Fast HEVC Coding Unit Decision Based On BP-Neural Network

Jing He, Wenkao Yang and Jing Wang

Department of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, 100044, China
13120070@bjtu.edu.cn, wkyang@bjtu.edu.cn, 11120158@bjtu.edu.cn

Abstract

High Efficiency Video Coding (HEVC) employs a highly flexible quad-tree coding block partitioning structure characterized by a coding unit (CU), prediction unit (PU) and transform unit (TU). Taking the computational complexity of CU splitting in HEVC into consideration, a novel fast CU selection algorithm based on a back-propagation neural network (BPNN) is proposed in this paper. To find the best prediction mode at the current CU, the BPNN classifier is designed for two kinds of outputs representing “split” and “unsplit”. Thus, whether to split the current CU or not is directly judged by the outputs of the BPNN classifier. With this new scheme, the unnecessary CU splitting process can be skipped in advance. Experiments and results show that the proposed algorithm could greatly reduce the high computational complexity and the average time savings reaches 47.75% in Random Access Main configuration and 41.94% in Low Delay B Main configuration with a BD-bitrate increase of 1.89%.

Key words: HEVC, CU splitting, BPNN classifier, BD-bitrate

1. Introduction

HEVC is the next generation video coding standard being defined by a joint collaborative team on video coding (JCT-VC)[1]. The aim of HEVC is to significantly improve coding efficiency compared to the existing video coding standards. It employs a highly flexible quad-tree coding block partitioning structure based on CU, PU and TU [2][3]. CU, whose size ranges from 64x64 to 8x8 pixels and allows recursive quad-tree splitting, is a basic coding unit of HEVC. However, to find an optimal CU structure, the HEVC test model (HM) [4] employs rate distortion optimization (RDO), a kind of exhaustive search, which significantly improves the coding efficiency of HEVC while resulting in great computation complexity to the encoder.

Recently, many fast CU selection algorithms have been developed to decrease the high computational complexity of CU size decision in HEVC. An early detection of SKIP mode was proposed in [5] by checking the differential motion vector (MV) and the coded block flag (CBF). A fast coding unit decision algorithm in either frame level or coding unit level to accelerate encoding procedure is introduced in [6] by J. Leng et al. The proposed algorithm provides on average 45% reduction in total encoding time. A fast CU size decision method based on improved Bayesian classification framework is shown in [7], whose total encoding time was reduced to 54.7% of HM 10.0. A CU-splitting early termination algorithm based on a weighted support vector machine (SVM) was introduced in [8], in which a CU splitting decision was considered as a binary classification problem. This method achieves notable RD performance but leads to higher computation cost. A pyramid motion divergence (PMD) based method is presented in [9]. It indeed reduces the computational complexity of RDO and accelerates the coding significantly, but brings the loss of PSNR.

This paper proposes a fast and efficient coding unit selection algorithm based on a
BP-neural network. BPNN, which is one of the most widely applied neural network models, has a simple structure and is easy to implement [10]. The CU splitting in HEVC can be more effective with help of the trained BPNN. Simulation results and comparative analysis show that the proposed algorithm significantly reduces the high computational complexity while maintaining high performance in coding quality.

The remainder of the paper is organized as follows. Section 2 gives a brief introduction of the coding unit in HEVC. Section 3 elaborates the fast efficient algorithm in detail. The simulation results and comparisons are presented in Section 4, and conclusions are drawn in Section 5.

2. Overview of Coding Unit in HEVC

The emerging HEVC employs a highly flexible quad-tree coding block partitioning structure. First, the pictures are divided into a sequence of coding tree units (CTUs) [4], which can better sub-partition the picture into variable sized structures, such as 64×64, 32×32 or 16×16. As shown in Figure 1, each CTU consists of one luma coding tree block (CTB), two chroma CTBs and the associated syntax element (SE), in arrangement with the ratio of 4:2:0. In contrast to a fixed size of 16×16 pixel macroblocks used for inter/intra coding in the previous video coding standards, such as H.264/AVC, the CU size in HEVC is variable with the maximize size of 64×64 pixels. The high-resolution video encoding particularly benefits from the availability of larger CTBs.

![Figure 1. The Makeup of a CTU](image)

A CTU may consist of one or more CUs. For HEVC, a quad-tree-based coding approach was introduced to help CU splitting, such that the leaves of the quad tree represent the CUs. If the leaf node flag is set to 1, the CU is partitioned into four sub-CUs; else, the leaf node with flag 0 is non-split, just resulting as a final CU. As shown in Figure 2, when coding scanning began, we adopted the zigzag scanning mode, which ensures that the current coding unit on the left, top or top left cell already completed, and can eliminate redundancies by the coded unit. CU splitting can be highly flexible, with sizes ranging from 64×64 to 8×8, and it can be divided into four CUs recursively up to the maximum allowable depth=4.

