Fall 8-16-2018

Improvement of Decision on Coding Unit Split Mode and Intra-Picture Prediction by Machine Learning

Wenchan Jiang

IMPROVEMENT OF DECISION ON CODING UNIT SPLIT MODE AND INTRA-PICTURE PREDICTION BY MACHINE LEARNING

A Thesis Presented to

The Faculty of the Computer Science Department

by

Wenchan Jiang

In Partial Fulfillment

of Requirements for the Degree

Master of Computer Science

Kennesaw State University
2018.08

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of, this thesis which involves potential financial gain will not be allowed without written permission.

_____

Wenchan Jiang

Notice to Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Wenchan Jiang

359 J Building, 1100 South Marietta Pkwy
Marietta, GA 30060

The director of this thesis is:

Prof. Ming Yang
Prof. Ying Xie

363 J Building, 1100 South Marietta Pkwy
Marietta, GA 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are required to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for the use of their patrons are required to see that each user records here the information requested.

Name of user	Address	Date	Type of use (examination only or copying)

IMPROVEMENT OF DECISIONS OF CODING UNIT MODE AND INTRA-PICTURE PREDICTION MODE BY MACHINE LEARNING

An Abstract of

A Thesis Presented to

The Faculty of the Computer Science Department

by

Wenchan Jiang

In Partial Fulfillment

of Requirements for the Degree

Master of Compute Science

Kennesaw State University

2018.08

Abstract

High efficiency Video Coding (HEVC) has been deemed as the newest video coding standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group. The reference software (i.e., HM) have included the implementations of the guidelines in appliance with the new standard. The software includes both encoder and decoder functionality.

Machine learning (ML) works with data and processes it to discover patterns that can be later used to analyze new trends. ML can play a key role in a wide range of critical applications, such as data mining, natural language processing, image recognition, and expert systems.

In this research project, in compliance with H.265 standard, we are focused on improvement of the performance of encode/decode by optimizing the partition of prediction block in coding unit with the help of supervised machine learning. We used Keras library as the main tool to implement the experiments. Key parameters were tuned for the model in our convolution neuron network. The coding tree unit mode decision time produced in the model was compared with that produced in HM software, and it was proved to have improved significantly. The intra-picture prediction mode decision was also investigated with modified model and yielded satisfactory results.

Keywords: Machine Learning, HEVC, H.265

IMPROVEMENT OF DECISIONS OF CODING UNIT PARTITION MODE AND INTRA-PICTURE PREDICTION MODE BY MACHINE LEARNING

A Thesis Presented to

The Faculty of the Computer Science Department

by

Wenchan Jiang

In Partial Fulfillment

of Requirements for the Degree

Master of Computer Science

Advisor: Prof. Ming Yang
Co-advisor: Prof. Ying Xie

Kennesaw State University

2018.07

Acknowledgement

I would like to express my sincere gratitude to my advisor Prof. Ming Yang. I still vividly remember his persuasive talk and suggestion in 2015 summer that made me finally decide to go to KSU. And it's his continuous support of my research from topic selecting to trouble shooting problems, and to thesis guidance so that I am able to finish this thesis. He has been a truly wonderful advisor and I really appreciate everything he did for me.

I also want to thank two other committee members: Prof. Ying Xie and Prof. Selena (Jing) He. Prof. Ying Xie is also my co-advisor and the teacher for two courses of Big Data Analytics and Text Mining. These courses have laid solid foundation of big data techniques and applications, which are very helpful in my research. Prof. Selena (Jing) He is the teacher for another two courses of Advanced Algorithm and Advanced Network. The knowledge and projects that I finished in these course enhanced my algorithm background and further facilitated my interview.

This thesis owes much to my wife Wenting Wang, my parents, and my in-laws. Without their support and encouragement, I could not possibly switch my career path from chemistry to computer science in my 30s and completed this master degree. In the past two years at KSU, we not only witnessed my study, internship, work-visa application, and final landing on a good job, but also our first boy Hank. And by the time I finish my thesis defense, I will be expecting my second boy Harry in 3 months. All of these haven become integrated as part of my life, and will be remembered forever. Life is a wonderful journey with family members!

Table of Contents

List of Figures

List of Tables

Chapter I

Introduction

For the recent couple of decades, with the rapid development of hardware, utilization of network techniques, and emerge of digital information era, multimedia technology has witnessed tremendous evolvement as well. Modern computers are capable of receiving, processing, and sending out texts, pictures, audio, and video information utilizing multimedia technology. Video comprises a significant portion among all the multimedia forms. Specially in current decade, people tend to stream video contents online in a real-time manner, such as YouTube, Netflix, and Hulu etc. So digital video compression is one of the key aspects of enabling fast and efficient exchange and distribution of video contents.

Video encoding/decoding lies in the core of video compression concept because video coding techniques provide efficient solutions to represent video data in a more compact and robust way so that the storage and transmission of video can be realized in less cost in terms of size, bandwidth and power consumption (Dass, Sign, & Kaushik, 2012).

Following the previous standard known as H.264, high efficiency video coding (i.e., HEVC) has been deemed as the newest video coding standard of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group (Sullivan, Ohm, Han, & Wiegand, 2012). HEVC has the potential to deliver better performance than earlier standards such as H.264/AVC. The reference software (i.e., HM) have included the implementations of the guidelines in appliance with the new standard. The software

includes both encoder and decoder functionality according to Fraunhofer Heinrich-Hertz-Institut (2015).

There are two important aspects in the video encoding standard: inter-picture prediction and intra-picture prediction. Each of these deals with the two fundamental types of redundancy in video compression, i.e., spatial and temporal redundancies. In the reference software HM, it will explore all the possibilities in a traversal and exhaustive manner to find the best partition and merge pattern for a specific prediction unit. It is a time-consuming process and will be difficult, if not impossible, to stream ultra HD video contents in real time using the new HEVC standard.

On the other hand, machine learning techniques work great with big data and processes them to discover patterns that can be later used to analyze the new trends. Machine learning can play a key role in a wide range of critical applications, such as data mining, natural language processing, image recognition, and expert systems (Konstantinova, 2014). Machine learning refers to neural networks with multiple hidden layers that can learn increasingly abstract representations of the input data (Buhuma, 2015). One famous example is training computer to recognize hand-written digits using Keras library, which has achieved as high accuracy as to 100% (Muqeet, 2017). Modern machine learning frameworks, e.g., convolutional neuron networks, will be an ideal candidate to deal with image data. Machine learning also has other business applications. Text-based searches, fraud detection, spam detection, handwriting recognition, image search, speech recognition, street View detection, and translation are all tasks that can be performed through machine learning (Kishor, 2017).

As indicated previously, in compliance with the newest HEVC standard, we utilized the machine learning technique in Keras framework, and we are focused on the improvement of the performance of encoding/decoding by improving the partition speed of prediction block in coding block, as well as in intra picture prediction.

Chapter II

High Efficiency Video Coding (H.265)

Video Encoding Principles

A video consists of a sequence of pictures (i.e., frames). When these frames are displayed rapidly in succession and if the frame rate is high enough to about 20-25 frames per second, the viewers have the illusion that motion is occurring. To be able to store and transmit large video sequences, they need to be compressed or encoded on the sender side, and then the video can be decoded at the receiver side to be displayed (2017). Hence there are two important processes involved: encode and decode.

Among them, video encoding technique plays an especially important role. Video compression enables digital video to be used in environments that cannot support raw, uncompressed video transmission and storage. For example, current Internet throughput rates make it difficult to process uncompressed video in real time, even at very low video frame rates and with very little video spatial resolution, and a 1.36 GB DVD can only be stored for less than a minute of the equivalent of the original video quality of the TV resolution and frame rate (216 Mbits/s) according to previous standard (2014). Video compression also enables people to use transmission and storage resources more efficiently. Although storage and transmission capacity continues to increase, compression remains the core of multimedia services for a long time to come. For instance, a typical 2-hour full high definition (HD) uncompressed video corresponds to 2 (hours) × 60 (minutes per hour)

× 60 (seconds per minute) × 25 (frame rate, frames per second) × 1920 × 1080 (frame size in pixels) × 3/2 (number of bytes per pixel) = 559,9 GB (Juurlink, et al., 2012). Therefore, we will focus on the improvement of video compression/encoding in this research paper.

Since the importance of video compression has been elaborated above, it's necessary to further discuss the feasibility of video compression. There are several redundancies in a video that will be discussed in detail as follows (Richardson I. E., 2008).

- Spatial redundancy. A static image, such as a human face, background, hair, as well as the brightness and color, are gently and gradually changing. Adjacent pixels and chrominance signal values are relatively close and are with strong correlation. Directly using the exact values of brightness and color information will result in more data and hence the spatial redundancy. If we choose one frame as reference and re-encode the following frames based on the reference frame, we can remove the redundant data and reduce significantly the average number of bits per pixel, which is commonly referred to as intra-frame encoding, i.e., to reduce spatial redundancy for data compression.
- Time redundancy. Video can be considered as a sequence of frame images in the time axis direction, and the correlation between adjacent frame images is also strong. This redundancy or in simpler words the 'repetition of information between frames' is exploited by all the video compression algorithms. The basic idea is not to encode the similar or the near similar pixel values which have already been encoded and transmitted (Bharamgouda, 2013). The techniques of

motion estimation and motion compensation satisfy the quality requirements of decoding and reconstructing images.

- Bits redundancy. Using the same bit to represent symbols of different probabilities can result in the waste of bits. The principle of variable-length coding such as Huffman coding is the example, i.e., shorter code for symbol of higher probability and longer code for the symbols of lower probability according to the Huffman coding explanation. (Fraenkel & Klein, 1990)
- Structural redundancy. There is also a relationship between the various parts of the image. Through this relationship, we can reduce the code redundancy using fractal image coding for example (Zhao, Wang, & Yuan, 2000).
- Visual redundancy. The human eyes are more sensitive to the brightness over colorfulness, still image over moving image, central parts over peripherals etc.

With all the redundancies known, it is a complicated process to identify and come up with an algorithm to reduce them, which is the constant pursuit for researchers. The redundancies mentioned above, especially spatial and temporal ones, are the cornerstone of all video compression standards, from MPEG2, MPEG4 to H.264 and H.265. These standards may as well be considered to have provided some combination of algorithms, simple or complex, to find out where the redundant information is, reduce/compress the redundancies as much as possible without trading off video qualify too much, and finally minimize the amount of data to be transferred.

Overview of H.265

As briefly discussed in previous paragraph, with an increasing growth of video streaming on the Internet over popular websites such as Netflix and YouTube, and with 4K cameras gaining new ground in the market, a considerable amount of storage and bandwidth is required. The new standard of HEVC (i.e., High Efficiency Video Coding), or H.265, promises a 50% storage reduction as its algorithm uses efficient coding by encoding video at the lowest possible bit rate while maintaining a high image quality level. Therefore, conceived to boost video streaming, H.265 is a video compression standard designed to substantially improve coding efficiency when compared to its precedent H.264 (Rodrigues, 2016).

H.265 still uses the widely accepted hybrid coding framework since H.264 as we have seen in previous chapter, including intra-frame prediction, inter-frame prediction based on motion compensation, transformation, entropy coding, and quantization. Figure 1 illustrates such similarities.

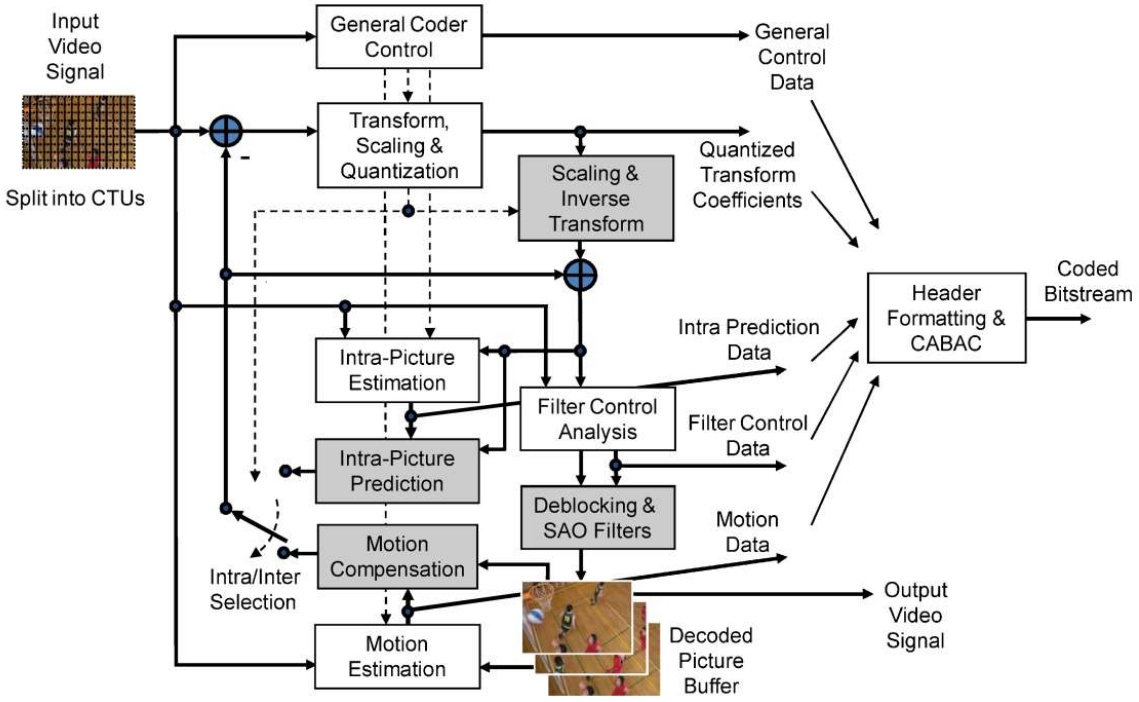


Figure 1. Typical HEVC video encoder *(Sullivan, Ohm, Han, & Wiegand, 2012)*

Compared to previous coding standard, H.265 is more advanced than H.264 in several ways. First, the main difference is that HEVC allows for further reduced file size, and therefore reduced required bandwidth, of your live video streams. Unlike H.264 macroblocks, H.265 processes information in what's called Coding Tree Units (CTUs). Whereas macroblocks can span 4 × 4 to 16 × 16 block sizes, CTUs can process as many as 64 × 64 blocks, giving it the ability to compress information more efficiently. Secondly, along with the improved CTU segmentation of larger size, H.265 also has better motion compensation and spatial prediction than H.264 does. This means that H.265 requires more advanced hardware to be able to compress the data. Fortunately, however, it also means that viewers with H.265 compatible devices will require less bandwidth and processing power to decompress that data and watch a high quality stream. This also enables the

streaming of 4K video over common network speeds (Rodrigues, 2016). Table 1 summarized the improvement of H.265 standard compared with H.264 in major aspects (Narang, 2013).

Category	H.264	H.265
Partition Size	Macroblock 16 × 16	Coding Unit 8 × 8 to 64 × 64
Partitioning	Sub-block down to 4 × 4	Prediction Unit Quadtree down to 4 × 4 square,
Intra Prediction	Up to 9 predictions	35 predictions
Transform	Integer DCT 8×, 4×4	Transform Unit square IDCT from 32×32 to 4×4 + DST Luma intra 4×4
Filters	Deblocking filter	Deblocking filter, Sample Adaptive Offset
Motion Prediction	Spatial Median (3 blocks)	Advanced Motion Neighbor Vector Prediction (AMVP) for both spatial and temporal
Entropy Coding	CABAC, CAVLC	CABAC

Table 1. Main Differences between H.264 and H.265

Coding Tree Units Structure

The central piece in previous standard is macroblock, containing a 16 × 16 block of luma samples and, in the usual case of 4:2:0 color sampling, two corresponding 8 × 8 blocks of chroma samples. In recent decade, we have much higher frame sizes to deal with since 4K production became practical and 8K is also promising. Even mobile device such as iPhone X's display is 5.8 inches with a 2436-by-1125-pixel resolution at 458 ppi. Therefore, larger

macroblocks are needed to efficiently encode the motion vectors for these frame size. On the other hand, blocks at the granularity of 4×4 are also essential to process prediction and transformation of small details.

Then it comes the replacement of macroblock. The analogue in H.265 is CTUs with size determined by the encoder and larger than a traditional macroblock. The CTU consists of a luma CTB and the corresponding chroma CTBs and syntax elements as shown in Figure 2 and 3 (Sullivan, Ohm, Han, & Wiegand, 2012).

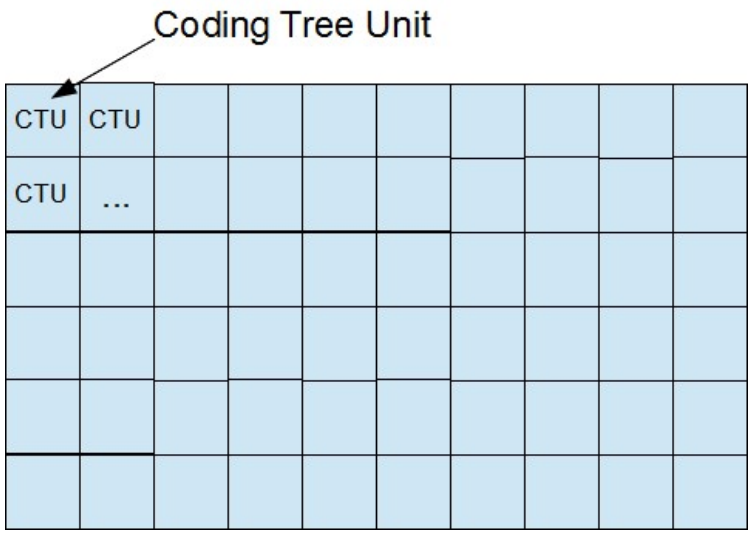


Figure 2. Coding Tree Unit in H.265 *(Moto, 2012)*

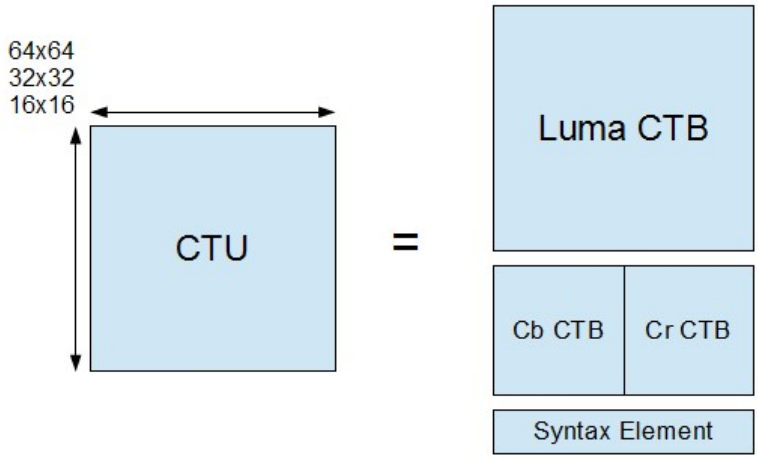


Figure 3. CTU and related CTB syntax *(Moto, 2012)*

The size L × L of a luma CTB can be chosen as L = 16, 32, or 64 samples, with the larger sizes typically enabling better compression. Each luma CTB still has the same size as CTU. HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quadtree-like signaling. Depending on a part of video frame, however, CTB may be too big to decide whether we should perform inter-picture prediction or intra-picture prediction. Thus, each CTB can be differently split into multiple CBs (Coding Blocks) and each CB becomes the decision making point of prediction type. For example, some CTBs are split to 16 × 16 CBs while others are split to 8 × 8 CBs. HEVC supports CB size all the way from the same size as CTB to as small as 8 × 8. Figure 4 illustrates how 64 × 64 CTB can be split into CBs.

