# HEVC BASED SCREEN CONTENT CODING AND TRANSCODING USING MACHINE LEARNING TECHNIQUES

KUANG WEI

PhD

The Hong Kong Polytechnic University

2019

The Hong Kong Polytechnic University

Department of
Electronic and Information Engineering

# HEVC Based Screen Content Coding and Transcoding Using Machine Learning Techniques

Kuang Wei

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

August 2019

# CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

______________________________ (Signed)

Kuang Wei (Name of student)

# Abstract

Screen content video is one of the emerging videos, and it usually shows mixed content with both of nature image blocks (NIBs) and computer-generated screen content blocks (SCBs). Since High Efficiency Video Coding (HEVC) is only optimized for NIBs while SCBs exhibit different characteristics, new techniques are necessary for SCBs. Screen Content Coding (SCC) extension was developed on top of HEVC to explore new coding tools for screen content videos. SCC employs two additional coding modes, intra block copy (IBC) mode and palette (PLT) mode for intra-prediction. However, the exhaustive mode searching makes the computational complexity of SCC increase dramatically. Therefore, in this thesis, some novel machine learning based techniques are suggested to simplify both encoding and transcoding of SCC.

A fast intra-prediction algorithm for SCC by content analysis and dynamic thresholding is firstly proposed. A scene change detection method is adopted to obtain a learning frame in each scene, and the learning frame is encoded by the original SCC encoder to collect learning statistics. The prediction models are tailor-made for the following frames in the same scene according to the video content and QP of the learning frame. Simulation results show that the proposed scheme can achieve remarkable complexity reduction while preserving the coded video quality.

Afterwards, we propose a decision tree based framework for fast intra mode decision by investigating various features in training sets. To avoid the exhaustive mode searching process, a framework with a sequential arrangement of decision trees is proposed to check each mode separately by inserting a classifier before checking a mode. As compared with the previous approaches that both IBC and PLT modes are checked for SCBs, the proposed coding framework is more flexible which facilitates either IBC or PLT mode to be checked for SCBs such that computational complexity is further reduced. Simulation results show that the proposed scheme can provide significant complexity saving with negligible loss of coded video quality.

To avoid the necessity of hand-crafted features, a deep learning based fast prediction network DeepSCC is then proposed by using convolutional neural network (CNN), which contains two parts, DeepSCC-I and DeepSCC-II. Before fed to DeepSCC, incoming coding units (CUs) are divided into two categories: dynamic coding tree units (CTUs) and stationary CTUs. For dynamic CTUs with different content as their collocated CTUs, DeepSCC-I takes raw sample values as the input to make fast predictions. For stationary CTUs with the same content as their collocated CTUs, DeepSCC-II additionally utilizes the optimal mode maps of the stationary CTU to further reduce the computational complexity. Simulation results show that the proposed scheme further improves the complexity reduction.

Finally, we propose a fast HEVC to SCC transcoder. To migrate the legacy screen content videos from HEVC to SCC to improve the coding efficiency, a fast transcoding framework is proposed by analyzing various features from 4 categories. They are the features from the HEVC decoder, static features, dynamic features, and spatial features. First, the CU depth level collected from the HEVC decoder is utilized to early terminate the CU partition in SCC. Second, a flexible encoding structure is proposed to make early mode decisions with the help of various features. Simulation results show that the proposed scheme dramatically shortens the transcoding time.

# Publications Arising from the Thesis

## International Journal Papers

1. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "DeepSCC: Deep Learning Based Fast Prediction Network for Screen Content Coding," IEEE Transactions on Circuits and Systems for Video Technology (Accepted on 2 July, 2019)

2. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Online-Learning-Based Bayesian Decision Rule for Fast Intra Mode and CU Partitioning Algorithm in HEVC Screen Content Coding," IEEE Transactions on Image Processing. (Accepted on 10 June, 2019)

3. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Machine Learning Based Fast Intra Mode Decision for HEVC Screen Content Coding Via Decision Trees," IEEE Transactions on Circuits and Systems for Video Technology. (Accepted on 20 February, 2019)

4. **Wei Kuang**, Yui-Lam Chan, and Sik-Ho Tsang, "Fast HEVC to SCC Transcoder by Early CU Partitioning Termination and Decision Tree Based Flexible Mode Decision for Intra-Frame Coding," IEEE Access, vol. 7, pp. 8773–8788, January 2019.

5. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Efficient Intra Bitrate Transcoding for Screen Content Coding Based on Convolutional Neural Network," IEEE Access, vol. 7, pp. 107211-107224, August 2019.

6. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Fast Intraprediction for High-Efficiency Video Coding Screen Content Coding by Content Analysis and Dynamic Thresholding," Journal of Electronic Imaging, vol. 27, no. 5, pp. 053029-1–053029-18, October 2018.

7. Sik-Ho Tsang, Yui-Lam Chan, **Wei Kuang**, "Mode Skipping for HEVC Screen Content Coding via Random Forest," IEEE Transactions on Multimedia, (Accepted on 7 March, 2019)

8. Sik-Ho Tsang, Yui-Lam Chan, **Wei Kuang**, and Wan-Chi Siu, "Reduced-Complexity Intra Block Copy (IntraBC) Mode with Early CU Splitting and Pruning for HEVC Screen Content Coding," IEEE Transactions on Multimedia, vol. 21, no. 2, pp. 269-283, July 2018.

## International Conference Papers

1. **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, “Fast HEVC to SCC Transcoding Based on Decision Trees,” in Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2018), San Diego, California, USA, July, 2018, pp.1-6.

2. **Wei Kuang**, Sik-Ho Tsang, Yui-Lam Chan and Wan-Chi Siu, “Fast mode decision algorithm for HEVC screen content intra coding,” in Proceedings of the IEEE International Conference on Image Processing (ICIP 2017), Beijing, China, September, 2017, pp.2473-2477.

3. Sik-Ho Tsang, **Wei Kuang**, Yui-Lam Chan and Wan-Chi Siu, “Decoder side merge mode and AMVP in HEVC screen content coding” in Proceedings of the IEEE International Conference on Image Processing (ICIP 2017), Beijing, China, September, 2017, pp.260-264.

4. Sik-Ho Tsang, **Wei Kuang**, Yui-Lam Chan and Wan-Chi Siu, “Fast HEVC screen content coding by skipping unnecessary checking of intra block copy mode based on CU activity and gradient,” in Proceedings of Signal and Information Processing Association Annual Summit and Conference (APSIPA 2016), Jeju, Korea, Dec. 2016, pp.1-5.

# Acknowledgments

I would like to express my sincere gratitude to my chief supervisor, Dr. Yui-Lam Chan, and my co-supervisor, Prof. Wan-Chi Siu for their continuous encouragement, supervision throughout my research work. They offered me valuable suggestions and guidance that make me complete my work successfully.

Also, I would like to express my sincere gratitude to Dr. Sik-Ho Tsang, Dr. Tsz-Kwan Lee, Dr. Hong-Bin Zhang, Dr. Huan Dou, Dr. Meng Yao, Dr. Jun-Jie Huang, Dr. Tian-Rui Liu, Mr. Song-Zhi Liu, Ms. Xue-Fei Yang, Mr. Chu-Tak Li, Mr. Li-Wen Wang, Mr. Ting-Tian Li, Mr. Hui-Wai Lam, Mr. Chao Shi, and other members in Center for Signal Processing. The sharing of ideas and experience with them has greatly contributed to make every success of my work.

Meanwhile, I am greatly appreciative of the Department of Electronic and Information Engineering and The Hong Kong Polytechnic University for providing me a comfortable working environment and for their financial support to my research work.

Finally, I am deeply indebted to my parents and friends for their continuous company and love. Without their understanding and patience, it is impossible for me to complete this research study.

# Table of Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| ITU-T | Telecommunication Standardization Sector of ITU |
| VCEG | Video Coding Experts Group |
| ISO | International Organization for Standardization |
| IEC | International Electrotechnical Commission |
| MPEG | Moving Picture Experts Group |
| AVC | Advanced Video Coding |
| HEVC | High Efficiency Video Coding |
| NIB | Natural Image Block |
| SCB | Screen Content Block |
| JCT-VC | Joint Collaborative Team on Video Coding |
| SCC | Screen Content Coding |
| Intra | HEVC Conventional Intra |
| CTU | Coding Tree Unit |
| CU | Coding Unit |
| QP | Quantization Parameter |
| IBC | Intra Block Copy |
| PLT | Palette |
| BV | Block Vector |
| PU | Prediction Unit |
| RD | Rate-Distortion |
| SCM | Screen Content Model |
| BDBR | Bjøntegaard Delta Bitrate |
| AI | All intra |
| CTC | Common Test Condition |
| CC | Camera-Captured Content |
| A | Animation |
| TGM | Text and Graphics with Motion |
| M | Mixed Content |
| SCM-8.3 | HEVC Test Model Version 16.12 Screen Content Model Version 8.3 |
| CBFT | Conventional Brute-Force Transcoder |
| FHST | Fast HEVC to SCC Transcoder |
| HOD | Histogram of Difference |
| DOH | Difference of Histogram |
| DT | Decision Tree |
| SAD | Sum of Absolute Differences |
| ME | Motion Estimation |
| MV | Motion Vector |
| AMVP | Advanced Motion Vector Prediction |
| MC | Motion Compensation |
| BV | Block Vector |
| ACT | Adaptive Color Transform |
| AMCP | Adaptive Motion Compensation Precision |
| RNIB | Rough Natural Image Block |
| RSCB | Rough Screen Content Block |
| WEKA | Waikato Environment for Knowledge Analysis |
| LD | Low Delay |
| CNN | Convolutional Neural Network |
| FHST | Fast HEVC to SCC Transcoder |
| LSTM | Long- and Short-Term Memory |

# Chapter 1 Introduction

## 1.1 Overview

Digital video has become one of the most popular media for content representation and distribution. However, the huge storage and transmission bandwidth requirements limit the application of videos in the raw form. To reduce the storage and transmission cost, video compression is desired. The Telecommunication Standardization Sector of ITU (ITU-T) Video Coding Experts Group (VCEG) and the International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) Moving Picture Experts Group (MPEG) have developed a series of video coding standards such as H.261, MPEG-1 Part 2, H.262/MPEG-2 Part 2, H.263, MPEG-4 Part 2, H.264/Advanced Video Coding (AVC), and H.265/High Efficiency Video Coding (HEVC) [1]–[11], which enable video encoders and decoders from different manufactures work together across a range of applications [12]. All these standards were originally developed for the compression of camera-capture videos. With the fast development of the Internet and wireless communication, an emerging video type, screen content video, is playing an essential role in many applications, such as cloud-mobile computing, remote education, video conference with document sharing, wireless or Wi-Fi screen mirroring [12], etc. In these applications, it requires transmitting the computer-generated screen content video from one device to another for display. Efficient coding of screen content videos with low complexity is crucial for the success of these applications. Screen content videos usually shows a mixed content of traditional natural image blocks (NIBs) and screen content blocks (SCBs), as shown in Figure 1.1. Compared with NIBs, SCBs have

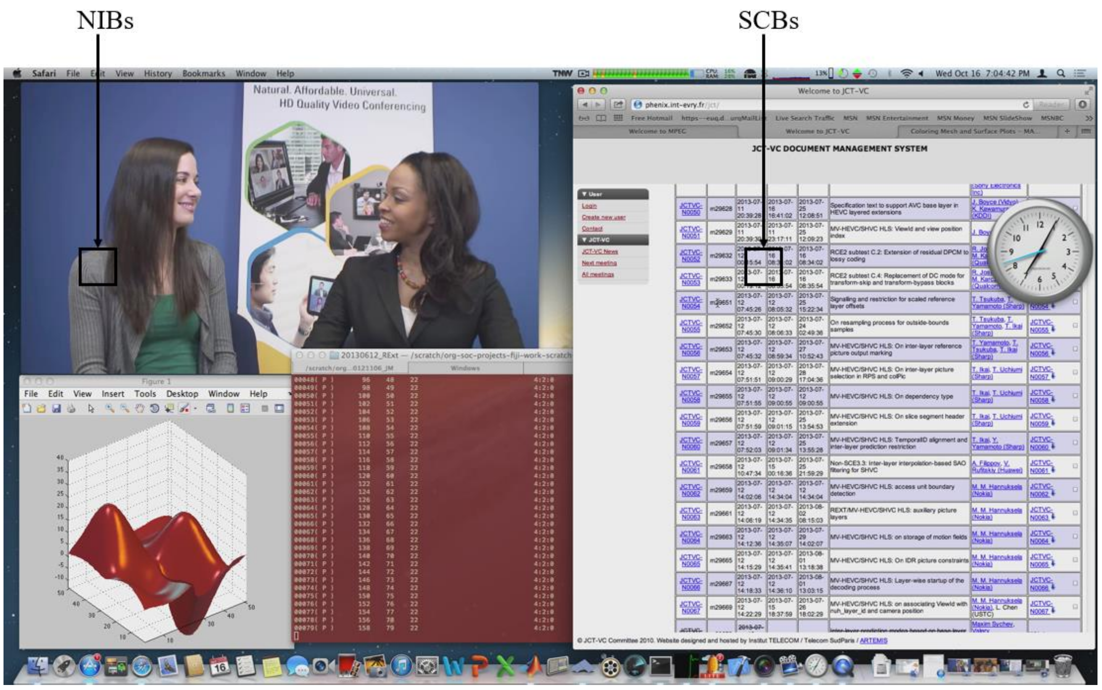


Figure 1.1: NIB and SCB in the first frame of “MissionControlClip3”.

different signal characteristics, including no sensor noise, large flat areas with a single color, many repeated patterns within one frame and limited colors. While NIBs can be well compressed by the HEVC conventional intra (Intra) mode, new techniques are necessary for SCBs. Therefore, the Joint Collaborative Team on Video Coding (JCT-VC) established jointly by ITU-T VCEG and ISO/IEC MPEG has developed Screen Content Coding (SCC) extension [13] of HEVC to explore new encoding tools for screen content videos since January 2014, and it was finalized in 2016.

## 1.2 Flexibility of Intra Coding Structures In SCC

SCC inherits the same flexible quadtree-based block partitioning scheme from HEVC with the introduction of several new coding modes. In SCC, frames are divided into non-overlapping coding tree units (CTUs), which are the basic coding units with the size of 64×64 pixels, i.e. depth level of 0. Then each CTU can be partitioned into four coding units (CUs) of equal size, and each CU can be further partitioned into four smaller

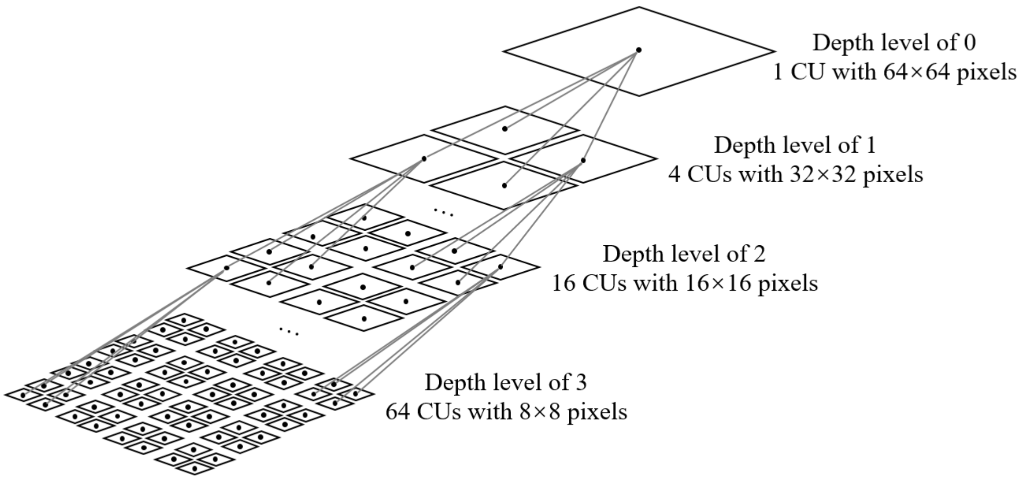


Figure 1.2: A CTU partition and its corresponding partitioning structure.

CUs recursively until the smallest CUs of 8×8 pixels are reached, i.e. depth level of 3. An example of the partition structure in a CTU is shown in Figure 1.2. The intra mode decision process is performed for CUs with different sizes recursively. To efficiently encode a CU, two additional modes, intra block copy (IBC) mode [15]–[21] and palette (PLT) [22]–[28] mode, are introduced. IBC mode is a block matching based intraframe approach as shown in Figure 1.3. IBC mode searches the reconstructed regions of the current frame to find the best reference block for the current CU. It includes three steps – IBCPredictor, IBC merge&skip (IBCM&S) and IBCSearch. IBCPredictor simply checks a set of block vectors (BVs) from the two last encoded CUs and the neighboring CUs of left, above, collocated, below left, above right and above left. IBCM&S is the intra version of the

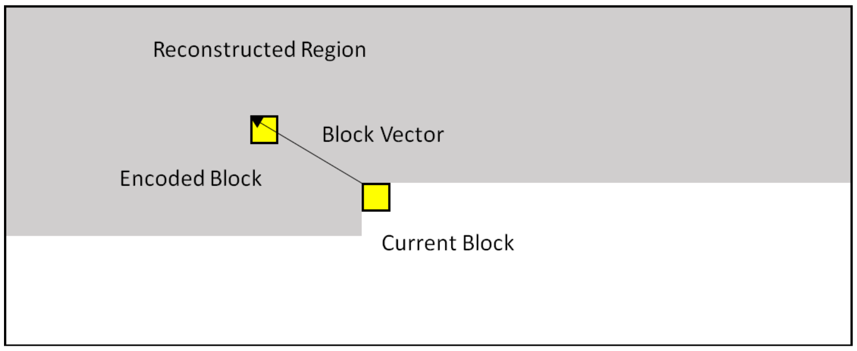


Figure 1.3: IBC mode in SCC. The current CU is predicted from a block in the reconstructed region.

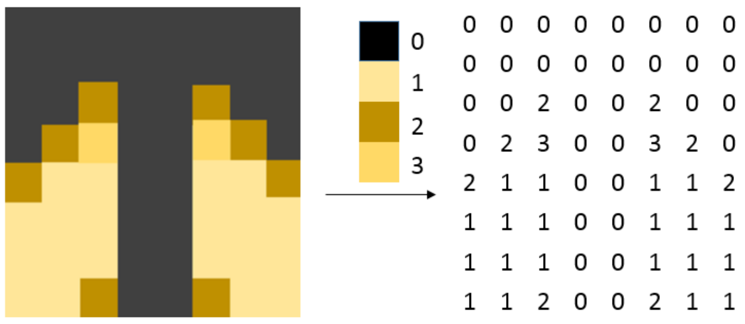


Figure 1.4: PLT mode in SCC. CUs are represented by base colors and an index map.

merge and skip mode for inter-prediction in HEVC, where IBCMerge signals residues to a SCC decoder but IBCSkip does not. IBCSearch finds the best matched block in the reconstructed region of the current frame for 16×16 CUs and 8×8 CUs, and it provides different searching strategies for different CU and prediction unit (PU) sizes. The searching strategies include the full vertical and horizontal searches, local vertical and horizontal 1D searches, and 2D pre-defined area search. For a 16×16 CU, only the 2N×2N PU with full vertical and horizontal searches are performed. It is due to the fact that a large CU size tends to have fewer repeated patterns within the same frame. For an 8×8 CU, additional PU sizes are allowed to find more repeated patterns. If it is a N×2N PU, only full vertical and horizontal searches are carried out. If it is a 2N×N or 2N×2N PU, local vertical and horizontal 1D searches, and 2D pre-defined area search within the current CTU and left CTU are performed. Besides, a hash value based fast searching method is implemented for 8×8 CUs with 2N×2N PUs, where only blocks having the same hash value as the current CU are searched. Therefore, IBCSearch comes with the highest computational complexity among the three steps. PLT mode is designed to improve the encoding efficiency for CUs with limited colors, as shown in Figure 1.4. Several representative colors in a CU are selected to form a palette table. Then an index map is generated to indicate the index of the representative color for each pixel location. In the encoding process of a CU, Intra mode, IBC mode and PLT mode are exhaustively

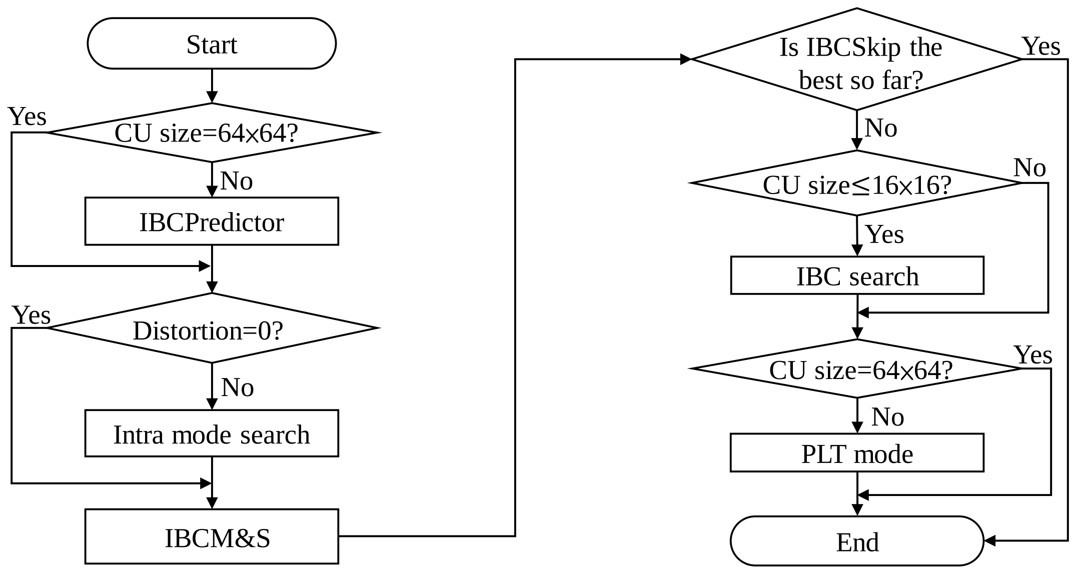


Figure 1.5: Encoding procedure implemented in SCM.

checked, and the encoding procedure implemented in the HEVC-SCC reference software, Screen Content Model (SCM), is shown in Figure 1.5. At the beginning, IBCPredictor is checked for CUs with sizes from 32×32 down to 8×8, If the distortion is zero after checking IBCPredictor, Intra mode inherited from HEVC is skipped. Otherwise, Intra mode is checked, which includes 33 directional modes, plus planar and DC modes. Then it is followed by checking block vector (BV) predictors of IBCM&S for all CUs. If IBCSkip is selected as the best mode among IBCPredictor, Intra, and IBCM&S, further mode searching is terminated. Otherwise, if the best mode is IBCPredictor, Intra or IBCMerge, the following IBCSearch and PLT modes are checked. Specifically, only CUs with sizes of 16×16 and 8×8 need to check IBCSearch. Finally, PLT mode is checked for CUs with sizes from 32×32 down to 8×8. In the mode searching process, the coding performance of each mode is evaluated by calculating a Lagrange rate-distortion (RD) cost function, $J_{mode}$, as

$$J_{mode} = D_{SSE} + \lambda \times B_{mode} \tag{1.1}$$

where $D_{SSE}$ denotes the sum of the squared error between the current CU and its reconstructed CU, λ is a Lagrange multiplier and $B_{mode}$ is the actual encoding bits for signaling the mode and the residues. The mode with the smallest RD cost is selected as the best mode of the CU, and its RD cost is represented as $J_{CU}$. All CU partitions in a CTU need to go through this mode searching process, and the final partitioning structure of a CTU is selected as the one with the smallest RD cost, and it is involved in the final encoding bitstream.

By adding coding tools specially designed for SCBs, SCC significantly improves the coding efficiency and it is expected to enhance HEVC to compress screen content videos. For example, the SCC reference software SCM version 4.0 achieves over 50% Bjøntegaard delta bitrate (BDBR) [29] compared with the HEVC reference software HM-16.4 for typical screen content sequences [13].

## 1.3 Limitations of SCC for Intra Coding

SCC achieves the coding gain than HEVC mainly from its adoption of more mode candidates. However, the additional mode candidates induce very high computational complexity. To analyze the additional complexity brought by IBC and PLT, we encode the first 100 frames of testing sequences with quantitation parameters (QPs) of 22, 27, 32, and 37 under all intra (AI) configuration, which are the settings recommended by common test conditions (CTC) for SCC [30]. The testing sequences are YUV 4:4:4 sequences which include camera-captured content (CC), animation (A), text and graphics with motion (TGM), and mixed content (M). In this thesis, reference software HEVC Test Model Version 16.12 Screen Content Model Version 8.3 (HM-16.12+SCM-8.3, hereafter called SCM-8.3 for the sake of simplicity) [31] is used. The test platform used for simulations was a HP EliteDesk 800 G1 computer with a 64-bit Microsoft Windows 10 OS running on an Intel Core i7-4790 CPU of 3.6 GHz and 32.0 GB RAM. Table 1.1 tabulates the BDBR and the encoding time difference, ΔTime, of the conventional SCC

Table 1.1: BDBR and ΔTime of SCC compared to SCC with both IBC and PLT disabled.

| Sequences | Type | BDBR | ΔTime |
|---|---|---|---|
| BasketballScreen | M | -52.47 | 87.50 |
| MissionControlClip2 | M | -48.78 | 99.70 |
| MissionControlClip3 | M | -66.92 | 87.90 |
| ChineseEditing | TGM | -60.68 | 104.04 |
| Console | TGM | -69.34 | 52.41 |
| Desktop | TGM | -83.47 | 66.60 |
| FlyingGraphics | TGM | -63.79 | 89.28 |
| Map | TGM | -25.87 | 143.37 |
| Programming | TGM | -52.96 | 73.50 |
| SlideShow | TGM | -23.38 | 52.94 |
| WebBrowsing | TGM | -80.32 | 66.16 |
| Robot | A | -2.67 | 112.42 |
| EBURainFruits | CC | -0.08 | 91.35 |
| Kimono1 | CC | 0.03 | 74.30 |
| **Average** | | **-45.05** | **85.82** |

increased by IBC and PLT compared to SCC with both IBC and PLT disabled. ΔTime is defined as the percentage difference of the encoding time with and without IBC and PLT modes. It can be observed that the BDBR is decreased by 45.05% on average and up to 83.47% which indicates that IBC and PLT are efficient coding tools for SC. However, the encoding time is also increased largely by 85.82% on average and up to 143.37% when both IBC and PLT are enabled.

## 1.4 Motivation and Objectives

In the encoding process, a SCC encoder needs to solve two problems:

1. Mode decision: Which mode should be chosen to encode the current CU to minimize RD cost?
2. CU partitioning decision: Should the current CU be partitioned into smaller sub-CUs to minimize RD cost?

To solve the above problems, the original SCC encoder adopts an exhaustive search method, where all mode candidates and all CU partitions are searched, and then the best coding structure is selected by comparing RD cost. For time-critical applications, the exhaustive mode searching approach implemented in a SCC encoder is not practical. Therefore, it is desired that the mode candidates and CU sizes can be checked adaptively according to the content being encoded in order to speed up the encoding process.

On the other hand, the full adoption of SCC may take several years while HEVC is still a widely used video compression standard. It motivates the development of the HEVC to SCC transcoder. First, there are a significant number of legacy screen content videos encoded by HEVC, and it is necessary to convert them from HEVC to SCC to achieve low-cost storage. Second, a transcoder is desirable to alleviate the traffic load between clients and clouds when users upload videos to a cloud server. Since the bandwidth resources are very expensive, it is necessary for users to convert HEVC bitstreams into SCC bitstreams with higher compression ratio for screen content videos. When users download videos from the cloud server, they can either use a device with a SCC decoder or let the server convert the bitstreams back to HEVC. Video transcoding can always be done by using a conventional brute-force transcoder (CBFT), which contains an original decoder and an original encoder. As shown in Figure 1.6, CBFT decodes the incoming HEVC bitstream first, and then completely re-encodes the decoded video into a SCC bitstream. Although this approach can be applied to any heterogeneous transcoding tasks with high RD performance, it brings high computational complexity due to the exhaustive search method. Therefore, it is desired that the decoder side information can be collected to simplify the re-encoding process of CBFT, as illustrated in the upper part of Figure 1.6.

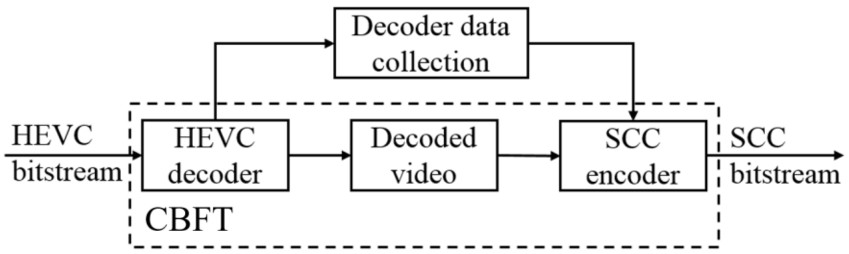


Figure 1.6: HEVC to SCC transcoder structure.

## 1.5 Organization of This Thesis

This thesis is divided into seven chapters. Prior to the description of the objectives and the main contributions in this thesis, Chapter 2 gives a broad literature review of video coding techniques that are related to this work. The new coding tools in SCC beyond HEVC are then introduced. Afterwards, a review of the previous fast video encoding works is given. At the end of this chapter, a review of the previous fast video transcoding works is also presented.

In Chapter 3, an online learning based fast SCC encoding algorithm is presented. To eliminate the checking of unnecessary modes and CU partitions, content-denpendent rules are derived according to the content being encoded. A scene change detection method is adopted to update the content-denpendent rules by obtaining the learning statistics from the first frame of a new scene.

In Chapter 4, a decision tree (DT) based fast prediction framework is designed. To get rid of the exhaustive mode searching process, a flexible mode decision framework is proposed by inserting a classifier before checking each mode. This framework facilitates the use of dynamic features containing the unique intermediate coding information of a coding unit, and it can make sepreate decision for IBC and PLT modes such that they are not grouped together to be checked for a SCBs. Therefore, computational complexity can be reduced.

In Chapter 5, a deep learning based fast prediction network DeepSCC is presented, which contains two parts, DeepSCC-I and DeepSCC-II. For CTUs having different content as their collocated CTUs, i.e., the sum of absolute differences (SAD) between the current CTU and its collocated CTU is larger than 0, they are called as dynamic CTUs and their sample values are fed to DeepSCC-I. Otherwise, they are called as stationary CTUs and their sample values are fed to DeepSCC-II. Since there exists strong correlations between the optimal modes of a stationary CTU and its collocated CTU, DeepSCC-II additionally utilizes the optimal mode maps of the collocated CTU to further reduce the computational complexity.

In Chapter 6, a fast HEVC to SCC transcoder (FHST) is presented to migrate the legacy screen content videos from HEVC to SCC. First, the CU partition in SCC is early terminated by utilizing the CU depth level collected from the HEVC decoder. Second, various features from 4 categories are obtained to improve the mode decision accuracy, which are the features from the HEVC decoder, static features, dynamic features, and spatial features. With the help of these features, a flexible transcoding structure is proposed to make early mode decisions by using DTs.

Chapter 7 is devoted to a conclusion of the work herein, and we summarize the contributions of the thesis. Suggestions are also included for further research in this area.

# Chapter 2 Literature Review

## 2.1 Background Research

To improve the compression rate of digital videos, various video coding standards such as H.264/AVC and HEVC have been proposed. These standards are designed to reduce the video size by eliminating data redundancy. For example, HEVC employs 33 intra-prediction directions and DC plus planer mode to predict the content of a CU based on spatial neighboring blocks. Nowadays, the proliferation of applications that use video devices to display a mixed content of NIBs and SCBs leads to the development of SCC extension. To further reduce the data redundancy beyond HEVC, SCC is developed as an extension of HEVC by including some additional coding tools. Consequently, more computational efforts are needed in data redundancy elimination. Therefore, fast algorithms for reducing the computational complexity in video coding are desired. In the literature, many efforts have been devoted to accelerating the encoding process.

This chapter is organized as follows. In the first section, a brief description of some fundamental concepts about video coding is presented, with an emphasis on the coding structure of HEVC. Then, we present the new coding tools in SCC beyond HEVC. Next, various works aimed at the complexity reduction in the encoding problem of HEVC and SCC are described. The last section reviews several existing fast video transcoding algorithms among various video standards.

## 2.2 Digital Video Compression Fundamentals

Digital video is stored and transmitted in digital form, which requires a large amount of data. Consequently, the cost of storing and transmitting a raw video is extremely high. To overcome the drawback of the raw video, spatial and temporal redundancy can be utilized to reduce the video size with negligible quality loss. Various video coding standards have been developed by ITU-T VCEG and ISO/IEC MPEG separately or jointly, such as H.261, MPEG-1 Part 2, H.262/MPEG-2 Part 2, H.263, MPEG-4 Part 2, H.264/AVC, and the recent HEVC. HEVC mainly inherits coding features from H.264/AVC, and then it induces the flexible quadtree-based block partitioning scheme to further improve coding efficiency beyond H.264/AVC. It is reported that HEVC achieves a 50% bitrate reduction over H.264/AVC with similar video quality [32]. As an extension of HEVC which is specially developed for screen content videos, SCC further reduces 50% bitrate reduction over HEVC with similar video quality for them. The block diagram of a HEVC/SCC video encoder is illustrated in Figure 2.1. Intra-prediction and inter-prediction are employed to remove the spatial and temporal redundancy, respectively. Then, the prediction error is signaled to a decoder by entropy coding after transform and quantization.

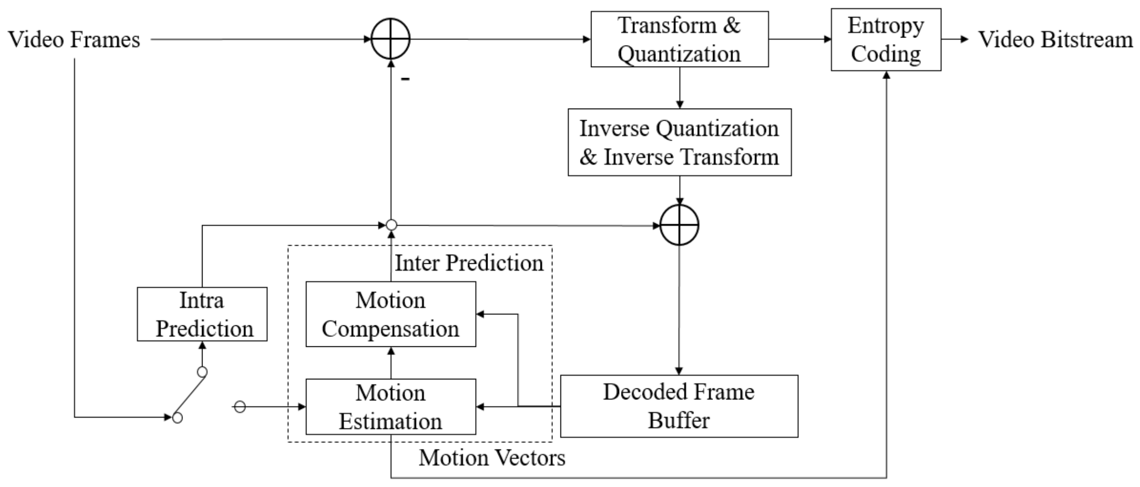


Figure 2.1: Block diagram of a HEVC/SCC video encoder.

### 2.2.1 Intra-Prediction for Spatial Redundancy Elimination

Intra-prediction is an efficient tool to reduce the spatial redundancy within a frame, where the content of a CU is predicted according to the reconstructed spatially neighboring samples. HEVC employs 33 different directional orientations and DC mode plus planner mode to reduce the redundancy, and they are referred to as Intra mode in this thesis. Then, only the residues between the current block and its predicted block are encoded. An example of a CU encoded by Intra mode is shown in Figure 2.2.

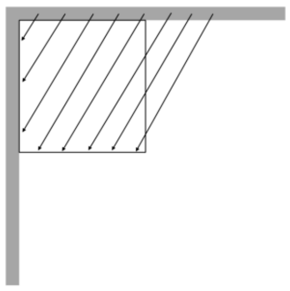


Figure 2.2: An example of a CU encoded by Intra mode.

### 2.2.2 Inter-Prediction for Temporal Redundancy Elimination

Inter-prediction is another efficient tool in HEVC, which removes temporal redundancy between adjacent frames. Instead of using reconstructed spatially neighboring samples, the reference pixels are from a previously coded frame. Inter-prediction is performed based on the assumption that objects are in translational motion among adjacent frames so that a CU in the current frame can find a similar block in the previously coded frame. HEVC performs Motion estimation (ME) to find the best matched block for the current CU, and the relative location between the current CU and the reference block is denoted by a motion vector (MV). ME can be done by merge & skip mode or advanced motion vector prediction (AMVP). Merge & skip mode derives several motion predictors from spatial and temporal neighboring blocks, and then directly uses them to find the best matched block. More specifically, merge mode signals residues to the decoder but skip mode does not. On the other hand, AMVP first derives a motion predictor from two spatial neighboring blocks, and then it further searches the best matched block around the motion predictor. Unlike merge & skip mode, the motion vector of AMVP is differentially coded with the motion vector predictor. After performing motion compensation (MC), only the residual block between the current block and its predicted block is encoded.

## 2.3 New Coding Tools in SCC Beyond HEVC

To improve the coding efficiency for screen content videos, SCC adopts four major coding tools beyond HEVC, which are IBC mode [15]–[21] and PLT mode [22]–[28], adaptive color transform (ACT) [33]–[37], adaptive motion compensation precision (AMCP) [38]–[42]. IBC and PLT modes significantly improve the coding efficiency of SCC but also induce great computational complexity.

IBC was first proposed in the contest of AVC/H.264 [43], but then removed because it is not efficient for camera-captured videos. SCC includes IBC as a coding tool similar to inter-prediction, but its search region is the reconstructed area in the current frames. The syntax for IBC mode is unified with inter-prediction but it adopts a different searching strategy. If IBC mode is chosen, a BV is signaled to denote the relative location between the match block and the current CU.

PLT mode is an effective mode for CUs with limited colors. Several representative colors in a CU are selected as base colors to form a palette table. Each entry in the palette table consists of three components of YUV or RGB. Those color values not in the palette table are treated as escape colors. Then, a palette index map is generated to send indices for base colors and escape colors. For base colors, only the indices in the palette are encoded. For escapes colors, the quantized color values are directly encoded.

ACT is developed to remove the inter-color component redundancy. A residual block is adaptively converted into a different color space, i.e., YCgCo. The RD cost function is employed to decide whether to code the residual signal in the original RGB/YUV color space or in the converted YCgCo color space.

AMCP adaptively switches motion vectors between full and fractional resolutions. Unlike screen camera-capture content where motion is continuous and motion vectors in the fractional resolution are necessary, computer-generated content usually has a granularity of one or more samples. By using AMCP, it eliminates the need to signal fractional motion vectors for computer-generated contents.

## 2.4 Complexity Reduction in the Encoding Problem

To simplify the encoding process of HEVC, a fast CU partitioning algorithm was proposed in [44] by using Bayesian decision rule. The CU partitioning process is early terminated by using joint online and offline learning. In [45], a fast mode decision algorithm was proposed to predict the RD cost and bit cost of a CU based on the statistical analysis. Then unnecessary modes are skipped according to the prediction. In [46], both the mode searching process and the CU partitioning process are terminated adaptively by analyzing the RD cost of the current CU. Although they work well for computational complexity reduction of HEVC, they are not suitable for SCC in which new coding modes such as IBC and PLT have been adopted.

To reduce the computational complexity of SCC, fast mode searching algorithms were designed in [47]–[49], and fast CU size decision algorithms were suggested in [50]–[52]. Then, various algorithms were integrated to make both fast mode decision and CU size decision in [53]–[56]. In [47], a new mode was proposed to fill a noiseless smooth CU by its boundary samples. In [48], a hash value is calculated to adaptively skip the local search process in IBC mode. In [49], IBC mode is skipped for zero activity CUs and low gradient CUs. In [50], a neural network based fast algorithm was proposed to make fast CU size decision by utilizing features that describe CU statistics and sub-CU homogeneity. However, high RD performance loss is induced by this approach. In [51], for static regions, collocated CU depth and mode information are utilized to predict the current CU size. Besides, an approach with the adaptive searching step was proposed to simplify the block matching process of IBC mode. However, this algorithm is not suitable for screen content videos with many dynamic regions. In [52], a fast CU size decision algorithm based on entropy was proposed. Some rules are firstly set based on entropy to make CU partitioning decision, and then the coding bits are used to improve the decision accuracy. The algorithms in [53]–[56] are mainly based on the assumption that NIBs select Intra mode while SCBs select IBC and PLT modes. They then classify CUs into NIBs and SCBs to make fast mode decision. In [53], a DT-based classifier was firstly designed to classify CUs into NIBs and SCBs, so that NIBs only check Intra mode while SCBs check both IBC and PLT modes. Besides, to speed up the encoding of NIBs, two classifiers were designed to predict the Intra mode direction from 35 prediction modes and early terminate the partitions of NIBs, respectively. In [54], a CU type classification is performed by CU content analysis. While IBC and PLT modes are skipped for some smooth NIBs, all modes are checked for SCBs and non-smooth NIBs. Then the depth information of temporal and spatial neighboring CUs, as well as coding bits, are utilized to make fast CU size decision. In [55], Intra mode is firstly checked for all CUs with 2N×2N PUs, and then an early CU partitioning decision is made. If a CU is classified as a partitioning CU, it directly goes to the next depth level. Otherwise, it is further classified as a SCB or NIB. If it is a SCB, both IBC and PLT modes are checked. If it is a NIB, only Intra mode for N×N PUs in the depth level of 3 is tested. In [56], neural network-based classifiers are trained to classify CUs in NIBs and SCBs. Again, IBC, PLT modes and a subset of Cintra mode are checked for SCBs, while only Cintra mode is checked for NIBs. Then, various heuristic rules based on the information from spatial and temporal adjacent CUs is proposed to make fast CU partitioning decision. Although the methods in [53]–[56] provide better performance compared with the previous works, they mainly focus on the fast encoding of NIBs. For SCBs, either both IBC and PLT modes or all modes need to be checked. Therefore, it is desired that mode candidates can be further reduced for SCBs.

