Deep Learning based Fast Mode Decision in HEVC Intra Prediction using Region Wise Feature Classification

by

SHIBA KUANAR

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

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2018

Deep Learning based Fast Mode Decision in HEVC Intra Prediction using Region Wise Feature Classification.

The members of the Committee approve the doctoral dissertation of SHIBA KUANAR.

Supervising Professors:

Dr. K.R.Rao

rao@uta.edu

Dr. Jonathan W Bredow

jbredow@uta.edu

Dr. Kambiz Alavi

alavi@uta.edu

Dr. William E Dillon

dillon@uta.edu

Dr. Howard Russell

hrussell@uta.edu

I dedicate this thesis to my beloved parents, Dr. Balakrishana Kuanar (father) and Santilata Kuanar (mother), who have inculcated in me the love for science and education, my son Pratik Kuanar and my wife Monalisa Bilas, who has been a source of support and encouragement. I love you all and I thank you for being there for me.

ACKNOWLEDGEMENTS

I would like to thank my supervising professor K.R. Rao for constantly motivating and encouraging me and also for his invaluable advice during the course of my doctoral studies.

I wish to thank my academic advisors Dr. Jonathan Bredow, Dr. Kambiz Alavi, Dr. William Dillon, and Dr. Howard Russell for their interest in my research and for taking time to serve in my dissertation committee.

I would also like to extend my appreciation to Dr. Athitsos Vassilis, giving a research opportunity at Vision Learning-Mining (VLM) Lab, UTA CSE department and also his research support and encouragement. I am indebted towards him for shaping my research life.

I am especially grateful to Dr. Nityananda Pradhan (NIMHANS Bangalore India) and Dr. Aravinda Mishra for their valuable help during my research and for the helpful discussions and invaluable comments. I also wish to thank Dr. Arabinda Mishra (Vanderbilt University) for taking the time to critically evaluate this manuscript.

I am grateful to all the teachers and professors who taught me during the years I spent in school/university, first in India and finally in the Unites States. I would like to thank Dr. Nityananda Pradhan for encouraging and inspiring me to pursue my Ph.D. graduate studies.

Finally, I would like to express my deep gratitude to my wife who has encouraged and inspired me, and being patient and understanding during my graduate studies. I would also like to dedicate this work to our beautiful son, Pratik, who is the joy of our lives.

Life in graduate school would not have been good if not for the help and encouragement of fellow graduate students. I would like to express my thanks to my lab members: Christopher Conly, Pat Jangyodsuk, Alex Dillhoff, and Amir Ghaderi. I am also thankful to Dr. Manfred Huber and Dr. Farhad Kamangar lab member: Mohsen Kheirandishfard. I also thank several of my friends who have helped me throughout my career. Above all, I want to thank God for providing me with all that I could ask for and more.

November 2nd, 2018

ABSTRACT

Deep Learning based Fast Mode Decision in HEVC Intra Prediction using Region Wise Feature Classification

SHIBA KUANAR, Ph.D.

The University of Texas at Arlington, 2018

Supervising Professor: K.R. Rao

The High Efficiency Video Coding (HEVC) standard has achieved best coding efficiency as compared to previous H.264/AVC standard. But the computational time of HEVC encoder has increased mainly because of the hierarchical quad-tree based structure, recursive search for finding the best coding units, and the exhaustive prediction search up-to 35 modes. These advances improve the coding efficiency, but result into a very high computational complexity. Furthermore selecting the optimal modes among all prediction modes are necessary for the subsequent rate distortion optimization process.

Therefore we propose a convolutional neural network (CNN) based algorithm which learns the region wise image features and performs a classification job. These classification results are later used in the encoder downstream systems for finding the optimal coding units in each of the tree blocks, and subsequently reduce the number of prediction modes. For our model training, we gathered a new dataset which includes diverse images for the better generalization of our results. The experimental results show that our proposed learning based algorithm reduces the encoder time up to 66.15 % with a minimal Bjontegaard Delta Bit Rate (BD-BR) loss of 1.34 % over the state-of-the-art machine learning approaches. Furthermore our method also reduces the mode selection by 45.91 % with respect to the HEVC baseline.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

With the emergence of next-generation digital media and fast development of multimedia technology, the high definition (HD) and ultra-high definition (UHD) resolution videos are becoming more and more popular. This explosive growth of video content creates an urgent need for a new and better coding technology. In order to meet this demand, the Joint Collaborative Team on Video Coding (JCT VC) has come up with High Efficiency Video Coding (HEVC) standard [2]. HEVC adopts various advanced coding techniques like, flexible quad-tree coding structure, sample adaptive offset, advanced motion vector predictions, and strives to achieve around 50% bitrate reduction as compared to previous video coding standard [2], [3]. Although the latest HEVC video coding standard supports block-based hybrid framework [3], the coding efficiency improvement came from recursive quad-tree based flexible CU partitioning of ranging from 8×8 to 64×64 [4]. To support more flexible coding segmentation, the HEVC standard provides a hierarchy of adaptive coding tree units (CTUs) which include non-overlapping coding units (CU) and prediction units (PU). Each CU can be recursively split into four sub-CUs and again further split into one or multiple PUs as shown in Fig. 1.1. For intra coding the CU supports two PU partitioning types $2N \times 2N$ and $N \times N$. The spatial intra prediction [3] in HEVC has been significantly improved with more fine granular predictions which include mode 0 for planar, mode 1 for DC and 33 angular modes. These modes are employed to adapt to different videos with diverse contents. According to HEVC standard, the rate-distortion optimization (RDO) process performs an iterative check for the CTU, all combinations of CU partitions, PU modes, and selects the optimal ones which are normally with the minimum RD cost. The above increased number of mode searches in all directions includes a top-down checking, bottom-up comparison process and hence requires a huge computation load. Again selecting smaller CU/PU size usually results in a large number of bits to signal the mode information. It might become an overhead to the network. Therefore, the encoder computation time becomes a big burden for the real-time video applications. So it is essential to optimize the original partition procedure of the quad-tree structure and lower the entire computational complexity of encoding process.

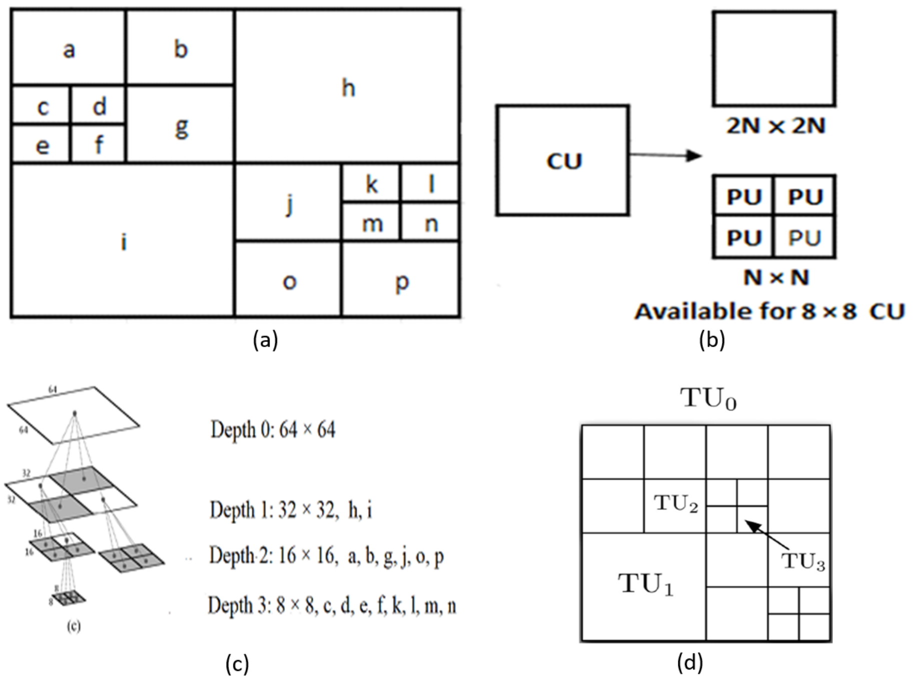


Figure 1.1: (a) CTU structure partition, (b) PU sizes for CU intra prediction, and (c) Hierarchical depth of a CTU divided into various CU sizes and (d) TU structure.

To reduce the computational time in HEVC intra prediction, many efficient algorithms have been proposed in the past decade. Piao et al. [5] introduced a rough mode decision method to pick N best candidates from 35 modes using Hadamard cost. Zhao et al [4] added most probable modes to the rough mode decision candidates by utilizing the direction information of image edges and mode correlations. These kinds of algorithms have positive effects on accelerating the intra prediction process, but they need to search all CU depths to find the optimal tree partition. Wang et al [6] introduced a three step prediction algorithm where the fast prediction of CU splits, and the low precision rough mode decision (RMD) were used to speed up the intra coding. In Shen et al [7] an effective CU size decision algorithm was adopted to bypass the intra prediction for large CUs, and an early determination strategy was used to reduce the number of candidate CU sizes. This strategy utilized texture homogeneity and coding information from the neighbors. Min et al [8] also released a fast CU size decision algorithm by utilizing both the global and local edge complexities in different directions to decide the CTU partitions. Xiong et al [9] proposed a pyramid motion divergence based faster CU selection algorithm for the HEVC inter prediction. They investigated the correlation between the CU splitting, motion divergences which was calculated using pixel motion vectors, and subsequently made a more precise CU splitting decision. This method worked effectively, but calculated the pixel wise optical flows which required additional hardware resources for the final implementation. In spite of above various algorithmic implementations, the encoder computation time is still high. On the other hand, the intra frame CU predictions are highly correlated to that of neighboring units in terms of texture, color, and shape cues which are not investigated in great detail. Hence, there is still some space for further reduction in encoder time.

In this thesis, we propose a deep CNN based algorithm to find the optimal CU depth and subsequently reduce the number of prediction modes search. Our CNN model learns the features from the input CTU image regions and classifies those, by using a softmax classifier. For our model parameter learning, we collected a diversified dataset [10] which contains thousands of images with different textures (homogeneous, granular and dynamic), object shapes (small or big) and colors. Textures, patterns, and colors are the key components in analyzing the natural objects and describe the low-level image attributes [11]. Textures in images represent the high-intensity variations and exhibit repeated structures. Again the number of details present in textures can be quantified based on the granularity labels and used as an attribute in our classification task. For our object detection, we are applying a region proposal network on top of the convolution layer (fifth) and follows the training steps explained in [12], [13]. For texture analysis, the low-level image cues are collected from the coarser convolution layer (third) and later merged to the fully connected (FC7) layer. Our approach effectively detects image texture and object shapes in CUs and classifies the spatial patterns into four classes. Based on the classification results our algorithm is able to predict the CU depth partition and later reduces the angular mode search which has a good tradeoff between RD costs and encoding time. We provide our results in terms of BD-BR [10], BD-PSNR, and encoder time-saving T in section 6.

1.2 Related Work

To the best of our knowledge in recent years, there are fewer deep learning based papers published on encoder intra predictions. But in general the HEVC intra prediction can be divided into two classes: heuristic based and learning based. In heuristic methods, some intermediate encoding features are learned and determines the CTU partitions. Cho et al [14] developed a fast splitting and pruning method at