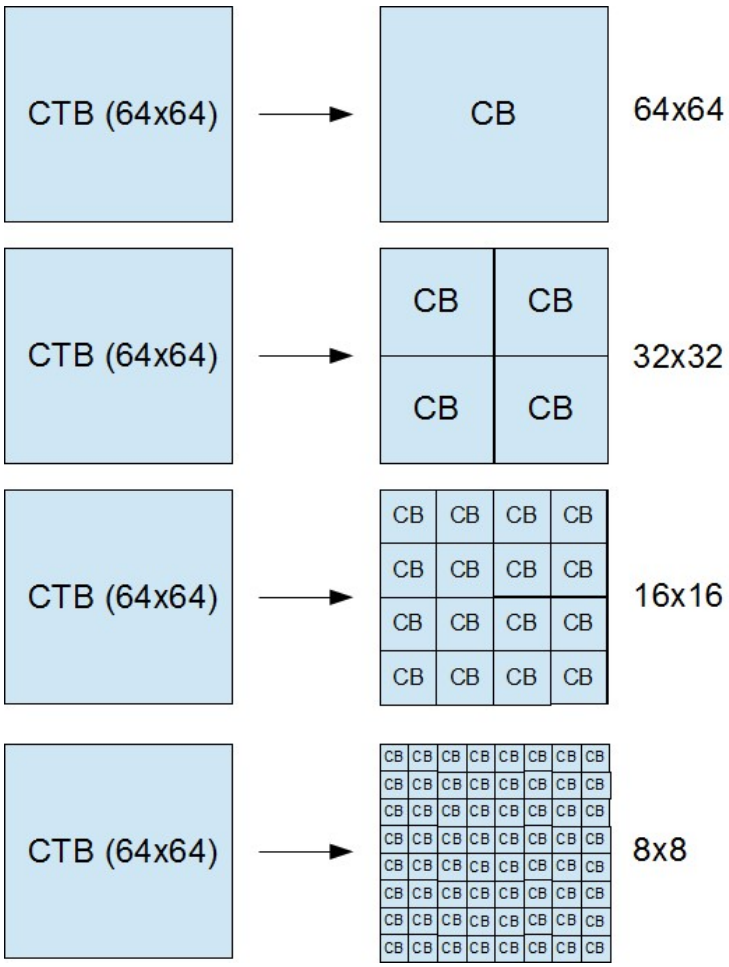


Figure 4. Illustration of CTBs split into CBs *(Moto, 2012)*

Partitioning of CTU

Each CTU can be further evenly divided into four square CUs, and one CU can be recursively divided into four small CUs according to a quadtree structure, shown in Figure 5. For color video with Y:U:V = 4:2:0, one CU consists of one CB of the luma samples, two CBs of the chroma samples and related syntax elements. One Luma CB is a pixel region of $2^N \times 2^N$ (where N is different in size from N in the CTU) and the corresponding

chroma CB is $2^{(N-1)} \times 2^{(N-1)}$ in the pixel area. The value of N is likewise determined in the encoder and transmitted in the sequence parameter set (SPS).

During encoding, at the CTU level, the split_cu_flags flag is transmitted to indicate whether the CTU is further divided into four CUs. Similarly, for a CU, a split_cu_flags flag is also used to indicate whether to divide further into sub-CUs, until split_cu_flags decreases to be 0 or a minimum CU size is reached. Therefore, the size range of a CU is: minimum size CU to the size of CTU. Generally, CU has minimum size of 8 (determined by the depth of the CTU) and CTU has the size of 64, so the size of the CU may be 8, 16, 32, and 64 at this time. The CU encoding is performed in a depth-first order, similar to the z-scan, as shown in the figure below: The right indicates the recursive quadtree division of the CTU and the left indicates the coding order of the CU in the CTU.

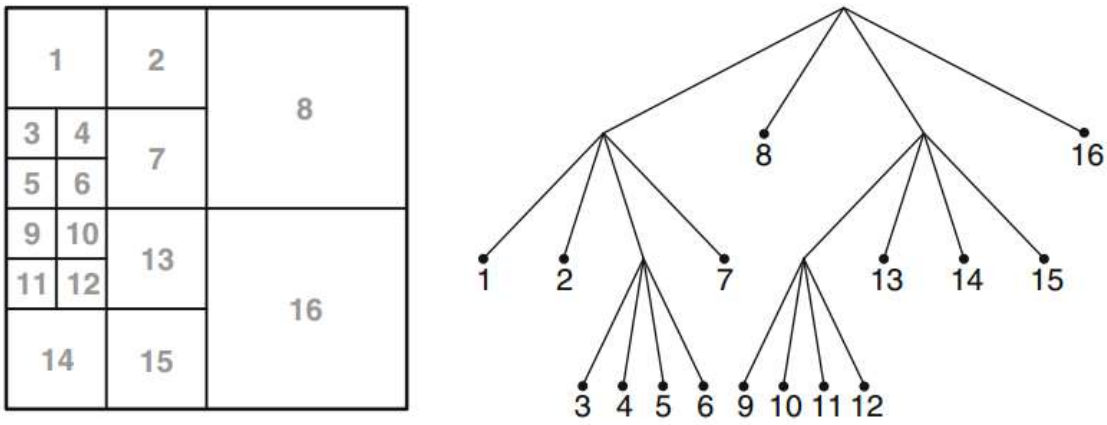


Figure 5. Left: partitioning of a 64 × 64 CTU into CUs; Right: Quadtree structure of the partitioning with numbers indicating the coding order of the CUs *(Schwarz, Schierl, & Marpe, 2014)*

Furthermore, the resolution (horizontal and vertical size) of the video sequence is also transmitted in the sequence parameter set. Such resolution has to be an integer multiple of

the minimum CU size, but it does not need to be an integer multiple of the CTU size. If the resolution does not represent an integer multiple of the CTU size, the CTUs at the borders are inferred to be split until the boundaries of the resulting blocks coincide with the picture boundary. All in all, the CU block is the basic unit for decision-making of intra-picture prediction or motion-compensated prediction. In that respect, CUs in H.265 are similar to macroblocks in older video coding standards but with variable sizes (Schwarz, Schierl, & Marpe, 2014).

Intra-Picture Prediction

Similar to H.264, H.265 intra prediction uses the reconstructed values of adjacent blocks to perform the predictions. Therefore, the coding mode selection and encoding are the key factors to be addressed in intra prediction. The biggest difference between H.265 and H.264 in intra-picture prediction is that H.265 adopts larger and more size selection to suit the characteristic content of ultra-high definition video and that supports more intra-prediction modes to be suitable for finer details.

The CB (i.e., Coding Block) can be split into size of M × M or (M/2 × M/2), as shown in Figure 6. The first one means that the CB is not split, so the PB (i.e., Prediction Block) has the same size as the CB. It is possible to use it in all CUs. The second partitioning means that the CB is split into four equally-sized PBs. This can only be used in the smallest 8 × 8 CUs. In this case, a flag is used to select which partitioning is used in the CU. Each resulting PB has its own intra prediction mode. The intra-prediction in luma component in

HEVC standard supports five types of PUs (Prediction Units): 4 × 4, 8 × 8, 16 × 16, 32 × 32, and 64 × 64.

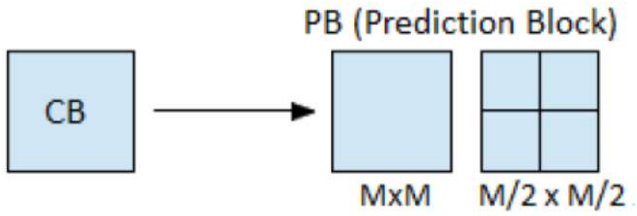


Figure 6. Illustration of CB to PB *(Rao, Thakur, & Adavi, 2016)*

Each type of PU corresponds to 35 types of prediction modes including planar mode, DC mode and 33 kinds of angular mode. Planar mode is developed from the plane mode in H.264/AVC and is improved to preserve continuities along the block edges. This mode in HEVC is known as mode 0. Planar mode uses two linear filters, horizontal and vertical, with the average of the two as the prediction for the current block of pixels. This mode is implemented as follows, as shown in Figure 7. The sample X is the first sample predicted as an average of the samples D and E, then the right column samples (blue samples) are predicted using bilinear interpolation between samples in D and X, and the bottom row samples (orange samples) are predicted using bilinear interpolation between samples in E and X. The remaining samples are predicted as the averages of bilinear interpolations between boundaries samples and previously coded samples (Rao, Thakur, & Adavi, 2016).

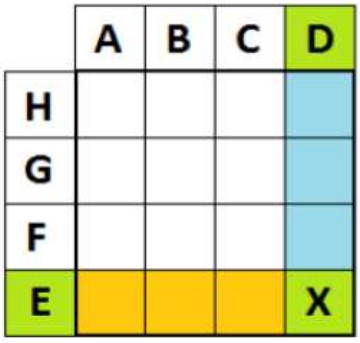


Figure 7. Planar intra prediction mode *(Ling, 2012)*

DC mode is suitable for large flat areas. This mode is also similar to the DC mode in H.264/AVC. It is efficient to predict plane areas of smoothly-varying content in the image, but gives a coarse prediction on the content of higher frequency components and as such it is not efficient for finely textured areas (Rao, Thakur, & Adavi, 2016). The current block prediction value can be obtained from the average value of the reference pixels on the top and left neighboring TBs (excluding the upper left and upper right corners), shown in Figure 8.

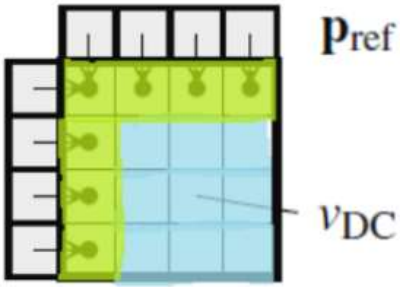


Figure 8. DC intra prediction mode *(Rao, Thakur, & Adavi, 2016)*

Angular intra-picture prediction in HEVC is designed to efficiently model different directional structures typically present in video and image content. The set of available prediction directions has been selected to provide a good trade-off between encoding complexity and coding efficiency for typical video material. The sample prediction process

itself is designed to have low computational requirements and to be consistent across different block sizes and prediction directions. This has been found especially important as the number of block sizes and prediction directions supported by HEVC intra coding far exceeds those of previous video codecs, such as H.264/AVC. In HEVC there are four effective intra prediction block sizes ranging from 4 × 4 to 32 × 32 samples, each of which supports 33 distinct prediction directions. A decoder must thus support 132 combinations of block sizes and prediction directions (Rao, Thakur, & Adavi, 2016). All of 33 angle modes are specific directions, wherein V0 (mode 26) and H0 (mode 10) are represented as vertical and horizontal directions. The rest of prediction modes can be seen as a variant of vertical or horizontal mode, with the variant offset value can be calculated accordingly (Han & Lainema, 2014).

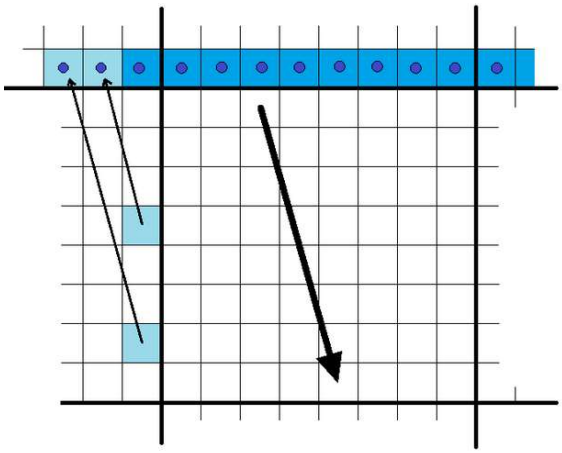


Figure 9. Illustration of angular modes *(Patel, Lad, & Shah, 2015)*.

Brief Summary

In summary, the purpose of HEVC intra-picture prediction is to eliminate spatial redundancy, which is further developed by the HEVC standard relative to H.264. In order to adapt the content characteristic of the high definition video, H.265 uses more flexible sizes of prediction block. Also to adapt to richer textures, H.265 specifies more prediction modes that correspond to different prediction directions. The total of 35 intra prediction modes are summarized in Figure 10.

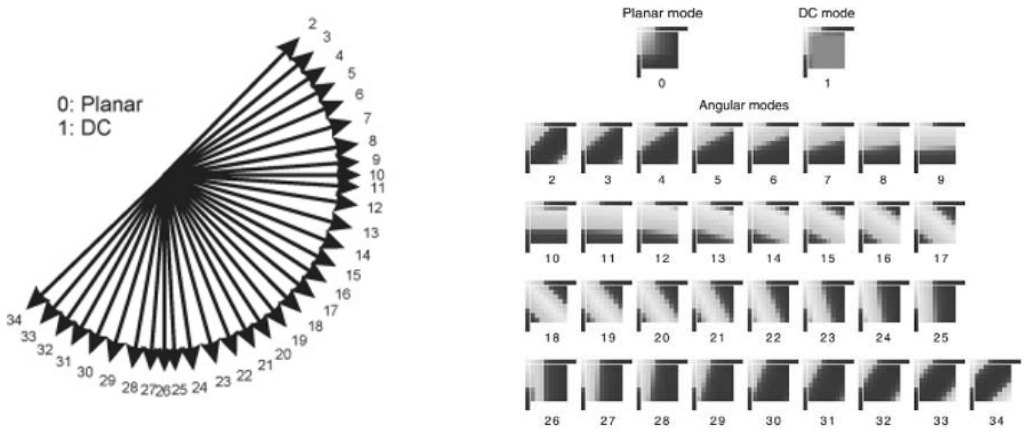


Figure 10. Summary of 35 modes in intra-picture prediction *(Han & Lainema, 2014)*

Chapter III

Machine Learning

Overview

Over the past decade machine learning has become one of the top trending of information technologies deeply integrated with our life. With the ever increasing amounts of data becoming available, it's reasonable to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress. In general, people observe, learn, and then master a skill. The computer-implemented data-derived algorithms are designed to allow machines (computers) to simulate human learning behaviors, acquire skills, and help predict and identify objects. In a broad sense, machine learning is a way of giving machine the learning ability to perform functions that traditional programming cannot directly accomplish. However, in practical terms, machine learning is a method of training a model by using the data, and then using the model for future prediction (Smola & Vishwanathan, 2008).

Types of Algorithm

There are many types of machines learning. Broadly, there are 3 types of machine learning in terms of the way of learning. The first one is supervised learning. Supervised learning algorithms try to model relationships and dependencies between the target

prediction output and the input features such that we can predict the output values for new data based on those relationships which it learned from the previous data sets. There are two subcategories under supervised learning: classification and regression. Classification algorithms include Decision Tree (e.g., banking credit assessment), Nearest Neighbor (e.g., face recognition), Support Vector Machine (e.g., red eye detection), and Neural Network (e.g., recognition of hand-written digits). Regression algorithms include Linear Regression and Non-linear Regression, both of which are widely used for sales or price prediction (Alpaydin, 2014).

The second type is unsupervised learning. In this algorithm, there does not exists any target or outcome variable to predict / estimate. Specifically, the computer is trained with unlabeled data. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Typical algorithms include K-means Clustering and Hierarchical, widely used in an effort to mine for rules, detect patterns, and summarize and group the data points which help in deriving meaningful insights and describe the data better to the users (Alpaydin, 2014). The first two types of machine learning are illustrated in Figure 11.

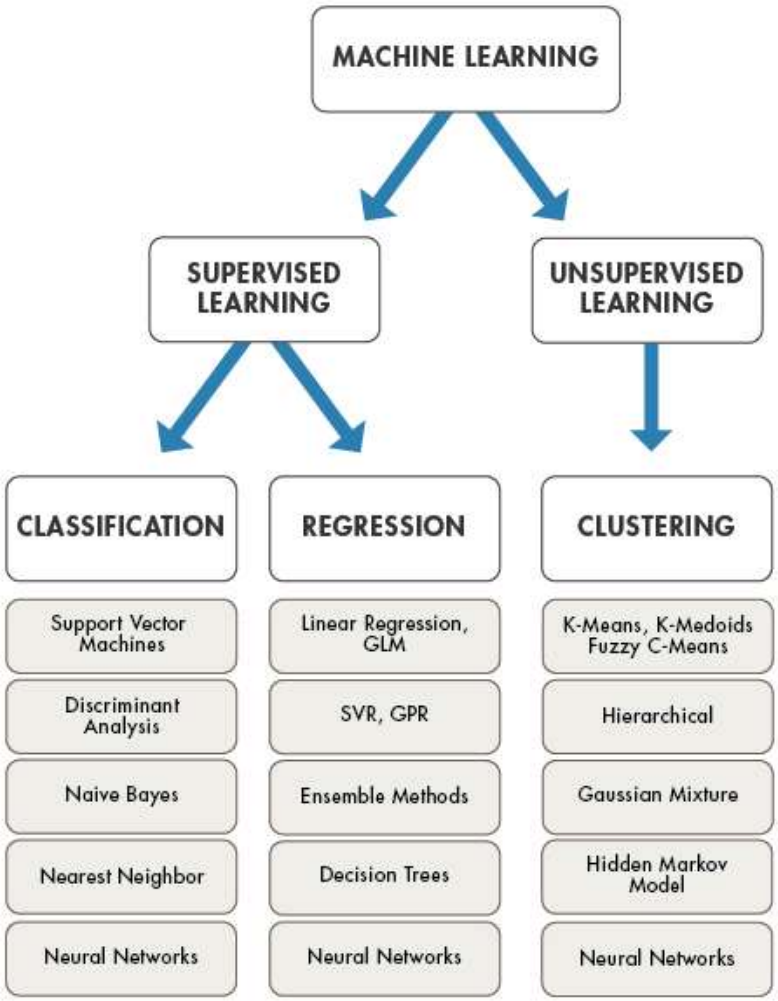


Figure 11. Summary of supervised and unsupervised learning *(Machine Learning in MATLAB, 2017)*

The third type is Reinforcement Learning, as shown in Figure 12. The machine using this algorithm is trained to make specific decisions. This method aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. In the process, the agent learns from its experiences of the environment until it explores the full range of possible

states. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal. Some applications of the reinforcement learning algorithms are computer played board games (Chess, Go), robotic hands, and self-driving cars. (Ray, 2017; Fumo, 2017; Alpaydin, 2014)

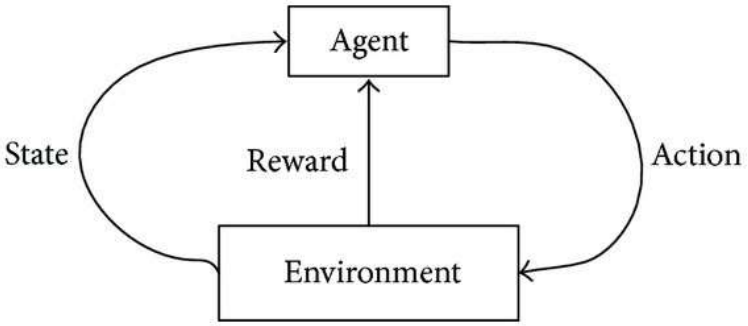


Figure 12. Illustration of reinforcement learning *(Fumo, 2017)*

Convolutional Neural Networks (CNN)

Overview

Image recognition is one of applications that supervised learning specializes as outlined in previous section. As we are processing videos under the new H.265 standard, we are dealing with frames of pictures essentially. Therefore, we will mainly focus on image recognition and how it is properly handled with one branch of machine learning, i.e., Convolutional Neural Network (CNN).

Regular neural networks receive an input (a single vector), and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores. Figure 13 illustrates how computer interprets an incoming image and classify an object in the picture.

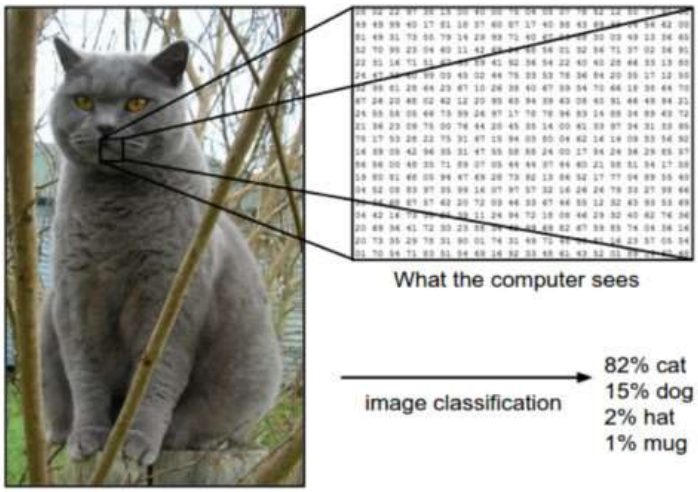


Figure 13. How computers interprets an image *(Ravindra, 2017)*

But regular neural nets don't scale well to full or even huge images. CNN makes the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. CNN roots biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel and Wiesel in 1962 where they showed that some individual neuronal cells in the brain responded (or fired) only in the presence of edges of a certain orientation. This idea of specialized components

inside of a system having specific tasks (the neuronal cells in the visual cortex looking for specific characteristics) is one that machines use as well, and is the basis behind CNNs (Deshpande, 2016).

