CNN based Bi-Directional Prediction for Complexity Reduction of High Efficiency Video Coding

Tharuki Rangana De Silva

A Thesis submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg

Abstract

Real-time video streaming has become the largest portion of internet traffic in recent years. Therefore, improving the efficiency of video coding remains an important research issue. Beyond the level of compression, there are two other factors that must be considered to determine the efficiency of a real-time video codec: decoded video quality and the computational complexity of the encoding and decoding processes.

Modern video codecs rely on inter-frame prediction for efficient coding. However, inter-frame prediction used in modern codecs is one of the most computationally expensive and time consuming operations. Convolutional neural networks (CNN) have been used in recent research for inter-frame prediction tasks. The CNN architectures in previous work have been used without regard to the model complexity and computational efficiency. The objective of this thesis is to develop a CNN based low complexity bi-prediction algorithm for video coding.

The contribution of this thesis consists of three parts. In the first part, a simple floating point CNN architecture has been developed to perform bi-prediction operation in video coding with an accuracy comparable to that produced by motion estimation and compensation used in modern video encoders. This architecture is then quantized to derive an integer arithmetic only CNN to further reduce the computational complexity. It is shown that the encoding time for integer CNN is considerably lower compared to the floating point CNN. The experimental results have shown that this conversion only causes a minor loss of prediction accuracy. In the final part, it is experimentally shown that the proposed integer arithmetic CNN bi-prediction algorithm has a lower computational cost and better video quality compared to the conventional motion estimation based bi-prediction. Further, it is shown that CNN based bi-prediction can contribute to a rate-distortion performance improvement in video coding.

Acknowledgements

First and foremost, I wholeheartedly thank my adviser Prof. Pradeepa Yahampath for allowing me to work with him, and for the continuous encouragement and support throughout my M. Sc program. A special thanks must be conveyed to the examining committee for accepting this thesis for review.

I would also like to acknowledge the support and inspiration given by the academic and non-academic staff of the Department of Electrical and Computer Engineering, Faculty of Graduate Studies and the University of Manitoba.

Last but not least, I would like to pay gratitude to the greatest inspirations of my life, my father Upali De Silva and my mother Chandrika Kudagamage. Without my parents support and encouragement, I would never be the person I am today. I would also like to thank my sister Rydma De Silva and my partner Thulana Kannangara for never giving up on me amidst all hardships. Many thanks to my family and friends, who have been immensely supportive throughout this journey.

Contents

List of Tables

List of Figures

List of Acronyms

CNN	**C**onvolutional **N**eural **N**etwork
HEVC	**H**igh **E**fficiency **V**ideo **C**oding
MCP	**M**otion **C**ompensated **P**rediction
HD	**H**igh **D**efinition
MV	**M**otion **V**ector
PU	**P**rediction **U**nit
CU	**C**oding **U**nit
RDO	**R**ate **D**istortion **O**ptimization
IoT	**I**nternet **o**f **T**hings
MCVC	**M**otion **C**ompensated **V**ideo **C**oding
VVC	**V**ersatile **V**ideo **C**oding
SVM	**S**upport **V**ector **M**achine
CTU	**C**oding **T**ree **U**nit
PSNR	**P**eak **S**ignal to **N**oise **R**atio
SSIM	**S**tructural **S**imilarity **I**ndex **M**easure
MSE	**M**ean **S**quare **E**rror
AMVP	**A**dvanced **M**otion **V**ector **P**rediction
MVD	**M**otion **V**ector **D**ifferences
MVP	**M**otion **V**ector **P**rediction
TZ	**T**est **Z**one
SAE	**S**um of **A**verage **E**rror

MER	**M**erge **E**stimation **R**egion
GOP	**G**roup **O**f **P**ictures
ReLU	**R**ectified **L**inear **U**nit

To my family and friends.

Chapter 1

Introduction

1.1 Video Compression

It is estimated that by 2022, video streaming will grow up to 88% of the total internet traffic [1]. Higher consumption of storage capacity and transmission bandwidth from video streaming has become a challenge due to the rapid growth of video applications and increasing demand for superior quality video. As a result, video traffic has become the biggest load on communication networks and data storage world-wide. This has made video compression a critical stage to guarantee the quality and latency of video streaming. Video compression or video encoding is the process of reducing the amount of data required to represent a video signal, prior to transmission or storage. The complementary operation, decompression or decoding, recovers a video signal from a compressed representation, prior to display.

A video signal is a sequence of images or frames, which are correlated with each other. Due to this high correlation in the video frames, a single frame within a

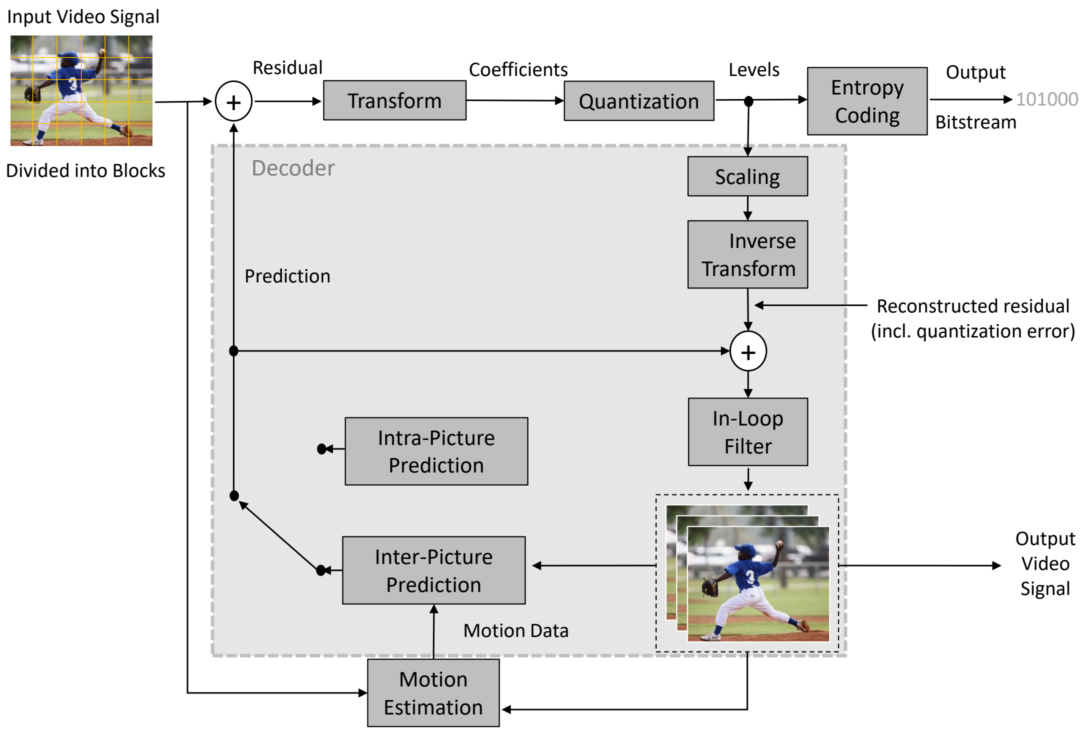


FIG. 1.1: Block diagram of a hybrid video encoder, including the modeling of the decoder within the encoder.

video sequence contains a notable number of similar neighboring pixels (spatial redundancy) and a high similarity is present between consecutive frames in a video sequence (temporal redundancy). Video compression methods are designed to eliminate the redundancy in each video frame as well as in the video sequence. All video compression methods since 1988 [2] have been based on the hybrid video coding principle, which is illustrated in Fig. 1.1. The term hybrid refers to the combination of two means used to reduce redundancy in the video signal, that is, prediction and transform coding with quantization of the prediction residual. Although prediction and transforms reduce the redundancy in the video signal by decorrelation, quantization decreases the amount of data required for transform coefficient representation by reducing their precision, ideally by removing only imperceptible details; in such case, it serves to reduce irrelevance in the data [2].

The hybrid video coder given in Fig. 1.1 can be characterized by the following building blocks.

Block partitioning is performed to divide the video frame into smaller blocks for the prediction and transform processes. Over the years, block partitioning has evolved to become more flexible by using different block sizes and shapes to enable adaptation to local neighborhood statistics. In the prediction stage, this allows an encoder to trade off the prediction accuracy (using small blocks) for the amount of data required to signal the prediction information to the decoder (using large blocks). In coding the residual differences, small blocks enable the coding of fine details and large ones can smoothen regions very efficiently by avoiding blocking artifacts.

Motion compensation or inter-picture prediction takes advantage of the redundancy that exists between the frames of the coded video sequence. In block based motion compensation illustrated in Fig. 1.2, for each block of a video frame, a corresponding (best matched) area from a previously decoded picture, which is the reference frame, is employed to predict the current block [3]. This process of finding the best matching block is known as motion estimation. Assuming that the content of a block moves between frames with translational motion, the displacement between the current block and the corresponding area in the reference frame is commonly referred to by a 2-D translational motion vector (MV) [3]. The encoder then signals the estimated MV data to the decoder.

Intra-picture or intra-frame prediction exploits the spatial redundancy that exists within a video frame by deriving the prediction for a block from already coded/decoded, spatially neighboring reference blocks. After deriving the intra-frame prediction mode, the encoder signals the estimated prediction information to the decoder.

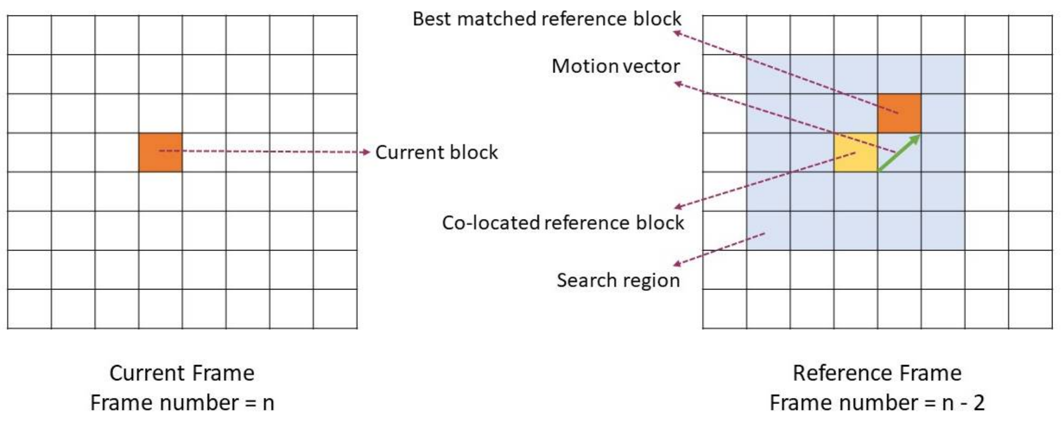


FIG. 1.2: Basic concept of block based motion compensation.

Transformation decorrelates a signal by transforming it from the spatial domain to a transformed domain (typically a frequency domain), using a suitable transform basis. Hybrid video coding standards apply a transform to the prediction residual, that is, the difference between the prediction and the original input video signal, as shown in Fig. 1.1. In the transform domain, the essential information typically concentrates into a small number of coefficients. At the decoder, the inverse transform needs to be applied to reconstruct the residual samples.

Quantization in hybrid video coding is typically applied to individual transformed residual samples, that is to transform coefficients, resulting in integer coefficient levels. As illustrated in Fig. 1.1, this process is applied at the encoder. At the decoder, the corresponding process is known as inverse quantization or simply as scaling, which restores the original value range without regaining the precision.

In-loop filtering is a filtering process that is applied to the reconstructed frame, as illustrated in Fig. 1.1, where the reconstructed frame is the combination of the reconstructed residual signal (with quantization error) and the prediction. This reconstructed frame after in-loop filtering can be stored to be used as a reference frame for inter-picture prediction of subsequent video frames. The main purpose of the filtering is to reduce the visual artifacts and decrease reconstruction errors of the reconstructed frame.

The above described video coding approach has been the basis for all video coding standards developed during the last 3 decades [4]. The state-of-the-art standards include H.265, also referred to as high efficiency video coding (HEVC) and H.266, referred to as versatile video coding (VVC). According to [5], the inter-frame prediction process is shown to consume 84% of encoding time of the motion compensated video coding (MCVC) process. This is due to the high complexity of the motion estimation and compensation techniques used in the inter-picture (or inter-frame) prediction process. Motion estimation is considered the most computationally expensive and resource hungry operation in the inter-frame prediction [6].

Although motion estimation plays an important role in MCVC, the high computational complexity makes it quite difficult to implement in real-time applications. Therefore, it is necessary to reduce the computational complexity and speed up the motion estimation operation. However, apart from regular block-wise shift motion, there usually exists inconsistent pixel-wise motion such as rotation and deformation between blocks, which will largely degrade the prediction performance in using motion estimation. In this thesis we focus on developing an alternative approach to prediction by using deep learning. This approach does not require motion estimation and therefore has the potential to be computationally more efficient.

Learning based approaches for video coding have been investigated due to the superior capability of convolutional neural networks (CNN) to solve computer vision and image processing problems compared to conventional model based approaches

[7]. In recent research, CNN based architectures have shown great potential in the estimation and representation of complex motion. While motion estimation needs precise per-pixel localization, it also requires finding correspondences between two input images. This involves not only learning image feature representations, but also learning to match them at different locations in the two images. According to [8], CNNs are capable of solving motion estimation problem as a supervised learning task. Different CNN architectures have been proposed to perform inter-frame prediction in prediction unit (PU) and coding unit (CU) levels in recent years. It has been observed that these CNN architectures improve the accuracy of inter-frame prediction in video coding standards.

Even though CNNs can be used for improving motion estimation in video encoders, they can also be used for direct non-linear prediction of video frames. A CNN can be pre-trained for universal prediction (using a large amount of training data). The same CNN can then be used in the encoder and the decoder. This approach can be more efficient for predictive video coding than motion estimation and compensation, as the latter approach requires a part of the coded bit stream to be used to signal motion information to the decoder.

1.2 Inter-frame prediction using CNNs

Inter-frame prediction in video coding can be performed by two methods; uni-prediction and bi-prediction. In uni-prediction a given frame is predicted from one of the temporally adjacent frames in a video sequence. In bi-prediction, a pair of temporally adjacent frames are used as reference frames for predicting a given frame. This thesis is concerned with using a CNN for bi-prediction. Therefore,

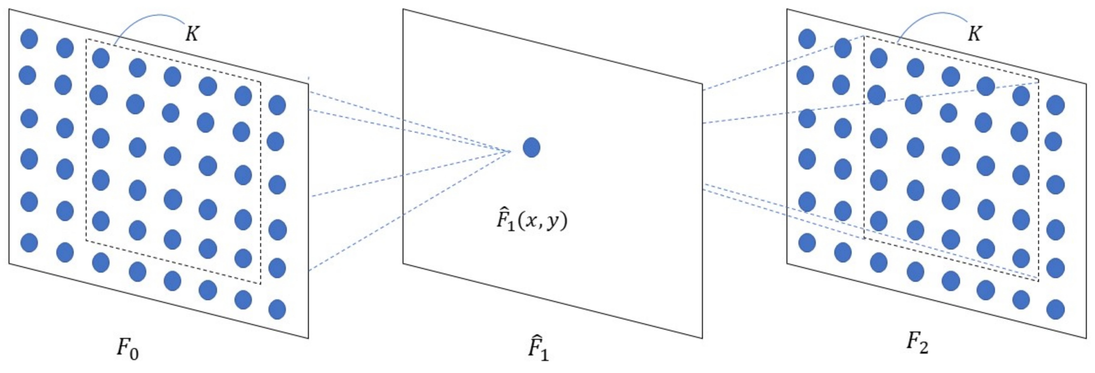


FIG. 1.3: Prediction of pixel value by convolution.

the process of using a CNN to perform this bi-prediction operation is explained in this section.

Given two video frames F_0 and F_2 in the video sequence, the intermediate frame $\hat{F}_1$ can be interpolated using the convolution operation. In this case, pixel interpolation is formulated as local convolution over the input frames F_0 and F_2 as shown in Fig. 1.3. The color of pixel (x, y) in the target frame to be interpolated can be obtained by convolving a filter kernal K over the input frames F_0 and F_2 centered at (x, y) position. This formulation of pixel interpolation as convolution has several advantages. The pixels of the target frame are derived by local convolution over the two input frames and therefore this method combines motion estimation and pixel synthesis into a single step. Second, the convolution filters provide flexibility to account for complex motions such as deformation, occlusion [9].

Estimating proper weights for the convolution filter K is essential to generate an accurate prediction of the intermediate frame $\hat{F}_1$. A CNN can be trained to estimate a proper convolution filter to synthesize each output pixel in the predicted frame. The output of this CNN is the prediction of the intermediate frame $\hat{F}_1$ and the inputs are the adjacent frames F_0 and F_2. The original intermediate frame

F_1 serves as the ground truth during the training stage. The prediction loss is then, back propagated to update the weights and biases of the network at each step. At the end of the training process, the estimated weights and biases of the convolution filters have the ability to predict the intermediate frame using the two adjacent frames. Further, this process can be extended to predict a set of frames using only two reference frames. This architecture has been used in this thesis to perform CNN based inter-frame prediction.

1.3 Motivation of this thesis

Since the introduction of AlexNet [10], CNNs have been appraised according to performance or accuracy. Thus, network architectures have evolved without regard to model complexity and computational efficiency. In most inter-frame prediction architectures proposed in previous literature, video frames are divided into small blocks and CNNs are used to perform inter-frame prediction at the block level. Therefore, CNNs are used at every block in a frame in order to predict a video frame. As CNN architectures perform predictions based on the localized pixel values, the inter-frame prediction process is performed in frame level using smaller convolution filters in this thesis. Then, computationally intensive and time consuming block level processing can be avoided.

To perform inter-frame predictions at the frame level, the CNN architecture must be able to output the intermediate video frame once the two adjacent video frames are given as inputs. Thus, both inputs and outputs must have the same resolution and the CNN architecture has to be optimized in both accuracy and complexity. A CNN architecture with minimum parameters and optimum prediction accuracy must be selected to perform bi-prediction operation with low complexity. Therefore, we have selected a U-Net CNN architecture inspired by [11] to perform bi-prediction. This CNN architecture has been tested with different hyper-parameters in this thesis, to identify the optimum bi-prediction algorithm for prediction accuracy and computational complexity.

Current state-of-the-art floating point arithmetic CNNs are not well suited for use on embedded (IoT) hardware platforms with limited computing resources. Floating point arithmetic costs much more computing power compared to integer arithmetic. In addition, the implementation of floating point arithmetic is platform dependent and this is a drawback for applications concerning interoperability, such as video coding [12]. On the other hand, integer networks provide the advantage of a smaller model, faster inference and cross platform consistency. In this thesis, we also investigate an integer arithmetic only CNN for bi-prediction, to reduce the complexity of the inter-frame prediction process.

In this thesis, the computational complexity of the CNN based bi-prediction method is compared with that of one of the simplest motion estimation algorithms available in a commonly used implementation of a H.265 standard video encoder, the HM 16.22 reference implementation. This is a platform independent analysis anchored on the number of integer arithmetic operations performed by both methods. In addition, we have experimentally evaluated the performance of the HEVC standard video encoder with both conventional motion estimation/compensation based bi-prediction and the proposed CNN based bi-prediction. These results show that the CNN approach can not only improve the encoding speed but can also provide noticeable gains in rate-distortion performance.

1.4 Outline of the thesis

The main focus of this thesis is to develop a low complexity bi-prediction architecture based on CNNs for video coding.

A literature review of the research work on different complexity reduction approaches used in video coding standards, motion estimation models for inter-frame prediction and learning based prediction models are presented in Chapter 2 of this thesis.

Chapter 3 describes different metrics used in this thesis to evaluate the performance of video codecs.

The bi-prediction techniques used in the HEVC standard HM 16.22 reference encoder are discussed and an analysis of the associated computational complexity is presented in Chapter 4.

In Chapter 5, the architecture, training procedure and the computational complexity of CNN bi-prediction algorithms are discussed. The method used to convert the floating point CNN to an integer-only CNN is also discussed.

The experimental results are presented in Chapter 6. Finally, the conclusions and contributions of this study are summarized, and avenues for further research are suggested in Chapter 7.

Chapter 2

Literature Review

2.1 HEVC Complexity Reduction Approaches

The HEVC standard saves approximately 50% of bit-rate at similar video quality compared to the previous generation H.264/AVC standard. This is achieved by using advanced video coding techniques which lead to a much higher computational complexity. According to [13], the encoding time of HEVC is 253% higher than H.264/AVC on average. The existing research on HEVC complexity reduction can be classified into two categories: heuristic and learning based approaches. In heuristic approaches, the brute-force rate distortion optimization search of coding unit (CU) partitioning has been simplified using some intermediate features. At the frame level, [14] proposed a fast coding unit decision algorithm that skips several CUs by analysing the utilization rate of CUs in previous frames. Kim et al. [15] proposed a fast pyramid motion divergence based CU selection algorithm to simplify the HEVC inter-frame prediction process. [16] considers computation friendly features such as rate distortion (RD) cost, inter mode prediction error and derives CU splitting decision based on the rule of minimizing the Bayesian risk. In addition to simplified CU partitioning, various heuristic approaches were proposed to reduce complexity of PU in inter-frame prediction. [17] has proposed a fast PU size decision approach which combines smaller PUs into larger PUs depending upon the video frame content. Estimating the PU partition with the maximum probability, on account of the coding block flag and the RD costs of encoded PUs is given in [18]. In addition, intra-frame and inter-frame prediction and in-loop filtering have been simplified to reduce the encoder complexity in [19–23].

Recently learning based fast coding approaches have been developed for complexity reduction in HEVC. Block partitioning and motion selection in video coding are modelled as data classification via online or offline training in these algorithms. For example, [24] has modelled the CU partition process of the intra mode as a binary classification problem with logistic regression. Support vector machines (SVM) have been used to perform CU partition classification in [25]. In [26], several HEVC domain features that are correlated with inter mode CU partitioning were explored and a joint classifier of SVM has been proposed to utilize these features to determine the CU depths. Further, Corrêa et al. [27] proposed three early termination schemes with data mining techniques to simplify the decision on the optimal coding tree unit (CTU) structures. These approaches can reduce computational time dedicated for brute force rate search of CU partitions by using well trained classification models. In addition, binary and multi-class SVM algorithms [28] have been proposed to predict both CU partition and PU mode with offline training. The encoding time of HEVC is further reduced in using this approach as online training computations are excluded from the video coding process. However, the above learning based approaches rely heavily on hand-crafted features that are related to the CU partition. Deep learning can be used to automatically extract CU partition related extensive features, rather than the limited hand-crafted features. In [29] and [30] a shallow CNN architecture has been used to predict the intra mode CU partition. The encoding time is further reduced in [31] by region wise feature classification using a CNN for intra mode prediction in CU partitions. [32] proposed a hierarchical CU partition map which reduces the computational time of the CU partition by identifying the complete CU partitions in a whole CTU in terms of one hierarchical CU partition map by running the trained ETH-CNN/ETH-LSTM model only once. The ETH-LSTM has been developed to learn the dependencies among CU partitions across frames for inter mode HEVC.

