Question 1: (20 Points) *Introduction to Videos and ffmpeg*

(a) Describe in your own words and on a basic level the following terms:

(a) **Resolution**

In the context of digital images, resolution describes how many pixels are in each given area of the displayed image (or usually, in the whole image). A digital image is made up out of squares (so called *pixels*) and more squares per given area, generally means higher image quality.

For example, resolution of “64/square inch” means that each square inch of the image is made of 64 individual squares.

In practice, resolution is often expressed as pixels for the entire video: width * height of pixels in each image. For example, resolution of 1600 * 900 means that the image is made of 1600 columns of pixels, each column contains 900 pixels.

The resolution of an image is the *highest* resolution the image can be displayed at, it does not mean that the image will be displayed in this resolution. If the display does not support the resolution of the image, the image will not be shown in its original resolution but in a downgraded resolution using the process of down sampling.

(b) **Frames per second**

A video is a sequence of images displayed one after another. The term frames per second describes how many (different) images are displayed in each second of the video.

For example, 60 frames per second means that each second 60 images are displayed one after another.

(c) **Bitrate**

In general the term bitrate describes the amount of data which is processed (/streamed/displayed) during a certain time frame (usually per second).

In the context of video, bitrate also describes how many bits of data are processed each given time frame which is generally the product of the image size for each frame and how many frames per seconds are in the video. For example, video bitrate of 3.8 Mbit/s means that each second, 3.8 megabits are processed.

(d) **Codec**

Codec is a package of software that implements encoding and decoding for an encoding standard, and so it provides the ability to store and display video files in different formats.

Every codec contains an **encoder**, for encoding video files and store them in a **container** that holds the video data and provides Meta data on it (encoding: representing the data in a specific digital way). A **decoder**, for decoding the file (decoding: restoring the original data from the encoded data)

(e) **Compression**

In the context of image/video compression, compression is the encoding process by which the size of an image or video file is digitally reduced.

For example:[1] the size of one-hour RGB video at 720p resolution, 24 bits depth and 30 fps without encoding will be - width * height * color-depth * fps * seconds = 1,280 * 720 * 24 * 30 * 3,600 = **278.09 GB**

Compressed, the video can be stored in a file of less than **6 GB** in size.

(f) Raw video

A raw video is a video file without any encoding, just digital representation (RGB for example) of a images shown in rapid succession.

For example, the calculation form the previous definition would be a raw video file.

(g) **Motion vector**

Motion vector describes motion related change between 2 2D images. Motion vector hold information about where a given object has come from, and can therefore fully describe a new image out of existing image, without to re-describe each pixel.

[1] https://github.com/leandromoreira/digital_video_introduction

For example, if we have a sequence of 3 images in which a ball is moving from the left to the right while all the rest is the same, we can describe the first and second image by the means of motion vectors out of the third image.

(b) How does the bitrate change if the resolution increases? Give a short explanation!

If the resolution increases the bit rate will increase as well.

Explanation: An increase in the resolution requires more data for the additional pixels of each frame, therefore more data needs to be processed in each given time frame. If we talk about a raw video, like in the 1.(a).6. example calculation, one can see the direct impact of the resolution, represented by width and height, in the product. Hence the overall bitrate would increase.

(c) How does the bitrate change if the frames per second decreases? Give a short explanation!

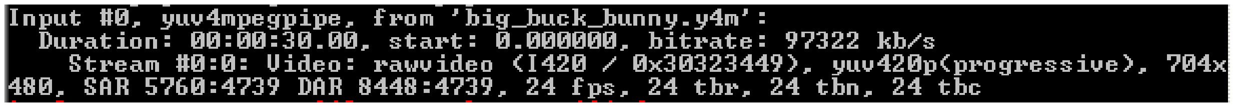
If the frame per second decrease, the bitrate will decrease as well.

Explanation: A decrease in the frame per second means less data needs to be processed at each given time frame, and so the bitrate will decrease. If we talk about a raw video, like in the 1.(a).6. example calculation, one can see the direct impact of the framerate in the product.

(d) Use ffmpeg to find out (1)-(3) from sub task (a) for the Big Buck Bunny clip.

Command – “ffmpeg -threads 1 -i big_buck_bunny.y4m -hide_banner”

Output –



```
Input #0, yuv4mpegpipe, from 'big_buck_bunny.y4m':
  Duration: 00:00:30.00, start: 0.000000, bitrate: 97322 kb/s
    Stream #0:0: Video: rawvideo (I420 / 0x30323449), yuv420p(progressive), 704x
480, SAR 5760:4739 DAR 8448:4739, 24 fps, 24 tbr, 24 tbn, 24 tbc
```

Resolution: 704 * 480

Frame per second: 24

Bitrate: 97322 kb/s

Question 2: (30 Points) *Moving Pictures and Motion Vectors Download* all images from https://service.inet.tu-berlin.de/owncloud/index.php/s/fGGIKX24ZOpqTdV. Transform them to YUV and only consider the Y-channels in the following.

Miscellaneous:

- ❖ The images are referred to by the last 2 digits of the image name
  For example: “0005091” → “91”
- ❖ difference between images is notated as “xx-xx”, for example, “01-00” is the difference between the matrix representations of image 01 (00005901.png) and 00 (00005900.png)

2(a) Compute the entropies for all of the images:

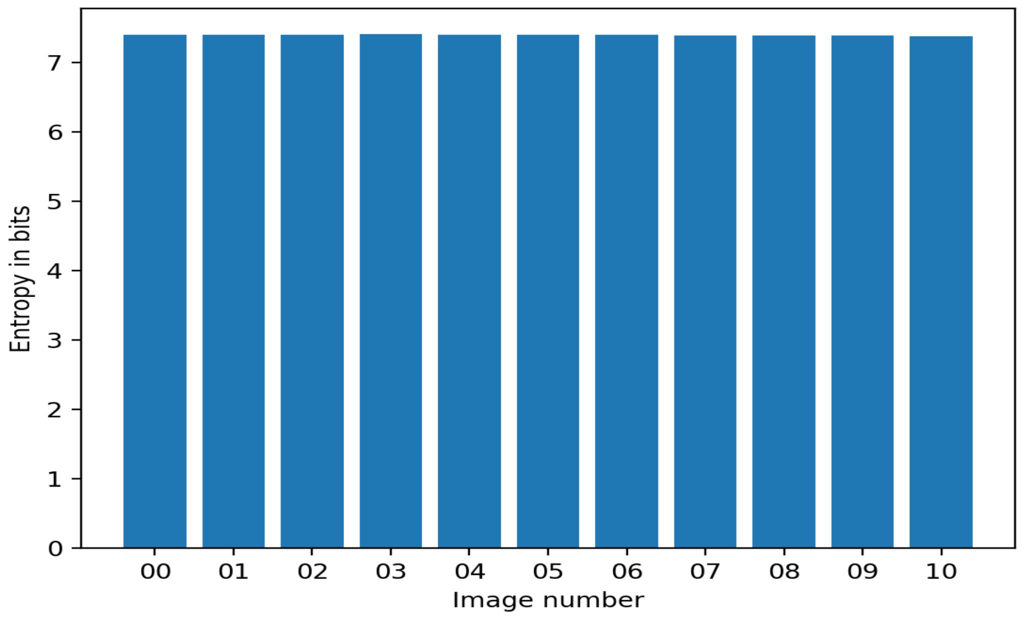
Solution 2(a):

