Multimedia Communications and Signal Processing

Deep Learning for Block Partitioning in HEVC

Report on Research Internship

Andreas Spruck

October 2016

Supervisor: Andreas Heindel, Sanjukta Ghosh

Contents

List of Abbreviations

HEVC	High Efficiency Video Coding
SVM	Support Vector Machine
RDO	Rate Distortion Optimization
CU	Coding Unit
QP	Quantization parameter
ReLu	Rectified linear unit

Chapter 1

Introduction

As the imaging and displaying technology gets better and better, the demand for efficient video coding techniques increases more and more, in order to be able to store the video data on a small data-space or transmit it over a channel with limited bandwidth. For this task the new H.265/HEVC standard is very popular, as it is capable of coding video data very efficiently. But with increasing resolution of the coded video also the coding time and the requirements regarding the hardware increase rapidly. So the aim of this work will be to investigate if machine learning can be employed to speed up the coding of video sequences and how accurate the model works.

A quiet promising approach therefore are neuronal networks. They become increasingly popular among researchers in the machine learning community, as there is no need to feed features into the network but only some raw data, based on which the network calculates an result by itself.

Chapter 2

Problem Statement

The problem that inspired this work is the high time and power consumption of the Rate Distortion Optimization which is an important part of the HEVC standard. This will be further explained in Section 3.1.1. The idea of this internship was to train a neural network in a manner, that it makes the decision whether a block shall be split or not based on certain input data delivered by the coder. All this is done without calculating the costs of the possible decisions. By this much power and time can be saved, as it is not necessary to calculate the costs for each block arrangement.

Chapter 3

Background

3.1 HEVC

The H.265/HEVC standard is the newest video coding standard, it was designed to match the growing demand of coding high resolution video sequences more efficiently. This task has recently gathered more attention, as services like internet video streaming portals become more popular among the consumers who desire a as high as possible video resolution. At the latest with the upcoming of new 4K TV-sets it became clear that it wouldn't be possible to realize streaming via internet with the old H.264 standard. Also in the new DVB-T2 television broadcast system the HEVC standard will be implied.

One of the many reasons why it is possible to code videos so efficiently with HEVC is that the size of the Coding Units (CU), which can be compared to makroblocks in H.264, may be freely chosen to be 64x64, 32x32, 16x16 or 8x8 pixels large. Due to this variety of CU-sizes there are many possible options of splitting a frame into CUs thinkable. Figure 3.1 shall convey an impression of how many different split schemas are possible for a single frame, by showing the variety of bolcksizes in relation to a whole video frame. This effect increases with growing resolution

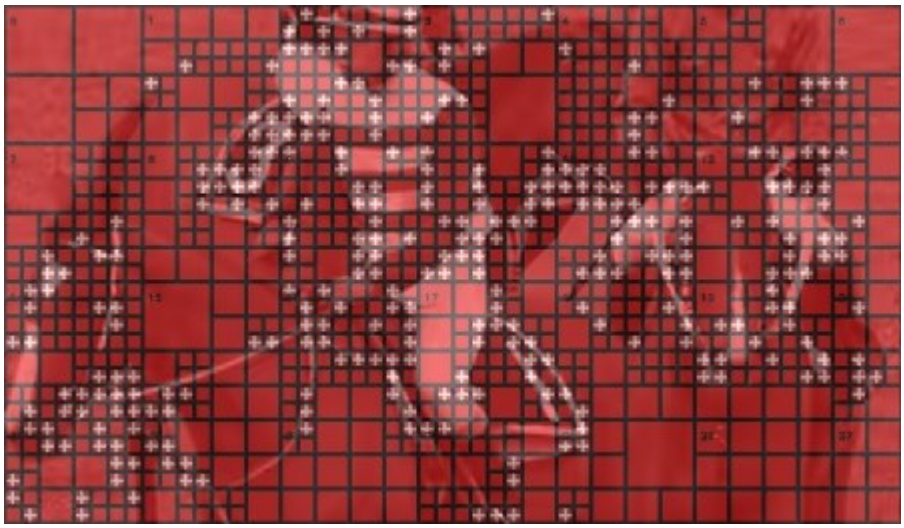

Figure 3.1: Example for CU partition of an intra coded frame. [SSW+14]

3.1.1 Rate Distortion Optimization in HEVC

Within the HEVC standard the Rate Distortion Optimization, short RDO, is used to obtain the in the rate-distortion sense optimal block partitioning schema for the current frame, meaning the distortion D shall be minimized under the constraint of a certain rate R_c.

$$\min D \text{ subject to } R < R_c \tag{3.1}$$

The optimization problem in equation (3.1) can be solved using the Lagrangian optimization where a distortion term is weighted against a rate term [SW98].

$$\min J \text{ where } J = D + \lambda R \tag{3.2}$$

Each solution of equation (3.2) for a given Lagrange multiplier λ is an optimal solution to equation (3.1) for a certain value of R_c.

