LOW COMPLEXITY H.264 ENCODER USING MACHINE LEARNING FOR STREAMING APPLICATIONS

by

SUCHETHAN SWAROOP VAIDYANATH

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON
May 2011

ACKNOWLEDGEMENTS

This thesis would not have been possible without the encouragement, inputs, guidance and support of a number of people. First and foremost, I would like to thank my thesis advisor Dr.K.R.Rao for taking me under his wings right from the first semester. His zeal to try new trends and his expertize in the field has always been an inspiration for me. I am truly grateful to him for giving me such a life changing opportunity.

I would like to thank Dr.Dongil Han for his expert guidance and financial support. Working with him has been an invaluable experience. I would like to thank all my team mates at Ericsson Inc for guiding me during my internship. Their constant support helped me in completing my thesis. My sincere thanks to Thejaswini, Pragnesh, Aruna , Vineeth, Vidya and Jennie for their inputs on this research.

Last but not least, I would like to thank my Mom and Dad and my brothers for being my backbone in whatever I have done till now.

April 8, 2011

ABSTRACT

LOW COMPLEXITY H.264 ENCODER USING MACHINE LEARNING FOR STREAMING APPLICATIONS

Suchethan Swaroop Vaidyanath, M.S

The University of Texas at Arlington, 2011

Supervising Professor: K.R.Rao

H.264, MPEG-4 part-10 or AVC, is the latest digital video codec standard which has proven to be superior than earlier standards in terms of compression ratio, quality, bit rates and error resilience [1]. Joint model (JM) reference software is used for academic reference and it was developed by the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG (Video coding experts group). The Intel IPP H.264 (Integrated Performance Primitives) is a product of Intel which uses Intel IPP libraries and SIMD instructions available on modern processors. The Intel IPP H.264 is multi-threaded and uses CPU optimized IPP routines. These two softwares are compared in terms of execution time and video quality of the decoded sequences. The metrics used for comparison are SSIM (Structural Similarity Index Metric), PSNR (Peak-to-Peak Signal to Noise Ratio), MSE (Mean Square Error), motion estimation time, encoding time, decoding time and the compression ratio of the H.264 file size (encoded output). The compression ratio of H.264 file is found to be less in JM software at various bit rates than in Intel IPP. Hence, it is preferred over Intel IPP for reduction in the motion estimation time during encoding.

Motion estimation takes about 60 to 70 percent of the encoding time. The time consuming Sum of Absolute Differences (SAD) method adopted in the H.264 encoder in JM 16.2 software is replaced with a classification rule using machine learning. This tree is implemented in the form of if-else statements in the motion estimation block of JM16.2. Hence, the motion estimation process is reduced to if else statements thereby reducing the encoding time. H.264 is a video codec format. Its corresponding .AAC (Advanced Audio Coding) audio format and the video format are then placed on a MP4 container using an open source tool called MP4box. This MP4 file can be streamed (after forming manifest files) using IIS (Internet Information Services) to achieve smooth low complexity streaming of media over the Internet.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Digital video has become the main stream and is being used in a wide range of applications including DVD, digital TV, HDTV, video telephony, and teleconferencing [7]. These digital video applications are feasible because of the advances in computing and communication technologies as well as efficient video compression algorithms. The rapid deployment and adoption of these technologies was possible primarily because of standardization and the economies of scale brought about by competition and standardization. Most of the video compression standards are based on a set of principles that reduce the redundancy in digital video.

Consider a digital video sequence at a standard definition TV picture resolution of 720x480 and a frame rate of 30 frames per second (FPS). If a picture is represented using the YUV color space with 8 bits per component or 3 bytes per pixel, size of each frame is 720x480x3 bytes. The disk space required to store one second of video is 720x480x3x30 = 31.1 MB. A one hour video would thus require 112 GB. To deliver video over wired and/or wireless networks, minimum bandwidth required is (31.18)/2 = 124.4 MHZ. In addition to these extremely high storage and bandwidth requirements, using uncompressed video will add significant cost to the hardware and systems that process digital video. Digital video compression is thus necessary even with exponentially increasing bandwidth and storage capacities. Fortunately, digital video has significant redundancies and eliminating or reducing those redundancies results in compression. Video compression is typically achieved by exploiting

- Spatial
- Temporal
- and statistical redundancies.

1.1 H.264

H.264 or AVC (Advanced Video Coding) is a digital video codec standard which is noted for achieving very high data compression [1]. It was developed by the ITU-T Video Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) as the product of a collective partnership effort known as the Joint Video Team (JVT) [1]. H.264 delivers stunning quality at remarkably low data rates. Ratified as part of the MPEG-4 standard (MPEG-4 Part 10), this ultra-efficient technology gives excellent results across a broad range of bandwidths, from 3G (3rd Generation) for mobile devices to iChat (instant chatting application) for video conferencing to HD for broadcast and DVD [6].

Massive Quality, Minimal Files

H.264 uses the latest innovations in video compression technology to provide incredible video quality from the smallest amount of video data. This means that crisp and clear videos can be watched in much smaller files, saving bandwidth and storage costs over previous generations of video codecs. H.264 delivers the same quality as MPEG-2 at a third to half the data rate and up to four times the frame size of MPEG-4 Part 2 at the same data rate [6].

Scalable [8] from 3G to HD and Beyond

H.264 achieves the best-ever compression efficiency for a broad range of applications, such as broadcast, DVD, video conferencing, video-on-demand, streaming and multimedia messaging. True to its advanced design, H.264 delivers excellent quality across a wide operating range, from 3G to HD and everything in between. Whether the need is high-quality video for a mobile phone, iChat, Internet, broadcast or satellite delivery, H.264 provides exceptional performance at impressively low data rates. Table 1.1 shows data rates used at various resolutions.

Table 1.1. H.264 data rates at various resolutions [6]

Use Scenario	Resolution and frame rate	Example data rates
Mobile Content	176x144, 10-15 fps	50-60Kbps
Internet/Standard Definition	352x288, 640x480, 24 fps	1-2Mbps
High Definition	1280x720, 24fps,	5-6 Mbps
Full High Definition	1920x1080, 24p, 24fps,	7-8 Mbps

1.2 The New Industry Standard

Already ratified as part of the MPEG-4 standard MPEG-4 Part 10 and the ITU-Ts latest video-conferencing standard, H.264 is now mandatory for the HD-DVD and Blu-ray specifications (the two formats for high-definition DVDs) and ratified in the latest versions of the DVB (Digital Video Broadcasters) and 3GPP (3rd Generation Partnership Project) standards. Numerous broadcast, cable, videoconferencing and consumer electronics companies consider H.264 the video codec of choice for their new products and services [6].

In this thesis, performance analysis and comparison of JM and Intel IPP H.264 softwares are implemented. Machine learning has been used to reduce the encoding time. The video multiplexed with audio can then be streamed thereby achieving low complexity streaming of media over the Internet.

CHAPTER 2

PROFILES AND WORKING OF H.264 CODEC

2.1 The AVC/H.264 profiles

The H.264/AVC standard includes the following sets of capabilities, which are referred to as profiles. They target specific classes of applications [9]:

- Constrained Baseline Profile (CBP): It is primarily for low-cost applications. This profile is used widely in videoconferencing and mobile applications. It corresponds to the subset of features that are common between the Baseline, Main, and High Profiles.
- Baseline Profile (BP): It is primarily for low-cost applications [10] that require additional error robustness. This profile is used rarely in videoconferencing and mobile applications, and it adds additional error resilience tools to the Constrained Baseline Profile. The importance of this profile is fading after the Constrained Baseline Profile has been defined.
- Main Profile (MP): This was originally intended as the mainstream consumer profile for broadcast and storage applications. The importance of this profile faded when the High profile was developed for these applications.
- Extended Profile (XP): This was intended as the streaming video profile. This profile has relatively high compression capability. It has some extra tricks for robustness to data losses and server stream switching.
- High Profile (HiP): This is the primary profile for broadcast and disc storage applications, particularly for high-definition television applications. This is the

profile adopted into HD DVD and Blu-ray Disc.There are four High Profiles (Fidelity range extensions) [21]. They are:

- High Profile: To support the 8-bit video with 4:2:0 sampling for applications using high resolution.
- High 10 Profile (Hi10P): Going beyond today's mainstream consumer product capabilities, this profile builds on top of the High Profile, adding support for up to 10 bits per sample of decoded picture precision.
- High 4:2:2 Profile (Hi422P): This profile primarily targets professional applications that use interlaced video. It builds on top of the High 10 Profile, adding support for the 4:2:2 chroma subsampling format while using up to 10 bits per sample of decoded picture precision.
- High 4:4:4 Predictive Profile (Hi444PP): This profile builds on top of the High 4:2:2 Profile, supporting up to 4:4:4 chroma sampling, up to 14 bits per sample, and additionally supporting efficient lossless region coding and the coding of each picture as three separate color planes.

Figure 2.1 shows the various profiles of H.264.

2.1.1 Common coding parts for all profiles

The common coding parts for the profiles are listed below [2]:

- I slice (Intra-coded slice): coded by using prediction only from decoded samples within the same slice.
- P slice (Predictive-coded slice): coded by using inter prediction from previously decoded reference pictures, using at most one motion vector and reference index to predict the sample values of each block [11].
- CAVLC (Context-based Adaptive Variable Length Coding) for entropy coding.

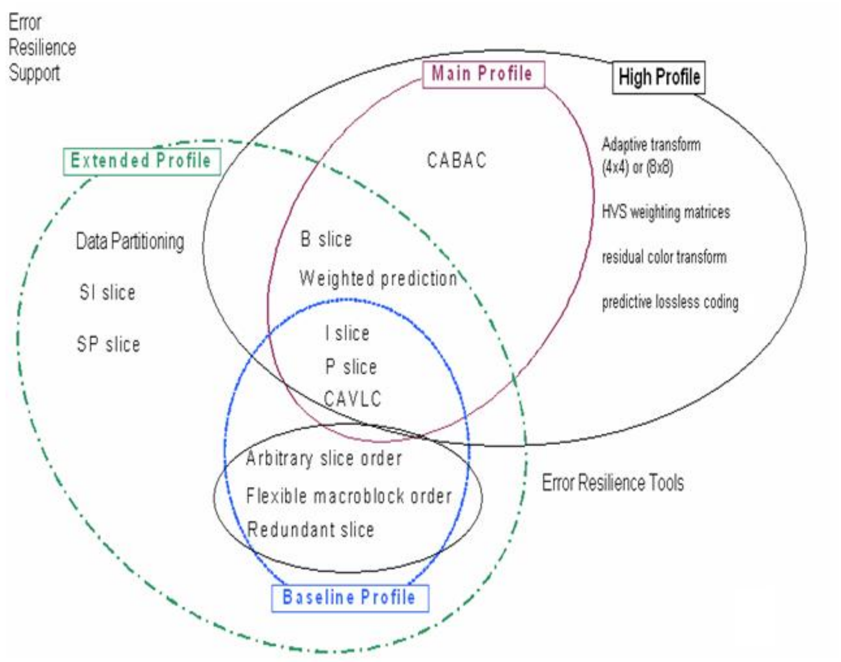


Figure 2.1. Various profiles of H.264.