The values of the images entropies in bits:

Image number	Entropy in bits
00	7.399868759259168
01	7.399149056362563
02	7.402273665239967
03	7.406282635822052
04	7.402505900254265

05	7.396249556478643
06	7.396094143157574
07	7.390947769286206
08	7.390346999161699
09	7.386110537875433
10	7.382121441124786

Individual entropies of images are very similar:



In all, the entropy values differ from 7,38 to 7,40 bits. It means we'd need around 7,4 * N bits of information to encode every image separately, when N is the number of pixels to encode.

For code see "q2_a.py"

2(b) Compute the entropies of the subsequent images' differences and visualize your results:

Solution 2(b):

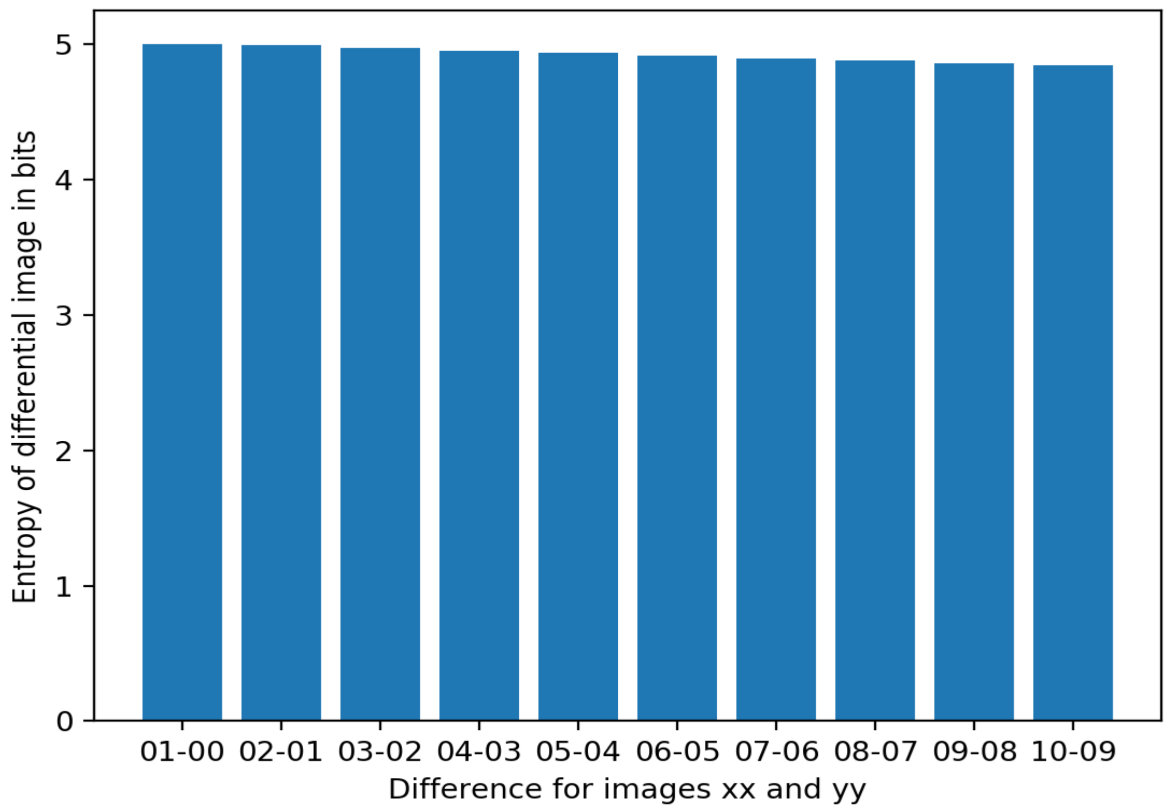
The entropies of the subsequent images' differences:

Image difference	**Entropy in bits**

01-00	4.998878662055725
02-01	4.992473278382965
03-02	4.976343811064164
04-03	4.954573053642677
05-04	4.939840387942192
06-05	4.914682514325139
07-06	4.895761531708138
08-07	4.88263293753002
09-08	4.858159577537898
10-09	4.842477014683971

Entropy of differential images is also very similar, but they keep decreasing:

Visualizing the results:



If we were to transmit only differences between images, we could extract some part of temporal entropy and reduce overall amount of information transmitted. The information gain would be equal to:

1 - (5/7.4) * 100% = 32,4%.

Information amount would be cut at around **1/3**.

