FAST DEPTH CODING IN 3D-HEVC USING DEEP LEARNING

A DISSERTATION
SUBMITTED TO THE
DEPARTMENT OF ELECTRONIC AND INFORMATION ENGINEERING
OF THE HONG KONG POLYTECHNIC UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Zhen-xiang WANG
December 2017

CERTIFICATE OF ORIGINALITY

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

_____(Signed)

_____(Date)

Abstract

The 3D Extension of the High Efficiency Video Coding standard (3D-HEVC), which has been finalized by the Joint Collaborative Team on Video Coding (JCT-VC) in February 2015, is the new industry standard for 3D applications. The 3D-HEVC provides plenty of advanced coding tools specifically for addressing the coding of auto-stereoscopic videos which have the format of multiple texture views plus depth maps which are responsible for synthesizing intermediate views with sufficient quality for auto-stereoscopic display. The provided tools take advantage of the statistical redundancies amongst texture views and depth maps in the video sequences, as well as the unique characteristics of depth maps to significantly shrink the bitrate while preserving the objective visual quality of the 3D videos. However, those tools with high capabilities in terms of compression come with high complexity of computation which has made the encoding time of 3D video sequences much longer than ever by traversing a lot more mode candidates than all the previous standards. Although the current encoding scheme in the 3D-HEVC standard is able to find the best intra mode candidate for each coding unit in depth maps, the cost of time for encoding is becoming a major obstacle for it to be applied to profitable products.

In this dissertation we address the aforementioned time cost issue by a new intra mode decision algorithm for depth maps, leveraging deep learning to train computational models built from neural network for predicting the best intra angular mode in depth map coding. The predicted intra angular mode is utilized to decide the most probable wedgelet by which the number of wedgelet candidates can be reduced by half. The size of the neural network has been carefully designed to balance the trade-off between the complexity and accuracy in the model prediction. Validation precision and confusion matrix are used to monitor the model training process. Top-k metric is adopted to make use of the predictions from the learned models. We have integrated learned models into the reference software of 3D-HEVC for experiments. The compiled executable binaries are able to harness the power of simultaneous computation of CPU, as well as parallel computation of GPU to accelerate the predictions. The simulation results show that the proposed fast depth coding algorithm provides 64.6% time reduction in average while the BD performance has a trivial decrease comparing with the state-of-the-art 3D-HEVC standard.

Acknowledgments

First and foremost, I would like to give sincere thanks to my supervisor, Dr.Yui-Lam Chan, for his extremely generous support, most insightful advices and innumerable yet constructive feedback. I learned from him to first identify a problem, by reading a vast amount of articles to know what people have achieved and what bottlenecks they have encountered. I learned how to read papers, how to organize them to become the inner comprehension. He guided me to use the machine learning approach to solve the problem that has been found in the first stage. Without his guidance I will not have the idea to learn the deep learning technology and apply it to optimize the video coding. His encyclopedic knowledge and charming personalities made him my mentor in both research and life. I wish to thank Dr.Sik-Ho Tsang, for our in-depth discussions from which I can always find useful clues for next step. His great expertise in video coding significantly benefits me during my intensive period of learning. Also I would like to thank my friends Alex and Jacky, for our extensive discussions about artificial intelligence and their applications. Special thanks to Sue, for all the good times we have been together. Those indispensable memories always shine in my life. Thank you my dear parents. In the light of your great love and constant encouragements, I am always bursting with confidence when challenges come.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Video is the medium to record, copy, playback, broadcast and display the motion images in an electronic style [1]. Watching videos is becoming an important way for human entertainment as well as education. The high definition (HD) and ultra high definition (UHD) videos are increasingly demanding nowadays. People prefer videos with higher resolution than those with lower resolution because HD videos provide better watching experience. However, challenges emerged for delivering videos with high definition. HD/UHD videos typically contain much more information in every picture frame than videos with lower resolution. More data needs to be squeezed into the same capacity for transmission. For example, the uncompressed video with the dimension 720×480 at 30 frames per second requires 0.03 gigabytes per second, while the uncompressed video with the dimension 2880×2048 at 120 frames per second requires 2.12 gigabytes per second. Since bitrate is proportional to system bandwidth for transmission [2], and heavily expanding the bandwidth is usually expensive, the significantly increased bitrate for transmitting the video data is becoming one of the major obstacles for HD video services.

To cope with the growing need for higher compression of moving pictures [3], Joint Collaborative Team on Video Coding (JCT-VC) [4] has finalized the High Efficiency Video Coding which is the newest international video coding standard for substantially ameliorate the compression performance against the previous standards. Comparing with H.264 Advanced Video Compression Standard [5], H.265 High Efficiency Video Coding Standard provides a reduction of fifty percent in terms of bitrate while maintaining the objective video quality at the same level.

Three-dimensional (3D) video has been introduced to market via lots of ways, including Blu-Ray disc, cable and satellite transmission, terrestrial broadcast, and streaming or downloading from the Internet [6]. 3D video provides the perception of depth information which augments the vividness of video contents. Currently most 3D videos in the market are using stereo display technology. Two similar views, one for left eye, the other for right eye, are presented at the same time with the multiplexing techniques enabling the adjustments of video geometry information [7] to provide the

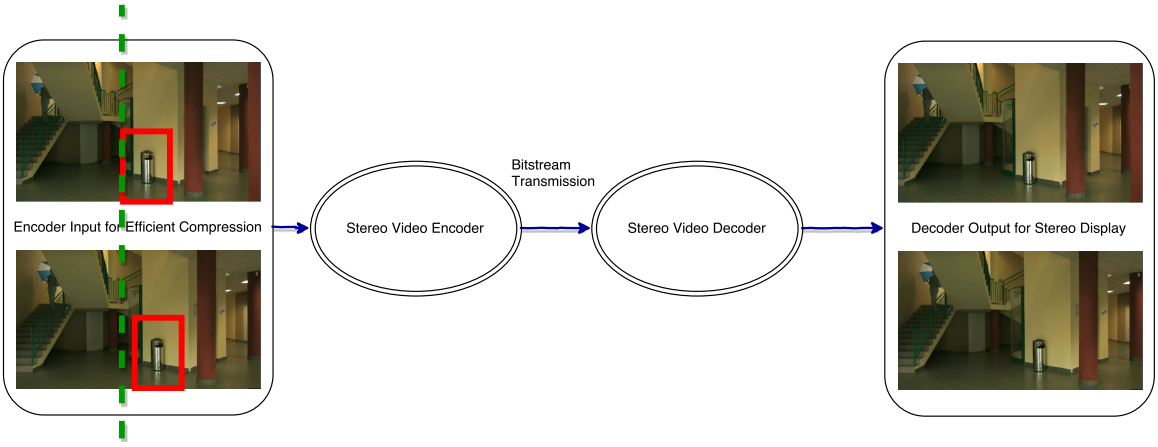


Figure 1.1: System Structure for transmitting videos targeting stereo display.

3D effect. Figure 1.1 illustrates the typical system structure for transmitting videos targeting stereo display. It can be observed that there exists a displacement between two views. The green vertical left margins of the red rectangles in two views at encoder side are different with each other. Such a displacement is the visual disparity for 3D perception. Stereoscopic videos [8] have achieved great profitability for movie theatres in recent years. For example, IMAX 3D has became the most popular one that offering the immersing multimedia experiences around the world. Special 3D glasses are needed for watching IMAX 3D movies. The current 3D film industry is very successful in terms of attracting customers, however, it is not the end of the story. Myopic people do not like to wear one more pair of glasses when watching 3D movies. Some people will experience discomfort after wearing 3D glasses for hours. To get rid of the undesired 3D glasses, autostereoscopic multi-view technology [8] is coming to the rescue. The two major different characteristics between stereo display and autostereoscopic display are listed in Table 1.1 [9]. The impact of available view amount for autostereoscopic display is shown in Table 1.2 [9]. Comparative ease can be brought to the 3D video audience since they do not need to wear 3D glasses for watching autostereoscopic videos. At each different view position, scenes with minor differences are available from multiple stereo pairs which are provided by autostereoscopic display [9]. As a result, when audience make moves for various view positions, scenes not viewable from the previous locations are revealed during the movements.

Table 1.1: Comparing characteristics of stereoscopic display and autostereoscopic display

Characteristic	Stereo Display	Autostereoscopic Display
Glass-Free	No	Yes
Multiple Stereo Pairs	No	Yes

Table 1.2: The impact of available view amount for autostereoscopic display

Characteristic	Small Number of Views	Large Number of Views
Seamless View Transition	No	Yes
High Quality of Scene Depth	No	Yes

The autostereoscopic multi-view display demands more than two views. With a sufficient amount of views present in autostereoscopic display, the disparities between every two adjacent views can be small enough to offer seamless transitions from scene to scene, such that when multiple views meet eyes sequentially, the scenes as a whole can be gorgeous. The visual quality of the autostereoscopic display is highly proportional to the number of available views. Due to limited available bandwidth, transmitting arbitrary number of views is not practical. Researchers have proposed a new format which only requires limited number of views and their associated depth maps for the capability of generating arbitrary amount of views. The typical system structure using this new format to compress and supply 3D video resources is shown in Figure 1.2. An enormous number of views in medium positions which are able to guarantee the high quality of 3D display can be synthesized from decoded texture frames in combination with decoded depth maps.

To employ multi-view plus depth format for 3D video, efficient compressing methods are needed, which leads to the 3D Extension of High Efficiency Video Coding Standard (3D-HEVC) by the Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) [10]. The 3D Extension of HEVC standard provides extra coding efficiency for encoding texture views along with the corresponding depth maps by using new tools. Those new tools exploit statistical redundancies between texture views and depth maps, and pay attention to the unique characteristics of depth

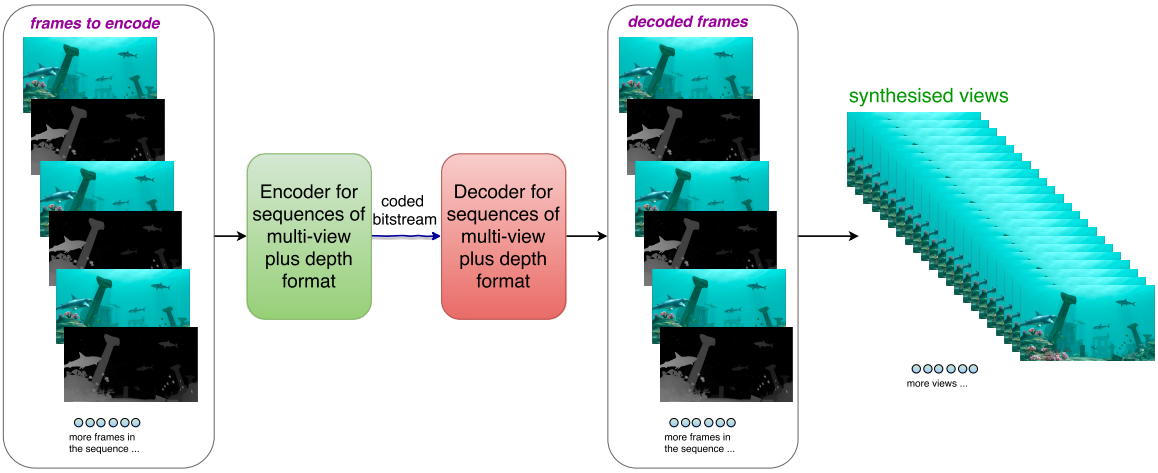


Figure 1.2: System Structure for transmitting videos of Multi-view Plus Depth format.

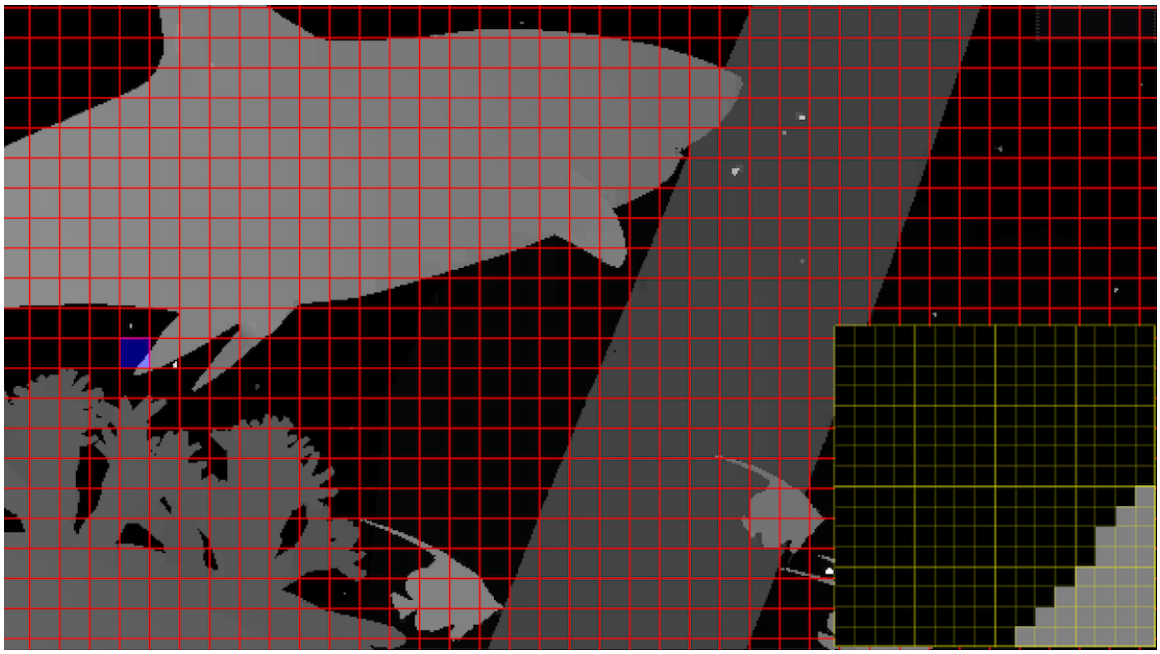


Figure 1.3: Example of wedgelet partition in a block of size 16×16 in a depth map from Shark video sequence. The tiny block highlighted by transparent blue color is magnified then shown in the right bottom corner of the depth map. Wedgelet partitions are straight lines that tries their best to fit the sharp edges in CU blocks.

maps, such as large homogeneous regions separated by sharp boundaries [11].

Depth map vividly conveys the distance between distant views and nearby views. Instead of presenting depth maps as intermediate views directly to audience, views in the medium positions are generated by Depth-Image-Based Rendering (DIBR) technique. The quality of depth maps is vital to the results produced by DIBR process. Corona artifacts (a.k.a. ringing artifacts) [9] can be discovered in synthesized views if sharpness of edge in depth maps can not be well preserved. Therefore, retaining edge sharpness in depth maps is the key to avoid the artifacts in synthesized views. There are totally 35 intra-picture prediction modes in HEVC, including DC mode, PLANAR mode and 33 Angular modes. Table 1.3 on page 6 specifies the indices and names for each of them. Figure 1.4 on page 5 illustrates the angle of each angular mode. In 3D-HEVC, all the 35 intra-picture prediction modes are retained while new intra-picture prediction modes and new residual coding methods have been applied to adapt encoding to special properties of depth maps.

Depth Modelling Mode (DMM), which is one of the new intra-picture prediction tools, is capable of conserving sharp edges in the shape of straight line. It provides a very dense set of straight partitions which will be looped through in order to find the most suitable candidate for each in the CU blocks in depth maps. Figure 1.3 presents an example of wedgelet partition in a depth map from Shark video sequence. The small block highlighted by blue color amongst the blocks separated

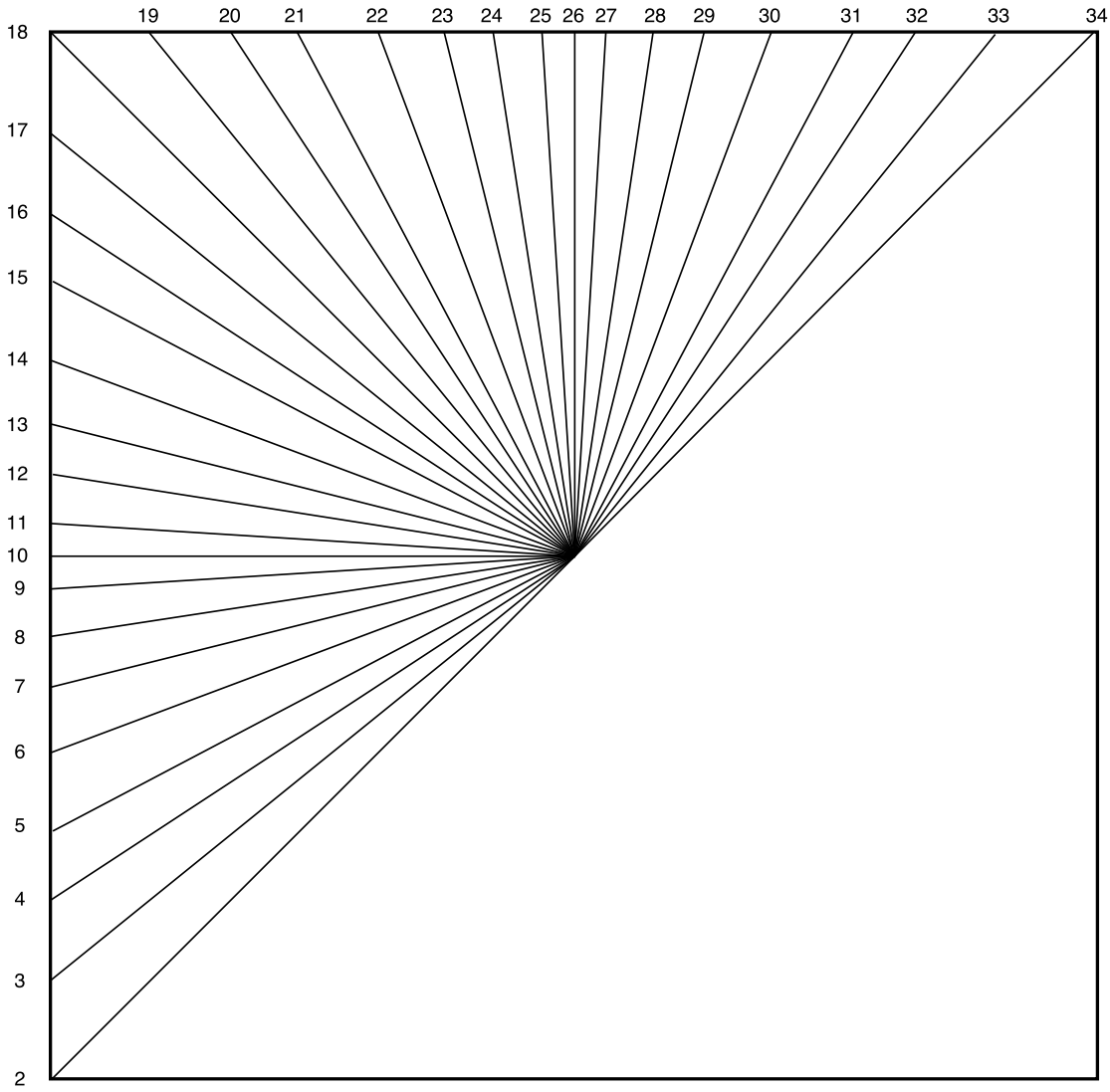


Figure 1.4: Intra angular modes in HEVC. All the angular modes are pointing to the center of the square. The modes adjacent to angular mode 10 and 26 are more dense than others to provide more flexibility for encoding vertical patterns and horizontal patterns.

Table 1.3: Indices and names for each of the intra-picture prediction mode in HEVC

Index of mode	Name of mode
0	PLANAR
1	DC
2..34	Angular (**X**), **X** = 2..34

by the red grid is magnified at the right-bottom position. Straight lines are used for the partitions in wedgelet mode. Figure 1.5 shows a sample of the contour partition from the same depth map shown in Figure 1.3. The partition pattern comprises contour line instead of straight line. Wedgelet partition and contour partition for depth maps are enabled by DMM1 and DMM4 separately.

1.1 Motivation

The idea of this work originates from the discovery of computational complexity inside the process for finding the best wedgelet in DMM1. The immense intricacy for searching the best wedgelet candidate results in a massive increase of encoding time. The time consumed for compressing a single depth map in 3D-HEVC encoder is roughly a sixfold increase relevant to the encoding time of a single texture frame wherein All-Intra configuration in $HTM16.2$ is used. To reduce the heavy time required in depth map coding, a computational model, which is a deep convolutional neural network, has been designed and trained to predict the most probable angular mode for each CU block. The predicted angular mode is utilized to further predict the most probable wedgelet candidates. The learned models exhibit 91.9% to 97.0% top-15 precision for various block sizes. It has been integrated into the reference software $HTM16.2$ of 3D-HEVC. The learned models can reduce roughly half of the wedgelet candidates. It provides 64.6% time reduction in average while the BD performance has a negligible decrease comparing with the original implementation of 3D-HEVC encoder.