2.1.2 Baseline Profile

The coding parts for the baseline profile are listed below:

- Common parts: I slice, P slice, CAVLC.
- FMO Flexible macroblock order: macroblocks may not necessarily be in the raster scan order. The map assigns macroblocks to a slice group.
- ASO Arbitrary slice order: the macroblock address of the first macroblock of a slice of a picture may be smaller than the macroblock address of the first macroblock of some other preceding slice of the same coded picture.
- RS Redundant slice: This slice belongs to the redundant coded data obtained by same or different coding rate, in comparison with previous coded data of same slice.

2.1.3 Extended Profile

The coding parts for the extended profile are listed below:

- Common parts : I slice, P slice, CAVLC.
- SP slice : specially coded for efficient switching between video streams, similar to coding of a P slice.
- SI slice: switched, similar to coding of an I slice.
- Data partition: the coded data is placed in separate data partitions, each partition can be placed in different layer unit.
- Flexible macroblock order (FMO), arbitrary slice order (ASO).
- Redundant slices (RS), B slice.
- Weighted prediction.

2.1.4 Main Profile

The coding parts for the main profile are listed below:

- Common parts: I slice, P slice, CAVLC.
- B slice (Bi-directionally predictive-coded slice): the coded slice by using inter prediction from previously-decoded reference pictures, using at most two motion vectors and reference indices to predict the sample values of each block.
- Weighted prediction: scaling operation by applying a weighting factor to the samples of motion-compensated prediction data in P or B slice.
- CABAC (Context-based Adaptive Binary Arithmetic Coding) for entropy coding.

2.1.5 High Profile

The coding parts for the high profile are listed below:

- Adds to the main profile

- 8x8 intra prediction
- Custom quantization
- Lossless video coding
- More YUV formats (4:4:4)

2.2 Encoder

The H.264 encoder includes two dataflow paths, a forward path and a reconstruction path.

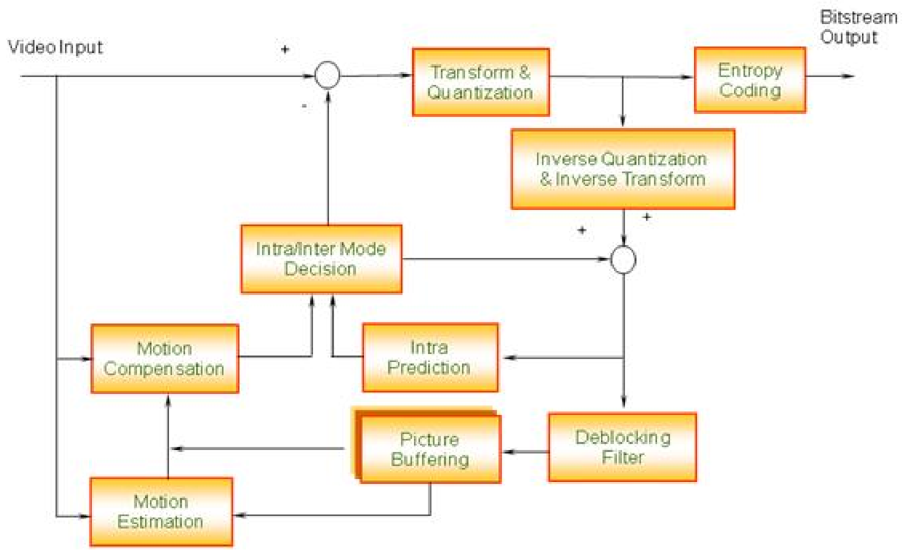


Figure 2.2. Encoder block diagram of H.264 [3].

Encoder (Forward Path)

An input frame is presented for encoding as shown in Figure 2.2. The frame is processed in units of a macroblock (corresponding to 16x16 pixels in the original image).

Each macroblock is encoded in intra or inter mode. In either case, a predicted macroblock P is formed based on a reconstructed frame. In intra mode, P is formed from samples in the current frame that have been previously encoded, decoded and reconstructed form P. The unfiltered samples are used to form P. In inter mode, P is formed by motion-compensated prediction from one or more reference frame(s). The prediction for each macroblock may be formed from one or more past or future frames (in time order) that have already been encoded and reconstructed [3].

The prediction P is subtracted from the current macroblock to produce a residual or difference macroblock. This is transformed (using a block transform) and quantized to give a set of quantized transform coefficients. These coefficients are reordered and entropy encoded. The entropy encoded coefficients, together with side information required to decode the macroblock (such as the macroblock prediction mode, quantizer step size, motion vector information describing how the macroblock was motion-compensated, etc) form the compressed bitstream. This is passed to a network abstraction layer (NAL) for transmission or storage.

Encoder (Reconstruction path)

As illustrated in Figure 2.2, the quantized macroblock coefficients are decoded in order to reconstruct a frame for encoding of further macroblocks. The coefficients are re-scaled and inverse transformed to produce a difference macroblock. This is not identical to the original difference macroblock. The quantization process introduces losses. The predicted macroblock P is added to the difference macroblock to create a reconstructed macroblock (a distorted version of the original macroblock). A deblocking filter is applied to reduce the effects of blocking distortion and reconstructed reference frame is created from a series of macroblocks [12].

2.3 Decoder

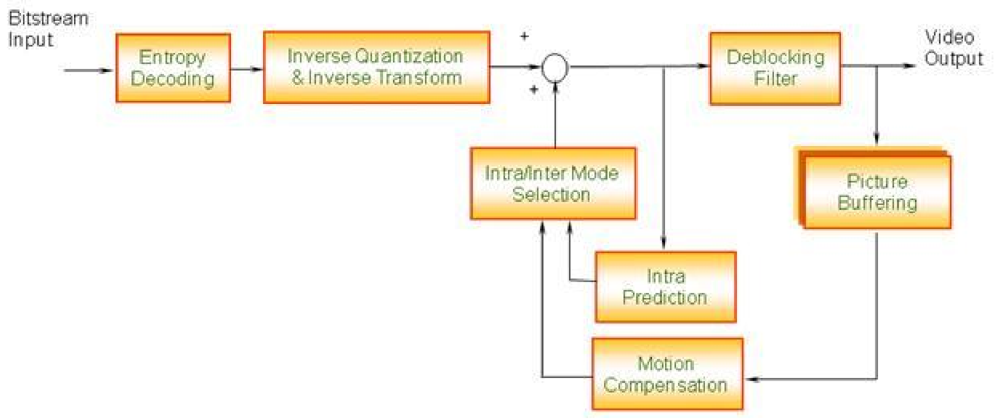


Figure 2.3. Decoder block diagram of H.264 [3].

The decoder receives a compressed bitstream from the NAL as shown in Figure 2.3. The data elements are entropy decoded and reordered to produce a set of quantized coefficients. These are rescaled and inverse transformed to give a difference macroblock. Using the header information decoded from the bit stream, the decoder creates a prediction macroblock P, identical to the original prediction P formed in the encoder. P is added to the difference macroblock and this result is given to the deblocking filter to create the decoded macroblock.

The purpose of the reconstruction path in the encoder is to ensure that both encoder and decoder use identical reference frames to create the prediction P. If this is not the case, then the predictions P in encoder and decoder will not be identical, leading to an increasing error or drift between the encoder and decoder [3].

2.4 Inter Prediction

Inter prediction includes both motion estimation (ME) and motion compensation (MC). The ME/ MC process performs prediction [13]. It generates a predicted version of a rectangular array of pixels, by choosing another similarly sized rectangular array of pixels from a previously decoded reference picture and translating the reference array to the position of the current rectangular array. The translation from other positions of the array in the reference picture is specified with quarter pixel precision.

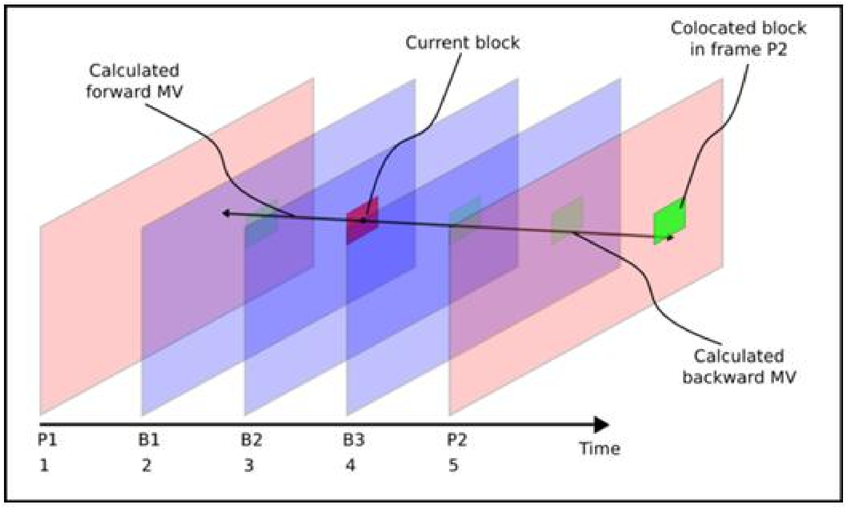


Figure 2.4. Multi-frame bidirectional motion compensation in H.264 [4].

H.264/AVC supports multi-picture motion-compensated prediction [14]. That is, more than one prior-coded picture can be used as a reference for motion-compensated prediction as shown in Figure 2.4. Up to 16 frames can be used as reference frames. In addition to the motion vector, the picture reference parameters are also transmitted.

Both the encoder and decoder have to store the reference pictures used for Inter-picture prediction in a multi-picture buffer. The decoder replicates the multi-picture buffer of the encoder, according to the reference picture buffering type and any memory management control operations that are specified in the bit stream. B frames use both a past and future frame as a reference. This technique is called bidirectional prediction. B frames provide most compression and also reduce the effect of noise by averaging two frames. B frames are not used in the baseline profile (Figure 1.1).

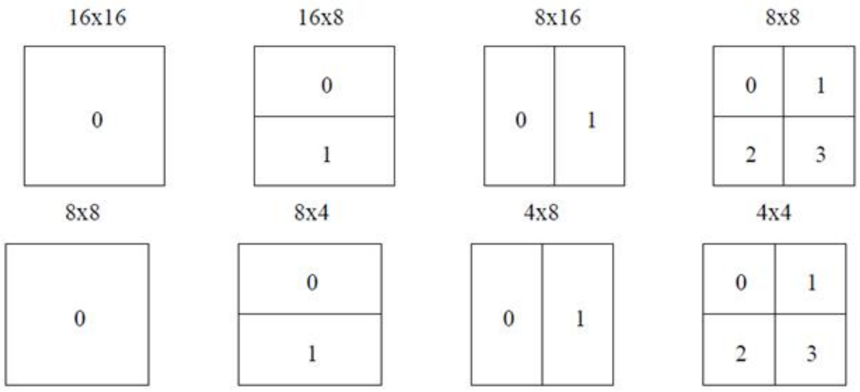


Figure 2.5. Block sizes used for motion compensation [5].

H.264 supports motion compensation block sizes ranging from 16x16 to 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 sizes as shown in Figure 2.5. This method of partitioning macroblocks into motion compensated sub-blocks of varying sizes is known as tree structured motion compensation.

