to the professors in the University of Oviedo, particularly to Irene and Antonio for their inestimable help in this project.

Finally, a mention for my colleagues at CERN, which are making my stance there an amazing journey, even though I've had to devote the first two weeks to conclude this project.

I don't want to forget anyone, so thanks in general to every person that has helped me or who has given me an opportunity at any time. ***Thank you****.*

Index

Table of figures

Table of tables

CHAPTER 1. INTRODUCTION

This chapter introduces the project briefly, showing the main motivations, the objectives pursued with its development, the work plan followed and the structure of this document.

1.1 Introduction

The amount of video traffic on the Internet has been dramatically increasing in the last few years, amounting to as much as 70% of all Internet traffic in 2015 [1]. With new technologies such as 4K, 3D and multiview video, higher frame rates, extended color gamut, and the emerge of new video streaming services, this percentage is expected to keep growing up to 82% by the year 2020.

This growth can be attributed to the high number of platforms with rapidly expanding user bases making use of video content for a wide variety of purposes: online education, like Coursera or edX; video on-demand, like Netflix or HBO; video sharing, like YouTube or Vimeo; or social networking, like Facebook or Instagram.

To be able to cope with this increase, the Joint Collaborative Team on Video Coding (JCT-VC) released the High Efficiency Video Coding (HEVC) standard [2] in 2013. HEVC doubles the compression performance of the previous standard H.264/MPEG-4 Part 10 – Advanced Video Coding (AVC) [3] (i.e. reduces the bitrate by 50% while maintaining the same objective quality) [4]. However, this comes at a cost, since HEVC requires more complex computations, being much slower at both encoding and decoding times, than its predecessor [5].

Considering the huge amount of content that is currently encoded using H.264/AVC, a system allowing conversion from H.264/AVC to HEVC is highly desirable. Moreover, there is a wide catalog of H.264/AVC encoders with a good tradeoff between performance and cost, like the open-source project x264 [6], so combining H.264/AVC coding with transcoding to HEVC lets us exploit the benefits of both standards.

Following this approach, information from the H.264/AVC stream can be extracted and reused to accelerate the decisions taken on the HEVC encoder, thus lowering the computational and storage complexities of the latter. This is the main objective pursued in this work.

1.2 Motivation

Without video coding standards, communication infrastructures would not be able to handle the massive amounts of video circulating in the network. The growth expectations evidence the necessity of new standards with biggest compression rates in order to reduce the bandwidth required for video transmission. The 50% bitrate reduction achieved by HEVC which, following some subjective quality metrics, can raise to 59% [7], is essential for handling these new consumption habits.

Therefore, the usage of HEVC needs to be as widely extended as possible, at least until an improved new coding standard emerges as a replacement. Of course, it is also necessary and crucial that videos currently encoded in H.264/AVC are transcoded to the HEVC standard. This project aims to present a contribution to this important issue.

The conversion of a sequence which is encoded with a specific standard and configuration is called transcoding [8]. Transcoding can be heterogeneous, when the standard used to encode the sequence is changed, or homogeneous, when only the configuration (resolution, frame rate, bit rate and/or any other parameter) is modified. This work is focused on heterogeneous transcoding, which is very relevant nowadays due to the great variety of devices with different decoding capabilities.

The transcoding process can be performed inefficiently in a cascade way (i.e. decoding the original H.264/AVC stream and re-encoding it with HEVC), but this requires a high amount of computational power and time. It is logical to propose the usage of some information from the original H.264/AVC encoded stream to speed up the HEVC encoding.

The extraction of knowledge from H.264/AVC for its later application in HEVC has already been discussed by some authors, being [9] one of the most relevant contributions and the starting point for this research project.

Machine Learning techniques, which are enjoy a huge popularity lately and are being really useful in very diverse fields, can be of inestimable help as an acceleration tool, assisting in some of the decisions taken during the transcoding processes and allowing to reduce times importantly with just small quality decrements. This proposal also makes an extensive use of these techniques.

1.3 Objectives

The main objective of this work is to improve the transcoding algorithm proposed in [9] by studying and comparing different Machine Learning (ML) algorithms and preprocessing techniques and then applying the most successful classifier to the transcoder. To address this objective, the following goals are proposed:

- **Goal 1:** to review some of the most common Machine Learning algorithms and preprocessing techniques that will be later applied.
- **Goal 2:** to explain the basic concepts of video coding, focusing on the different compression techniques and giving an overview of the different coding standards, with a more in-depth look at H.264/AVC and HEVC and their features and differences.
- **Goal 3:** to carry out a review of the state of the art of H.264/AVC to HEVC transcoding.
- **Goal 4:** to test and compare the performance of different algorithms over several sample datasets in order to identify the most interesting classifier to be applied in the transcoding process.
- **Goal 5:** to implement the most interesting classifier for its usage in the HEVC reference software.
- **Goal 6:** to test the performance of the encoder making use of the implemented classifier in comparison to the discussed state-of-the-art approaches.

1.4 Work plan

This project has followed a work plan divided in different ordered phases, starting with the understanding of the domain of the problem and continuing with the execution of different tests, the implementation phase and the final evaluation of the results achieved:

- **Phase 1:** understanding the basic concepts on video coding and compression techniques and standards, studying the relevant literature and familiarizing with the AFQLD algorithm [9].

- **Phase 2:** understanding the KDD process [10] and studying the different Machine Learning algorithms, learning their characteristics and advantages.

- **Phase 3:** automating and executing the tests of the ML algorithms over training datasets and starting the documentation related with the previous points while the tests are running, since they take a very high amount of time.

- **Phase 4:** reviewing the obtained results and implementing the most promising classifier into the proposed algorithm.

- **Phase 5:** testing the proposed algorithm over standardized sequences to measure and analyze its real performance, comparing it with the reference transcoder and the AFQLD algorithm. Meanwhile, incorporating the results obtained from phases 3 and 4 to the document.

- **Phase 6:** writing the chapter corresponding to the evaluation of the proposal based on the results from phase 5 and finishing the documentation by adding the conclusion chapter where the achievements obtained by the project are discussed, as well as the rest of the missing chapters.

The timespan in which each of the six phases was developed and the number of hours spent on them is shown in 1.

Phase	**Timespan**	**Hours**
Phase 1	February & early March 2018	40
Phase 2	Mid-late March 2018	20
Phase 3	April & May 2018	110[1]
Phase 4	Early June 2018	70
Phase 5	Mid-late June 2018	70
Phase 6	Early July 2018	40

Table 1.1. Schedule of the project.

[1] This refers to the time spent working actively. The time taken by the tests to run was much higher.

1.5 Structure of the document

This document is organized as follows. In the next chapter, an overview of some of the most popular Machine Learning algorithms is carried out, paying attention to the pros and cons of each one. Chapter 3 introduces some basic concepts about video coding and then develops the functioning of the two main coding standards this work is based on. Chapter 4 describes the experimentation made with the previously mentioned algorithms, comparing their performance. Additionally, experimental results over video sequences are given in Chapter 5. Lastly, some final conclusions can be read in Chapter 6. Finally, an appendix describing the contents of the attached ZIP file can be found.

CHAPTER 2. KDD

KDD, or *Knowledge Discovery in Databases* [10] is the process of finding and interpreting patterns or models in the data. Its main objective is to map low-level data, too voluminous to understand, into more compact, more abstract or more simple forms. This is done by using data mining algorithms to extract knowledge from a database, applying any required preprocessing or transformation to that database. The general process of KDD is outlined in Figure 2.1 and involves the repetition of the following steps:

1. Understanding the application domain and prior knowledge.
2. Creating a target dataset, where a subset of samples on which discovery is to be performed is selected.
3. Data cleaning and preprocessing, focused on removing noise from outliers and handling missing data fields.
4. Data reduction and projection, consisting on finding the useful features for the problem and reducing the number of features.
5. Choosing the data mining task, that is, deciding if the problem is a classification (like in this project), regression, clustering, etc.
6. Choosing the data mining algorithms, where the method for searching for patterns in the data is selected and its parameters are decided.
7. Data mining, i.e. applying the algorithms to search for patterns of interest.
8. Interpreting the obtained patterns.
9. Consolidating the knowledge acquired.

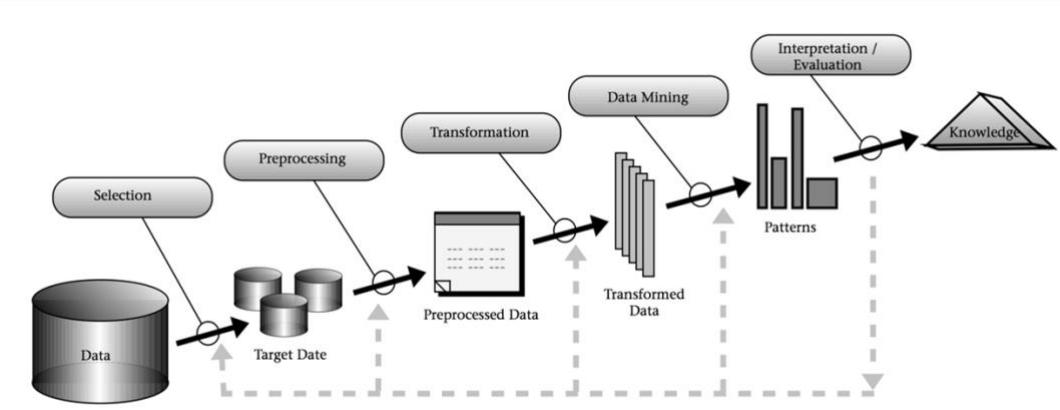

Figure 2.1. Steps of the KDD process [10].

2.1 Classifiers

In this section some concepts focused on steps 3 to 6 from the KDD process are explained in detail. Data preprocessing is going to be reviewed first, followed by an overview of the most popular and well-performing Machine Learning algorithms.

2.1.1 Data preprocessing

Before running a classification algorithm, it is important to consider the possibility of manipulating the dataset in order to improve its quality and maximize the knowledge extracted. The most common preprocessing methods are discretization and feature selection.

2.1.1.1 Discretization

Discretization consists on grouping continuous values into a defined set of intervals, so, making features discrete/categorical. Whenever a discretization is performed, there is a discretization error, which we try to minimize. There are different ways of performing the partitioning.

- Equal-width partitioning divides the features into n equal intervals. This can be problematic if we have almost all the data in the same partition. It is sensible to outliers.

- Equal-depth partitioning performs the division into n intervals each one containing approximately the same number of samples. This way the distribution of values is taken into account and it is not sensible to outliers.

- Entropy-based discretization uses entropy to find the best way to split the data. It calculates the value that maximizes the Information Gain (defined in 2.1.2.1) and splits by that value. The process is repeated until n intervals are created or no Information Gain is achieved. Unlike the other two methods, this one is supervised, that is, it takes into account the value of the target feature to make the discretization, usually yielding better results.

This is an important process since many Machine Learning algorithms are known to produce better models if continuous values are discretized first [11].

2.1.1.2 Feature selection

Also known as variable or attribute selection, feature selection consists on selecting a subset of the most relevant features to be used subsequently in model construction [12]. This allows us to simplify the models, shorten the training times, remove irrelevant features, reduce overfitting and, usually, improve the accuracy of the models.

It can be seen as a combination of a search algorithm for proposing new subsets with an evaluation algorithm to measure the quality of each subset, being this second part the one with the biggest impact on the results obtained. There are distinct categories of feature selection algorithms:

- Wrapper methods, popularized by Kohavi and John [13], evaluate subsets by running a Machine Learning algorithm on each of them, ranking their predictive power. Performance assessment is done using a test dataset or by cross-validation. Wrappers are computationally expensive when the number of features is large and have the risk of overfitting the model, but different search strategies can be used to reduce the amount of computation required. Greedy strategies are particularly beneficial and robust against overfitting and are presented in two flavors: forward selection, where the best features are iteratively incorporated into an initially empty set, and backward elimination, where, starting with the set of all features, the least promising ones are progressively eliminated.

- Filter methods [14] perform evaluation based on general features like the correlation with the variable to predict. They are robust to overfitting and require low computation time, although they tend to select redundant features since they do not consider the relationship between features.

- Embedded methods [15] combine the learning part with the feature selection part. Thus, a separate preprocessing step is not necessary. They use a Machine Learning algorithm which takes advantage of its own feature selection process and performs feature selection and classification at the same time. Decision trees such as C4.5 can be understood as embedded methods since feature selection is implicitly built into the algorithm.

2.1.2 Algorithms

In this section, six of the most common machine learning algorithms are presented, having a look at their functioning, the type of problems in which they excel, their differences and their advantages and disadvantages.

2.1.2.1 C4.5

C4.5 [16] is an algorithm for decision tree creation. It is one of the most widely used Machine Learning algorithms and it can be used on both discrete and continuous values.

Decision tree algorithms start with a root node which represents the whole training set. The algorithm chooses the feature that best predicts the target variable and divides the samples in groups depending on the selected feature's value. This way, the different branches of the tree are created. The branching process stops once every or almost every example of the node has the same class (i.e. when the target feature takes the same value), when there are no more features to study or when the tree has reached a given size.

With discrete features a branch is generated for each possible value of the feature, but when working with continuous features, one or several split points must be chosen.

Figure 2.2 shows an example of decision tree built with C4.5, whose objective variable is whether to go to play outdoors or not. The algorithm decides to use the outlook feature as root. Then, if it is overcast, play can be chosen directly while, if it is rainy, the algorithm performs a further splitting based on whether it is also windy or not. If the weather is sunny, since the humidity node is a continuous variable, it splits the samples on two ranges.

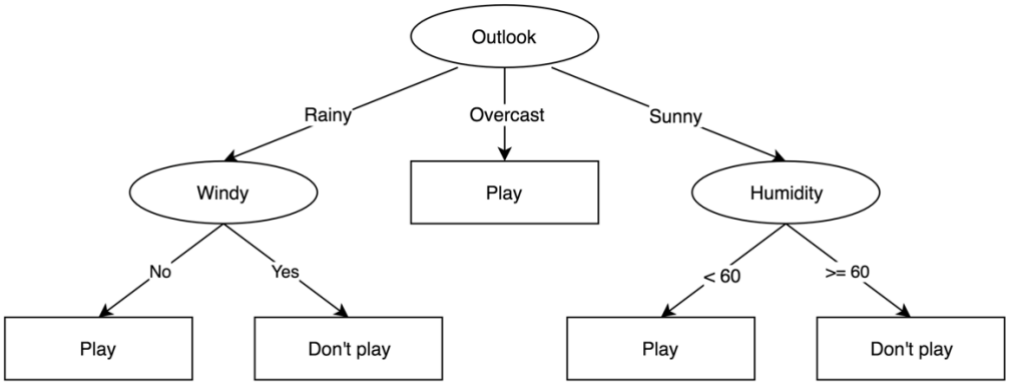

Figure 2.2. Example of a C4.5 decision tree.

Choosing the best variable

To choose the best variable, heuristic criteria are employed. In this case, C4.5 uses the Gain Ratio (GR) criterion, expressed as $GainRatio(A) = \frac{IG(A)}{SplitInfo(A)}$.

The Information Gain (IG) measures the reduction of entropy after using a variable to split the dataset. It is defined as $IG_Y(X) = E(Y) - E(X|Y)$, where $E(Y|X) = \sum_i P(X = v_i)\, E(Y|X = v_i)$. That is, the summation of the probability of the variable X taking each value multiplied by the entropy of the variable Y when X takes that value.

To understand the previous formula, the concept of entropy must be defined. Entropy indicates the homogeneity of a data sample, i.e. the predictability of data, and is represented as $E(X) = -\sum_{i=1}^{N} p_i \log_2 p_i$, where p_i stands for the probability of an object v_i. Thus, the lower the entropy, the easier to shape a prediction.

Finally, Split Info normalizes the Information Gain using the entropy of the partition, so, $SplitInfo(A) = -\sum_{i=1}^{v} \frac{|S_i|}{S} \log_2 \frac{|S_i|}{S}$, where A represents a given variable, S stands for the set of training samples, and S_i indicates the samples which take a given value of the attribute A. Higher Split Info indicates more uniform partition sizes.

The Gain Ratio (GR) criterion is applied to each attribute, selecting the one that achieves the highest value as a root. This process is applied recursively to each new node, until one of the stopping criteria is fulfilled. Note that GR is calculated again for all the attributes each time we want to split a new node. From this process one can notice, as mentioned before, that C4.5 performs feature selection implicitly during its execution.

An important fact to be aware of is the vulnerability of decisions trees to overfitting. A decision tree is said to overfit when, instead of being a generalized model, it corresponds

too closely to training dataset and its possible errors and, thus, performs poorly when facing new test data. Generally, the more features you have, the more likely the model will suffer from overfitting. C4.5 incorporates pruning techniques to try to reduce this problem.

Pruning can be divided on pre-pruning, consisting on stopping the construction of the decision tree before it overfits the training set following a given stopping condition (a depth limit, a maximum number of nodes, a minimum GR, etc.), and post-pruning, in which branches of the tree are removed once finished and replaced by leaves. The latter is the most recommended technique and is the one that C4.5 uses.

Advantages and disadvantages

Although vulnerable to overfitting, decision trees are flexible, only use the most relevant attributes, produce easy to interpret models, can be used for small and big datasets and are more efficient than more complex models. However, they are usually vulnerable to slight changes on the training set, and have the risk of overfitting, as just explained.

2.1.2.2 SVM

SVMs, or Support Vector Machines [17] use linear models to implement classes with non-linear separations, transforming the input space into a new space. Samples are represented as points in space, and the goal is to build a hyperplane w which acts as a decision surface so that the margin between the two classes is maximum. New samples will be predicted depending on which side of the space they fall on.

This algorithm can be seen as an extension of a Perceptron (2.1.2.5). A Perceptron is able to find a separator between two classes if it exists, while SVM manages to find the one with the maximum margin.

The separation margin is measured orthogonally from the hyperplane to the closest point at each side, as shown in Figure 2.3. Although infinite different hyperplanes cloud be built between two classes, choosing the one with the maximum margin provides higher robustness against outliers. A margin of error is also allowed so that said outliers do not influence the classification too much.

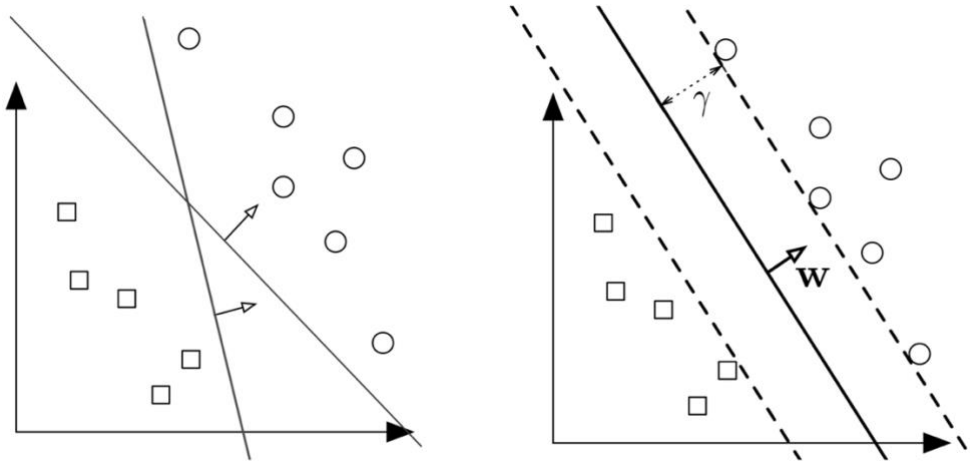

Figure 2.3. Left: Two different hyperplanes. Right: The maximum margin hyperplane w, where γ is the distance from the hyperplane to the closest point in either class [18]

Let $y_i(w^T x_i + b) \geq 1, \forall i$ be the decision frontier. The margin between the hyperplane and the closest point of class $+$ is $d^+ = \frac{|wx^+ + b|}{\|w\|} = \frac{1}{\|w\|}$. The distance to class – can be computed similarly. The width of the margin is then $d^+ + d^- = \frac{2}{\|w\|}$. The goal is to maximize this function.

Non-linear problems

If the problem is not linear, the space x_i is transformed into a higher-dimensional feature space $f(x_i)$ where linear separation of the data is possible.

Nonetheless, calculations on the feature space are costly due to high dimensionality. We reduce this impact by the use of kernels, similarity functions which take two inputs and return the similarity between them, that are provided to the algorithm by the user and which require less computation.

This is possible since many ML algorithms (like SVM) can be expressed entirely in terms of dot products and, a kernel function, under determinate conditions, can be expressed as a dot product in a feature space (Mercer's Theorem). This way, we can replace a dot product in the algorithm with a kernel easier to compute. This is called the *kernel trick.*

Diverse kernel functions can be applied. Some of them are polynomial kernels of degree d, denoted as $K(x, y) = (x^T y + 1)^d$; radial (RBF) kernels of width σ, expressed as $K(x, y) = \exp(-\frac{\|x-y\|^2}{2\sigma^2})$ or sigmoidal with parameters κ and θ, formulated as $K(x, y) =$

$\tanh(\kappa x^T y + \theta)$. Commonly, a low-degree polynomial kernel or an RBF kernel of reasonable width are good elections.

Figure 2.4 shows an example of transformation from the input space to a higher-dimensional feature space performed with a quadratic kernel. We can see how, after the transformation, it is possible to split the data in two classes with a hyperplane, something that was not possible in the original space.

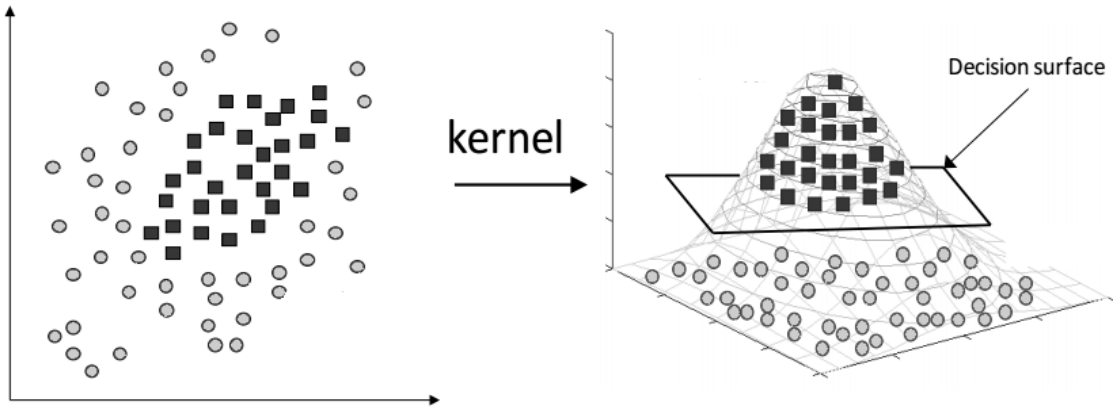

Figure 2.4. Transformation from a two dimensional to a three dimensional space [19].