Motivation for Wedgelet Candidates Reduction: The time consumed by the encoder from $HTM16.2$ for each view can be observed from command line outputs. Figure 1.6 shows a piece of command line outputs from the encoding process of Shark sequence. The numbers in red blocks stands for the encoding time of certain views, while the corresponding layer Id and Picture Order Count (POD) are in the green blocks. A repetitive pattern of the encoding time for each view can be observed every six numbers vertically. A simple calculation using six numbers within the top-most red block, $(90 + 69 + 69)/(90 + 69 + 69 + 14 * 3) \approx 0.84$, shows that approximately 84% total encoding time is busy with encoding depth maps. Similarly, it is reported in [12] that the encoding for depth maps consumes nearly 86% total 3D-HEVC encoding time. A trial of time profiling for 3D-HEVC encoder is performed using Instruments which is an application available on macOS. After encoding the Newspaper sequence for more than one hour, Figure 1.7 clearly shows

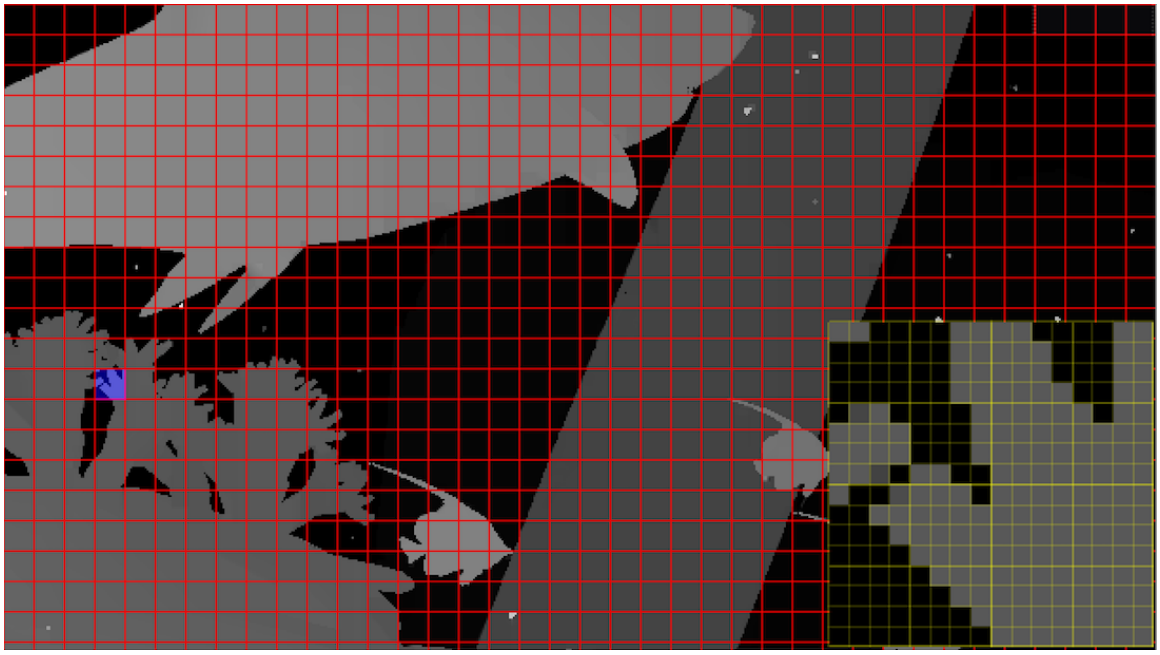


Figure 1.5: Example of contour partition in a block of size 16×16 in a depth map from Shark video sequence. The tiny block highlighted by transparent blue color is magnified then shown in the right bottom corner of the depth map. Contour partitions are irregular lines that tries their best to fit the sharp edges in CU blocks.

```
Layer   0   POC   143 TId: 0 ( I-SLICE, nQP 25 QP 25 )     932224  bits [Y  41.6950 dB    U  44.3646 dB    V  45.2432 dB] [ET    14 ] [L0 ] [L1 ]
Layer   1   POC   143 TId: 0 ( I-SLICE, nQP 34 QP 34 )      60160  bits [Y  45.0108 dB    U   0.0000 dB    V   0.0000 dB] [ET    89 ] [L0 ] [L1 ]
Layer   2   POC   143 TId: 0 ( I-SLICE, nQP 25 QP 25 )     932592  bits [Y  41.7195 dB    U  44.3713 dB    V  45.2450 dB] [ET    14 ] [L0 ] [L1 ]
Layer   3   POC   143 TId: 0 ( I-SLICE, nQP 34 QP 34 )      59616  bits [Y  44.0645 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   4   POC   143 TId: 0 ( I-SLICE, nQP 25 QP 25 )     932312  bits [Y  41.7154 dB    U  44.3855 dB    V  45.2770 dB] [ET    14 ] [L0 ] [L1 ]
Layer   5   POC   143 TId: 0 ( I-SLICE, nQP 34 QP 34 )      61920  bits [Y  44.1105 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   0   POC   144 TId: 0 ( I-SLICE, nQP 25 QP 25 )     933808  bits [Y  41.7343 dB    U  44.3498 dB    V  45.2481 dB] [ET    14 ] [L0 ] [L1 ]
Layer   1   POC   144 TId: 0 ( I-SLICE, nQP 34 QP 34 )      62288  bits [Y  44.6998 dB    U   0.0000 dB    V   0.0000 dB] [ET    90 ] [L0 ] [L1 ]
Layer   2   POC   144 TId: 0 ( I-SLICE, nQP 25 QP 25 )     932096  bits [Y  41.7303 dB    U  44.3655 dB    V  45.2215 dB] [ET    14 ] [L0 ] [L1 ]
Layer   3   POC   144 TId: 0 ( I-SLICE, nQP 34 QP 34 )      61496  bits [Y  43.9290 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   4   POC   144 TId: 0 ( I-SLICE, nQP 25 QP 25 )     931432  bits [Y  41.7151 dB    U  44.4146 dB    V  45.2913 dB] [ET    14 ] [L0 ] [L1 ]
Layer   5   POC   144 TId: 0 ( I-SLICE, nQP 34 QP 34 )      62440  bits [Y  43.7962 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   0   POC   145 TId: 0 ( I-SLICE, nQP 25 QP 25 )     927656  bits [Y  41.7556 dB    U  44.3820 dB    V  45.2680 dB] [ET    14 ] [L0 ] [L1 ]
Layer   1   POC   145 TId: 0 ( I-SLICE, nQP 34 QP 34 )      64216  bits [Y  44.6256 dB    U   0.0000 dB    V   0.0000 dB] [ET    90 ] [L0 ] [L1 ]
Layer   2   POC   145 TId: 0 ( I-SLICE, nQP 25 QP 25 )     928776  bits [Y  41.7512 dB    U  44.4091 dB    V  45.3110 dB] [ET    14 ] [L0 ] [L1 ]
Layer   3   POC   145 TId: 0 ( I-SLICE, nQP 34 QP 34 )      62608  bits [Y  44.0115 dB    U   0.0000 dB    V   0.0000 dB] [ET    68 ] [L0 ] [L1 ]
Layer   4   POC   145 TId: 0 ( I-SLICE, nQP 25 QP 25 )     929248  bits [Y  41.7290 dB    U  44.3781 dB    V  45.2933 dB] [ET    14 ] [L0 ] [L1 ]
Layer   5   POC   145 TId: 0 ( I-SLICE, nQP 34 QP 34 )      65928  bits [Y  44.0145 dB    U   0.0000 dB    V   0.0000 dB] [ET    70 ] [L0 ] [L1 ]
Layer   0   POC   146 TId: 0 ( I-SLICE, nQP 25 QP 25 )     928136  bits [Y  41.7692 dB    U  44.4027 dB    V  45.2841 dB] [ET    14 ] [L0 ] [L1 ]
Layer   1   POC   146 TId: 0 ( I-SLICE, nQP 34 QP 34 )      62480  bits [Y  44.4089 dB    U   0.0000 dB    V   0.0000 dB] [ET    89 ] [L0 ] [L1 ]
Layer   2   POC   146 TId: 0 ( I-SLICE, nQP 25 QP 25 )     923304  bits [Y  41.7896 dB    U  44.3323 dB    V  45.2464 dB] [ET    14 ] [L0 ] [L1 ]
Layer   3   POC   146 TId: 0 ( I-SLICE, nQP 34 QP 34 )      61344  bits [Y  43.5797 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   4   POC   146 TId: 0 ( I-SLICE, nQP 25 QP 25 )     927120  bits [Y  41.7560 dB    U  44.3389 dB    V  45.2676 dB] [ET    14 ] [L0 ] [L1 ]
Layer   5   POC   146 TId: 0 ( I-SLICE, nQP 34 QP 34 )      65248  bits [Y  43.6238 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   0   POC   147 TId: 0 ( I-SLICE, nQP 25 QP 25 )     918384  bits [Y  41.7766 dB    U  44.4255 dB    V  45.2442 dB] [ET    14 ] [L0 ] [L1 ]
Layer   1   POC   147 TId: 0 ( I-SLICE, nQP 34 QP 34 )      64480  bits [Y  44.3403 dB    U   0.0000 dB    V   0.0000 dB] [ET    90 ] [L0 ] [L1 ]
Layer   2   POC   147 TId: 0 ( I-SLICE, nQP 25 QP 25 )     926144  bits [Y  41.8009 dB    U  44.3506 dB    V  45.2566 dB] [ET    14 ] [L0 ] [L1 ]
Layer   3   POC   147 TId: 0 ( I-SLICE, nQP 34 QP 34 )      64000  bits [Y  43.5365 dB    U   0.0000 dB    V   0.0000 dB] [ET    69 ] [L0 ] [L1 ]
Layer   4   POC   147 TId: 0 ( I-SLICE, nQP 25 QP 25 )     923752  bits [Y  41.7702 dB    U  44.4198 dB    V  45.3113 dB] [ET    14 ] [L0 ] [L1 ]
Layer   5   POC   147 TId: 0 ( I-SLICE, nQP 34 QP 34 )      64664  bits [Y  43.6823 dB    U   0.0000 dB    V   0.0000 dB] [ET    70 ] [L0 ] [L1 ]
```

Figure 1.6: An example showing a piece of the command line outputs during the encoding process for Shark sequence.

Table 1.4: The percentages that DMM1 searching time has in total time occupied by **xCompressCU**

size	xCompressCU	Percentage
32×32	**XC2**	30.0%
16×16	**XC3**	25.6%
8×8	**XC4**	18.8%

97.8% time is used to compress the CUs recursively. The first recursive **xCompressCU** function (denoted as **XC1** thereafter) is for CUs of size 64×64, the second one (denoted as **XC2** thereafter) is targeting CUs of size 32×32, the third one (denoted as **XC3** thereafter) is dedicated to CUs of size 16×16, and the last one (denoted as **XC4** thereafter) is bound to CUs of size 8×8. It can observed from Figure 1.7 that the most time consuming part during the process of compressing depth CUs is DMM1 searching. The time percentage that DMM1 searching time has in total time occupied by **xCompressCU** is summarized in Table 1.4 wherein the summary for **XC1** is omitted since DMM1 is not applicable to CUs of size 64×64 in $HTM16.2$. [t] The major reason leading to the time consuming property of DMM1 searching is the View Synthesis Optimization (VSO) Method for improving quality of synthesized views [13], in which the Synthesized View Distortion Change (SVDC) is computed. The time percentage that VSO has in DMM1 searching are summarized in Table 1.5. In $HTM16.2$, a large number of wedgelet candidates are evaluated using VSO which introduces computational complexity. Intuitively, evaluating less wedgelet candidates can help with relieving the heavy burden of computation bared by encoder, thereby time reduction can be achieved.

Motivation for Using Deep Learning: Deep learning is a sub field of representation learning, which is in turn a major subset of machine learning [14]. Machine learning [15] has been applied to many scenarios in the domain of Artificial Intelligence (AI). Deep learning was found hard to proceed

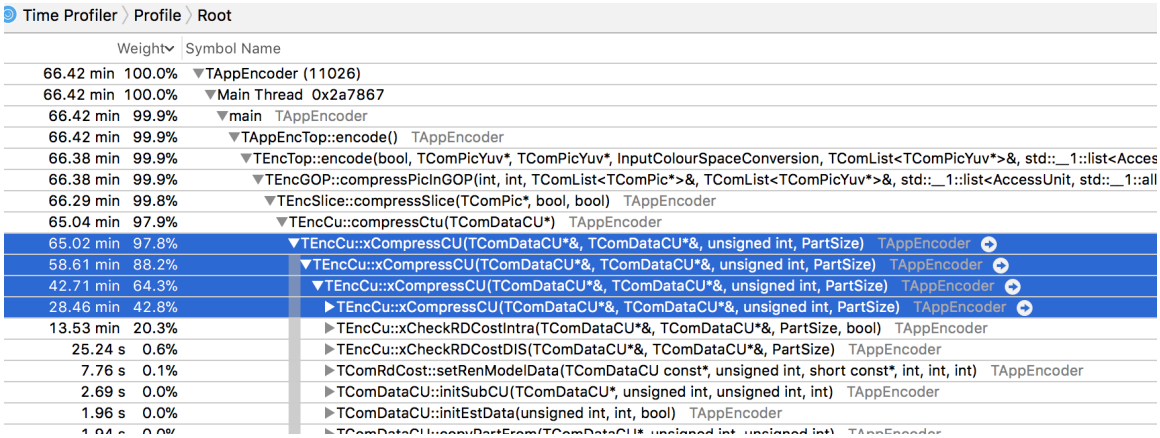


Figure 1.7: A screen capture of the time profiling information for Newspaper sequence.

Table 1.5: The time percentage that VSO has in DMM1 searching

size	Process	Percentage
32×32	VSO in DMM1 searching from **XC2**	80.1%
16×16	VSO in DMM1 searching from **XC3**	83.7%
8×8	VSO in DMM1 searching from **XC4**	78.8%

further in the late 1980s [16]. However, starting from 2012, it kicks off its glorious comeback. The deep Convolutional Neural Network (CNN) has won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) from 2012 to 2015, with the CNN architecture going deeper and deeper. The great achievements have attracted attentions from people all over the world and have made deep learning the most popular topic in our daily lives. Inspired by the fact that supervised deep learning can learn multiple layers of abstract representations in the visual recognition tasks, it should be applicable to recognize the angular modes of the intra-picture prediction in 3D-HEVC. The final DMM1 candidates selected in depth map coding are essentially determined by the angle pattern of depth blocks. If we can make use of deep learning to predict the most probable angles of the target pixel block, a large amount of angular modes and DMM1 wedgelet candidates can be naturally skipped by which the time saving can be achieved without sacrificing much of the coding performance.

Motivated by the discussions above, we adopt deep learning approach with deep convolutional neural network to accelerate depth map coding in 3D-HEVC.

1.2 Contribution and Dissertation Outline

We accelerate depth map coding using deep learning. The contributions of the dissertation are:

- A deep convolutional neural network with 32 layers comprising ResNet units [17] has been designed and trained to recognize the most probable angular mode of coding units (CUs) in intra-picture prediction in 3D-HEVC encoder. Each learned model has high top-15 precision which works well on tasks of recognizing intra angular patterns in 3D-HEVC.

- A way of integrating the learned model into $HTM16.2$ encoder has been suggested. By making use of Bazel [18] to compile the encoder binary, the data level parallelism (instead of concurrency) functionality in CPU as well as the parallel architecture in GPU are fully utilized for efficient matrix computations.

- An algorithm for fast depth map coding has been proposed and implemented. The simulation results show that the proposed algorithm is capable of reducing 64.6% of encoding time in wedgelet searching during 3D-HEVC encoding process while the BD performance only has a trivial decrease.

The first two contributions lay the foundation for the third one, which is the main objective of this work: to accelerate the depth map encoding process in 3D-HEVC.

Chapter 2 supplies the background of video coding history, video coding standards, and deep learning using artificial neural networks. Prior arts in video coding are surveyed in this chapter.

Chapter 3 describes the algorithm which has been implemented to collect the data to be used in deep learning. Data pre-processing steps are described in detail along with the reasons behind the scene. A rich set of visualizations are presented to illustrate the properties of the collected data.

Chapter 4 presents a deep convolutional neural network which has been applied in our deep learning. Both architecture and settings are covered in detail. The stopping criteria are shown with the training results. At the end of the chapter, the evaluation results for learned models are presented.

Chapter 5 shows the methods used to integrate learned models into the reference software of 3D-HEVC. The problem encountered during the integration is described, and the solution for the problem is presented. Simulation results comparing with the original $HTM16.2$ are given in this chapter.

Chapter 6 concludes the thesis.

Chapter 2

Background

To start with, we bring up what is video coding, why it is needed, and its challenges. Next we discuss what is deep learning, the history of deep learning and how it works for vision tasks. Furthermore, we introduce how we plan to apply deep learning to optimize video coding and why it should work. Lastly, a survey of related works on the topic of video coding is presented.

2.1 Video Coding

Video playback is the most straightforward way for human to perceive dynamic scenes which exist across a period of time. More than half of the neurons in human brains are born to process the visual information supplied by human eyes. It becomes effortless for human to know what is going on in video playback. Videos are made up of consecutive sets of image frames, which in turn are made up of pixel matrices.

Most of the time videos need to be stored for future use. In 1950s, video tapes were employed to store videos. Video tape is able to serve eight to twelve years before the quality of the stored video starts to degrade. In 1970s, laser disc appeared in the market as an alternative of video tapes. Start from laser disc, the video storage started its new era in digital world. In 1990s, DVDs were released into the market. Data is stored in spiralling tracks on the disc. A laser beam can be utilized to read the data. In addition, hard drives, flash drives and SD cards were also starting to become popular in the late 90s. Nowadays, the cloud storage is very common in daily lives. It is capable of storing data on the servers which are accessible from any devices via internet connections.

Although so many formats are available for video storage, they share a common feature: the more storage you use, the more extravagant it will be. Taking the cloud storage as an example, Google Cloud is one of the most popular cloud services. It provides cloud storage with a price of \$0.026 per GB/month [19] (this price appears on 21 Nov 2017, it may change in the future). If there is a video (120 frames per second, 4096×2160 resolution, 8 bits per RGB component) to be stored

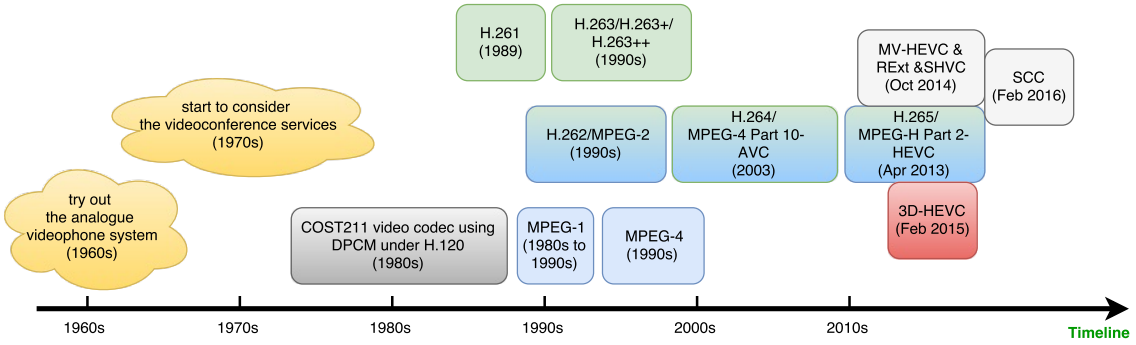


Figure 2.1: The brief history of the video coding standards

without any compression in Google Cloud, we need to pay a monthly fee of $(4096 \times 2160 \times 120 \times 60 \times 90 \times 3 \times 0.026)/(1024 \times 1024 \times 1024) \approx 416.47$ \$. Without doubt, this figure is relatively not acceptable for the purpose of video storage. High compression is needed to store the videos in a practical way.

From the other perspective, let us take the bandwidth into consideration. To deliver the uncompressed 4K video which has been mentioned in the previous paragraph, we need a bandwidth of $(4096 \times 2160 \times 120 \times 3)/(1024 \times 1024 \times 1024) \approx 2.97$ Gigabytes per second. The maximum bandwidth of Wireless 802.11ac, which is one of the common internet access technologies, is 1.3 Gigabytes per second [20]. Apparently, the wireless connection is not able to deliver such kind of 4K videos. High compression is desired to deliver the video on the internet.

Despite the fact that raw videos usually contain a large amount of data, a lot of redundancies exist. For every video sequence, two types of redundancies are ubiquitous: spacial redundancy and temporal redundancy. Video coding technologies are taking advantages of those redundancies to achieve efficient compression for video data. Many of the useful video coding technologies have been adopted by the international video coding standards, such as MPEG-4, H.264, H.265, etc.

Figure 2.1 shows the brief history of the video coding standards. In 1980s, the COST211 video codec, which was built on top of Differential Pulse Code Modulation (DPCM), was standardized under H.120 standard by CCITT (now known as ITU-T). In late 1989, the H.261 was completed and its success marked a milestone for video coding at low bit rate with fairly good quality [21]. The Motion Picture Experts Group (MPEG) kicked off the exploration of video storage, such as CD-ROMs. Their objective was to achieve a competitive performance with cassette recorders in terms of compression of videos wherein rich motions can be found. The framework of H.261 was used to start the codec design of MPEG-1. MPEG-2 is the successor after MPEG-1. It features high capabilities when handling video with high bitrate and high resolution. In MPEG-2, the encoder is allowed to make its own decision on the the number of bi-directionally predicted pictures according to a suitable coding delay. ITU-T found this technique applicable to telecommunication applications, as a result

MPEG-2 has been adopted as H.262 for telecommunications. Right after the MPEG-2 standard, MPEG-3 was designed mainly for coding of high definition videos. However, MPEG-3 was discarded due to the versatility of MPEG-2, which can be used to encode video of any resolution. In the late 1998, MPEG-4 was introduced as a standard to define compression of both audio and visual digital data. Later on MPEG-4 was divided into several parts during its continuously evolving. Among its sub-parts, MPEG-4 part 10 (a.k.a. Advanced Video Coding) is mainly for video compression. With the rising popularity of high definition videos, the new standard termed High Efficiency Video Coding (HEVC) for compressing videos in a more efficient way comparing with previous standards, such as H.264/AVC, has emerged under the efforts from the Joint Collaborative Team on Video Coding (JCT-VC). In the meanwhile, five extensions of the HEVC standard, comprising Format Range Extension (REext), Scalability Extension (SHVC), Multi-view Extension (MV-HEVC), 3D Extension (3D-HEVC), and Screen Content Coding Extension (SCC), have been finalized from 2014 to 2016 for fulfilling extra requirements in various video coding scenarios.

In this work, we focus on depth map coding in 3D-HEVC. DC mode, PLANAR mode, 33 angular modes and depth modeling modes have been embraced in depth map coding tools in 3D-HEVC. The DMM1 mode introduces a huge increase for the encoding time of 3D videos. Acceleration of depth map coding is needed.

2.2 Deep Learning

Deep learning is one of the approaches for representation learning (a.k.a. feature learning), which is essentially a method to learn from data. Numerous layers of computational units together with appropriate activating mechanism comprise the basic architecture for deep learning. Multitudinous data sets are needed for those computational architectures to learn data abstractions for tasks such as image classification, speech recognition, object detection, etc. Each layer learns a level of abstraction from the data sets using back-propagation algorithm [22]. Making use of those learned abstractions, the computational models are able to solve complicated problems which are typically non-linear and normally hard to solve by using specific rules that are designed by human experts in advance.

Deep learning has attracted wide attention from all over the world in recent years, not only because of the great achievements it has made in various application scenarios, but also due to the promise of an intelligent future it gives. Such a learning methodology makes people believe that the formation of wise machines, which they have long dreamed to possess, is possible. The growing data accessibility provides rich datasets for computational models to adjust their internal weights and bias until their predictions will have low error rate. On the other hand, the devices for computation are relatively affordable than in the previous years. With the help of those devices, accelerations of deep learning have been achieved. Consequently a bunch of time consuming deep learning architectures can be explored within an acceptable range of time.

In the ILSVRC-2012 competition [23], AlexNet [24] received the championship with the 15.3% top-5 error rate comparing to 26.2% error rate achieved by the runner-up. Such a large margin of error rate claimed a breakthrough in object recognition history. Since then, both academia and industry started to try out deep learning at a blistering pace, which in turn result in an increase of the submissions to ILSVRC-2013, in which ZF Net [25] was the winner. It fine-turned the architecture of AlexNet and gorgeous visualizations of trained models were presented. Both AlexNet and ZF Net are of the same structure which is built up by simply stacking computational layers while GoogLeNet [26] is composed of Inception modules. This new architecture was the most successful candidate in ILSVRC-2014. It not only reached a new height of object recognition but also started the optimization of the computational resources inside network by design. It consists of 22 layers, which was deeper than all the previous networks in ILSVRC. However, it was still not deep enough. In ILSVRC-2015, Residual Neural Network (ResNet) [17] with 152 layers won the championships in all the five main tracks. ResNet introduced a brand new notion into the neural network architecture named identity mapping. It is implemented by shortcut connections which prevent the degradation of training precision when the network goes deeper. Besides, the converging speed of ResNet is faster than the network built up with Inception modules when both contain similar size of learnable parameters.

Despite the fact that neural networks built up from Inception modules converge slower than those built up from ResNet modules, it is still worth it for a brief review of the valuable insights residing in the Inception networks. A typical incarnation of the first generation of Inception networks is named GoogLeNet [26]. It was intricately carved with a responsibility to win computer vision tasks in ILSVRC-2014, on which it performed better than all the other deep neural network architectures. The philosophical reflections, which intended to serve as guidelines for the construction of Inception networks, are highlighted below. Two major downsides of enlarged neural networks have been discussed in [26]. One is the higher chance of overfitting while the other is the strikingly increased requirements of computational resources. To overcome those drawbacks, based on the new ideas about how to construct reasonable architectures of neural networks in [27], new experiments orienting sparse network architecture have been conducted. One year later after GoogleNet hold the championship of ILSVRC-2014, an approach named Batch Normalization [28] was proposed by Google researchers to accelerate and ease the training of deep neural networks. The core idea behind Batch Normalization is to normalize the inputs to each layer for every batch of training data. More importantly, normalization essentially is matrix multiplications followed by adding biases. Due to this observation, the Batch Normalization, which is implemented as additional layers, has been made part of the network architecture. This rather novel method started a new chapter for the training of deep neural networks. With the adoption of Batch Normalization, higher learning rates no longer impede the convergence of the deep networks, oppositely faster training speed, which can achieve a better accuracy of prediction with considerably less time, is brought to scene. Additionally, in

some cases, it can even replace the Dropout [29] which is an effective method to prevent overfitting. The incorporation of Batch Normalization into the first generation of Inception network architecture result in the formation of Inception-v2 which improved the best accuracy on ImageNet classification with less training steps. In the same year, Inception-v3 [30] joined the party, the objective of which was to effectively leverage the power of additional computation by factorizing to smaller size convolutions and regularizing the classifier layer with the estimation of minor effect of label-dropout in the training process. The network architectures were scaled up in Inception-v3, which consequently imposed higher requirements of available computational resources. With the ResNet [17] stealing the show in ILSVRC-2015, the influence of the identity mapping in residual units on learning process has been investigated in [31]. The filter concatenation stage of Inception-v3 was replaced using shortcut connections which effectuated the layout of a new model named Inception-ResNet-v1. A more advanced version, which was named Inception-ResNet-v2, has a larger network size than the first version. Besides the mixed architectures of Inception-ResNet, a pure Inception incarnation named Inception-v4 was also presented with comparison to Inception-ResNet-v2. Both Inception-v4 and Inception-ResNet-v2 have significant gain of performance mainly benefiting from the enlarged size of network.