2.5 Intra Prediction

Intra prediction [15] exploits spatial redundancy between adjacent macroblocks in a frame. It predicts the pixel values as linear interpolation of pixels from adjacent

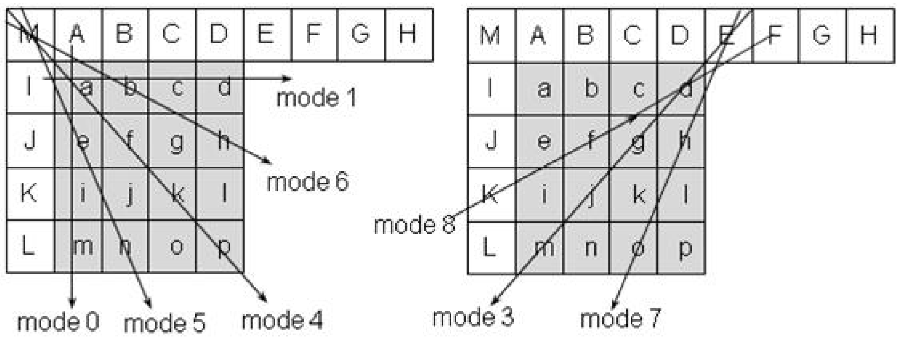


Figure 2.6. Intra 4*4 prediction modes and prediction directions.

edges of neighboring macroblocks that are decoded before the current macroblock. The interpolations are directional in nature, with multiple modes, each implying a spatial direction of prediction as shown in Figure 2.6. There are 9 prediction modes defined for a 4x4 block and 4 prediction modes defined for a 16x16 block. The union of all mode evaluations, cost comparisons and exhaustive search inside motion estimation(ME) causes a great amount of time spent by the encoder. Complex and exhaustive ME evaluation is the key to good performance achieved by H.264, but the cost is in the encoding time [16].

2.6 Test Sequences

- CIF (Common Intermediate Format) as illustrated in Figure 2.7 is a video format used in video conferencing systems. CIF is part of the ITU H.261 video-conferencing standard. It specifies a data rate of 30 frames per second (fps), with each frame containing 288 lines and 352 pixels per line. Hence it has a resolution of 352 x 288.
- QCIF (Quarter CIF) as illustrated in Figure 2.8 is related to CIF. Its resolution per frame is 144 lines, with 176 pixels per line. Multiplying 352 and 288 yields 101,376 total number of pixels in the CIF format, which is exactly four times the

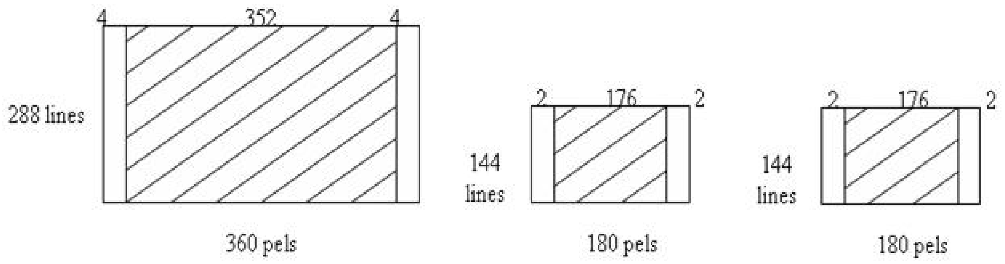


Figure 2.7. Supporting picture format-4:2:0 chroma sampling for CIF.

Table 2.1. Basic information for CIF sequence "football.yuv" [2]

Frame rate(fps)	30
Width	352
Height	258
Size(Mega bytes)	13
Number of Frames	90

number of pixels contained in a QCIF frame (25,344 pixels). The "Quarter" terminology is meant to indicate that QCIF frames contain quarter as many pixels as the CIF frame and thus take up less bandwidth.

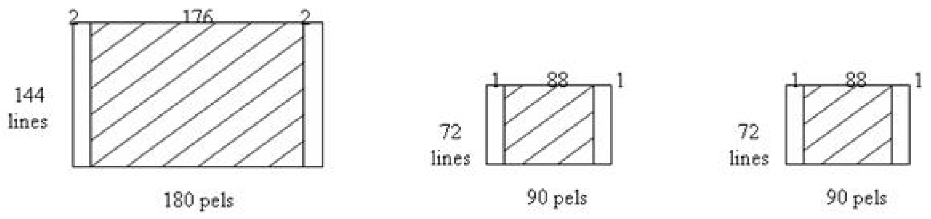


Figure 2.8. Supporting picture format:4:2:0 chroma sampling for QCIF.

Table 2.2. Basic information for QCIF sequence “foreman.yuv” [2]

Frame rate(fps)	30
Width	176
Height	144
Size(Mega bytes)	11.138
Number of Frames	300

2.7 Joint Model (JM) Reference Software

The JM [17] reference software is used to implement the H.264 codec. It has a configuration file through which the input parameters can be given such as the input sequence, frame rate, video resolution of the input sequence, bit rate, quantization parameter, profile to be used etc.

The command used to execute the H.264 encoder is:

- lencod f encoder.cfg

  The command used to execute the H.264 decoder is:

- ldecod i testfile.264 o testoutput.yuv

2.8 Intel IPP H.264 compiler

Intel Integrated Performance Primitives (Intel IPP) is an extensive library of highly optimized software functions for digital media and data-processing applications. Intel IPP [18] offers thousands of optimized functions covering frequently-used fundamental algorithms. Intel IPP functions are designed to deliver performance beyond what optimized compilers alone can deliver. It also has a parameter file called h264.par wherein the input parameters can be given.

The command used to execute the H.264 encoder is:

- umc_video_enc_con.exe h264 h264.par testfile.h264

  The command used to execute the H.264 decoder is:

- umc_h264_dec_con.exe -i testfile.h264 -o testoutput.yuv

2.9 Input Parameters

The parameters that were changed in the configuration file (JM) and the parameter file (Intel IPP) are

- Input sequence
- Frame width and height
- Frame rate
- YUV format
- Number of frames to be encoded
- Profile
- Bit rate
- Quantization parameter
- Intra period
- Number of reference frames to be used
- CABAC and bi-directional prediction are not included (as baseline profile is being used).

2.10 Summary

In this chapter, the working of the encoder and decoder are explained. The test sequences and the common parameters chosen to be used in both JM and Intel IPP H.264 encoders are also discussed. The next chapter is about the test results obtained from these two softwares.

CHAPTER 3

RESULTS OBTAINED FROM JM AND INTEL IPP

3.1 Results for ”Foreman.yuv” sequence from JM 16.2 and Intel IPP

The results obtained from JM are tabulated in Table 3.1 The results obtained from Intel IPP is tabulated in Table 3.2

Table 3.1. Results obtained from JM 16.2 for Foreman.yuv QCIF sequence

Bit rates used (KB/sec)	Motion estimation time (sec)	Encoding time (sec)	Decoding time (sec)	H.264 file size	Compression ratio
25	1396.664	1658.874	15.824	251 KB	44.37
50	1426.045	1829.835	18.384	501 KB	22.23
75	1442.001	1951.315	19.059	751 KB	14.83
100	1376.785	1941.312	17.805	1.001 MB	11.12
125	1398.593	2045.541	19.375	1.251 MB	8.9
150	1509.317	2280.951	19.307	1.501 MB	7.42

Table 3.2. Results obtained from Intel IPP H.264

Bit rates used (KB/sec)	Motion estimation time (sec)	Encoding time (sec)	Decoding time (sec)	H.264 file size	Compression ratio
25	2.51	3.41	0.1257	294 KB	37.92
50	2.58	3.31	0.1519	562 KB	19.82
75	3.07	3.93	0.1061	824 KB	13.52
100	2.88	3.74	0.1232	1.082 MB	10.3
125	2.94	3.77	0.1386	1.334 MB	8.35
150	3.25	4.13	0.1577	1.583 MB	7.04

3.1.1 Encoding time and ME time taken by JM 16.2

Figure 3.1 illustrates the comparison of encoding time and ME time taken by JM 16.2.

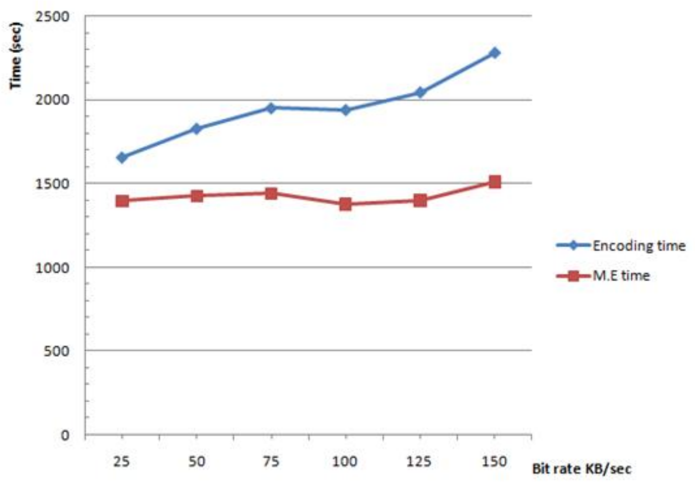


Figure 3.1. Encoding time and ME time taken by JM 16.2.

3.1.2 Encoding time and ME time taken by Intel IPP

Figure 3.2 illustrates the comparison of encoding time and ME time taken by Intel IPP for Foreman sequence.

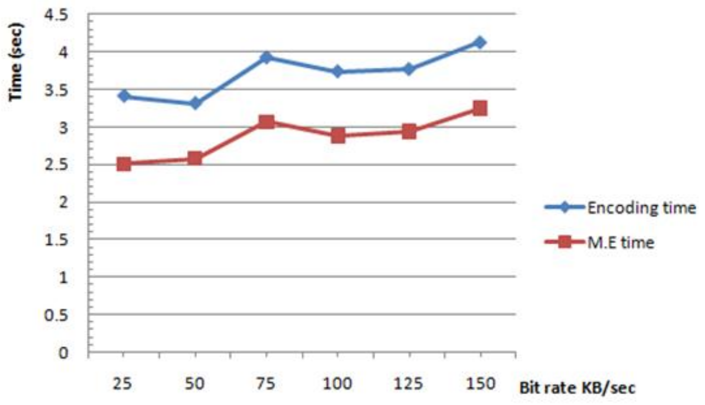


Figure 3.2. Encoding time and ME time taken by Intel IPP H.264.

3.1.3 Decoding time taken by JM 16.2 for Foreman sequence

Figure 3.3 illustrates the decoding time taken by JM 16.2.

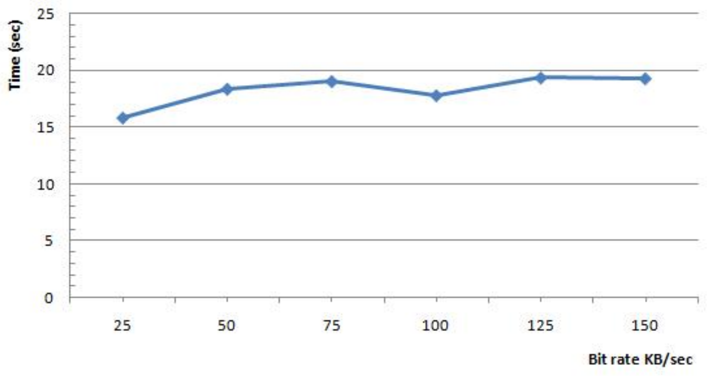


Figure 3.3. Decoding time taken by JM 16.2 for variable bit rates.

3.1.4 Decoding time taken by Intel IPP for Foreman sequence

Figure 3.4 illustrates the decoding time taken by Intel IPP.

