electronics

MDPI

Article

Bagged Tree and ResNet-Based Joint End-to-End Fast CTU Partition Decision Algorithm for Video Intra Coding

Yixiao Li [1,2], Lixiang Li [1,2,*], Yuan Fang [1,2], Haipeng Peng [1,2] and Nam Ling [3]

[1] Information Security Center, State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; lyx@bupt.edu.cn (Y.L.); fangyuan@bupt.edu.cn (Y.F.); penghaipeng@bupt.edu.cn (H.P.)

[2] National Engineering Laboratory for Disaster Backup and Recovery, Beijing University of Posts and Telecommunications, Beijing 100876, China

[3] Department of Computer Science and Engineering, Santa Clara University, Santa Clara, CA 95053, USA; nling@scu.edu

* Correspondence: lixiang@bupt.edu.cn; Tel.: +86-010-6228-2264

Abstract: Video coding standards, such as high-efficiency video coding (HEVC), versatile video coding (VVC), and AOMedia video 2 (AV2), achieve an optimal encoding performance by traversing all possible combinations of coding unit (CU) partition and selecting the combination with the minimum coding cost. It is still necessary to further reduce the encoding time of HEVC, because HEVC is one of the most widely used coding standards. In HEVC, the process of searching for the best performance is the source of most of the encoding complexity. To reduce the complexity of the coding block partition in HEVC, a new end-to-end fast algorithm is presented to aid the partition structure decisions of the coding tree unit (CTU) in intra coding. In the proposed method, the partition structure decision problem of a CTU is solved by a novel two-stage strategy. In the first stage, a bagged tree model is employed to predict the splitting of a CTU. In the second stage, the partition problem of a 32 × 32-sized CU is modeled as a 17-output classification task for the first time, so that it can be solved by a single prediction. To achieve a high prediction accuracy, a residual network (ResNet) with 34 layers is employed. Jointly using bagged tree and ResNet, the proposed fast CTU partition algorithm is able to generate the partition quad-tree structure of a CTU through an end-to-end prediction process, which abandons the traditional scheme of making multiple decisions at various depth levels. In addition, several datasets are used in this paper to lay the foundation for high prediction accuracy. Compared with the original HM16.7 encoder, the experimental results show that the proposed algorithm can reduce the encoding time by 60.29% on average, while the Bjøntegaard delta rate (BD-rate) loss is as low as 2.03%, which outperforms the results of most of the state-of-the-art approaches in the field of fast intra CU partition.

Keywords: video coding; fast coding unit (CU) partition; residual network (ResNet); high-efficiency video coding (HEVC); intra coding

check for updates

Citation: Li, Y.; Li, L.; Fang, Y.; Peng, H.; Ling, N. Bagged Tree and ResNet-Based Joint End-to-End Fast CTU Partition Decision Algorithm for Video Intra Coding. *Electronics* **2022**, *11*, 1264. https://doi.org/10.3390/electronics11081264

Academic Editor: Stefanos Kollias

Received: 4 March 2022
Accepted: 13 April 2022
Published: 16 April 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Video coding standards have been continuously developed and updated for decades to meet the increasing demand of the video market for videos with higher definition. In recent years, various coding standards have been invented and replaced, such as advanced video coding (AVC), high-efficiency video coding (HEVC), versatile video coding (VVC), audio-video coding standard (AVS), and AOMedia video 1 (AV1). All of these methods have considerable coding benefits; for example, developed by the Joint Collaborative Team on Video Coding (JCT-VC), VVC and HEVC are both able to achieve a 50% lower bit rate than their predecessors while maintaining the same video quality [1]. However, with the repeated transition of the video compression rate, the complexity of various video coding standards has increased dramatically due to the introduction of many efficient but complicated coding tools.

Block-based coding is one of the most important coding techniques and is commonly used in many popular video coding standards. Taking HEVC and VVC as an example, HEVC uses a quad-tree structure to partition a coding unit (CU). There are two options for a CU of size 16 × 16 (i.e., nonsplit or split into four sub-CUs of size 8 × 8). However, the options for a CU of size 32 × 32 increase to 17 ($1 + 2^4$), and for a coding tree unit (CTU), the options are as many as 83,522 ($1 + 17^4$). Specifically, a CTU in VVC can be as large as 128 × 128. VVC uses a quad-tree plus binary-tree (QTBT) structure to finish block partitioning [2]. This supports the use of horizontal and vertical binary-tree (BT) and ternary-tree (TT) structures to further expand the partition structure. The intra-coding complexity of VVC is, on average, 18 times higher than that of HEVC [3]. In block-based video coding standards, each CU can then be iteratively split into sub-CUs according to a specific partition structure. A flexible CU partition rule results in various size combinations for a CU. With many CU sizes and modes to be selected, rate-distortion optimization (RDO) is adopted to select the optimal CU partition structure for a CTU, along with prediction modes. During the encoding process, many video coding standards traverse all possible CU sizes and prediction modes, and then RDO is used to select the best combination with the minimum cost. As a result, RDO results in the greatest encoding complexity burden by exhaustive calculations in the CU size decision process, and it restricts the application of video coding standards in many real-time scenarios. Thus, it is necessary to design a fast CU partition algorithm for existing video coding standards.

Although the encoding complexity of VVC is far greater than that of HEVC, HEVC is used much more widely than VVC in industrial applications [4]. Considering the high complexity of HEVC and VVC, algorithms reducing their encoding time are urgently needed. However, the partition structure of a CTU in HEVC is simpler than that of VVC, although HEVC has a similar block-based coding structure to VVC, while there are differences in some partition cases. Furthermore, acceleration algorithms based on deep learning usually need the support of a graphics processing unit (GPU), and GPUs supporting HEVC hardware encoding–decoding are far more mature and popular than those of VVC. To validate the effectiveness of our approach, the implementation of HEVC is easier than that of VVC, and is also highly practical for industrial applications. Intra coding is frequently used in sequence encoding, in terms of the intra profile and the encoding of key reference frames in other profiles. One of the properties of intra coding is that the splitting decision is only related to the information in the current frame, which allows classifiers to predict the partition structure from pixels directly. In comparison, inter coding involves not only information regarding the current frame, but also factors from the time domain. This makes the splitting decision of inter CU another focus-of-research point, as intra coding is the basis of inter coding. As a result, we focus on the partition problem of HEVC intra coding in this paper.

In the past several years, many approaches have been proposed to reduce the massive encoding complexity of various standards, such as HEVC and VVC. Works focusing on fast 3D-HEVC encoding have been developed [5–8]. Some researchers proposed fast video coding methods from the perspective of hardware design. Zhang et al. [9] presented four algorithm adaptations and a fully parallel hardware architecture for an H.265/HEVC intra encoder, the first of its kind. In addition, Zhang et al. [10] also presented high-performance algorithm adaptations and a high-throughput hardware architecture for the HEVC intra encoders. Cai et al. [11] proposed an efficient intra mode decision algorithm for the parallel hardware architecture of the AVS3 intra encoder by processing CUs in parallel, including intra prediction and estimating the rate-distortion cost for mode decision. Sjövall et al. [12] introduced the first complete high-level synthesis (HLS) implementation for an HEVC intra encoder on FPGA, and designed a proof-of-concept system for hardware-accelerated HEVC encoding. Zummach et al. [13] proposed a hardware design for the AV1-constrained directional enhancement filter, targeting the real-time processing of 4K ultra-high-definition videos.

Heuristic-based CU depth decision approaches have been proposed and widely studied [14–20]. For example, Liu et al. [15] proposed a fast CU depth decision algorithm based on statistical analysis. They used a three-stage method to make the splitting result decision according to prior information. Shen et al. [21] proposed an early determination and a bypass strategy for CU size decisions by using the texture property of the current CU and coding information from neighboring CUs. In some approaches, the CU depth range was shorted, and some CU depth levels were skipped according to statistical information [22,23].

Although the heuristic-based fast methods have achieved many acceptable results, they cannot properly consider the partition property of various video sequences. In other words, there are too many factors to influence the partition result. Furthermore, these factors may change with different sequences, such that people do not usually know which combination of these factors has the best performance upon implementation. Generally, we only consider several key factors closely correlated with CU partition, and a small number of the considered factors may lead to a poor result.

The technique of classical machine learning algorithms is introduced in order to overcome the drawbacks of statistical-information-based methods. Support vector machine (SVM) models with three outputs are used to achieve a trade-off between bit distortion and encoding complexity [24,25]. In addition, to further improve the prediction accuracy, two or more SVM models are employed in each CU depth. Zhang et al. [26] employed two SVMs at each depth to make the decisions regarding early CU split and early CU termination. Zhu et al. [27] used the cascaded SVM and defined a misclassification cost and a risk area to jointly make a CU partition decision. Meanwhile, Grellert et al. [28] and Zhang et al. [29] also proposed SVM-based approaches which focused on features analysis. In addition, decision tree or data mining methods were also used to reduce the encoding complexity [30–34]. Furthermore, Fisher's linear discriminant analysis and the *k*-nearest neighbors classifier were employed in order to quickly decide on a CU partition [35], and Kim et al. [36] proposed a joint online and offline Bayesian decision rule-based fast CU partition algorithm. Moreover, Yang et al. [37] proposed an efficient low-complexity intra coding algorithm for VVC by using learning-based classifiers.

Although the classical machine learning model-based methods outperform the heuristic-based methods due to their advantages when dealing with high-dimensional problems, their inputs (i.e., features) play a very important role in the encoding results. In addition, the features' design is entirely manual and requires much experience. Additionally, there are many key features which are commonly used in existing works. Hence, in order to significantly improve the encoding performance, researchers should find more efficient features, which is usually quite difficult. Furthermore, one or more classifiers are generally needed for each depth, which requires much work and a long training time. Algorithms using the convolutional neural network (CNN) were proposed to address these problems [38–40]. Due to the properties of CNN when it comes to automatic region feature extraction, these algorithms have enormous advantages in image processing. CNN-based algorithms have also achieved many good encoding results.

Jamali et al. [41] used deep reinforcement learning to reduce the encoding complexity of HEVC intra coding. Amna et al. [42] proposed a LeNet5-based approach for fast intra coding. Liu et al. [43] devised a convolutional neural network (CNN)-based fast algorithm to prune no less than two partition modes for RDO processing on each CTU. However, its CNN structure is too shallow to fully learn the relationship between the image data and the partition structure. In addition, considering all available partition modes for a CU, only a minimum of two CU partition modes are pruned, which is not enough. In addition, the algorithm proposed by Kim et al. [44] used image data and an encoding-information-based vector data to train a CNN for the prediction of the CTU depth. However, in this algorithm, not only image data but also vector data need to be collected before the prediction phase, which requires more pre-encoding time. In addition, three kinds of CNN structures should be constructed, with each being designed for a CU of a certain depth. Therefore, at least three CNNs are needed for one video sequence. Furthermore, Xu et al. [45] proposed an

approach using a CNN and a long short-term memory (LSTM) network. Specifically, the CNN was used to predict the CU partition of intra coding, and the LSTM network was used to predict the CU partition for inter coding. In their algorithm, the CNN, especially the LSTM part, is highly complex, and requires time to be trained and refined.

As we can see, these CNN-based methods still complete the CU partition prediction at various depth levels, with one or more CNN models being needed for each CU depth. This means that at least three deep learning models are needed for a single video sequence. In other words, existing works still model the CU partition as a binary classification problem, which is a great waste of the CNN's abilities. An end-to-end approach is needed. A CTU in HEVC has 83,522 possible partition structures in total, which makes it hard for a deep learning classifier fed with CTU pixels to correctly choose the right partition structure through a single prediction. However, the number of possible partition structures of a 32×32 CU is 17, and the partition structure of a 32×32 CU is much easier for a deep learning classifier to predict than that of a CTU. Thus, we take a two-stage strategy to complete the CTU partitioning.

Although the name "two-stage" was mentioned by [24,46], the meaning of "two-stage" in our paper is quite different to theirs. Specifically, they both solve the estimation problem in a traditional way, by which the splitting decision of a CU is made depth by depth. In [24,46], "two-stage" means that an additional binary classification will be performed for each depth for CUs left undetermined in the first stage. In our manuscript, the two-stage strategy means we only need to perform a binary classification for a CTU, then the partition structure can be directly determined by a single prediction.

The first stage is the task of splitting decisions from the CTU to 32×32 CUs, which can be handled by a simple learning-based method. In this paper, we use a bagged tree at the CTU level due to its simple structure and fast prediction speed. Among traditional machine learning methods, prediction efficiency, time complexity, and implementation difficulty vary a great deal. However, compared with other models, the tree model has its unique advantages. First, it aligns with people's common sense when it comes to some classification problems. Thus, it can achieve quite a high accuracy among some particular problems through proper training. Second, it is much easier to train and takes less time to finish predictions, which results in negligible overhead times. Third, it is also easier to be implemented into existing works due to the simple "if…else…" structure-based prediction process. A deep learning network can also be used, but it contains far more parameters and is designed for difficult tasks. It would be a waste of ability if we also employed a deep learning network at the CTU level.