## 2.5 Complexity Reduction in the Transcoding Problem

In the literature, transcoding techniques can be divided into two categories: homogeneous transcoding and heterogeneous transcoding. Homogeneous transcoding refers to the conversion within the same format to meet a new functionality, such as different bit rates [57], different frame rates [58]–[60], different spatial resolutions[61], or even the insertion of new information such as watermarking [62] and error resilience layers [63], [64]. Heterogeneous transcoding refers to the bitstream conversion between different formats, and the HEVC to SCC transcoding belongs to this category. For heterogeneous transcoding, many fast transcoding algorithms have been proposed for different tasks, such as MPEG-2 to H.264 transcoding [65], [66], MPEG-2 to HEVC transcoding [67] and H.264 to HEVC transcoding [68]–[71]. These transcoders all focus on the fast CU partitioning decision because of different CU partitioning structures of the various standards. However, the fast HEVC to SCC transcoding is different from the previous transcoding problem due to the introduction of the new IBC and PLT modes. Therefore, new challenges are introduced in the HEVC to SCC transcoding problem.

To reduce the computational complexity of a HEVC to SCC transcoder, one possible way is to use various fast encoding algorithms [47]–[56] to replace the original SCC encoder of CBFT in Figure 1.6. These fast algorithms all utilize the SCC encoder side information only, and they are not optimal when applied to the HEVC to SCC transcoding problem. On the one hand, the information from SCC encoder side contains noise due to the lossy encoding and decoding of HEVC, and it may lead to high RD performance loss. On the other hand, it is desired that the information from the HEVC decoder side can be utilized to improve decision accuracy.

In the literature, there is only one paper [72] studying the fast transcoding scheme of HEVC to SCC, and it was designed for screen content videos in YUV 4:4:4 format. First, it directly maps the optimal CU size from HEVC to SCC and classifies CUs into non-partitioning CUs and partitioning CUs. For partitioning CUs, it utilizes a CU type classifier to further classify them into SCBs and NIBs like other fast mode decision algorithms [53]–[56]. SCBs check both IBC and PLT modes before going to the next depth level, while NIBs directly go to the next depth level. For non-partitioning CUs, only Intra mode is checked and then CU partitions are terminated. Besides, some thresholds are set to skip remaining modes and CU partitions if the bit cost of a CU is small. However, it has two drawbacks. First, it only utilizes some static features describing the current CU content to perform classifications while ignoring the information from the intermediate encoding stage and neighboring CUs. Therefore, the prediction accuracy is not high, and all mode candidates need to be checked for 8×8 CUs. Second, it treats the decision for IBC and PLT modes the same like other fast mode decision algorithms [53]–[56], where both IBC and PLT modes are checked for SCBs. In fact, a SCB will only select one mode from IBC and PLT modes, and higher re-encoding time reduction can be provided if mode candidates are further reduced for SCBs.

## 2.6 Chapter Summary

To reduce the computation complexity of the SCC encoding and transcoding problems, many recently proposed algorithms have been reviewed in this chapter. We started this chapter by reviewing the compression techniques employed in current video standards. Next, four new coding tools in SCC beyond HEVC were introduced. Specially, IBC and PLT modes significantly improve the coding efficiency for screen content videos, but they also induce great computational complexity. Afterward, various works aimed at reducing the encoding time of SCC were presented. The existing algorithms always perform simple CU type classification for making mode decision. Therefore, at least two modes, IBC and PLT, need to be checked for SCBs. To further reduce the encoding time, the classification between IBC-coded SCBs and PLT-coded SCBs should be made. In addition, the existing algorithms related to the fast video transcoding problem were reviewed, and there is only one work aimed at accelerating the transcoding of HEVC to SCC. It shows that there is plenty of room for improvement. Therefore, in the following chapters, we examine the possibility of further accelerating the encoding and transcoding process by various machine learning algorithms.

# Chapter 3 Determinations on Coding Structure by Online Learning

## 3.1 Introduction

It is well known that the characteristics of successive frames in a video are usually very similar. By taking this into account, content dependent rules can be considered to predict the optimal modes and CU partitions of the current frame. In this chapter, a fast intra-prediction algorithm by content analysis and dynamic thresholding is presented, where the prediction models are tailor-made according to the video content and QP. The organization of this chapter is as follows. Section 3.2 presents the motivation of this research work. Section 3.3 describes the details of the proposed coding scheme by content analysis and dynamic thresholding. Section 3.4 evaluates the performance of the proposed scheme, where simulation results are provided in BDBR and ΔTime. Finally, conclusions are given in Section 3.5.

Parts of the contents of this chapter are extracted from our published work [73]:

- **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Fast Intraprediction for High-Efficiency Video Coding Screen Content Coding by Content Analysis and Dynamic Thresholding," Journal of Electronic Imaging, vol. 27, no. 5, pp. 053029-1–053029-18, October 2018.

## 3.2 Motivation for Content Dependent Rules

Different sequences usually show very different characteristics, and they have different mode decision and CU partitioning decision. On the contrary, a video sequence

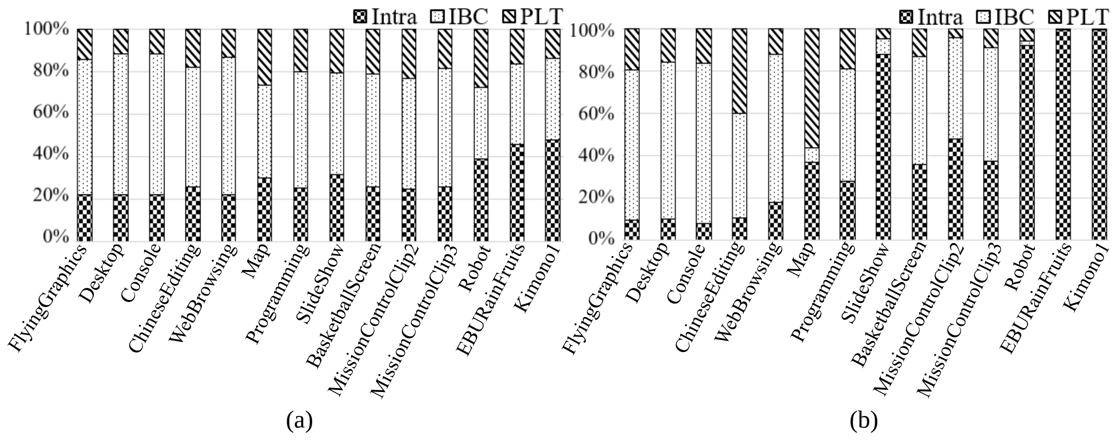


Figure 3.1: (a) Encoding time and (b) mode distributions at the depth level of 2 and QP of 32.

is usually composed of a series of similar frames, and they exhibit similar characteristics. If content dependent rules can be derived for different sequences, encoding time can be further reduced.

### 3.2.1 Content Dependent Mode Decision

To analyze the mode decision in SCC, the encoding time distribution and mode distribution at the depth level of 2 with QP of 32 are shown in Figure 3.1. It is noted that similar distributions can be observed with other QPs and depth levels. From Figure 3.1, it is observed that the distributions vary remarkably for different sequences. For example, in “Console”, Intra mode only takes up 7.55% of the mode distribution but it costs 22.09% of the encoding time. In “EBURainFruits”, IBC and PLT modes take up 38.03% and 15.94% of the encoding time while only negligible percentage of CUs (0.21% and 0.06%, respectively) select them, respectively. For “Map”, IBC mode only takes up 6.72% of the mode distribution while it costs 43.64% of the encoding time. Besides, it is observed that although IBC mode can be skipped for many CUs of “Map”, PLT mode is a frequently selected mode which takes up 56.51% of total modes. Although CU type classification algorithms such as [53]–[56] work well for “EBURainFruits” by skipping both IBC and

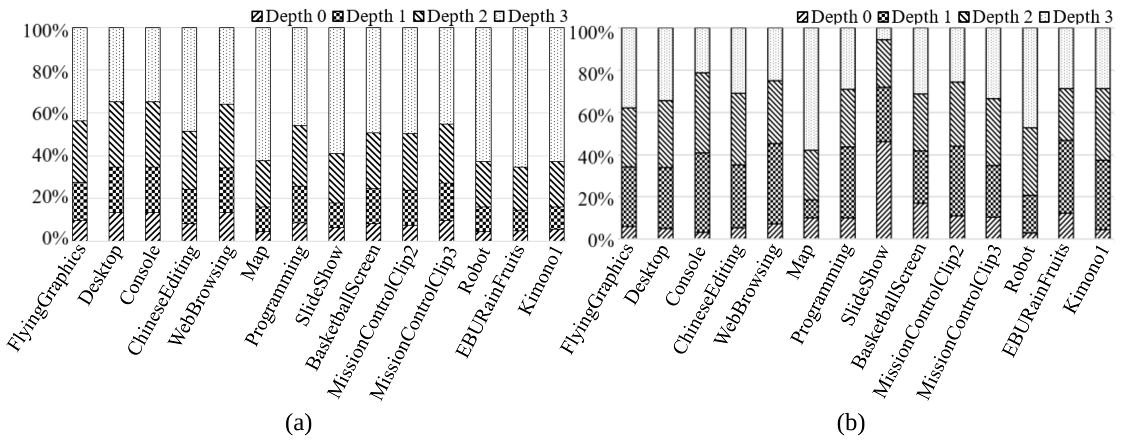


Figure 3.2: (a) Encoding time and (b) CU depth level distribution at QP of 32.

PLT modes for NIBs, they are not optimal for “Map” because their fixed rules treat IBC and PLT modes equally. The variability of distributions among various sequences in Figure 3.1 implies that content dependent rules should be beneficial for making fast mode decisions in addition to performing CU type classifications.

### 3.2.2 Content Dependent CU Size Decision

To analyze the CU size decision in SCC, the encoding time and optimal CU depth level distributions of all test sequences are presented in Figure 3.2 with QP of 32. As mentioned before, similar distributions can be observed with other QPs. It is observed that the computational complexity increases as the CU depth level gets higher, and the CU depth level of 3 takes up over 50% of the total encoding time for many sequences such as “Map”, “SlideShow”, “Robot”, “EBURainFruit”, and “Kimnono1”. However, the optimal CU depth level distributions of these sequences differ a lot. For example, 58.02% of CUs in “Map” select the depth level of 3, while only 5.65% of CUs in “SlideShow” are encoded by the depth level of 3. Again, it is desirable that content dependent rules can be derived to early terminate the CU partitioning process. For sequences with many CUs selecting lower depth levels like “SlideShow”, the content dependent rules with an adaptive thresholding technique will terminate more CU partitions to pursuit higher encoding time reduction.

## 3.3 Proposed Coding Scheme by Content Analysis and Dynamic Thresholding

Among those coding modes, Intra mode works well for NIBs while PLT and IBC modes are specially designed based on the characteristics of the repeated pattern and limited colors for SCBs. Therefore, PLT and IBC modes can be skipped for NIBs while Intra mode can be skipped for SCBs. In this section, this CU type classification will be carried out through a rough CU classification, followed by a fine-granular CU classification. Then, content dependent rules with adaptive thresholding based on background color ratio and RD cost are proposed to further speed up mode decisions and skip redundant partitions for efficiently encoded CUs, respectively.

### 3.3.1 Rough CU Classification

There are usually many sharp edges in a SCB while a NIB tends to be smoother. Based on this observation, high gradient pixels are adopted to classify CUs into rough NIBs (RNIBs) and rough SCBs (RSCBs). The use of high gradient pixels is first employed in [74] for an early version of PLT, which utilizes the number of high gradient pixels *HGN* in a CU to decide whether PLT should be tested for the CU. This is very similar to our purpose to distinguish regions with shape edges and smooth pixels for RSCBs and RNIBs, respectively. A pixel is defined as a high gradient pixel if the luminance difference of the current pixel $Y_{i,j}$ and one of the neighboring pixels $Y_{i\pm1,j}$ and $Y_{i,j\pm1}$ located at 0°, 90°, 180° and 270° is larger than a threshold $TH_S$

$$\left|Y_{i,j} - Y_{i\pm1,j}\right| > TH_S \quad \text{or} \quad \left|Y_{i,j} - Y_{i,j\pm1}\right| > TH_S \tag{3.1}$$

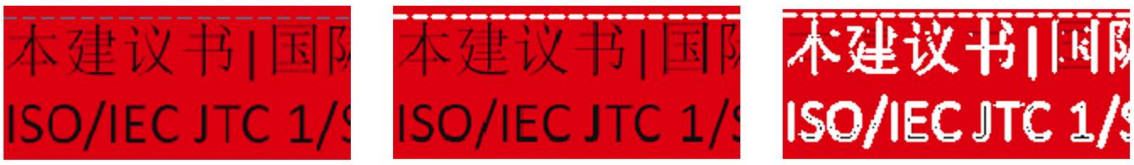


Figure 3.3: (a) Original content in "ChineseEditing" and its high gradient pixels detected by (b) luminance component, and (c) all components. High gradient pixels are shown by white pixels.

where $i$ and $j$ denote the row and column indices of the pixel. $TH_S$ is a threshold controlling the sharpness of the edges for detection, which is experimentally set to 64. Furthermore, if *HGN* in a CU is larger than *TH_HGN*, it is classified as a RSCB in the proposed algorithm. Otherwise, it is classified as a RNIB.

As shown in Figure 3.3(b), some sharp edges cannot be detected by high gradient pixels that only use the luminance component. But we find that they can be well detected if all components of a pixel are taken into consideration, which is clearly shown in the example shown in Figure 3.3(c). Therefore, instead of using the luminance component only, our proposed algorithm further improves it by utilizing all color components to avoid the missed detection of sharp edges. In the proposed algorithm, high gradient pixels are firstly detected by using each color component separately with $TH_S$, and then $HGN$ is set to the maximum number of high gradient pixels detected by the three components

$$HGN = max(HGN_Y, HGN_{Cr}, HGN_{Cb}) \tag{3.2}$$

where *max*(·) is the maximum function to return the largest value from a set of data. $HGN_Y$, $HGN_{Cr}$, and $HGN_{Cb}$ are the number of high gradient pixels detected by the components of Y, Cr and Cb, respectively. Since a larger CU usually has a larger number of high gradient pixels, the value of *TH_HGN* is set according to CU sizes of 2N×2N pixels as

$$TH\_HGN = \frac{2N\times 2N}{256}\times 4 \tag{3.3}$$

We perform the rough CU classification as

Table 3.1: Average mode distributions in each depth level for RSCBs and RNIBs.

| CU Type | Depth Level | PLT/IBC (%) | Intra (%) |
|---|---|---|---|
| RSCB | 0 | 47.04 | 52.64 |
|  | 1 | 87.91 | 12.09 |
|  | 2 | 89.14 | 10.86 |
|  | 3 | 80.65 | 19.35 |
| RNIB | 0 | 6.29 | 93.71 |
|  | 1 | 20.31 | 79.39 |
|  | 2 | 34.06 | 65.94 |
|  | 3 | 29.34 | 70.66 |

$$\mathrm{CU} \in \begin{cases} \mathrm{RSCB}, if\ HGN\ > TH\_HGN \\ \mathrm{RNIB}, if\ HGN\ \leq TH\_HGN \end{cases} \tag{3.4}$$

Thus, if *HGN* in a CU is larger than *TH_HGN*, it is classified as a RSCB in the proposed algorithm. Otherwise, it is classified as a RNIB. To analyze the mode distributions in terms of RNIBs and RSCBs, we encoded all 14 SCC standard sequences. The average mode distributions for RSCBs and RNIBs with QP of 32 are shown in Table 3.1, and similar distributions can be observed with other QPs. It is observed that RSCBs tend to select IBC or PLT mode while RNIBs tend to select Intra mode. However, it is also observed that the accuracy of the CU type classification is not high enough. For example, 52.64% of RSCBs still select Intra mode at the depth level of 0. Therefore, instead of making early mode decisions according to the rough CU type directly, it is adopted as a pre-processing step and then two fast mode decision techniques are built on the top of the rough CU classification result.

### 3.3.2 Fine-granular Coding Unit Classification

In this sub-section, the background color information in a CU is used to perform a more fine-granular CU type classification. For a CU, the background color is defined as the color with the highest occurrence frequency within the CU by considering all the three components. As shown in Figure 1.1, there is usually a large area filled with a background color for a SCB. If the background color of the current CU exists in all its four sub-CUs, it is more likely to be a SCB. Otherwise, if the background color of the current CU disappears in one of its sub-CUs, it is more likely to be a NIB. Therefore, by utilizing both the rough CU classification result and the background color, incoming CUs are further classified as SCBs, NIBs and Uncertain CUs as

$$\mathrm{CU} \in \begin{cases} \mathrm{SCB}, if\ \mathrm{CU} \in \mathrm{RSCB}\ \text{and}\ BC \in \{S_1 \cap S_2 \cap S_3 \cap S_4\} \\ \mathrm{NIB}, if\ \mathrm{CU} \in \mathrm{RNIB}\ \text{and}\ BC \notin \{S_1 \cap S_2 \cap S_3 \cap S_4\} \\ \text{Uncertain CU}, \qquad otherwise \end{cases} \tag{3.5}$$

where $BC$ denotes the background color of the current CU. $S_1$, $S_2$, $S_3$, and $S_4$ denote the sample spaces of its four sub-CUs. It should be noted that $\{S_1 \cap S_2 \cap S_3 \cap S_4\}$ would contain $BC$ only if all the sub-CUs contain the background color of the current CU. Therefore, a CU is classified as a SCB if it is a RSCB and its four sub-CUs contain the background color of the current CU. On the contrary, a CU is classified as a NIB if it is a RNIB and the background color of the CU disappears from at least one of its four sub-CUs. Otherwise, it is treated as an Uncertain CU and no early mode decision is made. Since IBC mode and PLT mode are designed for SCBs, they are skipped for NIBs. Similarly, Intra mode is designed for NIBs and it is skipped for SCBs. To evaluate the accuracy of the fine-granular CU classification technique, we analyzed the hit rate of mode decision by calculating the percentage of the CUs whose optimal mode from the original SCM-8.3 is not skipped by using the proposed technique. Table 3.2 shows the hit rate for all test sequences under QP of 32, and similar results can be observed with other QPs. It is observed that the hit rates vary from 87.96% to 100% for different sequences and depth levels. Besides, the average hit rates for the depth levels of 0, 1, 2, and 3 are 96.04%, 97.33%, 96.89% and 94.70%, respectively. Considering that IBC mode is a very efficient mode for encoding small NIBs with repeated patterns, we still check IBC mode for NIBs at the depth level of 3. By adding this condition, it is observed in Table 3.2 that the hit

Table 3.2: Hit rate of mode decision for various sequences by the fine-granular CU classification technique.

| Sequences | Depth=0 (%) | Depth=1 (%) | Depth=2 (%) | Depth=3 (%) | Depth=3 with additional IBC checking (%) |
|---|---|---|---|---|---|
| BasketballScreen | 92.97 | 93.69 | 94.48 | 92.67 | 97.73 |
| MissionControlClip2 | 96.80 | 94.41 | 94.07 | 92.73 | 97.95 |
| MissionControlClip3 | 93.20 | 95.38 | 93.72 | 95.32 | 98.25 |
| ChineseEditing | 87.96 | 96.66 | 97.47 | 96.45 | 98.24 |
| Console | 92.93 | 99.26 | 99.68 | 98.44 | 98.61 |
| Desktop | 98.19 | 99.64 | 98.25 | 97.20 | 99.43 |
| FlyingGraphics | 99.59 | 99.78 | 99.15 | 93.56 | 95.50 |
| Map | 98.43 | 97.20 | 94.73 | 92.13 | 96.90 |
| Programming | 90.20 | 94.12 | 96.60 | 94.23 | 97.63 |
| SlideShow | 97.30 | 96.32 | 93.79 | 89.20 | 93.98 |
| WebBrowsing | 97.60 | 97.11 | 98.97 | 97.88 | 99.21 |
| Robot | 99.94 | 99.34 | 96.48 | 91.31 | 98.54 |
| EBURainFruits | 99.50 | 99.66 | 99.58 | 98.00 | 99.87 |
| Kimono1 | 100 | 100 | 99.52 | 96.70 | 99.98 |
| **Average** | 96.04 | 97.33 | 96.89 | 94.70 | 97.99 |

rate for the depth levels of 3 is increased to 97.99%, and our experiments shows that the increase in BDBR brought by our proposed algorithm is reduced by 0.90%.

### 3.3.3 Mode Skipping Rule with Adaptive Thresholding

In this sub-section, based on the rough CU classification result, a content dependent rule with adaptive thresholding based on background color ratio *K* is derived to perform more flexible fast mode decisions. Let *BCN* be the number of pixels with the background color of a 2N×2N CU, then *K* is defined as

$$K = \frac{BCN}{2N\times 2N}. \tag{3.6}$$

Since SCBs usually have a large area filled with a background color, they have higher values of *K* than NIBs. Therefore, CUs with high values of *K* are more likely to select IBC and PLT modes while CUs with low values of *K* tend to select Intra mode. To analyze the distributions of Intra, IBC and PLT modes in terms of *K*, the first 100 frames of

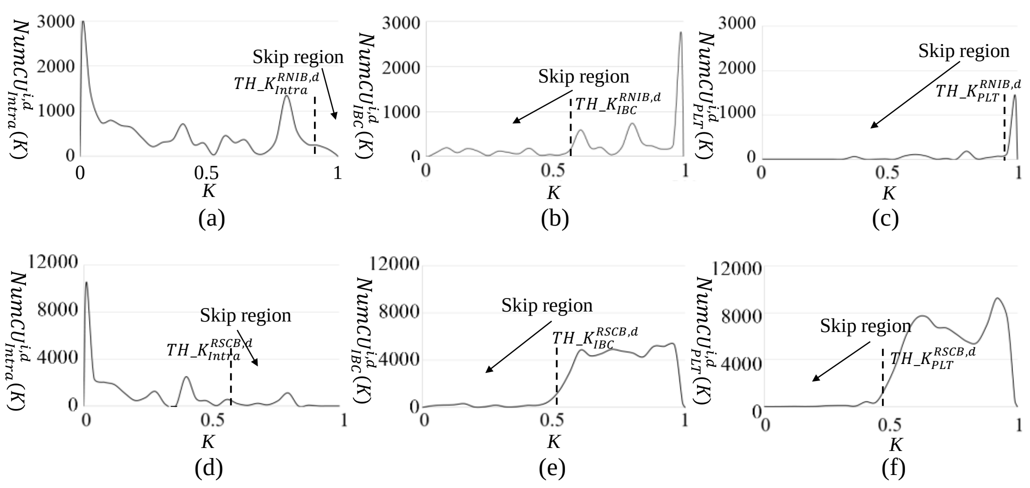


Figure 3.4: Distributions of (a) Intra mode, (b) IBC mode, and (c) PLT mode in RNIBs, and (d) Intra mode, (e) IBC mode, and (f) PLT mode in RSCBs in terms of $K$ for the first 100 frames of “WebBrowsing” encoded with QP of 32 and the depth level of 2.

“WebBrowsing” were encoded. Considering that RSCBs and RNIBs have different mode distributions, as shown in Table 3.1, the optimal mode distributions are investigated for RSCBs and RNIBs separately. Figure 3.4(a)–(c) and Figure 3.4(d)–(f) show the Intra, IBC, PLT mode distributions in RNIBs and RSCBs with QP of 32 and the depth level of 2, respectively, and similar distributions can be observed with other QPs and depth levels. It is noted that if a CU has same samples in each row or each column, it is regarded as a simple CU, as depicted in Figure 1.1. It is well known that simple CUs can be encoded by all modes efficiently with low complexity, they are excluded in Figure 3.4. It is observed that for both RNIBs and RSCBs, Intra mode is rarely selected in the region with high values of $K$, while IBC and PLT modes are rarely selected in the region with low values of $K$. More specifically, while IBC and PLT modes have similar distributions in RSCBs, they have different distributions in RNIBs. This explains why the mode skipping rule with adaptive thresholding should consider RSCBs and RNIBs separately. Compared with IBC mode, the number of RNIBs selecting PLT mode is relatively low, and they are concentrated on a narrower region with very high values of *K*. Therefore, for CUs in the depth level $d$ ($d \in \{0,1,2,3\}$) and rough type $t$ ($t \in \{\text{RNIB}, \text{RSCB}\}$), content dependent thresholds $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$, and $TH\_K_{PLT}^{t,d}$ of $K$ can be derived to perform mode skipping for Intra, IBC and PLT modes, respectively. Based on the observation from Figure 3.4(a) and Figure 3.4(d), the skip region of Intra mode is on the right side of $TH\_K_{Intra}^{t,d}$. By rounding the value of *K* down to one-hundredth of its original value, the ratio of Intra mode, $R_{Intra}^{t,d}$, to all modes in its skip region is calculated as

$$R_{Intra}^{t,d} = \frac{\sum_{K=TH\_K_{Intra}^{t,d}}^{1} NumCU_{Intra}^{t,d}(K)}{\sum_{K=TH\_K_{Intra}^{t,d}}^{1} NumCU_{All}^{t,d}(K)} \tag{3.7}$$

where $NumCU_{Intra}^{t,d}(K)$ denotes the number of Intra CUs with the background color ratio of *K*. $NumCU_{All}^{t,d}(K)$ denotes the number of all CUs with the background color ratio of *K*. On the other hand, as shown in Figure 3.4(b)–(c) and Figure 3.4(e)–(f), the skip regions of IBC and PLT modes are on the left side of $TH\_K_{IBC}^{t,d}$, and $TH\_K_{PLT}^{t,d}$. Then, for a given value of $TH\_K_{IBC}^{t,d}$ and $TH\_K_{PLT}^{t,d}$, the ratios of IBC and PLT modes, $R_{IBC}^{t,d}$ and $R_{PLT}^{t,d}$, to all modes in their own skip regions are calculated as

$$R_{IBC}^{t,d} = \frac{\sum_{K=0}^{TH\_K_{IBC}^{t,d}} NumCU_{IBC}^{t,d}(K)}{\sum_{K=0}^{TH\_K_{IBC}^{t,d}} NumCU_{All}^{t,d}(K)} \tag{3.8}$$

$$R_{PLT}^{t,d} = \frac{\sum_{K=0}^{TH\_K_{PLT}^{t,d}} NumCU_{PLT}^{t,d}(K)}{\sum_{K=0}^{TH\_K_{PLT}^{t,d}} NumCU_{All}^{t,d}(K)} \tag{3.9}$$

where $NumCU_{IBC}^{t,d}(K)$ and $NumCU_{PLT}^{t,d}(K)$ denote the number of IBC CUs, and PLT CUs with the background color ratio of *K*, respectively. It is noted that $NumCU_{All}^{t,d}(K)$ is calculated as the sum of $NumCU_{Intra}^{t,d}(K)$, $NumCU_{IBC}^{t,d}(K)$ and $NumCU_{PLT}^{t,d}(K)$. To reduce the computational complexity of the mode searching process, it is desired that the skip regions are set as large as possible. However, larger skip regions also induce larger

RD performance loss, because more modes are incorrectly skipped. Therefore, a confidence threshold $\alpha$ is set to control the skip region of each mode. For a given value of $\alpha$, $TH\_K_{Intra}^{t,d}$ is set to the minimum value by keeping $R_{Intra}^{t,d}$ smaller than $\alpha$ while $TH\_K_{IBC}^{t,d}$ and $TH\_K_{PLT}^{t,d}$ are set to the maximum values by keeping $R_{IBC}^{t,d}$ and $R_{PLT}^{t,d}$ smaller than $\alpha$.

For the first frame of a new scene in a sequence, the encoder follows the original encoding process to search all modes in all depth levels while collecting the statistics of $NumCU_{Intra}^{t,d}(K)$, $NumCU_{IBC}^{t,d}(K)$ and $NumCU_{PLT}^{t,d}(K)$. Then, based on the statistics in the first frame, the values of $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$ and $TH\_K_{PLT}^{t,d}$ are derived according to the confidence threshold $\alpha$. Finally, $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$ and $TH\_K_{PLT}^{t,d}$ are used to perform early mode skipping for the following frames. When a CU with the background color ratio of *K* is being encoded, intra mode is skipped if

$$K > TH\_K_{Intra}^{t,d}. \tag{3.10}$$

Similarly, IBC mode and PLT mode are skipped separately if

$$K < TH\_K_{IBC}^{t,d} \tag{3.11}$$

$$K < TH\_K_{PLT}^{t,d}. \tag{3.12}$$

To analyze the hit rate of the mode decision with our adaptive thresholding, a value of $\alpha$ is set to 0.05 in this sub-section, and the analysis of the effects on selecting $\alpha$ will be examined in Section 3.4.1. Table 3.3 shows the hit rate of mode decision for all test sequences under QP of 32, and similar distributions are observed with other QPs. It is observed that the hit rates vary from 90.65% to 100% for different sequences and depth levels. For the depth levels of 0, 1, 2, and 3, the average hit rates are 99.11%, 95.51%, 97.91% and 98.21%, respectively. Therefore, the mode skipping rule with the proposed adaptive thresholding technique will bring negligible RD performance loss to screen

Table 3.3: Hit rate of mode decision for various sequences by the proposed mode skipping rule with adaptive.

| Sequences | Depth=0 (%) | Depth=1 (%) | Depth=2 (%) | Depth=3 (%) |
|---|---|---|---|---|
| BasketballScreen | 98.72 | 95.77 | 98.62 | 98.65 |
| MissionControlClip2 | 99.02 | 91.64 | 97.26 | 97.99 |
| MissionControlClip3 | 98.32 | 92.62 | 96.69 | 98.40 |
| ChineseEditing | 98.31 | 91.01 | 96.02 | 97.83 |
| Console | 100 | 98.79 | 99.27 | 98.34 |
| Desktop | 98.67 | 98.45 | 99.50 | 98.47 |
| FlyingGraphics | 99.12 | 95.49 | 99.38 | 99.87 |
| Map | 99.25 | 90.65 | 97.68 | 98.86 |
| Programming | 98.18 | 92.98 | 99.43 | 98.89 |
| SlideShow | 98.68 | 94.88 | 95.43 | 97.04 |
| WebBrowsing | 99.27 | 95.82 | 98.05 | 98.06 |
| Robot | 100 | 99.02 | 94.65 | 98.36 |
| EBURainFruits | 100 | 99.98 | 99.48 | 97.91 |
| Kimono1 | 100 | 99.97 | 99.30 | 96.23 |
| **Average** | 99.11 | 95.51 | 97.91 | 98.21 |

content sequences. As compared with the works in [53]–[56] and our mode skipping rule using fine-granular CU classification in Section 3.3.2 where IBC and PLT modes are checked together for SCBs, the new mode skipping rule with adaptive thresholding facilitates to check only one mode (IBC or PLT mode) for the SCBs.

To avoid the situation that a CTU cannot be encoded, the decision for skipping all modes in a CU is considered as invalid, and then all mode candidates are checked for the CU. It is noted that, from our experimental results, only 0.03% CUs have all modes decided to be skipped after going through our mode skipping rule with adaptive thresholding.

### 3.3.4 Early Termination of CU Partitions with Adaptive Thresholding

To encode each CTU, the original encoding process needs to search all depth levels to determine the final CTU partition structure. However, if a CU is already encoded efficiently, the remaining partitions can be skipped without RD performance loss. For

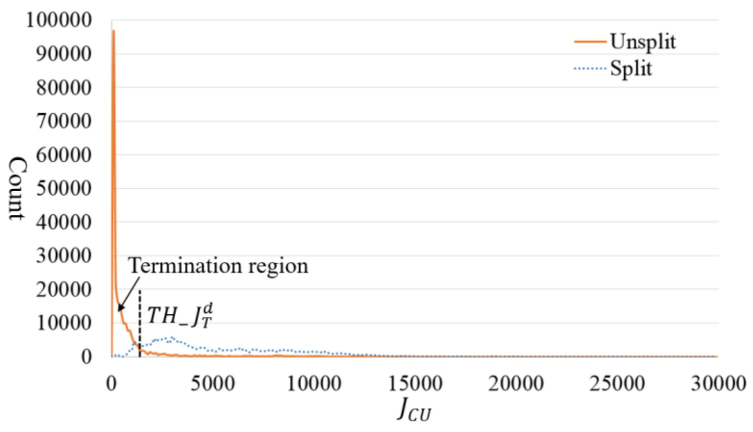


Figure 3.5: RD cost, $J_{CU}$, distributions in term of Unsplit CUs and Split CUs for the first 100 frames of "Desktop" encoded with QP of 32 and the depth level of 2.

traditional camera-captured videos, homogeneous content tends to select large CU sizes while small CU sizes are more likely to be selected to encode complex content. However, due to the adoption of new coding modes, complex content in screen content videos can also be encoded efficiently with large CU sizes by PLT mode and IBC mode. Therefore, instead of analyzing the content of a CU, we use the RD cost of a CU, $J_{CU}$, to perform early termination of CU partitions.

A CU with $d \in \{0,1,2\}$ is classified as an Unsplit CU if it has been efficiently encoded and further partitions are unnecessary. Otherwise, it is classified as a Split CU and it needs to continue partitioning to search the optimal size. To reveal the different $J_{CU}$ distributions in Unsplit CUs and Split CUs, the first 100 frames of "Desktop" were encoded. The results with QP of 32 are shown in Figure 3.5, and similar distributions can be observed with other QPs and depth levels. We can see that CUs with small values of $J_{CU}$ are most likely to be Unsplit CUs whereas CUs with relatively large values of $J_{CU}$ values are likely to be Split CUs. Thus, a content dependent threshold $TH\_J_T^d$ can be extracted to early terminate remaining partitions, and the termination region should be on the left side of $TH\_J_T^d$. Similar as Section 3.3.3, for reducing the computational complexity of the CU partitioning process, it is desirable that the termination region is as large as possible. However, a larger termination region leads to larger RD performance loss, because more CUs are terminated incorrectly. Therefore, a confidence threshold $\beta$ is set to control the termination region. By rounding the value of $J_{CU}$ into integer values which are one-hundredth of the original RD costs, the ratio of Split CUs, $R_{Split}^{d}$, to all CUs in the termination region is calculated as

$$R_{Split}^{d} = \frac{\sum_{J_{CU}=0}^{TH\_J_T^d} NumCU_{Split}^{d}(J_{CU})}{\sum_{J_{CU}=0}^{TH\_J_T^d} (NumCU_{Split}^{d}(J_{CU}) + NumCU_{Unsplit}^{d}(J_{CU}))} \tag{3.13}$$

where $NumCU_{Split}^{d}(J_{CU})$ and $NumCU_{Unsplit}^{d}(J_{CU})$ denote the number of Split CUs and Unsplit CUs with RD cost of $J_{CU}$. $TH\_J_T^d$ is set to the maximum value while keeping the value of $R_{Split}^{d}$ smaller than $\beta$.

By applying the original encoding process to the first frame of a scene, the statistics of $NumCU_{Split}^{d}(J_{CU})$ and $NumCU_{Unsplit}^{d}(J_{CU})$ are obtained to calculate the value of the content dependent threshold $TH\_J_T^d$ according to $\beta$. After the first frame, the previously extracted $TH\_J_T^d$ is used to terminate partitions for a CU. To evaluate the accuracy of the proposed early termination of the CU partition technique, the hit rate of CU partition was

Table 3.4: Hit rate of CU partition for various sequences by the proposed early termination of CU partition.

| Sequences | Depth=0 (%) | Depth=1 (%) | Depth=2 (%) |
|---|---|---|---|
| BasketballScreen | 99.31 | 99.10 | 98.83 |
| MissionControlClip2 | 99.29 | 99.22 | 98.24 |
| MissionControlClip3 | 98.58 | 98.90 | 98.69 |
| ChineseEditing | 99.83 | 98.77 | 98.43 |
| Console | 99.91 | 99.42 | 97.55 |
| Desktop | 98.96 | 99.25 | 98.72 |
| FlyingGraphics | 99.85 | 98.98 | 98.66 |
| Map | 99.34 | 98.32 | 98.12 |
| Programming | 99.94 | 99.96 | 98.49 |
| SlideShow | 98.98 | 98.98 | 98.05 |
| WebBrowsing | 99.86 | 98.29 | 97.85 |
| Robot | 100 | 99.90 | 99.41 |
| EBURainFruits | 100 | 99.26 | 98.87 |
| Kimono1 | 100 | 99.95 | 97.90 |
| **Average** | 99.56 | 99.16 | 98.42 |

analyzed by calculating the percentage of the CUs whose optimal partitions from the original SCM-8.3 are not falsely skipped by using the proposed technique. The hit rates are shown in Table 3.4 for all test sequences under QP of 32, and similar results are observed with other QPs. Here the value of $\beta$ is set to 0.05, and the detailed analysis of choosing $\beta$ will be studied in Section 3.4.1. It is observed that the hit rates for the early termination of the CU partition technique are very high, which vary from 97.55% to 100% for different sequences and depth levels. On average, hit rates of 99.56%, 99.16%, and 98.42% are provided for CUs in the depth levels of 0, 1, and 2, respectively.

### 3.3.5 Scene Change Detection for Adaptive Threshold Updating

In screen content videos, scene changes occur frequently such as document opening or closing, slideshow playing, etc. If the video content changes significantly, it is regarded as a scene change. A scene change makes the learning statistics change a lot, and the content dependent rules are not accurate. To correctly extract the values of $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$, $TH\_K_{PLT}^{t,d}$ and $TH\_J_{T}^{d}$, we use a simple yet efficient correlation measurement method, histogram of difference (HOD) [44], [75], to update the statistics adaptively for different scenes in a sequence. In HOD, the correlation between two adjacent frames is calculated by comparing the collocated luminance values of the two frames, and it is represented as

$$HOD = \frac{\sum_{l \notin [-\tau,\tau]} hod}{\sum_{l=-q+1}^{q-1} hod} \tag{3.14}$$

where $hod$ denotes the histogram of difference between two adjacent frames, and $q$ denotes the number of luminance levels. $\tau$ is a threshold used to select the pixels with a large difference, which is usually set to 32. The further the histogram of difference is distributed from the origin of $hod$, the more different the frames are. Therefore, a scene change is regarded to happen if

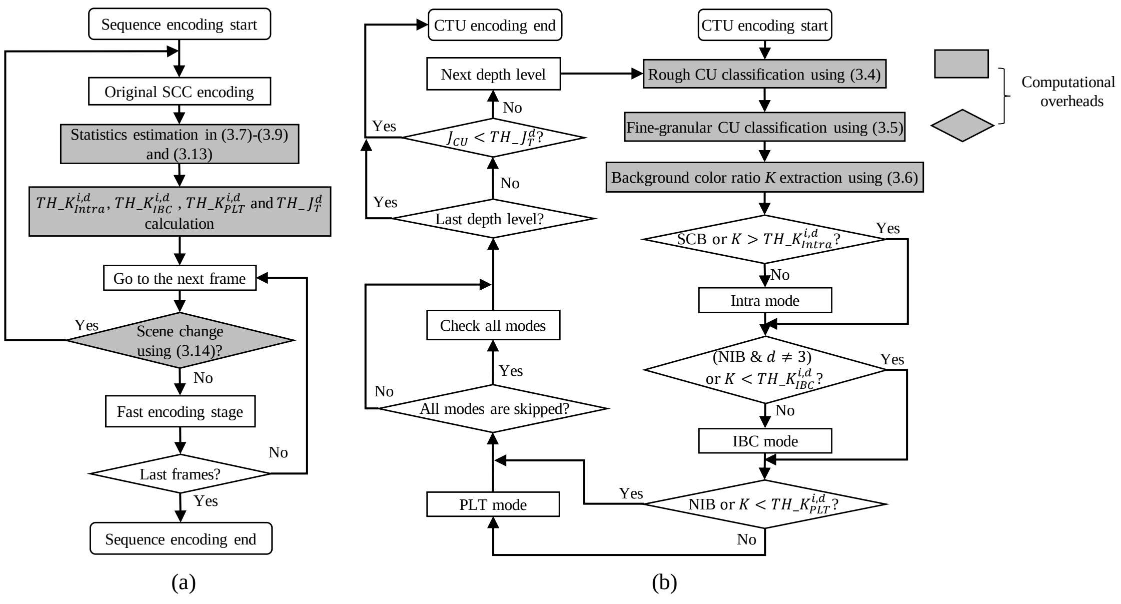

Sequence encoding start
Original SCC encoding
Statistics estimation in (3.7)-(3.9) and (3.13)
$TH\_K_{Intra}^{i,d}$, $TH\_K_{IBC}^{i,d}$, $TH\_K_{PLT}^{i,d}$ and $TH\_J_T^d$ calculation
Go to the next frame
Yes
Scene change using (3.14)?
No
Fast encoding stage
No
Last frames?
Yes
Sequence encoding end
(a)
CTU encoding end
Next depth level
No
Yes
$J_{CU} < TH\_J_T^d$?
No
Yes
Last depth level?
Check all modes
Yes
No
All modes are skipped?
PLT mode
CTU encoding start
Rough CU classification using (3.4)
Fine-granular CU classification using (3.5)
Background color ratio K extraction using (3.6)
SCB or $K > TH\_K_{Intra}^{i,d}$?
Yes
No
Intra mode
(NIB & $d \neq 3$) or $K < TH\_K_{IBC}^{i,d}$?
Yes
No
IBC mode
Yes
NIB or $K < TH\_K_{PLT}^{i,d}$?
No
(b)
Computational overheads

Figure 3.6: Flowcharts of (a) the overall algorithm in sequence level and (b) the fast encoding stage in CTU level.

$$HOD > TH\_HOD_{SC} \tag{3.15}$$

where $TH\_HOD_{SC}$ is a threshold to detect the scene change, and it is experimentally set to 0.2 in our proposed algorithm. Then, the first frame of the new scene is encoded by the original encoding process to update the statistics and calculate the values of $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$ , $TH\_K_{PLT}^{t,d}$ and $TH\_J_T^d$.

### 3.3.6 Flowcharts of Overall Algorithm

Based on the above analysis, our proposed algorithm is divided into two stages, which are the threshold updating stage and fast encoding stage, and the flowchart of the overall algorithm is shown in Figure 3.6. In the threshold updating stage, the first frame of a new scene is encoded by the original encoding process to search all modes in all depth levels while collecting the statistics in Equations (3.7)–(3.9) and (3.13). Then, the values of $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$ , $TH\_K_{PLT}^{t,d}$ and $TH\_J_T^d$ are calculated according to $\alpha$ and

$\beta$. In the fast encoding stage, the fine-granular CU classification technique and the content dependent thresholds are used to make fast mode and CU partition decisions.