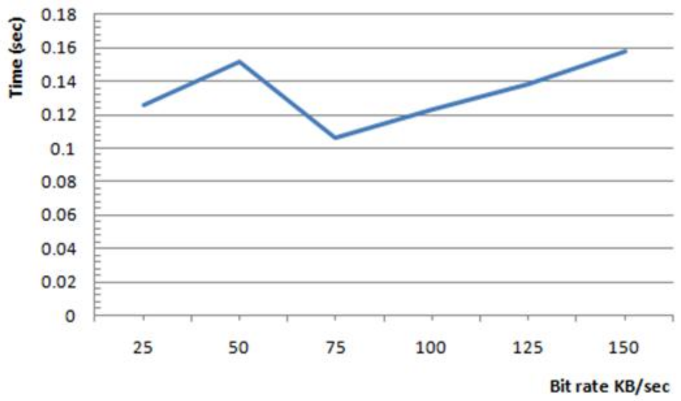


Figure 3.4. Decoding time taken by Intel IPP for variable bit rates.

3.1.5 H.264 file size comparison

Figure 3.5 illustrates the H.264 file size obtained from JM 16.2 and Intel IPP.

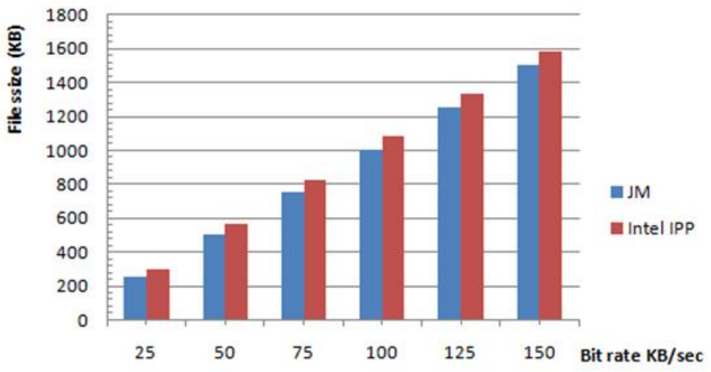


Figure 3.5. H.264 file size comparison of JM 16.2 and Intel IPP.

3.1.6 Compression ratios for JM and Intel IPP

Compression ratio has been calculated as the ratio between the original sequence file size to the H.264 file size. Figure 3.6 illustrates the compression ratios achieved by JM 16.2 and Intel IPP.

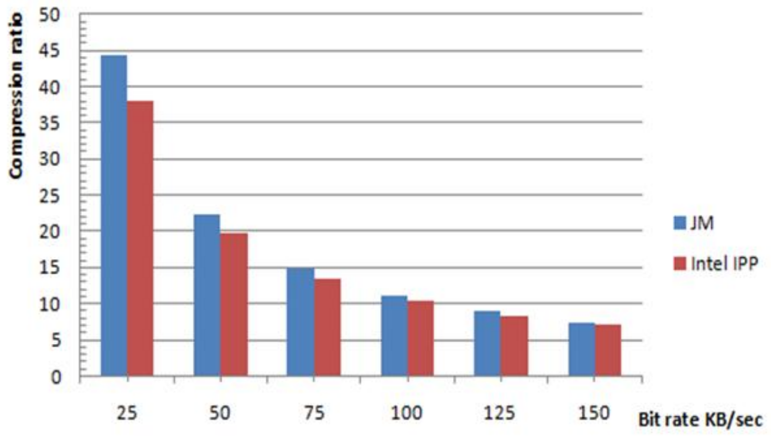


Figure 3.6. Compression ratio comparison for JM 16.2 and Intel IPP.

3.2 Quality Metrics

The 3 metrics used to compare the quality of the output are the structural similarity index [19] (SSIM), peak to peak signal to noise ratio (PSNR), and the Mean Square Error (MSE).

3.2.1 SSIM values obtained for JM 16.2 and Intel IPP

The values of SSIM obtained for JM 16.2 and Intel IPP H.264 at different bit rates are tabulated in Table 3.3.

Table 3.3. Values of SSIM obtained for JM 16.2 and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	0.9609	0.96297
50	0.98146	0.98186
75	0.98789	0.98876
100	0.99063	0.99221
125	0.99227	0.99419
150	0.9934	0.99546

The Figure 3.7 shows the plot of SSIM achieved by JM 16.2 and Intel IPP for Foreman sequence.

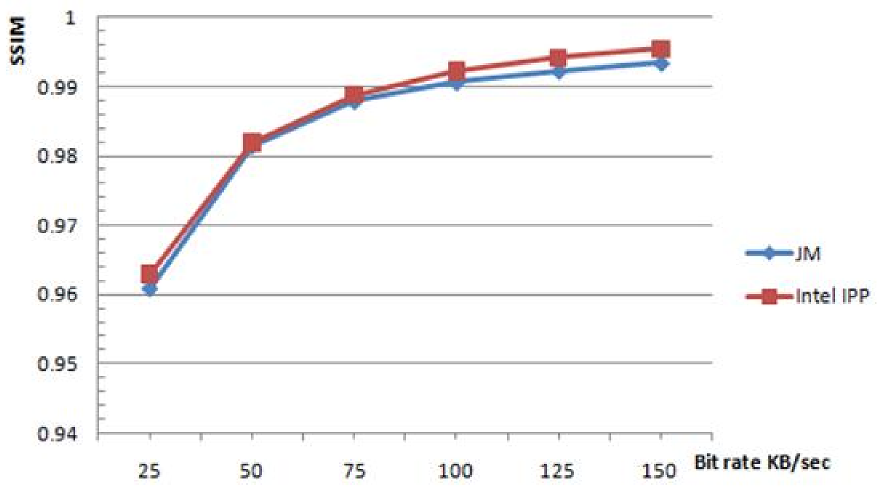


Figure 3.7. SSIM achieved by JM 16.2 and Intel IPP.

3.2.2 PSNR obtained for JM 16.2 and Intel IPP

The values of PSNR [20] in dB obtained for JM 16.2 and Intel IPP H.264 at different bit rates are tabulated in Table 3.4.

Table 3.4. Values of PSNR in dB obtained for JM 16.2 and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	37.18359	37.46751
50	41.13141	41.28917
75	43.17939	43.7612
100	44.37394	45.63609
125	45.26677	47.11219
150	45.95815	48.34548

The Figure 3.8 shows the plot of PSNR in dB achieved by JM and Intel IPP.

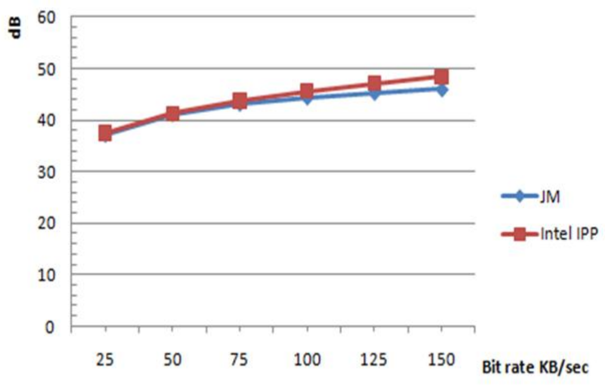


Figure 3.8. PSNR in dB achieved by JM 16.2 and Intel IPP.

3.2.3 MSE obtained for JM H.264 and Intel IPP

The values of MSE [21] obtained for JM 16.2 and Intel IPP H.264 at different bit rates are listed in Table 3.5.

Table 3.5. Values of MSE obtained for JM and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	12.43717	11.6501
50	5.01117	4.83241
75	3.1271	2.73502
100	2.37513	1.77612
125	1.93376	1.26433
150	1.64917	0.95177

The Figure 3.9 shows the plot of MSE achieved by JM 16.2 and Intel IPP obtained for Foreman sequence.

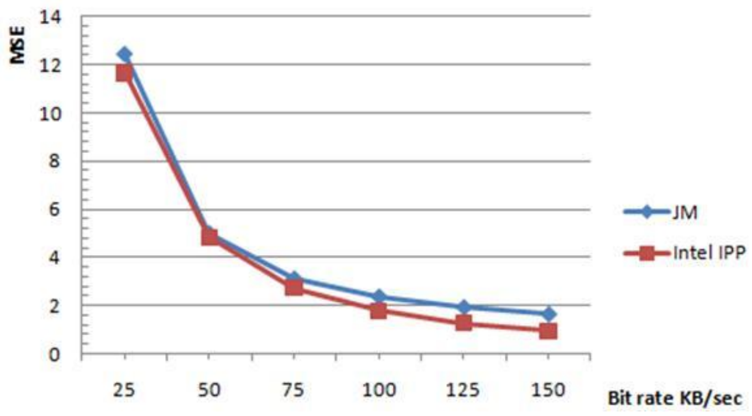


Figure 3.9. Comparison of MSE achieved by JM 16.2 and Intel IPP.

3.3 Results obtained for "Football.yuv" [2] sequence

The results obtained from JM 16.2 are listed in Table 3.6. The results

Table 3.6. Results obtained from JM 16.2 for Football CIF sequence

Bit rates used (KB/sec)	Motion estimation time (sec)	Encoding time (sec)	Decoding time (sec)	H.264 file size (KB)	Compression ratio
25	1867.852	1977.511	7.46	115	116.21
50	1895.832	2011.676	8.643	151	88.5
75	1870.535	1994.897	8.793	225	59.4
100	1787.468	1914.402	10.001	300	44.55
125	1762.494	1894.901	9.81	375	35.64
150	1752.587	1889.357	10.397	450	29.7

obtained from Intel IPP is tabulated in Table 3.7.

Table 3.7. Results obtained from JM 16.2 for Football CIF sequence

Bit rates used (KB/sec)	Motion estimation time (sec)	Encoding time (sec)	Decoding time (sec)	H.264 file size (KB)	Compression ratio
25	1.47	1.77	0.0483	88	151.87
50	1.96	2.27	0.0567	186	71.85
75	2.25	2.58	0.0642	280	47.73
100	2.38	2.71	0.0629	372	35.92
125	2.47	2.93	0.0628	465	28.74
150	2.53	2.86	0.0898	554	24.12

3.3.1 Encoding time and ME time taken by JM 16.2

Figure 3.10 shows the comparison of encoding time and ME time taken by JM 16.2 for Football sequence.

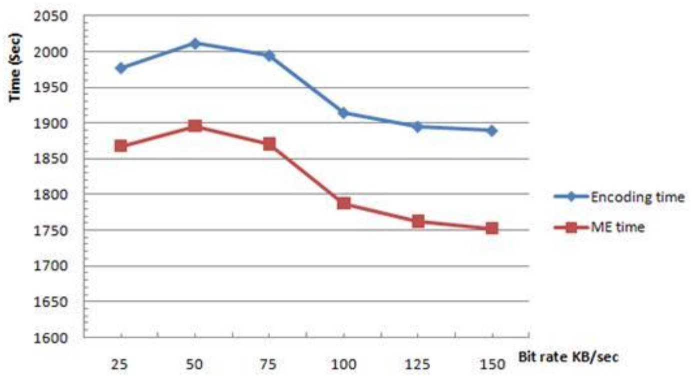


Figure 3.10. Encoding time and ME time taken by JM 16.2.

3.3.2 Encoding time and ME time taken by Intel IPP

Figure 3.11 shows the comparison of encoding time and ME time taken by Intel IPP for Football sequence.