All the above stated complexity reduction approaches focus on simplifying the rate distortion optimization (RDO) search of quad-tree partitioning process of the HEVC standard. According to [5], the inter-frame prediction is shown to consume 84% of encoding time while only 3% is taken by intra-frame prediction for HD sequences. The high encoding time during inter-frame prediction is largely due to the complexity of the state-of-the-art motion estimation and motion compensation techniques used in the HEVC standard. Motion estimation is an effective tool to find the best matching block in a reference frame to a given block, to reduce the temporal redundancy between successive frames of a video sequence. Although motion estimation plays an important role in video coding, the computational complexity of this process is very high. Therefore, it is necessary to reduce the computational complexity and speed up the motion estimation process. As a result, the common optimization methods used to reduce HEVC complexity, focus on decreasing the number of motion estimation operations during the inter-frame prediction process [5]. Therefore, alternate methods for video frame prediction have been investigated with the objective of reducing the complexity and improving the accuracy of prediction in the HEVC standard.

2.2 Advance Methods for Motion Estimation

In conventional approaches to motion estimation [33, 34], the prediction is derived directly from one individual reference block or a linear combination of the multiple reference blocks. Thus, the pixel-wise motion cannot be fully modeled and will result in a large residue between the prediction and the block to be coded, which costs a lot of bits in coding. Over the past few years, several new approaches have been proposed as alternatives to the state-of-the-art motion estimation in video coding. Yin et al. in [35] proposed a weighted prediction strategy where the weighting factors are derived from the neighboring blocks. This algorithm achieved significant coding performance improvement in the H.264/AVC standard. But the linear combination of the forward and backward predictions was not sufficient to handle the complicated motions that occur in practical scenarios. A local affine motion compensation framework for complex motion which supports multiple reference frames for HEVC has been proposed in [36]. However, each pixel in the prediction block derived by affine motion is still computed as the average of two corresponding pixels and this linear combination process largely ignores complicated motions. Multi-hypothesis prediction methods to reduce compression artifacts and improve the quality of block transform video coding have been proposed in [37–39]. But the irregular motion patterns of real world objects can not be effectively described by the block wise translational motion model used in the inter-frame prediction in the HEVC standard.

Deep learning has presented significant advantages in dealing with complex non linear tasks such as image classification [10], image super resolution [40], etc. Dosovitskiy et al. in [8] proposed a Flownet to directly predict optical flow from two input video frames. [9] proposes a robust video frame interpolation method using a deep CNN. These literature validate the ability of CNNs in estimation and representation of complex motion. Yan et al. in [41] introduced a CNN-based interpolation filter to improve the accuracy of half pixel interpolation in motion estimation. The performance of the filter module is further upgraded with the support of a deep neural network in [42]. These literature have shown the ability of CNNs to develop a motion estimation algorithm.

2.3 CNN Architectures for Frame Prediction

In previous work, different CNN architectures have been designed to perform prediction tasks. In particular, Zhou et al. [43] trained a CNN to predict the appearance flows, which was then used to reconstruct the target view. LSTMs have been used to extrapolate images [44], but the results were observed to be blurry. [45] and [46] have used adversarial training and unique loss functions to reduce blurriness. However, the prediction capability of deep learning were not fully exploited in the scenario of bi-directional motion compensation in these literature. In [11], a deep neural network was used to generate dense voxel flows to optimize frame interpolation results. An encoder-decoder (U-Net) CNN architecture is used in this approach where the output is the prediction of the target video frame and the input to the network is the two adjacent video frames. This CNN architecture has shown better results in prediction quality compared to the previous approaches.

CNN architectures have also been designed to perform inter-frame prediction at the CTU level. Among these approaches, adaptive separable convolution shows considerable superiority in terms of both interpolation quality and complexity. Niklaus et al. [47] formulated pixel interpolation as a local convolution process, and provided an adaptive convolution approach which combines motion estimation and pixel synthesis in a single step. A major drawback of adaptive convolution lies in its memory cost. As predictions are performed at CTU level, it is stated in [48] that more than 20 GB memory overhead is needed to generate the intermediate frame of a 1080p video sequence, which makes it impractical in real time applications. In view of this, Niklaus et al. [48] then proposed adaptive separable convolution algorithm which approximates 2D convolution kernels with two 1D kernels. In this manner, an N×N convolution kernel can be encoded using only 2N variables, showing considerable superiority than 2D convolution version. In [49], this adaptive seperable architecture has been used to perform inter-frame prediction in the HEVC encoder, and it was observed that the time complexity in the encoder increased by $30-45\%$ and further, the time complexity of the decoder increased by about 70 times. The burden in time complexity makes this method not applicable to real time scenarios. In [50], a CNN based method has been proposed to refine the motion compensation in video coding. This is a uni-prediction architecture used after motion compensation algorithm to improve the accuracy of the prediction. It was observed that better compression performance can be achieved by integrating this CNN architecture into the state-of-the-art motion estimation in HEVC. This architecture is highly complex as CNN adds additional computations to the existing inter-frame prediction process in the HEVC standard. State-of-the-art HEVC standard adopts the bi-prediction (B-frames) to improve the coding efficiency of the inter-frame prediction process. Recently, the prediction capability of deep learning has been exploited by [51] for bi-directional motion compensation. This approach has delivered better predictions compared to traditional bi-prediction in HEVC. The input to the proposed CNN in [51] is a PU. The computational expense of RDO search in quad-tree partitioning is present in this method as the predictions are performed at PU level. Therefore, the computational complexity of an HEVC encoder is further increased in using this method.

Based on the results reported in the literature, we can draw the following conclusion regarding the use of CNNs for inter-frame prediction of a HEVC standard encoder. If a very deep CNN as required for good inter-frame prediction is used at PU or CU levels, the computational complexity and the memory requirements of the associated RDO process can increase significantly. This will make the method impractical in real-time video coding applications. In view of this observation, this thesis considers in Chapter 5, the use of deep CNNs for frame level prediction, rather than for PU or CU level predictions.

Chapter 3

Metrics for Assessing Performance of Video Codecs

The quality of a compressed video signal and the compression efficiency of the video encoder play an important role in determining the performance of a video coding method. This chapter describes the different metrics we will use to evaluate the performance of video codecs in this thesis. The two objective metrics, peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM) will be used for video quality assessment. Compression efficiency of video codecs will be evaluated by Bjontegaard metrics.

3.1 Video Quality Estimation Models

3.1.1 PSNR

The PSNR performance metric [52] is very commonly used to quantify the video quality. PSNR is defined as

$$PSNR = 10\ \log_{10}\left(\frac{MAX_I}{MSE(g,p)}\right)\ (dB),$$

where MAX_I is the maximum pixel value possible in a video frame, and $MSE(g,p)$ is the mean square error (MSE) between the ground truth image g and predicted image p. Let $g(i,j)$ and $p(i,j)$ denote the pixel values at the location (i,j) in the ground truth image and the predicted image respectively, where $i = 1,..,N$, $j = 1,..,M$. Then, the MSE is given by

$$MSE(g,p) = \frac{1}{NM}\sum_i \sum_j \left((g(i,j) - p(i,j)\right)^2.$$

The PSNR, which is a pixel-wise error measure can be better for evaluating noisy images [53].

3.1.2 SSIM

SSIM is an image quality measure based on certain features of the human visual system, proposed by Wang et al. in [54]. It is an improvement over measures like MSE and PSNR [55]. The main function of the human visual system is the extraction of structural information from the viewing field. The SSIM is an attempt to capture this principle in the same way. SSIM is a full reference objective video quality metric which measures the structural similarity between two video frames. SSIM was selected in this thesis as it has been found to be sometimes more useful in relative performance evaluation studies [55], [56].

The SSIM computes a score for each individual pixel, using a window of neighboring pixels. These scores can then be averaged to produce a global score for the entire image. The SSIM is usually normalized to produce a score in the (-1, 1) interval, where a value closer to 1 indicates a high similarity between the two images.

3.2 Rate Distortion Performance Assessment

The compression performance of a video codec is usually measured by the average distortion it introduces at a given bit rate. This is referred to as RD performance. A set of metrics that is widely used to compare the RD performance of two different video codecs or the same video codec at two different settings is Bjontegaard metrics [57] which involve two parts: BD-PSNR and BD-Rate.

Given a reference codec with two different settings, BD-Rate or BD-PSNR values are calculated as follows:

1. Several RD points are calculated for the reference codec with setting 1 and setting 2. At least four points must be computed. These points should be obtained using the same quantizers when comparing two versions of the same codec.

2. Additional points outside of the range should be discarded.

3. The rates are converted into log-rates.

4. A piece-wise cubic hermite interpolating polynomial is fitted to the RD points for each setting to produce functions of log-rate in terms of distortion.

The metric score ranges are computed as follows:

1. In comparing two versions of the same codec, the overlap is the intersection of the two curves, bounded by the chosen quantizer points.
2. The log-rate is numerically integrated over the metric range for each curve, using at least 1000 samples and trapezoidal integration.
3. The resulting integrated log-rates are converted back to linear rate, and then the percent difference is calculated from the setting 1 to the setting 2.

BD-PSNR (dB): This metric evaluates the quality gains made by the codec setting 1 compared to the codec setting 2 at the same bit-rate. So if the BD-PSNR value is 5 dB, then it means that the second settings gives 5 dB more quality than the first setting at the same bit-rate. The calculations can also be done with other objective quality metrics such as the SSIM (BD-SSIM).

BD-Rate (%): This metric evaluates the bit-rate savings at the same quality and is also referred to as the BD-BR value. If the quality of both codec settings are the same, then BD-Rate is used to find which setting uses more bits and how much. If the value of BD-Rate is 50%, it means that setting 2 needs 50% more bits than setting 1 to achieve the same quality.

In the upcoming chapters, these metrics will be used to compare the video quality and compression efficiency of video codecs which use motion-estimation based prediction and CNN based prediction.

Chapter 4

Complexity Analysis of Bi-Prediction in a HEVC Standard Encoder

As discussed in Chapter 1, one of the most time consuming operations in a modern hybrid video encoder is the motion estimation used for predicting video frames from reference frames. In this chapter we present an analysis of the computational complexity of one of the most common implementations of the state-of-the-art HEVC encoder, namely HM 16.22 reference software. This video encoder will be used in the experimental work presented in Chapter 6. We determine the number of addition and multiplication operations required for motion estimation based bi-prediction process in HM 16.22 reference encoder.

HEVC standard follows the same basic hybrid video coding approach as shown in Fig. 1.1. Block based motion compensated prediction (MCP) assumes that all motions inside a block are homogeneous and any moving object is larger than the prediction block size. In inter-frame prediction, suitably chosen blocks of pixels in already coded reference frames are used as the predictors for blocks in the frame. The concept of MCP based on a translational motion model is illustrated in Fig. 4.1. In the translational model, a block of pixels in a given frame is assumed to be a translated version of a similarly sized block in another already coded, temporally close frame referred to as the *reference frame*, as shown in Fig. 4.1. The motion information of the block is represented by a vector $(\Delta x, \Delta y)$, where Δx and Δy represent the horizontal and vertical distances between the given block and the predictor block respectively. In a video encoder, the reference frames are stored in a buffer referred to as the *reference picture list* where each stored frame is identified by a *reference picture index*. The difference in display order of the reference picture and the current picture to be encoded is given by a reference picture index Δt. In the HEVC standard, the motion vectors and the reference indices define the motion data required for predicting a block. The motion vectors are also allowed to have a fractional-pixel accuracy to capture the motion of objects more accurately. When motion compensation is to be performed at fractional accuracy, the pixels in the reference picture are interpolated to obtain the reference blocks.

In a video codec, the motion data for each predicted block must also be encoded and conveyed to the decoder. Since motion data of neighboring blocks can be highly correlated, modern video codecs employ predictive coding of motion data. In the HEVC standard, efficient predictive coding of motion data is achieved by using a procedure referred to as *advanced motion vector prediction* (AMVP). Furthermore, so called *inter prediction block merging* is used to reduce blocking artifacts in decoded images. Even though these techniques improve the codec performance, they also increase the computational complexity of motion estimation.

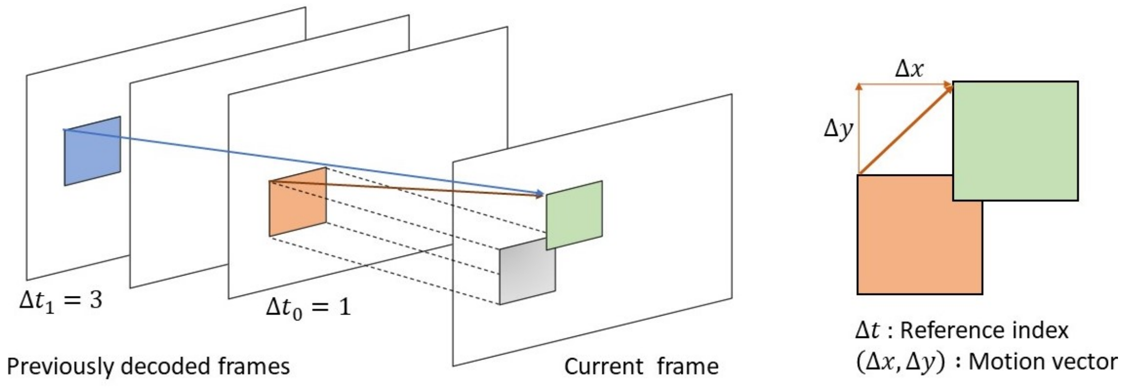

$\Delta t_1 = 3$
$\Delta t_0 = 1$
Previously decoded frames
Current frame
Δx
Δy
Δt : Reference index
$(\Delta x, \Delta y)$: Motion vector

Fig. 4.1: Inter-frame prediction using the translational motion model.

Inter-frame prediction can be performed by two methods: uni-prediction and bi-prediction. In the perceptive of coding efficiency, bi-prediction provides higher efficiency compared to uni-prediction [58]. In the case of bi-prediction, two sets of motion data are derived using motion estimation process, and the final prediction for a given block is obtained by averaging or weighted prediction, see section 4.3.5. In this thesis, we focus only on bi-prediction for inter-frame prediction.

The block-based motion estimation is essentially a search algorithm. The only way to ensure that the best matching predictor block is found, is to use an exhaustive search, commonly referred to as a *full search*. In a typical full search algorithm, all possible blocks inside a given search range in a reference frame are compared with the block to be predicted, to find the best matching block in terms of a cost function. It has been shown in [59] that when a full search is used, upto 96% of the encoding time can be consumed by motion estimation. Therefore, reduced complexity sub-optimal search algorithms are used to reduce computational time. One such reduced complexity search method widely used is the test zone (TZ) search algorithm described in section 4.3.4. This algorithm is used in the HM 16.22 reference codec that will be used in our experimental work.

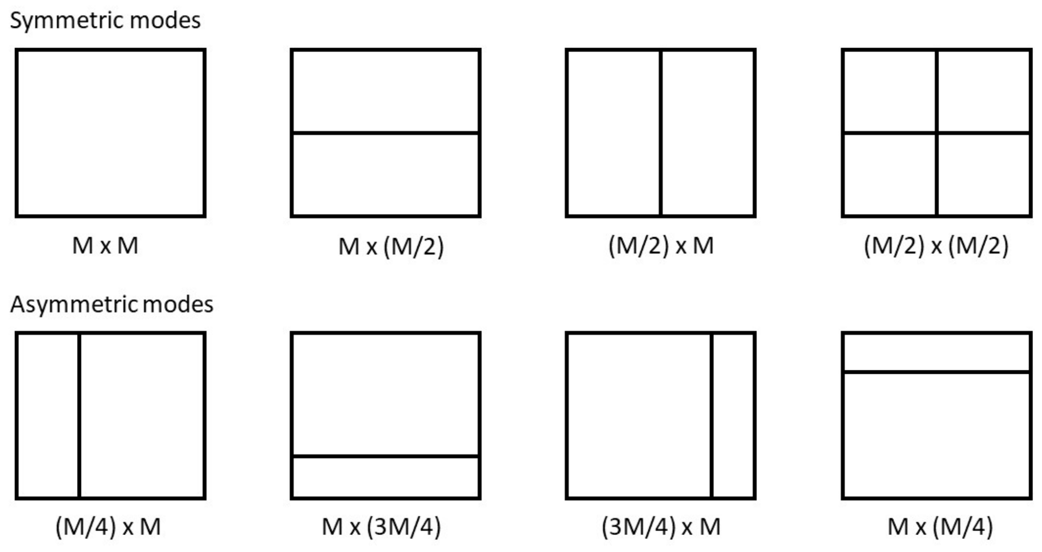


Fig. 4.2: PU modes supported in the HEVC standard.

4.1 Overview of HEVC Block Partitioning

In HEVC standard, each video frame is partitioned to blocks of size $2^N \times 2^N$ where N can be set to either 4, 5, or 6. These partitions are known as coding tree units (CTUs). While a CTU can be coded as a single block, each CTU may also be further sub-divided into smaller blocks through a hierarchical quad-tree partitioning process to improve the prediction efficiency. The blocks in the final partitioning are referred to as coding units (CU). That is, a complete CTU may form a single CU, or a CTU may be recursively split into 4 equally sized smaller CUs, until the minimum CU size of 8×8 is reached. The partitioning decision at every level of the recursive process is based on the RD benefit.

Since a single set of motion data may not be able to accurately predict a complete CU, a CU may be further split into smaller prediction units (PU). A PU is a block of pixels that uses the same motion parameters for inter-frame prediction. For each PU, a single set of motion parameters is signaled in the encoder output bit stream

[60]. In the HEVC standard there are 8 possible PU modes available for splitting a CU (see Fig. 4.2), even though the actual PU modes used by the encoder depends on how one configures the encoder. In symmetric PU modes, a CU is either predicted as a single block or further split into two or four equal size PUs. The $(M/2) \times (M/2)$ mode is only supported for the minimum CU size which should be larger than 8×8 luma samples [60]. In asymmetric modes, the square section of a CU is divided into (1/4) and (3/4) units as shown in Fig. 4.2. These modes are useful when only a small part of a CU shows significantly different motion vectors. Asymmetric PU modes are only supported if the the selected minimum CU size is greater than 8×8 luma samples. Thus, 8×4 and 4×8 are the smallest PU sizes that can be used in inter-frame prediction.

In general the encoding process involved in an HEVC encoder depends on how the encoder configuration is set by the user and can be very complicated. The HM 16.22 reference implementation of the HEVC codec allows multiple options for hierarchical block-partitioning and motion estimation which offer varying levels of performance-complexity trade-offs to a user. In this chapter, we focus on the simplest possible option to facilitate a fair comparison of the complexities between motion estimation based bi-prediction and CNN based bi-prediction. For a comprehensive description of the complete HEVC encoding process, see [61].

4.2 Minimum-complexity Encoder Configuration

In this thesis we use the following simple set of settings (see Fig. 4.3) in the HM 16.22 reference implementation of a HEVC codec, to keep the computational complexity of motion estimation to a minimum.

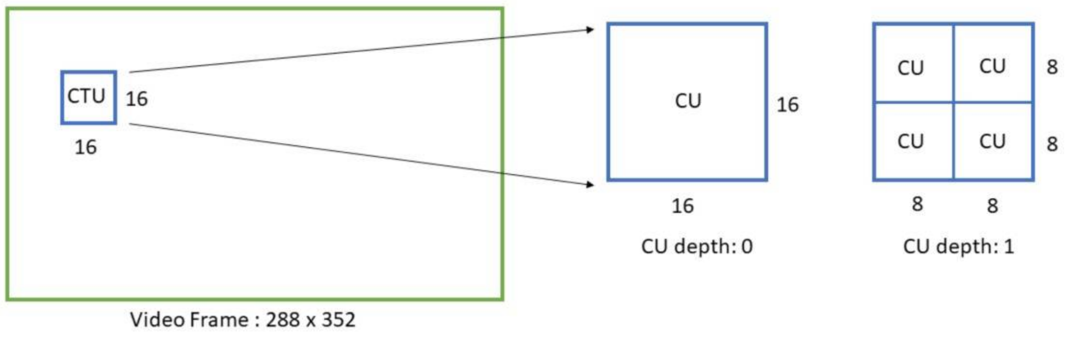


Fig. 4.3: CU partition modes considered in this study.

- CTU size is 16×16.
- The Maximum CU partitioning depth is 1. That is, a CTU can only be further divided into 4 sub-blocks. Therefore, the available CU sizes are restricted to 16×16 and 8×8 as shown in Fig. 4.3.
- PU depth for inter-frame prediction is 1 and only symmetric PU modes are used. The PU sizes allowed for inter-frame prediction are given in Table 4.1. Note that the PU size 4×4 is not available for the 8×8 CU size, as it is smaller than the minimum PU size, see section 4.1.

With this simple block-partitioning scheme, the computational complexity of the motion estimation based frame bi-prediction can be determined by identifying the number of addition and multiplication operations required for the prediction of one PU. The total computational complexity of predicting a single frame can then be obtained by multiplying this number by the number of PUs in a frame. In order to determine the number of computations required for bi-prediction of a single PU, we need to consider the following aspects:

- Advanced motion vector prediction (AMVP) and inter-prediction block merging processes used to determine the candidate blocks.

Table 4.1: PU modes allowed in the minimum-complexity encoder configuration.

CU size	Available PU sizes
16×16	16×16, 8×8, 8×16, 16×8
8×8	8×8, 8×4, 4×8

- Search algorithm used to find the best matching predictor blocks from the two reference frames.
- Pixel interpolation process used for motion estimation with fractional pixel accuracies.

4.3 Bi-prediction Process

In this section, we briefly describe the main operations involved in the bi-prediction process that have to be considered in determining the computational complexity. The prediction of a PU starts with the initialization of motion vectors using motion vector predictors (MVPs) identified in the AMVP process.

4.3.1 Advanced motion vector prediction (AMVP)

In an HEVC encoder, AMVP involves a process known as *motion vector competition* to select the best prediction for a given PU. This is a complicated process due to the quad-tree partitioning of a CTU, where a single PU can have multiple neighboring PUs of different sizes, therefore identifying the best predictor may be difficult. AMVP is used to simplify the motion vector competition for such a complex block structure.

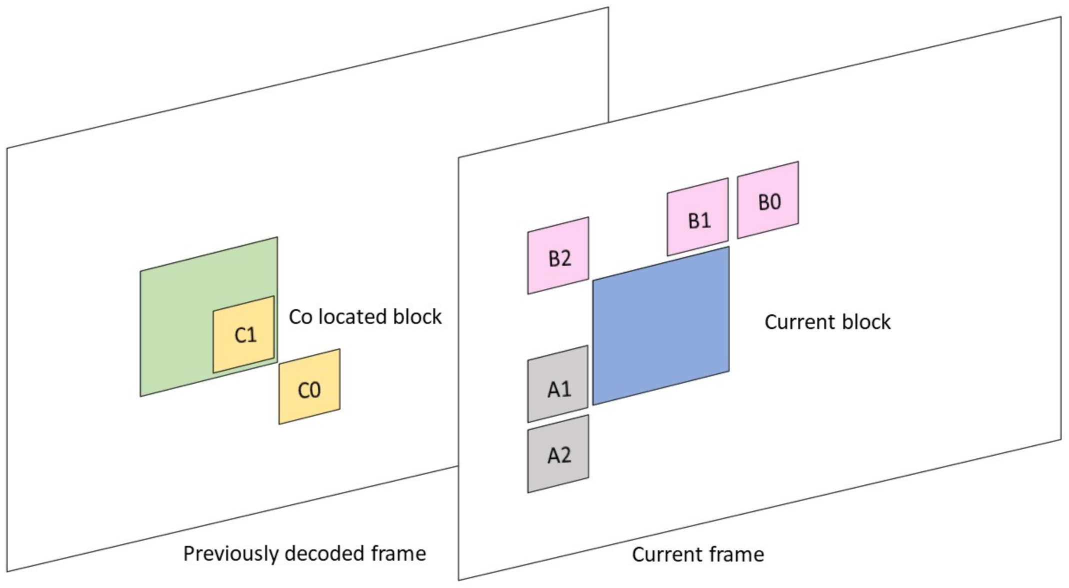


Fig. 4.4: Candidate blocks in AMVP.