## 3.4 Experimental Results and Discussions

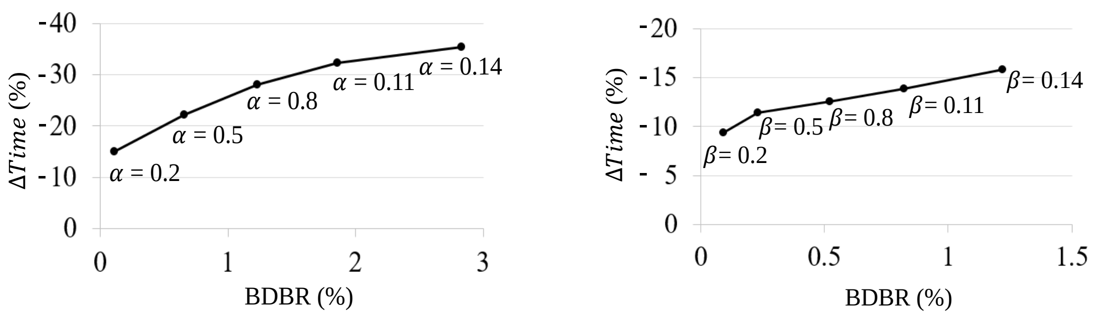
The proposed fast intra-prediction has been implemented in SCM-8.3 for simulations. BDBR and encoding time reduction, $\Delta Time$, with QPs at 22, 27, 32, and 37 were compared with that of the original SCM-8.3. $\Delta Time$ is defined as

$$\Delta Time = \frac{1}{4}\sum_{QP}\left(\frac{Time_{NEW,QP} - Time_{REF,QP}}{Time_{REF,QP}}\right) \times 100\% \qquad (3.16)$$

where $Time_{NEW,QP}$ denotes the encoding time of a new algorithm with a value of QP, and $Time_{REF,QP}$ is the encoding time of the original SCM-8.3 with a value of QP. Since sequences in TGM and M are videos containing both pictorial content and textual content, while sequences in A and CC are pictorial content videos, we will show the average performance results for TGM+M and A+CC, respectively. Three kinds of experiments were conducted to analyze the performance of the proposed algorithm. First, the values of thresholds were tested to find a good trade-off between the coding efficiency and encoding complexity. Second, the computational overhead of the proposed algorithm is analyzed. Third, the performance of the proposed overall algorithm was evaluated by comparing with the existing fast intra-prediction methods. Fourth, the contribution of different proposed techniques was assessed. Fifth, the performance of the proposed algorithm with the adoption of fast encoding in learning frames is investigated.

### 3.4.1 Threshold Determination

In our proposed fast intra-prediction algorithm, two confidence thresholds $\alpha$ and $\beta$ are used to balance the computational complexity reduction and the high coding efficiency of SCC. Figure 3.7 shows the individual performance of the proposed



(a) (b)

Figure 3.7: Performacne of (a) mode skipping rule with adaptive thresholding and (b) early termination of CU partitions with adaptive thresholding with different value of $\alpha$ and $\beta$.

techniques with difference values of $\alpha$ and $\beta$. It is observed that as the value of $\alpha$ increases from 0.2% to 0.14%, the proposed mode skipping rule with adaptive thresholding provides 15% - 35% encoding time reduction with 0.11% - 2.83% BDBR increment. As the value of $\beta$ increases from 0.2% to 0.14%, the proposed early termination of CU partitions with adaptive thresholding provides 9% to 16% encoding time reduction with 0.09% to 1.22% BDBR increment.

To achieve a good trade-off between the coding efficiency and encoding complexity, different values of $\alpha$ and $\beta$ were tested to evaluate the performance of the overall

Table 3.5: Performances with different values of $\alpha$ and $\beta$.

| Sequences | $\alpha$=0.025, $\beta$=0.025 | | $\alpha$=0.025, $\beta$=0.075 | | $\alpha$=0.05, $\beta$=0.05 | | $\alpha$=0.075, $\beta$=0.025 | | $\alpha$=0.075, $\beta$=0.075 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 0.95 | -22.89 | 1.17 | -25.29 | 1.14 | -25.21 | 1.55 | -26.91 | 1.75 | -28.97 |
| MissionControlClip2 | 1.12 | -23.71 | 1.14 | -27.72 | 1.64 | -28.73 | 2.26 | -28.12 | 2.36 | -30.80 |
| MissionControlClip3 | 0.67 | -23.81 | 0.89 | -26.69 | 1.15 | -29.82 | 1.53 | -30.45 | 1.71 | -31.94 |
| ChineseEditing | 0.31 | -25.66 | 0.45 | -27.45 | 0.43 | -33.77 | 0.63 | -38.34 | 0.74 | -38.89 |
| Console | 0.42 | -32.11 | 1.01 | -34.34 | 0.69 | -35.90 | 0.48 | -35.50 | 1.08 | -37.20 |
| Desktop | 0.67 | -31.06 | 1.20 | -33.39 | 1.21 | -33.68 | 1.14 | -37.85 | 1.65 | -39.50 |
| FlyingGraphics | 1.11 | -20.43 | 1.36 | -22.27 | 1.27 | -23.25 | 1.24 | -23.15 | 1.57 | -23.63 |
| Map | 1.24 | -25.49 | 1.46 | -26.86 | 1.54 | -31.56 | 1.81 | -32.17 | 1.26 | -32.49 |
| Programming | 0.49 | -19.92 | 0.63 | -22.04 | 0.73 | -23.85 | 0.82 | -25.03 | 2.04 | -26.99 |
| SlideShow | 3.27 | -55.10 | 3.44 | -58.03 | 3.63 | -58.82 | 5.36 | -58.06 | 5.84 | -60.55 |
| WebBrowsing | 0.82 | -32.89 | 1.89 | -35.10 | 1.83 | -41.76 | 1.87 | -43.96 | 3.32 | -45.91 |
| Robot | 0.77 | -24.27 | 0.79 | -24.32 | 1.21 | -39.00 | 2.35 | -50.40 | 2.40 | -50.89 |
| EBURainFruits | 0.20 | -48.15 | 0.23 | -51.12 | 0.23 | -52.92 | 0.27 | -52.96 | 0.30 | -55.29 |
| Kimono1 | 0.10 | -42.02 | 0.15 | -46.72 | 0.12 | -44.97 | 0.10 | -41.17 | 0.15 | -46.67 |
| **Average (*TGM*+*M*)** | 1.01 | -28.46 | 1.33 | -30.83 | 1.39 | -33.30 | 1.70 | -34.50 | 2.12 | -36.08 |
| **Average (*A*+*CC*)** | 0.36 | -38.15 | 0.39 | -40.72 | 0.52 | -45.63 | 0.91 | -48.18 | 0.95 | -50.95 |
| **Average (*ALL*)** | 0.87 | -30.54 | 1.13 | -32.95 | 1.20 | -35.95 | 1.53 | -37.43 | 1.87 | -39.27 |

Table 3.6: Performances with different values of $TH_S$.

| Sequences | $TH_S$=16 | | $TH_S$=32 | | $TH_S$=48 | | $TH_S$=64 | | $TH_S$=80 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 1.40 | -25.88 | 1.22 | -25.05 | 1.03 | -24.39 | 1.14 | -25.21 | 1.26 | -25.15 |
| MissionControlClip2 | 2.18 | -26.75 | 1.74 | -30.25 | 1.61 | -28.22 | 1.64 | -28.73 | 1.78 | -28.02 |
| MissionControlClip3 | 1.23 | -26.05 | 1.08 | -27.81 | 1.04 | -28.83 | 1.15 | -29.82 | 1.23 | -29.65 |
| ChineseEditing | 0.43 | -31.59 | 0.42 | -32.14 | 0.45 | -33.45 | 0.43 | -33.77 | 0.49 | -32.58 |
| Console | 0.74 | -34.57 | 0.71 | -34.56 | 0.71 | -35.59 | 0.69 | -35.90 | 0.65 | -33.37 |
| Desktop | 0.83 | -32.38 | 0.81 | -33.22 | 1.15 | -33.53 | 1.21 | -33.68 | 1.03 | -32.80 |
| FlyingGraphics | 1.27 | -22.82 | 1.57 | -23.99 | 1.41 | -23.53 | 1.27 | -23.25 | 1.17 | -21.59 |
| Map | 1.54 | -37.35 | 2.25 | -34.68 | 1.66 | -31.90 | 1.54 | -31.56 | 1.58 | -29.99 |
| Programming | 0.73 | -21.16 | 1.01 | -22.37 | 0.87 | -22.88 | 0.73 | -23.85 | 0.64 | -23.22 |
| SlideShow | 3.63 | -57.98 | 3.78 | -58.63 | 3.87 | -59.25 | 3.63 | -58.82 | 4.22 | -57.65 |
| WebBrowsing | 1.82 | -41.70 | 2.09 | -43.97 | 1.73 | -43.25 | 1.83 | -41.76 | 2.00 | -41.22 |
| Robot | 1.21 | -33.94 | 1.74 | -40.18 | 1.20 | -37.24 | 1.21 | -39.00 | 1.26 | -39.18 |
| EBURainFruits | 0.23 | -54.43 | 0.28 | -55.76 | 0.28 | -54.10 | 0.23 | -52.92 | 0.24 | -51.82 |
| Kimono1 | 0.11 | -44.11 | 0.07 | -44.10 | 0.10 | -44.80 | 0.12 | -44.97 | 0.11 | -43.07 |
| **Average (*TGM+M*)** | 1.44 | -32.57 | 1.52 | -33.33 | 1.42 | -33.17 | 1.39 | -33.30 | 1.46 | -32.29 |
| **Average (*A+CC*)** | 0.52 | -44.16 | 0.70 | -46.68 | 0.53 | -45.38 | 0.52 | -45.63 | 0.54 | -44.69 |
| **Average (*ALL*)** | 1.24 | -35.05 | 1.34 | -36.19 | 1.22 | -35.78 | 1.20 | -35.95 | 1.26 | -34.95 |

algorithm, and the results are shown in Table 3.5. It is expected that as the values of $\alpha$ and $\beta$ increase, more reduction in encoding time can be achieved at the expense of BDBR increase. The proposed algorithm shows encoding time saving from 30.54% to 39.27%, while BDBR is increased from 0.87% to 1.87% with $\alpha$ and $\beta$ varying from 0.025 to 0.075. Therefore, the proposed algorithm has the advantage that it can make the trade-off between the RD performance and time saving by setting different confidence threshold values. If the encoding time is the key issue, a large value can be set to the confidence thresholds. Otherwise, a small value can be set to preserve the high coding efficiency of SCC. To balance the encoding time and coding efficiency, we set the values of $\alpha$ and $\beta$ to 0.05 for later discussions.

In the rough CU classification, $TH_S$ is set to 64 empirically to detect high gradient pixels. We also list the performance of our proposed algorithm with different values of $TH_S$ in Table 3.6. It is observed that the overall performance varies little with different values of $TH_S$ from 16 to 80, and the least increase in BDBR of 1.20% is provided with $TH_S$ of 64, as adopted in our proposed algorithm.

### 3.4.2 Analysis for Computational Overheads

Except the encoding process, the proposed algorithm includes the additional processes such as statistics estimation for threshold updating, rough CU classification, fine-granular CU classification, background color ratio $K$ extraction and scene change detection, as highlighted in Figure 3.6. During threshold updating, the first frame of the new scene is encoded by the original SCM-8.3, and various statistics are then collected to calculate the content dependent thresholds $TH\_K_{Intra}^{t,d}$, $TH\_K_{IBC}^{t,d}$, $TH\_K_{PLT}^{t,d}$ and $TH\_J_T^d$ according to $\alpha$ and $\beta$. This process is only applied to the first frame with scene change. It is noted that the calculation of these content dependent thresholds is very simple, and it is about 0.66% on average of the encoding time required for the first frame of the new scene compared to the first frame encoded by the original SCM-8.3. Table 3.7 further analyzes these computational overheads, which are calculated as the ratio of the overhead to the overall encoding time of the proposed algorithm. Results in Table 3.7 show that the threshold updating process only consumes about 0.015% on average of the overall encoding time of the proposed algorithm. On the other hand, the rough CU classification, fine-granular CU classification, background color ratio extraction and scene change detection are necessary to be carried out in all frames. The proportion of these overheads is also listed in Table 3.7, which is about 0.885% on average of the overall encoding time of the proposed algorithm. It can be concluded that the encoding time required for these overheads is negligible. Moreover, in the following evaluation of the proposed algorithm, the encoding time consumed by these processes and all frames are included when calculating the complexity reduction.

Table 3.7: Computational overheads in the overall encoding time of the proposed algorithm.

| Sequences | Proportion (%) | |
|---|---|---|
| | Threshold updating | Classification tasks/ *K* extraction/ scene change detection |
| BasketballScreen | 0.007 | 0.705 |
| MissionControlClip2 | 0.024 | 0.857 |
| MissionControlClip3 | 0.007 | 0.751 |
| ChineseEditing | 0.004 | 0.350 |
| Console | 0.003 | 0.299 |
| Desktop | 0.003 | 0.287 |
| FlyingGraphics | 0.013 | 0.272 |
| Map | 0.006 | 0.616 |
| Programming | 0.008 | 0.583 |
| SlideShow | 0.038 | 0.782 |
| WebBrowsing | 0.009 | 0.623 |
| Robot | 0.021 | 1.592 |
| EBURainFruits | 0.031 | 1.893 |
| Kimono1 | 0.042 | 2.784 |
| **Average** | 0.015 | 0.885 |

### 3.4.3 Performance Evaluation

To validate the efficiency of the proposed algorithm, our overall algorithm is compared with the state-of-the-art fast intra-prediction algorithms [52]–[55] for SCC. It is noted that they were implemented in different reference software from ours in their original publications. Zhang *et al.*'s method [52] was simulated using HM-12.1+RExt-5.1 rather than SCM, while Duanmu *et al.*'s method [53], Lei *et al.*'s method [54] and Yang *et al.*'s method [55] were simulated using SCM-4.0, SCM-2.0 and SCM-5.0, respectively. There are numerous enhancements, speed-up techniques and codes clean-up in SCM-8.3 compared with the older versions. In the older versions, the BV signal in IBC mode was not unified with the inter mode which only has left and above BVs as predictors without IBCM&S. Consequently, incoming CUs always need to check the time-consuming IBCSearch and PLT modes without early termination. Moreover, N×N IBCSearch was done after 2N×N search while it is eliminated in SCM-8.3. In addition, the older versions enable PLT mode in the depth level of 0 while it is disabled in SCM-8.3 because of the

Table 3.8: Performance comparisons with the state-of-the-art fast intra-prediction algorithms.

| Sequences | Zhang [52] | | Duanmu [53] | | Lei [54] | | Yang [55] | | Proposed | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 0.45 | -11.98 | 1.25 | -22.43 | 1.46 | -24.83 | 3.00 | -31.54 | 1.14 | -25.21 |
| MissionControlClip2 | 0.40 | -20.50 | 2.86 | -33.90 | 1.71 | -25.49 | 2.51 | -38.54 | 1.64 | -28.73 |
| MissionControlClip3 | 0.37 | -11.28 | 2.03 | -24.61 | 1.69 | -33.81 | 2.90 | -34.15 | 1.15 | -29.82 |
| ChineseEditing | 0.14 | -3.40 | 1.10 | -17.47 | 0.99 | -18.96 | 5.47 | -31.19 | 0.43 | -33.77 |
| Console | 2.64 | -8.23 | 1.87 | -28.12 | 2.87 | -23.40 | 6.27 | -35.91 | 0.69 | -35.90 |
| Desktop | 0.67 | -4.94 | 2.19 | -26.24 | 1.97 | -23.85 | 7.38 | -42.83 | 1.21 | -33.68 |
| FlyingGraphics | 0.54 | -3.24 | 0.98 | -20.13 | 1.72 | -18.13 | 4.30 | -34.16 | 1.27 | -23.25 |
| Map | 0.97 | -10.66 | 1.55 | -19.16 | 1.23 | -20.05 | 4.71 | -27.38 | 1.54 | -31.56 |
| Programming | 0.44 | -11.76 | 1.89 | -22.16 | 2.50 | -22.92 | 3.69 | -34.45 | 0.73 | -23.85 |
| SlideShow | 0.36 | -46.92 | 2.82 | -52.47 | 2.32 | -55.58 | 5.00 | -53.00 | 3.63 | -58.82 |
| WebBrowsing | 0.79 | -6.99 | 1.91 | -28.17 | 6.02 | -26.75 | 2.84 | -41.66 | 1.83 | -41.76 |
| Robot | 0.43 | -17.89 | 1.18 | -29.36 | 5.21 | -46.91 | 0.59 | -28.19 | 1.21 | -39.00 |
| EBURainFruits | 0.21 | -18.96 | 0.88 | -26.47 | 1.76 | -48.58 | 0.17 | -25.89 | 0.23 | -52.92 |
| Kimono1 | 0.14 | -26.67 | 1.23 | -25.75 | 1.52 | -75.55 | 0.13 | -36.18 | 0.12 | -44.97 |
| **Average (*TGM+M*)** | 0.71 | -12.72 | 1.86 | -26.81 | 2.23 | -26.71 | 4.37 | -36.80 | 1.39 | -33.30 |
| **Average (*A+CC*)** | 0.26 | -21.17 | 1.10 | -27.19 | 2.83 | -57.01 | 0.30 | -30.09 | 0.52 | -45.63 |
| **Average (*ALL*)** | 0.61 | -14.53 | 1.70 | -26.89 | 2.36 | -33.20 | 3.50 | -35.36 | 1.20 | -35.95 |

occasional use. Due to those differences, we re-implemented them into SCM-8.3 for fair comparisons. Table 3.8 shows the detailed performance comparisons in terms of BDBR and $\Delta Time$. It is observed that our proposed algorithm can achieve up to 58.82% encoding time reduction as compared with the anchor SCM-8.3. TGM+M sequences show 33.30% complexity reduction with 1.39% increase in BDBR. Specifically, it provides 33.68%, 35.90% and 33.77% encoding time reduction for "Desktop", "Console" and "ChineseEditing", while the BDBR is increased by 1.21%, 0.69% and 0.43%, which is slightly better than the performance of skipping Intra mode entirely. Besides, A+CC sequences show 45.63% complexity reduction with 0.52% increase in BDBR, and it is similar to the performance of skipping IBC+PLT modes entirely where 46.99% encoding time is reduced with 0.81% increase in BDBR. On average, 35.95% encoding time can be saved with a negligible increase in BDBR of 1.20%. Zhang *et al.*'s method [52] only optimizes the CU partitioning process, so that it shows the least encoding time reduction by only 14.53%, and it is not enough considering the high computational complexity of the SCC encoder. Duanmu *et al*.'s method [53] shows a less encoding time saving of 26.89%, and a higher increase in BDBR of 1.70% than the proposed algorithm. Particularly, only 27.19% complexity reduction is shown for A+CC sequences, which is much smaller than our algorithm. The reason is that unlike Duanmu *et al*.'s method [53] where fixed rules for mode skipping are applied to all sequences, our algorithm calculates the thresholds for mode skipping adaptively by extracting the statistics in the first frame of a scene. For A+CC sequences, because almost no CU selects IBC and PLT modes in the first frame,  the dynamic thresholds can skip IBC and PLT modes for all CUs in the following frames, and it leads to much higher encoding time reduction than Duanmu *et al*.'s method [53]. Lei *et al.*'s method [54] and Yang *et al*.'s method [55] achieve similar encoding time reduction as our algorithm, which are 33.20% and 35.36%, respectively. However, it is observed that their methods bring a much higher increase in BDBR than the proposed algorithm, which are 2.36% and 3.50%, respectively. Again, they also set many fixed rules to skip some mode candidates and CU partitions without taking the distinct characteristics of each sequence into account. Besides, they focus on the fast encoding scheme for A+CC sequences but not for TGM+M sequences. While both Lei *et al.*'s method [54] and Yang *et al*.'s method [55] can skip IBC and PLT modes for NIBs, they need to check Intra mode (either for 2N×2N PUs or all PUs), IBC mode and PLT mode for SCBs. Comparatively, the proposed fine-granular CU classification technique can skip both IBC and PLT modes for NIBs and skip Intra mode for SCBs. Then, the proposed mode skipping rule with the adaptive thresholding technique allows the case that only one mode is checked for SCBs, which further reduces the encoding time.

In the proposed algorithm, a scene change detector is utilized to update the content dependent thresholds adaptively. While similar performances are observed for sequences

Table 3.9: Performances of the proposed overall algorithm with and without scene change detector.

| Sequences | Without HOD | | With HOD | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| MissionControlClip2 | 1.72 | -32.71 | 1.64 | -28.73 |
| MissionControlClip3 | 1.55 | -29.89 | 1.15 | -29.82 |
| FlyingGraphics | 1.35 | -24.34 | 1.27 | -23.25 |
| SlideShow | 3.77 | -53.64 | 3.63 | -58.82 |
| **Average** | 2.10 | -35.15 | 1.92 | -35.16 |

without obvious scene changes by enabling and disabling the scene change detector, the performances for sequences with more than 3 scene changes are shown in Table 3.9 for comparison. It is observed that by enabling the scene change detector, the BDBR increment is reduced by 0.18% for those sequences without affecting the encoding time on average. Specially, "SlideShow" shows performance improvement in both BDBR and $\Delta Time$ because more suitable values of the content dependent thresholds are updated with the scene change detector.

Furthermore, we also investigate the performance of the proposed algorithm by replacing HOD by difference of histogram (DOH) [75]. For DOH, it calculates the absolute sum of the histogram difference between two adjacent frames, $F_a$ and $F_b$, by using luma samples, and the value of DOH is given as the ratio of the absolute sum of the histogram difference to all histograms of $F_a$

$$DOH(F_a, F_b) = \frac{\sum_{l=0}^{q-1} |h_a(l) - h_b(l)|}{\sum_{l=0}^{q-1} h_a(l)} \tag{3.16}$$

where $q$ is the number of luma level, $h_a$ and $h_b$ are the histograms of $F_a$ and $F_b$. If DOH is larger than $TH\_HOD_{SC}$, which is usually set to 0.25, a scene change is regarded to happen. The results are shown in Table 3.10. It is observed that DOH cannot detect the scene change and update the learning statistics adaptively.

Table 3.10: Performances of the proposed overall algorithm with HOD and DOH.

| Sequences | HOD | | DOH | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| MissionControlClip2 | 1.27 | -23.25 | 1.72 | -32.04 |
| MissionControlClip3 | 3.63 | -58.82 | 1.52 | -28.46 |
| FlyingGraphics | 1.64 | -28.73 | 0.51 | -8.83 |
| SlideShow | 1.15 | -29.82 | 4.91 | -56.87 |
| **Average** | 1.92 | -35.16 | 2.17 | -31.55 |

### 3.4.4 Contribution Analysis of Different Techniques

To further investigate the contribution of each individual proposed technique and their combinations, simulations were performed for another 4 settings with $\alpha$=0.05, $\beta$=0.05 and $TH\_P_{DIFF}$=64: fine-granular CU classification (TECH1), mode skipping rule with adaptive thresholding (TECH2), early termination of CU partitions with adaptive thresholding (TECH3), and TECH1+TECH2. The performance of each setting is shown in Table 3.11. Besides, the performance of the overall algorithm is also shown in the table as TECH1+TECH2+TECH3.

TECH1 provides 17.72% encoding time saving with 0.55% increase in BDBR on average. TGM+M sequences show 18.00% complexity reduction with 0.64% increase in BDBR, while A+CC sequences show 16.68% complexity reduction with 0.24% increase in BDBR. We can see that it works well for the sequences with almost pure textual content, which contains both sharp edges and large background area in the same CUs, such as “Console”, “ChineseEditing” and “Desktop”. Significant encoding time reduction can be achieved by 23.50%, 24.96% and 22.88% with a small increase in BDBR by 0.45%, 0.24% and 0.18% for these three sequences, respectively. Besides, TECH1 can encode CC sequences “EBURainFruits” and “Kimono1” efficiently with almost no increase in BDBR. No RD performance loss is observed for “Kimono1” with encoding time saved by 17.13%,

Table 3.11: Performance of the proposed algorithm with different settings.

| Sequences | TECH1 | | TECH2 | | TECH3 | | TECH1+TECH2 | | TECH1+TECH2+ TECH3 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 0.82 | -16.94 | 0.23 | -10.69 | 0.10 | -4.43 | 0.99 | -19.83 | 1.14 | -25.21 |
| MissionControlClip2 | 0.68 | -12.62 | 1.05 | -12.90 | 0.12 | -11.25 | 1.52 | -22.58 | 1.64 | -28.73 |
| MissionControlClip3 | 0.47 | -18.06 | 0.56 | -14.47 | 0.13 | -5.80 | 0.99 | -24.67 | 1.15 | -29.82 |
| ChineseEditing | 0.18 | -22.88 | 0.20 | -22.35 | 0.13 | -3.14 | 0.32 | -32.61 | 0.43 | -33.77 |
| Console | 0.24 | -24.96 | 0.20 | -21.78 | 0.38 | -6.85 | 0.34 | -29.68 | 0.69 | -35.90 |
| Desktop | 0.45 | -23.50 | 0.77 | -22.54 | 0.28 | -4.64 | 0.99 | -28.70 | 1.21 | -33.68 |
| FlyingGraphics | 0.99 | -17.60 | 0.09 | -8.21 | 0.16 | -3.42 | 1.11 | -20.04 | 1.27 | -23.25 |
| Map | 0.77 | -12.81 | 0.58 | -18.49 | 0.22 | -4.62 | 1.28 | -26.87 | 1.54 | -31.56 |
| Programming | 0.44 | -14.42 | 0.22 | -8.79 | 0.08 | -6.39 | 0.58 | -18.44 | 0.73 | -23.85 |
| SlideShow | 1.76 | -10.81 | 2.38 | -13.83 | 0.28 | -41.37 | 3.64 | -17.58 | 3.63 | -58.82 |
| WebBrowsing | 0.24 | -23.39 | 0.88 | -29.07 | 0.63 | -5.94 | 1.22 | -36.33 | 1.83 | -41.76 |
| Robot | 0.67 | -18.78 | 0.92 | -30.52 | 0.01 | -3.74 | 1.22 | -34.99 | 1.21 | -39.00 |
| EBURainFruits | 0.05 | -14.13 | 0.18 | -43.83 | 0.02 | -9.09 | 0.21 | -44.38 | 0.23 | -52.92 |
| Kimono1 | 0.00 | -17.13 | 0.10 | -37.83 | 0.03 | -5.09 | 0.10 | -39.66 | 0.12 | -44.97 |
| **Average (*TGM+M*)** | 0.64 | -18.00 | 0.65 | -16.65 | 0.23 | -8.90 | 1.18 | -25.21 | 1.39 | -33.30 |
| **Average (*A+CC*)** | 0.24 | -16.68 | 0.40 | -37.39 | 0.02 | -5.97 | 0.51 | -39.68 | 0.52 | -45.63 |
| **Average (*ALL*)** | 0.55 | -17.72 | 0.60 | -21.09 | 0.18 | -8.27 | 1.04 | -28.31 | 1.20 | -35.95 |

and BDBR is increased by only 0.05% for “EBURainFruits” with encoding time saved by 14.13%.

For the TECH2, 21.09% encoding time can be saved while BDBR is increased by 0.60% on average. TGM+M sequences show 16.65% complexity reduction with 0.65% increase in BDBR while A+CC sequences show a very high complexity reduction of 37.39% with 0.40% increase in BDBR. CC sequences “EBURainFruits” and “Kimono1” show the largest encoding time reduction by 43.83% and 37.83% while the increases in BDBR are only 0.18% and 0.10%, respectively. The reason is that IBC mode and PLT mode are rarely selected for CC sequences when encoded by the original SCC encoder. Then in the fast encoding stage, the content dependent thresholds $TH\_K_{PLT}^{t,d}$ and $TH\_K_{IBC}^{t,d}$ are set to large values, so that PLT mode and IBC mode are skipped for almost all CUs in the following frames.

By combining the two mode decision techniques together, TECH1+TECH2 provides 28.31% encoding saving with 1.04% increase in BDBR on average. TGM+M sequences show 25.21% complexity reduction with 1.18% increase in BDBR, while A+CC sequences show 39.68% complexity reduction with 0.51% increase in BDBR. When compared with the individual performance of TECH1 and TECH2, higher encoding time is provided by TECH1+TECH2, especially for TGM+M sequences which contains both NIBs and SCBs. It is noted that TECH1 performs the conventional CU classification like method [53]–[55] where both IBC and PLT modes are checked for SCBs. Comparatively, TECH2 provides a more flexible mode checking scheme by deriving content dependent thresholds, where the decisions of IBC and PLT modes can be different. By combining them together, 7.21% and 8.56% higher encoding time are reduced for TGM+M sequences compared with TECH1 and TECH2, respectively. This proves that TECH1 and TECH2 can complement each other to provide higher encoding time reduction.

For TECH3, 8.27% encoding time saving can be achieved with the increase in BDBR by 0.18% on average. TGM+M sequences show 8.90% complexity reduction with 0.23% increase in BDBR while A+CC sequences show 5.97% complexity reduction with 0.02% increase in BDBR. We can see from Table 3.11 that “SlideShow” is the most benefited sequence by employing this approach, and 41.37% encoding time can be saved while BDBR is only increased by 0.28%. As analyzed in Figure 3.2, most CUs in “SlideShow” are determined with sizes of 64×64 and 32×32. Therefore, many CU partitions can be terminated by deriving the content dependent thresholds $TH\_J_T^d$ in the fast encoding stage.

### 3.4.5 Addption of Fast Encoding in Learning Frames

In the proposed algorithm, the encoder follows the original encoding process for the learning frames. The reason is that we want to collect the correct learning statistics, and then the derived content dependent rules can be applied to the following frames for fast encoding. To reduce the computational complexity in the learning frames, some hand-craft

Table 3.12: Performance of applying the proposed fine-granular CU classification in learning frames.

| Sequences | Fast in learning frames | | Proposed | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 1.46 | -25.69 | 1.14 | -25.21 |
| MissionControlClip2 | 1.12 | -33.34 | 1.64 | -28.73 |
| MissionControlClip3 | 1.40 | -32.65 | 1.15 | -29.82 |
| ChineseEditing | 0.35 | -32.05 | 0.43 | -33.77 |
| Console | 0.68 | -36.51 | 0.69 | -35.90 |
| Desktop | 1.38 | -34.52 | 1.21 | -33.68 |
| FlyingGraphics | 2.13 | -27.73 | 1.27 | -23.25 |
| Map | 2.38 | -30.94 | 1.54 | -31.56 |
| Programming | 1.24 | -26.08 | 0.73 | -23.85 |
| SlideShow | 3.78 | -56.71 | 3.63 | -58.82 |
| WebBrowsing | 1.97 | -41.28 | 1.83 | -41.76 |
| Robot | 1.25 | -40.07 | 1.21 | -39.00 |
| EBURainFruits | 0.28 | -52.33 | 0.23 | -52.92 |
| Kimono1 | 0.13 | -44.95 | 0.12 | -44.97 |
| **Average ($ALL$)** | 1.40 | -36.78 | 1.20 | -35.95 |

rules or offline learning-based rules can be applied. However, it makes the learning statistics not accurate. Table 3.12 shows the peformacen of applying the proposed fine-granular CU classification in the learning frames, and it is observed that the encoding time reduction is further improved by 0.83% with BDBR further increased by 0.2%. The encoding time are similar because the number of learning frames is very small.

## 3.5 Chapter Summary

In this chapter, a fast intra-prediction algorithm is proposed based on both content analysis and dynamic thresholding using AI configuration. To skip unnecessary modes for a CU, two early mode decision techniques are proposed based on the rough CU classification, where mode candidates are checked adaptively according to the fine-granular CU types and content dependent thresholds, respectively. Then, content dependent thresholds of RD cost are derived to make early termination of CU partitions. Experimental results under AI configuration show that the proposed algorithm can achieve 35.95% encoding time reduction with 1.20% negligible increase in BDBR on average for typical screen content videos compared with the reference software SCM-8.3.

# Chapter 4 Determinations on Coding Structure by Decision Trees

## 4.1 Introduction

We have observed from the previous chapter that mode decision in SCC is related to CU content, and Chapter 3 utilizes several static features describing CU content reduce mode candidates. In this chapter, dynamic features revealing the unique intermediate coding information of a CU are further explored. The exhaustive mode searching process can then be avoided by a sequential arrangement of DTs, which is constructed to check each mode separately with the insertion of a classifier before checking a mode. In contrast to the approaches that both IBC and PLT modes are examined for SCBs, the proposed coding arrangement becomes more flexible and allows either IBC or PLT mode to be checked for SCBs to further reduce the computational complexity of mode decision in SCC.

The rest of this chapter is organized as follows. Section 4.2 analyzes the flexibility of different fast encoding frameworks, and it discusses the advantage of the proposed framework. The proposed idea of utilizing both static features and dynamic features to make mode decision is presented in Section 4.3. First, we give the classification principle of DT-based classifiers. Second, the proposed dynamic features and their advantage are presented. Third, the details of the proposed DT-based mode decision classifiers are given. Forth, a DT constraint technique by using spatial mode correlation is introduced to improve the accuracy of the proposed classifiers. Section 4.4 shows the simulation results of the proposed algorithm. Finally, Section 4.5 concludes this chapter.

Parts of the contents of this chapter are extracted from our published work [76]:

- **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, "Machine Learning Based Fast Intra Mode Decision for HEVC Screen Content Coding Via Decision Trees," IEEE Transactions on Circuits and Systems for Video Technology, 2019.

## 4.2 Flexibility of Different Frameworks

The previous fast SCC encoding algorithms are mainly focused on fast CU size decision and fast mode decision made by CU type classification, as shown in Figure 4.1 (a) and (b), respectively. However, these frameworks are not flexible and are difficult to achieve a good trade-off between the computational complexity and coding efficiency. For fast CU size decision approaches [50]–[52], all modes are either checked or skipped together in a CU as shown in Figure 4.1(a). For fast mode decision approaches [53]–[56] using CU type classification in Figure 4.1(b), the screen content modes, IBC mode and PLT mode, are either checked or skipped together. In screen content videos, some CUs are very difficult to be decided whether they are SCBs or NIBs even by human beings,

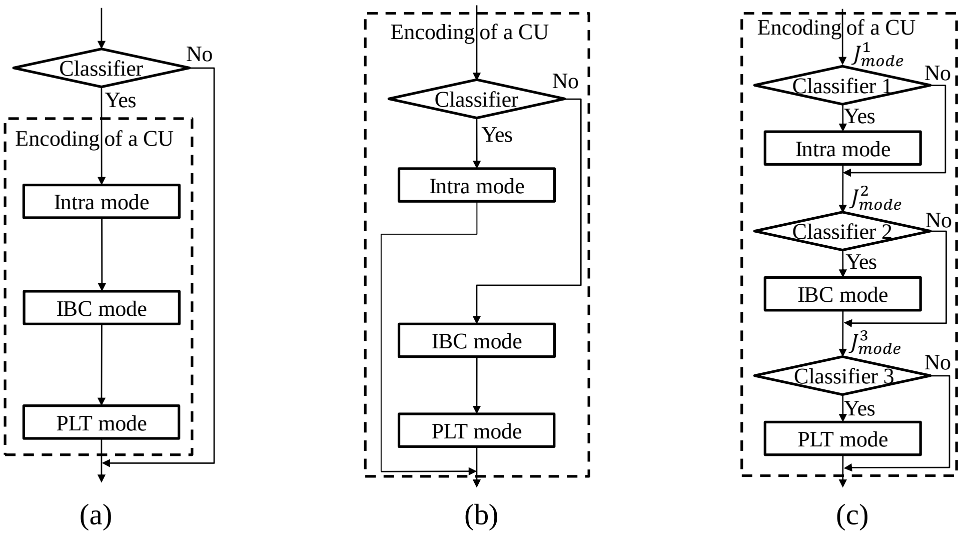


Figure 4.1: CU encoding flowcharts of various fast SCC encoding algorithms. (a) Typical fast CU size decision algorithm [50]–[52], (b) typical fast mode decision algorithm by CU type classification [53]–[56], and (c) proposed fast mode decision algorithm.

and the CU type classification approaches are not efficient for these CUs. On the contrary, our proposed framework provides larger flexibility by inserting a classifier before checking a mode in a CU, as shown in Figure 4.1(c). By deriving the dynamic features right before checking a mode, more accurate decision is made. On the one hand, encoding time can be further reduced by allowing the case that only one mode is checked for a SCB. On the other hand, RD performance can be improved by allowing PLT mode to be checked for a SCB even if IBC mode is wrongly skipped. It is also noted that the values of the dynamic features are changing as a CU goes through different classifiers, and only our framework in Figure 4.1(c) can adopt these dynamic features proposed in this work.

## 4.3 Proposed DT Based Framework

Since there are numerous mode candidates in different CU sizes, it is difficult to manually select the optimal features and classification criteria to build accurate mathematical models. To solve this problem, 11 features, which are related to the mode decision, are proposed to train various DT-based classifiers from offline learning. Therefore, the optimal features and classification criteria are reasonably selected based on the training data. In the test phase, the trained classifiers are implemented in SCM to make fast mode decision.

### 4.3.1 Description of the Classifier Using DT

DT is one of the most popular machine learning algorithms. In this chapter, we utilize a DT as the classifier, because it comes with low complexity in the testing phase and can be easily implemented into a SCC encoder as a set of “if-then-else” conditions. A DT-based classifier is a flowchart-like tree structure, as shown in Figure 4.2. It is composed of a root node, internal nodes and leaf nodes. For each non-leaf node, i.e., a

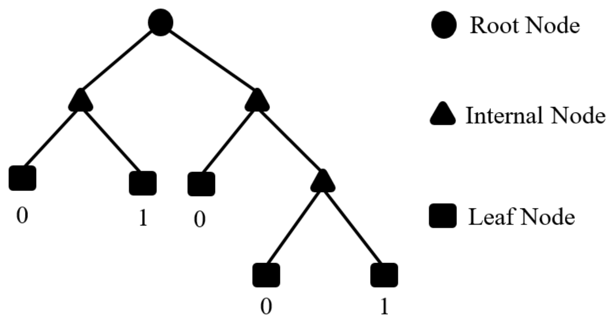


Figure 4.2: General structure of a DT-based classifier.

root node or an internal node, it denotes a test on a feature of the incoming sample. Each branch after a non-leaf node denotes the outcome of the test, and each leaf node denotes a class label. In the specific case of the mode selection problem in a CU, the class label of 1 or 0 represents whether the target mode is checked or not. The classifiers based on DTs are trained by the C4.5 algorithm [77] in the Waikato Environment for Knowledge Analysis (WEKA) [78] version 3.8. To generate training frames which reflect the characteristic of SCC sequences, 8 frame-skipped sequences are formed by extracting the first frame of each second from the 8 sequences, which are “ChineseEditing”, “FlyingGraphics”, “SlideShow”, “BasketballScreen”, “MissionControlClip2”, “Robot”, “EBURainFruits”, “Map”. Training frames from different sequences were encoded by the original SCM-8.3 encoder with QPs at 22, 27, 32, and 37 using AI configuration to generate training data.

If a node of a DT only contains samples from one class, it is defined to have pure samples. Otherwise, the impurity is calculated to represent how impure the samples in the node are. To reduce the impurity of the node, a feature $f$ with a classification threshold $TH_f$ is selected to further classify the samples into two child nodes, and the impurity reduction is calculated by comparing the impurities of two child nodes and the parent node. In the training process of a DT, the impurity reduction by splitting a parent node to two child nodes is calculated iteratively for each feature $f$ with a classification threshold

$TH_f$. The larger the impurity reduction is, the better the feature and the classification threshold are. In the C4.5 algorithm, the impurity is calculated by entropy. Then the impurity reduction with $f$ and $TH_f$ is measured by the gain ratio $GainRatio(f, TH_f)$

$$GainRatio(f, TH_f) = \frac{InfoGain(f, TH_f)}{SplitInfo(f, TH_f)} \tag{4.1}$$

where $InfoGain(f, TH_f)$ is the information gain by splitting a node $n_0$ into its child nodes $n_1$, $n_2$ using a feature $f$ with a threshold $TH_f$. It is calculated by the entropy reduction after splitting as

$$InfoGain(f, TH_f) = En(n_0) - \sum_i \frac{N_{n_i}}{N_{n_0}} En(n_i) \tag{4.2}$$

where $N_{n_0}$ and $N_{n_i}$ represent the number of samples in the node $n_0$ and child nodes $n_i$, $i \in \{1,2\}$. Let $p(\omega_j)$ be the probability of training samples belonging to the class $\omega_j$ in a node $n$, $j \in \{0,1\}$. The entropy $En(n)$ in the node $n$ is calculated as

$$En(n) = -\sum_{j=0}^{1} p(\omega_j) log_2 \, p(\omega_j). \tag{4.3}$$

The normalization term $SplitInfo(f, TH_f)$ is defined by

$$SplitInfo(f, TH_f) = -\sum_{i=1}^{2} \frac{N_{n_i}}{N_{n_0}} log_2 \frac{N_{n_i}}{N_{n_0}}. \tag{4.4}$$

The best feature and the threshold are selected as the ones with maximum gain ratio to split a node. A DT is trained node by node, and the splitting of a node is terminated if the number of training samples arrived the node is less than or equal to 1% of the total training samples. Then a reduced error pruning process [79] is performed to prune the DT backward to avoid overfitting. After generating a DT, the classification accuracy of the tree is given by a 10-fold cross-validation process [80], which calculates the percentage of correctly classified samples in the total training samples.

### 4.3.2 Proposed Dynamic and Static Features

In general, the precision of SCC mode decision in a classification task is highly dependent on the feature space used to train the model. In most of the machine learning algorithms adopted in mode decision of video coding, the features extracted from a CU is always determined by its static content, such as background color number, gradient, etc. These features are called as static features in this thesis.

In contrast, we find that the probability of selecting IBC mode as the optimal one depends on the spatial location of the current CU, as shown in Figure 4.3. Assume that $CU_A$ and $CU_B$ in the example of Figure 4.3 have the same static content. Even though the static features extracted from $CU_A$ and $CU_B$ are the same, the mode decision of these two CUs may be different. For example, $CU_A$ may select PLT mode while $CU_B$ with the same content is likely to select IBC mode. The reason is that the search window of $CU_B$ is larger, and it results in a higher chance to find a good repeated pattern with very low RD cost by using IBC mode. By taking this specific characteristic of screen content videos into account, we propose to extract the IBC mode flag of the current CU before checking the target mode, $Flag_{IBC}$, as the dynamic feature $f_1$. If the best mode so far of the current CU before checking the target mode is a sub-mode (i.e. IBCPredictor, IBCM&S, or IBCSearch) of IBC mode, $Flag_{IBC}$ is set to 1. Otherwise, $Flag_{IBC}$ is set to 0. It is noted

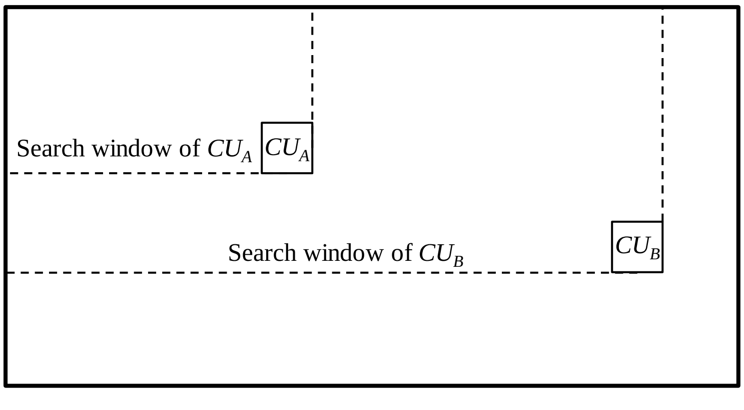


Figure 4.3: Two CUs with same content in a frame.

that, for $CU_B$ in Figure 4.3, the chance of $Flag_{IBC}$ equal to 1 is higher as compared with that of $CU_A$ even though they the have same content. Therefore, this feature may vary according to the spatial location, and it is considered as a dynamic feature.