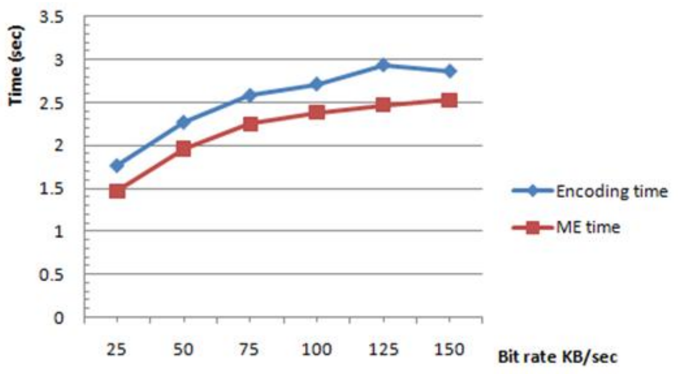


Figure 3.11. Encoding time and ME time taken by Intel IPP.

3.3.3 Decoding time taken by JM 16.2 for Football sequence

Figure 3.12 illustrates the decoding time taken by JM 16.2.

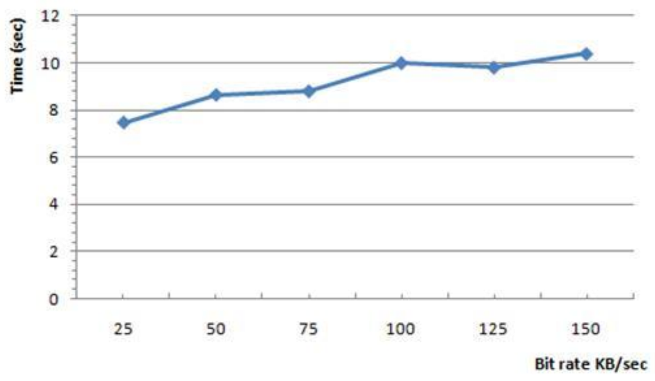


Figure 3.12. Decoding time taken by JM 16.2 for Football sequence.

3.3.4 Decoding time taken by Intel IPP

Figure 3.13 illustrates the decoding time taken by Intel IPP.

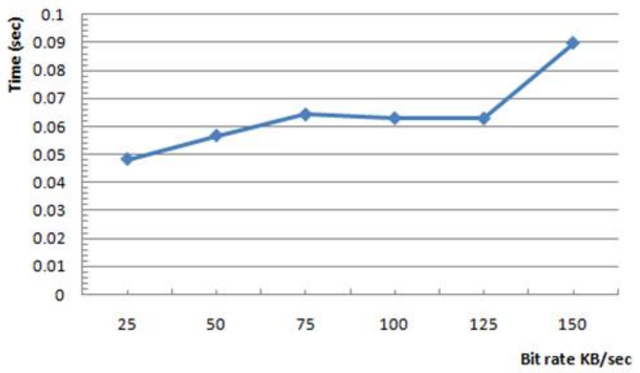


Figure 3.13. Decoding time taken by Intel IPP for Football sequence.

3.3.5 H.264 file size comparison

Figure 3.14 illustrates the H.264 file size obtained from JM 16.2 and Intel IPP

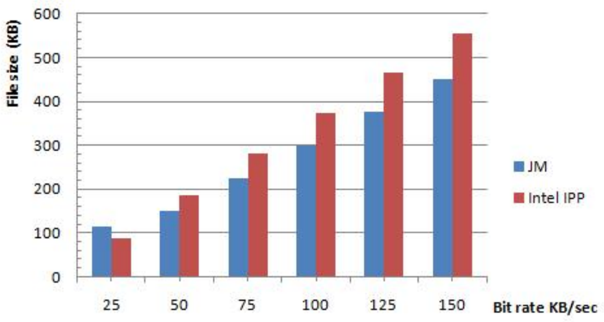


Figure 3.14. H.264 file size obtained from JM 16.2 and Intel IPP.

3.3.6 Compression ratios for JM 16.2 and Intel IPP

Compression ratio has been calculated as the ratio between the original sequence file size and the H.264 file size. Figure 3.15 illustrates the compression ratio achieved by JM 16.2 and Intel IPP.

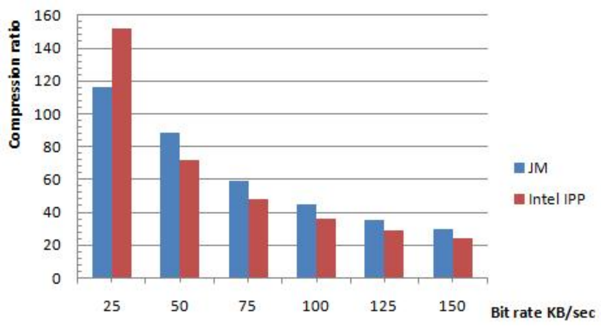


Figure 3.15. Compression ratio achieved by JM 16.2 and Intel IPP.

3.4 Quality Metrics

3.4.1 SSIM obtained for JM 16.2 and Intel IPP H.264

The values of SSIM obtained for JM 16.2 and Intel IPP H.264 at different bit rates are listed in Table 3.8.

Table 3.8. SSIM values obtained for JM 16.2 and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	0.78131	0.76785
50	0.82761	0.8471
75	0.8734	0.8876
100	0.90238	0.91482
125	0.92068	0.93117
150	0.93351	0.94339

The graph shows the plot of SSIM achieved by JM 16.2 and Intel IPP for different bit rates in Figure 3.16.

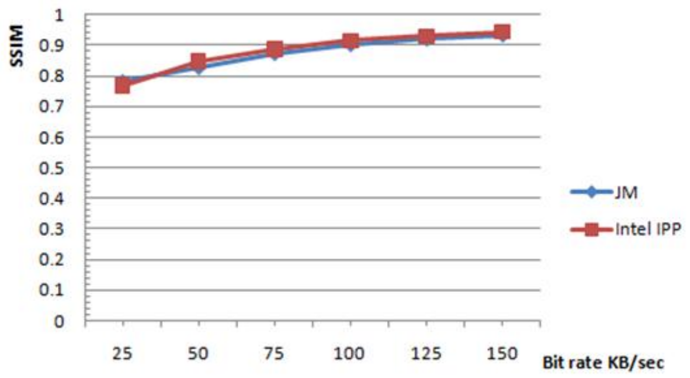


Figure 3.16. SSIM achieved by JM 16.2 and Intel IPP for Football sequence.

3.4.2 PSNR obtained for JM 16.2 and Intel IPP H.264

The values of PSNR in dB obtained for JM 16.2 and Intel IPP H.264 at different bit rates are listed in Table 3.9.

Table 3.9. PSNR values in dB obtained for JM 16.2 and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	29.33657	29.15126
50	30.98771	32.14291
75	33.01121	34.08814
100	34.73708	35.63852
125	36.00283	36.81034
150	37.02637	37.82698

The graph shows the plot of PSNR in dB achieved by JM and Intel IPP for different bit rates in Figure 3.17.

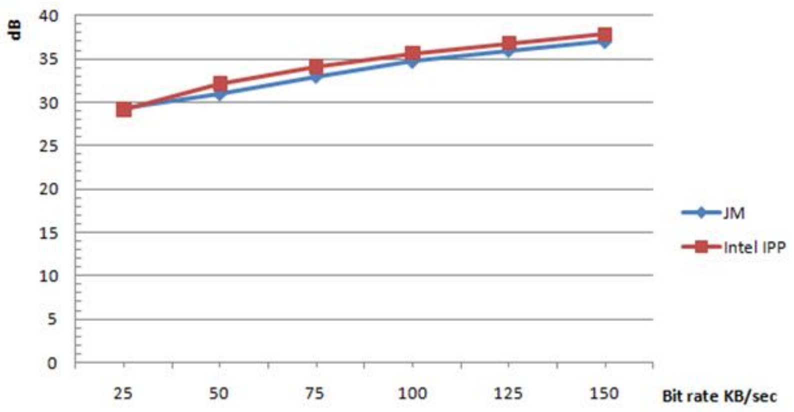


Figure 3.17. PSNR in dB achieved by JM and Intel IPP.

3.4.3 MSE obtained for JM H.264 and Intel IPP H.264

The values of MSE obtained for JM 16.2 and Intel IPP H.264 at different bit rates are listed in Table 3.10.

Table 3.10. MSE values obtained for JM 16.2 and Intel IPP

Bit rates used (KB/sec)	JM	Intel IPP
25	75.7571	79.05947
50	51.79758	39.69989
75	32.50572	25.36683
100	21.84603	17.75123
125	16.32292	13.55333
150	12.89565	10.72463

The graph shows the plot of MSE achieved by JM 16.2 and Intel IPP for different bit rates in Figure 3.18 .

3.5 Discussion

The Intel IPP H.264 is multi-threaded and uses CPU-optimized IPP routines. It uses Intel IPP libraries to exploit benefits from SIMD (Single Instruction Multiple

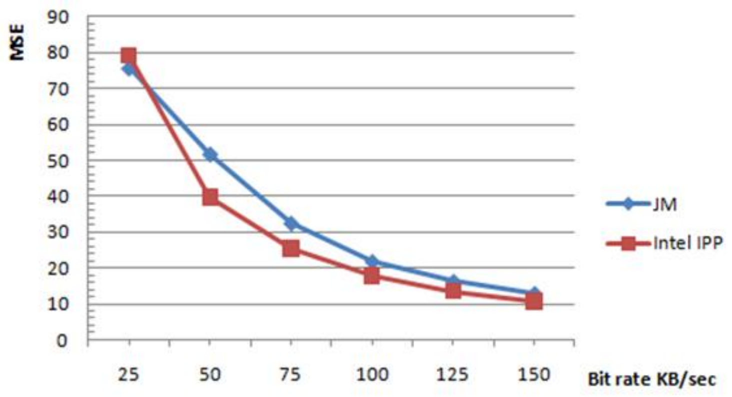


Figure 3.18. MSE achieved by JM 16.2 and Intel IPP.

Data) instructions available on modern processors.In addition to this, it is also able to utilize multiple cores doing work in parallel. Hence, it is much faster than JM. The quality of the video is also better than the JM output in terms of SSIM, PSNR and MSE. The only place where the JM 16.2 scores over Intel IPP H.264 is the compression size of H.264 file. The maximum size difference found was 104 KB (CIF sequence with bit rate of 150 KB/sec).

3.6 Summary

The Intel IPP compiler is highly optimized and it is more than 500 times faster than the JM software for encoding time and decoding time. The quality of the output video is tested in terms of MSE, PSNR, and SSIM for different video streams namely CIF and QCIF. Intel IPP H.264 clearly emerges as the winner against JM 16.2 in all aspects except the compression size of a H.264 file. The next chapter explains the applications of machine learning to the JM software to reduce the encoding time.

CHAPTER 4

MACHINE LEARNING

4.1 Machine Learning

Machine learning [22], is a branch of artificial intelligence that is concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data, such as from sensor data or databases. A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distributions. Data can be seen as examples that illustrate relations between observed variables. A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data. The difficulty is that the set of all possible behaviors given all possible inputs is too large to be covered by the set of observed examples (training data). Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases.

4.1.1 Decision Tree Learning

Decision tree learning [23], used in statistics, data mining and machine learning, uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. More descriptive names for such tree models are classification trees or regression trees. In these tree structures, leaves represent classifications and branches represent conjunctions of features that lead to those classifications.

