Wrocław University of Science and Technology

Faculty of Information and Communication Technology

Field of study: **INA**

Speciality: **-**

BACHELOR THESIS

The algorithms for video streaming quality improvement based on neural networks

Jan Pawłowski

Supervisor
Ph.D. Maciej Zięba

hevc, neural networks, artifact removal, intra frames

Wrocław 2022

Abstract

Major part of internet traffic consists of streamed videos, which are compressed lossily with standards such as HEVC. Lossy compression produces characteristic artifacts, which in HEVC are reduced by Deblocking Filter and Sample Adaptive Offset. These filters can be substituted with neural networks such as VRCNN, which produces good visual results, but is too slow to perform inference in real-time on HD input.

This work proposes a neural network architecture for artifact removal in HEVC intra coding on domain-specific videos, that is inspired by VRCNN. Proposed architecture is able to perform inference more than 30 times per second on HD input on mainstream hardware. It yields better results compared to Deblocking Filter and Sample Adaptive Offset in HEVC. Tests show that the amount of improvement is heavily dependent on the nature of domain.

Streszczenie

Dużą część ruchu internetowego stanowią strumieniowane filmy, kompresowane stratnie za pomocą standardów takich jak HEVC. Kompresja stratna powoduje powstanie charakterystycznych artefaktów, które w HEVC są redukowane za pomocą filtra deblokującego i adaptacyjnego przesunięcia próbek. Filtry te mogą zostać zastąpione sieciami neuronowymi, takimi jak do tego przeznaczony VRCNN, który zwraca dobre wizualne rezultaty, jednak nie jest dostatecznie wydajny, by działać w czasie rzeczywistym na rozdzielczości HD na powszechnie posiadanych kartach graficznych.

W pracy zaproponowano architekturę sieci neuronowej zainspirowaną przez VRCNN, służącą do usuwania artefaktów w standardzie HEVC w filmach z obrębu jednej dziedziny. Proponowana architektura jest w stanie przeprowadzać inferencję na danych o rozdzielczości HD ponad 30 razy na sekundę na powszechnie posiadanych kartach graficznych. Zaproponowana architektura zwraca lepsze wyniki, niż filtr deblokujący i adaptacyjne przesunięcie próbek w HEVC. Testy pokazują, że różnica jest w dużym stopniu uzależniona od charakteru dziedziny.

Contents

Introduction

Studies indicate the 60% of all downstream internet traffic consists of video[1]. Because of this it is important to achieve as good stream quality as possible for a given bitrate. Commonly used standards such as H.264/AVC or H.265/HEVC have proven to perform very well and deliver good image quality, while immensely reducing used bandwidth. Additionally AVC and HEVC can be decoded on most modern devices such as computers, smart TVs and smartphones. Because of this they gained a wide popularity. This work is focused mostly on HEVC as it is newer and produces better visual results than AVC.

With the recent rise of popularity of Neural Networks, attempts have been made to use them for video compression. Because conventional coding standards such as AVC or HEVC perform very well in general, it is often considered to replace a specific stage in the HEVC pipeline, instead of replacing these coding standards entirely. One example of such replacement is VRCNN[2] or RHCNN[18] which are replacements for DBF and SAO filters of HEVC. Unfortunately these solutions are often unable to perform inference in real-time.

The goal of this work is to propose a Neural Network architecture that replaces DBF and SAO filters of HEVC, which produces perceptually better results and is able to perform inference on 1280x720 frames at least 30 times per second on mainstream hardware. Inference speed of neural networks is heavily dependent on i.e. the amount of FLOPS (Floating Point Operations Per Second) that a GPU can perform. Nvidia GTX 1060 will be the target hardware as it is the most commonly used GPU according to the Steam Hardware Survey of October 2021[2] This work will focus on creating models specialized in one domain of data, rather than creating one model suitable for any kind of video. This helps models to produce good visual results while being relatively not lightweight. While there is no strict definition of a ”data domain” in this context, one can assume that multiple frames captured from one game share a lot when it comes to visual features and thus can be considered to be withing the same domain. This approach could be applicable in real life scenarios, given the popularity of game streaming services like Microsoft xCloud, Nvidia GeForce Now and Amazon Luna or live streaming services like Twitch, where it is well known what game is being streamed. According to https://twitchtracker.com/games/32982 there have been 129 million hours of Grand Theft Auto V streamed on Twitch in November of 2021.

[1] Source: https://www.sandvine.com/hubfs/Sandvine˙Redesign˙2019/Downloads/2020/Phenomena/Mobile%20Phenomena%20Report%201H%202020%2020200219.pdf

[2] Source: https://store.steampowered.com/hwsurvey/

H.265/HEVC

This chapter contains a brief introduction to the relevant parts of HEVC coding standard.

Video coding standard H.265/HEVC (High Efficiency Video Coding) is a successor to a widely popular standard H.264/AVC [15] (Advanced Video Coding). It is computationally more expensive, but achieves significantly better quality at the same bitrate, example of which is shown in figure 1.1. HEVC performs lossy compression and the quality and weight of compressed video is dependent on the value of *QP* (Quantization Parameter). Higher QP values correspond to stronger compression and worse image quality.

In order to achieve better compression, HEVC exploits spatial and temporal redundancy in videos.

Figure 1.1: Frame compressed by AVC (left image) and HEVC (right image) at identical bitrates. Note more details on HEVC.

1.1 Exploiting spatial redundancy

Within a single frame there are often a lot of similar patterns. In order to exploit this, a frame is divided into a quad tree consisting of Coding Units (CUs) that can can have a size of 8×8, 16×16, 32×32 or 64×64, dependent on what the encoder deems most optimal. Generally areas of images with less information are assigned larger chunks, while areas with more details are divided into smaller chunks. This can be seen in figure 1.2. This mechanism is different compared to AVC, which uses fixed-size 16×16 macroblocks.

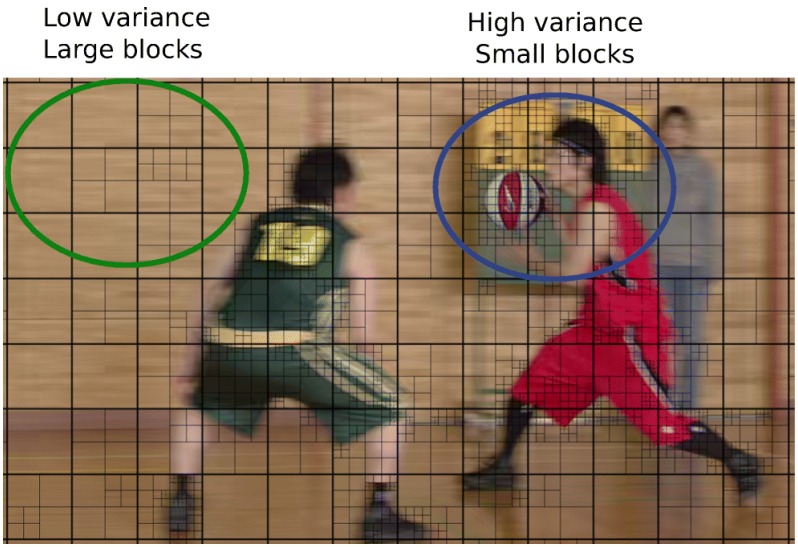


Figure 1.2: Frame split into Coding Units. Source: [14].

Then each block can be compressed with the help of *intra-frame prediction*, which means that the contents of current block are predicted, based on reference pixels of nearby blocks. Reference pixels locations are shown in figure 1.3. Note that not all reference pixels are always available at the time of making intra prediction. In that case their value is estimated based on nearest available reference pixels.

Only the difference (residue) between the actual content of a block and the intra prediction needs to be encoded, which contains significantly less information and thus is more compressible.

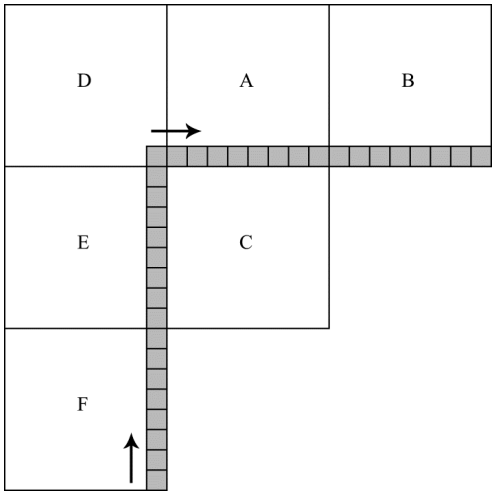


Figure 1.3: Reference pixels locations relative to the block being coded (denoted C). Arrows indicate the order in which reference pixels are accessed. Source: https://www.elecard.com/page/spatial˙intra˙prediction˙in˙hevc.

How the intra-frame prediction is calculated is based on a *mode* selected by the encoder. There are 35 various prediction modes available in HEVC, which are described in detail in [5].

1.1.1 Mode 0

Also called Planar mode. A bilinear interpolation using reference pixels is used to calculate value of each pixel in the block. Since reference pixels are not available to the right and below current block, reference pixels to the right are assumed to have a value of the leftmost reference pixel in block B on figure 1.3 and reference pixels below current block are assumed to have a value of the topmost reference pixel in block F on figure 1.3.

1.1.2 Mode 1

Also called DC mode. In this mode values are predicted by calculating mean average of reference pixels. Detailed description can be found in [5].

1.1.3 Modes 2-34

Modes 2-34 are angular modes. In these modes each pixel in the block is assigned a value of a reference pixel, determined by the direction associated with selected angular mode. Directions associated with each mode are shown in figure 1.4.