The AMVP candidate list is generated by selecting the most suitable MVPs from different types of predictors as given below.

- *Upto 2 spatial candidate MVPs that are derived from 5 spatial neighboring blocks*: Motion vector differences (MVD) between the motion vector (Δx, Δy) of the current block and the MVPs of the 5 spatial candidates A_1, A_2, B_0, B_1 and B_2 (illustrated in Fig. 4.4) have to be calculated. Given a spatial candidate, the MVD components are calculated as

$$MVD_x = \Delta x \; - \; MVP_x,$$
$$MVD_y = \Delta y \; - \; MVP_y,$$

where MVP_x and MVP_y are the MVP components of the spatial candidate. If the reference picture index of the current block is different to that of the candidate block then the motion vector should be scaled according to the temporal distance between the current reference picture and the candidate reference picture [62]. In frame level prediction, this calculation is not performed for spatial candidates as all blocks in one frame are predicted using the same reference picture set. Two candidates out of A_1, A_2 and B_0, B_1, B_2, with the minimum error are included in the AMVP candidate list.

- *One temporal candidate MVP is derived from two temporally co-located blocks*: In the HEVC standard, the blocks to the bottom right (C_0) and the blocks in the center of the co-located blocks (C_1) (illustrated in Fig. 4.4) are determined to provide a good temporal motion vector predictor for the current block [61]. Motion vectors of the temporal candidate MVPs can not be directly used for the MVD calculation as the reference pictures of the current block and candidate blocks can be different. Motion vectors of the candidate MVPs need to be scaled according to the temporal distances between the candidate reference picture and the current reference picture. Then the MVD is calculated for scaled temporal MVPs with respect to the current block MV, and the candidate with minimum error is added to the AMVP list.

- *Zero motion vectors are added to the candidate list when neither spatial nor temporal candidates are available.*

4.3.2 Inter-prediction block merging

Inter-frame prediction process in HEVC standard is based on prediction of PUs using similar-sized blocks from temporally neighboring reference frames. The complex quad-tree partitioning process introduces a large number of borders which can interrupt the motion information in a video frame. A block merging algorithm is used to identify a list of merge candidate blocks which improves the accuracy of inter-frame prediction through joint description. A merge candidate comprises of all motion data: motion vectors and reference picture indices. Merge candidate lists consist of spatial and temporal candidates. Initially, 4 spatial candidates are derived from 5 spatially neighboring blocks. Then, the motion data of the 4 candidate blocks are copied to the merge candidate list. Two temporal merge candidates are added to the list in bi-prediction mode. These candidates are selected out of 2 co-located blocks from each reference frame. No motion vector scaling is done but the motion information of the two candidates are copied to the candidate list. When a frame is bi-predicted, additional candidates can be generated by combining the reference list 0 motion data of one candidate, with the reference list 1 motion data of the other. If the list is not full after adding bi-predictive candidates then zero motion vector candidates are added finally.

4.3.3 Fractional sample interpolation

HEVC standard supports motion estimation at fractional-pixel accuracy to capture the continuous motions more accurately. The standard supports motion vectors with quarter-pixel accuracy for luma component and one-eighth pixel accuracy for chroma components. Unlike the older H.264/AVC standard, quarter pixel values in the HEVC standard are derived without using half pixel values, but by directly applying a 7 or 8 tap filter on integer pixels.

In HEVC bi-prediction, if one of the prediction blocks from the two prediction lists has sub-pixel accuracy then instead of averaging each prediction block at the precision of the input bit depth in the interpolation process, they are averaged at the higher precision. Longer filter tap lengths increase the number of arithmetic operations required to obtain the interpolated sample. Interpolation filter in HEVC requires 20% more multiplication and addition operations compared to the H.264/AVC filter for 8-bit video [60].

4.3.4 TZ search algorithm

TZ search algorithm has been adopted in the HM 16.22 reference implementation of the HEVC encoder that is considered in this thesis, to perform fast motion estimation using only integer arithmetic operations. This algorithm is divided into four steps as follows.

1. Prediction motion vector selection

   A motion vector predictor is used to identify the initial search center for a given PU block for which a prediction is to be computed. This predictor is derived using the AMVP candidate list which includes both temporal and spatial candidates. A temporary best predictor is computed by deriving the mean of the motion vectors of the spatial candidates [63]. This best predictor candidate is used as the start point of the TZ search. If only the spatial candidates are less than 2, the temporal candidates are considered for this mean calculation.

2. Zonal search

   After identifying the initial starting point, the next step is to determine the search range and the search pattern for an initial grid search. This step involves using a search window of either a diamond or a square shape where the size of search area varies from 1 up to the search range. In HM 16.22 encoder implementation, the maximum search range is 64. In this encoder implementation, the distance between the search points are multiples of 2,

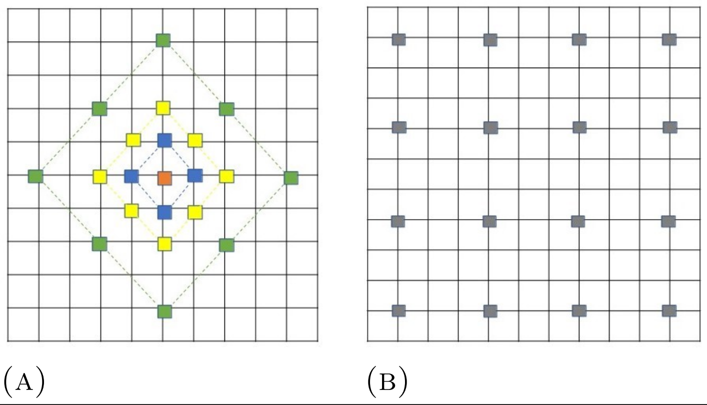


Fig. 4.5: TZ search methods: (A) Eight point diamond search, (B) Raster search with stride length = 3.

which leads to 8 search points at each step for either square or diamond shape search. The diamond search patterns for distance one to three are shown in Fig. 4.5(A). The distance between the current search center and the minimum distortion point out of all points in a given pattern is stored in the *Bestdistance* variable. If *Bestdistance*=0 after the search is completed, no additional steps are conducted.

3. Raster search

   A parameter *iRaster* is used to specify the maximum distance that is allowed between the current search center and the minimum distortion point in the zonal search. If the *Bestdistance* found in the zonal search is greater than zero but less than *iRaster*, then a refinement step is carried out directly [64]. A raster scan is performed using *iRaster* value as the stride length if *Bestdistance* > *iRaster*. This is a form of full search conducted using stride length as the gap between the search points. A raster search with a stride length of 3 is illustrated in Fig. 4.5(B).

With the minimum-complexity encoder configuration that we consider in this thesis, the refinement stage of the TZ search algorithm is not considered. The best estimate for the motion vector to predict the current PU from a PU of the given reference video frame is identified by the TZ search process.

4.3.5 Weighted sample prediction

After the TZ search, two sets of motion data are derived for the current PU for bi-prediction. These two sets of motion data identify the best matching PUs from the two reference frames. Weighted prediction is applied to these two PUs to derive the final prediction of the current PU as

$$\hat{P} = P_{L1}[x][y] * w_1 + P_{L2}[x][y] * w_2 + (O_1 + O_2), \tag{4.1}$$

where P_{L1} and P_{L2} are the PUs from reference frame 1 and reference frame 2 respectively, x and y are the spatial coordinates of the PUs. Weighting factors w_1, w_2 and offsets O_1, O_2 are determined according to the reference picture lists. In HEVC standard, the weight and offset parameters are explicitly specified and no derivation is required. Deriving the weighted prediction from the two motion data sets is the final step of the bi-prediction process.

4.4 Computational Complexity of Bi-Prediction

Motion estimation in HEVC is performed at PU level where PUs are obtained by hierarchical partitioning of a video frame. Several factors contribute to the high computational complexity of this process.

1. A RD cost has to be computed at each level of quad-tree partitioning.

2. RD cost computation at a given CU level has to be repeated for all 8 PU modes shown in Table 4.1.

3. AMVP and inter-prediction block merging processes require that candidate lists with both spatial and temporal candidates be derived.

4. Each TZ search algorithm consists of zonal and raster search procedures, in which a distortion calculation is performed at each point. This complex search algorithm is considered the most computationally expensive process in the inter-frame prediction.

In the following, we determine the number of addition and multiplication operations involved in the computation steps of the motion estimation, for a single PU in the minimum-complexity encoder configuration described in section 4.2.

4.4.1 Motion vector prediction

This analysis is based on the assumption that every PU has the minimum number of MVP candidates allowed in the HEVC standard. Therefore, 7 MVP candidates are present for every PU in the current frame, where 5 of them are neighboring PUs from the same frame (spatial candidates) and 2 candidates are from the co-located PUs from each reference frame (temporal candidates). We assume that the size of MVP candidates are always equal to the size of the PU being predicted, which is a reasonable assumption for the encoder configuration in section 4.2. In the analysis to follow, we consider the 8×8 PU block size, as this PU block size is common to both CU sizes given in Table 4.1. It is assumed that half-pixel and quarter-pixel interpolations are used in fractional-accuracy motion estimation.

Table 4.2: Filters used in HEVC standard for fractional-pixel interpolation.

Position	Filter coefficients
1/4	$-1, 4, -10, 58, 17, -5, 1, 0$
2/4	$-1, 4, -11, 40, 40, -11, 4, -1$
3/4	$0, 1, -5, 17, 58, -10, 4, -1$

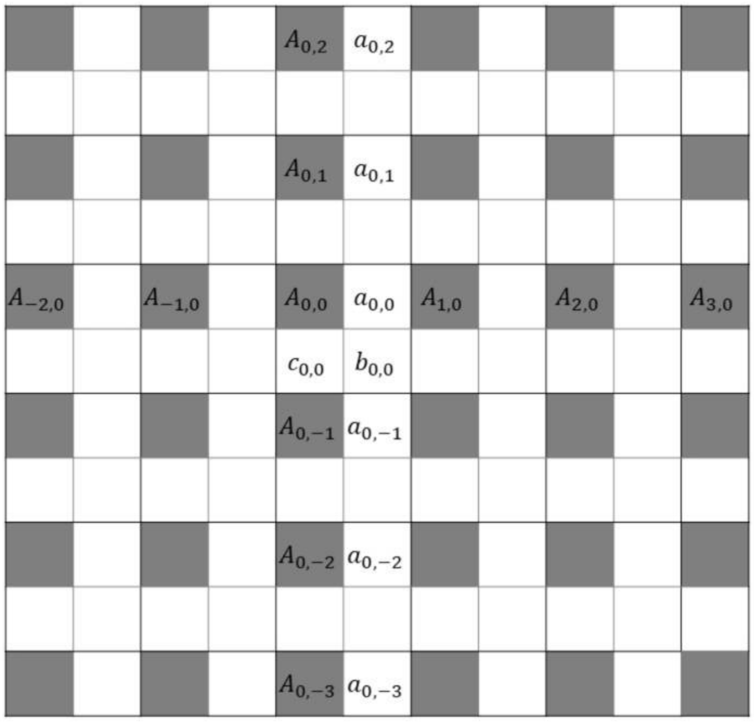


Fig. 4.6: Integer pixels (shaded blocks with upper-case letters) and fractional-pixels (un-shaded blocks with lower-case letters) for half-pixel interpolation.

Half-pixel interpolation

In the HEVC standard, half-pixel interpolation is performed using the 8-tap filter given in Table 4.2 (filter in position 2/4). Half pixel interpolations performed to generate 3 fractional pixels are shown in Fig. 4.6. The fractional position $a_{0,0}$ is interpolated by horizontally applying a 1D filter as

$$a_{0,0} = -A_{-3,0} + 4A_{-2,0} - 11A_{-1,0} + 40A_{0,0} + 40A_{1,0} - 11A_{2,0} + 4A_{3,0} - A_{4,0}.$$

Similarly, the fractional position $c_{0,0}$ is interpolated by applying the same filter vertically to obtain

$$c_{0,0} = -A_{0,-3} + 4A_{0,-2} - 11A_{0,-1} + 40A_{0,0} + 40A_{0,1} - 11A_{0,2} + 4A_{0,3} - A_{0,4}.$$

The position $b_{0,0}$ is interpolated by, first applying a horizontal 1D filter eight times, to determine the values of $a_{0,-3}$, $a_{0,-2}$, $a_{0,-1}$, $a_{0,0}$, $a_{0,1}$, $a_{0,2}$, $a_{0,3}$ and $a_{0,4}$ separately, and then applying a vertical 1D filter to these values. Accordingly

$$b_{0,0} = -a_{0,-3} + 4a_{0,-2} - 11a_{0,-1} + 40a_{0,0} + 40a_{0,1} - 11a_{0,2} + 4a_{0,3} - a_{0,4}.$$

It follows that each half-pixel interpolation requires 8 multiplications and 7 additions.

Quarter-pixel interpolation

Quarter-pixel interpolation is performed by using all three filters given in Table 4.2 [65]. For example, the positions $a_{0,0}$, $b_{0,0}$, $c_{0,0}$ illustrated in Fig. 4.7 are derived by using 1D horizontal filters as,

$$\begin{aligned}
a_{0,0} &= -A_{-3,0} + 4A_{-2,0} - 10A_{-1,0} + 58A_{0,0} + 17A_{1,0} - 5A_{2,0} + A_{3,0},\\
b_{0,0} &= -A_{-3,0} + 4A_{-2,0} - 11A_{-1,0} + 40A_{0,0} + 40A_{1,0} - 11A_{2,0} + 4A_{3,0} - A_{4,0},\\
c_{0,0} &= A_{-2,0} - 5A_{-1,0} + 17A_{0,0} + 58A_{1,0} - 10A_{2,0} + 4A_{3,0} - A_{4,0}.
\end{aligned}$$

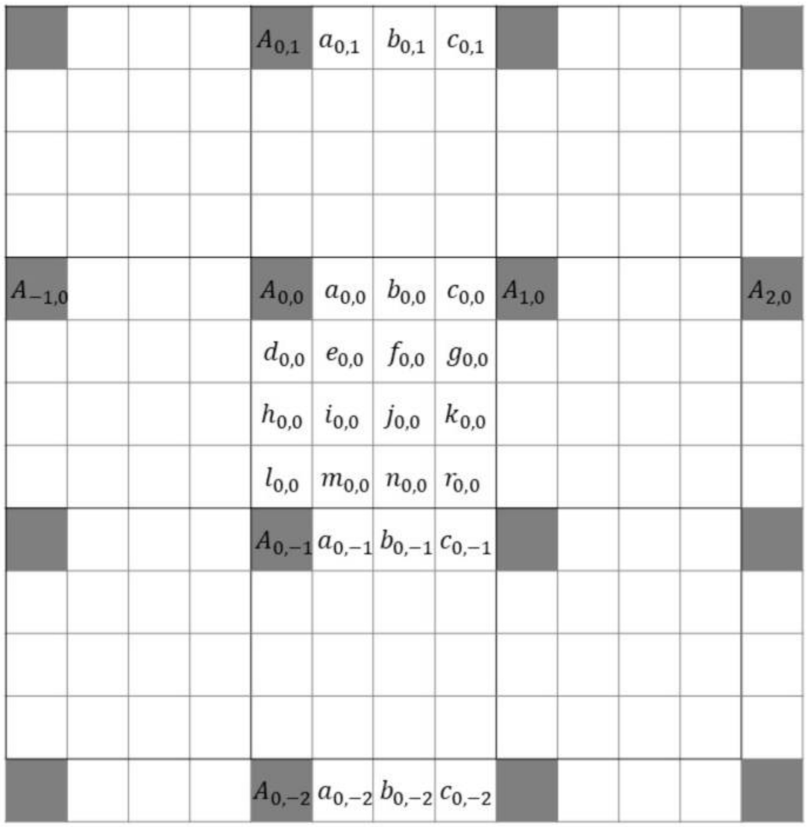

$A_{0,1}$ $a_{0,1}$ $b_{0,1}$ $c_{0,1}$
$A_{-1,0}$ $A_{0,0}$ $a_{0,0}$ $b_{0,0}$ $c_{0,0}$ $A_{1,0}$ $A_{2,0}$
$d_{0,0}$ $e_{0,0}$ $f_{0,0}$ $g_{0,0}$
$h_{0,0}$ $i_{0,0}$ $j_{0,0}$ $k_{0,0}$
$l_{0,0}$ $m_{0,0}$ $n_{0,0}$ $r_{0,0}$
$A_{0,-1}$ $a_{0,-1}$ $b_{0,-1}$ $c_{0,-1}$
$A_{0,-2}$ $a_{0,-2}$ $b_{0,-2}$ $c_{0,-2}$

FIG. 4.7: Integer pixels (shaded blocks with upper-case letters) and fractional-pixels (un-shaded blocks with lower-case letters) for quarter-pixel interpolation.

The positions $d_{0,0}$, $h_{0,0}$, $l_{0,0}$ are derived by applying the same 8-tap filter in vertical direction. Next, the positions $e_{0,0}$, $i_{0,0}$, $m_{0,0}$, $f_{0,0}$, $j_{0,0}$, $n_{0,0}$, $g_{0,0}$, $k_{0,0}$ and $r_{0,0}$ are derived by applying the 8-tap filter to the fractional-pel pixels $a_{0,i}$, $b_{0,i}$ and $c_{0,i}$ in vertical direction respectively as

$$e_{0,0} = -a_{-3,0} + 4a_{-2,0} - 10a_{-1,0} + 58a_{0,0} + 17a_{1,0} - 5a_{2,0} + a_{3,0},$$

$$i_{0,0} = -a_{-3,0} + 4a_{-2,0} - 11a_{-1,0} + 40a_{0,0} + 40a_{1,0} - 11a_{2,0} + 4a_{3,0} - a_{4,0},$$

$$m_{0,0} = a_{-2,0} - 5a_{-1,0} + 17a_{0,0} + 58a_{1,0} - 10a_{2,0} + 4a_{3,0} - a_{4,0},$$

where $i = -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ 4$.

Each quarter-pixel interpolation requires 8 multiplications and 7 additions similar to half-pixel interpolation [65]. The total number of half-pixel interpolations (n_{half}) and quarter-pixel interpolations ($n_{quarter}$) performed are given by

$$n_{half} = (2H - 1) \times (2W - 1) - (H \times W),$$
$$n_{quarter} = (4H - 3) \times (4W - 3) - (H \times W),$$

where H and W are the height and width of a PU respectively. For the 8×8 PU size we consider $H = W = 8$, and hence $n_{half} = 161$ and $n_{quarter} = 777$. Therefore, 8 multiplications and 7 additions are required for every half- or quarter-pixel interpolation, and a total of 1288 multiplications and 1127 additions are performed for half-pixel interpolation of one PU. It follows that, 5439 additions and 6216 multiplications are required for quarter-pixel interpolation of a single PU.

MVP candidate selection

For MVP candidate selection, fractional pixel interpolation is performed for all 7 candidates and the current PU. Thus, the number of arithmetic operations for one PU will be repeated for all candidate PUs. The total required multiplications and additions for fractional pixel interpolation of spatial and temporal candidates are given in Table 4.3.

To find the best 3 MVP candidates, the sum of average error (SAE),

$$SAE = \sum_{i=1}^{M} |r_i - c_i|, \tag{4.2}$$

TABLE 4.3: Integer operations involved in AMVP for one PU.

Process name		Additions	Multiplications	Total
Spatial	Half pixel interpolation	5635	6440	12075
	Quarter pixel interpolation	27195	31080	58275
	SAE calculation	11285	0	11285
Temporal	Half pixel interpolation	2254	2576	4830
	Quarter pixel interpolation	10878	12432	23310
	SAE calculation	4514	0	4514
	Scaling	12	8	20
Total computations		61773	52536	114309

where c_i and r_i are pixels in the current PU and the candidate PU respectively, is calculated for all spatial and temporal candidates at full and sub-pixel accuracies, and M is the number of pixels in a PU. For full, half and quarter pixel accuracies the values of M for an 8×8 PU are given by,

$$\begin{aligned} M_{full} &= H \times W = 64, \\ M_{half} &= (2H-1) \times (2W-1) = 225, \\ M_{quarter} &= (4H-3) \times (4W-3) = 841. \end{aligned} \tag{4.3}$$

At each accuracy level, $2M-1$ additions are required to compute the SAE between the current block and one candidate block. To find the best matching block for the current block, SAEs are calculated between the current block and all the candidate blocks. The total number of integer arithmetic operations required to calculate the SAEs are given in Table 4.3. After this computation, the two minimum SAE candidates out of the spatial candidates and the minimum SAE candidate out of the temporal candidates are added to the AMVP candidate list.

Spatial candidates can be directly added to the AMVP candidate list but before adding temporal candidates, a scaling operation must be performed as explained previously in section 4.3.1. In bi-prediction, scaling must be performed on temporal candidates from both reference frames. A scaling operation requires 6 additions and 4 multiplications per motion vector as described in Appendix A. Thus, for the 2 temporal candidates, this operation has to be performed twice to scale both motion vectors. A total of 114309 integer operations are performed to identify the AMVP candidates for one 8×8 PU. The total integer operations required to perform AMVP in all available PU sizes are given in Table 4.5.

4.4.1.1 Block merging

For spatial candidates, 5 neighboring PUs are selected from the merge estimation region (MER) of the same frame as the current PU. The SAEs between the current PU and each of the 5 candidates are calculated at both full resolution and fractional-pixel resolutions using (4.2). Four minimum SAE candidates out of the spatial candidates are added to the merge candidate list. As the temporal candidates, 2 PUs are considered from each reference frame and the best merge candidate is selected based on the minimum SAE criterion. The same set of computations used in AMVP are used for block merging as explained in section 4.3.2. Thus, the number of computations required for block merging is equal to the total number of integer operations in AMVP for spatial and temporal candidates except for the scaling operation. Scaling operation for motion vectors is not performed for temporal candidates in block merging as explained in section 4.3.2. The total number of integer computations required for block merging process is given in Table 4.5.

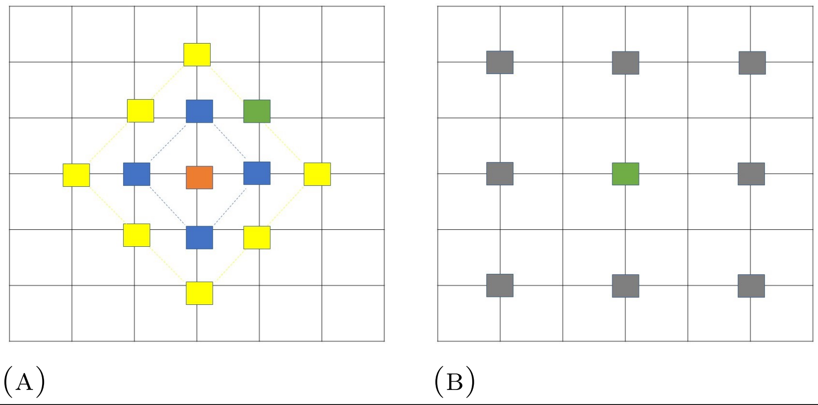


Fig. 4.8: TZ search points in a 48×48 search region for 8×8 PUs: (A) Zonal search, (B) Raster search with stride length = 2.