In the decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining [23], a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making.

Advantages of a decision tree:

- Simple to understand and interpret.
- Able to handle both numerical and categorical data.Other techniques are usually specialized in analysing datasets that have only one type of variable. Eg. relation rules can be used only with nominal variables while neural networks can be used only with numerical variables.

  Limitations:
- Decision-tree learners can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning are necessary to avoid this problem. Pruning is a technique in machine learning that reduces the size of decision trees by removing sections of the tree that provide little power to classify instances. The dual goal of pruning is reduced complexity of the final classifier as well as better predictive accuracy by the reduction of overfitting and removal of sections of a classifier that may be based on noisy or erroneous data.

4.2 WEKA

Weka (Waikato Environment for Knowledge Analysis) [24] is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. The original version was primarily designed as a tool for analyzing data from agricultural domains but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research. Advantages of Weka include:

- Free availability under the General Public License
- Portability: because it is fully implemented in the Java programming language and thus runs on almost any modern computing platform
- A comprehensive collection of data preprocessing and modeling techniques
- Ease-of-use due to the graphical user interfaces it contains

The philosophy behind WEKA is to move away from supporting a computer science or machine learning researcher, and towards supporting the end user of machine learning. The end user is someone typically, an agricultural scientist [25] with an understanding of the data and sufficient knowledge of the capabilities of machine learning to select and investigate the application of different techniques. In order to maintain this philosophy, WEKA concentrates on ensuring that the implementation details of the machine learning algorithms and the input formats they require are hidden from the user. Hence, the WEKA software is easy to use.

4.3 C4.5 Algorithm

C4.5 is an algorithm [26] used to generate a decision tree. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. C4.5 uses the divide and conquer method [22] to construct a tree from a set, S, of training instances. If all instances in S belong to the same class, the decision tree is a leaf labeled with that class. Otherwise the algorithm uses a test to divide S into several non-trivial partitions. Each of the partitions becomes a child node of the current node and the tests separating S are assigned to the branches.

Conditional entropy is used to provide a measure of the correlation between features and class and between features. If H(X) is the entropy of a feature X and H(X/Y) the entropy of a feature X given the occurrence of feature Y the correlation between two features X and Y can then be calculated using the symmetrical uncertainty:

$$C(X/Y) = \frac{H(X) - H(X/Y)}{H(Y)} \tag{4.1}$$

The class of an instance is considered to be a feature. The goodness of a subset is then determined as:

$$G_{subset} = \frac{k\bar{r}_1}{\sqrt{k + (k-1)\bar{r}_2}} \tag{4.2}$$

where k is the number of features in a subset, $\bar{r}_1$ is the mean feature correlation with the class and $\bar{r}_2$ is the mean feature correlation. When $\bar{r}_1$ is larger and the $\bar{r}_2$ is smaller, the classification results of subset is better. It yields a better classification tree.

The C4.5 algorithm constructs the decision tree with a divide and conquer strategy. In C4.5, each node in a tree is associated with a set of cases. Also, cases are assigned weights to take into account unknown attribute values. At the beginning, only the root is present and associated with the whole training set ΓS and with all case weights equal to 1.0. At each node, the following divide and conquer algorithm is executed trying to exploit the locally best choice, with no backtracking allowed [5].

4.3.1 Pseudo code of the C4.5 Tree-Construction Algorithm

```
1. Compute ClassFrequency(T);
```

```
2. if ‘OneCase or ‘FewCases’
   Return a leaf;
   Create a decision node N;
3. For Each Attribute A,
   ComputeGain(A);
4. N.test=AttributeWithBestGain;
5. if N.test is continuous;
   Find threshold;
6. For Each T’ in the splitting of T
7. if T’ is Empty; Child of N is a leaf
   Else
8. Child of N =FormTree(T’);
9. ComputeErrors of N; Return N
```

Let T be the set of cases associated at the node. The weighted frequency freq (C_i,T) is computed (Step (1)) of cases in T whose class is C_i for i ϵ [1, N$class$].

In all cases, (Step (2)) in T belongs to the same class as C_j (or the number of cases in T is less than a certain value), then the node is a leaf, with associated class C_j. The classification error of the leaf is the weighted sum of cases in T whose class is not C_j. If T contains cases belonging to two or more classes (Step (3)), then the information gain of each attribute is calculated. For discrete attributes, the information gain is relative to the splitting of cases in T into sets with distinct attribute values. For continuous attributes, the information gain is relative to the splitting of T into two subsets, namely, cases with an attribute value not greater than a local threshold and cases with an attribute value greater than a certain local threshold, which is determined during information gain calculation. The attribute with the highest information gain (Step (4)) is selected for the test at the node. Moreover, in case a continuous attribute is selected, the threshold is computed (Step (5)) as the greatest value of the whole training set that is below the local threshold [5].

A decision node has s children if T_i (where i[1,s]) are the sets of the splitting produced by the test on the selected attribute (Step(6)). Obviously, s=2 when the selected attribute is continuous, and s=h for discrete attributes with h known values. For i=[1,s], if T_i is empty, (Step(7)) the child node is directly set to be a leaf, with the associated class. This is the most frequent class at the parent node and classification error 0.
If T_i is not empty, the divide and conquer approach consists of recursively applying the same operations (Step (8)) on the set consisting of T_i plus those cases in T with an unknown value of the selected attribute. Note that cases with an unknown value of the selected attribute are replicated in each child with their weights proportional to the proportion of cases in T_i over cases in T with a known value of the selected attribute. Finally, the classification error (Step (9)) of the node is calculated as the sum of the error of classifying all cases in T as belonging to the most frequent class in T, then the node is set to be a leaf and all sub trees are removed. The information gain of an attribute a for a set of cases T is calculated as follows: if a is discrete, and T_i (where i [1,s]) are the subsets for T consisting of cases with distinct known value for attribute a, then:

$$\mathrm{gain} = \mathrm{info(T)} - \sum_{\mathrm{i=1}}^{\mathrm{s}} \frac{|\mathrm{T_i}|}{|\mathrm{T}|} \times \mathrm{info(T_i)} \tag{4.3}$$

where

$$\mathrm{info(T)} = -\sum_{\mathrm{j=1}}^{\mathrm{N_{class}}} \frac{\mathrm{freq(C_j, T)}}{|\mathrm{T}|} \times \log_2\left(\frac{\mathrm{freq(C_j, T)}}{|\mathrm{T}|}\right) \tag{4.4}$$

is the entropy function. The function freq(C_j,T), is the function calculating the frequency of classifying case T as C_j. While having an option to select information gain, by default, however, C4.5 considers the information gain ratio of splitting T_i (where i ϵ [1,s]), which is the ratio of information gain to its split information:

$$\mathrm{Split(T)} = -\sum_{\mathrm{i=1}}^{\mathrm{s}} \frac{|\mathrm{T_i}|}{|\mathrm{T}|} \times \log_2\left(\mathrm{p}\frac{|\mathrm{T_i}|}{|\mathrm{T}|}\right) \tag{4.5}$$

where p represents the probability function.

4.4 Approach

The implementation of using decision trees as if-else statements is done on JM 16.2. Machine learning is used to exploit the spatial and temporal redundancies in video in order to make optimal mode decisions by replacing the sum of absolute differences (SAD) and other cost evaluations by if else statements in the motion estimation block. The flow chart shown in Figure 4.1 sums up the approach incorporated in this project.

The motion estimation process takes upto 70 percent of the encoding time in H.264 [27]. Hence, C4.5 algorithm is used to reduce the complexity of determining the mode decisions. The statistics for each 16x16 macroblock of the first four frames of the video sequence is calculated. The statistics being the mean, variance, variance of means for all the sub macroblock sizes in the macroblock, mean of the adjacent macroblocks, variance of the adjacent macroblocks and variance of means for all

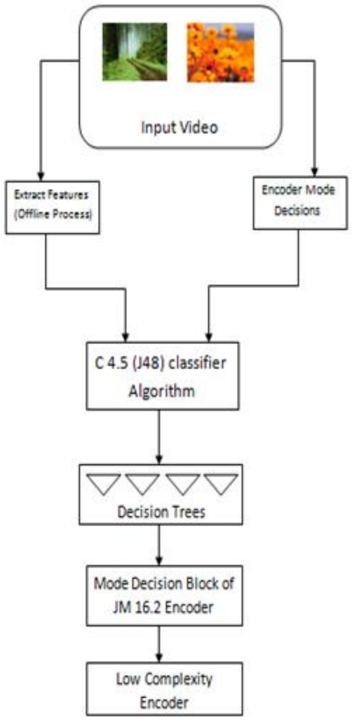

Input Video
Extract Features (Offline Process)
Encoder Mode Decisions
C 4.5 (J48) classifier Algorithm
Decision Trees
Mode Decision Block of JM 16.2 Encoder
Low Complexity Encoder

Figure 4.1. Flow chart used to achieve the low complexity encoder.

the submacroblock sizes in the adjacent blocks. The modes for the same first four frames from the video sequences are determined from the H.264 encoder in the JM 16.2 software. These modes and the determined statistics are collectively given as attributes for training in the WEKA tool. This is an offline process. The WEKA tool uses the C4.5 (J48) classifier algorithm to determine the mode decision tree. An attempt has been made to determine a universal tree that can give relatively accurate mode decisions to any video sequence. To demonstrate this,different combinations of video sequences for training the mode decision trees and later testing the mode decision trees. Table 5.1 summarizes the results. The attributes most commonly considered for mode decision in all the entries in the table are considered to determine the mode decision for the universal mode decision tree. This tree is implemented in the form of if else statements in the motion estimation block of JM16.2. Hence, the motion estimation process is reduced to if else statements.

Following are the parameters that are used during encoding:

- High profile - The primary profile for broadcast and disc storage applications, particularly for high-definition television applications.
- Bi directional Prediction - H.264/AVC supports multi-picture motion-compensated prediction. That is, more than one prior-coded picture can be used as a reference for motion-compensated prediction. B frames use both a past and future frame as a reference.
- Context adaptive binary arithmetic coding (CABAC) - is a form of entropy coding used in H.264/MPEG-4 AVC video encoding. It is a lossless compression technique. It is notable for providing much better compression than most other encoding algorithms.

This thesis considers only sub macroblock modes (8x8, 8x4, 4x8 and 4x4) for machine learning. The motion estimation code which contains this block is replaced by if else statements. This idea of using machine learning to reduce the motion estimation time has been taken from [5]. In this thesis, SD (Standard Definition) sequences are used for training purposes to build a stronger tree along with the results obtained from [5]. Parameters such as bi-directional prediction, high profile of H.264 and CABAC are used to improve the video quality of the sequence. The speedups of the encoding time and the ME time are smaller than that in [5] because of the use of these parameters.