For code see "q2_b.py"

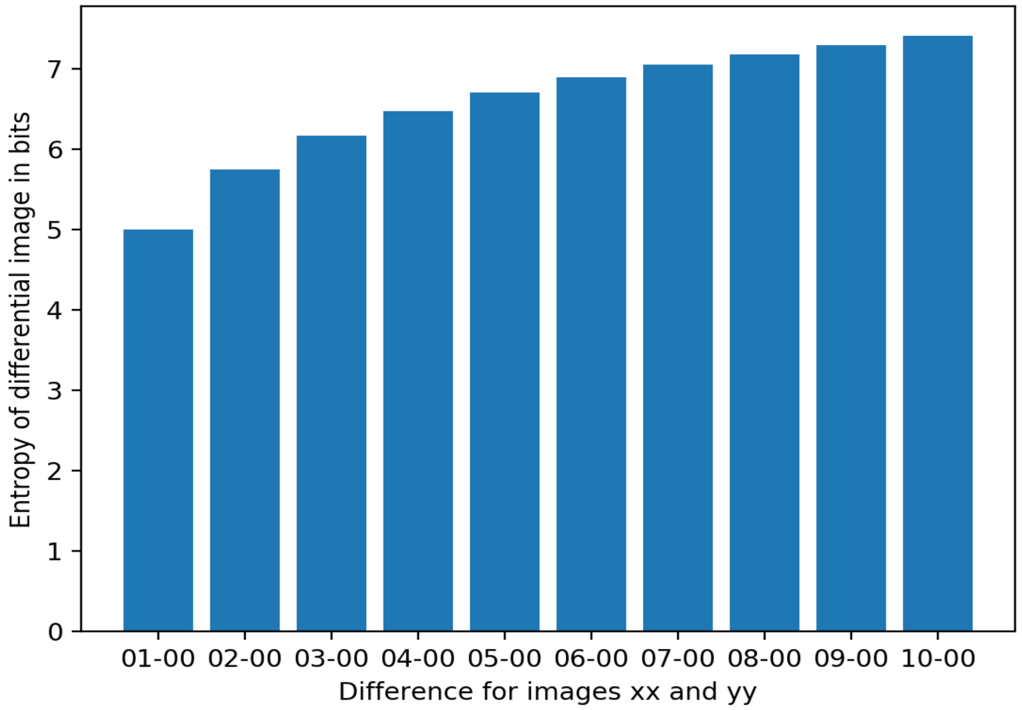
2(c) Compute the entropies of the difference of all images to the first one and visualize your results:

Solution 2(c):

The entropies of the difference of all images to the first one

Image difference	**Entropy in bits**
01-00	4.998878662055725
02-00	5.742178914597863
03-00	6.168458800585598
04-00	6.467441282740993
05-00	6.700940461626027
06-00	6.890982913050517
07-00	7.046384106476886
08-00	7.177977253387062
09-00	7.2952676009239905
10-00	7.40214935561599

We can see that difference is increasing with frame series. Neighbor frames are often very similar, but this difference grows as we move further away from the original frame:



The plot shows, that even with only a little difference between images, it’s useless to send difference only with the first frame. It’s effective in conserving data only up to 3 frames ahead, until the difference exceeds 6 bits.
After that, difference would be so diverse that it takes almost the size of the whole 1 frame, i.e. 6,5 to 7,4 bits. After 10 frames it’s a completely different image.

Comparing two entropy extraction methods presented by (b) and (c), we can make a conclusion that:

> **1)** Subsequent images' difference in (b) is very effective in terms of data overhead reduction, i.e. it proposes a better compression capability.
>
> **2)** Images' difference to only the first one in (c) is more robust in terms of data loss, i.e. we can lose couple of frames without any quality drop.

Noticeably, that in both cases this case we can preserve the original image quality!
As a result, there is a trade-off between a transmission data rate and a live image quality.
In this way, the first method is better suited for a video file encoding, where all data is transmitted over reliable TCP connection.
The second method is better suited for a live video streaming, where a robustness of a real time video transmission is important.

For code see "q2_c.py"

2(d) implement a function called *EstMotion*. For all macro blocks in the next image (00005901.png), it finds the best hit in the current image (00005900.png). The best hit is the minimum value of a cost function. Your function returns for each macro block the distance to the best hit as (x,y)-vector. Consider the following two cost functions:

(h) CF1: Mean difference
(i) CF2: Absolute mean difference

Solution 2(d):

You can find a source code of *EstMotion* function written in Python3 programming language below.

As argument **it takes**:

- Numpy array representation of the next image and current image

- Cost function defining the best hit estimation

- Dimension of macro block in pixels which is by default 16.

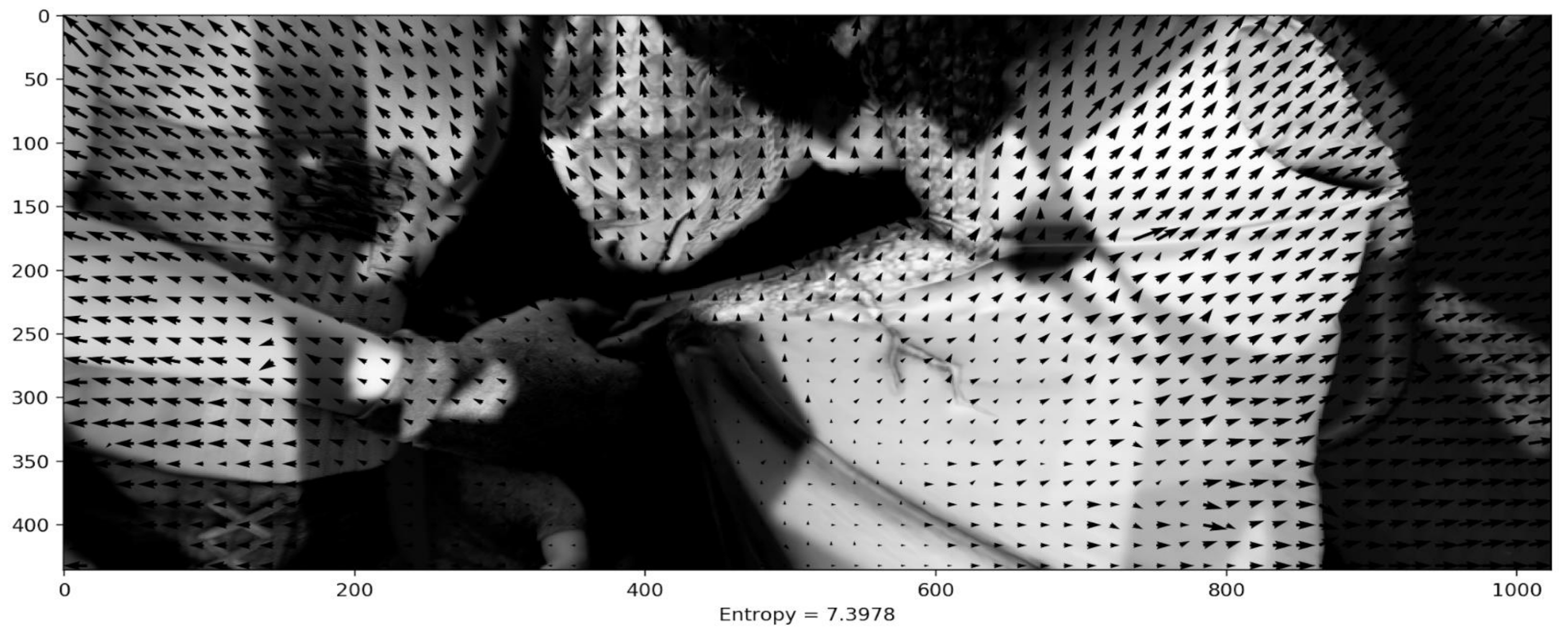
The macro block is a size of search area in pixels around which in the current image should be performed a search for a best hit. Wikipedia [https://en.wikipedia.org/wiki/Block-matching_algorithm] claims that macroblock size of 16 pixels and a search area of p = 7 represent a typical parameter input for a block matching algorithm.