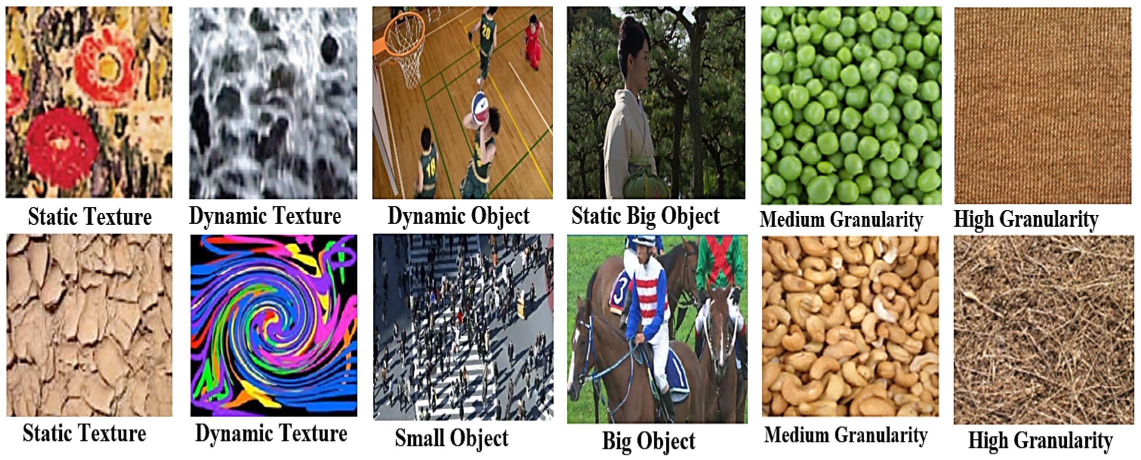
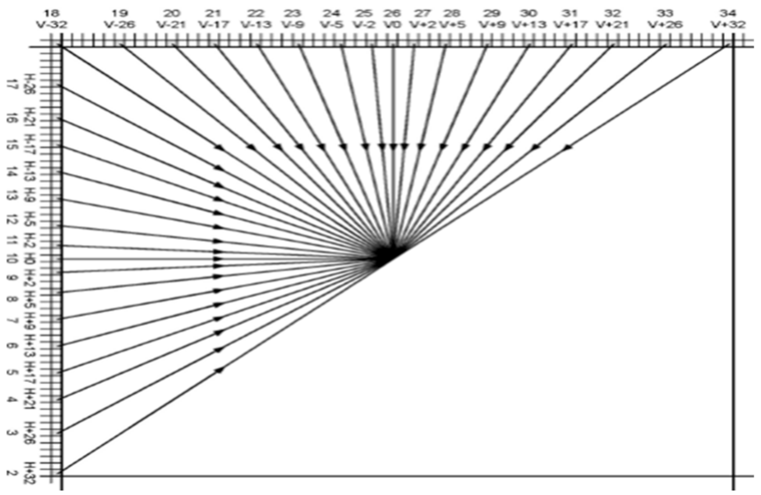
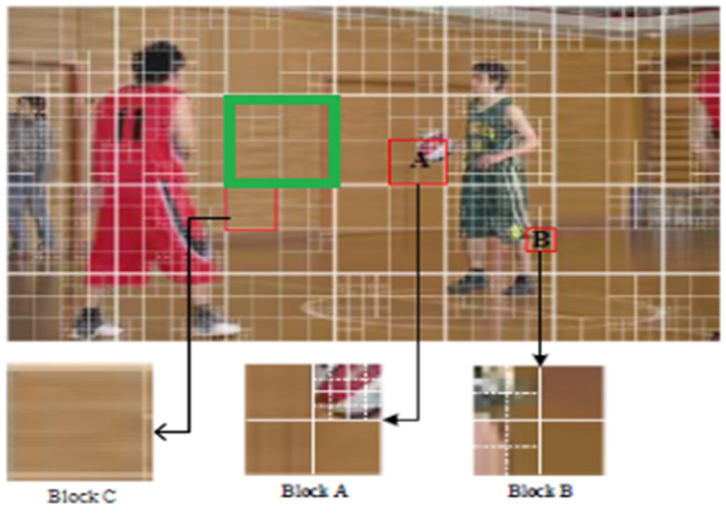


Figure 1.2: Discussions of the implications of the results shown in the figure should be left for the main text.

each CU depth level according to a Bayes decision rule based on RD costs (full and low complexity). Khan et al [15] proposed a content-based scheme that adaptively combined smaller PUs into larger PUs by recursively comparing the RD cost. The arbitrariness of the above heuristic methods is found difficult in correlating the intermediate features and achieving desired RD performance. To solve such problems the machine learning based methods have emerged to reduce the intra mode coding complexity. Correa et al [16] proposed a partition structure optimization scheme based on decision tree mining technique and simplified the optimal CTU structure. Zhang et al [17] introduced the learning based CU depth decision method with a joint support vector machine (SVM) classifier. Hu et al [18] modeled the optimal CU partition as a binary classifier using logistic regression and further alleviated the encoder computation. In all the above methods, some handcrafted features were manually extracted to predict the partition patterns. Again these handcrafted features rely heavily on the prior knowledge about their relationships with CTU partitions. To overcome the above difficulties the CNN based architectures are introduced. Liu et al [19] have developed two convolutional layered shallow CNN architectures to predict the CTU structure. Li et al [1] introduced a CNN based model for complexity reduction on the CU partition with a new CHIP data set.



a) Angular prediction modes from 2 to 34 and associated displacement parameters.



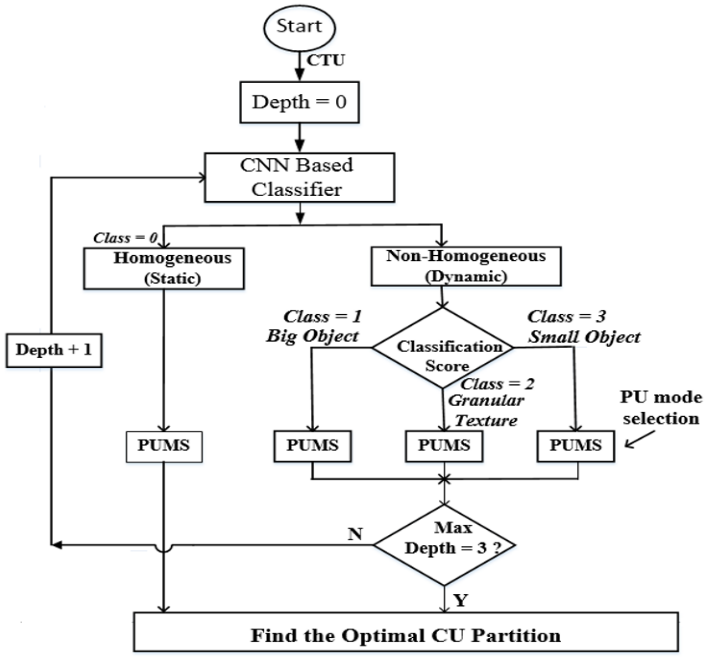
b) CU partitions of "BasketballPass Frame" given by HM16.9 with HEVC Intra coding and QP =22

Figure 1.3: (a) Prediction modes (angular) from 2 to 34 and associated displacement parameters, (b) CU partitions of "BasketballPass Frame" given by HM16.9 with HEVC Intra coding and QP = 22

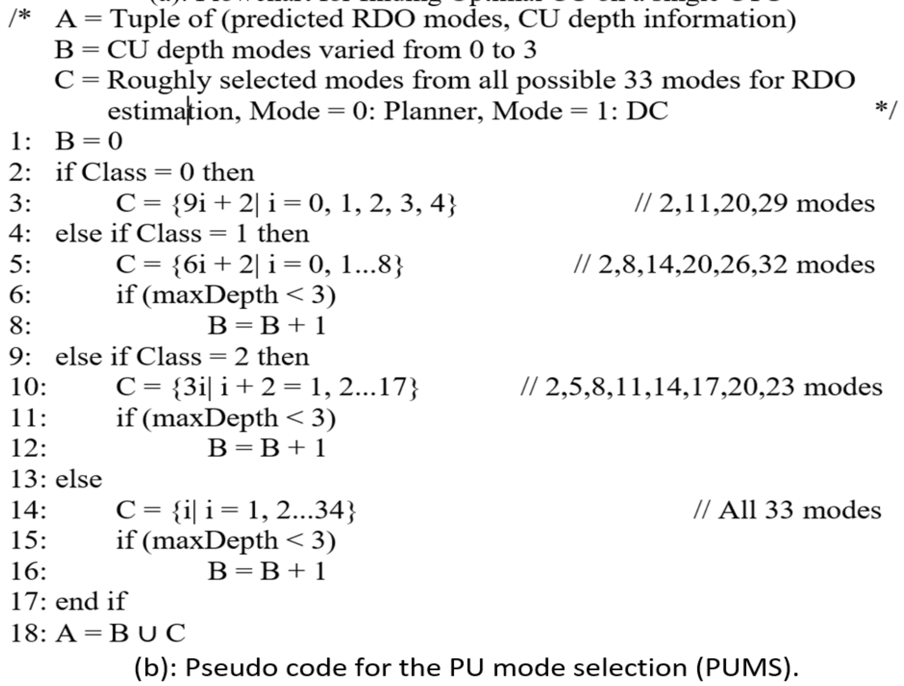
1.3 Overview of Intra Prediction

In HEVC the intra prediction is employed to remove spatial redundancies. Again the angular prediction is adapted to efficiently model different structures present in the frames. Hence, the selection of the number of prediction directions provides a trade off between the encoding time and coding efficiency. The prediction of the target block is conducted by referring the neighboring samples within a frame and the boundary region pixels. The prediction directions have angles measured by the displacement of rows in PU and its reference rows both in the vertical and horizontal directions [3]. As a result, HEVC provides up to 35 distinct directions for each PU blocks in Fig. 1.3 (a). Again image textures have consistent orientation and might cover several neighboring blocks. Hence, it is desirable to analyze the texture feature statistics of the neighboring blocks and use those to improve the intra mode decision. In order to accelerate the encoding process with an acceptable RD cost, it is necessary to make a detailed analysis of CU size decision process.

A CTU represents the basic processing unit in the encoding and consists of both Luma and Chroma components. The recursive splitting of CTU includes a series of non-overlapping coding units (CUs) with variable sizes and further splits into prediction units (PUs). As shown in Fig. 1.1, CTU can either contain a single CU or multiple CUs with size ranging from 64×64 to 8×8. As a result, a single 64×64 CTU can have 85 possible CUs in brute search. Finally, the CU partition which has the minimum RD cost among all combinations of CU partitions will be selected as the optimal CUs for our encoding process. With further split of CU, a typical HEVC reference encoder must check a total of $40 + 41 + 42 + 43 + 44 = 341$ PU blocks to find the optimal combination of CU/PU partitions. Each PU includes 35 modes (Fig. 1.3b) for its angular prediction and carry the information related to prediction



(a): Flowchart for finding Optimal CU on a single CTU



```
/*  A = Tuple of (predicted RDO modes, CU depth information)
    B = CU depth modes varied from 0 to 3
    C = Roughly selected modes from all possible 33 modes for RDO
        estimation, Mode = 0: Planner, Mode = 1: DC                    */
1:  B = 0
2:  if Class = 0 then
3:       C = {9i + 2| i = 0, 1, 2, 3, 4}              // 2,11,20,29 modes
4:  else if Class = 1 then
5:       C = {6i + 2| i = 0, 1...8}               // 2,8,14,20,26,32 modes
6:       if (maxDepth < 3)
8:             B = B + 1
9:  else if Class = 2 then
10:      C = {3i| i + 2 = 1, 2...17}        // 2,5,8,11,14,17,20,23 modes
11:      if (maxDepth < 3)
12:            B = B + 1
13: else
14:      C = {i| i = 1, 2...34}                         // All 33 modes
15:      if (maxDepth < 3)
16:            B = B + 1
17: end if
18: A = B ∪ C
```

(b): Pseudo code for the PU mode selection (PUMS).

Figure 1.4: (a): Flowchart for finding Optimal CU on a single CTU, and 4 (b): Pseudo code for the PU mode selection (PUMS).

processes. Again the intra mode operation in HM software [20] is performed by using all best possible CU/PU sizes and modes to find the ones with the least RD cost. These exhaustive encoding iterations of the Rate-distortion optimization (RDO) process considers the encoding possibilities and compares all of them in terms of bit rate and image quality. Therefore, both the nested structure and encoding process are responsible for the overall increase in computational time and would cause a great burden to encoder. However, if we are able to predict most probable CUs according to the relevant depth information and reduce mode checking range of PUs in advance, then the computational complexity of the intra prediction would be significantly reduced. So our final objective is to find the optimal CUs and the best PU prediction modes at each iteration. Fig. 1.4 (a) shows our proposed algorithm flowchart and the subsequent pseudo-code loop for PU mode selection (PUMS). In our algorithm, the CU depth splits are considered based on four classification scores ranging from 0 to 3.

The video frames are usually of different textures, object shapes and change over time. The homogeneous or static image regions like an open sky or background (Fig. 3 (b) green box) do not contain much image textural information. Hence, further CU depth split is avoided on these static regions and only a few simple directional modes are chosen as optimal for RDO estimation. On the other hand, the dynamic and granular regions contain rich textures. These image regions need further splitting into CU/PUs and find the video contents. Again it is observed that the object motion mostly exists at image edge regions. Hence, an accurate object and texture detection measure is included in those dynamic regions with a subsequent increase in the number of modes. Therefore, an exhaustive CU search is not necessary except at rich image regions. As a result, the whole process saves lot of encoder computation time. In our work, we introduce a CNN based architecture which is able to predict classes based on the region feature statistics. The depth prediction helps in reducing a number of search modes in PU by skipping some superfluous modes. According to pseudocode loop (Fig. 4 (a), (b)), the number of PU prediction directions at class = 0 is reduced to four modes and some mode reduction also followed at other class labels. Fig. 1.3 (b) displays the final CU partitions for 'BasketballPass' video sequence encoded by HEVC HM 16.9 [20]. The block A is a CU of size 32×32 and split into four 16×16 CUs. The sub-CU of Block A (first row and second column) has relatively high texture complexity and further split into smaller blocks. Similarly, the block B has a size of 16×16, which has two sub-CUs (first column) with high complexity and is split into small blocks. From these scenarios, we would find that if there is any texture complexity difference among the sub-CUs, then the current CU is split into smaller CUs. Thus, the complexities of the four sub-CUs are also important during the quad-tree CU size decision process.

CHAPTER 2

PROPOSED CNN NETWORK

2.1 Introduction

We propose a deep CNN architecture which hypothesizes the texture and objects location features using an efficient region proposal technique and then classifies the features using a soft-max classifier (Figs. 2.1, and 2.2). Our end to end process comprises three steps: 1) region proposal network for region predictions, 2) object shape detection, and 3) texture feature merging. Our object detection procedure is inspired by the state-of-the-art detection technique explained in [12], [13] and extracts feature representations from the input images through a series of coarser to finer convolutions. For texture feature calculation, we added a separate order less Fisher Vector pooling operation (i.e. a region based texture descriptor) to our network on top of the convolution features. The object and texture features are later merged at a fully connected (FC) layer. Finally, the output probabilistic scores are classified into four discriminate classes which in turn help to predict the CU depth splitting in our HEVC intra prediction.