The second stage is the decision of the partition structure for a 32×32 CU. A 32×32 CU can be partitioned 17 ways, such that the decision can be predicted by deep learning techniques as a multi-classification task. Because the differences among these 17 classes are very small, partition structures are similar to each other, which makes it hard for a sample network to solve the end-to-end CU partition problem. More layers usually means a higher accuracy; ResNet can contain many layers while maintaining a good convergence performance. We chose a ResNet with 34 layers as the classifier to achieve a high prediction accuracy, which is suitable for our prediction task.

In this paper, we construct a deep structure to explore the learning capacity of ResNet. Meanwhile, we propose an end-to-end solution for the CTU partition, which models the CU partition as an unprecedented multi-classification problem. To verify the effectiveness of our proposed method, we implement it into HEVC intra coding. First, bagged tree models are employed to classify a CTU to sharply shorten the classification categories. Then, an end-to-end ResNet is trained to predict the final partition structure of a 32×32 CU instead of predicting the splitting decision of a CU at a certain depth.

The main contributions of this paper are presented as follows:

1. Adopting a two-stage prediction strategy with a combination of bagged tree and ResNet, the proposed algorithm can achieve more accurate prediction results. It reduces the CU partition categories sharply and effectively, and abandons the traditional scheme of making multiple decisions at various depth levels;
2. For the first time, the partition task of a 32 × 32 CU is modeled as a 17-class problem, for there are 17 ways in total to partition a 32 × 32 CU. The partition structure can be decided by an end-to-end ResNet model through a single prediction;
3. A general solution to CU partitioning in video intra coding is proposed, and is verified on HEVC. In a similar way, it can be implemented on other block-based video coding standards and reduce encoding times.

This paper is organized as follows. Section 2 introduces the quad-tree-based partition structure and gives a brief review of bagged tree and ResNet. Section 3 describes the proposed bagged tree and ResNet-based end-to-end joint fast CTU partition algorithm. Section 4 reports the experimental results. The conclusions are summarized in Section 5.

2. Background

In this section, we first describe the CU partition technique in HEVC. Then, we give a brief review of the bagged tree method, followed by a short introduction of the ResNet model.

2.1. CU Partition of HEVC

To encode a video sequence, each frame among a sequence is divided into multiple non-overlapping squares of size 64 × 64. As the largest CU, a CU of size 64 × 64 is also called a CTU. As is known, a larger CU can save more semantic information, while smaller CUs can achieve more precise pixel prediction values. Thus, HEVC employs a recursive splitting process to traverse all the possible partition results of a CU. Figure 1 shows the recursive splitting process of a CTU, and this process terminates when it reaches the smallest CU, the size of which is usually configured before encoding, and defaults to 8 × 8. Therefore, there are, in total, four kinds of CUs, all of which have different sizes, i.e., 64 × 64, 32 × 32, 16 × 16, and 8 × 8, shown as squares of different colors in Figure 1.

According to a quad-tree structure, HEVC employs RDO to make the optimal partition decision for a CTU. The left part of Figure 2 shows a partition example of a CTU, and the right part of Figure 2 is the corresponding quad-tree structure.

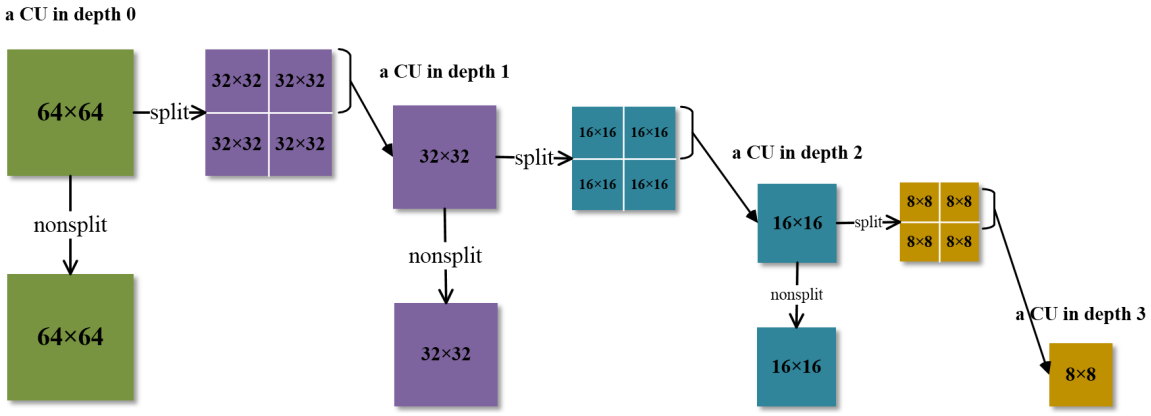

Figure 1. Recursive partition process of a CTU in the HEVC intra coding.

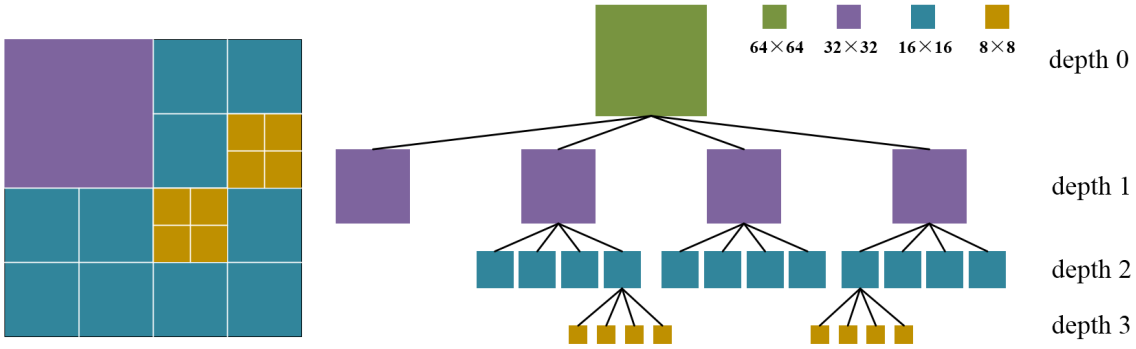

Figure 2. A partition example of a CTU and the corresponding quad-tree splitting structure in the HEVC intra coding.

2.2. Bagged Tree

Traditional machine learning methods have shown great advantages in classification tasks. As one of the most famous classical machine learning models, the decision tree is widely used in various application scenarios, such as image recognition and data mining. As its name suggests, the decision tree employs a tree structure to make a classification decision. For a classification task, each leaf node in a decision tree model represents a target class, and each parent node contains an attribute along with a threshold. In this way, the decision-making process becomes deeper, and the predicted labels are generated.

However, the decision tree is likely to overfit the training set. The bagging technique is introduced to address this problem. As a result, the bagged tree model is generated [47] as an ensemble of many decision tree classifiers, each of which is trained with a random subset of the training dataset. After training, the final prediction of a bagged tree model for an instance is made by taking the majority vote of the predictions from all individual decision trees of the input sample. Figure 3 shows the structure of a bagged tree model consisting of n decision trees, and also illustrates how the bagged tree generates the final prediction for an instance. Since the bagging technique decreases the variance of the model without increasing the bias, the performance of the bagged tree model is better than that of a single decision tree model. Therefore, in the proposed joint fast CU partition algorithm, the bagged tree is used in the first phase to predict the splitting result of a CTU.

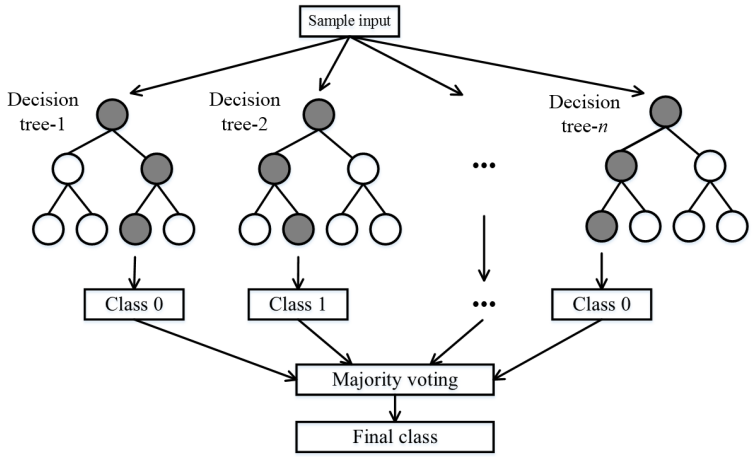

Figure 3. The structure of a bagged tree model. An illustration of how the final prediction for a sample input is generated by such a model, which is composed of multiple decision trees, each trained on a random subset of data.

One of the most important features of the bagged tree model is that the importance of features can be evaluated during the training process without having to repeatedly train

models with different feature combinations, as is required by most other feature-selection techniques. Caruana et al. [48] proposed a feature importance measurement technique, referred to as multiple counting, for a bagged tree model. In multiple counting, the importance value for every feature of a single decision tree model is calculated according to the number of data samples decided by this feature in all decision tree nodes. To compute the importance value of a feature for the bagged tree model, the importance values of a feature on the individual decision trees are summed and normalized by the feature entropy to ensure the comparability of features with different numbers of values. More details about the feature importance measurement technique are available in [48].

2.3. Residual Network

In recent years, the CNN has been a hot topic due to its excellent performance in solving image problems, such as image recognition and object detection. Compared with traditional machine learning methods, the CNN extracts effective features by using filters of various sizes instead of hand-crafted ones. Using the convolution, the CNN also reduces the massive parameters required in a neural network. However, when deeper networks start converging, previous research found that with the increment in network depth, the training error became higher, and the accuracy degraded rapidly [49].

To address this problem, He et al. [49] proposed ResNet using a deep residual learning framework. Figure 4a shows the structure of a basic building block in ResNet. With the introduction of a shortcut in Figure 4a, a deep network, constructed by many building blocks, can converge easily, and the degradation problem is solved. A ResNet is constructed by a stack of numbers of building blocks, as shown in Figure 4a, and He et al. [49] provided two kinds of such building blocks (Figure 4b,c), by which a well-performed ResNet of expected depth can be generated. The building block in Figure 4b is used to build a relative deep network (ResNet-34). The building block in Figure 4c, called "bottleneck", is used to stack ResNet-50/101/152, which represents ResNets containing 50, 101, and 152 layers, respectively.

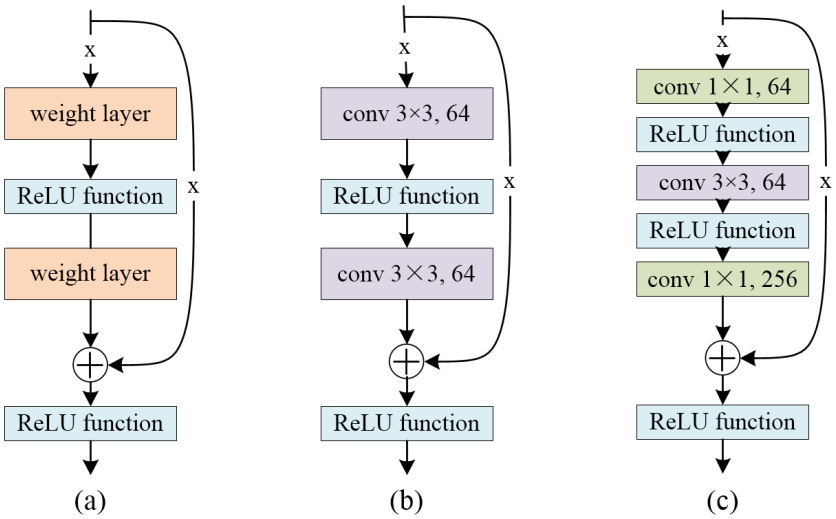

Figure 4. Structure of a building block in ResNet and two examples of a basic block for ResNet constructions of different depths [49]. (**a**) The basic structure with a shortcut for ResNet. (**b**) An example of a basic block used to build a shallow ResNet. (**c**) The "bottleneck" used to construct a deep ResNet by stacking.

3. The Proposed Bagged Tree and ResNet-Based Joint CTU Partition Method

In this section, we describe the proposed fast CTU partition algorithm for HEVC intra coding. First, we illustrate the overall process, which explains how the bagged tree model and the ResNet model work jointly. Then, effective features are designed to train the

bagged tree models. Furthermore, the architecture of the ResNet is designed to achieve a good prediction performance. Finally, the databases are constructed to train and validate our bagged tree and ResNet models. Using these databases, we train 20 bagged tree models (each is used for sequences under the same resolution and quantization parameter (QP) value). In addition, we also train four ResNet models with the same architecture, and each model is generated when a certain QP is set at different values, i.e., 22, 27, 32, and 37.

3.1. Flowchart of Our Fast Splitting Method

The flowchart of the proposed two-stage CTU partition algorithm is shown in Figure 5. As can be seen from Figure 5, there are roughly two phases in the proposed bagged tree and ResNet-based joint CTU partition algorithm.