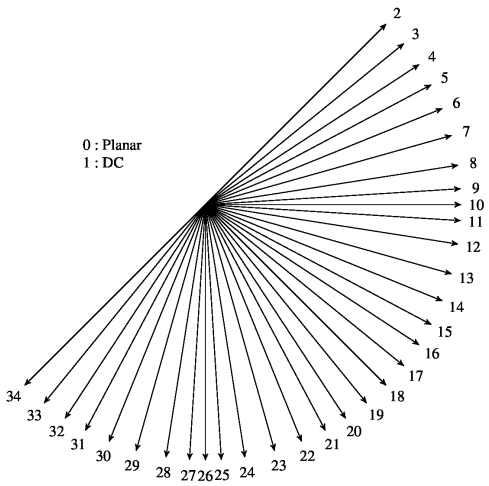


Figure 1.4: Directions corresponding to each angular mode. Source: http://www.hevcbook.de/intra-prediction/.

1.2 Exploiting temporal redundancy

In videos it is common to see frames that are close to each other in time, being very similar or even identical to each other in many parts of the image. This temporal redundancy can be exploited and information from other frames can be reused in order minimize the amount of information that needs to be encoded. To achieve this three types of frames are used:

- I-frames (Intra-coded frames): include complete frame, with no reference to other frames,
- P-frames (Predictive frames): store information about changes relative to prior I-frames or P-frames,
- B-frames (Bi-predictive frames): can reference both prior and subsequent frames.

Prediction made with reference to other frames, rather than other blocks in current frame is called *inter prediction*. Important detail related to inter prediction is that any frame that is being referenced is already improved by in-loop filters such as DFB and SAO (described in section 1.3) and thus the improvements

carry to inter-predicted frames. This is obviously not the case for any filter that is applied out-of-loop (just before the frame is displayed). Later parts of this work focus on I-frames.[1].

1.3 Deblocking Filter in HEVC

HEVC is block based and hence it produces blocking artifacts. These artifacts are reduced by an in-loop *Deblocking Filter*[8] (DBF). DFB algorithm is complex and has multiple branchings. The algorithm considers blocks' boundaries in 4-pixel long sections and uses blocks of 4×4 pixels, adjacent to the boundary line on each side of it. These adjacent blocks are referred to as P and Q and their pixels are denoted as shown in figure 1.5

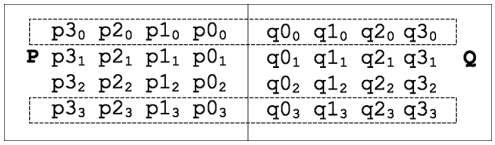


Figure 1.5: P and Q blocks of size 4×4 on each side of boundary. Only rows marked with dashed lines are used for selecting deblocking type. Source: [8].

General idea is that low spatial activity on both sides of the boundary combined with a discontinuity at the boundary itself indicates a blocking artifact as shown in figure 1.6.

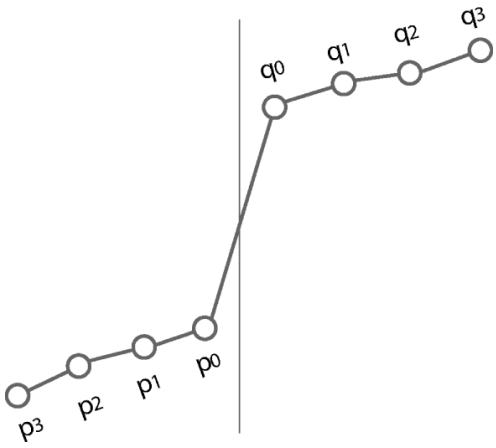


Figure 1.6: Example of a blocking artifact on block boundary. Source: [8].

Given that pixels are denoted as in figure 1.5 then spatial activity of $i-th$ row is defined as:

$$|p_{2,i} - 2p_{1,i} + p_{0,i}| + |q_{0,i} - 2q_{1,i} + q_{2,i}|, \tag{1.1}$$

which is an estimation of how linear the relation of pixels on each side of the boundary is. Based on spatial activity of first and third a deblocking decision is made. A filtering is then applied in order to smooth out the discontinuity. Details of this operation are described in [8].

[1] An explanation of inter-prediction works can be found in an online Elecard Video Compression Book at https://www.elecard.com/page/inter˙frame˙prediction˙inter˙in˙hevc and https://www.elecard.com/page/motion˙compensation˙in˙hevc

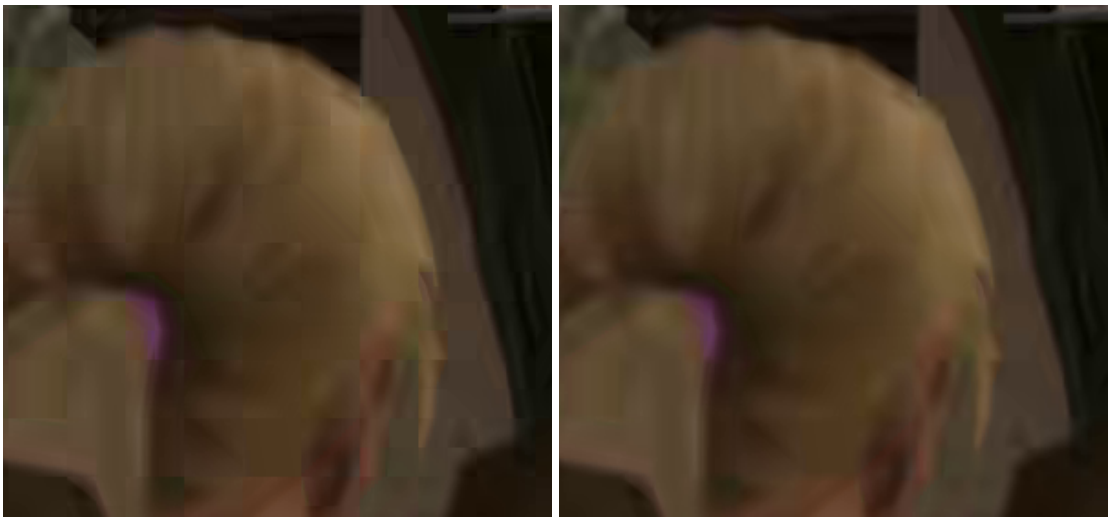


Figure 1.7: Example image without deblocking (left) and with deblocking (right).

1.4 Sample Adaptive Offset in HEVC

Sample Adaptive Offset [7] (SAO) is an in-loop filter that is applied after DBF. It aims to further reduce artifacts such as ringing artifacts. Example ringing artifact is shown in figure 1.8.

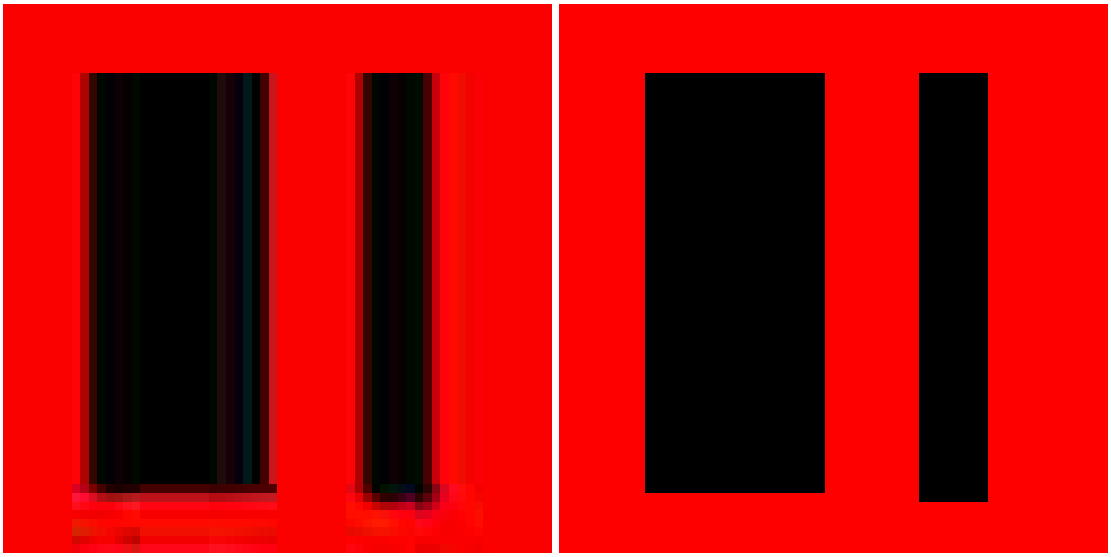


Figure 1.8: Image exhibiting ringing artifacts (left) and image without ringing artifacts (right).

Unlike DBF which does not require any additional information to be transferred, SAO does require sending additional data. There are two types of SAO: Band Offset (BO) and Edge Offset (EO). It is decided per block, which one is used.

1.4.1 Band Offset (BO)

Pixels are divided into 32 evenly spaced bins (bands), based on their intensity. Average offset between reference image and decoded pixels is calculated for each band in current block. Encoder selects four consecutive bands and transfers offsets corresponding to them, alongside with number of the first band. On decoder side, the offsets are added to pixels assigned to corresponding bands.

1.4.2 Edge Offset (EO)

There are four EO categories, each is demonstrated on figure 1.9

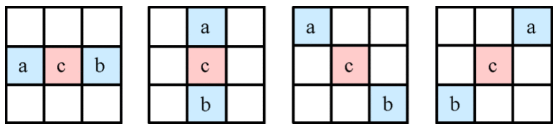


Figure 1.9: Patterns corresponding to four EO categories (from left to right): horizontal, vertical, 135° diagonal and 45° diagonal. Current sample is denoted c. Two reference neighboring samples are denoted a and b. Source: [7].