Fig. 2.1 gives the overview of our CNN based joint classification model, which includes both texture and object features. The upper half of the Fig. 2.1 describes the region proposal network (RPN) which proposes a set of regions of interest (ROI) for object detection. The RPN network and its training procedure are more elaborately explained in Fig. 2.3. As shown on the lower half of Fig. 2.1, the proposed ROIs from

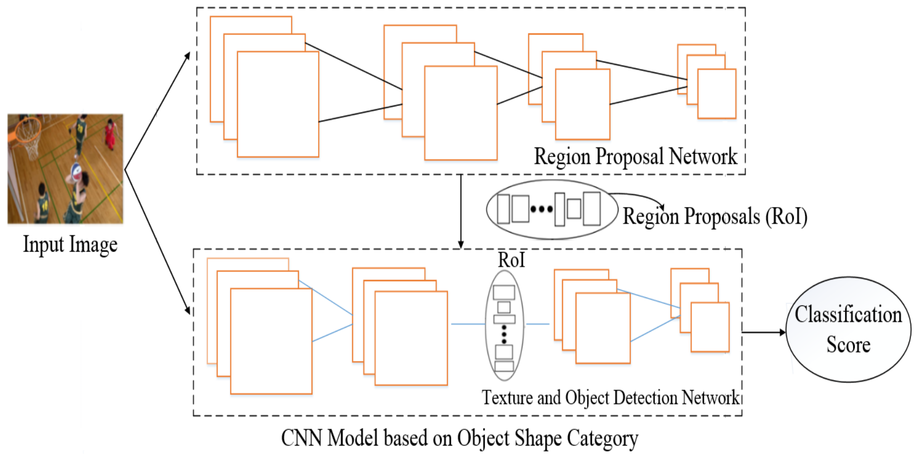


Figure 2.1: Overview of our CNN based joint classification model by exploiting the region category information.

RPN are merged into the main flow of CNN model and classified into four classes. The classification task details are illustrated in Fig. 2.2.

2.1.1 Review of CNN Architecture

For our CNN model design we empirically experimented with the different number of layers, convolution receptive fields (filter kernel size), and a combination of both. Finally, we came up with an architecture which operates jointly in an end to end framework (Fig. 6). Our architecture mimics ZFnet [21] model and includes a scalable architecture with five convolution layer and three fully-connected (FC) layers. Each layer in the network is responsible for a specific task and learns features starting from low-level features (blobs, edges, color pieces) to high- level features (rough object structures) as we move into deeper and deeper convolution layers. The input image to ConvNet is a fixed size of 224×224 and passed through a stack of convolution layers with various kernel sizes as shown in the Fig. 2.2. Multiple convolution layers are stacked together and then followed by a 2×2 max-pool layer with a stride of

2. To restore the spatial resolution, intermediate convolution layer inputs are zero padded. All the hidden layers are equipped with Rectified Linear Unit non-linearity (ReLU, max (0, x)) [22] and applied threshold on the filter responses. The number of parameters, and the memory required for each network layer are summarized in Fig. 2.4. The memory requirements assume that each parameter is of 8 bytes or a double precision floating-point number.

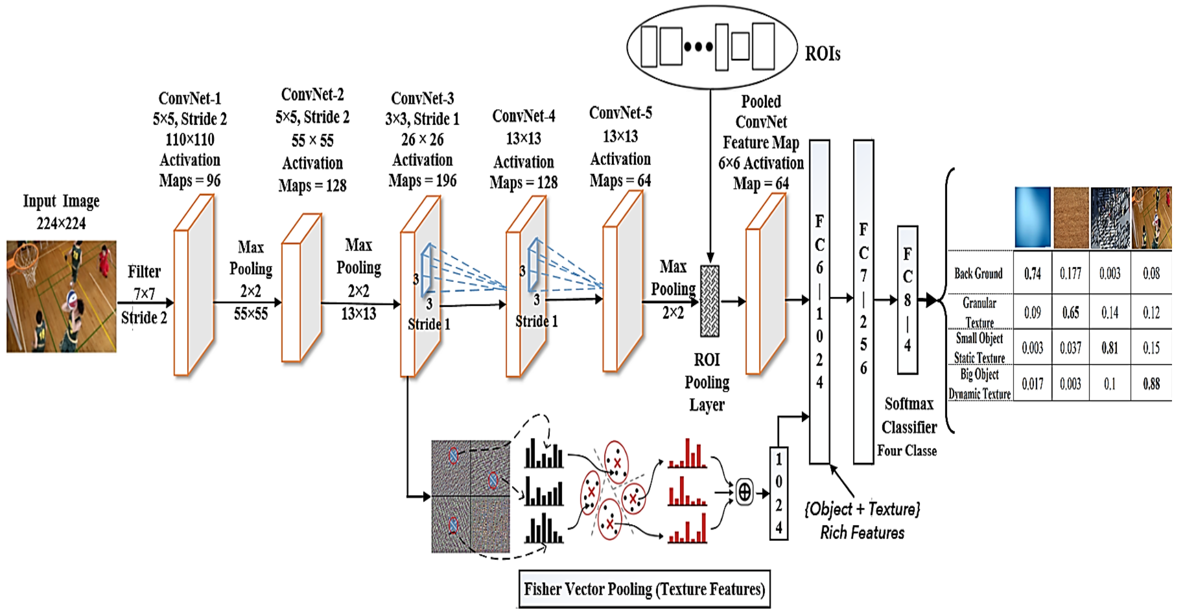

ROIs
Input Image 224×224
Filter 7×7 Stride 2
ConvNet-1 5×5, Stride 2 110×110 Activation Maps = 96
Max Pooling 2×2 55×55
ConvNet-2 5×5, Stride 2 55 × 55 Activation Maps = 128
Max Pooling 2×2 13×13
ConvNet-3 3×3, Stride 1 26 × 26 Activation Maps = 196
3 3 Stride 1
ConvNet-4 13×13 Activation Maps = 128
3 3 Stride 1
ConvNet-5 13×13 Activation Maps = 64
Max Pooling 2×2
ROI Pooling Layer
Pooled ConvNet Feature Map 6×6 Activation Map = 64
FC6 | 1024
FC7 | 256
FC8 | 4
Softmax Classifier Four Classe
Back Ground 0.74 0.177 0.003 0.08
Granular Texture 0.09 0.65 0.14 0.12
Small Object Static Texture 0.003 0.037 0.81 0.15
Big Object Dynamic Texture 0.017 0.003 0.1 0.88
1024
{Object + Texture} Rich Features
Fisher Vector Pooling (Texture Features)

Figure 2.2: Our Deep CNN architecture with Object detection and Texture classification.

During the training process, the network follows a forward path which stores the gradients w.r.t. to its parameter. The stored values are then used in the back propagation path. As a result, the memory requirement is doubled as shown in Fig. 2.4 which is sufficient for our GPU machine with 12 GB on board.

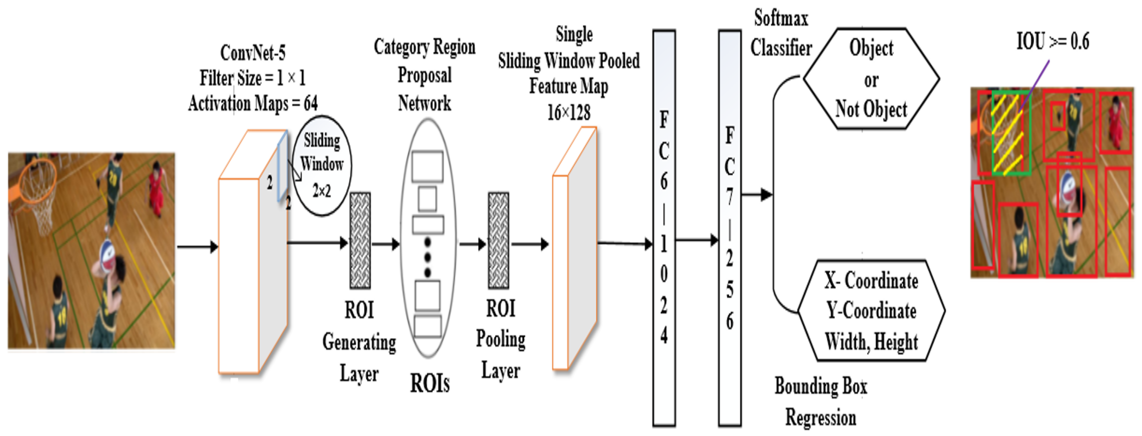
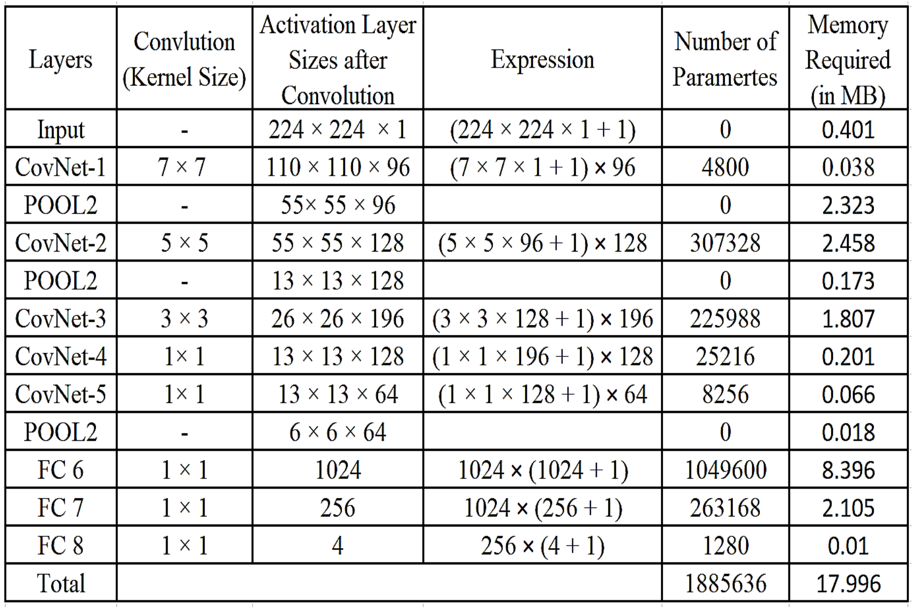


Figure 2.3: Region Proposal Network with Bounding Box Regression and object classification using BasketballDrill sequence.

2.1.2 Object Detection

The first step of the object shape detection network is to identify the regions of interest (ROI) from the last convolution layer. In our implementation, the output of ConvNet5 feature map forms the basis for the ROI network. These ROIs serves as an attention model and proposes the possible potential object locations, which are examined in the final step for object detection. In our implementation, we consider ROI as the rectangular window and defined by a tuple(x, y, h, w), where (x, y) specifies its bottom-left corner and (h, w) its height and width. Fig. 2.3 illustrates the region proposal network which simultaneously predicts object bounds and region wise object-ness score at each position.

Our region proposal network (RPN) takes an input image and generates a dense grid of rectangle anchor regions with specified size and aspect ratio. Looking into the HEVC CTU depth partition structure we consider a range of anchor box scales from 16×16, 32×32, 64×64 to 128×128 (Fig. 2.5). Each anchor box is accommodated



Layers	Convlution (Kernel Size)	Activation Layer Sizes after Convolution	Expression	Number of Paramertes	Memory Required (in MB)
Input	-	224 × 224 × 1	(224 × 224 × 1 + 1)	0	0.401
CovNet-1	7 × 7	110 × 110 × 96	(7 × 7 × 1 + 1) × 96	4800	0.038
POOL2	-	55× 55 × 96		0	2.323
CovNet-2	5 × 5	55 × 55 × 128	(5 × 5 × 96 + 1) × 128	307328	2.458
POOL2	-	13 × 13 × 128		0	0.173
CovNet-3	3 × 3	26 × 26 × 196	(3 × 3 × 128 + 1) × 196	225988	1.807
CovNet-4	1× 1	13 × 13 × 128	(1 × 1 × 196 + 1) × 128	25216	0.201
CovNet-5	1× 1	13 × 13 × 64	(1 × 1 × 128 + 1) × 64	8256	0.066
POOL2	-	6 × 6 × 64		0	0.018
FC 6	1 × 1	1024	1024 × (1024 + 1)	1049600	8.396
FC 7	1 × 1	256	1024 × (256 + 1)	263168	2.105
FC 8	1 × 1	4	256 × (4 + 1)	1280	0.01
Total				1885636	17.996

Figure 2.4: Analysis of Network Parameters in CNN

with four aspect ratios (0.5, 0.5), (1, 2), (2, 1), (1, 1). To generate the region proposals a 2×2 sliding window is used over all positions on ConvNet5 feature maps, with the center coinciding with the center of proposed regions. Each window is then mapped to a 128-d dimensional vector which comprises 64 positive anchors and 64 negative anchors. The above 128 vectors are then connected to FC6 and FC7 layers. The last FC7 layer is then fed into 1) a box classification layer with two probability scores and 2) a bounding box regression layer with four coordinates. The above process is repeated for 16 times (4 box scales 4 aspect ratios) at a single position and continued for all 20×30 ConvNet-5 features locations. Overall our prediction process generates $20 \times 30\ \times 16$ possible anchor boxes. The total number of anchor boxes are reduced to 1000 by using non-maxima suppression technique and ignore the boxes which cross the image boundary. The anchor box is a mechanism to judge an image region whether it contains an object or not. For our RPN training, we assign a binary class label to each anchor. A positive label is assigned to an anchor box that has an intersection over union (IoU) overlap (w.r.t its ground truth) higher than 0.6. Similarly, for an anchor box whose IoU ratio lower than 0.3 is assigned with a negative label. Anchor boxes which are neither positive nor negative (e.g. background) do not contribute to our training (Fig. 2.5). During our training time the above hyper-parameter values are tuned to best fit the different types of objects, which we are trying to find.