From a broad perspective, CNNs will take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. The output can be a single class or a probability of classes that best describes the image. Figure 14 illustrate the structure of a typical 3-layer neural network.

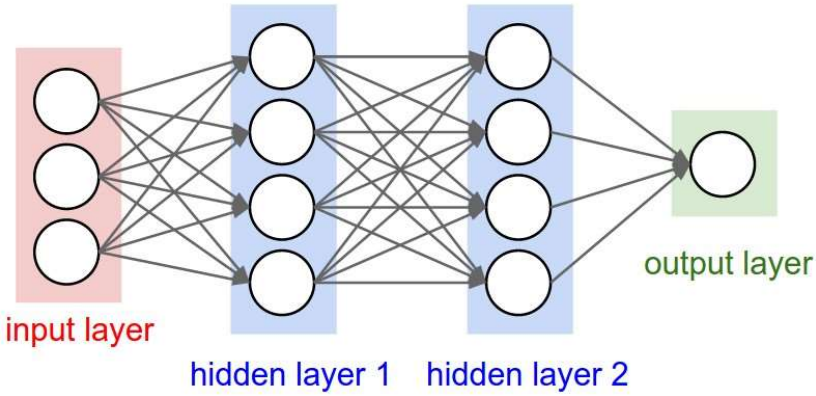


Figure 14. Illustration of the structure of a neural network *(Karpathy, 2017)*

Convolutional Layer

CNNs make the image processing computationally manageable through filtering the connections by proximity. The first layer in a CNN after the input is always a convolutional layer. The input is a 32 × 32 × 3 array of pixel values. Rather than linking every input to every neuron, CNNs restrict the connections intentionally so that any one neuron accepts the inputs only from a small subsection of the layer before it (say like 5 × 5 or 3 × 3 pixels). Hence, each neuron is responsible for processing only a certain portion of an image. This is almost how the individual cortical neurons function in your brain. Each neuron responds

to only a small portion of your complete visual field). This subsection is called a filter or a kernel and the region that it covering is called the receptive field. Now this filter is also an array of numbers called weights or parameters. And the depth of this filter has to be the same as the depth of the input, so the dimensions of this filter is 5 × 5 × 3. The filter is sliding, or convolving, around the input image, it is multiplying the values in the filter with the original pixel values of the image. These multiplications are all summed up to be a single number, which is just representative of when the filter is at the top left of the image. This process is repeated, generating a number for every location on the image. After all the locations are covered, a 28 × 28 × 1 array of numbers known as feature map will be produced, because there are 784 different locations that a 5 × 5 filter can fit on a 32 × 32 input image. And these 784 numbers are mapped into a 28 × 28 array (Deshpande, 2016; Ravindra, 2017).

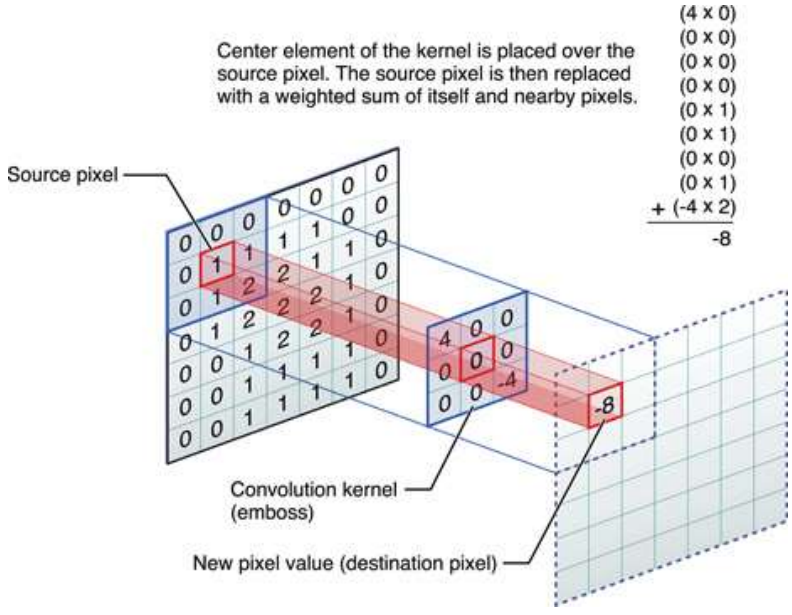


Figure 15. Illustration of a convolutional layer with 3 × 3 filter *(Nielsen, 2015)*

Pooling Layer

It is a common practice to have a pooling layer in-between successive Convolutional layers in a CNN. It can also be considered as a downsampling layer. The goal of a pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The most common form is a pooling layer with filters of size 2 × 2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case take a max over 4 numbers. The depth dimension remains unchanged. In addition to max pooling, the pooling units can also perform other functions: such as average pooling or even L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice (Karpathy, 2017).

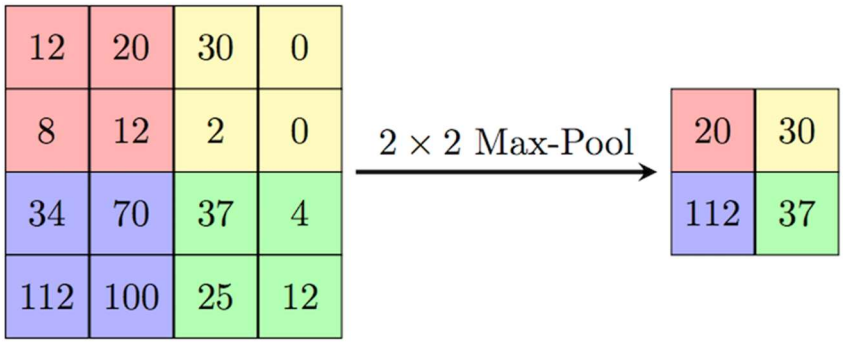


Figure 16. Illustration of Max Pooling Layer *(Veličković, 2017)*

Activation Layer

Activation layer is a node added to the output end of any neural network. It can also be attached in between two neural networks. Without the activation layer, each layer in the

neural network only makes a linear transformation, and after the multi-layer input is superposed, it is still a linear transformation. Even for a perceptron with a hidden layer, the output is still a complicated linear function to try to represent the actual result. Because the expression of linear model is not enough, the activation function can introduce nonlinear factors. There are many different non-linear activation functions, mainly divided on the basis of their range or curves to meet various purposes. For example, Sigmoid $f(x) = \frac{1}{1+e^{-x}}$ will look like S-shape, and this function is differentiable. Therefore, it is especially useful for models where we have to predict the probability as an output since probability of anything exists only between the range of 0 and 1. Another example is ReLU (Rectified Linear Unit) activation function. It is the most used activation function in almost all the convolutional neural networks or deep learning. It is zero when input is less than zero and it is equal to input when the input is above or equal to zero, which in turns affects the resulting graph by not mapping the negative values appropriately (Sharma, 2017; Di, 2013). Some of the typical activation functions are summarized below.

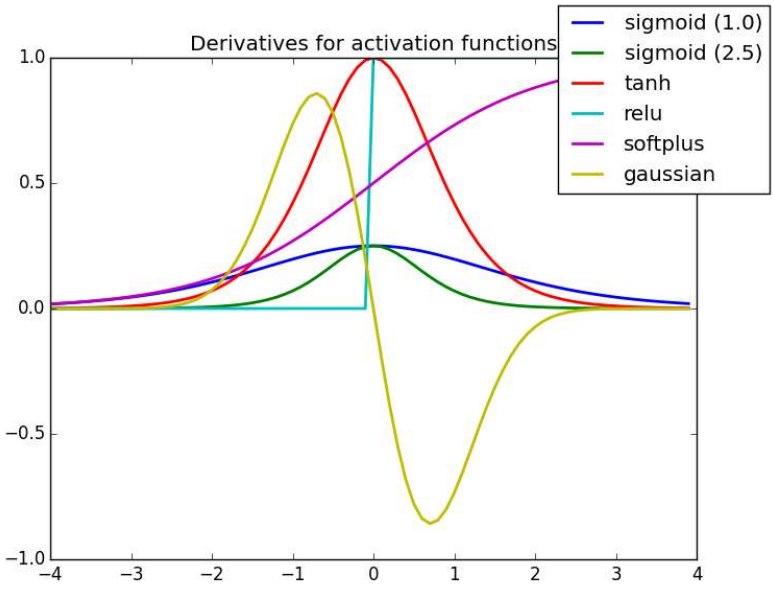


Figure 17. Example of activation functions *(Sharma, 2017)*

Fully Connected Layer

The image pixel-array is downsampled and utilized as the regular fully connected neural network's input. Since the input's size has been reduced dramatically using pooling and convolution, the input has fallen into a category that a normal network will be able to handle while still preserving the most significant portions of data. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. This layer basically takes an input volume and outputs an N dimensional vector where N is the number of classes that the program has to choose from. For example, if you wanted a digit classification program, N would be 10 since there are 10 digits. Each number in this N dimensional vector represents the probability of a certain class. If the resulting vector for a digit classification program is

[0 .1 .1 .75 0 0 0 0 0 .05], then this represents a 10% probability that the image is 1, a 10% probability that the image is 2, a 75% probability that the image is 3, and a 5% probability that the image is 9. In principle, a fully connected layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when it computes the products between the weights and the previous layer, and it generates the correct probabilities for the different classes (Deshpande, 2016; Ravindra, 2017).

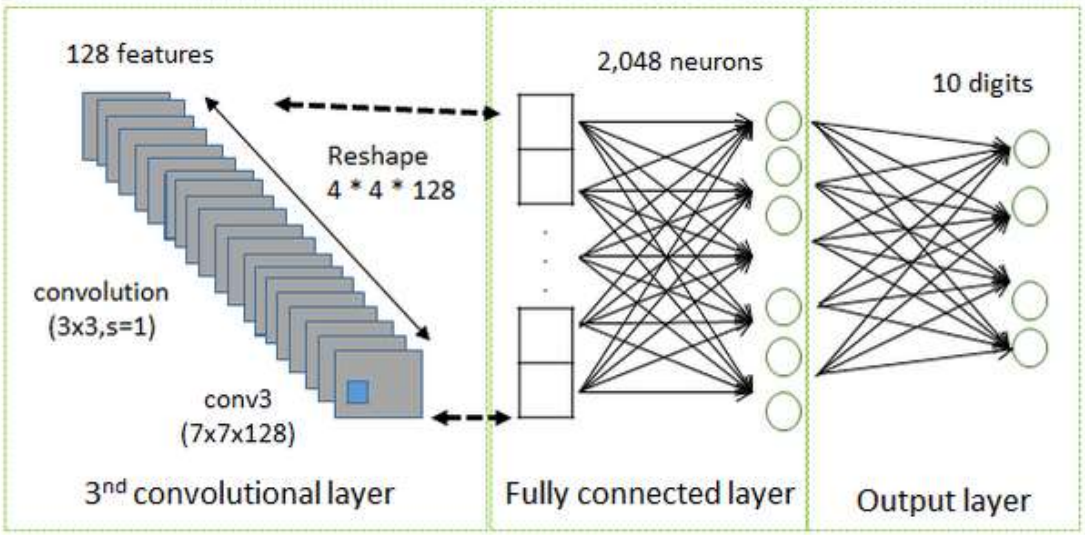
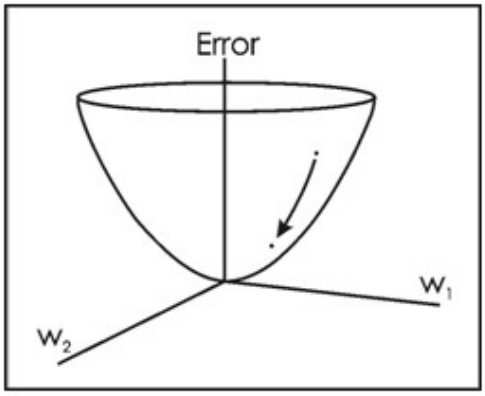


Figure 18. Data flow into fully connected layer *(Zaccone, 2017)*

Training and Testing

Training and testing are probably most important components in a CNN. The way the computer is able to adjust its filter values (i.e., weights) is through a training process called backpropagation. By backpropagation, the filters in the first convolutional layer know where to look for edges and curves, and the fully connected layer know what activation maps to look at.

Before the CNN starts, the weights or filter values are randomized. The filters don't know to look for edges and curves. So in the beginning, we need to correctly label each individual image. This idea of being given an image and a label is the training process that CNNs go through. So backpropagation can be divided into 4 distinct sections, the forward pass, the loss function, the backward pass, and the weight update. During the forward pass, a N × N array of image of number is passed into the whole network. Since all of the weights were randomly initialized and the output will have no preference for any number yet. Then it follows the loss function. Since the data have both an image and a label. If the training image is number 3, then the label for this image will be a vector of [0 0 0 1 0 0 0 0 0 0]. A common definition of loss function is MSE (mean squared error), i.e., $E_{total} = \Sigma\frac{1}{2}(target - output)^2$. The total loss will be very high for the initial couple rounds of training images. In order to get to a point where the predicted label is the same as the training label, the amount of loss needs to be minimized (Deshpande, 2016). It is essentially a calculus problem, and the solution is to find out which weights most directly contributed to the loss of the network as shown in Figure 19.



$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

Figure 19. Left: 3-D visualization of loss function with 2 parameters (w_1 and w_2); Right: Mathematical representation of dL/dW where L is the total loss and W is the weight *(Deshpande, 2016)*

A backward pass through the network is performed to determine the most significant weights contributed most to the loss and funds ways to adjust them to decrease the loss. After the derivative is obtained, the weights can be updated in the opposite direction of the gradient. The parameter of learning rate is also carefully and properly chosen. This is because a large learning rate means bigger steps taken to update the weights and it may require less time for the model to converge on an optimal set of weights. On the other hand, if a too high learning rate may be not precise enough and miss the converging point (Surmenok, 2017).

The aforementioned process of forward pass, loss function, backward pass, and parameter update is considered as one training iteration. In a CNN, such process will repeat for a certain number of iterations for each batch of training images. Ideally, when such iteration finishes on the last batch of samples, all the weights across the layers should be finely tuned and the CNN is expected to be trained well enough so that the model can predict the results with high accuracy.

Brief Summary

The concepts and related algorithms of machine learning are briefly overviewed in this chapter. Convolutional neural network is introduced with regard to its principles and implementation. Key components in a CNN are discussed in detail including convolutional layer, pooling layer, activation function, and fully connected layer. A typical CNN process

including all the important components as discussed before is depicted in Figure 20, which illustrates how a picture is fed into a CNN, processed in different layers and finally resulting in a predicted label per node.

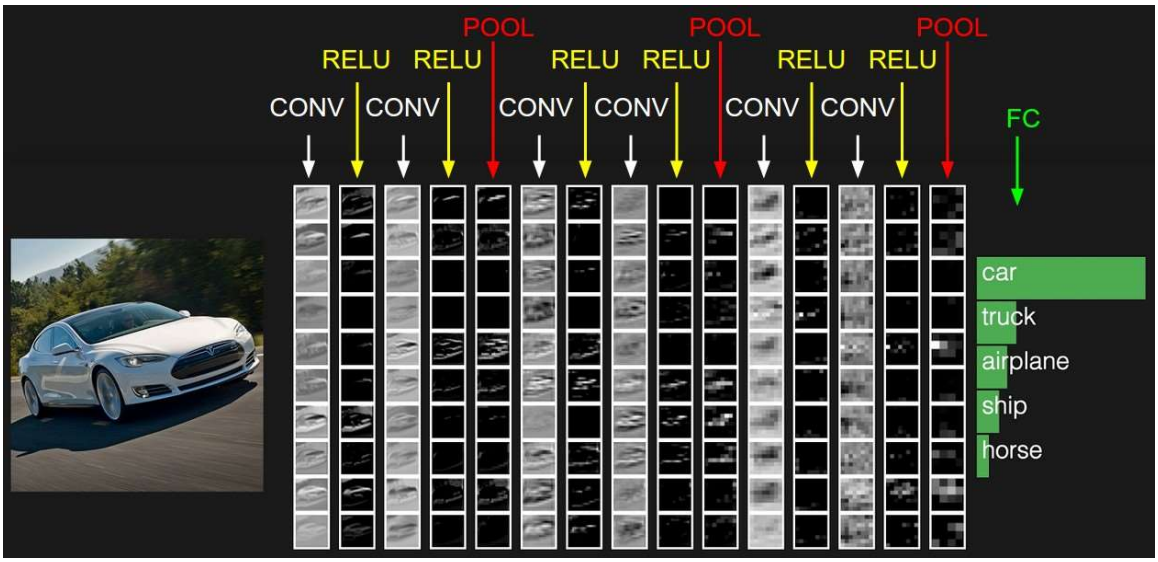


Figure 20. Illustration of a typical CNN and the layers *(Karpathy, 2017)*

Chapter IV

Literature Review

In the field of improving encoding in HEVC/H.265 with machine learning technique, there exist some related literatures aiming to solve the complexity issues in the newest standard through all different routes. Momcilovic et al proposed a novel fast Coding Tree Unit partitioning for HEVC/H.265 encoder (Momcilovic, Roma, Sousa, & Milentijevic, 2015). In their paper, it decouples the requirement of any pre-training and yields a high adaptivity to the dynamic changes in video contents because it depends on efficient sampling strategy and run-time trained neural networks for fast coding units splitting decisions. It claims to reduces the HEVC/H.265 encoding time for up to 65% with negligible rate-distortion penalties.

In another study the authors proposed a machine learning based method for fast CU partition decision using features that describe CU statistics and sub-CU homogeneity (Duanmu, Ma, & Wang, 2015). A "preprocessing" module implementing such proposed scheme sits on top of the Screen Content Coding reference software. Comparatively, the experiment resutls can achieve 36.8% complexity reduction on average with only 3.0% bit-rate increase. In Alam's work (Alam, Nguyen, Hagan, & Chandler, 2015), a fast convolutional-neural-network based quantization strategy for HEVC was proposed. A network trained on data derived from an improved contrast gain control mode was employed to predict local artifact visibility. They further utilized the contrast gain control model to propose a structural facilitation model to capture effects of recognizable structures

on distortion visibility. Their results provided on average 11% improvements in compression efficiency for spatial luma channel of HEVC while requiring almost one hundredth of the computational time of an equivalent gain control model.

Liu et al. proposed a fast algorithm based on convolution neural network to decrease no less than two CU partition modes in each CTU for full rate-distortion optimization (RDO) processing, therefore reducing the encoder's hardware complexity (Liu, et al., 2016; Liu, Yu, Chen, & Wang, 2016). As their algorithm does not depend on the correlations among CU depths or spatially nearby CUs, it was friendly to the parallel processing and did not deteriorate the rhythm of RDO pipelining. The proposed algorithm can save 63% Intra encoding time at the cost of the averaged 2.66% BDBR increase.

Furthermore, another research group proposed a fast coding unit (CU) depth decision algorithm for intra coding of HEVC using an artificial neural network (ANN) and a support vector machine (SVM) (Chen, Fang, Liu, & Chang, 2016). In their research, machine learning provided a systematic approach for developing a fast algorithm for early CU splitting or termination to reduce intra coding computational complexity. Experiment results showed that the proposed fast algorithm saves at most 48.5% and on average 33% encoding time with a 1.55% Bjontegaard delta bit rate (BDBR) loss compared with HM 15.0.