4.4.1.2 TZ search algorithm

To perform the TZ search, first the mean of AMVP spatial candidates is calculated to identify the initial search center. The position (x, y) of the initial search center is derived using the positions of the two spatial candidates s_1 and s_2 in the current frame as

$$center(x, y) = \frac{s_1(x, y) + s_2(x, y)}{2}.$$

Zonal search is performed over a search range of 48×48 pixels around the initial search center. It is assumed that the zonal search is conducted in diamond pattern only. This results in the lowest possible complexity. In this case, the diamond search is performed until the maximum possible stride length of 2 is reached, see Fig. 4.8(A). For the search area size 48×48, 8×8 PU size, and the stride length 2, that we consider, the TZ search can only be performed for 12 PU blocks as illustrated in Fig. 4.8(A). Therefore, the SAEs between the current PU and only the 12 neighboring PUs have to be computed. The PU with the minimum SAE identified in the diamond search is used as the search center for the raster search, which is performed inside a search range using a stride length of 2. Thus, the

Table 4.4: Integer operations required for TZ search for the bi-prediction of one 8×8 PU.

Process	Add	Multiply	In bi-prediction		Total
			Add	Multiply	
Half pixel interpolation	22540	25760	45080	51520	96600
Quarter pixel interpolation	108780	124320	217560	248640	466200
SAE calculation	45140	0	90280	0	90280
Total			352920	300160	653080

SAEs between the current PU and 8 other PUs have to be computed in the raster search. That is, if we consider the green block in Fig. 4.8(A) to be the PU with the minimum SAE found in the zonal search, then a raster search is conducted with the 8 neighboring PUs around the minimum SAE PU as indicated in Fig. 4.8(B), as we consider 48×48 search region and the stride length 2.

The SAE calculations are performed for full, half and quarter pixel resolutions to identify the minimum SAE PU during both zonal and raster searches. For a 8×8 PU, SAE between the current PU and the 20 neighboring blocks must be computed, which includes 12 neighboring PUs during the zonal search and 8 neighboring PUs during the raster search. Therefore, SAE and interpolation computations have to be performed for a total of twenty 8×8 PUs during the zonal and raster searches. The total number of integer operations performed for one 8×8 PU in zonal and raster searches are given in Table 4.4. Note that, in bi-prediction, the TZ search has to be performed twice, one per each reference frame. Therefore, the number of integer computations will be doubled as given in Table 4.4. As we are considering the simplest possible motion estimation configuration in an HEVC encoder, the integer computations in the refinement stage of the TZ search are not taken into account in this analysis.

4.4.1.3 Weighted prediction

Weighted prediction as given by (4.1) is applied to the two minimum SAE PUs from the reference frame 1 and reference frame 2, found in the TZ search. This produces the final prediction for the current PU. The pixel values in each minimum cost block are multiplied with the respective weights and the offsets are added as given in (4.1). Therefore, weighted prediction requires 64 multiplications and 64 additions per 8×8 minimum SAE PU block (each pixel value must be multiplied by a weight followed by the addition of an offset). The weighted pixels of the minimum SAE blocks from the two reference frames are then added, to produce the prediction for the current PU which requires another 64 addition operations. In total, weighted prediction results in 192 additions and 128 multiplications as given in Table 4.5. The weighted prediction process completes the computational steps required for the bi-prediction of a single 8×8 PU.

The total number of integer operations performed for a single PU in all available PU modes, at each step of the bi-prediction process are given in Table 4.5. The computational complexity of the motion estimation in HEVC standard highly depends on the quad-tree partitioning structure used in the model. Although we set the CTU size to 16×16, the CU and PU sizes can vary from one CTU to another according to the RD optimization. Therefore, we considered two different cases in order to analyze the computational complexity required to predict a single video frame.

Case 1: Least complexity

The least complexity case for this minimum-complexity encoder occurs when the CU size is selected as 16×16 and the maximum CU partitioning depth is set to 0

($MaxCuDepth = 0$). Consider, the PU mode selection process given in Fig. 4.9. In this case, the video frame is divided into CUs of size 16×16. There will be 4 PU modes available for selecting the best coding option. The modes are 16×16, 16×8, 8×16 and 8×8 as given in Table 4.1. Thus, the SAE for each of the four available PU modes have to be calculated to select the minimum SAE PU mode as given in the Fig. 4.9. In this thesis, we consider CIF resolution video, which has a frame size of 288×352 pixels. Therefore, if we consider the PU mode 16×16 alone, a single frame will have 396 PU blocks. Similarly, for 16×8, 8×16 and 8×8 PU modes there will be 792, 792 and 1584 PU blocks respectively. Therefore, the total number of integer computations required in this case is

$$T_{ic} = \sum_{i \,\varepsilon\, PUmodes} I_{PU}(i) \times N_{PU}(i), \tag{4.4}$$

where I_{PU} is the total number of integer operations required for the i^{th} PU mode and N_{PU} is the number of PU blocks in a video frame for the i^{th} PU mode.

We note that for the HEVC encoding configuration in Table 4.6, the minimum possible computational complexity is about 6×10^9 integer operations per CIF video frame.

Table 4.5: Integer operations for bi-prediction for all available PU modes.

PU size	8 x 8 PUs		8 x 4 or 4 x 8 PUs		8 x 16 or 16 x 8 PUs		16 x 16 PUs	
Process	Add	Multiply	Add	Multiply	Add	Multiply	Add	Multiply
Advanced Motion Vector Prediction								
1. Spatial candidates								
Half pel - candidates	5635	6440	2555	2920	11795	13480	24675	28200
Quarter pel - candidates	27195	31080	12075	13800	57435	65640	121275	138600
SAE calculation	11285	0	5125	0	23605	0	49365	0
2. Temporal candidates								
Half pel -candidates	2254	2576	1022	1168	4718	5392	9870	11280
Quarter pel - candidates	10878	12432	4830	5520	22974	26256	48510	55440
SAE calculation	4514	0	2050	0	9442	0	19746	0
Scaling	12	8	12	8	12	8	12	8
Merge Candidate List								
1. Spatial candidates								
Half pel- candidates	5635	6440	2555	2920	11795	13480	24675	28200
Quarter pel- candidates	27195	31080	12075	13800	57435	65640	121275	138600
SAE calculation	11285	0	5125	0	23605	0	49365	0
2. Temporal candidates								
Half pel -candidates	2254	2576	1022	1168	4718	5392	9870	11280
Quarter pel - candidates	10878	12432	4830	5520	22974	26256	48510	55440
SAE calculation	4514	0	2050	0	9442	0	19746	0
TZ search								
Half pel- candidates	45080	51520	30660	35040	141540	161760	78960	90240
Quarter pel- candidates	217560	248640	144900	165600	689220	787680	388080	443520
SAE calculation	90280	0	61500	0	94420	0	157968	0
Weight prediction	192	128	96	48	512	256	1024	512
Total	476398	405352	292482	247512	1185642	1171240	1172926	1001320
Total per PU size	881998		539994		2356882		2174246	

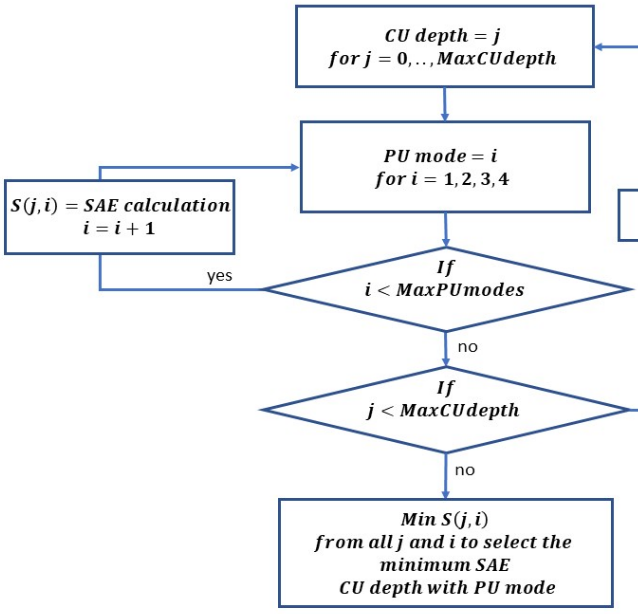


Fig. 4.9: CU partitioning and PU mode selection based on the minimum SAE.

Table 4.6: Integer operations required for bi-prediction of one frame in the least-complexity case for CIF resolution video.

Available PU sizes	Computations per PU	Number of PUs	Total
16×16	2174246	396	8.6×10^8
16×8	2356882	792	1.9×10^9
8×16	2356882	792	1.9×10^9
8×8	881998	1584	1.4×10^9
Total computations per frame			6×10^9

Case 2: Highest complexity

The highest complexity case for this minimum-complexity encoder occurs when CU partitioning is done by considering a maximum CU partitioning depth of 1. As given in Fig. 4.9, the SAE calculations are first performed for all 4 PU modes of CU

TABLE 4.7: Integer operations required for bi-prediction of one frame in the highest-complexity case for CIF resolution video.

Available PU sizes	Computations per PU	Number of PUs	Total
16×16	2174246	396	8.6×10^8
16×8	2356882	792	1.9×10^9
8×16	2356882	792	1.9×10^9
8×8	881998	1584	1.4×10^9
4×8	539994	3168	1.7×10^9
8×4	539994	3168	1.7×10^9
Total computations per frame			9.5×10^9

size 16×16 (at $CUdepth = 0$) and then the SAE calculations are performed for all available PU modes for CU size 8×8 (at $CUdepth = 1$). The available PU modes for each CU size are given in Table 4.1. For the video frames of size 288×352, considered in this thesis, the number of PU blocks per frame in considering different PU modes are given in Table 4.7. The number of computations per PU block has been multiplied by the number of PU blocks per frame (4.4), to calculate the total number of integer computations used in this case.

We note that for the HEVC encoding configuration in Table 4.7, the maximum possible computational complexity is about 9.5×10^9 integer operations per CIF video frame.

Chapter 5

A Study of a Low Complexity CNN Architecture for Video Frame Bi-Prediction

In video coding algorithms inter-frame prediction is performed in a block-wise manner. Typically size of these blocks are small compared to the frame size, as the translational motion assumption which underpins classical motion estimation can be poor for large block sizes. For example, the minimum block sizes used by the HEVC standard are 4×8 and 8×4. Motion of objects occurring in natural video sequences are a combination of translations, rotations and deformations. Furthermore, the moving object can be smaller than the block size used for motion estimation. In such cases, the block matching based motion estimation can result in poor prediction performance. Deep learning architectures have been used for bi-prediction tasks in recent works [51]. One of the most widely used approaches to deep learning based predictions is CNNs. In contrast to motion estimation based prediction, CNNs can be used for prediction at frame level. Furthermore, while traditional motion estimation used in most video coding algorithms such as the HEVC standard is linear, CNNs perform nonlinear prediction which can capture complex motion in natural video. This chapter presents an investigation of an efficient CNN architecture for video frame bi-prediction. The goal of the study presented in this chapter has been to identify the best CNN architecture for video frame bi-prediction, with both prediction accuracy as well as the computational complexity in mind. This study involves the evaluation and comparison of several candidate CNN architectures.

CNNs used for video frame bi-prediction in all previous work reported in the literature have used floating point arithmetic operations. In state-of-the-art video codecs, most of the operations, including motion estimation are implemented as integer operations to reduce the computational complexity to levels required for real-time video compression. Floating point CNNs may thus not be appropriate for such video coding applications. In fact any advantage gained by improved prediction may be offset by high computational complexity of floating point arithmetic. With minimal computational complexity in mind, the main objective of this thesis is to explore the possibility of using integer CNNs for frame bi-prediction in video compression. Section 5.3 presents a quantitative analysis for computational complexity of the CNN bi-prediction architecture used in this thesis. This analysis is used for the experimental study presented in Chapter 6, where the computational complexity of CNN bi-prediction is compared with that of the simplest motion estimation option available in the HEVC (HM 16.22) reference encoder as described in Chapter 4.

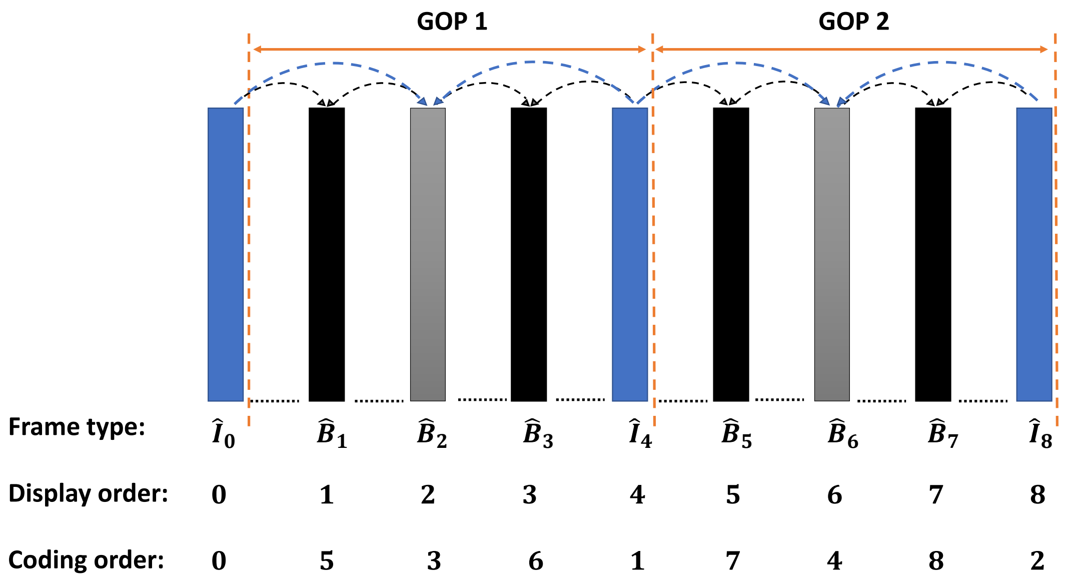


FIG. 5.1: Hierarchical bi-prediction at the decoder.

5.1 Hierarchical Bi-Prediction Using CNNs

In this chapter, we assume that the video encoder is operated on the group of pictures (GOP) structure as shown in Fig. 5.1, where the prediction process for two consecutive GOPs, GOP1 and GOP2 are illustrated. Without a loss of generality, we consider the first two GOPs in a video sequence. In this encoding scheme, the first frame of the GOP is encoded as a key-frame. In video coding, the key-frames are encoded as still images, without reference to any other frame. These are referred to as I-frames. All other frames in the GOP are predicted using two reference frames. These are referred to as bi-predicted frames (B-frames). The bi-prediction process for the first two GOPs in a video sequence is shown in Fig. 5.1. Every two consecutive GOPs are predicted and coded as shown here.

When a trained CNN is incorporated into a video encoder, it will be performing closed-loop prediction, as shown in Fig. 5.2. This means that the input frames

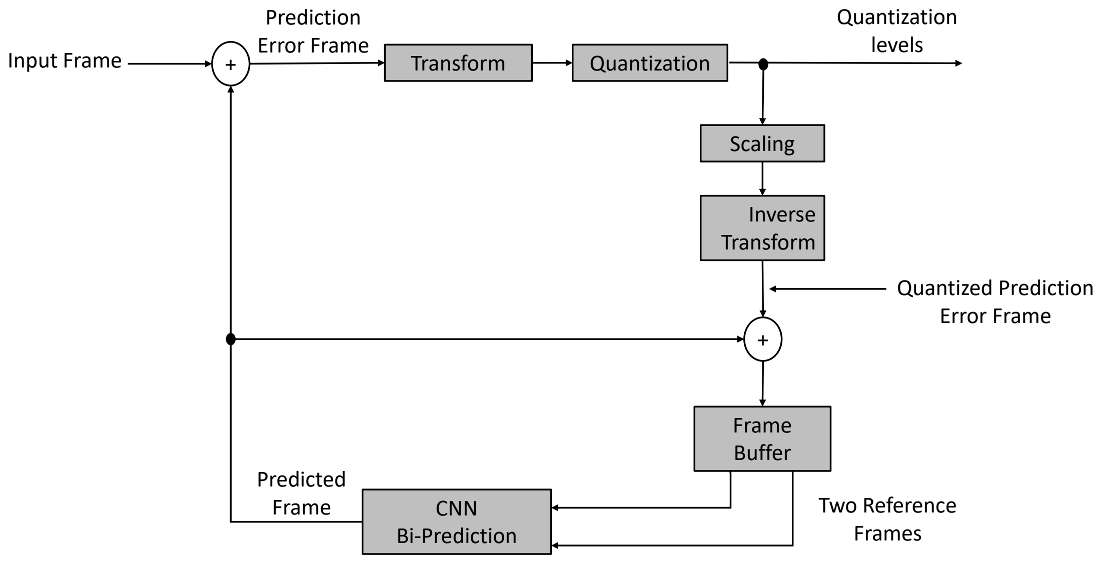


FIG. 5.2: Closed loop prediction.

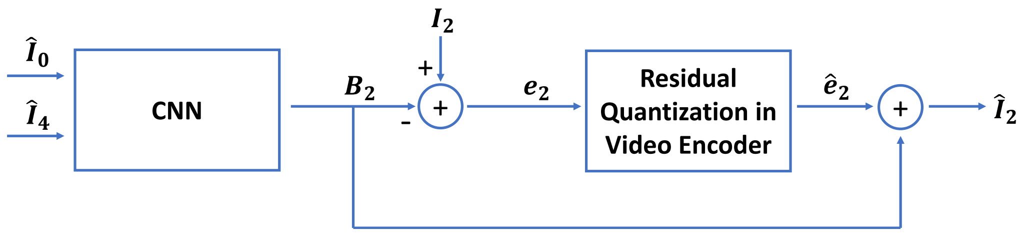


FIG. 5.3: Level 1 of bi-prediction architecture.

to the CNN will be the encoded versions of the original frames. Let us consider the closed-loop hierarchical bi-prediction process for GOP 1. Let I_0, I_1, I_2, I_3 and I_4 be the original frames of this GOP. In this case, the bi-prediction hierarchy consists of 2 levels as shown in Figs. 5.3 and 5.4. In these diagrams, for I_i where $i = 1, 2, 3, 4$: B_i is the bi-prediction, e_i is the prediction error and $\hat{e}_i$ is the quantized version of the prediction error. In level 1, the encoded versions of the input frames $\hat{I}_0$ and $\hat{I}_4$ are used as the inputs to bi-predict the middle frame I_2. In level 2, the reconstructed frame $\hat{I}_2$ from level 1 is used with $\hat{I}_0$ and $\hat{I}_4$ separately to bi-predict I_1 and I_3 respectively, as shown in Fig. 5.4.

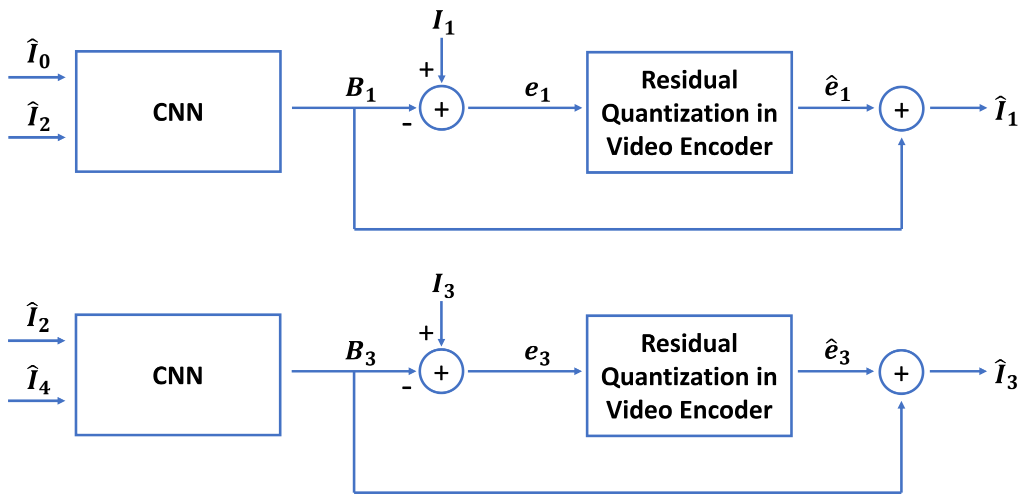


FIG. 5.4: Level 2 of bi-prediction architecture

Our description above applies to closed-loop prediction in a video encoder. Note that in closed-loop prediction, the predictor inputs are dependent on the predictor itself. Therefore, it is actually not possible to train the CNNs to minimize the closed-loop prediction error. However, if the bit-rate of the video encoder is not very low, we can expect the CNNs trained to minimize the open-loop prediction error which will also be good for closed-loop prediction. We therefore considered the open-loop prediction error during the CNN training stage for bi-prediction. As we only consider the compression of monochrome video sequences, in the following sections we will focus on the prediction of monochrome video using CNNs.

5.2 U-Net Architecture for Bi-Prediction

5.2.1 Overview of the Architecture

In the literature review presented in Chapter 2, we have discussed the different types of CNN architectures for video frame bi-prediction. We require a CNN architecture with low complexity and high quality of bi-prediction performance to fulfill the objective of this thesis. The CNN architecture used in this thesis is based on the U-Net architecture for video frame interpolation originally proposed in [11]. We selected this architecture for bi-prediction due to the following reasons as we discussed in Chapter 2. Bi-prediction can be performed at the video frame level and therefore, the quad-tree partitioning structure used in the HEVC encoder is not considered for bi-prediction. U-Net architecture has a limited number of layers and parameters and therefore the computational complexity and memory cost are low enough to be used in real-time video coding. Further, U-Net architecture has shown better results in prediction quality compared to many CNN architectures proposed in previous work. We further reduced the number of parameters in the U-net architecture proposed in [11] to reduce the complexity of the bi-prediction operation. More specifically, in this thesis, we use a reduced number of layers and filters in the U-Net architecture.

The U-Net architecture consists of two parts, namely the encoder and the decoder, as illustrated in Fig. 5.5. In the encoder side of the network, each processing unit contains both convolutional and max-pooling layers. The input feature map sizes are down-sampled in this side. In the decoder side, the feature maps are up-sampled to the size of the reference video frames. Each processing unit of the decoder side contains an up-convolutional layer followed by a convolutional layer.