Even though the result which is obtained by this method is optimal in the rate distortion sense, the computational complexity increases rapidly with increasing resolution of the videos. As the number of pixels quadruples with doubled resolution and every

possible combination has to be tested in order to find the optimum. Therefore in the following it will be tried to replace the RDO during the block partitioning by a neural network even though the result will then be no longer be optimal. But we can not get completely rid of the RDO, as it is still used on every block we obtained with the network to obtain the optimal prediction mode and partitioning schema for the prediction and transform units. But this is still an advantage compared to the previous method, as there equation (3.2) would have to be evaluated for every possible block partitioning schema and for each of these as above mentioned for the prediction and transform unit partitioning as well as for the prediction mode. So by introducing the network the number of possible combinations decreases significantly.

3.2 Deep Learning

As professor Yang explained in his talk [Yan16], deep learning is a branch of machine learning. Machine learning is a branch of signal processing where the computer is not explicitly told what to do but rather given the ability to learn it by itself from examples. This is in principal the same way we humans learn. The aim of neural networks is to imitate the function of the human brain, even only on a way lower level. In light of increasing computational power deep neural networks became popular again.

There is no unique definition when a neural network is called a deep neural network, but at least two layers are needed to call it deep, as it is depicted in Figure 3.2.

We will use a feedforward network here, which means that there is no feedback within the network. Also we will use a supervised learning approach here, which means, that we have the decision from the RDO available for the training data. By this we can develop a cost function that will be minimized by the network.

In order to avoid overfitting the network will be tested on independent test data and the accuracy will be calculated during this. Overfitting leads to a low accuracy during the test phase, as the model is only suited for the training data. Meaning the network is too specialized on the training data and is not able to recognize a pattern within the

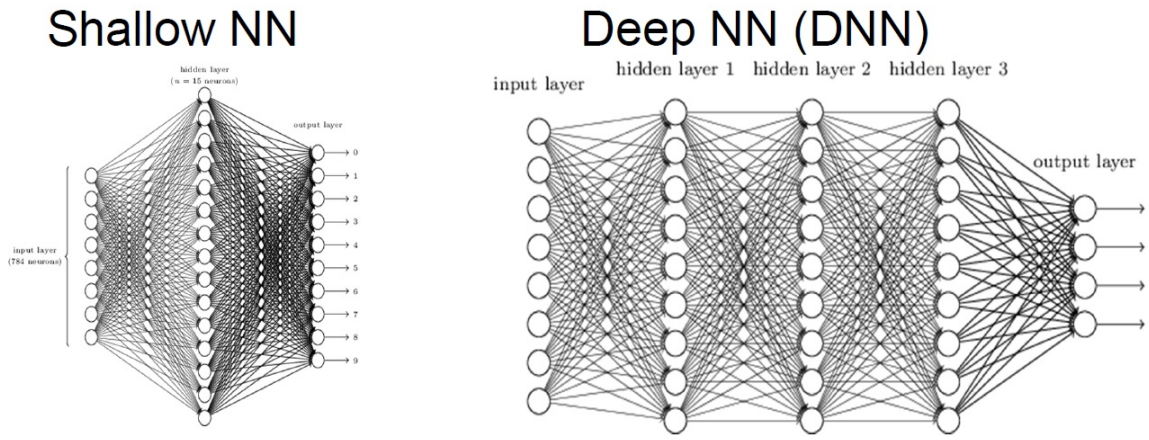

Figure 3.2: Comparison between a shallow and a deep neural network [Yan16].

training data. This leads to a poor performance during the test phase, as the network is not able to generalize.

Chapter 4

Computational Framework

Caffe is a framework for deep learning networks developed by the Berkeley Vision and Learning Center [JSD+14]. Caffe is a very fast implementation. It is also possible to switch between CPU and GPU mode by simply setting a flag. Another reason for choosing Caffe as framework is that it has Matlab, Python und commandline interfaces and is written in C++. Moreover it is built in a modular manner, which might ease the process of integrating it into the encoder during later projects and keeps it well arranged and easy to understand during the process of building the model.