On the other hand, there also exist some research focused on the intra-picture prediction algorithm as well. Song *et al*. reported a CNN-based arithmetic coding method for intra prediction modes in HEVC (Song, Liu, Li, & Wu, 2017). In their research, a customized CNN is used to predict the probability distribution of the intra prediction modes, and multi-

level arithmetic codec is adopted to compress the intra prediction modes with the predicted probability. Their results showed up to 9.9% bits saving compared with context-adaptive binary arithmetic coding (CABAC). Similar work had been shown in another group, as they down-sampled a CTU before being compressed by normal intra coding, and then up-sampled by CNN-based technique to its original resolution in order to enable adaptive sampling rates for different CTUs (Li, et al., 2017). Specifically, a two-stage up-sampling process is proposed in accordance to the block-level down/up-sampling, and study the coding parameters setting of the down-sampled CTUs for pursing frame-level rate-distortion optimization. Their results lead to on average 5.5% BD-rate reduction on common test sequences and on average 9.0% BD-rate reduction on ultra-high definition (UHD) test sequences.

Moreover, Reuze et al. proposed to address the complexity issues and enhance the intra mode signaling in intra-picture prediction in HEVC from another perspective of (Reuze, Philippe, Deforges, & Hamidouche, 2016). The proposed solution introduced new decision tree process by adding new tests and new labels not considered in HEVC. Their solution provided a systematic way to find the best signaling scheme for a given set of data, and the results reduced the BD-Rate by 0.38% in all Intra coding configuration.

Compared with existing efforts that applied machine learning in video encoding, our proposal has the following two unique features: 1) trying to take advantages of superiority of the-state-of-the-art deep CNN technology on image content detection to enhance content-based video encoding; 2) trying to use deep CNN as the primary technique for multiple content-relevant tasks in video encoding within the framework of H.265/HEVC.

We are hoping to explore an innovative field by combining the best of two worlds, and inspire more of practical application in video streaming area.

Chapter V

Experimental Section

Two main concepts, i.e., HEVC and CNN, have been introduced and discussed in detail in previous chapters. HEVC is focused on video compression, and CNN is in the category of machine learning. In this chapter, we will connect these two fields and apply the techniques of CNN into video compression, aiming to increase the encoding speed for both CU partitioning and intra-picture prediction.

Objectives

There are several goals that we will accomplish in this chapter.

- Exam the code logic in the reference software HM in compliance with HEVC standard.
- Obtain the results generated by HM to get the data and labels for CNN training data of CU partitioning and intra-picture prediction.
- Build the CNN framework based on Keras library in Python.
- Tune the weight parameters such as shuffle impact, iteration number, and learning rate.
- Train and test the CNN with data obtained from HM as input and label.
- Predict on CU partitioning and intra-picture prediction using the model trained from CNN.

Implementation in HM Software

In this section, we will mainly focus on the details of CU partitioning and intra-picture prediction and how they are implemented in HM software. Studying the implementation in HM and obtaining the corresponding training data are essentials steps for the following experimental steps.

CU Partitioning in HM

CTU and CU are the most important concepts in our research. The CTU is the basic processing unit similar to MB in prior standards. The size N of the CTUs is chosen by the encoder, ranging from 16 × 16, 32 × 32, 64 × 64, generally with the last one as default. CTU may be too big to decide whether we should perform inter-picture prediction or intra-picture prediction. Thus, each CTU can be differently split into multiple CUs (Coding Units) and each CU becomes the decision making point of prediction type. The size of the CU can range from the same size as the CTU to a minimum size (8 × 8).

As mentioned above, the CTU is further partitioned into multiple CUs to adapt to various local characteristics. But before we discuss how they are split, another important concept should be addressed first: rate distortion ratio. This is the standard that determines how CUs are split. The rate distortion ratio is represented in the formula $J = d + \lambda R$. In this equation, d means distortion, the smaller this value is, the better video quality it has. R indicates rate, and smaller rate means less storage room in video compression. λ indicates LaGrange operator and it's an adjustment parameter. In theory, the best scenario will be smaller rate and smaller distortion, which means smaller J. But d and R are related to each

other, which means smaller d will lead to greater R, and vice versa. All these factors should be included and weighed when we decide how to split a CTU.

For example, for a 32 × 32 CTU, supposedly $CU_{0,0}$ is the optimal splitting mode, then we need to calculate its corresponding rate distortion ratio of $J_{0,0}$. The first subscription of 0 indicates the CU's current depth, and the second 0 indicates that the CU size is the same as CTU size. If the second subscription number is 1, it means CTU will be divided in to 4, and 2 for division into 8 and so forth. Then suppose $CU_{0,0}$ is split into four smaller CUs of size of 16 × 16: $CU_{1,0}$, $CU_{1,1}$, $CU_{1,2}$ and $CU_{1,3}$. Take $CU_{1,0}$ as example first, we need to calculate the cost of rate distortion ratio of $J_{1,0}$. Then $CU_{1,0}$ is supposedly split into four smaller CUs of size of 8 × 8: $CU_{2,0}$, $CU_{2,1}$, $CU_{2,2}$ and $CU_{2,3}$. Now since every CU is 8 × 8, it is already the smallest CU requiring no further split. We have reached the leaf node of the quad-tree.

Then we will need to calculate the cost of rate distortion ratio for each CU of $J_{2,0}$, $J_{2,1}$, $J_{2,2}$, $J_{2,3}$. The sum of $J_{2,0} + J_{2,1} + J_{2,2} + J_{2,3}$ will be compared against $J_{1,0}$, which is the cost of rate distortion ratio before the split. If $J_{1,0}$ is greater than the sum of $J_{2,0} + J_{2,1} + J_{2,2} + J_{2,3}$, then $CU_{1,0}$ will be split into four smaller CUs. Otherwise, CU will retain and need not to be split. The same procedure applies to other CUs such as $CU_{1,1}$, $CU_{1,2}$, and $CU_{1,3}$ to determine if they need to be further split. After $CU_{1,0}$, $CU_{1,1}$, $CU_{1,2}$, and $CU_{1,3}$ have all been determined, we should recursively track back up to the parental level of these 4 CUs. The cost of rate distortion ratio of $J_{0,0}$ will be compared against the sum of $J_{1,0} + J_{1,1} + J_{1,2} + J_{1,3}$ to decide if $CU_{0,0}$ needs to be further split based on the same principle. Such recursive procedure will be repeated for all the CUs in a CTU, as well as for all CTUs in one fame,

and finally the optimal split mode for each CU within one video sequence will be obtained. This processing order of CUs can be interpreted as a depth-first traversing in a Zig-Zag order in the coding tree structure as shown in Figure 21 below. Figure 22 also illustrates a broader view of how CTU partitioning works within a frame.

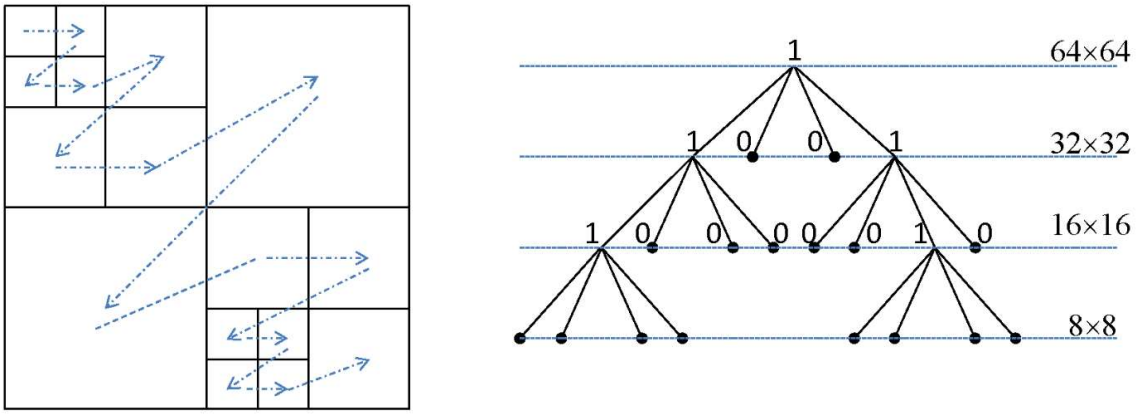


Figure 21. Example of CTU partitioning and processing order when size of CTU is equal to 64×64 and minimum CU size is equal to 8×8. (a) CTU partitioning. (b) Corresponding coding tree structure *(Kim, Min, Lee, Han, & Park, 2012)*

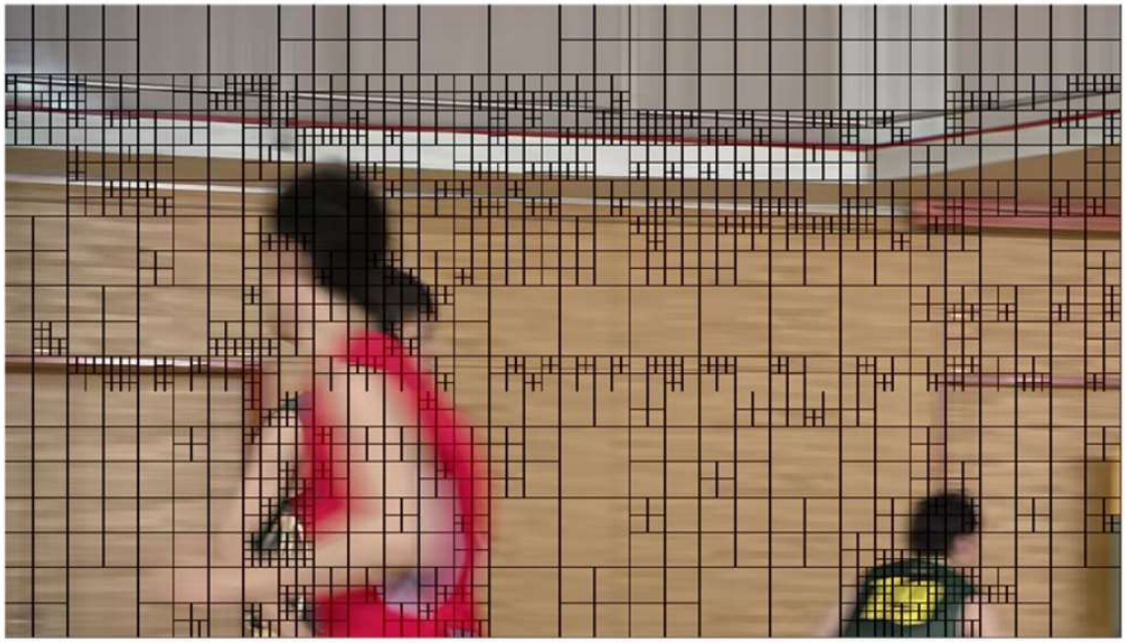


Figure 22 Illustration of CTU splits into CUs in one frame *(Zhang, Zhai, & Liu, 2017)*

Intra-prediction in HM

The intra-picture prediction uses the previously decoded boundary samples from spatially neighboring block in order to predict a new prediction block PB. So the first frame of a video sequence and the first picture at each clean random access point into a video sequence are coded using only intra-picture prediction. HEVC employs 35 different intra modes to predict a PB (prediction block) including 33 angular modes, 1 planar mode and one DC mode. All the intra prediction modes use the same set of reference samples, which are extracted at the boundary from the upper and left blocks adjacent to the current PU. For the diagonal directions the top-left corner sample may also be used. Also, it is possible to use the lower left and above right, if they are available from preceding decoding operations.

Specifically, for luma components, HEVC supports 5 different Pus: 4 × 4, 8 × 8, 16 × 16, 32 × 32, and 64 × 64, and each of PU size can have 35 prediction modes.

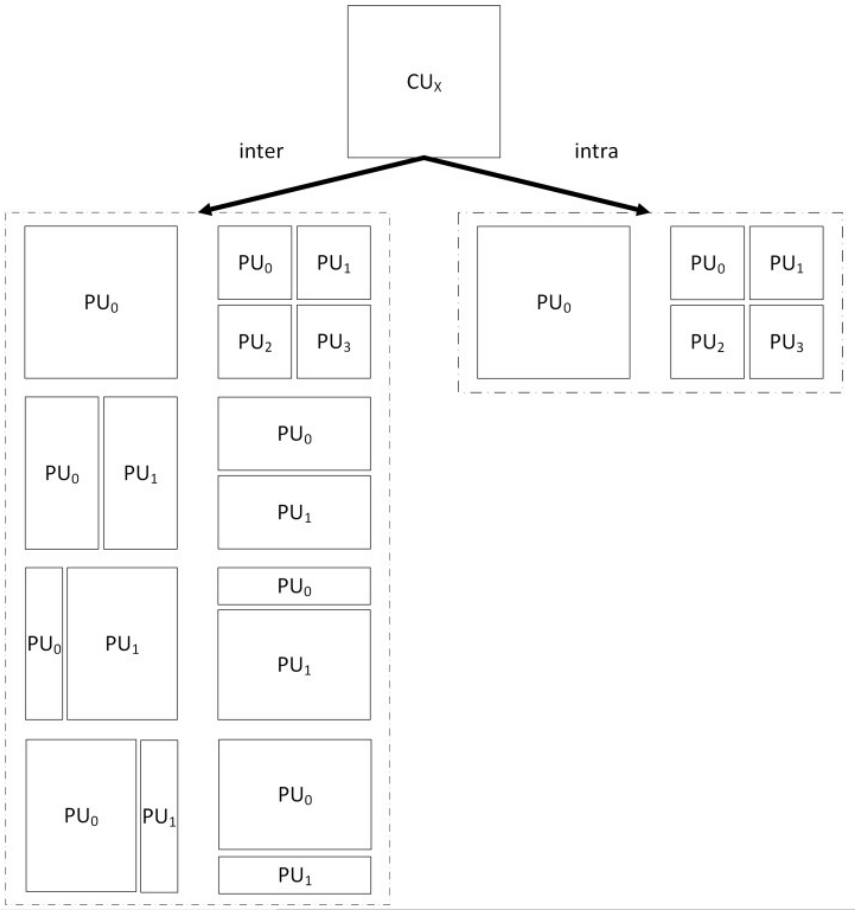


Figure 23. Organization of H.265/HEVC syntax of CU into PUs *(Abramowski, 2016)*

A typical intra-prediction procedure includes 3 steps: 1) determine the availability of reference pixels adjacent to the current PU and process accordingly; 2) filtering reference pixels; 3) calculate the predicted pixel value of the current PU from the filtered reference pixels. For example, reference samples are denoted as $R_{x,y}$ with (x,y) having its origin one pixel above and to the left of the block's top-left corner. Similarly, $P_{x,y}$ is used to denote a predicted sample value at a position (x,y), as illustrated in Figure 24.

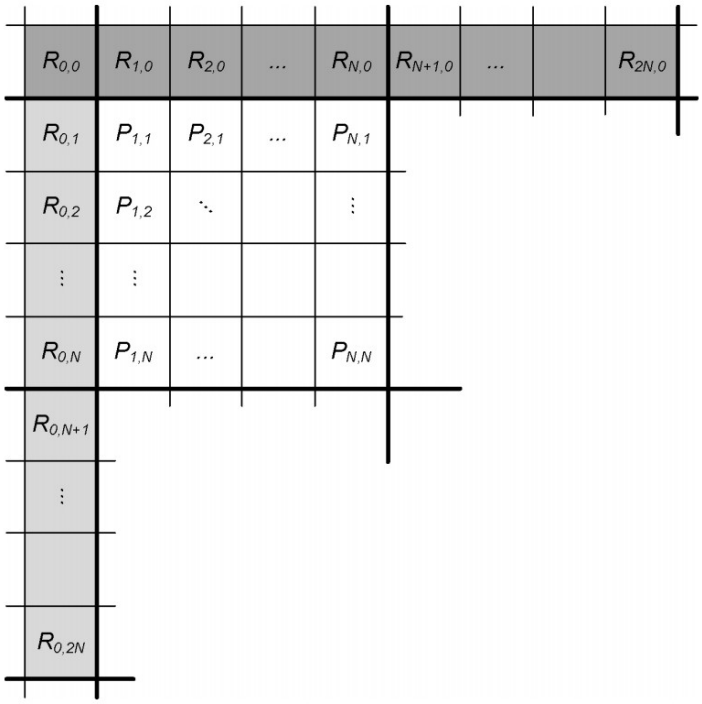


Figure 24. Reference samples Rx,y used in prediction to obtain predicted samples Px,y for a block of size N × N samples *(Lainema, Bossen, Han, Min, & Ugur, 2012)*.

Neighboring reference samples may be unavailable for intra-picture prediction, for example, at picture or slice boundaries, or at CU boundaries when constrained intra prediction is enabled. Missing reference samples on the left boundary are generated by repetition from the closest available reference samples below (or from above if no samples below are available). Similarly, the missing reference samples on the top boundary are obtained by copying the closest available reference sample from the left. If no reference sample is available for intra prediction, all the samples are assigned a nominal average sample value for a given bit depth (Lainema, Bossen, Han, Min, & Ugur, 2012).

The intra sample prediction process in HEVC is performed by extrapolating sample values from the reconstructed reference samples utilizing a given directionality. In order to

simplify the process, all sample locations within one prediction block are projected to a single reference row or column depending on the directionality of the selected prediction mode (utilizing the left reference column for angular modes 2 to 17 and the above reference row for angular modes 18 to 34). For arbitrary number of directions: each predicted sample *Px,y* is obtained by projecting its location to a reference row of pixels applying the selected prediction direction and interpolating a value for the sample at 1/32 pixel accuracy. Interpolation is performed linearly utilizing the two closest reference sample as illustrated in equation below, where w_y is the weighting between the two reference samples corresponding to the projected subpixel location in between $R_{i,0}$ and $R_{i+1,0}$, and >> denotes a bit shift operation to the right.

$$Px,y = ((32 - w_y) \cdot R_{i,0} + w_y \cdot R_{i+1,0} + 16) >> 5 \quad (1)$$

Reference sample index i and weighting parameter w_y are calculated based on the projection displacement d associated with the selected prediction direction as in the following equation 2, where & denotes a bitwise AND operation.

$$\begin{aligned} c_y &= (y \cdot d) >> 5 \\ w_y &= (y \cdot d) \,\&31 \\ i &= x + c_y \end{aligned} \quad (2)$$

It should be noted that parameters c_y and w_y depend only on the coordinate y and the selected prediction displacement d. Both parameters remain constant when calculating predictions for one line of samples and only equation (1) needs to be evaluated per sample basis. When the projection points to integer samples, the process is even simpler and consists of only copying integer reference samples from the reference row. Both equations

(1) and (2) define how the predicted sample values are obtained in the case of vertical prediction (modes 18 to 34) when the reference row above the block is used to derive the prediction. Prediction from the left reference column (modes 2 to 17) is derived identically by swapping the *x* and *y* coordinates in (1) and (2) (Lainema, Bossen, Han, Min, & Ugur, 2012).

The set of Most Probable Modes (MPM) is typically created based on the modes chosen for left and top neighbor CUs. Its cardinality must be three, assuming $S = \{M_1, M_2, M_3\}$, so it is complemented by the first non-present mode from planar, DC and the mode corresponding to the vertical direction. The only exception occurs when both neighbors' optimal modes are equal and angular. In such situation, MPM consists of this mode and two modes representing directions adjacent to it. An intra prediction mode is written to the stream as a flag, indicating whether it is included in MPM, and the value. The value identifies the specific mode either inside or outside the MPM set.

Experiment Methods

The abovementioned sections discussed the principle and the procedures of how CTUs are split (i.e., partitioned) and how intra-picture prediction implements in reference software HM in HEVC. The problem is such procedures require a lot of recursion and it introduced too much computational complexity. In a real world scenario such as streaming services, it will be insufficient to encode/decode at a low speed. So in this paper, we proposed to adapt the most popular deep learning techniques to replace traditional split

procedure of CTUs, as well as intra-picture prediction implementations. Among all the deep learning techniques, we chose CNN due to its high efficiency and convenience.

Convolutional Neural Network (CNN) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. A CNN is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. A CNN is easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

We downloaded 13 videos samples as the training and testing data. For each sample video, it is split into 80% as training dataset and 20% as testing dataset. Every video size is 352 × 288. Then every frame of the video is divided into CTU with size of 32 × 32. For a training data includes total of 3150 frames, and each frame has 99 CTUs, there are over 300 K+ CTUs available for training. The testing data has 260 frames, giving us 25K+ CTUs.