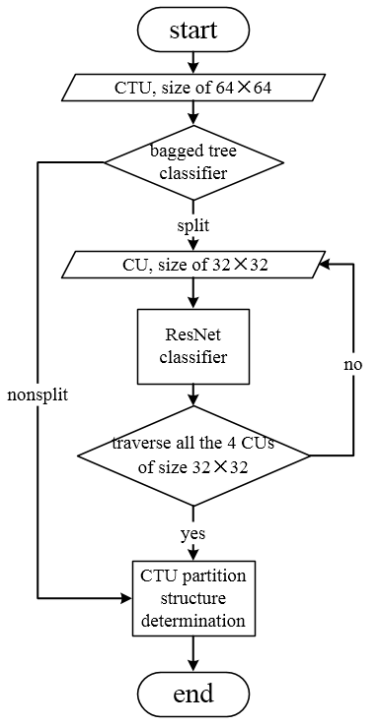

Figure 5. Flowchart of the proposed bagged tree and ResNet-based joint fast CTU partition algorithm.

In the first stage, a bagged tree-based classifier is used to predict the splitting status of a CTU, and the output is either split or nonsplit. Specifically, if a CTU is predicted to be of the nonsplit class, it goes to the CTU partition structure determination process directly, as shown in Figure 5, and the final partition is a whole 64 × 64 CTU without any splitting. On the other hand, if a CTU is predicted to be split by the bagged tree classifier, it will immediately be split into four sub-CUs with size 32 × 32. Sequentially, each of these four CUs will be passed to the second stage for further processing.

In the second stage, a 32 × 32-sized CU is fed to the finely trained ResNet classifier. As a result, of 17 labels, each of which represents a partition structure for a CU with size 32 × 32, one label is the output by ResNet. Similarly, all four sub-CUs of a CTU predicted to be split in the first stage are classified, and four corresponding predicted labels are generated. Then, these four labels, as well as the parent CTUs, are processed further.

A CTU partition structure determination process is adopted by following the two above-mentioned stages. According to the four labels generated by ResNet in stage 2, the partition structure of the corresponding parent CTU is decided. Figure 6 shows the flowchart of the proposed algorithm. The predicted label (PL) output by ResNet for a CU of size 32 × 32 is from 1 to 17. The corresponding partition quad-tree structure is illustrated in Figure 6, where the PL values of four sub-CUs (numbered 1, 2, 3, and 4) are assumed to be 1, 5, 17, and 2. Once

a partition of a CTU is generated, an optimal rate-distortion cost can be calculated directly without many comparisons. Then, an encoding process is executed.

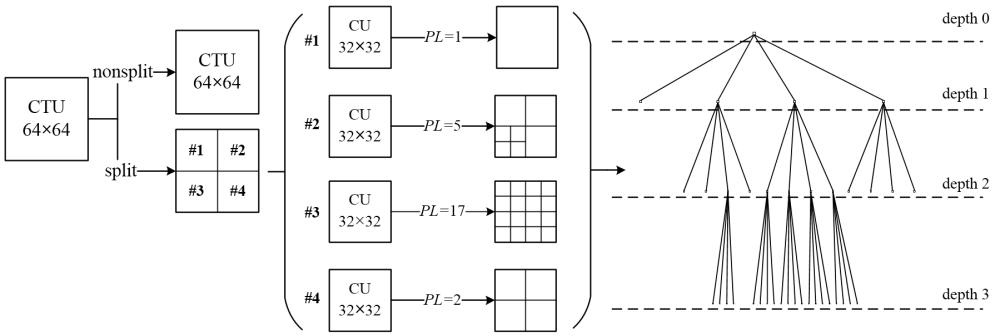

Figure 6. An example of CTU partition structure determination process with the corresponding partition quad-tree. *PL* is the predicted label for a 32 × 32 CU output by ResNet. *PL* is set as an integer from 1 to 17.

Compared with traditional recursive RDO, the proposed algorithm effectively skips the unnecessary search for different combinations of CU sizes. In addition, compared with other existing fast partition methods implemented on a depth level, this paper proposes a novel end-to-end two-stage CTU partition algorithm by using the bagged tree and ResNet techniques. In this way, the CTU partition result can be calculated through a well-trained bagged tree model and a fine-tuned ResNet model. As a result, we not only significantly reduce the time spent on RDO, but also spend less time on training the required learning models compared with existing works, which train as many as three or more models on a depth level.

3.2. Features Design for Bagged Tree Model

The traditional machine learning methods rely heavily on handcrafted features. Many existing works have developed a number of useful attributes for CU splitting label prediction. Based on these existing features, in our previous work [50], we also designed several novel features which were proven to be effective. As a result, in this paper, we select a total of 29 feature candidates used in [50], which are listed in Table 1 along with their corresponding meanings. These feature candidates are from four fields: information from neighboring CTUs, side information during the pre-encoding process, statistical information of CTU pixels, and information from pixel filtering results. We will give a brief introduction of these feature candidates in the following paragraphs, and more details are available in our previous work [50].

The results of [51] suggest that information from neighboring CTUs is useful for the decision making of current CU in regard to partitions. Thus, the features *m_nbCtuAboRd*, *m_nbCtuLefRd*, *m_nbCtuAblRd*, and *m_nbCtuAbrRd* are extracted, which represent the RD cost of the neighboring CTUs located at the the above, left, above-left and above-right regions of the current CTU, respectively. In addition, for a target CTU, *m_nbCtuAboDepth*, *m_nbCtuLefDepth*, *m_nbCtuAblDepth*, and *m_nbCtuAbrDepth* denote the average depth values of its above, left, above-left, and above-right neighbored CTUs, respectively.

In addition, some bypass results during the encoding process are also important and widely used [52]. Hence, we pre-encode the current CTU with PLANAR mode and use several useful encoding results as key features. As a result, the features *totalCost*, *totalDistortion*, and *totalBins* represent the total cost, the total distortion, and the number of bits under PLANAR mode, respectively. Furthermore, *m_aveCBF* is the coded block flag (CBF) of the current CTU encoded with PLANAR mode. Moreover, HEVC uses the Hadamard transformation to quickly estimate the encoding performance, so the encoding cost, the distortion, and the number of bits generated under Hadamard conditions are extracted as features, which are denoted as *m_costHadamard*, *m_sadHadamard*, and *m_bitsHadamard*, respectively.

Table 1. Feature candidates and corresponding descriptions.

Index	Feature Candidates	Feature Description
1	*m_varMeanSub*	variance of four sub-CUs' mean
2	*m_varVarSub*	variance of four sub-CUs' variance
3	*m_aveCBF*	average value of current CU's coded block flag
4	*m_nbCtuAboRd*	RD cost of current CU's above CTU
5	*m_nbCtuLefRd*	RD cost of current CU's left CTU
6	*m_nbCtuAblRd*	RD cost of current CU's above-left CTU
7	*m_nbCtuAbrRd*	RD cost of current CU's above-right CTU
8	*m_nbCtuAboDepth*	depth of current CU's above CTU
9	*m_nbCtuLefDepth*	depth of current CU's left CTU
10	*m_nbCtuAblDepth*	depth of current CU's above-left CTU
11	*m_nbCtuAbrDepth*	depth of current CU's above-right CTU
12	*totalCost*	total cost of current CU encoded with planar
13	*totalDistortion*	total distortion of current CU encoded with planar
14	*totalBins*	total bins of current CU encoded with planar
15	*m_costHadamard*	Hadamard cost of planar mode
16	*m_sadHadamard*	distortion of residual after Hadamard transfer
17	*m_bitsHadamard*	Hadamard bits of planar mode
18	*m_edgeSobel*	edge detection result using Sobel
19	*m_nmse*	mean square error of neighbor pixels
20	*m_dcom*	mean of gradients of four directions
21	*m_numInterestPoint*	number of interesting points of current CU
22	*m_haarSumx*	sum of horizontal value after Haar wavelet transfer
23	*m_haarSumy*	sum of vertical value after Haar wavelet transfer
24	*m_haarSumxy*	sum of diagonal value after Haar wavelet transfer
25	*m_haarSumAbsx*	sum of horizontal absolute value after Haar
26	*m_haarSumAbsy*	sum of vertical absolute value after Haar
27	*m_haarSumAbsxy*	sum of diagonal absolute value after Haar
28	*m_meanMain*	mean of current CU
29	*m_varMain*	variance of current CU

Some classic statistical data, such as mean, variance, and gradient, can reflect the content complexity of a CU and the difference among the four sub-CUs comprising one CU. Thus, for a target CTU, we calculate the mean and the variance of all pixel values as *m_meanMain* and *m_varMain*, respectively, which are calculated according to the following equations:

$$m\_meanMain = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{1}$$

$$m\_varMain = \frac{1}{n}\sum_{i=1}^{n} (p_i - m\_meanMain)^2 \tag{2}$$

where p_i is the luminance value of the ith pixel, and n is the number of pixels in the current CU.

Furthermore, the variance of the four sub-CU means is calculated and denoted as *m_varMeanSub*, and *m_varVarSub* is the variance of the four sub-CU variances. These values are calculated through the following equations:

$$m\_varMeanSub = \frac{1}{4}\sum_{i=1}^{4} (meanSub_i - m\_meanMain)^2 \tag{3}$$

$$m\_varVarSub = \frac{1}{4}\sum_{i=1}^{4} (varSub_i - meanVarSub)^2 \tag{4}$$

where $varSub_i$ represents the variance of luma pixels within the ith CU, and *meanVarSub* is the mean of all four sub-CU variances (i.e., $varSub_1$, $varSub_2$, $varSub_3$, and $varSub_4$).

In addition, we also reuse the feature *m_nmse*, which was designed and proved to be efficient in [25]. *m_nmse* is the mean of squared errors between each pixel and the mean of its eight neighboring pixels of a CTU. It is calculated as follows:

$$\bar{p}_{i,j} = \frac{1}{8}\begin{pmatrix} p_{i-1,j-1} + p_{i-1,j} + p_{i-1,j+1} \\ +p_{i,j-1} + p_{i,j+1} \\ +p_{i+1,j-1} + p_{i+1,j} + p_{i+1,j+1} \end{pmatrix} \tag{5}$$

$$m\_nmse = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(p_{i,j} - \bar{p}_{i,j}\right)^2 \tag{6}$$

where $p_{i,j}$ is the luma value of the pixel located at (i, j) in the current CU. N is the side length of a CU, which is 64 in the case of a CTU.

In addition, we also apply several kinds of filters to the pixels of a CTU and process these filter results to obtain useful features, which can reflect the pixel change, the content complexity, and even the edge information of a target CTU. As a result, *m_edgeSobel* and *m_dcom* are features calculated from the responses of Sobel filters, widely used in edge detection. Figure 7 shows four Sobel filters for detecting edges in different directions. Each filter is applied with overlap on every 3 × 3 square pixel block within the current CU. Figure 7a is an example of the 3 × 3 square pixel block. *m_edgeSobel* and *m_dcom* are calculated through the following equations:

$$g_h = -a - 2b - c + g + 2h + i \tag{7}$$

$$g_v = -a - 2d - g + c + 2f + i \tag{8}$$

$$g_{45} = 2a + b + d - f - h - 2i \tag{9}$$

$$g_{135} = b + 2c - d + f - 2g - h \tag{10}$$

$$m\_edgeSobel = \frac{1}{(N-2)^2}\sum_{k=0}^{(N-2)^2}\left(\left(g_h^k\right)^2 + \left(g_v^k\right)^2\right)^{1/2} \tag{11}$$

$$m\_dcom = \frac{1}{(N-2)^2}\sum_{k=0}^{(N-2)^2}\left(\left|g_h^k\right| + \left|g_v^k\right| + \left|g_{45}^k\right| + \left|g_{135}^k\right|\right) \tag{12}$$

where N is the pixel number of a CU along a side, which is 64 in the case of a CTU. k denotes the kth 3 × 3 square pixel block within the current CU.

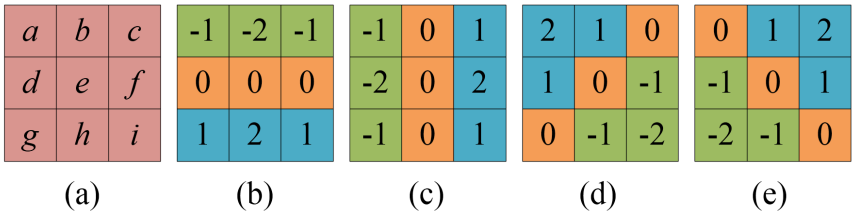

Figure 7. Sobel filters for different directions. (**a**) Example of a 3 × 3 square pixel block to be filtered by Sobel filters. (**b**) Sobel filter for horizontal direction. (**c**) Sobel filter for vertical direction. (**d**,**e**) Sobel filters for 45° and 135°, respectively.

As we can see, *m_edgeSobel* can reflect information about the horizontal and vertical edges, and *m_dcom* represents the comprehensive gradient information of the current CU from four directions. Moreover, Bay et al. [53] systematically analyzed the significant advantage of the Haar wavelet on the edge changing and scene shading of a picture. Therefore, we performed three Haar filters of different directions on the pixels in a target CTU. Figure 8b–d show the Haar filters of horizontal, vertical, and diagonal directions, respectively. Figure 8a is an example of a 2 × 2 square pixel block on which the Haar filters

are performed. Specifically, we first split a CTU into 2 × 2 non-overlapped sub-squares. Then, each sub-square was filtered by the same Haar filter to obtain the corresponding filter responses of a certain direction. The responses of a certain filter on a sub-square are calculated according to the following equations:

$$d_x = a + b - c - d \tag{13}$$

$$d_y = a - b + c - d \tag{14}$$

$$d_{xy} = a - b - c + d \tag{15}$$

where d_x, d_y, and d_{xy} are the filter responses of horizontal, vertical, and diagonal directions, respectively.