4.5 Summary

This chapter has discussed decision tree learning and the Weka tool. The C4.5 algorithm and the approach incorporated in this thesis are covered. The next chapter discusses the experimental results obtained after the application of machine learning to the test sequences.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Machine Learning

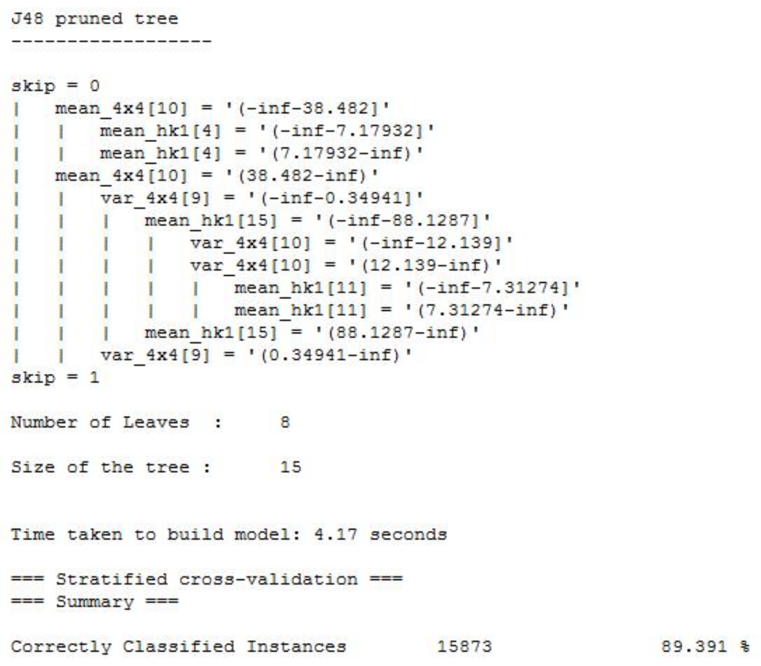
Table 5.1 shows the WEKA tool results for various combinations of videos sequences as training and test sequences. The if-else statements derived from the mode decision trees were used to replace the mode decision block in the JM encoder. The assessment metrics like PSNR, MSE, SSIM and file compression ratios for the H.264 video encoder as in JM 16.2 and the encoder based on machine learning are tabulated in Tables 5.2 through 5.5.

Table 5.1. WEKA results for various combinations of video sequences

Training Sequence 1	% Accuracy* for training Seq 1	Training Sequence 2	% Accuracy* for training Seq 2	Test Sequence	% Accuracy*
Container_cif	98.131	———	———	Waterfall_cif	95.0042
Stefan_cif	88.3838	Tempete_cif	85.0444	Container_cif	90.1812
Bus_cif	70.6861	Foreman_cif	80.7645	Mobile_cif	77.188
Waterfall_cif	90.5636	———	———	Bus_cif	83.0469
Susie_sd	89.391	———	———	Foreman_cif	86.2947
Rosebowl_sd	88.172	Susie_SD	89.391	Container_cif	82.610

% Accuracy * refers to the accuracy in determining the mode decision using machine learning in comparison to the mode decision in JM 16.2 encoder.

Classification tree for the Susie [2] sequence is shown in Figure 5.1.



```
J48 pruned tree
------------------

skip = 0
|   mean_4x4[10] = '(-inf-38.482]'
|   |   mean_hk1[4] = '(-inf-7.17932]'
|   |   mean_hk1[4] = '(7.17932-inf)'
|   mean_4x4[10] = '(38.482-inf)'
|   |   var_4x4[9] = '(-inf-0.34941]'
|   |   |   mean_hk1[15] = '(-inf-88.1287]'
|   |   |   |   var_4x4[10] = '(-inf-12.139]'
|   |   |   |   var_4x4[10] = '(12.139-inf)'
|   |   |   |   |   mean_hk1[11] = '(-inf-7.31274]'
|   |   |   |   |   mean_hk1[11] = '(7.31274-inf)'
|   |   |   mean_hk1[15] = '(88.1287-inf)'
|   |   var_4x4[9] = '(0.34941-inf)'
skip = 1

Number of Leaves  :     8

Size of the tree :      15


Time taken to build model: 4.17 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances       15873               89.391 %
```

Figure 5.1. Classification tree for Susie sequence from Weka tool.

The simulation results obtained using JM 16.2 and JM using machine learning (4 frames) is shown in Table 5.2.

Table 5.2. Encoding time and ME time comparison

Seq No	Sequence	Encoding time(sec) for JM16.2 without machine learning	Encoding time (sec) machine learning Sequence 2	ME time (sec) for JM 16.2 without machine learning	ME time (sec) using machine learning
1	Bus_cif	1924.628	1520.071	1374.73	914.470
2	Interview_cif	2107.122	1593.827	1249.716	861.554
3	Mother_Daughter _qcif	534.741	406.563	441.163	287.108
4	Foreman_qcif	419.325	330.050	305.298	204.238

The motion estimation time for the sequences is illustrated in the figure 5.2:

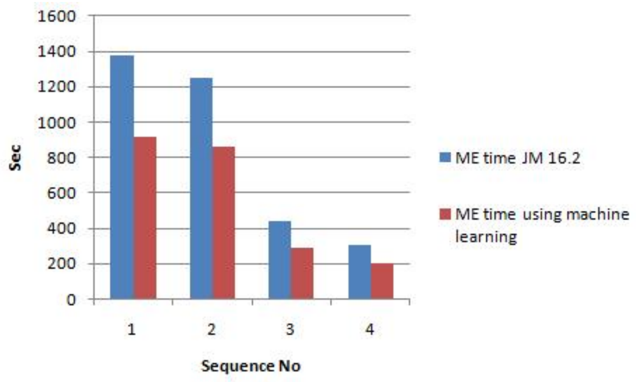


Figure 5.2. Motion Estimation time for sequences.

The speed up in the encoding time and the motion estimation time by using machine learning are tabulated in Table 5.3.

Table 5.3. Speedup in encoding time and ME time using machine learning

Seq No	Sequence	Speedup in Encoding time (%)	Speedup in ME time (%)
1	Bus_cif	21.02	33.48
2	Interview_cif	24.36	31.06
3	Mother_Daughter _qcif	23.97	34.92
4	Foreman_qcif	21.29	33.102

5.2 Quality Metrics

The three metrics used to compare the quality of the output are the structural similarity index (SSIM), the peak to peak signal to noise ratio (PSNR), and the Mean Square Error (MSE).

5.2.1 PSNR and MSE results obtained

The simulation results obtained with respect to PSNR in dB and MSE are tabulated in Table 5.4.

Table 5.4. Comparison of PSNR and MSE

Sequence Number	Sequence	PSNR(dB) using JM 16.2 encoder	PSNR (dB) using JM 16.2 encoder	MSE using JM 16.2 encoder	MSE using machine learning
1	Bus_cif	39.124	39.109	12.278	12.411
2	Interview_cif	37.912	37.898	13.182	13.345
3	Mother_Daughter _qcif	35.763	35.759	16.854	16.861
4	Foreman_qcif	33.257	33.210	17.002	17.993

The PSNR in dB obtained for JM 16.2 and JM using machine learning is shown in figure 5.3.

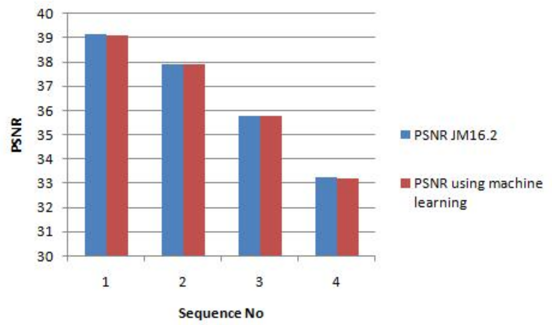


Figure 5.3. PSNR obtained from JM 16.2 and JM using machine learning.

The MSE obtained for JM 16.2 and JM using machine learning is plotted in figure 5.4.

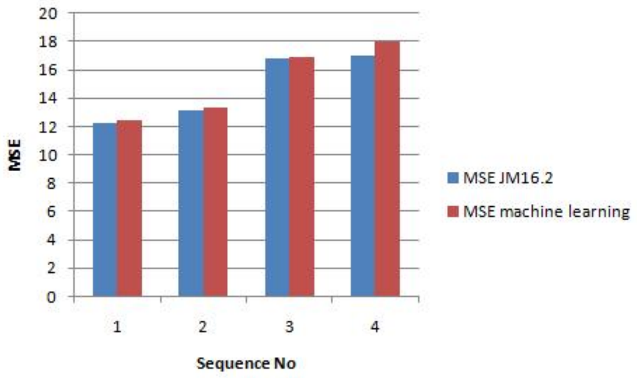


Figure 5.4. Plot of MSE from JM 16.2 and JM using machine learning.

5.2.2 SSIM results

The simulation results obtained with respect to SSIM are tabulated in Table 5.5.

Table 5.5. SSIM comparison of JM 16.2 and JM using machine learning

Sequence Number	Sequence	SSIM using JM 16.2 encoder	SSIM using machine learning	% decrease
1	Bus_cif	0.9568	0.9565	0.0003
2	Interview_cif	0.9607	0.9601	0.0006
3	Mother_Daughter _qcif	0.9412	0.9397	0.00158
4	Foreman_qcif	0.9383	0.936	0.0024

% decrease** refers to the percentage decrease in SSIM using machine learning in comparison to the SSIM as obtained in JM 16.2 encoder.

SSIM in JM 16.2 and JM using machine learning for sequences is shown in figure 5.5.

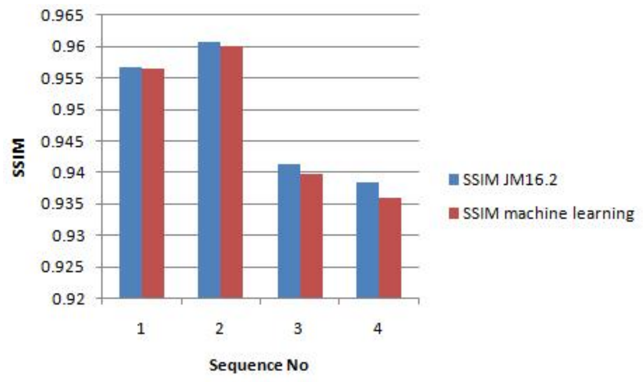


Figure 5.5. Plot of SSIM in JM 16.2 and JM using machine learning.

100 frames of the video Interview.yuv was encoded using JM 16.2 and JM using machine learning. The following values were captured:

- Speedup in encoding time: 13.8 %
- Speedup in ME time: 24.17 %
- % decrease in SSIM: 0.0006

5.3 Discussion

Tables 5.2 through 5.5 tabulate the results of encoding 4 frames of video sequences. The average speedup in the encoding time for 4 frames is 22.66 %. The average speedup in the ME time for 4 frames is 34.027 %. It is observed that whenever the number of frames to encode is increased, the average speedups in encoding time and ME time come down. For example, the speedup in the encoding time for 100 frames of Interview sequence is 13.8 %. The speedup in ME time for the same sequence is 24.17%. This variation is because training has been done only for four frames. Training using 100 frames was not done since overfitting problems were encountered. Overfitting does not help to build a general classification tree.

5.4 Summary

In this chapter, the results obtained using machine learning are discussed. The video quality of the sequences is also tabulated and illustrated using SSIM, PSNR and MSE as the metrics. The next chapter deals with streaming of media over the internet.