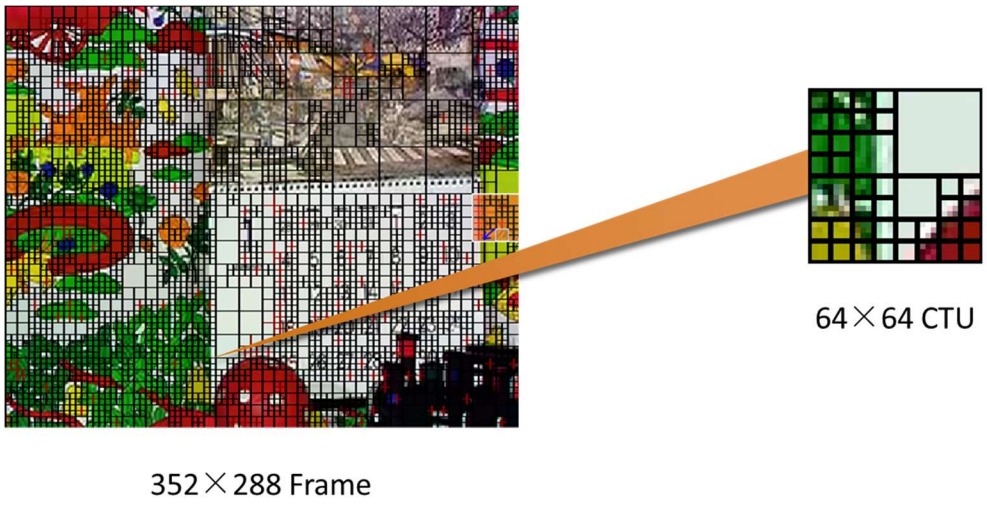


Figure 25. Illustration of one frame divided into CTUs in experiment *(Abramowski, 2016)*

For a typical frame in our experiment, it will be divided into 99 CTUs. Basically, for CU split pattern experiment, firstly, we extract the luma values for each CTU as the input for CNN, and then use the depth values, i.e., the split pattern as the output in this CNN. In our research, we mainly utilized the famous example: MNIST. It is a great dataset for getting started with deep learning and computer vision. The dataset consists of pair, "handwritten digit image" and "label". Digit ranges from 0 to 9, meaning 10 patterns in total. For handwritten digit image: each one is gray scale image with size 28 × 28 pixel. And for label: each is actual digit number this handwritten digit image represents. It is either 0 to 9. The training dataset will be trained against each corresponding label, until then the trained model will predict label for new incoming data with high accuracy.

Our experiments are designed in such way that luma values for each CTU will be used as aforementioned "handwritten digit image" (i.e., the input), and split pattern/intra prediction mode will be used as "label" (i.e., the output). For CU split pattern, there are total of 17 patterns ("labels") for a 32 × 32 CTU with minimum of 4 × 4 CU. For intra prediction modes, there are total of 35 possibilities ("labels"). Also in a similar way, the input data will be trained to generate a compiled model that will be used to predict the labels for new inputs.

One of the most powerful and easy-to-use Python libraries for developing and evaluating deep learning models is Keras. It wraps the efficient numerical computation libraries Theano and TensorFlow. The advantage of this is mainly that you can get started with neural networks in an easy and fun way. In our experiment, we use Python as the language and Theano as the backend.

The basic principles of designing convolution layers, hidden layers, fully connected layers etc. are mainly based on what has been implemented in MNIST with Keras. We included two convolution layers, one max pooling layer after each convolution layer (note that max pooling layer is removed in intra-picture prediction). There are several parameters that are fine-tuned for both split pattern and intra mode predictions, as will be discussed in the next section.

Results and Discussion

CU Partitioning Pattern

For this research, the skeleton of the model we trained has 2 convolution layers and each is followed by a max pooling layer. After the fully connected layer, the model will be fed with training data and begin the fit process. During building the model, there are several parameters involved: max pooling, dropout, number of convolution layers, and shuffle. We investigated all of them, and determined their influences on the training results and accuracy.

Shuffle.

Shuffle is a parameter in Keras fit function. It is boolean (whether to shuffle the training data before each epoch) or str (for 'batch'). 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Generally, if data are highly correlated, (e.g., every class in order), shuffle is needed. you need to shuffle or your training results will be bad. But if your dataset is already shuffled, then setting it to false is logical.

We have experimented with this parameter with all 7 video clips in our research. We take the results of 3 representatives out of 7 clips and they are shown below.

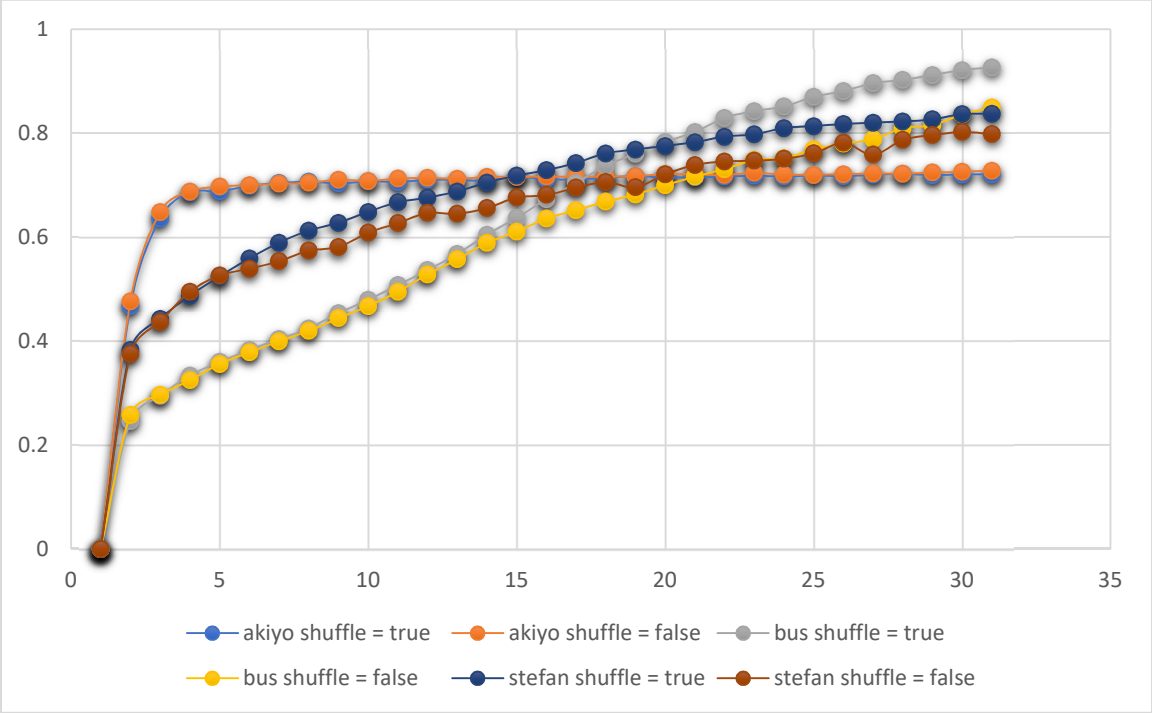


Figure 26. Shuffle impact on training accuracy

By comparing the results with shuffle as True with those of shuffle as False, it can be observed that the testing accuracies are generally higher when shuffle is set as True. The trend is not that obvious for Akiyo video clip since the accuracy would reach about 70% and there is no major difference between true/false shuffle parameter. But for Bus and Stefan video clips, it is quite obvious that the accuracy is higher when shuffle is true. Similar trend is also observed for the rest 4 video clips. By default, in Keras, shuffle is set as true for training dataset but not the validation dataset. So we will leave it as true for the rest of our research experiments.

Max Pooling.

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping sub-regions of the initial representation.

In our experiments of split pattern prediction, max pooling is employed after each layer by using a 2 × 2 filter. This is because similar accuracy will be achieved with or without employing max pooling layer, but it only needs about half of execution time with max pooling layer in place. Similar trends are observed in other clips, and we take Akiyo clip as an example and summarized below.

time of each epoch (s)	10	23
accuracy of 30 epochs (%)	70.8	71.4

Table 2. Comparison of execution time for max pooling

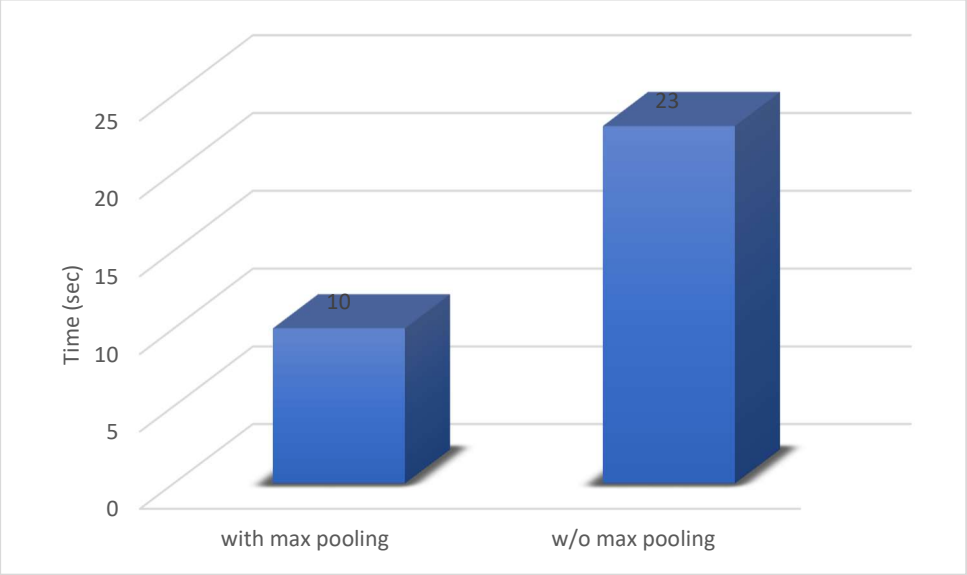


Figure 27. Training time comparison with/without pooling

Dropout.

A simple and powerful regularization technique for neural networks and deep learning models is dropout. It aims to reduce the complexity of the model with the goal to prevent overfitting. During a dropout, certain units (neurons) in a layer are randomly deactivated with a certain probability. So, if half of the activations of a layer is set to zero, the neural network won't be able to rely on particular activations in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations; the network can't rely on the particular neurons and the combination (or interaction) of these to be present. Also training will be faster with dropout set.

We also tuned this parameter in our experiments. 3 different dropout ratios are chosen: 0.5, 0.25, and 0.1. The comparison results are summarized as below. As we can see from the chart that when dropout ratio is set to 0.5, the accuracy will be much lower than the other two. When dropout ratio is changed to 0.25 and 0.1, the accuracy is significantly

improved. It is indicated that the accuracy will be highest when dropout ratio is set to be 0.1.

We also considered the execution time difference, since it takes shorter time for a higher dropout ratio as mentioned before. In our experiment, such difference is marginal among these 3 candidates, (i.e., ~10s per epoch across the board). Therefore, we set dropout ratio as 0.1 by taking all the factors into account.

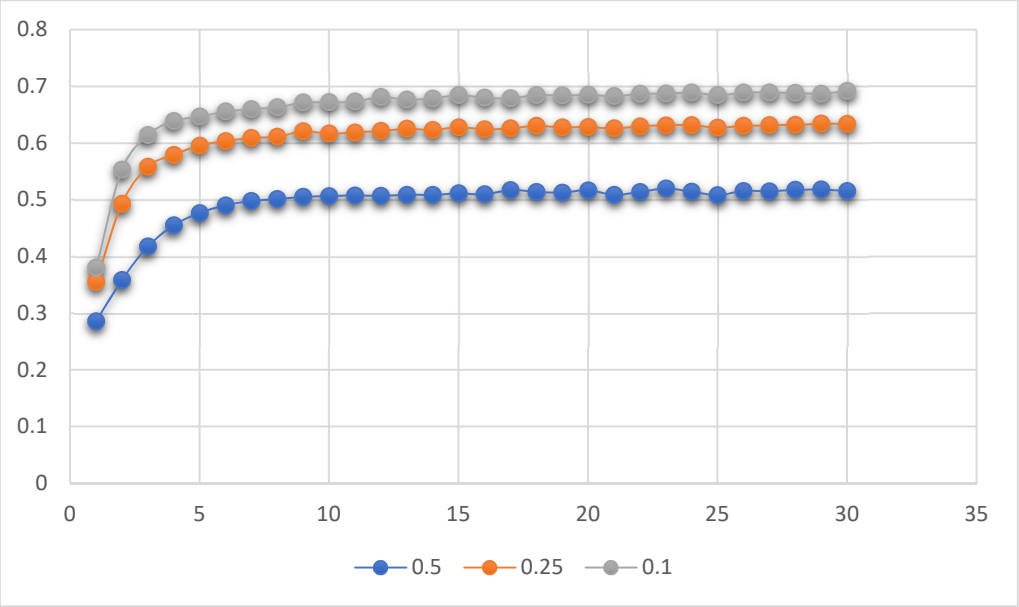


Figure 28. Dropout ratio impact on accuracy for Akiyo sample

It should also be noted that dropout is only applied during training, and we need to rescale the remaining neuron activations. Specifically, if 50% of the activations in a given layer is set to zero, we need to scale up the remaining ones by a factor of 2. Finally, if the training has finished, the complete network should be used for testing.

Number of Layers.

In deep learning, selection of optimal number of layers and neurons is also one of hyperparameters that can be fine-tuned. A method is to add layers until it starts to overfit the training set. Then it is time to add dropout or another regularization method. The idea is that once your network overfitting you're sure that it is powerful enough for your task. The dropout helps to prevent feature co-adaptation and therefore avoid over-fitting. The number of neurons in each layer is not really sensible. Usually a bit more or as much neurons should be put on the first layer than inputs, and the number should decrease slowly until the last layer.

Ideally, the best method is to use someone else's architecture and adjust from there. But there isn't a paper that fits our problem, then we have to create our own model and fine-tune the number of layers. This is the painstaking process of a righteous field that is more empirical than theoretical.

In our research, we adopted the method of trial and error. We tried different combinations of parameters and keep the one with the lowest loss value or better accuracy on the validation set. We tried two possibilities of 2 and 3 fully connected layers. For 2 layers, it has 512 and 17 hidden units for the fully connected layers. Whereas for 3 layers, it has 512, 128, and 17 hidden units for each layer. The results are summarized below.

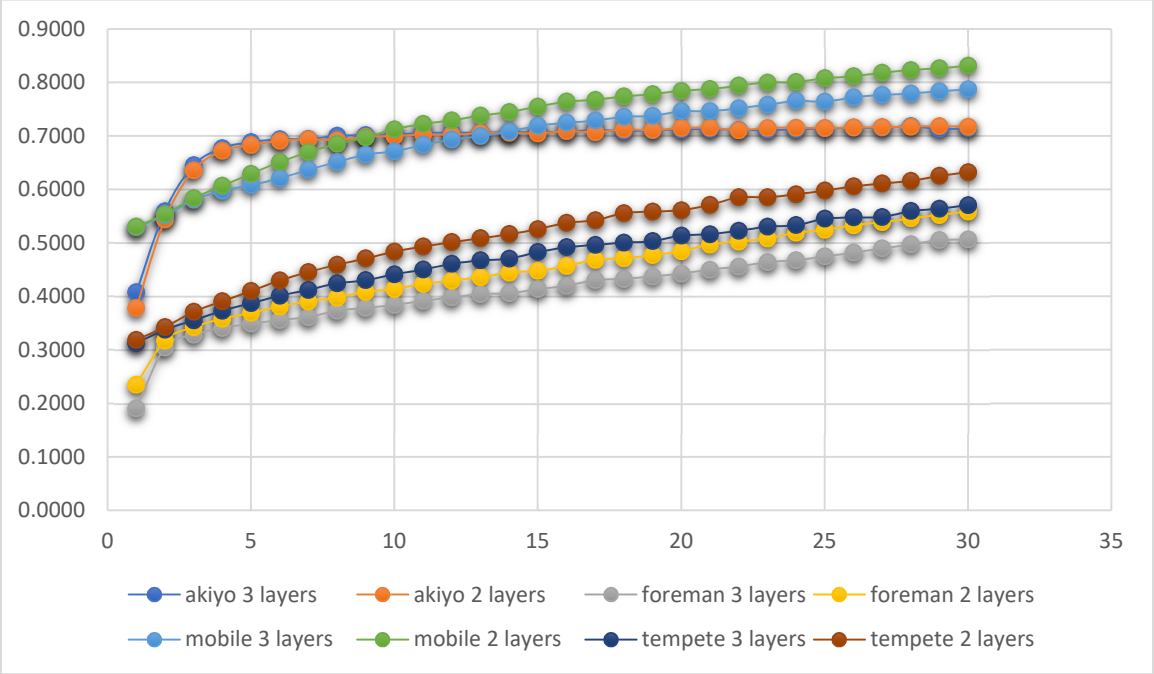


Figure 29. Accuracy comparison between different number of layers

From the figure above, we can clearly see that for the dataset that we trained, the accuracy is almost the same for Akiyo clip, but for the other samples it is slightly higher when we use 2 fully connected layers compared to 3 layers. It might be because 2 layers is sufficient to handle the dataset without sacrificing the accuracy.

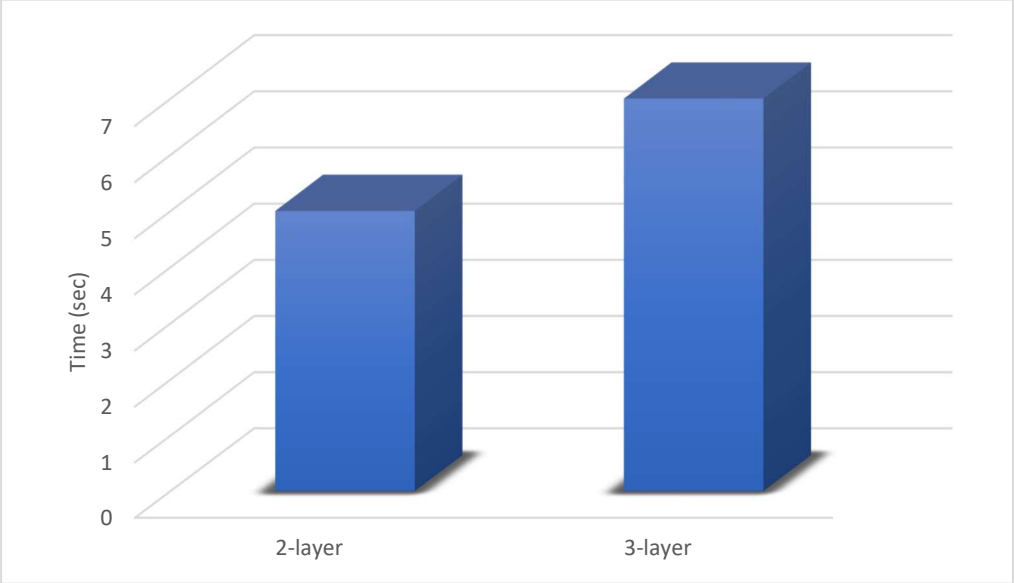


Figure 30. Comparison of training time per epoch

Another benefit using less fully connected layers is less amount of execution time. In our case, average execution time for 3-layer is 7 seconds per epoch, and it is about 5 seconds for 2-layer per epoch, which results in 28% decrease of time when we choose 2-layer. Therefore, we will use 2 fully connected layers across our experiments.

Training and Testing Results.

After fine tuning the parameters as discussed above, we have determined these parameters and settle down on the CNN's structure shown in the figure below.