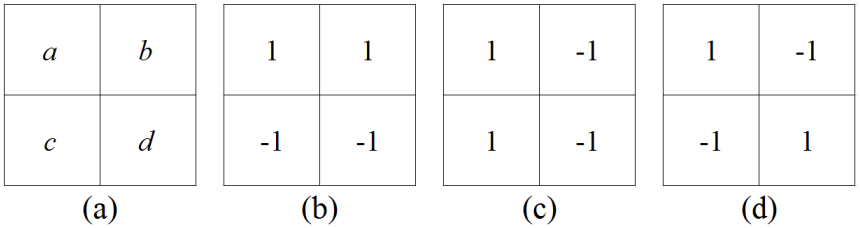

Figure 8. Haar filters for different directions. (**a**) Example of a 2 × 2 square pixel block to be filtered by Haar filters. (**b**) Haar filter for horizontal direction. (**c**) Haar filter for vertical direction. (**d**) Haar filter for diagonal direction.

Based on these responses, we can sum them to obtain the features *m_haarSumx*, *m_haarSumy*, *m_haarSumxy*, *m_haarSumAbsx*, *m_haarSumAbsy*, and *m_haarSumAbsxy*, according to the following equations:

$$m\_haarSumx = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} d_x^k \tag{16}$$

$$m\_haarSumy = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} d_y^k \tag{17}$$

$$m\_haarSumxy = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} d_{xy}^k \tag{18}$$

$$m\_haarSumAbsx = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} \left|d_x^k\right| \tag{19}$$

$$m\_haarSumAbsy = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} \left|d_y^k\right| \tag{20}$$

$$m\_haarSumAbsxy = \frac{1}{\left(\frac{N}{2}\right)^2} \sum_{k=1}^{\left(\frac{N}{2}\right)^2} \left|d_{xy}^k\right| \tag{21}$$

where N is the number of pixels of the current CU along one side, which is 64 in the case of a CTU.

Furthermore, the number of interest-related points within a picture can reflect people's visual perception, while its influence on the CU splitting goes unnoticed. So, we extract the interest-related points of the feature *m_numAveInterestPoint* for a target CTU by using a particular filter on a CTU. The feature *m_numAveInterestPoint* represents the average number of interest points for each pixel of a CTU. This reflects how much attention people will pay to a CTU and how many details a CTU contains. Three filters, as shown in Figure 9, are used on each of the current CTUs, and three corresponding results D_{xx}, D_{yy}, D_{xy} are obtained as the filter responds in the horizontal, vertical, and diagonal directions, respectively. We obtain the final value of the feature *m_numAveInterestPoint* for the current CTU by using the following equations:

$$P(i,j) = \left| D_{xx}D_{yy} - \left(0.9D_{xy}\right)^2 \right| \tag{22}$$

$$B(i,j) = \begin{cases} 0, P(i,j) < t \\ 1, P(i,j) \geq t \end{cases} \tag{23}$$

$$m\_numAveInterestPoint = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} B(i,j) \tag{24}$$

where $P(i,j)$ is the interest value of the pixel located at (i,j), and $B(i,j)$ is the Boolean value of the decision to create an interest point for pixel (i,j). t is the threshold to judge the interest point. N, the value of which is 64, represents the size of the current CTU. Because the original interest point detection method uses more complicated filters to obtain $P(i,j)$, which is much more time consuming, the relative weight of 0.9 is used to minimize the error between them.

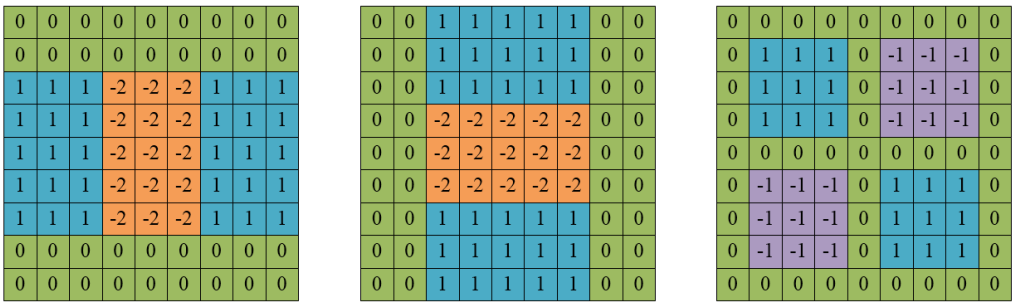

Figure 9. Three filters used for interest point detection in horizontal, vertical, and diagonal directions. (**a**) The filter used in the horizontal direction. (**b**) The filter used in the vertical direction. (**c**) The filter used in the diagonal direction. (**a**–**c**) are performed on each pixel of a CTU, and the corresponding filtering responses are D_{xx}, D_{yy}, and D_{xy}, respectively.

3.3. Features Selection for Bagged Tree Model

These 29 feature candidates, as shown in Table 1, are too numerous to be implemented. Hence, these candidates are ranked by the feature importance measurement technique introduced in Section 2.2 during the bagged tree training phase.

According to the ranking results, we find that the importance values are divided between the 10th and the 11th most-important features. In addition, the importance values of the 10 most-important features stay at a relatively high level, while the importance values for the rest of the feature candidates are generally very low. Thus, we pick the 10 most-important features as our final attributes. Hence, each bagged tree model has its own unique feature set consisting of the 10 most-important features.

Table 2 lists the top-10 features of the classifiers for the 5 resolution classes (i.e., A, B, C, D and E) and different QP values. The importance values of the corresponding features are

also shown in Table 2. These top 10 features of classifiers for different QPs and resolution classes are almost the same, while their rankings change.

Table 2. Ten most-important features selected for different QPs and resolution classes with corresponding importance values.

Resolution Class A								
	QP22		**QP27**		**QP32**		**QP37**	
Rank	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**
1	*m_aveCBF*	0.773	*m_aveCBF*	0.417	*m_aveCBF*	0.361	*var_var_sub*	0.255
2	*var_var_sub*	0.218	*var_var_sub*	0.292	*var_var_sub*	0.242	*m_aveCBF*	0.198
3	*meanMain*	0.128	*varMain*	0.078	*varMain*	0.107	*totalBins*	0.124
4	*varMain*	0.085	*m_numAveInterestPoint*	0.077	*m_haarSumxy*	0.078	*varMain*	0.083
5	*m_nbCtuAboDepth*	0.082	*meanMain*	0.066	*m_numAveInterestPoint*	0.066	*m_haarSumxy*	0.068
6	*m_haarSumy*	0.070	*m_nbCtuAblDepth*	0.063	*m_nbCtuAblDepth*	0.065	*m_haarSumx*	0.067
7	*m_numAveInterestPoint*	0.057	*m_haarSumx*	0.050	*m_haarSumAbsxy*	0.060	*m_numAveInterestPoint*	0.058
8	*m_haarSumx*	0.056	*m_nbCtuLefDepth*	0.042	*m_nbCtuAbrDepth*	0.057	*m_nbCtuAboDepth*	0.054
9	*m_haarSumxy*	0.055	*m_nbCtuAbrRd*	0.041	*meanMain*	0.049	*totalCost*	0.051
10	*m_haarSumAbsxy*	0.042	*m_haarSumxy*	0.039	*totalBins*	0.047	*m_haarSumy*	0.051
Resolution Class B								
	QP22		**QP27**		**QP32**		**QP37**	
Rank	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**
1	*m_aveCBF*	0.738	*m_aveCBF*	0.565	*m_aveCBF*	0.384	*m_nbCtuLefDepth*	0.282
2	*m_nbCtuLefDepth*	0.288	*m_nbCtuLefDepth*	0.213	*m_nbCtuLefDepth*	0.231	*m_aveCBF*	0.254
3	*m_nbCtuAboDepth*	0.150	*var_var_sub*	0.173	*var_var_sub*	0.208	*var_var_sub*	0.201
4	*var_var_sub*	0.078	*m_nbCtuAboDepth*	0.095	*m_nbCtuAboDepth*	0.119	*m_nbCtuAboDepth*	0.097
5	*m_nmse*	0.071	*totalBins*	0.076	*totalCost*	0.088	*totalBins*	0.084
6	*m_haarSumAbsx*	0.065	*m_haarSumAbsx*	0.060	*totalBins*	0.080	*varMain*	0.065
7	*m_haarSumx*	0.053	*m_numAveInterestPoint*	0.051	*m_numAveInterestPoint*	0.048	*totalDistortion*	0.058
8	*meanMain*	0.045	*totalCost*	0.049	*m_haarSumAbsx*	0.047	*m_nmse*	0.051
9	*totalDistortion*	0.045	*m_nmse*	0.045	*meanMain*	0.044	*meanMain*	0.048
10	*m_numAveInterestPoint*	0.042	*meanMain*	0.041	*m_nmse*	0.033	*m_numAveInterestPoint*	0.046
Resolution Class C								
	QP22		**QP27**		**QP32**		**QP37**	
Rank	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**
1	*m_aveCBF*	0.767	*m_aveCBF*	0.625	*m_aveCBF*	0.509	*m_aveCBF*	0.463
2	*var_var_sub*	0.164	*var_var_sub*	0.203	*var_var_sub*	0.226	*var_var_sub*	0.162
3	*m_nbCtuAboDepth*	0.073	*m_haarSumAbsy*	0.079	*m_haarSumAbsy*	0.094	*totalBins*	0.151
4	*m_nmse*	0.060	*m_nmse*	0.059	*totalBins*	0.093	*varMain*	0.112
5	*var_mean_sub*	0.039	*m_nbCtuAboDepth*	0.044	*totalCost*	0.064	*m_haarSumAbsy*	0.100
6	*meanMain*	0.036	*m_haarSumy*	0.040	*m_nmse*	0.060	*meanMain*	0.066
7	*varMain*	0.032	*totalBins*	0.031	*varMain*	0.040	*m_nmse*	0.060
8	*m_numAveInterestPoint*	0.023	*var_mean_sub*	0.030	*meanMain*	0.036	*m_numAveInterestPoint*	0.042
9	*totalDistortion*	0.021	*meanMain*	0.030	*m_numAveInterestPoint*	0.033	*m_haarSumAbsxy*	0.038
10	*m_haarSumAbsxy*	0.019	*varMain*	0.026	*totalDistortion*	0.030	*totalCost*	0.034
Resolution Class D								
	QP22		**QP27**		**QP32**		**QP37**	
Rank	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**
1	*m_aveCBF*	0.500	*m_aveCBF*	0.260	*var_var_sub*	0.173	*m_aveCBF*	0.204
2	*var_var_sub*	0.089	*var_var_sub*	0.158	*varMain*	0.067	*totalBins*	0.122
3	*varMain*	0.043	*varMain*	0.063	*m_nmse*	0.065	*var_var_sub*	0.115
4	*m_nmse*	0.016	*m_haarSumAbsx*	0.047	*m_haarSumAbsxy*	0.051	*totalCost*	0.107
5	*meanMain*	0.007	*m_dcom*	0.038	*m_aveCBF*	0.046	*meanMain*	0.055
6	*totalCost*	0.007	*totalBins*	0.035	*totalBins*	0.044	*m_nmse*	0.050
7	*m_costHadamard*	0.005	*var_mean_sub*	0.027	*totalCost*	0.027	*totalDistortion*	0.034
8	*m_sadHadamard*	0.005	*m_nmse*	0.024	*totalDistortion*	0.026	*m_haarSumAbsxy*	0.026
9	*m_numAveInterestPoint*	0.005	*m_numAveInterestPoint*	0.022	*m_haarSumAbsx*	0.018	*var_mean_sub*	0.025
10	*totalBins*	0.004	*m_haarSumAbsxy*	0.017	*m_sadHadamard*	0.016	*varMain*	0.022
Resolution Class E								
	QP22		**QP27**		**QP32**		**QP37**	
Rank	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**	**Feature**	**Importance**
1	*m_aveCBF*	0.363	*var_var_sub*	0.401	*var_var_sub*	0.413	*var_var_sub*	0.321
2	*var_var_sub*	0.262	*m_nbCtuLefDepth*	0.192	*m_nbCtuLefDepth*	0.220	*m_nbCtuLefDepth*	0.219
3	*m_nbCtuLefDepth*	0.104	*m_aveCBF*	0.188	*m_aveCBF*	0.201	*m_nbCtuAblDepth*	0.168
4	*meanMain*	0.090	*m_nbCtuAboDepth*	0.164	*meanMain*	0.099	*meanMain*	0.137
5	*m_bitsHadamard*	0.083	*m_nbCtuAbrDepth*	0.089	*totalBins*	0.083	*m_aveCBF*	0.121
6	*m_nbCtuAblRd*	0.078	*meanMain*	0.088	*m_nbCtuAboDepth*	0.060	*m_bitsHadamard*	0.100
7	*m_nbCtuAboRd*	0.059	*totalBins*	0.067	*m_nbCtuAblDepth*	0.055	*varMain*	0.079
8	*totalBins*	0.058	*m_numAveInterestPoint*	0.063	*m_numAveInterestPoint*	0.055	*m_nmse*	0.070
9	*m_nbCtuLefRd*	0.052	*m_nbCtuAbrRd*	0.061	*varMain*	0.052	*totalBins*	0.065
10	*m_haarSumy*	0.049	*varMain*	0.056	*totalDistortion*	0.048	*totalCost*	0.063

3.4. ResNet Model Designing and Training

Four ResNets with the same architecture are used, which dedicate video sequences under different QP values, i.e., 22, 27, 32, and 37, respectively. The ResNets used in this

work are based on the 34-layer ResNet in [49]. The specific architecture of the ResNet is shown in Figure 10. Each convolution block in Figure 10 represents a convolution process followed by a normalization process and a rectified linear unit (ReLU) calculation. The ResNet used in this work consists of 34 layers, and takes a luma CU of size 32 × 32 as the input. With a fully connected layer at the very end, the ResNet outputs a vector to describe the probability of a CU belonging to each class. Using this probability vector and the ground truth vector of the target CU, the cross entropy loss is calculated according to the following equation:

$$\mathcal{L} = \text{loss}(V_1, V_2) = -\log\left(\frac{\exp(V_1 \times V_2)}{\sum_{i=1}^{17} \exp(V_1 \times i)}\right) \tag{25}$$

where V_1 is the probability vector output by ResNet, which contains the probabilities of a CU belonging to every class; V_2 is the class label ground truth of a CU, in the form of one-hot encoding; and $\mathcal{L}$ is the cross entropy loss between V_1 and V_2. The cross entropy loss is used to measure the difference between the predicted label and the ground truth of the 32 × 32 CU. Since it is a convex function, the global minimum can be easily calculated by taking a partial derivative. Generally, training the ResNet model involves finding suitable parameters to minimize the cross entropy loss of all the training samples.