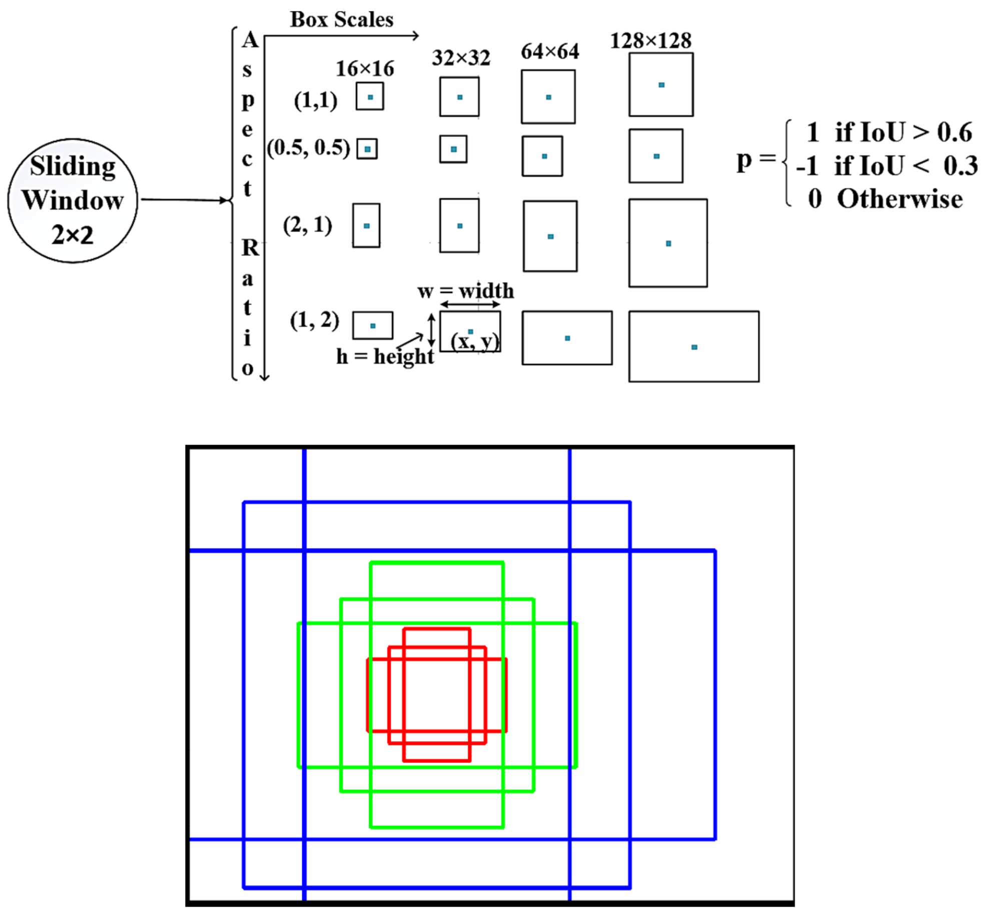


Figure 2.5: Generated 16 anchor boxes for each sliding window on the ConvNet5 feature map.

2.1.3 Texture Feature Calculation

Textures have an important role in characterizing many natural objects, particularly for those objects that best qualifies a pattern such as a granularity, static and dynamicity as illustrated in Fig. 2. Fisher Vector (FV) has been widely used to aggregate the local descriptors of an image into a global representation in large-scale image retrieval [23], [24]. The FV pools the local features densely within the described regions, and is therefore more apt at describing textures. Traditionally the convolution layers are regarded as filter banks and considered the best for feature learning [11], [25]. So on the top of our ConvNet3 layer, we build representations using the FV pooling which is commonly done in the bag-of-words approach. Our pooling operation is order less, single scale i.e. 32×32, and hence suitable for the textures [8]. We extracted 32×32 patches sampled with a stride of 16 pixels. The above 32×32 patch size is considered, as it is reasonable for the CU size. For our texture analysis we calculate the texture descriptors based on FV's, over the ConvNet-3 (Fig. 6) and collected the low level image cues. The FV pools the local features densely within the described regions and are therefore apt at describing the image textures. Our network results in 256 dimensional local features for Fisher vector computation and pools into a representation with 32 Gaussian components. Finally, it results in 16K dimensional descriptors which are much higher than the FC6 layer (1024-d) and is highly redundant. So the FV dimension is projected to 1024-d by using the principal component analysis (PCA) technique. After above dimension reduction the effective dimension of FC and FV are comparable and are merged at FC6 layer.

Let $X_i = \{x_{ij}\}_{j=1}^{m_i}$ to be a set of local SIFT descriptors (e.g. scale-invariant feature transform descriptor) [26] [29] extracted from an image, where $x_{ij} \in R^D$, and m_i is the number of local descriptors. We first revisit the traditional FV $\phi(\mathrm{X}_i)$, which encodes a set of local descriptors X_i extracted from an image by fitting a K - component Gaussian Mixture Model (GMM) $u_\lambda(\mathrm{x}) = \sum_{k=1}^{K} \mathrm{w}_k^G\ u_k(\mathrm{x})$ to the local descriptors. FV encodes the derivatives of loglikelihood with respect to its parameter set [23], denoted as $\lambda = \{\mathrm{w}_k^G,\ \mu_k(\mathrm{x}),\ \sum_k, k = 1, 2...K\}, where\ \mathrm{w}_k^G \in R,\ \mu_k \in R^{D\times1},\ \sum_k \in R^{D\times D}$. The $w_k^G,\ \mu_k,\ \sum_k$ are the mixture weight, mean vector and co-variance matrix of the GMM model, and $\mu_k =$ diag (σ_k^2), $\sigma_k \in R^{D\times1}$. We assume that the co-variance matrices are diagonal, and denoted by the variance vector σ_k^2. In particular, the proposed Fisher Layer makes two simplifications to the original FV: 1) all GMM components have equivalent weights; 2) co-variance matrices of all the Gaussian components share the same determinant so that the k^{th} Gaussian distribution $u_k(x_{ij})$ can be written as:

$$\mathrm{u}_k(x_{ij}) = \tfrac{1}{(2\pi)^{D/2}}\ exp\{\tfrac{1}{2}(\mathrm{x}_{ij}\ \text{-}\mu_k)^T \textstyle\sum_k^{-1}(\mathrm{x}_{ij}\ \text{-}\mu_k)\} \tag{2.1}$$

The gradients of a single descriptor x_{ij} w.r.t the parameters u_k and σ_k of the simplified GMM can be written as:

$$G_{\mu_k}^{x_{ij}} = \gamma_j(k)[w_k^G \odot (x_{ij} + b_k)] \tag{2.2}$$

$$G_{\mu_k}^{x_{ij}} = \gamma_j(k)[(w_k^G \odot (x_{ij} + b_k))^2 - 1] \tag{2.3}$$

Where $\odot$ is an element-wise product operation, $w_k^G = \frac{1}{\sigma_k}$, $b_k = -u_k$. w_k^G and b_k are sets of learn-able parameters for each Gaussian component k. γ_j(k) is the posterior probability, and can be written as follows:

$$\begin{aligned} \gamma_j(k) &= \frac{u_k(x_{ij})}{\sum_{n=1}^{K} u_k(x_{ij})} \\ = \frac{exp\{-\frac{1}{2}(w_k^G \odot (x_{ij} + b_k)^T)(w_k^G \odot (x_{ij} + b_k)\}}{\sum_{n=1}^{K} exp\{-\frac{1}{2}(w_k^G \odot (x_{ij} + b_k)^T)(w_k^G \odot (x_{ij} + b_k)\}} \end{aligned} \tag{2.4}$$

For any single local descriptor x_{ij}, itś output Fisher layer can be denoted as:

$$\phi(x_{ij}) = [G_{\mu_1}^{x_{ij}T}, ..., G_{\mu_k}^{x_{ij}T}; G_{\mu_1}^{\gamma_{ij}T}, ..., G_{\gamma_k}^{x_{ij}T}] \tag{2.5}$$

As a result, the final FV $\phi(\mathrm{X}_i)$ of the local feature set X_i from an image is the mean-pooling of all local representations in X_i, i.e. $\phi(\mathrm{X}_i) = \frac{1}{m_i}\sum_{j=1}^{m_i}\phi(x_{ij})$. All the operations in Fisher layer are differentiable and the parameters w_k and b_k can be derived via back-propagation. Before using the representation into a linear model, the aggregated descriptor $\phi(\mathrm{X}_i)$ is divided by its norm $\parallel \phi(\mathrm{X}_i) \parallel^2$ i.e. L2 normalized as described in [23].

2.1.4 Loss Function

A multi-objective loss function is employed to train our region proposal network (RPN) for both feature classification and bounding box regression. The loss function is given below:

$$\mathrm{L}(\mathrm{p}_i, {p_i}^*, t_i, {t_i}^*, \theta) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, {p_i}^*, \theta) + \lambda_1 \frac{1}{N_{loc}}\sum_i {p_i}^* \times L_{cls}(t_i, {t_i}^*, \theta) + \lambda_2 L_{FV_i} \tag{2.6}$$

p_i = Probability (predicted) of an anchor i being an object.

${p_i}^*$ = Ground truth label 1 for +ve anchor or 0 for −ve anchor

N_{cls} = Number of anchors in mini batch of 128

N_{loc} = Number of total anchors i.e. 1000

λ_1 and λ_2 = Regularization parameters

i = Index of an anchor box in a mini batch

θ = Represents the network weight and bias parameters

L_{cls} = Classification loss

L_{loc} = Bounding box regression loss

L_{FV} = L2 normalization on Fisher vector space

t_i is a 4d vector representing the four coordinates of a predicted bounding box and ${t_i}^*$ is that of the ground truth box, associated with a positive anchor. A simple bounding box regression is adapted such that t_i and ${t_i}^*$ become close to each other and improves localization performance after a couple of iterations. The classification loss L_{cls} is the cross-entropy loss $-\{logp_{i_{p^*_{i,\theta}}}\}$ which is determined over two classes (object or not object). The term L_{loc} is defined as the regression loss over a tuple of true bounding-box regression target for class ${p_i}^*$ with co-ordinate $t_i(x, y, w, h)$, and a predicted tuple ${t_i}^*(x', y', w', h')$. We used the L1 smoothing function, as defined in [12], and calculated the loss between the predicted bounding anchor box and the ground truth. Our final bounding box regression L_{loc} loss is defined below:

$$L(t_i, {t_i}^*, \theta) = \sum_{i \in \{x,y,w,h\}} L_1({t_i}^* - t_i) \text{ in which}$$

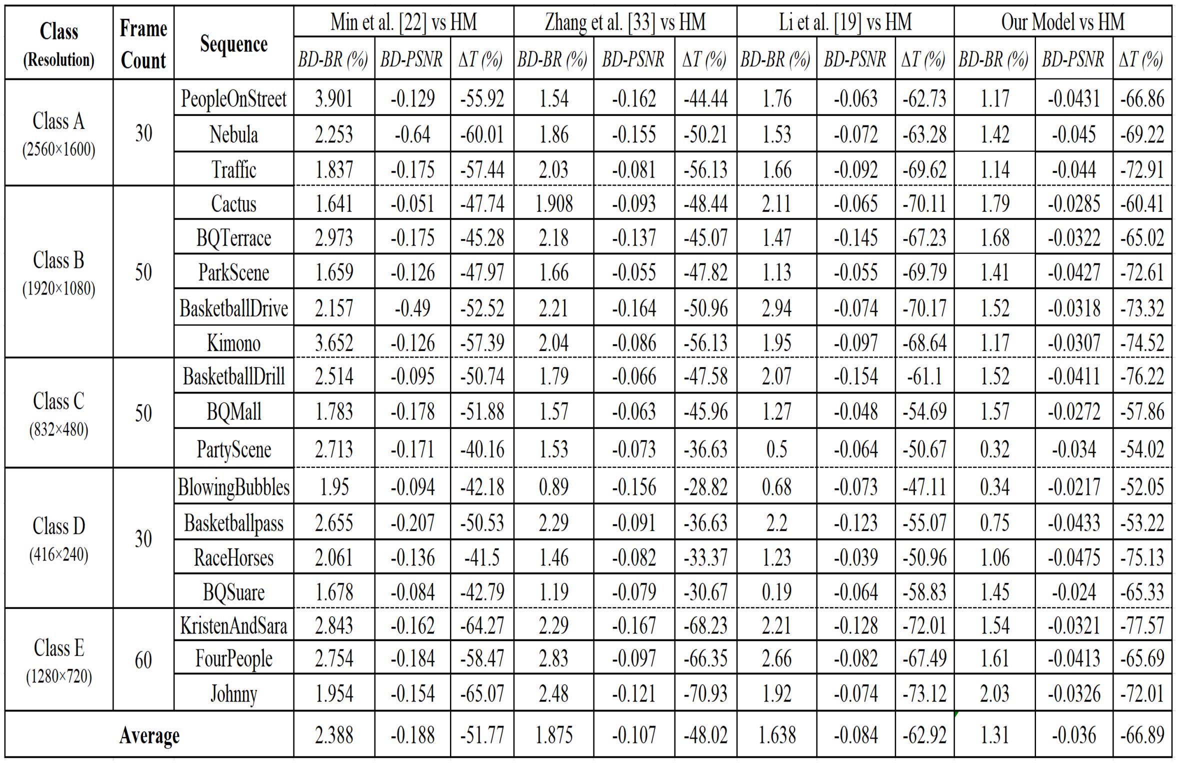
$$L_1(x) = 0.5\ x^2\ if\ |x| < 1\ , else\ |x| - 0.5$$