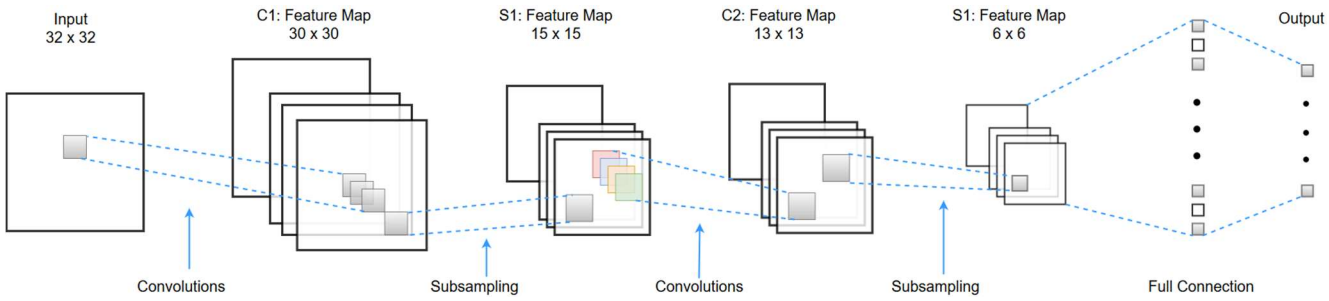


Figure 31. CNN structure in this research

Specifically, the network includes 2 main layers, and each main layer has 1 convolution layer and 1 max pooling (i.e., subsampling) layer. For each convolution layer, we use 32 of 3 × 3 filters to extract the feature map. The activation function is ReLU across the board. And the border mode is set as “valid”. The “valid” means there is no padding around input or feature map, while “same” means there are some padding around input or feature map, making the output feature map's size same as the input's. In our research, there two don’t make difference when used to build the model and train the data. For max pooling layer, the pool size is 2 × 2 with strides of 2 × 2. The border mode is also set to be ‘valid’, just the same as in convolution layer. After two main layers, it finally comes with the fully connected layers where the hidden units will be “flatten” and directed to the output of 17 labels.

In our experiment, the inputs are 32 × 32 CTU. For a typical sample video clip with 300 frames and each frame with size of 352 × 288 pixel, there will be 11 × 9 × 300 = 29700 CTUs generated by reference software HM and available to use. Specifically, the input will be the matrix of pixel value of 32 × 32 CTU as demonstrated in figure below. During the process of building the model, such big data set will be split into 80/20 portion of training/test data set. Therefore, the test data set can be used to validate the training results.

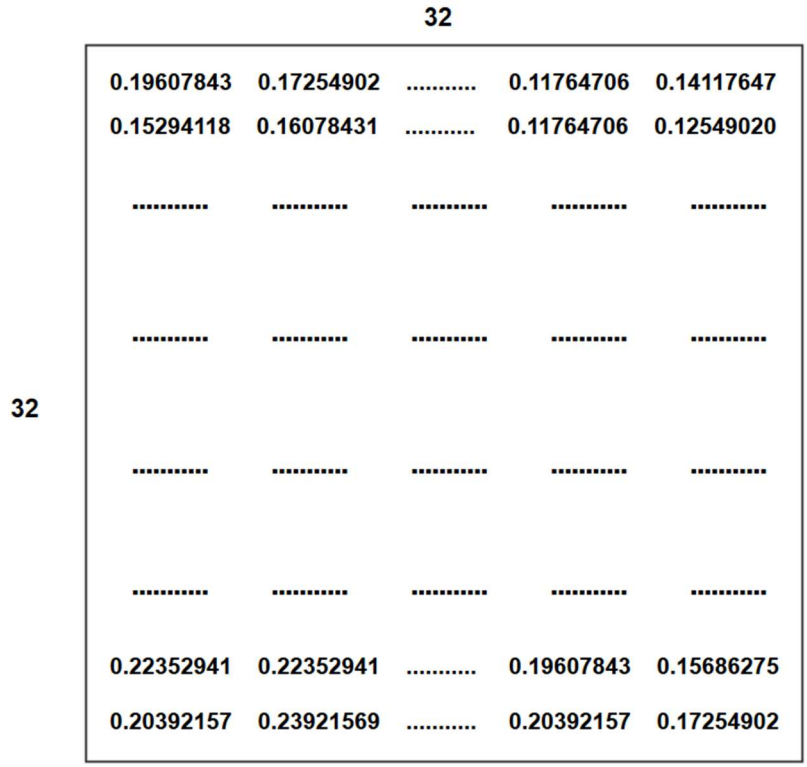


Figure 32. Illustration of 32× 32 CTU from sample video clip

For output, there are total of 17 possible combinations for a 32 × 32 CTU, and will be explained in detail. At the top level, it is one combination per se, meaning no split. Then drilling one level down, the CTU will be split evenly into 4 smaller CTUs of 16 × 16 in size, which is another combination. Furthermore, each of 16 × 16 CTU can be decided at this point whether it needs to be split into even smaller one or not. For 4 of these 16 × 16 CTU blocks, each one has two possibilities of split or not, and therefore there are total of $2^4 = 16$ combinations. However, considering one of these 16 combinations is that none of these 4 CTU blocks will split, which is the case mention already previously. Therefore, for a 32 × 32 CTU block, there are total of 1 + 1 + 16 – 1 = 17 combinations. The possible combinations are illustrated in the figure below.

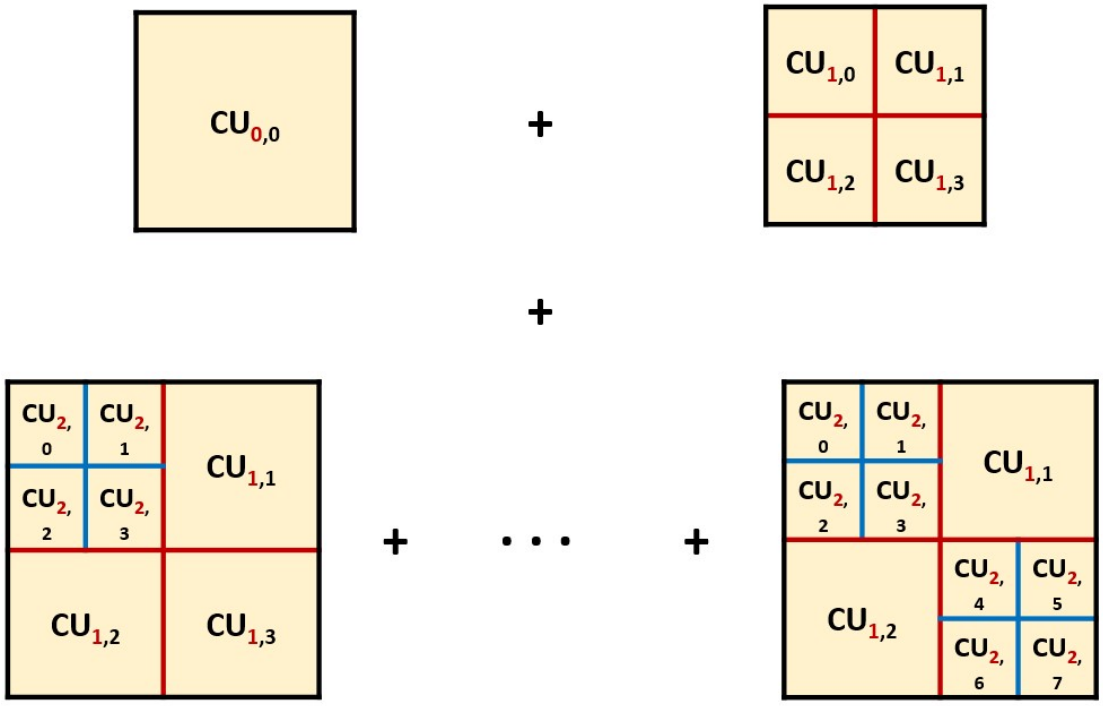


Figure 33. Illustration of possible split patterns

In a typical training process, the input data will be fed into the Keras model with the parameters configured as mentioned before. The number of training iterations has been tested with different values. For some sample video clips, only 50 iterations (i.e., ~ 25 min) can yield ~ 90% training accuracy. For example, videos of bus, mobile, flower etc. can quickly achieve high training accuracy as summarized below. The reason may be because these video clips do not have many variations in terms of pixel changes and ranges. So the model can quickly learn the relationship between the input pattern and split pattern with relatively high accuracy.

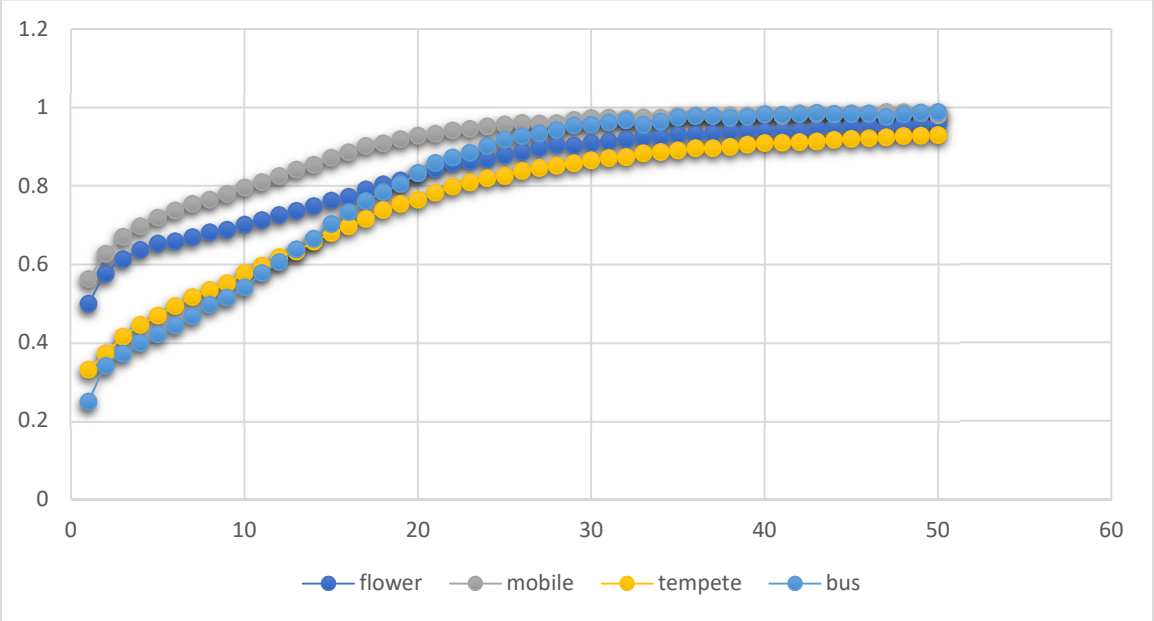


Figure 34. Training accuracy of ~ 90% in 50 cycles

For most of the sample video clips, it generally requires about 100, sometimes even 200, iterations to achieve ~ 90% training accuracy. For example, video clips of container, hall, Stefan etc. will achieve training accuracy above 90% only after 200 iterations. It is suspected that these video clips are more complex in terms of pixel variations and ranges. Therefore, it is harder for the model to connect the dots between input pattern and split pattern, and requires more time and repetition to correct itself.

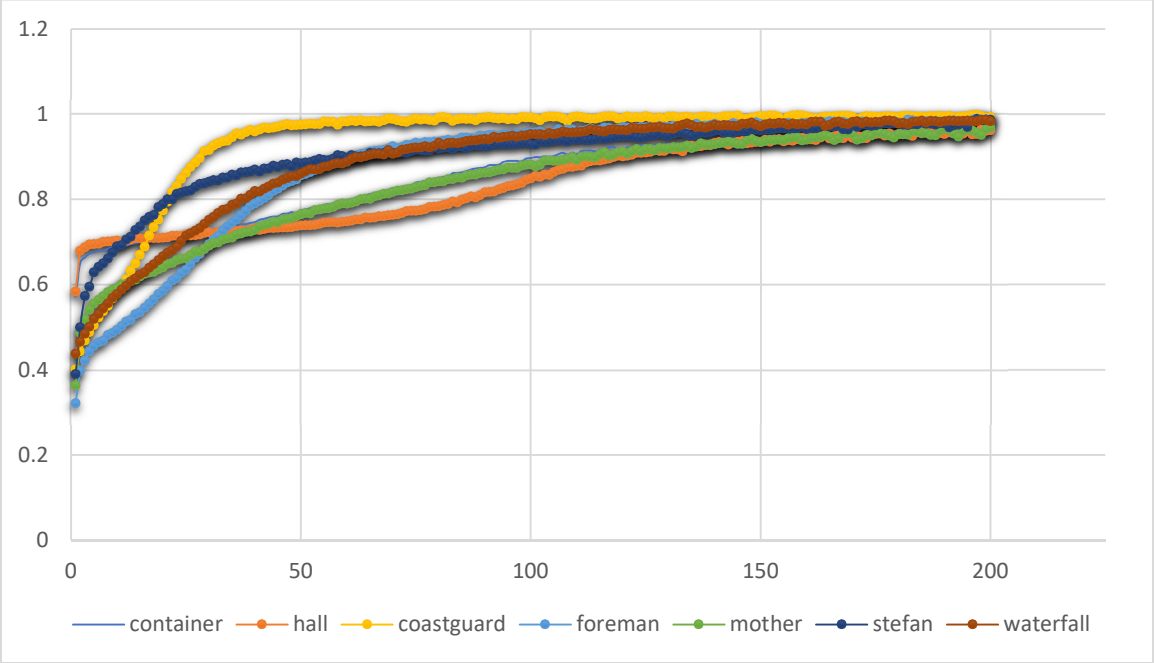


Figure 35. Training accuracy of above 90% in 200 cycles

Testing accuracy is also in alignment with the training accuracy. Testing results on most of the video sample clips achieved reasonably high accuracy of ~ 90%. Only the samples of akiyo, silent, and news have relatively lower accuracies of 77.8%, 82.4%, and 73.8%.

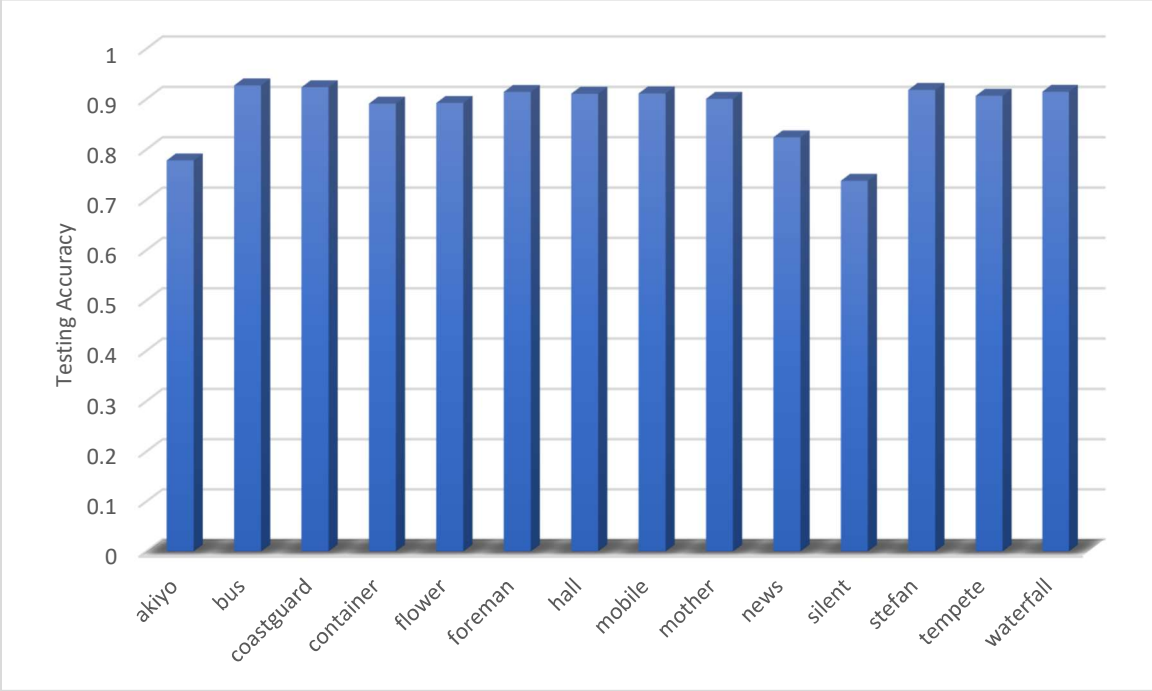


Figure 36. Comparison of testing accuracies for different sample video

It was initially suspected that overfitting may have been involved in the training process. However, examining the data of training accuracy carefully indicated that these 3 sample video does not generate good training results in the first place, all of which are around 80%. The training accuracies got stile and further training does not yield significant increase of accuracy. Considering the sample video clips in our research only have 30 frames, which is a relatively small sample pool, it suggests that future training with a larger sample video including more frames should help to improve the testing accuracy.

In addition to the good training accuracy against the sample video clips, the speed of split one CTU has been improved significantly after training the model. The split pattern of a specific CTU in the reference software HM is exhaustive and recursive. It means HM will calculate the rate distortion cost of every possible split pattern (i.e., 17 combinations in total), and further compare these cost values recursively to obtain the best split pattern

based on the minimum rate distortion cost. Understandably, it is a very time-consuming process, and it will be unrealistic to compress and transfer videos of ultra-high definition integrating H.265 standard in a real-time streaming manner. There will be a lot of overhead upfront that is unbearable and unfortunate to apply HEVC new standard in a broader spectrum. Therefore, we introduced the CNN and the training model with the hope of reducing the time to obtain the best split pattern for a specific CTU. Our results indicated that, after the training, within average of 400 microseconds (i.e., 10^{-6} second) the model is able to predict the split pattern for one specific CTU with over 90% accuracy. It is significantly improved from 4000 microseconds per CTU in HM, which is obtained recursively.

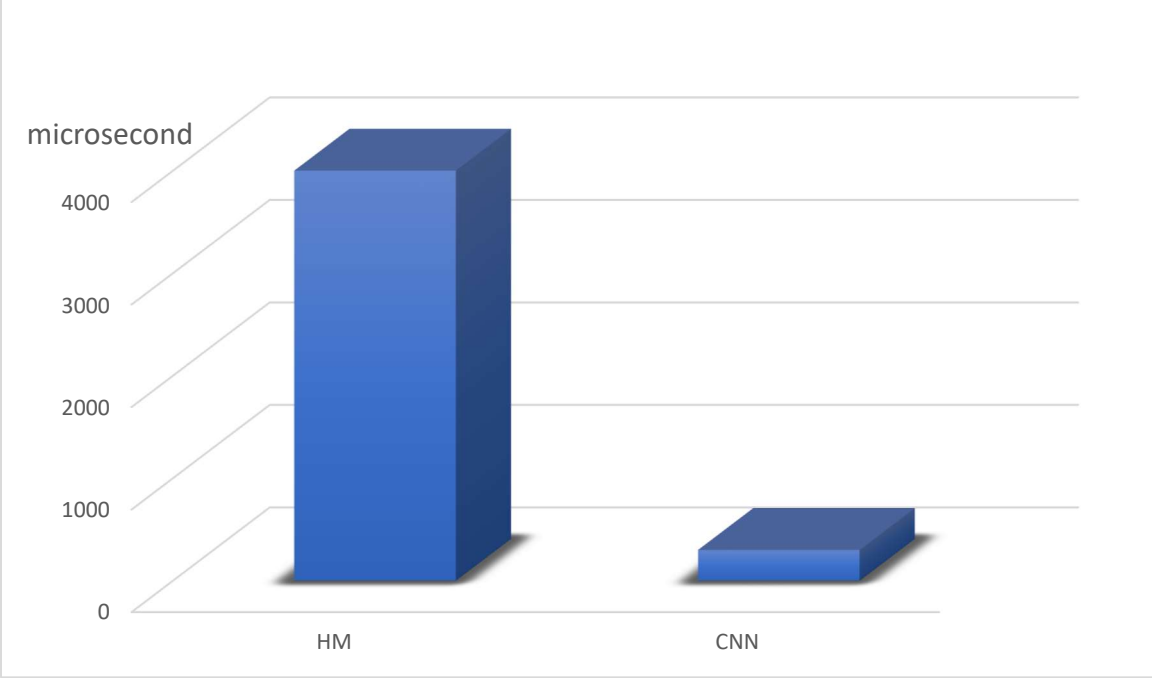


Figure 37. Speed of determining split pattern before and after CNN

Intra-picture Prediction

After it is proved that applying CNN in predicting split pattern of CTU yield satisfactory results, we moved further and expand our research to intra prediction. Although we plan to use similar methodology as we used in split pattern prediction, there are several differences between split pattern and intra-picture prediction. The first one is we have more output possibilities in intra-picture prediction (i.e., 35) compared with 17 in split pattern. This difference requires us to make corresponding code changes in Keras. Another significant difference is input size. Although the input CTU size is determined by HM stage and is set to 32 × 32 initially in intra prediction, the same as split pattern, it needs to be further divided into smaller sizes. This is related to the nature of intra prediction data, which is 4 × 4 matrix for one 32 × 32 CTU. Most often these 4 values in the matrix differ from each other, and they cannot be considered as a whole to represent the actual intra prediction mode. Therefore, every block in such 4 × 4 matrix will be singled out and extracted as 1 value. After adapting this method to process the output data, we also need to divide the input 32 × 32 CTU into 4 sections accordingly, yielding 8 × 8 CTU blocks.