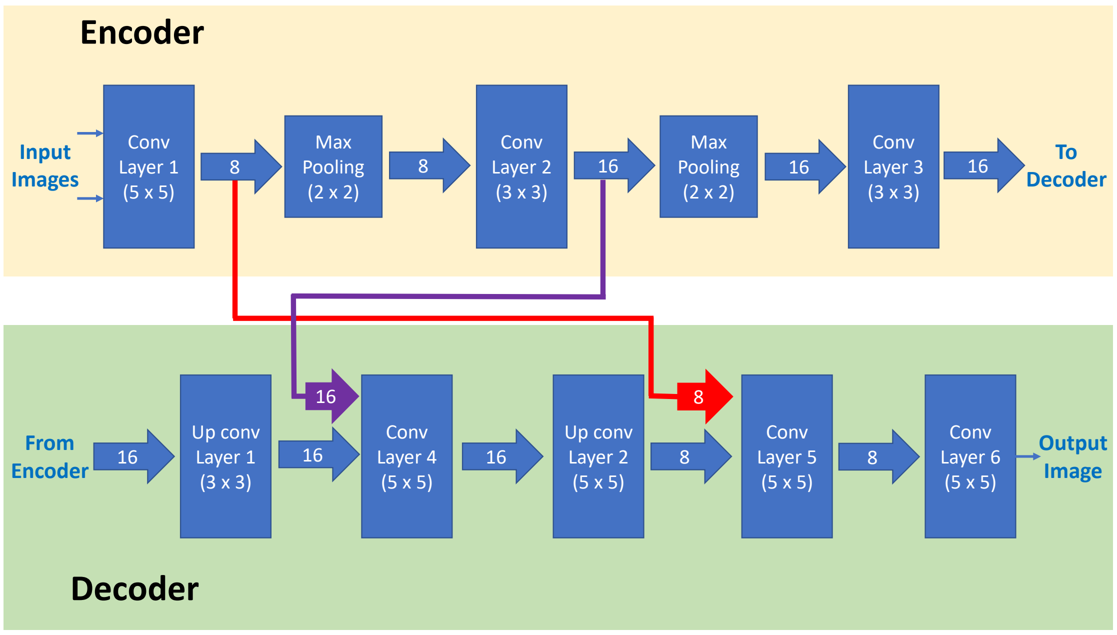


FIG. 5.5: Encoder and decoder in U-Net architecture.

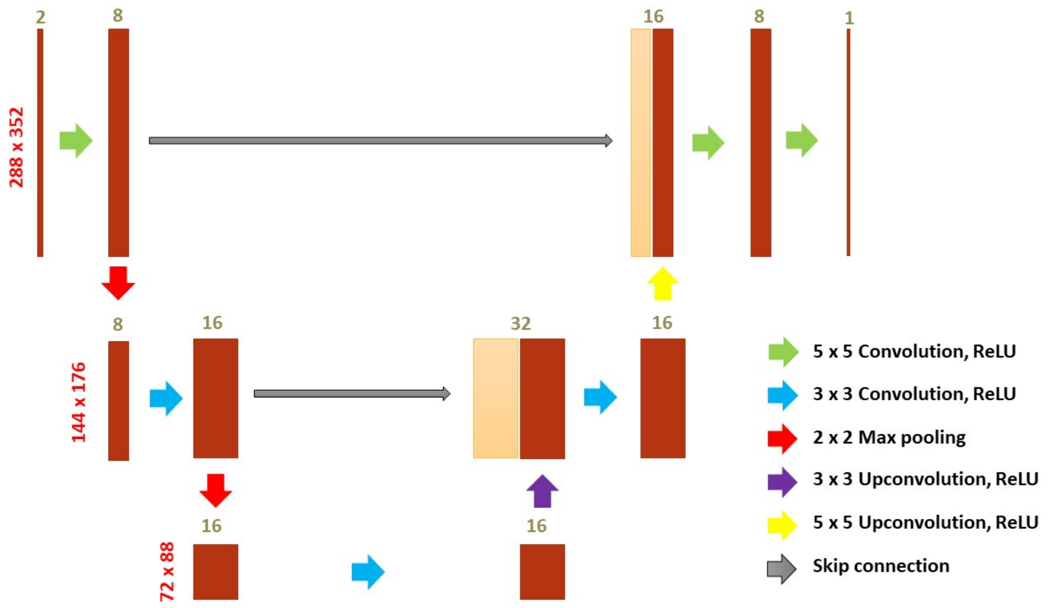


FIG. 5.6: U-Net CNN architecture.

As the network performs down-sampling of the input and up-sampling of the output (so that the reference frames and predicted frame are of the same size), the network parameters are independent of the video frame resolution. As shown in Fig. 5.6, the network consist of 6 convolutional layers, 2 up-convolutional layers and 2 max pooling layers. The number of channels at each of the layers are denoted above the feature maps of the respective layers in this figure. In section 5.2.2, we present experimental results demonstrating the complexity vs performance trade-off when the number of channels and layers are varied. These results show that it is possible to further reduce the complexity of the U-net architecture compared to the original architecture in [11], with only a small degradation in the prediction accuracy.

The convolution, up-convolution and max-pooling filter sizes that we have used is given in Fig. 5.6. This diagram also illustrates the down-sampling and up-sampling of the feature maps. Convolution filter dimensions in the 1st and 2nd levels of the encoder are 5×5 and 3×3 respectively. In using different filter dimensions, different scales of motion can be detected. Each convolution operation is followed by batch normalization, a rectified linear unit (ReLU) and a 2×2 max pooling operation with a stride length of 2, to reduce feature map dimensions as given in Fig. 5.6. This down-sampling encoder architecture can extract the global (low resolution) features of a large receptive field of the input frames as the information propagates through the layers towards the bottleneck between the encoder and the decoder. If there is a considerable movement of an object that occurs between the two input frames, the bottleneck layer will be able to capture a feature related to this motion [11]. In the bottleneck layer the convolution operation has an effective receptive field of 72×88 pixels from the original input frames. In the decoder section, an up-convolutional layer is introduced after every convolution operation. Up-convolutional layer performs up-sampling at each level, to finally generate a prediction with same resolution as the input video frames. Both up-convolution and convolution kernels are of the same dimensions, 3×3

and 5×5 respectively. ReLUs have been used as the activation functions in all the convolutional and up-convolutional layers except for the last convolutional layer, where the tanh non-linearity has been used. The tanh non-linearity will restrict the pixel value of output to $(-1, +1)$. This network output is converted to the standard $16-235$ range of the 8-bit pixel depth of the luma component to generate the final predicted frame.

To better maintain spatial information flow through the network, skip connections are added between the corresponding convolutional and up-convolutional layers. Specifically, the corresponding up-convolutional layers and convolutional layers are concatenated together before being fed forward. The convolution that is followed by skip connection and up-convolution will warp the low resolution features to high resolutions. In addition, skip connections can assure that gradients will flow backward, which mitigates the vanishing gradient problem when training the network [66].

5.2.2 Training and Validation

In this thesis CNN models have been trained and implemented using the Pytorch platform [67]. Publicly available UCF101 database [68] was selected to train the CNN model.

- *Data set:* The UCF101 database is mainly used for action recognition tasks and has 13320 video clips with resolution 320×240. The video clips have been recorded at 25 frames per second and are available in RGB format. UCF101 database has slow to fast motion video sequences of a large variety of action types. Therefore, training with this database can generalize the network to tackle complex motions in video sequences. OpenCV and Pytorch software were used to preprocess the original video files and to form the training data set, including applying transforms for normalization and random flipping [69].

- *Training loss functions:* The choice of the loss function for training a CNN in Pytorch generally defaults to the mean squared error (MSE) loss. This is due to the many desirable properties of this loss function. But MSE has its limitations. MSE correlates poorly with the image quality as perceived by a human observer [70]. In designing U-Net our objective is to obtain high performance in video frame reconstruction and therefore, we studied the effect of different loss functions for better network performance. The U-Net architecture has been trained using three different loss functions including the MSE and their impact on the overall bi-prediction performance have been evaluated. The loss functions considered are

  - MSE (or $l2$ loss),
  - $l1$ loss,
  - Structural similarity index measure (SSIM).

  These loss functions are widely used in training CNNs for image processing applications [70]. Importantly, they are differentiable ($l1$ loss is differentiable everywhere except at the origin), which is a requirement for the propagation of the training algorithm to work.

- *Best choice of loss function:* The overall performance obtained by the network under different combinations of training loss functions and performance metrics are given in Table 5.1. When the network was trained using $l1$ loss instead of the MSE loss, the quality of the output image has been better according to both PSNR and SSIM metrics, for the validation data set. The

Table 5.1: Performance of the proposed CNN architecture with different loss functions.

Loss function		MSE	$l1$ loss	SSIM
PSNR (dB)	Training	22.3	22.5	22.1
	Validation	22.1	22.2	22
SSIM	Training	0.78	0.79	0.78
	Validation	0.77	0.78	0.77

network trained with SSIM as the loss function, has performed either at par or slightly worse than the network trained with $l1$ loss, for the validation data set. These results indicate that the $l1$ loss outperforms the other loss functions for this U-Net CNN architecture. The better performance of $l1$ loss function compared to other loss functions is also discussed in [70]. Thus, $l1$ loss has been adopted as the loss function for training, in the rest of this thesis.

- *Performance criteria for evaluation:* PSNR and SSIM metrics have been used to evaluate the performance of the trained CNN. These metrics are widely used in image processing tasks as discussed in Chapter 3. As these two metrics evaluate the image quality in two different aspects, both of these metrics were used to evaluate the CNN network output.

- *Effect of hyper-parameters:* The results of the Adam optimizer are generally better than every other optimization algorithm. This optimization has faster computation time, and require fewer parameters for tuning. Therefore our network has been trained using the Adam optimization criterion [71] for different sets of hyper-parameter selections. These are learning rate α, the betas β_1, β_2 of the Adam algorithm and the weight decay parameter λ. A mini-batch size of 5 has been used and the CNN model was trained for 100

TABLE 5.2: Best training and validation performance observed for U-Net architecture with different combinations of hyper-parameters.

Network		Network f1	Network f2	Network f3
Epochs		100	100	100
Learning Rate (α)		0.001	0.002	0.001
Betas (β_1, β_2)		0.85, 0.997	0.95, 0.995	0.95, 0.995
Weight decay (λ)		0.001	0.0008	0.001
Training	Loss	0.094	0.098	0.088
	PSNR	24.8	24.4	24.9
	SSIM	0.76	0.72	0.76
Validation	Loss	0.072	0.079	0.07
	PSNR	22.8	22.6	23.4
	SSIM	0.67	0.67	0.68

epochs. A fixed set of 8% samples from the dataset has been used to form the validation set while the remaining samples are used as the training set. The network was trained by initializing weights using Xaviar initialization [72] and data augmentation was enabled. The training and validation results for the 3 best performing networks with different hyper-parameter values are given in Table 5.2. The highest PSNR and SSIM values for the validation set have been observed with Network f3. Further, Network f3 was also trained for 500 epochs to analyze any performance improvement with a longer training time, but no improvements were observed.

- *Impact of network complexity:* In order to investigate the potential performance improvements in increasing the network complexity, U-Net structures with more layers and channels were trained and tested. The same hyper-parameter values used with the Network f3 were used here as well. The evaluated architectures are as follows.

– *High channel U-Net architecture:* A two level U-Net architecture with the same filter kernel sizes as Network f3 is used here. The number of channels were increased to 16 and 32 in the 2 levels respectively. The number of addition and multiplication operations in this architecture was about 12 times higher than in Network f3, but validation performance was increased only by 0.01 dB in PSNR. As a PSNR gain below 0.5 dB is visually insignificant, we note that this more complex architecture does not yield any useful performance improvement.

– *3 level U-Net architecture:* One more level was added to the Network f3 architecture. The filter kernel sizes used in the 3 levels of the encoder side are 5×5, 5×5 and 3×3 respectively and vice versa in the decoder side. The number of channels used in the 3 levels of the encoder are 16, 32, and 64. The number of addition and multiplication operations in this architecture was about 8 times higher than in the Network f3, but the validation performance was not as good. CNN models tend to overfit the training data set of fixed size, as the network depth increases. This overfitting can be the reason for performance degradation in this case.

The goal of this study is to minimize the computational complexity of the CNN architecture with a minimal sacrifice in prediction performance. Therefore, Network f3 has been selected as the best low complexity floating point CNN architecture for video frame bi-prediction.

5.3 Computational Complexity of U-Net Architecture

In this section, the computational complexity of U-Net architecture shown in Fig. 5.6 is established, in terms of the number of addition and multiplication operations. The objective is to compare the complexities of CNN based bi-prediction and the simplest motion estimation algorithm option in HM 16.22 encoder described in Chapter 4.

5.3.1 Convolutional layer complexity

Convolutional layers perform two-dimentional (2D) convolution between a given input matrix and a filter kernel. Consider an input matrix of size $H \times W$ introduced to a convolutional layer with filter kernels of size $H_k \times W_k$ as given in Fig. 5.7. The input can have several channels. As an example, in our CNN architecture we input 2 video frames of size 288×352 concatenated together as a 2 channel input, as given in Fig. 5.7(A). Further, a convolutional layer can have multiple filters, but all these filters have the same dimensions $H_k \times W_k$. To perform convolution, the filter moves across the input matrix. This is a 2D shift operation in both x and y directions and the filter kernel is shifted in stride length S. The stride length is the number of pixels shifted by the filter kernel over the input matrix. During this shift operation, the dot product between the overlapped part of the input matrix and the filter kernel (F) is performed at every shift position. The result of this dot product computes the features of the output feature map of the convolutional layer. Consider the feature O_1 in the output of the convolutional layer given in Fig. 5.7(C). The convolution operation between the input matrix and the filter

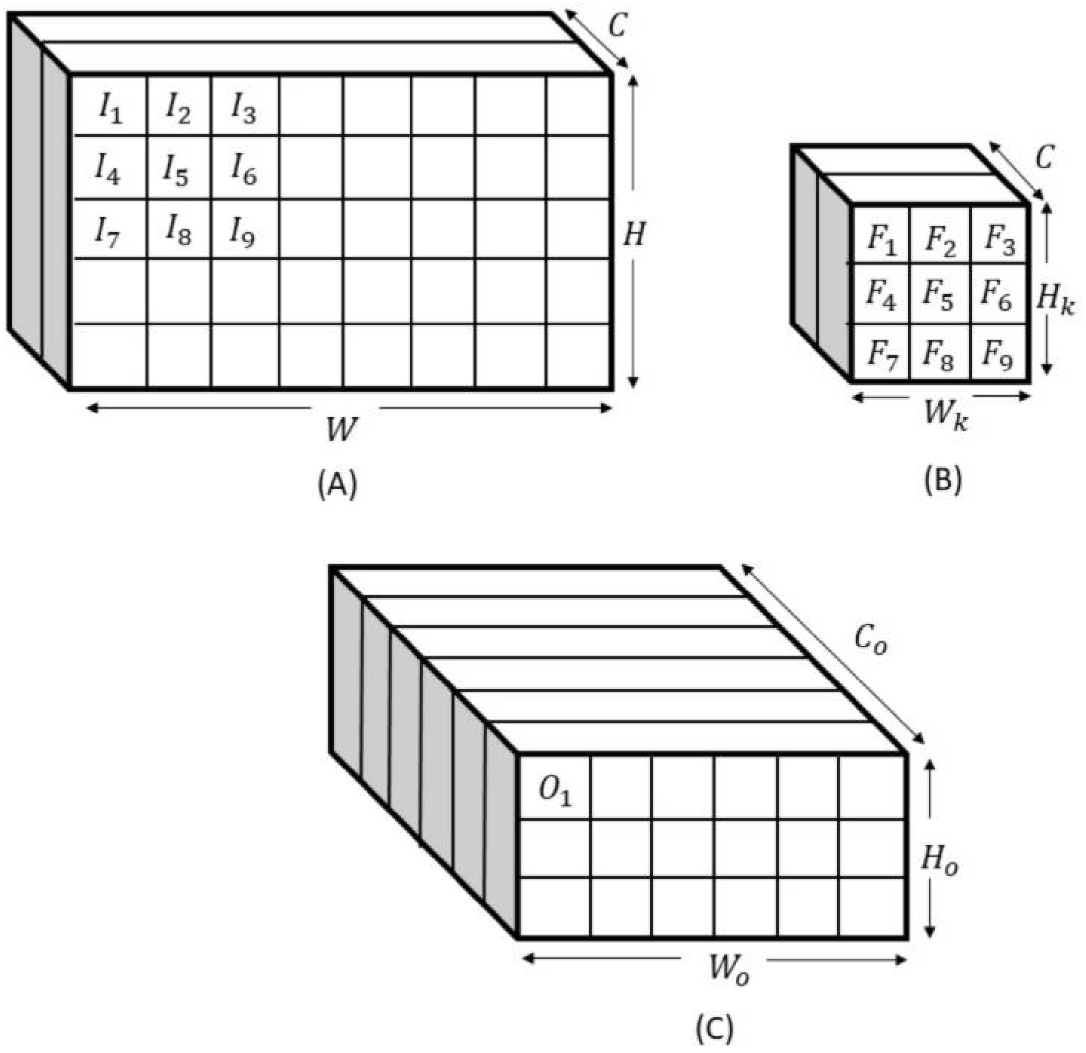


FIG. 5.7: Convolution operation: (A) Input to the convolutional layer, (B) Filter kernel, (C) Output of the convolutional layer.

kernel is given by

$$O_1 = \sum_{n=1}^{C} I(n) \, . \, F(n) \, , \tag{5.1}$$

where C is the number of channels in the input, $I(n)$ and $F(n)$ are the overlapped part of the input matrix and filter kernel in the channel n respectively, as illustrated in Fig. 5.7. If we assume that the same filter is used on all channels, the number of multiplications (n_{M,O_1}) and additions (n_{A,O_1}) required to compute one feature (O_1) in the output are given by

$$\begin{aligned} n_{M,O_1} &= (H_k \times W_k) \times C \, , \\ n_{A,O_1} &= ((H_k \times W_k) \times C) - 1 \, , \end{aligned} \tag{5.2}$$

where $(H_k \times W_k)$ is the number of parameters in the filter kernel. Further, the height and width of the output feature map of the convolutional layer are respectively given by

$$\begin{aligned} H_o &= \frac{H - H_k + 2P}{S} + 1 \, , \\ W_o &= \frac{W - W_k + 2P}{S} + 1 \, , \end{aligned} \tag{5.3}$$

where P is the zero-padding added to the input. Zero padding denotes the number of additional zero pixels added to the input feature map to increase the dimensions of the input matrix. P has been multiplied by 2 to add these zero pixels in both sides of the x and y dimensions, of the input matrix. Every filter in the convolutional layer will generate an output of dimensions $H_o \times W_o$ individually. Suppose the number of filters in the convolutional layer is C_o. Then the total number of features in the layer output is $H_o \times W_o \times C_o$. Therefore, the total number of multiplications ($n_{M,conv}$) and additions ($n_{A,conv}$) performed in the convolutional layer to generate the output are given by

$$\begin{aligned} n_{M,conv} &= H_k \times W_k \times C \times H_o \times W_o \times C_o \, , \\ n_{A,conv} &= ((H_k \times W_k \times C) - 1) \times H_o \times W_o \times C_o \, . \end{aligned} \tag{5.4}$$

As an example, consider the convolutional layer 1 of the U-Net architecture shown in Fig. 5.5. The layer parameters are as follows. Input height (H) and width (W)

are 288 and 352 respectively. Two channels are present in the input (two adjacent frames) and therefore $C = 2$. The height (H_k) and width (W_k) of the filter kernal are 5 and 5 respectively. There are 8 such filters present in the convolutional layer, so $C_o = 8$. The stride length and padding in this layer are $S = 1$ and $P = 2$ respectively. Thus from (5.3), we derive that $H_o = 288$ and $W_o = 352$ and from (5.4) we derive that $n_{M,conv} = 40550400$ and $n_{A,conv} = 39739392$.

In the U-Net architecture there are six convolutional layers as illustrated in Fig. 5.5. i.e. convolutional layer 1, convolutional layer 2, convolutional layer 3, convolutional layer 4, convolutional layer 5 and convolutional layer 6. Using (5.4) we can determine the number of addition and multiplication operations required in each layer, which are summarized in Table 5.3.

5.3.2 Upconvolutional layer complexity

Upconvolutional or transpose-convolutional layer is used to up-sample the input feature map using a filter kernal. Consider an input of size $H_i \times W_i$ where H_i is the height and W_i is the width. If this input is given to a upconvolutional layer with filter kernels of dimensions $H_k \times W_k$, then for every feature in the input feature map, $H_k \times W_k$ number of output values are computed. This process can be explained as follows. The n^{th} output generated by the feature I_1 in the input feature map is

$$O_{1,n} = I_1 \times F_n, \tag{5.5}$$

where F_n is the n^{th} parameter in the filter kernel. Therefore, for a single feature I_1 in the input feature map, there will be $H_k \times W_k$ output features computed. The multiplication operation in (5.5) is performed for all features in the input feature

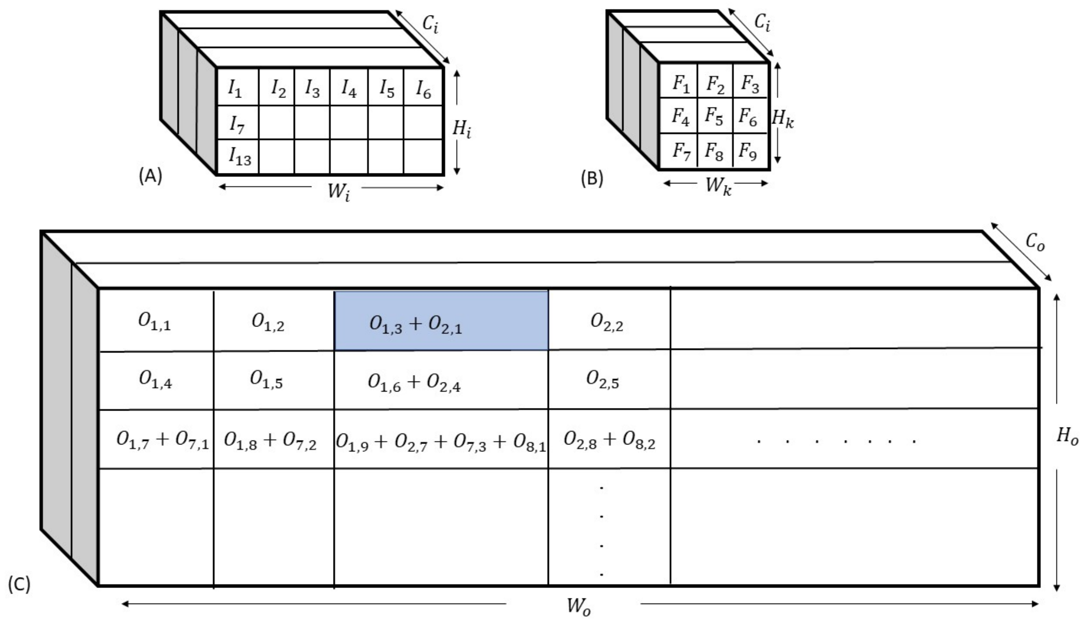

(A)
C_i
I_1 I_2 I_3 I_4 I_5 I_6
I_7
I_{13}
H_i
W_i
(B)
F_1 F_2 F_3
F_4 F_5 F_6
F_7 F_8 F_9
H_k
W_k
(C)
C_o
$O_{1,1}$ $O_{1,2}$ $O_{1,3}+O_{2,1}$ $O_{2,2}$
$O_{1,4}$ $O_{1,5}$ $O_{1,6}+O_{2,4}$ $O_{2,5}$
$O_{1,7}+O_{7,1}$ $O_{1,8}+O_{7,2}$ $O_{1,9}+O_{2,7}+O_{7,3}+O_{8,1}$ $O_{2,8}+O_{8,2}$
H_o
W_o

Fig. 5.8: Upconvolution operation: (A) Input to the upconvolutional layer, (B) Filter kernal, (C) Output of the upconvolutional layer.

map. Therefore, the total number of multiplication operations performed in the upconvolution operation is given by

$$n_{M,upconv} = (H_i \times W_i \times C_i) \times (H_k \times W_k) \times C_o, \tag{5.6}$$

where $H_i \times W_i \times C_i$ is the total number of features in the input feature map, and C_o is the number of filters in the upconvolutional layer. Consider the case of $H_k = 3$ and $W_k = 3$ as given in Fig. 5.8. In this case, a total of 9 values will be derived from the input feature I_1 which are denoted by $O_{1,1}$ to $O_{1,9}$ in Fig. 5.8. Similarly, for the I_2 feature, 9 values $O_{2,1}$ to $O_{2,9}$ are computed as shown in Fig. 5.8. In this case the stride length is $S = 2$.