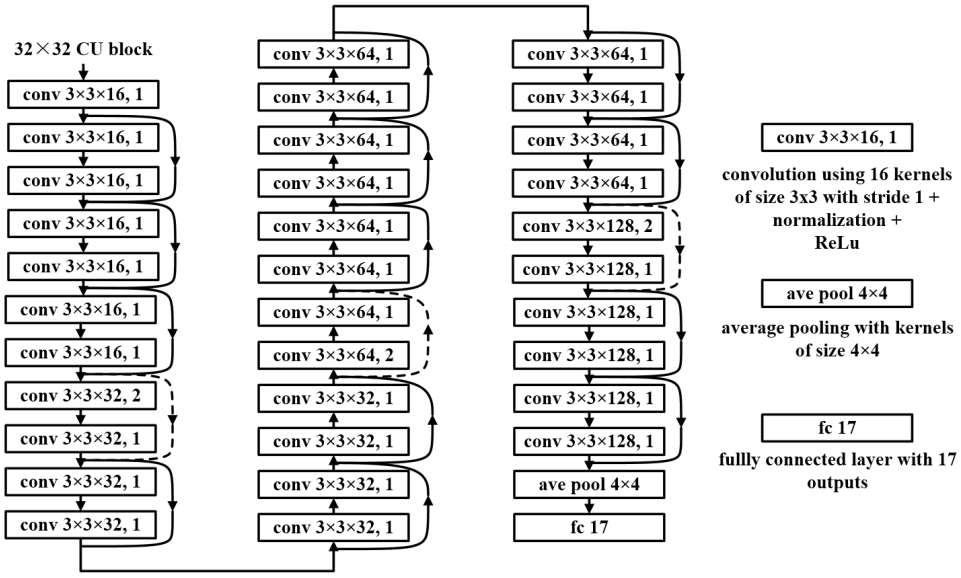

Figure 10. Architecture of the ResNet used in this paper. The dotted lines represent the shortcuts which increase the dimensions.

We use the deep learning framework PyTorch to train the ResNets in this paper. The Adam optimizer is used with a learning rate of 0.01. Specifically, the models are trained on 40 epochs with a batch size of 32.

3.5. Database Generation for Training and Validation

In this paper, the bagged tree and the ResNet are used jointly to make an end-to-end decision of the partition structure for a CTU. Specifically, the bagged tree is used to make the splitting decision for a 64 × 64 CU, while the ResNet is used to determine the partition structure of a 32 × 32 CU through a single prediction. Thus, two kinds of databases are needed. One is constructed for the bagged tree model and the other is constructed for the ResNet model.

Because resolution and QP both have a strong influence on the splitting of a CU, we cannot train a prediction model without considering them. In this paper, we train one

bagged tree model for each resolution and each QP value, and train one ResNet model for each QP value. In practical applications, people can pick specific models according to their own encoding scenarios. It would be ideal to train a general model for every situation, but it is difficult to maintain a high encoding performance while achieving generality, and a much bigger training set is required, including all possible resolutions and QPs. In addition, the strategy of training a single model for a certain condition can improve the prediction accuracy in order to achieve a better encoding performance. Moreover, it provides people with more specific solutions according to their own encoding tasks.

First, five sequences of different resolutions are picked from the standard test sequences established by JCT-VC [54] as training sequences: *Traffic* (2560 × 2600), *ParkScene* (1920 × 1080), *BasketballDrill* (832 × 480), *BQSquare* (416 × 240), and *FourPeople* (1280 × 720). Then, these five sequences are encoded by HM16.7 under all intra configurations but four QPs (22, 27, 32, and 37).

As for the databases used to train bagged tree models, we extract 10 key features of each CTU among a training sequence under a particular QP and resolution class. In addition, the splitting label of a CTU is also extracted to form the training database. So, there are 20 databases in total, each of which is constructed for a QP value and a resolution class, as shown in Table 3. DB_{xy} represents the database used to train the resolution class x and QP y bagged tree model. For example, DB_{A22} is the database constructed to train the bagged tree model, which is used to make the splitting decision of the CTUs in a sequence under resolution A and QP 22.

Table 3. Databases for bagged tree models of each QP and resolution class.

Resolution Class	QP22	QP27	QP32	QP37
A (2560 × 1600)	DB_{A22}	DB_{A27}	DB_{A32}	DB_{A37}
B (1920 × 1080)	DB_{B22}	DB_{B27}	DB_{B32}	DB_{B37}
C (832 × 480)	DB_{C22}	DB_{C27}	DB_{C32}	DB_{C37}
D (416 × 240)	DB_{D22}	DB_{D27}	DB_{D32}	DB_{D37}
E (1280 × 720)	DB_{E22}	DB_{E27}	DB_{E32}	DB_{E37}

Specifically, to form the database used to train the bagged tree models, 36,000 CTU samples are randomly picked from the training sequences encoded by HM16.7. To balance the database, the ratio between the samples of split and nonsplit labels in each database is 1. When there are not enough samples in the whole encoded sequence, all of the nonsplit samples are picked, and the same number of split samples is randomly picked.

As for the databases constructed to train the ResNet, due to the end-to-end prediction structure, each partition structure of a 32 × 32 CU corresponds to a class label. There are 17 classes in total, and these classes, together with the corresponding partition structures, are shown in Figure 11. As we can see from Figure 11, these classes cover all the possible partition results for a CU of size 32 × 32. In this way, a partition structure can be predicted by ResNet through a single prediction.

In the databases for ResNet, each sample includes the luma values of a 32 × 32 CU and its ground truth label according to the encoding results. To form the database, all 32 × 32 CUs in a training sequence are collected. Different from the 20 bagged tree models, each ResNet model is designed for the sequences encoded under a certain QP, so four ResNet models are required in total. Hence, four databases are generated (denoted as DB-22, DB-27, DB-32, and DB-37), and constructed for the sequences encoded with QP 22, 27, 32, and 37, respectively. Taking the generation of DB-22, for example, we firstly encode five training sequences with QP 22. Then, the luma values and class labels of 32 × 32 CUs from these five sequences are all collected to obtain DB-22. Similarly, DB-27, DB-32, and DB-37 are generated. It is worth noting that the difference among these four databases lies only in the

labels. The luma pixel values between different databases are the same, because they are all from the same training sequences.

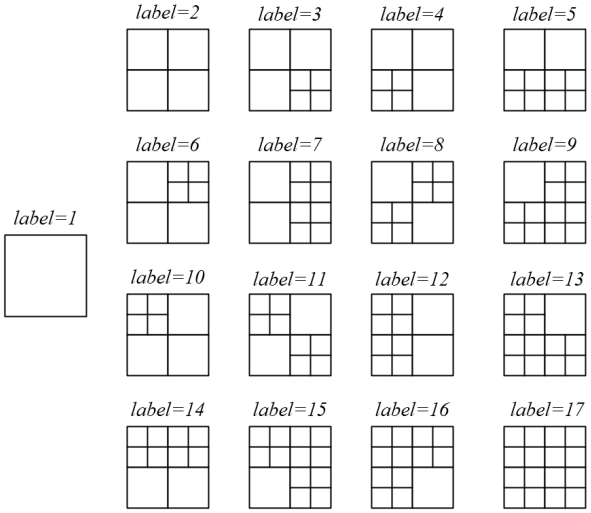

Figure 11. Class labels and corresponding partition structures of a 32 × 32 CU for ResNet multi-classification. Variate *label* is the class index of different partitions for a CU of size 32 × 32.

After all databases are generated, each is divided into two parts: the training datasets and validation datasets. The number of samples in the training dataset accounts for 90% of the total number of samples, and samples in the validation dataset account for 10%. In this paper, chroma CUs are split in the same manner as luma CUs. Hence, the models and datasets for chroma are not considered.

4. Experimental Results

In order to verify the performance of the proposed bagged tree and ResNet joint fast algorithm (BTRNFA), we implemented this algorithm in the HEVC reference encoder HM16.7. The original software HM16.7 is available online [55]. In addition, the deep learning framework PyTorch is used to perform the training and the prediction of ResNet models. Test sequences were selected from the HEVC common test conditions (CTC) [54].

Experiments were conducted on an Aliyun host, which has a Windows 64-bit operating system. The host has four kernels of Intel(R) Xeon(R) Platinum 8269CY CPU @ 2.50 GHz, 3.10 GHz with a 16 GB memory, and is also equipped with an NVIDIA RTX 2080s GPU. Coding parameters were set as the default, and all results were generated under the all-intra main configuration. The number of frames to be encoded was set as the maximal value for each sequence according to the CTC. When HM16.7 begins to encode a video sequence, the python code of ResNet starts running on the GPU to predict the partition structures of all frames in the current video sequence. Due to the computing capability of a GPU, the ResNet outputs the partition structure of a CTU before the encoder begins to encode it. We sum the inference time of ResNet and the time spent by the encoder together as the final encoding time of the proposed algorithm.

There are three versions of the proposed BTRNFA. The first version is denoted as $\text{BTRNFA}_{\text{DT}}$, in which only the bagged tree model is active. In this situation, the CUs of depth 0 are partitioned according to the predicted labels output by a bagged tree model, and CUs of depth 1 and 2 are processed by RDO. The second version is $\text{BTRNFA}_{\text{ResNet}}$, in which only the ResNet model for the CUs of depths 1 and 2 is active. RDO is performed on CUs of depth 0. The third version is $\text{BTRNFA}_{\text{joint}}$, which is also the most aggressive version. In the $\text{BTRNFA}_{\text{joint}}$, both the bagged tree model and the ResNet model are activated. As a result, $\text{BTRNFA}_{\text{joint}}$ can achieve the greatest reduction in the encoding time.

First, $\text{BTRNFA}_{\text{DT}}$, $\text{BTRNFA}_{\text{ResNet}}$, and $\text{BTRNFA}_{\text{joint}}$ were carried out for performance comparison. We analyzed the contributions of the bagged tree model and the ResNet model

on the proposed joint algorithm. In addition, we also compared the $\text{BTRNFA}_{\text{joint}}$ algorithm with the CNN-based state-of-the-art CU depth decision algorithms, i.e., the CU partition mode decision algorithm (PMDA) in [43] and the complexity reduction method (CRM) in [45]. To measure the rate-distortion performance, the BD-rate was calculated. The fast algorithm usually leads to an increase in the BD-rate, and the BD-rates of all sequences are averaged to reflect an overall encoding quality. The greater the increase in BD-rate is, the worse the encoding quality. Complexity reduction was measured by the savings in encoding time (denoted as TS), which was calculated according to the following equation:

$$TS = \frac{time_{ref} - time_{pro}}{time_{ref}} \times 100\% \tag{26}$$

where $time_{pro}$ is the sequence encoding time of the proposed algorithm, and $time_{ref}$ is the encoding time spent by the reference encoder. Encoding time is calculated by the function "clock", with the standard header "time.h" for C, which measures the time the encoding process takes from start to finish. The referring time of ResNet is measured in the Python code by using Python package "time". $time_{pro}$ equals the sum of the referring time of ResNet and the encoding time outputted by the encoder.

4.1. Performance of Three Versions of The Proposed Algorithm

In this section, we discuss the BD-rate loss and time-saving performance of three versions of the proposed BTRNFA (i.e., $\text{BTRNFA}_{\text{DT}}$, $\text{BTRNFA}_{\text{ResNet}}$, and $\text{BTRNFA}_{\text{joint}}$). The experimental results of each version are shown in Table 4.