2.3 Related Work

In this section, the prior arts working on video coding optimizations are reviewed. The differences between this work and others are compared.

Before the occurrence of Depth Modeling Mode (DMM) and View Synthesis Optimization (VSO) in [32], a lot of literature on depth map coding, which have been published, are mainly focusing on improving the effectiveness of depth map coding. Based on the observation that the depth map is characterized by vast smooth regions, which are separated by sharp edges, an algorithm to effectively encode homogeneous regions has been proposed in [33]. It improves coding performance for depth map by copying pixel values for homogeneous blocks from values of neighboring reference pixels. In [34], Depth Lookup Table (DLT) has been proposed for encoding the depth map in the 3D-HEVC standard. It offers the benefit of 1.3% bitrate reduction. To further improve the coding performance for depth map, more effectual tools for depth map coding are needed. Depth Modeling Mode (DMM) and View Synthesis Optimization (VSO) are proposed in [32]. VSO provides 17% bitrate reduction in average, while DMM provides 6% average savings on bitrate. Although the introductions of DMM and VSO bring the effectiveness of depth map coding into a new level, the computational complexity increases a lot due to the complex nature of VSO. Consequently, the time cost of depth map encoding becomes fairly expensive.

The computational complexity of depth map coding raised the question of whether it is possible to reduce the computational complexity for saving encoding time. In [35], a fast wedgelet searching

scheme achieves significant reduction for computational complexity with minor decrease on BD performance. It takes advantage of the results from Sum of Absolute Transform Difference (SATD) to reduce the wedgelet searching candidates. Rough RD cost from Rough Mode Decision (RMD) is used as mode selection threshold in [36] to speed up the bi-partition modes decision. A two-step fast searching approach for wedgelet partition appears in [37]. It features a coarse search in conjunction with a further refinement step. Another fast approach for wedgelet searching [38] is to make use of Most Probable Mode (MPM) to reduce wedgelet searching candidates. Since intra angular modes will lead to ringing artifacts when they are utilized for depth map coding, the idea of skipping intra angular prediction by making use of edge detector is shown in [39]. Bayesian classifier is used in [40] to alleviate the computational complexity of intra mode decision in 3D-HEVC. The optimal mode of parent prediction unit (PU) in the hierarchical quad-tree coding structure has been utilized to select the mode for child prediction unit (PU) in [41], and early decision for segment-wise DC coding is used together to achieve faster depth intra coding. Edge classification in Hadamard transform domain is used in [42] to skip the DMM decision process conditionally. The minimum RD cost of the candidate from full-RD searching list is taken as the threshold to bypass DMM decision in [43]. Most probable region for DMM1 mode decision is identified with the help of sharp edges in [44], and DMM3 is skipped when depth prediction unit (PU) does not match with co-located texture counterpart. Variance is utilized in [45] to estimate the most promising sub-region for DMM1. Corner point is used for fast quad-tree decision of depth intra coding in [46]. Variance distribution is studied in [12], based on which the method termed Squared Euclidean distance of variances (SEDV) is proposed to substitute the long-standing View Synthesis Optimization (VSO) process. Besides, a new scheme termed probability-based early depth intra mode decision (PBED) is employed to skip modes and the RD cost in Rough Mode Decision (RMD) is used to terminate segment-wise depth coding (SDC) [34] as early as possible. The correlation between depth map and texture view is explored in [47] to palliate the sophistication of compression for depth map. In [48], comparing RD cost with pre-calculated threshold for fast intra mode decision together with early decision for the CU depth are used to accelerate the encoding. Making use of RD cost results of the angular modes, only the most promising DMMs are evaluated in [49] and, moreover, an innovative method using golden ratio to further improve the depth map coding is proposed. The characteristics of depth map are studied in [50], as a result only four conventional intra modes are used for depth map intra coding and only six directions are used in DMM1 searching. Block edge along with the border gradient are used together in [51] to surge the speed of depth map coding. Information of neighboring blocks and threshold which is derived from lots of experiments are used in [52] for improving depth map coding.

Despite the aforementioned works which are using heuristic approaches, machine learning approaches are also applied for video coding optimization. In [53], the decision for the depth of coding units (CUs) in High Efficiency Video Coding (HEVC) is modeled as a classification problem which

can be solved by machine learning approach. A shallow convolutional neural network (CNN) is used in [54] to determine coding unit depth in High Efficiency Video Coding (HEVC) while in [55], a deeper convolutional neural network together with long- and short-term memory (LSTM) network are employed to address the same issue. In additional to the works which are targeting the coding unit depth decision using machine learning approaches, it is found in [56] that deep learning is used for the intra mode selection in Screen Content Coding (SCC) Extension of High Efficiency Video Coding (HEVC).

The researching focus in this thesis is the same as [56]. However, there are three significant differences between two works. Firstly, the network in this thesis, which consists of 32 layers, is much deeper than the 4-layer network in [56]. Secondly, unlike the server-client setup in [56], we have managed to integrate the learned models into the codebase of $HTM16.2$. Executable binary can be obtained which is totally self-contained in the sense that they are not relying on the remote server to do the prediction, just the binary itself is capable of doing prediction for mode selection in depth maps. Lastly, in this work, deep learning is for depth map coding in Three Dimension Extension of High Efficiency Video Coding (3D-HEVC) while in [56] the learning is for Screen Content Coding (SCC) Extension of High Efficiency Video Coding (HEVC).

Chapter 3

Prepare the Data for Deep Learning

Deep learning heavily relies on data. It only works when a considerably large set of data can be provided. Since we are using supervised learning which is the most popular form of deep learning for the time being, our datasets must be well labeled. Normally a large dataset contains enormous yet different classes, with each class has its own label. The labels are used to adjust the inner parameters of a deep model according to the loss function during the training process. We need to prepare a large set of labeled data before we can start to train the deep model. In this chapter, we start with data collection, in which data source and algorithm for collecting data are shown in detail. After that the necessary pre-processing steps for the collected data are described. Lastly plenty of visualizations for the collected raw data are presented with discussions explaining the reason for the data pre-processing.

3.1 Data Collection

To collect the data for training a deep model to predict the most probable intra angular directions for depth CUs, we need to clear up two questions:

- 1. Where does the data come from?
- 2. How to collect the data from the source?

In this section, above two questions are answered one by one.

3.1.1 Source of Data

The data are collected from four video sequences as shown in Table 3.1 on page 19. Balloons

Table 3.1: Source of data for deep learning

#	Name of the Sequence	Resolution	Number of Frames
1	Balloons	1024×768	300
2	Kendo	1024×768	300
3	Poznan Street	1920×1088	250
4	Undo Dancer	1920×1088	250

sequence and Kendo sequence are of the resolution 1024×768 while Poznan Street sequence and Undo Dancer sequence are of the resolution 1920×1088. All the available frames from above four video sequences are used to collect data. The collected data for each sequence will be separated into three sets for training, testing and validating in deep learning. The training data set is used for deep model to learn the desired representations by the back-propagation algorithm [57], during which the inner parameters, which typically are weights and bias, are moving along the gradient descent direction. After the learned model will be obtained, the validating datasets are used to fine-tune the hyper-parameters. With reasonably adjusted hyper-parameters, the training process will be performed again to obtain a better model. After certain loops of the train-validate schedule, the testing dataset will be used to evaluate the final learned model which by then shall not be further turned any more. After learned model will be applied to the testing datasets, the performance results of evaluation can indicate the generalization of the learning model.

3.1.2 Algorithm for Collecting Data

In this subsection, the algorithm used for collecting data are presented. We collect data from coding unit (CU) level by encoding four video sequences shown in Table 3.1. Most of the hybrid video coding features from HEVC remain unchanged in 3D-HEVC, including the sizes allowed for each type of block. There are totally four types of blocks, i.e., coding tree unit (CTU), coding unit (CU), prediction unit (PU) and transform unit (TU). Table 3.2 shows the allowed sizes for each type of block. A CTU itself can be used as a single CU while in some scenarios it can be split into multiple

Table 3.2: Allowed sizes of each type of block

Block Type	Allowed Sizes				
CTU			16×16	32×32	64×64
CU		8×8	16×16	32×32	64×64
PU	4×4	8×8	16×16	32×32	64×64
TU	4×4	8×8	16×16	32×32	

CUs [58]. CU sits on top the PU partition structure. The quad-tree splitting syntax allows CUs of

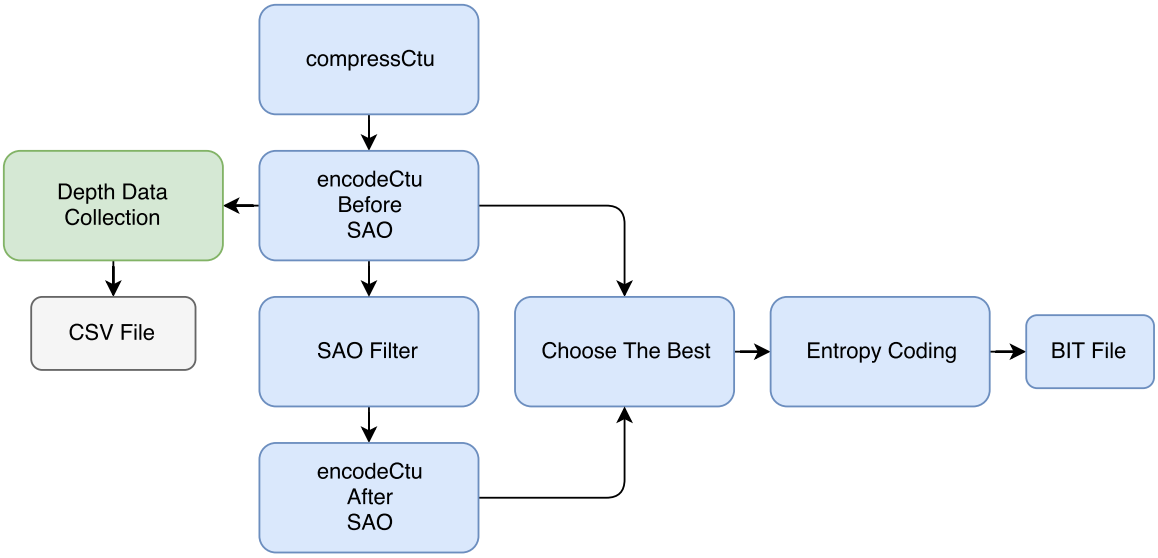


Figure 3.1: Data collecting diagram.

largest size to be further split into smaller sizes. CU can be partitioned to form TUs recursively in residual coding. The maximum coding unit (CU) size in 3D-HEVC is 64×64. The encoder selects the minimum allowed CU size based on the syntax in the sequence parameter sets (SPS). For Luma CU samples, the minimum allowed size is larger than or equal to 8×8. The encoder first needs to make the basic decision of whether to code a block with inter-picture or intra-picture prediction at CU level. After that the best mode of the intra-picture prediction is obtained at PU level. The maximum block size allowed for DMM1 is 32×32 while the minimum block size allowed is 4×4. Hence we need to collect data for each block size from 4×4, 8×8, 16×16 to 32×32. To collect data, we first need to identify to which part in the reference software we should insert the data collecting module. Since supervised learning has been chosen, labels for data samples are also required to be collected. The label is the best mode that has been chosen by $HTM16.2$ encoder. Figure 3.1 illustrates the relationships among the core modules for collecting data. The rectangular blocks with light blue background are the modules from $HTM16.2$ while others are the newly added modules. In the module of *compressCtu*, the encoder recursively invokes another module named *xCompressCtu*, which is not on the diagram, to try different kinds of CU, PU, TU and to decide the best mode for them. Once the *compressCtu* module will be finished, all the needed data can be obtained in the *encodeCtu Before SAO* module. During the data collecting process, only Luma samples in depth blocks are used since we are trying to reduce the computational complexity of depth map encoding. As shown in Figure 3.2 on page 21, the Luma samples are flattened from rectangle CU blocks into a single row which will be subsequently written into associated CSV file.

The detailed implementation for the module of *depth data collection* is shown in Algorithm 1 on

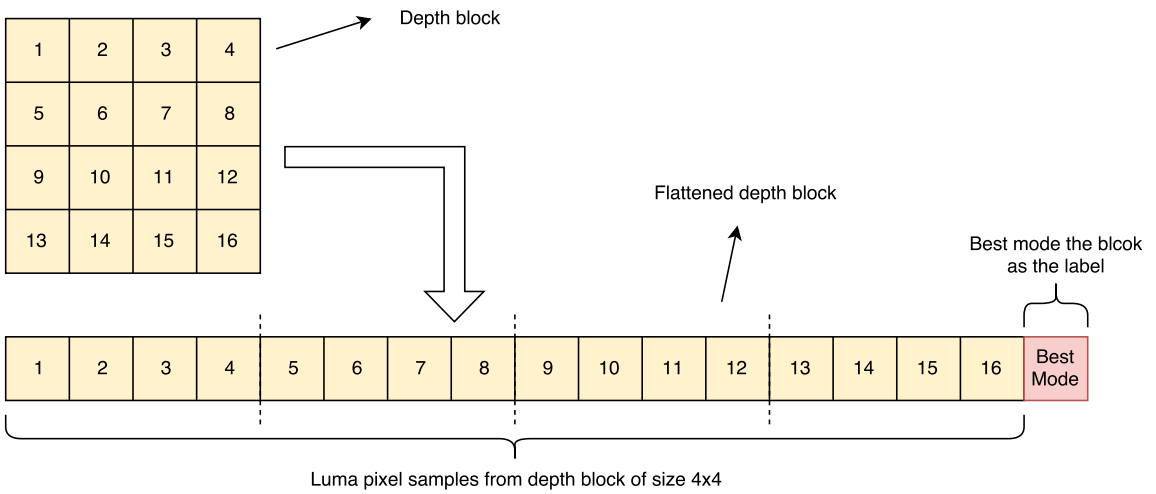


Figure 3.2: Flatten Luma samples into one line and append best mode at the end.

page 39. The inputs (i.e., the CU data structure, the absolute partition index and the corresponding quad-tree depth) to the data collecting workflow are exactly the inputs to the module of *encodeCtu*. Firstly, the partition mode of the CU, which can further indicate the partition number to either be one or four, is obtained. After that, if the partition number is equal to one, which means the current CU has not been split and it has its own best mode, the luma pixel samples of this block are collected into a single row in the CSV file. In the end of the same row, the best mode of the CU is signaled; else if the partition number is equal to four, which means the current CU has been split to form four smaller sub-blocks with each sub-block has its own best mode, the luma samples for each sub-block are collected into four different rows in the CSV file with the last value in each row to be the best mode for each sub-block. $HTM16.2$ [59] is used in this work, which is the newest version of the reference software of 3D-HEVC. Considering we are focusing on the intra prediction, All-Intra configuration is used.

3.2 Data Visualization

Visualizing data is a good way for human to better understand and memorize information. Complicated patterns hidden in the data are easier to be discovered when they are aesthetically presented via the graphical format. The collected data are visualized with the hope of discovering more information from them. Discussions based on the observations of visualized blocks, which have convinced us of the necessity of data pre-processing in Section 3.3 before training the deep models, are given.

3.2.1 Visualized Data

We have four sets of data which are from blocks of size 4×4, 8×8, 16×16 and 32×32. The intra prediction modes in 3D-HEVC include DC mode, PLANAR mode, 33 angular modes, DMM1 and DMM4. In this thesis, several indices are assigned to each mode: 0 for DC mode, 1 for PLANAR mode, [2,34] for 33 angular modes, 35 for DMM1 and 36 for DMM4.

After all the four sets of data have been visualized, it is found that four corresponding sets of visualizations give same hints which are discussed in Subsection 3.2.2. For this reason, only the visualizations of blocks of size 8×8 are shown. There are totally 37 visualizations presented, from Figure 3.3, Figure 3.4, ... to Figure 3.39.

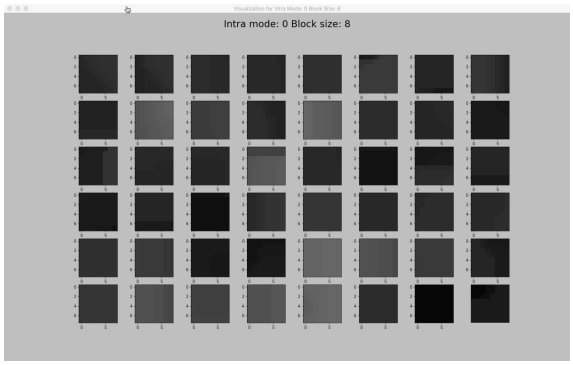


Figure 3.3: Visualizations for blocks tagged with intra DC.

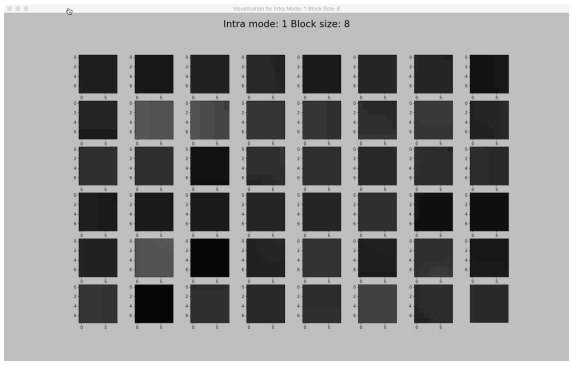


Figure 3.4: Visualizations for blocks tagged with intra PLANAR.

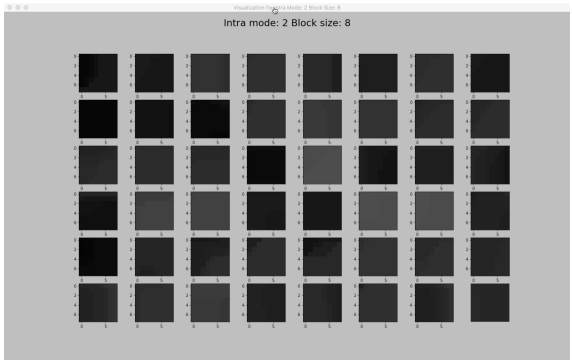


Figure 3.5: Visualizations for blocks tagged with intra mode 2.

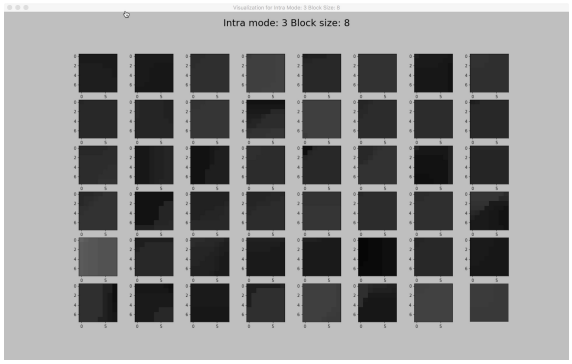


Figure 3.6: Visualizations for blocks tagged with intra mode 3.

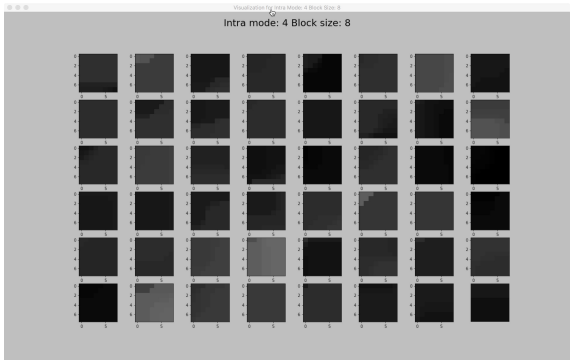


Figure 3.7: Visualizations for blocks tagged with intra mode 4.

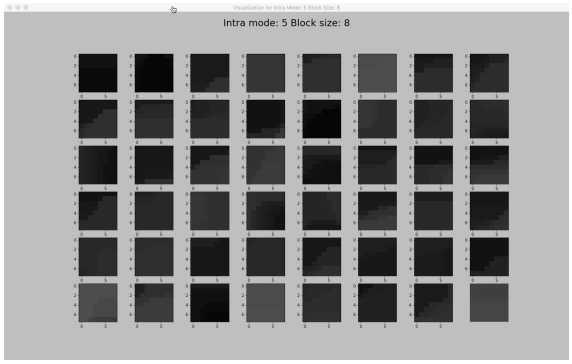


Figure 3.8: Visualizations for blocks tagged with intra mode 5.

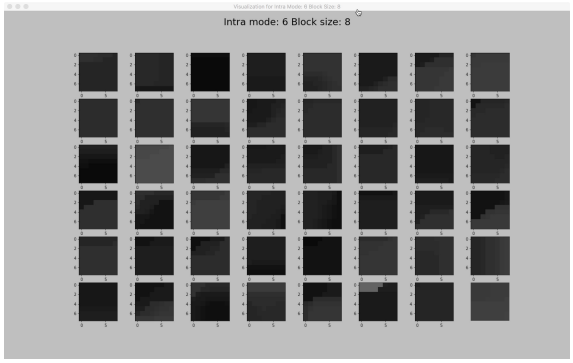


Figure 3.9: Visualizations for blocks tagged with intra mode 6.

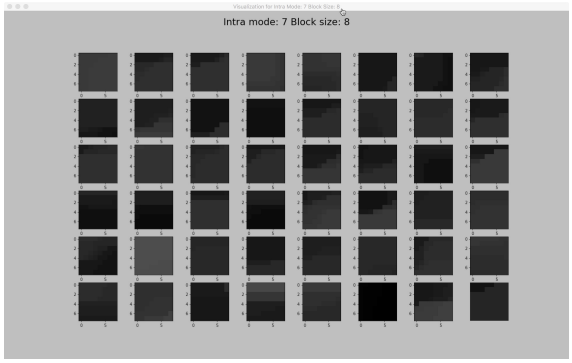


Figure 3.10: Visualizations for blocks tagged with intra mode 7.

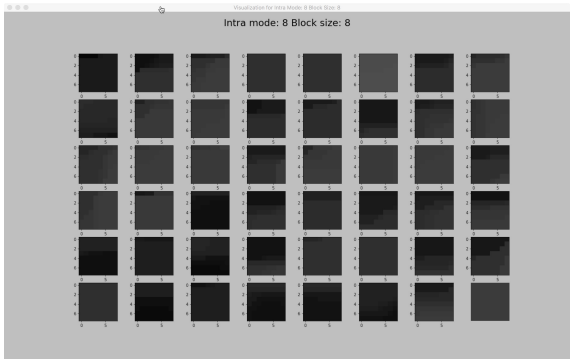


Figure 3.11: Visualizations for blocks tagged with intra mode 8.