Consider the shaded feature in the output feature map in Fig. 5.8(C). From the multiplication between I_1 and F_3 the output value $O_{1,3}$ is derived for this feature. Then, from the multiplication between I_2 and F_1 another output value $O_{2,1}$ is derived for the same feature. Then, these two output values are added to derive the shaded feature in Fig. 5.8(C). Therefore, the number of addition operations performed to derive features of the output can vary from one feature to another as given in figure Fig. 5.8(C). Therefore, the total number of addition operations in the upconvolutional layer is derived by calculating the sum of the number of addition operations per every feature in the output feature map. The size of the output feature map ($H_o \times W_o$) generated by an upconvolution layer is given by,

$$\begin{aligned} H_o &= ((H_i - 1) \times S) + H_k - 2P + P_o, \\ W_o &= ((W_i - 1) \times S) + W_k - 2P + P_o, \end{aligned} \tag{5.7}$$

where S is the stride length, P is the zero padding added to the input and P_o is the zero padding added to the output to obtain the required dimensions for the layer output.

As an example, consider the upconvolutional layer 2 in the U-Net architecture in Fig. 5.6. The height (H_i) and width (W_i) of the input feature map are 144 and 176 respectively. There are 16 channels (C_i) in the input. This layer consists of 8 filter kernels (C_o) with height (H_k) and width (W_k) equal to 5. Then, according to (5.6) the total number of multiplications ($n_{M,upconv2}$) performed in upconvolutional layer 2 is 81100800. The stride length (S), padding (P) and output padding (P_o) in this layer are 2, 2, 1 respectively. Thus, the size of the output feature map generated in this layer is 288×352 according to (5.7). For each pixel position of this output, the number of additions performed vary as indicated in Fig. 5.8. Thus, in upconvolutional layer 2 it has been observed that 1, 2, 3, 4, 5 and 8 additions are possible for different output feature derivations. This is illustrated in Fig. 5.9. This is a 9×11 size output derived from an upconvolutional layer with the same stride length and filter kernel size as the upconvolutional layer 2. For blue features in Fig. 5.9, only a single value is computed using (5.5). For white features, 9 values are computed by (5.5) using different combinations of the input features and the filter kernel parameters. Therefore, 8 addition operations are performed to derive the value of the features indicated in white in the output. Taking the above result into account, we selected the highest number of possible additions per output feature in each upconvolutional layer. For the upconvolutional layer 2, it is 8 additions per output feature. This is used as the number of additions ($n_{A,feature}$) to derive one output feature in the upconvolutional layer 2. Therefore, the total number of addition operations performed to derive the output in the upconvolutional layer 2 is given by

$$\begin{aligned} n_{A,upconv} &= n_{A,feature} \times Number\ of\ output\ features, \\ &= n_{A,feature} \times (H_o \times W_o \times C_o). \end{aligned} \tag{5.8}$$

Thus, using this equation the number of additions for upconvolution layer 2 is computed as 6488064 additions.

In the U-Net CNN architecture there are two upconvolutional layers as illustrated in Fig. 5.6: upconvolutional layer 1 and upconvolutional layer 2. According to (5.6), the integer multiplication operations performed at each upconvolutional layer are given in Table 5.3. The number of addition operations for every upconvolutional layer shown in this table has been computed by considering the stride length and filter kernel dimensions specified for the upconvolution operations.

5.3.3 Other operations in U-Net CNN architecture

The other operations performed in the U-Net architecture are max pooling, activations (ReLU or tanh), and concatenation. Max pooling is a logical operation. In

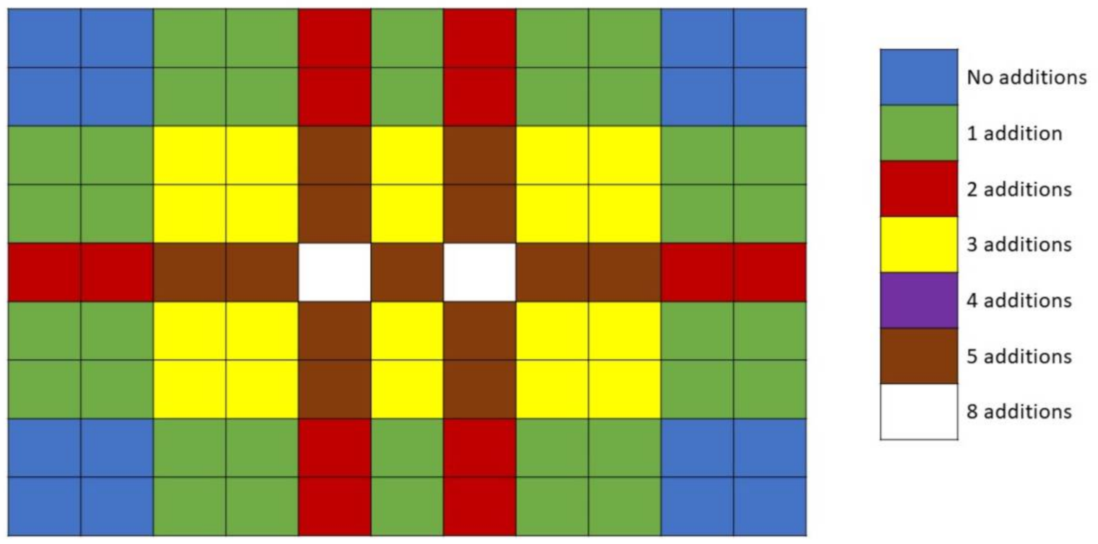


Fig. 5.9: Number of additions per output feature for a 9×11 upconvolution output.

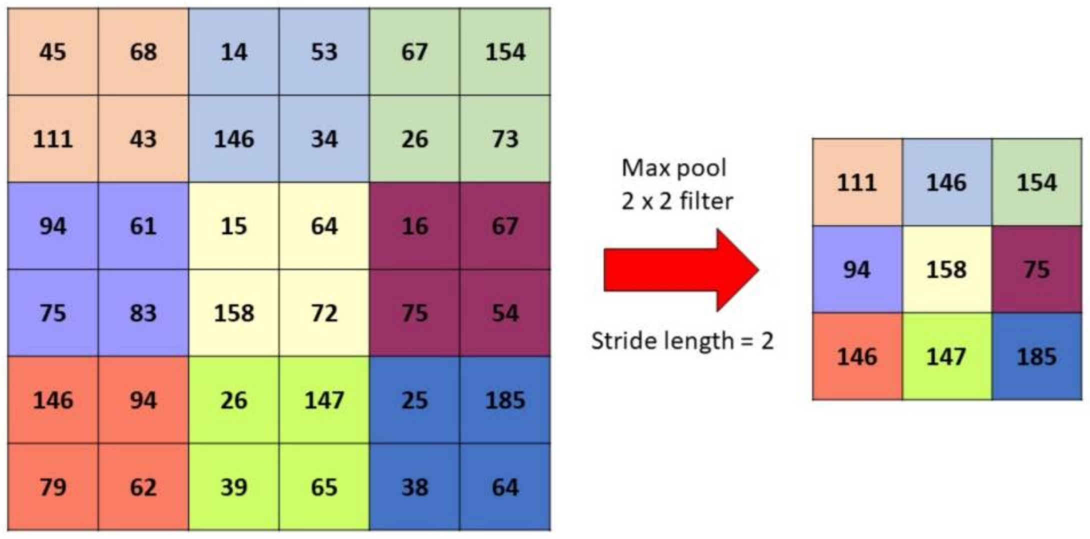


Fig. 5.10: Max pooling operation by a filter kernel of size 2×2 with stride length 2.

this architecture, max pooling is used to replace a 2×2 block of pixel values with the maximum pixel value as illustrated in Fig. 5.10. Thus, no arithmetic computations are required in this operation. But max pooling is a form of sub-sampling, and it is used to reduce the feature map size in the encoder section of the CNN architecture. As 2×2 filter kernels with stride length 2 are used for max pooling operation throughout the U-Net, the height and width of the input feature maps are reduced to half of the dimensions at the output as shown in Fig. 5.10.

CNN activation functions are mapping operations. Therefore, in ReLU and tanh activations, additions or multiplications are not performed. Concatenation operation is used in the decoder section of the U-Net CNN architecture. This operation is used to join the output of the upconvolutional layers (in the decoder section of U-Net) end-to-end channel wise with the convolutional layer outputs (in the encoder section of U-Net) channeled through skip connections as illustrated in Fig. 5.6. No computations are performed in the concatenation operation. The number of output channels in the convolutional layers of the encoder side is set equal to the number of output channels in parallel upconvolutional layers of the decoder side ($C_{conv} = C_{upconv}$) in the U-Net architecture. Therefore, the number of channels is doubled after the concatenation operation as the convolution and upconvolution outputs with same dimensions ($H \times W \times C$) are fused channel wise in concatenation operation. This is illustrated in Fig. 5.11.

Considering all the above mentioned operations performed in the U-Net architecture, the total number of arithmetic operations performed to bi-predict one video frame are given in Table 5.3. In the max pooling layer 1 and the max pooling layer 2, the height and width of the feature maps are reduced to half of the input dimensions, but the number of channels remains the same. In the concatenation layer 1 and the concatenation layer 2, the number of channels are doubled, but the height and width of the input feature maps remain the same. The number of additions and multiplications performed in convolution and upconvolution

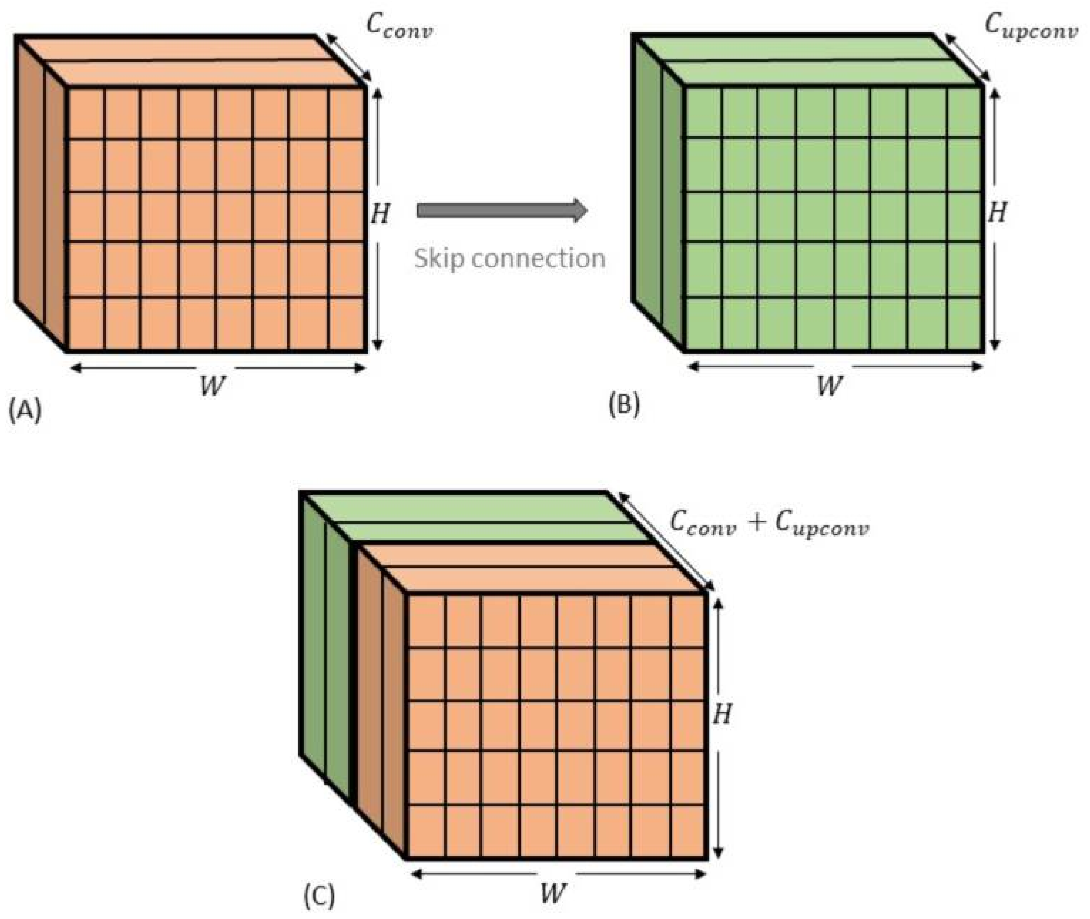


FIG. 5.11: Concatenation operation: (A) Convolutional layer output feature map, (B) Upconvolutional layer output feature map, (C) Concatenated feature map.

operations are computed as explained in section 5.3.1 and section 5.3.2 respectively. Therefore, in order to bi-predict one video frame using the U-Net CNN architecture, about 1.2×10^9 additions and multiplications are required.

TABLE 5.3: Integer computations performed in U-Net CNN to bi-predict one intermediate frame using the two adjacent frames.

Layer	H_i	W_i	H_k	W_k	S	P_o	P	C_i	C_o	H_o	W_o	Multiplications	Additions
Convolution - layer 1	288	352	5	5	1		2	2	8	288	352	40550400	39739392
Max pooling - 1	288	352	2	2	2			8	8	144	176	0	0
Convolution - layer 2	144	176	3	3	1		1	8	16	144	176	29196288	28790784
Max pooling - 2	144	176	2	2	2			16	16	72	88	0	0
Convolution - bottleneck layer	72	88	3	3	1		1	16	16	72	88	14598144	14496768
Upconvolution - layer 1	72	88	3	3	2	1	1	16	16	144	176	14598144	3244032
Concatenation - 1	144	176						16	32	144	176	0	0
Convolution - layer 3	144	176	3	3	1		1	32	16	144	176	116785152	116379648
Upconvolution - layer 2	144	176	5	5	2	1	2	16	8	288	352	81100800	6488064
Concatenation - 2	288	352						8	16	288	352	0	0
Convolution - layer 4	288	352	5	5	1		2	16	8	288	352	324403200	323592192
Convolution - layer 5	288	352	5	5	1		2	8	1	288	352	20275200	20173824
Total computations												641507328	552904704
												1194412032	

5.4 U-Net CNN Architecture with Integer Arithmetic Only Operations

For real-time video coding, low computational complexity of the video encoder is a critical requirement. As such state-of-the-art video codecs rely on integer arithmetic for all time consuming operations such as motion estimation. From this perspective, floating-point CNNs may not be competitive with efficient integer only motion estimation algorithms. To this end, in this section we investigate the use of integer only CNNs for video frame bi-prediction. There is also another motivation for using integer operations in the implementation of a video encoder. Floating-point operations tend to produce hardware dependent results whereas integer operations typically do not. Thus an integer only video encoder will perform consistently across all hardware platforms.

5.4.1 CNN quantization in Pytorch

CNN quantization refers to the representation of weights, bias values and activation outputs using finite-precision integer values in a CNN. In this case, all additions and multiplications are carried out on integer variables resulting in an increase in the speed of execution. However, this results in an inevitable loss of accuracy and hence a loss of prediction accuracy compared to a CNN using floating-point representations.

The Pytorch platform used in our study implements CNN models using high precision 32-bit floating point arithmetic (FP32). However, Pytorch does support 8-bit arithmetic (INT8) as well [73]. In general, there are three approaches to generating an integer precision CNN model in Pytorch.

1. *Dynamic quantization:* This is the simplest form of quantization where the weights are quantized ahead of time but the activations are dynamically quantized during the operation of the CNN, referred to as the *inference* stage. This is used for situations where the model execution time is dominated by loading weights from memory rather than computing the matrix multiplications.

2. *Post training static quantization:* This is the most commonly used form of quantization where the weights are quantized ahead of time and the scale factor and bias for the activation tensors are pre-computed based on observing the behavior of the model during a calibration process. Static quantization is mainly used when both memory bandwidth and savings in computations are significant in a CNN.

3. *Quantization-aware training:* During training, all calculations are done in floating point, with fake quantization modules used to model the effects of quantization by clamping and rounding to simulate the effects of INT8. After model conversion, weights and activations are quantized. It is commonly used with CNNs and yields higher accuracy compared to other methods.

In this thesis we use the simpler approach of static quantization to convert the previously trained and validated floating point CNN model to integer arithmetic.

Let q be a finite-precision unsigned binary representation of the real-value r. Then, the quantization operation can be described by the affine mapping given in Fig. 5.12, where s is a scaling factor that maps the range of the binary representation to that of the real representation, and Z is an off-set representing the value zero in the binary representation. This ensure that the real value zero is quantized to the binary representation 0, which allows efficient implementation of the zero-padding operation. The off-set Z has the same finite-precision resolution

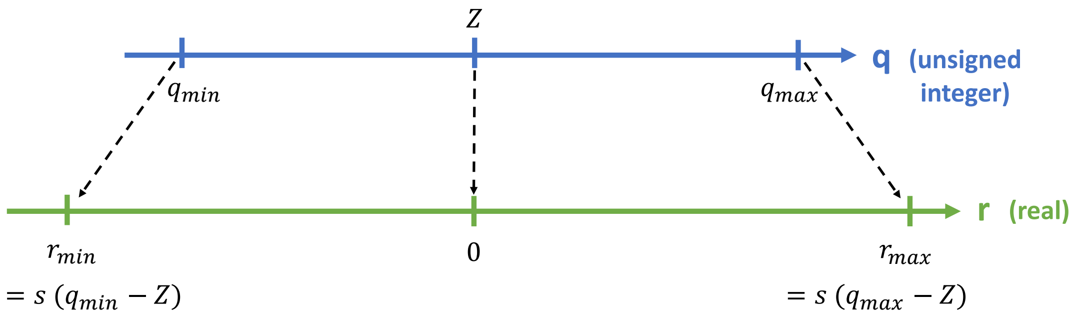


Fig. 5.12: Quantization mapping.

as q. On the other hand, the scale factor s has to be represented as a floating point value to prevent the severe degradation of the accuracy of computations. However, as described in section 5.4.1.1 it is possible to implement arithmetic operations without using such floating point representations.

In our research, we will consider 8-bit binary interpretations of real parameters for a high computational efficiency. However quantizing some bias variables to 32-bit accuracy is required for acceptable performance, see section 5.4.1.2.

5.4.1.1 Integer-arithmetic-only matrix multiplication

Consider the multiplication of two square $N \times N$ matrices of real numbers r_1 and r_2, to compute $r_3 = r_1 r_2$. We denote the entries of each of these matices r_α where $\alpha = 1, 2, \text{or } 3$ as $r_\alpha^{(i,j)}$ for $1 \leq i, j \leq N$, and the quantization parameters with which they are quantized as S_α and Z_α. The quantized entries are denoted by $q_\alpha^{(i,j)}$. Then according to quantization mapping

$$r_\alpha^{(i,j)} = S_\alpha(q_\alpha^{(i,j)} - Z_\alpha),$$

and using this, the definition of matrix multiplication can be written as

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1)\, S_2(q_2^{(j,k)} - Z_2),$$

which can be rewritten as

$$q_3^{(i,k)} = Z_3 + M \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2), \tag{5.9}$$

where the multiplier M is defined as

$$M = \frac{S_1 S_2}{S_3}.$$

In (5.9) the only non-integer is the multiplier M. As a constant that depends only on the quantization scales S_1, S_2, and S_3, it can be computed offline. It has been empirically found that M always lies in the interval $(0, 1)$ [74], and can therefore be expressed in the normalized form

$$M = 2^{-n} M_0,$$

where M_0 is in the interval $[0.5, 1)$ and n is a non-negative integer. The normalized multiplier M_0 now lends itself well to being expressed as a fixed point multiplier. For example, if 32-bit integers (INT32) are used (depends on the hardware capability), the integer representing M_0 is the INT32 value nearest to $2^{31} M_0$. Since $M_0 \geq 0.5$, this value is always at least 2^{30} and will therefore always have at least 30 bits of relative accuracy. Multiplication by M_0 can thus be implemented as a fixed point multiplication in Pytorch software. Multiplication by 2^{-n} can be implemented with an efficient bit shift, but accurate round-to-nearest behavior can be expected from this operation.

5.4.1.2 Implementation of a fused layer

Following [74], in this section we explicitly define the data types of all quantities involved and modify the quantized matrix multiplication in (5.9) to merge the bias addition and activation function evaluation directly into it. Consider q_1 matrix to be the weights and the q_2 matrix to be the activations. Both the weights and activations are 8-bit unsigned integers (UINT8), with modified zero-points we could have similarly chosen INT8. Accumulating products of UINT8 values requires a 32-bit accumulator, and we choose a signed type for the accumulator. The sum in (5.9) is thus of the form

$$\text{INT32} = \text{INT32} + \text{UINT8} \times \text{UINT8}.$$

In order to have the quantized bias-addition be the addition of an INT32 bias into this INT32 accumulator, the bias-vector is quantized such that:

- INT32 is used as the quantized data type.
- 0 is used as the zero-point Z_{bias}.
- Scale S_{bias} is of the same data type as that of the accumulators, which is the product of the scales of the weights and input activations.

In the notation of section 5.4.1.1,

$$\begin{aligned} S_{bias} &= S_1 S_2, \\ Z_{bias} &= 0. \end{aligned}$$

Although the bias-vectors are quantized as 32-bit values, they account for only a tiny fraction of the parameters in a CNN. Furthermore, the use of higher precision for bias vectors meets a real need: as each bias-vector entry is added to many output activations, any quantization error in the bias-vector tends to act as an overall bias (i.e. an error term with non zero mean), which must be avoided in order to preserve good end-to-end CNN accuracy.

With the final value of the INT32 accumulator, there remain three things left to do: scale down to the final scale used by the 8-bit output activations, cast down to UINT8 and apply the activation function to yield the final 8-bit output activation. The down-scaling corresponds to multiplication by the multiplier M in (5.9). As explained in section 5.4.1.1, it is implemented as a fixed-point multiplication by a normalized multiplier M_0 and a rounding bit-shift. Afterwards, a saturating cast to UINT8, saturating to the range $[0, 255]$ is performed. We focus on activation functions that are mere clamps such as ReLU. Fusing of mathematical functions are further discussed in [74].

Static quantization quantizes weights to INT8 and activations to UINT8. To prepare the model, Pytorch modules *QuantStub* and *DeQuantStub* are used in the Network f3. These modules are used to quantize the inputs and de-quantize the output of the final layer. Activations are fused to the preceding layers where possible, to improve both model accuracy and performance. Convolution operation was fused with ReLU operation at each layer. Quantization configuration was set to *qnnpack*, which specifies how weights and activations should be quantized. Modules were added to observe the activation tensors during calibration stage using *torch.quantization.prepare()* method. Then the model was calibrated using the validation dataset. Finally, the calibrated model was converted with the *torch.quantization.convert()* method. This method quantizes the weights, computes and stores the scale and bias value to be used in each activation tensor, and replaces key operators with quantized implementations.

In the next chapter, the bi-prediction performance and encoding times of floating-point CNN and integer arithmetic only CNN are experimentally compared.

Chapter 6

Experimental Results and Discussion

The main objective of this thesis has been to investigate the possibility of reducing the computation complexity of a video codec by replacing the classical motion estimation-based bi-prediction by the U-Net CNN described in Chapter 5. This chapter presents experimental results which compare the prediction performance and encoding times of the CNN approach with the state-of-the-art low-complexity TZ search motion estimation algorithm described in section 4.3.4.