In addition, the dynamic RD cost of the best mode so far in the current CU before checking the target mode, $J_{mode}$, is another dynamic feature $f_2$ for fast mode decision. Similarly, $J_{mode}$ of $CU_B$ in Figure 4.3 is likely to become smaller since it is easier to get a good repeated pattern in the reconstructed area. Besides, $J_{mode}$ is not only related to the spatial location but varies during the encoding process. For instance, $J_{mode}^2$ to Classifier 2 in Figure 4.1(c) may be different from $J_{mode}^3$to Classifier 3 since $J_{mode}^3$ has already gone through Intra mode and IBC mode while only intra mode is tested for computing $J_{mode}^2$. The variation property is well suited for our proposed framework in Figure 4.1(c) in which the values of this dynamic feature entered to various DTs are different. This new arrangement is of great importance to SCC mode decision process using classification, which will be verified in the following sections. $J_{mode}$ reveals the unique intermediate coding information of a CU, and its value varies as the CU goes through different DTs. By implementing DTs right before the target mode, the most updated values of these dynamic features ($f_1$ and $f_2$) are obtained for different trees to improve the decision accuracy.

By using the proposed framework with DTs prior to checking a mode in a CU, the new dynamic features with the following nine static features are then selected based on our prior knowledge for the training of DTs in Figure 4.1(c).

$f_3$**:** Background color number *BCN*. The background color in a CU is defined as the color with the highest occurrence frequency within the CU, and *BCN* is calculated by counting the number of the background color pixels.

$f_4$**:** Distinct color number *DCN*. *DCN* is calculated by counting the pixels in a CU with different sample values.

For *BCN* and *DCN*, all three components of a pixel (Y, U, V in YVV 4:4:4 or R, G, B in RGB 4:4:4) are stacked to form a 24-bit sample value. For sequences in YUV 4:2:0 format, only the luminance component is utilized as an 8-bit sample value.

$f_5 - f_8$**:** High gradient pixel number $HGN_0$, $HGN_1$, $HGN_2$, $HGN_3$. The high gradient pixel is utilized to detect sharp edges in a CU, as in Equation (3.1). To detect edges with different sharpness in our proposed algorithm, 4 different high gradient pixel numbers, $HGN_0$, $HGN_1$, $HGN_2$, and $HGN_3$, are calculated by counting high gradient pixels with $TH_S$ of 4, 8, 16, and 32, respectively. Considering that the proposed algorithm is a machine learning based approach, it lets the DT select the features to be used based on the off-line training. It implies to select which value(s) of $TH_S$ to be used in each DT. Therefore, we do not need to manually select which particular value(s) of $TH_S$ in the final DTs.

It is noted that sequences in RGB 4:4:4 format are converted to YUV 4:4:4 format to get the luminance component.

$f_9 - f_{10}$**:** CU horizontal and vertical activities *HorAct* and *VerAct*. They have been used for skipping IBC mode adaptively in the original SCM-8.3 and defined as

$$HorAct = \sum_{i=0}^{2N} \sum_{j=0}^{2N-1} |Y_{i,j} - Y_{i,j+1}| \tag{4.5}$$

$$VerAct = \sum_{j=0}^{2N} \sum_{i=0}^{2N-1} |Y_{i,j} - Y_{i+1,j}|. \tag{4.6}$$

$f_{11}$**:** CU variance *Var*. *Var* can well represent the smoothness of a CU, which is defined as

$$Var = \frac{1}{2N \times 2N} \sum_{j=0}^{2N} \sum_{i=0}^{2N} (Y_{i,j} - \bar{Y})^2 \tag{4.7}$$

where $\overline{\mathrm{Y}}$ is the average luminance value over the current CU.

$f_3$– $f_{11}$ are static features which have fixed values in a CU. Therefore, they are obtained once for a CU and shared among different DTs.

### 4.3.3 Fast Mode Decision Design

To make fast mode decision in SCC, the selection of Intra mode, IBC mode, and PLT mode is investigated by adopting different DTs in our new coding framework. Then, a DT for performing CU type classification is trained at the last depth level to avoid the situation that all modes are skipped for a CTU. The proposed framework inserts a classifier before checking a target mode. Unlike the existing methods that inserts a classifier before the entire mode-checking, it can extract features that reflect the intermediate coding information, such as the best RD cost so far and the IBC flag so far. Those dynamic features help to improve the decision accuracy. In this sub-section, the detailed design of the new coding framework is discussed.

#### A. Feature Analysis

Among the three coding modes in SCC, Intra mode is the only mode inherited directly from HEVC. While Intra mode is very efficient for NIBs, IBC mode and PLT mode are both specially designed for SCBs. To perform fast mode decision, a common idea is to classify CUs into NIBs and SCBs by analyzing their content characteristics. Then IBC and PLT modes are checked for SCBs while Intra mode is checked for NIBs. However, such an approach is not optimal since IBC and PLT modes are always checked together for SCBs.

To understand the distributions of Intra, IBC and PLT modes over different features, we randomly selected 300,000 16×16 CUs from the training samples, and the number of the CUs with each mode is 100,000. First, the mode distributions over the dynamic features obtained right before the target mode were investigated. Figure 4.4 shows the

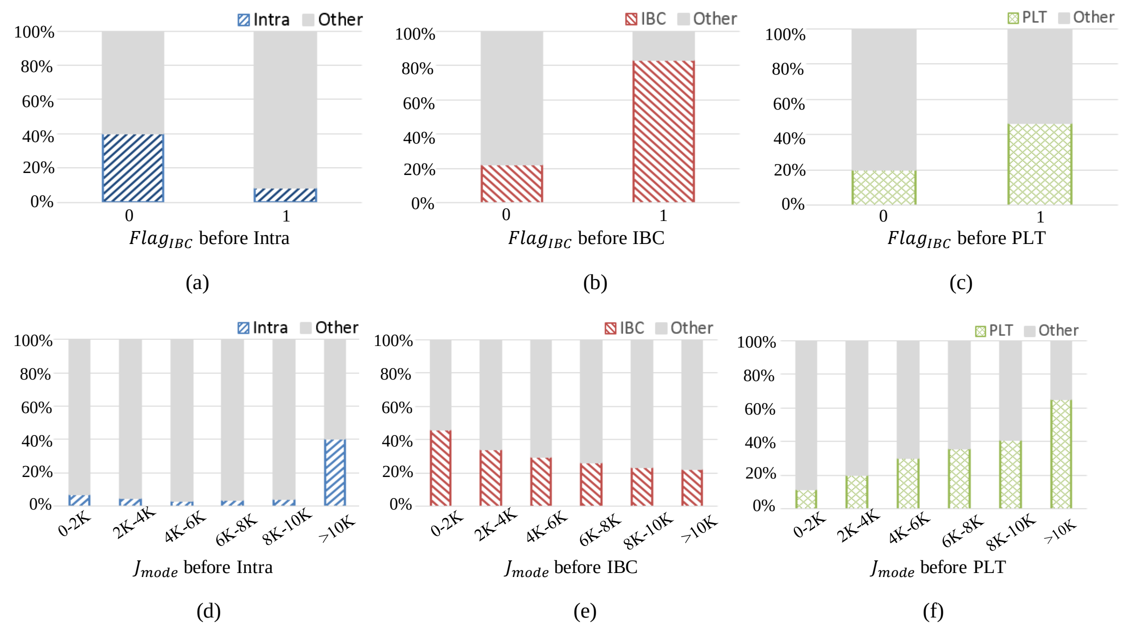


Figure 4.4: The percentages of the target mode and other modes in terms of $Flag_{IBC}$ (a)–(c) and $J_{mode}$ (d)–(e) for 16×16 CUs.

percentages of CUs selecting the target mode and other modes in terms of $Flag_{IBC}$ (Figure 4.4 (a)–(c)) and $J_{mode}$ (Figure 4.4 (d)–(e)). If $Flag_{IBC}$ before checking the target mode is 1, the CU is more likely to be a SCB, otherwise, it more likely to be a NIB. Therefore, it is observed in Figure 4.4(a) that the percentage of Intra mode is very low if $Flag_{IBC}$ before checking Intra mode is 1. On the contrary, the percentages of IBC mode and PLT mode are low if $Flag_{IBC}$ before checking the target mode is 0, as shown in Figure 4.4(b) and Figure 4.4(c), respectively. Before checking Intra mode, $J_{mode}$ is highly correlated to $Flag_{IBC}$. If IBCPredictor does not provide a valid BV for a CU, $Flag_{IBC}$ would be 0 and the value of $J_{mode}$ becomes very large. Otherwise, the value of $J_{mode}$ is relatively small if $Flag_{IBC}$ is 1. Therefore, the percentage of CUs selecting Intra mode is very low if the value of $J_{mode}$ is small, as shown in Figure 4.4(d). It is also observed in Figure 4.4(e) that if $J_{mode}$ before checking IBC mode is very large, the percentage of IBC mode would be low. The reason is that for CUs with very large values of $J_{mode}$, they usually have complex texture, and it is difficult to find repeated patterns for the complex

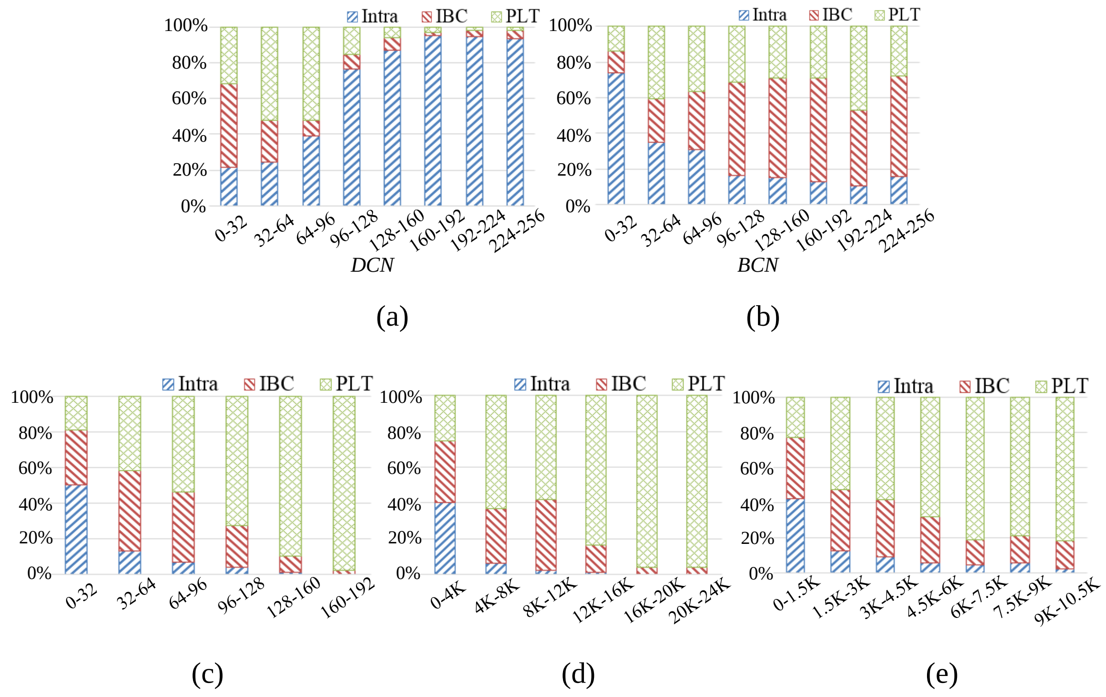


Figure 4.5: Intra, IBC and PLT mode distributions in terms of (a) Distinct color number *DCN*, (b) high gradient pixel number $HGN_3$, (c) horizontal activity *HorAct*, and (d) CU variance *Var* for 16×16 CUs.

texture CUs by IBC mode. Besides, Figure 4.4(f) shows that the percentage of PLT mode is low for CUs with a small value of $J_{mode}$. The reason is that the CU with a small value of $J_{mode}$ before checking PLT mode may have been efficiently encoded and the checking of PLT mode becomes unnecessary. The discrepancy between Figure 4.4(e) and Figure 4.4(f) verifies that PLT mode and IBC mode have different characteristics and should not use the same classifier when the dynamic features are adopted.

Then the mode distributions of the static features shared among different DTs were also investigated. Figure 4.5 shows the mode distributions in terms of 5 representative features: (a) *DCN*, (b) *BCN* (c) $HGN_3$, (d) *HorAct* and (e) *Var*. It is observed that the percentage of Intra mode increases as *DCN* gets larger, or *BCN*, $HGN_3$, *HorAct*, and *Var* get smaller. The reason is that Intra mode is designed for NIBs, and they tend to have larger *DCN*, smaller *BCN* and be smoother. Besides, it is also observed that the percentage of PLT mode is much higher than IBC mode when CUs get more complex, such as CUs with larger values of $HGN_3$, *HorAct* and $Var$. It further implies that PLT mode and IBC mode should not share the same classifier for SCC intra mode selection that is always adopted in the algorithms proposed in the literature [53]–[55].

Based on these observations, we trained DTs in the proposed coding framework to adaptively check Intra mode, IBC mode and PLT mode separately.

## B. Training of DT-based Classifier

As described before, IBC mode contains three steps, which are IBCPredictor, IBCM&S and IBCSearch. While the step of the IBCPredictor only checks several BV predictors and our experiment shows that it takes up only 1.24% of the total encoding time, and the computational complexities of IBCM&S and IBCSearch are relatively high. Therefore, we always check IBC-Step1 as the first check. By collecting the most updated features, two sets of DTs are generated inside IBC mode to adaptively check IBCM&S and IBCSearch. After generating the DTs for all modes, they are implemented in the SCM-8.3 encoder to perform fast mode decision. Before checking a mode, the incoming CU goes through the DT for the mode to decide whether it should be tested. If the outcome or the class label of the DT is 1, it is involved in the mode searching process. Otherwise, the current CU does not check the target mode so that the computational complexity brought by this mode is reduced. However, there is a case that all modes are decided to be skipped for a CTU when all mode DTs are implemented, and finally the CTU cannot be encoded. To solve this problem, a CU type DT is also trained at the last depth level, and at least one possible mode is assigned to the CU if all modes are skipped for it. The CU type DT can classify incoming CUs into NIBs and SCBs. If the outcome for a CU is a

Table 4.1: Training data number for each DT.

| CU Size | Intra | IBC | | PLT | CU Type |
|---|---|---|---|---|---|
| | | Merge & Skip | Search | | |
| 64×64 | 28452 | 14224 | | | 64×64 |
| 32×32 | 216072 | 111980 | | 80804 | 32×32 |
| 16×16 | 715548 | 573848 | 168736 | 219192 | 16×16 |
| 8×8 | 3166280 | 2724108 | 1522712 | 453080 | 8×8 |

Table 4.2: Depth of each DT.

| CU Size | Intra | IBC | | PLT | CU Type |
|---|---|---|---|---|---|
| | | Merge & Skip | Search | | |
| 64×64 | 14 | 13 | | | |
| 32×32 | 7 | 4 | | 6 | |
| 16×16 | 8 | 1 | 10 | 6 | |
| 8×8 | 9 | 9 | 6 | 7 | 7 |

NIB, i.e., 1, Intra mode is checked for it. Otherwise, IBC and PLT modes are both checked for it.

As SCC supports CU sizes of 64×64, 32×32, 16×16, and 8×8, 4 DTs are trained for CUs with different sizes. To avoid the data imbalance problem caused by more training samples in one class than in the other [81], we let 50% of the training samples come from CUs with the target mode as their optimal modes, and they are treated as the positive data. The other 50% of training samples come from the CUs which are not encoded by the target mode, and they are treated as the negative data. Besides, for the training of the CU type DT at the last depth level, the positive training data are from NIBs, i.e. CUs encoded by Intra mode, while the negative data are from SCBs, i.e. CUs encoded by IBC or PLT mode.

The training data number and the depth of each DT are shown in Table 4.1 and Table 4.2, respectively. Since a frame can be partitioned into more CUs with a small size than CUs with a large size, more training data are obtained as the CU size gets smaller. Besides, we can see that the largest depth of the trained DTs is 14, which means the decision for a mode is made after going through at most 14 “if-then-else” conditions. Therefore, the

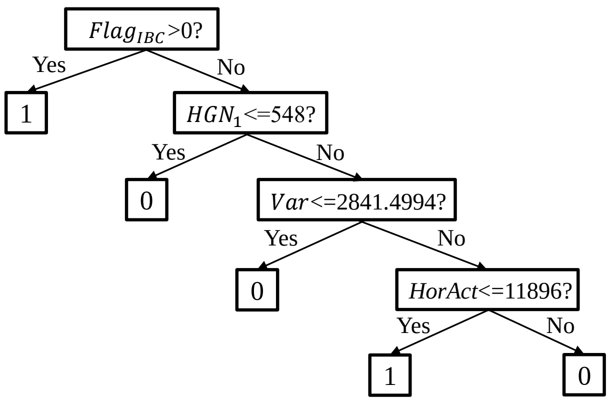


Figure 4.6: IBCM&S mode DT for 32x32 CUs.

computational complexity brought by those DTs is negligible. As an example, the DT based IBCM&S mode classifier for 32×32 CUs is shown in Figure 4.6, and other trained classifiers can be found in our website [82].

## C. Feature Subset Selection

When training classifiers for different tasks, the valid features are quite different, and the performance of a classifier is very sensitive to the features utilized to train the classifier. Therefore, to eliminate the impact of irrelevant or redundant features and provide a better understanding of the valid features for each mode decision, a feature subset selection [83] approach is applied.

We implemented the feature subset selection in WEKA using the wrapper evaluation with a greedy search strategy, which is computationally advantageous and robust against overfitting. The feature subset selection consists of the following steps:

Step 1: Initialize the feature subset set $\mathrm{F}^k$=$\emptyset$ at $k$=0.

Step 2: Find the best remaining feature $f$ which provides the largest accuracy increase when added to $\mathrm{F}^k$.

Step 3: $k$++ and $\mathrm{F}^k$=$\mathrm{F}^{k-1}\cup\{f\}$.

Step 4: Iterate step 2 and step 3 until the classifier accuracy is no longer improved.

Table 4.3: The gain ratio of each feature for each DT.

| DT | | $Flag_{IBC}$ | $J_{mode}$ | *BCN* | *DCN* | $HGN_0$ | $HGN_1$ | $HGN_2$ | $HGN_3$ | *HorAct* | *VerAct* | *Var* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Intra | 64×64 | ╲ | ╲ | 0.044 | 0.029 | ╲ | 0.071 | 0.094 | | 0.074 | 0.065 | 0.054 |
| | 32×32 | 0.179 | ╲ | 0.043 | 0.022 | ╲ | ╲ | ╲ | 0.084 | 0.045 | 0.046 | |
| | 16×16 | ╲ | 0.102 | 0.035 | 0.029 | ╲ | ╲ | ╲ | ╲ | 0.029 | 0.025 | 0.027 |
| | 8×8 | 0.180 | ╲ | 0.051 | 0.046 | ╲ | ╲ | 0.005 | ╲ | 0.020 | 0.019 | 0.025 |
| IBCM&S | 64×64 | ╲ | 0.065 | 0.099 | | 0.084 | ╲ | ╲ | ╲ | ╲ | 0.097 | 0.100 |
| | 32×32 | 0.669 | ╲ | ╲ | ╲ | ╲ | 0.045 | ╲ | ╲ | 0.070 | ╲ | 0.066 |
| | 16×16 | 0.431 | ╲ | ╲ | ╲ | ╲ | ╲ | ╲ | ╲ | ╲ | ╲ | ╲ |
| | 8×8 | 0.178 | 0.014 | 0.032 | 0.019 | ╲ | ╲ | 0.024 | ╲ | 0.017 | 0.018 | 0.025 |
| IBCSearch | 16×16 | 0.275 | 0.007 | 0.028 | 0.030 | ╲ | ╲ | 0.019 | 0.023 | 0.010 | 0.016 | 0.022 |
| | 8×8 | 0.281 | | 0.039 | 0.027 | ╲ | ╲ | 0.061 | 0.067 | 0.035 | 0.036 | ╲ |
| PLT | 32×32 | 0.001 | 0.048 | 0.064 | 0.041 | ╲ | 0.026 | ╲ | ╲ | 0.035 | ╲ | ╲ |
| | 16×16 | 0.166 | 0.038 | 0.035 | 0.025 | 0.009 | ╲ | 0.034 | ╲ | 0.031 | ╲ | ╲ |
| | 8×8 | 0.105 | 0.027 | 0.029 | 0.021 | ╲ | ╲ | 0.031 | 0.039 | 0.027 | 0.023 | ╲ |
| CU Type | 8×8 | ╲ | ╲ | 0.058 | 0.045 | ╲ | ╲ | ╲ | ╲ | 0.031 | 0.028 | ╲ |

Table 4.3 shows the valid features of each DT, and the importance of each valid feature is also shown in this table by measuring its gain ratio. It is observed that the proposed dynamic features, $Flag_{IBC}$ and $J_{mode}$ obtained right before the target mode, are quite important for most DTs, with the gain ratio up to 0.669 and 0.102, respectively. This verifies that the dynamic features play a critical role in the decision process with the introduction of the new coding framework. By adopting the feature subset selection approach, the number of features fed into a DT is reduced to 1–8, and 6.07 on average. Compared with the original feature set with 11 features, the feature number is reduced by 44.92%, and the impact of the feature subset selection approach in terms of coding performance will be discussed in Section 4.4.4.

### D. Accuracy of DTs

The decision accuracy for each DT is shown in Table 4.4. We can see from the table that the accuracies of those DTs vary from 75.44% to 94.51%, where the decision accuracies for 64×64 CUs are relatively low. The reason is that there are many CUs with pure horizontal edges, pure vertical edges or a single color in the training data set of 64×64 CUs, and they are difficult for making classification. However, this kind of CUs can be encoded efficiently by all modes with very low computational complexity. Therefore, the

Table 4.4: Decision accuracy for each DT.

| CU Size | Intra (%) | IBC | | PLT (%) | CU Type (%) |
|---|---|---|---|---|---|
| | | Merge & Skip (%) | Search (%) | | |
| 64×64 | 75.44 | 82.31 | | | |
| 32×32 | 87.95 | 94.51 | | 82.27 | |
| 16×16 | 83.34 | 84.57 | 81.89 | 82.48 | |
| 8×8 | 78.23 | 83.13 | 85.66 | 79.83 | 83.17 |

Table 4.5: Incorrect decision for each DT.

| CU Size | Intra (%) | IBC | | PLT (%) |
|---|---|---|---|---|
| | | Merge & Skip (%) | Search (%) | |
| 64×64 | 3.81 | 5.00 | | |
| 32×32 | 2.88 | 3.77 | | 5.33 |
| 16×16 | 4.07 | 13.14 | 8.73 | 5.06 |
| 8×8 | 9.95 | 6.07 | 6.75 | 6.60 |

low decision accuracy of 64×64 CUs will not lead to large computational complexity or coding efficiency degradation.

There are two kinds of false classifications for each mode DT. One is the missed detection, which is the case that the optimal mode of a CU is not a certain mode, but it is not detected, and the mode is checked for the CU redundantly. Although the missed detection leads to the increase in computational complexity, it does not bring RD performance loss. The other is the incorrect decision, which is the case that the optimal mode of a CU is a certain mode, but the mode is skipped for the CU incorrectly. The incorrect decision leads to RD performance loss because the optimal mode for a CU is skipped. The incorrect decision rate of each DT is shown in Table 4.5, and we can see that only 3.12% to 13.14% of the mode decision leads to RD performance loss.

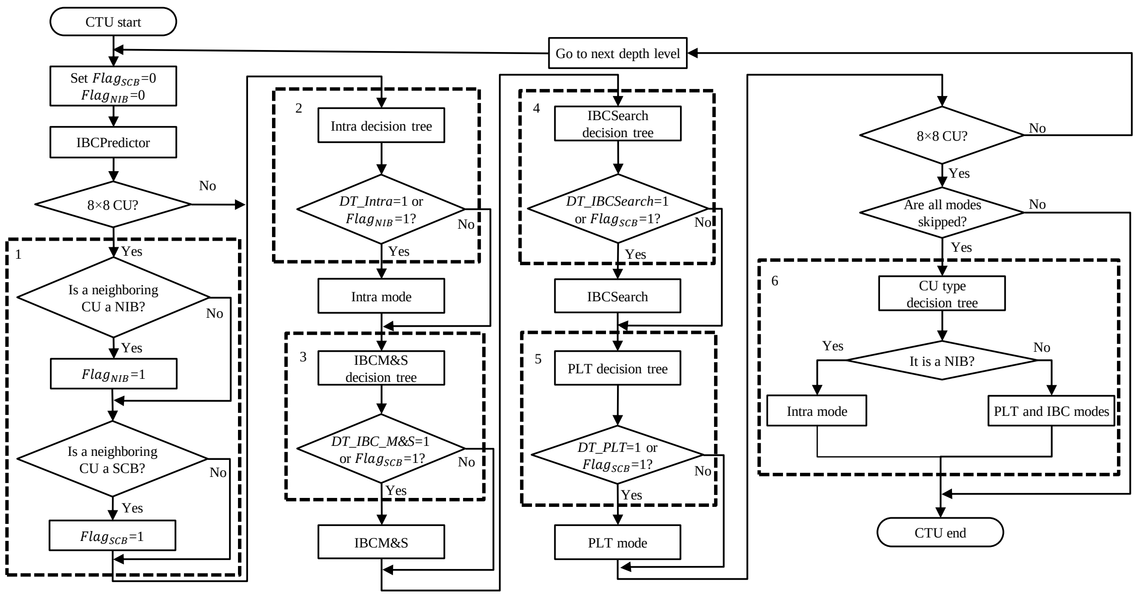
### 4.3.4 DT Constraint

To reduce the RD performance loss caused by skipping the optimal mode for a CU incorrectly, a DT constraint technique based on the spatial content correlation is derived for CUs with the size of 8×8 in this sub-section.

Table 4.6: Same content neighboring CU number distributions for 8×8 CUs.

| CU content | 0 (%) | 1 (%) | 2 (%) |
|---|---|---|---|
| NIB | 7.58 | 18.12 | 74.30 |
| SCB | 4.28 | 15.08 | 80.64 |

There is usually a strong spatial content correlation in screen content videos. A CU with the neighbor of NIBs is very likely to be a NIB while a CU with the neighbor of SCBs is very likely to be a SCB. To prove the strong spatial correlation, we encoded the frame-skipped sequences with QPs at 22, 27, 32, and 37 by using the original SCM-8.3 encoder. For each CU, the optimal modes of its top and left neighboring CUs were recorded. We treat a CU selecting Intra mode as a NIB, and a CU selecting IBC mode or PLT mode as a SCB. If a top or left neighboring CU has the same type of content as the current CU, we call it a same content neighboring CU. Table 4.6 shows the spatial content correlation of 8×8 CUs by giving the distributions of the same content neighboring CU number. We can see from the table that over 90% CUs have one or two same content neighboring CUs. Only 7.58% of SCBs and 4.28% of NIBs have no same content neighboring CU. Therefore, when encoding an 8×8 CU, if one of its neighboring CUs from the top and left selects Intra mode, i.e. $Flag_{NIB}$=1, we additionally check Intra mode for it based on the outcomes of DTs, and if one of its neighboring CUs from the top and left selects PLT or IBC mode, i.e. $Flag_{SCB}$=1, we additionally check IBC mode and PLT mode for it based on the outcomes of DTs. Although there is also the strong spatial correction of optimal modes for large CU sizes, it is unnecessary to check more mode candidates for them in order to achieve higher encoding reduction. For a large CU, if DTs assign an incorrect mode to it, it still has a chance to select good modes when partitioned into 8×8 CUs by using the DT constraint technique, so that the RD performance loss brought by the incorrect decision of large CU is decreased. The impact of the DT constraint technique will be discussed in Section 4.4.4.



Proposed techniques: 1. The DT constraint, 2. Intra mode DT, 3. IBCM&S DT, 4. IBCSearch mode DT, 5. PLT mode DT, and 6. CU type DT.

Figure 4.7: Flowchart of the proposed fast mode decision algorithm in a CTU.

All proposed techniques are treated as additional mode checking conditions based on the original encoding process when implemented in SCM-8.3. As a summary, the flowchart of the proposed fast mode decision algorithm is shown in Figure 4.7, where $Flag_{SCB}$, and $Flag_{NIB}$ are used to denote the outcome of the DT constraint technique, and *DT_Intra*, *DT_IBCM&S*, *DT_IBCSearch*, and *DT_PLT* are used to denote the outcomes of the DTs for Intra, IBCM&S, IBCSearch, and PLT modes, respectively. For simplicity, the original mode checking conditions in SCM-8.3 are not shown in this figure.

## 4.4 Experimental Results and Discussions

Four sets of experiments have been conducted to analyze the performance of the proposed work from different aspects. First, a study on the different number of training sequences is discussed. Second, the performance of the proposed framework is evaluated by comparing it with existing fast SCC encoding algorithms. Third, the contribution of

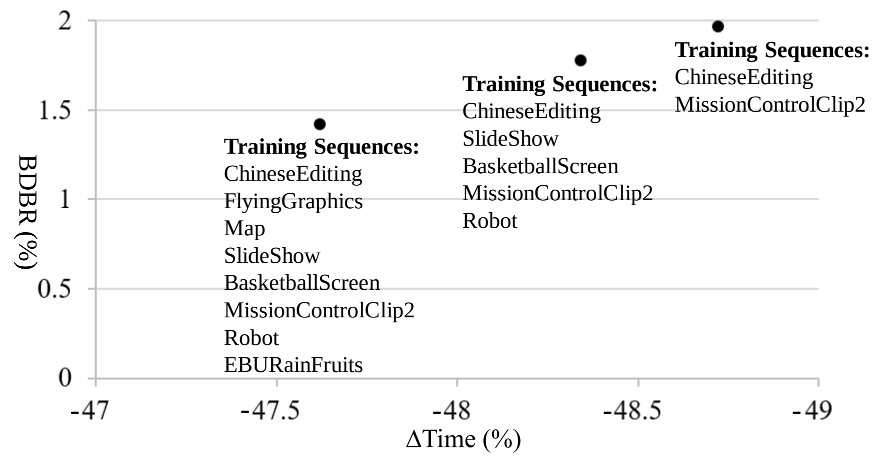


Figure 4.8: Simulation results with two, five, eight training sequences.

each individual mode decision algorithm is analyzed. At last, the efficiency of the feature subset selection and DT constraint techniques is validated.

### 4.4.1 Study on Different Training Set

To understand the impact of the training sequences to the performance of the proposed algorithm, we gradually reduce the number of training sequences and then compare their performances. Figure 4.8 shows the simulation results with two, five, eight training sequences, respectively. It is observed that the proposed algorithm can provide relatively good performance even though two training sequences are used, where 48.72% encoding time is reduced with 1.97% increase in BDBR. Besides, it is observed that using more training sequences helps to reduce the increase in BDBR. When training sequences are increased from two to eight, the increase in BDBR is reduced from 1.97% to 1.42%.

### 4.4.2 Performance Evaluation

Table 4.7 shows the performance of the proposed framework using 8 training sequences in Figure 4.8 and four state-of-the-art SCC fast intra-prediction algorithms [52]–[55] in terms of BDBR and ΔTime, where the largest value of ΔTime in each sequence is marked in boldface. It is observed that our proposed framework shows the

Table 4.7: Performance comparisons with the state-of-the-art fast intra-prediction algorithms.

| Sequences | Zhang [52] | | Duanmu [53] | | Lei [54] | | Yang [55] | | Proposed | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen (*T*) | 0.45 | -11.98 | 1.25 | -22.43 | 1.46 | -24.83 | 3.00 | -31.54 | 1.87 | **-48.60** |
| MissionControlClip2 (*T*) | 0.40 | -20.5 | 2.86 | -33.9 | 1.71 | -25.49 | 2.51 | -38.54 | 2.51 | **-47.30** |
| MissionControlClip3 (*NT*) | 0.37 | -11.28 | 2.03 | -24.61 | 1.69 | -33.81 | 2.90 | -34.15 | 1.68 | **-52.21** |
| ChineseEditing (*T*) | 0.14 | -3.40 | 1.10 | -17.47 | 0.99 | -18.96 | 4.30 | -34.16 | 0.60 | **-53.06** |
| Console (*NT*) | 2.64 | -8.23 | 1.87 | -28.12 | 2.87 | -23.40 | 7.38 | -42.83 | 0.60 | **-54.14** |
| Desktop (*NT*) | 0.67 | -4.94 | 2.19 | -26.24 | 1.97 | -23.85 | 6.27 | -35.91 | 1.03 | **-62.34** |
| FlyingGraphics (*T*) | 0.54 | -3.24 | 0.98 | -20.13 | 1.72 | -18.13 | 5.47 | -31.19 | 1.56 | **-52.13** |
| Map (*T*) | 0.97 | -10.66 | 1.55 | -19.16 | 1.23 | -20.05 | 2.84 | **-41.66** | 1.36 | -31.89 |
| Programming (*NT*) | 0.44 | -11.76 | 1.89 | -22.16 | 2.50 | -22.92 | 4.71 | -27.38 | 2.20 | **-48.94** |
| SlideShow (*T*) | 0.36 | -46.92 | 2.82 | -52.47 | 2.32 | **-55.58** | 3.69 | -34.45 | 3.76 | -35.67 |
| WebBrowsing (*NT*) | 0.79 | -6.99 | 1.91 | -28.17 | 6.02 | -26.75 | 5.00 | -53.00 | 0.98 | **-57.23** |
| Robot (*T*) | 0.43 | -17.89 | 1.18 | -29.36 | 5.21 | -46.91 | 0.59 | -28.19 | 1.51 | **-47.19** |
| EBURainFruits (*T*) | 0.21 | -18.96 | 0.88 | -26.47 | 1.76 | **-48.58** | 0.17 | -25.89 | 0.16 | -39.07 |
| Kimono1(*NT*) | 0.14 | -26.67 | 1.23 | -25.75 | 1.52 | **-75.55** | 0.13 | -36.18 | 0.05 | -36.93 |
| Average (*NT*) | 0.84 | -11.65 | 1.85 | -25.84 | 2.76 | -34.38 | 4.40 | -38.24 | 1.09 | -51.97 |
| **Average (*TGM+M*)** | 0.71 | -12.72 | 1.86 | -26.81 | 2.23 | -26.71 | 4.37 | -36.80 | 1.65 | -49.41 |
| **Average (*A+CC*)** | 0.26 | -21.17 | 1.10 | -27.19 | 2.83 | -57.01 | 0.30 | -30.09 | 0.57 | -41.06 |
| **Average (*ALL*)** | 0.61 | -14.53 | 1.70 | -26.89 | 2.36 | -33.20 | 3.50 | -35.36 | 1.42 | -47.62 |

Sequences with *T* are videos used to generate training frames. Sequences with *NT* are unseen sequences to the trained DTs.

best performance compared with other SCC fast intra-prediction algorithms [52]–[56], and it provides the largest encoding time reduction for 10 sequences out of 14 sequences. Compared with the anchor SCM-8.3, our proposed framework achieves up to 62.34% encoding time reduction on the “Desktop” sequence. On average, 47.62% encoding time reduction is obtained with a negligible increase in BDBR of 1.42%. Zhang *et al.*’s method [52] adopts the fast CU size decision framework shown in Figure 4.1(a), and it is observed in Table 4.7 that it only reduces the encoding time by 14.53% on average. Compared with Duanmu *et al.*’s method [53], Lei *et al.*’s method [54] and Yang *et al.*’s method [55] which adopt the hybrid method by combining the frameworks in Figure 4.1(a) and (b) for fast CU size decision and fast mode decision based on CU type classification, the proposed framework substantially outperforms them in both coding efficiency and computation complexity. Duanmu *et al.*’s method [53] provides 26.89% encoding time reduction while BDBR is increased by 1.70% on average. When compared with the anchor SCM-8.3, our proposed framework shows 22.60% larger encoding time reduction with 0.21% smaller increase in BDBR than Duanmu *et al.*'s method [53] for sequences in TGM+M. For sequences in A+CC, the proposed framework shows 13.87% larger encoding time reduction with 0.53% smaller increase in BDBR than Duanmu *et al.*'s method [53]. Lei *et al.*'s method [54] achieves 33.20% encoding time reduction while BDBR is increased by 2.36% on average. Although Lei *et al.*'s method [54] shows a larger encoding time reduction than the proposed framework for sequences in A and CC, the increase in BDBR is about 4 times higher than the proposed framework. For the sequences in TGM+M, Lei *et al.*'s method [54] also shows a very high increase in BDBR while the encoding time is only reduced by 26.71%. Yang *et al.*'s method [55] shows 35.36% encoding time reduction with a very high increase in BDBR of 3.50% on average. Since it always checks Intra mode for 2N×2N PUs, it brings only 0.30% increase in BDBR to the sequences in A+CC. However, the BDBR of the sequences in TGM+M is increased by 4.37% due to the low decision accuracy for SCBs.

It should be noted that our proposed DTs were trained by the frames in sequences marked with *T*, while the sequences were not used for training are marked with *NT*. It is observed in Table 4.7 that our proposed framework provides similar performance for the training sequences and the unseen sequences. Besides, the best performance of our proposed framework is not achieved for the training sequences but for the unseen "Desktop" sequence where 62.34% encoding time is reduced with 1.03% negligible increase in BDBR. Specifically, the average performances of the *NT* sequences are also shown in Table 4.7. The *NT* sequences show 51.97% encoding time reduction with 1.09% increase in BDBR, which outperforms algorithms in [52]–[55]. This shows that the proposed framework is generalizable to the unseen sequences. It is noted that the 14 sequences in CTC [30] are carefully selected to be representatives for other screen content sequences, and all existing fast SCC encoding algorithms always utilize some sequences

Table 4.8: Performance comparisons with the state-of-the-art fast intra-prediction algorithms for sequences not in CTC.

| Sequences | Zhang [52] | | Duanmu [53] | | Lei [54] | | Yang [55] | | Proposed | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BigBuckBunnyStudio | 0.64 | -20.44 | 1.90 | -31.34 | 2.58 | **-44.83** | 1.98 | -35.08 | 1.78 | -41.94 |
| ClearTypeSpreadsheet | 0.54 | -1.14 | 1.81 | -22.83 | 0.72 | -20.86 | 7.67 | -41.76 | 0.71 | **-62.56** |
| EBULupoCandlelight | 0.33 | -36.23 | 1.18 | -41.42 | 3.41 | **-66.49** | 0.43 | -43.05 | 0.13 | -38.35 |
| CadWaveform | 1.22 | -6.19 | 6.40 | -33.64 | 4.66 | -19.97 | 4.85 | -36.16 | 0.50 | **-58.40** |
| PcbLayout | 0.96 | -10.77 | 2.58 | -36.07 | 3.08 | -27.30 | 4.95 | -38.08 | 1.67 | **-48.82** |
| PptDocXls | 1.35 | -4.31 | 1.47 | -24.01 | 1.29 | -17.38 | 4.16 | -32.72 | 0.74 | **-55.70** |
| RealTimeData | 3.32 | -6.19 | 1.55 | -26.15 | 2.11 | -22.43 | 7.11 | -34.65 | 0.82 | **-50.62** |
| VideoConferencingDocSharing | 0.37 | -3.60 | 1.57 | -24.57 | 3.63 | -19.93 | 7.06 | -34.76 | 0.55 | **-58.27** |
| Viking | 0.33 | -18.66 | 1.00 | -30.41 | 5.00 | **-65.84** | 0.47 | -29.61 | 1.36 | -45.02 |
| WordEditing | 0.36 | -10.30 | 0.97 | -23.17 | 1.24 | -24.42 | 4.21 | -42.10 | 1.57 | **-53.71** |
| **Average ($ALL$)** | 0.94 | -11.78 | 2.04 | -29.36 | 2.78 | -32.95 | 4.29 | -36.80 | 0.98 | -51.34 |

from CTC [30] for both training and testing. To further show the generalization of the proposed algorithm to other screen content sequences, ten more test sequences [84]–[88] that are not included in CTC [30] were evaluated. The results are shown in Table 4.8 with comparison to the existing fast SCC encoding algorithms [52]–[55], where the largest value of ΔTime in each sequence is marked in boldface. It is observed that the proposed algorithm again outperforms the fast SCC encoding algorithms [52]–[55], and it provides the largest encoding time reduction for seven sequences out of the ten test sequences. Although Lei *et al.*'s method [54] shows a larger encoding time reduction in the other three sequences, the increase in BDBR is remarkably higher than the proposed framework. On average, the fast SCC encoding algorithms [52]–[55] reduce 11.78%–36.80% encoding time with 0.94%–4.29% increase in BDBR. Comparatively, the proposed algorithm reduces 51.34% encoding time with only 0.98% increase in BDBR. Again, this confirms the generalization ability of the proposed algorithm.

The proposed algorithm includes additional processes of feature extraction and decision determination for making fast mode decision, and these computational overheads are further analyzed and summarized in Table 4.9. It is observed that the

Table 4.9: Average computational overheard of the proposed algorithm.

| Computational Overhead | Proportion (%) | |
|---|---|---|
| | Feature Extraction | Decision Determination |
| | 1.32 | 0.01 |

average computational overhead proportions of feature extraction and decision determination are only 1.32% and 0.01%, respectively. It is noted that these computational overheads have been counted in Table 4.7 to calculate the encoding time reduction.

We also extend our work to support sequences in YUV 4:2:0 and RGB 4:4:4 formats based on the same methodology, and their performances are summarized in Table 4.10. It is observed that for sequences in YUV 4:2:0 and RGB 4:4:4 formats, encoding time of 41.68% and 49.98% is reduced with 1.68% and 1.41% increase in BDBR on average, respectively. The results are very similar to that of YUV 4:4:4 sequences, which demonstrates the proposed framework is generalizable to other color formats. Since the fast SCC encoding algorithms [52]–[55] only investigated the fast prediction for YUV 4:4:4 sequences, we cannot make comparisons for sequences in YUV 4:2:0 and RGB 4:4:4 formats.