Furthermore, because intra-picture prediction will take into account the neighboring pixels around current CTU block, we need to further modify 8 × 8 CTU and expand it to accommodate neighbors based on different positions of current CTU as illustrated in Figure 38.

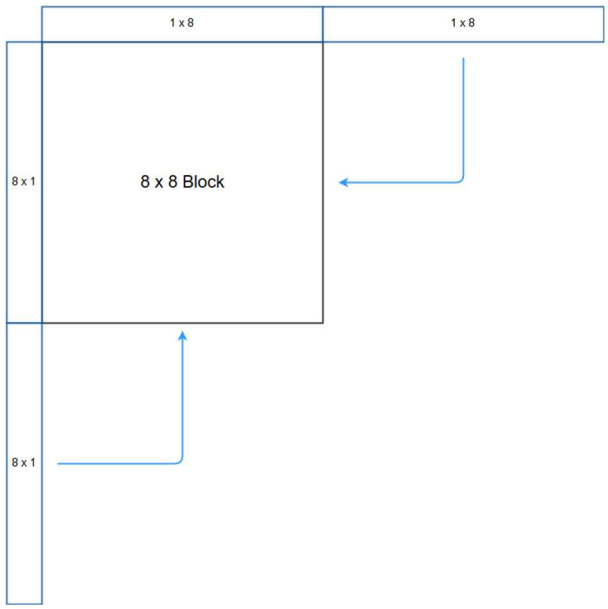


Figure 38. Expanding CTU with neighboring blocks in intra-picture prediction

In general, four neighboring strips of pixels: top, top right, left, lower left will be padded to the current CTU block. Among them, top and left strips will be added directly without any modification. Top right and lower left will be "bent and folded" to be padded to the current CTU block. Then there will be four corners that are left blank, and they will be arbitrarily assigned to value of 0.5. The general guidance for padding CTU in intra prediction is outline above, but there are several special scenarios that need special attention, as illustrated in Figure 39.

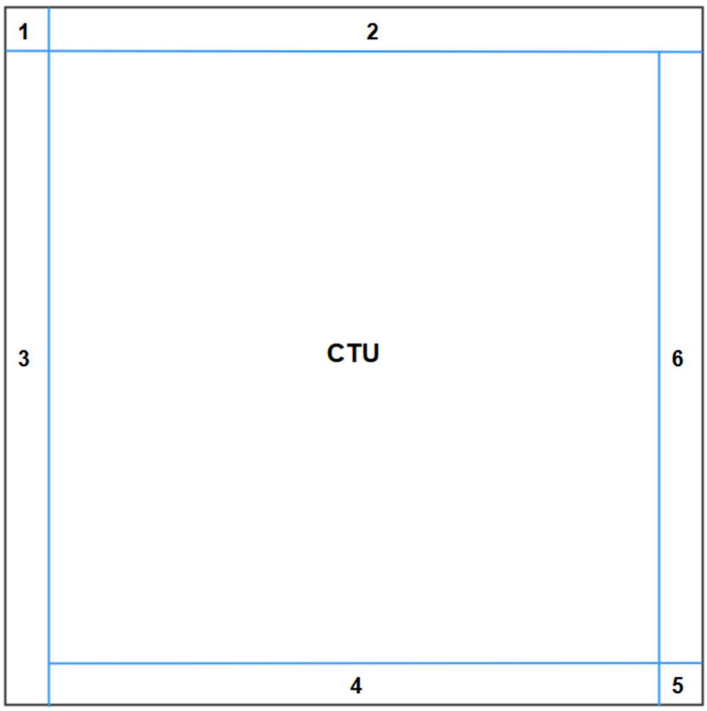
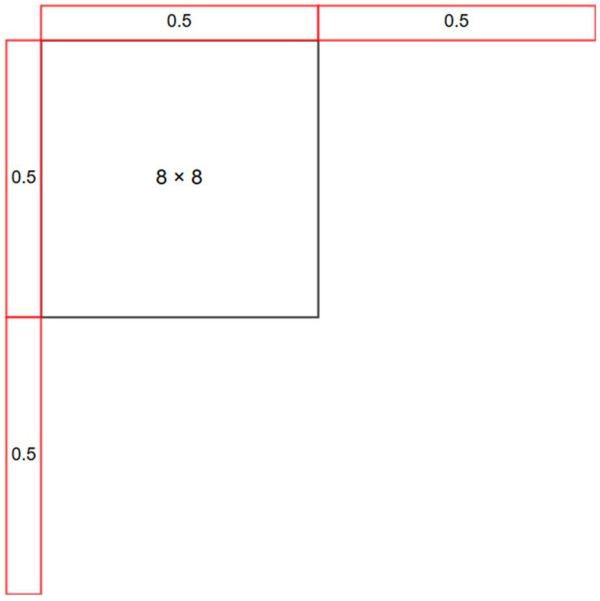


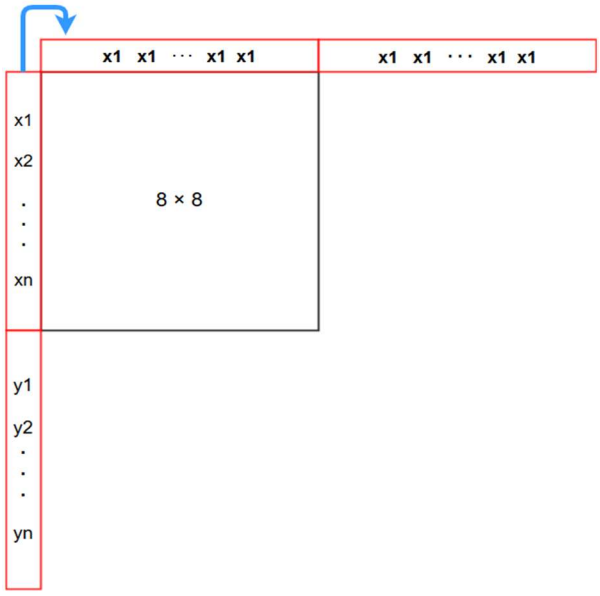
Figure 39. Special locations in a CTU

Location 1 is when current sub-block is located at the top left corner of parent 32 × 32 CTU. In this case, all the neighboring blocks (i.e., the left and lower left, top, and top right strips) do not exist, and they will be assigned to value of 0.5 across the board. Location 2 is all but the first block in the first row of parent CTU. In this case, only the top and top right strips do not exist, and they will be assigned to the top pixel value of the left 1 × 8 strip. Location 3 is all the sub-blocks in the first column except the one in the first row. They do not have left and lower left strips. And they will be padded by the first pixel of top strip. Location 4 is last row except first and last sub-blocks. Since they don't have lower left strip, which will be padded by last pixel value of left strip. Location 5 is bottom right sub-block, it does not have top right and lower left, and will be padded with last pixel value of left strip and top strip. Location 6 is last column except the first and last sub-block, they don't have top right strip, and will be padded by last pixel value of the top strip. All of

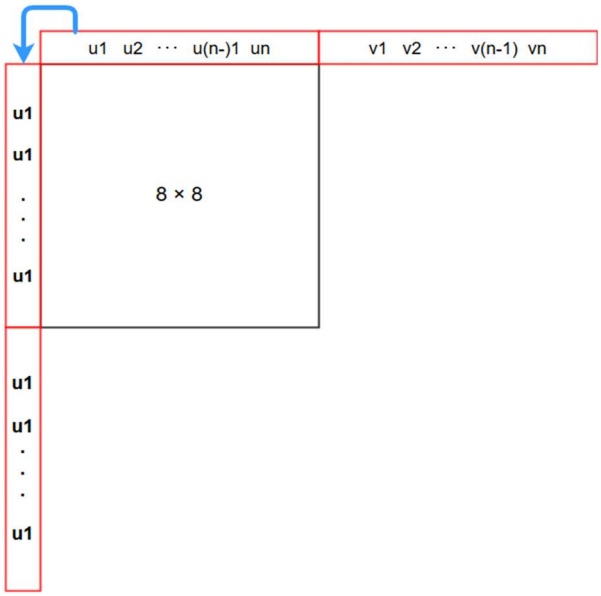
these mentioned special locations and their related padding patterns are summarized in Figure 40.



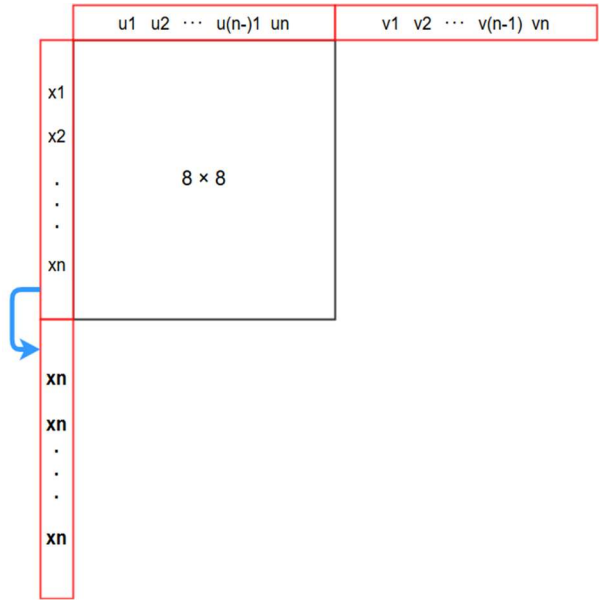
Location 1



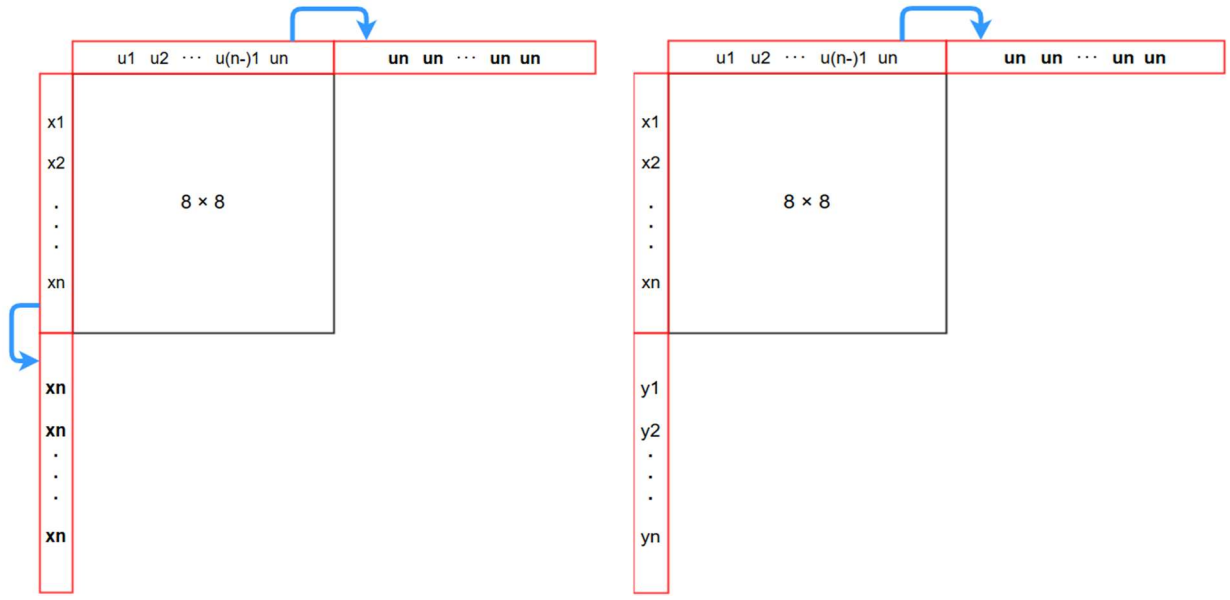
Location 2



Location 3



Location 4



Location 5 Location 6

Figure 40. Summary of special locations in CTU padding patterns

In brief, one specific 32 × 32 CTU will be divided into 4 of 8 × 8 blocks, and each of these 8 × 8 sub-blocks will be padded with neighboring pixels according to its special location, which will yield 10 × 10 sub-blocks. These expanded (i.e., padded) blocks will be used as input data source, and corresponding intra prediction result will be used as output data source for later training purpose.

CNN with only Fully Connected Layers.

We used similar Keras model as mentioned before in application of intra-prediction. However, in our preliminary experiments, the training results didn't yield good results, and the training accuracy quickly converged to about 30%. We thought it might be related to the parameters in the model, and we have tried to tune the parameters in all various combinations, but these attempts didn't not help and the accuracy still converged very quickly.

After careful analysis and examination, we suspect that since the input size is only 10 × 10, if we use max pooling layers in between convolution layers, the input size will be too small for further convolution. In other words, no matter how we tune the parameters, such as the number of filters, it does not make any difference at all because convolution is not able to extract the features and does not give useful higher-level information any more. Also the purpose of introducing max-pooling or mean-pooling layers is to reduce the noise and minimize the influence of overfitting, because the training accuracy is sensitive to some of the input errors.

Therefore, we first changed to try with residual network. Residual network was chosen to be tested simply because it does not have the sub-sampling pooling layers, and is very well-known for its ability to extend to deep network without sacrificing the training accuracy. We won't go into too much of detail about ResNet in this paper, but it's necessary to cover some basics of it.

With network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error. Instead of hoping each stack of layers directly fits a desired underlying mapping, we explicitly let these layers fit a residual mapping. The original mapping is recast into $F(x)+x$. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers (He, Zhang, Ref, & Sun, 2016).

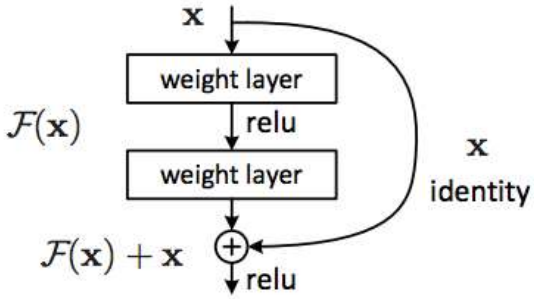


Figure 41. Residual learning: a building block *(He, Zhang, Ref, & Sun, 2016)*

Therefore, we had experimented with ResNet that was used in MNIST (i.e., digit recognition) (Kweon, 2016). Video clip of bus was picked as the sample, we ran the training process for up to 600 iterations, but the training accuracy reached ~ 46% and stayed stable as is. Actually, the accuracy reached ~ 35% after 50 iterations, and only slightly increased 11% between 50 and 600 iterations. So we don't use ResNet in the following experiments.

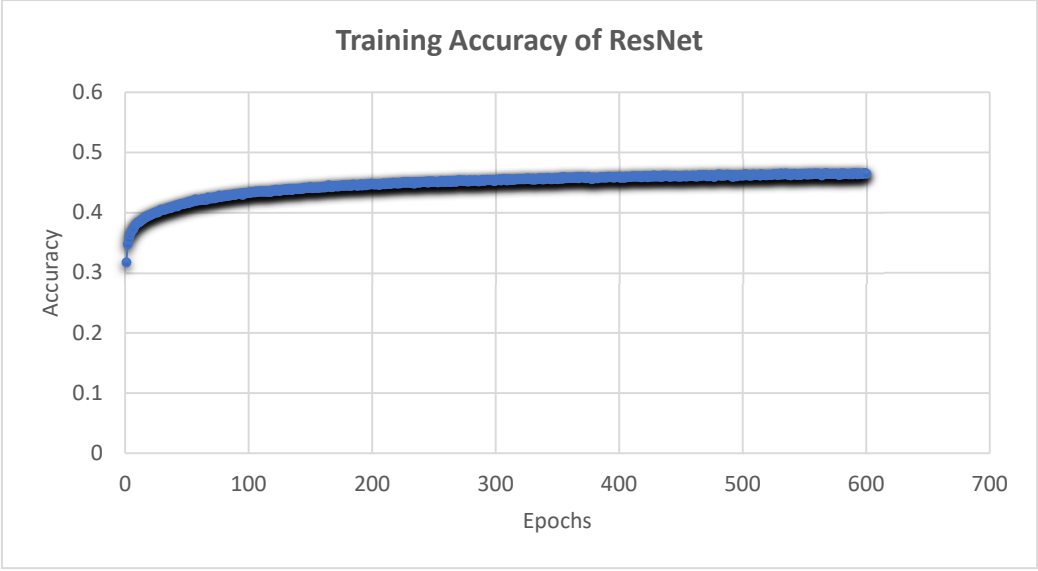


Figure 42. Training accuracy using ResNet

In the experiment ensuing, we still adopt the skeleton of Keras model that was used in split pattern prediction, but with some modifications to fit into the new scenario. The pooling layer is discarded in favor of architecture that only consists of repeated CONV layers. Convolution network has been used without pooling layers when input size is too small for further feature extraction. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers (Labs, 2017).

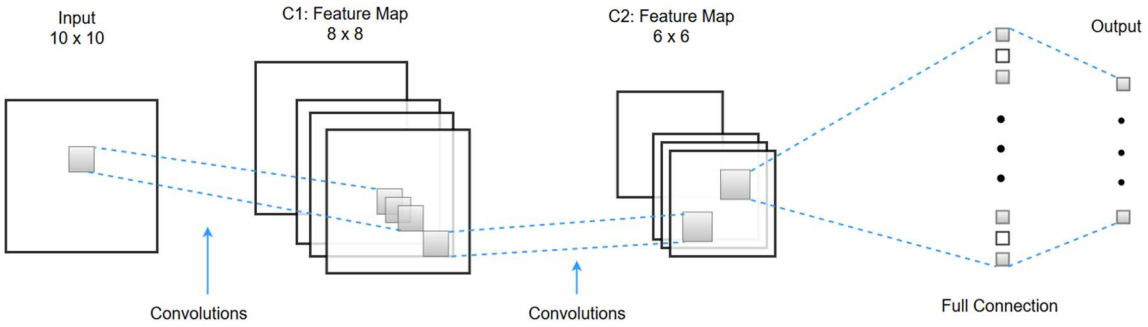


Figure 43. CNN structure without pooling layers

From the structure outlined above, we can see here is one big difference, i.e., no pooling is present in this model. The network includes only 2 convolution layers. For each convolution layer, we use 32 and 64 of 3 × 3 filters respectively to extract the feature map. The activation function is also ReLU across the board. And the border mode is set as "valid". It finally comes with the fully connected layers where the hidden units will be "flatten" and directed to the output of 35 labels.

In our experiment, for a typical sample video clip with 300 frames and each frame with size of 352 × 288 pixel, there will be 475200 of 8 × 8 CTUs generated by reference software HM and available to use, which will be further expanded/padded to 10 × 10. During the

process of building the model, such big data set will be split into 80/20 portion of training/test data set, and the test data set can be used to validate the training results. For output, it is relatively simpler than split pattern prediction since there are only 35 predefined possibilities, which are generated directly from reference software HM.

Training and Testing Results.

We conducted the experiments with several of the sample video clips. The accuracy can go as high as 92% in 800 iterations as shown in Figure 44. The experiment halted at 800 iterations because each iteration takes about 300 seconds to run. The highest accuracy achieved is bus video clip, and the lowest training accuracy is 66% for container clip. The possible reason may be container clip is less complicated or "diversified" such that the model is not fed with enough information within the same period of time. However, one thing to note is that the trend of the training accuracy curve is still upward, and the accuracy will get further improved give more time of training. Another thing that should be noted is that training time is quite long, averaging 300 seconds per iteration. This is because the pooling layers are not present in the model, and it increased the data amount and computational complexity, and we used CPU as the only computing method. Such issue may be addressed by introducing GPU as the main computing power and computation in a distributed manner should improve the time as suggested by others (Shaikh, 2017).