It returns a tuple of three arrays. First is the array containing distance vectors, (x, y) tuples, for the best hit for each macroblock as it is stated in the task. The second array contains starting positions of these destination vectors, also (x, y) tuples. The third array is the “predicted” picture based on the motion estimations, i.e. the “predicted” next picture constructed out of blocks of current picture.

```
import numpy as np


def CF1(diff):
    return np.mean(diff)

def CF2(diff):
    return np.mean(np.abs(diff))

def EstMotion(next, current, cost_func, step = 16, p = 7):
    iLen, jLen = next.shape
```

```
prediction = np.zeros(next.shape, dtype = np.int32)
locations = []
distances = []

for i in range(0, iLen, step):
    for j in range(0, jLen, step):
        next_block = next[i:i+step, j:j+step]
        # search area
        sa = (i-p,i+p), (j-p,j+p)
        min_cost = np.inf
        best_hit = None
        l = [i,j]
        d = [0,0]

        for k in range(sa[0][0], sa[0][1]):
            for m in range(sa[1][0], sa[1][1]):
                search_block = current[k:k+step, m:m+step]

                if search_block is None or search_block.shape != next_block.shape:
                    continue
                diff = next_block - search_block
                t = cost_func(diff)
                if t < min_cost:
                    min_cost = t
                    best_hit = search_block
                    d = [k-i, j-m]

        locations.append(l)
        distances.append(d)
        prediction[i:i+step, j:j+step] = best_hit

return distances, locations, prediction
```

On the picture below we can see the result picture with motion vectors layered on it with *axes.quiver* matplotlib function. Vectors do not represent an exact block pixel-to-pixel movement. They are scaled based on average vector length for a better visual representation. However, this option can be changed to a pixel-to-pixel movement in source code of q2_d.py file.



For code see "q2_d.py"

2(e) Compare CF1 and CF2 for blocks of a size of 16x16. For which of the cost functions is the prediction error, i.e. the entropy between predicted image and actual image, larger?

Solution 2(e)

We calculate a prediction error for search area of size 7. It offers quite good results on a given input and does not cause any artifacts, which could affect the entropy measurement.
Results are presented below.

Prediction error for CF1: 5.957368005027437
Prediction error for CF2: 2.5136462876560755

This result comes from the fact, that on a very contrast search blocks we can receive that the average of -100 and 100 is smaller than 0 and 1.

For code see "q2_e.py"

2(f) Vary the block size and compare the efforts to transmit the motion blocks and compare the entropy of the error pattern. Which block size minimizes the data to be transmitted?

Solution 2(f)

We used a search area of size 5, and compared block sizes of 4, 8, 12, 16, 24 and 32.
Results follow.

<u>Block size: 4</u>
Prediction error CF2: 2.2999040447211168
Data units to be transmitted: 168,432.45428096925 (in vectors, 1 vector can be encoded in 8 bits)

Block size: 8
Prediction error CF2: 2.488793214937137
Data units to be transmitted: 40,989.0836694677 (in vectors, 1 vector can be encoded in 8 bits)

Block size: 12
Prediction error CF2: 2.5586089987113647
Data units to be transmitted: 18,333.69938635774 (in vectors, 1 vector can be encoded in 8 bits)

Block size: 16
Prediction error CF2: 2.5901765631562492
Data units to be transmitted: 10,256.973109654582 (in vectors, 1 vector can be encoded in 8 bits)

Block size: 24
Prediction error CF2: 2.661312387735342
Data units to be transmitted: 4,643.694180898848 (in vectors, 1 vector can be encoded in 8 bits)

Block size: 32
Prediction error CF2: 2.6815074705549606
Data units to be transmitted: 2,505.816543865725 (in vectors, 1 vector can be encoded in 8 bits)

We refer to a data unit, as the amount of information transmitted differs, depending on the amount of bits used to encode the distance vector.
For example, 4 bits can be used to encode one value for the search area of size 7. One bit represents the positive or negative sign of the value, the other 3 represent the number from 0 to 7. Altogether it takes 8 bit, as there are two vector values, per 1 macroblock.
The data units' size can be multiplied by 8 to receive an exact amount of information to be transmitted.

We can see that the biggest block size minimizes the amount of data that should be transmitted, while still preserving a relatively good prediction error compared to the most granular block size of 4.

For code see "q2_f.py"

Solution 3:

(a)

For the encoding of the videos a bash script was written. See code/q3_convert_video.sh

(b) Rate each video on scale of 1-5:

	crf 18	**crf 24**	**crf 30**	**crf 36**	**crf 42**	**crf 48**
Daniil	5	4	4	3	2	1
Marius	4	4	4	2	1	1
Sivan	5	5	3	2	1	1

(c)

The satisfaction with the mp4 video resulting from the encoding with crf value 48 was low across the whole group.

You could observe major flaws and missing details. Flaws here mean that the original imagery was so degraded that the actual objects were no longer visible, i.e. the leaf floating on the river or the bunny sleeping in the cave under the tree.

Additionally, all images seemed extremely pixelated and the video appeared slowed and laggy down due to multiple artifacts of older frames being visible for an extended amount of time.
Moving macroblocks are made visible and worsen the 3d experience. Video resembles more a kaleidoscopic picture, rather than a meaningful frame series.

In details, our observations were:

(a) In the beginning the trees draw lines on the zoom down.

(b) The background is unclear and smudged.

(c) You cannot see the river bed.

(d) The Pupils of the bird are not properly visible.