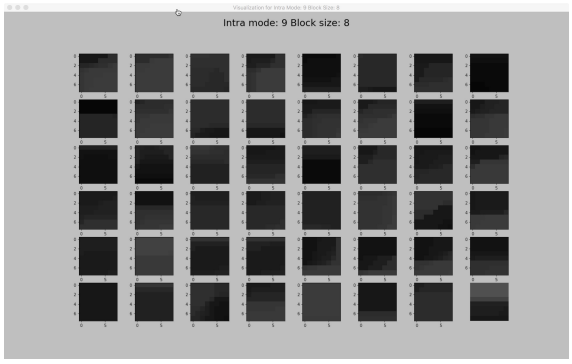


Figure 3.12: Visualizations for blocks tagged with intra mode 9.

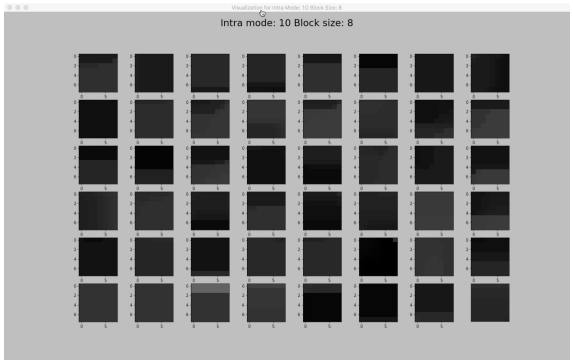


Figure 3.13: Visualizations for blocks tagged with intra mode 10.

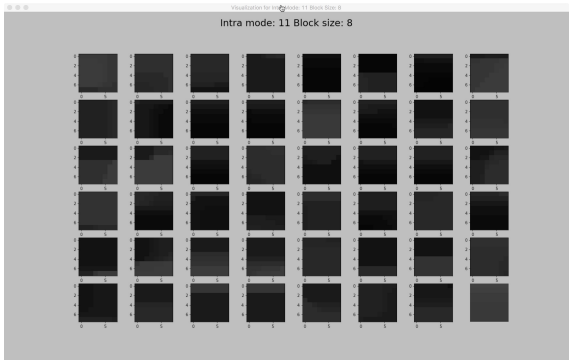


Figure 3.14: Visualizations for blocks tagged with intra mode 11.

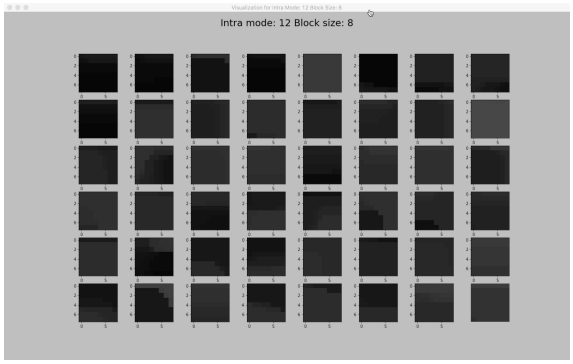


Figure 3.15: Visualizations for blocks tagged with intra mode 12.

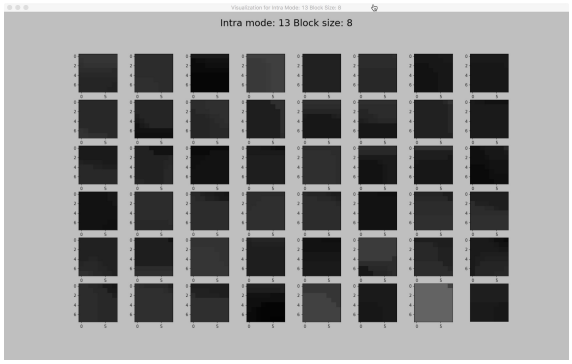


Figure 3.16: Visualizations for blocks tagged with intra mode 13.

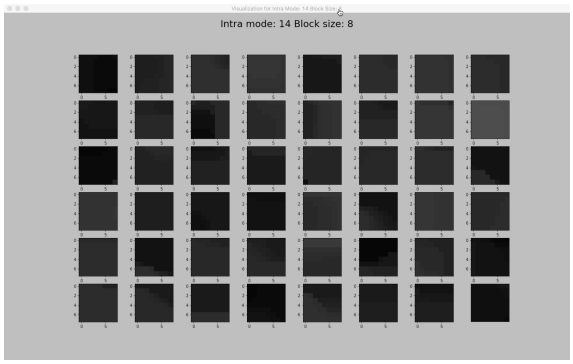


Figure 3.17: Visualizations for blocks tagged with intra mode 14.

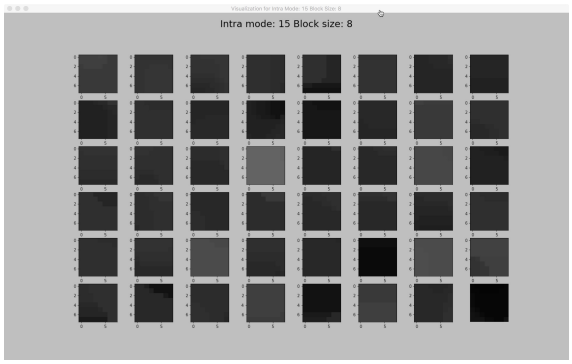


Figure 3.18: Visualizations for blocks tagged with intra mode 15.

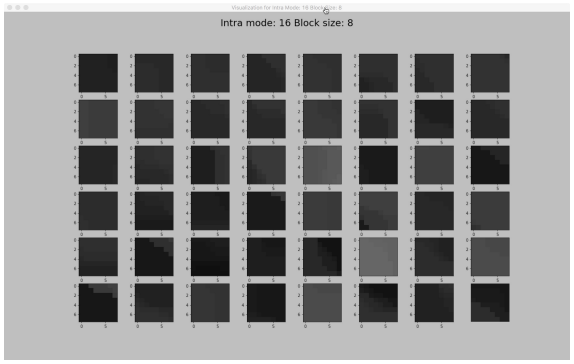


Figure 3.19: Visualizations for blocks tagged with intra mode 16.

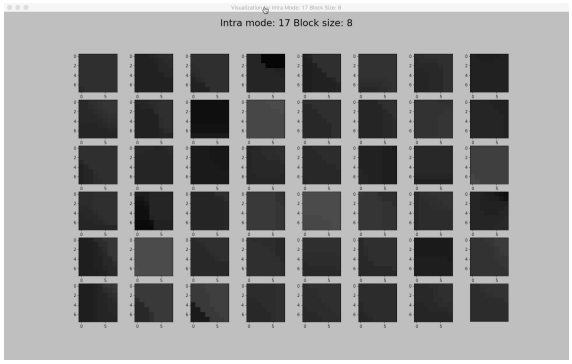


Figure 3.20: Visualizations for blocks tagged with intra mode 17.

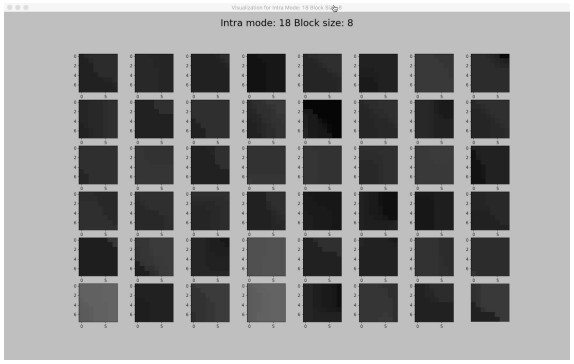


Figure 3.21: Visualizations for blocks tagged with intra mode 18.

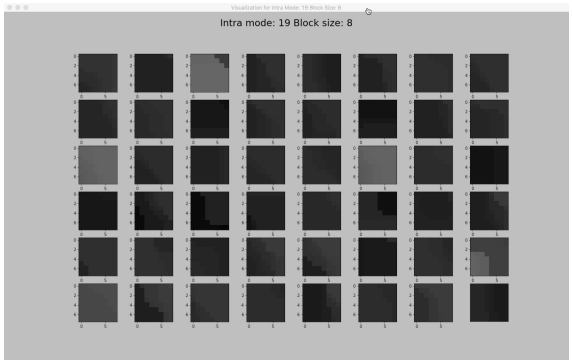


Figure 3.22: Visualizations for blocks tagged with intra mode 19.

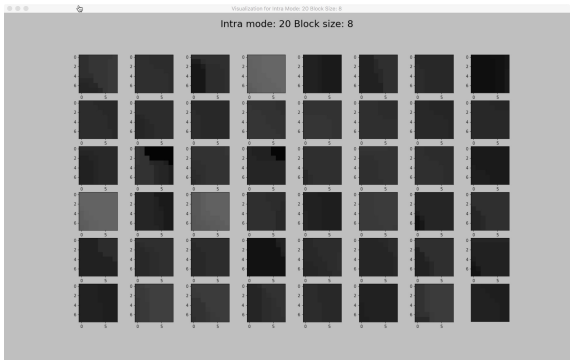


Figure 3.23: Visualizations for blocks tagged with intra mode 20.

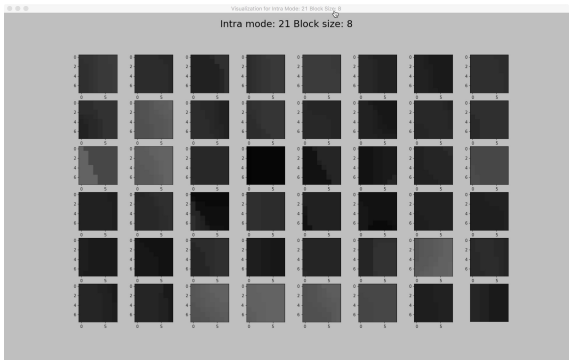


Figure 3.24: Visualizations for blocks tagged with intra mode 21.

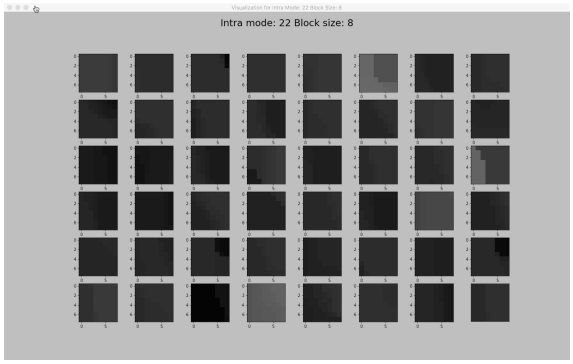


Figure 3.25: Visualizations for blocks tagged with intra mode 22.

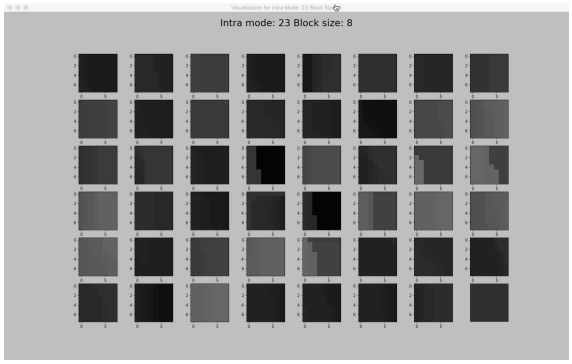


Figure 3.26: Visualizations for blocks tagged with intra mode 23.

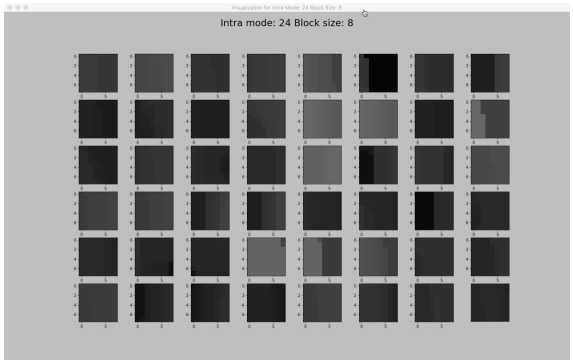


Figure 3.27: Visualizations for blocks tagged with intra mode 24.

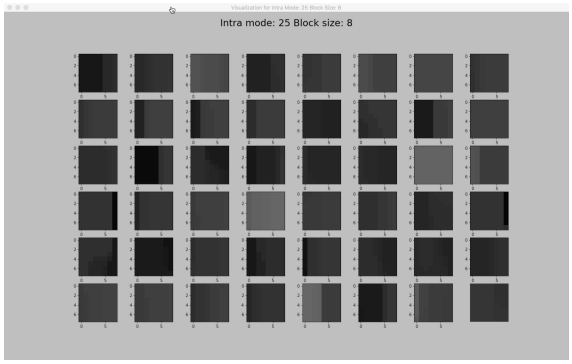


Figure 3.28: Visualizations for blocks tagged with intra mode 25.

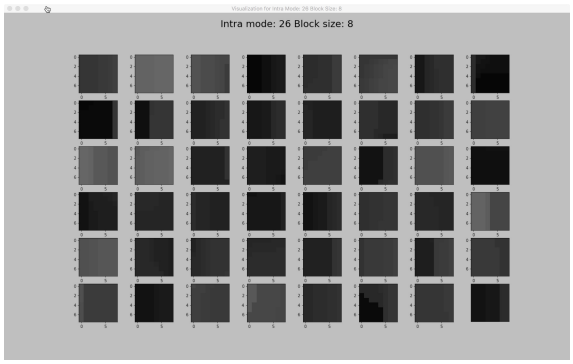


Figure 3.29: Visualizations for blocks tagged with intra mode 26.

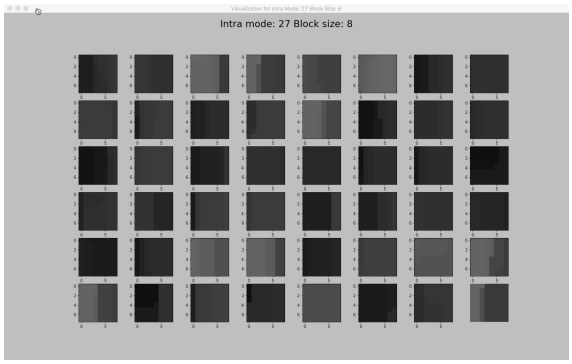


Figure 3.30: Visualizations for blocks tagged with intra mode 27.

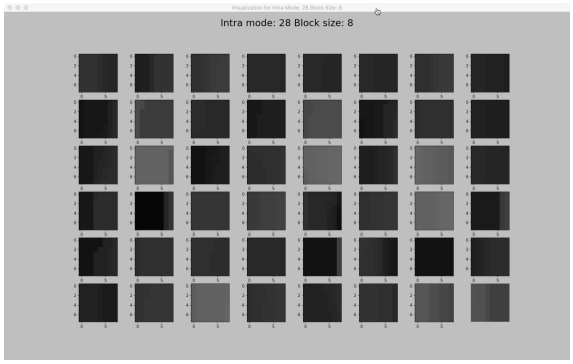


Figure 3.31: Visualizations for blocks tagged with intra mode 28.

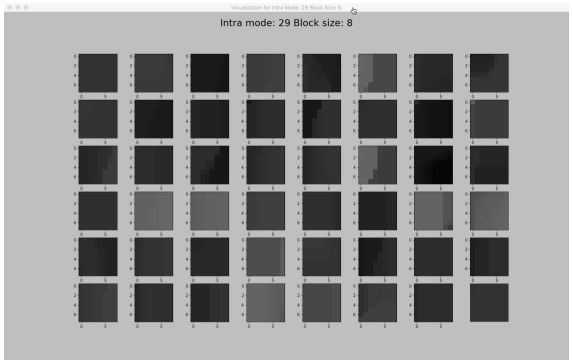


Figure 3.32: Visualizations for blocks tagged with intra mode 29.

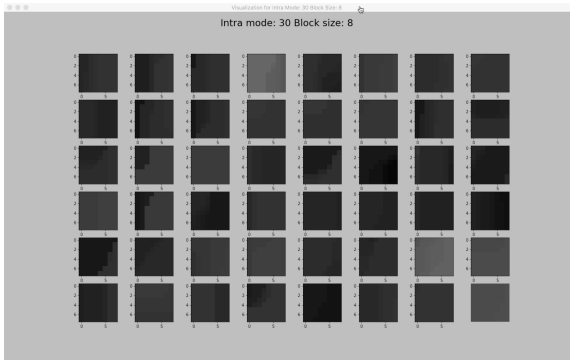


Figure 3.33: Visualizations for blocks tagged with intra mode 30.

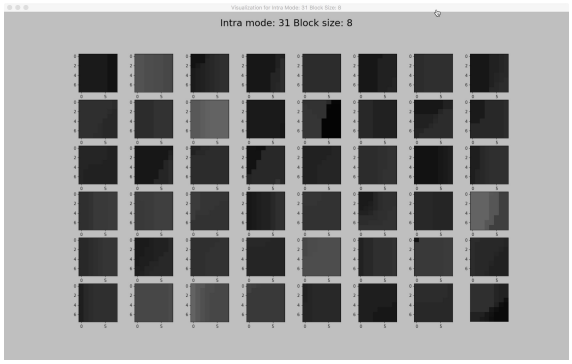


Figure 3.34: Visualizations for blocks tagged with intra mode 31.

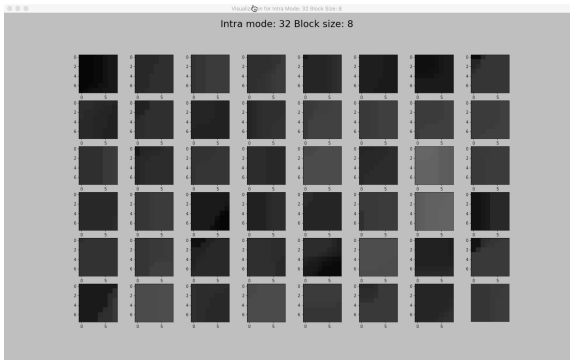


Figure 3.35: Visualizations for blocks tagged with intra mode 32.

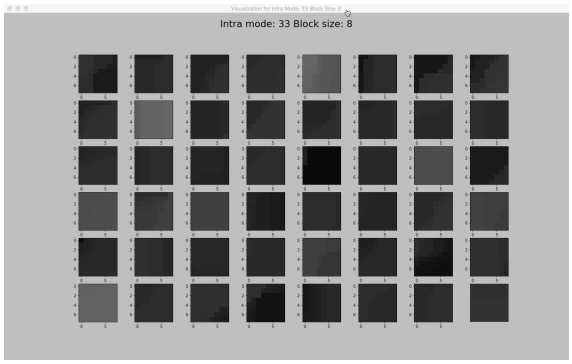


Figure 3.36: Visualizations for blocks tagged with intra mode 33.

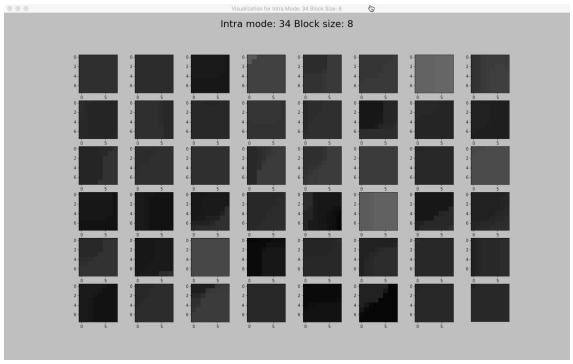


Figure 3.37: Visualizations for blocks tagged with intra mode 34.

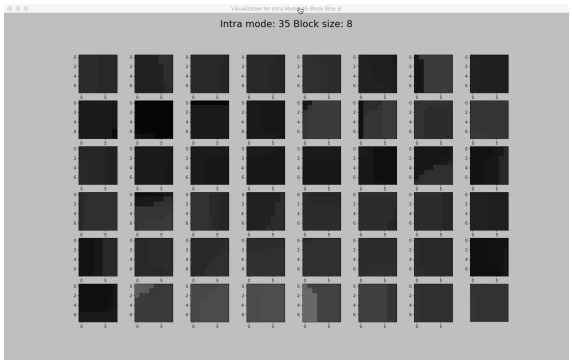


Figure 3.38: Visualizations for blocks tagged with DMM1

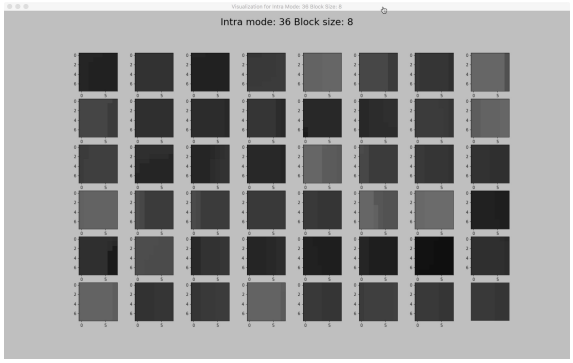


Figure 3.39: Visualizations for blocks tagged with DMM4

3.2.2 Discussion

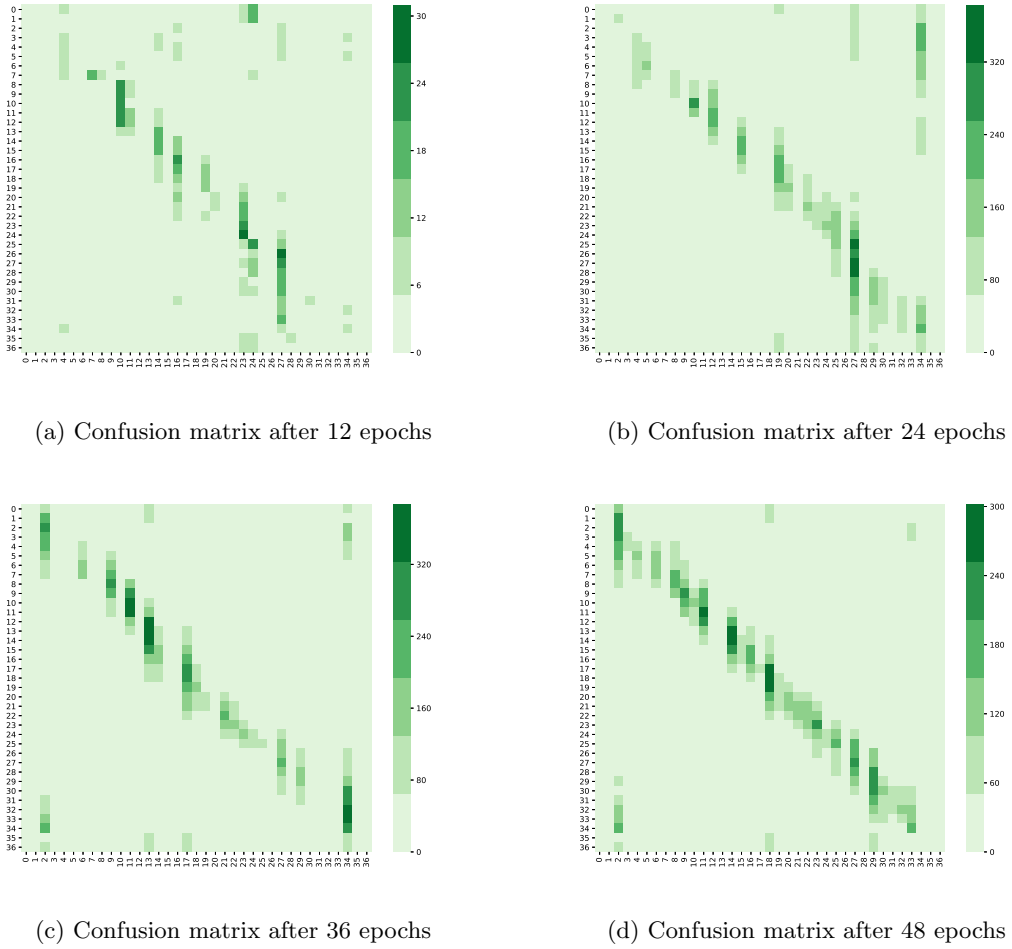
From the visualizations of blocks of size 8×8, it is found that within blocks of each single angular mode, part of them have sharp edges that can be easily perceived by human while the others are very smooth such that their edges can not be clearly seem. When blocks with such a mixed style are fed into the convolutional neural networks, the prediction accuracy is always not ideal enough to be used inside the reference software of 3D-HEVC. However, the same model learns well on other benchmark datasets such as MNIST [60] and CIFAR [61]. There are two explanations to this phenomenon. One is that the mix of extreme smoothness and clear sharpness inside blocks of each single intra prediction mode yields impure datasets such that the neural networks are not able to figure out the intrinsic abstractions layer by layer. The other is that our network size is not deep enough to have the capability of learning representations for such a mixed style. Meanwhile, the size of the dataset cannot satisfy large networks due to the limited information in the training dataset. The size of collected data is not abundant such that there is no chance of feeding a large dataset to the computational model for the time being. Moreover, larger neural networks require more computational power which can be very expensive. Combining the two considerations above, eliminating extreme smoothness is the way to go, by which the blocks with vague edges are removed from datasets.