![Figure 2. CUs and the Corresponding Quad Tree](image)
In accordance with the partition mode, a CU may contain one or more PUs[4]. In Figure 3, there are eight partition modes used for inter-coded CUs and two partition modes for intra-coded CUs. Intra-coded CUs with PART_2Nx2N contains one PU, while a CU with PART_NxN contains four PUs. The other modes contain two PUs.

However, how to find an optimal CU splitting is becoming one of the most urgent problems that HEVC encoders are faced with. The exhaustive RD cost calculation is one of the preferred methods. Though it significantly improves the coding efficiency, it brings high computational complexity [11]. In this paper, a fast CU splitting algorithm based on BPNN is proposed. The decision to partition or not partition the current CU is directly depend on the outputs of the BPNN classifier. With this framework, the unnecessary CU splitting process can be easily skipped in advance.

3. BP-Based CU Selection Optimization Algorithm

A schematic diagram of the fast CU selection optimization algorithm is performed as shown in Figure 4.

(1) Calculate the RD cost for the possible prediction mode (Intra, Skip, Inter) and code the current CU with depth = depth;
(2) Select the mode with the minimum cost as the PU partition mode at the current CU size.
(3) Predict and determine whether or not to split the current CU by the outputs of the BPNN classifier. In other words, the BPNN classifier designed with feature values
produces two kinds of results: “split” and “unsplit” for the input CU.

(4) If the current CU belongs to “split,” with a size of 8x8 pixels, it shall be divided into four sub-CUs, and the depth increases by 1. Then repeating steps (1) to (4) for the four sub-CUs to decide whether to continue further splitting.

(5) Otherwise, terminate CU splitting and exit the process.

3.1. Feature Selection

In this section, we focus on the problem of selecting features to construct the BPNN classifier. As we all know, classifiers with proper features as specific parameters will not only help effective classification and save storage memory, but also reduces the training and prediction time. Hence, feature selection plays a critical role to implement our new algorithm.

CU splitting is influenced by a few features, such as the sum absolute transform difference (SATD) of the prediction residual of Inter_2N×2N mode, RD cost of Inter_2N×2N mode, coded block flag (CBF) of Inter_2N×2N mode, Co-located of CU depth, variance of four sub-CUs’ SATD (SAD) of the prediction residual of Inter_2N×2N mode and so on. With comparison and analysis, the three candidate features, SATD of the prediction residual of Inter_2N×2N mode, denoted as $f_{\text{SATD}}$; variance of four sub-CUs’ SATD of the prediction residual of Inter_2N×2N mode, denoted as $f_{\text{VAR}}$; and CBF of Inter_2N×2N mode, denoted as $f_{\text{CBF}}$ are selected as input feature vectors of the BPNN classifier.

In the process of feature selection, sequential forward selection (SFS) [12] was utilized to determine the three feature values. First, the feature set is initialized as empty. Second, in each iteration process, a quantized feature with optimal value is added to the feature set. Third, the iterative splitting process continues until the number of feature set is satisfied.

3.2. Threshold Function

When preparing a BPNN classifier, the threshold and the output probability density function must be carefully considered [13]. As illustrated above, the BPNN classifier should output two kinds of results: “split” and “unsplit.” After comparing different classification principles of BP neural classifiers, we adopted a binary BP neural classifier, which is designed with two kinds of output, +1 and -1, respectively. Correspondingly, the sign function expressed as $f(x) = \text{sgn}(x)$ [14] is selected as our threshold function. Based on the sign function, an improved BP neural classifier model was constructed as in Figure 5.

![Figure 5. A Binary BP Neural Classifier](image)

In this figure, $x_i$ is the input $w_i$ is the weight value, i.e., synaptic connection strength,
and the threshold function is \( f(x) = \text{sgn}(x) \).

Set the threshold: \( \theta = \omega_0 \), \( W = (\omega_1, \omega_2, \omega_3, \omega_4)^T \), \( X = (x_1, x_2, x_3, 1)^T \), then

\[
y = f(W^T X) = \begin{cases} 
1 & W^T X > 0 \\
-1 & W^T X < 0
\end{cases} \Rightarrow X \in \Omega_1 \\
\Omega_2
\]

(1)

3.2. Thresholds and Feature Weights

As can be seen from Figure 5, the optimal feature weights and the thresholds can be obtained through training the BP neural classifier. The steepest descent method [15] was introduced in this paper, and the perceptron training algorithm was as follows:

(1) Initialization

First, choose a training set \( \{X_1, X_2, \ldots, X_n\} \), \( \Omega_1 \) and \( \Omega_2 \) represent two different categories. If \( x_i \in \Omega_1 \), then \( x_i \) is multiplied by \(-1\).