Advantages and disadvantages

SVMs are easy to train, versatile (because they accept varied kernel functions), valid for non-linear problems, and scale well for high-dimensional data. Conversely, they depend strongly on the chosen kernel function, and are not efficient if the number of features is very huge in number in comparison with the training samples. Also, unlike with decision trees, the resulting models are difficult to interpret.

2.1.2.3 Random Forest

Random Forests [20] are one of the most popular Machine Learning algorithms. Like the term *forest* indicates, they build a multitude of decision trees, usually trained with the *bagging* method and then merge them together, averaging the results.

Bagging

While the predictions of a single tree are highly sensitive to noise in the training set, the average obtained from many trees is not, as long as the trees are not correlated. To reduce correlation, as it can be seen in Figure 2.5, Random Forests create random subsets of the

features, build several smaller trees using those subsets and then average the results of all the subtrees. This way, the risk of overfitting is lowered.

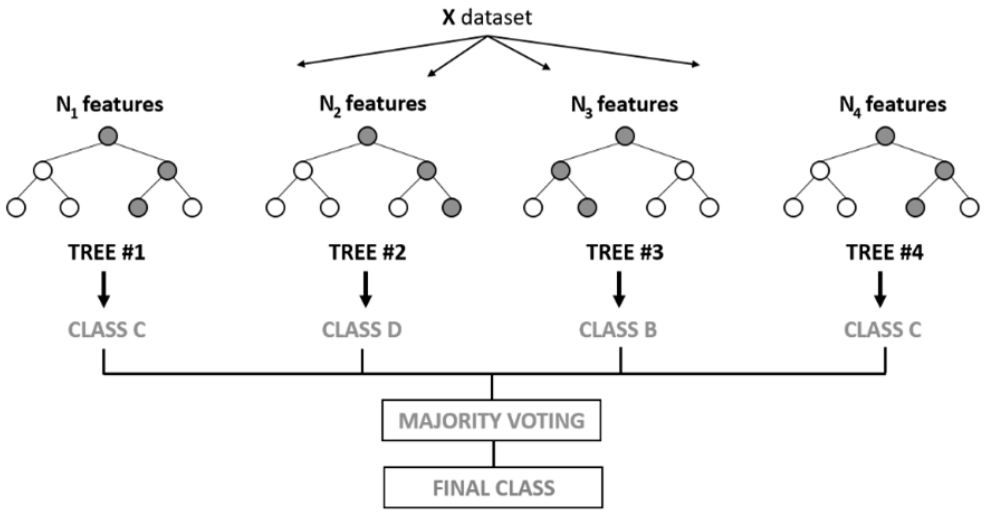

Figure 2.5. Example of a Random Forest [21].

Therefore, when splitting a node, instead of searching for the best overall feature, the algorithm looks for the best feature among a random subset of features, creating a higher diversity which generally improves the model's quality.

The creation of a Random Forest consists on the following steps:

1. Randomly select k features from the total m features, being $k \ll m$.
2. Calculate the node d using the best split point among those k features.
3. Split the node.
4. Repeat 1 to 3 until l level of nodes has been obtained.
5. Repeat 1 to 4 n times to create n randomly generated trees.

Then, for making a prediction, the next procedure is followed:

1. Classify the test features following the rules stablished by each tree, storing the outcome.
2. Calculate the votes for each prediction.
3. Select the most voted outcome as the final prediction.

Advantages and disadvantages

Random Forests are easy to use, fast to train (individual trees can be trained in parallel), efficient also on large datasets and, as explained, robust against overfitting. On the contrary, they are not easily interpretable, and a large number of trees can make the algorithm slower and ineffective for real-time predictions, although it is generally fast enough for most real-world applications.

2.1.2.4 KNN

The KNN or K-nearest neighbors [22] algorithm is the most basic instance-based learning method. It bases classification in analyzing the classes of the neighboring samples. KNN is useful when samples can be represented in a vector space or when they are characterized by few features (commonly, less than 20).

Instead of performing generalization, instance-based learning algorithms compare new problem instances with instances seen in training, which have been stored in memory. It is easy to see that SVM methods also fall in this category.

The samples are represented in an n-dimensional space, where n is the number of features. Three inputs are required: the training set, a formula to calculate the distance between each pair of samples (Euclidean, Manhattan, etc.) and k, which is the number of nearest neighbors to return.

Unlike most algorithms, KNN does not perform any operation during training time. Instead, when a new example has to be classified, the distance from that example to each of the training samples is calculated, the k nearest samples are identified, and a decision is made based on the classes of them. Decision on the nearest samples can be made by simply taking the majority class or by weighting each example depending on the distance, for instance with a weight $w = \frac{1}{d^2}$.

The election of a good k is key for the satisfactory performance of the algorithm, since a small k will be sensitive to noise and a big one will make boundaries between classes less distinct. The general recommendation is to experiment with different odd values.

Figure 2.6 illustrates an example with two features, in the x and y axes, using $k = 3$ and $k = 5$. In the first case, the new example is assigned to the triangle class, while in the second case it is classified as square class.

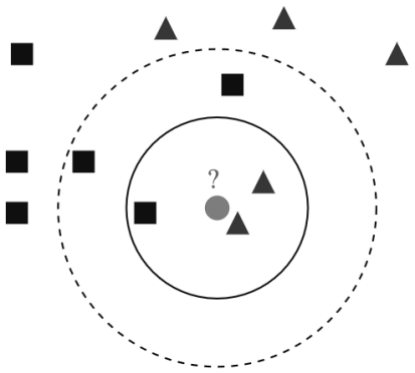

Figure 2.6. Example of a KNN classification. The circle represents the element to classify, while the squares and triangles represent the two classes [23].

It is necessary to normalize attributes so that the distance is not dominated by a particular variable. For instance, if we have two attributes, height and wage, which range, respectively, from 1.5m to 1.8m and from 10,000€ to 1,000,000€, the wage attribute is going to have a much greater influence on distance.

Normalization is performed by applying the formula $X_s = \frac{X-Min}{Max-Min}$ for each variable X, where Max and Min are the maximum and the minimum value of that variable throughout all the training samples.

Of course, discrete features with n values must be transformed into $n-1$ boolean features when applying the algorithm.

Advantages and disadvantages

KNN can learn very complex functions and does not lose information. However, it is slow at classifying (although preordering and indexing can help) and is affected by irrelevant features and outliers.

2.1.2.5 Neural Networks

Artificial Neural Networks [24] are inspired on the human brain. They are based on a series of "neurons" connected by edges. A neuron receives a signal and can process it before transmitting it to other connected neurons. Signals are commonly real numbers, and the output of each neuron is computed by a non-linear function applied to the sum of its inputs.

Neurons

Like biological neurons, an artificial neuron, also called perceptron, has input (dendrites) and output (axon) connections. Using the input values, the neuron makes a computation and then outputs a single result. Thus, a neuron can be understood as a mathematical function.

Internally, the neuron performs a weighted sum of all the input values. This sum depends on the weight assigned to each incoming connection. Therefore, every input can affect the neuron with a different intensity. The sum is then passed to an *activation function* (non-linear), which gives the output value.

Having explained these characteristics, it is easy to understand that a neuron can be formulated as $y = \varphi(\sum_{i=0}^{m} w_i x_i)$ where φ is the activation function, w is the vector of weights and x is the vector of inputs. Figure 2.7 details the functioning of a neuron.

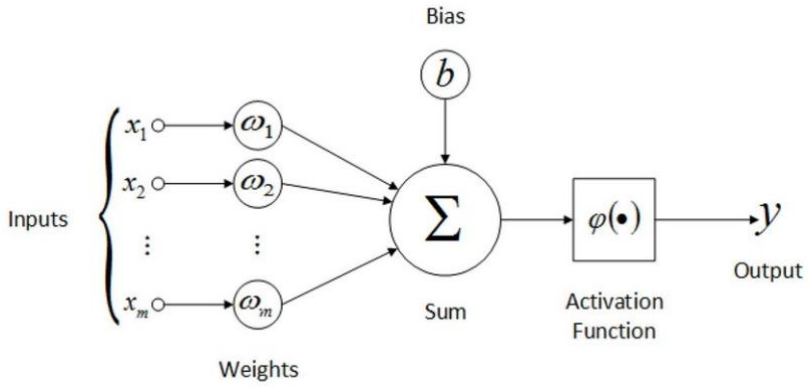

Figure 2.7. Artificial Neuron [25].

Multilayer Perceptron

A Multilayer Perceptron, shown in Figure 2.8, is a class of feed-forward (finite, acyclic) Neural Network which has at least three fully-connected layers. The first layer is called input layer, which communicates to one or more hidden layers, where the actual processing is performed. Finally, the last layer is called the output layer. Each node in a layer connects with a certain weight to every node in the following layer.

Two neurons placed on the same layer receive the same input information from the previous layer and send the output to the following layer. Using a multi-layer structure allows us to work with hierarchic knowledge. The more layers the network is composed of, the more complex the elaborated knowledge can be. This is what gives the name to Deep Learning.

The activation function must be non-linear. Otherwise, stacking many hidden layers would be useless, since the final output would still be a linear combination of the original input data (equivalent to a single layer), thus not being able to correctly classify data in non-linear problems.

Learning occurs by changing the weights each time a piece of data is processed, based on the error in the output compared to the expected result. This procedure is called *backpropagation*.

Changing the weight on the input edges allows us to modify the resulting function but is not enough if we want to shift it. This is why bias units are introduced in every non-output layer as neurons with output value 1 and with no input connections, which are connected to the standard neurons (as always, with a given weight). They fulfill an equivalent role to that of the constant b of any linear function $y = ax + b$.

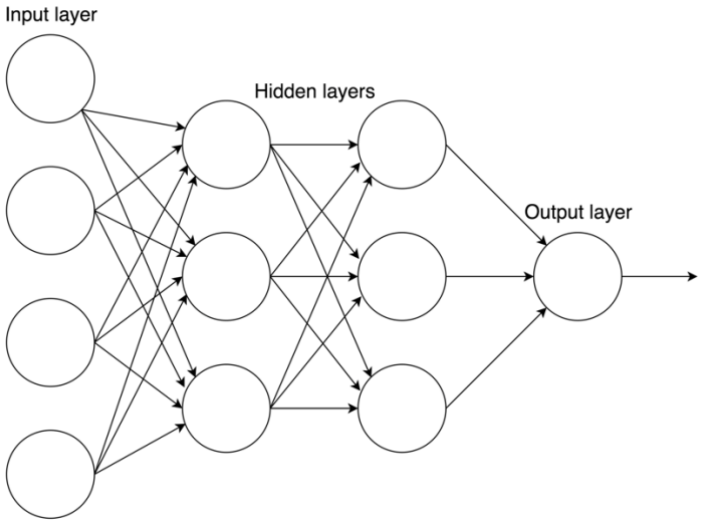

Figure 2.8. Architecture of a Multilayer Perceptron.

Advantages and disadvantages

In general, Neural Networks are some of the most used ML systems nowadays, backed by copious amounts of literature, and good to model non-linear data with a large number of input features (like image classification).

Nonetheless, they are black-box (it is not possible to know what is happening inside), hard to train, computationally expensive, slower than other alternatives like SVM or decision trees, and usually require large training datasets.

2.1.2.6 Naïve-Bayes

Alongside decision trees and neural networks, Naïve-Bayes [26, 27] is one of the most used Machine Learning algorithms. It is a probabilistic model which assumes that all the features are independent on each other given the class, as it can be seen in Figure 2.9. This assumption works surprisingly well in most cases, even if the features are not actually independent, and makes the algorithm easier to apply.

Let $X = (x_1, \dots, x_n)$ be a sample with n independent features. The probability of a hypothesis being true or, in this case, the probability of the sample belonging to the class $c_i \in \{c_1, \dots, c_k\}$, can be represented as $P(c_i|X)$. This is called the posterior probability and is the value we want to obtain.

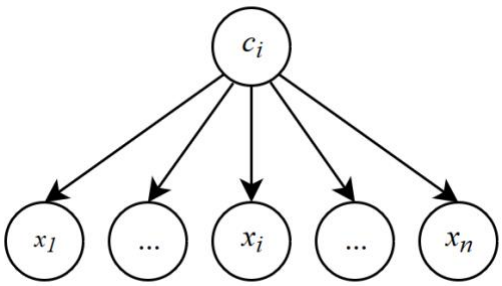

Figure 2.9. Dependency relationships between the features in a Naïve-Bayes classifier. The features are independent on each other.

So, given a sample X, the classifier would predict that the sample belongs to the class with the highest posterior probability, that is, $c^* = \underset{1 \le i \le k}{\operatorname{argmax}} P(c_i|X)$.

According to Bayes' theorem, this probability can be decomposed as $c^* = \underset{1 \le i \le k}{\operatorname{argmax}} \frac{P(X|c_i)\,P(c_i)}{P(X)}$. This way, we can infer the target probability from known probabilities. We can then get rid of the denominator since it is constant for all classes and transform the numerator into $P(X, c_i)$.

To compute $P(X, c_i)$ we need to apply the chain rule, which allows us to rewrite the joint distribution using conditional probabilities. Thus, for a given class, $P(X, c_i) = P(x_1, \dots, x_n, c_i) = P(x_1|x_2, \dots, x_n, c_i)P(x_2, \dots, x_n, c_i) = \cdots =$
$P(x_1|x_2, \dots, x_n, c_i)P(x_2|x_3, \dots, x_n, c_i) \dots P(x_{n-1}|x_n, c_i)P(c_i)$.

As we can expect, given data sets with many attributes, this computation would be expensive (we would need to compute the product of $n + 1$ conditional properties). However, assuming conditional independence among features, it is easy to see that

$P(x_j|x_{j+1}, \dots, x_n, c_i)$ is equivalent to $P(x_j|c_i)$. This way, we can approximate the distribution as simply $c^* = \underset{1 \le i \le k}{\operatorname{argmax}} P(c_i) \prod_{j=1}^{n} P(x_j|c_i)$.

Advantages and disadvantages

Naïve-Bayes classifiers are computationally fast, easy to train (even with a small dataset) and simple to understand, they work well with high dimensional data and they are not sensitive to irrelevant features. On the other hand, performance is not as good if the assumption of independence is not met.

CHAPTER 3. VIDEO CODING

The goal pursued by video coding is to reduce the amount of redundant information in video sequences while maintaining adequate quality standards and adapting to the changing formats where the video is to be consumed.

Multiple researches and advancements have been conducted regarding video, image and audio coding. The always growing popularity of video content has been motivating this continuous progress, creating the need for fastest and more efficient compression techniques and coding standards.

First, it is necessary to understand the general concepts behind video coding, in order to get deeper into the most relevant techniques and standards used in the industry, from the more archaic versions to modern standards like HEVC. Therefore, in this section, the basic concepts about video and, thus, image, are briefly introduced, along with the different techniques used for encoding and compression. A more in-depth analysis of the H.264/AVC and HEVC standards is then performed, since they are the two standards this project is based on.

3.1 General concepts

Images are composed of a set of pixels. In monochromatic images, each pixel has a brightness value, usually represented with 8 bits, while in colored images, each pixel has commonly three components (R, G and B), each component defined by 8 bits. The different combinations of these three color components allow the representation of any color perceived by the human eye.

A video is made of a temporal sequence of images, called frames. Human perception of the quality of a video signal is based on the resolution (size), the frame frequency (FPS) and the scan type (interlaced or progressive). The number of frames per second needed to transmit a feeling of fluency in the movements is between 25 and 30 frames [28]. Interlaced video distinguishes between high field, made of the odd-numbered lines of the frame, and low field, made of the even-numbered lines, and is suited for sequences with a lot of movement. Progressive video, instead, does not include the concept of field, and is more useful for sequences with higher detail and less movement and is the most commonly used scan type nowadays.

In the domain of video transmission, the bitrate, measured in bits per second, is defined as the amount of information processed or transformed in relation with the time.

3.2 Compression techniques

We live in a world where image and video resolutions are constantly increasing at an even faster rate than the advancements in hardware and communication networks. Therefore, it is very important to be able to compress signals effectively.

Video compression refers to the process carried out by the various standards in order to reduce the amount of information representing a sequence, thus reducing the bitrate, while maintaining a good quality for the target application.

There are many compression techniques, which depending on the reversibility of the encoding-decoding process can be classified into lossless and lossy.

- On the one hand, lossless strategies provide an exact reproduction of the content, where the output of the encoding and decoding process is a faithful copy of the original, being therefore a reversible process.

- On the other hand, when using lossy techniques part of the information is lost, so the information recovered after encoding and decoding is not identical to the original. They are also known as irreversible processes.

The most commonly used techniques in the main compression standards are included in the latter group, since the compression rates achieved are much higher.

A further classification can be performed considering the nature of each technique.

- **Sample coding**, in which only information from pixels or individual samples is used for compressing the original signal.

- **Perceptive coding**, based on human psycho-visual perception.
- **Transform coding**, which transform information to a different domain where information, from a perceptive point of view, is gathered in a small number of samples.
- **Predictive coding**, which exploits the temporal and spatial correlation of signals to code the information efficiently.
- **Sub-band coding**, in which the signal is divided into several frequency bands and each band is then compressed according to its relevance.
- **Vector quantization**, in which a set of samples (vector) is coded based on a list with pre-stablished values, trying to minimize the introduced error. This technique is common in low-resource devices.

Some of these techniques are explained in-depth in the following paragraphs.

3.2.1 Sample coding

3.2.1.1 Quantization

Quantization is a lossy compression technique which divides the continuous input value range in subranges each with a unique output value. The factor of quantization (Qstep) determines the amount of compression to be applied. This value is defined in relation with a quantization parameter (QP), whose maximum value is determined in each standard and varies from one to another. The higher the QP value, the most intense the compression applied to the input signal, and the biggest the granularity of the *Qstep*, the higher the flexibility to control the output bitstream. However, high values entail a higher coding cost, so it is important to find an appropriate balance.

Figure 3.1 shows a basic example of application of linear quantization with a compression factor of 8:2 to a sample 8-bit image, so that the resulting image is encoded with only 2 bits.

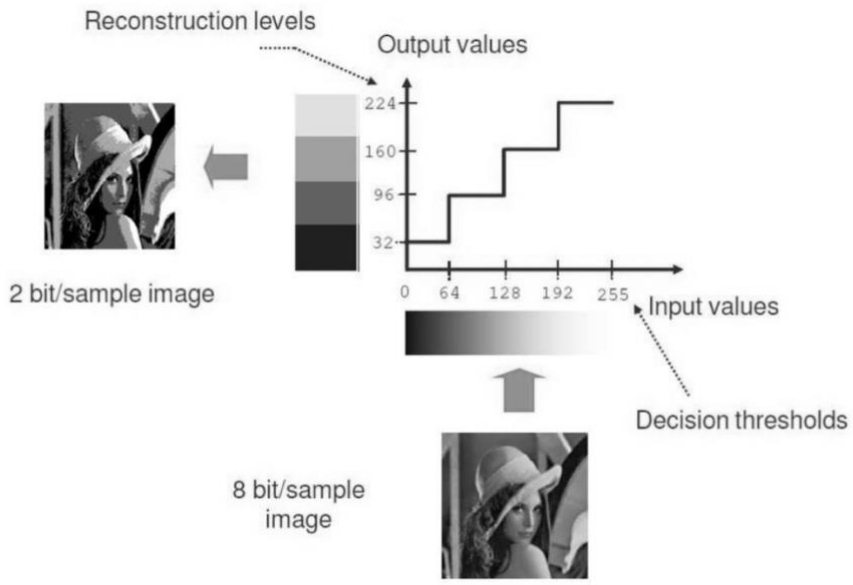

Figure 3.1. Linear quantization with a compression factor of 8:2 [29].

3.2.1.2 Run-length coding

Run-length coding, or RLC (also known as Run-Length Encoding or RLE), is a very simple lossless data compression technique, useful when long sequences of the same value are expected. Its consists on coding the longitude between two different values, instead of all the identical values of that sequence. For instance, a long sequence like $WWWWWWWWBWWWWWWWWBWWWWWWWWWWWWWWWWWWWWBBWW$ can be represented in a more compact way: $8W1B7W1B18W2B2W$. This is especially useful in combination with other techniques which yield long sequences of repeated values, like quantization.

3.2.1.3 Entropy coding

Entropy coders focus on assigning variable-length code (VLC) words to each sample. Shorter words are assigned to the most likely sample values and longer words are used for those which are less likely. The shorter words do not constitute prefixes in the longer words, so the classifier can recognize them and delimit their bounds.

A probability model is used to assign code words. The better the model, the greatest the level of compression achieved. If the model is wrong, the result can even end up being an expansion of the representation.

The most commonly used algorithms for entropy coding are Huffman coding and Arithmetic coding. All these techniques are lossless, but do not usually achieve more than a

factor of compression of two, so they are used in combination with others to obtain a greater compression.

The algorithm used in Huffman coding is the following. First, the values are ordered according to their probabilities and then the two less likely symbols are grouped, creating a new symbol whose probability is the sum of both. The second step is repeated until only one symbol is left. This is illustrated in Figure 3.2.