The regression loss term ${p_i}^* \times L_{Loc}$ is only activated for positive anchors (${p_i}^* = 1$) and disabled at negative anchors. The third term L_{FV} is the L2 normalization on Fisher vector space. We chose the L2 norm because it is the natural norm associated with the dot-product. So our final loss function L in equation (1) consists of p_i, t_i, θ and ϕ terms only. In our implementation, the two terms L_{cls} and L_{loc} are normalized by N_{cls} and N_{loc} respectively. λ_1 and λ_2 are the regularization parameters to balance the three losses in eq. 2.1. At the region generation steps, we set the regularization values $\lambda_1 = 7$ and $\lambda_2 = 0.001$, which implies that we are biased towards better box locations. After RPN training we have thousand proposed bounding boxes all over input image as shown in Fig. 2.3. Our objective is also to detect both small and big objects present on frames. The small object detection is quite challenging for the proposed regions like 8×8 or 16×16. To overcome this we consider a bigger context region i.e. 32×32, 64×64 anchor box over a small object size like 8×8 and followed the technique explained in ContextNet [27]. So our proposed rectangular region justifies the above context based technique for small object detection too. Our RPN + CNN network is able to achieve detection accuracy up to 65.28% mean Average Precision (mAP) over the above proposed regions per image. At test time our model is able to generate region proposals in forty-two milliseconds and achieves classification with near real-time computation.

Table 2.1: Summary of Six step CNN training Process

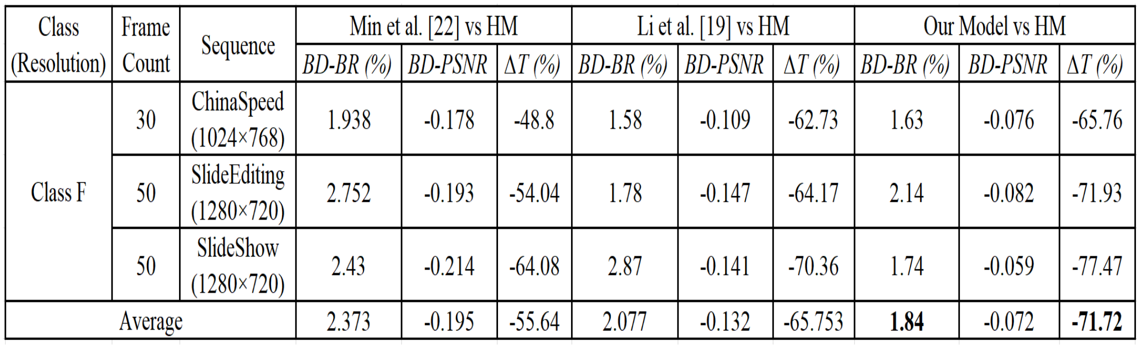
Step 1: Pre-train a Deep CNN model (ZFnet) for initializing basic layers in Step 2 and Step 3.
Step 2: Train CNN for the region proposal generation
Step 3: Train CNN for object shape detection using region proposals obtained from Step 2.
Step 4: Fine-tune CNN for region proposal generation, by sharing CNN feature layer weights trained in Step 3.
Step 5: Fine-tune CNN for region wise object and texture detection using region proposals obtained from Step 4, with feature layers fixed.
Step 6: Output the unified CNN classification as the final single model, by training jointly in Step 4 and Step 5.

On the final classification step, an end to end training is performed by using the predicted bounding box regions from the first stage (Fig. 2.3). The image regions (CTU/CU) are then classified into four classes by using the Softmax classifier. For our model training, a six-step process is included for the joint optimization and detailed steps are summarized in Table 2.1 . In steps 2 and 3, the object proposal and detection networks are separately trained. After fined tuning steps 4, and 5, both the networks shared their parameters for CNN feature extraction. Finally, two separate networks are combined into a unified network and perform our classification task as seen in Fig. 2.2 and step 6 of Table 2.1. All the training hyper-parameters i.e. IoU, anchor box scale, aspect ratio, regularization parameters step size, and learning rate values are included in section IV and V.



Class (Resolution)	**Frame Count**	**Sequence**	Min et al. [22] vs HM			Zhang et al. [33] vs HM			Li et al. [19] vs HM			Our Model vs HM		
			BD-BR (%)	*BD-PSNR*	*ΔT (%)*	*BD-BR (%)*	*BD-PSNR*	*ΔT (%)*	*BD-BR (%)*	*BD-PSNR*	*ΔT (%)*	*BD-BR (%)*	*BD-PSNR*	*ΔT (%)*
Class A (2560×1600)	30	PeopleOnStreet	3.901	-0.129	-55.92	1.54	-0.162	-44.44	1.76	-0.063	-62.73	1.17	-0.0431	-66.86
		Nebula	2.253	-0.64	-60.01	1.86	-0.155	-50.21	1.53	-0.072	-63.28	1.42	-0.045	-69.22
		Traffic	1.837	-0.175	-57.44	2.03	-0.081	-56.13	1.66	-0.092	-69.62	1.14	-0.044	-72.91
Class B (1920×1080)	50	Cactus	1.641	-0.051	-47.74	1.908	-0.093	-48.44	2.11	-0.065	-70.11	1.79	-0.0285	-60.41
		BQTerrace	2.973	-0.175	-45.28	2.18	-0.137	-45.07	1.47	-0.145	-67.23	1.68	-0.0322	-65.02
		ParkScene	1.659	-0.126	-47.97	1.66	-0.055	-47.82	1.13	-0.055	-69.79	1.41	-0.0427	-72.61
		BasketballDrive	2.157	-0.49	-52.52	2.21	-0.164	-50.96	2.94	-0.074	-70.17	1.52	-0.0318	-73.32
		Kimono	3.652	-0.126	-57.39	2.04	-0.086	-56.13	1.95	-0.097	-68.64	1.17	-0.0307	-74.52
Class C (832×480)	50	BasketballDrill	2.514	-0.095	-50.74	1.79	-0.066	-47.58	2.07	-0.154	-61.1	1.52	-0.0411	-76.22
		BQMall	1.783	-0.178	-51.88	1.57	-0.063	-45.96	1.27	-0.048	-54.69	1.57	-0.0272	-57.86
		PartyScene	2.713	-0.171	-40.16	1.53	-0.073	-36.63	0.5	-0.064	-50.67	0.32	-0.034	-54.02
Class D (416×240)	30	BlowingBubbles	1.95	-0.094	-42.18	0.89	-0.156	-28.82	0.68	-0.073	-47.11	0.34	-0.0217	-52.05
		Basketballpass	2.655	-0.207	-50.53	2.29	-0.091	-36.63	2.2	-0.123	-55.07	0.75	-0.0433	-53.22
		RaceHorses	2.061	-0.136	-41.5	1.46	-0.082	-33.37	1.23	-0.039	-50.96	1.06	-0.0475	-75.13
		BQSuare	1.678	-0.084	-42.79	1.19	-0.079	-30.67	0.19	-0.064	-58.83	1.45	-0.024	-65.33
Class E (1280×720)	60	KristenAndSara	2.843	-0.162	-64.27	2.29	-0.167	-68.23	2.21	-0.128	-72.01	1.54	-0.0321	-77.57
		FourPeople	2.754	-0.184	-58.47	2.83	-0.097	-66.35	2.66	-0.082	-67.49	1.61	-0.0413	-65.69
		Johnny	1.954	-0.154	-65.07	2.48	-0.121	-70.93	1.92	-0.074	-73.12	2.03	-0.0326	-72.01
Average			2.388	-0.188	-51.77	1.875	-0.107	-48.02	1.638	-0.084	-62.92	1.31	-0.036	-66.89

Figure 2.6: BD-BR, BD-PSNR (dB), Time Reduction T (%) for our Proposed Algorithm on HM16.9 Main Profile Video Sequences and other state-of-the-art Models at QP = 22, 27, 32, 37



Class (Resolution)	Frame Count	Sequence	Min et al. [22] vs HM			Li et al. [19] vs HM			Our Model vs HM		
			BD-BR (%)	*BD-PSNR*	*ΔT (%)*	*BD-BR (%)*	*BD-PSNR*	*ΔT (%)*	*BD-BR (%)*	*BD-PSNR*	*ΔT (%)*
Class F	30	ChinaSpeed (1024×768)	1.938	-0.178	-48.8	1.58	-0.109	-62.73	1.63	-0.076	-65.76
	50	SlideEditing (1280×720)	2.752	-0.193	-54.04	1.78	-0.147	-64.17	2.14	-0.082	-71.93
	50	SlideShow (1280×720)	2.43	-0.214	-64.08	2.87	-0.141	-70.36	1.74	-0.059	-77.47
Average			2.373	-0.195	-55.64	2.077	-0.132	-65.753	**1.84**	-0.072	**-71.72**

Figure 2.7: Table IV: BD-BR, BD-PSNR (dB) and T (%) for our Proposed Algorithm on HM16.9, Class F videos at QP =22, 27, 32, 37

2.2 CNN NETWORK TRAINING

For our CNN training we randomly select 20K images from PASCAL VOC 2007 object data sets [28], texture data sets like DynTex (comprehensive database of Dynamic Textures) [29] , and DTD (describable textures dataset) [30]. The ground truth images are collected by wrapping into various image regions. For our experiment we consider only the luminance Y frames and extracted patches of size 224×224. As our model mimics the ZFnet, we try to maintain the input patch size as 224×224 for our better network simplification. Again we know from the paper [2] , that the maximal CTU sizes in HEVC can go up to 64×64. Initially we have a plan to introduce an image pyramid [31] as an input to CNN which can accommodate various CTU shapes. But we convinced that the above pyramid because of network complexity and moved it as a future study. As a result the entire CTU is feed into to our CNN trained model. Though our patch size is larger than the current CTU size, but it helps training our deeper model and gives a simple model as compared to recent CNN models explained in [1] and [32]. We believe that our bigger patch size might accommodate bigger CTU/CU sizes on the future video coding standard. To use the data set efficiently we adopt three data augmentation techniques 1) horizontal flipping 2) scaling by a factor of 0.7, 0.5, 0.4, and 3) rotation by degrees of 90, 180 and 270. Then our training set is multiplied into $2 \times 4 \times 4 = 32$ times to that of the original data set. The objective loss functions L (θ) on equation (1) is minimized using the stochastic gradient descent with standard back propagation [33].

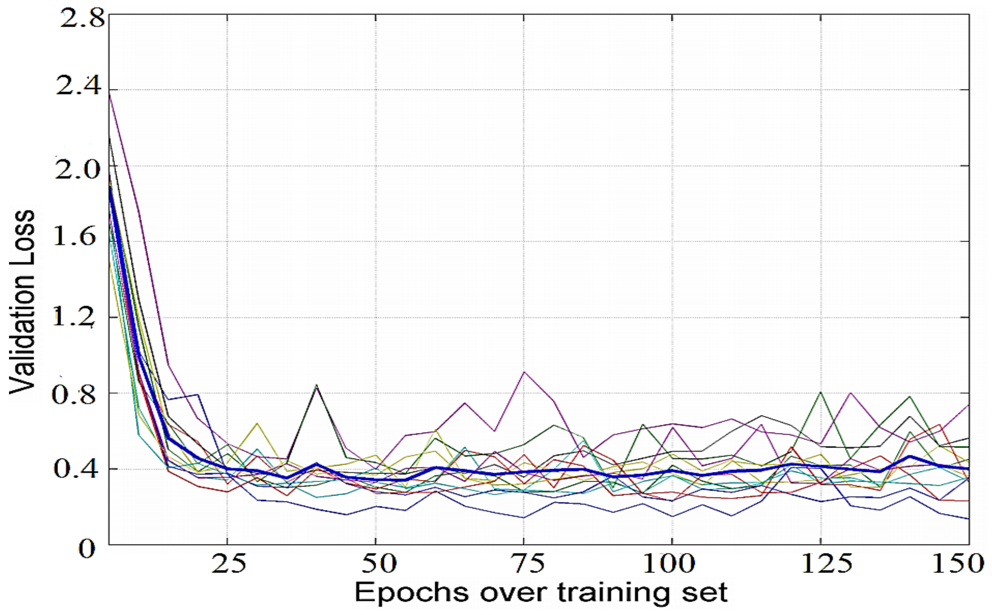


Figure 2.8: A plot of training data Epochs and validation loss over all cross validation folds.