Each sample c is assigned a category based on it's neighbouring reference samples a and b according to this formula:

$$\text{category}(a,b,c) = \begin{cases} 1 & c < a \wedge c < b \\ 2 & (c < a \wedge c = b) \vee (c = a \wedge c < b) \\ 3 & (c > a \wedge c = b) \vee (c = a \wedge c > b) \\ 4 & c > a \wedge c > b \\ 0 & \text{otherwise} \end{cases} . \tag{1.2}$$

Similarly to BO, an average offset is calculated, transmitted and then added at the by the decoder for each category, except for category 0, for which no action is performed. Notice that both BO and EO require sending one identifier and four offset values.

Neural Networks and replacements for HEVC filters

This chapter contains a brief introduction to neural networks and an analysis of a neural network architecture that replaces DBF and SAO filters of HEVC and other useful architectures.

2.1 Brief introduction to Neural Networks

Neural Networks (NNs) are a subset of Machine Learning and they have proven their ability to approximate very complex functions with the usage of artificial neurons which are inspired by how real neurons function.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a subset of Neural Networks commonly used for image processing. They take advantage of spatial proximity of features, by sliding a kernel (of the same dimensionality as the input) across the input and calculating dot product of the filter with the part of input it's aligned with in order to obtain a new feature map (also of the same dimensionality as the input).

2.1.2 Batch Normalization

Batch Normalization[4] layer normalizes its inputs across each channel individually by subtracting its mean and dividing by its standard deviation during training. These values are tracked with a running mean, which is used when performing evaluation. Batch Normalization is very useful as it prevents exploding and vanishing values and their gradients.

2.1.3 Activation function

Activation function is applied in order to introduce non-linearity. Non-linearity is required in order to approximate complex functions.

One commonly used activation function is *ReLU* which is defined as:

$$\mathrm{ReLU}(x) = \begin{cases} x & x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.1}$$

Papers such as [16] indicate that activation function *LeakyReLU* defined as:

$$\mathrm{LeakyReLU}_{\alpha}(x) = \begin{cases} x & x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.2}$$

can performs better than ReLU, because ReLU suffers from dying neurons problem. A neuron can die, when it's activated by ReLU and it happens that is always receives negative values as it's input. If that happen, then the neuron always outputs the value of 0 and never changes this behaviour, which is a direct

consequence of its zero derivative on negative inputs, which is shown in equation 2.3. LeakyReLU has non-zero derivative in all of its domain and thus does not suffer from the dying neuron problem.

$$\frac{\partial \,\mathrm{ReLU}(x)}{\partial x} = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$\frac{\partial \,\mathrm{LeakyReLU}_\alpha(x)}{\partial x} = \begin{cases} 1 & x > 0 \\ \alpha & \text{otherwise} \end{cases} \tag{2.4}$$

An activation function useful for limiting the range of model's output to $(-1, 1)$ is tanh which can be written as:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{2.5}$$

2.2 UNet

UNet[10] is an architecture originally designed to perform segmentation of medical images. The name originates from the shape of the architecture, which is shown in figure 2.1. It implements an *encoder-decoder* structure with skip connections between the two. Encoder extracts semantic information about the image, while loosing spatial information. Decoder uses skip connections to regain lost spatial information of features produced by the encoder while upscaling said features. This architecture was truly groundbreaking at the time of its release and multiple other architectures have been inspired by it.

UNet can produce output maps that contain both semantic and spatial information. Additional benefit of implementing the encoder-decoder structure is the vastly increased field of view, compared to networks that operate on a single scale of images. It also means that majority of operations are performed on features of resolution significantly lower than resolution of input. This property can lead to an architecture that performs inference significantly faster compared to a model without any downscaling operations.

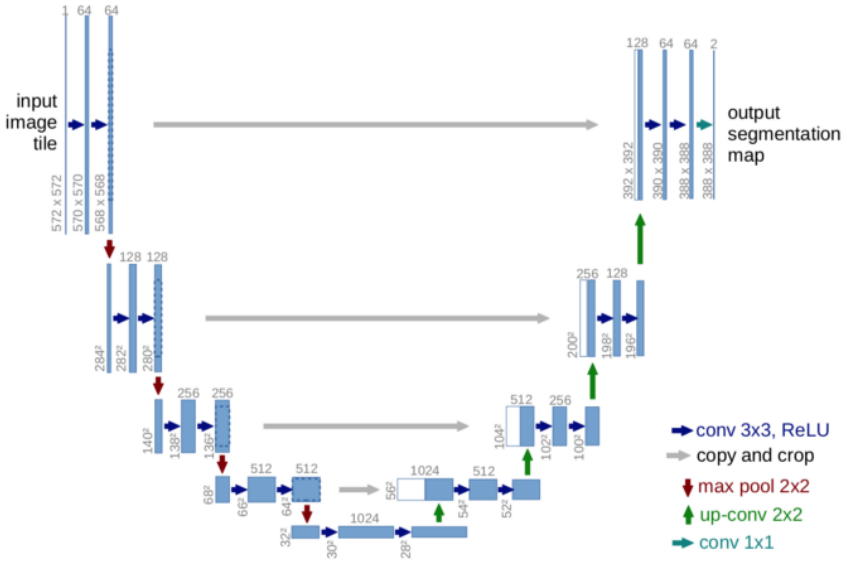
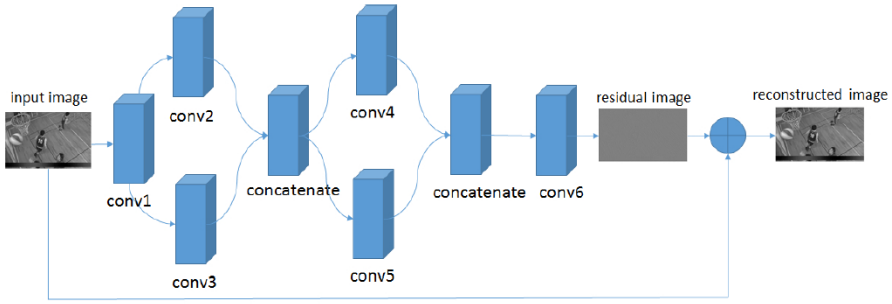


Figure 2.1: Original U-Net architecture. Source: [10]

2.3 Substitutions for DBF and SAO: VRCNN and VRCNN-BN

There are different architectures meant to replace different parts of HEVC pipeline, but this work will focus on replacing DBF and SAO filters. VRCNN[2] (Variable-filter-size Residue-learning Convolutional Neural Network) and VRCNN-BN[20] (VRCNN with Batch Normalization) are CNNs that use variable filter size to better work with variable CUs size in HEVC, although it is very shallow and thus has small field of view. Deeper architectures like RHCNN [18] have generally performed better, at the cost of having significantly more parameters. MMS-net [6] uses two parallel sub-networks of different scales. VRCNN and VRCNN-BN will be explored in more depth.

VRCNN is an architecture meant to replace DBF and SAO filter and remove compression artifacts in intra coded frames. It is inspiered by AR-CNN [17], but it introduces variable filter size, which effectively means that two convolutions with different kernel sizes are performed in parallel branches, which are then concatinated. The architecture is shown in figure 2.2.



Layer	Layer 1	Layer 2		Layer 3		Layer 4
Conv. module	conv1	conv2	conv3	conv4	conv5	conv6
Filter size	5×5	5×5	3×3	3×3	1×1	3×3
# filters	64	16	32	16	32	1
# parameters	1600	25600	18432	6912	1536	432
Total parameters	54512					

Figure 2.2: Architecture of VRCNN from [2]. Note that conv2 has different filter size than conv3 and conv4 has different filter size than conv5.

As the name suggests VRCNN (Variable-filter-size Residue-learning Convolutional Neural Network) works by predicting the residue between the luminance channel of compressed and reference frame. Processing only the luminance channel was chosen, since in YUV colorspace it is the channel with most perceptual significance, which can be seen in figure 2.3.

(a) Complete frame (b) Y: luminance channel (c) U: chrominance channel (d) V: chrominance channel

Figure 2.3: Example frame split into Y (luminance) and U and V (chrominance) channels. Note how difficult it is to see any details on U and V channels compared to Y channel.

VRCNN is implemented as an out-of-loop filters for simplicity and therefore is tested only on intra frames. That is because for intra frames there is no difference between an in-loop and out-of-loop filters.

VRCNN-BN (VRCNN with Batch Normalization) proposed in [20] is a modification of VRCNN, which notably adds batch normalization layers described in section 2.1.2 to each block and suggests training two separate models: one for luma channel, and one for chroma channels. Its architecture look as shown in 2.4.