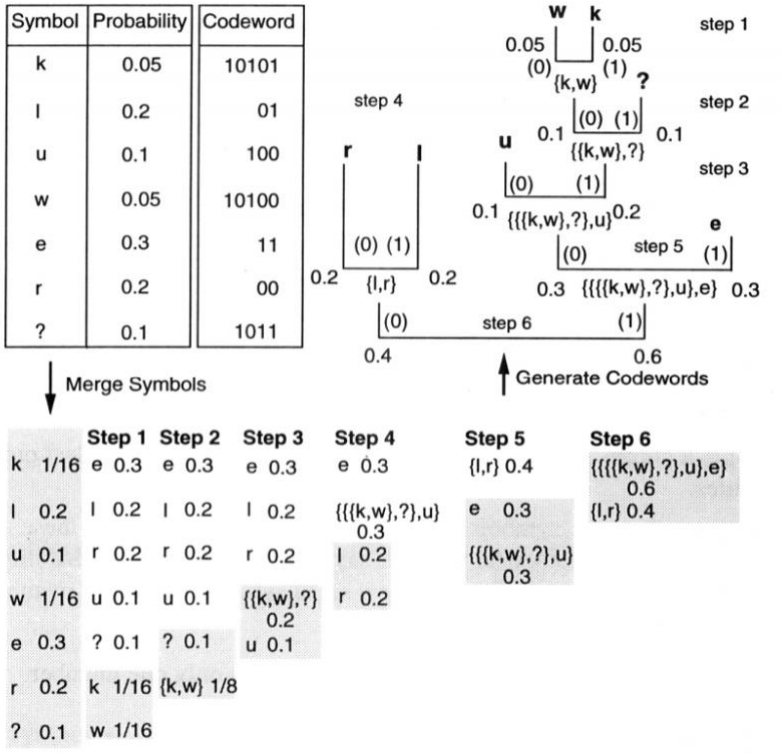

Figure 3.2. Example of the procedure of Huffman code creation [29].

3.2.2 Perceptive coding

As mentioned in 3.1, when using the RGB color format, each color is represented by a combination of red, green and blue, making use of 3 matrices and a total of 24 bits to represent the color of each pixel. However, the amount of information needed to depict the tonality of the millions of pixels that an image can be composed of makes this representation unreasonable in a world where image and video resolutions are constantly increasing at an even faster rate than the advancements in hardware and communication networks. We need ways to compress the signal, and working with luminance and chrominance is one of the most basic techniques.

The human eye is more sensitive to brightness (luminance) than to color difference (chrominance), so this technique is based on transforming the color space from RGB to a more lightweight format: YUV.

The YUV format represents the signal as one luminance (Y) and two chrominances (U and V), performing a transformation in the color space from RGB to the new format, taking into consideration that the human eye is more sensitive to green and less sensitive to blue. Then, it compresses (subsamples) only chrominances at half or one fourth of the luminance values, which remain unchanged. This yields an imperceptible or virtually negligible decrease in subjective quality.

The most commonly used subsampling method is 4:2:0, although many others are possible. Some examples are:

- 4:4:4: The three components are left with the same frequency. There is no information saving.
- 4:2:2: Both chrominances are sampled at half the frequency of the luminance, in horizontal direction. The bandwidth of the signal obtained from this subsampling is reduced in one third without a remarkable difference in quality.
- 4:2:0: The chrominances are subsampled in both horizontal and vertical direction, being, thus, one fourth the initial size. The bandwidth is reduced in half.
- 4:1:1: In this case, chrominances are sampled at one fourth of the luminance signals in horizontal direction. As with 4:2:0, the bandwidth reduction is of one half.

An example of color space transformation from RGB to YUV with a 4:2:0 subsampling method and the resulting information saving is shown in Figure 3.3.

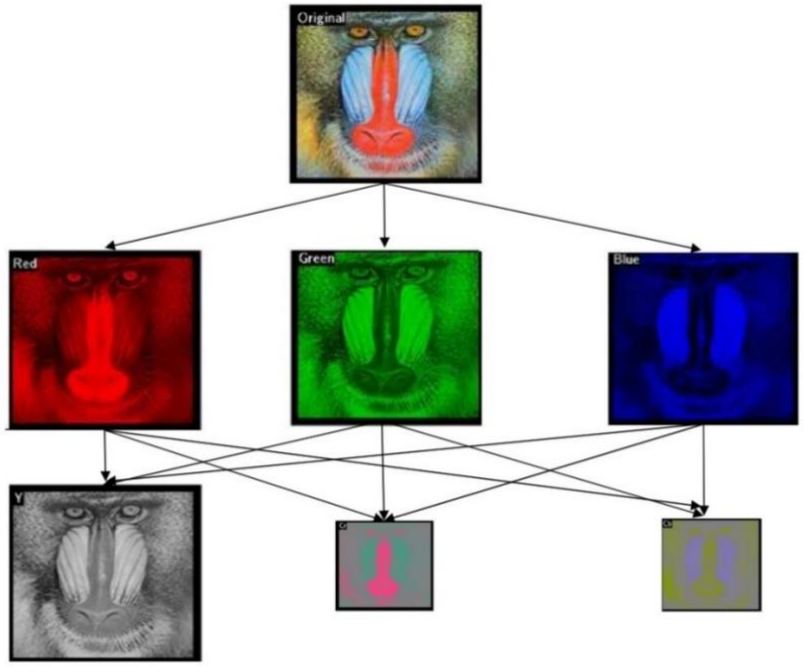

Figure 3.3. Example of transformation from RGB to YUV with a 4:2:0 subsampling method [30].

3.2.3 Transform coding

Transform coding is a (mostly) lossless compression technique that aims to reduce spatial redundancy. For this, the signal is transformed to a different domain, in which just a few coefficients contain most of the information and the rest of the coefficients have negligible values. Thereby, in the new domain, the signal representation will be much more compact and able to be represented with only some of the coefficients.

The most common quick transform is the Direct Cosine Transform (DCT). Despite its popularity, DCT does not provide the best results compared with other transforms such as the Karhunen-Loeve Transform (KLT), but it is used in most standards given its implementation simplicity and its independence of the processed image.

This kind of transforms are very costly, with an $O(n\ log(n))$ complexity. To reduce the amount of operations required, the image is divided in, usually, 8x8 blocks and then the transform is applied to each block separately.

In the case of the DCT, the resulting 8x8 matrices are composed of values that divide the colors of the image in frequencies ranging from the lowest, which represents the most sensitive elements for the human eye (slow changes), in the upper left corner, to the highest, representing less sensitive areas (quick changes), in the lower right corner. The value of the top left corner (DC) indicates the value of the dominant color and the rest of the values (AC)

correspond to color changes across the block. The change from the spatial domain to the frequency domain fruit of the application of a DCT is shown in Figure 3.4.

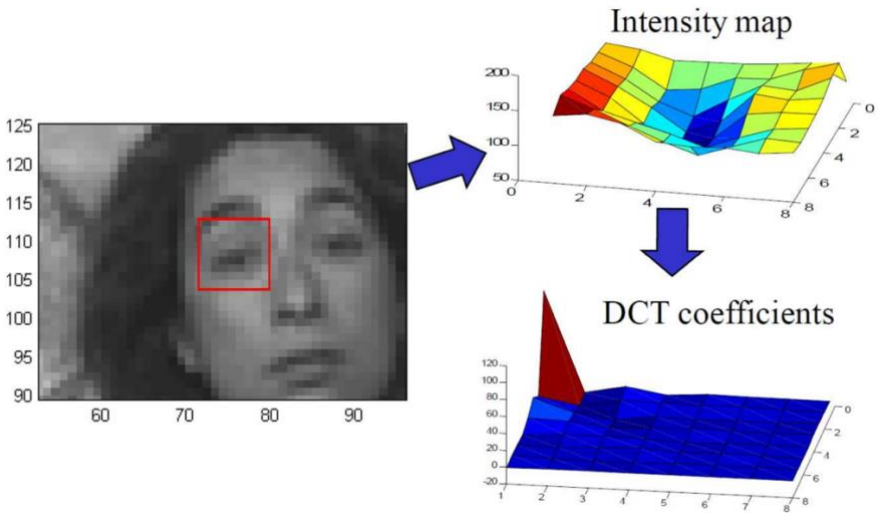

Figure 3.4. Compactation of the information after applying a DCT [29].

After applying transform coding, a quantization process commonly takes place, followed by RLC, since quantization produces lots of zeroes on AC coefficients. This is usually followed by an entropy coding before sending the information. A complete example can be seen in Figure 3.5.

3.2.4 Predictive coding

Predictive techniques strive to eliminate redundancy in the spatiotemporal domain in which samples are represented. They predict a sample's value through its spatial or temporal neighborhood and only code the difference between predicted and current values, since the variation between the values of two consecutive samples tends to be slight. These differences are called residues, and their range is usually lower than that of individual samples, which leads to a reduction in the number of bits used for encoding said range.

As mentioned, these techniques work in both the spatial and temporal domains and can, thus, be classified into two groups: intra-frame and inter-frame predictions.

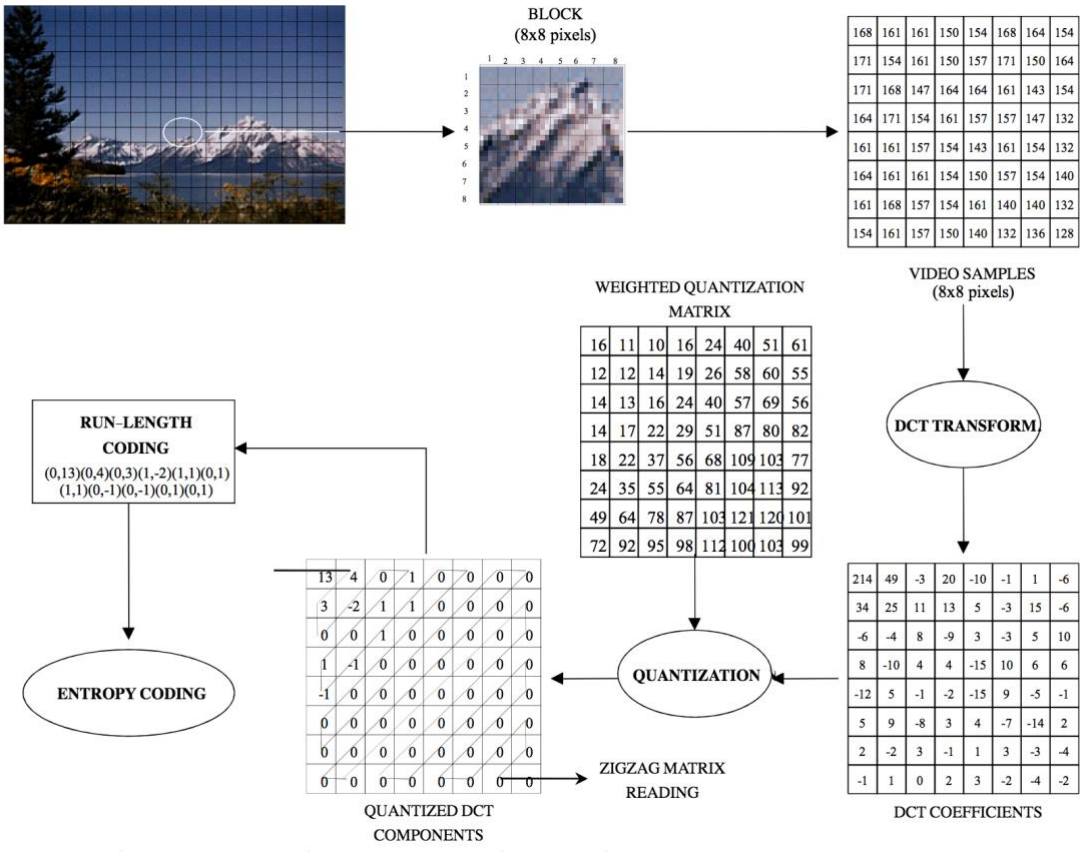

Figure 3.5. Example of a coding system based on the DCT [29].

3.2.4.1 Intra-frame prediction

Intra-frame prediction for a block of pixels is performed by making use of the available neighboring (or frontier) pixels, whose values are extrapolated to form a predictor. It is used to eliminate **spatial** redundancy, that is, to eliminate correlative information to regions inside a frame.

Commonly, pixels from the line above the current block and pixels from the most recent column of reconstructed blocks are combined to generate a prediction which is then subtracted to the real sample in order to only encode the residue. This process is exemplified in Figure 3.6.

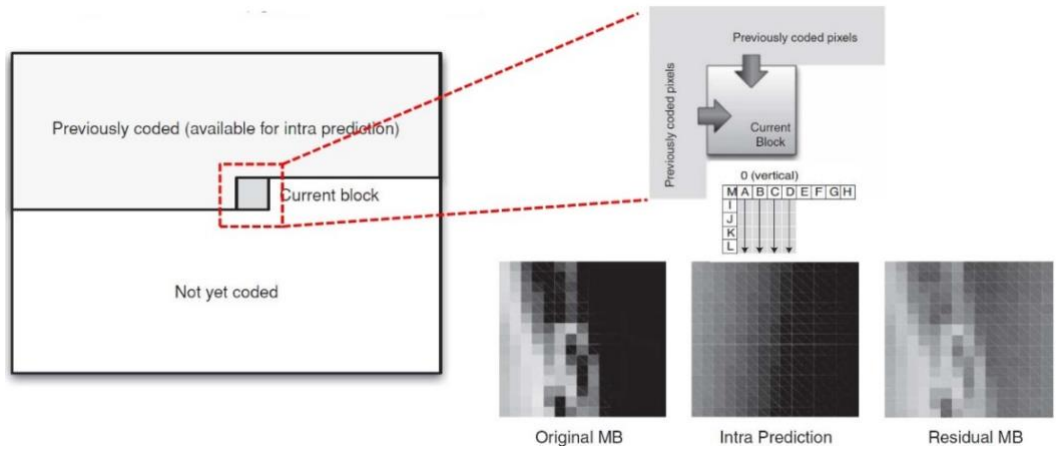

Figure 3.6. Example of spatial prediction using a vertical mode [29].

3.2.4.2 Inter-frame prediction

As suggested by its name, the goal of inter-frame prediction is to eliminate the redundancy in the **temporal** domain, for which adjacent images are analyzed. It is the compression technique with the highest computational cost.

The information contained in two adjacent images is very similar. This means the temporal redundancy among them is very big, since the same components of the scene are supposed to appear in consecutive frames, although maybe in different positions. Motion estimation and compensation techniques help eliminating this redundancy by evaluating in an efficient manner the movement of the objects from one image to the following.

With the motion estimation method, a prediction image is created by means of motion estimation (taking the current image and the preceding one as inputs) and a motion compensation. The difference between the current and the predicted image is called motion-compensated residue. In a typical video sequence, this residue will have smaller, close to zero numbers, coded with just a few bits.

Motion estimation is often performed by macroblock (MB) comparison. A macroblock usually consists of four 8x8-pixel blocks (making a total of 16x16 pixels) of luminance information and a variable number of chrominance blocks, depending on the sampling format. The area of the adjacent frame with greater similarity to the current macroblock is called predictor. The difference between the current macroblock and the prediction constitutes the motion-compensated residue, which is then sent to the decoder along with a motion vector (MV) pointing to the matched macroblock so that it can recognize the relative position of the prediction in relation to the original macroblock.

Despite smaller macroblocks often lead to more accurate predictions, that is, to smaller residues, they also imply a higher amount of motion vectors, which means a bitrate increase.

Therefore, some video coding standards use different macroblock sizes, as detailed in the following sections.

In conclusion, motion estimation is the process of looking for a prediction using search algorithms (block matching algorithms) and stablishing a search area (in order to reduce their high cost), while motion compensation is the application of the obtained MVs and residues to get a new image.

A graphical representation is shown in Figure 3.7. There, the motion vector expresses the movement of the object and the illumination change will be the obtained residue. It can be noted, in line with what was commented before, that other compression techniques such as DCT or RLC are applied after inter-frame prediction. This is a common pattern in most video coding standards.

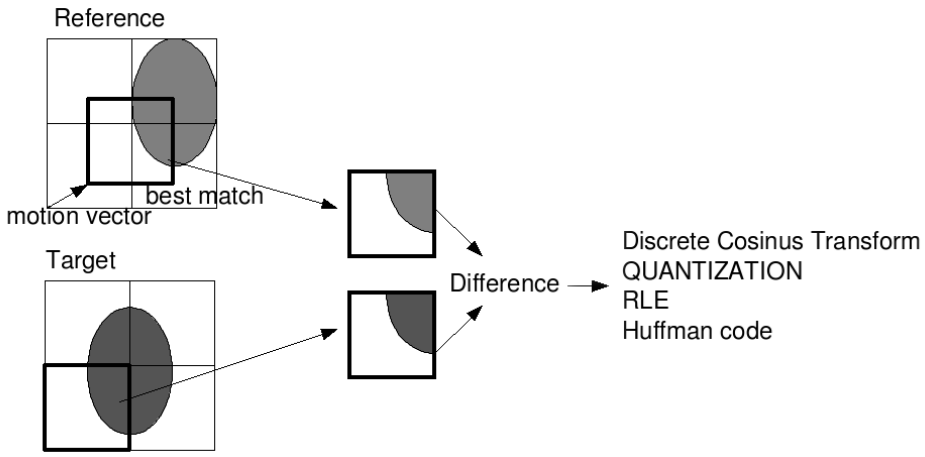

Figure 3.7. Inter-frame prediction process. Several compression techniques are applied with posteriority [31].

It is important to note that the most computationally expensive phase of temporal prediction is motion estimation, especially the block matching algorithms.

It is important to dedicate some space to talk about some of the problems that can arise from the inter-frame prediction process:

- If the block matching algorithm fails to find a suitable match (which is especially common in scene change images), the size of the motion vector plus the residue will be greater than the default encoding. In this case, the encoder would send the raw encoding for that block instead.

- If the matched block has also been encoded applying intra-frame prediction, possible errors from that encoding will be propagated to the next block.

To address these drawbacks, the idea of intra-frame is introduced. An intra-frame, or I-frame, is an only intra coded frame, which serves as an anchor image, avoiding the propagation of errors which can harm image fidelity and supporting decoder synchronization. This leads to the concept of Group of Pictures (GOP), a layout containing, commonly, three types of images: I-frames, P-frames and B-frames.

P-frames allow forward inter-frame prediction with only one reference image (the previous one) and B-frames allow bidirectional temporal prediction using two reference images (the previous and the following). The frame arrangement in the GOP depends on the standard, being *IBBPBB*... the most common structure.

3.3 Video coding standards

Different standards for both image and video coding have been developed in the last decades, published by the International Telecommunication Union (ITU) and the International Organization for Standardization and International Electrotechnical Commission (ISO/IEC).

Some of these standards include JPEG/JPEG2000 for image compression and, chronologically, H.120, H.261, MPEG-1, MPEG-2, H.263 or MPEG-4 (part 2) for video coding:

- H.120 [32] was the first digital video compression standard. It was developed by the ITU in 1984. A revision with improved features was released in 1988. The video was not of adequate quality for practical use, but the knowledge acquired by the organization was important for the birth of its successors.

- H.261 [33] was first proposed by the ITU in 1988. It was focused on videoconference and videotelephony, working at 15 frames per second and using low resolution formats. It used a DCT transform followed by quantification, RLC and an entropy coding (VLC). The majority of modern video standards are based on H.261.

- MPEG-1 [34] was the first standard proposed by the ISO/IEC, in 1993. It was based on H.261 and it introduced bidirectional prediction (B-frames) and was mainly applied to CD-ROM storage. This was also the origin of the GOP.

- MPEG-2 [35], proposed in 1995, is a continuation of the previous standard. It was the first standard approved by both organizations (although in ITU's terminology it is named H.262) and is focused on coding video with associated audio for digital television and DVD.

- H.263 (ITU, 1996) [36] is based on H.261 and includes several improvements on the encoding tools and was able to describe a video scene as a series of objects. It was not designed for a particular application but more like an all-rounder standard.

- MPEG-4 (part 2, ISO/IEC, 1999) [37] is a standard for object-based video coding, which aimed to work independently of the transmission or storage medium.

With the increment on video distribution through the Internet, improving compression standards proved necessary. To achieve that goal, a joint team was created between the ITU and the ISO/IEC, giving rise to the Joint Video Team (JVT).

This team, with the objective of developing a more flexible specification in terms of compression and computational cost, able to act over applications of different nature and with improved resistance to errors, published the H.264/AVC (Advanced Video Coding) recommendation in 2004, which managed to reduce bitrate by 50% when compared to MPEG-2 while maintaining the same image quality. Figure 3.8 represents a timeline of some of the different standards published by the two organizations and those developed by the joint team.

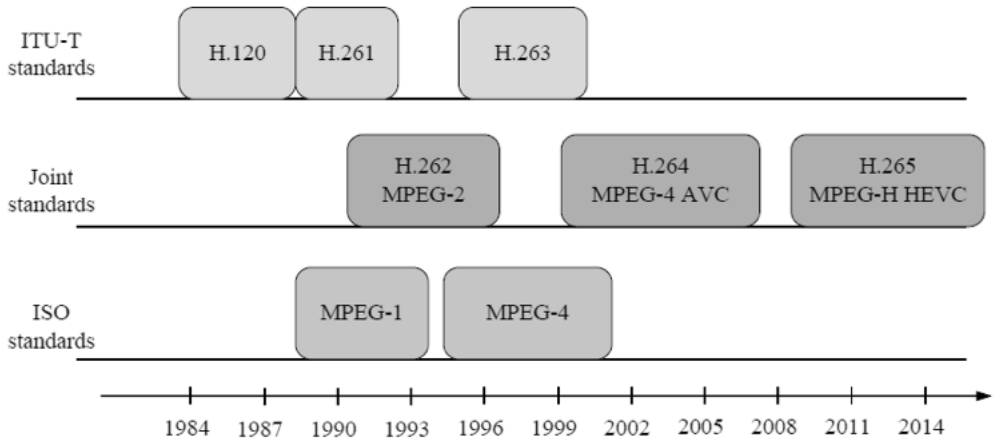

Figure 3.8. Timeline of the emergence of some of the most important video standards.

3.3.1 H.264/AVC

H.264/AVC is a widely used standard, developed in 2003. Its main applications are, for instance, HD television and Blu-Ray Discs, although it is sufficiently versatile to be able to adapt to all kinds of applications. Several extensions were published in the following years, including functionalities for scalability, which allow to adapt the video to a wide

variety of transmission circumstances and devices with different screen sizes and computation capabilities.

It includes the concept of Network Abstraction Layer (NAL) [38], which enables to maintain the syntax of the video sequence in any kind of network. This data is included in NAL units, which contain a number of bits starting with a header that indicates the number of bits contained. NALs allow the distribution of data in various separate packages, improving the resiliency against transmission failures.

Similar to previous standards, H.264/AVC is based on a block-based coding, but with several efficiency improvements like intra-frame prediction, motion compensation with different macroblock sizes, motion vectors with precision of ¼ or ⅛ pixel, temporal prediction based on multiple frames (for P-frames), more flexibility for B-frames allowing hierarchical patterns, multiple transforms on 4x4 blocks and additional entropy coding techniques. These new features are explained in the following paragraphs.