Mode DC, PLANAR, DMM1 and DMM4 need special attention. For DC and PLANAR, since most of their blocks have weak edges with patterns that look very similar to angular modes, intuitively it seems not practical to require neural network to learn to distinguish them from angular modes. For DMM1 which is particularly designed for depth maps, it is noticed that lots of their blocks contain straight edges with arbitrary positions. Hence the learned model may predict DMM1 into any of the 33 angular modes according to the angle of the partition line in DMM1 blocks. DMM4 is another dedicated mode for depth maps. Most of their blocks feature contour partitions instead of straight lines. Some contours nevertheless cannot show their clear characteristics that can discriminate themselves from angular modes which have some curvilinear distortions.

In fact, according to our experiments, the deep neural network, which has an architecture that works well on benchmark datasets, does not work well on the collected dataset comprising modes [DC, PLANAR, 2, ..., 34, DMM1, DMM4]. During the training process, the validations are performed in a regular frequency to monitor the performance of the learned model.

Confusion matrix [62] is obtained after each validation. The matrix summarizes the performance of the learned model by presenting the classification distribution for each class. Figure 3.40 shows the confusion matrices generated during validation. It turns out the neural networks will misclassify mode 0, 1, 35, 36 into angular modes instead of giving predictions that are identical to their label of ground truth. To cure this illness, it has been decided to remove those four modes from target classes.

Different from the object recognition problems, when convolutional neural networks are employed



(a) Confusion matrix after 12 epochs

(b) Confusion matrix after 24 epochs

(c) Confusion matrix after 36 epochs

(d) Confusion matrix after 48 epochs

Figure 3.40: Confusion matrices obtained in the validation are shown here. Each vertical bar to the right of each sub-figure indicates the relationship between color thickness and number of samples. The color thickness of a block with coordinate (i, j) indicates how many samples have been classified as mode j while their real tag is mode i. For each row in the matrix, the probabilities of all blocks sum up to 100%. From epoch 12 to epoch 48, most of the thicknesses in matrices have gradually been aggregated to the main diagonal, but there are nevertheless four exceptional classes that predictions for them have never been right in all the confusion matrices. Those four classes are exactly mode DC, PLANAR, DMM1 and DMM4 which are tagged with mode index 0, 1, 35 and 36 separately.

to predict the intra patterns, some popular data pre-processing steps such as random crop, vertical flip are not applicable anymore. For example, angular mode 18 in Figure 3.21 on page 25 and angular mode 34 in Figure 3.37 on page 28 would be identical to each other in terms of the edge angle if either is vertically flipped.

Angular mode 2 and angular mode 34 are on the same diagonal such that they cannot be taken as two separate classes in our case. For this reason, it is decided to remove blocks of angular mode 34 from the datasets. Instead of directly predicting angular mode 34, when the prediction result is angular mode 2, there will be a further comparison between mode 2 and mode 34 using conventional encoder decision.

3.3 Data Pre-processing

According to the discussions in Subsection 3.2.2 on page 29, which are based on the observations of the visualized data in Subsection 3.2.1 on page 22, several pre-processing actions need to be taken to clean up the collected data for deep learning. The pre-processing steps are very crucial to the success of the learning for the convolutional neural network. Without pre-processing, it would be impossible to achieve a satisfactory model performance for prediction tasks in our work. In this section, the data pre-processing are explained in detail.

Table 3.3: Information of the datasets after merging

#	Name of the file	Size	Samples	Usage
1	size04.csv	206 MegaBytes	3,675,428	train,test,validate
2	size08.csv	513 MegaBytes	2,372,324	train,test,validate
3	size16.csv	1.25 GigaBytes	1,439,773	train,test,validate
4	size32.csv	2.02 GigaBytes	567,554	train,test,validate

Right after encoding four video sequences for data collection, four CSV files would be obtained for each of them. Every CSV file contains two unique identifications, one is *size of blocks*, the other is *name of the source video sequence*. For example, one of CSV files shall include depth Luma data of blocks of size 16×16 from *Newspaper* video sequence. In the first step of data-preprocessing, all the CSV files having the same identification of *size of blocks* shall be merged into a single dataset. The specifications of the four CSV files after merging are shown in Table 3.3 on page 31. There is no data for blocks of size 64×64 since the largest block size for DMM is 32×32. As the size of block increases, the total volume of the collected data is growing while the total number of samples of the collected data is decreasing. This phenomenon is reasonable in the sense that the decreasing speed of the number of samples is slower than the increasing speed of the volume of every single

record. More specifically, from block size 4×4 to 8×8, the number of samples decreased by 1.55 times; however, there is a fourfold increase of the volume size for each sample.

The statistics of the datasets after the first step of merging are shown in Table 3.9 on page 35 and Table 3.10 on page 36. Each of them is unsorted or sorted in terms of the percentage of each mode. Each mode can be quickly found in Table 3.9 on page 35 by looking at the first column. From Table 3.10 on page 36, it can be observed that the size differences of collected samples for each mode vary in a large range. For example, the blocks of mode 0 are 96.9 times as many the blocks of mode 16 while it is only 1.27 times as many the blocks of mode 35 for blocks of size 32×32. This is introducing the topic of imbalanced learning [63] in which the data size of each class can be very different.

According to the discussions presented in Subsection 3.2.2, mode 0, 1, 34, 35 and 36 will be removed from datasets in the second step. The specifications of the data after the second step are shown in Table 3.4. More than half of the collected data are removed.

Table 3.4: Information of the datasets after removing mode 0, 1, 34, 35 and 36

#	Name of the file	Size	Samples	Usage	Percent of removed data (%)
1	msize04.csv	75.7 MegaBytes	3,675,428	train,test,validate	64
2	msize08.csv	130.8 MegaBytes	2,372,324	train,test,validate	74
3	msize16.csv	377.3 MegaBytes	1,439,773	train,test,validate	70
4	msize32.csv	708.7 MegaBytes	567,554	train,test,validate	65

In the third step, we start to remove the smooth blocks. The reasons have been discussed in Subsection 3.2.2. Algorithm 2 on page 40, which is based on the coarse edge sharpness analysis in [54], has been designed to define the sharpness of depth blocks.

The difference between Algorithm 2 and the coarse edge sharpness analysis in [54] is that the first one takes the average sharpness value of top-k small square regions inside a single block as its block sharpness while the second one sums all the sharpness values of all small square regions inside a single block as its sharpness. In depth maps, the sharpness typically is contributed by few sharp edges. Lots of regions in a single sharp block are smooth regions. Hence it makes sense to only evaluate the top-k average instead of a sum of all. An empirical threshold of 15 has been chosen to tag a block as either smooth or sharp. Blocks tagged with smooth will be removed from datasets while the others will be kept. The specifications of the data after the third step of removing smooth blocks are shown in Table 3.5. Almost half of the data are removed.

Table 3.5: Information of the datasets after removing smooth blocks

#	Name of the file	Size	Samples	Usage	Percent of removed data (%)
1	smsize04.csv	36.3 MegaBytes	616,281	train,test,validate	54
2	smsize08.csv	91.2 MegaBytes	403,277	train,test,validate	33
3	smsize16.csv	210.9 MegaBytes	232,806	train,test,validate	46
4	smsize32.csv	235.4 MegaBytes	65,481	train,test,validate	79

The statistics of the datasets after the third step are shown in Table 3.11 on page 37 and Table 3.12 on page 38. It is clear that after several pre-processing steps, the issue of imbalanced learning [63] still exists. To tackle the issue, we truncate datasets to make the size of each class being equal. In the last step, remaining 32 modes are tagged with integers [0, 1, 2, ..., 31] to obtain new indices which starting from zero for the convenience of deep learning. At the same step, we separate datasets into training, testing and validating datasets. Datasets for blocks of 32×32 are too small to be fed into the deep neural network. They are consequently discarded. The descriptions for the finalized datasets are shown in Table 3.6, Table 3.7 and Table 3.8.

Table 3.6: Information of the finalized datasets from blocks of size 4×4

#	Name of the file	Size	Samples	Usage
1	train04.csv	7.9 MegaBytes	$4,092 \times 32$	train
2	test04.csv	1.1 MegaBytes	600×32	test
3	val04.csv	1.1 GigaBytes	600×32	validate

Table 3.7: Information of the finalized datasets from blocks of size 8×8

#	Name of the file	Size	Samples	Usage
1	train08.csv	46.3 MegaBytes	$6,303 \times 32$	train
2	test08.csv	3.8 MegaBytes	600×32	test
3	val08.csv	3.8 GigaBytes	600×32	validate

Table 3.8: Information of the finalized datasets from blocks of size 16×16

#	Name of the file	Size	Samples	Usage
1	train16.csv	89.9 MegaBytes	$3,075 \times 32$	train
2	test16.csv	15.9 MegaBytes	600×32	test
3	val16.csv	15.9 GigaBytes	600×32	validate

Table 3.9: Unsorted statistics of datasets obtained after merging

Block Size	4×4		8×8		16×16		32×32	
Mode Idx	Samples	Percent (%)	Samples	Percent (%)	Samples	Percent (%)	Samples	Percent (%)
0	717,274	19.52	642,520	27.08	459,291	31.90	158,824	27.98
1	482,776	13.14	249,061	10.50	164,050	11.39	57,528	10.14
2	97,629	2.66	25,101	1.06	12,868	0.89	4,561	0.80
3	17,991	0.49	12,489	0.53	7,946	0.55	2,863	0.50
4	14,375	0.39	11,688	0.49	8,642	0.60	2,175	0.38
5	15,849	0.43	13,428	0.57	8,829	0.61	2,164	0.38
6	17,144	0.47	15,318	0.65	9,768	0.68	2,958	0.52
7	18,187	0.49	17,238	0.73	15,988	1.11	6,625	1.17
8	19,146	0.52	15,785	0.67	20,357	1.41	11,642	2.05
9	23,462	0.64	12,362	0.52	18,207	1.26	18,195	3.21
10	42,752	1.16	9,740	0.41	7,978	0.55	18,972	3.34
11	23,727	0.65	12,836	0.54	17,696	1.23	21,142	3.73
12	21,992	0.60	17,837	0.75	23,143	1.61	13,262	2.34
13	24,613	0.67	20,254	0.85	19,260	1.34	6,740	1.19
14	22,620	0.62	17,784	0.75	13,851	0.96	2,995	0.53
15	21,169	0.58	18,268	0.77	12,834	0.89	2,073	0.37
16	20,289	0.55	15,418	0.65	10,214	0.71	1,639	0.29
17	21,869	0.60	17,501	0.74	10,010	0.70	1,977	0.35
18	52,552	1.43	19,889	0.84	9,862	0.68	1,998	0.35
19	23,871	0.65	16,171	0.68	9,797	0.68	2,170	0.38
20	22,992	0.63	15,656	0.66	10,227	0.71	1,925	0.34
21	25,416	0.69	16,706	0.70	11,978	0.83	2,504	0.44
22	27,593	0.75	16,446	0.69	12,251	0.85	2,925	0.52
23	33,250	0.90	16,783	0.71	12,744	0.89	3,707	0.65
24	40,677	1.11	17,262	0.73	12,257	0.85	4,457	0.79
25	36,018	0.98	13,841	0.58	7,812	0.54	5,123	0.90
26	404,933	11.02	70,417	2.97	30,896	2.15	17,897	3.15
27	48,843	1.33	17,062	0.72	12,205	0.85	7,414	1.31
28	34,217	0.93	24,051	1.01	16,725	1.16	6,169	1.09
29	37,756	1.03	23,486	0.99	15,647	1.09	5,436	0.96
30	30,910	0.84	20,972	0.88	12,782	0.89	3,945	0.69
31	35,102	0.95	20,499	0.86	13,331	0.93	3,475	0.61
32	25,756	0.70	18,811	0.79	12,254	0.85	3,289	0.58
33	33,270	0.91	19,088	0.80	11,943	0.83	3,526	0.62
34	73,107	1.99	31,114	1.31	15,848	1.10	5,382	0.95
35	789,662	21.48	710,089	29.93	299,368	20.79	126,427	22.28
36	276,639	7.53	139,353	5.87	70,914	4.93	23,450	4.13

Table 3.10: Sorted statistics of datasets obtained after merging

Block Size		4×4			8×8			16×16			32×32	
Row idx	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)
0	4	14,375	0.39	10	9,740	0.41	25	7,812	0.54	16	1,639	0.29
1	5	15,849	0.43	4	11,688	0.49	3	7,946	0.55	20	1,925	0.34
2	6	17,144	0.47	9	12,362	0.52	10	7,978	0.55	17	1,977	0.35
3	3	17,991	0.49	3	12,489	0.53	4	8,642	0.60	18	1,998	0.35
4	7	18,187	0.49	11	12,836	0.54	5	8,829	0.61	15	2,073	0.37
5	8	19,146	0.52	5	13,428	0.57	6	9,768	0.68	5	2,164	0.38
6	16	20,289	0.55	25	13,841	0.58	19	9,797	0.68	19	2,170	0.38
7	15	21,169	0.58	6	15,318	0.65	18	9,862	0.68	4	2,175	0.38
8	17	21,869	0.60	16	15,418	0.65	17	10,010	0.70	21	2,504	0.44
9	12	21,992	0.60	20	15,656	0.66	16	10,214	0.71	3	2,863	0.50
10	14	22,620	0.62	8	15,785	0.67	20	10,227	0.71	22	2,925	0.52
11	20	22,992	0.63	19	16,171	0.68	33	11,943	0.83	6	2,958	0.52
12	9	23,462	0.64	22	16,446	0.69	21	11,978	0.83	14	2,995	0.53
13	11	23,727	0.65	21	16,706	0.70	27	12,205	0.85	32	3,289	0.58
14	19	23,871	0.65	23	16,783	0.71	22	12,251	0.85	31	3,475	0.61
15	13	24,613	0.67	27	17,062	0.72	32	12,254	0.85	33	3,526	0.62
16	21	25,416	0.69	7	17,238	0.73	24	12,257	0.85	23	3,707	0.65
17	32	25,756	0.70	24	17,262	0.73	23	12,744	0.89	30	3,945	0.69
18	22	27,593	0.75	17	17,501	0.74	30	12,782	0.89	24	4,457	0.79
19	30	30,910	0.84	14	17,784	0.75	15	12,834	0.89	2	4,561	0.80
20	23	33,250	0.90	12	17,837	0.75	2	12,868	0.89	25	5,123	0.90
21	33	33,270	0.91	15	18,268	0.77	31	13,331	0.93	34	5,382	0.95
22	28	34,217	0.93	32	18,811	0.79	14	13,851	0.96	29	5,436	0.96
23	31	35,102	0.95	33	19,088	0.80	29	15,647	1.09	28	6,169	1.09
24	25	36,018	0.98	18	19,889	0.84	34	15,848	1.10	7	6,625	1.17
25	29	37,756	1.03	13	20,254	0.85	7	15,988	1.11	13	6,740	1.19
26	24	40,677	1.11	31	20,499	0.86	28	16,725	1.16	27	7,414	1.31
27	10	42,752	1.16	30	20,972	0.88	11	17,696	1.23	8	11,642	2.05
28	27	48,843	1.33	29	23,486	0.99	9	18,207	1.26	12	13,262	2.34
29	18	52,552	1.43	28	24,051	1.01	13	19,260	1.34	26	17,897	3.15
30	34	73,107	1.99	2	25,101	1.06	8	20,357	1.41	9	18,195	3.21
31	2	97,629	2.66	34	31,114	1.31	12	23,143	1.61	10	18,972	3.34
32	36	276,639	7.53	26	70,417	2.97	26	30,896	2.15	11	21,142	3.73
33	26	404,933	11.02	36	139,353	5.87	36	70,914	4.93	36	23,450	4.13
34	1	482,776	13.14	1	249,061	10.50	1	164,050	11.39	1	57,528	10.14
35	0	717,274	19.52	0	642,520	27.08	35	299,368	20.79	35	126,427	22.28
36	35	789,662	21.48	35	710,089	29.93	0	459,291	31.90	0	158,824	27.98

Table 3.11: Unsorted statistics of datasets obtained after removing smooth blocks

Block Size	4×4		8×8		16×16		32×32	
Mode Idx	Samples	Percent (%)	Samples	Percent (%)	Samples	Percent (%)	Samples	Percent (%)
0	0	0.00	0	0.00	0	0.00	0	0.00
1	0	0.00	0	0.00	0	0.00	0	0.00
2	28,569	4.64	14,319	3.55	6,418	2.76	1,444	2.21
3	6,302	1.02	8,170	2.03	4,275	1.84	879	1.34
4	5,292	0.86	7,503	1.86	4,625	1.99	929	1.42
5	5,730	0.93	8,511	2.11	4,302	1.85	872	1.33
6	6,632	1.08	10,450	2.59	4,998	2.15	1,080	1.65
7	7,361	1.19	11,455	2.84	7,077	3.04	2,302	3.52
8	7,870	1.28	11,580	2.87	7,595	3.26	2,861	4.37
9	10,022	1.63	9,954	2.47	5,768	2.48	3,588	5.48
10	19,385	3.15	8,059	2.00	4,427	1.90	2,769	4.23
11	11,591	1.88	10,272	2.55	6,840	2.94	4,299	6.57
12	10,234	1.66	12,985	3.22	9,746	4.19	3,159	4.82
13	11,248	1.83	14,037	3.48	10,408	4.47	2,500	3.82
14	10,838	1.76	12,570	3.12	8,151	3.50	1,532	2.34
15	10,328	1.68	12,086	3.00	7,059	3.03	1,222	1.87
16	9,930	1.61	10,393	2.58	6,237	2.68	1,103	1.68
17	11,132	1.81	11,701	2.90	6,198	2.66	1,321	2.02
18	28,674	4.65	13,292	3.30	6,192	2.66	1,180	1.80
19	12,181	1.98	11,106	2.75	6,478	2.78	1,364	2.08
20	12,661	2.05	10,773	2.67	6,510	2.80	1,298	1.98
21	14,320	2.32	11,436	2.84	7,234	3.11	1,551	2.37
22	15,903	2.58	11,567	2.87	7,882	3.39	1,778	2.72
23	16,902	2.74	11,845	2.94	8,804	3.78	2,333	3.56
24	21,987	3.57	12,093	3.00	8,941	3.84	2,909	4.44
25	19,726	3.20	10,167	2.52	5,089	2.19	2,705	4.13
26	190,471	30.91	40,592	10.07	14,439	6.20	4,821	7.36
27	23,172	3.76	12,240	3.04	6,943	2.98	3,322	5.07
28	17,158	2.78	17,156	4.25	10,722	4.61	2,909	4.44
29	17,788	2.89	16,161	4.01	9,801	4.21	2,114	3.23
30	14,012	2.27	14,060	3.49	8,341	3.58	1,554	2.37
31	14,527	2.36	13,054	3.24	7,459	3.20	1,269	1.94
32	11,072	1.80	11,944	2.96	7,149	3.07	1,264	1.93
33	13,263	2.15	11,746	2.91	6,698	2.88	1,250	1.91
34	0	0.00	0	0.00	0	0.00	0	0.00
35	0	0.00	0	0.00	0	0.00	0	0.00
36	0	0.00	0	0.00	0	0.00	0	0.00

Table 3.12: Sorted statistics of datasets obtained after removing smooth blocks

Block Size		4×4			8×8			16×16			32×32	
Row idx	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)	Mode idx	Samples	Percent (%)
0	0	0	0.00	0	0	0.00	0	0	0.00	0	0	0.00
1	1	0	0.00	1	0	0.00	1	0	0.00	1	0	0.00
2	34	0	0.00	34	0	0.00	34	0	0.00	34	0	0.00
3	35	0	0.00	35	0	0.00	35	0	0.00	35	0	0.00
4	36	0	0.00	36	0	0.00	36	0	0.00	36	0	0.00
5	4	5,292	0.86	4	7,503	1.86	3	4,275	1.84	5	872	1.33
6	5	5,730	0.93	10	8,059	2.00	5	4,302	1.85	3	879	1.34
7	3	6,302	1.02	3	8,170	2.03	10	4,427	1.90	4	929	1.42
8	6	6,632	1.08	5	8,511	2.11	4	4,625	1.99	6	1,080	1.65
9	7	7,361	1.19	9	9,954	2.47	6	4,998	2.15	16	1,103	1.68
10	8	7,870	1.28	25	10,167	2.52	25	5,089	2.19	18	1,180	1.80
11	16	9,930	1.61	11	10,272	2.55	9	5,768	2.48	15	1,222	1.87
12	9	10,022	1.63	16	10,393	2.58	18	6,192	2.66	33	1,250	1.91
13	12	10,234	1.66	6	10,450	2.59	17	6,198	2.66	32	1,264	1.93
14	15	10,328	1.68	20	10,773	2.67	16	6,237	2.68	31	1,269	1.94
15	14	10,838	1.76	19	11,106	2.75	2	6,418	2.76	20	1,298	1.98
16	32	11,072	1.80	21	11,436	2.84	19	6,478	2.78	17	1,321	2.02
17	17	11,132	1.81	7	11,455	2.84	20	6,510	2.80	19	1,364	2.08
18	13	11,248	1.83	22	11,567	2.87	33	6,698	2.88	2	1,444	2.21
19	11	11,591	1.88	8	11,580	2.87	11	6,840	2.94	14	1,532	2.34
20	19	12,181	1.98	17	11,701	2.90	27	6,943	2.98	21	1,551	2.37
21	20	12,661	2.05	33	11,746	2.91	15	7,059	3.03	30	1,554	2.37
22	33	13,263	2.15	23	11,845	2.94	7	7,077	3.04	22	1,778	2.72
23	30	14,012	2.27	32	11,944	2.96	32	7,149	3.07	29	2,114	3.23
24	21	14,320	2.32	15	12,086	3.00	21	7,234	3.11	7	2,302	3.52
25	31	14,527	2.36	24	12,093	3.00	31	7,459	3.20	23	2,333	3.56
26	22	15,903	2.58	27	12,240	3.04	8	7,595	3.26	13	2,500	3.82
27	23	16,902	2.74	14	12,570	3.12	22	7,882	3.39	25	2,705	4.13
28	28	17,158	2.78	12	12,985	3.22	14	8,151	3.50	10	2,769	4.23
29	29	17,788	2.89	31	13,054	3.24	30	8,341	3.58	8	2,861	4.37
30	10	19,385	3.15	18	13,292	3.30	23	8,804	3.78	24	2,909	4.44
31	25	19,726	3.20	13	14,037	3.48	24	8,941	3.84	28	2,909	4.44
32	24	21,987	3.57	30	14,060	3.49	12	9,746	4.19	12	3,159	4.82
33	27	23,172	3.76	2	14,319	3.55	29	9,801	4.21	27	3,322	5.07
34	2	28,569	4.64	29	16,161	4.01	13	10,408	4.47	9	3,588	5.48
35	18	28,674	4.65	28	17,156	4.25	28	10,722	4.61	11	4,299	6.57
36	26	190,471	30.91	26	40,592	10.07	26	14,439	6.20	26	4,821	7.36

Algorithm 1: Collect data

Input: CU data structure pcCU, absolute partition index of CU uiAbsPartIdx, quad-tree depth uiDepth