The layer weight parameters are initialized from a zero-mean Gaussian distribution with 0.05 standard deviation. The weight matrices are updated in terms of learning rate and weight gradients. The momentum parameter was set to 0.99 and weight decay to 0.001 [34]. Our implementation is derived from publicly available python/C++ based Tensorflow framework [35] and the training is performed on an NVIDIA K40 GPU machine. The multi-core training operation is carried out by splitting the training images into batches and parallelly process the task on each core. After the batch gradients are computed at each core, they are averaged to obtain the gradient of a full batch. The mini batch sizes are set to 128 anchors, which are extracted from two images and contain both positive and negative anchors as shown in [12]. The mode parameters are converged after around 1100 iterations with twenty-four epochs. The entire model training process took around 22 hours on our GPU system. Fig. 2.8 shows the validation loss along with number of epochs over the training data set. The thick blue line is the average loss over all the cross validation folds. The trained files along with different quantization parameter (QP) parameters are compiled and integrated on a locally installed HM software. For our implementation we modified the "getPartitionSize", "getPredictionMode" and "get-Depth" procedures in "TEncSlice.cpp", "TEncSearch.cpp", "TEncCU.cpp" modules and respective header files on HM software 16.9 [20]. During the testing phase, all the encoder parameter changes occurred at the "encoder_infra_main·cfgȷconfiguration file.

2.3 RESULTS

To evaluate our algorithm performance we compared our model output and three other benchmark models on the HEVC HM16.9 reference software [36]. All the experiments are performed on a desktop computer (Windows operating systems) with Intel I7 2.99GHz processor and NVIDIA Tesla K40 GPU @875 MHz graphics card with 12 GB memory, CUDA version 8.0. The experimental results for various test sequences are presented in this section. Our proposed region wise feature classification algorithm was compared with the recent CNN method by T. Li et al [1], and the heuristic methods Zhang et al [17] and Min et al [8]. Our experiment strictly followed the common test conditions defined in [37] and HM [20] software.

To verify the performance of our algorithm we tested with the standard video sequences of class A, B, C (WVGA), D (WQVGA), E, and separately with the Class F videos. A group of experiments were carried out with QP = 22, 27, 32 and 37 on the above recommended video sequences and the results are shown in Tables 2.6, and 2.7. The BD-BR and Bjøntegaard delta peak signal-to-noise ratio (BD-PSNR) are applied to evaluate the RD performance of different schemes and was compared with the original HM output. The encoder time- saving ΔT is calculated to measure the time reduction of the tested methods. The coding efficiency loss was measured by using BD-BR [10] and the encoder time saving was derived by,

$$\Delta T = \frac{1}{4}\sum_{i=1}^{4} \frac{T_{prep}(QP_i) - T_{org}(QP_i)}{T_{org}(QP_i)} * 100\%$$

where T_{org} denotes the encoding time consumed by original HM and T_{prop} denotes the algorithms considered in our simulation. As shown in Table 2.6, our algorithm was able to achieve on an average of 66.89% encoder time saving along all the intra encoding experiments. Since we focused on intra coding performance (AI mode), the experiments were carried out for all intra I-frame sequences. In our simulation, we included two phases of coding performance evaluation. One was to make the comparison between the coding performances of our proposed algorithm and the state-of-the-art algorithms. The other was the number of mode reductions comparison between the proposed algorithm and the HEVC baseline.

2.3.1 Evaluation of Encoding Time Complexity Reduction

The BD-BR, BD-PSNR, and encoding time complexity reduction (ΔT) results of our algorithm are tabulated in Tables 2.6, 2.7 and successively compared with Min et al. [8], Zhang et al. [17], and Li et al. [1] methods. It is observed that our deep CNN based approach saved more encoding time at four given different QPs i.e QP = 22, 27, 32, 37. On an average our deep learning approach (66.15%) outperformed other three approaches, (62.92 %) for [1], (-51.77 %) for [8], and (-48.02%) for [17] in terms of encoding time complexity reduction. We noticed that our CNN approach consumed less time than that of shallow CNN Li et al. [1] model. It might be reason that, the Li et al. [12] requires RDO search for the decision by splitting CTU from 64×64 to 16×16, and learned three separate CNN models for obtaining the classification (three scales). We further found that the gap of time-savings between our and other heuristic approaches become larger at given QPs. Perhaps it is due to rich feature learning at various image regions and efficient classification task of our CNN model. In addition, we can see from Table 2.7 that our algorithm was able to reduce the average encoding time as compared to [1] and [17] on all the class F video data sets. The number of modes evaluated on our RDO process are analyzed and the results are displayed in Table 2.8. It is observed that an average of four modes are evaluated in our RDO process as compared to seven modes in HEVC i.e. reduced up to 45.91% modes.

In our simulation, we also included Class F video sequences which are called the non-camera capture uncompressed source video sequences. Even though the encoding time complexity reduction is still retained on the Class F sequences, but observed that the increase of BD-BR on class F (1.84%) was larger than the conventional class A-E video sequences (1.34%) for all intra coding (Table 2.6 and 2.7). This was probably due to different content distributions between Class F video and other classes. We will address the above subjects on our next research exploration and will come up with an updated algorithm with better encoding time.

2.3.2 Computational Complexity of CNN

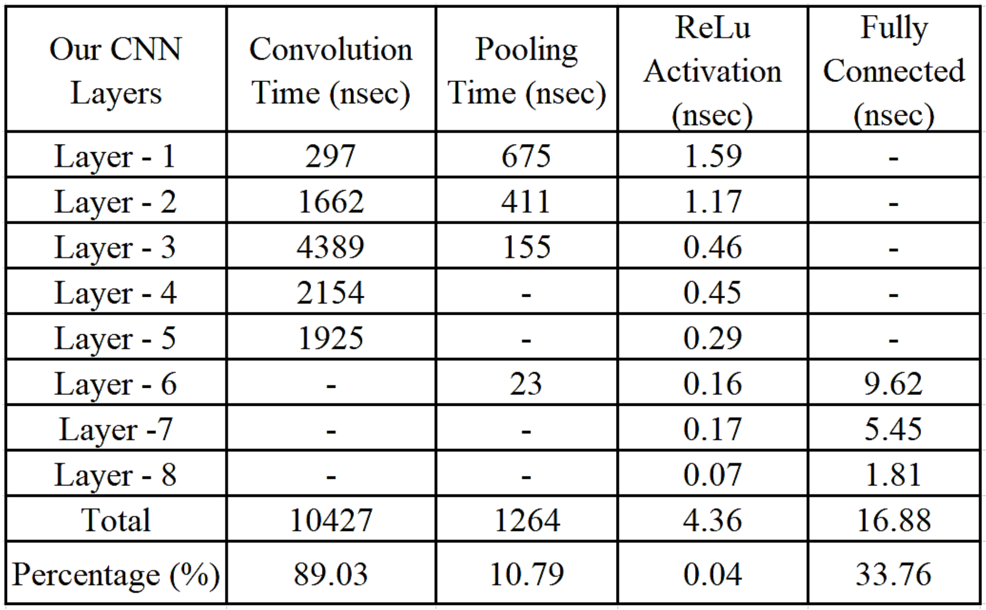
The recent success of CNN models in multimedia applications, promises to improve the effective algorithm implementation and reduces the computational cost. One direction is to 1) improve the CNN algorithm is by using GPUs and field-programmable gate arrays (FPGAs) as in [38], [39], and [40], and the other direction is to reduce the number of basic mathematical operations needed in CNN layer computations. Here, we analyze the time complexity of CNN by counting number of floating-point operations which include only multiplications and additions. So we investigate the linear algebraic properties in CNN computations without analyzing the hardware cost.

The CNN layers mainly comprise three steps: convolution, pooling, and activation rectified linear unit (ReLU). The convolution is simply multiplication and accumulation with additive operations, and depends on six factors: 1) Size of the filter or kernel ($X \times Y$), 2) Number of filters (N), 3) Size of the input feature map ($F_X \times F_Y$), 4) Number of input feature maps (n), 5) Number of channels per input feature map (C), 6) Stride i.e. S (step by which the filter slides over input feature map). As explained in [41], the number of multiplication and addition operations in a convolution stage can be calculated as:

$$(F_x - X + S)/S \times (F_y - Y + S)/S \times C \times N \times n \tag{2.7}$$

We calculated the execution time of convolution steps in detail and understand how the overall execution time is affected. A breakdown of run time is given in Table

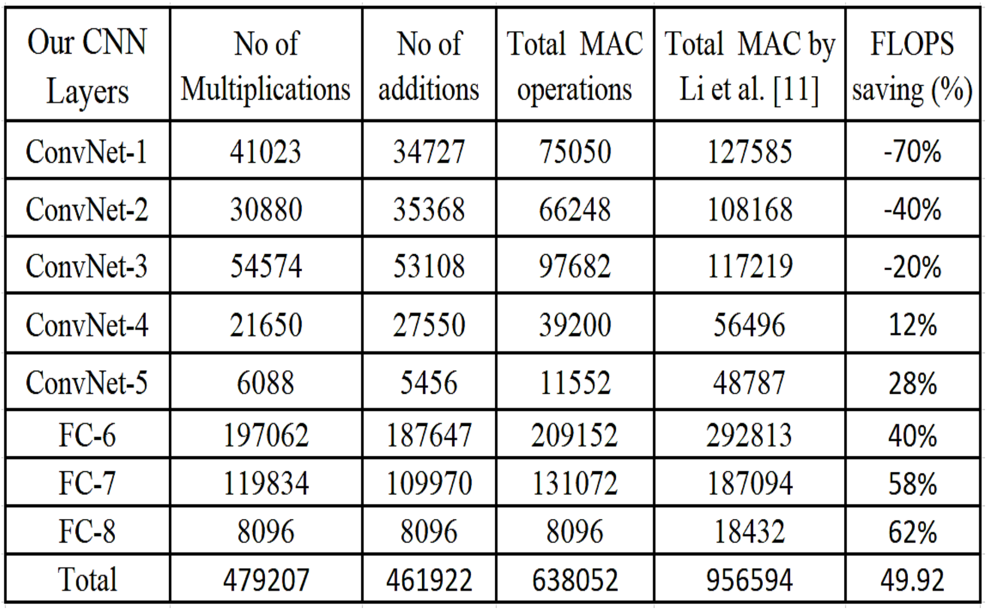
2.10. We also observe that the overall run time is dominated by convolution operations and accounts approximately 90% of the total computation time in a given layer and is liable for most of the execution time.



Our CNN Layers	Convolution Time (nsec)	Pooling Time (nsec)	ReLu Activation (nsec)	Fully Connected (nsec)
Layer - 1	297	675	1.59	-
Layer - 2	1662	411	1.17	-
Layer - 3	4389	155	0.46	-
Layer - 4	2154	-	0.45	-
Layer - 5	1925	-	0.29	-
Layer - 6	-	23	0.16	9.62
Layer -7	-	-	0.17	5.45
Layer - 8	-	-	0.07	1.81
Total	10427	1264	4.36	16.88
Percentage (%)	89.03	10.79	0.04	33.76

Figure 2.10: Breakdown of CNN run time in nanosecond, for feature classification of image size = 224×224

To understand the time complexity of our model we counted the number of floating-point operations across each layer, which include the number of additions and multiplications. The Table VII reports the numbers of floating-point operations for each layer of CNN and performs a total of around 0.63×10^6 floating-point operations, including 453,858 additions and 471,143 multiplications. To better understand the total encoding time, we implemented CNN in a GPU mode. In our implementation we used Intel I7 desktop which holds NVidia K40 GPU inside. The K40 GPU full performance is double-precision with 1.43 Tera Floating point operations per second (TFLOPS). But the floating point operations in our CNN convolution stages need double precision (FP 64). Again each multiplication or addition operation is considered as a FLOP on its own. Since each convolution is simply a multiplication and accumulation operation, now it takes two FLOPs. We followed the convolution specification in Fig. 2.4 and calculated total number of MAC operations in our model by using the Eq. 2.7. As shown on Table 2.10, the model requires 638K MACs to process a 224×224 image patch. As a result, the time taken by K40 GPU machine is $t = \frac{2\times638\times10^3}{10\times10^1 2} = 126.8$ nano-seconds to execute the convolution stages. We compare our model performance (run time) with Li et al model [1] and it is observed that our CNN model performs fewer floating-point operations by at least two orders of magnitude.



Our CNN Layers	No of Multiplications	No of additions	Total MAC operations	Total MAC by Li et al. [11]	FLOPS saving (%)
ConvNet-1	41023	34727	75050	127585	-70%
ConvNet-2	30880	35368	66248	108168	-40%
ConvNet-3	54574	53108	97682	117219	-20%
ConvNet-4	21650	27550	39200	56496	12%
ConvNet-5	6088	5456	11552	48787	28%
FC-6	197062	187647	209152	292813	40%
FC-7	119834	109970	131072	187094	58%
FC-8	8096	8096	8096	18432	62%
Total	479207	461922	638052	956594	49.92

Figure 2.11: Number of multiplication and accumulation (MAC) operations in CNN layers for a single image patch of 224×224

A comparison of our CNN model with the Li et al model [1] is performed in terms of GFLOPS savings and is shown in Fig. 2.12. We used different convolution kernel sizes and feature maps in Fig. 6, Table-I and calculated % FLOP saving in each convolution layers. From Fig. 2.12 it is observed that the Li et al model [1] may not bring benefits on higher matrix size (activation map), but could lead to more than 100% overhead, especially when the size of the matrix is small. But in our model setting a greater benefit is achieved due to the element-wise multiplications and redefined granularities of matrix elements.