Image partitioning is performed at various levels. An image is divided into several slices composed of a sequence of 16x16 macroblocks. Macroblocks are the basic processing unit and can be further divided into smaller units (16x8, 8x16, 8x8, 8x4, 4x8 or 4x4 pixels). They are composed of luminance and chrominance pixels, and they can depend on other macroblocks in the same slice, but not on others from outside their own slice.

3.3.1.1 Intra-frame prediction improvements

Following the classification from previous sections, macroblocks and slices can be classified into I, P or B. A slice is said to be of type I if all its macroblocks are of type I (intra predicted). If some of the macroblocks are coded using inter prediction with at least one past motion-compensated prediction, the slice is said to be of type P. Lastly, it is said to be of type B when some macroblocks are coded using inter prediction with at least two (past and/or future) motion-compensated predictions.

If a block or macroblock is coded as intra, a prediction is performed for that block or MB taking the previously coded blocks or MBs in that image. Only samples belonging to the same slice can be used for the sake of slice independency. This means there will exist spatial propagation of errors, but only along the same slice.

The prediction can be formed for a macroblock or for each 4x4 block. Different intra prediction models are available depending on the size of the block:

- For 16x16 luminance macroblocks, 4 prediction modes can be used: vertical, horizontal, DC (adequate for flat surfaces) and plane. See Figure 3.10 for more detail.

- For chrominance, the same modes apply, but for 8x8 blocks. The same prediction is applied to both chrominance blocks.

- For 4x4 luminance blocks there are 9 prediction modes more suitable for highly detailed areas, displayed in Figure 3.9 along with an example.

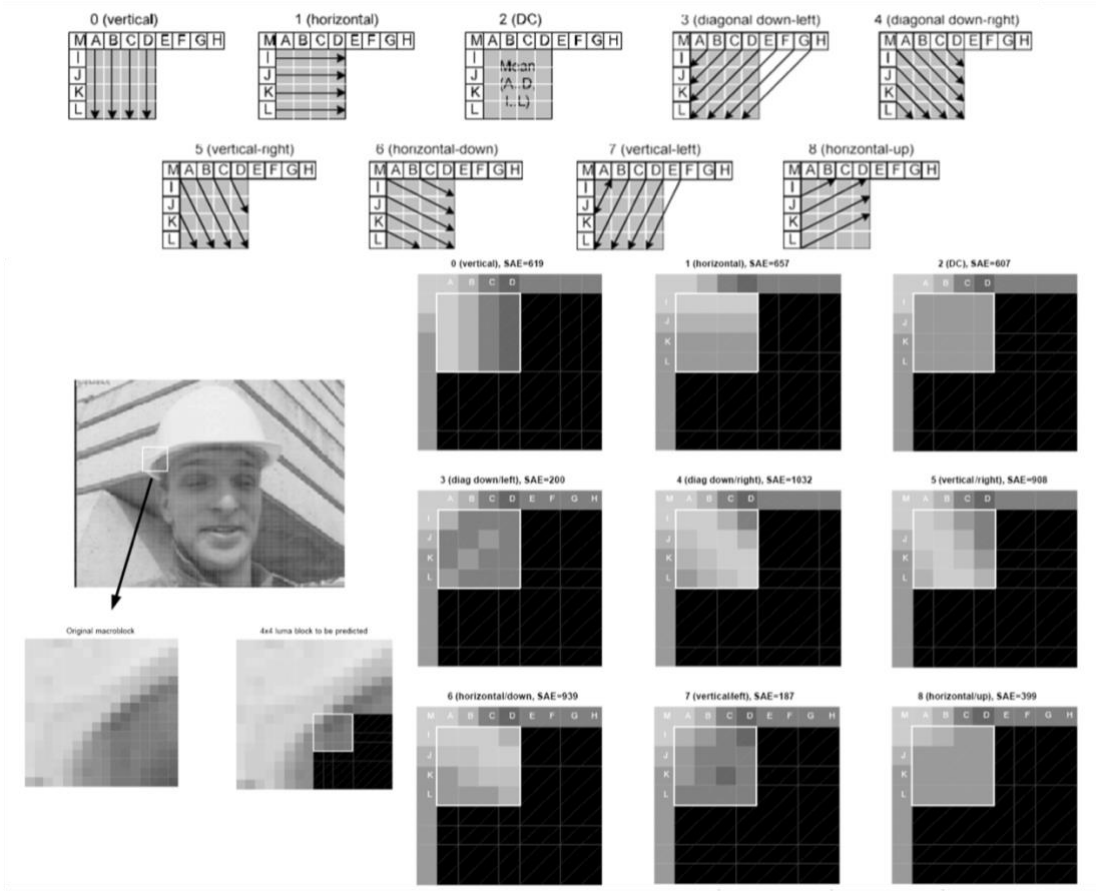

Figure 3.9. Above: the nine prediction modes which can be used for 4x4 luminance blocks. Below: an example of intra prediction comparing every mode. The selected mode is 7, since it provides the most accurate results [39].

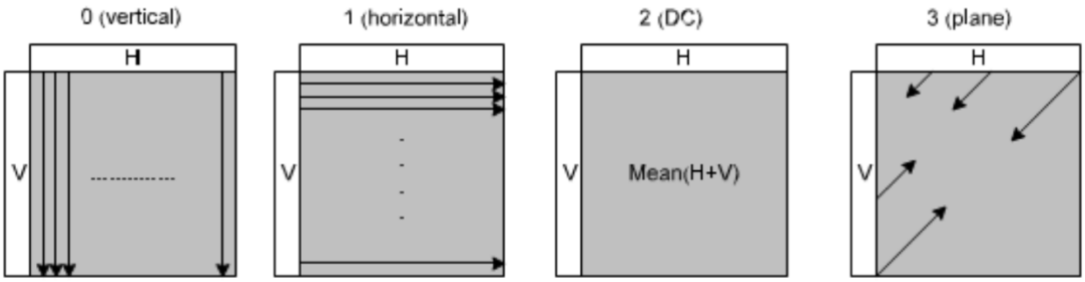

Figure 3.10. The four prediction modes available for 16x16 luminance macroblocks. In case 2, the average of some neighbor pixels is calculated [39].

3.3.1.2 Inter-frame prediction improvements

Regarding inter prediction, up to 16 motion vectors can be generated for each MB for motion estimation, which are coded differentially but only in relation with MBs of the same slice. The encoder selects the most appropriate partitioning for each macroblock to maximize the coding efficiency. For this, every possible partitioning is tried in order to choose the best one, rocketing the computational cost of the process. The different splitting possibilities can be clearly seen in Figure 3.11.

Figure 3.11. Different coding modes in H.264/AVC: intra, skip and the different inter splittings [39].

H.264/AVC supports motion estimation based on multiple frames, which means that more than one past image may be used simultaneously to make a prediction, up to 16. Nevertheless, this has a negative impact on memory since both encoder and decoder need to store all those images temporarily.

Moreover, B-frames can now serve as reference for other frames as well as use two weighted references which can be two from the past, two from the future, or one from each.

3.3.1.3 Other features

H.264/AVC uses three transform types depending on the type of residue to encode: Hadamard 4x4, Hadamard 2x2 and Integer DCT 4x4.

The standard also proposes the application of scalar quantization, meaning that the same Qstep is used for all the transformed coefficients from the MB. The quantization factor is selected from a standardized table with 52 carefully defined values (see Table 3.1) which define the relation between QP and Qstep. The values were chosen so that each increment in the Qstep provides a reduction of 12.5% in the bitrate and the Qstep value is doubled for each increment of 6 in the QP. The QP can be different for luminance and chrominance blocks.

QP	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Qstep	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625
QP	**9**	**10**	**11**	**12**	**…**	**18**	**…**	**24**	**…**
Qstep	1.75	2	2.25	2.5	…	5	…	10	…
QP	**30**	**...**	**36**	**...**	**42**	**…**	**48**	**…**	**51**
Qstep	20	…	40	…	80	…	160	…	224

Table 3.1. QP and Qstep sizes in H.264/AVC.

Besides, a deblocking filter is applied in order to smooth sharp edges which can appear between macroblocks as a result of the use of different MB sizes for motion estimation and the application of DCT and quantization.

Additionally, regarding entropy coding techniques, two new options are proposed apart from the traditional VLC: *Context Adaptive VLC* (CAVLC) and *Context-based Adaptive Binary Arithmetic Coding* (CABAC). CAVLC requires less processing than CABAC and yields better results than previous entropy coders. On the other hand, CABAC uses dynamic probabilistic models for most of the symbols, exploiting the correlation between them via the creation of contexts, and providing a 5-15% bit-rate saving with respect to CAVLC [40].

Finally, a block diagram of AVC, gathering all the described processes, can be seen in Figure 3.12.

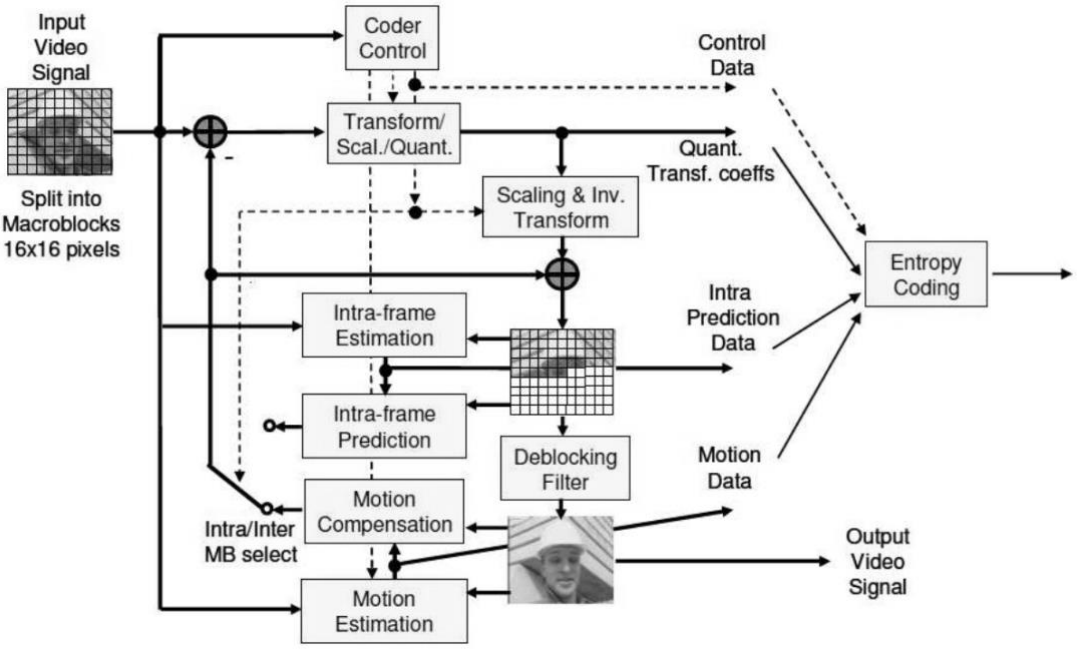

Figure 3.12. Block diagram of the H.264/AVC encoding process showing the different compression techniques used [39].

Of course, efficiency comes at a cost since, due to some of these techniques, memory and computational complexity is 4 times higher on the encoder and 3 times higher on the decoder with respect to MPEG-2.

3.3.2 HEVC

With bigger display resolutions like 4K (3840 x 2160 pixels) or 8K (7680 x 4320 pixels), a growing variety of devices (smartphones, tablets, televisions, computers…) and a huge increase in the amount of video traffic on the Internet which exceeded 50% of all traffic on 2012, 70% in 2015 and, as mentioned at the beginning of this document, is expected to exceed 82% in 2020, a newer standard featuring higher compression, better quality, improved parallel processing architectures and more image and color formats was needed.

Therefore, a new group with experts from ITU and ISO/IEC, the JCT-VC, was created. They published in 2013 the High Efficiency Video Coding (HEVC) standard, also known as H.265, which fulfills the above requirements and is able to maintain quality while reducing the bitrate by 50% with respect to H.264/AVC.

Although HEVC preserves most of its predecessor's functionalities, it also implements many new features. The main ones, summarized and compared to those of H.264/AVC in

Figure 3.13, are intra prediction with 35 modes, a residual quadtree, 64x64 to 8x8 coding units, or large-size transforms.

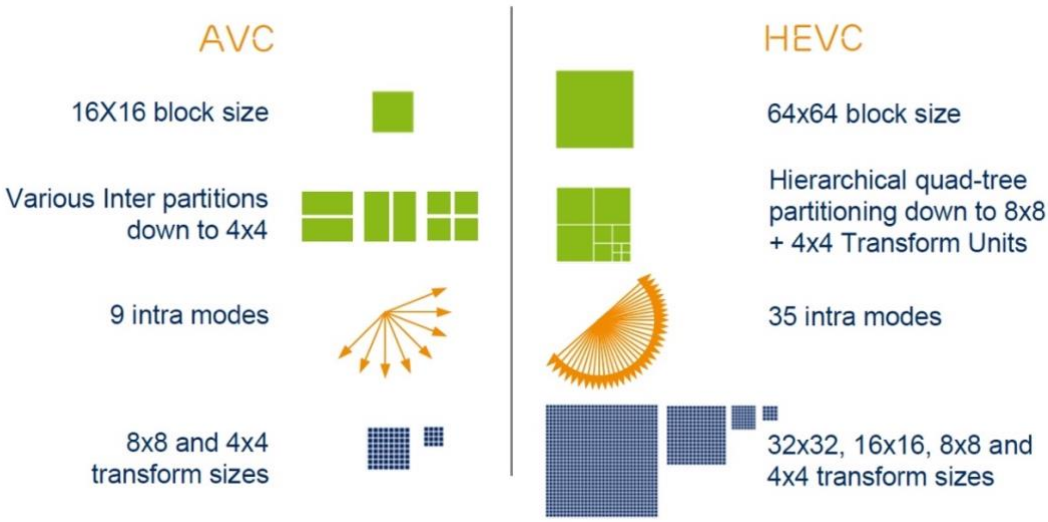

Figure 3.13. Comparison of the main features of HEVC and H.264/AVC [39].

3.3.2.1 Coding quadtree

HEVC leaves the concept of macroblock aside and introduces three very relevant concepts: Coding Units (CUs), Prediction Units (PUs) and Transform Units (TUs), providing high flexibility to adapt predictions to the peculiarities of each image.

- Coding Units define a sub-partition of the image in squared variable-sized regions, from 64x64 to 8x8 pixels which can have independent coding modes. They contain one or more PUs and TUs.

- Prediction Units are the elementary **prediction** unit. They are defined after the last partitioning level of the CU, that is, in the last level of the partitioning tree (quadtree) of the CU. For intra prediction, square, symmetric splitting is permitted. For inter prediction, PUs allow not only square or rectangular splitting (like in H.264/AVC), but also asymmetric splitting (¼ / ¾).

- Transform Units are the units where **transform** and **quantization** are applied. The shape of a TU depends on the partitioning mode of the PU (since in asymmetry cases the CU cannot be divided into 4 equal-sized blocks, but in 16) and its size can go from 32x32 to 4x4. A PU can contain several TUs, and this is also true the other way around. Note that TUs are stablished starting from the

corresponding CU, and not from the PUs. All this behavior can be better understood by analyzing Figure 3.14.

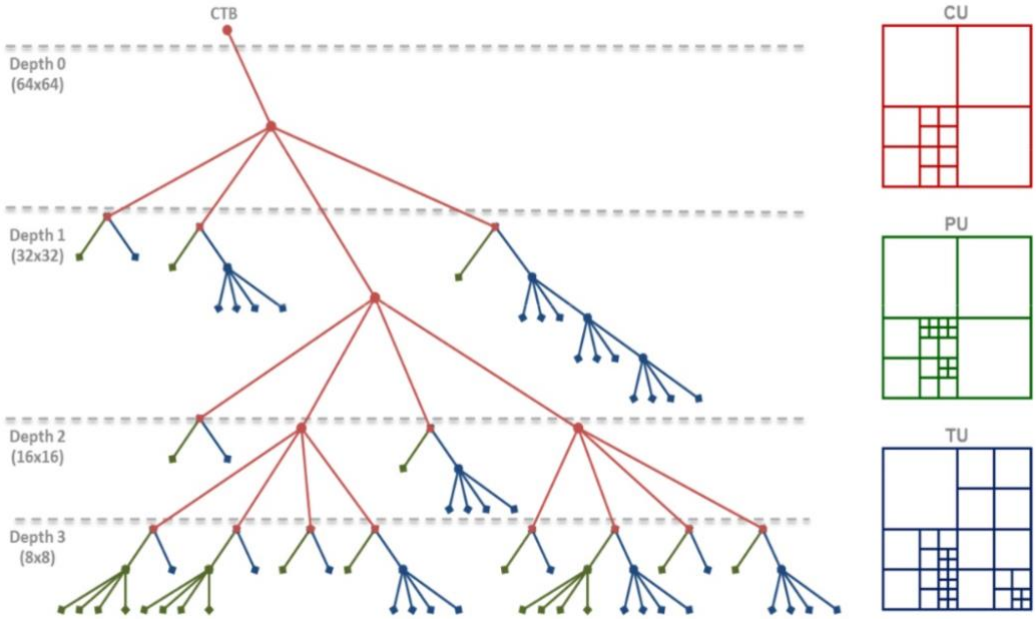

Figure 3.14. Example of a HEVC quadtree showing the splitting in the three different unit types [39].

These three kinds of elements are contained in structures called Coding Tree Units (CTUs), which contain one Coding Tree Block (CTB) for each color component (usually three blocks: Y, U and V) and correspond conceptually to H.264/AVC's macroblocks. A CTU maintains the same size as the luminance (Y) CTB, since chrominance CTBs may be subsampled, and it contains at least one CU, that can be subsequently divided. Thus, each CTU acts as a root of a quadtree[2].

At the time of choosing the next CU when applying the coding techniques, the leaves are traversed from left to right.

The symmetric and asymmetric splitting of PUs and their relationship with TUs can be seen in Figure 3.15, where the three types of units are represented. Figure 3.16 shows the comparison in terms of block sizes between H.264/AVC and HEVC.

[2] A quadtree is a tree where each parent has exactly four children.

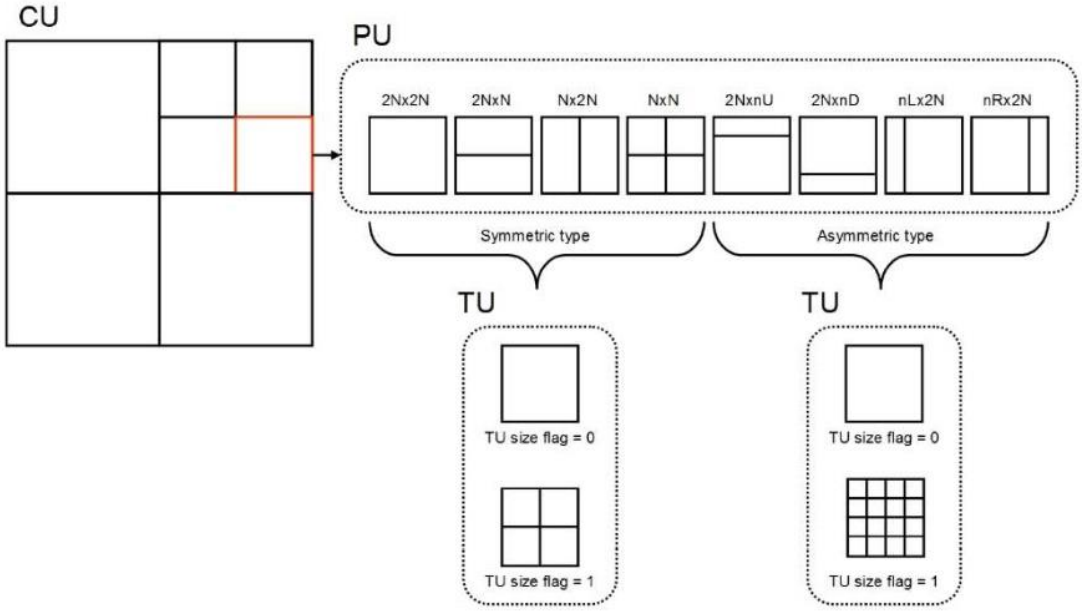

Figure 3.15. Relationship between CUs, PUs and TUs [39].

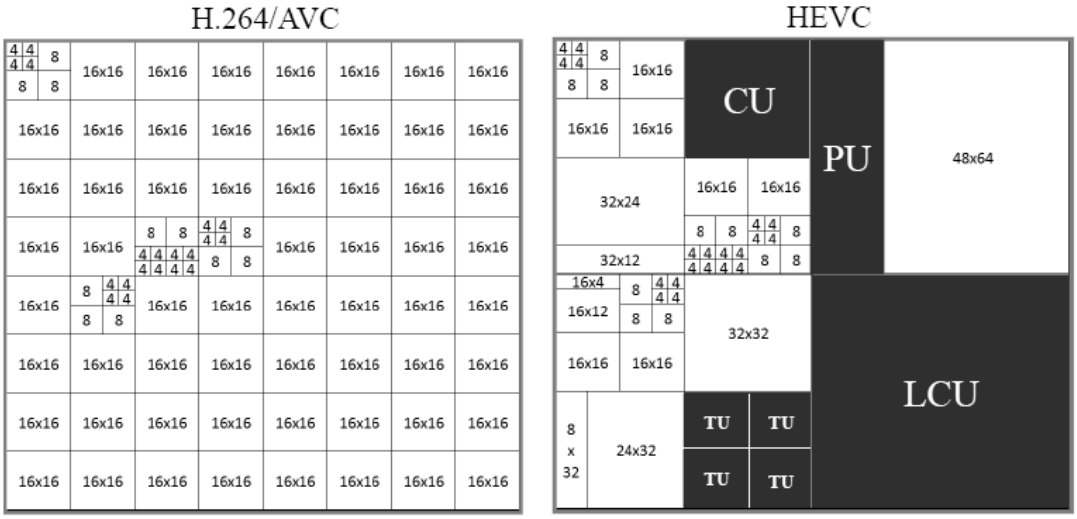

Figure 3.16. Block size comparison between H.264/AVC and HEVC [39].

3.3.2.2 Prediction improvements

Similar to H.264/AVC, different number of intra prediction modes, in this case of a total of 35, are available for each PU size. They are described in Table 3.2. The plane prediction mode in HEVC supports all block sizes defined in the standard, unlike in H.264/AVC, where it can only be applied to blocks of 16x16 pixels.