Since Intra-prediction is also needed in inter frame coding, Figure 4.9 also shows the impact of the proposed algorithm on inter frame coding under Low Delay (LD) configuration. BDBR and ΔTime of five typical sequences in YUV 4:4:4 format are

Table 4.10: Average ΔTime and BDBR of the proposed algorithm for YUV 4:2:0 and RGB 4:4:4 sequences under CTC.

| Sequence Categories | YUV 4:2:0 | | RGB 4:4:4 | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Average (TGM+M) | 1.66 | -37.92 | 1.58 | -48.98 |
| Average (A+CC) | 1.79 | -59.82 | 0.80 | -55.30 |
| Average (ALL) | 1.68 | -41.29 | 1.41 | -49.98 |

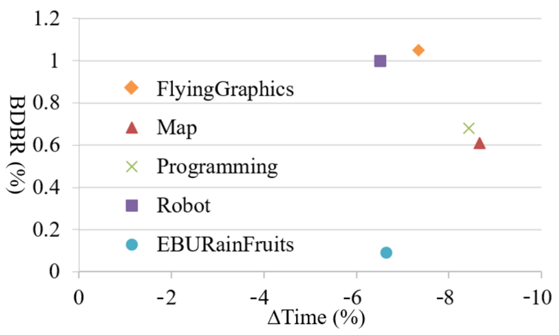


Figure 4.9: BDBR and ΔTime of the proposed algorithm under LD configuration.

shown in Figure 4.9, and similar results are observed for other sequences. It is observed that the proposed algorithm reduces 6.52%–8.44% encoding time with negligible increase in BDBR, which implies the proposed algorithm also benefits to inter frame coding.

### 4.4.3 Performance of the Individual Mode Decision Algorithm

To further investigate the contribution of each mode decision algorithm, additional experiments were performed by implementing DTs for IBC mode, PLT mode, IBC+PLT

Table 4.11: Performance of each individual mode decision algorithm and their combinations for YVU 4:4:4 sequences.

| Sequences | IBC Mode Decision | | PLT Mode Decision | | PLT+IBC Mode Decision | | Intra Mode Decision | | Overall Framework | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen (*T*) | 1.05 | -22.76 | 0.21 | -8.98 | 1.31 | -31.58 | 0.56 | -17.85 | 1.87 | -48.60 |
| MissionControlClip2 (*T*) | 1.65 | -21.72 | 0.58 | -11.66 | 2.10 | -33.55 | 0.74 | -14.50 | 2.51 | -47.30 |
| MissionControlClip3 (*NT*) | 0.91 | -24.09 | 0.30 | -8.28 | 1.04 | -31.53 | 0.67 | -19.42 | 1.68 | -52.21 |
| ChineseEditing (*T*) | 0.31 | -26.52 | 0.11 | -2.01 | 0.45 | -27.40 | 1.21 | -24.14 | 1.56 | -52.13 |
| Console (*NT*) | 0.34 | -13.97 | 0.06 | -4.24 | 0.57 | -17.97 | 0.85 | -14.56 | 1.36 | -31.89 |
| Desktop (*NT*) | 0.87 | -21.64 | 0.93 | -6.17 | 1.88 | -27.62 | 0.37 | -20.84 | 2.20 | -48.94 |
| FlyingGraphics (*T*) | 2.26 | -16.23 | 0.41 | -11.09 | 3.09 | -28.26 | 0.71 | -8.31 | 3.76 | -35.67 |
| Map (*T*) | 0.67 | -31.58 | 0.11 | -3.83 | 0.78 | -34.83 | 0.25 | -22.56 | 0.98 | -57.23 |
| Programming (*NT*) | 0.35 | -21.79 | 0.27 | -1.37 | 0.62 | -23.08 | 0.17 | -29.54 | 0.60 | -53.06 |
| SlideShow (*T*) | 0.15 | -26.56 | 0.12 | -2.12 | 0.25 | -27.68 | 0.37 | -27.81 | 0.60 | -54.14 |
| WebBrowsing (*NT*) | 0.45 | -33.40 | 0.37 | -1.58 | 0.65 | -34.64 | 0.33 | -27.74 | 1.03 | -62.34 |
| Robot (*T*) | 0.57 | -23.08 | 0.67 | -17.97 | 1.38 | -43.02 | 0.12 | -3.11 | 1.51 | -47.19 |
| EBURainFruits (*T*) | 0.12 | -26.12 | 0 | -11.03 | 0.13 | -38.15 | 0.02 | 0.22 | 0.16 | -39.07 |
| Kimono1(*NT*) | 0.04 | -24.28 | 0 | -9.76 | 0.04 | -37.48 | 0.04 | 2.51 | 0.05 | -36.93 |
| **Average (*NT*)** | 0.52 | -26.93 | 0.31 | -5.29 | 0.77 | -32.30 | 0.34 | -19.31 | 1.09 | -51.97 |
| **Average (*TGM+M*)** | 0.82 | -23.66 | 0.32 | -5.58 | 1.16 | -28.92 | 0.57 | -20.66 | 1.65 | -49.41 |
| **Average (*A+CC*)** | 0.24 | -24.49 | 0.22 | -12.92 | 0.52 | -39.55 | 0.06 | -0.13 | 0.57 | -41.06 |
| **Average (*ALL*)** | 0.70 | -23.84 | 0.30 | -7.15 | 1.02 | -31.20 | 0.46 | -16.26 | 1.42 | -47.62 |

Sequences with *T* are videos used to generate training frames. Sequences with *NT* are unseen sequences to the trained DTs.

modes, Intra mode, respectively, and the results are shown in Table 4.11. We can see from the table that IBC mode DTs provide the largest encoding time reduction, followed by Intra mode and PLT mode, which are 23.84%, 16.26% and 7.15%, respectively. When the DTs of IBC mode and PLT mode are both implemented, sequences in A+CC show 39.55% encoding time reduction, which is nearly the same as the results of the overall framework with all DTs enabled. It is because nearly all CUs in A+CC sequences are encoded by Intra mode, and it shows that the IBC and PLT DTs can efficiently skip these CUs. Smaller encoding time is saved for sequences in TGM+M because they contain many SCBs, so that fewer IBC mode and PLT mode are skipped. In contrast, only 0.13% encoding time reduction is observed for sequences in A+CC when only the Intra mode DTs are implemented. For sequences with almost pure SCBs, such as “ChineseEditing”, “Console” and “Desktop”, over 27% encoding time is saved by using Intra mode DTs. The reason is that almost all CUs in these sequences can skip Intra mode and large encoding time reduction is achieved. Furthermore, Table 4.11 shows that the overall framework provides 23.52%–34.60% larger encoding time reduction for “ChineseEditing”, “Console” and “Desktop”, as compared with the results that only intra mode DTs are enabled. The reason is that besides the Intra mode which is not suitable for encoding SCBs, the overall framework considers PLT mode and IBC mode separately and then further skips unnecessary PLT mode and IBC mode for SCBs. To support this statement, we investigated the mode decision of the proposed overall framework by encoding all sequences with QPs of 22, 27, 32, 37, and the average distribution of mode decision is shown in Table 4.12.

It is observed that in the depth level of 3, IBC or PLT mode is always checked with other modes because of the DT constraint technique. However, the proposed overall framework is very efficient for larger CU sizes. In the depth level of 0, 56.84% CUs

Table 4.12: Mode decision distribution of the proposed overall algorithm for YVU 4:4:4 sequences.

| CU Size | Intra only | IBC only | PLT only | Intra+IBC | Intra+PLT | IBC+PLT | Intra+IBC+PLT | No Mode |
|---|---|---|---|---|---|---|---|---|
| 64×64 | 27.60 | 6.70 |  | 8.86 |  |  |  | 56.84 |
| 32×32 | 41.79 | 1.15 | 41.52 | 0.08 | 4.30 | 3.674 | 0.02 | 7.48 |
| 16×16 | 53.23 | 3.89 | 24.31 | 1.49 | 3.06 | 12.18 | 1.03 | 0.82 |
| 8×8 | 52.06 | 0 | 0 | 4.01 | 1.97 | 27.14 | 14.81 | 0 |

directly go to the next depth level because only Intra mode and IBC mode exist. In the depth levels of 1 and 2, 42.67% and 28.20% CUs select either IBC mode or PLT mode, respectively. By further reducing the mode candidates for SCBs in the proposed overall framework, it provides 20.48%–38.49% larger encoding time reduction for “ChineseEditing”, “Console” and “Desktop” compared with [53]–[55], as shown in Table 4.7.

### 4.4.4 Evaluation of Feature Subset Selection and DT Constraint

To validate the efficiency of the feature subset selection and the DT constraint techniques, experiments were performed by implementing the overall framework without the feature subset selection and DT constraint techniques, and the results are shown in Table 4.13. Compared with the case without performing feature subset selection, the proposed overall framework provides 3.97% larger encoding time reduction with 0.13% decrease in BDBR. Therefore, better performance is provided by adopting the feature subset selection technique, because the impact of irrelevant or redundant features is removed. Besides, it is observed that the DT constraint technique helps to reduce BDBR increase of the proposed framework at the cost of less encoding time reduction. On average, the encoding time saving of the proposed framework is slightly reduced from 52.90% to 47.62% while the increase in BDBR is reduced from 3.07% to 1.42% by implementing the DT constraint technique. Specifically, we can see that the performance improvement for

Table 4.13: Performances of the proposed algorithm with other settings for YVU 4:4:4 Sequences.

| Sequences | Without Feature Subset Selection | | Without DT Constraint | | Overall Framework | |
|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BasketballScreen | 0.60 | -48.06 | 1.77 | -57.46 | 1.87 | -48.60 |
| MissionControlClip2 | 0.81 | -48.06 | 1.75 | -59.07 | 2.51 | -47.30 |
| MissionControlClip3 | 1.00 | -58.62 | 3.06 | -66.93 | 1.68 | -52.21 |
| ChineseEditing | 1.68 | -42.28 | 3.35 | -55.79 | 1.56 | -52.13 |
| Console | 1.52 | -28.83 | 3.38 | -40.19 | 1.36 | -31.89 |
| Desktop | 2.17 | -43.10 | 4.98 | -55.51 | 2.20 | -48.94 |
| FlyingGraphics | 3.93 | -33.84 | 6.94 | -56.91 | 3.76 | -35.67 |
| Map | 1.25 | -57.68 | 2.17 | -60.75 | 0.98 | -57.23 |
| Programming | 1.79 | -43.58 | 4.72 | -54.01 | 0.60 | -53.06 |
| SlideShow | 3.13 | -42.57 | 4.85 | -51.85 | 0.60 | -54.14 |
| WebBrowsing | 1.98 | -45.97 | 3.52 | -56.63 | 1.03 | -62.34 |
| Robot | 1.61 | -45.69 | 2.11 | -48.76 | 1.51 | -47.19 |
| EBURainFruits | 0.15 | -37.16 | 0.29 | -39.13 | 0.16 | -39.07 |
| Kimono1 | 0.05 | -35.68 | 0.07 | -37.64 | 0.05 | -36.93 |
| **Average (*TGM+M*)** | 1.81 | -44.78 | 3.68 | -55.92 | 1.65 | -49.41 |
| **Average (*A+CC*)** | 0.60 | -39.51 | 0.82 | -41.84 | 0.57 | -41.06 |
| **Average (*ALL*)** | 1.55 | -43.65 | 3.07 | -52.90 | 1.42 | -47.62 |

sequences in A+CC is limited, but sequences in TGM+M gain large benefits from the DT constraint technique and the increase in BDBR is reduced by 2.03%. The reason is that IBC and PLT modes are very effective in 8×8 SCBs. Therefore, even a small incorrect decision rate of IBC and PLT modes in 8×8 SCBs leads to large RD performance loss. By using the DT constraint technique, additional mode candidates are available at the last depth level (depth level of 3), and the RD performance loss brought by the incorrect decision is reduced effectively.

## 4.5 Chapter Summary

In this chapter, a machine learning based fast mode decision framework is proposed for SCC. To avoid the exhaustive mode searching process, a flexible intra mode decision framework is proposed by utilizing a sequential arrangement of mode classifiers. Compared with the traditional methods that IBC and PLT modes are both checked for

SCBs, we insert a DT before checking each mode with the help of new dynamic features, so that the decision of each mode is made separately, and it allows the case that only one mode is checked for a SCB. Experimental results have shown that the proposed framework can provide an average computational complexity reduction of 47.62% with a negligible increase in BDBR of 1.42%.

# Chapter 5 Determinations on Coding Structure by Convolutional Neural Network

## 5.1 Introduction

The DT-based fast SCC encoding algorithm in Chapter 4 and some other recent algorithms heavily rely on the limited number of hand-crafted features or heuristic rules. However, they have the risk that human may ignore some important features during feature extraction. In this chapter, we present a deep learning based fast prediction network, DeepSCC, to reduce the computational complexity of SCC. DeepSCC takes raw sample values as the input, and it makes fast predictions for all CUs of a CTU in a single test. It contains much more trainable parameters than the traditional machine learning based approaches, so that it is able to make more accurate classification. This chapter is started by introducing the difference between dynamic and stationary CTUs. Then we present the proposed deep learning network DeepSCC. Next, experimental results of the proposed algorithm are provided. Finally, conclusion is given for this chapter.

Parts of the contents of this chapter are extracted from our submitted work [89]:

- **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, “DeepSCC: Deep Learning Based Fast Prediction Network for Screen Content Coding,” IEEE Transactions on Circuits and Systems for Video Technology (Accepted on 2 July, 2019).

Table 5.1: Percentage of stationary areas in different sequences.

| Categories | Sequences | Stationary CTU (%) |
|---|---|---|
| TGM | ChineseEditing | 93.41 |
| | Console | 62.72 |
| | Desktop | 78.57 |
| | FlyingGraphics | 2.50 |
| | Map | 79.20 |
| | Programming | 48.11 |
| | SlideShow | 75.41 |
| | WebBrowsing | 96.43 |
| M | BasketballScreen | 86.80 |
| | MissionControlClip2 | 83.82 |
| | MissionControlClip3 | 73.78 |
| A | Robot | 0 |
| CC | EBURainFruits | 0 |
| | Kimono1 | 0 |
| **Average (*TGM*+*M*)** | | 70.98 |
| **Average (*A*+*CC*)** | | 0 |
| **Average (*ALL*)** | | 55.77 |

## 5.2 Difference Between Dynamic and Stationary CTUs

Unlike the traditional camera-captured sequences only containing dynamic CTUs which show different content in adjacent frames, screen content sequences contain many stationary CTUs, i.e., the sum of SAD between the current CTU and its collocated CTU is 0. Table 5.1 shows the percentage of stationary CTUs in different sequences. It is observed that sequences in A+CC only contain dynamic CTUs, while 70.98% CTUs in TGM+M sequences are stationary CTUs. To simplify the encoding of stationary CTUs, an intuitive idea is to directly encode stationary CTUs with the same optimal modes of the collocated CTUs. However, this approach brings high RD performance loss because whether a CU selects the same mode as its collocated CU is related to its actual content. For example, a SCB with simple texture usually has many repeated patterns within the current frame while a SCB with complex texture has few repeated patterns. If the collocated CU of a simple SCB selects IBC mode, this SCB usually select IBC mode. On

Table 5.2: Performance of encoding stationary CTUs with the same optimal modes of the collocated CTUs of stationary areas in different sequences.

| Categories | Sequences | BDBR (%) | ΔTime (%) |
|---|---|---|---|
| TGM | ChineseEditing | 3.44 | -51.81 |
| | Console | 4.20 | -47.83 |
| | Desktop | 6.16 | -57.22 |
| | FlyingGraphics | 0.13 | -1.28 |
| | Map | 4.94 | -49.39 |
| | Programming | 2.52 | -31.84 |
| | SlideShow | 13.01 | -43.26 |
| | WebBrowsing | 8.55 | -65.98 |
| M | BasketballScreen | 7.38 | -47.51 |
| | MissionControlClip2 | 12.40 | -44.42 |
| | MissionControlClip3 | 6.77 | -48.67 |
| A | Robot | 0 | 0 |
| CC | EBURainFruits | 0 | 0 |
| | Kimono1 | 0 | 0 |
| **Average (*TGM*+*M*)** | | 6.32 | -44.37 |
| **Average (*A*+*CC*)** | | 0 | 0 |
| **Average (*ALL*)** | | 4.96 | -34.95 |

the contrary, if the collocated CU of a complex SCB selects IBC mode, this SCB may select PLT mode since its very limited repeated patterns can be disappeared in the current frame. Table 5.2 shows BDBR and the change in encoding time, ΔTime, brought by this approach compared with the original SCM-8.3. It is observed that for sequences in TGM+M that contain many stationary CTUs, this approach provides 44.37% encoding time reduction, but it brings 6.32% increase in BDBR. Although the algorithms in [51], [54], [56] utilize some heuristic rules to reduce the RD performance loss brought by this approach, such as disabling the fast approach every ten frames to avoid error propagation [51], and jointly analyzing the coding information from the collocated CU and spatial neighboring CUs [54], [56], they still do not achieve a good trade-off between encoding time reduction and BDBR. To further improve the performance for stationary CTUs, it is desired that the optimal mode of the collocated CTU and the actual CTU content are jointly analyzed.

## 5.3 Proposed Deep Learning based Network DeepSCC

Generally, humans are sensitive to the difference between SCBs and NIBs, so that many approaches [53]–[56] have successfully differentiated SCBs from NIBs relying on a limited number of hand-crafted features. However, it is inefficient to make further classification inside SCBs with hand-crafted features because humans are less sensitive to the difference between IBC-coded SCBs and PLT-coded SCBs. To overcome the limitation of hand-crafted features, a deep learning based fast prediction network DeepSCC is proposed, which contains two parts, DeepSCC-I and DeepSCC-II. DeepSCC-I is used to make predictions for dynamic CTUs, while DeepSCC-II is used to make predictions for stationary CTUs. Since the proposed DeepSCC contains many trainable parameters and learns extensive features, it is able to make the more accurate mode decision of Intra, IBC, and PLT rather than the simple CU type classification of NIBs and SCBs. The previous fast prediction approaches of SCC always make predictions in the CU level, which means the derived model is tested for multiple times to make fast prediction for a single CTU. The drawback of this strategy is that it scarifies the encoding time reduction due to the multiple tests of the derived models. To reduce the computation overhead, the proposed DeepSCC directly outputs 85 labels for 85 CUs of a CTU in a single test. Since a CU can either skip all modes or select one mode from Intra, IBC, and PLT, each predicted label contains the probabilities of four classes, i.e., $P(Allskip)$, $P(Intra)$, $P(IBC)$, and $P(PLT)$, in accordance with the probabilities for skipping all modes, and checking Intra, IBC, PLT modes, respectively. Noted that *Allskip* is not an actual mode, and a SCC encoder will not employ it to encode a CU. We include class *Allskip* to denote the case that video content should be encoded in other depth levels, so that all modes of Intra, IBC, and PLT in the current CU can be skipped for computational complexity reduction. Figure 5.1 illustrates the structure of the proposed DeepSCC,

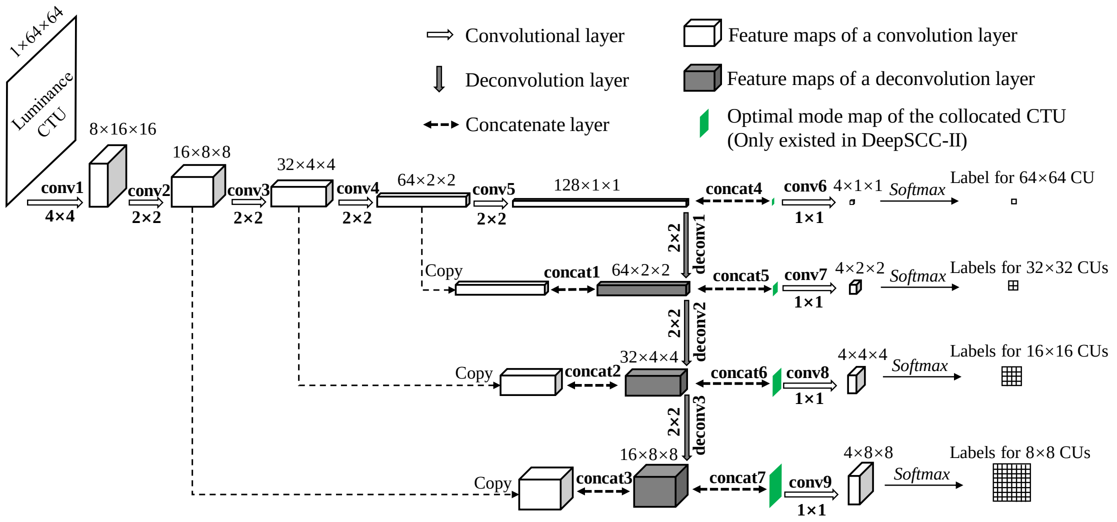


Figure 5.1: Structure of DeepSCC. The optimal mode maps of the collocated CTU only exist in DeepSCC-II, which is denoted by green blocks.

where the kernel sizes and feature map dimensions are also presented. The only difference between DeepSCC-I and DeepSCC-II is that the optimal mode maps of the collocated CTU are concatenated to the extracted feature maps before going through the convolution layers conv6–conv9 in DeepSCC-II, which is denoted by green color. The details of DeepSCC are given in the following sub-sections.

### 5.3.1 DeepSCC-I for Dynamic CTU

As shown in Figure 5.1, DeepSCC-I takes the luminance component of a CTU as the input. It is noted that the luminance component of a CTU is preprocessed by mean removal before it is fed to DeepSCC-I. Finally, it is able to output 85 labels for 85 CUs with different sizes, where each label shows the probabilities of selecting different modes. DeepSCC-I is composed of nine convolutional layers (conv1–conv9), three deconvolutional layers (deconv1–deconv3), and three concatenating layers (concat1–concat3). Each convolutional or deconvolutional layer is followed by a rectified linear unit (ReLU) activation function, except for conv6–conv9, where softmax is utilized to generate the output labels. The details of these layers are presented as followings.

*Convolutional layers:* At the beginning, the luminance component of a CTU goes through five convolutional layers, i.e., conv1–conv5, to generate feature maps. As shown in Figure 5.1, the kernel size of conv1 is 4×4 and the kernel sizes of conv2–conv5 are 2×2. The strides of conv1–conv5 are set to the width of the kernel sizes for non-overlapping convolutions, in accordance with the non-overlapping CU partitioning structure. By using this arrangement, the receptive field of each node in a feature map is always equal to a CU size, so that the feature maps of conv2–conv5 reflect the local features of CUs from 8×8 to 64×64, respectively. It is noted that the spatial down-sampling in DeepSCC-I is achieved by convolutions with strides, rather than the deterministic spatial pooling functions. Therefore, it allows the network to learn its own spatial down-sampling strategy. At each down-sampling step, we double the channel number of feature maps. After concatenating the feature maps of convolutional layers and deconvolutional layers, conv6–conv9 incorporate those feature maps and generate the last set of feature maps with the kernel size of 1×1 and stride of 1. Each layer of conv6–conv9 outputs four feature maps since each CU contains four classes. Finally, the feature maps of conv6–conv9 are used to output the predicted labels after going through a softmax function.

*Deconvolutional layer*s: In contrast to the convolution layer which reduces the size of a feature map, a deconvolutional layer is used to enlarge the size of a feature map. After generating the 128 feature maps of conv5 with the size of 1×1, three deconvolutional layers i.e., deconv1–deconv3, are used to enlarge the feature maps of conv5 using the kernel size of 2×2 and stride of 2. Since the receptive field of each node in the feature maps of conv5 is the entire CTU, the receptive field of each node in the feature maps of deconv1–deconv3 also becomes the entire CTU, and they reflect the global features for CUs with size from 32×32 to 8×8, respectively. The global features help to improve the prediction accuracy because there exists spatial content correlation in a CTU. For example, if other CUs are SCBs in a CTU, the current CU is more likely to be a SCB and it would check IBC or PLT mode. On the contrary, if other CUs are NIBs in a CTU, the current CU is more likely to be a NIB and it would check Intra mode. At each feature map enlarging step, we halve the channel number of feature maps. Finally, the global feature maps and local feature maps have the same dimension for each CU size.

*Concatenating layers*: DeepSCC-I adopts three concatenating layers, i.e., concat1–concat3, to concatenate the global feature maps and local feature maps for CUs with sizes from 32×32 to 8×8, respectively.

As shown in Figure 5.1, DeepSCC-I outputs 1, 4, 16, and 64 labels for a CTU, in accordance with the hierarchical CTU partitioning structure in Figure 1.2, which contains 1 CU of 64×64 pixels, 4 CUs of 32×32 pixels, 16 CUs of 16×16 pixels, and 64 CUs of 8×8 pixels.

### 5.3.2 DeepSCC-II for Stationary CTU

As analyzed in Section 5.2, directly encoding a stationary CTU with the same optimal modes of the collocated CTU leads to a very high increase in BDBR. To address this problem, the optimal mode maps of the collocated CTU are jointly analyzed with the actual CTU content to reduce the BDBR loss for stationary CTUs. By defining the indices for classes of *Allskip*, *Intra*, *IBC* and *PLT* as 0, 1, 2 and 3, an example of a collocated CTU and its optimal mode maps is shown in Figure 5.2. To obtain the optimal mode maps of a collocated CTU, its optimal modes are analyzed in four depth levels. Since the CTU in Figure 5.2 is not encoded as a single 64×64 CU, the optimal mode map for the 64×64 CU has the class index of *Allskip*, which means all modes are skipped in the 64×64 CU. Then, there are two 32×32 CUs encoded by PLT mode and Intra mode in the CTU,

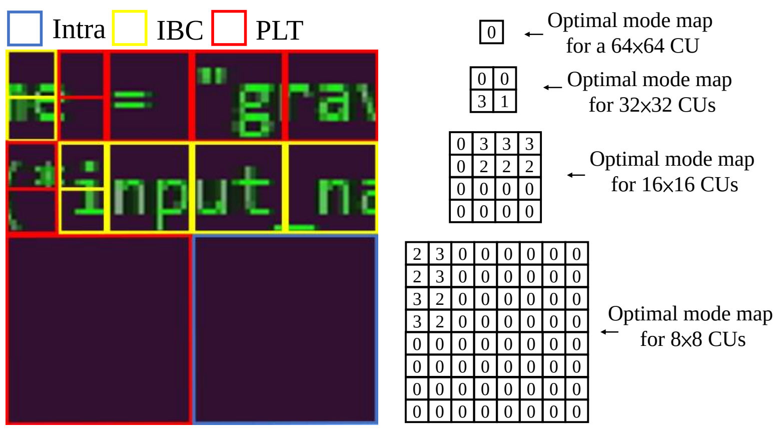


Figure 5.2: A collocated CTU and its optimal mode maps.

respectively, so that the class indices of the corresponding positions in the optimal mode map for 32×32 CUs are 3 and 1, which denote *PLT* and *Intra*, respectively. The other two positions in this optimal mode map still contain the index of *Allskip* since they are not encoded as 32×32 CUs. This process is repeated until the four optimal mode maps are all generated in the collocated CTU.

It is noted that the four optimal mode maps of the collocated CTU have the same size as the corresponding feature maps from conv2–covn5 and deconv1–deconv3. To utilize the optimal mode correlation between the current stationary CTU and its collocated CTU, the four optimal mode maps of the collocated CTU are concatenated to the corresponding feature maps of the current CTU by using four concatenate layers concat4–concat7, as shown in Figure 5.1. After using conv6–conv9 to incorporate those feature maps and the optimal mode maps, a softmax function is used to output the predicted labels.

### 5.3.3 Training Strategy for DeepSCC

To avoid the overlapping between the training set and testing set, we selected 12 training sequences from [84]–[88], [90] which are not included in CTC [30] to generate the training samples. Based on the content classification criterion of CTC, we also classify the 12 training sequences into the four categories of TGM, M, A and CC, and they are

Table 5.3: Training sequences for DeepSCC.

| Categories | Sequences | Resolution | No. of Frame |
|---|---|---|---|
| TGM | ClearTypeSpreadsheet | 1920×1080 | 300 |
| | PptDocXls | 1920×1080 | 200 |
| | RealTimeData | 1920×1080 | 600 |
| | WordEditing | 1920×1080 | 600 |
| | VideoConferencingDocSharing | 1280×720 | 300 |
| M | BigBuck | 1920×1080 | 400 |
| | KristenAndSaraScreen | 1920×1080 | 600 |
| | MissionControlClip1 | 2560×1440 | 600 |
| A | Viking | 1280×720 | 300 |
| CC | EBULupoCandlelight | 1920×1080 | 250 |
| | Seeking | 1920×1080 | 250 |
| | ParkScene | 1920×1080 | 240 |

shown in Table 5.3. Then, the 14 sequences in CTC are used as the testing sequences to evaluate the performance of the proposed DeepSCC. A single model of DeepSCC is trained for QPs of 22, 27, 32, and 37 by using training data from the four QPs. For each training sequence, 50 frames were extracted with equal intervals, and they were encoded by the original SCM-8.3 with QPs of 22, 27, 32 and 37 to obtain the ground truth labels. Finally, 750,000 CTUs were generated with their ground truth labels to train DeepSCC-I, while 440,000 CTUs with their ground truth labels and the optimal mode maps of the collocated CTUs were obtained to train DeepSCC-II.

The training process of DeepSCC was implemented in Caffe [91]. A GPU of GeForce GTX 1080 Ti was used to accelerate the training process, and then it was disabled in the testing phase, so that only a CPU was used to evaluate the performance of DeepSCC. To make the maximum use of GPU memory, a large batch size of 1024 CTUs was adopted. The loss of an *i-th* training sample in a batch is defined as the sum of cross-entropy over all labels in four depth levels, and it is represented by

$$l_i = f(\omega^0, \hat{\omega}^0) + \sum_{j=0}^{3} f\left(\omega^{1\_j}, \hat{\omega}^{1\_j}\right) + \sum_{j=0}^{15} f(\omega^{2\_j}, \hat{\omega}^{2\_j}) + \sum_{j=0}^{63} f(\omega^{3\_j}, \hat{\omega}^{3\_j}) \quad (5.1)$$

where $\omega^0$ and $\omega^{1\_j}$, $\omega^{2\_j}$, $\omega^{3\_j}$ denote the ground truth classes of the CU in the depth level of 0, and the *j-th* CU in the depth levels of 1, 2, 3, respectively. Similarly, $\hat{\omega}^{0\_j}$, $\hat{\omega}^{1\_j}$,

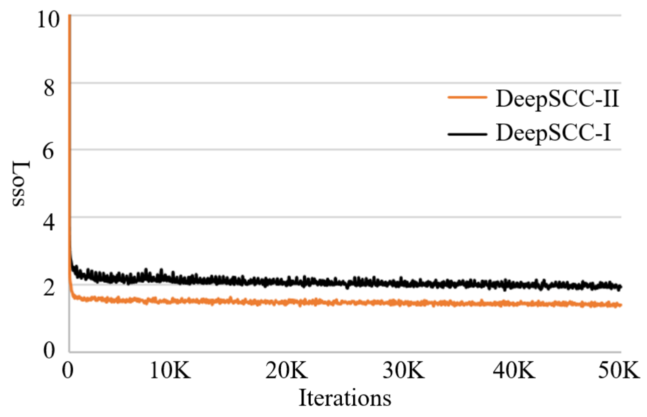


Figure 5.3: Training loss of DeepSCC-I and DeepSCC-II alongside iterations.

$\widehat{\omega}^{2\_j}$, and $\widehat{\omega}^{3\_j}$ denote the predicted classes of the corresponding CUs. $f()$ represents the cross-entropy function between the ground truth class and predicted class, and it is represented as

$$f(\omega, \widehat{\omega}) = -\sum_k y(\omega_k = \omega)\, log(P(\omega_k = \widehat{\omega})) \qquad (5.2)$$

where $k$ denotes the class index. $y(\omega_k = \omega)$ is 1 if $\omega_k$ is the same as the ground truth class $\omega$, otherwise, $y(\omega_k = \omega)$ is 0. $P(\omega_k = \widehat{\omega})$ denotes the probability that $\omega_k$ is the same as the predicted class $\widehat{\omega}$. By averaging the loss over all training samples in one batch, the loss function $L$ is written as

$$L = \frac{1}{N}\sum_{i=1}^{N} l_i \qquad (5.3)$$

where $N$ is the number of training samples in one batch. All trainable parameters in DeepSCC are initialized by the "msra" filter [92]. Then, Adam optimizer [93] is adopted to update the trainable parameters in DeepSCC with the default values of momentum and momentum2, which are 0.9 and 0.999, respectively. A weight decay of 0.005 is used to alleviate the overfitting problem. Instead of using the conventional learning rate policy of "Step", we adopt the learning rate policy of "Poly" as in [94], and the learning rate in each iteration ($iter$), $lr_{iter}$, is

$$lr_{iter} = lr_{base} \times (1 - \frac{iter}{max_{iter}})^{power} \qquad (5.4)$$

where $lr_{base}$ is the base learning rate of 0.01, $power$ is set to 0.9, and $max_{iter}$ is set to 50,000. Started from 0.01, $lr_{iter}$ is gradually reduced to 0 as the iteration increases.

The training losses of DeepSCC-I and DeepSCC-II calculated by Equation (5.3) are shown in Figure 5.3. Compared with the traditional classification task, the mode decision framework only has three class, and it is easier to make classification, so it is observed that the training processes of DeepSCC-I and DeepSCC-II converge very fast. Besides, the final loss of DeepSCC-II is smaller than DeepSCC-I, because DeepSCC-II additionally utilizes the optimal mode maps of the collocated CTUs. Although DeepSCC-I can reduce encoding time for all CTUs by only taking sample values as the input, we only enable it for dynamic CTUs. For stationary CTUs, DeepSCC-II is enabled instead of DeepSCC-I because it has a smaller loss. The advantage of DeepSCC-II over DeepSCC-I for stationary CTUs is further discussed in Section 5.4.3.

### 5.3.4 Content-adaptive Threshold

To make fast prediction for an input CTU, the proposed DeepSCC outputs 85 labels for 85 CUs, and each label contains four probabilities, i.e., $P(\omega)$, $\omega \in \{Allskip, Intra, IBC, PLT\}$. In the testing phase, a threshold $\alpha_x$ is used to decide whether a CU needs to check the mode *x*, $x \in \{Intra, IBC, PLT\}$. If the probability of checking a mode *x* is smaller than the value of $\alpha_x$, i.e., $P(\omega=x)<\alpha_x$, the mode *x* is regarded as unnecessary, and the current CU does not check it for encoding time reduction. It should be noted that the selection of the class *Allskip* is not directly decided but depended on the probabilities of checking other classes from $\{Intra, IBC, PLT\}$. If the probabilities of checking all classes from $\{Intra, IBC, PLT\}$ are smaller than $\alpha_x$, the optimal class of the CU becomes *Allskip*, and the mode checking for the CU can be skipped.

In SCC, NIBs and SCBs usually show the concentrated distribution in a frame, and there exists an optimal mode correlation in spatial neighboring CUs. Therefore, $\alpha_x$ is

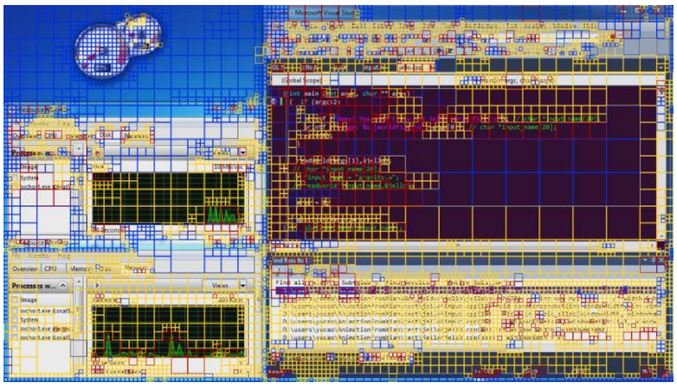


Figure 5.4: Optimal mode in the first frame of "Programming". Intra, IBC and PLT modes coded CUs are noted by blue, yellow and red blocks, respectively.

treated as a content-adaptive threshold, and its value is adjusted by utilizing the spatial optimal mode correlation. The mode distribution of the first frame in "Programming" is shown in Figure 5.4, and it was encoded by the original SCM-8.3 with QP of 22. It is observed that many CUs select the same modes as their top or left CUs at the same depth levels. Besides, many IBC-coded CUs and PLT-coded CUs are mixed together because IBC and PLT modes are both valid mode candidates for SCBs. Based on this observation, the value of $\alpha_x$ for a CU is decided by the optimal modes of its top and left neighboring CUs at the same depth level

$$\alpha_x = \alpha_{base} - I_x \times \alpha_{decay} \tag{5.5}$$

where $I_x$ is a content-adaptive parameter, $\alpha_{base}$ and $\alpha_{decay}$ are two predefined parameters that control the value of $\alpha_x$. The impact of their values to DeepSCC is discussed in Section 5.4.1. Since IBC and PLT modes show mixed distribution, they are grouped together to decide the value of $I_x$. For $x$ ∈{IBC, PLT}, $I_x$ is represented as

$$I_x = \begin{cases} 1, if\ \omega_{left} \in \{IBC, PLT\}\ or\ \omega_{top} \in \{IBC, PLT\} \\ 0, \qquad otherwise \end{cases} \tag{5.6}$$

For $x \in \{Intra\}$, $I_x$ is represented as

$$I_x = \begin{cases} 1, if\ \omega_{left} \in \{Intra\}\ or\ \omega_{top} \in \{Intra\} \\ 0, \qquad oherwise \end{cases} \tag{5.7}$$

where $\omega_{left}$ and $\omega_{top}$ are the optimal mode classes of the left and top neighboring CUs. By using the content-adaptive threshold $\alpha_x$, a CU has a larger chance to be coded by the optimal modes of its left and top CUs.

Since the proposed DeepSCC treats the case of skipping all modes as the class *Allskip* in mode decision, the CU partitioning decision is integrated into DeepSCC. If DeepSCC selects the class *Allskip* for a CU, it means that the current depth level is not optimal and the mode checking of the CU is skipped. Therefore, additional testing of another model specially designed for CU partitioning decision as in [53]–[56] is not necessary, and it further reduces the testing time. Before a CU in the depth level of 0, 1, or 2 continues the partitioning process shown in Figure 1.2, the labels of CUs in the deeper depth levels are analyzed to perform CU partitioning decision. If an area of a CU always selects the class *Allskip* in all deeper depth levels, the CU cannot be encoded if it continues partitioning. Therefore, we early terminate the CU partitioning process to avoid unnecessary computation.

### 5.3.5 Memory Overhead of DeepSCC

To make fast prediction, the trained Caffe model needs to be invoked in SCM-8.3. The memory overhead of DeepSCC comes from two parts, which are the size of the parameters stored in the Caffe model and the size of generated feature maps when running DeepSCC. If a CTU is a dynamic CTU, the Caffe model of DeepSCC-I is invoked, which takes up 348.47KB. To store the generated feature maps, 47.66KB is needed by using the double-precision floating point which requires 8B in C language. Therefore, the memory overhead is 396.13KB for DeepSCC-I. On the other hand, the Caffe model of DeepSCC-II is invoked for stationary CTUs, which takes up 348.80KB, and the associated feature maps require 48.32KB. Therefore, the memory overhead is 397.12KB for DeepSCC-II. Comparatively, a video frame with the resolution of 2560×1440 pixels takes up 108,00KB

Table 5.4: Validation sequences for DeepSCC.

| Categories | Sequences | Resolution | No. of Frame |
|---|---|---|---|
| TGM | BitstreamAnalyzer | 1920×1080 | 300 |
| | Doc | 1280×720 | 500 |
| | Web | 1280×720 | 500 |
| M | KimonoError2 | 2560×1440 | 500 |
| CC | BirdsInCage | 1920×1080 | 600 |
| | DucksAndLegs | 1920×1080 | 300 |
| | Traffic | 2560×1440 | 60 |
| | VenueVu | 1920×1080 | 300 |

(2560×1440×3÷1024). Therefore, the memory overhead percentages of DeepSCC-I and DeepSCC-II over the frame memory are only 3.67% and 3.68%, respectively.

## 5.4 Experimental Results and Discussions

To evaluate the performance of the proposed DeepSCC, it has been implemented in SCM-8.3 [31], and the DNN tool of OpenCV 3.4.1 is used to invoke the trained Caffe model in SCM-8.3. The trained Caffe model and the source code of the proposed DeepSCC can be found on our website [95]. It should be noted that no GPU but only a CPU is enabled for making fair comparisons. Three sets of experiments have been conducted to analyze the performance of the proposed DeepSCC. First, a series of ablation experiments were performed to decide the optimal structure of DeepSCC by using validation sequences [86], [88], [90], [96] in Table 5.4. Second, the performance of DeepSCC is evaluated by comparing it with existing fast SCC prediction algorithms. Third, the performances of the individual DeepSCC-I and DeepSCC-II are analyzed.

### 5.4.1 Ablation Study

In this sub-section, various experiments were performed to decide the optimal structure of the proposed DeepSCC by using the validation sequences shown in Table 5.4.

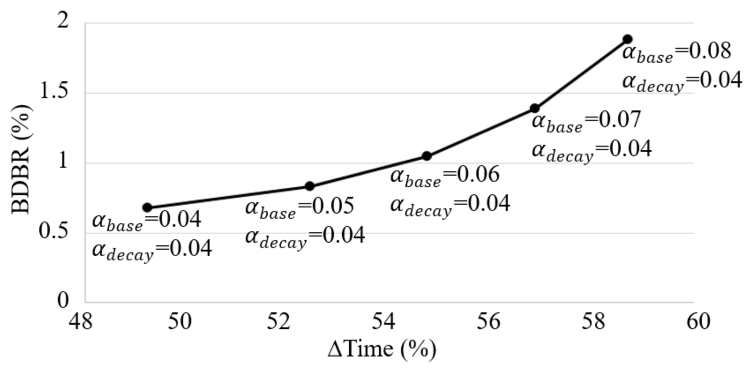


Figure 5.5: Performance of DeepSCC with various values of $\alpha_{base}$ and the fixed value of $\alpha_{decay}$.