4.1 Layers

Caffe encloses predefined layers which can be used to build a network as it is desired by the user. The most important ones, which were also used during this work shall be further explained in the following sections.

4.1.1 Data Layer

The data layer is a layer, which enables reading input data from disk. The supported file formats are HDF5 and LevelDB. With these Layers it is possible to read test and training data in order to train the model and test is after that. Therefore the data

has to be saved either in the HDF5 or LevelDB format, these are both scientific file formats which support large tables and allow by this huge amounts of data, which will be explained in more detail in Section 5.1.

4.1.2 Inner Product Layer

The inner product (IP) layer is also called the fully connected layer as for example seen in Figure 3.2. It takes the input data as simple vector and produces a output vector with width one [JSD+14]. The IP-Layers used here have each 1000 neurons.

4.1.3 ReLu

The abbrevation ReLu stands for rectified linear unit. The rectifier is a an activation function which is defined as

$$f(x) = max(0, x).$$

Where x is the input of the ReLu. This is also called the ramp function. This layer introduces a non-linearity into the generated network [DSH13].

4.1.4 Accuracy Layer

The accuracy layer gives the output of the network compared to the desired target outcome [JSD+14]. By this the performance of the network can easily evaluated during the test phase, and by this one can directly see whether the desired target accuracy has been reached or not and how much effort has to be further invested into fine-tuning the model.

4.1.5 Softmax with Loss

The softmax function is also known as the normalized exponential function, this function gives a smoothed version of the max function, that's why it's called softmax [Bis06].

Chapter 5

Model Training

5.1 Data Setup

The data used for the generation of the model is based on the features used in [ZKW+15] for training their support vector machine. In more detail this means that the data used are:

- label
- skip flag
- coded blog flag
- rate distortion cost
- distortion
- bits
- motion vector
- average neighboring rate distortion cost
- average neighboring depth

- QP

These features are given in a txt-flie and have to be converted into a HDF5 file fist in order to use it with Caffe. This is done with a Matlab script which is appended too. This skript mainly uses the store2hdf5 function which comes along with Caffe. But in order to process the data stored in the txt-file it has to be preprocessed such that every feature has an own column. This has been done using Excel here and afterwards saving it as CSV formatted file in order to make it available for Matlab. By doing so the file and with that the data can be read automatically into a cell array using textscan. One of the main issues was to find the matching dimensions required for the HDF5 file. The file consists of two data-sets, label and data, those can be read and processed separately within the model. The data-set label contains the outcome of the RDO, and data contains the other nine features shown before, which are used to feed the network. The problem with the dimensions are that Caffe expects four dimensional input data which is quite unintuitive, another problem is, that the size of the dimension has to be arranged in a different manner in Matlab than it is described in Caffe. The dataset label needs the dimensions of $(n \times 1 \times 1 \times 1)$ and data of $(n \times 1 \times 1 \times 9)$, where n is the number of samples.

5.2 Network

As mentioned before the most common layers needed for building a neural network come with the program. Building a model can be done by defining a sequence of layers within a protobuf-file. The version used here can be further investigated on the appended disk. A sketch of the network described in the protobuf-file can be seen in Figure 5.1. This network is an example of supervised learning, as we use the outcome of the decision to enhance the learning process of the network, which is represented by the label-blob in Figure 5.1.

We use a rather simple network here in order to keep the time needed for training and later running the model low. Furthermore this is the first try of using a network in such