PU size	64x64	32x32	16x16	8x8	4x4
# prediction modes	5	34	34	34	17

Table 3.2. Number of prediction modes available for each PU size.

The modes use data from previously decoded neighboring prediction blocks from the same picture. All TUs within a given PU shall use the same intra prediction mode for each component [41]. For chroma components, the encoder can select the best out of 5 prediction modes: plane, DC, horizontal, vertical or a direct copy of the mode used for luminance.

Talking about inter prediction, the ¼ pixel precision from H.264/AVC is maintained in this standard, allowing to improve the quality of the generated motion vectors. If the area to predict is not exactly displaced by one or more complete pixels, but occupying parts of different pixels, this technique is going to help getting a more accurate prediction.

3.3.2.3 Other improvements

Parallelization is of foremost importance since the computational cost to achieve such a high bitrate reduction over H.264/AVC is not at all low. Current single-core technology allows decoding H.264/AVC 1080p sequences but is not enough to face real-time HEVC 2160p video decoding. Thus, HEVC includes new and improved parallelization strategies to replace those that were available in H.264/AVC.

Other important additions in HEVC are, for instance, the Sample Adaptive Offset (SAO), which complements the deblocking filter preserved from H.264/AVC, the use of CABAC as the only method for entropy coding, and the utilization of the Discrete Sine Transform (DST) besides the usual DCT.

3.3.2.4 Common Test Conditions

To test and compare the functioning of different coding software based on HEVC, a set of normalized test conditions, namely Common Test Conditions (CTC) [42], is defined. They include predefined configuration parameters and video sequences of different resolutions and characteristics, in an attempt to represent a good variety of real-life applications. The predefined prediction structures, which are illustrated in Figure 3.17, are:

- **All intra (AI)**: all the frames of the video are encoded as type I. It is an appropriate configuration for application with low delay and high bitrate, like post-production and edition, where direct access to all frames is needed.

- **Random Access (RA)**: a hierarchical prediction structure with B-frames is used. It is adequate for applications with high coding efficiency but a higher delay, like

broadcasting or streaming, due to frame reordering. An I-frame is introduced periodically to allow random access and limit the propagation of errors.

- **Low Delay P (LDP)**: the first image is of type I while the rest are P-frames. Only predictions with respect to the past are permitted, so no frame reordering is needed. It is apt for low-delay, high-efficiency applications, like videoconferencing.

- **Low Delay B (LDB)**: follows the same distribution and rules of LDP but replacing P-frames with B-frames.

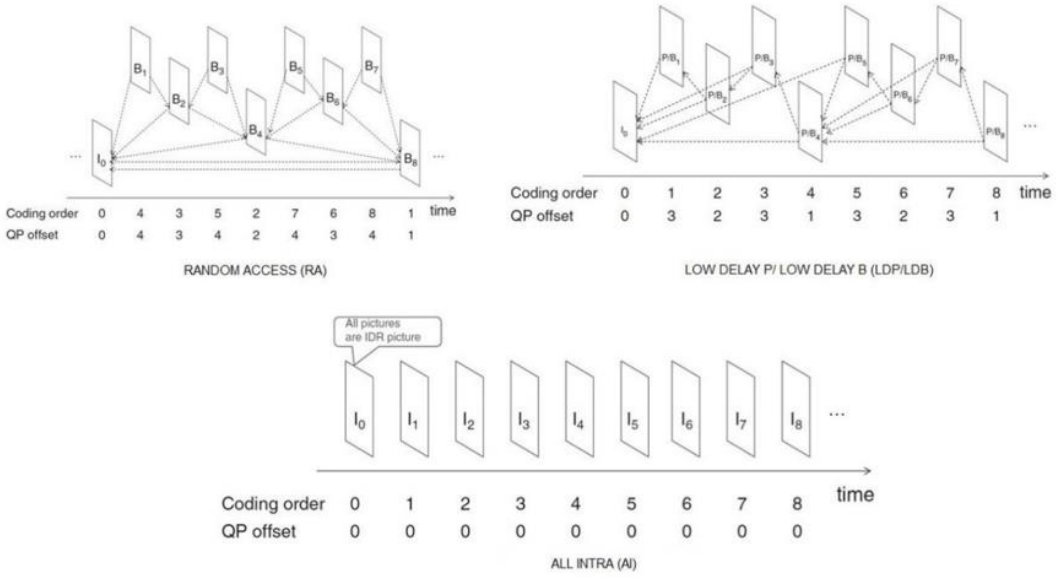

Figure 3.17. Comparison of the different prediction structures allowed in the CTC [39].

3.4 State of the art of video transcoding

Some studies about H.264/AVC to HEVC transcoding have hitherto been conducted. These proposals aim to reduce the time complexity with respect to a cascade transcoding by avoiding or accelerating operations in the second part of the transcoding process (the HEVC encoding stage) by making use of information extracted from the previous decoding stage. A chronological review of the most relevant ones follows.

The first approach came out in 2012 and proposed the exploitation of the information of H.264/AVC's bit streams [43]. For inter frames, it tries to estimate the best CU splitting, PU mode and motion vector of each PU partition using the H.264/AVC residue, modes and motion vectors. For intra frames, the CU and PU candidates are reduced. This proposal manages to accelerate the process by 70-80%, but at the expense of a BD-rate [44] of 30%,

what means that to maintain the same objective quality, the bit rate is increased by 30%. This fact makes this proposal inviable since it counters the 50% bitrate reduction from HEVC while also increasing the complexity of both encoder and decoder.

In [45], the performance of the motion vector reuse technique is studied. This is one of the most popular techniques for heterogeneous transcoding. Moreover, the paper proposes a new algorithm focused on reusing the motion vectors and a similarity metric to decide which CU partitions should be tested. The speedup achieved by this proposal is a relevant 4.13x, with a RD impact of 10.92%.

A different approach, focused on video surveillance (given the static backgrounds that characterize this type of sequences) and based on region feature analysis, is taken in [46]. This algorithm divides each frame in three regions, containing full CTU units, based on the correlation between image complexity and the coding bits of the H.264/AVC stream. The searching depth of each CTU is then decided depending on the region type. Then, the motion vectors are used to select the PU partitions optimally. This provides a 1.93x speedup compared with the eighth version of the reference software for HEVC coding (HM 8.0) [47] with a BD-rate increment of 1.73%.

In [48], a transcoder for multi-core processors is detailed. This standard takes advantage of the input H.264/AVC stream for motion estimation and mode decision. Frames are divided in several rectangular regions to be processed in parallel. This approach can provide a 70x speedup over the reference encoder HM 8.1, with a PSNR reduction of only 0.02dB.

Peixoto *et al.* [49] proposed two alternatives to map H.264/AVC macroblocks to HEVC CUs using machine learning techniques. The first one builds the model offline that is later used in the transcoding process. The second alternative uses a dynamic training with two phases: a training stage where a full re-encoding is performed while the information from the bitstreams of both standards is recorded and then used to build a model, and a transcoding stage where that model is applied to classify the HEVC CU partitions. Experimentation shows a 2.26x speedup with a 3.6% RD loss.

An extension of [45, 49] introducing a fresh approach was published in [50]. This work involves the usage of the first frames of the sequence to compute the parameters to be provided to the transcoder. This way the knowledge is specific to the sequence. Two types of mapping are proposed: in the first solution a single H.264/AVC parameter is used to determine the HEVC partitioning using a dynamic thresholding, achieving 3.08x speedup with a 16.8% bitrate increase whilst, in the second, linear discriminant functions are used to map the H.264/AVC coding parameters to HEVC partitions. This last solution is called *Proposed Transcoder for Content Modelling using Linear Discriminant Functions* (PTCM-LDF).

Again, Peixoto *et al.* [51] proposed a further step on the H.264/AVC to HEVC transcoding field with a solution focused on two main modules: a CU classification module which performs H.264/AVC macroblock to HEVC CU mapping relying on a Machine Learning technique, and an early termination technique based on a statistical model of the HEVC RD cost. The process is also based on two stages (training and transcoding). This method obtains a 3.83x speedup, yielding a RD loss of 4%.

Another relevant work in the field is [52], which proposes a *Fast Quadtree Level Decision* (FQLD) algorithm which, in line with the previous two proposals, extracts data from the H.264/AVC decoder to improve the decision-making on CU splitting in HEVC making use of Machine Learning techniques, in this case, a Naïve-Bayes probabilistic classifier. Results show a 2.28x speedup with an increment of 4.8% on BD rate.

3.4.1 AFQLD algorithm

Díaz-Honrubia *et al.* refined in [9] the algorithm they previously proposed in [52], giving place to the Adaptive Fast Quadtree Level Decision algorithm (AFQLD). The AFQLD algorithm is the basis on which this project is being developed, so its functioning is going to be explained in more detail.

As many of the previous algorithms show, data mining techniques can be very helpful to optimize the time needed by the HEVC encoding process when performing a transcoding. AFQLD uses probabilistic models able to provide more flexible classifiers which can adapt to the particularities of each scene.

The objective of these models is to decide whether a CU has to be split or not. If the splitting is performed, we descend one level in the quadtree. Otherwise, the current level is chosen as the maximum allowed depth. This decision function is designed by following a KDD process. It can be translated to a supervised classification problem with a target variable that can take the value C_S (split), or C_N (do not split) based on a set of features taken from the H.264/AVC stream and the HEVC encoding process.

The decision can be carried out in a different way depending on the type of frame being processed and the depth of the quadtree, so several classifiers are needed for this task. Since achieving a 100% precision rate is not realistic, some PU computations are performed preventively. Thus, if the algorithm chooses to split, only the skip and 2Nx2N PUs are checked at levels 0 and 1, while all PUs are checked at level 2 of the quadtree. If the algorithm chooses C_N, all the PUs are evaluated and the algorithm for the current CTU finalizes. This way, if some of the RD costs calculated for higher levels is better than the best RD cost for the final level, the quadtree can return to the best one. The functioning of the algorithm is schematized in Figure 3.18.

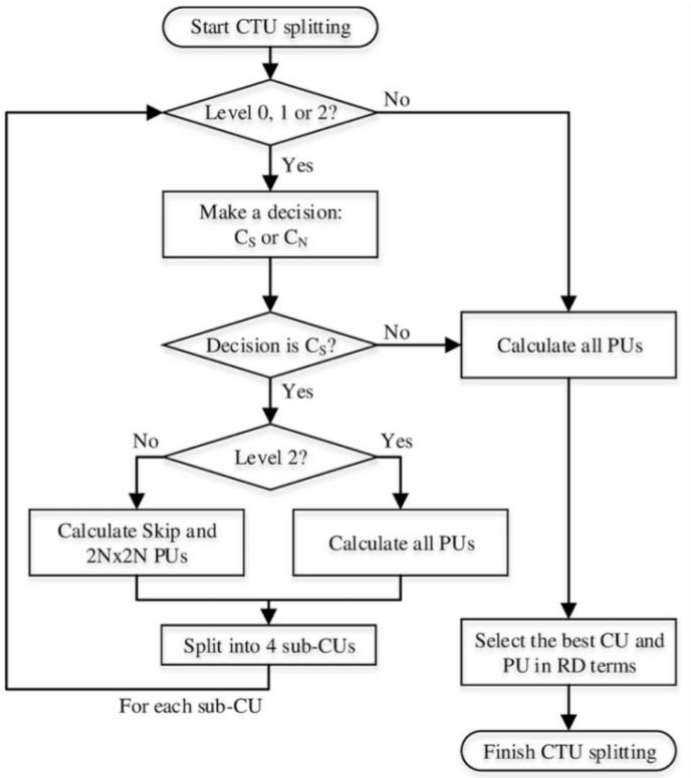

Figure 3.18. Diagram of the AFQLD algorithm [9].

Classifiers are built for levels 0 and 1. At level 2, CU size is 16x16 pixels, which corresponds to the macroblock size in H.264/AVC, so the algorithm just mimics it (it does not introduce any novelty in that case). This division in levels was considered since CUs from upper levels tend to split more frequently than those in the lower levels.

Therefore, eight different datasets, one for each combination of CU depth level (0 or 1) and residual energy level (1, 2, 3 or 4), are generated from a varied range of standardized sequences and then used to train the models using a RA configuration. Thus, each frame in RA configuration can be identified with a different energy level depending on its hierarchical layer (frames in higher layers tend to use bigger CU sizes, that is, they tend to choose not to split more frequently). One classifier is used for each of the combinations.

Some examples of features provided to the classifier are the number of bits used to encode all the MBs for the current CU after applying CABAC, the sum of all motion vectors contained in the frame, or the Lagrangian cost of choosing skip.

However, incorrectly deciding not to split should be costlier than the opposite kind of error because, if we decide to split, the speed is decreased but the quality is preserved. AFQLD adjusts the costs of making every decision dynamically, depending on the sequence. These costs are learned at the beginning of the sequence, making the HEVC encoder decide the partitioning for a maximum of one GOP, which is then compared with the predicted

partitioning. Since only one GOP can be used for learning the costs, making the extra expense negligible in comparison with the benefit obtained from the cost sensitive evaluation.

This algorithm achieves a 2.31x speedup (or, equivalently, a time reduction of 56.7%) with a BD-rate penalty of 3.4%. Also, it can be partially implemented, only at the first depth level, achieving a 26.7% time saving with a negligible BD rate increment of 0.8%.

CHAPTER 4. PROPOSED TRANSCODER

The main objective of this project is to research and evaluate the application of different Machine Learning algorithms in the AFQLD algorithm, apart from the proposed Naïve-Bayes. Therefore, in this chapter, the diverse Machine Learning algorithms reviewed in 2.1.2 are tested and compared in order to identify the most beneficial one. Later, the chosen algorithm will be implemented into AFQLD. More specifically, this work focuses on improving the *Make a decision* step seen in Figure 3.18 for 64x64 and 32x32 CUs (the decision mechanism of the 16x16 CUs is still kept as a copy from the H.264/AVC decisions).

4.1 Machine Learning algorithms testing

Tests have been carried out for different combinations of data preprocessing techniques and algorithms. This has been the most time-consuming phase of the project, given the large amount of time needed for training some of the most resource-intensive ones.

4.1.1 Comparison metrics

In order to be able to make a comparison among the algorithms, some metrics are necessary. The precision rate, the ROC curve and the F-measure are some of the most commonly used metrics in the field of ML.

The accuracy measures the precision of a classifier in terms of the percentage of right guesses. However, this is not always a good metric by itself, since a high accuracy does not necessarily mean that the classifier is actually good, because it can be overfitting the training data. Usually, extremely high accuracies tend to be a bad indicator.

The *Receiving Operating Characteristic* (ROC) was developed in World War II to analyze noisy signals in order to characterize the relation between hits and false alarms. Now it is considered a powerful metric for measuring the goodness of ML models. It represents the true positive rate and the false positive rate as axes y and x of a graph, respectively. The true positive rate is expressed as $TPR = \frac{TP}{TP+FN}$, where TP are the true positives (elements correctly predicted to belong to a given class) and FN are the false negatives (samples belonging to a given class but predicted not to be members of said class). The false positive rate is formulated as $FPR = \frac{FP}{FP+TN}$, where FP are the false positives (samples not belonging to a given class but predicted to belong to it) and TN are the true negatives (elements correctly predicted not to belong to a given class). These values are usually represented in a *confusion matrix*, as shown in Table 4.1. A curve is traced based on those two variables, and the area below is measured. The bigger the area (i.e. the closest to 1), the better the model. Figure 4.1 shows an example of a ROC curve.

↓ *Actual class / Predicted class* →	**Positive**	**Negative**
Positive	**TP**	FN
Negative	FP	**TN**

Table 4.1. Confusion matrix.

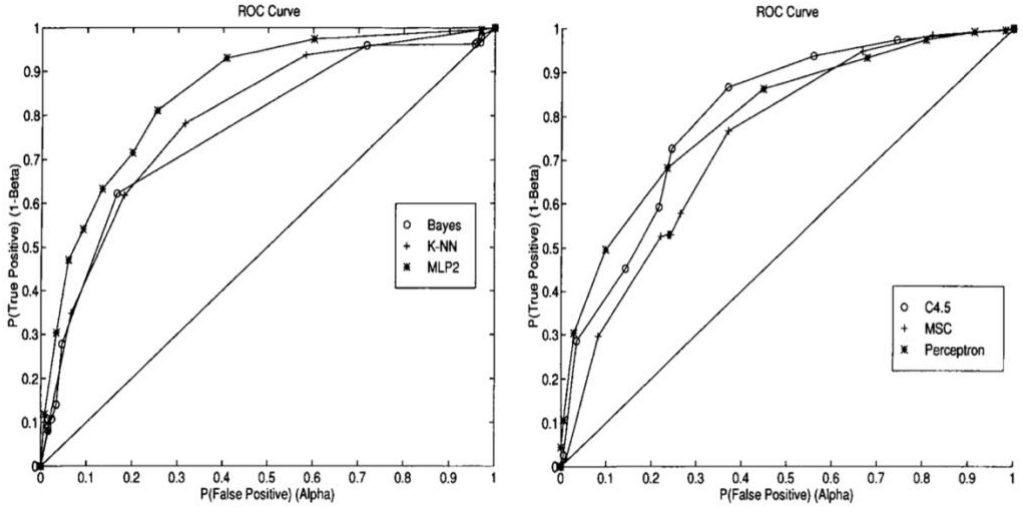

Figure 4.1. Example of ROC curves for different algorithms over a particular dataset. The diagonal corresponds to a random guessing [53].

The F-measure is written as $FM = 2\frac{P \cdot R}{P+R}$, that is, the harmonic mean of the precision, $P = \frac{TP}{TP+FP}$, which is the percentage of samples classified as a given class which actually belong to that class, and $R = \frac{TP}{TP+FN}$, which is the proportion of samples which actually belong to a class and were correctly predicted to belong to said class (refer to Table 4.1 for

a better understanding). Of course, since big P and R values are desired, the higher the value obtained, the better the model.

4.1.2 Evaluating an algorithm

The most common technique for algorithm evaluation is cross-validation [54], in particular, k-fold cross-validation. This technique divides the training set in k (usually $k = 5, 10, ...$) subsets of the same size. Then, the algorithm is trained with $k - 1$ subsets and is tested with the remaining one, as shown in Figure 4.2. The process is repeated k times, each time with a different test subset. Finally, the average error is calculated and used as a measure of the accuracy of the learning algorithm. This is the technique used for this work's tests.

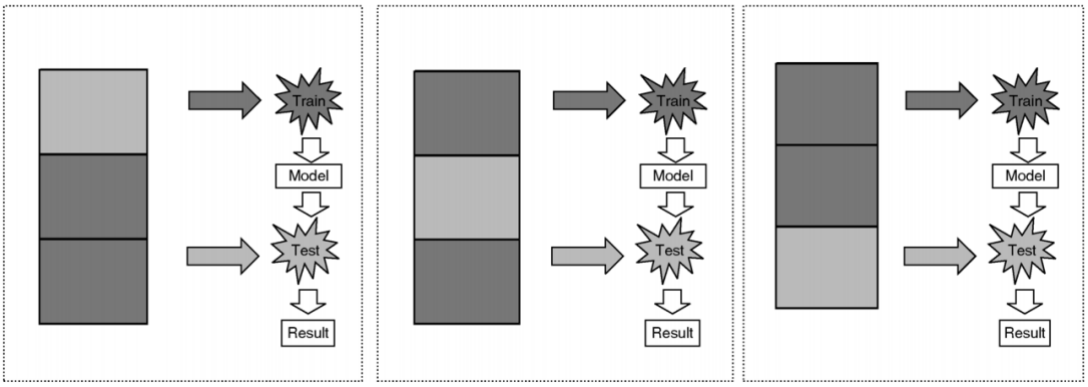

Figure 4.2. Procedure of three-fold cross-validation [54].

4.1.3 Test Results

The results obtained from the execution of the different Machine Learning algorithms are presented in this section in terms of accuracy, F-measure, and area under the ROC curve. The main the idea behind the selection of these algorithms is to perform an extensive test comprising a varied catalog of algorithms and data preprocessing methods so that the main categories of Machine Learning algorithms are covered. A geometric mean of the three metrics is also calculated and will be the main criterium followed for comparing the algorithms. A geometric mean is especially useful when comparing different values with different numeric ranges, which is the case of the three units used for these tests. It can give a meaningful average in contrast with the arithmetic mean which would favor greatly the unit with a bigger value range and lower the significance of the other values.

An extensive set of tests has been executed. The tested algorithms were Naïve-Bayes (used by the default implementation of AFQLD), C4.5, SVM with linear, radial and second-

degree polynomial kernels, Random Forest, KNN and Multilayer Perceptron. All of them were tested with different preprocessing combinations: no preprocessing, discretization, correlation based[3] feature selection (FS), feature selection with a wrapper method, discretization combined with correlation-based feature selection and discretization combined with wrapper feature selection. The discretization algorithm used is the entropy-based supervised version proposed by Fayyad and Irani [55], which generates intervals of different widths and selects the number of intervals of each feature automatically. The feature selection with the wrapper method is performed using the tested algorithm as the subrogate classifier and with a greedy forward strategy (starting with the empty set and iteratively incorporating the best feature in each step), this way, the knowledge transferred between the feature selection phase and the model building phase is maximized. Finally, the evaluation of each algorithm was made using a 10-fold cross-validation, in order to obtain results with enough representativeness.

Every combination was executed over each of the eight datasets $DS_{i,j}, i \in \{0,1\}, j \in \{1,2,3,4\}$, where i indicates the level in the quadtree and j indicates the residual energy level, following the division made in the original AFQLD algorithm and described in 3.4.1. The tests were executed using the Weka suite [56], a well-known open-source machine learning framework that provides many utilities for processing data and training models. Weka has not only a graphical interface, which was used in the preliminary steps of the experimentation, but also a Java API. This last option was chosen to automate the testing process, with and approach similar to the pseudocode shown in Figure 4.3. Note that FS with wrapper is done individually for each classifier, while the rest preprocessing methods are applied just once for each dataset.

However, the process of model construction of some of the algorithms was extremely time consuming for the most complicated datasets (in $DS_{i,j}$, the bigger i and j, the more difficult to make a prediction) so some of the tests had to be interrupted before finishing, since the times were completely unapproachable, in the order of weeks for each configuration. Nevertheless, the tests were performed in an incremental way, so new configurations were considered depending on the results obtained with previous ones (for example, if polynomial SVM performs badly with FS, the configurations not performing this kind of preprocessing are tested before) in an effort for optimizing the quality of the information gathered under the time constraints of the project.