Second, \( x_i = (x_{i1}, x_{i2}, x_{i3})^T \) is a \((3+1)\) dimensional augmented vector in which \( x_{i1}, x_{i2}, x_{i3} \) represents the three factors of \( x_i \), and \( 1 \) is the augmented matrix.

Third, make weight vector \( W_i = (\omega_{i1}, \omega_{i2}, \omega_{i3}, \omega_{i4})^T \) a \((3+1)\) dimensional vector in which weight function \( \omega_{i1}, \omega_{i2}, \omega_{i3} \) corresponds to \( x_{i1}, x_{i2}, x_{i3} \) respectively, and \( \omega_{i4} \) is the threshold, which makes the classification decision.

(2) The algorithm steps

First, set the step \( k=1 \) and the increment \( C \) is a constant \((C>0)\). Prepare each component of the initial augmented weight vector \( W \) with a small arbitrary value.

Second, calculate the function value \( W^T(k)X_i \) when different training modes \( X_i \) is input.

Third, adjust the augmented weight vectors as follows:

\[
W(k+1) = \begin{cases} 
W(k) + \rho X_k & W^T(k)X_k \leq 0 \\
W(k) & W^T(k)X_k \geq 0
\end{cases}
\]

Where \( \rho \) is the learn rate, a random number between 0 and 1.

(3) If \( k < N \), \( k=k+1 \), go back to the step (2). If \( k=N \), make sure the function values \( W^T(k)X \) can be accurately classified for all the training modes \( \{X_1, X_2, \ldots, X_n\} \). If yes, go to end; else, \( k=1 \), go back to step (2).

3.3. Training Process

HEVC in HEVC provides multiple sequences, which is convenient for us to train the classifiers. However, the more training samples are used, the longer training time will be taken [16]. Therefore, it is important to choose the proper number of training samples to reduce the training time while achieving fast CU splitting. To determine the reasonable number of training samples in our algorithm, we experimented on HM9.0 with test sequences Cactus, BQMall and Four People. The optimal feature weight and the threshold are determined by the input feature vectors and optimal CU splitting. In the process, test samples were randomly selected from the feature set and divided into M subsets in which M-1 subsets were used to train BP neural classifiers, while the rest one was selected as a detection subset to judge whether the trained data is optimal. Repeated the above process and recorded the accuracy of the trained data each time. Thus, the integrated CV (Cross Validation) rate of this classifier was obtained. This method is also known as cross-validation [17].
As shown in Figure 6, with the number of samples increasing, the accuracy rate was not increasing continuously, but fluctuating around a fixed value. The accuracy rate intended to be unchangeable when the samples reached 20,000. Hence, we chose 20,000 training samples to train the BP neural classifier. As illustrated above, \( f_{\text{SATD}}, f_{\text{VAR}}, \) and \( f_{\text{CBF}} \) were finally selected as the feature vectors of CU splitting. Thus, the parameters of the binary BP neural classifier contain \( \omega_1, \omega_2, \omega_3, \) and \( \omega_0 \), where \( \omega_1, \omega_2, \omega_3, \omega_0 \) are the weight values of \( f_{\text{SATD}}, f_{\text{VAR}}, \) and \( f_{\text{CBF}} \) respectively and \( \omega_0 \) is the classification threshold.

For different quantization parameters (QP), the threshold of CU splitting is also different. With QP ranging from 0 to 51, HM encoder can get the corresponding classification threshold which is saved as a matrix, denoted as \( T_{\text{BP}} \). During the training process of the BP neural classifier with 20,000 selected samples, we can get the corresponding values of the four parameters \( \omega_1, \omega_2, \omega_3, \omega_0 \) when the most accurate rate completes, and record them as table\( W_1, W_2, W_3 \) and \( T_{\text{BP}} \) respectively. These four tables make it convenient to search the feature values and thresholds with different QP in our experiments.