We can see from Table 4 that $\text{BTRNFA}_{\text{DT}}$ achieves a 0.25% BD-rate loss with an encoding time reduction of 22.37% on average. Its BD-rate loss is the smallest among the three versions of the proposed algorithm, as are its encoding time savings. The average BD-rate loss of 0.25% shows that our bagged tree model is trained well and has a high prediction accuracy in test sequences. However, the time-savings results are not the highest, because only the bagged tree model is activated in this version of the proposed algorithm, and the encoding complexity among the CU partitioning of depths 1 and 2 is not considered reduced.

On the contrary, the bagged tree model is disabled in $\text{BTRNFA}_{\text{ResNet}}$, and only the ResNet model is active. In this case, RDO is performed on each CTU while the partition structure of CUs with depth 1 is generated by the end-to-end prediction. This version of the proposed algorithm achieves a BD-rate loss of 1.81% with a time saving of 49.83% on average, compared with the original algorithm in HM16.7. We can see from Table 4 that, compared with $\text{BTRNFA}_{\text{DT}}$, the time savings of $\text{BTRNFA}_{\text{ResNet}}$ increase by about 27.46%, with a 1.56% sacrifice in terms of BD-rate. The results given in Table 4 show that the prediction accuracy of ResNet is not quite so high that further BD-rate loss is caused. On the other hand, we can also see that more complexity can be saved with CU depths 1 and 2 than that in a CU with depth 0.

$\text{BTRNFA}_{\text{joint}}$ is the most aggressive version, in which both the bagged tree model for CTU and the ResNet model for CU in depth 1 are activated. Its results are shown in the last two columns of Table 4. $\text{BTRNFA}_{\text{joint}}$ achieves a complexity reduction as great as 60.29%, with a BD-rate loss of only 2.03%. Since it has the best time-savings results of the three algorithm versions, its BD-rate loss performance is acceptable.

As we can see, the performance of the proposed $\text{BTRNFA}_{\text{joint}}$ varies with the sequences listed in Table 4. To determine the kinds of videos suitable for $\text{BTRNFA}_{\text{joint}}$, such that good results can be obtained, we use the ratio of time savings versus BD-rate loss to conduct further analysis. As shown in Table 4, $\text{BTRNFA}_{\text{joint}}$ performs better in videos of resolution class C and D than those of A, B, and E, which means our method has better performance in low-resolution videos, which may be due to the high prediction accuracy of CTUs in low-resolution videos. In addition, observing performance differences on sequences of the same resolution class, we can see that *BQTerrace* shows better results than *BasketballDrive*. Considering BD-rate and time savings jointly, *PartyScene* and *BlowingBubbles* outperform

BQMalll and *BasketballPass*, respectively. It is easy to find the sequences *BasketballDrive*, *BQMalll*, and *BasketballPass*, which all contain multiple fast-moving objects in a scene. Sequences which contain much still or slow-moving content allow our method to achieve good results easily. Thus, we can conclude that low-resolution videos with less moving content are suitable for the proposed method.

Table 4. BD-rate loss and time-saving performance of three versions of the proposed BTRNFA.

Class	Sequence	$\text{BTRNFA}_{\text{DT}}$		$\text{BTRNFA}_{\text{ResNet}}$		$\text{BTRNFA}_{\text{joint}}$	
		BD-Rate (%)	**TS (%)**	**BD-Rate (%)**	**TS (%)**	**BD-Rate (%)**	**TS (%)**
A	*PeopleOnStreet*	0.16	17.21	2.19	51.85	2.28	62.21
	Average	0.16	17.21	2.19	51.85	2.28	62.21
B	*Kimono*	1.12	55.06	1.66	68.65	2.50	79.95
	Cactus	0.34	21.91	2.39	56.13	2.66	66.62
	BasketballDrive	0.98	32.31	3.12	62.14	3.83	73.16
	BQTerrace	0.11	20.24	1.68	52.62	1.77	62.89
	Average	0.64	32.38	2.21	59.88	2.69	70.66
C	*BQMall*	0.04	14.52	1.73	45.14	1.83	55.16
	PartyScene	0.00	10.79	0.61	33.08	0.62	42.80
	RaceHorses	0.08	18.65	1.38	49.70	1.51	60.50
	Average	0.04	14.65	1.24	42.64	1.32	52.82
D	*BasketballPass*	0.01	10.17	1.19	38.84	1.23	46.82
	BlowingBubbles	0.00	8.90	0.50	27.16	0.50	34.59
	RaceHorses	0.01	10.50	0.90	36.06	0.92	44.17
	Average	0.01	9.86	0.86	34.02	0.88	41.86
E	*Johnny*	0.24	39.92	3.24	64.64	3.54	75.59
	KristenAndSara	0.19	30.62	2.89	61.73	3.07	72.80
	Average	0.21	35.27	3.07	63.19	3.30	74.19
F	*BasketballDrillText*	0.00	11.22	1.42	43.85	1.46	53.99
	SlideEditing	0.16	17.47	2.11	57.36	2.37	67.39
	SlideShow	0.25	22.18	2.27	55.94	2.45	65.94
	Average	0.41	16.96	1.93	52.38	2.09	62.44
Overall Average		0.23	21.35	1.83	50.30	2.03	60.29

As for the performance of our method on other QP settings, on the one hand, we can use the current models to directly encode videos under other QPs. This does not affect the RD performance, but only affects the benefit of time savings. On the other hand, we can train one model for a particular QP. This will obviously result in a better RD performance and more time savings, since it is targeted to the QP. In short, experiments on existing QPs have verified the effectiveness of our algorithm. For other QPs, our method can achieve a better trade-off between RD performance and time savings by training the corresponding models.

4.2. Prediction Accuracy and Inference Overhead of the Proposed Method

To analyze the performance of the proposed $\text{BTRNFA}_{\text{joint}}$ at each stage, we present the prediction accuracy of the bagged tree and ResNet models, as well as the overhead GPU inference time.

Firstly, Table 5 shows the prediction accuracy of different bagged tree models for CUs in depth 0. The average accuracy of these 20 bagged tree models is 89.86% on the splitting prediction for CUs in depth 0. The prediction accuracy is quite stable, at around 90% for four QP values, which can be observed from the last row of Table 5. In addition, we find these four bagged tree models trained for resolution class A all perform below average, perhaps because the video sequence used for training is not representative.

Table 5. Prediction accuracy of different bagged tree models.

Resolution Class	QP22	QP27	QP32	QP37
A (2560 × 1600)	90.66%	94.13%	94.09%	94.79%
B (1920 × 1080)	79.21%	77.15%	78.25%	78.95%
C (832 × 480)	95.15%	95.44%	96.42%	96.44%
D (416 × 240)	98.83%	98.78%	98.12%	96.91%
E (1280 × 720)	66.11%	83.50%	91.99%	92.26%
Average	85.99%	89.80%	91.78%	91.87%

In addition, from different levels, Table 6 presents the prediction accuracy of four ResNets. On the overall prediction level, four ResNets show quite stable accuracy performances, which are distributed between 56% and 58%. Although the overall accuracy is low, their accuracy on different depth levels is quite good. In Table 6, the prediction level of depth 1 refers to the split flag prediction (i.e., split or non-split) of CUs in depth 1. Furthermore, the prediction level depth 2 refers to the splitting decision prediction for CUs in depth 2, which is also a binary decision. As we can see from depth level 1 of Table 6, the highest split flag prediction accuracy for CU in depth 1 is 89.34% of the ResNet trained for QP 27. Even the lowest split flag prediction accuracy for CU in depth 1 is as high as 86.43%. Furthermore, the accuracy of four ResNets on depth level 2 is similar to that of depth level 1. It can be observed that, although the overall prediction accuracy of four ResNet models is not high, the prediction accuracy of the splitting decision at each depth level is quite high. This is because the difference between classes is not obvious in the multi-classification task for CU partitioning, and a slight prediction error will not lead to the splitting decision error at each depth level. The high prediction accuracy of ResNets at each depth level also verifies the correctness of our algorithm.

Table 6. Prediction accuracy from different levels of four ResNets.

Prediction Level	QP22	QP27	QP32	QP37
Overall	57.20%	58.86%	58.12%	56.51%
Depth 1	86.43%	89.34%	88.68%	86.82%
Depth 2	85.36%	86.51%	86.76%	86.64%

This paper aims to reduce the encoding complexity by deciding the partitioning structure of a CTU early, so the complexity introduced by the ResNet model must be considered. As a result, an appropriate network structure, as shown in Figure 10, is designed, and the encoding time of the proposed method includes the corresponding inference time, which is shown in Table 7. It can be observed from Table 7 that the inference overhead of four ResNets is about 1% of the original HM encoder. It is worth noting that the inference process is carried out on a GPU, so the consumption is very low.

Table 7. Inference overhead compared with the original HM16.7 encoding time.

	QP22	QP27	QP32	QP37
Percentage	0.85%	1.00%	1.12%	1.24%

4.3. Performance Comparison with CNN-Based State-of-the-Art Algorithms

This section firstly compares the encoding performance between the proposed $\text{BTRNFA}_{\text{joint}}$ and two other CNN-based intra mode fast decision algorithms: PMDA and CRM. These two algorithms, compared to $\text{BTRNFA}_{\text{joint}}$, are implemented on different versions of reference software. In order to make a fair comparison, we apply the proposed $\text{BTRNFA}_{\text{joint}}$

to these reference software versions of the two comparison objects, respectively. The corresponding results are presented in the following paragraphs.

The PMDA in [43] reduces the hardware complexity of the encoder by decreasing more than two CTU partition modes for RDO. It achieves a 60.86% complexity reduction on average, and the BD-rate loss increases by as much as 2.74% on HM12.0. Compared with their results, the proposed BTRNFA$_{joint}$ saves about 0.73% in terms of BD-rate loss, with a negligible encoding time increment of 0.54% on average. The detailed results shown in Table 8 indicate that BTRNFA$_{joint}$ exceeds PMDA in BD-rate with a similar time savings.

Table 8. Performance comparison between the proposed BTRNFA$_{joint}$ and PMDA under HM12.0; all intra coding.

Class	Sequence	PMDA		BTRNFA$_{joint}$	
		BD-Rate (%)	**TS (%)**	**BD-Rate (%)**	**TS (%)**
A 2560 × 1600	*PeopleOnStreet*	2.27	61.80	2.29	63.30
	Average	2.27	61.80	2.29	63.30
B 1920 × 1080	*Kimono*	2.47	62.60	2.51	79.68
	Cactus	2.59	60.40	2.65	66.73
	BasketballDrive	3.49	70.90	3.84	72.23
	BQTerrace	2.26	62.40	1.77	61.83
	Average	2.70	64.08	2.69	70.12
C 832 × 480	*BQMalll*	2.93	57.80	1.82	56.26
	PartyScene	2.19	51.10	0.61	44.59
	RaceHorses	2.08	56.20	1.50	61.09
	Average	2.40	55.03	1.31	53.98
D 416 × 240	*BasketballPass*	2.89	59.30	1.20	48.24
	BlowingBubbles	2.54	53.60	0.49	36.27
	RaceHorses	2.43	53.60	0.89	45.26
	Average	2.62	55.50	0.86	43.26
E 1280 × 720	*Johnny*	4.42	72.20	3.55	75.40
	KristenAndSara	3.12	69.30	3.06	73.25
	Average	3.77	70.75	3.30	74.33
Overall Average		2.74	60.86	2.01	60.32

As for the encoding complexity-reducing approach, CRM, in [45], an early terminated hierarchical CNN is proposed to decide the CU partitioning. The encoding complexity of the HEVC intra mode decision is dramatically reduced by replacing the RDO with the predicted partitioning result. On average, 61.91% of the encoding complexity is saved with 2.24% of the BD-rate loss increment, compared with the original HM16.5. Comparison results are shown in Table 9. Compared with CRM, the proposed BTRNFA$_{joint}$ achieves 0.22% less BD-rate loss, at 2.02%, with only 2.60% more encoding time. In addition, only one ResNet model is required in the BTRNFA$_{joint}$, while the number of CNN models used in CRM is as many as three, which means that the proposed algorithm has advantages no matter the model training or encoder implementation.

Generally speaking, the proposed BTRNFA$_{joint}$ outperforms two CNN-based state-of-the-art algorithms. On average, an additional 0.73% and 0.22% in the BD-rate is saved, compared with PMDA and CRM, respectively. Although the time savings of BTRNFA$_{joint}$ are very similar, our approach shows an advance in terms of the model number required in encoding by introducing an end-to-end partitioning solution. Thus, the complexity of model training and implementation is quite low and is competitive for the proposed BTRNFA$_{joint}$.