Even so, the total time spent in the testing phase was 557 hours, which would correspond to almost 24 days of continued execution. This statistic does not take into account the tests that had to be stopped due to their enormous processing times. Note that these times do not correspond to the time that the model takes to classify each sample when working

[3] Based on the assumption that good feature sets contain features highly correlated with the classification, but not among them.

online[4] but to the time taken by the algorithm to build a model in an offline[5] setting, which would be then implemented into the transcoder. For instance, it is easy to see from the explanation in 2.1.2.2 that SVM takes a long time to build the model, since it has to perform quite demanding computations, but is otherwise very fast in the subsequent testing phase.

```
0:  for all datasets do
1:      for all preprocessingLists do
2:          for all preprocessings in preprocessingList do
3:              if preprocessing is not FS with wrapper then
4:                  Apply preprocessing to dataset
5:              end if
6:          end for
7:          for all classifiers do
8:              if preprocessing is FS with wrapper then
8:                  Apply preprocessing to dataset using classifier
9:              end if
10:             Build classifier model with dataset
11:             10-fold cross-validate model with dataset
12:             Save results
13:         end for
14:     end for
15: end for
```

Figure 4.3. Pseudocode of the script developed for Weka tests automation.

An exhaustive view of the results yielded by those tests is shown in Tables 4.2 – 4.9. The parameters used in the Weka suite are the defaults in all cases except in the feature selection with wrapper, where a forward *greedy stepwise* search method is employed. *FS* stands for correlation-based feature selection, while *FS (wrapper)* is the version using the wrapper method. The rows corresponding to the SVM algorithm indicate the kernel used between parentheses. The headers %, F, ROC and GM stand for accuracy, F-measure, area under the ROC curve, and the geometric mean of the three values. The best result obtained for each dataset is highlighted in bold.

[4] Performed at runtime, while the transcoder is running.
[5] Performed prior to the transcoding process.

$DS_{0,1}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	88,31	0,91	0,96	4,26	91,81	0,93	0,98	4,38
C4.5	96,94	0,97	0,86	4,33	97,21	0,97	0,91	4,42
SMO (Linear)	96,61	0,97	0,81	4,23	97,44	0,97	0,85	4,32
SMO (Poly)	96,83	0,97	0,80	4,21	96,28	0,96	0,82	4,24
SMO (RBF)	-	-	-	-	97,31	0,97	0,82	4,27
Random Forest	97,30	0,97	0,99	**4,54**	97,33	0,97	0,98	4,53
KNN	95,93	0,96	0,81	4,21	96,64	0,97	0,93	4,42
Multilayer Perceptron	96,90	0,97	0,98	4,51	96,82	0,97	0,98	4,51
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	90,29	0,92	0,97	4,32	-	-	-	-
C4.5	96,89	0,97	0,94	4,45	97,24	0,97	0,94	4,46
SMO (Linear)	96,37	0,96	0,76	4,13	-	-	-	-
SMO (Poly)	96,33	0,96	0,73	4,07	96,85	0,97	0,80	4,21
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	97,31	0,97	0,98	4,53	97,47	0,97	0,99	**4,54**
KNN	95,78	0,96	0,80	4,19	96,48	0,96	0,93	4,41
Multilayer Perceptron	96,46	0,96	0,98	4,50	96,66	0,97	0,95	4,46
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	94,54	0,95	0,98	4,46	96,71	0,97	0,97	4,50
C4.5	97,06	0,97	0,96	4,48	97,51	0,97	0,97	4,52
SMO (Linear)	97,11	0,97	0,80	4,23	97,29	0,97	0,84	4,30
SMO (Poly)	97,10	0,97	0,84	4,29	97,54	0,97	0,86	4,34
SMO (RBF)	97,02	0,97	0,80	4,21	96,97	0,97	0,80	4,21
Random Forest	96,91	0,97	0,98	4,51	97,38	0,97	0,97	4,51
KNN	96,91	0,97	0,97	4,50	97,30	0,97	0,97	4,51
Multilayer Perceptron	96,80	0,97	0,98	4,50	97,20	0,97	0,98	4,52

Table 4.2. Results of the tested ML algorithms for the dataset $DS_{0,1}$.

$DS_{0,2}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	70,69	0,75	0,92	3,65	80,04	0,83	0,94	3,96
C4.5	91,68	0,92	0,90	4,22	92,21	0,92	0,94	4,30
SMO (Linear)	91,26	0,91	0,81	4,07	92,29	0,92	0,80	4,08
SMO (Poly)	91,66	0,91	0,80	4,05	90,19	0,90	0,82	4,06
SMO (RBF)	90,57	0,90	0,78	3,99	91,96	0,91	0,78	4,03
Random Forest	92,46	0,92	0,96	**4,35**	92,05	0,92	0,95	4,32
KNN	89,53	0,90	0,80	4,01	90,98	0,91	0,88	4,17
Multilayer Perceptron	91,23	0,91	0,95	4,29	91,70	0,91	0,94	4,29
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	73,20	0,77	0,93	3,74	89,73	0,90	0,92	4,21
C4.5	91,69	0,91	0,93	4,28	92,29	0,92	0,93	4,30
SMO (Linear)	90,86	0,91	0,81	4,06	91,22	0,91	0,80	4,06
SMO (Poly)	91,26	0,91	0,79	4,02	91,61	0,91	0,81	4,08
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	91,62	0,91	0,96	4,31	91,36	0,91	0,93	4,26
KNN	88,69	0,89	0,79	3,95	91,35	0,91	0,93	4,26
Multilayer Perceptron	91,22	0,91	0,94	4,28	91,38	0,91	0,94	4,27
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	85,96	0,87	0,95	4,14	91,83	0,92	0,95	4,30
C4.5	91,99	0,92	0,94	4,30	92,16	0,92	0,92	4,27
SMO (Linear)	91,31	0,91	0,78	4,01	92,06	0,92	0,80	4,07
SMO (Poly)	91,86	0,92	0,81	4,08	-	-	-	-
SMO (RBF)	91,49	0,90	0,75	3,95	91,62	0,91	0,76	3,97
Random Forest	91,67	0,91	0,94	4,29	92,23	0,92	0,95	4,32
KNN	91,64	0,91	0,93	4,27	92,33	0,92	0,95	4,32
Multilayer Perceptron	91,18	0,91	0,95	4,28	92,27	0,92	0,95	4,32

Table 4.3. Results of the tested ML algorithms for the dataset $DS_{0,2}$.

$DS_{0,3}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	71,46	0,73	0,91	3,63	80,86	0,82	0,94	3,97
C4.5	90,69	0,91	0,92	4,23	90,88	0,91	0,94	4,27
SMO (Linear)	89,68	0,90	0,87	4,12	91,16	0,91	0,86	4,15
SMO (Poly)	90,39	0,90	0,86	4,13	89,70	0,90	0,87	4,12
SMO (RBF)	89,03	0,89	0,86	4,09	90,90	0,91	0,85	4,13
Random Forest	91,63	0,92	0,97	**4,33**	90,88	0,91	0,96	4,30
KNN	88,84	0,89	0,85	4,07	89,85	0,90	0,92	4,20
Multilayer Perceptron	90,35	0,90	0,96	4,28	90,61	0,91	0,95	4,28
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	73,91	0,76	0,94	3,75	87,74	0,88	0,94	4,17
C4.5	90,95	0,91	0,95	4,28	91,00	0,91	0,95	4,28
SMO (Linear)	89,35	0,89	0,87	4,11	90,09	0,90	0,85	4,11
SMO (Poly)	90,08	0,90	0,85	4,10	90,19	0,90	0,85	4,11
SMO (RBF)	87,76	0,88	0,86	4,06	-	-	-	-
Random Forest	90,41	0,90	0,96	4,28	90,62	0,90	0,95	4,28
KNN	87,97	0,88	0,84	4,02	-	-	-	-
Multilayer Perceptron	90,05	0,90	0,95	4,26	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	84,84	0,86	0,94	4,09	90,39	0,90	0,95	4,26
C4.5	90,77	0,91	0,95	4,27	91,11	0,91	0,94	4,27
SMO (Linear)	90,51	0,90	0,86	4,12	91,07	0,91	0,86	4,15
SMO (Poly)	90,93	0,91	0,87	4,15	-	-	-	-
SMO (RBF)	90,61	0,90	0,85	4,12	-	-	-	-
Random Forest	90,69	0,91	0,95	4,28	91,15	0,91	0,96	4,30
KNN	90,56	0,90	0,94	4,26	91,20	0,91	0,96	4,30
Multilayer Perceptron	90,40	0,90	0,96	4,28	90,81	0,91	0,96	4,29

Table 4.4. Results of the tested ML algorithms for the dataset $DS_{0,3}$.

$DS_{0,4}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	75,61	0,75	0,87	3,66	80,31	0,80	0,92	3,91
C4.5	91,18	0,91	0,94	4,28	90,63	0,91	0,95	4,28
SMO (Linear)	89,29	0,89	0,90	4,15	90,94	0,91	0,91	4,22
SMO (Poly)	90,35	0,90	0,91	4,20	89,67	0,90	0,90	4,16
SMO (RBF)	88,03	0,88	0,88	4,09	90,69	0,91	0,91	4,21
Random Forest	92,11	0,92	0,98	4,36	90,89	0,91	0,97	4,31
KNN	89,05	0,89	0,89	4,14	89,98	0,90	0,94	4,24
Multilayer Perceptron	90,78	0,91	0,97	4,31	90,94	0,91	0,96	4,30
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	79,41	0,79	0,95	3,90	-	-	-	-
C4.5	90,64	0,91	0,94	4,27	-	-	-	-
SMO (Linear)	89,08	0,89	0,89	4,13	-	-	-	-
SMO (Poly)	88,94	0,89	0,89	4,12	-	-	-	-
SMO (RBF)	88,01	0,88	0,88	4,09	-	-	-	-
Random Forest	88,38	0,88	0,95	4,20	92,33	0,92	0,98	**4,37**
KNN	87,37	0,87	0,88	4,06	-	-	-	-
Multilayer Perceptron	89,88	0,90	0,95	4,25	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	86,81	0,87	0,96	4,16	89,94	0,90	0,96	4,26
C4.5	90,43	0,90	0,95	4,27	91,19	0,91	0,96	4,31
SMO (Linear)	89,80	0,90	0,90	4,16	90,79	0,91	0,91	4,21
SMO (Poly)	90,49	0,90	0,90	4,20	-	-	-	-
SMO (RBF)	89,29	0,89	0,89	4,14	-	-	-	-
Random Forest	90,54	0,91	0,96	4,28	90,98	0,91	0,97	4,31
KNN	90,56	0,91	0,96	4,28	90,79	0,91	0,96	4,30
Multilayer Perceptron	90,39	0,90	0,96	4,28	90,77	0,91	0,97	4,31

Table 4.5. Results of the tested ML algorithms for the dataset $DS_{0,4}$.

$DS_{1,1}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	55,66	0,60	0,74	2,92	71,38	0,74	0,76	3,42
C4.5	80,90	0,79	0,69	3,54	82,73	0,79	0,74	3,64
SMO (Linear)	-	-	-	-	82,10	0,78	0,55	3,28
SMO (Poly)	-	-	-	-	80,60	0,79	0,60	3,37
SMO (RBF)	-	-	-	-	82,34	0,75	0,51	3,17
Random Forest	83,12	0,80	0,80	**3,76**	82,32	0,80	0,76	3,68
KNN	76,51	0,77	0,60	3,28	79,98	0,78	0,68	3,49
Multilayer Perceptron	82,39	0,79	0,78	3,71	78,54	0,78	0,73	3,54
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	58,58	0,63	0,75	3,03	-	-	-	-
C4.5	82,06	0,79	0,71	3,59	-	-	-	-
SMO (Linear)	-	-	-	-	-	-	-	-
SMO (Poly)	-	-	-	-	-	-	-	-
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	82,61	0,80	0,77	3,70	82,45	0,77	0,73	3,59
KNN	76,43	0,76	0,60	3,27	-	-	-	-
Multilayer Perceptron	82,43	0,78	0,78	3,69	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	72,53	0,75	0,77	3,47	82,48	0,78	0,74	3,63
C4.5	83,01	0,79	0,76	3,68	-	-	-	-
SMO (Linear)	82,11	0,78	0,55	3,28	82,11	0,78	0,55	3,28
SMO (Poly)	82,98	0,79	0,57	3,35	-	-	-	-
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	80,79	0,79	0,74	3,60	83,21	0,79	0,78	3,71
KNN	80,86	0,79	0,70	3,54	83,19	0,79	0,78	3,71
Multilayer Perceptron	-	-	-	-	82,63	0,79	0,75	3,65

Table 4.6. Results of the tested ML algorithms for the dataset $DS_{1,1}$.

$DS_{1,2}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	54,38	0,53	0,68	2,70	63,43	0,64	0,70	3,06
C4.5	67,42	0,67	0,66	3,11	68,48	0,68	0,70	3,20
SMO (Linear)	67,54	0,64	0,59	2,95	65,93	0,62	0,57	2,87
SMO (Poly)	67,31	0,65	0,60	2,97	66,86	0,67	0,64	3,06
SMO (RBF)	64,19	0,51	0,50	2,54	68,59	0,67	0,63	3,07
Random Forest	69,74	0,69	0,75	**3,30**	67,75	0,67	0,71	3,18
KNN	62,94	0,63	0,60	2,87	65,79	0,65	0,66	3,06
Multilayer Perceptron	68,21	0,68	0,73	3,23	65,36	0,65	0,68	3,07
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	55,78	0,55	0,69	2,76	-	-	-	-
C4.5	67,56	0,67	0,70	3,17	-	-	-	-
SMO (Linear)	-	-	-	-	-	-	-	-
SMO (Poly)	66,38	0,64	0,59	2,91	-	-	-	-
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	66,65	0,66	0,69	3,12	65,93	0,65	0,68	3,08
KNN	61,03	0,61	0,59	2,79	-	-	-	-
Multilayer Perceptron	66,86	0,66	0,71	3,15	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	63,74	0,64	0,71	3,08	67,35	0,67	0,70	3,15
C4.5	67,84	0,67	0,71	3,19	68,84	0,68	0,72	3,23
SMO (Linear)	-	-	-	-	66,01	0,62	0,57	2,86
SMO (Poly)	67,73	0,67	0,62	3,04	-	-	-	-
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	65,88	0,66	0,69	3,10	68,11	0,67	0,72	3,20
KNN	66,08	0,66	0,67	3,09	68,21	0,68	0,71	3,21
Multilayer Perceptron	-	-	-	-	67,39	0,66	0,70	3,15

Table 4.7. Results of the tested ML algorithms for the dataset $DS_{1,2}$.

$DS_{1,3}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	54,94	0,52	0,66	2,66	61,46	0,61	0,68	2,96
C4.5	66,51	0,67	0,67	3,10	66,51	0,66	0,69	3,12
SMO (Linear)	65,03	0,64	0,63	2,97	65,92	0,66	0,64	3,03
SMO (Poly)	65,18	0,65	0,63	2,99	64,93	0,65	0,64	3,01
SMO (RBF)	64,31	0,64	0,62	2,94	66,72	0,66	0,65	3,06
Random Forest	68,88	0,68	0,75	**3,29**	65,11	0,65	0,69	3,08
KNN	62,46	0,63	0,62	2,89	63,94	0,64	0,66	2,99
Multilayer Perceptron	66,53	0,66	0,71	3,16	63,69	0,64	0,67	3,01
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	55,41	0,52	0,67	2,69	-	-	-	-
C4.5	65,89	0,66	0,69	3,10	-	-	-	-
SMO (Linear)	62,97	0,63	0,62	2,91	-	-	-	-
SMO (Poly)	62,05	0,59	0,58	2,77	-	-	-	-
SMO (RBF)	63,06	0,63	0,62	2,90	-	-	-	-
Random Forest	64,29	0,64	0,68	3,03	63,64	0,63	0,67	3,00
KNN	60,01	0,60	0,59	2,78	-	-	-	-
Multilayer Perceptron	64,56	0,64	0,69	3,06	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	61,29	0,61	0,69	2,95	64,84	0,65	0,69	3,07
C4.5	65,83	0,65	0,70	3,10	67,57	0,67	0,70	3,17
SMO (Linear)	-	-	-	-	63,23	0,61	0,59	2,84
SMO (Poly)	65,58	0,65	0,64	3,01	-	-	-	-
SMO (RBF)	-	-	-	-	63,23	0,61	0,59	2,84
Random Forest	63,58	0,63	0,68	3,01	67,01	0,67	0,71	3,16
KNN	63,40	0,63	0,66	2,98	67,14	0,67	0,71	3,17
Multilayer Perceptron	-	-	-	-	66,51	0,67	0,70	3,15

Table 4.8. Results of the tested ML algorithms for the dataset $DS_{1,3}$.

$DS_{1,4}$	%	F	ROC	GM	%	F	ROC	GM
	None				**Discretization**			
Naïve-Bayes	55,03	0,48	0,57	2,46	57,72	0,56	0,64	2,74
C4.5	65,21	0,65	0,67	3,05	63,73	0,64	0,66	2,99
SMO (Linear)	57,44	0,53	0,57	2,60	61,04	0,61	0,61	2,83
SMO (Poly)	61,16	0,61	0,61	2,83	63,88	0,63	0,64	2,96
SMO (RBF)	58,38	0,56	0,58	2,67	62,76	0,62	0,63	2,90
Random Forest	67,65	0,68	0,74	3,24	63,41	0,63	0,69	3,02
KNN	62,44	0,62	0,62	2,90	63,08	0,63	0,66	2,97
Multilayer Perceptron	62,62	0,63	0,67	2,97	62,73	0,63	0,67	2,97
	Feature Selection				**Feature Selection (wrapper)**			
Naïve-Bayes	56,01	0,50	0,57	2,52	-	-	-	-
C4.5	64,61	0,64	0,68	3,04	-	-	-	-
SMO (Linear)	57,22	0,53	0,57	2,59	-	-	-	-
SMO (Poly)	56,74	0,52	0,56	2,55	-	-	-	-
SMO (RBF)	56,56	0,52	0,56	2,55	-	-	-	-
Random Forest	63,18	0,63	0,68	3,00	67,89	0,68	0,74	**3,25**
KNN	58,99	0,59	0,59	2,74	-	-	-	-
Multilayer Perceptron	61,28	0,61	0,65	2,90	-	-	-	-
	Discretization + FS				**Discretization + FS (wrapper)**			
Naïve-Bayes	57,40	0,54	0,64	2,71	62,34	0,62	0,65	2,94
C4.5	61,98	0,61	0,64	2,90	64,57	0,64	0,67	3,04
SMO (Linear)	-	-	-	-	59,84	0,59	0,60	2,77
SMO (Poly)	61,39	0,61	0,61	2,83	-	-	-	-
SMO (RBF)	-	-	-	-	-	-	-	-
Random Forest	61,75	0,61	0,64	2,89	64,08	0,64	0,69	3,04
KNN	61,83	0,61	0,64	2,89	64,16	0,64	0,69	3,04
Multilayer Perceptron	-	-	-	-	63,63	0,64	0,69	3,03

Table 4.9. Results of the tested ML algorithms for the dataset $DS_{1,4}$.

As shown in the tables, following the geometric mean of the accuracy, F-measure and ROC, the algorithm with, in general, the most remarkable performance over the datasets is Random Forest with no preprocessing, surpassing the Naïve-Bayes classifier used in the original AFQLD (Naïve-Bayes with discretization and wrapper FS) in all cases but $DS_{0,4}$ and $DS_{1,4}$, where its performance is almost on pair with the winner solution. KNN with discretization and FS with wrapper and some versions of SVM also perform well in a few datasets but, unlike Random Forest, this performance only holds for specific cases.

Random Forest also works satisfactorily after performing a feature selection with a wrapper method but is surpassed in terms of geometrical mean by the version without preprocessing in most of the datasets, as the tables reflect. Therefore, the chosen algorithm to be implemented into the transcoder is the latter.

Of course, the results obtained in practice may differ from the estimations made in this test process, since the fitness of the models obtained depends on the quality of the training datasets, and it is difficult to generate datasets which represent all the different variants one can encounter in a video sequence, but they can help us identify the most interesting one *a priori*.

4.2 Implementation of the classifier

Having selected the algorithm, it is time to implement it in the HEVC encoder inside the AFQLD algorithm. First, it is important to understand its structure and how its implementation connects with the HEVC reference coding software version 16.2 (HM 16.2) [47].

AFQLD is implemented in C. It takes the information extracted from the H.264/AVC bitstream as input and processes every CU in each frame, one by one. If the current level is 0 or 1, the input information related with each frame and that proper of the CU itself is passed to the classifier which takes the decision of whether to split or not. These decisions about partitioning are stored in a specific file, where the depth of each CU is indicated slice by slice. *0* indicates a 64x64 CU and bigger numbers represent an extra partitioning level. The CUs are traversed in a *Z* order (left to right, top to bottom). For instance, *1233332211* refers to the splitting shown in Figure 4.4. A real (reduced) fragment of one of these files can be seen in Figure 4.5.

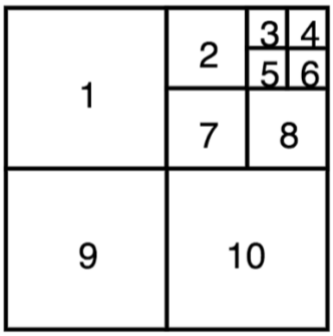

Figure 4.4. Example of a CU partitioning. The numbers indicate the order in which the CU is traversed.