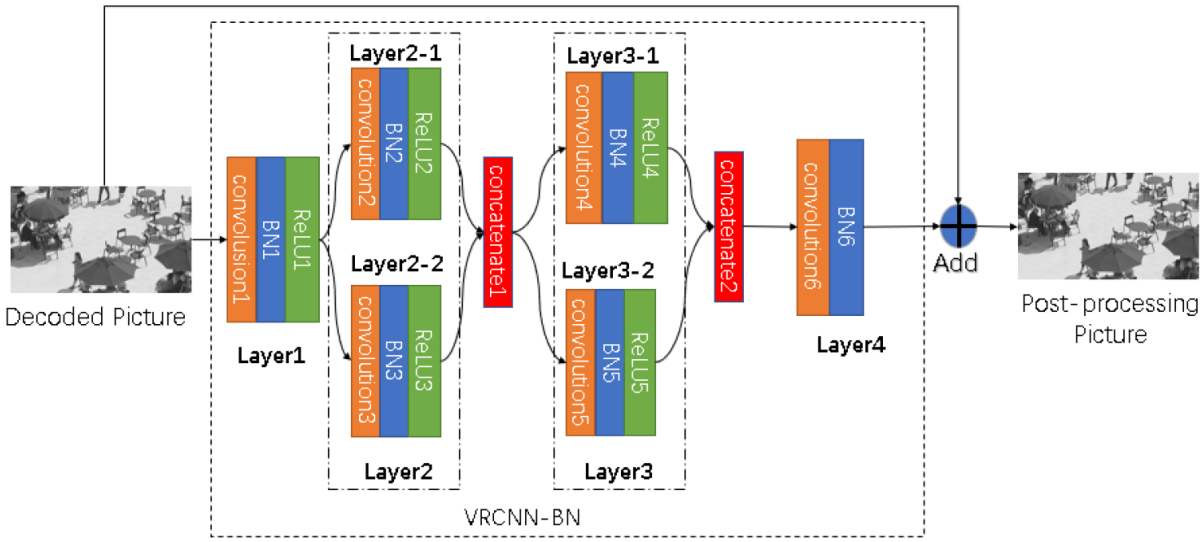


Figure 2.4: VRCNN-BN architecture. Source: [20]

Unfortunately neither of these architectures is able perform inference in real-time on 1280x720 frames on mainstream hardware. That is despite it being a very shallow architecture. Main problem of VRCNN and VRCNN-BN is that they have a lot of large filter, all of which operate on full resolution frames. Working on high resolution frames is important to produce high resolution output image, but it should be possible to gain some semantic understanding of processed frame in resolution lower than native, thus allowing for faster inference, without making the model overly simplified.

Proposed architecture: VFS-UNet

Goal of this work it to propose an architecture that will improve quality of streamed video, by replacing DBF and SAO in HEVC. For simplicity sake however, similarly to VRCNN, propsed architecture is not implemented in-loop and thus all test were performed in AI (All Intra) mode[1]. It must be able to perform inference of 1280x720 frames in real time, that is at lest 30 times per second on GTX 1060.

There are a couple of key ideas that were applied while designing the proposed architecture. Each one is explained and motivated.

3.1 Specializing model for one domain

Generally as models become lightweight, they are also becoming less capable and cannot generalize as well. Simplifying the task that the model performs obviously allows for simplification of the model's architecture, which is desired when the goal is to speed up the model. Such simplification of performed task might be narrowing range of supported data to a given domain. While it is difficult to define a domain in this context, for testing purposes, a domain is considered a set of frames, captured from the same videogame. Example frames for the same domain are shown in 3.1 and images for different domain in 3.2.

Figure 3.1: Two images considered from the same domain

[1] In theory however, it could be implemented as an in-loop filter

Figure 3.2: Two images considered from different domains

While difficult to fully quantify in, it can be clearly seen that images from the same domain have a consistent style of image and share common structures and features. Images from different domains on the other hand might share no common features at all.

As already mentioned in section 2, solutions specialized in one specific domain could have a legitimate used in live-streaming services like Twitch or game streaming services like Amazon Luma. In order to allow for a simplified architecture to perform well, an assumption is made that a given model is specialized in one domain of data. When testing, a separate model is prepared for each tested dataset and each tested QP value.

An inconvenient consequence of having a specializing model per domain is that weights of a corresponding model must be transferred beforehand. This however would need to be performed only once and thus would not be very problematic in real-life use.

3.2 Architecture description

Proposed architecture is meant to remove compression artifacts from frames and do it within a time budget. Similarly to VRCNN, it predicts a residue, which is then added to input frame.

Proposed architecture has an encoder-decoder structure similarly to UNet described in section 2.2. In order to simplify the model it has a small amount of filters in each layer and unlike UNet, it includes only one layer on each level of encoder and decoder, instead of two used in the original paper. This greatly reduces complexity and allows for significantly faster inference.

Additionally proposed architecture incorporates Variable Filter Size for every used convolution. This effectively means each regular convolution is substituted with two parallel convolutions with kernels of different sizes. Because of the UNet-like basic structure and the usage of Variable Filter Size, proposed architecture will be referred to as VFS-UNet (Variable Filter Size UNet) for simplicity.

VRCNN removed artifacts only on luma channel of frames. VRCNN-BN removed artifacts from all channel, but still requires performing one pass per channel, so three passes per frame. Data in each channel is related to other channels, so it would be more optimal to perform a single pass for all channels. VFS-UNet processes all channel of a frame in a single pass.

Let $\text{conv}_{k\times k}$ denote a convolution with filter size of $k \times k$ and a padding value of $\frac{k-1}{2}$.

3.2.1 Encoder

Encoder Block

Let each building block of encoder be denoted E_c. Then block E_c consists of:

- two parallel convolutions $\text{conv}_{3\times3}$ and $\text{conv}_{5\times5}$, both having $\frac{c}{2}$ filters and a stride value of 2,
- concatination of two branches,
- batch normalization,
- LeakyReLU activation function.

Note that given block $E_{C_{out}}$ and its input of shape (C_{in}, H, W), then its output will take shape of $(C_{out}, \frac{H}{2}, \frac{H}{2})$, which means that spatial resolution of input is halved on each axis.

Blocks configuration

Encoder consists of four stacked blocks: E_{12}, E_{16}, E_{24}, E_{48}.

3.2.2 Decoder

Decoder Block

Decoder blocks make use of **Pixel shuffling**[11], which is an operation of rearranging pixels in layer of size (r^2C, H, W) into a size of (C, rH, rW). This one of ways to increase resolution of a layer using neural networks. Ordering of pixels during pixel shuffling is shown in figure 3.3. In each block value of r is set to 2, so spatial resolution is doubled on each axis when processed by each block.

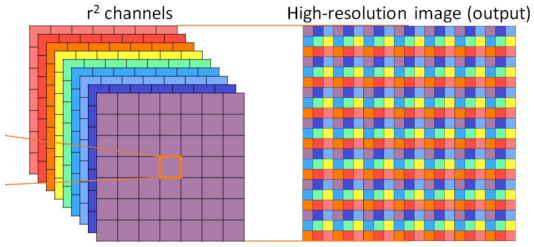


Figure 3.3: Visualization of pixel shuffling operation. It converts multiple low resolutions layers into higher resolution layer. Source: [11]

Let each building block of encoder be denoted D_c. Then block D_c consists of:

- two parallel convolutions $\text{conv}_{3\times3}$ and $\text{conv}_{5\times5}$, both having $2c$ filters and a stride value of 1,
- pixel shuffling outputs of both convolutions
- concatination of two branches,
- batch normalization,
- LeakyReLU activation function,
- concatinating output of activation function with the output of corresponding encoder block

Block configuration

Decoder consists of four stacked blocks: D_{48}, E_{24}, E_{16}, E_8.

3.2.3 Convolutions after decoder

After decoder produced its output, the following steps are executed in order to achieve final, improved frame:

- decoder's output is passed through two parallel convolutions: $\text{conv}_{3\times3}$ and $\text{conv}_{5\times5}$, both having 3 filters,
- concatination of two branches,
- passing data through a single $\text{conv}_{1\times1}$ with 3 filters,
- applying tanh activation function in order to obtain predicted residue,
- adding predicted residue to input image.

Architecture of VFS-UNet is show in figure 3.4. It contains 201 191 parameters, which is roughly 4 times more than VRCNN.

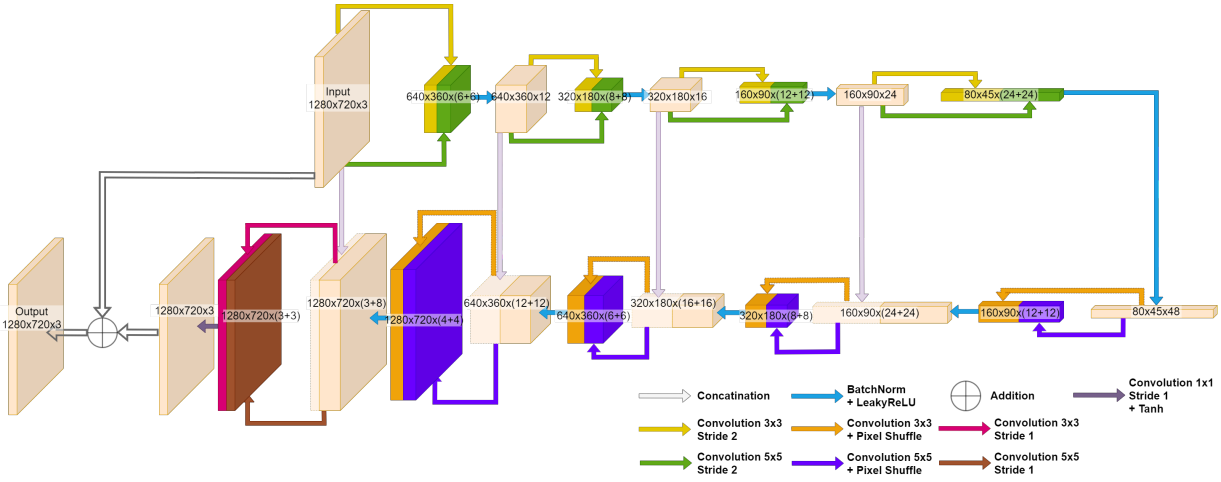


Figure 3.4: Proposed architecture: VFS-UNet

Additionally a very simple trick can be performed to check VFS-UNet's field of view of view without performing any calculations or analysis. Instead we can find it, by:

- initializing weights of filters in all convolutions to 1,
- setting bias and standard deviation in batch normalization to 0 and 1 respectively,
- inferring an image containing only zeros, except for center pixel, which has a value of 1,
- binarizing output image, be setting all nonzero values to 1.