In addition, we also compare our work with two other machine-learning-based methods, RDNet and TCA, in terms of encoding performance. RDNet was proposed by Yao et al. [56] using a CNN network named RDNet. TCA was also a CNN-based CU partition approach, proposed by Zhang et al. [57]. RDNet and TCA both employ deep learning as their key techniques, which is necessary to carry out the comparisons. The encoding results of each algorithm are shown in Table 10. The proposed algorithm BTRNFA$_{joint}$ outperforms RDNet, in terms of encoding time savings, by 5.29%, while the BD-rate of our approach slightly increases,

to about 0.19% more than that of RDNet. Our algorithm may not defeat RDNet thoroughly, but it is competitive. Compared with TCA, $BTRNFA_{joint}$ performs well. Their encoding performances are quite close. Considering the overall average, TCA causes as little as 0.1% less BD-rate loss compared with our method, and displays 2.14% more time savings. As for performances on the sequence in class A, the differences between TCA and $BTRNFA_{joint}$ are negligible. In addition, $BTRNFA_{joint}$ causes less BD-rate loss on class D and TCA causes less BD-rate loss on class B. They both have resolution categories that they excel in. It is worth noting that $BTRNFA_{joint}$ has an advantage in terms of the number of models when encoding a video sequence. TCA uses a three-stage strategy, such that it has to employ three CNN models while encoding a video sequence. $BTRNFA_{joint}$ only needs one bagged tree model and a ResNet model due to the two-stage strategy.

Table 9. Performance comparison between the proposed $BTRNFA_{joint}$ and CRM under HM16.5; all intra coding.

Class	Sequence	CRM		$BTRNFA_{joint}$	
		BD-Rate (%)	TS (%)	BD-Rate (%)	TS (%)
A 2560 × 1600	*PeopleOnStreet*	2.37	61.00	2.28	62.08
	Average	2.37	61.00	2.28	62.08
B 1920 × 1080	*Kimono*	2.59	83.54	2.53	79.78
	Cactus	2.27	60.96	2.65	65.98
	BasketballDrive	4.27	76.32	3.84	71.71
	BQTerrace	1.84	64.72	1.77	60.73
	Average	2.74	71.39	2.69	69.55
C 832 × 480	*BQMalll*	2.09	58.42	1.82	54.80
	PartyScene	0.66	44.50	0.61	42.48
	RaceHorses	1.97	57.12	1.50	60.21
	Average	1.57	53.35	1.31	52.50
D 416 × 240	*BasketballPass*	1.84	56.42	1.25	46.79
	BlowingBubbles	0.62	40.54	0.50	34.43
	RaceHorses	1.32	55.76	0.93	44.11
	Average	1.26	50.91	0.89	41.78
E 1280 × 720	*Johnny*	3.82	70.68	3.55	75.15
	KristenAndSara	3.46	74.86	3.05	72.74
	Average	3.64	72.77	3.30	73.95
Overall Average		2.24	61.91	2.02	59.31

Table 10. Encoding performance comparison with RDNet and TCA.

Class	Sequence	RDNet		TCA		$BTRNFA_{joint}$	
		BD-Rate (%)	TS (%)	BD-Rate (%)	TS (%)	BD-Rate (%)	TS (%)
A 2560 × 1600	*PeopleOnStreet*	2.20	57.53	2.29	63.12	2.28	62.21
	Average	2.20	57.53	2.29	63.12	2.28	62.21
B 1920 × 1080	*Kimono*	1.40	83.53	1.93	69.17	2.50	79.95
	Cactus	1.95	52.72	2.18	59.40	2.66	66.62
	BasketballDrive	3.94	74.29	3.58	65.08	3.83	73.16
	BQTerrace	1.19	47.96	1.48	63.26	1.77	62.89
	Average	2.12	64.63	2.29	64.23	2.69	70.66
C 832 × 480	*BQMall*	1.33	33.08	1.15	55.02	1.83	55.16
	PartyScene	0.36	33.66	1.07	57.15	0.62	42.80
	RaceHorses	1.66	36.28	1.43	61.02	1.51	60.50
	Average	1.12	34.34	1.22	57.73	1.32	52.82
D 416 × 240	*BasketballPass*	1.85	57.06	1.79	59.08	1.23	46.82
	BlowingBubbles	0.85	37.87	0.58	58.01	0.50	34.59
	RaceHorses	0.98	42.99	1.15	59.04	0.92	44.17
	Average	1.23	45.97	1.17	58.71	0.88	41.86
E 1280 × 720	*Johnny*	3.42	77.55	3.42	66.32	3.54	75.59
	KristenAndSara	2.66	74.00	2.91	69.45	3.07	72.80
	Average	3.04	75.78	3.17	67.89	3.30	74.19
Overall Average		1.83	54.50	1.92	61.93	2.02	59.79

4.4. Comparisons of CU Partition Results

To compare the differences in the CU partition results between the original algorithm in HM16.7 and the proposed BTRNFA$_{\text{joint}}$ algorithm, we encode the 180th frame of the sequence *RaceHorses* (416 × 240) with QP 22 by using these two algorithms. Figure 12 presents the CU partition results of the original HM16.7. In Figure 12, the black line represents the CU boundaries. Figure 13 shows the CU partition results of the proposed BTRNFA$_{\text{joint}}$. In Figure 13, we use the gold and red lines to represent the differences of the CU boundaries between the original HM16.7 and the BTRNFA$_{\text{joint}}$.

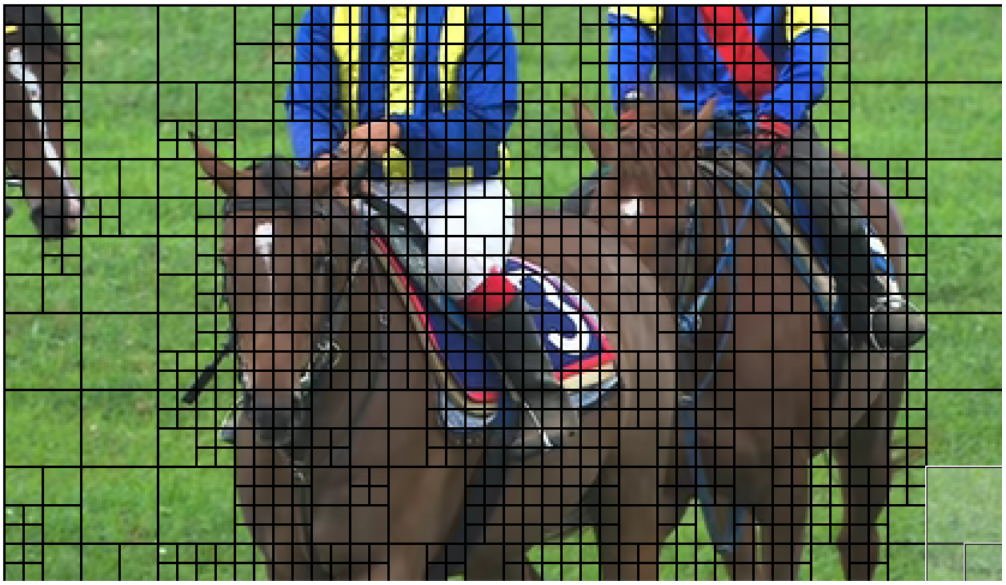

Figure 12. Partition results of the 180th frame in the sequence *RaceHorses* (416 × 240), which is encoded by the original HM 16.7 with QP 22. The black line represents the splitting boundaries between CUs.

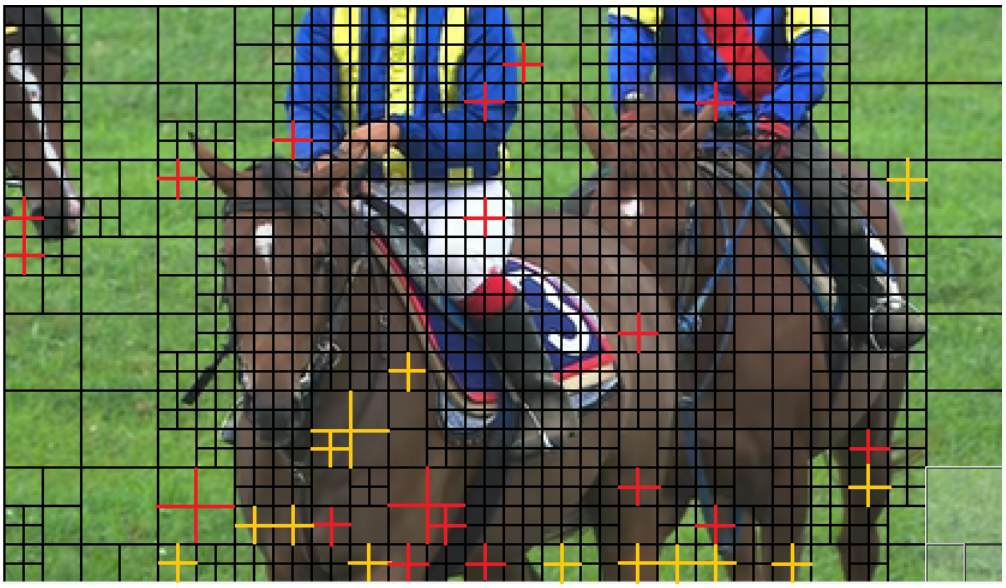

Figure 13. Partition results of the 180th frame in the sequence *RaceHorses* (416 × 240), which is encoded by the third version BTRNFA$_{\text{joint}}$ of the proposed algorithm with QP 22. The black line represents the same partition results as those of the original HM16.7. The gold line represents the boundaries of CUs, which are split by the original HM16.7 but are not split by the BTRNFA$_{\text{joint}}$. The boundaries of CUs, which are non-split by the original HM16.7 but are split by the BTRNFA$_{\text{joint}}$, are shown with a red line.

It can be observed from Figure 13 that the same CU partition results take up a quite large portion. Setting the results in Figure 12 as the baseline, we observe that almost all the partition results upon the background and plain areas stay the same as those in Figure 13. This observation indicates that our bagged tree model and ResNet model achieve a high accuracy on CUs with flat contents. In addition, as we can see, the differences between Figures 12 and 13 mainly exist among the edge areas. This means that our algorithm should be improved, in terms of the prediction of some particular CUs which cross the boundaries of objects. In addition, we also count the bits of this frame encoded by the original HM16.7 and the proposed algorithm, respectively. The frame encoded by the proposed approach takes 145,352 bits, which outnumbers the 144,552 bits of the original. This means that our approach loses 800 bits on this frame. The increase of 800 bits is reasonable since there are some partition errors which result in a decrease in compression ratio.

In general, compared with the optimal CU partition results generated by the original HM16.7, as shown in Figure 12, there are not many differences in the CU partition results generated by the proposed BTRNFA$_{\text{joint}}$. Our BTRNFA$_{\text{joint}}$ algorithm performs well, and its CU partition results are satisfactory.

5. Conclusions

In this paper, we propose a bagged tree and ResNet-based joint end-to-end CTU partition structure prediction algorithm for HEVC intra coding. First, we construct effective datasets for our models. Then, the bagged tree model is designed to predict the splitting flag of a CTU. In addition, a ResNet-based model is designed and used on each CU of size 32×32, so that the partition structure can be determined through a single prediction. In this way, the traditional brute-force RDO is replaced, so that much encoding complexity is saved. Compared with the original HM16.7, the proposed algorithm saves as much as 60.29% encoding time, with only 2.03% BD-rate loss on average. Furthermore, our approach defeats four CNN-based state-of-the-art fast algorithms when making intra mode decisions, which shows the great application prospects of our approach. However, the ResNet used in our approach is on a large scale, which introduces huge complexity in terms of training and prediction. To reduce the number of trainable parameters, we will use pruning techniques to prune the ResNet. Transfer learning techniques will also be considered to increase the generalization ability of our ResNet in future work.

Author Contributions: Conceptualization, Y.L., L.L. and H.P.; formal analysis, Y.L. and Y.F.; software, Y.L. and Y.F.; supervision, H.P. and N.L.; writing—original draft preparation, Y.L. and H.P.; writing—review and editing, L.L. and N.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported in part by the National Key Research and Development Program of China (grant no. 2020YFB1805403), the National Natural Science Foundation of China (grant nos. 62032002, 61972051 and 61932005), the Natural Science Foundation of Beijing Municipality (grant no. M21034) and the 111 Project (grant no. B21049).