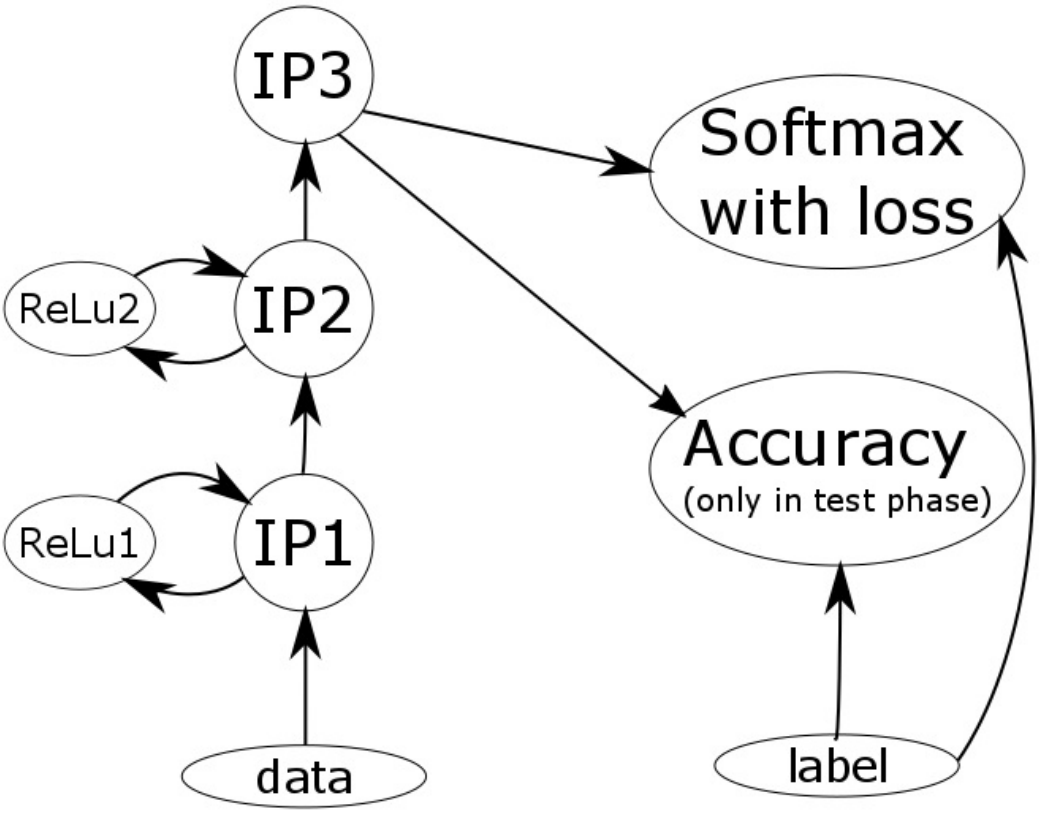

Figure 5.1: Graphical representation of the build model

a case and because of this it should be investigated how such a simple model performs in this kind of application.

That this is just a first attempt to get a guess of the possible performance is also the reason why the data layer takes HDF5-files as input. This requires that the output of the coder needs to be saved in order to feed it later into the network, for a final implementation it might be thinkable that the coder output is fed directly into the network without intermediate storing on the hard drive, meaning the network becomes a directly implemented part of the coder.

Chapter 6

Results

Due to some problems while implementing the network and especially formatting the data, the network was only tested for one class C sequence, the Basketball Drill with a resolution of 832x480 and 50fps for depth 0. The training data for this sequence consists of three other class C sequences also for depth 0. These sequences are:

- BQMall with a resolution of 832x480 and 60 fps
- PartyScene with a resolution of 832x480 and 50 fps
- RaceHorses with a resolution of 832x480 and 30 fps

For the generation of the model 450 000 training iterations were used. With the system described in this report it was possible to achiev a accuracy of 66.25%. The hyperparameters used here, which lead to this result are:

- starting learning rate: 0.01
- learning rate decrement-factor: 0.1
- weight decay: 0.0005
- batch size: 32
- total number of training examples: 53508