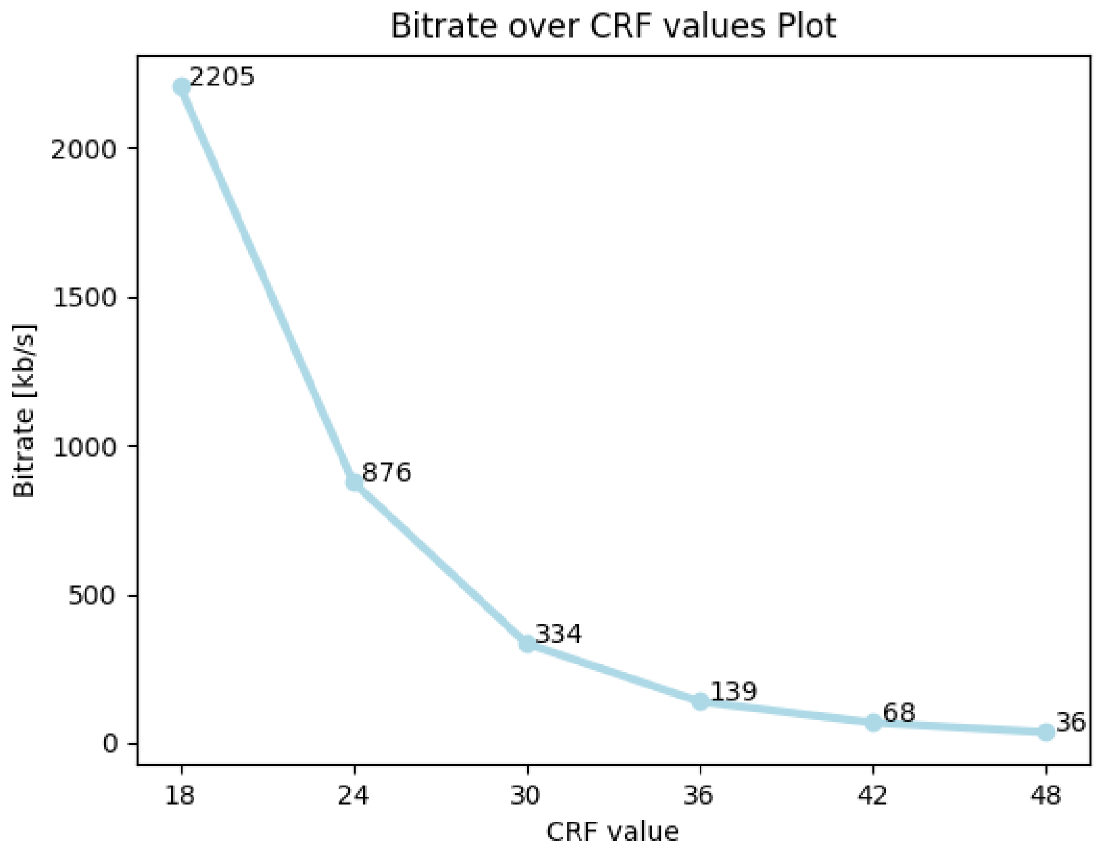
(e) The text “THE PEACH OPEN MOVIE PROJECT PRESENTS” is barely readable.

(f) The caption at the end seems to pop up bit by bit.

(g) Throughout the video the grass looks more like moss.

(d)

Crf	18	24	30	36	42	48
bitrate	2205 kb/s	876 kb/s	334 kb/s	139 kb/s	68 kb/s	36 kb/s



<u>What can we see:</u>

We can see that higher CRF has a lower bitrate. This happens due to CRF defining a quantization factor by which all the frames are quantified. The higher the quantization factor is, the less bits we need for encoding each frame, and so the bitrate decreases.

We also noticed that CRF with the value 30 presents a great trade-off between video quality and bitrate reduction. While members of our group were satisfied with video experience by this value, its size was reduced by more than 5 times.

Solution 4:

(a) For the encoding of the videos a bash script was written. See code/q4_a.sh
Average bitrates are: 2205 kbps, 876 kbps, 334 kbps 139 kbps, 68 kbps, 36 kbps.

(b)

```
#options: 704x480 fps=24/1 timebase=1/24 bitdepth=8 cabac=1 ref=1 deblock=1:0:0
analyse=0x1:0 me=dia subme=2 psy=1 psy_rd=1.00:0.00 mixed_ref=0 me_range=16
chroma_me=1 trellis=0 8x8dct=0 cqm=0 deadzone=21,11 fast_pskip=1 chroma_qp_offset=0
threads=1 lookahead_threads=1 sliced_threads=0 nr=0 decimate=1 interlaced=0
bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1
weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=24 scenecut=40 intra_refresh=0
rc_lookahead=40 rc=abr mbtree=1 bitrate=2204 ratetol=1.0 qcomp=0.60 qpmin=0 qpmax=69
qpstep=4 ip_ratio=1.40 aq=1:1.00

in:0 out:0 type:I dur:2 cpbdur:2 q:22.08 aq:4.00 tex:252 mv:487 misc:101 imb:1320 pmb:0
smb:0 d:- ref:;

in:4 out:1 type:P dur:2 cpbdur:2 q:20.99 aq:9.52 tex:77 mv:80 misc:267 imb:11 pmb:0
smb:1309 d:- ref:0 ;

in:2 out:2 type:B dur:2 cpbdur:2 q:20.99 aq:5.00 tex:0 mv:0 misc:152 imb:0 pmb:0 smb:1320
d:- ref:0 ;

in:1 out:3 type:b dur:2 cpbdur:2 q:20.99 aq:6.00 tex:0 mv:0 misc:152 imb:0 pmb:0 smb:1320
d:- ref:0 ;

in:3 out:4 type:b dur:2 cpbdur:2 q:20.99 aq:6.00 tex:0 mv:0 misc:152 imb:0 pmb:0 smb:1320
d:- ref:0 ;

in:5 out:5 type:P dur:2 cpbdur:2 q:12.99 aq:3.25 tex:209699 mv:3967 misc:398 imb:961
pmb:135 smb:224 d:- ref:0 w:5,99,-32 ;
```

```
in:6 out:6 type:i dur:2 cpbdur:2 q:17.16 aq:6.14 tex:46279 mv:3468 misc:141 imb:1320 pmb:0
smb:0 d:-

[...]
```