### A. Threshold Determination

As aforementioned in Section 5.3.4, a content-adaptive threshold $\alpha_x$ is used to eliminate unnecessary mode candidates in a CU, and its value is controlled by two predefined parameters $\alpha_{base}$ and $\alpha_{decay}$. First, a fixed value of $\alpha_{decay}$ is applied to analyze the impact of $\alpha_{base}$, and the results are shown in Figure 5.5. It is observed that as the value of $\alpha_{base}$ increases, more encoding time is reduced at the cost of a larger increase of BDBR. Besides, when the gap between $\alpha_{base}$ and $\alpha_{decay}$, i.e., $\alpha_{base} - \alpha_{decay}$, is large, BDBR increases quickly. For example, when the gap between $\alpha_{base}$ and $\alpha_{decay}$ increases from 0 to 0.02, the encoding time is further reduced by 5.50% while BDBR is further increased by only 0.37%. When the gap between $\alpha_{base}$ and $\alpha_{decay}$ increases from 0.02 to 0.04, the encoding time is further reduced by 3.96% while BDBR is further increased by 0.86%. Therefore, we limit the gap between $\alpha_{base}$ and $\alpha_{decay}$ to a small value to balance the encoding time reduction and increase in BDBR, and the results are shown in Table 5.5. It is observed that the DeepSCC is complexity scalable and it provides 46.60%–56.34% encoding time reduction with BDBR increased by 0.48–1.33%. In the following sub-sections, $\alpha_{base}$ is set to 0.05 and $\alpha_{decay}$ is set to 0.04 for further discussions, where 52.35% encoding time is reduced with 0.83% increase in BDBR.

Table 5.5: Performance of the proposed DeepSCC for validation sequences with different values of $\alpha_{base}$ and $\alpha_{decay}$.

| Sequences | $\alpha_{base}$=0.03 $\alpha_{decay}$=0.02 | | $\alpha_{base}$=0.03 $\alpha_{decay}$=0.01 | | $\alpha_{base}$=0.05 $\alpha_{decay}$=0.04 | | $\alpha_{base}$=0.05 $\alpha_{decay}$=0.03 | | $\alpha_{base}$=0.07 $\alpha_{decay}$=0.06 | | $\alpha_{base}$=0.07 $\alpha_{decay}$=0.05 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 0.43 | -40.74 | 0.44 | -41.49 | 0.79 | -45.29 | 0.91 | -45.43 | 1.42 | -48.60 | 1.52 | -49.35 |
| Doc | 0.38 | -45.05 | 0.34 | -44.13 | 1.35 | -50.47 | 1.52 | -51.56 | 2.06 | -55.60 | 2.13 | -56.70 |
| Web | 0.86 | -47.17 | 1.03 | -48.32 | 1.43 | -52.71 | 1.57 | -54.04 | 2.71 | -56.66 | 2.99 | -57.82 |
| KimonoError2 | 0.66 | -31.63 | 0.70 | -29.75 | 0.76 | -37.24 | 0.82 | -37.14 | 0.90 | -39.98 | 0.93 | -39.98 |
| BirdsInCage | 0.04 | -49.37 | 0.04 | -49.95 | 0.09 | -59.07 | 0.10 | -59.03 | 0.14 | -63.01 | 0.14 | -63.10 |
| DucksAndLegs | 0.17 | -62.23 | 0.17 | -62.22 | 0.24 | -64.21 | 0.24 | -64.48 | 0.33 | -65.51 | 0.33 | -65.76 |
| Traffic | 0.57 | -51.02 | 0.59 | -51.33 | 0.90 | -56.96 | 0.91 | -56.64 | 1.16 | -60.05 | 1.16 | -60.37 |
| VenueVu | 0.76 | -45.55 | 0.81 | -45.93 | 1.08 | -52.85 | 1.10 | -52.60 | 1.36 | -56.77 | 1.40 | -57.67 |
| **Average (*TGM+M*)** | 0.58 | -41.15 | 0.63 | -40.92 | 1.08 | -46.43 | 1.21 | -47.04 | 1.77 | -50.21 | 1.89 | -50.96 |
| **Average (*A+CC*)** | 0.39 | -52.04 | 0.40 | -52.36 | 0.58 | -58.27 | 0.59 | -58.19 | 0.75 | -61.34 | 0.76 | -61.73 |
| **Average (*ALL*)** | 0.48 | -46.60 | 0.52 | -46.64 | 0.83 | -52.35 | 0.90 | -52.62 | 1.26 | -55.77 | 1.33 | -56.34 |

### B. Decoupling Local Features and Global Features

The proposed DeepSCC utilizes convolutional layers and deconvolutional layers to extract local features and global features in a CTU, respectively. Then, they are concatenated together to predict the mode labels. To evaluate the importance of the proposed structure, two sets of experiments were performed by decoupling local features and global features, i.e., removing concat1–concat3 from DeepSCC. First, only the feature maps of conv2–conv5 are fed to concat4–concat7 so that only local features are utilized to make mode prediction. Second, only the feature maps of conv5 and deconv1–

Table 5.6: Performance comparison of LFDeepSCC, GFDeepSCC and DeepSCC.

| Sequences | LFDeepSCC | | GFDeepSCC | | DeepSCC | |
|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 0.82 | -45.20 | 0.83 | -40.86 | 0.79 | -45.29 |
| Doc | 1.44 | -49.02 | 1.79 | -50.18 | 1.35 | -50.47 |
| Web | 1.77 | -51.11 | 1.45 | -51.61 | 1.43 | -52.71 |
| KimonoError2 | 0.87 | -37.28 | 0.82 | -39.75 | 0.76 | -37.24 |
| BirdsInCage | 0.03 | -29.38 | 0.11 | -58.02 | 0.09 | -59.07 |
| DucksAndLegs | 0.03 | -20.02 | 0.19 | -61.32 | 0.24 | -64.21 |
| Traffic | 0.29 | -35.24 | 0.93 | -52.94 | 0.90 | -56.96 |
| VenueVu | 0.57 | -27.93 | 1.10 | -50.58 | 1.08 | -52.85 |
| **Average (*TGM+M*)** | 1.23 | -45.65 | 1.22 | -45.60 | 1.08 | -46.43 |
| **Average (*A+CC*)** | 0.23 | -28.14 | 0.58 | -55.72 | 0.58 | -58.27 |
| **Average (*ALL*)** | 0.73 | -36.90 | 0.90 | -50.66 | 0.83 | -52.35 |

3 are fed to concat4–concat7 so that only global features are utilized to make mode prediction. Let us call them LFDeepSCC and GFDeepSCC, respectively, and their performances are shown in Table 5.6. It is observed that LFDeepSCC provides 36.90% encoding time reduction with 0.73% increase in BDBR. DeepSCC outperforms it by providing a much higher encoding time reduction of 52.35% with a similar increase in BDBR. GFDeepSCC also shows worse performance than DeepSCC by providing 50.66% encoding time reduction with 0.90% increase in BDBR. Therefore, concatenating the local features and global features helps to improve the performance of the proposed DeepSCC.

### C. Term Normalization in Loss Function

In Equation (5.1), the loss function of a training sample is derived as the sum of cross-entropy over all labels in the four depth levels, and the terms for different depth levels are not normalized. For example, the loss function contains only one term in the depth level of 0 while it contains 64 terms in the depth level of 4. The reason that we do not normalize the loss function to let the terms of different depth levels have equal weight is that the mode classifications in deeper depth levels are more complex. Therefore, the

Table 5.7: Performance comparison of DeepSCC with and without term normalization in loss function.

| Sequences | DeepSCC with term normalization | | DeepSCC without term normalization | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 1.82 | -38.81 | 0.79 | -45.29 |
| Doc | 0.92 | -47.90 | 1.35 | -50.47 |
| Web | 1.10 | -49.04 | 1.43 | -52.71 |
| KimonoError2 | 0.82 | -33.19 | 0.76 | -37.24 |
| BirdsInCage | 0.07 | -51.96 | 0.09 | -59.07 |
| DucksAndLegs | 0.22 | -61.15 | 0.24 | -64.21 |
| Traffic | 0.72 | -40.78 | 0.90 | -56.96 |
| VenueVu | 0.92 | -44.79 | 1.08 | -52.85 |
| **Average (*TGM+M*)** | 1.17 | -42.23 | 1.08 | -46.43 |
| **Average (*A+CC*)** | 0.48 | -49.67 | 0.58 | -58.27 |
| **Average (*ALL*)** | 0.82 | -45.95 | 0.83 | -52.35 |

loss function without term normalization will be naturally more focused on the mode classification of small CUs. To prove its advantage, Table 5.7 shows the performance of DeepSCC using loss function with term normalization. It is observed that the original DeepSCC outperforms DeepSCC with term normalization by providing 6.4% more encoding time reduction with almost the same increase in BDBR.

### D. Feature Fusion Function

To join the convolution features, deconvolution features, and optimal mode maps of the collocated CTU, the concatenating layer is adopted in DeepSCC. It is one of the most widely used feature fusion layers and it can join feature maps with the arbitrary channel number. An alternative way is to use element wise addition layer which can only join two sets of feature maps with the equal channel numbers. Therefore, element wise addition layers can be adopted to join convolution features and deconvolution features since they have equal channel numbers, then they are concatenated to the optimal mode maps of the collocated CTU. Table 5.8 shows the results of DeepSCC with element wise addition layers. It is observed it almost shows the same results as the original DeepSCC. Therefore, different feature fusion functions have a minor impact on the DeepSCC.

Table 5.8: Performance comparison of DeepSCC with different feature funsion functions.

| Sequences | DeepSCC with element wise addition layer | | DeepSCC with concatenating layer | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 0.78 | -44.69 | 0.79 | -45.29 |
| Doc | 1.24 | -49.20 | 1.35 | -50.47 |
| Web | 1.60 | -52.15 | 1.43 | -52.71 |
| KimonoError2 | 0.81 | -37.30 | 0.76 | -37.24 |
| BirdsInCage | 0.09 | -61.91 | 0.09 | -59.07 |
| DucksAndLegs | 0.22 | -64.27 | 0.24 | -64.21 |
| Traffic | 0.78 | -54.61 | 0.90 | -56.96 |
| VenueVu | 1.00 | -53.16 | 1.08 | -52.85 |
| **Average (*TGM+M*)** | 1.11 | -45.84 | 1.08 | -46.43 |
| **Average (*A+CC*)** | 0.52 | -58.49 | 0.58 | -58.27 |
| **Average (*ALL*)** | 0.82 | -52.16 | 0.83 | -52.35 |

Table 5.9: Performance comparison of different learning policies.

| Sequences | DeepSCC with "Step" | | DeepSCC with "Poly" | |
|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 1.30 | -47.50 | 0.79 | -45.29 |
| Doc | 1.48 | -50.01 | 1.35 | -50.47 |
| Web | 1.68 | -51.02 | 1.43 | -52.71 |
| KimonoError2 | 0.73 | -36.25 | 0.76 | -37.24 |
| BirdsInCage | 0.09 | -57.68 | 0.09 | -59.07 |
| DucksAndLegs | 0.32 | -63.11 | 0.24 | -64.21 |
| Traffic | 0.82 | -53.89 | 0.90 | -56.96 |
| VenueVu | 1.12 | -50.01 | 1.08 | -52.85 |
| **Average (*TGM*+*M*)** | 1.30 | -46.20 | 1.08 | -46.43 |
| **Average (*A*+*CC*)** | 0.59 | -56.17 | 0.58 | -58.27 |
| **Average (*ALL*)** | 0.94 | -51.18 | 0.83 | -52.35 |

### E. Learning Policy

In the training process of DeepSCC, the learning rate policy of "Poly" is adopted rather than the conventional "Step". To evaluate the efficiency of this strategy, experiments were done by training DeepSCC with "Step" with the same values of $lr_{base}$ and $max_{iter}$, and the learning rate is multiplied by 0.1 every 10,000 iterations. The performance comparison is shown in Table 5.9. It is observed that DeepSCC with "Step" achieves 51.18% encoding time reduction with 0.94% increase in BDBR. By replacing "Step" with "Poly", DeepSCC shows slightly better performance of 1.17% larger encoding time reduction and 0.11% smaller increase in BDBR.

### F. Number of Channels

The proposed DeepSCC has the advantage of automatically learning useful features by using extensive learnable parameters, which is controlled by the number of channels in each layer. If a small number of channels are employed, DeepSCC may run into the underfitting problem. On the contrary, if a larger number of channels are employed, DeepSCC may run into the overfitting problem. As shown in Figure 5.1, the channel number of the feature maps after going through conv1 is 8. Before fed to concat4–concat7,

Table 5.10: Performance comparison of DeepSCC with different number of channels.

| Sequences | NumChannel/4 | | NumChannel/2 | | Original DeepSCC | | NumChannel×2 | | NumChannel×4 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| BitstreamAnalyzer | 1.35 | -38.14 | 1.67 | -38.9 | 0.79 | -45.29 | 1.36 | -47.26 | 1.10 | -47.93 |
| Doc | 1.36 | -49.86 | 1.32 | -50.52 | 1.35 | -50.47 | 1.31 | -47.23 | 1.35 | -47.88 |
| Web | 1.32 | -50.67 | 1.50 | -51.76 | 1.43 | -52.71 | 1.45 | -51.23 | 1.41 | -50.56 |
| KimonoError2 | 0.78 | -35.67 | 0.79 | -38.04 | 0.76 | -37.24 | 0.68 | -36.72 | 0.91 | -36.82 |
| BirdsInCage | 0.08 | -55.92 | 0.07 | -66.94 | 0.09 | -59.07 | 0.07 | -55.16 | 0.09 | -50.04 |
| DucksAndLegs | 0.17 | -43.67 | 0.18 | -62.15 | 0.24 | -64.21 | 0.23 | -64.74 | 0.29 | -63.38 |
| Traffic | 0.53 | -49.04 | 0.79 | -56.14 | 0.90 | -56.96 | 0.96 | -52.78 | 1.02 | -52.23 |
| VenueVu | 1.10 | -49.85 | 1.12 | -54.27 | 1.08 | -52.85 | 1.10 | -53.85 | 1.11 | -45.24 |
| **Average (*TGM+M*)** | 1.20 | -43.59 | 1.32 | -44.81 | 1.08 | -46.43 | 1.20 | -45.61 | 1.19 | -45.80 |
| **Average (*A+CC*)** | 0.47 | -49.62 | 0.54 | -59.88 | 0.58 | -58.27 | 0.59 | -56.63 | 0.63 | -52.72 |
| **Average (*ALL*)** | 0.84 | -46.60 | 0.93 | -52.34 | 0.83 | -52.35 | 0.90 | -51.12 | 0.91 | -49.26 |

the channel number of the feature maps is doubled if they go through a convolutional layer or halved if they go through a deconvolutional layer. To evaluate the impact of the channel number in DeepSCC, another four sets of experiments were performed, i.e., multiplying the channel number of each layer before concat4–concat7 by 1/4, 1/2, 2, 4, and they are denoted as NumChannel/4, NumChannel/2, NumChannel×2, NumChannel×4, respectively. The performance comparison of DeepSCC with the different number of channels is shown in Table 5.10. It is observed that the original DeepSCC shows slightly better performance than the networks with the other number of channels. As the channel number increases, the performance of DeepSCC is improved first because of underfitting, and then it is dropped because of overfitting. Therefore, DeepSCC with the proposed channel number achieves a good trade-off.

### 5.4.2 Performance of DeepSCC

Table 5.11 shows the performance of the proposed DeepSCC for training sequences. It is observed that DeepSCC provides 50.17% encoding time reduction with 1.13% negligible increase in BDBR. As compared with the results for validation sequences in Table 5.5, DeepSCC provides similar performance for both training sequences and

Table 5.11: Performance of the proposed DeepSCC for training sequences.

| Training Sequences | BDBR (%) | ΔTime (%) |
|---|---|---|
| ClearTypeSpreadsheet | 1.01 | -53.59 |
| PptDocXls | 1.99 | -45.60 |
| RealTimeData | 1.04 | -40.91 |
| WordEditing | 1.40 | -53.54 |
| VideoConferencingDocSharing | 1.86 | -52.61 |
| BigBuck | 1.18 | -42.48 |
| KristenAndSaraScreen | 0.90 | -46.69 |
| MissionControlClip1 | 1.37 | -47.43 |
| Viking | 1.78 | -54.40 |
| EBULupoCandlelight | 0.25 | -53.60 |
| Seeking | 0.30 | -52.65 |
| ParkScene | 0.47 | -58.51 |
| **Average (*TGM+M*)** | 1.34 | -47.86 |
| **Average (*A+CC*)** | 0.70 | -54.79 |
| **Average (*ALL*)** | 1.13 | -50.17 |

validation sequences. This shows that the proposed DeepSCC is generalizable to the unseen sequences.

Then, to evaluate the performance of the proposed DeepSCC, it is directly compared with four state-of-the-art SCC fast intra-prediction algorithms [51], [53]–[55]. The results for 14 testing sequences in YUV 4:4:4 format are shown in Table 5.12. It is observed that DeepSCC outperforms the SCC fast intra-prediction algorithms [51], [53]–[55] by providing 48.81% encoding time reduction with only 1.18% increase in BDBR. Specifically, DeepSCC provides 46.12% and 58.69% encoding time reduction with 1.30% and 0.76% increase in BDBR for sequences in TGM+M and A+CC, respectively. Comparatively, Zhang *et al.*'s method [51] shows 1.25% increase in BDBR, which is similar to the proposed DeepSCC. However, DeepSCC outperforms it by providing 15.62% larger encoding time reduction. Since Zhang *et al.*'s method [51] strongly relies on CUs having similar content as their collocated CUs, it shows very limited encoding time reduction for sequences with almost only dynamic regions, such as "FlyingGraphics", "Robot", "EBURainFruits", and "Kimono1", where only 4.60%, 12.04%, 16.48%, and 0.46% encoding time is reduced. Comparatively, DeepSCC can efficiently address

Table 5.12: Performance of different algorithms compared with SCM-8.3 for sequences in YUV 4:4:4 format.

| Sequences | Zhang [51] | | Duanmu [53] | | Lei [54] | | Yang [55] | | DeepSCC | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) | BDBR (%) | $\Delta Time$ (%) |
| ChineseEditing | 0.65 | -49.73 | 1.10 | -17.47 | 0.99 | -18.96 | 4.30 | -34.16 | 1.07 | -48.80 |
| Console | 3.36 | -39.35 | 1.87 | -28.12 | 2.87 | -23.40 | 7.38 | -42.83 | 1.06 | -41.85 |
| Desktop | 1.95 | -47.94 | 2.19 | -26.24 | 1.97 | -23.85 | 6.27 | -35.91 | 1.00 | -53.46 |
| FlyingGraphics | 0.84 | -4.60 | 0.98 | -20.13 | 1.72 | -18.13 | 5.47 | -31.19 | 0.99 | -30.76 |
| Map | 0.85 | -36.95 | 1.55 | -19.16 | 1.23 | -20.05 | 2.84 | -41.66 | 1.79 | -36.36 |
| Programming | 1.16 | -40.44 | 1.89 | -22.16 | 2.50 | -22.92 | 4.71 | -27.38 | 0.87 | -42.74 |
| SlideShow | 1.39 | -44.15 | 2.82 | -52.47 | 2.32 | -55.58 | 3.69 | -34.45 | 2.78 | -55.36 |
| WebBrowsing | 2.05 | -51.73 | 1.91 | -28.17 | 6.02 | -26.75 | 5.00 | -53.00 | 0.88 | -54.09 |
| BasketballScreen | 1.06 | -41.84 | 1.25 | -22.43 | 1.46 | -24.83 | 3.00 | -31.54 | 1.27 | -46.78 |
| MissionControlClip2 | 1.29 | -39.08 | 2.86 | -33.90 | 1.71 | -25.49 | 2.51 | -38.54 | 1.56 | -51.16 |
| MissionControlClip3 | 1.05 | -39.91 | 2.03 | -24.61 | 1.69 | -33.81 | 2.90 | -34.15 | 1.01 | -45.96 |
| Robot | 0.92 | -12.04 | 1.18 | -29.36 | 5.21 | -46.91 | 0.59 | -28.19 | 1.81 | -49.43 |
| EBURainFruits | 0.71 | -16.48 | 0.88 | -26.47 | 1.76 | -48.58 | 0.17 | -25.89 | 0.29 | -55.94 |
| Kimono1 | 0.15 | -0.46 | 1.23 | -25.75 | 1.52 | -75.55 | 0.13 | -36.18 | 0.17 | -70.69 |
| **Average (*TGM*+*M*)** | 1.42 | -39.61 | 1.86 | -26.81 | 2.23 | -26.71 | 4.37 | -36.80 | 1.30 | -46.12 |
| **Average (*A*+*CC*)** | 0.59 | -9.66 | 1.10 | -27.19 | 2.83 | -57.01 | 0.30 | -30.09 | 0.76 | -58.69 |
| **Average (*ALL*)** | 1.25 | -33.19 | 1.70 | -26.89 | 2.36 | -33.20 | 3.50 | -35.36 | 1.18 | -48.81 |

dynamic CTUs, and it provides 30.76%, 49.73%, 55.94% and 70.68% encoding time reduction for those sequences. Duanmu *et al.*'s method [53], Lei *et al.*'s method [54] and Yang *et al.*'s method [55] all eliminate the mode candidates for a CU by classifying it into a NIB or a SCB, and at most one mode, i.e., Intra mode, is skipped for a SCB. On the contrary, DeepSCC directly performs the mode classification rather than the simple CU type classification, so IBC and PLT modes are no longer always checked together for a SCB. As a result, DeepSCC outperforms the fast algorithms [53]–[55] by providing 21.92%, 15.61% and 13.45% larger encoding time reduction with 0.52%, 1.18% and 2.32% smaller increase in BDBR, respectively.

Furthermore, we make an indirect comparison of the proposed DeepSCC with Huang *et al.*'s method [56] because we do not have the source code of their approach. However, the proposed DeepSCC is implemented in the same reference software as Huang *et al.*'s method [56], SCM-8.3, which makes the indirect comparison to be fair. Huang *et al.*'s method [56] adopts a hybrid framework of neural network-based classifiers for CU type

Table 5.13: Indirect comparison for sequences in YUV 4:4:4 format.

| Sequences | Huang [56] | | Proposed DeepSCC | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| ChineseEditing | | | 1.07 | -48.80 |
| Console | | | 1.06 | -41.85 |
| Desktop | 0.84 | -46.48 | 1.00 | -53.46 |
| FlyingGraphics | 1.10 | -43.45 | 0.99 | -30.76 |
| Map | 1.25 | -42.60 | 1.79 | -36.36 |
| Programming | 2.05 | -53.66 | 0.87 | -42.74 |
| SlideShow | 1.54 | -68.38 | 2.78 | -55.36 |
| WebBrowsing | 0.99 | -55.33 | 0.88 | -54.09 |
| BasketballScreen | 0.87 | -39.83 | 1.27 | -46.78 |
| MissionControlClip2 | 1.47 | -46.39 | 1.56 | -51.16 |
| MissionControlClip3 | 1.63 | -39.42 | 1.01 | -45.96 |
| Robot | 2.52 | -40.31 | 1.81 | -49.43 |
| EBURainFruits | 0.67 | -50.56 | 0.29 | -55.94 |
| Kimono1 | 1.35 | -65.74 | 0.17 | -70.69 |
| **Average (*Huang [56]'s sequences*)** | 1.36 | -49.34 | 1.20 | -49.39 |
| **Average (*ALL*)** | | | 1.18 | -48.81 |

classification and various heuristic rules to make CU partitioning decisions. Comparatively, the proposed DeepSCC integrates the mode decision and CU partitioning decision into the same network. Therefore, DeepSCC is easier for implementation than Huang *et al.*'s method [56]. As observed in Table 5.13, Huang *et al.*'s method [56] provides 49.34% encoding time reduction with 1.36% increase in BDBR for their selected sequences. However, the proposed DeepSCC outperforms Huang *et al.*'s method [56] by providing nearly the same encoding time reduction with 0.16% less increase in BDBR. Besides, the training sequences of Huang *et al.*'s method [56] are partly overlapped with its testing sequences, where "WebBrowsing" and "Kimono1" are utilized as both training sequences and testing sequences. On the contrary, the training sequences and testing sequences of the proposed DeepSCC are totally different, which avoids the situation of overfitting.

Table 5.14 shows the prediction accuracy of the proposed DeepSCC by calculating the percentage of the areas encoded by the same mode as the original SCM-8.3. It is observed that the prediction accuracy of DeepSCC is very high and it varies from 94.47%

Table 5.14: Prediction accuracy of the proposed DeepSCC.

| Sequences | QP=22 | QP=27 | QP=32 | QP=37 |
|---|---|---|---|---|
| ChineseEditing | 96.79 | 96.85 | 96.53 | 96.03 |
| Console | 97.77 | 97.71 | 97.58 | 96.62 |
| Desktop | 98.10 | 97.96 | 97.81 | 97.49 |
| FlyingGraphics | 97.67 | 97.70 | 97.71 | 97.39 |
| Map | 96.57 | 96.86 | 97.22 | 97.05 |
| Programming | 96.27 | 96.83 | 97.07 | 97.08 |
| SlideShow | 96.40 | 97.29 | 97.79 | 98.17 |
| WebBrowsing | 98.58 | 98.12 | 98.40 | 98.07 |
| BasketballScreen | 96.79 | 96.91 | 97.67 | 97.38 |
| MissionControlClip2 | 96.17 | 96.88 | 97.58 | 97.59 |
| MissionControlClip3 | 96.86 | 97.32 | 97.60 | 97.75 |
| Robot | 94.47 | 96.48 | 97.46 | 97.52 |
| EBURainFruits | 97.11 | 97.86 | 98.40 | 98.30 |
| Kimono1 | 96.53 | 98.38 | 98.70 | 98.71 |
| **Average (*TGM+M*)** | 97.09 | 97.31 | 97.54 | 97.33 |
| **Average (*A+CC*)** | 96.07 | 97.57 | 98.19 | 98.18 |
| **Average (*ALL*)** | 96.86 | 97.37 | 97.68 | 97.51 |

to 98.71% for different sequences with different QPs. On average, the prediction accuracy is 96.86%, 97.37%, 97.68%, 97.51% for QP of 22, 27, 32, 37, respectively. Since the proposed DeepSCC is trained by using a mixed training data from 22, 27, 32 and 37, it shows stable accuracy for the testing sequenecs under the four QPs.

Figure 5.6 shows the RD curve and ΔTime for four sequences over different QPs by using DeepSCC, and it is noted that other sequences have similar results. It is observed that the RD curves of DeepSCC are very close to those of the original SCC encoder, which indicates that DeepSCC has a negligible influence on video quality. Besides, ΔTime varies little over different QPs for all sequences. Therefore, DeepSCC provides stable performance in both high and low bitrate cases.

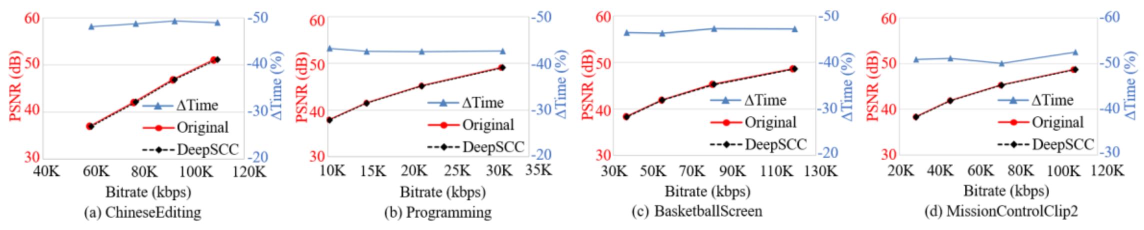


Figure 5.6: RD curve and ΔTime of the proposed DeepSCC for “ChineseEditing”, “Programming”, “BasketballScreen”, and “MissionControlClip2”.

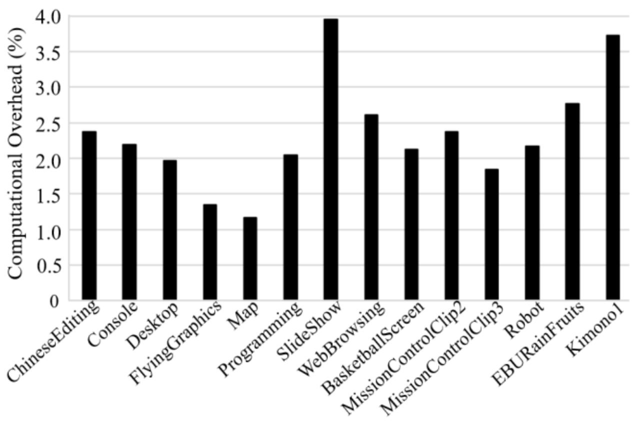


Figure 5.7: Computational overhead of the proposed DeepSCC.

Figure 5.7 shows the computational overhead of the proposed DeepSCC, which is calculated as the ratio of running DeepSCC for mode prediction to the total encoding time of various sequences. Since DeepSCC adopts non-overlapping convolutions and outputs 85 labels in a single test, the computational overhead is very low, which is from 1.17% to 3.94% of the total encoding time for all test sequences. It is noted that the computational overhead is included to calculate the total encoding time of the proposed DeepSCC for all simulations.

Table 5.15 shows the performance of DeepSCC applied to sequences in RGB 4:4:4 and YUV 4:2:0 formats. While the luminance samples of sequences in YUV 4:2:0 format are directly input to DeepSCC, color space conversion is performed for sequences in RGB 4:4:4 format to get the luminance samples. It should be noted that DeepSCC is only trained by sequences in YUV 4:4:4 format. However, DeepSCC shows good generalization for sequences in YUV 4:2:0 and RGB 4:4:4 formats, where 46.49% and 43.69% encoding time can be reduced with only 1.13% and 1.29% increase in BDBR, respectively. Since YUV 4:4:4 is the most widely adopted format for screen content sequences and most existing fast SCC prediction algorithms do not support sequences in other formats, we cannot make the comparison for sequences in YUV 4:2:0 and RGB 4:4:4 formats.

Table 5.15: Performance of DeepSCC for sequences in RGB 4:4:4 and YUV 4:2:0 formats.

| Sequences | RGB 4:4:4 | | YUV 4:2:0 | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| ChineseEditing | 1.08 | -43.97 | 1.31 | -42.53 |
| Console | 0.95 | -36.47 | 1.42 | -34.97 |
| Desktop | 1.30 | -49.15 | 1.30 | -47.70 |
| FlyingGraphics | 1.14 | -30.59 | 0.83 | -23.48 |
| Map | 1.33 | -32.45 | 1.19 | -38.78 |
| Programming | 1.26 | -40.75 | 0.67 | -40.41 |
| SlideShow | 2.43 | -53.07 | 2.68 | -52.93 |
| WebBrowsing | 1.20 | -48.36 | 1.01 | -50.78 |
| BasketballScreen | 1.11 | -43.34 | 1.23 | -46.30 |
| MissionControlClip2 | 1.52 | -47.71 | 1.42 | -48.54 |
| MissionControlClip3 | 0.87 | -42.98 | 0.95 | -43.16 |
| Robot | 1.29 | -48.71 | 0.86 | -57.90 |
| ChinaSpeed | | | 1.94 | -40.55 |
| EBURainFruits | 0.22 | -58.85 | | |
| Kimono1 | 0.12 | -74.47 | | |
| **Average (*TGM*+*M*)** | 1.29 | -42.62 | 1.27 | -42.69 |
| **Average (*A*+*CC*)** | 0.54 | -60.68 | 1.40 | -49.23 |
| **Average (*ALL*)** | 1.13 | -46.49 | 1.29 | -43.69 |

### 5.4.3 Performance of Individual DeepSCC-I and DeepSCC-II

The proposed overall DeepSCC utilizes DeepSCC-I and DeepSCC-II to make separate predictions for dynamic CTUs and stationary CTUs. To show the advantage of this arrangement, two sets of experiments were performed by only enabling DeepSCC-I and DeepSCC-II for all CTUs, respectively. The results are shown in Table 5.16. When applying DeepSCC-II to all CTUs, a very high increase in BDBR of 3.96% is brought since the mode correlation between the current CTU and the collected CTU is not guaranteed. Although some sequences contain very high percentages of stationary CTUs, they still suffer from very high increases of BDBR. For example, "ChineseEditing" contains 93.41% stationary CTUs, and it shows 6.11% increase in BDBR by implementing the individual DeepSCC-II for all CTUs. When applying DeepSCC-I to all CTUs, it provides 41.86% encoding time reduction with 1.03% increase in BDBR. It proves that DeepSCC-I can address both dynamic CTUs and stationary CTUs by only

Table 5.16: Performance of the individual DeepSCC-I and DeepSCC-II.

| Sequences | DeepSCC-I | | DeepSCC-II | | Proposed Overall DeepSCC | |
|---|---|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| ChineseEditing | 0.69 | -35.49 | 6.11 | -43.62 | 1.07 | -48.80 |
| Console | 0.83 | -33.95 | 4.92 | -26.79 | 1.06 | -41.85 |
| Desktop | 0.64 | -42.75 | 4.08 | -44.78 | 1.00 | -53.46 |
| FlyingGraphics | 0.98 | -30.48 | 6.58 | -0.78 | 0.99 | -30.76 |
| Map | 1.62 | -25.32 | 4.10 | -33.52 | 1.79 | -36.36 |
| Programming | 0.76 | -33.65 | 5.40 | -26.75 | 0.87 | -42.74 |
| SlideShow | 3.19 | -51.07 | 8.73 | -40.50 | 2.78 | -55.36 |
| WebBrowsing | 0.49 | -42.58 | 7.13 | -51.63 | 0.88 | -54.09 |
| BasketballScreen | 0.88 | -37.05 | 1.00 | -38.29 | 1.27 | -46.78 |
| MissionControlClip2 | 1.36 | -40.62 | 3.44 | -39.25 | 1.56 | -51.16 |
| MissionControlClip3 | 0.66 | -36.94 | 2.20 | -32.25 | 1.01 | -45.96 |
| Robot | 1.81 | -49.78 | 1.19 | 0 | 1.81 | -49.43 |
| ChinaSpeed | 0.29 | -55.67 | 0.37 | 0 | 0.29 | -55.94 |
| EBURainFruits | 0.17 | -70.69 | 0.21 | 0 | 0.17 | -70.69 |
| Kimono1 | 0.69 | -35.49 | 6.11 | -43.62 | 1.07 | -48.80 |
| **Average (*TGM*+*M*)** | 1.10 | -37.27 | 4.88 | -34.38 | 1.30 | -46.12 |
| **Average (*A*+*CC*)** | 0.76 | -58.71 | 0.59 | 0 | 0.76 | -58.69 |
| **Average (*ALL*)** | 1.03 | -41.86 | 3.96 | -27.01 | 1.18 | -48.81 |

take the luminance samples as the input. However, it shows less encoding time reduction compared with the overall DeepSCC, especially for sequences with many stationary CTUs. For example, the proposed overall DeepSCC shows 13.91% larger encoding time reduction for “ChineseEditing” than the individual DeepSCC-I. Therefore, the proposed overall DeepSCC which integrates DeepSCC-I and DeepSCC-II together helps to improve coding performance.

## 5.5 Chapter Summary

In this chapter, a deep learning based fast prediction network DeepSCC is proposed to reduce the computational complexity of SCC. To avoid the exhaustive mode search in a CTU, DeepSCC outputs 85 labels for 85 CUs of the CTU in a single test. For dynamic CTUs, DeepSCC-I is designed to take the luminance samples of a CTU as the input. For stationary CTUs, DeepSCC-II additionally utilizes the optimal mode maps of the collocated CTUs for further performance improvement. Compared with the traditional fast SCC prediction algorithms heavily relying on the limited number of hand-crafted features or heuristic rules, the proposed DeepSCC automatically learns useful features from the input. With extensive trainable parameters, DeepSCC is able to make direct mode decision for Intra, IBC, and PLT rather than the simple CU type classification. Experimental results show that the proposed DeepSCC provides an average computational complexity reduction of 48.81% with a negligible increase in BDBR of 1.18%, and the computational overhead of DeepSCC is less than 4% of the total encoding time.

# Chapter 6 Determinations on Coding Structure for HEVC-SCC Transcoding

## 6.1 Introduction

HEVC has dominated the market for many years and it leaves many legacy screen content videos encoded by HEVC. Therefore, this chapter presents a fast HEVC to SCC transcoder FHST that migrate the legacy screen content videos from HEVC to SCC to improve the coding efficiency. FHST analyzes various features from 4 categories to early terminate CU partitions and makes early mode decision. They are the features from the HEVC decoder, static features, dynamic features, and spatial features. First, the CU depth level collected from the HEVC decoder is utilized to early terminate the CU partition in SCC. Second, a flexible encoding structure is proposed to make early mode decisions with the help of various features. In this chapter, we start by presenting the data available from HEVC decoder. We then proceed to present our novel fast transcoder FHST. Next, the experimental results of the proposed algorithm are provided. Finally, the conclusion is given for this chapter.

Parts of the contents of this chapter are extracted from our published work [97]:

- **Wei Kuang**, Yui-Lam Chan, Sik-Ho Tsang, and Wan-Chi Siu, “Fast HEVC to SCC transcoder by early CU partitioning termination and decision tree based flexible mode decision for intra-frame coding,” IEEE Access, vol. 7, pp. 8773–8788, Jan. 2019.

## 6.2 Data Available from HEVC Decoder

When a HEVC encoder encodes a CU at the depth level of $d$, where $d \in \{0,1,2,3\}$, it calculates the residual block $Res$ between the predicted CU, $CU_{Pred}$, and the original CU, $CU_{Orig}$,

$$Res = CU_{Orig} - CU_{Pred} \tag{6.1}$$

After transformation and quantization, the quantized transform coefficients $C$ are obtained

$$C = (DResD^T) \otimes S_f \oslash Q \tag{6.2}$$

where $D$ is the transformation matrix, $S_f$ is the forward scaling matrix, $Q$ is the quantization matrix, $\otimes$ is the element wise multiplication operator, and $\oslash$ is the element-wise division operator. Finally, the HEVC encoder signals the quantized transform coefficients $C$ to represent the CU.

In the HEVC decoder side, the quantized transform coefficients $C$ are obtained for each CU by decoding the HEVC bitstream. After the corresponding inverse transformation and dequantization processes, the reconstructed residual block $Res'$ is obtained as

$$Res' = D^T(C \otimes \mathrm{Q} \otimes S_i)D \tag{6.3}$$

where $S_i$ is the inverse scaling matrix. Finally, a reconstructed CU, $CU_{Reco}$ , is represented as

$$CU_{Reco} = Res' + CU_{Pred} \tag{6.4}$$

To simplify the re-encoding process of SCC, the data from the HEVC decoder side can be utilized, such as the optimal depth level $d$, transform coefficients $C$ and the reconstructed residual block $Res'$.

## 6.3 Proposed FHST

To meet the challenge of computation-constrained applications, it is desired that the features from both the HEVC decoder and the SCC encoder are collected to simplify the re-encoding process. First, the features are utilized to early terminate the CTU partitions. Second, the features are also used to skip unnecessary mode candidates in a CU.

### 6.3.1 Early CU Partitioning Termination

Although HEVC and SCC share the same CTU partitioning structure, the optimal CU size decided by HEVC may change during the transcoding process, and an example is shown in Figure 6.1. It is observed that due to the adoption of the new coding modes, SCC allows inhomogeneous content to select larger CU sizes than HEVC. However, we also notice that the most optimal CUs decided by HEVC do not continue partitioning in SCC. Therefore, the CU partitioning process in SCC can be early terminated by utilizing the decoder side information of HEVC.

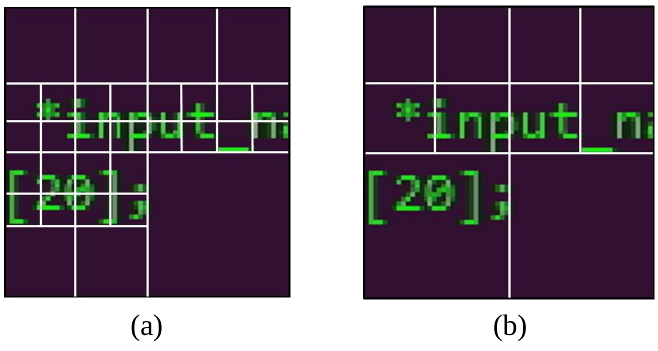


Figure 6.1: The partitioning structure of a CTU encoded by (a) HEVC and (b) SCC.

To derive the early CU partitioning termination rule, we encoded and decoded 13 typical SCC test sequences [30] by HEVC reference software HM-16.12 [98] with QPs of 22, 27, 32, and 37 under AI configuration, and then the decoded sequences were re-encoded by SCC reference software SCM-8.3 with the same QPs. Table 6.1 shows the average

Table 6.1: Performance of the proposed DeepSCC for training sequences.

| $d_{HEVC}$ | Partitioned to $d_{SCC}$ = 1 (%) | Partitioned to $d_{SCC}$ = 2 (%) | Partitioned to $d_{SCC}$ = 3 (%) |
|---|---|---|---|
| 0 | 11.02 | 0.91 | 0.23 |
| 1 | | 4.83 | 0.25 |
| 2 | | | 3.25 |

percentages of the further partitioned CUs from HEVC to SCC, where $d_{HEVC}$ and $d_{SCC}$ represent the CU depth level in HEVC and SCC, respectively. It is observed that for CUs with $d_{HEVC}$ of 1 or 2, they rarely continue partitioning in SCC. However, for CUs with $d_{HEVC}$ of 0, 11.02% of them are partitioned to $d_{SCC}$ of 1. Based on this observation, we set the early CU partitioning termination rule for different CU sizes adaptively by limiting the maximum CU depth level $d_{SCC}^{max}$ allowed to be checked as

$$d_{SCC}^{max} = \begin{cases} 1, & if\ d_{HEVC} = 0 \\ d_{HEVC}, & otherwise \end{cases}. \tag{6.5}$$

Therefore, for CUs with $d_{HEVC}$ of 1 or 2, further partitions are not allowed in SCC. For CUs with $d_{HEVC}$ of 0, FHST only allows them to be partitioned to $d_{SCC}$ of 1, and then further partitions are terminated.

### 6.3.2 Flexible Mode Decision

We propose a flexible mode decision structure in our algorithm, where the decision of each mode is considered separately. The objective of the flexible mode decision technique is to design a decision model for each mode, so that it can assist the transcoder in making the decision of checking a mode or, on the contrary, skipping a mode. Therefore, in the re-encoding process, two classes are defined for a mode *x*, where $x \in$ {Intra, IBCM&S, IBCSearch, PLT}, i.e., checking the mode ($\omega_x$) and skipping the mode ($\overline{\omega_x}$). By collecting features from both the HEVC decoder and the SCC encoder, the objective can be solved as a supervised classification task, and the model *G* is represented as

$$G(f_1, f_2, f_3, \dots, f_n) \rightarrow \{\omega_x, \overline{\omega_x}\} \tag{6.6}$$

where $f_i$ represents the features used to generate the model and $i=1,\dots,n$. Therefore, the decision of each mode is made adaptively by inserting a model $G$ before checking a mode.