CHAPTER 6

SMOOTH STREAMING

6.1 Smooth Streaming

Smooth streaming [28] is the Microsoft implementation of adaptive streaming technology, which is a form of Web-based media content delivery that uses standard HTTP (Hypertext Transfer Protocol). Instead of delivering media as full-file downloads, or as persistent (and thus stateful) streams, the content is delivered to clients as a series of MPEG-4 (MP4) fragments that can be cached at edge servers. Smooth streaming-compatible clients use special heuristics to dynamically monitor current network and local PC conditions and seamlessly switch the video quality of the smooth streaming presentation that they receive. As clients play the fragments, network conditions may change (for example, bandwidth may decrease) or video processing may be impacted by other applications that are running. Clients can immediately request that the next fragment come from a stream that is encoded at a different bit rate to accommodate the changing conditions. This enables clients to play the media without stuttering, buffering, or freezing. As a result, users experience the highest-quality playback available, with no interruptions in the stream.

Smooth streaming, an IIS (Internet Information Services) Media Services extension, enables adaptive streaming of media to Silverlight and other clients over HTTP. Smooth streaming provides a high-quality viewing experience that scales massively on content distribution networks. It offers code-free deployment and simplified content management for content creators and content delivery networks. For end users, the improved video viewing experience will bring the reliability and quality to their favorite video Web sites [28].

6.2 MP4

MPEG-4 Part 14 or MP4 file format [29], is a multimedia container format standard specified as a part of MPEG-4. It is most commonly used to store digital video and digital audio streams, especially those defined by MPEG, but can also be used to store other data such as subtitles and still images. MPEG-4 Part 14 allows streaming over the Internet. The widely supported data streams are

- Video: MPEG-4 Part 10 (h.264), MPEG-4 Part 2
- Audio: Advanced Audio Coding (AAC - MPEG-4 Part 3 Subpart 4)

There are several reasons to use MP4 format [28]:

- MP4 is a lightweight container format with less overhead.
- It is based on a widely used standard, making 3rd party adoption and support more straightforward.
- It is designed with H.264 video codec support in mind. H.264 is an industry leading video compression standard that has been adopted across a broad range of Microsoft products [28],including Silverlight 3, Windows 7, Xbox 360, Zune and MediaRoom.
- MP4 is designed to natively support payload fragmentation within the file.

The .264 file is multiplexed with its corresponding .aac file using an open source software called MP4Box [30] to get a file in MP4 format. MP4 media file can be transcoded into four files namely

- *.ismv: It contains video and audio, or only video. There is usually one ISMV file per encoded video bit rate.
- *.isma: It contains only audio.
- *.ism : It describes the relationships between the media tracks, bit rates and files on disk . It is only used by the IIS Smooth Streaming server and not by clients.
- *.ismc : It describes the available streams to the client: the codecs used, bit rates encoded, video resolutions, markers, captions, etc. It is the first file delivered to the client.

  These four files are also called smooth streaming manifest files. Because they are based on XML, they are highly extensible. Among the features already included in the current smooth streaming format specification is the support for the following [28]:

- VC-1, WMA, H.264 and AAC codecs
- Multi-language audio tracks
- Alternate video and audio tracks (for example, multiple camera angles, director's commentary, etc.)
- Multiple hardware profiles (for example, a bit rate targeted at different playback devices)
- Live encoding and streaming

Smooth Streaming Playback Microsoft's adaptive streaming prototype (used for NBC Summer Olympics 2008 in Beijing) relied on physically chopping up long video files into small file chunks. To retrieve the chunks for the Web server, the player client simply needed to download the files in a logical sequence: 00001.vid, 00002.vid, 00003.vid, and so on. Smooth Streaming uses a more sophisticated file format and server design. The videos are no longer split up into thousands of file chunks, but are instead "virtually" split into fragments (typically one fragment per video GOP (Group of pictures)) and stored within a single contiguous MP4 file.

The first thing a player client requests from the Smooth Streaming server is the *.ismc client manifest. The manifest tells it which codecs were used to compress the content (so that the client runtime can initialize the correct decoder and build the playback pipeline), what bit rates and resolutions are available, and a list of the available chunks (with either their start times or durations). The fragment start offset is an agreed-upon time unit (usually 100 nanoseconds (ns)). This value is known from the client manifest. After receiving the client request, IIS Smooth Streaming looks up the quality level (bit rate) in the corresponding *.ism server manifest and maps it to a physical *.ismv or *.isma file on the hard disk drive. It then loads the MP4 fragment corresponding to the requested start time offset. This is how the chunks of video/audio are requested from the server [28].

The bit rate switching is what makes adaptive streaming more effective. The server plays no part in the bit-rate switching process. The client-side code looks at chunk download times (video is downloaded in chunks or fragments), buffer fullness, rendered frame rates, and other factors, and decides when to request higher or lower bit rates from the server.The IIS was set up and smooth streaming of media over the Internet was successfully done. But creating the transcoded files (ism files) using the MP4 file could not be achieved since there is no open source software or exe file available to execute it [28].

6.3 Summary

The .264 file is multiplexed with its corresponding .aac video to form an MP4 video. Smooth streaming using IIS is setup and the consistent playback without stutter, buffering or “last mile” congestion is observed in the video that comes with the Smooth Streaming setup.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The compression ratio of H.264 file is found to be better in JM software at various bit rates than the Intel IPP H.264. Hence, it is chosen ahead of the Intel IPP for reduction in the motion estimation time using machine learning. The 'Submacroblock modes' in JM 16.2 software are replaced by the if-else statements (which were obtained from WEKA tool). It is observed that using bi-directional prediction, CABAC and encoding using the high profile of H.264 decreases the motion estimation time by a smaller factor compared to not using these parameters. From table 5.3, the average speedup in the encoding time for 4 frames is 22.66 %. The average speedup in the ME time for 4 frames is 34.027 %.

It is observed that whenever the number of frames to encode is increased, the average speedups in encoding time and ME time come down. This is verified by encoding 100 frames of the Interview sequence. The average decrease in SSIM is 0.00122. The .264 video and .aac audio formats are placed on a MP4 container format. The MP4 file can be streamed (after converting them to manifest files) using IIS (Internet Information Services) to achieve smooth low complexity streaming of media over the Internet.

7.2 Future Work

Only sub macroblock modes for ME (Fig 2.5) have been used for classification. A classification which involves all the modes can be devoloped to reduce the encoding time and with little compromise on the video quality. An attempt to use decision tree softwares like Orange [31] and Sipina [32] can also be made, rather than the Weka tool used in this thesis.

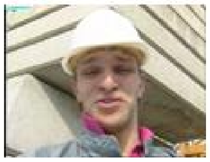
APPENDIX A

VIDEO SEQUENCES USED IN THIS THESIS

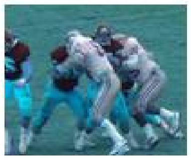
The video sequences considered in this thesis are as follows.

Bus_cif

Container_cif



Foreman_qcif



Football_cif

Mother-Daughter_qcif

Susie_SD

Waterfall_cif

Interview_cif

Rosebowl_sd

Stefan_cif

REFERENCES

[1] S. Kwon, A. Tamhankar, and K. Rao, “Overview of H. 264/MPEG-4 part 10,” *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 186–216, April 2006.

[2] “http://trace.eas.asu.edu/yuv/index.html,” video test sequences (YUV 4:2:0).

[3] “http://www.vcodex.com/files/,” working of H.264 codec.

[4] “http://en.wikipedia.org/wiki/bi directional prediction,” bi directional prediction.

[5] T. Purushotham, “Low complexity H.264 encoder using machine learning,” *M.S. Thesis, E.E Dept, UTA 2010.*

[6] “http://www.apple.com/quicktime/technologies/h264/,” for H.264 codec reference.

[7] “http://encyclopedia.jrank.org/articles/pages/6922/video-coding-techniques-and-standards.html,” about Video compression.

[8] M. Wien, H. Schwarz, and T. Oelbaum, “Performance analysis of SVC,” *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 17, no. 9, pp. 1194–1203, 2007.

[9] A. Ravi, “Performance analysis and comparison of the Dirac video codec with H. 264/MPEG-4 Part 10 AVC,” *M.S. Thesis, E.E Dept, UTA*, 2009.

[10] D. Marpe, T. Wiegand, and G. Sullivan, “The H. 264/MPEG-4 AVC standard and its applications,” *IEEE communications magazine*, vol. 44, pp. 134–143, 2006.

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

[12] I. Richardson, *H. 264 and MPEG-4 video compression.* Wiley Online Library, 2003.

[13] A. Puri, X. Chen, and A. Luthra, “Video coding using the H. 264/MPEG-4 AVC compression standard,” *Signal Processing-Image Communication*, vol. 19, no. 9, pp. 793–850, October 2004.

[14] T. Schierl, C. Hellge, S. Mirta, K. Gruneberg, and T. Wiegand, “Using H. 264/AVC-based scalable video coding (SVC) for real time streaming in wireless IP networks,” in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 3455–3458.

[15] P. Merkle, A. Smolic, K. Muller, and T. Wiegand, “Efficient prediction structures for multiview video coding,” *IEEE Transactions on Circuits and Systems for Video Technology,*, vol. 17, no. 11, pp. 1461–1473, November 2007.

[16] P. Carrillo, H. Kalva, and T. Pin, “Low complexity H. 264 video encoding, Applications of Digital Image Processing,” in *Proc. of SPIE*, vol. 7443.

[17] “http://iphome.hhi.de/suehring/tml/jmreferencedocumentation manual.

[18] “http://software.intel.com/en-us/,” for Intel IPP software.

[19] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing,*, vol. 13, no. 4, pp. 600–612, April 2004.

[20] Z. Wang, L. Lu, and A. Bovik, “Video quality assessment based on structural distortion measurement,” *Signal processing: Image communication*, vol. 19, no. 2, pp. 121–132, February 2004.

[21] Z. Wang, H. Sheikh, and A. Bovik, "Objective video quality assessment," *The Handbook of Video Databases: Design and Applications*, pp. 1041–1078, 2003.

[22] J. Quinlan, *C4. 5: programs for machine learning*, 1993.

[23] "http://en.wikipedia.org/wiki/machine learning," for Machine learning.

[24] "http://www.cs.waikato.ac.nz/ml/weka/," for WEKA datamining tool.

[25] G. Holmes, A. Donkin, and I. Witten, "Weka: A machine learning workbench," in *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, December 1994, pp. 357–361.

[26] Y. Ma, Z. Qian, G. Shou, and Y. Hu, "Study of information network traffic identification based on C4. 5 algorithm," in *4th International Conference on Wireless Communications, Networking and Mobile Computing.* IEEE, October 2008, pp. 1–5.

[27] T. Purushotham and K. Rao, "Low complexity H.264 using machine learning," *SPA Poland*, September 2010.

[28] "http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/," for Smooth Streaming.

[29] "http://www.iso.org/iso/," for MP4 format.

[30] "http://www.videohelp.com/tools/mp4box," for MP4Box software.

[31] "http://orange.biolab.si/doc/modules/orngtree.htm," for Orange machine learning tool.

[32] "http://eric.univ-lyon2.fr/r̃icco/sipina.html," for Sepina decision tree software.

BIOGRAPHICAL STATEMENT

Suchethan Swaroop K.V was born in Bangalore, India in 1986. He received his B.S. degree from Visvesvaraya Technological University, India in Electronics and Communication Engineering in 2007. He worked in Infosys technologies Limited as a Software Engineer for two years (July 07 - July 09). He worked as a Graduate Research Assistant under Dr.K.R.Rao in Multimedia Processing Lab at UT Arlington from Fall 09 to Spring 10. He is now pursuing his internship at Ericsson Inc since Summer 2010. His interests are in the field of multimedia applications and data communication.