Final image is shown on figure 3.5. The result is a 172×172 square filled with ones, which represents the field of view of the that proposed architecture.

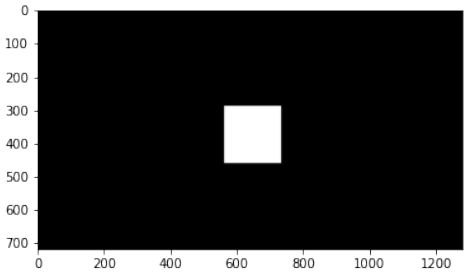


Figure 3.5: Visual representation of field of view of VFS-UNet. White block is representative of model's 172×172 pixel field of view.

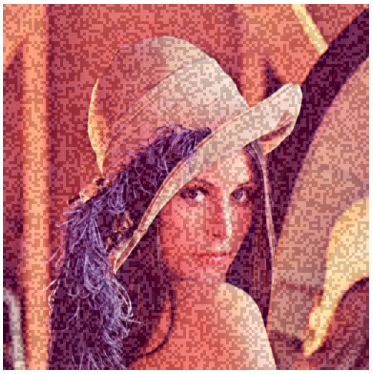
3.3 Used loss function

Authors of [19] suggest that using a combination of ℓ_1 loss and MS-SSIM$_{\text{loss}}$ produces good visual results. Their loss is described as:

$$\text{MIXED}_{\text{loss}}(X,Y) = \alpha\,\text{MS-SSIM}_{\text{loss}}(X,Y) + (1-\alpha)\ell_1(X,Y), \tag{3.1}$$

with proposed value of α being 0.84. MIXED$_{\text{loss}}$ is meant to capture the best features of its components, which on their don't always work well, example of which is shown on figure 3.6.

(a) Original image $\ell_1 = 0.00$, MS-SSIM$_{\text{loss}} = 0.000$



(b) Image with random binary noise $\ell_1 = 0.0784$, MS-SSIM$_{\text{loss}} = 0.4017$

(c) Image with uniform bias $\ell_1 = 0.0784$, MS-SSIM$_{\text{loss}} = 0.0031$

Figure 3.6: Notice identical ℓ_1 on 3.6b and 3.6c, despite very different visual quality. On the other hand, MS-SSIM$_{\text{loss}}$ is very insensitive to the uniform bias in 3.6c, which is picked up well by ℓ_1.

3.3.1 ℓ_1 component

ℓ_1 loss is described by this simple equation:

$$\ell_1(X,Y) = \frac{1}{N}\sum_{i=0}^{N}|X_i - Y_i| \tag{3.2}$$

where N is the number of pixels in input image. According to [19] ℓ_1 can yield better visual results compared to ℓ_2, because it does not over-penalize errors.

Since ℓ_1 loss is calculated per pixel it is quite good at preserving general colors and brightness, but sometimes fails to detect noise. Example of this is shown in 3.6.

3.3.2 MS-SSIM$_{\text{loss}}$ loss component

SSIM

SSIM was originally published in [12] and is a differentiable function meant to quantify the visibility of differences between distorted and reference images. Unlike ℓ_1 or ℓ_2, which compare images pixel by pixel, SSIM tries to mimic human visual perception, that is focused mostly on the structural information of an image.

To compare two images authors make three comparisons, that are later combined: luminance, contrast and structure. These values are calculated locally on patches of size $k \times k$[2]. The number of pixels in patch is denoted N. Each pixel in patch is weighted by circular-symmetric Gaussian weighting function denoted ω with standard deviation of 1.5 normalized so that $\sum_{i=1}^{N} \omega_i = 1$.

Luminance of each patch is denoted μ and calculated as

$$\mu_x = \sum_{i=1}^{N} \omega_i x_i \tag{3.3}$$

Standard deviation is used as an estimate of contrast and is denoted σ.

$$\sigma_x = \sqrt{\sum_{i=1}^{N} (x_i - \mu_x)^2} \tag{3.4}$$

Final function consists of three components that are combined:

$$\text{SSIM}(x, y) = f(l(x, y), c(x, y), s(x, y)) \tag{3.5}$$

where l is comparison of luminance, c is comparison of contrast and s of structure.

Luminance comparison for patches x and y is defined as follows:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{3.6}$$

where C_1 is a constant used for numerical stability. This equation is related to Weber's law, which states that as the stimulus gets stronger, the larger increase of its intensity is needed to perceive a difference in sensation.

Similarly, constrast comparison for patches x and y is defined as follows:

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{3.7}$$

where again C_2 is a constant used for numerical stability.

Structural comparison:

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{3.8}$$

[2]Authors propose window of size 11×11

where

$$\sigma_{xy} = \sum_{i=1}^{N} \omega_i (x_i - \mu_x)(y_i - \mu_y) \tag{3.9}$$

This equation is equivalent to Pearson correlation coefficient between x and y. Result is not divided by N, because $\sum_{i=1}^{N} \omega_i = 1$. Finally

$$\text{SSIM}(x, y) = l(x, y)^{\alpha} c(x, y)^{\beta} s(x, y)^{\gamma}, \tag{3.10}$$

where authors propose $\alpha = \beta = \gamma = 1$. Then:

$$\text{SSIM}(x, y) = l(x, y)c(x, y)s(x, y)a = \frac{(2\mu_x\mu_y + C_1)(2\sigma_x\sigma_y + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.11}$$

Mean SSIM (MSSIM, not to be confused with MS-SSIM) is defined as

$$\text{Mean-SSIM}(X, X) = \frac{1}{M} \sum_{j=1}^{M} \text{SSIM}(X_j, Y_j) \tag{3.12}$$

where X and Y are distorted and reference images, M is the number of pixels in each one and X_j, Y_j are patches centered around j-th pixel in image.

MS-SSIM

MS-SSIM described in [13] stands for Multi-Scale SSIM. It applies a series of low-filter passes and down-scalings by a factor of 2 on each axis to obtain reference and distorted images of multiple scales. Authors proposed using five scales, indexing them from 1 (original image) to 5 (smallest image). Luminance, contrast and structure values are calculated and denoted as l_i, c_i, s_i respectively. MS-SSIM is defined as:

$$\text{MS-SSIM}(x, x) = l_M(X, Y)^{\alpha_5} \prod_{j=1}^{5} (c_i(x, y)^{\beta_i} s_i(x, y)^{\gamma_i}), \tag{3.13}$$

where authors proposed $\beta_1 = \gamma_1 = 0.0448$, $\beta_2 = \gamma_2 = 0.2856$, $\beta_3 = \gamma_3 = 0.3001$, $\beta_4 = \gamma_4 = 0.2363$ and $\alpha_5 = \beta_5 = \gamma_5 = 0.1333$.

Given all of above the $\text{MS-SSIM}_{\text{loss}}$ can be defined as:

$$\text{MS-SSIM}_{\text{loss}}(X, Y) = 1 - \text{Mean-MS-SSIM}(X, Y) \tag{3.14}$$

Datasets

This chapter contains description of collected datasets including their quantity and how they was collected.

Data quality and quantity are very important factors when it comes to training neural networks. Even more so in this work, since presented solution is designed to be specialized for certain domain of data. Because of this each dataset needs to represent associated domain well.

Each dataset is a collection of high quality captures from a chosen game, each one from different genre. Clips have been recorded using Nvidia Experience software at 720p@30fps at 50Mbps, which is a very high bitrate for this resolution and framerate. To make validation as fair as possible, entire clips were used for validation, instead of individual frames. Using individual frames might have caused very similar images to exist in training and validation datasets. Number of frames collected for each training and validation dataset is shown in 4.1.

	GTA V	Katana Zero	Team Fortress 2	Trackmania
train dataset:	182 337	208 202	-	161 908
valid dataset:	13 217	23 733	-	33 691

Table 4.1: Number of frames in each collected dataset

High quality clips are compressed using ffmpeg with HEVC encoding. Four QP settings are used: 32, 35, 37 and 39, where higher number indicates smaller bitrate and worse quality.

Average bitrate of each dataset on each preset is shown in 4.2. It is very important to note, that recorded bitrates are much higher compared to what they would be in real use-case, because of the usage of All-Intra mode.

	GTA V	Katana Zero	Team Fortress 2	Trackmania
QP=39 **without** dbf and sao:	5 456 kbps	4 653 kbps	-	5 560 kbps
QP=39 **with** dbf and sao:	5 532 kbps	4 696 kbps	-	5 606 kbps
QP=37 **without** dbf and sao:	6 880 kbps	5 779 kbps	-	6 787 kbps
QP=37 **with** dbf and sao:	6 961 kbps	5 832 kbps	-	6 839 kbps
QP=35 **without** dbf and sao:	8 771 kbps	7 329 kbps	-	8 393 kbps
QP=35 **with** dbf and sao:	8 813 kbps	7 330 kbps	-	8 406 kbps
QP=32 **without** dbf and sao:	12 359 kbps	10 041 kbps	-	11 091 kbps
QP=32 **with** dbf and sao:	12 412 kbps	10 042 kbps	-	11 106 kbps

Table 4.2: Bitrate of validation datasets dependent on preset

Implementation and training

This chapter briefly describes implementation and training of VFS-UNet.