## A. Algorithm Description

We implement a DT-based mode decision model right before checking a mode. Therefore, some dynamic features showing the intermediate coding information, such as the RD cost and IBC mode flag, can be employed to make mode decisions. Figure 6.2 shows the DT-based *x* mode decision model of our proposed algorithm, where $x \in \{$Intra, IBCM&S, IBCSearch, PLT$\}$. It can be seen from Figure 6.2 that the *x* mode decision model contains two parts, which are the *x* mode classifier and the Spatial-Info classifier. The *x* mode classifier utilizes the features from the current CU to make the decision of a mode *x*, including features from both the HEVC decoder side and the SCC encoder side, and the class label is set to $\omega_x$ if the outcome of the *x* mode classifier is 1. Besides, we

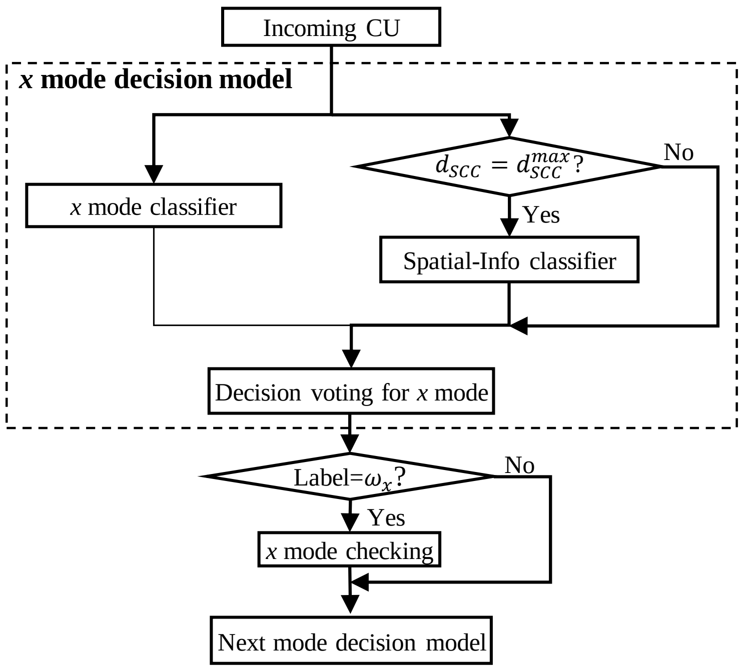


Figure 6.2: DT-based x mode decision model.

find that the mode decision accuracy for CUs arrived at their last depth levels, i.e., $d_{SCC}^{max}$, is very important to reduce the RD performance loss. The reason is that if the optimal mode is skipped for a large CU, it can still find relatively good modes when it continues partitioning and arrives at its last depth level. To achieve a good trade-off between the computational complexity and coding efficiency, we additionally train a set of Spatial-Info classifiers for CUs arrived at their last depth levels. We define CUs coded by Intra mode as NIBs and CUs coded by IBC or PLT mode as SCBs. The Spatial-Info classifier is trained by utilizing the spatial features to decide whether the current CU is a NIB or a SCB. If it is a SCB, the class label is set to $\omega_x$ for $x$∈{IBCM&S, IBCSearch, PLT}. Otherwise, the class label is set to $\omega_x$ for $x$∈{Intra}. After going through the classifiers, a decision voting strategy is adopted for making decisions. The label of the *x* mode decision model is set to $\omega_x$ if at least one classifier outputs $\omega_x$. More discussions on the structure of the *x* mode decision model are provided in Section 6.4.3.

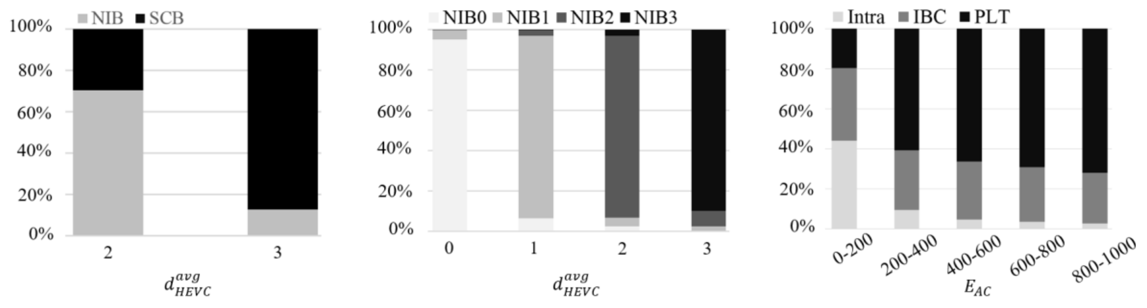
### B. Feature Selection

***<u>Features from the HEVC decoder:</u>***

$f_1$: The average CU depth level of HEVC $d_{HEVC}^{avg}$, which is represented as

$$d_{HEVC}^{avg} = \frac{\sum_{d_{HEVC}} Area(d_{HEVC}) \times d_{HEVC}}{2\text{N}\times2\text{N}} \tag{6.7}$$

where $Area(d_{HEVC})$ represents the area with $d_{HEVC}$ in a CU, and 2N×2N represents the size of the CU. In HEVC, screen content is encoded by small CUs due to their inhomogeneity. Therefore, CUs with larger values of $d_{HEVC}^{avg}$ are more likely to be SCBs. To verify this claim, 10000 16×16 NIBs and 10000 16×16 SCBs were randomly selected from the training set, and the distributions of NIBs and SCBs over $d_{HEVC}^{avg}$ are given in Figure 6.3(a). Since CUs with $d_{HEVC}^{avg}$ of 0 and 1 are usually encoded as 64×64 or 32×32 CUs in SCC, Figure 6.3(a) only shows the 16×16 CU type distributions over $d_{HEVC}^{avg}$ of 2



(a) (b) (c)

Figure 6.3: (a) NIB and SCB distribution over $d_{HEVC}^{avg}$ for 16×16 CU size, (b) Optimal CU depth level distribution of NIBs over $d_{HEVC}^{avg}$, and (c) Mode distribution over $E_{AC}$.

and 3. It is observed that many CUs with $d_{HEVC}^{avg}$ of 2 are NIBs while most CUs with $d_{HEVC}^{avg}$ of 3 are SCBs. Besides, the CU depth level distribution of NIBs in SCC over $d_{HEVC}^{avg}$ is investigated by randomly selecting 10000 NIBs at each depth level, and the results are shown in Figure 6.3(b), where NIB0, NIB1, NIB2 and NIB3 denote the NIBs encoded at the depth levels of 0, 1, 2, and 3 in SCC, respectively. It is observed that that NIBs from HEVC are very likely to be encoded at the same depth levels in SCC. Therefore, Intra mode at other depth levels is very likely to be skipped for NIBs.

$f_2$: The AC coefficient energy $E_{AC}$ of the quantized transform coefficients $C$, which is defined as the sum of square of the AC coefficients

$$E_{AC} = \sum_{c_{i,j} \in C,\, c_{i,j} \neq c_{0,0}} c_{i,j}^2 \tag{6.8}$$

where $c_{i,j}$ is a quantized transform coefficient with the row index of $i$ and column index of $j$ in $C$, and $c_{0,0}$ is the DC coefficient. SCBs have many high frequency components and they contain higher values of $E_{AC}$ than NIBs. Besides, we also notice that while many IBC coded CUs are smooth, PLT coded CUs are usually more complex, and they have even higher values of $E_{AC}$ than IBC coded CUs. Statistics that support this claim are shown in Figure 6.3(c), where 10000 Intra, 10000 IBC and 10000 PLT coded 16×16 CUs were randomly selected from the training set. Therefore, $E_{AC}$ provides a chance to differentiate PLT mode from IBC mode.

$f_3$: The number of zeros in the residual block $N_{ZR}$, which is also adopted in the fast transcoding algorithm [72], and it is defined as

$$N_{ZR} = \sum_{r_{i,j} \in Res'} \delta(r_{i,j}, 0) \tag{6.9}$$

where $r_{i,j}$ is an element with row index of $i$ and column index of $j$ in the reconstructed residual block $Res'$, and Kronecker delta $\delta\left(r_{i,j}, 0\right)$ is represented as

$$\delta(a, b) = \begin{cases} 0, \text{ if } a \neq b \\ 1, \text{ if } a = b \end{cases} \tag{6.10}$$

Since SCBs have many uniform background pixels, they tend to have a larger value of $N_{ZR}$.

***<u>Static features:</u>***

$f_4$–$f_7$: High gradient pixel number $HGN_0$, $HGN_1$, $HGN_2$, $HGN_3$. The high gradient pixel is utilized to detect sharp edges in a CU, as in Equation (3.1). To detect edges with different sharpness in the proposed algorithm, 4 different high gradient pixel numbers, $HGN_0$, $HGN_1$, $HGN_2$, and $HGN_3$, are calculated by counting high gradient pixels with $TH_S$ of 8, 16, 32 and 64, respectively. As analyzed in Chapter 4, SCBs have a larger high gradient pixel number because they have many sharp edges. Besides, because PLT coded CUs are usually more complex, they also have a larger high gradient pixel number than IBC coded CUs.

$f_8$: Distinct color number $N_{DC}$, which is also used in fast SCC encoding algorithms [54], [55] and the fast transcoding algorithm [72]. It is calculated by counting the pixels with different luminance values. Since SCBs contain limited colors, they usually have a smaller value of $N_{DC}$ than NIBs.

***<u>Dynamic features:</u>***

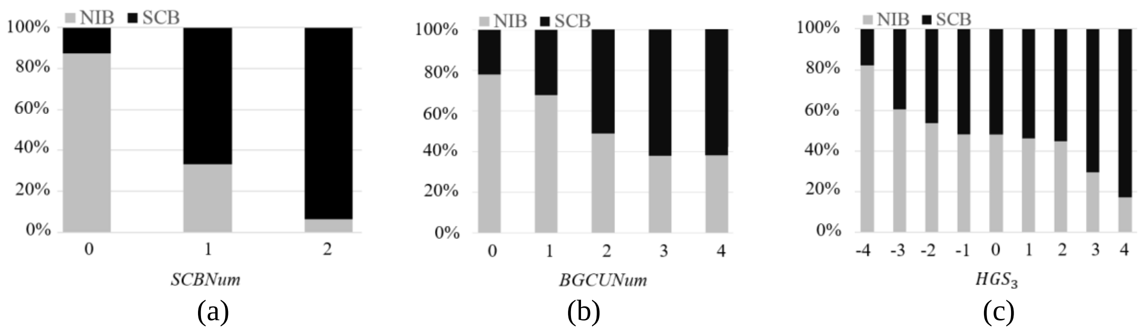


Figure 6.4: NIB and SCB distribution over (a) SCBNum, (b) BGCUNum and (c) $HGS_3$.

$f_9$– $f_{10}$: The RD cost and IBC flag of the best mode, $J_{mode}$ and $Flag_{IBC}$, respectively, before checking the target mode in the current CU. As analyzed in Chapter 4, they reflect the intermediate coding information of a CU.

***<u>Spatial features:</u>***

$f_{11}$: The neighboring SCB number, *SCBNum*, by counting the top and left neighboring CUs. *SCBNum* is denoted by

$$SCBNum = \delta(m_L, IBC) + \delta(m_L, PLT) + \delta(m_T, IBC) + \delta(m_T, PLT) \tag{6.11}$$

where $m_L$ and $m_T$ represent the optimal modes of the left and top CUs of the current CU, respectively. The current CU is more likely to be a SCB if it has SCB neighbors, and the evidence is shown in Figure 6.4(a), where 10000 16×16 NIBs and 10000 16×16 SCBs were randomly selected from the training set.

$f_{12}$: The same background sub-CUs number, *BGCUNum*, by counting its four sub-CUs which have the same background color as the current CU. *BGCUNum* is defined as

$$BGCUNum = \sum_{i=0}^{3} \delta(Y_{BC,d}, Y_{BC,d+1}^{i}) \tag{6.12}$$

where $Y_{BC,d}$ and $Y_{BC,d+1}^{i}$are the background colors of the current CU at the depth level of *d* and its *i-th* sub-CU at the depth level of *d* + 1, respectively. We define the background color in a CU/sub-CU as the luminance value with the highest occurrence frequency within the CU/sub-CU. If more sub-CUs have the same background color as the current CU, it is more likely to be a SCB. To support our claim, 10000 16×16 NIBs and 10000 16×16 SCBs were randomly selected, and their distributions over *BGCUNum* are shown in Figure 6.4(b).

$f_{13}$– $f_{16}$: The high gradient pixel strength, $HGS_0$, $HGS_1$, $HGS_2$ and $HGS_3$, which are calculated by considering if the neighboring CUs from the left, right, top and bottom contain high gradient pixels with $TH_S$ of 8, 16, 32 and 64, respectively. Let us call a CU that contains high gradient pixels as a high gradient CU (HGCU). It is observed that if the current CU is a HGCU, the more neighboring HGCUs it has, the more likely it is a SCB. Otherwise, if the current CU is a non-HGCU, the more neighboring non-HGCUs it has, the more likely it is a NIB. We set the initial value of $HGS_i$ ($i \in\{0,1,2,3\}$) to 0. If the current CU is a HGCU, $HGS_i$ is set to a non-negative value, and its absolute value is calculated by counting the number of HGCUs from the left, right, top and bottom neighboring CUs. Otherwise, if the current CU is a Non-HGCU, $HGS_i$ is set to a non-positive value, and its absolute value is calculated by counting the number of Non-HGCUs from the left, right, top and bottom neighboring CUs. Therefore, as the value of $HGS_i$ goes from small to large, the probability of the current CU being a SCB is increased, and statistics that give the evidence in this observation is shown in Figure 6.4(c), where 10000 16×16 NIBs and 10000 16×16 SCBs were randomly selected.

### C. Training Implementation

As in Chapter 4, we select DTs as the classification model. In our case that decides whether a mode is checked or not, a CU with several features is input to the DT. Each non-leaf node runs a test on a feature, and each branch denotes an outcome of the test. After going through a series of tests, the CU comes to a leaf node, and a class label of $\omega_x$ or $\overline{\omega_x}$ is assigned to it. Specifically, the class label is decided as the label of majority training samples in a leaf node, and the decision accuracy of a leaf node is denoted by the percentage of correctly classified samples in it. The training data for building DTs are selected from 5 sequences, which are “Console”, “Desktop”, “Map”, “MissionControlClip2”, and “Robot”. To generate the training data, 10 frames were extracted from each sequence with an equal time interval. Those frames were firstly encoded by HM-16.12 with QPs of 22, 27, 32 and 37, and then the decoded frames were re-encoded by SCM-8.3 with the same QPs. When training the $x$ mode classifier, the positive data come from CUs encoded by $x$ mode, and the negative data are from CUs encoded by other modes. Therefore, the class label of $x$ mode is $\omega_x$ if the outcome is 1. Otherwise, the class label is $\overline{\omega_x}$. To train the Spatial-Info classifier, the positive data are collected from SCBs while the negative data are collected from NIBs. Therefore, for SCBs, i.e., $x$∈{IBCM&S, IBCSearch, PLT}, the class label is set to $\omega_x$ if the outcome is 1. For NIB, i.e., $x$∈{Intra}, the class label is set to $\omega_x$ if the outcome is 0. To avoid the data imbalance problem [81] caused by more training samples in one class than the other, we set the numbers of positive and negative training data to be equal. To balance the coding efficiency and computational complexity, we set two confidence thresholds $\alpha$ and $\beta$ in the Spatial-Info classifier and $x$ mode classifier, respectively. If the accuracy of a decision made by the Spatial-Info classifier or $x$ mode classifier is lower than the value of $\alpha$ or $\beta$, the class label for mode $x$ is set to $\omega_x$ regardless of its outcome. As SCC inherits the CTU hierarchical partitioning structure from HEVC, which supports 4 different CU sizes from 8×8 to 64×64, the classifiers for each mode were trained for CUs with different sizes, respectively.

As a summary, the flowchart of our proposed algorithm is shown in Figure 6.5, where the DT-based $x$ mode decision model is shown in Figure 6.2. A CU goes through a mode decision model before checking a mode. If the label given by the decision model is $\omega_x$, the mode $x$ would be checked. Otherwise, it would be skipped. Besides, when the

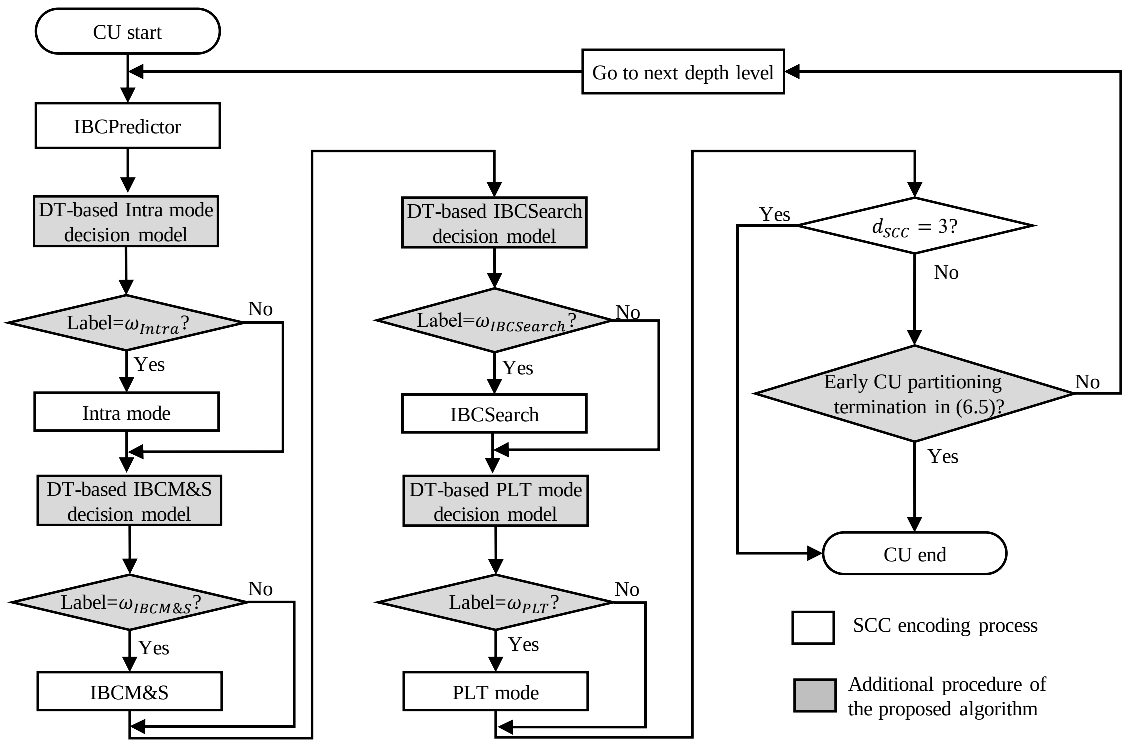

CU start
Go to next depth level
IBCPredictor
DT-based Intra mode decision model
Label=$\omega_{Intra}$?
Yes
No
Intra mode
DT-based IBCM&S decision model
Label=$\omega_{IBCM\&S}$?
Yes
No
IBCM&S
DT-based IBCSearch decision model
Label=$\omega_{IBCSearch}$?
Yes
No
IBCSearch
DT-based PLT mode decision model
Label=$\omega_{PLT}$?
Yes
No
PLT mode
$d_{SCC}$ = 3?
Yes
No
Early CU partitioning termination in (6.5)?
Yes
No
CU end
SCC encoding process
Additional procedure of the proposed algorithm

Figure 6.5: Flowchart of the proposed FHST.

CU partitioning termination rule described in Equation (6.5) is satisfied, the encoding process of this CU is finished.

## 6.4 Experimental Results and Discussions

Since there are many legacy screen content videos encoded by HEVC in YUV4:2:0 format, we conducted various experiments to evaluate the transcoding performance of YUV4:2:0 screen content videos. We implemented our proposed FHST in HM-16.12 and HM-16.12+SCM-8.3, and all experiments were conducted with QPs of 22, 27, 32 and 37 under AI configuration and CTC [30]. All test sequences were firstly encoded by HM-16.12 with QPs of 22, 27, 32 and 37, and then the decoded frames were re-encoded by the proposed FHST with the same QPs. The coding efficiency and re-encoding time of the proposed FHST were compared with the conventional brute-force transcoder CBFT, and they are measured by BDBR and re-encoding time increase, ΔTime, in percentage

(%). It is noted that BDBR is calculated by comparing the HEVC decoded video and the final transcoded video.

### 6.4.1 Confidence Threshold Determination

To achieve a good trade-off between the coding efficiency and computational complexity, two confidence thresholds $\alpha$ and $\beta$ were set in the Spatial-Info classifier and *x* mode classifier, respectively. If the accuracy of a decision made by the Spatial-Info classifier or *x* mode classifier is smaller than the value of $\alpha$ or $\beta$, the class label for mode *x* is set to $\omega_x$ regardless of its outcome. In this sub-section, the performance of our proposed FHST is investigated by adopting a greedy searching strategy. The default values of $\alpha$ and $\beta$ are always 0.5 in a DT. First, the value of $\alpha$ was fine-tuned with $\beta$ = 0.5. Second, the value of $\beta$ was fine-tuned with $\alpha$ set to the value providing the best performance. The performances with different values of $\alpha$ and $\beta$ are shown in Table 6.2. It is observed that FHST with the default values of confidence thresholds ($\alpha$ = 0.5, $\beta$ = 0.5) provides 53.19% re-encoding time reduction with BDBR increased by 1.89%. By adjusting the values of $\alpha$

Table 6.2: Performance of the proposed FHST for YUV 4:2:0 sequences with different threshold values.

| Sequences | Tuning of $\alpha$ ($\beta$=0.5) | | | | | | Tuning of $\beta$ ($\alpha$=0.75) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\alpha$=0.50 | | $\alpha$=0.75 | | $\alpha$=0.95 | | $\beta$=0.55 | | $\beta$=0.60 | | $\beta$=0.65 | |
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 3.39 | -40.86 | 2.89 | -39.57 | 2.51 | -33.33 | 2.83 | -39.26 | 2.82 | -39.01 | 2.57 | -33.54 |
| Desktop (*T*) | 1.18 | -40.67 | 1.15 | -40.38 | 0.99 | -32.14 | 1.12 | -40.27 | 1.12 | -39.21 | 0.55 | -33.54 |
| Map (*T*) | 1.29 | -59.36 | 0.11 | -57.02 | 0.17 | -38.11 | 0.16 | -54.92 | 0.16 | -54.81 | 0.03 | -54.58 |
| MissionControlClip2 (*T*) | 1.96 | -55.88 | 1.11 | -53.46 | 0.88 | -43.40 | 1.15 | -52.72 | 0.99 | -52.17 | 0.89 | -49.10 |
| Robot (*T*) | 1.94 | -78.58 | 2.00 | -74.87 | 2.01 | -53.17 | 2.04 | -72.31 | 2.05 | -72.22 | 2.09 | -73.85 |
| BasketballScreen (*NT*) | 2.06 | -55.17 | 1.15 | -52.95 | 0.77 | -39.80 | 1.12 | -51.48 | 1.02 | -51.07 | 0.80 | -48.44 |
| ChineseEditing (*NT*) | 1.11 | -37.68 | 0.80 | -36.16 | 0.57 | -29.01 | 0.77 | -35.68 | 0.75 | -35.05 | 0.45 | -29.50 |
| ChinaSpeed (*NT*) | 1.36 | -63.28 | 0.99 | -60.02 | 0.86 | -46.05 | 0.97 | -58.55 | 0.97 | -58.54 | 0.93 | -57.19 |
| FlyingGraphics (*NT*) | 2.74 | -41.28 | 2.08 | -40.13 | 1.28 | -31.24 | 1.95 | -39.18 | 1.81 | -38.52 | 1.49 | -32.24 |
| MissionControlClip3 (*NT*) | 2.33 | -50.53 | 1.70 | -48.81 | 1.42 | -37.29 | 1.69 | -47.97 | 1.60 | -46.97 | 1.31 | -43.07 |
| Programming (*NT*) | 1.70 | -47.30 | 0.88 | -45.88 | 0.60 | -37.50 | 0.87 | -45.04 | 0.81 | -44.62 | 0.67 | -40.39 |
| SlideShow (*NT*) | 1.50 | -71.22 | 0.71 | -68.71 | 0.70 | -59.47 | 0.66 | -67.00 | 0.60 | -66.92 | 0.63 | -66.16 |
| WebBrowsing (*NT*) | 1.92 | -49.59 | 1.62 | -48.15 | 1.42 | -39.32 | 1.87 | -47.49 | 1.74 | -47.02 | 0.98 | -42.32 |
| **Average (*T*)** | 1.95 | -55.07 | 1.45 | -53.06 | 1.31 | -40.03 | 1.46 | -51.90 | 1.43 | -51.48 | 1.23 | -48.92 |
| **Average (*NT*)** | 1.84 | -52.00 | 1.24 | -50.10 | 0.95 | -39.96 | 1.24 | -49.05 | 1.16 | -48.59 | 0.91 | -44.91 |
| **Average (*ALL*)** | 1.89 | -53.19 | 1.32 | -51.24 | 1.09 | -39.99 | 1.32 | -50.14 | 1.26 | -49.70 | 1.03 | -46.46 |

*T*: Sequence used for training. *NT*: Sequence not used for training.

and $\beta$ ($\alpha$ = 0.75, $\beta$ = 0.65), BDBR increment is reduced to 1.03%, and the re-encoding time is reduced by 46.46%.

Besides, it is also observed from Table 6.2 that the proposed FHST provides similar performance for sequences used for training (*T*) and sequences not used for training (*NT*). For example, with $\alpha$ = 0.75 and $\beta$ = 0.5, the average re-encoding time reductions of *T* and *NT* sequences are 53.06 % and 50.10%, while the BDBR of *T* and *NT* sequences are increased by 1.45% and 1.24%, respectively. It proves that the training process of our proposed FHST does not run into overfitting, and it can be well applied to other screen content sequences. In the following sub-sections, we set $\alpha$ to 0.75 and $\beta$ to 0.5 for further discussions, which provides 51.24% encoding time reduction with 1.32% increase in BDBR on average.

### 6.4.2 Performance Comparison of YUV 4:2:0 Format

To evaluate the efficiency of FHST, we compared it with CBFT in which the original SCC encoder of CBFT in Figure 1.6 is replaced by the fast SCC encoding algorithms in [51], [54], [55]. It is noted that the work in [72], which is the only existing fast HEVC to SCC transcoding algorithm, is only worked for sequences in YUV 4:4:4 format, and the comparison will be shown later in Section 6.4.6. Table 6.3 shows the performance comparisons based on HM-16.12 and HM-16.12+SCM-8.3. Compared with the fast SCC encoding algorithms in [51], [54], [55], the proposed FHST additionally utilizes features from the HEVC decoder to improve prediction accuracy, and it is observed that the performance of our proposed FHST outperforms the fast SCC encoding algorithms [51], [54], [55]. On average, the proposed FHST provides 51.24% re-encoding time reduction with a negligible increase in BDBR of 1.32%. Comparatively, Zhang *et al.*'s algorithm [51], Lei *et al.*'s algorithm [54] and Yang *et al.*'s algorithm [55] all bring very high increase in

Table 6.3: Performance comparison of the proposed FHST with different fast SCC encoding algorithms for YUV 4:2:0 sequences.

| Sequences | CBFT + Zhang [51] | | CBFT + Lei [54] | | CBFT + Yang [55] | | Proposed FHST | |
|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 2.61 | -24.29 | 2.25 | -18.97 | 5.43 | -38.62 | 2.89 | -39.57 |
| Desktop (*T*) | 1.60 | -27.90 | 1.32 | -19.59 | 4.10 | -41.38 | 1.15 | -40.38 |
| Map (*T*) | 1.58 | -27.64 | 4.05 | -21.45 | 5.13 | -32.55 | 0.11 | -57.02 |
| MissionControlClip2 (*T*) | 2.19 | -28.03 | 2.71 | -28.18 | 2.69 | -40.41 | 1.11 | -53.46 |
| Robot (*T*) | 11.37 | -15.22 | 12.85 | -34.15 | 4.90 | -36.15 | 2.00 | -74.87 |
| BasketballScreen (*NT*) | 1.69 | -29.43 | 1.41 | -24.06 | 3.17 | -37.80 | 1.15 | -52.95 |
| ChineseEditing (*NT*) | 0.52 | -31.19 | 0.59 | -17.52 | 3.32 | -35.54 | 0.80 | -36.16 |
| ChinaSpeed (*NT*) | 0.82 | -22.00 | 0.47 | -27.31 | 1.32 | -41.14 | 0.99 | -60.02 |
| FlyingGraphics (*NT*) | 0.41 | -12.75 | 1.10 | -16.97 | 3.68 | -36.31 | 2.08 | -40.13 |
| MissionControlClip3 (*NT*) | 1.57 | -26.09 | 1.78 | -22.26 | 2.95 | -37.19 | 1.70 | -48.81 |
| Programming (*NT*) | 1.29 | -27.91 | 1.57 | -23.32 | 4.00 | -38.57 | 0.88 | -45.88 |
| SlideShow (*NT*) | 1.89 | -43.19 | 4.88 | -51.60 | 4.05 | -57.34 | 0.71 | -68.71 |
| WebBrowsing (*NT*) | 2.35 | -37.28 | 2.85 | -24.01 | 5.58 | -40.34 | 1.62 | -48.15 |
| **Average (*T*)** | 3.87 | -24.62 | 4.64 | -24.47 | 4.45 | -37.82 | 1.45 | -50.06 |
| **Average (*NT*)** | 1.31 | -28.73 | 1.83 | -25.88 | 3.50 | -40.53 | 1.24 | -52.10 |
| **Average (*ALL*)** | 2.30 | -27.15 | 2.91 | -25.34 | 3.87 | -39.49 | 1.32 | -51.24 |

BDBR, where 27.15%, 25.34% and 39.49% re-encoding time is saved with 2.30%, 2.91% and 3.87% increase in BDBR, respectively. The fast SCC encoding algorithms [51], [54], [55] only utilize features from the SCC encoder, and they heavily rely on the assumption that the computer-generated content is noiseless. However, this assumption does not hold for decoded videos due to the lossy encoding and decoding of HEVC. Therefore, the fast SCC encoding algorithms [51], [54], [55] provide less improvement.

### 6.4.3 Discussion on the Structure of *x* Mode Decision Model

Apart from the *x* mode decision model in Figure 6.2, there are other possible structures, such as training a single DT by utilizing all features (FHST2) and applying the Spatial-Info classifier to all depth levels (FHST3). For FHST2, all features in Section 6.3.2 are used to train the *x* mode classifier without the Spatial-Info classifier. For FHST3, the condition for checking the Spatial-Info classifier in Figure 6.2 is removed. However, it is found that they fail to achieve a good tread-off between the re-encoding time and RD performance. The performances of FHST2 and FHST3 are shown in Table 6.4 with the

Table 6.4: Performance of the proposed algorithm with other structures.

| Sequences | FHST2 | | FHST3 | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 8.90 | -50.22 | 3.00 | -25.73 |
| Desktop (*T*) | 4.95 | -46.04 | 0.50 | -24.01 |
| Map (*T*) | 5.14 | -71.20 | 1.38 | -47.84 |
| MissionControlClip2 (*T*) | 6.62 | -61.22 | 1.68 | -44.98 |
| Robot (*T*) | 3.95 | -78.73 | 2.30 | -67.94 |
| BasketballScreen (*NT*) | 8.12 | -58.85 | 1.71 | -41.86 |
| ChineseEditing (*NT*) | 3.71 | -43.45 | 0.64 | -21.85 |
| ChinaSpeed (*NT*) | 6.00 | -68.85 | 0.83 | -52.27 |
| FlyingGraphics (*NT*) | 8.97 | -48.04 | 2.18 | -22.73 |
| MissionControlClip3 (*NT*) | 7.39 | -53.60 | 1.99 | -35.34 |
| Programming (*NT*) | 4.68 | -55.15 | 1.49 | -32.57 |
| SlideShow (*NT*) | 6.61 | -74.33 | 1.40 | -63.47 |
| WebBrowsing (*NT*) | 2.82 | -50.03 | 1.18 | -32.69 |
| **Average** | 5.99 | -58.43 | 1.56 | -39.48 |

default values of confidence thresholds ($\alpha$ = 0.5, $\beta$ = 0.5). It is observed that FHST2 brings a very high increase in BDBR of 5.99%. The reason is that FHST2 is strongly affected by error propagation due to the adoption of spatial features into the single DT. For example, the DT would directly skip IBC and PLT modes for the current CUs if the neighboring CUs are NIBs. Comparatively, FHST avoids the error propagation by treating the spatial features as additional features, and they are utilized to train another DT. On the other hand, FHST3 needs to check more mode candidates by applying the Spatial-Info classifiers to all depth levels. Therefore, it provides a limited encoding time reduction of 39.48%. Comparatively, the proposed FHST achieves a good trade-off between the encoding time and RD performance by applying the Spatial-Info classifier to the last depth level. With the default values of confidence thresholds, FHST provides 53.19% re-encoding time reduction with BDBR increased by 1.89%. By setting $\alpha$ to 0.75 and $\beta$ to 0.5, FHST has an even smaller increase of BDBR than FHST3, where 51.24% re-encoding time is reduced with BDBR increased by 1.32%. Therefore, we adopt it as the optimal structure to the *x* mode decision model.

### 6.4.4 Performance of the Individual Technique

In this sub-section, the performances of the early CU partitioning termination technique and the flexible mode decision technique are evaluated separately, and the results are shown in Table 6.5. It is observed that the proposed flexible mode decision technique achieves 45.11% re-encoding time reduction with BDBR increased by 1.28% on average. Besides, the early CU partitioning termination technique provides 17.31% re-encoding time reduction while BDBR is increased by 0.60% on average. More specifically, it provides the largest re-encoding time reduction of 46.65% for “SlideShow”. The reason is that “SlideShow” contains many smooth areas, which are encoded with many large CUs by HEVC. Therefore, with the help of the decoder side information of HEVC, many CU partitions in SCC are early terminated, and it leads to large re-encoding time reduction. Furthermore, to understand the re-encoding time reduction of our proposed FHST in low and high bit rate cases, we compared the re-encoding time reduction with different QPs. Figure 6.6 shows the results of 4 sequences including “BasketballScreen”, “ChinaSpeed”, “Desktop” and “WebBrowsing”, and similar results

Table 6.5: Performance of each proposed technique for YUV 4:2:0 sequences.

| Sequences | Early CU partitioning termination | | Flexible mode decision | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 1.82 | -9.06 | 2.32 | -37.20 |
| Desktop (*T*) | 0.25 | -5.95 | 1.09 | -39.03 |
| Map (*T*) | 0.35 | -14.03 | -0.09 | -52.90 |
| MissionControlClip2 (*T*) | 0.37 | -23.11 | 1.65 | -44.50 |
| Robot (*T*) | 2.55 | -34.82 | 1.20 | -62.00 |
| BasketballScreen (*NT*) | 0.15 | -16.11 | 1.62 | -48.28 |
| ChineseEditing (*NT*) | 0.13 | -6.21 | 0.87 | -34.52 |
| ChinaSpeed (*NT*) | 0.23 | -24.12 | 0.94 | -50.99 |
| FlyingGraphics (*NT*) | 0.31 | -5.65 | 2.08 | -38.93 |
| MissionControlClip3 (*NT*) | 0.83 | -13.26 | 1.65 | -44.29 |
| Programming (*NT*) | 0.18 | -14.25 | 0.89 | -41.27 |
| SlideShow (*NT*) | 0.30 | -46.65 | 0.73 | -47.42 |
| WebBrowsing (*NT*) | 0.35 | -11.79 | 1.74 | -45.13 |
| **Average** | 0.60 | -17.31 | 1.28 | -45.11 |

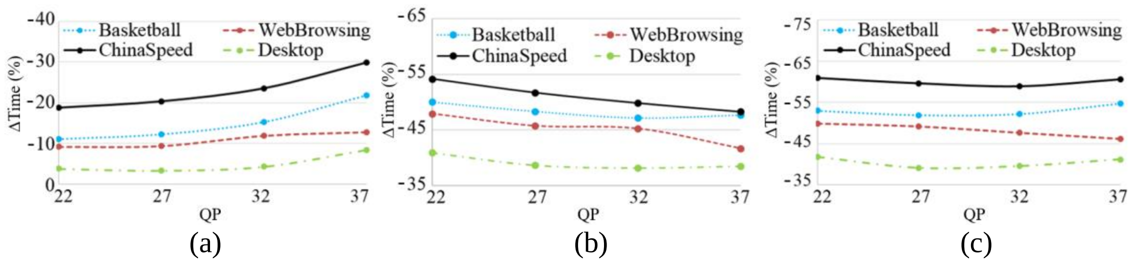


Figure 6.6: Re-encoding time reduction of (a) early CU partitioning termination technique, (b) flexible mode decision technique, and (c) the proposed overall algorithm over 4 QPs.

are observed for other sequences. The re-encoding time reductions provided by the early CU partitioning termination technique, flexible mode decision technique, and overall algorithm are shown in Figure 6.6(a), (b) and (c), respectively. It is observed in Figure 6.6(a) that the early CU partitioning termination technique provides more re-encoding time reduction as QP increases. The reason is that many CUs are encoded with large sizes by HEVC with a large value of QP, and more CUs are early terminated in SCC by using the early CU partitioning termination technique. On the contrary, the re-encoding time reduction provided by the flexible mode decision technique decreases as QP increases. The reason is that static features contain more noise as QP increases, which leads to a decrease of the decision accuracy. Although the re-encoding reduction provided by each sub-algorithm is different as QP changes, it is observed in Figure 6.6(c) that the re-encoding reduction of the overall algorithm varies little across different values of QP. Therefore, our proposed FHST has stable performance as the bit rate varies.

Another way to evaluate the proposed FHST is to investigate the hit rates of the proposed techniques compared with the CBFT transcoder. In this sub-section, the hit rates of the early CU partitioning termination and flexible mode decision techniques are given by calculating the percentages of the areas encoded by the same mode as in CBFT, and the results are shown in Table 6.6. It is observed that the average hit rates of the proposed early CU partitioning termination and flexible mode decision techniques are all above

Table 6.6: Hit rates of the proposed techniques for YUV 4:2:0 sequences.

| Sequences | Early CU partitioning termination (%) | | | | Flexible mode decision (%) | | | |
|---|---|---|---|---|---|---|---|---|
| | QP22 | QP27 | QP32 | QP37 | QP22 | QP27 | QP32 | QP37 |
| Console (*T*) | 95.78 | 93.67 | 92.72 | 96.30 | 91.27 | 90.43 | 87.97 | 88.81 |
| Desktop (*T*) | 98.03 | 98.25 | 98.40 | 97.78 | 92.90 | 91.71 | 88.37 | 85.96 |
| Map (*T*) | 98.65 | 98.62 | 98.10 | 98.02 | 94.40 | 95.00 | 94.79 | 94.45 |
| MissionControlClip2 (*T*) | 97.63 | 97.69 | 97.96 | 98.18 | 92.11 | 90.88 | 92.91 | 93.85 |
| Robot (*T*) | 97.46 | 97.32 | 97.64 | 97.84 | 95.89 | 94.74 | 94.64 | 93.71 |
| BasketballScreen (*NT*) | 97.49 | 98.06 | 98.08 | 98.00 | 92.90 | 91.54 | 90.38 | 90.51 |
| ChineseEditing (*NT*) | 98.09 | 98.34 | 98.50 | 98.54 | 91.12 | 90.81 | 90.91 | 90.50 |
| ChinaSpeed (*NT*) | 99.06 | 94.44 | 94.75 | 94.71 | 94.71 | 92.90 | 91.30 | 89.44 |
| FlyingGraphics (*NT*) | 98.50 | 98.52 | 98.28 | 97.20 | 92.16 | 90.77 | 89.03 | 88.87 |
| MissionControlClip3 (*NT*) | 98.05 | 98.14 | 98.27 | 98.19 | 91.92 | 91.97 | 91.73 | 91.39 |
| Programming (*NT*) | 97.50 | 97.83 | 98.15 | 97.84 | 93.58 | 92.84 | 91.79 | 91.12 |
| SlideShow (*NT*) | 98.90 | 98.91 | 99.06 | 99.04 | 95.55 | 95.81 | 95.98 | 96.23 |
| WebBrowsing (*NT*) | 98.03 | 97.78 | 98.50 | 98.46 | 92.81 | 92.07 | 88.12 | 87.81 |
| **Average (*T*)** | 97.51 | 97.11 | 96.96 | 97.62 | 93.31 | 92.55 | 91.74 | 91.36 |
| **Average (*NT*)** | 98.20 | 97.75 | 97.95 | 97.75 | 93.09 | 92.33 | 91.16 | 90.74 |
| **Average (*ALL*)** | 97.94 | 97.51 | 97.57 | 97.70 | 93.17 | 92.42 | 91.37 | 90.97 |

90%. More specifically, the hit rate of the early CU partitioning termination technique varies from 92.72% to 99.06%, while the hit rate of the flexible mode decision technique varies from 85.96% to 96.23% for different sequences with different QPs. Besides, the average hit rate of the early CU partitioning termination technique varies little under different QPs, while the average hit rate of the flexible mode decision technique is increased from 90.97% to 93.17% as QP gets smaller. It is due to the fact that the decoded HEVC videos contain less noise as QP gets smaller, and the static features utilized in our proposed mode decision models can describe the CU content characteristics more precisely.

Since the flexible mode decision technique contributes significantly to our proposed FHST, the mode decision of FHST is further studied. The decision of each mode is visualized, and it is compared with the mode decision made by CBFT. Figure 6.7 and Figure 6.8 show the mode decision of a region in “ChinaSpeed” and a region in “Programming”, respectively, at the depth level of 2 and QP of 22. Figure 6.7(a) and Figure 6.8(a) show the optimal mode decided by CBFT, where Intra, IBC and PLT modes are denoted by blue, purple and yellow blocks, respectively. It should be noted that for CUs without any denoted color, they are not encoded at the depth level of 2. Figure

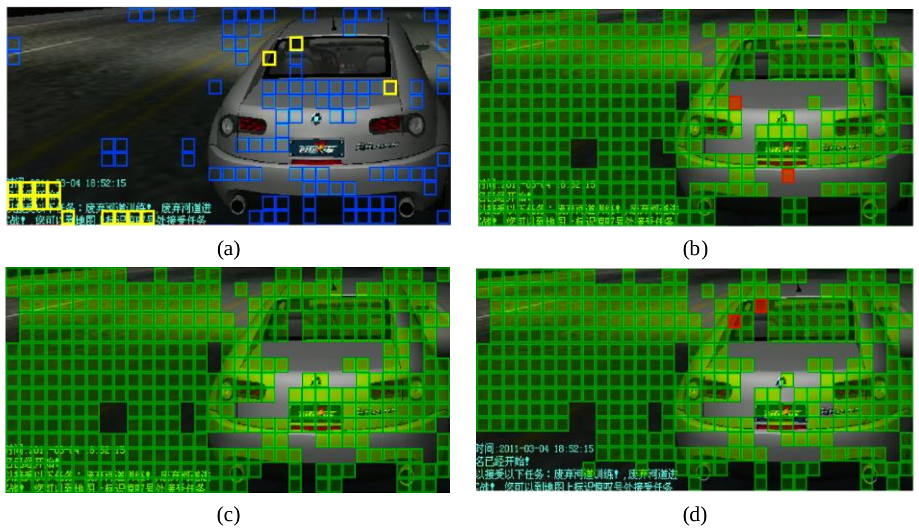


Figure 6.7: Mode decisions of a region in “ChinaSpeed” with the depth level of 2 and QP of 22. (a) Mode decision of CBFT, where Intra, IBC and PLT modes are denoted by blue, purple and yellow blocks. (b) Intra mode skipped CUs, (c) IBC mode skippped CUs, (d) PLT mode skipped CUs decided by our proposed FHST, where CUs with incorrectly and correctly skipped mode are denoted by red shaded blocks and green shaded blocks, respectively.