Output: Flattened luma pixel values of each block together with the index of its best intra mode in each row of the output csv file

```
1  begin
2      for each CU in depth maps do
3          uiCuSize ←getCUSize(pcCU, uiDepth)
4          pOrgPel ←getYPelCU(pcCU)
5          if DISFlag ≡ 0 then
6              partitionMode ←getPartitionSize(pcCU, uiAbsPartIdx)
7              if partitionMode ≡ sizeOfNByN then
8                  iPartNum ← 4
9              else
10                 iPartNum ← 1
11             for j ← 0 to iPartNum do
12                 iDir[j] ← getIntraDir(pcCU, uiAbsPartIdx)
13             if iPartNum ≡ 1 then
14                 Create a new csv file, append the value of uiDepth at the end of the name of
                    the new csv file
15                 for y ← 0 to uiCuSize do
16                     for x ← 0 to uiCuSize do
17                         Write pOrgPel[x] into row_m in csv file
18                     pOrgPel ← pOrgPel + iStride
19                 Write iDir[0] into the end of row_m in the csv file
20             else
21                 Create a new csv file, append the value of (uiDepth + 1) at the end of the name
                    of the new csv file
22                 sizeOfSubBlk ←getSizeOfSubBlk(pcCU, uiDepth)
23                 for j ← 0 to iPartNum do
24                     if j ≡ 0 then
25                         yStartPos ← 0 & xStartPos ← 0 & yEndPos ← sizeOfSubBlk & xEndPos
                            ← sizeOfSubBlk
26                     else if j ≡ 1 then
27                         yStartPos ← 0 & xStartPos ← sizeOfSubBlk & yEndPos ← sizeOfSubBlk
                            & xEndPos ← sizeOfSubBlk × 2
28                     else if j ≡ 2 then
29                         yStartPos ← sizeOfSubBlk & xStartPos ← 0 & yEndPos
                            ← sizeOfSubBlk × 2 & xEndPos ← sizeOfSubBlk
30                     else if j ≡ 3 then
31                         yStartPos ← sizeOfSubBlk & xStartPos ← sizeOfSubBlk & yEndPos
                            ← sizeOfSubBlk × 2 & xEndPos ← sizeOfSubBlk × 2
32                 for y ← yStartPos to yEndPos do
33                     for x ← xStartPos to xEndPos do
34                         Write pOrgPel[x] into row_m in the csv file
35                     pOrgPel ← pOrgPel + iStride
36                 Write iDir[j] into the end of row_m in the csv file
```

Algorithm 2: Sharpness analysis for single block

Input: Luma pixel values stored inside an one dimensional array FlattenLumaVals,
Output: Boolean value ISSMOOTH

```
1  begin
2      BLKSIZE ← |√getLength(FlattenLumaVals)|
3      SquareLumaVals ←reshape(FlattenLumaVals)
4      TotalSharpness ← 0
5      ArrayOfSharpness ← ∅
6      for i ← 0 to BLKSIZE − 1 do
7          for j ← 0 to BLKSIZE − 1 do
8              HorSharpness x ← SquareLumaVals [i][j] + SquareLumaVals [i + 1][j] −
                SquareLumaVals [i][j + 1] − SquareLumaVals [i + 1][j + 1]
9              VerSharpness x ← SquareLumaVals [i][j] + SquareLumaVals [i][j + 1] −
                SquareLumaVals [i + 1][j] − SquareLumaVals [i + 1][j + 1]
10             Sharpness ← HorSharpness^2 + VerSharpness^2
11             TotalSharpness ← TotalSharpness + Sharpness
12             ArrayOfSharpness ← pushValIntoArray(ArrayOfSharpness, Sharpness)
13     ArrayOfSharpness ← sortArray(ArrayOfSharpness)
14     ArrayOfSharpness ← getTopKEleAndFormNewArray(ArrayOfSharpness, BLKSIZE × 2)
15     ArrayOfSharpness ← removeEleLessThanEight(ArrayOfSharpness)
16     SIZEOFARRAY ← getSizeOfArray(ArrayOfSharpness)
17     if SIZEOFARRAY ≡ 0 then
18         BlockSharpness ← 0
19     else
20         BlockSharpness ← getMeanOfArray(ArrayOfSharpness)
21     if BlockSharpness > 15 then
22         ISSMOOTH ← 0
23     else
24         ISSMOOTH ← 1
```

Chapter 4

Train the Deep Model for Prediction

The convolutional neural networks (CNNs) consist of neurons that have weights and bias, which can be trained using large datasets for solving computer vision tasks. In this chapter, our architecture of deep convolutional neural network for intra mode prediction is illustrated. Then the hyper-parameters of our deep model is introduced. At the end of the chapter, the stopping criteria and training results are presented.

4.1 The Architecture of CNN

There exist many kinds of convolutional neural networks while the major difference that distinguish them from each other is the uniqueness of each architecure. Figure 4.1 on page 42 illustrates the basic unit of convolutional neural networks. The light blue cubics represent the Luma pixel values from a single coding unit (CU) of size 8×8. The yellow cubics show a kernel of size 3×3 that slides over the CU vertically and horizontally. At each position, the kernel calculates a weighted sum of all its inputs, then adds a bias. The output from the region it covers will be fed into a neuron right below the covered region. There is a big difference between multilayer perceptron and CNN the weights of the neurons are shared in the latter case. For example, all the neurons in Figure 4.1 on page 42 are reusing the same patch of parameters, i.e., weights and biases. For a single kernal of size 3×3, it has 9 weights. Apparently this amount of learnable parameters is just not enough to learn the representations. More degrees of freedom are needed to enhance the learning capacity of the neural network. For this reason, a large number of kernels will be used instead of one single kernal. If many kernels are stacked in a convolutional neural network such that the architecture looks very deep, the network is called deep neural network instead of simply neural network which typically

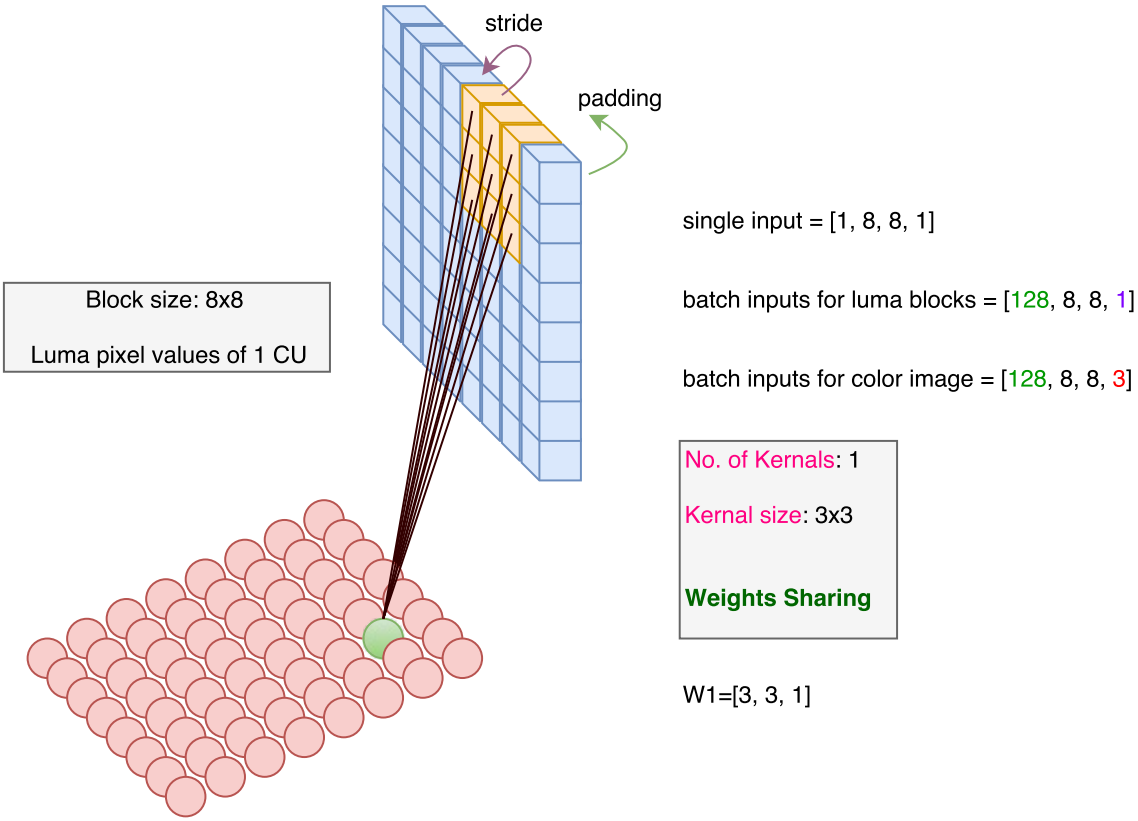


Figure 4.1: The basic unit of the convolutional neural networks.

refers to shallow ones. In deep convolutional neural network, each convolutional layer comprising multiple kernels which subsequently produces a three dimensional volume of the outputs. After the convolutional layer, there normally exist an activating layer where the specified activation function will be applied to each single output inside the output volume from the convolutional layer. Followed by the activating layer, there is a pooling layer. In the previous years, the maximum pooling and average pooling are popular methods for reducing the dimension of the outputs from convolutional layers. Nowadays, instead of applying a conventional pooling, a convolutional layer of stride > 1 is utilized. Such a combination of convolution-activation-pooling will be replicated as many times as you wish for a single convolutional neural network. In the tail of the network, a fully-connected layer will be used to compute the probability of each target class for the classification problems.

Our network is built with above description as guideline except the fact that we have adopted the identity mapping $h(\mathbf{x}_l) = \mathbf{x}_l$ in the Residual Neural Network from [17]. It is shown in [64] that the identity mapping in residual units can enable the direct propagation of information from one layer to any other layers. Such a nice property brings vital benefits for deep neural networks such that the saturation of accuracy can be largely alleviated. As a result, ultra-deep models are able to learn better representations to solve vision tasks. Besides, with the identity mapping, the network is able to converge faster hence the training time required to train a deep model can be reduced. Figure 4.2 on page 46 shows two types of the residual units in which the identity mapping is implemented by shortcut connections from the beginning of the block to the end of the same block.

The architecture of the deep convolutional neural network used for intra mode prediction in our work is shown in Figure 4.3 on page 47. The first layer consists of 16 kernels where the receptive fields are simply stacked on top of each other. In the middle of the network, there are three variants of resnet unit, each unit is replicated five times to deepen the network for increasing learning capacities. Batch Normalization is employed to regularize data inputs to each of the layers. If a layer has n-dimensional input $x = (x_1, x_2, \ldots, x_n)$, for each element in the input the batch normalization applies:

$$\hat{x}_m = \frac{x_m - E_b(x_m)}{\sigma_b(x_m) + \varepsilon} \tag{4.1}$$

$$BN(x_m) = \alpha(\hat{x}_m) + \beta \tag{4.2}$$

Here E_b is the mini-batch mean, σ_b is the mini-batch variance, ε is for preventing a division by zero. The idea is to compute the average and variance on a mini-batch of inputs, after that center and re-scale the weighted sum added with bias, lastly the learnable scale and offset are placed into the network to restore expressiveness. If we put $\alpha = \sigma_b(x_m)$ and $\beta = E_b(x_m)$ into equation 4.2, we will have $BN(x_m) = x_m$. In equation 4.1, x_m depends on weights, biases and input vectors, $E_b(x_m)$ depends input vectors together with only one set of weights and biases in a mini-batch. Therefore, batch normalization is differentiable relatively to weights, biases, scales and offsets. It

can be implemented as additional layers in neural networks since gradient descent algorithms still will work. The activation function for input units and hidden units is the most popular one named Relu, while the activation function in the output layer is softmax. After all the convolutional layers, global average pooling has been performed before the fully-connected layer. For each of the neurons in the fully-connected layer, the softmax function is applied to calculate the probability which tells how likely the class corresponding to that neuron is the truth. The activation Y_m of the mth output neuron in the fully-connected layer is

$$Y_m = softmax(L_m) \tag{4.3}$$

$$= \frac{e^{L_m}}{\sum_{a}^{A} e^{L_a}} \tag{4.4}$$

where A denotes the total number of classes, and L_m is the weighted sum of all related pixels adding a bias:

$$L_m = X_n \cdot W_{nm} + b \tag{4.5}$$

During training, the output Y_m from softmax layer will be fed into the cross-entropy loss function:

$$E_{\text{xent}} = -\sum_{m}^{A} T_m \log Y_m \tag{4.6}$$

where T_m is the label of ground truth for each class from training dataset. The learning happens when each of the training steps starts to minimize the loss of cross entropy using backpropagation.

In backpropagation, we first calculate the gradient of loss with respect to the weight that connects

neuron n to output neuron m in the last layer (a.k.a top layer):

$$\frac{\partial E_{\text{xent}}}{\partial W_{nm}} = \frac{\partial E_{\text{xent}}}{\partial L_m} \cdot \frac{\partial L_m}{\partial W_{nm}} \tag{4.7}$$

$$= \sum_{r}^{A} \frac{\partial E_{\text{xent}}}{\partial Y_r} \cdot \frac{\partial Y_r}{\partial L_m} \cdot X_{nm} \tag{4.8}$$

$$= (\frac{\partial E_{\text{xent}}}{\partial Y_m} \cdot \frac{\partial Y_m}{\partial L_m} + \sum_{m \neq r} \frac{\partial E_{\text{xent}}}{\partial Y_r} \cdot \frac{\partial Y_r}{\partial L_m}) \cdot X_{nm} \tag{4.9}$$

$$= (-\frac{T_m}{Y_m} \cdot Y_m \cdot (1 - Y_m) + \sum_{m \neq r} \frac{-T_r}{Y_r} \cdot (-Y_r \cdot Y_m)) \cdot X_{nm} \tag{4.10}$$

$$= (-T_m \cdot (1 - Y_m) + \sum_{m \neq r} T_r \cdot Y_m) \cdot X_{nm} \tag{4.11}$$

$$= (-T_m + T_m \cdot \ Y_m + \sum_{m \neq r} T_r \cdot Y_m) \cdot X_{nm} \tag{4.12}$$

$$= (-T_m + \sum_{r} T_r \cdot Y_m) \cdot X_{nm} \tag{4.13}$$

$$= (-T_m + Y_m \sum_{r} T_r) \cdot X_{nm} \tag{4.14}$$

$$= (-T_m + Y_m) \cdot X_{nm} \tag{4.15}$$

Here X_{nm} is the activation vector from neuron n that has been connected to mth output neuron. At this time, a fraction of the gradient will be used to update the weights and biases in the network to minimize the loss of cross entropy. In addition, the gradient of the weight W_{qp} that connects input neuron q to neuron p in hidden layers can be calculated:

$$\frac{\partial E_{\text{xent}}}{\partial W_{qp}} = \frac{\partial E_{\text{xent}}}{\partial L_p} \cdot \frac{\partial L_p}{W_{qp}} \tag{4.16}$$

$$= \sum_{r}^{A} \frac{\partial E_{\text{xent}}}{\partial L_r} \cdot \frac{\partial L_r}{\partial Y_p} \cdot \frac{\partial Y_p}{\partial L_p} \cdot \frac{\partial L_p}{W_{qp}} \tag{4.17}$$

$$= \sum_{r}^{A} (Y_r - T_r) \cdot W_{pr} \cdot (Y_p \cdot (1 - Y_p)) \cdot (X_{qp}) \tag{4.18}$$

The rest gradient calculations can be derived in the similar manner such that all the gradients in the network can be obtained.

4.2 Settings of the Network

In this section, the hyper-parameters, which have been used for deep learning, are given. The configurations are fine-tuned results against the validation datasets shown in Table 3.7 on 33 and

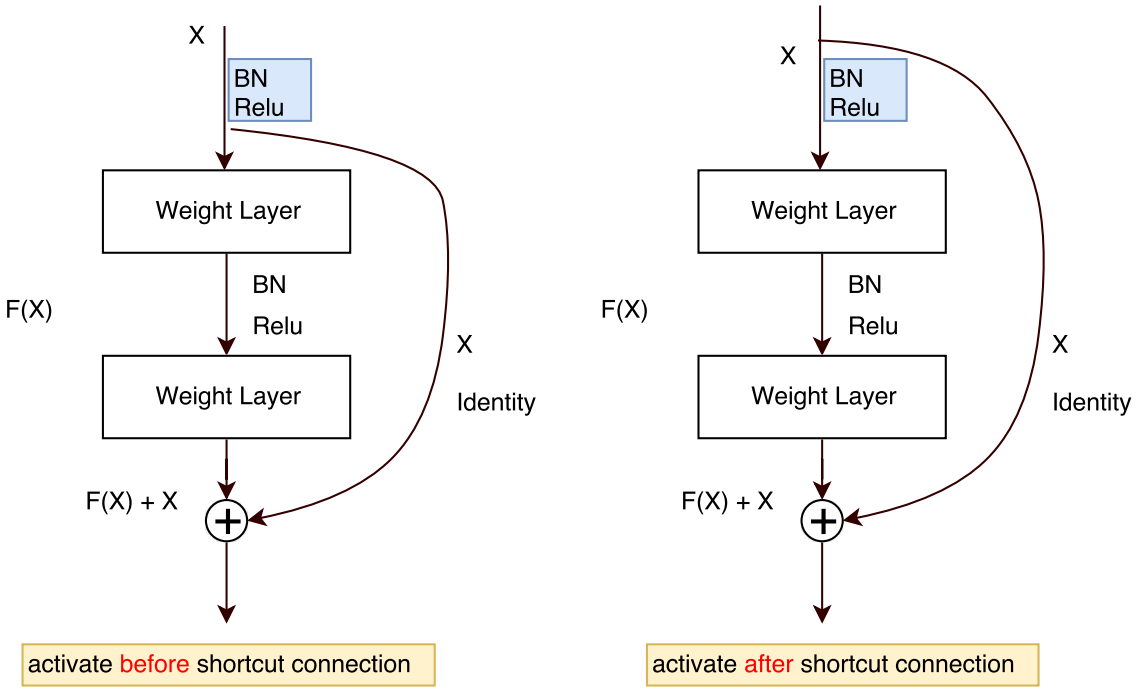


Figure 4.2: Two types of the residual units are shown here. BN stands for Batch Normalization while Relu stands for Rectified Linear Unit.

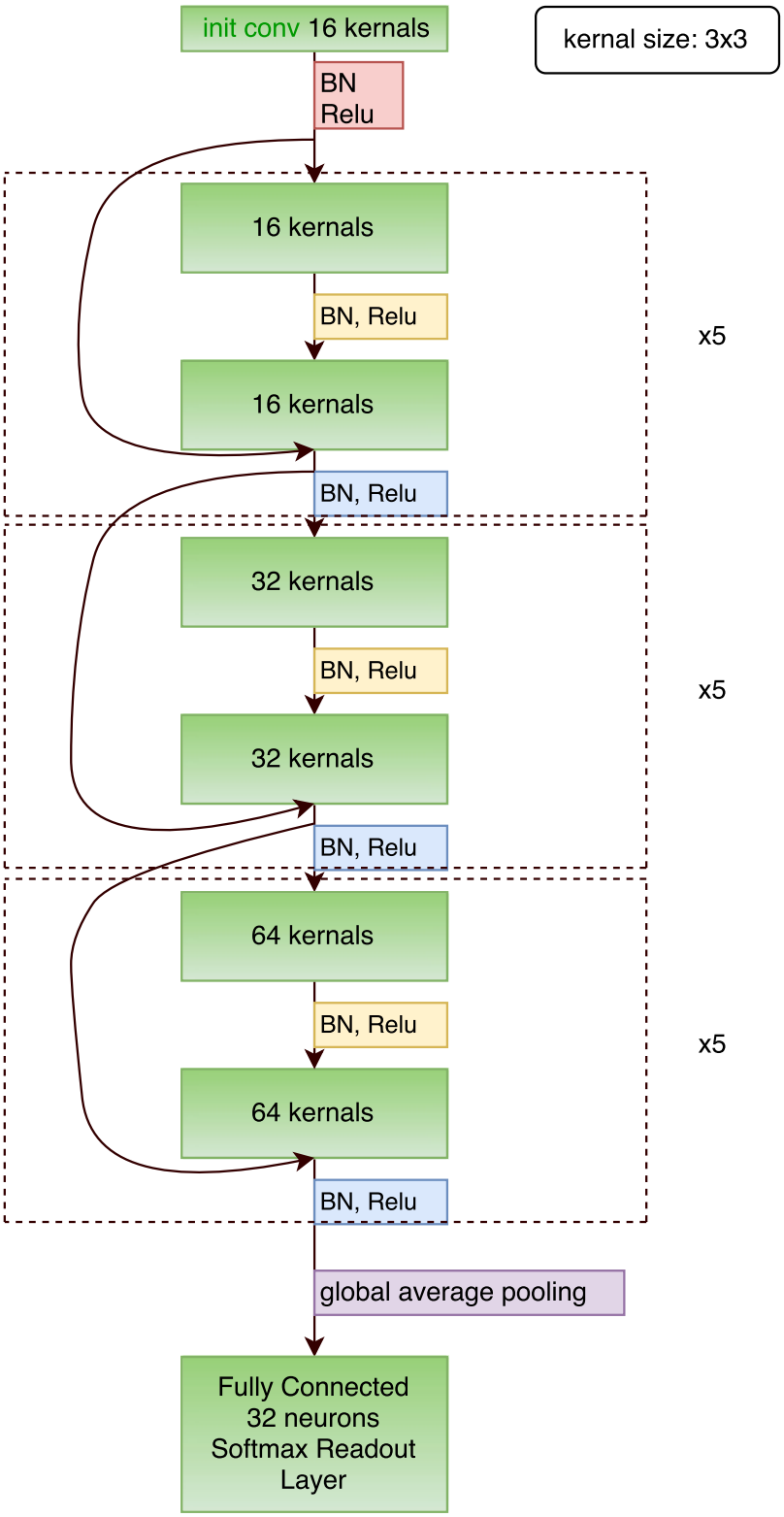


Figure 4.3: The architecture of the deep convolutional neural network used for intra mode prediction in our work.

Table 3.8 on 34 to achieve the best performance.

It is interesting to notice that in this network, the size of any reception field is always 3×3. It is found in [30] that larger kernel can be factorized into smaller kernels. A kernal with such a size is the smallest field which is capable of capturing the positional information [65]. All the strides are set to be one, which means no down-sampling will be performed. The depth CU that has been tagged as DMM1 is essentially a feature itself. Therefore there is no need to down-sampling the CU into smaller sizes. The output size of each convolutional layer will not be changed. For the inputs to the convolutional layers, zero padding (a.k.a SAME padding) is used. It is a padding algorithm which has the objective to pad zeros evenly for each direction. No data augmentation is used for the inputs. The kernal parameters are randomly initialized with a normal distribution where the mean is zero while the standard deviation is calculated using $\sqrt{2 \div (a \times a \times n)}$. Here the a represents the kernel size while n denotes the number of filters in the corresponding convolutional layer. The weights of the kernels are regularized by a decay of 0.0002. No transfer learning is used, which means all the learnable parameters are trained from scratch. The batch size used for training is 128. The loss function used is cross-entropy which has a better performance for classification problems than other distance functions such as L1 and L2. Stochastic gradient descent is used with a momentum of 0.9. The learning rate decay policy is shown in Table 4.1.

Table 4.1: Learning rate policy

Global step	Learning rate
<20,000	0.01
<40,000	0.001
<60,000	0.0001
Others	0.00001

4.3 Stopping Criteria and Training Result

During training, we keep eyes on various statistics to ensure the network is truly learning high dimensional representations instead of being trapped in a undesired status where nothing can be learned. Tensorflow [66] has been used in this work to implement the deep convolutional neural network. It has its own dedicated suite of visualization tools named Tensorboard which is very helpful with understanding and monitoring the training process. It has been configured to record the top-k precision on the validation datasets during training.