### 3.4. BP-Based CU Optimal Splitting Algorithm

With unknown parameters being determined, the BP neural classifier can be loaded to HM test model. The BP-based fast CU partition algorithm is as follows:

1. The classification thresholds of BP neural classifier are initialized with \( T_{\text{BP}} \) corresponding to the QP;
2. Start each CTU with depth=0;
3. Then, calculating the RD cost of the Inter_2Ncostat with depth=0 partition s of the current CU to get the feature vectors: \( F=[f_{\text{SATD}}, f_{\text{VAR}}, f_{\text{CBF}}] \);
4. Put the feature vectors \( F \) into the BP neural classifier. Thus, CU splitting decision is depended on the threshold.
   - **Case 1:** if the output of BP neurons classifier is 1, which means the current CU belongs to the Set \( \Omega_1 \), then split the CU into four sub-CUs. Repeat step (3) for the four sub-CUs to determine whether or not to further split.
   - **Case 2:** if the output is -1, the CU belongs to the Set \( \Omega_2 \) and will not be split; go to step (5).
   5. Calculate the RD cost of the possible prediction modes, \( \text{Inter}_2N, \text{Inter}_N \times nt, \text{Inter}_2Ne \) and so on, and select the mode with the minimum cost as the PU partition mode at the current CU size.
   6. Exit the recursive CU splitting process and go back to step (s2), proceed to code next CTU.
4. Experimental results

The proposed algorithm was implemented on HM9.0 and experimented with Low Delay B main and Random Access Main, including BasketballDrive, BQTerrace, BasketballDrill, Kimono, RaceHorses and RaceHorsesC, when QP was set to 28, 32, 36 and 40 respectively. The coding performance is measured by BD bit rate (BDBR) [18-20] and the encoder complexity is measured by the time savings $\Delta T$, which is calculated as:

$$\Delta T = \frac{\text{Time(proposed)} - \text{Time(anchor)}}{\text{Time(anchor)}} \times 100\%$$

$\Delta T$ is the encoding time of Anchor and BP-based Cu Splitting (BP-CU) in Random Access

Table 1. The Performance of BP-CU Optimization Algorithm in Low Delay

<table>
<thead>
<tr>
<th>QP</th>
<th>Anchor</th>
<th>BP-CU</th>
<th>ΔP (%)</th>
<th>BDBR-Y (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2062.</td>
<td>2065.</td>
<td>20</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>35.68</td>
<td>35.65</td>
<td>.92</td>
<td>.03</td>
</tr>
<tr>
<td>3</td>
<td>1109.</td>
<td>1113.</td>
<td>26</td>
<td>.61</td>
</tr>
<tr>
<td>2</td>
<td>32.87</td>
<td>32.79</td>
<td>.92</td>
<td>.61</td>
</tr>
<tr>
<td>18</td>
<td>1.89</td>
<td>1.60</td>
<td>.14</td>
<td>.49</td>
</tr>
<tr>
<td>28</td>
<td>26.61</td>
<td>25</td>
<td>.25</td>
<td>.49</td>
</tr>
<tr>
<td>3</td>
<td>586.2</td>
<td>30.69</td>
<td>.25</td>
<td>.49</td>
</tr>
<tr>
<td>91</td>
<td>3.52</td>
<td>1.46</td>
<td>.25</td>
<td>.49</td>
</tr>
<tr>
<td>28</td>
<td>2.24</td>
<td>1.33</td>
<td>.25</td>
<td>.49</td>
</tr>
<tr>
<td>2</td>
<td>366.8</td>
<td>367.5</td>
<td>2.2</td>
<td>.32</td>
</tr>
<tr>
<td>8</td>
<td>28.55</td>
<td>28.45</td>
<td>.13</td>
<td>.02</td>
</tr>
<tr>
<td>7</td>
<td>29.61</td>
<td>29.52</td>
<td>.12</td>
<td>.02</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>6</td>
<td>.13</td>
<td>.02</td>
</tr>
</tbody>
</table>
As shown in table 1, in LowDelay, as compared to HM9.0, in terms of encoding complexity, which can be seen from Figure 7 and table 1, the time savings at most reaches 69.64% with BQTerrace when QP is set to 40, while the least time savings reaches 20.92% with RaceHorses when QP is set to 28, resulting in an average of 41.94% time saving. Moreover, the BDBR also increases 1.89% at the same time with a slight PSNR loss. So it is reasonable to conclude that this algorithm is able to speed up the encoding procedure significantly.
Figure 8. The Encoding Time of Anchor and BP-CU in Random Access