(a) resolution 704x480

(b) fps 24/1 - frames per second

(c) timebase 1/24 - how long a frame is shown

(d) bitdepth 8 - color per channel

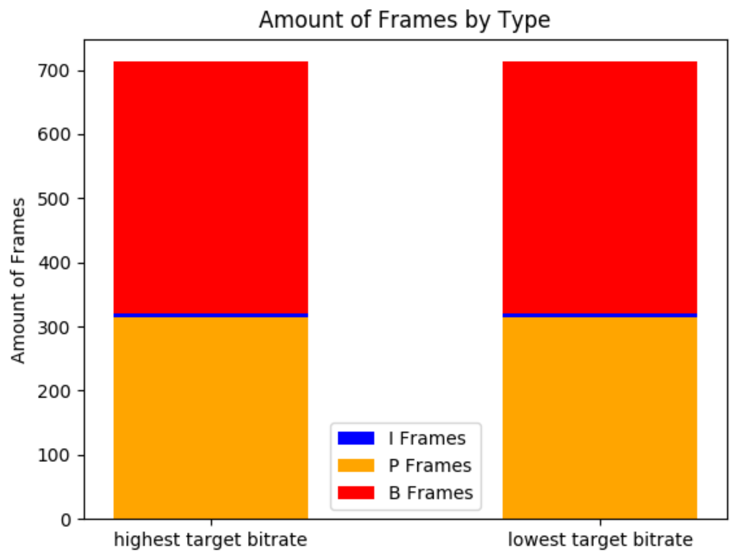
(e) threads 1 – CPUs used

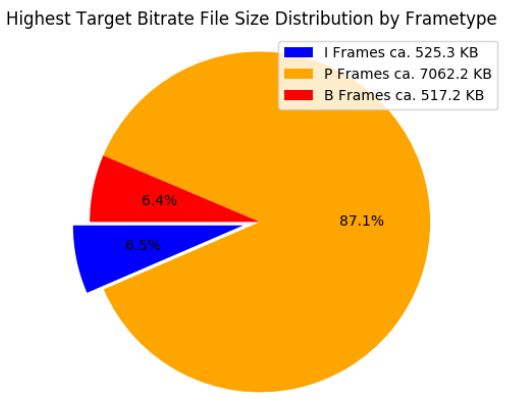
(f) there seems to be different I/i and B/b type frames

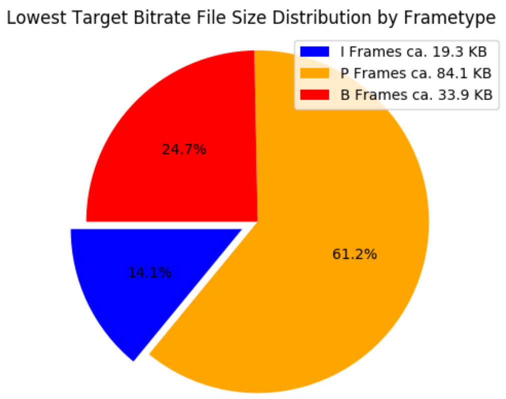
(ii) Interesting about I-Frames: they always have the same in and out value. So their corresponding frame in the raw video is the same number the frame is encoded at.

(c)

Below you can find a visualization of the fraction of I-frames, P-frames, and B-frames on the total number of frames for both our video with the highest and lowest target bitrate (2205 kbit/second and 36 kbit/second). The total amount of frames is 720.







We observed that the distribution is not changing between the lowest and highest target bitrate. We therefore conclude that the bitrate has no influence on the frame type distribution, i.e. of the I, P and B frames.

In the two pie charts you can see the distribution of the different frame types on the file size. Here we observed large shifts in the distribution.

With the bitrate increase the contribution of P, I and B frames has also changed. I-frames contribute in size now less than before, as well as B-frames. For the latter the contribution has decreased drastically. P-frames now also contribute almost 90% of the file size, instead of 60% as for a lower bitrate.

solution 5

5(a) Investigate and visualize the average quality of the six videos, using the structural similarity (SSIM) and PSNR. For this task, you can use ffmpeg built-in functionalities.

Solution5(a):

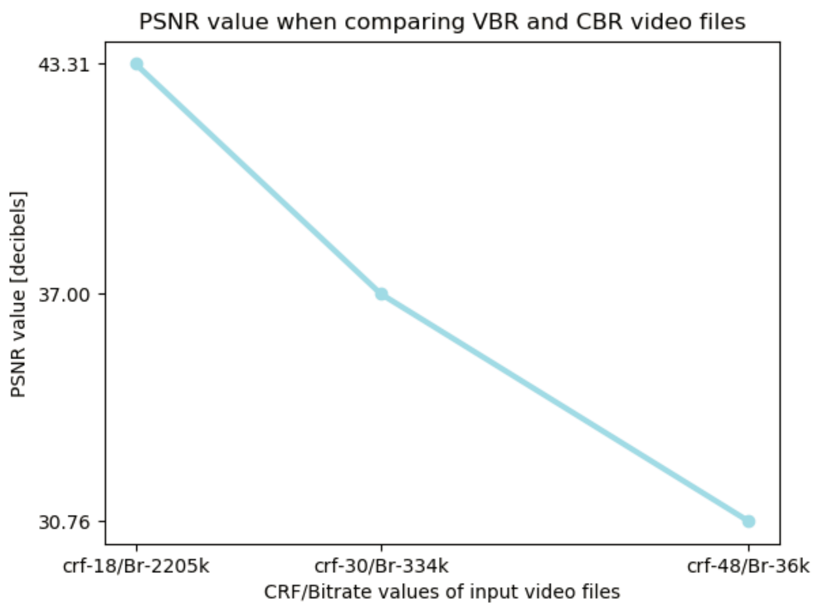
The commands for getting the PSNR and the SSIM respectively, for each 2 corresponding videos:

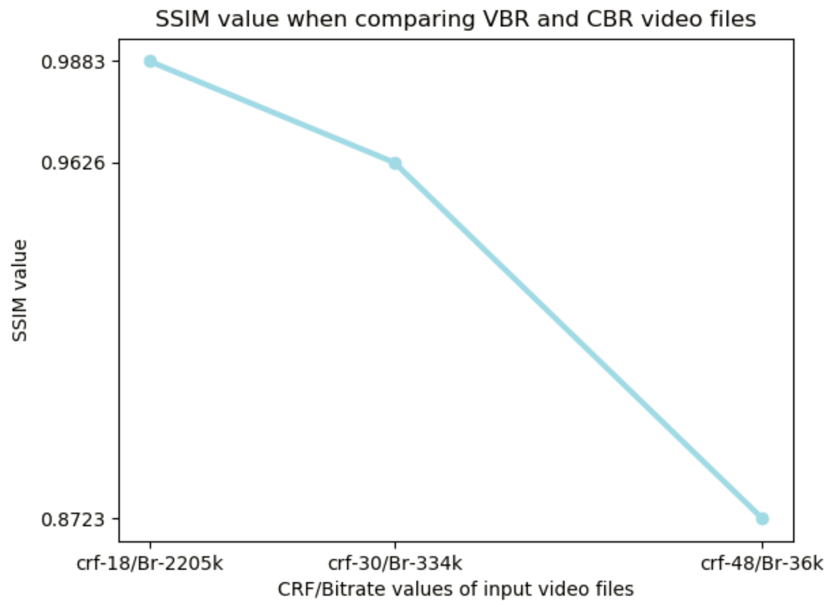
Note: the commands were executed via NodeJS wrapper for FFMpeg (fluent-ffmpeg)

```
ffmpeg -threads 1 -progress", "pipe:1" -i original_video –i reference_video -filter_complex "psnr" -f null –
```

```
ffmpeg -threads 1 -progress", "pipe:1" -i original_video –i reference_video -filter_complex "ssim" -f null –
```

video original	video reference	PSNR	SSIM
VBR - CRF = 18	CBR – Br = 2205k	43.310996	0.988287 (19.313301)
VBR - CRF = 30	CBR – Br = 334k	36.998585	0.962550 (14.265522)
VBR - CRF = 48	CBR – Br = 36k	30.755535	0.872278 (8.937359)





We can see that both matrices predict low perceived quality difference between the better quality videos (crf-18 and CBR 2205k) but as the quality decrees, the predicted perceived different increased, until it is very significant for the lowest quality videos (crf-48 and CBR 36k)

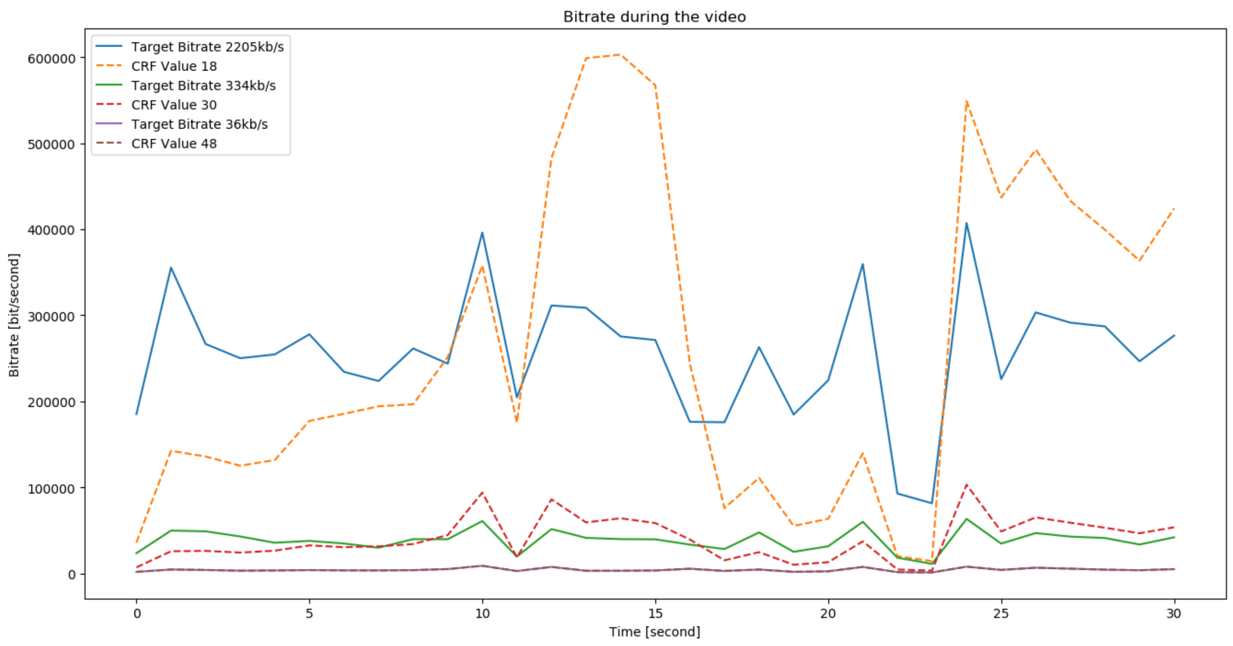
A possible explanation could be the fact that the VBR encoded video can ‘spend’ the available bits as he wish, and so to dedicate more bits for complex scenes, and less for simple scenes, while the CBR encoded video has to ‘spend’ the same amount of bits on both, leading to significant perceived quality difference of the complex scenes. However, if that was the case, we would except to see a clear quality difference between the videos, in favor of the VBR video, which we could not observe. Since the CBR is not a ‘real’ CBR, it is hard to be sure.

5(b) Visualize the bitrate for each video on a per-second scale

Command for getting the video bitrate per frame:

```
ffprobe -unit -select_streams v -show_entries packet=size,duration_time:stream=duration -of compact video-file
```

After extracting the bitrate per frame using FFProbe, the bitrate per second is calculated by adding together the bits streamed over each 24 frames.



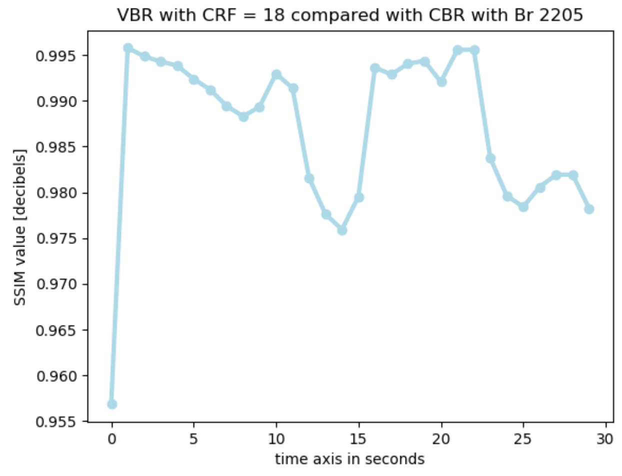
Note: as can be seen, the bit rate of our CBR videos is less varied than the VBR videos, but it isn't completlly constant. we tried several ways, but were unable to achive "real" CBR for MP4 using FFMpeg.
Side note: the flag *"nal-hrd=cbr"* does not work with mp4 output.