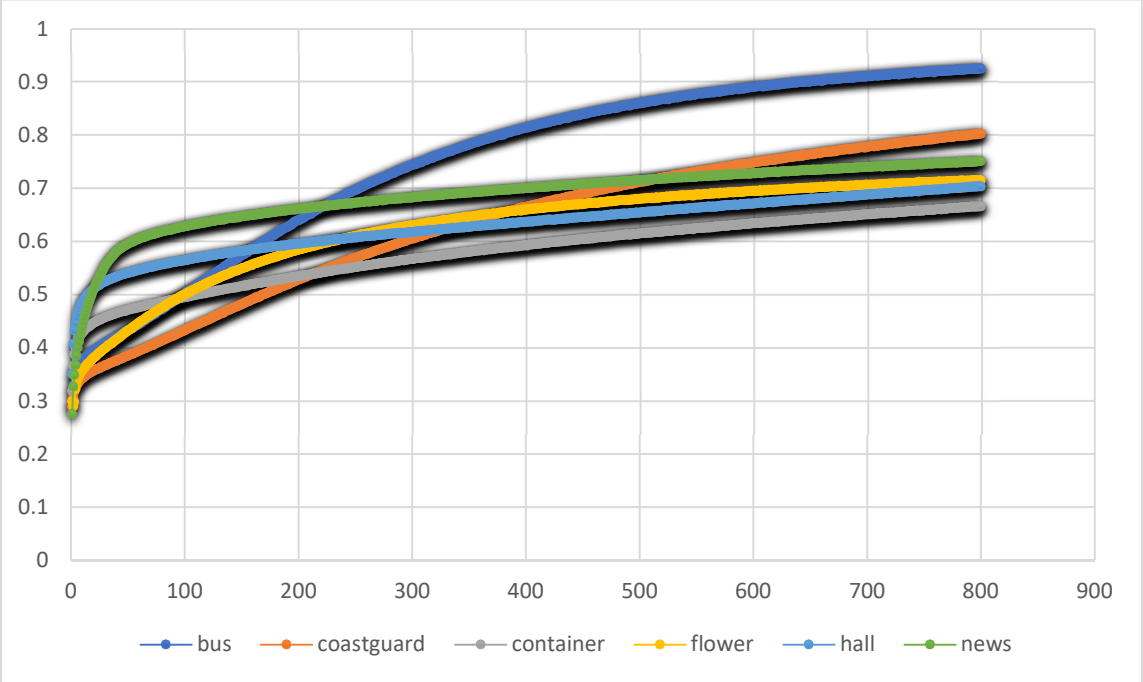


Figure 44. Training accuracy of intra-picture prediction for sample video clips

We then moved to the validate the testing accuracies with the trained model against the testing samples. The testing accuracies are also in align with the training accuracy. The highest testing accuracy is achieved for bus video clip, and the lowest is container. The testing accuracy is in align with training, such that it is expected to improve given more training time.

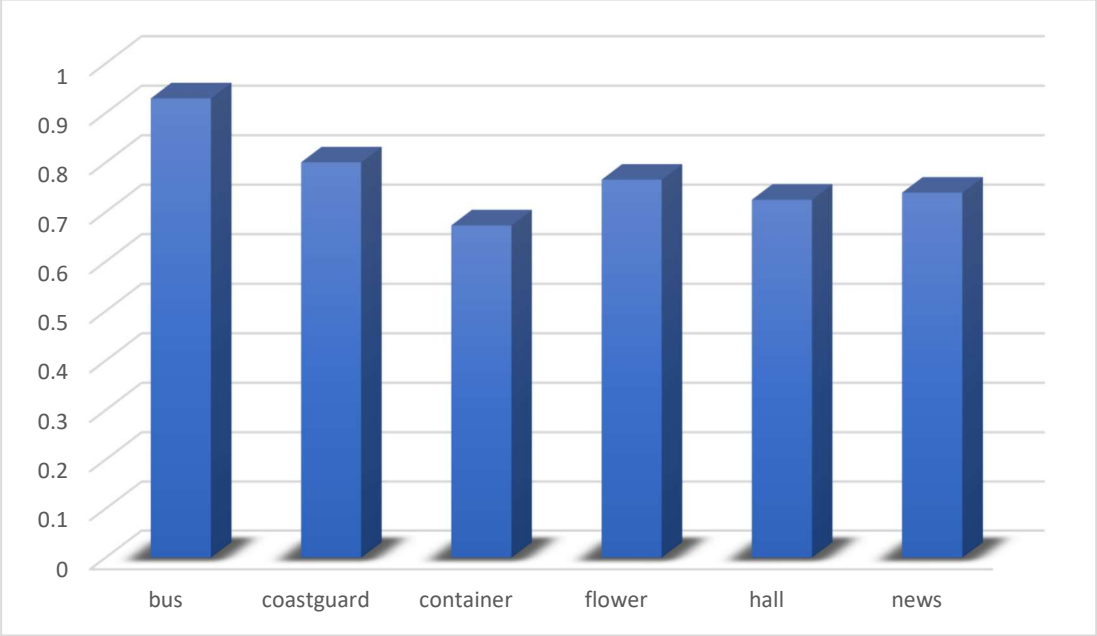


Figure 45. Testing accuracies for intra-picture prediction of sample videos

The speed of intra-picture prediction is also compared before and after introduction of CNN. Based on the results from reference software HM, the speed is 500 microseconds per CU block. The model can have average of 60 microseconds for intra-picture prediction of each block with 92% accuracy.

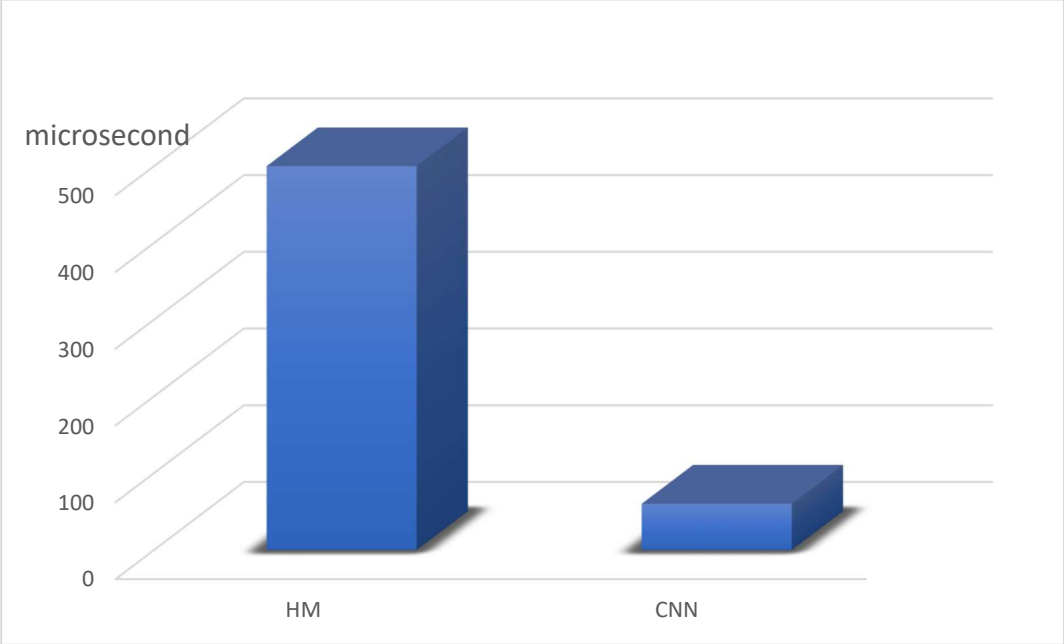


Figure 46. Speed of intra-picture prediction before and after CNN

It's improved 88% in terms of speed. It should be noted again that the accuracy can still be improved using more training iterations. In summary, it is very promising as well to apply machine learning techniques and integrate them into intra-picture prediction of H.265.

Chapter VI

Conclusion

In this paper, we introduced new video encoder/decoder standard H.265, also known as HEVC (i.e., High Efficiency Video Coding). Such new standard is designed to meet the increasing demand of storage and streaming of ultra-high definition videos. In its core, it is composed of 2 major components: CU split pattern prediction and intra-prediction. We then discussed the advantages of CNN (convolution neuron network). Since we are also dealing with large amount of video sample data, it's realistic to apply CNN techniques into this research.

We investigated the application of CNN in these two core elements: split pattern and intra-prediction, aiming to improve the prediction speed of these operations as they are exhaustive and time-consuming in reference software HM. Our results in CU split pattern prediction indicated that after fine tuning the model in Keras framework, most of the training accuracy of the model reached above 90% and even as high as 99.9%. The prediction speed also increased from 4000 microseconds to 400 microseconds, a significant 90% improvement. In intra-prediction, we removed the subsampling layers to better fit this specific scenario, and the training accuracy also reached above 90% and prediction speed increased from 500 microseconds to 60 microseconds. It is concluded that employment of CNN in HEVC is beneficial and promising.

There is another core component in HEVC, i.e., the inter-frame prediction. It is expected to apply CNN technique in this area in further work. Also, we can adopt some other deeper or more advanced neuron network such as ResNet that we have preliminarily experimented

in this paper. What should be noted is we may introduce GPU as the computing methods in a distributive manner to improve the lengthy training time during intra-picture prediction.

References

Lainema, J., Bossen, F., Han, W.-J., Min, J., & Ugur, K. (2012). Intra Coding of the HEVC Standard. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 1792-1801.

Abramowski, A. (2016). A survey over possible intra prediction optimizations in the H.265/HEVC encoder. *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments.* Wilga, Poland.

Alam, M. M., Nguyen, T. D., Hagan, M. T., & Chandler, D. M. (2015). A perceptual quantization strategy for HEVC based on a. *convolutional neural network trained on natural images*.

Alpaydin, E. (2014). Introduction. In E. Alpaydin, *Introduction to Machine Learning.* The MIT Press.

Bharamgouda, S. (2013, 08 05). *Video Compression (Temporal Redundancy) Simplified.* Retrieved from www.quora.com: https://simply5.quora.com/Video-Compression-Temporal-Redundancy-Simplified

Buhuma, N. (2015, January). *Deep Learning in a Nutshell – what it is, how it works, why care?* Retrieved from https://www.kdnuggets.com/2015/01/deep-learning-explanation-what-how-why.html

Chen, Z.-Y., Fang, J.-T., Liu, Y.-C., & Chang, P.-C. (2016). Machine Learning-based Fast Intra Coding Unit Depth Decision for High Efficiency Video Coding. *Journal of Information Science and Engineering*.

Dass, R., Sign, L., & Kaushik, S. (2012). Video Compression Technique. *International Journal of Scientific & Technology Research, 1*(10), 114-119.

Deshpande, A. (2016, July 20). *A Beginner's Guide To Understanding Convolutional Neural Networks.* Retrieved from https://github.com/adeshpande3: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Di, W. (2013, September 6). *Rectifier Nonlinearities.* Retrieved from Vanessa's Imiloa: https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities/

Duanmu, F., Ma, Z., & Wang, Y. (2015). 2015 IEEE International Conference on Image Processing (ICIP). *2015 IEEE International Conference on Image Processing (ICIP)*.

Encoding/Decoding Explanation of ITU-T H.264/MPEG-4 AVC. (2014, 3 28). Retrieved from CSDN.net: http://blog.csdn.net/chinadragon76/article/details/22408727

Fraenkel, A. S., & Klein, S. T. (1990). Bidirectional Huffman Coding. *The Computer Journal*, 296-307. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Huffman_coding

Fumo, D. (2017, June 15). *Types of Machine Learning Algorithms You Should Know.* Retrieved from Towards Data Science: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

Han, W.-J., & Lainema, J. (2014). Intra-Picture Prediction in HEVC. In V. Sze, M. Budagavi, & G. J. Sullivan, *HIgh Efficiency Video Coding (HEVC) Algorithms and Architectures* (p. 93). Springer.

He, K., Zhang, X., Ref, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Las Vegas, NV, USA : 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

High Efficiency Video Coding (HEVC). (2015). Retrieved from Fraunhofer HHI: Fraunhofer Heinrich-Hertz-Institut: https://hevc.hhi.fraunhofer.de/HM-doc/index.html

Juurlink, B., Alvarez-Mesa, M., Chi, C. C., Azevedo, A., Meenderinck, C., & Ramirez, A. (2012). Understanding the Application: An Overview of the H.264 Standard. In B. Juurlink, M. Alvarez-Mesa, C. C. Chi, A. Azevedo, C. Meenderinck, & A. Ramirez, *Scalable Parallel Programming Applied to H.264/AVC Decoding* (p. 5). Springer Science & Business Media.

Karpathy, A. (2017). *CS231n Convolutinal Neural Networks for Visual Recognition.* Retrieved from http://cs231n.github.io: http://cs231n.github.io/convolutional-networks/

Kim, I.-K., Min, J., Lee, T., Han, W.-J., & Park, J. (2012). Block Partitioning Structure in the HEVC Standard. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 1697-1706.

Kishor, N. (2017, May 23). *Understanding the differences between AI, machine learning, and deep learning*. Retrieved from http://houseofbots.com/news-detail/362-Understanding-the-differences-between-AI-machine-learning-and-deep-learning

Konstantinova, N. (2014, March 22). *Machine learning explained in simple words*. Retrieved from http://nkonst.com/machine-learning-explained-simple-words/

Kweon, M. (2016). *https://github.com/kkweon*. Retrieved from Github: https://github.com/kkweon/mnist-competition#resnet

Labs, M. (2017, March). *How these researchers tried something unconventional to come out with a smaller yet better Image Recognition.* Retrieved from Medium: https://medium.com/@matelabs_ai/how-these-researchers-tried-something-unconventional-to-came-out-with-a-smaller-yet-better-image-544327f30e72

Li, J., & Wang, P. (2017, 08 10). *Introduction to Next Generation of Video Compression Standard HEVC(H.265).* Retrieved from wenku.baidu.com: https://wenku.baidu.com/view/5462911f55270722192ef775.html

Li, Y., Liu, D., Li, H., Li, L., Wu, F., Zhang, H., & Yang, H. (2017). Convolutional Neural Network-Based Block Up-sampling for Intra Frame Coding. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 1-14.

Ling, N. (2012). High Efficiency Video Coding and its 3D extension: A research perspective. *IEEE Conference on Industrial Electronics and Applications*, (pp. 2150-2155). Singapore.

Liu, Z., Yu, X., Chen, S., & Wang, D. (2016). CNN oriented fast HEVC intra CU mode decision. *2016 IEEE International Symposium on Circuits and Systems*.

Liu, Z., Yu, X., Gao, Y., Chen, S., Ji, X., & Wang, D. (2016). CU Partition Mode Decision for HEVC Hardwired Intra Encoder Using Convolution Neural Network. *IEEE Transactions on Image Processing*.

Machine Learning in MATLAB. (2017). Retrieved from MathWorks: https://www.mathworks.com/help/stats/machine-learning-in-matlab.html?requestedDomain=true

Momcilovic, S., Roma, N., Sousa, L., & Milentijevic, I. (2015). Run-Time Machine Learning for HEVC/H.265 Fast Partitioning Decision. *2015 IEEE International Symposium on Multimedia*.

Moto. (2012, 10 28). *HEVC – What are CTU, CU, CTB, CB, PB, and TB?* Retrieved from CODE: Sequoia: https://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/

Muqeet, A. (2017, April). *Handwritten Digit Recognition using Keras*. Retrieved from https://www.kaggle.com/abdulmuqeet/handwritten-digit-recognition-using-keras/versions

Narang, N. (2013, 10 01). *What is the difference between HEVC (H.265) and H.264 (MPEG-4 AVC).* Retrieved from M&E Industry Trends, Technology and Research: http://www.mediaentertainmentinfo.com/2013/10/4-concept-series-what-is-the-difference-between-hevc-h-265-and-h-264-mpeg-4-avc.html/

Nielsen, M. (2015). Using neural nets to recognize handwritten digits. In M. Nielsen, *Neural Networks and Deep Learning.* Determination Press. Retrieved from http://neuralnetworksanddeeplearning.com: http://neuralnetworksanddeeplearning.com/chap1.html

Patel, D., Lad, T., & Shah, D. (2015). Review on Intra-prediction in High Efficiency Video Coding (HEVC) Standard. *International Journal of Computer Applications*, 27-30.

Persistently, S. (2017, 07 21). *H.265/HEVC —— 帧内预测*. Retrieved from www.jianshu.com: https://www.jianshu.com/p/d19d7eb3844a

Pourazad, M. T., Doutre, C., Azimi, M., & Nasiopoulos, P. (2012, 07). HEVC: the New Gold Standard for Video Compression. *IEEE Consumer Electronic Magazine*, pp. 36-46.

Puch, A. (2017, 08 23). *HEVC (H.265) vs. AVC (H.264) - What's the Difference?* Retrieved from Boxcast: https://www.boxcast.com/blog/hevc-h.265-vs.-h.264-avc-whats-the-difference

Rao, S. K., Thakur, N., & Adavi, S. K. (2016). *HEVC Intra Prediction.*

Ravindra, S. (2017, August). *How Convolutional Neural Networks Accomplish Image Recognition?* Retrieved from https://www.kdnuggets.com/: https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html

Ray, S. (2017, September 9). *Essentials of Machine Learning Algorithms (with Python and R Codes).* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

Reuze, K., Philippe, P., Deforges, O., & Hamidouche, W. (2016). Intra prediction modes signalling in HEVC. *2016 Picture Coding Symposium (PCS)*.

Richardson, I. E. (2003). Video Coding Concepts. In I. E. Richardson, *H.264 and MPEG-4 Video Compression* (p. 27). Wiley.

Richardson, I. E. (2008). *H.264 Advanced Video Compression Standard.* Wiley.

Rodrigues, A. (2016, 06 09). *H.264 vs H.265 - A technical comparison. When will H.265 dominate the market?* Retrieved from Medium: https://medium.com/advanced-computer-vision/h-264-vs-h-265-a-technical-comparison-when-will-h-265-dominate-the-market-26659303171a

Schwarz, H., Schierl, T., & Marpe, D. (2014). Block Structures and Parallelism. In V. Sze, M. Budagavi, & G. J. Sullivan, *High Efficiency Video Coding (HEVC) Algorithms and Architectures* (p. 58). Springer.

Shaikh, F. (2017, May 18). *Why are GPUs necessary for training Deep Learning models?* Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/

Sharma, S. (2017, September 6). *Activation Functions: Neural Networks.* Retrieved from Towards Data Science: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Smola, A., & Vishwanathan, S. (2008). Introduction. In A. Smola, & S. Vishwanathan, *Introduction to Machine Learning.* Cambridge University Press.

Song, R., Liu, D., Li, H., & Wu, F. (2017). *Neural Network-Based Arithmetic Coding of Intra Prediction Modes in HEVC.* Retrieved from Cornell University Library: https://arxiv.org/pdf/1709.05737.pdf

Sullivan, G. J., Ohm, J.-R., Han, W.-J., & Wiegand, T. (2012). Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology, 22*(12), 1649-1668.

Surmenok, P. (2017, 11 12). *Estimating an Optimal Learning Rate For a Deep Neural Network.* Retrieved from Towards Data Science: https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

Veličković, P. (2017, 03 20). *Deep learning for complete beginners: convolutional neural networks with keras.* Retrieved from Data Science Course London: https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

Video. (2017). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Video

When to Choose Deep Learning. (n.d.).

Yang, J.-F., Wang, H.-M., & Tseng, C.-H. (2006, 08). High Quality and Fast H.264/AVC Video Encoder：Enhanced Intra 4x4 Mode Decision for H.264/AVC coders. *IEEE Transaction on Circuit and Systems for Video Technology, 2*(1), 1.

Zaccone, G. (2017). CNN architecture. In G. Zaccone, *Deep Learning with TensorFlow.*

Zhang, M., Zhai, X., & Liu, Z. (2017). Fast and adaptive mode decision and CU partition early termination algorithm for intra-prediction in HEVC. *EURASIP Journal on Image and Video Processing*, 1-11.

Zhao, Y., Wang, H., & Yuan, B. (2000, April). Advances in Fractal Image Coding. *Acta Electronica Sinica, 28*(4), 1-7.