Table 2. The Performance of BP-Based CU Optimization Algorithm in Random Access

<table>
<thead>
<tr>
<th></th>
<th>QP</th>
<th>Anchor</th>
<th>BP-CU</th>
<th>Λ (%)</th>
<th>BDBR (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BitRate (kbps)</td>
<td>PSNR (dB)</td>
<td>Time (s)</td>
<td>BitRate (kbps)</td>
</tr>
<tr>
<td>Race-Horse</td>
<td>2 8</td>
<td>1835.35</td>
<td>35.40</td>
<td>1.7</td>
<td>1838.52</td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>1005.26</td>
<td>34.09</td>
<td>1.1</td>
<td>1006.12</td>
</tr>
<tr>
<td></td>
<td>3 6</td>
<td>570.33</td>
<td>30.97</td>
<td>1.3</td>
<td>569.95</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>365.80</td>
<td>29.24</td>
<td>1.0</td>
<td>365.30</td>
</tr>
<tr>
<td>Basket-Dmill</td>
<td>2 8</td>
<td>1590.48</td>
<td>36.95</td>
<td>2.9</td>
<td>1594.60</td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>869.93</td>
<td>34.90</td>
<td>2.5</td>
<td>874.53</td>
</tr>
<tr>
<td></td>
<td>3 6</td>
<td>546.13</td>
<td>32.58</td>
<td>2.2</td>
<td>546.36</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>359.41</td>
<td>30.88</td>
<td>1.7</td>
<td>358.70</td>
</tr>
<tr>
<td>Basket-Dmill</td>
<td>2 8</td>
<td>5585.16</td>
<td>37.31</td>
<td>14.1</td>
<td>5643.78</td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>2946.83</td>
<td>35.77</td>
<td>13.8</td>
<td>3005.76</td>
</tr>
<tr>
<td></td>
<td>3 6</td>
<td>1817.61</td>
<td>34.25</td>
<td>13.9</td>
<td>1847.83</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>1192.62</td>
<td>32.66</td>
<td>12.6</td>
<td>1328.39</td>
</tr>
<tr>
<td>Kimo-no</td>
<td>2 8</td>
<td>1959.40</td>
<td>39.49</td>
<td>11.0</td>
<td>1957.12</td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>1071.64</td>
<td>38.01</td>
<td>10.2</td>
<td>1071.55</td>
</tr>
<tr>
<td></td>
<td>3 6</td>
<td>642.67</td>
<td>35.65</td>
<td>9.7</td>
<td>641.36</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>415.94</td>
<td>33.96</td>
<td>8.9</td>
<td>414.66</td>
</tr>
<tr>
<td>BQ-Terrace</td>
<td>2 8</td>
<td>7309.98</td>
<td>35.14</td>
<td>15.6</td>
<td>7228.81</td>
</tr>
<tr>
<td></td>
<td>3 2</td>
<td>2870.88</td>
<td>34.43</td>
<td>15.1</td>
<td>2836.54</td>
</tr>
<tr>
<td></td>
<td>3 6</td>
<td>1627.74</td>
<td>32.60</td>
<td>15.6</td>
<td>1604.79</td>
</tr>
<tr>
<td></td>
<td>4 0</td>
<td>1021.01</td>
<td>31.05</td>
<td>14.8</td>
<td>1005.55</td>
</tr>
</tbody>
</table>

Ave. 47.75 1.88
Figure 8 depicts the encoding time of sequences in Random Access with HM and our proposed algorithm, and it can be seen that the encoding time of standardized HM ranges from 1.07h to 16.95h, while the time of the new algorithm ranges from 0.75h to 9.68h. From Table 2, it is obvious that when QP is set to 28, with different test sequences and different QP, the time reduction varies from 25.77% to 65.92%, and the BDBR varies from 1.69% to 2.11%. As a result, our proposed fast CU decision algorithm achieves an average of 47.75% time reduction, while maintaining an average increase of 1.88% BDBR.

As a result, compared with the RDO employed by HM, different video test sequences have different motion and texture features, resulting in the encoding time of our BP based method being reduced by 1.5 to 3 times and the compression performance measured by BDBR increased from 1.63% to 2.24%, resulting in an average increase of 1.89%, while the RD loss is no more than 1.9%. That is to say, this fast method based on BPNN for CU splitting can effectively reduce encoding complexity while maintaining high compression performance.

5. Conclusion

A novel fast coding unit splitting algorithm for the HEVC encoding is proposed in this paper. The BP neural network, one of the most popular neural network models in practice, which has a simple structure and is easy to implement, is introduced. In this algorithm, the CU splitting decision is directly made by the trained BPNN classifier, which helps HEVC avoid unnecessary search efforts. Compared with the existing HEVC standard, our new algorithm greatly decreases the computation complexity and improves compression efficiency with almost negligible RD loss. On average, approximately one half of the encoding time can be saved. Moreover, the algorithm also has a good robust performance, providing accurate CU splitting in different test sequences.

References