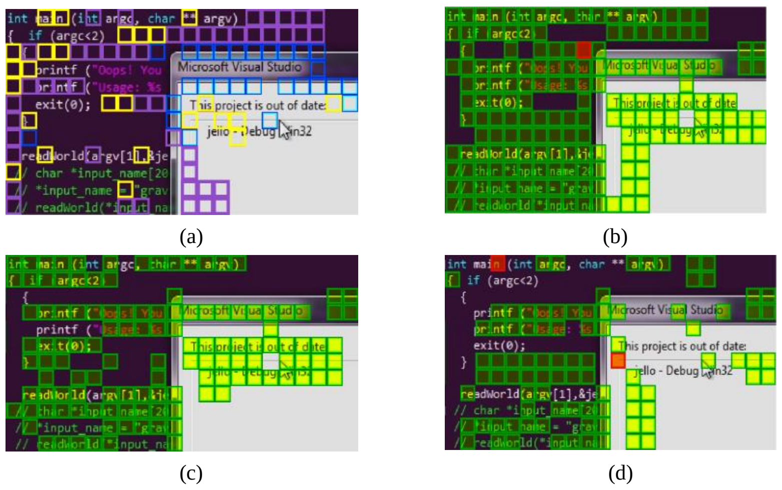


Figure 6.8: Mode decisions of a region in “Programming” with the depth level of 2 and QP of 22. (a) Mode decision of CBFT, where Intra, IBC and PLT modes are denoted by blue, purple and yellow blocks. (b) Intra mode skipped CUs, (c) IBC mode skippped CUs, (d) PLT mode skipped CUs decided by our proposed FHST, where CUs with incorrectly and correctly skipped mode are denoted by red shaded blocks and green shaded blocks, respectively.

6.7(b)–(d) and Figure 6.8(b)–(d) show the Intra mode skipped CUs, IBC mode skipped CUs, PLT mode skipped CUs decided by our proposed flexible mode decision technique, where CUs with incorrectly and correctly skipped modes are denoted by red shaded blocks and green shaded blocks, respectively. It is observed in Figure 6.7(b) that Intra mode is skipped for many NIBs, because $d_{HEVC}^{avg}$ of these CUs are not equal to the current depth level in SCC, as analyzed in Section 6.3.2. Besides, IBC mode and PLT mode are skipped for many CUs as shown in Figure 6.7(c) and (d). However, almost all SCBs are well detected and decided to check PLT mode by the proposed FHST. It is also noted that some smooth CUs need to check all modes, as shown in Figure 6.7(b)–(d). The reason is that those smooth CUs may select any mode in the training frames so that it is difficult to make flexible mode decisions. When compared with the mode decisions made by CBFT, only 4 CUs are incorrectly skipped, while the remaining 106 CUs are correctly skipped and then encoded by the optimal modes correctly.

For the region containing many SCBs in “Programming”, it is observed in Figure 6.8(b) that almost all SCBs are well detected and Intra mode is skipped for them. Besides, it is observed in Figure 6.8(c) and (d) that many SCBs are decided to check one mode either from IBC or PLT mode, so that the re-encoding time is further reduced when compared with the fast mode decision algorithms [54], [55], [72] that only perform CU type classification. When compared with the mode decisions made by CBFT in Figure 6.8(a), only 3 CUs are incorrectly skipped, while the remaining 91 CUs are encoded by their optimal modes correctly. Therefore, many redundant mode candidates are skipped by the proposed FHST, while the optimal modes are well kept.

### 6.4.5 Discussion on the Feature Importance

Our proposed FHST utilizes features from 4 categories to reduce the re-encoding time of SCC, which are features from the HEVC decoder, static features, dynamic features, and spatial features. In this sub-section, we discuss the importance of each

Table 6.7: Performance of the proposed transcoder for YUV 4:2:0 sequences with different feature combination.

| Sequences | No spatial features | | No decoder features | | No dynamic features | | No static features | | All features | |
|---|---|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 7.64 | -47.86 | 2.18 | -36.30 | 3.57 | -38.49 | 6.22 | -47.88 | 2.89 | -39.57 |
| Desktop (*T*) | 3.57 | -45.50 | 1.28 | -38.05 | 1.50 | -36.56 | 3.34 | -49.20 | 1.15 | -40.38 |
| Map (*T*) | 5.01 | -62.03 | 2.58 | -56.83 | 1.62 | -54.42 | 3.14 | -62.92 | 0.11 | -57.02 |
| MissionControlClip2 (*T*) | 4.50 | -58.86 | 2.95 | -44.05 | 1.50 | -51.18 | 2.82 | -57.09 | 1.11 | -53.46 |
| Robot (*T*) | 2.24 | -77.97 | 6.75 | -58.27 | 1.97 | -73.92 | 1.86 | -76.56 | 2.00 | -74.87 |
| BasketballScreen (*NT*) | 4.63 | -58.30 | 3.50 | -47.58 | 1.63 | -49.84 | 3.22 | -57.13 | 1.15 | -52.95 |
| ChineseEditing (*NT*) | 3.41 | -43.47 | 1.84 | -34.05 | 0.96 | -35.57 | 2.47 | -44.26 | 0.80 | -36.16 |
| ChinaSpeed (*NT*) | 2.50 | -66.12 | 1.44 | -52.32 | 1.30 | -58.16 | 1.41 | -64.07 | 0.99 | -60.02 |
| FlyingGraphics (*NT*) | 7.06 | -49.20 | 2.67 | -39.00 | 2.28 | -38.67 | 5.02 | -47.29 | 2.08 | -40.13 |
| MissionControlClip3 (*NT*) | 4.43 | -52.92 | 2.68 | -44.46 | 1.73 | -45.68 | 3.19 | -54.04 | 1.70 | -48.81 |
| Programming (*NT*) | 3.89 | -51.72 | 2.21 | -40.24 | 1.24 | -42.97 | 2.30 | -52.06 | 0.88 | -45.88 |
| SlideShow (*NT*) | 4.97 | -72.30 | 5.99 | -43.01 | 1.17 | -66.99 | 2.30 | -71.28 | 0.71 | -68.71 |
| WebBrowsing (*NT*) | 4.38 | -52.11 | 2.10 | -43.66 | 1.77 | -45.29 | 4.24 | -53.43 | 1.62 | -48.15 |
| ***Average*** | 4.48 | -56.80 | 2.94 | -44.45 | 1.71 | -49.06 | 3.19 | -56.71 | 1.32 | -51.24 |
| ***PFactor*** | 12.68 | | 15.12 | | 28.69 | | 17.78 | | 38.82 | |
| ***IFactor*** | Spatial features 2.06 | | Decoder features 1.57 | | Dynamic features 0.83 | | Static features 1.18 | | | |

feature category by removing it from our proposed FHST firstly and then observing the performance improvement when it is added back. In general, better performance is denoted by larger re-encoding time reduction and smaller BDBR increase. Therefore, we adopt a similar performance factor, *PFactor*, as in [99] to denote the coding performance

$$PFactor = -\frac{\Delta\mathrm{Time}}{\mathrm{BDBR}}. \tag{6.13}$$

A larger value of $PFactor$ represents better performance. Then based on *PFactor*, we calculate the importance factor, *IFactor*, of each feature category by

$$IFactor = \frac{PFactor_{FA} - PFactor_{FR}}{PFactor_{FR}} \tag{6.14}$$

where $PFactor_{FR}$ and $PFactor_{FA}$ are the performance factors of feature removed and feature added back transcoders, respectively. Table 6.7 presents the results of the proposed FHST when either one category of features is removed. It should be noted that removing spatial features means disabling the Spatial-Info classifier in Table 6.7. It is observed that all feature categories are helpful for improving coding performance, and the transcoder with all features implemented has the best performance with *PFactor* of 38.82. For the transcoder without spatial features, decoder features, dynamic features and static features, 56.80%, 44.45%, 49.06% and 56.71% re-encoding time are saved while the BDBR is increased by 4.48%, 2.94%, 1.71%, and 3.19%, respectively. Therefore, the most important feature category to the proposed FHST is spatial features and then followed by decoder features, static features and dynamic features, whose *IFactors* are 2.06, 1.57, 1.18 and 0.83, respectively.

### 6.4.6 Performance Comparison of YUV 4:4:4 Format

In this sub-section, the proposed FHST is further extended to support fast transcoding of screen content videos in YUV4:4:4 format, where “Console”, “Desktop”, “Map”, “MissionControlClip2”, “Robot” were used to generate training data, and $\alpha$ is set to 0.75, $\beta$ is set to 0.5. Similarly, the fast algorithms in [51], [54], [55] are used to replace the original SCC encoder of CBFT in Figure 1.6 for comparison. Besides, we also compared FHST with the fast HEVC to SCC transcoding algorithm [72], which is only designed for YUV4:4:4 format. Considering that Duanmu *et al.*’s algorithm [72] is the only existing fast HEVC to SCC transcoding algorithm, and it was implemented in HM-16.4 [100] and HM-16.4+SCM-4.0 [101], we re-implemented all other algorithms in the same reference software as Duanmu *et al.*’s algorithm [72] to make fair comparisons.

Table 6.8 shows the comparisons of the proposed FHST with the fast SCC encoding algorithms [51], [54], [55] under CTC [30]. It is also observed that the performance of FHST is much better than the fast SCC encoding algorithms [51], [54], [55]. For the *T* sequences, 53.50% re-encoding time is saved with 1.42% increase in BDBR. For the *NT* sequences, similar performance is obtained, where 55.29% re-encoding time is reduced with BDBR increased by 1.15%. It again proves that the proposed FHST is generalizable to the sequences which are not used in training. On average, the proposed FHST provides 54.65% re-encoding time reduction with a negligible increase in BDBR of 1.25%.

Table 6.8: Performance comparison of the proposed transcoder with different fast SCC encoding algorithms for YUV 4:4:4 sequences.

| Sequences | CBFT + Zhang [51] | | CBFT + Lei [54] | | CBFT + Yang [55] | | Proposed FHST | |
|---|---|---|---|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 1.67 | -20.26 | 2.24 | -21.93 | 4.79 | -30.15 | 2.36 | -50.24 |
| Desktop (*T*) | 1.36 | -30.16 | 1.35 | -23.21 | 4.08 | -35.06 | 0.52 | -56.92 |
| Map (*T*) | 1.21 | -28.28 | 3.51 | -20.04 | 7.26 | -15.73 | 1.00 | -45.79 |
| MissionControlClip2 (*T*) | 2.22 | -29.55 | 2.84 | -28.02 | 3.87 | -29.56 | 1.50 | -54.29 |
| Robot (*T*) | 10.89 | -12.63 | 12.67 | -30.92 | 5.47 | -19.31 | 1.71 | -60.26 |
| BasketballScreen (*NT*) | 1.34 | -30.95 | 1.19 | -22.38 | 3.80 | -25.11 | 0.92 | -52.01 |
| ChineseEditing (*NT*) | 0.33 | -22.84 | 0.61 | -19.42 | 3.63 | -24.90 | 0.59 | -47.26 |
| EBURainFruits (*NT*) | 7.38 | -18.61 | 7.34 | -34.72 | 2.88 | -20.61 | 2.23 | -67.92 |
| FlyingGraphics (*NT*) | 0.45 | -4.39 | 1.08 | -19.14 | 4.56 | -26.84 | 1.03 | -45.53 |
| Kimono1(*NT*) | 10.29 | -3.79 | 8.39 | -41.82 | 4.18 | -27.14 | 1.39 | -56.99 |
| MissionControlClip3 (*NT*) | 1.83 | -26.93 | 1.43 | -23.64 | 3.08 | -29.30 | 1.24 | -54.57 |
| Programming (*NT*) | 1.28 | -28.42 | 1.76 | -22.19 | 7.27 | -26.35 | 1.08 | -50.74 |
| SlideShow (*NT*) | 1.99 | -41.03 | 1.76 | -48.79 | 5.81 | -45.94 | 1.37 | -63.83 |
| WebBrowsing (*NT*) | 1.57 | -40.16 | 3.21 | -25.38 | 5.65 | -34.30 | 0.50 | -58.80 |
| **Average (*T*)** | 3.47 | -24.18 | 4.52 | -24.82 | 5.09 | -25.96 | 1.42 | -53.50 |
| **Average (*NT*)** | 2.94 | -24.12 | 2.97 | -28.61 | 4.54 | -28.94 | 1.15 | -55.29 |
| **Average (*ALL*)** | 3.13 | -24.14 | 3.53 | -27.26 | 4.74 | -27.88 | 1.25 | -54.65 |

Comparatively, Zhang *et al.*'s algorithm [51], Lei *et al.*'s algorithm [54] and Yang *et al.*'s algorithm [55] provide 24.14%, 27.26% and 27.88% re-encoding time reduction with BDBR increased by 3.13%, 3.53%, and 4.74%, respectively. Then, based on the same reference software of HEVC and SCC, we made an indirect comparison between our proposed FHST and the only existing fast HEVC to SCC transcoding algorithm [72], and the results are presented in Table 6.9. It is observed when compared with CBFT, Duanmu *et al.*'s algorithm [72] achieves 47.93% re-encoding time reduction for their selected sequences while BDBR is increased by 2.14%. Comparatively, our proposed FHST achieves 54.01% re-encoding time reduction for their selected sequences while BDBR is only increased by 1.11%. Compared with Duanmu *et al.*'s algorithm [72] which only utilizes features from the HEVC decoder and static features, our proposed FHST additionally utilizes spatial features and dynamic features, so that more accurate decision is provided. Besides, Duanmu *et al.*'s algorithm [72] always checks both IBC and PLT modes for SCBs. However, we allow the case that only one mode is checked for SCBs, as observed in Figure 6.7(c), (d) and Figure 6.7(c), (d), so that higher re-encoding time reduction is provided.

Table 6.9: Performance comparison of the proposed transcoder with other transcoder for YUV 4:4:4 sequences.

| Sequences | Duanmu [72] | | Proposed FHST | |
|---|---|---|---|---|
| | BDBR (%) | ΔTime (%) | BDBR (%) | ΔTime (%) |
| Console (*T*) | 1.85 | -49.0 | 2.36 | -50.24 |
| Desktop (*T*) | 1.72 | -48.1 | 0.52 | -56.92 |
| Map (*T*) | | | 1.00 | -45.79 |
| MissionControlClip2 (*T*) | | | 1.50 | -54.29 |
| Robot (*T*) | | | 1.71 | -60.26 |
| BasketballScreen (*NT*) | 3.13 | -46.1 | 0.92 | -52.01 |
| ChineseEditing (*NT*) | | | 0.59 | -47.26 |
| EBURainFruits (*NT*) | | | 2.23 | -67.92 |
| FlyingGraphics (*NT*) | 1.94 | -50.1 | 1.03 | -45.53 |
| Kimono1(*NT*) | | | 1.39 | -56.99 |
| MissionControlClip3 (*NT*) | | | 1.24 | -54.57 |
| Programming (*NT*) | 1.05 | -42.9 | 1.08 | -50.74 |
| SlideShow (*NT*) | 2.21 | -51.4 | 1.37 | -63.83 |
| WebBrowsing (*NT*) | 3.08 | -47.9 | 0.50 | -58.80 |
| **Average ([72]'s sequences)** | 2.14 | -47.93 | 1.11 | -54.01 |
| **Average (*ALL*)** | | | 1.25 | -54.65 |

## 6.5 Chapter Summary

In this chapter, a fast HEVC to SCC transcoder FHST is proposed by early CU partitioning termination and flexible mode decision. Four categories of features are collected from both the HEVC decoder side and the SCC encoder side to simplify the transcoding process. First, an early CU partitioning termination technique is proposed to map the optimal CU size from HEVC to SCC. Then, a flexible encoding structure is proposed where DTs are generated to check each mode candidate adaptively in SCC. With the help of various features from the four categories and the flexible encoding structure, higher re-encoding time can be reduced with less RD performance loss compared with other algorithms. Experimental results show that the proposed FHST provides 51.24% and 54.65% re-encoding time reduction with a negligible increase in BDBR of 1.32% and 1.25% for YUV 4:2:0 and YUV 4:4:4 screen content sequences, respectively.

# Chapter 7 Conclusions and Future Work

In this thesis, we have carried out a research study on the fast mode and CU partitioning decision of SCC. To achieve high coding efficiency for screen coding videos, SCC adopts an exhaustive searching strategy among all depth levels and mode candidates. By implementing the proposed algorithms, unnecessary mode candidates and CU sizes are eliminated such that coding complexity can be reduced. In each chapter, the motivation of the proposed algorithm is firstly given, and then the proposed algorithms with the corresponding rationales were introduced with illustrations. Finally, simulation results were provided to show the effectiveness of the proposed algorithms. In this chapter, we first highlight the main contributions of this thesis. Then, some possible directions that could be the focus of the future research are discussed.

## 7.1 Contributions of the Thesis

In the objective of computational complexity reduction in SCC, our contributions chiefly include constructive machine learning based proposals of (1) an online learning based fast prediction algorithm which extract content dependent rules from learning frames; (2) a flexible encoding framework by sequential arrangement of DTs; (3) a deep learning based fast prediction network, DeepSCC, which contains much more trainable parameters than the traditional machine learning based approaches; (4) a fast HEVC to SCC transcoder FHST that migrates the legacy screen content videos from HEVC to SCC to improve the coding efficiency.

In particular, our conclusions are:

- An online learning based fast SCC encoding algorithm was explored in Chapter 3.

Since the mode and CU partitioning decisions in the same scene have a high correlation, the first frame in a scene is used to derive content dependent rules. Then these rules are applied to the following frames in the same scene to eliminate unnecessary checking of mode candidates and CU size. The content-dependent rules can achieve high prediction accuracy since it is tailor-made for a certain scene. Simulation results have proven that this algorithm succeeds in reducing encoding time with negligible RD performance loss.

- A flexible encoding framework by a sequential arrangement of DTs was proposed in Chapter 4, and it explores both static features that describing CU content and dynamic features that reveal the unique intermediate coding information of a CU. To utilize dynamic features for prediction, this framework checks each mode separately by inserting a classifier before checking a mode, and it facilitates either IBC or PLT mode to be checked for SCBs. Simulation results show that computational complexity is further reduced.
- To avoid the risk that humans may ignore some important features when doing feature extraction, a deep learning based fast prediction network DeepSCC was presented in Chapter 5. It directly extracts features from the raw pixels by using extensive learnable parameters, and it is able to make the more accurate mode decision of Intra, IBC, and PLT rather than the simple CU type classification of NIBs and SCBs. By outputting labels for all CUs in a CTU in a single test, the computational overhead of DeepSCC is less than 4% by only using a CPU for testing, and simulation results show that 48.18% encoding time is reduced with a negligible BDBR increase of 1.18%.
- In Chapter 6, a fast HEVC to SCC transcoder FHST was proposed to migrate the legacy screen content videos from HEVC to SCC for coding efficiency improvement.

A flexible mode decision framework is adopted and various features from 4 categories are employed, which are the HEVC decoder, static features, dynamic features, and spatial features. On the one hand, high decision accuracy is achieved because mode decision is considered from different aspects by utilizing features from more than one category. On the other hand, high computational complexity is reduced because the flexible structure considers the decision of each mode separately. Simulation results show that FHST provides 51.24% and 54.65% re-encoding time reduction with 1.32% and 1.25% negligible BDBR loss for YUV 4:2:0 and YUV 4:4:4 screen content sequences.

## 7.2 Future Work

With the successful techniques proposed and evaluated in this thesis, we now provide some related directions for our future studies.

### 7.2.1. Fast Inter-Prediction of SCC

In this thesis we propose various fast algorithms focused on the fast intra-prediction of SCC, and we can extend our proposed algorithms to inter-prediction. In inter-prediction, a CU needs to check more mode candidates, such as hash base ME, merge & skip, inter 2N×2N, inter 2N×N, inter N×2N. Since the mode selection of a CU in inter-prediction is strongly affected by the temporal correlation, the mode selection for two CUs with the same content can be different. Therefore, we can insert a decision tree based mode classifier before each target mode to decide whether a mode should be checked or not. Furthermore, for CNN based framework, the long- and short-term memory (LSTM) can be adopted to utilize the temporal correlation for fast inter-prediction.

### 7.2.2. Fast Bitrate Transcoding of SCC

With the proliferation of cloud-based video streaming technology, it is important to support screen content distributions from the cloud server to multiple clients. Those clients may have different bandwidths that prefer screen content videos with different quality levels. To meet the bandwidths of different clients, a cloud server needs to generate multiple bitstreams with reduced quality levels from the original high-quality video stream. As illustrated in Figure 7.1, each client can flexibly choose the most suitable stream given the network condition. For this purpose, fast bitrate transcoding of SCC is considered as our next work. Bitrate transcoding is a technology that transforms a high bit-rate video to a low bit-rate video with the same video format. To control the quality of the transcoded video, the value of QP is adjusted when doing video re-encoding. We can first decode the original bitstream of the high-quality video while collecting the decoder side information, such as optimal mode and CU size. Then, we re-encode the decoded video by a larger value of QP with help of decoder side information to reduce the re-encoding time.

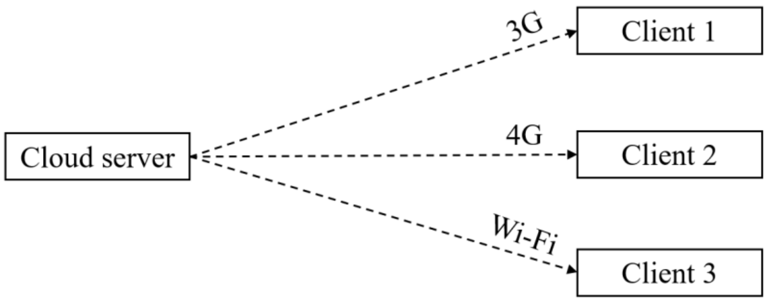


Figure 7.1: Adaptive screen content distribution over cloud.

### 7.2.3. GPU Based Parallel Encoding of SCC

In this thesis, techniques are proposed to accelerate the encoding and transcoding of SCC by using CPU without parallel processing. To further reduce the computational complexity, GPU can be utilized as a co-processor to assist CPU for parallel encoding.

GPU is a highly parallel multi-threaded and many-core processor that has the tremendous computational ability. Before the encoding process, GPU can be enabled to find the rough optimal BV in IBC mode and the optimal direction of Intra mode for multiple CUs based on the uncompressed reference samples. Although this prediction is not accurate because the intermediate coding statistics might not be exactly the same as the true encoding process, this pre-processing step limits the search range. Then CPU is enabled to fine-tune the optimal mode in the limited search range provided by GPU.

# References

[1] ISO/IEC 10918-1, “Information technology -- digital compression and coding of continuous-tone still images: requirements and guidelines”, 1994.

[2] Y. M. Lei and M. Ouhyoung, “Software-based motion JPEG with progressive refinement for computer animation,” IEEE Transactions on Consumer Electronics, vol. 40, pp. 557-562, Aug. 1994.

[3] ISO/IEC 11172-2, “Information technology -- coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s -- Part 2: Video,” 1993.

[4] ISO/IEC 13818-2, “Information technology -- generic coding of moving pictures and associated audio information: Video,” 1996.

[5] Video Codec for Audiovisual Services at p×64 kbit/s, ITU-T Recommendation H.261., 1993.

[6] Video Coding for Low Bitrate Communication, ITU-T Recommendation H.263, May. 1997.

[7] ISO/IEC 14496-2, “Information technology – coding of audio-visual objects – Part 2: Video,” 2001.

[8] ISO/IEC 14496-10, “Information technology – coding of audio-visual objects – Part 10: Advanced Video Coding,” 2003.

[9] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol.13, no.7, pp.560-576, Jul. 2003.

[10] Iain E. G. Richardson, “H.264 and MPEG-4 video vompression: video voding for next-generation multimedia,” Wiley, 2003.

[11] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[12] Y. Lu, S. Li, and H. Shen, “Virtualized screen: A third element for cloud-mobile convergence,” IEEE Multimedia, vol. 18, no. 2, pp. 4–11, Feb. 2011.

[13] J. Xu, R. Joshi, and R. A. Cohen, “Overview of the emerging HEVC screen content coding extension,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 1, pp. 50–62, Jan. 2016.

[14] J. Lainema, F. Bossen, W.-J. Han, J. Min, and K. Ugur, “Intra coding of the HEVC standard,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1792–1801, Dec. 2012.

[15] M. Budagavi and D.-K. Kwon, “AHG8: Video coding using intra motion compensation,” document JCTVC-M0350, Apr. 2013.

[16] C. Pang, K. Rapaka, J. Sole, and M. Karczewicz, “SCCE1: Test 3.6—Block vector coding for intra block copy,” document JCTVC-R0186, Jul. 2014.

[17] K. Rapaka, M. Karczewicz, C. Pang, K. Miyazawa, A. Minezawa, and S. Sekiguchi, “Non-CE1: Block vector voding for intra block copy,” document JCTVC-S0143, Oct. 2014.

[18] X. Xu, T.-D. Chuang, S. Liu, and S. Lei, “Non-CE2: Intra BC merge mode with default candidates,” document JCTVC-S0123, Oct. 2014.

[19] C. Pang, Y.-K. Wang, V. Seregin, K. Rapaka, M. Karczewicz, X. Xu, S. Liu, S. Lei, B. Li, and J. Xu, “Non-CE2: Intra block copy and inter signalling unification,” document JCTVC-T0227, Feb. 2015.

[20] X. Xu, S. Liu, T.-D. Chuang, and S. Lei, “Block vector prediction for intra block copying in HEVC screen content coding,” Data Compression Conference, Apr. 2015, pp. 273-282.

[21] X. Xu et al., “Intra block copy in HEVC screen content coding extensions”, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp. 409-419, Dec. 2016.

[22] Z. Ma, W. Wang, M. Xu, and H. Yu, “Advanced screen content coding using color table and index map”, IEEE Transactions Image Processing, vol. 23, no. 10, pp. 4399-4412, Oct. 2014.

[23] Y.-C. Sun, T.-D. Chuang, P. Lai, Y-W. Chen, S. Liu, Y.-W. Huang, and S. Lei, “Palette mode — A new coding tool in screen content coding extensions of HEVC”, IEEE International Conference on Image Processing, pp. 2409-2413, Sept. 2015.

[24]L. Guo, W. Pu, F. Zou, J. Sole, M. Karczewicz, and R. Joshi, “Color palette for screen content coding”, IEEE International Conference on Image Processing, pp. 5556-5560, Oct. 2014.

[25]X. Xiu, Y. He, R. Joshi, M. Karczewicz, P. Onno, C. Gisquet, and G. Laroche, “Palette-Based Coding in the Screen Content Coding Extension of the HEVC Standard”, IEEE International Conference on Image Processing, pp. 5556-5560, Apr. 2015.

[26]S. Ye, Z. Chen, W. Zhang, and L. Xu, “Parallel palette mode decoding for HEVC SCC”, IEEE International Symposium on Circuits and Systems, pp. 2551-2554, May. 2016.

[27] W. Pu, R. Joshi, V. Seregin, F. Zou, J. Sole, Y.-C. Sun, T.-D. Chuang, P. Lai, S. Liu, S.-T. Hsiang, J. Ye, and Y.-W. Huang, “Palette mode coding in HEVC screen content coding extension”, IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp. 420-432, Dec. 2016.

[28] L. Guo, M. Karczewicz, and J. Sole, “RCE3: Results of test 3.1 on palette mode for screen content coding,” document JCTVC -N0247, Jul. 2013.

[29] G. Bjontegaard, “Calculation of average PSNR differences between rd-curves,” document VCEG-M33, VCEG, Mar. 2001.

[30] H. -P. Yu, R. Cohen, K. Rapaka, and J. -Z Xu, “Common test conditions for screen content coding”, document JCTVC-X1015-r1, May. 2016.

[31] HM-16.12+SCM-8.3, HEVC test model version 16.12 screen content model version 8.3, [Online], available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.12+SCM-8.3/.

[32] J. Vanne, M. Viitanen, T. D. Hämäläinen, and A. Hallapuro, “Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1885–1898, Dec. 2012.

[33] H. S. Malvar, G. J. Sullivan, and S. Srinivasan, “Lifting-based reversible color transformations for image compression,” in SPIE Applications of Digital Image Processing. International Society for Optical Engineering, Aug. 2008.

[34] A. Minezawa, S. Sekiguchi, and T. Murakami, AHG5/AHG8: “On RGB to YCbCr conversion for screen contents,” document JCTVC-N0115, ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 WP3, Aug. 2013.

[35] L. Zhang, J. Chen, J. Sole, M. Karczewicz, X. Xiu, Y. He, and Y. Ye, “SCCE5 Test 3.2.1: In-loop color-space transform,” document JCTVC-R0147, Jul. 2014.

[36] L. Zhang, J. Chen, J. Sole, M. Karczewicz, X. Xiu, and J.-Z. Xu, “Adaptive color-space transform for HEVC screen content coding,” Data Compression Conference, Apr. 2015, pp. 233-242.

[37] L. Zhang, X. Xiu, J. Chen, M. Karczewicz, Y. He, Y. Ye, J. Xu, J. Sole, and W.-S. Kim, “Adaptive color-space transform in HEVC screen content coding,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp.446–459, Dec. 2016.

[38] Y.-J. Cho, J.-H, KO, H.-G Yu, J.-H. Lee, D.-J. Park and S.-H. Jun, “Adaptive motion vector resolution based on the rate-distortion cost and coding unit depth,” IEEE International Advance Computing Conference, Feb. 2014. pp.1000–1003.

[39] Z. Wang, J. Ma, F. Luo, and S. Ma, “Adaptive motion vector resolution prediction in block-based video coding,” Visual Communications and Image Processing, Dec. 2015, pp.1–4.

[40] Z. Wang, J. Zhang, N. Zhang, and S. Ma, “Adaptive motion vector resolution scheme for enhanced video coding,” Data Compression Conference, Mar. 2016. pp.101–110.

[41] B. Ray, J. Jung, and M.-C. Larabi, “A block level adaptive MV resolution for video coding,” IEEE International Conference on Multimedia and Expo, Jul. 2017. pp.49–54.

[42] B. Li, J. Xu, G. Sullivan, Y. Zhou, and B. Lin, "Adaptive motion vector resolution for screen content," document JCTVC-S0085, Oct. 2014.

[43] S. L. Yu and C. Chrysafis, "New intra prediction using intra-macroblock motion compensation," document JVT-C151r1, May 2002.

[44] H.-S. Kim, and R.-H. Park, "Fast CU partitioning algorithm for HEVC using an online-learning-based Bayesian decision rule," IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 1, pp. 130–138, Jan. 2016.

[45] S-H. Jung, and H.W. Park, "a fast mode decision method in HEVC using adaptive ordering of modes," IEEE Transactions on Circuits and Systems for Video Technology, vol. 26, no. 10, pp. 1846–1858, Oct. 2016.

[46] Q. Hu, X.-Y. Zhang, Z.-R. Shi, and Z.-Y. Gao, "Neyman-Pearson-based early mode decision for HEVC encoding," IEEE Transaction on Multimedia, vol. 18, no. 3, pp.379–391, Mar. 2016.

[47] S.-H. Tsang, Y.-L. Chan, and W.-C. Siu, "Fast and efficient intra coding techniques for smooth regions in screen content coding based on boundary prediction samples," IEEE International Conference on Acoustics, Speech and Signal Processing, Apr. 2015, pp.1409–1413.

[48] S.-H. Tsang, Y.-L. Chan, and W.-C. Siu, "Hash based fast local search for intra block copy (IntraBC) mode in HEVC screen content coding," Signal and Information Processing Association Annual Summit and Conference, Dec. 2015, pp. 396–400.

[49] S.-H. Tsang, W. Kuang, Y.-L. Chan and W.-C. Siu, "Fast HEVC screen content coding by skipping unnecessary checking of intra block copy mode based on CU activity and gradient," Signal and Information Processing Association Annual Summit and Conference, Dec. 2016, pp.1–5.

[50] F. Duanmu, Z. Ma, and Y. Wang, "Fast CU partition decision using machine learning for screen content compression," IEEE International Conference on Image Processing, Sept. 2015, pp. 4972–4976.

[51] H. Zhang, Q. Zhou, N.-N Shi, F. Yang, X. Feng, and Z. Ma, “Fast intra mode decision and block matching for HEVC screen content compression,” IEEE International Conference on Acoustics, Speech and Signal Processing, Mar. 2016, pp.1377–1381.

[52] M. Zhang, Y. Guo, and H. Bai, “Fast intra partition algorithm for HEVC screen content coding,” IEEE Visual Communications and Image Processing Conference, Dec. 2014, pp. 390–393.

[53] F. Duanmu, Z. Ma, and Y. Wang, “Fast mode and partition decision using machine learning for intra-frame coding in HEVC screen content coding extension,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp.517–531, Dec. 2016.

[54] J. Lei, D. Li, Z, Pan, Z. Sun, S. Kwong, and C. Hou, “Fast intra prediction based on content property analysis for low complexity HEVC-based screen content coding,” IEEE Transactions on Broadcasting, vol. 63, no.1, pp.48–58, Mar. 2017.

[55] H. Yang, L. Shen, and P. An, “An efficient intra coding algorithm based on statistical learning for screen content coding”, IEEE International Conference on Image Processing, Sept. 2017, pp. 2468–2472.

[56] C. Huang, Z. Peng, F. Chen, Q. Jiang, G. Jiang and Q. Hu, “Efficient CU and PU decision based on neural network and gray level co-occurrence matrix for Intra prediction of screen content coding,” IEEE Access, vol. 6, pp. 46643 - 46655, Aug. 2018.

[57] J. D. Cock, S. Notebaert, P. Lambert, R. V. Walle, “Architectures for fast transcoding of H.264/AVC to quality-scalable SVC streams,” IEEE Transaction on Multimedia, vol. 11, no. 7, pp. 1209-1224, Nov. 2009.

[58] T.-K. Lee, C.-H. Fu, Y.-L. Chan, and W.-C. Siu, “A new motion vector composition algorithm for fast-forward video playback in H.264,” International Symposium on Circuits and Systems, Jun. 2010, pp. 3649-3652.

[59] K. T. Fung, Y. L. Chan, W. C. Siu, “Low-complexity and high-quality frame-skipping transcoder for continuous presence multipoint video conferencing,” IEEE Transaction on Multimedia, vol. 6, no. 1, pp. 31-46, Feb. 2006.

[60] K.-T. Lai, Y.-L. Chan, and W.-C. Siu, “New architecture for dynamic frame-skipping transcoder,” IEEE Transaction on Image Processing, vol. 11, no. 8, pp. 886–900, 2002.

[61] H. Shu and L.-P. Chau, “The realization of arbitrary downsizing video transcoding,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 16, no. 4, pp. 540-546, Apr. 2006.

[62] D. Xu and P. Nasiopoulos, “Logo insertion transcoding for H.264/AVC compressed video,” IEEE International Conference on Image Processing, pp. 3693-3696, Nov. 2009.

[63] T. Shanableh, T. May, and F. Ishtiaq, “Error resiliency transcoding and decoding solutions using distributed video coding techniques,” Signal Processing: Image Communication, vol. 23, no. 8, pp. 610-623, 2008.

[64] Y.-L. Chan, H.-K Cheung and W.-C. Siu, “Compressed-domain techniques for error-resilient video transcoding using RPS,” IEEE Transaction on Image Processing, vol. 18, no. 2, pp. 357–370, 2009.

[65] G. Fernandez-Escribano, H. Kalva, P. Cuenca, L. Orozco-Barbosa, and A. Garrido, “A fast MB mode decision algorithm for MPEG-2 to H.264 P-frame transcoding,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 18, no. 2, pp. 172-185, Feb. 2008.

[66] G. Fernandez-Escribano, H. Kalva, J. Martinez, P. Cuenca, L. Orozco-Barbosa, and A. Garrido, “An MPEG-2 to H.264 video transcoder in the baseline profile,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 20, no. 5, pp. 763-768, 2010.

[67] T. Shanableh, E. Peixoto, and E. Izquierdo, “MPEG-2 to HEVC video transcoding with content-based modeling,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 7, pp. 1191-1196, 2013.

[68] H. Yuan, C. Guo, J. Liu, X, Wang, and S. Kwong, “Motion-homogeneous based fast transcoding method from H.264/AVC to HEVC,” IEEE Transactions on Multimedia, vol. 19, no 7, pp. 1416-1430, 2017.

[69] F. Zhang, Z. Shi, X, Zhang, and Z, Guo, “Fast H.264/AVC to HEVC transcoding based on residual homogeneity,” in Proc. Int. Conf. Audio, Language and Image Processing, Jul. 2014, pp. 765-770.

[70] A. Nagaraghatta, Y. Zhao, G. Maxwell, and S. Kannangara, “Fast H.264/AVC to HEVC transcoding using mode merging and mode mapping,” in Proc. Int. Conf. Consumer Electronics, Sept. 2015, pp. 765-770.

[71] P. Xing, Y. Tian, X. Zhang, Y. Wang, and T. Huang, “A coding unit classification based AVC-to-HEVC transcoding with background modeling for surveillance videos,” IEEE Visual Communications and Image Processing Conference, Nov. 2013, pp. 1-6.

[72] F. Duanmu, Z. Ma, W. Wang, M. Xu, and Y. Wang, “A novel screen content fast transcoding framework based on statistical study and machine learning,” IEEE International Conference on Image Processing, Sept. 2016, pp. 4205-4209.

[73] W. Kuang, Y.-L. Chan, S.-H. Tsang, and W.-C. Siu “Fast intraprediction for high-efficiency video coding screen content coding by content analysis and dynamic thresholding,” Journal of Electronic Imaging, vol. 27, no. 5, pp. 053029-1–053029-18, 8 October 2018.

[74] Z. Pan, H. Shen, Y. Lu, S. Li, and N. Yu, “A low-complexity screen compression scheme for interactive screen sharing,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 6, pp. 949–960, Jun. 2013.

[75] J. W. Lee, and B. W. Dickinson, “Temporally adaptive motion interpolation exploiting temporal masking in visual perception,” IEEE Transaction on Image Processing, vol. 3, no. 5, pp. 513–526, Sept. 1994.

[76] W. Kuang, Y.-L. Chan, S.-H. Tsang, and W.-C. Siu, “Machine learning based fast intra mode decision for HEVC screen content coding via decision trees,” IEEE Transactions on Circuits and Systems for Video Technology, early access, 2019.

[77] J. R. Quinlan, “C4.5: Programs for machine learning,” Morgan Kaufmann, 1993.

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

[79] Y. Mansour, “Pessimistic decision tree pruning based on tree size,” International Conference on Machine Learning, pp. 195–201, 1997.

[80] P. Burman, “A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods,” Biometrika, vol. 76, no. 3, pp. 503-514, Sept. 1989.

[81] N. Japkowicz, “The class imbalance problem: Significance and strategies,” International Conference on Artificial Intelligence, 2000, pp. 111–117.

[82] Machine Learning Based Fast Intra Mode Decision for HEVC Screen Content Coding Via Decision Trees. [Online]. Available at: http://www.eie.polyu.edu.hk/~ylchan/research/DT-FastSCC/.

[83] I. Guyon, and A. Elisseeff, “An introduction to variable and feature selection,” The Journal of Machine Learning Research, vol. 3, pp. 1157–1182, Mar. 2003.

[84] J. Guo, L. Zhao, and T. Lin, “Response to B1002 call for test materials: Five test sequences for screen content video coding”, document JVET-C0044, May. 2016.

[85] R. Cohen, “AHG8: 4:4:4 game content sequences for HEVC range extensions development”, document JCTVC- N0294, Aug. 2013.

[86] A. M. Tourapis, D. Singer, and K. Kolarov, “New test sequences for screen content coding”, document JCTVC-O0222, Nov. 2013.

[87] H. -P. Yu, W. Wang, X. Wang, J. Ye, and Z. Ma, “AHG8: New 4:4:4 test sequences with screen content”, document JCTVC- O0256, Nov. 2013.

[88] K. Sharman, and K. Suehring, “Common test conditions”, document JCTVC-X1100, May. 2016.

[89] W. Kuang, Y.-L. Chan, S.-H. Tsang, and W.-C. Siu, “DeepSCC: Deep Learning Based Fast Prediction Network for Screen Content Coding,” IEEE Transactions on Circuits and Systems for Video Technology, 2019.

[90] J. Guo, L. Zhao, and T. Lin, “Response to B1002 Call for test materials: Five test sequences for screen content video coding”, document JVET-C0044, May. 2016.

[91] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” ACM International Conference on Multimedia, USA, Nov. 2014, pp. 675–678.

[92] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” IEEE International Conference on Computer Vision, Dec. 2015, pp. 1026–1034.

[93] D. P. Kingma and J. L. Ba, “Adam: a method for stochastic optimization,” International Conference on Learning Representations, May 2015, pp. 1–13.

[94] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, no. 4, pp. 834–848, Apr. 2018.

[95] DeepSCC: Deep learning based fast prediction network for screen content coding. Available at: http://www.eie.polyu.edu.hk/~ylchan/research/DeepSCC/.

[96] W. Ding, Y. Shi, and B. Yin, “YUV444 test sequences for screen content”, document JCTVC-M0431, Apr. 2013.

[97] W. Kuang, Y.-L. Chan, S.-H. Tsang, and W.-C. Siu, “Fast HEVC to SCC transcoder by early CU partitioning termination and decision tree based flexible mode decision for intra-frame coding,” IEEE Access, vol. 7, pp. 8773–8788, 3 Jan. 2019.

[98] HM-16.12, HEVC test model version 16.12, [Online], available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.12/.

[99] G. Correa, P. A. Assuncao, L. V. Agostini, and L. A. da Silva Cruz, “Fast HEVC encoding decisions using data mining,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 25, no. 4, pp. 660-673, Apr. 2015.

[100] HM-16.4, HEVC test model version 16.4, [Online], available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.4/.

[101] HM-16.4+SCM-4.0, HEVC test model version 16.4 screen content model version 4.0, [Online], available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.4+SCM-4.0/.