Architecture described in section 3 and shown in figure 3.4 was implemented using PyTorch[9] and PyTorch Lightning[3]. Source code can be found under https://github.com/janpawlowskiof/VFS-UNet.

Library Ray[1] was used to speed up the training with the use of distributed computing. Four machines were used to perform parallel training.

Batch size of 8 was chosen and four machines were performing the training, meaning that each machine processed batches of only two elements.

For each dataset, a model was first trained for QP=37 preset for 100 000 steps. Then weights of that trained model were loaded and fine-tuned for 20 000 steps for QP=39, QP=35 and QP=32 presets.

[1] https://www.ray.io/

Benchmarks

This chapter contains benchmarks of trained models on different datasets compared to the baseline of DFB with SAO. All benchmarks were performed with code available at https://github.com/janpawlowskiof/VFS-UNet.

6.1 Metrics

Two commonly used metrics for estimating quality of compressed video are PSNR and MS-SSIM. MS-SSIM was already described in section 3.3.2. PSNR (Peak Signal to Noise Ratio) is a commonly used function for estimating quality of an image. It is usually expressed in decibel scale. For reference RGB image X and distorted image Y it is calculated simply as

$$\mathrm{PSNR}(X,Y) = 10\log_{10}\frac{3N\,{\mathrm{MAX_X}}^2}{\sum_{i=1}^{N}(X_{Ri}-Y_{Ri})^2+\sum_{i=1}^{N}(X_{Gi}-Y_{Gi})^2+\sum_{i=1}^{N}(X_{Bi}-Y_{Bi})^2}, \tag{6.1}$$

where pixels of X can be in range $[0, {\mathrm{MAX_X}}^2]$.
Frames used for calculating metrics for DBF+SAO are obtained by encoding and decoding reference clips from validation dataset using HEVC with both DBF and SAO filters **enabled**.

Frames used for calculating metrics for VFS-UNet are obtained by encoding and decoding reference clips from validation dataset using HEVC with both DBF and SAO filters **disabled** and then passing decoded frames through a VFS-UNet trained for the domain, from which clips originate.

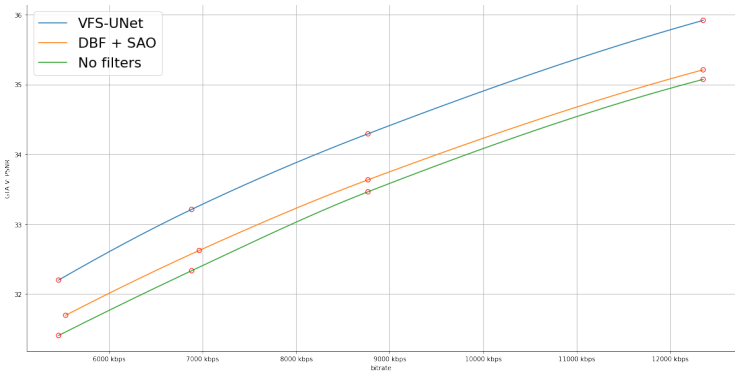
PNSR and MS-SSIM are calculated for VFS-UNet and DBF+SAO with four QP settings: 32, 35, 37 and 39. Results are listed in table 6.1 for PSNR and 6.2 for MS-SSIM. It is important to note that, clips used for calculating these metrics are not necessarily of identical bitrate. Improvement that takes bitrate into account is calculated in section 6.1. Plot that shows PSNR and MS-SSIM metrics with relation to bitrate have been plotted in figure 6.3 and figure 6.4 respectively.

	GTA V DBF+SAO	GTA V VFS-UNet	Trackmania DBF+SAO	Trackmania VFS-UNet	Team Fortress 2 DBF+SAO	Team Fortress 2 VFS-UNet	Katana Zero DBF+SAO	Katana Zero VFS-UNet
QP=32:	35,211 5	35,919 2	34,021 7	34,782 8	0	0	35,514 7	35,911 4
QP=35:	33,636 8	34,295 8	32,512 8	33,239 6	0	0	34,027 5	34,223 8
QP=37:	32,625 7	33,213 3	31,530 4	32,185 2	0	0	32,983 6	33,018 0
QP=39:	31,697 9	32,201 4	30,582 9	31,182 8	0	0	32,115 9	32,220 1

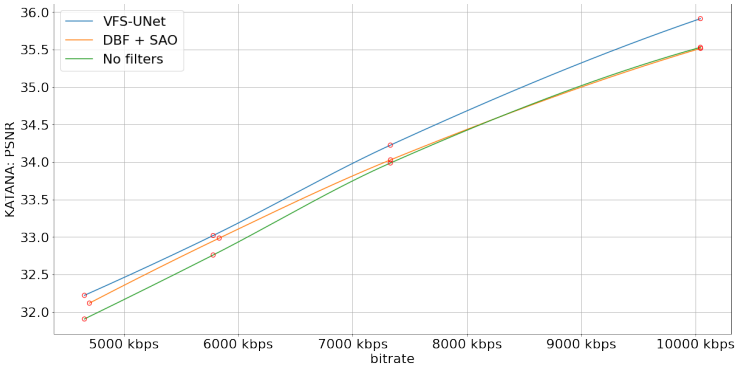
Figure 6.1: Achieved PSNR (dB) on validation datasets

	GTA V DBF+SAO	GTA V VFS-UNet	Trackmania DBF+SAO	Trackmania VFS-UNet	Team Fortress 2 DBF+SAO	Team Fortress 2 VFS-UNet	Katana Zero DBF+SAO	Katana Zero VFS-UNet
QP=32:	0,976 9	0,981 4	0,954 9	0,959 6	0	0	0,962 0	0,965 2
QP=35:	0,967 7	0,973 4	0,950 2	0,956 3	0	0	0,956 0	0,959 8
QP=37:	0,959 3	0,966 1	0,946 0	0,953 4	0	0	0,950 2	0,954 3
QP=39:	0,950 2	0,957 6	0,942 0	0,950 2	0	0	0,944 5	0,949 6

Figure 6.2: Achieved MS-SSIM on validation datasets



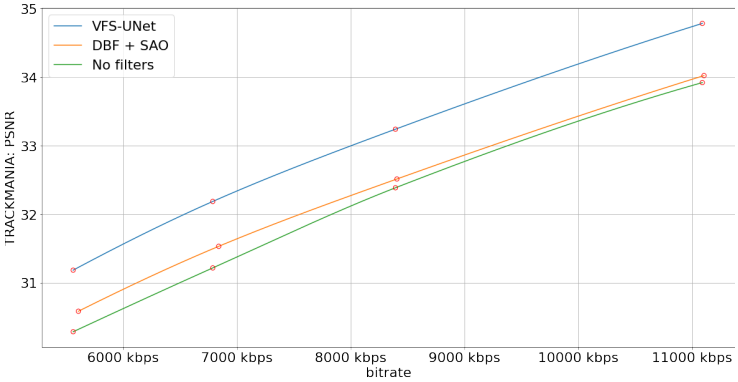
(a) GTA V



(b) Katana Zero

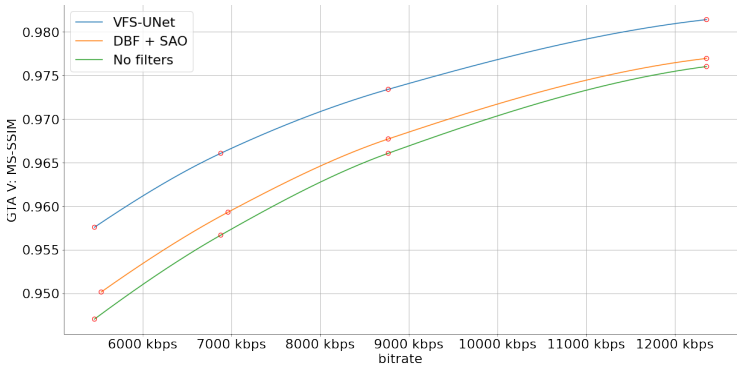
PLACeHOLDER

(c) Team Fortress 2

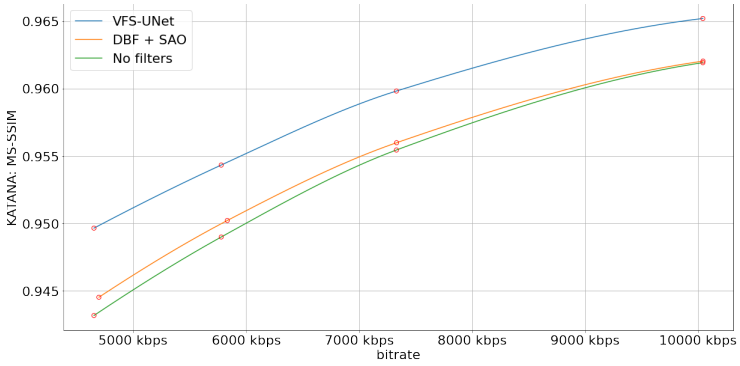


(d) Trackmania

Figure 6.3: PNSR plotted with relation to bitrate. Measurement points are marked with red dots. Drawn line is a third degree polynomial fit into measurement points, which is used to interpolate values of these functions.