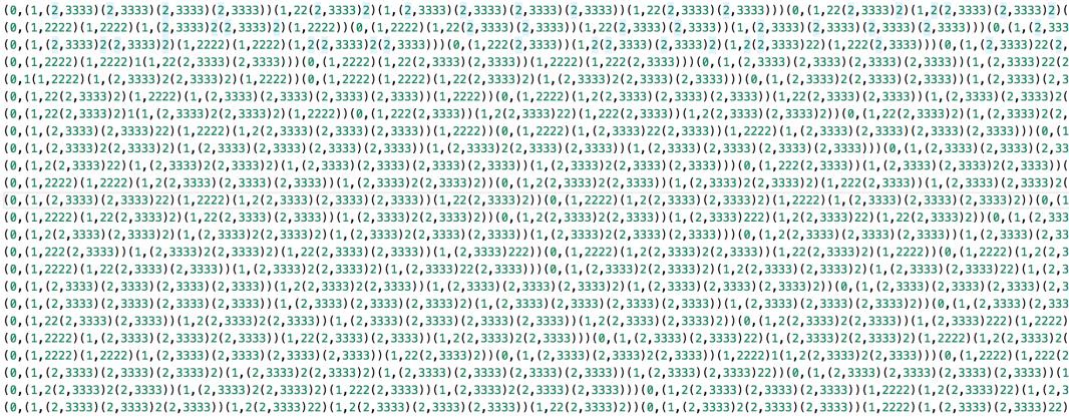

Figure 4.5. Fragment of a configuration file with the CU splitting information.

Besides the CU partitioning file, another file is generated including information about the PU splitting. Remember that more or less PU splittings are calculated depending on the decision previously taken by the classifier (refer to Figure 3.18). The information is stored as digits ranging from 0 to 7 and corresponding to the partition types shown in Table 4.10. A letter or any other character different from those mentioned in the table indicate that all the PUs have to be calculated by the encoder. Note that skip is always checked independently of the partition type defined.

Num.	**Partition type in HM**	**Description**
0	2Nx2N	Symmetric motion partition, 2Nx2N
1	2NxN	Symmetric motion partition, 2NxN
2	Nx2N	Symmetric motion partition, Nx2N
3	NxN	Symmetric motion partition, NxN
4	2NxnU	Asymmetric motion partition, 2Nx(N/2) + 2Nx(3N/2)
5	2NxnD	Asymmetric motion partition, 2Nx(3N/2) + 2Nx(N/2)
6	nLx2N	Asymmetric motion partition, (N/2)x2N + (3N/2)x2N
7	nRx2N	Asymmetric motion partition, (3N/2)x2N + (N/2)x2N

Table 4.10. Correspondence between numbers in the PU splitting configuration file and PU partition types.

Both files will be then fed to a modified version of HM 16.2 which performs the partitioning based on the information in them.

In the case of the Random Forest models, given the enormous size and higher implementation complexity compared to those from Naïve-Bayes, and in order to optimize the implementation times, a manual implementation into the C program was not viable. Instead, Weka was used for loading previously generated model files and classifying each instance. Since it does not provide a C API, Java had to be used for this purpose, taking advantage of the *Java Native Interface* (JNI) [57] for intercommunication between both

languages. This way, for each CU, the C code calls a Java function that receives the features of the given CU and returns a decision (to split or not).

JNI is a programming framework that allows a program written in Java to interact with other programs written in other languages, like C or C++. Internally, it runs a Java Virtual Machine (JVM) in which the Java code is executed. This technology is mainly used in situations where specific libraries or platform-specific features are not available in Java or, the other way around, when a Java library needs to be used in software written in a different language.

So, when the modified AFQLD algorithm is run, it starts by stablishing the connection to the Java code, running the JVM and calling a setup function which prepares the data structures to be used in the Java code for storing the input parameters and making the classification using the Weka API and loads the models that will be used for this task, which were generated and persisted prior to the execution of the algorithm. Then, at the time of classifying each CU, the classification method is called from the C code, the data structures are filled with the values of each feature and a call to the Weka API is made from the Java code, returning the result of the classification of the instance as an integer (1: split, or 0: do not split) which is then sent back to the C program, which continues the processing and writes the result in the output files mentioned previously in this section.

CHAPTER 5. PERFORMANCE EVALUATION

Having tested several Machine Learning algorithms, generated the models, and implemented them into the HEVC encoder, its performance on real video sequences can be tested and measured with different metrics. In this chapter, the most common metrics for measuring video and encoder quality are presented, followed by the results and evaluation of the proposed algorithm and the simulation setup utilized.

5.1 Metrics

Several metrics are used for measuring and comparing the performance of different video coding processes and the quality of the resulting video. They can be objective, like the BD-rate or the PSNR, or subjective.

5.1.1 Subjective video quality evaluation

An obvious way to measure the quality of a video sequence is the perception of a person. This technique uses human subjects to evaluate the visual quality of a video be means of questionnaires. It is useful for measuring the perceptual video quality (which cannot be really measured by objective mathematical techniques) but has the downside of the subjectivity of the results obtained, which can vary among subjects and depending on conditions such as the ambient illumination, the viewing distance or the device in which the video is being displayed.

In order to perform this kind of tests, a minimum of 24 voters is required by the ITU recommendations. These voters have to be non-expert and they should be shown a carefully selected set of sequences that best represent the particular application being evaluated.

Therefore, these tests are quite complicated to carry out and they are usually only used when the objective quality is known to be worse in the tested encoder but the subjective quality does not vary. Nevertheless, all the official papers and recommendations from the standardization committee use objective quality metrics.

5.1.2 Peak Signal-to-Noise Ratio

The term Peak Signal-to-Noise Ratio, or PSNR, refers to the maximum power of a signal and the power of a distorting noise that affects the quality of its representation after compression, processing or transmission. It is usually expressed in decibels (dB) using a logarithmic scale. This metric does not correlate perfectly with the perceived visual quality but is a valid quality measure as long as the video content and coding standard used are the same.

It is defined via the Mean Squared Error (MSE). Given a $m \cdot n$ monochrome image I and its noisy representation K, we can define MSE as $MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(I(i,j) - K(i,j))^2$ and then define the PSNR as $PSNR = 10log_{10}(\frac{MAX_I^2}{MSE})$, where MAX_I is the maximum possible pixel value of the image (for instance, 255 for 8-bit images). For color images, the MSE is the sum over all squared value differences divided by image and by three.

PSNR has been traditionally used in analog audio-visual systems and, although it has some limitations on digital video, it keeps being used due to its low complexity and easy measurability, particularly to measure the differences in video quality of a sequence for a specified bitrate between different coding solutions. The biggest the PSNR, the higher the quality.

5.1.3 BD-rate

The BD-rate, or Bjøntegaard Distortion-rate [44], measures the increment in bitrate for maintaining the same objective quality. It is the weighted average of the Y, U and V components, taking into account that the luminance has four times the size of the chrominances (the video format used is 4:2:0). It is represented as

$BD\text{-}rate = \frac{4\,BD\text{-}rate_Y + BD\text{-}rate_U + BD\text{-}rate_V}{6}$. It is commonly used for measuring the coding efficiency of an encoder with respect to a reference encoder.

It is based on the RD (Rate Distortion) curve (an example can be seen later for instance in Figure 5.1). The RD is a double metric that indicates what distortion (in terms of objective quality with respect to the original) is present in an encoded sequence for a given bitrate. The curves are usually elaborated with four points, corresponding to QP values of 22, 27, 32 and 37, as specified in the CTC. The BD-rate basically measures the area between the RD curves of two encoders.

5.1.4 Speedup

The speedup serves as a metric for expressing the time reduction of an encoder with respect to another. It is expressed as $Speedup = \frac{t_{anchor}}{t_{proposed}}$, where t_{anchor} is the time taken by the encoder used as reference and $t_{proposed}$ is the execution time of the proposed transcoder.

5.2 Results and analysis

As mentioned in 3.3.2, in order to homogenize comparisons between experiments, the JCT-VC defined the *Common Test Conditions* [42]. The evaluation of the transcoder has been made following the criteria detailed in that specification. The QP values used are 22, 27, 32 and 37 and the configuration is Random Access *main 10* (which, unlike *main*, allows asymmetric TU splitting).

The transcoder has been tested with the following sequences of the different classes defined in the CTC:

- Class A: *Traffic*. 150 frames at 30 FPS. 2560x1600 pixels.
- Class B: *ParkScene*. 240 frames at 24 FPS. 1920x1080 pixels.
- Class C: *PartyScene*. 500 frames at 50FPS. 832x420 pixels.
- Class C: *RaceHorsesC*. 300 frames at 30 FPS. 832x480 pixels.
- Class D: *BlowingBubbles*. 500 frames at 50 FPS. 416x240 pixels.
- Class D: *RaceHorses*. 300 frames at 30 FPS. 416x240 pixels.

For encoding and decoding the H.264/AVC sequences, the JM 18.4 software [58] has been used, while HM 16.2 [47] was used to encode the sequence with HEVC. The process followed consists on the following steps:

1. Encode the original YUV sequence in H.264 using JM 18.4.
2. Decode the H.264/AVC-encoded sequence, obtaining a YUV' file and the information needed for the classifier.
3. Encode the YUV' file in HEVC with the anchor transcoder.
4. Encode the YUV' file in HEVC with the AFQLD transcoder.
5. Encode the YUV' file in HEVC with the proposed transcoder.
6. Compare the anchor, AFQLD and proposed transcoders, obtaining the BD-rate and the speedup achieved.

The measurements have been made on a six-core Intel Core i7-3930K CPU running at 3.20 GHz. The results in terms of BD-rate and speedup are shown in Table 5.1.

Sequence	**AFQLD**		**Proposed transcoder**	
	BD-rate (%)	**Speedup**	**BD-rate (%)**	**Speedup**
Traffic (A)	3.0	2.71	3.4	3.20
ParkScene (B)	3.3	2.52	4.7	2.94
PartyScene (C)	1.3	1.96	4.5	2.30
RaceHorsesC (C)	1.3	1.46	5.8	1.98
BlowingBubbles (D)	1.4	1.96	3.4	2.21
RaceHorses (D)	1.3	1.66	5.0	1.79
Average	1.93	2.05	4.47	2.40

Table 5.1. Comparison in BD-rate and speedup between AFQLD and the proposed transcoder.

Although the table shows inferior results in terms of BD-rate for the simpler sequences when compared to AFQLD, it can be noticed that the proposed transcoder performs well in the class A sequence. In that case, the BD-rate is 3.4% (i.e. it requires 3.4% more bits to encode the sequence than the reference HEVC encoder) while AFQLD achieves a 3% but, however, the speedup is considerably higher. Therefore, this proposal looks promising for the most complex category of sequences, with resolutions beyond 1080p, where it can be useful for reducing the encoding times while increasing slightly the bitrate required. Given the high computational time needed for encoding the class-A sequences, this speedup is especially influential.

While in terms of content migration from H.264/AVC to HEVC this kind of sequence is not the most common (the majority of the legacy video has resolutions up to 1080p, and commonly not higher than 720p), the proposed transcoder could be used for reducing significantly the time taken to encode new sequences in HEVC by encoding them first in H.264/AVC, which requires much less computational resources and, thus, less CPU time, and then converting it to HEVC, obtaining as output a sequence which takes advantage of all the improvements from the HEVC standard while also exploiting the benefits of the previous standard in terms of coding velocity.

In the rest of the sequences, the proposed transcoder manages to increase the speedup at the cost of a more notable increase in BD-rate. Thus, it is not useful as an all-rounder solution for these kind of sequences, but (mainly for FullHD 1080p resolutions, as seen in the *ParkScene* sequence) rather as an alternative for specific situations in which the velocity of the encoding process is of foremost importance.

The following graphs (Figures 5.1 – 5.6) show the results of the simulations in terms of RD for each individual sequence. Figure 5.7 illustrates the speedup achieved in each sequence in comparison with the original implementation of AFQLD.

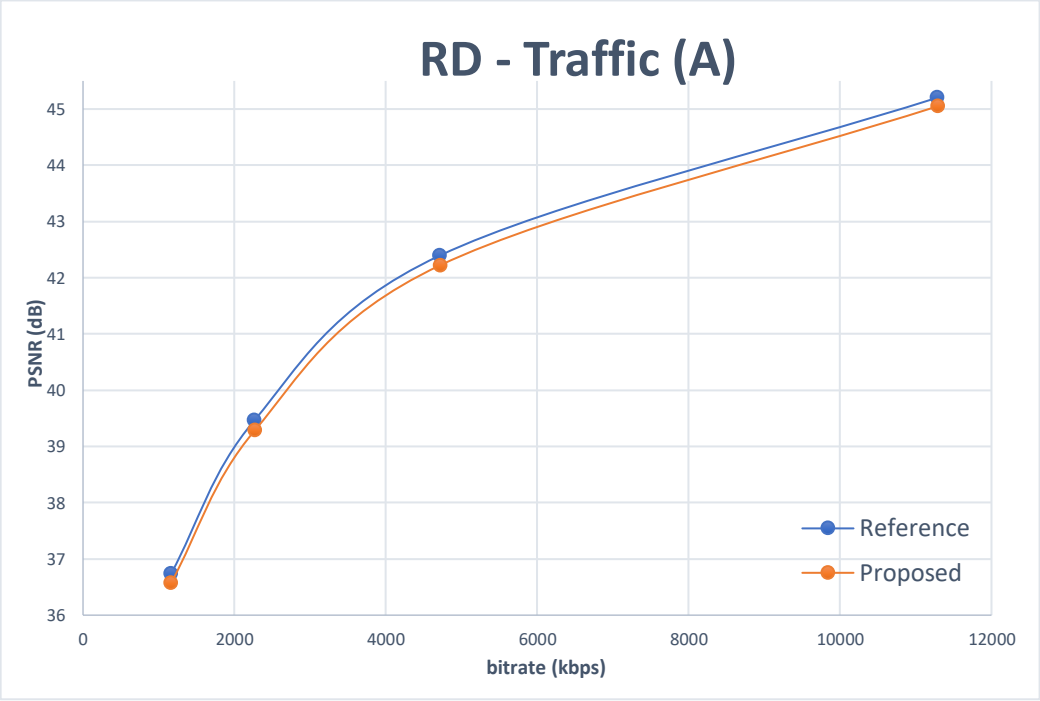

Figure 5.1. RD graph of the Traffic sequence.

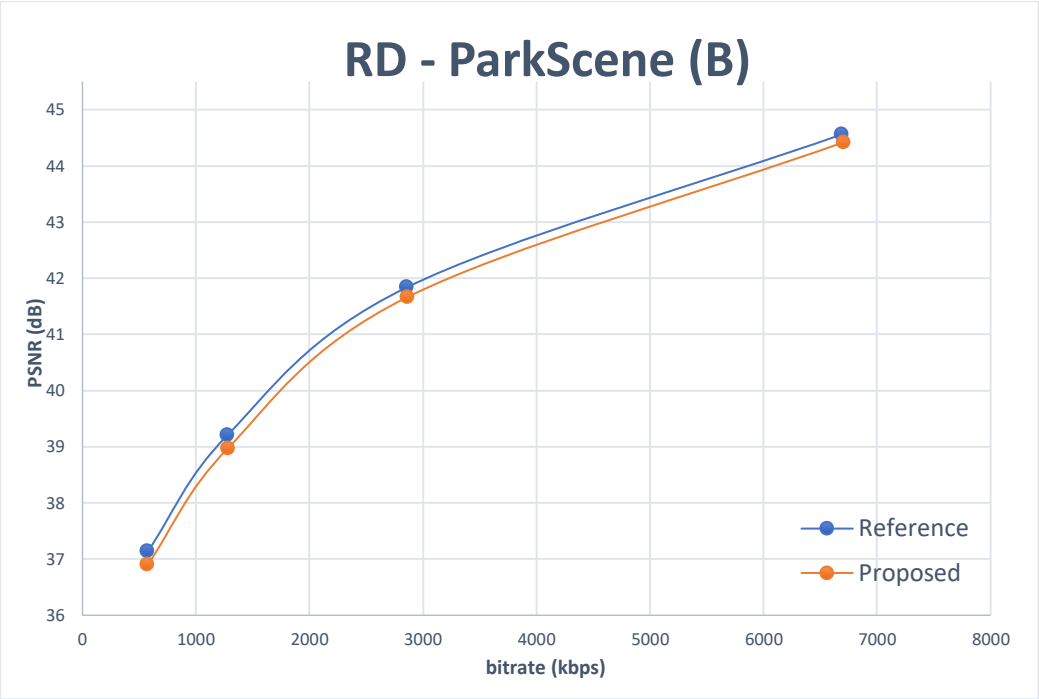

Figure 5.2. RD graph of the ParkScene sequence.

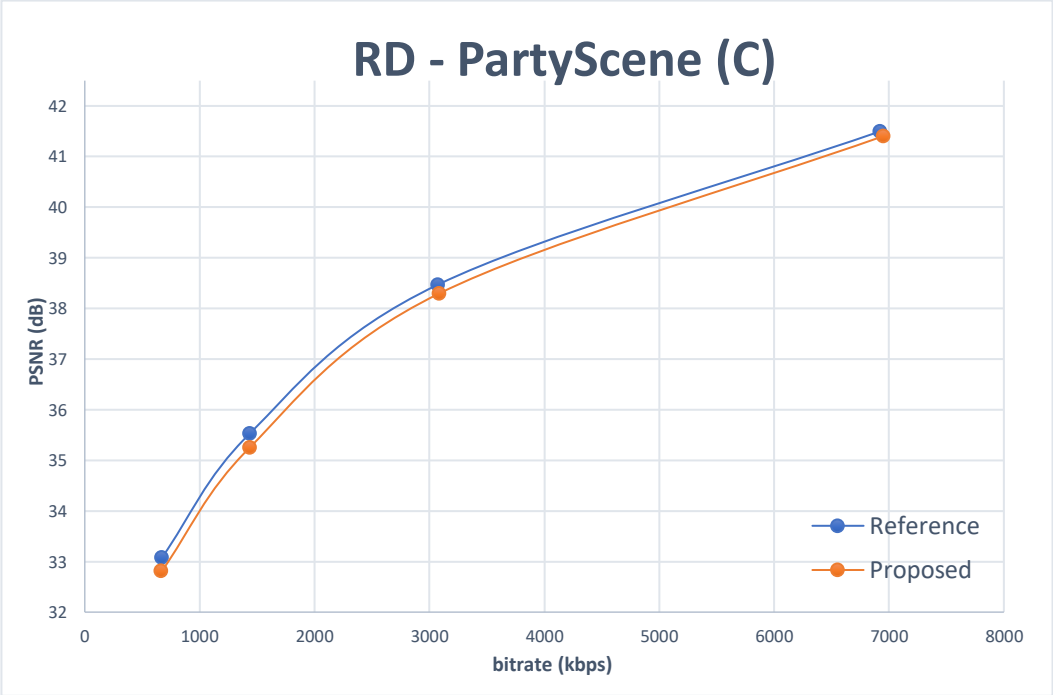

Figure 5.3. RD graph of the PartyScene sequence.

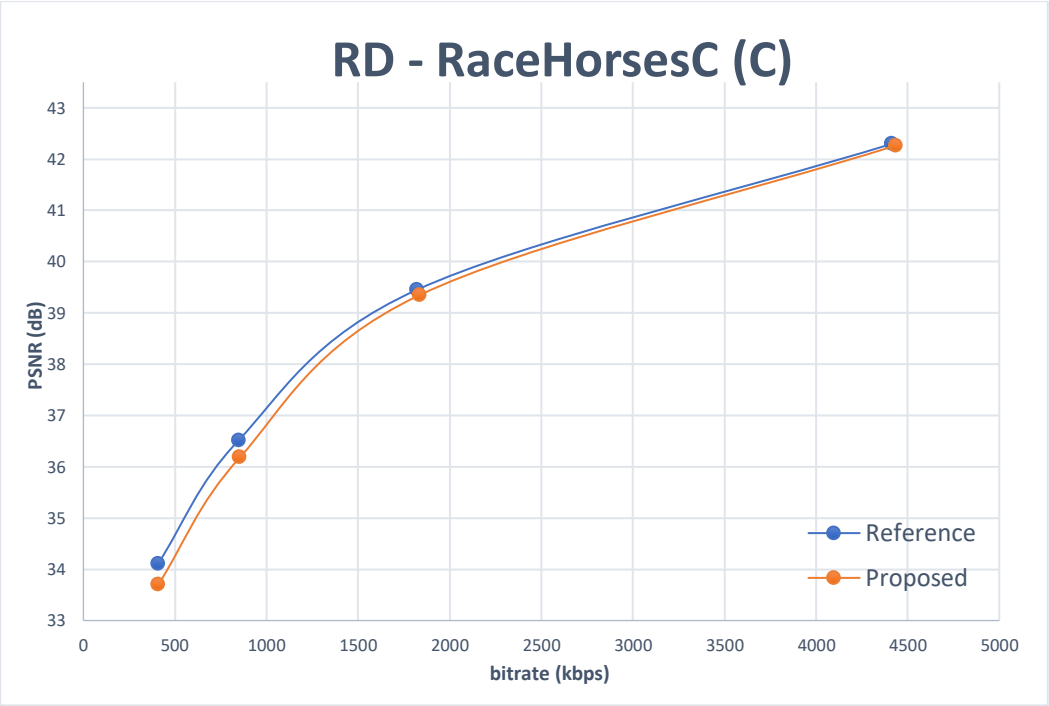

Figure 5.4. RD graph of the RaceHorsesC sequence.

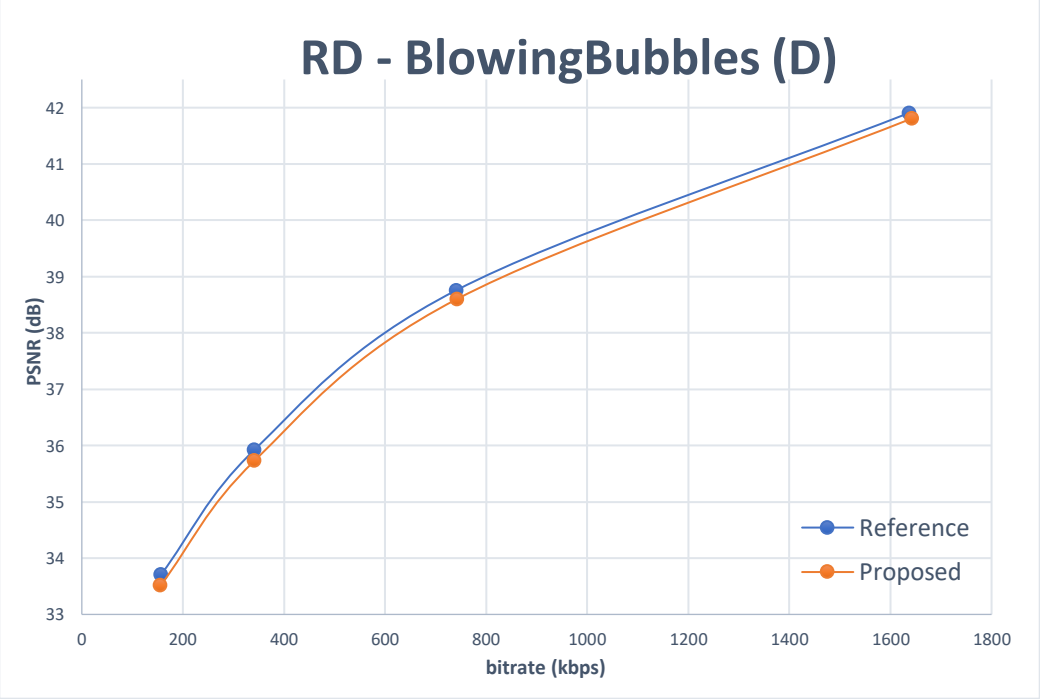

Figure 5.5. RD graph of the BlowingBubbles sequence.