The performance evaluations are conducted in this chapter as follows. In section 6.1 the bi-prediction performance of floating point CNN and integer arithmetic only CNN algorithms are evaluated and compared. The RD performance of the encoder with different bi-prediction techniques are evaluated in section 6.2. In section 6.3, the complexity of CNN based bi-prediction methods are compared with the motion estimation.

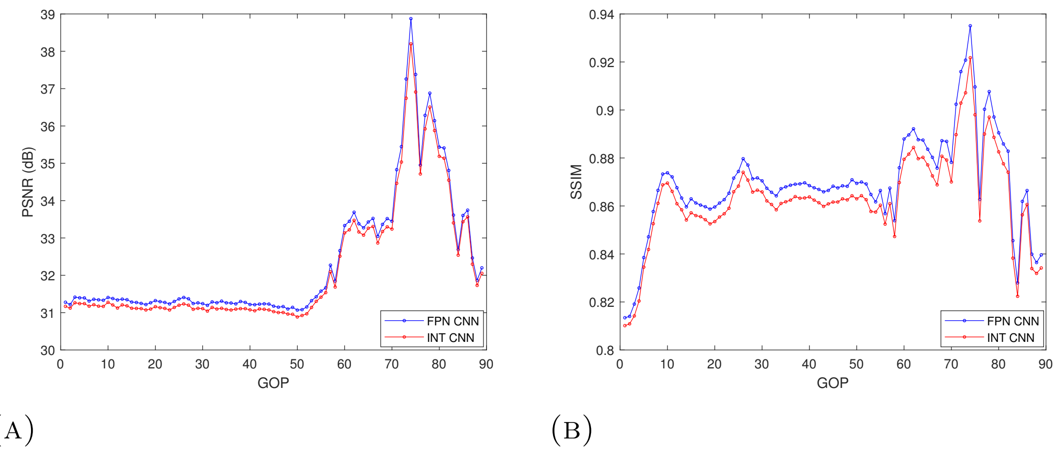
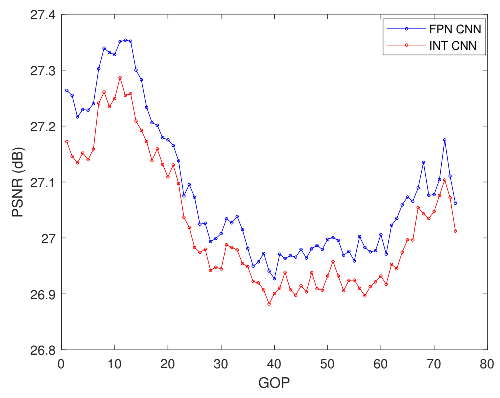


Fig. 6.1: GOP-wise bi-prediction performance for Football sequence: (A) PSNR, (B) SSIM.

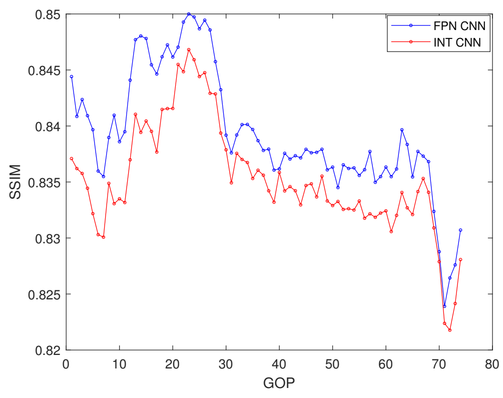
6.1 Performance Evaluation of U-Net CNN Architecture

In this section, we experimentally evaluate the prediction performance of the U-Net CNN architecture Network f3 described in section 5.2. Both floating point and integer-only (8-bit integer arithmetic) U-Net architectures are evaluated and compared.

The performance of these algorithms were evaluated using 5 different video sequences from Xiph.org video test media [75]. Football, Mobile, News, Flower and Stefan are the selected video sequences. These sequences were chosen as they cover different types of motion, speeds, objects, camera motion, and scene changes. We consider bi-prediction of frames inside a GOP in a hierarchical manner as shown in Fig. 5.1. As our goal is to compare the reduced complexity integer-only U-Net with the full-complexity floating point U-Net, in this section we only consider the hierarchical predictions that are initiated from the original (uncoded) I frames. This means that all results in this section applies to open-loop prediction.

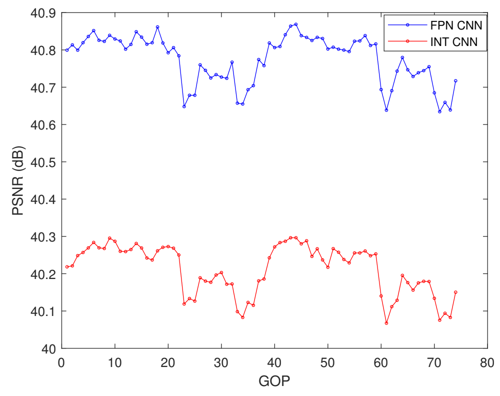


(A)

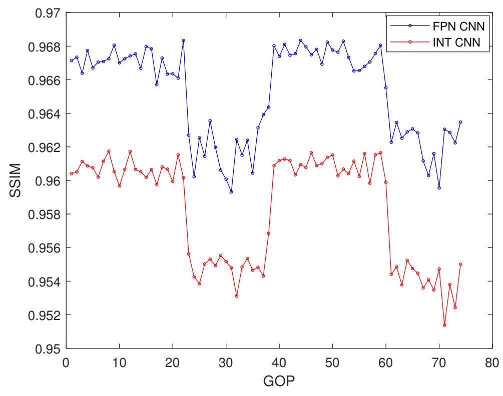


(B)

Fig. 6.2: GOP-wise bi-prediction performance for Mobile sequence: (A) PSNR, (B) SSIM.



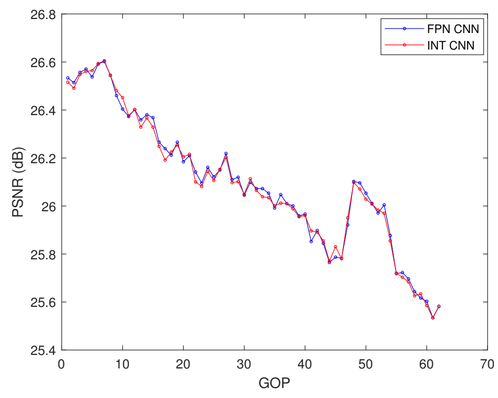
(A)



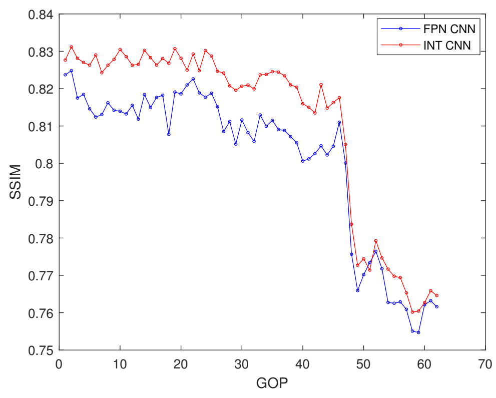
(B)

Fig. 6.3: GOP-wise bi-prediction performance for News sequence: (A) PSNR, (B) SSIM.

Figs. 6.1 to 6.5 compare the PSNR and SSIM performance of the two U-Net architectures. For the simplicity of presentation, these figures show the PSNR and SSIM averaged over each GOP in the video sequence, where a GOP consists of 4 frames as shown in Fig. 5.1. The per-GOP PSNR and SSIM refer to the averages over the three predicted frames B1, B2, and B3. The bi-prediction performance of the floating point CNN is evaluated by PSNR and SSIM metrics as given in Table 6.1. For video with 8-bit resolution a PSNR of 30 dB is considered good.

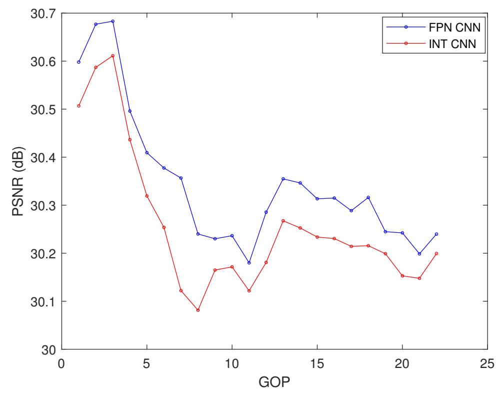


(A)

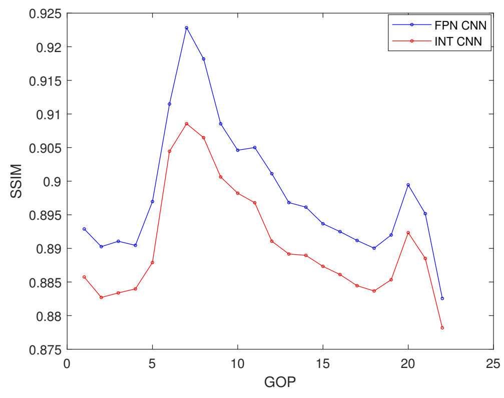


(B)

FIG. 6.4: GOP-wise bi-prediction performance for Flower sequence: (A) PSNR, (B) SSIM.



(A)



(B)

FIG. 6.5: GOP-wise bi-prediction performance for Stefan sequence: (A) PSNR, (B) SSIM.

Accordingly, floating point CNN based bi-prediction performs well in Football, and Stefan video sequences and the bi-prediction performance in News sequence is nearly transparent. The other two sequences have average performance according to the PSNR evaluation. The SSIM performance of the floating point CNN bi-prediction given in Table 6.1 is fairly good for all the sequences. A low SSIM performance is observed for the Flower sequence, for which the PSNR was also the lowest compared to other sequences. Next, the performance loss in using integer

Table 6.1: Average bi-prediction performance of the floating point CNN.

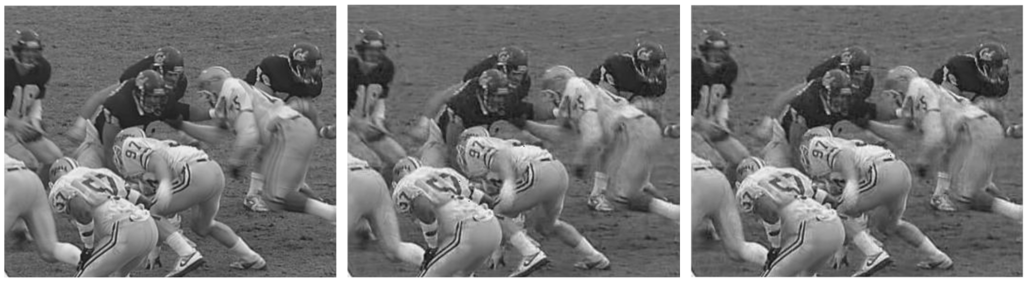
Sequence	PSNR (dB)	SSIM
Football	32.4	0.71
Mobile	27.1	0.75
News	40.8	0.70
Flower	26.1	0.54
Stefan	38.4	0.72

Table 6.2: Loss in bi-prediction performance of the integer arithmetic CNN compared to the floating point CNN.

Sequence	PSNR loss (dB)	SSIM loss (%)
Football	0.2	0.5
Mobile	0.1	0
News	0.6	1
Flower	0	0
Stefan	0.1	0.5

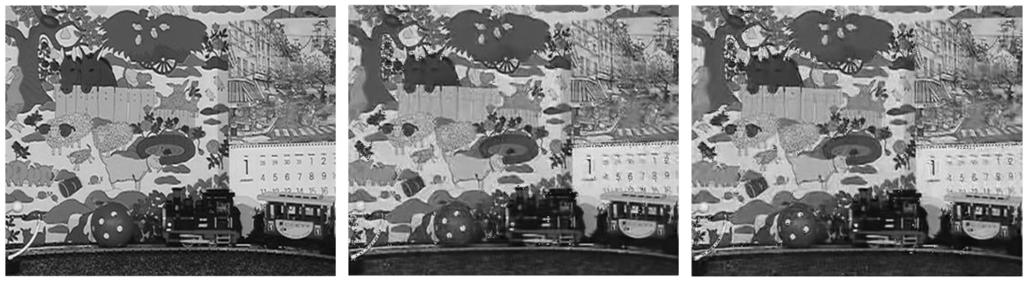
arithmetic only CNN for bi-prediction in place of floating point CNN is given in Table 6.2. The PSNR performance loss and SSIM performance loss for any given video sequence are below 0.6 dB and 1% respectively. For some sequences, no performance loss is observed with PSNR or SSIM metrics. Meaningful difference in visual quality can be observed if the PSNR and SSIM performance loss is greater than 0.5 dB and 1% respectively. Therefore, we can safely conclude that switching the bi-prediction architecture from floating point CNN to integer arithmetic only CNN does not result in a considerable performance loss.

To better understand the bi-prediction performance in using the CNN based architectures, the visualizations of bi-predicted frames of the video sequences are given in Figs. 6.6 to 6.10. The visual quality of bi-predictions from both CNN based architectures are very similar. Therefore, a significant difference in visual



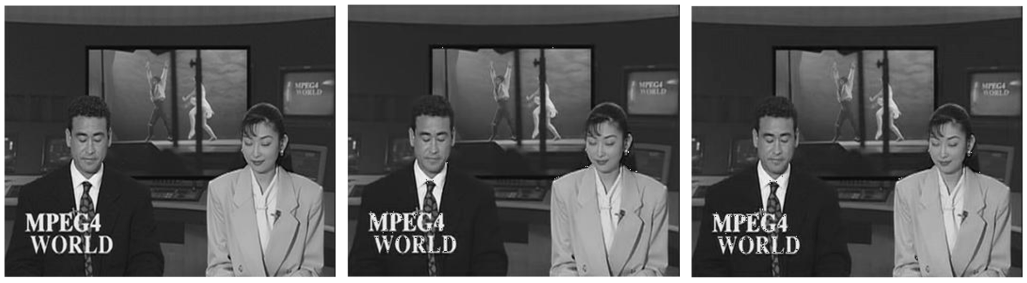
(A) (B) (C)

Fig. 6.6: Bi-predicted frames in Football sequence: (A) Original frame, (B) Bi-predicted frame from floating point CNN, (C) Bi-predicted frame from integer CNN.



(A) (B) (C)

Fig. 6.7: Bi-predicted frames in Mobile sequence: (A) Original frame, (B) Bi-predicted frame from floating point CNN, (C) Bi-predicted frame from integer CNN.



(A) (B) (C)

Fig. 6.8: Bi-predicted frames in News sequence: (A) Original frame, (B) Bi-predicted frame from floating point CNN, (C) Bi-predicted frame from integer CNN.

performance can not be observed between floating point CNN and integer CNN based bi-predictions.

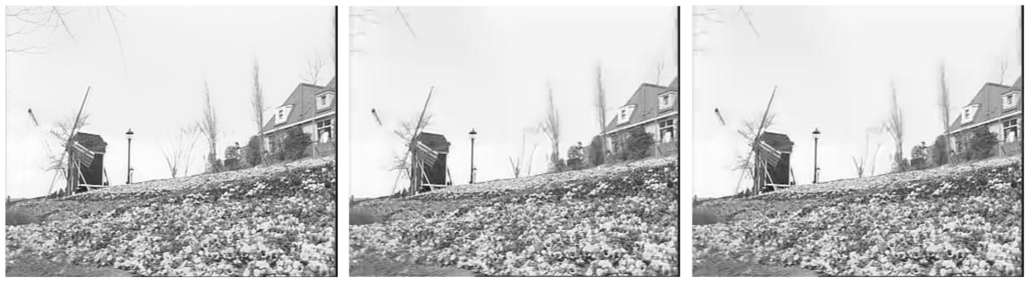


Fig. 6.9: Bi-predicted frames in Flower sequence: (A) Original frame, (B) Bi-predicted frame from floating point CNN, (C) Bi-predicted frame from integer CNN.

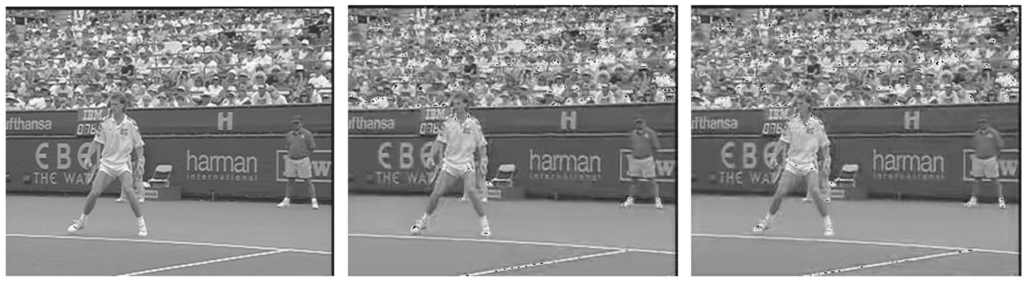


Fig. 6.10: Bi-predicted frames in Stefan sequence: (A) Original frame, (B) Bi-predicted frame from floating point CNN, (C) Bi-predicted frame from integer CNN.

6.2 Video Coding Experiments

6.2.1 Comparison of RD performance

In this section we evaluate and compare the performance of video compression using the HM 16.22 video encoder with motion estimation based bi-prediction and the same video encoder with motion estimation replaced by U-Net CNN based bi-prediction. When using U-Net based bi-prediction, the prediction error frames have been encoded as still frames (I-frames). In the following we refer to motion estimation based coding as ME, floating point U-Net based coding as FPN CNN

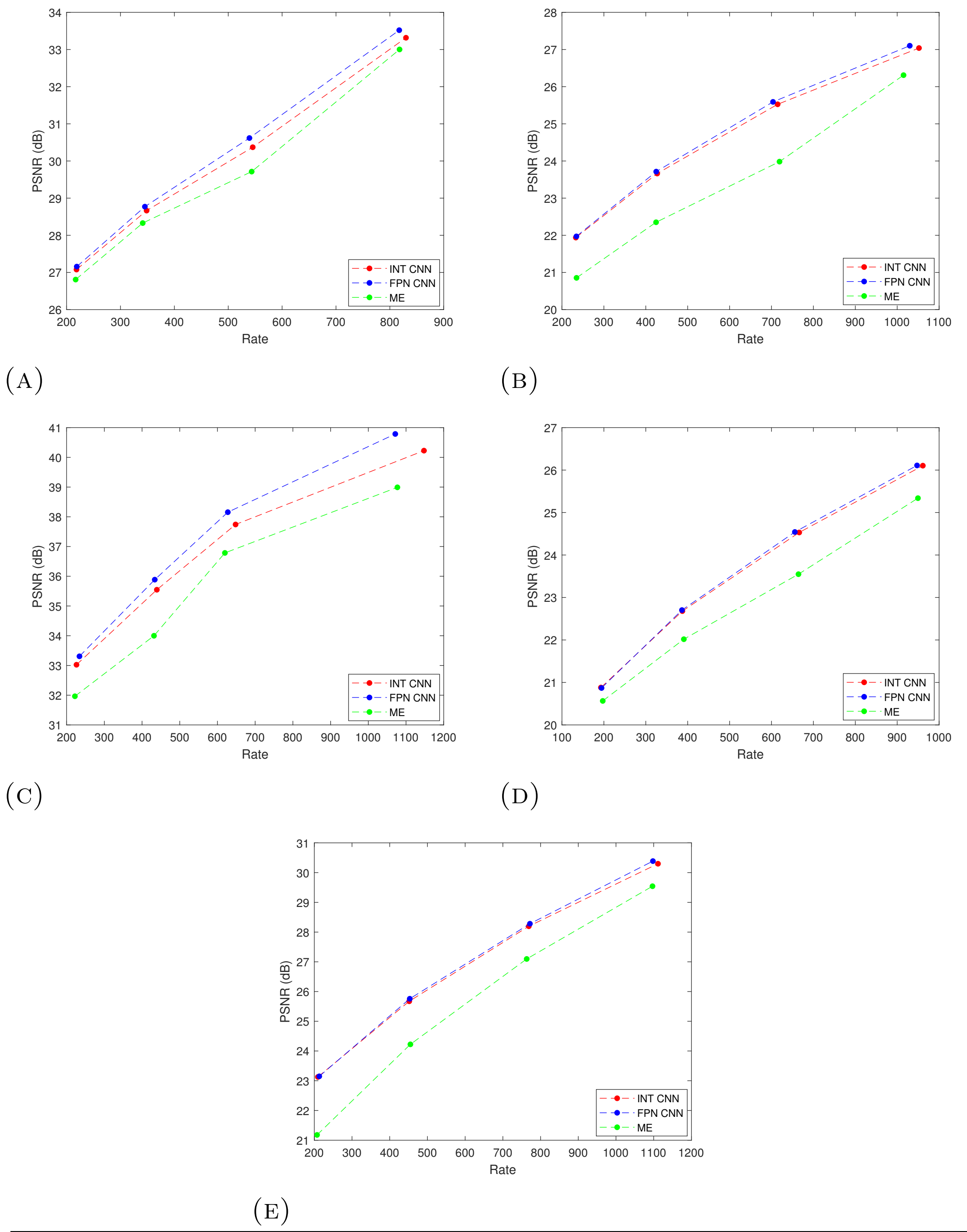


FIG. 6.11: Bit rate versus PSNR curves: (A) Football sequence, (B) Mobile sequence, (C) News sequence, (D) Flower sequence, (E) Stefan sequence.

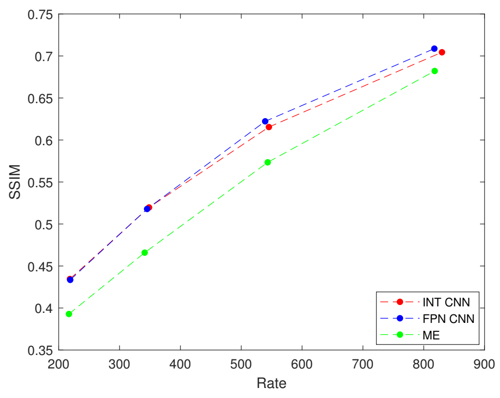
and integer U-Net based coding as INT CNN.

Table 6.3: Comparison of BD-PSNR performance between video codecs based on different bi-prediction methods.

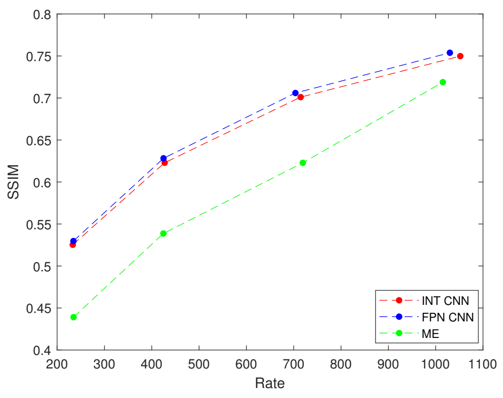
Sequence	FPN CNN to INT CNN BD-PSNR (dB)	FPN CNN to ME BD-PSNR (dB)	INT CNN to ME BD-PSNR (dB)
Football	-0.2	-0.6	-0.4
Mobile	-0.1	-1.3	-1.3
News	-0.5	-1.6	-1.1
Flower	-0.1	-0.7	-0.7
Stefan	-0.1	-1.5	-1.4

Fig. 6.11 shows the RD performance for the five video sequences we have considered. These results indicate that CNN-based bi-prediction consistently outperforms, the motion-estimation by noticeable margins. The main reason for this RD performance gain is that, with motion estimation, a part of the output bit-rate must be used to signal the motion information to the decoder, which is not required when CNN based prediction is used.