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

HEVC	High-efficiency video coding
VVC	Versatile video voding
AVC	Advanced video coding

AV1	AOMedia video 1
AV2	AOMedia video 2
AVS	Audio-video standards
JCT-VC	Joint collaborative team on video coding
CTU	Coding tree unit
CU	Coding unit
QTBT	Quad-tree plus binary tree
RDO	Rate-distortion optimization
SVM	Support vector machine
ResNet	Residual network
CNN	Convolutional neural network
LSTM	Long short-term memory network
QP	Quantization parameter
CBF	Coded block flag
CTC	Common test conditions
BD-rate	Bit-distortion rate
BTRNFA	Bagged tree and ResNet joint fast algorithm
$BTRNFA_{DT}$	BTRNFA version in which only bagged tree is active
$BTRNFA_{ResNet}$	BTRNFA version in which only ResNet is active
$BTRNFA_{joint}$	BTRNFA version in which bagged tree and ResNet are both active
PMDA	The algorithm proposed by [43]
CRM	The algorithm proposed by [45]
RDNet	The algorithm proposed by [56]
TCA	The algorithm proposed by [57]

References

1. Sullivan, G.J.; Ohm, J.; Han, W.; Wiegand, T. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Trans. Circuits Syst. Video Technol.* **2012**, *22*, 1649–1668. [CrossRef]
2. Fan, Y.; Chen, J.; Sun, H.; Katto, J.; Jing, M. A Fast QTMT Partition Decision Strategy for VVC Intra Prediction. *IEEE Access* **2020**, *8*, 107900–107911. [CrossRef]
3. Wu, S.; Shi, J.; Chen, Z. HG-FCN: Hierarchical Grid Fully Convolutional Network for Fast VVC Intra Coding. *IEEE Trans. Circuits Syst. Video Technol.* **2022**. [CrossRef]
4. Video Developer Report 2021. Available online: https://go.bitmovin.com/video-developer-report (accessed on 12 January 2022).
5. Chen, M.J.; Lin, J.R.; Hsu, Y.C.; Ciou, Y.S.; Yeh, C.H.; Lin, M.H.; Kau, L.J.; Chang, C.Y. Fast 3D-HEVC Depth Intra Coding Based on Boundary Continuity. *IEEE Access* **2021**, *9*, 79588–79599. [CrossRef]
6. Saldanha, M.; Sanchez, G.; Marcon, C.; Agostini, L. Fast 3D-HEVC Depth Map Encoding Using Machine Learning. *IEEE Trans. Circuits Syst. Video Technol.* **2020**, *30*, 850–861. [CrossRef]
7. Bakkouri, S.; Elyousfi, A. FCM-Based Fast Texture CU Size Decision Algorithm for 3D-HEVC Inter-Coding. In Proceedings of the 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), Antalya, Turkey, 2–5 November 2020; pp. 1–5. [CrossRef]
8. Fu, C.H.; Chan, Y.L.; Zhang, H.B.; Tsang, S.H.; Xu, M.T. Efficient Depth Intra Frame Coding in 3D-HEVC by Corner Points. *IEEE Trans. Image Process.* **2021**, *30*, 1608–1622. [CrossRef] [PubMed]
9. Zhang, Y.; Lu, C. Efficient Algorithm Adaptations and Fully Parallel Hardware Architecture of H.265/HEVC Intra Encoder. *IEEE Trans. Circuits Syst. Video Technol.* **2019**, *29*, 3415–3429. [CrossRef]
10. Zhang, Y.; Lu, C. High-Performance Algorithm Adaptations and Hardware Architecture for HEVC Intra Encoders. *IEEE Trans. Circuits Syst. Video Technol.* **2019**, *29*, 2138–2145. [CrossRef]
11. Cai, Z.; Gao, W. Efficient Fast Algorithm and Parallel Hardware Architecture for Intra Prediction of AVS3. In Proceedings of the 2021 IEEE International Symposium on Circuits and Systems (ISCAS), Daegu, Korea, 22–28 May 2021; pp. 1–5. [CrossRef]
12. Sjövall, P.; Lemmetti, A.; Vanne, J.; Lahti, S.; Hämäläinen, T.D. High-Level Synthesis Implementation of an Embedded Real-Time HEVC Intra Encoder on FPGA for Media Applications. *ACM Trans. Des. Autom. Electron. Syst.* **2022**, *27*, 1–34. [CrossRef]
13. Zummach, E.; Palau, R.; Goebel, J.; Palomino, D.; Agostini, L.; Porto, M. Efficient Hardware Design for the AV1 CDEF Filter Targeting 4K UHD Videos. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–5. [CrossRef]
14. Lee, D.; Jeong, J. Fast intra coding unit decision for high efficiency video coding based on statistical information. *Signal Process. Image Commun.* **2017**, *55*, 121–129. [CrossRef]
15. Liu, Z.; Lin, T.L.; Chou, C.C. HEVC coding-unit decision algorithm using tree-block classification and statistical data analysis. *Multimed. Tools Appl.* **2017**, *76*, 9051–9072. [CrossRef]
16. Ahn, S.; Lee, B.; Kim, M. A novel fast CU encoding scheme based on spatiotemporal encoding parameters for HEVC inter coding. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *25*, 422–435. [CrossRef]

17. Min, B.; Cheung, R.C. A fast CU size decision algorithm for the HEVC intra encoder. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *25*, 892–896.
18. Xiong, J.; Li, H.; Wu, Q.; Meng, F. A fast HEVC inter CU selection method based on pyramid motion divergence. *IEEE Trans. Multimed.* **2013**, *16*, 559–564. [CrossRef]
19. Sergeev, V.M.; Dvorkovich, A.V. The Fast CU Splitting Method Based on CU Size, Quantization Parameters and Difference of Variances. In Proceedings of the 2021 International Conference Engineering and Telecommunication (En T), Dolgoprudny, Russia, 24–25 November 2021; pp. 1–4. [CrossRef]
20. Wu, T.; Liu, S.; Wang, F.; Wang, Z.; Wang, R.; Wang, R. Fast CU Partition Decision Algorithm for AVS3 Intra Coding. *IEEE Access* **2021**, *9*, 7540–7549. [CrossRef]
21. Shen, L.; Zhang, Z.; Liu, Z. Effective CU size decision for HEVC intracoding. *IEEE Trans. Image Process.* **2014**, *23*, 4232–4241. [CrossRef]
22. Shen, L.; Liu, Z.; Zhang, X.; Zhao, W.; Zhang, Z. An effective CU size decision method for HEVC encoders. *IEEE Trans. Multimed.* **2012**, *15*, 465–470. [CrossRef]
23. Shen, L.; Zhang, Z.; An, P. Fast CU size decision and mode decision algorithm for HEVC intra coding. *IEEE Trans. Consum. Electron.* **2013**, *59*, 207–213. [CrossRef]
24. Zhang, Y.; Pan, Z.; Li, N.; Wang, X.; Jiang, G.; Kwong, S. Effective data driven coding unit size decision approaches for HEVC INTRA coding. *IEEE Trans. Circuits Syst. Video Technol.* **2017**, *28*, 3208–3222. [CrossRef]
25. Liu, X.; Li, Y.; Liu, D.; Wang, P.; Yang, L.T. An adaptive CU size decision algorithm for HEVC intra prediction based on complexity classification using machine learning. *IEEE Trans. Circuits Syst. Video Technol.* **2017**, *29*, 144–155. [CrossRef]
26. Zhang, T.; Sun, M.T.; Zhao, D.; Gao, W. Fast intra-mode and CU size decision for HEVC. *IEEE Trans. Circuits Syst. Video Technol.* **2016**, *27*, 1714–1726. [CrossRef]
27. Zhu, L.; Zhang, Y.; Kwong, S.; Wang, X.; Zhao, T. Fuzzy SVM-based coding unit decision in HEVC. *IEEE Trans. Broadcast.* **2017**, *64*, 681–694. [CrossRef]
28. Grellert, M.; Zatt, B.; Bampi, S.; da Silva Cruz, L.A. Fast coding unit partition decision for HEVC using support vector machines. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *29*, 1741–1753. [CrossRef]
29. Zhang, Y.; Kwong, S.; Wang, X.; Yuan, H.; Pan, Z.; Xu, L. Machine learning-based coding unit depth decisions for flexible complexity allocation in high efficiency video coding. *IEEE Trans. Image Process.* **2015**, *24*, 2225–2238. [CrossRef] [PubMed]
30. Qin, J.; Bai, H.; Zhang, M.; Zhao, Y. Fast intra coding algorithm for HEVC based on decision tree. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2017**, *100*, 1274–1278. [CrossRef]
31. Zhang, D.; Duan, X.; Zang, D. Decision tree based fast CU partition for HEVC lossless compression of medical image sequences. In Proceedings of the 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, China, 11–13 October 2017; pp. 1–6.
32. Correa, G.; Assuncao, P.A.; Agostini, L.V.; da Silva Cruz, L.A. Fast HEVC encoding decisions using data mining. *IEEE Trans. Circuits Syst. Video Technol.* **2014**, *25*, 660–673. [CrossRef]
33. Ruiz, D.; Fernández-Escribano, G.; Adzic, V.; Kalva, H.; Martínez, J.L.; Cuenca, P. Fast CU partitioning algorithm for HEVC intra coding using data mining. *Multimed. Tools Appl.* **2017**, *76*, 861–894. [CrossRef]
34. Zhu, S.; Zhang, C. A fast algorithm of intra prediction modes pruning for HEVC based on decision trees and a new three-step search. *Multimed. Tools Appl.* **2017**, *76*, 21707–21728. [CrossRef]
35. Lee, D.; Jeong, J. Fast CU size decision algorithm using machine learning for HEVC intra coding. *Signal Process. Image Commun.* **2018**, *62*, 33–41. [CrossRef]
36. Kim, H.S.; Park, R.H. Fast CU partitioning algorithm for HEVC using an online-learning-based Bayesian decision rule. *IEEE Trans. Circuits Syst. Video Technol.* **2015**, *26*, 130–138. [CrossRef]
37. Yang, H.; Shen, L.; Dong, X.; Ding, Q.; An, P.; Jiang, G. Low complexity CTU partition structure decision and fast intra mode decision for versatile video coding. *IEEE Trans. Circuits Syst. Video Technol.* **2020**, *30*, 1668–1682. [CrossRef]
38. Galpin, F.; Racapé, F.; Jaiswal, S.; Bordes, P.; Le Léannec, F.; François, E. CNN-Based Driving of Block Partitioning for Intra Slices Encoding. In Proceedings of the 2019 Data Compression Conference (DCC), Snowbird, UT, USA , 26–29 March 2019; pp. 162–171.
39. Huang, C.; Peng, Z.; Chen, F.; Jiang, Q.; Jiang, G.; Hu, Q. Efficient CU and PU decision based on neural network and gray level co-occurrence matrix for intra prediction of screen content coding. *IEEE Access* **2018**, *6*, 46643–46655. [CrossRef]
40. Shi, J.; Gao, C.; Chen, Z. Asymmetric-Kernel CNN Based Fast CTU Partition for HEVC Intra Coding. In Proceedings of the 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5.
41. Jamali, M.; Coulombe, S.; Sadreazami, H. CU Size Decision for Low Complexity HEVC Intra Coding based on Deep Reinforcement Learning. In Proceedings of the 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 9–12 August 2020; pp. 586–591. [CrossRef]
42. Amna, M.; Imen, W.; Ezahra, S.F. LeNet5-Based approach for fast intra coding. In Proceedings of the 2020 10th International Symposium on Signal, Image, Video and Communications (ISIVC), Saint-Etienne, France, 7–9 April 2021; pp. 1–4. [CrossRef]
43. Liu, Z.; Yu, X.; Gao, Y.; Chen, S.; Ji, X.; Wang, D. CU partition mode decision for HEVC hardwired intra encoder using convolution neural network. *IEEE Trans. Image Process.* **2016**, *25*, 5088–5103. [CrossRef] [PubMed]
44. Kim, K.; Ro, W.W. Fast CU depth decision for HEVC using neural networks. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *29*, 1462–1473. [CrossRef]

45. Xu, M.; Li, T.; Wang, Z.; Deng, X.; Yang, R.; Guan, Z. Reducing complexity of HEVC: A deep learning approach. *IEEE Trans. Image Process.* **2018**, *27*, 5044–5059. [CrossRef]
46. Qi, Z.; Sun, J.; Duan, Y.; Guo, Z. A two-stage fast CU size decision method for HEVC intracoding. In Proceedings of the IEEE International Workshop on Multimedia Signal Processing (MMSP), Xiamen, China, 19–21 October 2015; pp. 1–6.
47. Shah, S.A.A.; Aziz, W.; Arif, M.; Nadeem, M.S.A. Decision Trees Based Classification of Cardiotocograms Using Bagging Approach. In Proceedings of the 2015 13th International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, 14–16 December 2015; pp. 12–17. [CrossRef]
48. Caruana, R.; Elhawary, M.; Munson, A.; Riedewald, M.; Sorokina, D.; Fink, D.; Hochachka, W.M.; Kelling, S. Mining citizen science data to predict orevalence of wild bird species. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 20–23 August 2006; pp. 909–915.
49. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
50. Li, Y.; Li, L.; Fang, Y.; Peng, H.; Yang, Y. Bagged Tree Based Frame-Wise Beforehand Prediction Approach for HEVC Intra-Coding Unit Partitioning. *Electronics* **2020**, *9*, 1523. [CrossRef]
51. Shen, X.; Yu, L. CU splitting early termination based on weighted SVM. *EURASIP J. Image Video Process.* **2013**, *2013*, 4. [CrossRef]
52. Grellert, M.; Bampi, S.; Correa, G.; Zatt, B.; da Silva Cruz, L.A. Learning-based complexity reduction and scaling for HEVC encoders. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 1208–1212.
53. Bay, H.; Tuytelaars, T.; Van Gool, L. Surf: Speeded up robust features. *European Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 404–417.
54. Bossen, F. Common test conditions and software reference configurations. In Proceedings of the 12th Meeting of the JCT-VC, Geneva, Switzerland, 14–23 January 2013.
55. HEVC Reference Software HM16.7. Available online: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.7/ (accessed on 6 September 2016).
56. Yao, C.; Xu, C.; Liu, M. RDNet: Rate–Distortion-Based Coding Unit Partition Network for Intra-Prediction. *Electronics* **2022**, *11*, 916. [CrossRef]
57. Zhang, Y.; Wang, G.; Tian, R.; Xu, M.; Kuo, C.C.J. Texture-Classification Accelerated CNN Scheme for Fast Intra CU Partition in HEVC. In Proceedings of the 2019 Data Compression Conference (DCC), Snowbird, UT, USA, 26–29 March 2019; pp. 241–249. [CrossRef]