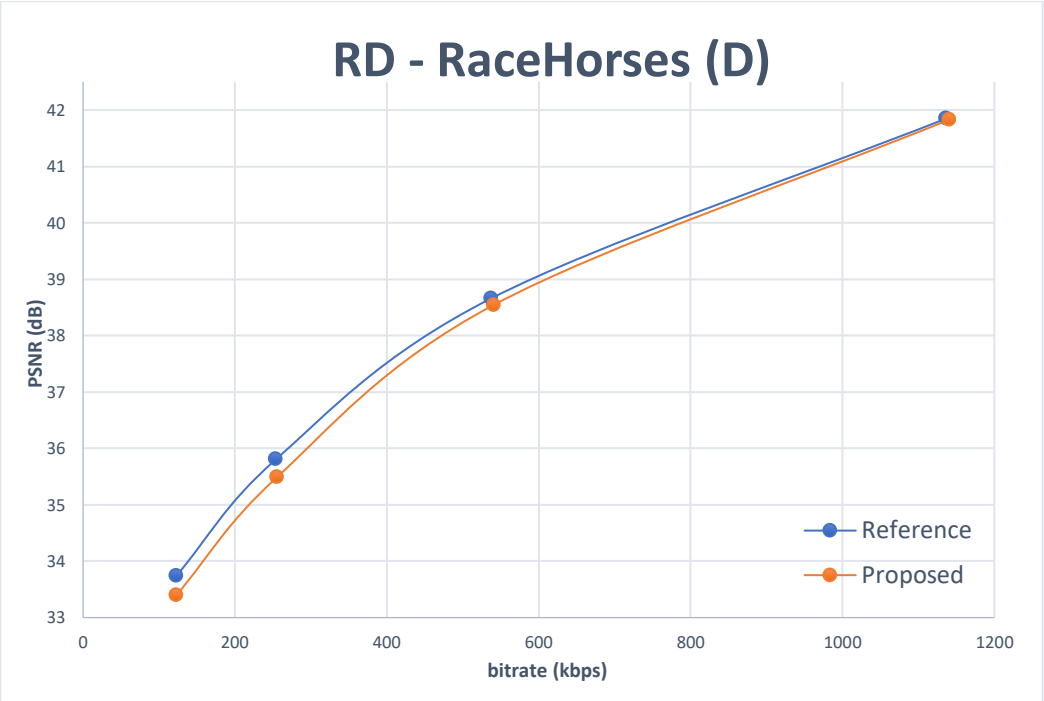

Figure 5.6. RD graph of the RaceHorses sequence.

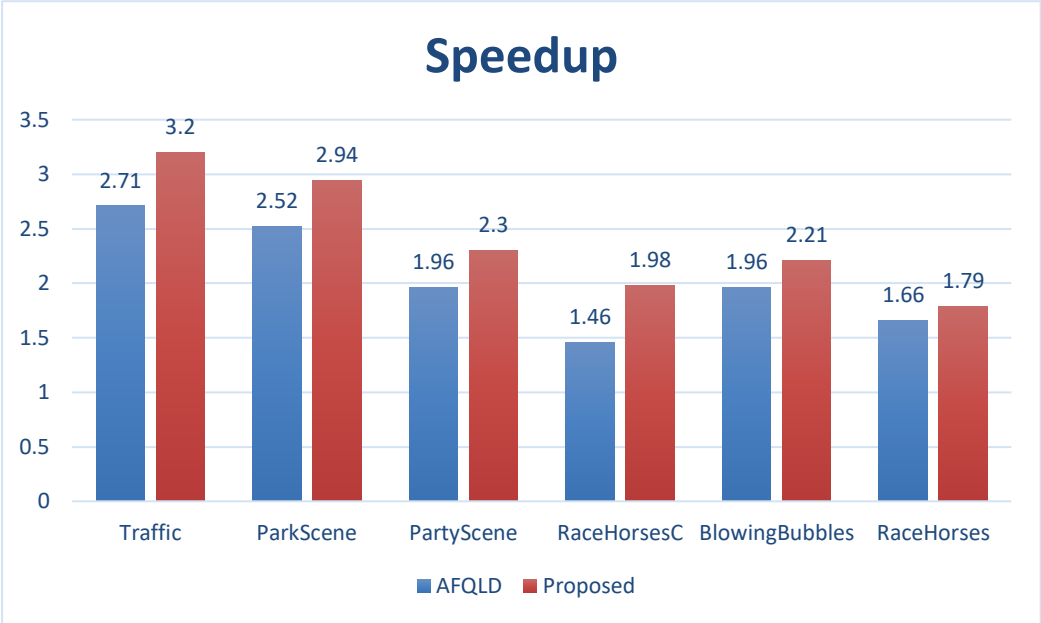

Figure 5.7. Comparison of the speedup obtained by AFQLD and the proposed transcoder.

In the RD graphs above we can se that the curves are very close to each other, which indicates that there is not a big loss in terms of RD but, in accordance with the results displayed in Table 5.1, we can see that the RD curve for the proposed transcoder is slightly below the curve for the reference transcoder. This means that the proposal achieves a lower PSNR value, obtaining a lower video quality with the same bitrate as the reference transcoder.

However, the speedup achieved over it is quite significant, and, as shown in Figure 5.7, surpasses AFQLD in every sequence. This is what makes this transcoder useful for several transcoding and encoding situations.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

This last chapter ends this document by reviewing the progress made throughout the project and analyzing the benefits and applications of the proposed solution. Finally, some possible extensions to the work developed are proposed.

6.1 Conclusions

In this project, a new proposal for video transcoding from H.264/AVC to HEVC based on the AFQLD algorithm has been formulated and tested. This proposal uses a ML algorithm to help the HEVC encoder take decisions based on the information fetched from the H.264/AVC decoder, which allows to accelerate the process prominently at the expense of an admissible increase in bitrate.

First, several Machine Learning practices and algorithms have been studied and reviewed. This provided the necessary knowledge for the subsequent testing and application of some of them in the proposed transcoder.

Then, a global vision of the video coding field of study has been provided, describing the most common coding standards and compression techniques, but focusing on the important improvements introduced by H.264/AVC and HEVC, which are the two protagonists of this work. Also, the state of the art of video transcoding has been reviewed, making a quick overview of the most relevant contributions on the topic and highlighting the diversity of the previous approaches. Here, the main features of the AFQLD algorithm have also been introduced with more detail.

After exposing the basics, the previously discussed Machine Learning algorithms have been tested over datasets with features extracted from the H.264/AVC decoder, yielding promising results, with several algorithms ranking higher than the Naïve-Bayes classifier used by default in AFQLD. The most relevant algorithm, Random Forest, has been implemented in the HEVC encoding process, requiring an interconnection solution between AFLQD's C code and the Weka utility written in Java.

The proposed transcoder has been then tested with some video sequences, obtaining interesting results, better in some respects than the state-of-the-art approaches, mainly for the higher-resolution sequences, where a 3.20x speedup was obtained with respect to the 2.71x originally obtained by AFQLD, while increasing the bitrate only by an extra 0.4%.

As a final conclusion, the project succeeded in fulfilling the original objectives while providing as a result an interesting transcoding alternative that can be used in the most demanding scenarios, adequate for some content migration situations and, even more, as a quicker way to encode new sequences in HEVC, going through a prior H.264/AVC encoding-decoding phase.

6.2 Future work

Having mentioned the set of conclusions obtained from this project, several improvements and extensions can be developed over it:

- Testing other algorithms not considered in this work, like CART or C5.0. The comparison performed was extensive enough to cover the main families of Machine Learning algorithms but could be extended to include more variants. Also, other preprocessing techniques could be covered, and more tests could be performed with the availability of high-performance machines able to reduce substantially the great time required for building and testing the models.

- Implementing the classification models natively in C in order to avoid the usage of Java and the consequent overhead introduced by the interconnection process and the calls to the Weka API.

- Making the algorithm adaptive to the sequence that is being coded, a characteristic that has not been inherited from the original AFQLD algorithm, using for instance an initial cost analysis.

REFERENCES

[1] Cisco, "Visual Networking Index: Forecast and Methodology, 2015–2020," 2016.

[2] ITU, "High Efficiency Video Coding," *Document ITU-T Rec. H.265 and ISO/IEC 23008-2 (Version 1),* 2013.

[3] "Advanced Video Coding for Generic Audiovisual Services," *document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) (Version 22),* 2012.

[4] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan and T. Wiegand, "Comparison of the coding efficiency of video coding standards— Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1669-1684, 2012.

[5] F. Bossen, B. Bross, K. Sühring and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, no. 12, pp. 1685-1696, 2012.

[6] L. Merritt and R. Vanam, X264: A High Performance H.264/AVC Encoder, Seattle, 2006.

[7] T. K. Tan, R. Weerakkody, M. Mrak, N. Ramzan, V. Baroncini, J. R. Ohm and G. Sullivan, "Video Quality Evaluation Methodology and Verification Testing of HEVC Compression Performance," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 26, no. 1, pp. 76-90, 2016.

[8] A. Vetro, C. Christopoulos and H. Sun, "Video transcoding architectures and techniques: An overview," *IEEE Signal Processing Magazine,* vol. 20, no. 2, pp. 18-29, 2003.

[9] A. J. Díaz-Honrubia, J. L. Martínez, P. Cuenca, J. A. Gamez and J. M. Puerta, "Adaptive Fast Quadtree Level Decision Algorithm for H.264 to HEVC Video Transcoding," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 26, no. 1, pp. 154-168, 2016.

[10] U. Fayyad, G. Piatetsky-Shapiro and a. P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine,* vol. 17, no. 3, pp. 37-54, 1996.

[11] S. Ramírez-Gallego, S. García, H. Mouriño-Talín, D. Martínez-Rego, V. Bolón-Canedo, A. Alonso-Betanzos, J. M. Benítez and F. Herrera, "Data Discretization: Taxonomy and Big Data Challenge," *WIREs Data Mining and Knowledge Discovery,* vol. 6, no. 1, pp. 5-21, 2016.

[12] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection," *Journal of Machine Learning Research,* vol. 3, pp. 1157-1182, 2003.

[13] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence,* vol. 97, no. 1-2, pp. 273-324, 1997.

[14] N. Sánchez-Maroño, A. Alonso-Betanzos and M. Tombilla-Sanromán, "Filter Methods for Feature Selection – A Comparative Study," in *International Conference on Intelligent Data Engineering and Automated Learning*, Birminghan, 2007.

[15] T. N. Lal, O. Chapelle, J. Weston and A. Elisseeff, "Embedded Methods," *Studies in Fuzziness and Soft Computing,* vol. 207, pp. 137-165, 2006.

[16] J. R. Quinlan, C4.5: Programs for Machine Learning, San Francisco: Morgan Kaufmann Publishers, Inc., 1993.

[17] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995.

[18] "Lecture 9: SVM," Cornell University, [Online]. Available: http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote09.html. [Accessed 30 6 2018].

[19] "Simple tutorial on SVM and Parameter Tuning in Python and R," Hackerearth, [Online]. Available: https://www.hackerearth.com/blog/machine-learning/simple-tutorial-svm-parameter-tuning-python-r/. [Accessed 30 6 2018].

[20] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[21] "Random Forest Classifier - Machine Learning," Global Software Support, [Online]. Available: http://www.globalsoftwaresupport.com/random-forest-classifier-bagging-machine-learning/. [Accessed 30 6 2018].

[22] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory,* vol. 13, no. 1, pp. 21-27, 1967.

[23] "K-nearest neighbors algorithm," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. [Accessed 30 6 2018].

[24] P. J. Braspenning, F. Thuijsman and A. J. M. M. Weijters, Artificial Neural Networks: An Introduction to ANN Theory and Practice, Heidelberg: Springer, 1995.

[25] R. M. S. d. Oliveira, R. C. F. Araújo, F. J. B. Barros, A. P. Segundo, R. F. Zampolo, W. Fonseca, V. Dmitriev and F. S. Brasil, "A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges," *Journal of Microwaves, Optoelectronics and Electromagnetic Applications,* vol. 16, no. 3, 2017.

[26] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE,* vol. 49, no. 1, pp. 8-30, 1961.

[27] P. M. Domingos and M. J. Pazzani, "Beyond independence: conditions for the optimality of the simple Bayesian classifier," in *Proceedings of the International Conference on Machine Learning*, Bari, 1996.

[28] Y. Liu, G. Zhai and X. L. D. Zhao, "Frame Rate and Perceptual Quality for HD Video," in *PCM 2015: 16th Pacific-Rim Conference on Multimedia*, Gwangju, 2015.

[29] P. Á. C. Cuenca, "Técnicas de Compresión de Imágenes, Vídeo y Audio," 2013.

[30] P. Á. C. Cuenca, "Captura, Formatos y Métricas de las señales Audiovisuales," 2013.

[31] "Inter frame," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Inter_frame. [Accessed 30 6 2018].

[32] ITU, "Codecs for Videoconferencing Using Primary Digital Group Transmission," *Document ITU-T Rec. H.120,* 1993.

[33] ITU, "Video Codec for Audiovisual Services at p x 64 kbits," *Document ITU-T Rec. H.261,* 1993.

[34] ISO/IEC, "Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s," *Communications of the ACM,* 1993.

[35] ISO/IEC, "Generic coding of moving pictures and associated audio information," 2000.

[36] ITU, "Video coding for low bit rate communication," *Document ITU-T Rec. H.263,* 2005.

[37] ISO/IEC, "Coding of audio-visual objects. Part 2: Visual," 2004.

[38] T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, 2003.

[39] P. Á. C. Cuenca, "Codificación Avanzada de Vídeo," 2013.

[40] X. Tian, T. M. Le and Y. Lian, "Review of CAVLC, Arithmetic Coding, and CABAC," in *Entropy Coders of the H.264/AVC Standard. Signals and Communication Technology*, Heidelberg, Springer, 2011, pp. 29-39.

[41] I. K. Kim, K. McCann, K. Sugimoto, B. Bross and W. J. Han, "HM9: High Efficiency Video Coding (HEVC) Test Model 9 Encoder Description," Shangai, 2012.

[42] F. Bossen, "Common test conditions and software reference configurations," Ginebra, 2011.

[43] D. Zhang, B. Li, J. Xu and H. Li, "Fast Transcoding from H.264 AVC to High Efficiency Video Coding," in *IEEE International Conference on Multimedia and Expo*, Melbourne, 2012.

[44] G. Bjøntegaard, "Improvements of the BD-PSNR Model," in *ITU-T SG16 Q6*, Berlin, 2008.

[45] E. Peixoto and E. Izquierdo, "A complexity-scalable transcoder from H.264/AVC to the new HEVC codec," in *IEEE International Conference on Image Processing*, Orlando, 2013.

[46] W. Jiang, Y. Chen and X. Tian, "Fast transcoding from H.264 to HEVC based on region feature analysis," *Multimedia Tools and Applications*, vol. 73, no. 3, pp. 2179-2200, 2013.

[47] "HM Reference Software," [Online]. Available: https://hevc.hhi.fraunhofer.de/HM-doc/. [Accessed 30 6 2018].

[48] T. Shen, Y. Lu, Z. Wen, L. Zou, Y. Chen and J. Wen, "Ultra Fast H.264/AVC to HEVC Transcoder," in *Data Compression Conference*, Snowbird, 2013.

[49] E. Peixoto, B. Macchiavello, E. M. Hung, A. Zaghetto, T. Shanableh and E. Izquierdo, "An H.264/AVC to HEVC video transcoder based on mode mapping," in *IEEE International Conference on Image Processing*, Melbourne, 2013.

[50] E. Peixoto, T. Shanableh and E. Izquierdo, "H.264/AVC to HEVC Video Transcoder Based on Dynamic Thresholding and Content Modeling," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 1, pp. 99-112, 2014.

[51] E. Peixoto, B. Macchiavello, E. M. Hung and R. L. d. Queiroz, "A fast HEVC transcoder based on content modeling and early termination," in *IEEE International Conference on Image Processing*, Paris, 2014.

[52] A. J. Díaz-Honrubia, J. L. Martínez, J. M. Puerta, J. A. Gámez, J. D. Cock and P. Cuenca, "Fast quadtree level decision algorithm for H.264/HEVC transcoder," in *IEEE International Conference on Image Processing*, Paris, 2014.

[53] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145-1159, 1997.

[54] P. Refaeilzadeh, L. Tang and H. Liu, "Cross-Validation," in *Encyclopedia of Database Systems*, Springer, 2009, pp. 532-538.

[55] U. M. Fayyad and K. B. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *International Joint Conference on Artificial Intelligence*, Chambery, 1993.

[56] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10-18, 2009.

[57] "Java Native Interface," Oracle, [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/guides/jni/. [Accessed 30 6 2018].

[58] "JM Reference Software," [Online]. Available: http://iphome.hhi.de/suehring/tml/. [Accessed 30 6 2018].

ACRONYMS

AFQLD	Adaptive Fast Quadtree Level Decision algorithm.
AI	All Intra. One of the predefined frame structures in the CTC.
AVC	Advanced Video Coding standard (H.264/AVC).
BD-rate	Bjøntegaard Distortion-rate. Measures the difference in bitrate between two encoders to achieve the same objective quality.
CABAC	Context-based Adaptive Binary Arithmetic Coding. An entropy coder used in H.264/AVC.
CAVLC	Contect Adaptive Variable Length Coding. An entropy coder used in H.264/AVC.
CTB	Coding Tree Block. Block of samples.
CTU	Coding Tree Unit. Composed of three CTUs (one for luminance and two for chrominance).
CU	Coding Unit. Unit that definesa sub-partition of an image in HEVC.
CTC	Common Test Conditions. Document that defines a common framework for testing and comparing different video coding standards.

DCT	Discrete Cosine Transform. The most popular transform coding technique. Used in H.264/AVC and HEVC.
FPS	Frames per second.
FQLD	Fast Quadtree Level Decision algorithm.
FS	Feature Selection. Process of selecting the most interesting features from a data source for a subsequent classification.
GOP	Group Of Pictures. Frame layout containing different types of frames.
GR	Gain Ratio. Used by C4.5 to find the most interesting features.
H.264	H.264/AVC. Coding standard jointly developed by the ITU-T and the ISO/IEC.
HD	High Definition. 1280x720 pixels resolutions.
HEVC	High Efficiency Video Coding. Coding standard successor of H.264/AVC and jointly developed by the ITU-T and the ISO/IEC.
HM	HEVC test Model. Reference software implementing the HEVC standard.
IEC	International Electrotechnical Commision. Normalization entity in the electric and electronic fields.
IEEE	Institute of Electrical and Electronics Engineers. International association focused on standardization and development in technical fields.
IG	Information Gain. Used to calculate the Gain Ratio.
ISO	International Organization for Standardization. Composed by several national standardization organizations. Creates international standards.
ITU	International Telecommunication Union. Organism of the United Nations responsible of regulating telecommunications worldwide.

JCT-VC	Joint Collaborative Team on Video Coding. Group of experts from the ITU-T and the ISO/IEC responsible of the creation of the HEVC standard.
JM	Joint Model. Reference software implementing the H.264/AVC standard.
JVT	Joint Video Team. Group of experts from the ITU-T and the ISO/IEC who created the H.264/AVC coding standard.
KLT	Karhunen-Loeve Transform. Transform coding technique.
LDB	Low Delay B. One of the predefined frame structures in the CTC.
LDP	Low Delay P. One of the predefined frame structures in the CTC.
MB	Macroblock. Basic processing unit in H.264/AVC.
ML	Machine Learning. Branch of the Artificial Intelligence focused on developing techniques that allow machines to learn.
MV	Motion vector. Used in inter-frame prediction to represent more efficiently the movement between frames.
MPEG	Moving Picture Experts Group. Group of experts of the ISO/IEC focused on developing audio and video standards.
NAL	Network Abstraction Layer. Allows to maintain the video sequence syntax in any network.
PTCM-LDF	Proposed Transcoder for Content Modelling using Linear Discriminant Functions. H.264/AVC to HEVC Transcoder proposed by Peixoto *et al.*
PSNR	Peak Signal to Noise Ratio. Metric used to measure the objective quality of a video sequence.
PU	Prediction Unit. Elementary prediction unit in HEVC.
QP	Quantization Parameter. Impacts the compression ratio of an encoder.

RA	Random Access. One of the predefined frame structures in the CTC.
RD	Rate-Distortion. Analytical expression for how much compression can be achieved using lossy compression methods.
RGB	Red Green Blue. Color model.
RLC	Run Length Coding. Lossless data compression technique.
RLE	Run Length Encoding. Refers to RLC.
SAO	Sample Adaptive Offset. Filter which reduces distortion and improves the general quality of the frames. Included in the HEVC standard.
TU	Transform Unit. Units where transform and quantization are applied in HEVC.
VLC	Variable Length Coding. A kind of entropy coding based on assigning variable-length words to each sample.
YUV	Color format representing the signal as one luminance (Y) and two chrominances (U and V).

APPENDIX A. **CONTENT OF THE ZIP FILE**

Along with this document comes a ZIP file including:

- The Java code used for automating the algorithm testing done in 4.1.3 (*ML algorithms testing code*).
- The modified C code of AFQLD and the Java code with which it communicates to perform the classification (*Classification code*).
- The files which indicate the CU and PU partitioning to the encoder (*CU and PU files*).
- The modified version of the HM software used in 5.2 (*HM16*).
- The training datasets used for building the ML models (*Transcoding databases)*.
- The Random Forest models used for online classification (*Models*).
- The output TXT files from the HM software which indicate the bitrate, PSNR and time consumed to encode a sequence (*HM results*).