Figure 4.4 on page 50 is a screen capture of the Tensorboard user interface which contains the top-20 validation precision for blocks of size 4×4. It can be seem that top-20 precision cannot go beyond 0.85. Figure 4.5 on page 50 further shows the top-28 validation precision for blocks of

the same size. Although the highest precision can reach 0.96 in Figure 4.5, it is not applicable in our case since it involves 28 classes to achieve this acceptable precision. Only four classes would be excluded. Due to this observation, it is decided to keep the intra mode decision process unmodified for blocks of size 4×4 in $HTM16.2$.

Figure 4.6 on page 51 shows the top-16 validation precision for blocks of size 8×8. After 20,000 global steps, the top-16 precision curve in Figure 4.6 is very steady. The precision value is higher than 0.92. It means if this model can be employed to predict the most probable intra angular directions, half of the angular directions can be skipped for the Rate Distortion Cost evaluation process by which a bunch of time shall be saved. Figure 4.7 on page 51 shows the top-16 validation precision for blocks of size 16×16. After 100,000 global steps, the top-16 precision in Figure 4.7 has reached 0.9654 which is considered as an excellent result in our application scenario. Once again, the half of the intra angular modes can be bypassed if this model can be applied in $HTM16.2$. For the blocks of size 32×32, the dataset obtained for them does not have enough data for the deep learning. Instead of using a dedicated model for them, the model trained from blocks of size 16×16 has been reused. The blocks of size 32×32 are resized to size 16×16 using Bilinear Interpolation.

Apart from monitoring validation precision, confusion matrix is the other useful criteria to help making decision on when to stop the training. Figure 4.8 on page 52 shows the confusion matrix after 173 epochs of training for blocks of size 8×8. Figure 4.9 on page 53 shows the confusion matrix after 396 epochs of training for blocks of size 16×16. Both of the two confusion matrices exhibit good property which shows most of the predictions fall in the top-k predictions.

4.4 Evaluate the Learned Models

Two models have been obtained after the training process. One of the models is trained from blocks of size 8×8 and is applied to blocks with the same size. The other one is trained from blocks of size 16×16, however, it is applied to blocks of size 16×16 and size 32×32. The top-k precision on testing datasets for various block sizes is reported in Table 4.2 on page 54 (see also Figure 4.10 on page 55). Notice that the maximum size for DMM1 wedgelet prediction is 32×32, hence there is no need to perform evaluation for blocks of size 64×64. To evaluate performance on blocks of size 32×32, four resizing methods, i.e., Bilinear Interpolation, Nearest Neighbor Interpolation, Bicubic Interpolation, and Area Interpolation, have been tried. It turns out the differences among the four resizing methods are so trivial in terms of the prediction accuracy. Here Bilinear Interpolation has been chosen to be applied in our work. It can be observed that starting from top-14, the error rates are all below 10%. And the model learns well for larger block sizes. When the models are integrated to the encoder, if a larger value of $\mathbf{k}$ is used, fewer modes can be excluded which can result in a small time reduction in the encoding process. In the contrary, if smaller value of $\mathbf{k}$ is used, the prediction accuracy may not be high enough to ensure the encoder performance. In our work top-15 has been

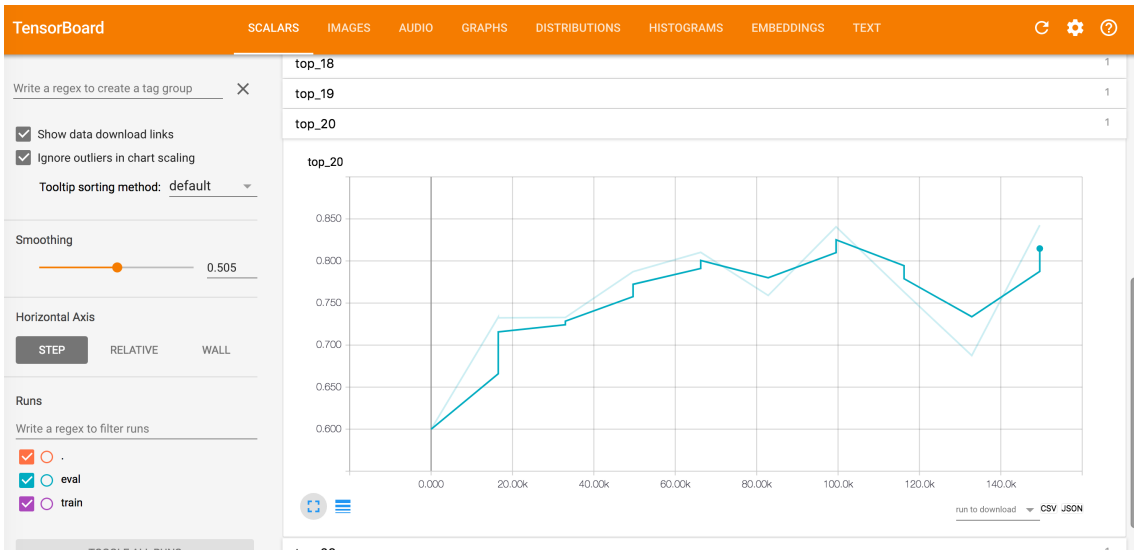


Figure 4.4: Top-20 precision on validation dataset for blocks of size 4×4.

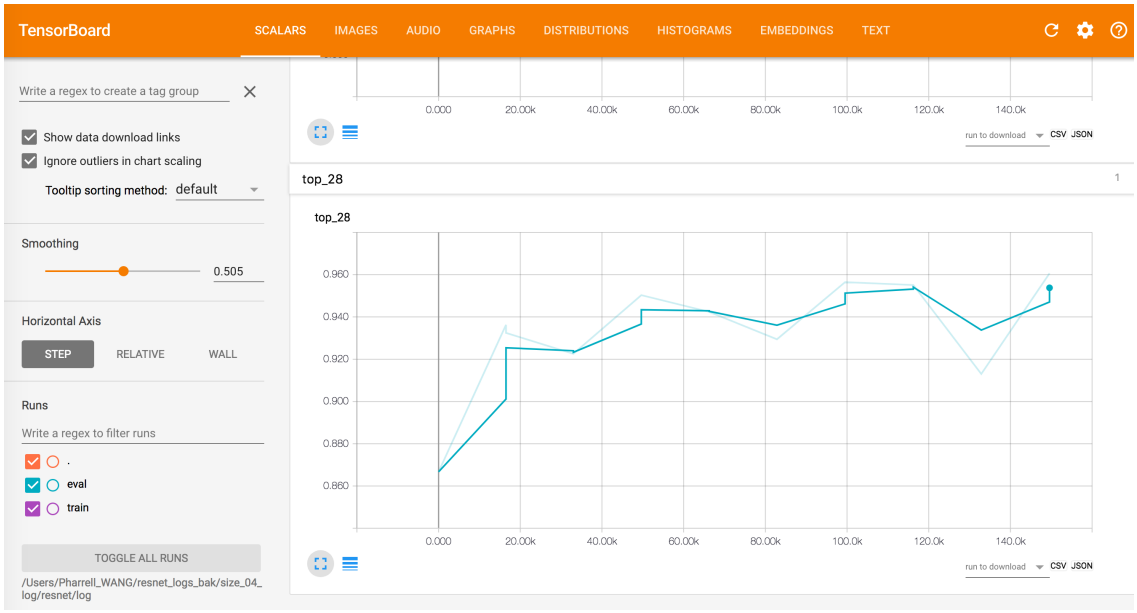


Figure 4.5: Top-28 precision on validation dataset for blocks of size 4×4.

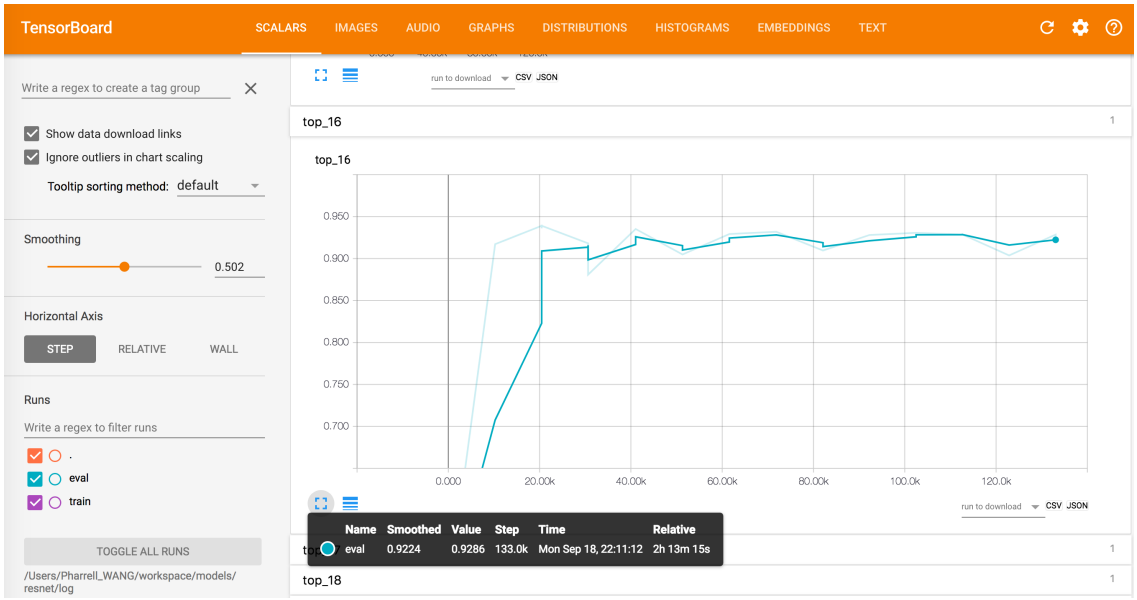


Figure 4.6: Top-16 precision on validation dataset for blocks of size 8×8.

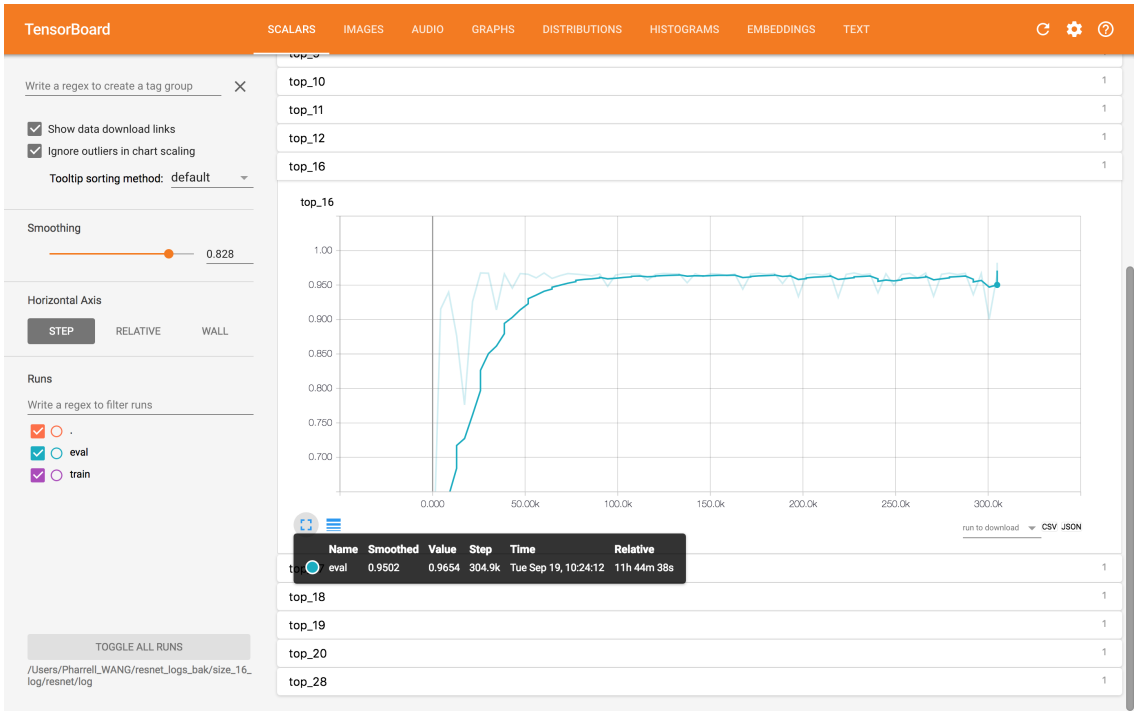


Figure 4.7: Top-16 precision on validation dataset for blocks of size 16×16.

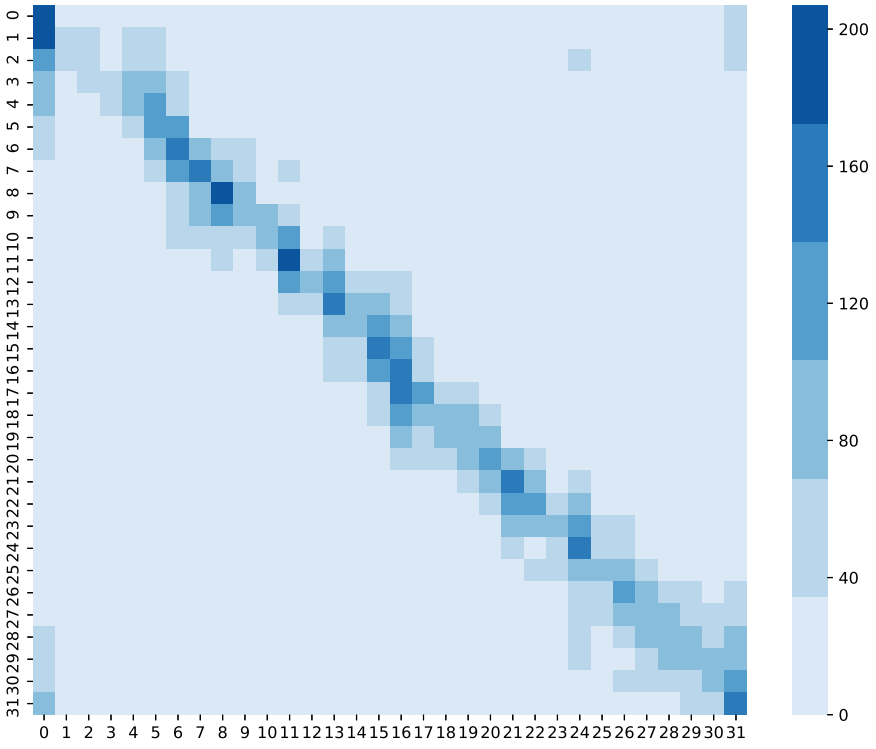


Figure 4.8: Confusion matrix after 173 epochs of model training for blocks of size 8×8.

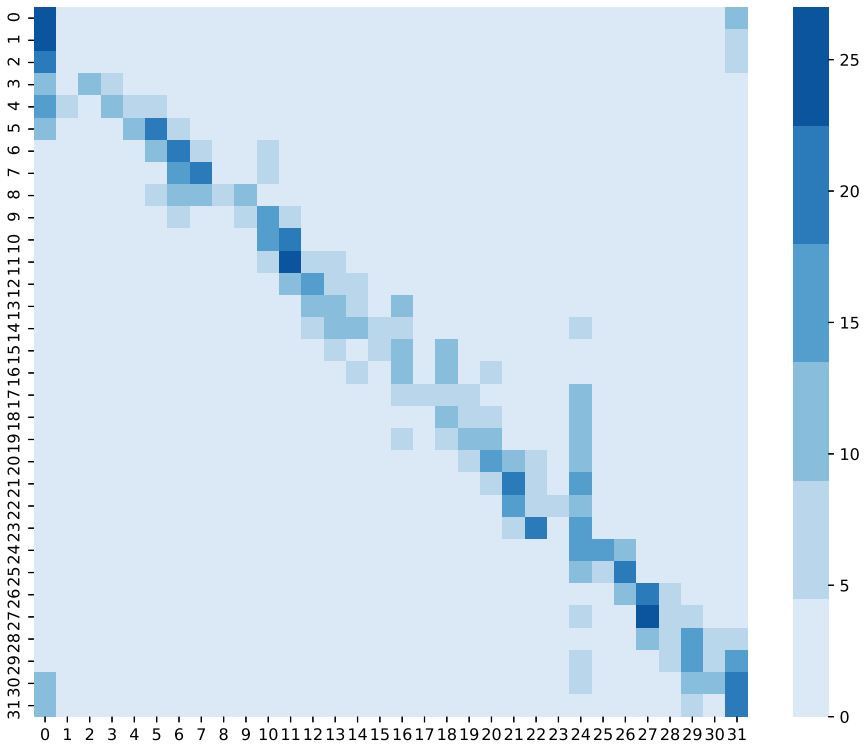


Figure 4.9: Confusion matrix after 396 epochs of training for blocks of size 16×16.

Table 4.2: Top-k precision on testing datasets for various block sizes

#	Top-k	Size 8×8	Size 16×16	Size 32×32
1	Top-5	0.650	0.801	0.819
2	Top-6	0.711	0.842	0.860
3	Top-7	0.759	0.873	0.891
4	Top-8	0.795	0.895	0.912
5	Top-9	0.823	0.912	0.927
6	Top-10	0.846	0.924	0.938
7	Top-11	0.867	0.934	0.947
8	Top-12	0.884	0.942	0.955
9	Top-13	0.897	0.949	0.961
10	Top-14	0.909	0.955	0.966
11	Top-15	0.919	0.961	0.970
12	Top-16	0.929	0.965	0.973
13	Top-17	0.936	0.970	0.977
14	Top-18	0.944	0.974	0.980
15	Top-19	0.950	0.977	0.983
16	Top-20	0.955	0.980	0.985

adopted which is responsible for balancing the trade-off between coding performance and the time cost in the encoder.

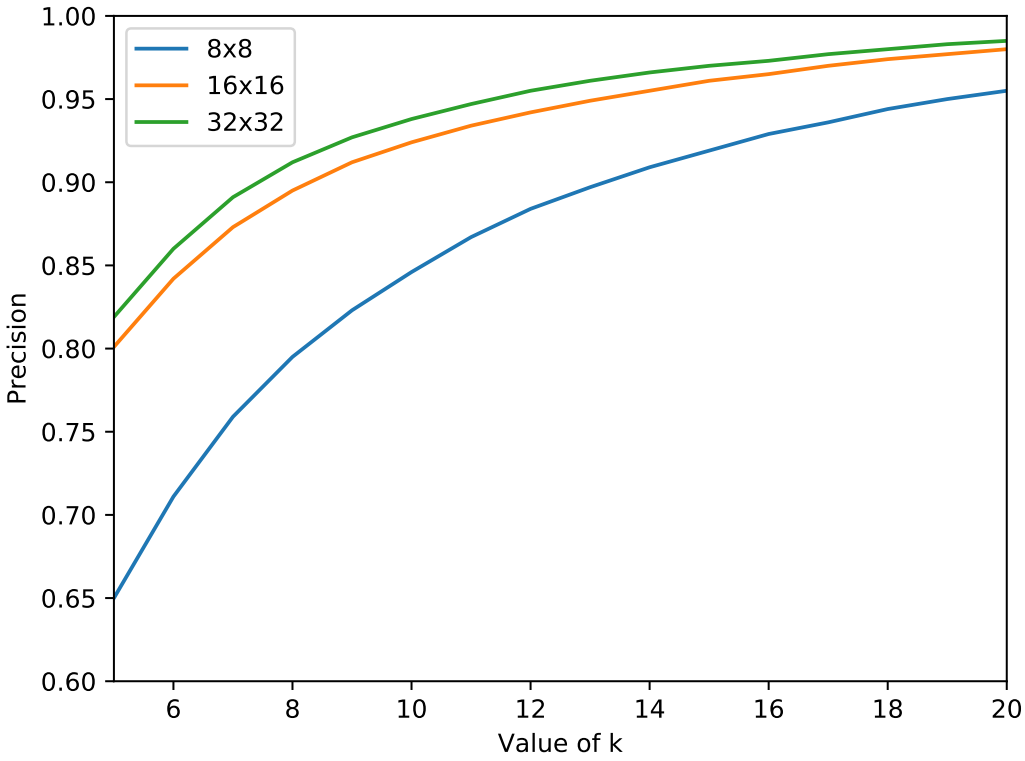


Figure 4.10: Top-k precision on testing datasets for various block sizes.

Chapter 5

Employ the Learned Deep Model

After we have obtained the learned models which exhibit desired behaviors, their integration into the reference software is carried out. In this chapter, the time cost of the model prediction is analyzed and optimizing methods are described. After that we explain the details of the algorithm for integration. Lastly the results of experiments are presented with discussions.

5.1 Analysis and Optimization for Prediction Time

It is intuitive to think in this way: every time when the encoder encounters a new block in the video frames, the learned models are required to perform the prediction for that block. However, after this idea has been implemented, it turns out the time cost of the encoder with neural engine even gets much higher than the encoder without neural engine. We found that the idea of one prediction for one block actually introduces a problem that impedes the integration to work as expected: a new session is initialized for a prediction, and the initialization for a single session in Tensorflow C++ APIs is really time expensive. The other idea which is a better way would be a single session for a batch of blocks.

The time cost of session initialization has been investigated and compared between the two ideas aforementioned. The learned model for blocks of size 8×8 is loaded to run predictions for blocks of size 8×8 from one single frame in one video sequence with the resolution to be 1024×768. Totally 12288 blocks to predict. When compiling the executable binary of the encoder powered by neural engine, it can be compiled against CPU or GPU when the compilation tool called Bazel [18] is utilized. Even more, with Bazel, the AVX and SSE4.2 can be employed to accelerate the parallel computations in one instruction at single moment in CPU. They offer more efficient matrix computations to CPU. Six experiments have been conducted and the results are shown in Table 5.1.

Obviously the best way is to compile the executable binary against AVX, SSE4.2 and GPU, at

Table 5.1: Time cost of session initialization for two ideas based on three sets of compiling configurations

#	Idea	Plain CPU	AVX,SSE4.2	AVX,SSE4.2,GPU
1	init 1 session for 12288 blocks	21.37s	15.56s	2.03s
2	init 12288 sessions for 12288 blocks	47.81s	33.91s	55.26s

the same time use the idea of one session for multiple blocks.

5.2 The Integration of the Learned Model

We have integrated two learned models into $HTM16.2$ which is the newest version of the reference software of 3D-HEVC by the time this thesis is written. More specifically, ResNet engines have been integrated to ***TAppEncoder*** in $HTM16.2$ for depth map angular modes [2, 34] prediction and the DMM1 searching process. Wedgelet slope is employed to reduce the number of wedgelet candidates to be evaluated in DMM1 searching process. Since top-15 is used, roughly half of the candidates are skipped during the encoding time. The details of the proposed fast depth coding algorithm is shown in Figure 5.1 on page 59.

Initially, before the encoding of a depth map will start, its Luma pixel values are accessed by the learned models to predict best modes for all the suitable blocks inside it. We call it **Batch Prediction**. The model learned from blocks of size 8×8 is used to predict the best modes for blocks of size 8×8 while the model learned from blocks of size 16×16 is used to predict the best modes for blocks of size 16×16 and size 32×32. For example, in a video of resolution 1024×768, each depth frame contains $12,288$ blocks of size 8×8, $3,072$ blocks of size 16×16 and 768 blocks of size 32×32. During depth map encoding, the learned models perform predictions for all the $12,288$ blocks of size 8×8, $3,072$ blocks of size 16×16 and 768 blocks of size 32×32. The prediction results are stored in a map with three elements comprising size of block, CU position x, and CU position y as the key, and best mode as the value. The information in this map will be retrieved using key to get the corresponding value when recursively encoding CUs in this depth map. For instance, given that the map is name as Map, the CU size is 8×8 and CU position is $[x_c, y_c]$, we can retrieve the best mode of this CU by $BestMode_c = Map[8, x_c, y_c]$. This map will be re-initialized to an empty map every time a new depth frame will appear in the encoding. Why is the implementation predicting best modes for all the $12,288$ blocks of size 8×8, $3,072$ blocks of size 16×16 and 768 blocks of size 32×32 when the depth frame will be encoded instead of predicting best mode for each CU when the CU will be encoded? The first reason is, as shown in Section 5.1, the time cost is not acceptable if the prediction is performed for each CU instead of for the whole frame at each time. The other reason is that when a depth frame will be encoded, the CU partition information is not

known to the encoder. For this reason, it is necessary to perform predictions for all the possible CU candidates to ensure that no matter how the CU partition scheme will be, the information stored in the map will be sufficient to help with the encoding.