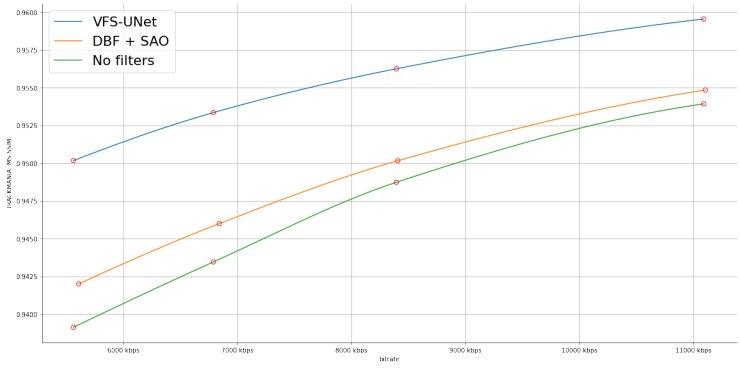
(a) GTA V



(b) Katana Zero

PLACeHOLDER

(c) Team Fortress 2



(d) Trackmania

Figure 6.4: MS-SSIM plotted with relation to bitrate. Measurement points are marked with red dots. Drawn line is a third degree polynomial fit into measurement points, which is used to interpolate values of these functions.

BD-Rate

BD-Rate[1] (Bjontegaard Delta-Rate) is commonly used for quantifying improvement over another video compression method. Calculating it requires measurement of a given metric at four different values of QP. First rates are converted into logarithmic rate and having a third degree polynomial fit into resulting points. That polynomial is then numerically integrated over. Finally, results are converted back into linear scale. This value is calculated for both VFS-UNet and compared to the value yielded by DBF+SAO which will be used as a baseline. Calculated BD-Rates are shown in 6.5 showing a relatively large bitrate savings.

	GTA V	Katana Zero	Team Fortress 2	Trackmania
BD PSNR:	0.6493	0.1893	-	0.7171
BD Rate PSNR:	−13.02 %	−3.93 %	-	−13.27 %
BD MS-SSIM:	0.00611	0.00404	-	0.00653
BD Rate MS-SSIM:	−17.77 %	−16.46 %	-	−32.96 %

Figure 6.5: BD-rates

As can be seen the amount of improvement is heavily dependent on the dataset. This demonstrates that the complexity of artifact removal task is heavily dependent on the domain. For example on Katana zero dataset there is very little improvement, especially on PSNR. This is probably due to the fact that this specific dataset is loosing most of the details in the scene, which causes relatively incapable model to perform poorely. Metrics do not show that in this dataset a lot of the details are lost (see figure 6.1) due to the large amounts of uniform colored surfaces in each frame that do not exhibit significant compression artifacts. Example of this is shown in figure 6.6.

(a) Fragment of reference frame

(b) Fragment of frame compressed with QP=37

Figure 6.6: Notice how many visually significant details have been lost during compression.

6.2 Performance

Target speed of proposed solution was processing 1280x720 frames at 30fps using NVIDIA GTX 1060. Inference speed on different hardware is demonstrated in table 6.7. Target inference speed was reached and a significant speedup was achieved over VRCNN and VRCNN-BN. During this benchmark, models were set to evaluation mode and no gradients were calculated. No other optimizations were performed. VRCNN and VRCNN-BN architectures were reimplemented in order to perform benchmarks.

	GTX 1060	RTX 2080 Ti	GTX 1050 Ti
VFS-UNet:	31.80 fps	108.61 fps	22.09 fps
VRCNN:	0 fps	61.22 fps	12.45 fps
VRCNN-BN:	0 fps	15.38 fps	3.35 fps

Figure 6.7: Inference speed comparison of 1280×720 frames using PyTorch 1.10 with Cuda 10.2 backend. Results are presented in frames per second. Note that VRCNN-BN requires three passes to be performed.

6.3 Visual comparisons

Metrics like PSNR are just an estimations of perceptual quality. Measuring perceptual quality by humans is very difficult to quantify, but it is very important as ultimately the goal of VFS-UNet is to produce visually pleasing results. A collection of images is shown in figures 6.8, 6.9, 6.10 and **??** presenting a comparison between reference images, images containing compression artifacts with no processing applied, images with DBF+SAO applied and images with VFS-UNet applied.

6.4 Benchmarks on data outside of learnt domain

A quick test is performed by calculating metrics on data from outside of learnt domain.
When processing data from outside of learnt domain BD-PSNR of 0.2319 was achieved by models trained on GTA V dataset tested on Trackmania dataset and -0.2418 by model trained on Trackmania dataset tested on Katana Zero dataset. Unsurprisingly achieved results are significantly worse compared to results shown in table 6.1. In the second case the results were even worse compared to DBF+SAO.

Figure 6.8: Visual comparisons between VFS-UNet and DBF+SAO on GTA V dataset. Images from left to right are full reference frame, close up on reference frame, close up on frame processed by VFS-UNnet and close up on frame processed by DBF+SAO.

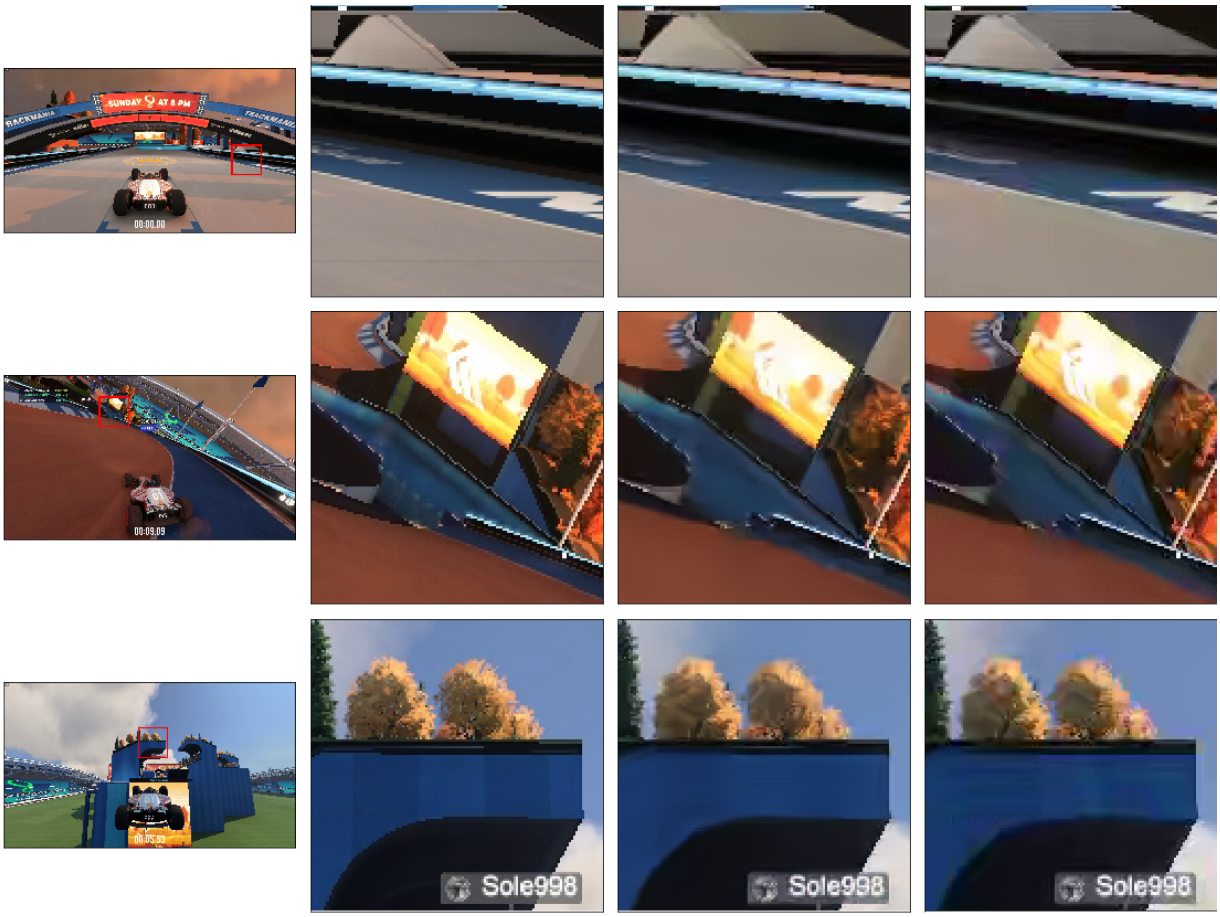


Figure 6.9: Visual comparisons between VFS-UNet and DBF+SAO on Trackmania dataset. Images from left to right are full reference frame, close up on reference frame, close up on frame processed by VFS-UNnet and close up on frame processed by DBF+SAO. Notice how in the second row reflection of the railing was accidentally removed by VFS-UNet, while it was not removed by DBF+SAO.