- total number of test data: 17836

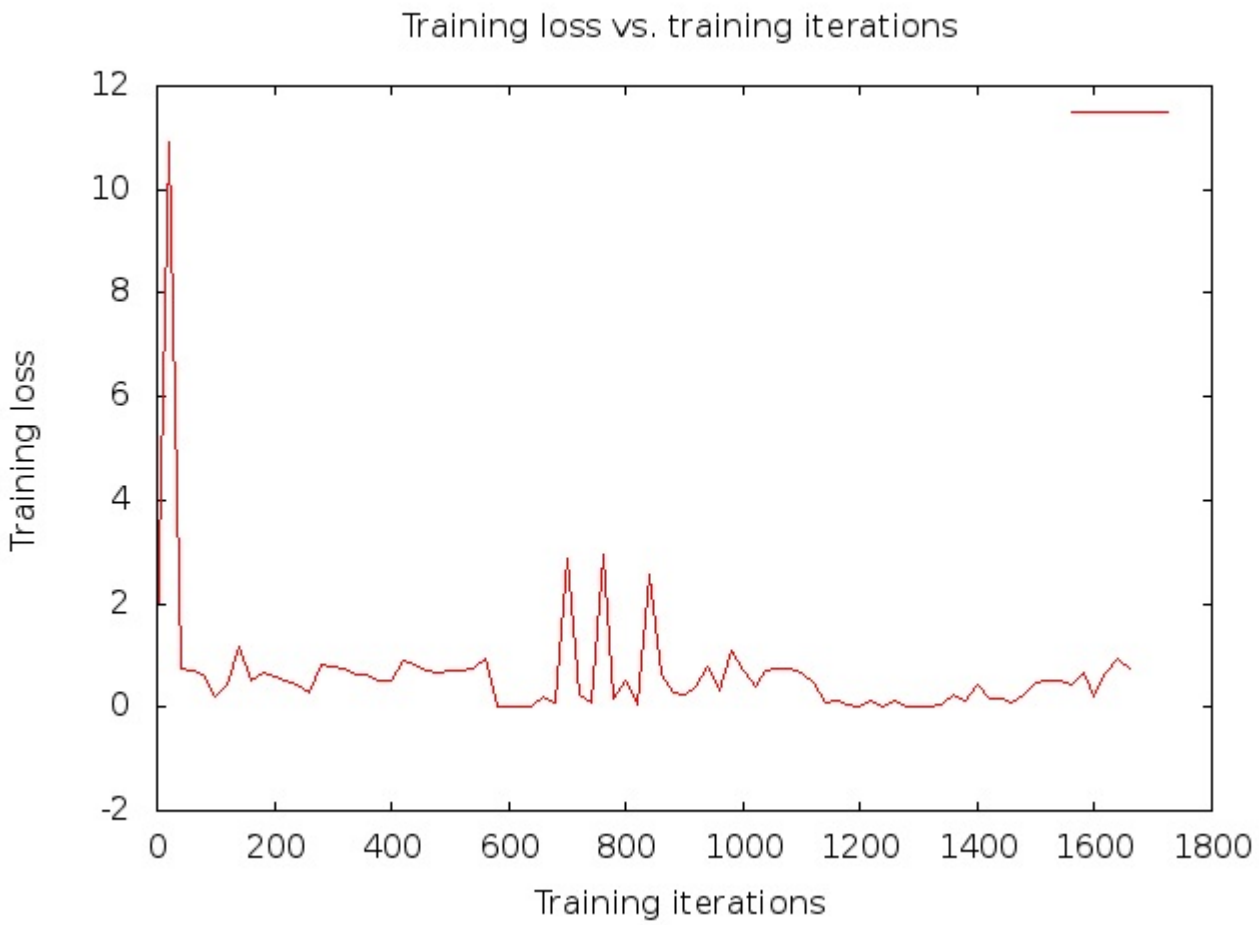

Figure 6.1: Training loss vs. training iterations

Chapter 7

Outlook

The result of 66.25% is not as good as expected as it is only slightly better than a random guess which would have an accuracy of 50%, so further work should focus on increasing the accuracy. Therefore various methods might be possible. In the first step the hyperparameters, which are used for generating the model should be varied in order to enhance the results. The following step would be to train and test the model on other sequences in order to see if the results obtained here are reproducible. If the results stay on such a low level the values set in the solver file should be varied, such as the number of training iterations or learning rate. If these methods lack of success too another possibility would be to modify the architecture of the whole network.
Even if this looks like much effort that must be invested once again it might be worth the work as the split decision is much faster using neural networks once the model was generated than using the RDO for this process.

Bibliography

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[DSH13] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613. IEEE, 2013.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[SSW+14] D. Springer, W. Schnurrer, A. Weinlich, A. Heindel, J. Seiler, and A. Kaup. Open Source HEVC Analyzer for Rapid Prototyping (HARP). In *IEEE Int. Conf. on Image Processing (ICIP)*, Paris, France, October 2014.

[SW98] Gary J Sullivan and Thomas Wiegand. Rate-distortion optimization for video compression. *Signal Processing Magazine, IEEE*, 15(6):74–90, 1998.

[Yan16] Bin Yang. Deep learning. Guest Lecture Ferienakademie 2016, Course 7: Virtual and Augmented Reality, 2016.

[ZKW+15] Yun Zhang, Sam Kwong, Xu Wang, Hui Yuan, Zhaoqing Pan, and Long Xu. Machine learning-based coding unit depth decisions for flexible complexity

allocation in high efficiency video coding. *IEEE Transactions on Image Processing*, 24(7):2225–2238, 2015.