Next, for each depth block of size 8×8, 16×16 and 32×32, top-15 modes together with PLANAR, DC modes are added to a candidate list for Rough Mode Decision (RMD). For blocks of other sizes, the original encoder workflow is applied. If mode 2 is inside the RMD List, mode 34 will be added to RMD List due to the reason that it is of the same angle as mode 2. Else if mode 2 is not inside the RMD List, mode 34 will not be added to RMD List. The top-15 modes here is constructed by making reference to the top-1 mode. For example, if the top-1 mode is mode 10, the adjacent 14 modes will be added to the list of top-15. Hence the list of modes will be [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. After that, PLANAR and DC modes will be added to the list of modes to form RMD List. In our case, mode 33 and mode 2 are considered to be adjacent to each other. Therefore, if the top-1 mode is mode 2, the list of modes will be [9, 8, 7, 6, 5, 4, 3, 2, 33, 30, 29, 28, 27, 26, 25]. After that, mode 34, PLANAR and DC modes will be added to RMD List. This method is based on the observations that the top-k predictions in confusion matrix will always be the adjacent ones.

Then Sum of Absolute Transform Difference (SATD) is used to obtain the best 3 candidates for blocks of size 8×8 and 16×16, 2 candidates for blocks of size 32×32. The number of candidates to be evaluated further is reduced for blocks of different types comparing with the original implementation, where 8 candidates will be selected for blocks of size 8×8 and 16×16, 3 candidates will be selected for blocks of size 32×32. The decision of candidates reduction in this step is due to the consideration that the model prediction has already narrowed the searching range for mode candidates with a high precision such that there is no need to evaluate a large number of candidates in the further steps.

After SATD, the best prediction of the block is utilized to construct a range of slopes. Denote the index of the top-1 mode from the model prediction as N_1, a slope range will be constructed using the angular modes within $[N_1 - 7, N_1 + 7]$, where the two boundaries are inclusive. In the further step, when performing the DMM1 coarse searching, the wedgelets which have the slopes out of the constructed slope range are excluded from View Synthesis Optimization (VSO) which is the source of the computational complexity in the original encoder. The DMM1 refinement following the coarse search is unmodified. In the end, the best mode for the depth map to be coded in the encoding process will be obtained. This algorithm can reduce the number of wedgelet candidate by half. Besides, the conventional intra angular mode decision for depth maps is also accelerated using the same prediction from neural networks.

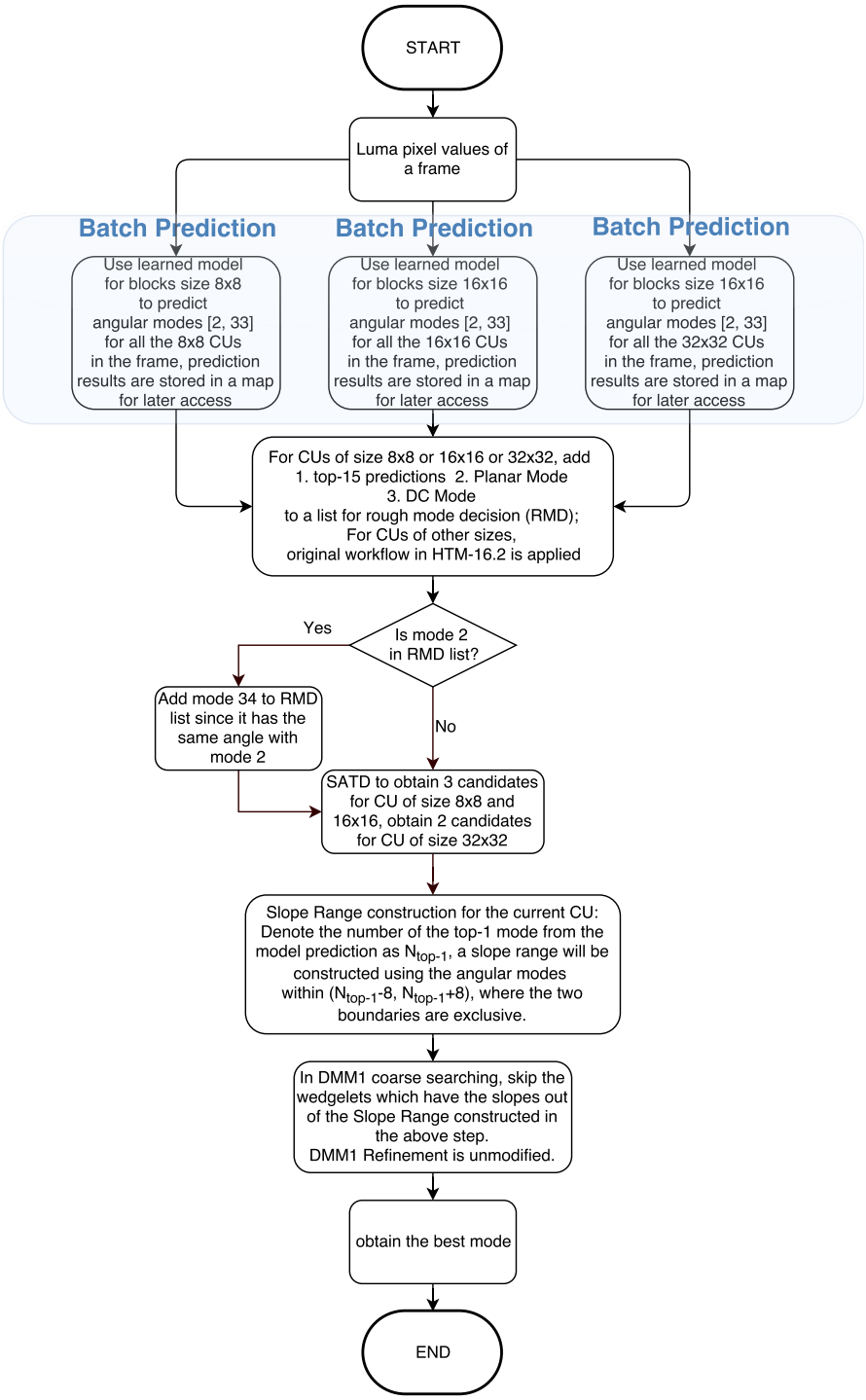


Figure 5.1: Flowchart for proposed fast depth coding algorithm.

5.3 Results of Experiments

The experiments are conducted on four video sequences shown in Table 5.2 to make sure every sample that will be predicted has never been seen by the learned models. Two metrics including **BD-BR** from [67] and **BD-PSNR** from [68] are used in the evaluations for the proposed fast depth coding algorithm. **BD-BR** calculates the average difference between Rate Distortion curves of two sets of synthesized views where one set of views are synthesized from the reconstructed views from original encoder while the other set of views are synthesized from the reconstructed views from the encoder powered by deep learning. **BD-PSNR** is employed to assess the subjective quality of synthesized views. The common test condition defined in [69] is used in our experiments. All the frames from four video sequences in experiments are encoded as I-Frames.

Table 5.2: Video sequences used for experiments

#	Name of the Sequence	Resolution	Usage	Number of Frames
1	Newspaper	1024×768	Simulation	300
2	GhostTownFly	1920×1088	Simulation	250
3	PoznanHall2	1920×1088	Simulation	200
4	Shark	1920×1088	Simulation	300

Table 5.3 shows the time saving for DMM1 wedgelet searching process together with the coding performance. Table 5.4 presents the time saving for total encoding and the coding performance.

Table 5.3: Time saving for wedgelet searching and coding performance of proposed method

	Time saving for DMM1					
Sequences	QP34	QP39	QP42	QP45	BD-BR	BD-PSNR
Newspaper	63.76	64.94	71.98	74.14	0.98%	-0.02dB
PoznanHall2	71.08	71.08	66.36	71.27	1.64%	-0.05dB
GhostTownFly	62.00	56.20	51.60	58.87	0.65%	-0.02dB
Shark	63.55	58.66	63.26	63.34	1.04%	-0.03dB

From the experimental statistics shown in the two tables above, our fast depth coding algorithm achieves an average time reduction of 64.6% in wedgelet searching during 3D-HEVC encoding while the BD performance only has a trivial decrease. It is noticed that the values of **BD-PSNR** almost stayed unchanged comparing with the original implementation from the reference software of 3D-HEVC. The models trained by deep learning were trying their best to keep the originality of the pixel blocks that they have seen during the encoding process. However, the learned models have no idea of how to adjust their prediction results by taking the bitrate into consideration. For this reason, the perceptual visual qualities of the videos are well preserved while the performance on

Table 5.4: Time saving for total encoding and coding performance of proposed method

Sequences	Time saving for total encoding				BD-BR	BD-PSNR
	QP34	QP39	QP42	QP45		
Newspaper	27.33	27.20	27.78	27.38	0.98%	-0.02dB
PoznanHall2	29.00	28.34	28.75	19.04	1.64%	-0.05dB
GhostTownFly	32.52	31.19	31.25	32.07	0.65%	-0.02dB
Shark	38.94	31.26	37.33	36.86	1.04%	-0.03dB

bitrate decreased a little as a penalty. In each single encoding activity for a video sequence, the time saving percent for whole encoding process is lower than the time saving percent for DMM1 but still the overall results achieved by the proposed fast depth coding algorithm can be considered as excellent when comparing with the original implementation of 3D-HEVC.

Chapter 6

Conclusion

In this dissertation we have presented a fast intra mode decision algorithm to address the time cost issue of depth map coding in 3D-HEVC, leveraging the learned models, which are constructed from deep convolutional neural networks, to predict the wedgelet angles for the CUs in the depth maps. The predictions from the learned models are capable of reducing the number of DMM1 wedgelet candidates by half. The size of the neural network has been carefully designed to balance the trade-off between complexity and accuracy of model prediction. Validation precision and confusion matrix are at the core of the stopping criteria. Top-k metric is adopted to make use of the predictions from the learned models. The learned models are integrated into the reference software of 3D-HEVC for the simulations. The compiled executable binaries are able to harness the power of simultaneous computation of CPU, as well as parallel computation of GPU to accelerate the predictions. The simulation results show that the proposed algorithm provides 64.6% time reduction in average for DMM1 wedgelet searching in depth maps while the BD performance has a trivial decrease comparing with the most recent implementation of 3D-HEVC standard.

Bibliography

[1] *Video*, Web Page, 2017. [Online]. Available: http://hidefnj.com/video.html.

[2] C. E. Shannon, *The mathematical theory of communication*. Urbana: Urbana : University of Illinois Press, 1949.

[3] "Itu-t recommendation database," 2017. [Online]. Available: http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=12905&lang=en.

[4] "Jct-vc - joint collaborative team on video coding," 2017. [Online]. Available: http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jctvc.aspx.

[5] I. E. Richardson, *The H.264 Advanced Video Compression Standard*, 2nd ed.., ser. H.264 Advanced Video Compression Standard 2e. Hoboken: Hoboken : Wiley, 2010.

[6] A. Vetro, T. Wiegand, and G. J. Sullivan, "Overview of the stereo and multiview video coding extensions of the h.264/mpeg-4 avc standard," *Proceedings of the IEEE*, vol. 99, no. 4, pp. 626–642, 2011, ISSN: 0018-9219. DOI: 10.1109/JPROC.2010.2098830.

[7] J. Konrad and M. Halle, "3-d displays and signal processing," *IEEE Signal Processing Magazine*, vol. 24, no. 6, pp. 97–111, 2007, ISSN: 1053-5888. DOI: 10.1109/msp.2007.905706.

[8] I. Sexton and P. Surman, "Stereoscopic and autostereoscopic display systems," *Signal Processing Magazine, IEEE*, vol. 16, no. 3, pp. 85–99, 1999, ISSN: 1053-5888. DOI: 10.1109/79.768575.

[9] Mu, amp, X, K. Ller, P. Merkle, and T. Wiegand, "3-d video representation using depth maps," *Proceedings of the IEEE*, vol. 99, no. 4, pp. 643–656, 2011, ISSN: 0018-9219. DOI: 10.1109/JPROC.2010.2091090.

[10] "Jct-3v - joint collaborative team on 3d video coding extension development," 2017. [Online]. Available: http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jct3v.aspx.

[11] G. Tech, K. Ying Chen, J.-R. Muller, A. Ohm, A. Vetro, and A. Ye-Kui Wang, "Overview of the multiview and 3d extensions of high efficiency video coding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 26, no. 1, pp. 35–49, 2016, ISSN: 1051-8215. DOI: 10.1109/TCSVT.2015.2477935.

[12] H.-B. Zhang, C.-H. Fu, Y.-L. Chan, S.-H. Tsang, and W.-C. Siu, "Probability-based depth intra mode skipping strategy and novel vso metric for dmm decision in 3d-hevc," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2016, ISSN: 1051-8215. DOI: 10.1109/TCSVT.2016.2612693.

[13] H. Dou, Y.-L. Chan, K.-B. Jia, and W.-C. Siu, "Segment-based view synthesis optimization scheme in 3d-hevc," *Journal of Visual Communication and Image Representation*, vol. 42, pp. 104–111, 2017, ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2016.11.012.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, ser. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016, xxii, 775 pages, ISBN: 9780262035613 0262035618.

[15] B. Schlkopf, C. J. C. Burges, and A. J. Smola, *Advances in kernel methods : support vector learning*. Cambridge, Mass. ; London: Cambridge, Mass. ; London : MIT Press, 1999.

[16] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks : the official journal of the International Neural Network Society*, vol. 61, p. 85, 2015.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[18] *Bazel*, Web Page, 2017. [Online]. Available: https://www.bazel.build/.

[19] *Google cloud*, Web Page, 2017. [Online]. Available: https://cloud.google.com/storage/.

[20] C. Y. Chou, "Advances in grid and pervasive computing: First international conference, gpc 2006," in. 2006, ISBN: 3540338098. [Online]. Available: https://en.wikipedia.org/wiki/Bandwidth_(computing)#Network_bandwidth_capacity.

[21] M. Ghanbari, *Video coding: an introduction to standard codecs*. London: Institution of Electrical Engineers, 1999.

[22] Y. A. LeCun, L. Bottou, G. B. Orr, and K. R. Mller, "Efficient backprop," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700, pp. 9–48, 2012, ISSN: 03029743. DOI: 10.1007/978-3-642-35289-8-3.

[23] *Imagenet large scale visual recognition competition*, Web Page, 2017. [Online]. Available: http://www.image-net.org/challenges/LSVRC/.

[24] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017, ISSN: 0001-0782. DOI: 10.1145/3065386.

[25] Generic, 2014. DOI: 10.1007/978-3-319-10590-1_53.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[27] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," *CoRR*, vol. abs/1310.6343, 2013. [Online]. Available: `http://arxiv.org/abs/1310.6343`.

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014, ISSN: 1532-4435.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015.

[31] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016.

[32] K. Muller, P. Merkle, G. Tech, and T. Wiegand, *3d video coding with depth modeling modes and view synthesis optimization*, Generic, 2012.

[33] S. H. Tsang, Y. L. Chan, and W. C. Siu, "Efficient intra prediction algorithm for smooth regions in depth coding," *Electronics Letters*, vol. 48, no. 18, pp. 1–2, 2012, ISSN: 00135194. DOI: `10.1049/el.2012.1768`.

[34] F. Jager, *Simplified depth map intra coding with an optional depth lookup table*, Generic, 2012. DOI: `10.1109/IC3D.2012.6615142`.

[35] Z. Mengmeng, Z. Chuan, X. Jizheng, and B. Huihui, *A fast depth-map wedgelet partitioning scheme for intra prediction in 3d video coding*, Generic, 2013. DOI: `10.1109/ISCAS.2013.6572473`.

[36] Z. Gu, J. Zheng, N. Ling, and P. Zhang, *Fast bi-partition mode selection for 3d hevc depth intra coding*, Generic, 2014. DOI: `10.1109/ICME.2014.6890324`.

[37] P. Merkle, K. Muller, and T. Wiegand, *Coding of depth signals for 3d video using wedgelet block segmentation with residual adaptation*, Generic, 2013. DOI: `10.1109/ICME.2013.6607429`.

[38] Z. Gu, J. Zheng, N. Ling, and P. Zhang, *Fast depth modeling mode selection for 3d hevc depth intra coding*, Generic, 2013. DOI: `10.1109/ICMEW.2013.6618267`.

[39] G. Sanchez, M. Saldanha, G. Balota, B. Zatt, M. Porto, and L. Agostini, *Complexity reduction for 3d-hevc depth maps intra-frame prediction using simplified edge detector algorithm*, Generic, 2014. DOI: `10.1109/ICIP.2014.7025649`.

[40] H. R. Tohidypour, M. T. Pourazad, and P. Nasiopoulos, *A low complexity mode decision approach for hevc-based 3d video coding using a bayesian method*, Generic, 2014. DOI: `10.1109/ICASSP.2014.6853726`.

[41] H. Zhang, S. H. Tsang, Y. L. Chan, C. H. Fu, and W. C. Siu, *Early determination of intra mode and segment-wise dc coding for depth map based on hierarchical coding structure in 3d-hevc*, Generic, 2015. DOI: 10.1109/APSIPA.2015.7415297.

[42] C. S. Park, "Edge-based intramode selection for depth-map coding in 3d-hevc," *IEEE Trans. Image Process.*, vol. 24, no. 1, pp. 155–162, 2015, ISSN: 1057-7149. DOI: 10.1109/TIP.2014.2375653.

[43] C. S. Park, "Efficient intra-mode decision algorithm skipping unnecessary depth-modelling modes in 3d-hevc," *Electronics Letters*, vol. 51, no. 10, pp. 756–758, 2015, ISSN: 00135194. DOI: 10.1049/el.2014.3874.

[44] H.-B. Zhang, C.-H. Fu, Y.-L. Chan, S.-H. Tsang, W.-C. Siu, and W.-M. Su, "Efficient wedgelet pattern decision for depth modeling modes in three-dimensional high-efficiency video coding," *Journal of Electronic Imaging*, vol. 25, no. 3, p. 033 023, 2016, ISSN: 1017-9909. DOI: 10.1117/1.JEI.25.3.033023.

[45] C. H. Fu, H. B. Zhang, W. M. Su, S. H. Tsang, and Y. L. Chan, "Fast wedgelet pattern decision for dmm in 3d-hevc," in *2015 IEEE International Conference on Digital Signal Processing (DSP)*, pp. 477–481, ISBN: 1546-1874. DOI: 10.1109/ICDSP.2015.7251918.

[46] H. B. Zhang, Y. L. Chan, C. H. Fu, S. H. Tsang, and W. C. Siu, "Quadtree decision for depth intra coding in 3d-hevc by good feature," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1481–1485. DOI: 10.1109/ICASSP.2016.7471923.

[47] Q. Zhang, M. Chen, X. Huang, N. Li, and Y. Gan, "Low-complexity depth map compression in hevc-based 3d video coding," *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, pp. 1–14, 2015. DOI: 10.1186/s13640-015-0058-5.

[48] K. K. Peng, J. C. Chiang, and W. N. Lie, "Low complexity depth intra coding combining fast intra mode and fast cu size decision in 3d-hevc," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 1126–1130. DOI: 10.1109/ICIP.2016.7532533.

[49] P. Liu, G. He, S. Xue, and Y. Li, "A fast mode selection for depth modelling modes of intra depth coding in 3d-hevc," *VCIP 2016 - 30th Anniversary of Visual Communication and Image Processing*, pp. 0–3, 2017, ISSN: 9781509053162. DOI: 10.1109/VCIP.2016.7805548.

[50] Q. Zhang, N. Zhang, T. Wei, K. Huang, X. Qian, and Y. Gan, "Fast depth map mode decision based on depthtexture correlation and edge classification for 3d-hevc," *Journal of Visual Communication and Image Representation*, vol. 45, pp. 170–180, 2017, ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2017.03.004.

[51] G. Sanchez, M. Saldanha, G. Balota, B. Zatt, M. Porto, and L. Agostini, "Dmmfast: A complexity reduction scheme for three-dimensional high-efficiency video coding intraframe depth map coding," *Journal of Electronic Imaging*, vol. 24, no. 2, p. 023 011, 2015, ISSN: 1017-9909. DOI: 10.1117/1.JEI.24.2.023011.

[52] G. Sanchez, L. Jordani, C. Marcon, and L. Agostini, "Dfps: A fast pattern selector for depth modeling mode 1 in three-dimensional high-efficiency video coding standard," *J. Electron. Imaging*, vol. 25, no. 6, 2016, ISSN: 1017-9909. DOI: 10.1117/1.JEI.25.6.063011.

[53] Y. Zhang, S. Kwong, X. Wang, H. Yuan, Z. Pan, and L. Xu, "Machine learning-based coding unit depth decisions for flexible complexity allocation in high efficiency video coding," *Image Processing, IEEE Transactions on*, vol. 24, no. 7, pp. 2225–2238, 2015, ISSN: 1057-7149. DOI: 10.1109/TIP.2015.2417498.

[54] Z. Liu, X. Yu, Y. Gao, S. Chen, X. Ji, and D. Wang, "Cu partition mode decision for hevc hardwired intra encoder using convolution neural network," *Image Processing, IEEE Transactions on*, vol. 25, no. 11, pp. 5088–5103, 2016, ISSN: 1057-7149. DOI: 10.1109/TIP.2016.2601264.

[55] M. Xu, T. Li, Z. Wang, X. Deng, and Z. Guan, "Reducing complexity of HEVC: A deep learning approach," *CoRR*, vol. abs/1710.01218, 2017. arXiv: 1710.01218. [Online]. Available: http://arxiv.org/abs/1710.01218.

[56] T. Laude and J. Ostermann, "Deep learning-based intra prediction mode decision for hevc," ISBN: 9781509059669. DOI: 10.1109/PCS.2016.7906399.

[57] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 396–404. [Online]. Available: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf.

[58] G. J. Sullivan, J. Ohm, T. Woo-Jin Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012, ISSN: 1051-8215. DOI: 10.1109/TCSVT.2012.2221191.

[59] *Official 3d-hevc repository*, Web Page, 2017. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/tags/HTM-16.2/.

[60] Y. LeCun, C. Cortes, and C. J. Burges, *The mnist database of handwritten digits*, Web Page. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[61] A. Krizhevsky, V. Nair, and G. Hinton., *The cifar-10 and cifar-100 datasets*, Web Page. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[62] *Confusion matrix*, Web Page, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Confusion_matrix.

[63] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009, ISSN: 1041-4347. DOI: 10.1109/TKDE.2008.239.

[64] K. He, X. Zhang, S. Ren, and J. Sun, *Identity mappings in deep residual networks*, Generic, 2016. DOI: 10.1007/978-3-319-46493-0_38.

[65] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[66] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[67] G. Bjontegaard, *Calculation of average psnr differences between rd curves*, Generic, 2001.

[68] C. Chun-Hsien, "A perceptually tuned subband image coder based on the measure of just-noticeable-distortion profile," in *Proceedings of 1994 IEEE International Symposium on Information Theory*, p. 420. DOI: 10.1109/ISIT.1994.395035.

[69] F. Jager, *Description of core experiment 5 (ce5) on depth intra modes*, Generic, 2013.