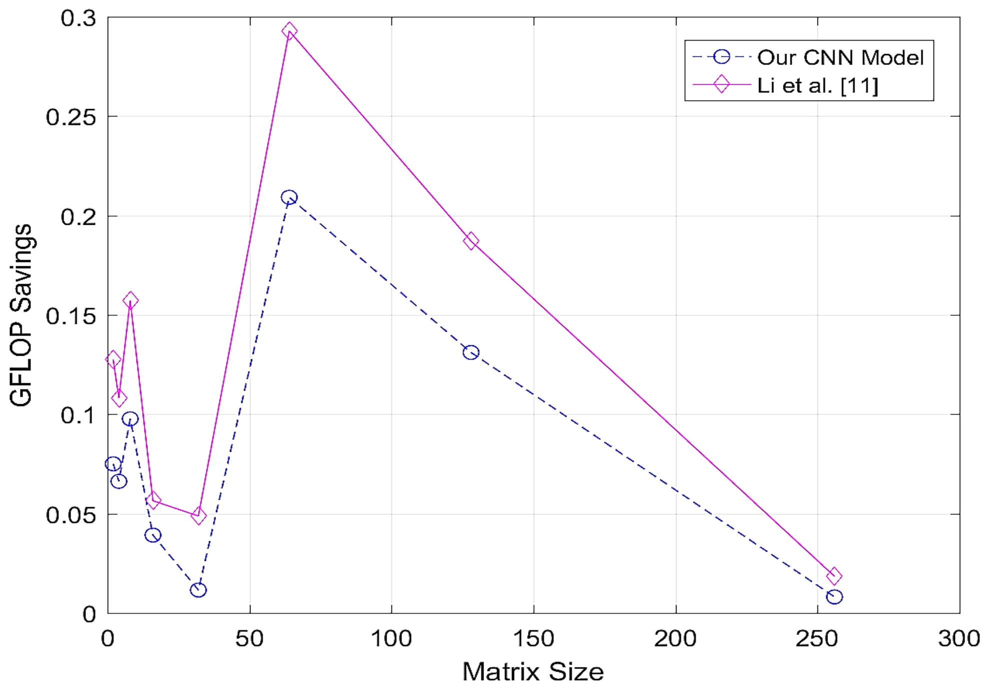


Figure 2.12: Comparison of our CNN model with Li et al. model [1] in terms of GFLOPS saving

2.3.3 Evaluation of Rate-Distortion Optimization

In addition to encoding time reduction, the rate distortion (RD) performance is also a critical metric for our evaluation. In our simulation, we compared the RD performance of our method w.r.t. other methods in terms of BD-BR and BD-PSNR. We can see from Table 2.6 that the BD-BR increment of our deep CNN approach is averaged at 1.34% for the test sequences and outperformed other machine learning methods (1.875 %) [17], (1.638 %) [1], and (2.388 %) [8].

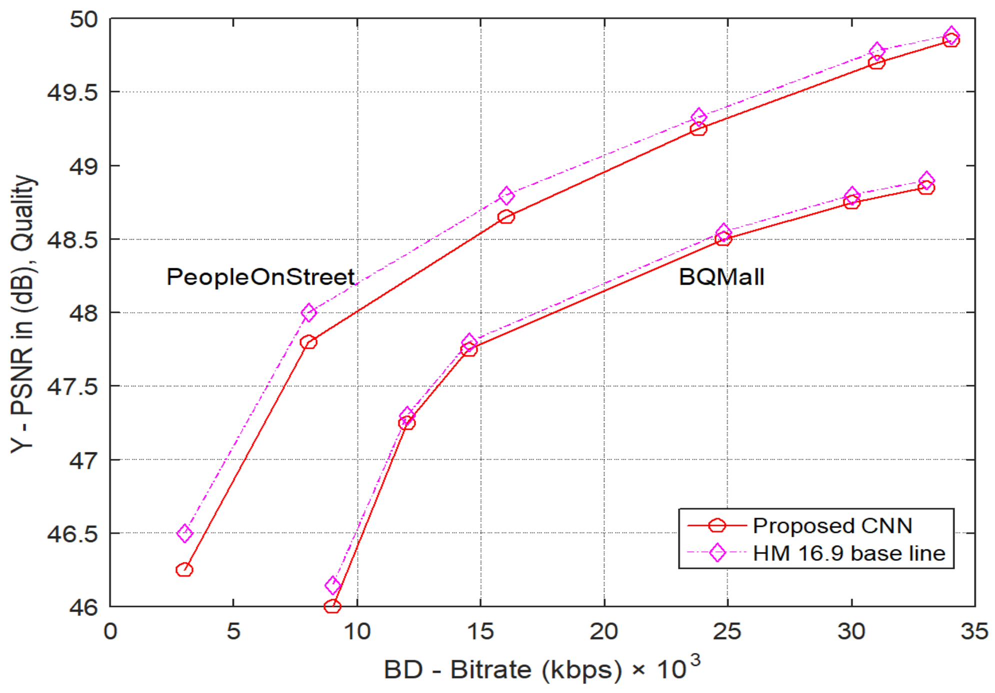


Figure 2.13: RD curves of proposed by our CNN algorithm and that of HEVC base line.

On the other hand our approach incurred -0.044 dB PSNR degradation for videos which is far better than those of (-0.084 dB) for [1], (-0.188 dB) for [8], and

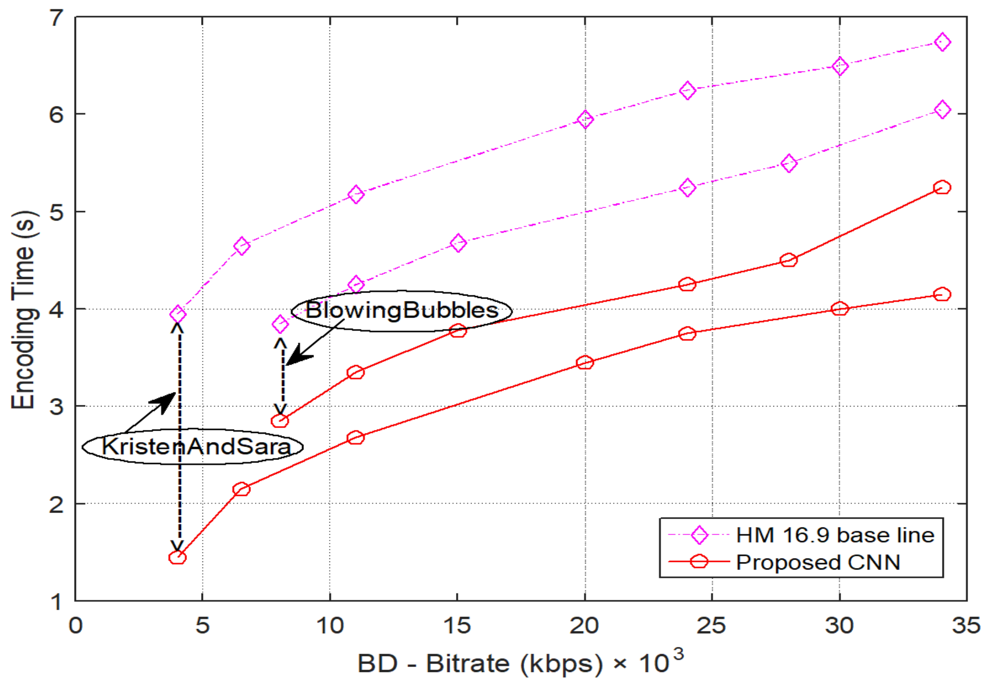


Figure 2.14: Comparison between the bit rate vs. encoding time of our CNN algorithm and that of HEVC baseline

(-0.107 dB) for [17]. Though Li et al [1] are able to achieve a significant encoding time reduction, a notable degradation is found on the RD findings. The main reason of the above degradation is that it could not maintain sufficient high prediction classifier accuracy for different QPs. Thus, our approach performed best among the three approaches by benefiting from the deeper CNN structures with better transfer parameter learning from our large diversified data set. The improvement of the RD performance by our approach is mainly attributed to the optimal prediction accuracy of CU partition and from efficient PU mode search. The RD curves of our proposed algorithm and the original test model HM16.9 are shown in Fig. 2.13. The BD-bitrate vs PSNR graph included the best case “BQMall” and the worst case “PeopleOnStreet” in terms of RD performance. We observed that the proposed model achieves better image quality with the original HEVC reference encoder throughout different QPs. Again Fig. 2.14 shows the BD-Bitrate vs. encoding time graphs at four different QPs which justified that our CNN model can efficiently reduce the encoding time as compared to original HEVC test model. Furthermore, in our model the encoding time reduction is observed for the high-activity sequences such as “KristenAndSara”, “RaceHorses”, and for the low-activity sequences like “BasketballDrill”, “Kimono1”, and “SlideShow”. The minimum encoding time reduction is observed at “Blowing-Bubbles”.

2.4 CONCLUSIONS

In this thesis, we presented a CNN based algorithm which learns the special image features to predict the optimal intra frame CU partitions and reduces the number of mode searches instead of conventional brute-force search. The experimental results show that our deep learning approach performed better than the state-of-the-art machine learning processes in terms of BD–BR, BD–PSNR, encoding time and mode search in RD evaluation. The improvement of RD performance by our approach is mainly due to high prediction accuracy of CTU partitions. The evaluation of video test sequences demonstrated that our approach is able to outperform the previously studied state-of-the-art machine learning models [1], smadi:spk30, [8] in terms of reducing encoder time. Again we compared the number of RDO modes selected by our model to that of HEVC baseline and achieved up to 45.91% reduction in CU/PU mode search process. Although our deep learning based network shows a possible alternative to faster mode selection, more careful training strategies are important for the real-time usages. The videos usually vary widely in the temporal dimension. So to capture the long-term temporal dynamics within and between frames a bi-directional long short-term memory (LSTM) will be very effective [42], [43]. In a future direction: we would also like to extend our work to 1) inter frame (P and B) predictions by using broader data set categories and recurrent LSTM network [44], which is a widely used learning-based algorithm for video analysis, 2) introduce an image pyramid [31] as input to CNN to handle different image scales and resolutions.

APPENDIX A

JENSEN'S INEQUALITY FOR CONVEX FUNCTIONS AND SYSTEM INSTRUCTIONS

In this appendix, we present a procedure for improving the bounds obtained by the application of Jensen's inequality. The methiod is based on the idea of reducing the thickness of a convex region into many thinner convex regions.

A.1 Convex Functions

A real valued function f is defined to be convex over an interval $\Omega = [\alpha, \beta]$ if

$$\lambda\Phi\{x_1) + (1-\lambda)\Phi(x_2) \geq \Phi(\lambda x_1 + (1-\lambda)x_2\}. \tag{A.1}$$

If the above inequality is reversed or

$$\lambda\Phi(x_1) + (1-\lambda)\Phi(x_2) \leq \Phi(\lambda x_1 + (1-\lambda)x_2), \tag{A.2}$$

then Φ is called concave.

A.2 Jensen's Inequality for Convex Functions

Let x be a random variable with a finite mean. If $\Phi(x)$ is real-valued convex function, then

$$E[\Phi(x)] \geq \Phi\left(E[x]\right) \tag{A.3}$$

where $E[.]$ is the mathematical expectation.

A.3 System Instructions

In order to run the core system the following dependencies are required:

- Python 3.6.
- Keras
- Tensorflow

- Numpy
- Python Imaging Library (PIL)
- h5py
- imgaug
- opencv-python

The system has only been tested with Ubuntu 14.04 (Linux Singularity box), but it should be possible to run on both Windows and Linux as long as the listed dependencies have been installed. Ubuntu is highly recommended because of a more convenient installation process.

A Nvidia GPU is also highly recommended for running the system. In most instances a GPU can give considerable speed improvements when training compared to a CPU. This is critical when having to deal with large datasets and models with millions of parameters. In order for Tensorflow Keras to efficiently use your GPU while training, CUDA Toolkit has to be installed.

A graphical user interface can also be utilized for monitoring the training. Running experiments can be stopped from this user interface, as well as a debugging option which displays examples and model predictions. In addition all experimental data and results are stored as JSON, and can be viewed in the user interface. This includes, a loss per epoch plot, precision and recall graph and hyper-parameter configuration.

For installation instructions, please read the included README files of the repositories.