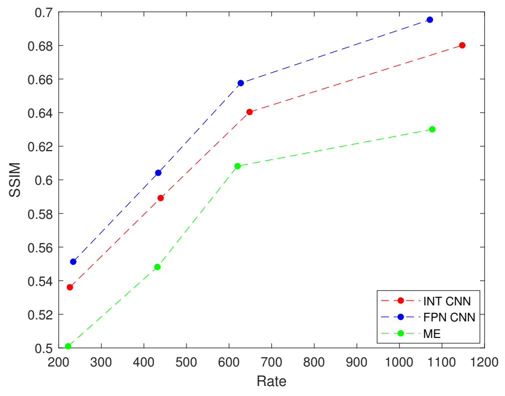
Table 6.3 summarizes the relative performance gains among the three coding methods in terms of the BD-PSNR. The BD-PSNR drops by 0.2 dB for the Football sequence in changing the bi-prediction algorithm from FPN CNN to INT CNN. For the same sequence the BD-PSNR drops by 0.4 dB when the bi-prediction algorithm is changed from INT CNN to ME. By considering performance for the 5 sequences, we can observe that, a maximum reduction of 0.5 dB in BD-PSNR has occurred in changing the bi-prediction architecture from FPN CNN to INT CNN. But in changing the bi-prediction method from FPN CNN to ME, the maximum reduction in PSNR for the reconstructed video sequence is 1.6 dB. Thus, we can observe that even in using INT CNN for bi-prediction, a PSNR performance gain can be achieved compared to the motion estimation based bi-prediction at the same bit-rate.



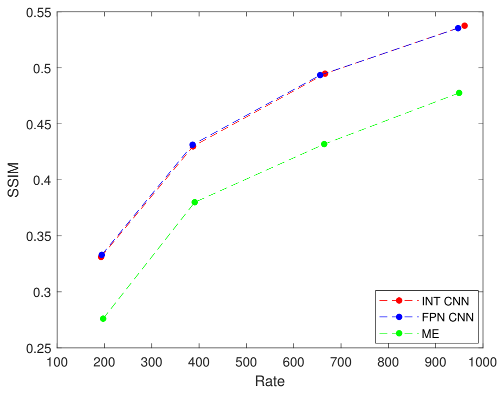
(A)



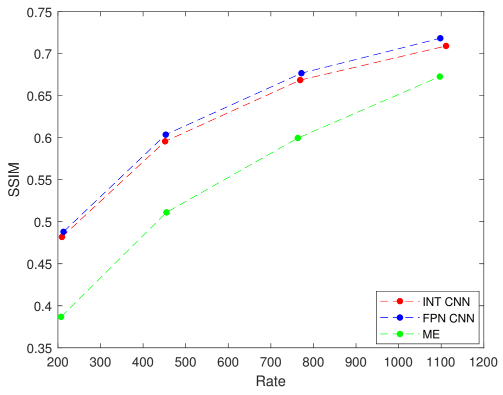
(B)



(C)



(D)



(E)

FIG. 6.12: Bit rate versus SSIM curves: (A) Football sequence, (B) Mobile sequence, (C) News sequence, (D) Flower sequence, (E) Stefan sequence.

Table 6.4: Comparison of BD-SSIM performance between the codecs based on different bi-prediction methods.

Sequence	FPN CNN to INT CNN BD-SSIM	FPN CNN to ME BD-SSIM	INT CNN to ME BD-SSIM
Football	0	-0.05	-0.05
Mobile	-0.01	-0.08	-0.08
News	-0.02	-0.05	-0.04
Flower	0	-0.06	-0.06
Stefan	-0.01	-0.09	-0.08

SSIM values achieved with video coding, using the three bi-prediction methods are given in Fig. 6.12. The SSIM results follow the same pattern as the PSNR results. The FPN-CNN method performs the best with all 5 video sequences at every bit-rate tested. Consider the SSIM performance for the Football sequence given in Fig. 6.12(A). At the average bit-rate of 820 kbps, FPN-CNN and INT-CNN coding reaches a SSIM performance of 0.71 and 0.70 respectively. But the SSIM achieved by ME based coding is only 0.68. In both Mobile sequence (Fig. 6.12(B)) and Stefan sequence (Fig. 6.12 (E)), about 10% of SSIM performance reduction can be observed in ME based coding compared to CNN based coding. If we consider the Flower sequence SSIM performance given in Fig. 6.12 (D), both INT CNN coding and FPN CNN coding perform the same at every bit rate considered.

BD-SSIM performance between FPN CNN, INT CNN and ME based codecs at a fixed bit-rate are summarized in Table 6.4. FPN CNN coding has the highest BD-SSIM for all 5 sequences. When the bi-prediction architecture is changed from FPN CNN to INT CNN, the reduction in average SSIM is less than 0.01 which is a negligible performance loss in visual analysis. But a significant reduction in average SSIM, greater than 0.06 can be observed in using ME based coding, when compared to FPN CNN coding. Therefore, even INT CNN coding performs better than ME based coding at the same bit-rate, according to BD-SSIM evaluation.

Table 6.5: Comparison of bit rate savings between the codecs based on different bi-prediction methods.

Sequence	FPN CNN to INT CNN BD-BR (%)	FPN CNN to ME BD-BR (%)	INT CNN to ME BD-BR(%)
Football	5.2	16.4	11.9
Mobile	2.5	33.1	31.6
News	10.6	25.6	19.7
Flower	1.1	21.8	21
Stefan	1.3	24.6	23.7

Next we evaluated the BD-BR or the bit-rate savings in using different bi-prediction approaches. Table 6.5 illustrates the bit-rate saving achieved by changing the bi-prediction method while maintaining the same video quality. For example, FPN CNN based coding can save about 16% in bit-rate over ME based coding for the Football sequence. As the Table 6.5 shows, the CNN-based bi prediction can result in a greater saving of bit-rate compared to ME based bi-prediction for the same video quality. The additional bit-rate penalty in ME based coding is due to the fact that with ME, a fraction of the output bit-rate has to be used to signal motion information to the decoder.

In summary, our BD-PSNR, BD-SSIM, and BD-BR evaluations have shown that the CNN-based bi-prediction improves the RD performance of the HM 16.22 video encoder. Furthermore, the performance loss resulting from restricting the CNN to integer arithmetic is only marginal. In the next section, we present experimental results demonstrating that the computational complexity of the video encoder can be reduced by using an integer CNN for bi-prediction.

TABLE 6.6: Number of integer arithmetic operations required to predict one video frame.

Algorithm	Motion estimation bi-prediction		CNN bi-prediction
	Highest complexity	Lowest complexity	
Total operations	9.5×10^9	6×10^9	1.2×10^9

6.3 Comparison of Computational Complexity

In section 4.4, we determined the number of addition and multiplication operations involved in a reduced complexity motion estimation based bi-prediction algorithm that is available in the HEVC standard HM 16.22 encoder. In section 5.3, we determined the number of operations that will be required for bi-prediction using U-Net CNN architecture. The results are summarized in Table 6.6. These numbers predict that motion estimation may require about $5 - 8$ times more arithmetic operations than the CNN based approach to predict a single video frame. While it is difficult to experimentally verify the number of arithmetic operations involved in the HM 16.22 encoders motion estimation algorithm, it is possible to get a sense of computational complexity by measuring the time taken by the encoder to code a sequence of video frames.

Table 6.7 presents the time taken by the HM 16.22 encoder for encoding the 5 test video sequences. Note that the encoder parameters used in this experiment, are common to all three methods. Therefore, the differences in encoding time are due to bi-prediction of B-frames and transform coding the corresponding prediction errors. As can be seen from Table 6.7, both floating point CNN and integer arithmetic only CNN can perform bi-prediction faster than the simplest, reduced complexity motion estimation algorithm available in HM 16.22 encoder.

As explained in Chapter 4, the exhaustive search for PU mode decision and

TABLE 6.7: Encoding times for HM 16.22 encoder with ME, and CNN based bi-predictions.

Sequence	Average bit rate (kbps)	Encoding time (s)		
		ME	FPN CNN	INT CNN
Football (357 frames)	218	534.2	403.3	218.4
	345	532.7	422.1	348.5
	540	532.7	455.2	390.3
	820	591.9	433.3	390.4
Mobile (297 frames)	234	444.1	320.3	282.2
	425	453.5	333.5	296.5
	710	500.7	363.1	322.4
	1030	590.7	397.3	358.1
News (297 frames)	230	392.8	303.3	271.5
	433	441.4	341.8	303.3
	620	459.7	333.1	301.4
	1070	475.8	392.6	346.2
Flower (249 frames)	194	359.1	287.5	241.5
	386	377.8	298.6	254.6
	655	390.3	314.7	281.1
	947	417.3	327.1	284.6
Stefan (89 frames)	213	133.2	115.4	96.6
	453	141.3	127.8	74.9
	771	139.3	128.3	108.6
	1090	154.4	137.9	116.6

advanced features in motion vector determination are both highly computationally expensive operations in the motion estimation process. This causes an increase in the encoding time during the bi-prediction. With CNNs, a high computational effort is required during the off-line training phase. The computational complexity involved using the trained CNN for bi-prediction inside the video coding-loop is however much lower. As can be seen from Table 6.7, converting the trained CNN from 32-bit floating point arithmetic to 8-bit integer arithmetic leads to a further reduction in encoding times.

Chapter 7

Conclusion and Future Work

7.1 Contributions and Conclusions

1. Inter-frame prediction based on motion estimation is considered as the most computationally expensive and time consuming operation in state-of-the-art video coding standards. In this thesis, a low complexity CNN architecture has been investigated which is inspired by the previously proposed U-Net CNN architecture. In experimental evaluations, it was found that this algorithm is able to provide satisfactory prediction performance.

2. Majority of CNN architectures reported in the literature, including the original U-Net use floating point arithmetic. Floating point arithmetic costs much more computing power compared to integer arithmetic. In this thesis, an integer arithmetic only CNN has been derived by post training static quantization of the floating point CNN. Only a minor video quality performance reduction was observed in this conversion. When used in a video encoder for bi-prediction of B-frames, the encoding time achieved with the

integer only CNN was observed to be about 8% less than that of the floating point version.

3. We have analyzed the computational complexity of both motion estimation and CNN based bi-prediction and determined the number of additions and multiplications required in each algorithm. Our results show that CNN based approach involves much less computational complexity than the motion estimation algorithm.

4. The experimental results show that the proposed integer arithmetic CNN based bi-prediction algorithm saves a considerable computational cost while maintaining a better video quality compared to the simplest motion estimation model available in the HEVC standard HM 16.22 video encoder. Further, the proposed bi-prediction algorithm has shown to improve the RD performance.

7.2 Future Work

1. The experimental and analytical results in Chapter 6 highlight that low complexity bi-prediction algorithms for video coding can be derived using U-Net CNN architectures. Different CNN architectures must be further analyzed to identify whether a higher reduction of complexity is possible for bi-prediction process in video coding.

2. Post training static quantization method is employed to convert a trained floating point CNN architecture to integer arithmetic CNN in this thesis. Quantization aware training method which has greater accuracy, can be used to convert a floating point CNN to integer arithmetic CNN during the training time.

3. The bi-prediction accuracy of CNN based algorithm and motion estimation based algorithm are not compared in this thesis. This comparison would be a useful step to further analyze and improve the bi-prediction performance of the video coding process.

Appendix A

Motion Vector Scaling

Let A and B be the temporal MVP candidates for the current block derived from the AMVP process. Before adding these candidates to the AMVP candidate list, it is checked whether any of the candidate blocks contain a reference index that is equal to the reference index of the current block. If reference indices from candidate blocks are pointing to a different reference picture than the reference index of the current block, the associated motion vector cannot be used as is. Therefore, the motion vectors need to be scaled according to the temporal distances between the candidate reference picture and the current reference picture. The candidate motion vector mv_{cand} is scaled according to a scale factor S_f as

$$mv = sign(mv_{cand} \,.\, S_f) \,.\, ((|mv_{cand} \,.\, S_f| \; + \; 2^7) \; >> \; 8),$$

where the scale factor is calculated as

$$S_f = clip(-2^{12},\ 2^{12} - 1,\ (t_b \,.\, t_x \ +\ 2^5)\ >>\ 6),$$
$$t_x = \frac{2^{14}\ +\ |\frac{t_d}{2}|}{t_d},$$

based on the temporal distance between the current picture and the reference picture of the candidate block t_d and the temporal distance between the current picture and the reference picture of the current block t_b. The temporal distance is expressed in terms of the difference between the picture order count values which define the display order of the pictures. This factoring allows pre-computation of scale factor at PU level since it only depends on the reference picture list structure signaled in the PU header.

.

Bibliography

[1] "Cisco annual internet report - cisco annual internet report (2018–2023) white paper." `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`, Mar 2020.

[2] B. Bross, J. Chen, J.-R. Ohm, G. J. Sullivan, and Y.-K. Wang, "Developments in international video coding standardization after AVC, with an overview of versatile video coding (VVC)," *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1463–1493, 2021.

[3] S. Xia, W. Yang, Y. Hu, and J. Liu, "Deep inter prediction via pixel-wise motion oriented reference generation," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 1710–1774, 2019.

[4] 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. 793–849, 2004.

[5] K. Saurty, P. C. Catherine, and K. M. S. Soyjaudah, "Inter prediction complexity reduction for HEVC based on residuals characteristics," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 10, 2016.

[6] I. Chakrabarti, *Motion Estimation for Video Coding Efficient Algorithms and Architectures.* Studies in Computational Intelligence, 590, Cham: Springer International Publishing, 1st ed. 2015. ed., 2015.

[7] S. Liu, W.-H. Peng, and L. Yu, “Guest editorial introduction to special section on learning-based image and video compression,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 1785–1788, 2020.

[8] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, “Flownet: Learning optical flow with convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, 2015.

[9] S. Niklaus, L. Mai, and F. Liu, “Video frame interpolation via adaptive convolution,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2270–2279, 2017.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, p. 84–90, May 2017.

[11] Z. Liu, R. A. Yeh, X. Tang, Y. Liu, and A. Agarwala, “Video frame synthesis using deep voxel flow,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 4473–4481, 2017.

[12] H. Zhao, D. Liu, and H. Li, “Efficient integer-arithmetic-only convolutional neural networks,” *CoRR*, vol. abs/2006.11735, 2020.

[13] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, “Performance and computational complexity assessment of high-efficiency video encoders,”

IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1899–1909, 2012.

[14] J. Leng, L. Sun, T. Ikenaga, and S. Sakaida, “Content based hierarchical fast coding unit decision algorithm for hevc,” in *2011 International Conference on Multimedia and Signal Processing*, vol. 1, pp. 56–59, 2011.

[15] J. Xiong, H. Li, Q. Wu, and F. Meng, “A fast hevc inter CU selection method based on pyramid motion divergence,” *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 559–564, 2014.

[16] X. Shen, L. Yu, and J. Chen, “Fast coding unit size selection for HEVC based on bayesian decision rule,” in *2012 Picture Coding Symposium*, pp. 453–456, 2012.

[17] M. U. K. Khan, M. Shafique, and J. Henkel, “An adaptive complexity reduction scheme with fast prediction unit decision for HEVC intra encoding,” in *2013 IEEE International Conference on Image Processing*, pp. 1578–1582, 2013.

[18] H.-M. Yoo and J.-W. Suh, “Fast coding unit decision algorithm based on inter and intra prediction unit termination for HEVC,” in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 300–301, 2013.

[19] H. Zhang and Z. Ma, “Fast intra mode decision for high efficiency video coding (HEVC),” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 4, pp. 660–668, 2014.

[20] J. Vanne, M. Viitanen, and T. D. Hämäläinen, “Efficient mode decision schemes for HEVC inter prediction,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 9, pp. 1579–1593, 2014.

[21] L. Shen, Z. Zhang, and Z. Liu, "Effective CU size decision for HEVC intracoding," *IEEE Transactions on Image Processing*, vol. 23, no. 10, pp. 4232–4241, 2014.

[22] K. Miyazawa, T. Murakami, A. Minezawa, and H. Sakate, "Complexity reduction of in-loop filtering for compressed image restoration in HEVC," in *2012 Picture Coding Symposium*, pp. 413–416, 2012.

[23] C.-S. Park, "Edge-based intramode selection for depth-map coding in 3D-HEVC," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 155–162, 2015.

[24] Q. Hu, Z. Shi, X. Zhang, and Z. Gao, "Fast HEVC intra mode decision based on logistic regression classification," in *2016 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, pp. 1–4, 2016.

[25] D. Liu, X. Liu, and Y. Li, "Fast CU size decisions for HEVC intra frame coding based on support vector machines," in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp. 594–597, 2016.

[26] 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," *IEEE Transactions on Image Processing*, vol. 24, no. 7, pp. 2225–2238, 2015.

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

[28] L. Zhu, Y. Zhang, Z. Pan, R. Wang, S. Kwong, and Z. Peng, "Binary and multi-class learning based low complexity optimization for HEVC encoding," *IEEE Transactions on Broadcasting*, vol. 63, no. 3, pp. 547–561, 2017.

[29] 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," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5088–5103, 2016.

[30] T. Laude and J. Ostermann, "Deep learning-based intra prediction mode decision for HEVC," in *2016 Picture Coding Symposium (PCS)*, pp. 1–5, 2016.

[31] S. Kuanar, K. R. Rao, M. Bilas, and J. Bredow, "Adaptive CU mode selection in HEVC intra prediction: A deep learning approach," *Circuits, systems, and signal processing*, vol. 38, no. 11, pp. 5081–5102, 2019.

[32] M. Xu, T. Li, Z. Wang, X. Deng, R. Yang, and Z. Guan, "Reducing complexity of hevc: A deep learning approach," *IEEE Transactions on Image Processing*, vol. 27, no. 10, pp. 5044–5059, 2018.

[33] 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, 2003.

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

[35] P. Yin, A. Tourapis, and J. Boyce, "Localized weighted prediction for video coding," in *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4365–4368 Vol. 5, 2005.

[36] L. Li, H. Li, Z. Lv, and H. Yang, "An affine motion compensation framework for high efficiency video coding," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 525–528, 2015.

[37] X. Zhang, R. Xiong, W. Lin, J. Zhang, S. Wang, S. Ma, and W. Gao, "Low-rank-based nonlocal adaptive loop filter for high-efficiency video compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 10, pp. 2177–2188, 2017.

[38] X. Zhang, W. Lin, R. Xiong, X. Liu, S. Ma, and W. Gao, "Low-rank decomposition-based restoration of compressed images via adaptive noise estimation," *IEEE Transactions on Image Processing*, vol. 25, no. 9, pp. 4158–4171, 2016.

[39] X. Zhang, R. Xiong, W. Lin, S. Ma, J. Liu, and W. Gao, "Video compression artifact reduction via spatio-temporal multi-hypothesis prediction," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 6048–6061, 2015.

[40] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[41] N. Yan, D. Liu, H. Li, and F. Wu, "A convolutional neural network approach for half-pel interpolation in video coding," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.

[42] C. Jia, S. Wang, X. Zhang, S. Wang, and S. Ma, "Spatial-temporal residue network based in-loop filter for video coding," *2017 IEEE Visual Communications and Image Processing (VCIP)*, Dec 2017.

[43] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *Computer Vision – ECCV 2016*, Lecture Notes in

Computer Science, pp. 286–301, Cham: Springer International Publishing, 2016.

[44] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using LSTMs," in *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 843–852, 2015.

[45] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," in *4th International Conference on Learning Representations, ICLR 2016*, (San Juan, Puerto Rico), 2016.

[46] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[47] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive convolution," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 2270–2279, IEEE Computer Society, Jul 2017.

[48] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 261–270, 2017.

[49] L. Zhao, S. Wang, X. Zhang, S. Wang, S. Ma, and W. Gao, "Enhanced motion-compensated video coding with deep virtual reference frame generation," *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 4832–4844, 2019.

[50] S. Huo, D. Liu, F. Wu, and H. Li, "Convolutional neural network-based motion compensation refinement for video coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2018.

[51] Z. Zhao, S. Wang, S. Wang, X. Zhang, S. Ma, and J. Yang, "CNN-based bi-directional motion compensation for high efficiency video coding," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2018.

[52] J. L. Martínez, P. Cuenca, F. Delicado, and F. Quiles, "Objective video quality metrics: A performance analysis," in *2006 14th European Signal Processing Conference*, pp. 1–5, 2006.

[53] A. Horé and D. Ziou, "Image quality metrics: PSNR vs. SSIM," in *2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, 2010.

[54] Z. Wang and A. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, 2002.

[55] 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, 2004.

[56] Z. Wang, L. Lu, and A. C. Bovik, "Video quality assessment based on structural distortion measurement," *Signal Processing: Image Communication*, vol. 19, no. 2, pp. 121–132, 2004.

[57] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," in *ITU-T SG 16 Q.6 document VCEG-M33, 13th VCEG meeting*, (Austin, Texas, USA), 2001.

[58] H. Tao, J. Qian, L. Yu, and H. Wang, “Bi-prediction enhancement with deep frame prediction network for versatile video coding,” in *2021 Data Compression Conference (DCC)*, pp. 374–374, 2021.

[59] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, “Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC),” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, 2012.

[60] M. Wien, *High Efficiency Video Coding: Coding Tools and Specification.* Springer Publishing Company, Incorporated, 2014.

[61] *High Efficiency Video Coding (HEVC) Algorithms and Architectures.* Integrated Circuits and Systems, Cham: Springer International Publishing, 1st ed. 2014. ed., 2014.

[62] S. Wang, Z. Wang, F. Luo, S. Wang, S. Ma, and W. Gao, “Enhanced motion vector prediction for video coding,” in *2018 IEEE Fourth International Conference on Multimedia Big Data (BigMM)*, pp. 1–5, 2018.

[63] R. Khemiri, N. Bahri, F. Belghith, F. E. Sayadi, M. Atri, and N. Masmoudi, “Fast motion estimation for HEVC video coding,” in *2016 International Image Processing, Applications and Systems (IPAS)*, pp. 1–4, 2016.

[64] N. Doan, T. S. Kim, C. E. Rhee, and H.-J. Lee, “A hardware-oriented concurrent tz search algorithm for high-efficiency video coding,” *EURASIP journal on advances in signal processing*, vol. 2017, no. 1, pp. 1–17, 2017.

[65] H. Lv, R. Wang, X. Xie, H. Jia, and W. Gao, “A comparison of fractional-pel interpolation filters in HEVC and H.264/AVC,” in *2012 Visual Communications and Image Processing*, pp. 1–6, 2012.

[66] H. Ahn and C. Yim, "Convolutional neural networks using skip connections with layer groups for super-resolution image reconstruction based on deep learning," *Applied Sciences*, vol. 10, p. 1959, 03 2020.

[67] "Pytorch." `https://pytorch.org/`.

[68] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," in *CRCV-TR-12-01*, Nov 2012.

[69] "Writing custom datasets, dataloaders and transforms." `https://pytorch.org/tutorials/beginner/data_loading_tutorial.html`.

[70] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss functions for image restoration with neural networks," *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 47–57, 2017.

[71] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[72] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, 01 2010.

[73] "Quantization." `https://glaringlee.github.io/quantization.html`.

[74] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

[75] "Xiph.org video test media [derf's collection]." `https://media.xiph.org/video/derf/`.