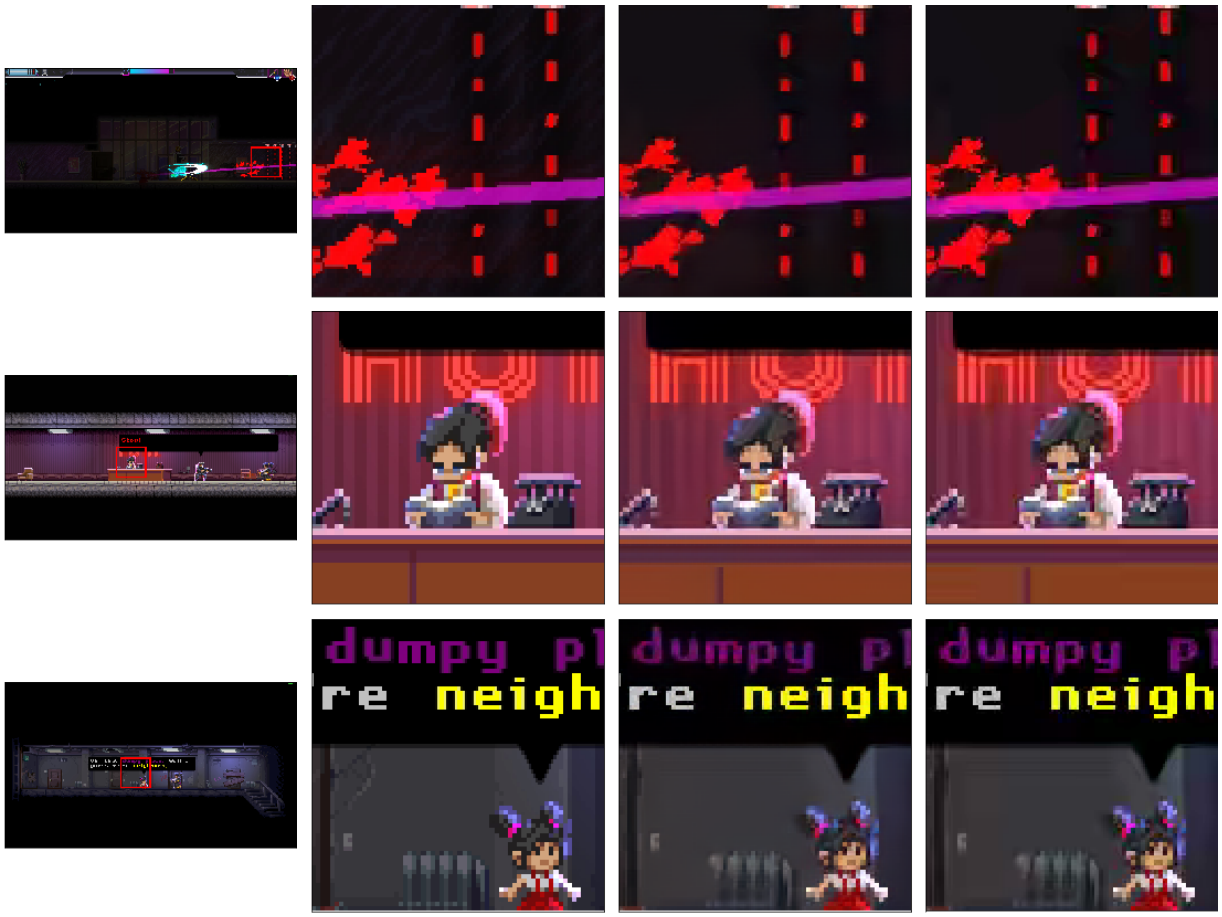
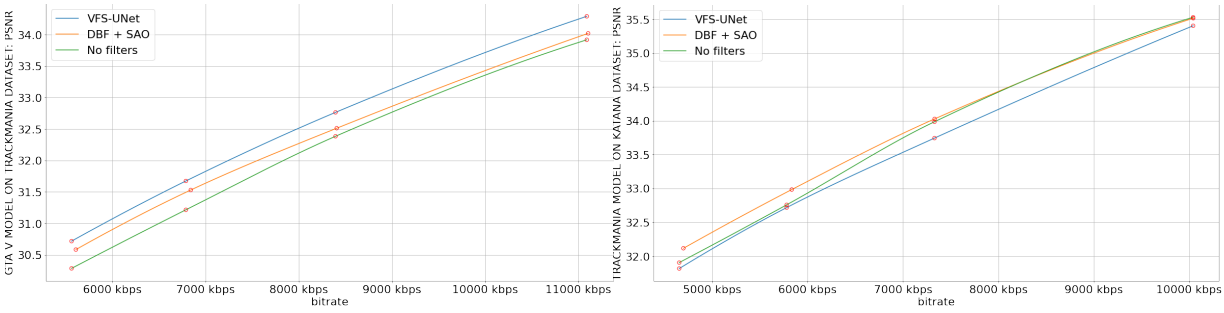


Figure 6.10: Visual comparisons between VFS-UNet and DBF+SAO on Katana Zero dataset. Images from left to right are full reference frame, close up on reference frame, close up on frame processed by VFS-UNnet and close up on frame processed by DBF+SAO.



(a) PSNR: GTA V model on Trackmania

(b) PSNR: Trackmania model on Katana Zero

Figure 6.11: Achieved PSNR when processing data from outside of learnt domain.

Summary and possible further refinements

This chapter includes summary of this work, alongside with possible further refinements.

In this work a CNN architecture was proposed that is designed for artifact removal on intra frames, and is meant to replace DBF and SAO filters in HEVC pipeline. Proposed architecture is inspired by VRCNN, but incorporates an encoder-decoder structure of UNet. Additionally models are specialized in specific domains, instead of creating one universal model. It achieves better PSNR and MS-SSIM metrics compared to DBF and SAO HEVC filters. It was designed with the performance target of performing inference on 1280×720 frames at least 30 times per second on GTX 1060 and it achieves that target. It is significantly faster compared to VRCNN[2] or RHCNN[18]. There are however further refinements that can be made to make proposed solution better.

Notably while the target resolution of 1280×720 is not unreasonable it is relatively low compared to modern standards for movies for example, which are watched in resolutions up to 3840×2160. For game streaming on the other hand the target framerate of 30 is low, as currently games are most commonly streamed at 60 frames per second, instead of the 30 that this work was targeting. It is obvious that further refinements would be required in order to apply this solution in a real use-case.

7.1 Quantization

Currently when testing, inference is performed using fp32 precision. A lot of performance can be gained from quantization. Quantization refers to replacing fp32 weights and inputs with types that are of lower precision and faster to operate on, for example fp16 or even int8.

7.2 Exporting model to TensorRT

TensorRT[1] is an SDK for high performance inference of deep learning models. It can give a significant performance boost as it performs optimizations dependent on the system's configuration.

7.3 Performing hyperparameter search on the number of filters in model

Proposed version of VFS-UNet has a fixed number of filters in each layer. Performing a hyperparameter tuning on the number of filters could help achieve faster performing model that yields similarly visual results.

[1] https://developer.nvidia.com/tensorrt

7.4 Targeting systems with tensor cores

Neural networks can be accelerated greatly with tensor cores that are present for example on NVIDIA RTX 20 series and 30 series graphics cards. NVIDIA uses these tensor cores for example to accelerate their DLSS 2.0 (Deep Learning Super Sampling). Graphics cards that include tensor cores are becoming more mainstream and it might make sense to target only systems that include them.

7.5 Implementing VFS-UNet in-loop

VFS-UNet was implemented out-of-loop and thus tested only on intra-frames. For further testing it should be implemented in-loop in order to evaluate its performance on inter prediction. This however will require significantly different training of the model, which would be significantly more difficult as model influences its own input during inter-prediction.

Bibliography

[1] G. BJONTEGAARD. Calculation of average psnr differences between rd-curves. *VCEG-M33*, 2001.

[2] Y. Dai, D. Liu, F. Wu. A convolutional neural network approach for post-processing in hevc intra coding. *Lecture Notes in Computer Science*, strona 28–39, Dec 2016.

[3] W. Falcon, The PyTorch Lightning team. PyTorch Lightning, 3 2019.

[4] S. Ioffe, C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[5] Y. Jiang, D. Llamocca, M. Pattichis, G. Esakki. A unified and pipelined hardware architecture for implementing intra prediction in hevc. 04 2014.

[6] J. Kang, S. Kim, K. M. Lee. Multi-modal/multi-scale convolutional neural network based in-loop filter design for next generation video codec. *2017 IEEE International Conference on Image Processing (ICIP)*, strony 26–30, 2017.

[7] C. ming Fu, E. Alshina, E. Alshin, Y. wen Huang, C. yeh Chen, C. yang Tsai, C. wei Hsu, S. min Lei, J. hoon Park, W. jin Han. Sample adaptive offset in the hevc standard.

[8] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, G. Van der Auwera. Hevc deblocking filter. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22:1746–1754, 12 2012.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. Pytorch: An imperative style, high-performance deep learning library. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett, redaktorzy, *Advances in Neural Information Processing Systems 32*, strony 8024–8035. Curran Associates, Inc., 2019.

[10] O. Ronneberger, P. Fischer, T. Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[11] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[12] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4):600–612, 2004.

[13] Z. Wang, E. Simoncelli, A. Bovik. Multiscale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, wolumen 2, strony 1398–1402 Vol.2, 2003.

[14] Z. Wang, S. Wang, J. Zhang, S. Wang, S. Ma. Effective quadtree plus binary tree block partition decision for future video coding. *2017 Data Compression Conference (DCC)*, strony 23–32, 2017.

[15] T. Wiegand, G. Sullivan, G. Bjontegaard, A. Luthra. Overview of the h.264/avc video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[16] B. Xu, N. Wang, T. Chen, M. Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[17] K. Yu, C. Dong, C. C. Loy, X. Tang. Deep convolution networks for compression artifacts reduction, 2016.

[18] Y. Zhang, T. Shen, X. Ji, Y. Zhang, R. Xiong, Q. Dai. Residual highway convolutional neural networks for in-loop filtering in hevc. *IEEE Transactions on Image Processing*, 27(8):3827–3841, 2018.

[19] H. Zhao, O. Gallo, I. Frosio, J. Kautz. Loss functions for neural networks for image processing, 2018.

[20] H. Zhao, M. He, G. Teng, X. Shang, G. Wang, Y. Feng. A cnn-based post-processing algorithm for video coding efficiency improvement. *IEEE Access*, 8:920–929, 2020.