The URL of these repositories are listed below:

- https://github.com/fizyr/keras-retinanet
- https://github.com/fizyr/keras-retinanet/tree/master/keras_retinanet
- https://github.com/fizyr/keras-retinanet/blob/master/keras_retinanet/bin/train.py

APPENDIX B

ESTIMATION of BD - PSNR and BD - BITRATE

In this appendix, we provide the computation of BD- PSNR and BD bit rate [10], [51]. BD-PSNR (Bjontegaard – PSNR) and BD-bit rate (Bjontegaard – bit rate) metrics are used to compute the average gain in PSNR and the average per cent saving in bit rate between two rate-distortion graphs respectively and is an ITU-T approved metric [10]. This method was developed by Bjontegaard and is used to gauge compression algorithms from a visual aspect in media industry and referenced by many multimedia engineers. The MATLAB code is available online [51].

```
    function avg_diff = bjontegaard(R1,PSNR1,R2,PSNR2,mode)
% R1,PSNR1 - RD points for curve 1
% R2,PSNR2 - RD points for curve 2
% mode -
%   'dsnr' - average PSNR difference
%   'rate' - percentage of bit rate saving between data set 1 and
%   data set 2
% % avg_diff - the calculated Bjontegaard metric ('dsnr' or 'rate')
%
% (c) 2010 Giuseppe Valenzise
%
% References:
%
% [1] G. Bjontegaard, Calculation of average PSNR differences between
% RD-curves (VCEG-M33)
% [2] S. Pateux, J. Jung, An excel add-in for computing Bjontegaard metric and
% its evolution
% convert rates in logarithmic units
```

```
lR1 = log(R1);
lR2 = log(R2);
switch lower(mode)
case 'dsnr'
%PSNRmethod
p1 = polyfit(lR1, PSNR1, 3);
p2 = polyfit(lR2, PSNR2, 3);
%integration interval
min_int = min([lR1; lR2]);
max_int = max([lR1; lR2]);
%find integral
p_int1 = polyint(p1);
p_int2 = polyint(p2);
int1 = polyval(p_int1, max_int) - polyval(p_int1, min_int);
int2 = polyval(p_int2, max_int) - polyval(p_int2, min_int);
%find avg diff
avg_diff = (int2-int1)/(max_int-min_int);
case'rate'
%ratemethod
p1 = polyfit(PSNR1,lR1,3);
p2 = polyfit(PSNR2,lR2,3);
%integrationinterval
min_int = min([PSNR1; PSNR2]);
max_int = max([PSNR1; PSNR2]);
%findintegral
p_int1 = polyint(p1);
```

```
p__int2 = polyint(p2);

int1 = polyval(p__int1, max__int) - polyval(p__int1, min__int);

int2 = polyval(p__int2, max__int) - polyval(p__int2, min__int);

%findavgdiff

avg__exp__diff = (int2-int1)/(max__int-min__int);

avg__diff = (exp(avg__exp__diff)-1)*100;

end
```

REFERENCES

[1] M. Xu, Z. W. T. Li, X. Deng, R. Yang, and Z. Guan, “Reducing complexity of hevc: A deep learning approach,” *IEEE Transactions on Image Processing (TIP)*, vol. 27, pp. 5044 – 5059, Oct. 2018.

[2] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, “Overview of the high efficiency video coding (hevc) standard,” *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, pp. 1649–1668, Sept. 2012.

[3] J. Lainema, F. Bossen, W. J. Han, J. Min, and K. Ugur, “Intra coding of the HEVC standard,” *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 22, pp. 1792–1801, Sept. 2012.

[4] L. Zhao, L. Zhang, S. Ma, and D. Zhao, “Fast mode decision algorithm for intra prediction in hevc,” *Visual Communications and Image Processing (VCIP) Conference*, pp. 1–4 , Tainan, Taiwan, 2011.

[5] Y. Piao, J. Min, and J. Chen, “Encoder improvement of unified intra prediction,” *Joint Collaborative Team on Video Coding (JCT-VC) - C207*, pp. 7–15, Sept. 2010.

[6] Y. Wang, X. Fan, L. Zhao, S. Ma, D. Zhao, and W. Gao, “A fast intra coding algorithm for hevc,” *IEEE International Conference on Image Processing (ICIP)*, pp. 4117–4121, La Defense, Paris, France, 2014.

[7] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang, “An effective cu size decision method for hevc encoders,” *IEEE Transactions on Multimedia*, vol. 15, pp. 465–470, Feb. 2013.

[8] B. Min and R. C. C. Cheung, “A fast cu size decision algorithm for the hevc intra encoder,” *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 25, pp. 892–896, May 2015.

[9] J. Xiong, H. Li, Q. Wu, and F. Meng, “A fast hevc inter cu selection method based on pyramid motion divergence,” *IEEE Transactions Multimedia*, vol. 16, pp. 559–564, Feb 2014.

[10] G. Bjontegaard, “Calculation of Average PSNR Differences between R-D Curves, document VCEG-M33, ITU-T VCEG 13th Meeting,” Sept. 2001.

[11] M. Cimpoi, S. Maji, and A. Vedaldi, “Deep filter banks for texture recognition and segmentation,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 25, June Boston, 2015.

[12] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn, towards real-time object detection with region proposal networks,” *Conference on Neural Information Processing Systems (NIPS)*, May 2015.

[13] R. Girshick, “Fast R-CNN,” *IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, Santiago, Chile, Dec. 2015.

[14] S. Cho and M. Kim, “Fast cu splitting and pruning for suboptimal cu partitioning in hevc intra coding,” *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 23, pp. 1555–1564, Sept. 2013.

[15] M. Khan, M. Shafique, and J. Henkel, "An adaptive complexity reduction scheme with fast prediction unit decision for HEVC intra encoding," *IEEE International Conference on Image Processing (ICIP), Melbourne, Australia*, 2013.

[16] 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 (TCSVT)*, vol. 25, pp. 660–673, Apr. 2015.

[17] 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 (TIP)*, vol. 7, pp. 2225–2238, July 2015.

[18] Q. Hu, Z. Shi, X. Zhang, and Z. Gao, "Fast hevc intra mode decision based on logistic regression classification," *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Nara, Japan*, 2016.

[19] Z. Liu, X. Yu, S. Chen, and D. Wang, "CNN oriented fast HEVC intra CU mode decision," *International Symposium on Circuits and Systems (ISCAS) , Montreal, QC, Canada*, 2016.

[20] "JCT-VC, HEVC HM Software, [online]. available:," 2016. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.9/

[21] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *IEEE European Conference on Computer Vision (ECCV)*, pp. 818–833, Zurich, Switzerland, Sept. 2014.

[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," *International Conference on Machine Learning (ICML)*, vol. 86, pp. 807–814, Haifa, Israel, 2010.

[23] F. Perronnin, J. Sanchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," *European Conference on Computer Vision (ECCV), Greece*, Sept. 2010.

[24] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 32, pp. 1704 –1716, Sept. 2012.

[25] J. Balle, A. Stojanovic, and J.-R. Ohm, "Models for static and dynamic texture synthesis in image and video compression," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, pp. 1353–1365, Aug. 2011.

[26] D. G. Lowe, "Object recognition from local scale-invariant features," *IEEE International Conference on Computer Vision (ICCV), Kerkyra, Greece*, 1999.

[27] C. Chen, M. Liu, O. Tuzel, and J. Xiao, "R-cnn for small object detection," *Asian Conference on Computer Vision (ACCV)*, pp. 214–230 , Taiwan, Taiwan, May 2016.

[28] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," 2007.

[29] R. Péteri, S. Fazekas, and M. J. Huiskes, "Dyntex: A comprehensive database of dynamic textures," *Pattern Recognition Letters*, vol. 31, pp. 1627–1632, Elsevier, Sep 2010.

[30] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," *IEEE, Computer Vision and Pattern Recognition (CVPR)*, Columbus, Ohio, June, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *European Conference on Computer Vision (ECCV)*, pp. 346–361, Zurich, Switzerland, Sept. 2014.

[32] S. Kuanar and C. Conly, and K. R. Rao, "Deep Learning based HEVC in-loop Filtering for Decoder Quality Enhancement," *IEEE Picture Coding Symposium (PCS)*, pp. 346–361, San Francisco, 2018.

[33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, pp. 2278–2324, Nov. 1998.

[34] S. Kuanar and K. R. Rao, and C. Conly, "Fast Mode Decision in HEVC Intra Prediction using Region wise CNN Feature Classification," *IEEE International Conference on Multimedia & Expo (ICME), San Diego, USA*, 2018.

[35] M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," Nov. 2016. [Online]. Available: https://arxiv.org/abs/1603.044672016

[36] Y. Shan and E. H. Yang, "Fast HEVC intra coding algorithm based on machine learning and Laplacian transparent composite model," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2642–2646 , New Orleans, USA, May 2017.

[37] F. Bossen, "Common hm test conditions and software reference configurations," *JCT-VC document L1100*, vol. 31, Jan. 2013.

[38] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *Proceedings ACM, Special Interest Group on Computer Architecture (SIGARCH) Computer Architecture News*, vol. 38, pp. 247–257, NY, USA, 2010.

[39] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *IEEE International Conference on Computer Design (ICCD)*, pp. 13–19 , USA, Oct. 2013.

[40] C. Farabet and C. Poulet and J. Y. Han, and Y. LeCun, "Cnp: An fpga-based processor for convolutional networks," *IEEE International Conference on Field Programmable Logic and Applications, Prague, Czech Republic*, Sept. 2009.

[41] V. Sze, Y. H. Chen, T. J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. of the IEEE*, vol. 105, pp. 2295 – 2329, Dec. 2017.

[42] S. Kuanar, V. Athitsos, N. Pradhan, A. Mishra, and K.R.Rao, "Cognitive Analysis of Working Memory Load from EEG by a Deep Recurrent Neural Network," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2576–2580, Calgary, Canada, 2018.

[43] N. Srivastava, E. Mansimov, and R. Salakhutdinov, "Unsupervised learning of video representations using LSTMs," *International Conference on Machine Learning (ICML)*, vol. 3, Jan. 2016. [Online]. Available: https://arxiv.org/abs/1502.04681

[44] S. Bharat, J. Michael, M.Tim, T. Oncel, and S. Ming, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," *IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1961–1970, Las Vegas, USA, 2016.

[45] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *International Conference Artificial Neural Networks and Machine Learning (ICANN)*, pp. 281–290, Springer, Sept. 2014.

[46] S. Kuanar, A. Vassilis, M. Bilas, D. Mahapatra, and K.R. Rao, "Night-time haze and glow removal using deep dilated convolutional network," *Image and Vision Computing (IVC), Elsevier*, 2018.

[47] S. Kuanar, K.R. Rao, C. Conly, and M. Bilas, "Deep convolutional neural network based fast mode decisions in hevc intra prediction," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT),*.

[48] S. Kuanar, K.R. Rao, M. Bilas, and J. Bredow, "Cnn based adaptive cu mode decision in hevc intra prediction," *Circuits, Systems, and Signal Processing (CSSP), Springer*, 2018.

[49] K.R. Rao, J.J. Hwang, and D.N. Kim, *High Efficiency Video Coding and other emerging standards.* River Publishers, USA, 2017.

[50] Humberto Ochoa Dominguez and K.R. Rao, *Versatile Video Coding.* River Publishers, USA, 2019.

[51] "BD metrics open source code. [online]. available:." [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/27798bjontegaardmetric/content/bjontegaard.m

BIOGRAPHICAL STATEMENT

Shiba Kuanar was born in Chandanpur Puri, Orissa India, in 1978. He received his Bachelor of Engineering (B.E.) degree from Indira Gandhi Institute of Technology (IGIT) Sarang, India (State Govt. College) in 2000, his Master of Science (M.S.) degree from The University of Texas at Arlington in 2005 in Electrical Engineering. From 2006 to 2012, he worked around six years with different telecommunication industries like Verizon Communication, AT&T, Amdocs USA and India, and finally IBM India. His telecommunication industry experience is mainly in operational support system (OSS), business support system (BSS) and their implementation on Fiber to node and Fiber to premises network. From 2012 to 2013, he worked as a project associate at the department of Psycho pharmacology, National Institute of Mental Health and Neuro sciences (NIMHANS) Bangalore, a premier institute of India. His research work was mainly focused on tomography image restoration using interactive algorithms. To enhance the research career further, Shiba got into Ph.D. graduate program in Spring 2014 at Electrical Engineering department, University of Texas Arlington. Besides that, Shiba also performed two graduate internships in summer, and fall 2018 at University of Texas Southwestern medical school and worked on bio-medical image analysis using various machine learning techniques.

ACRONYMS

BD-BR	Bjontegaard Delta Bit Rate
CNN	Convolutional neural networks
ConvNet	Convolutional networks
CTU	Coding tree unit
CU	Coding unit
DTD	Describable textures dataset
DynTex	Comprehensive database of DynamicTextures
HEVC	High Efficiency Video Coding
FC	Fully connected
FPGAs	Field programmable gate arrays
FV	Fisher Vector
GMM	Gaussian Mixture Model
GPU	Graphics processing unit
JCT-VC	Joint Collaborative Team on Video Coding
L1	Least Absolute Deviation
L2	Least Squares

LSTM	Long short-term memory
PCA	Principal component analysis
PSNR	Peak Signal to Noise Ratio
PU	Prediction units
PUMS	PU mode selection
RDO	Rate-distortion optimization
ReLU	Rectified Linear Unit
RMD	Rough mode decision
ROI	Region of interest
RPN	Region proposal network
SIFT	Scale-invariant feature transform
SVM	Support vector machine
TFLOPS	Tera Floating point operations per second
w.r.t	with respect to