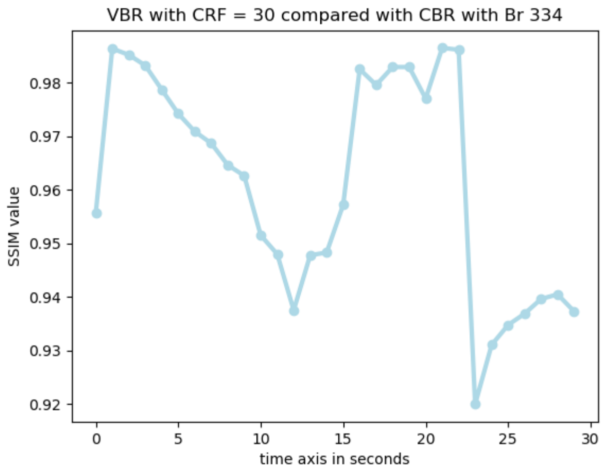
5(c) Visualize the SSIM for each video on a per-second scale

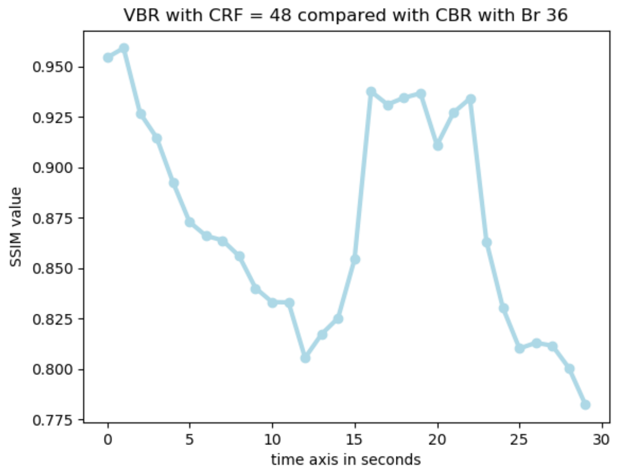
Command for getting ssim for each 2 frames into a file:

```
ffmpeg -i input-file -i input-file -lavfi ssim="stats_file=ssim.log" -f null –
```

After extracting the ssim per frame, the average ssim per second is calculated by calculating tha average over each 24 frames.





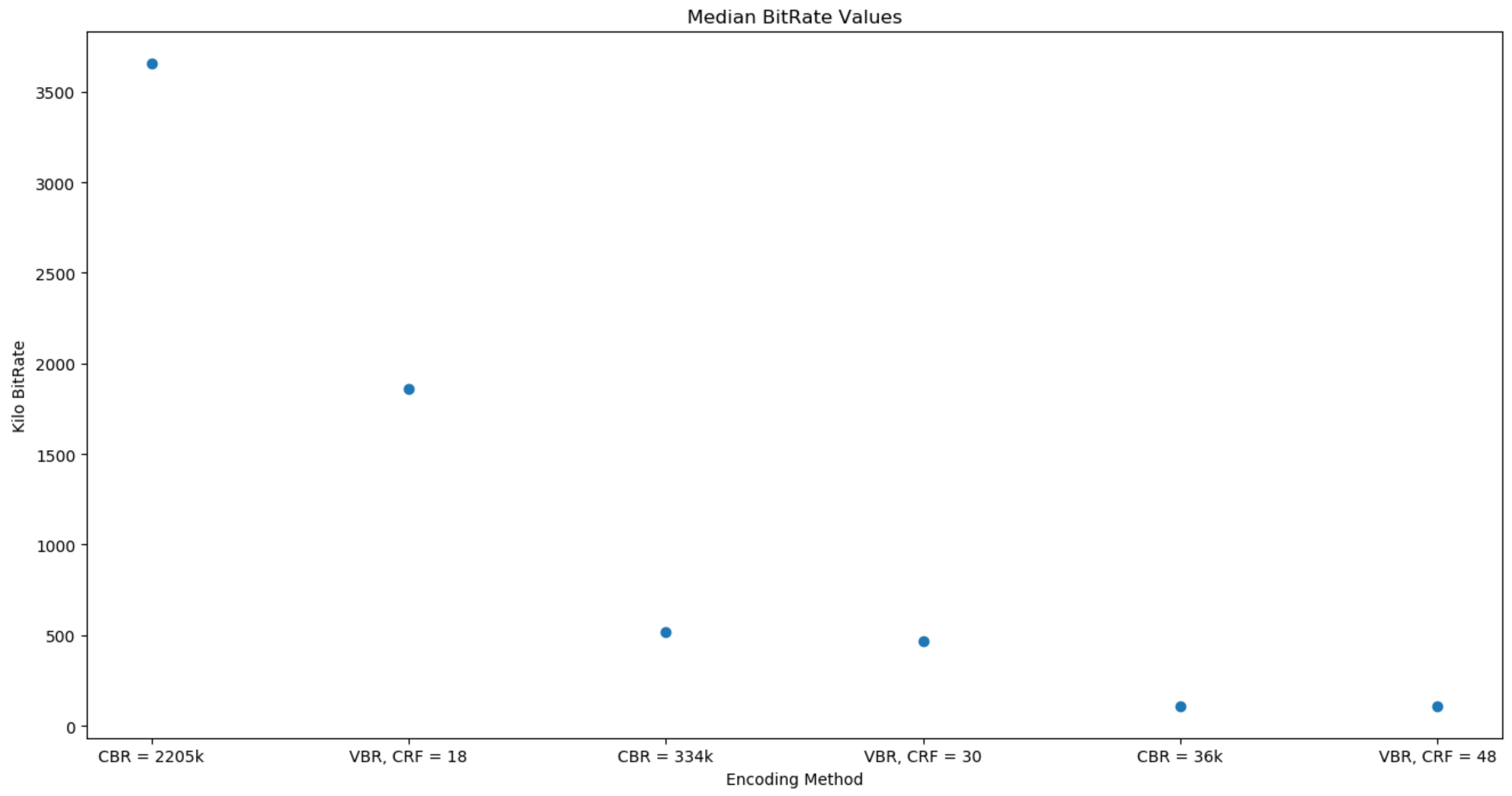


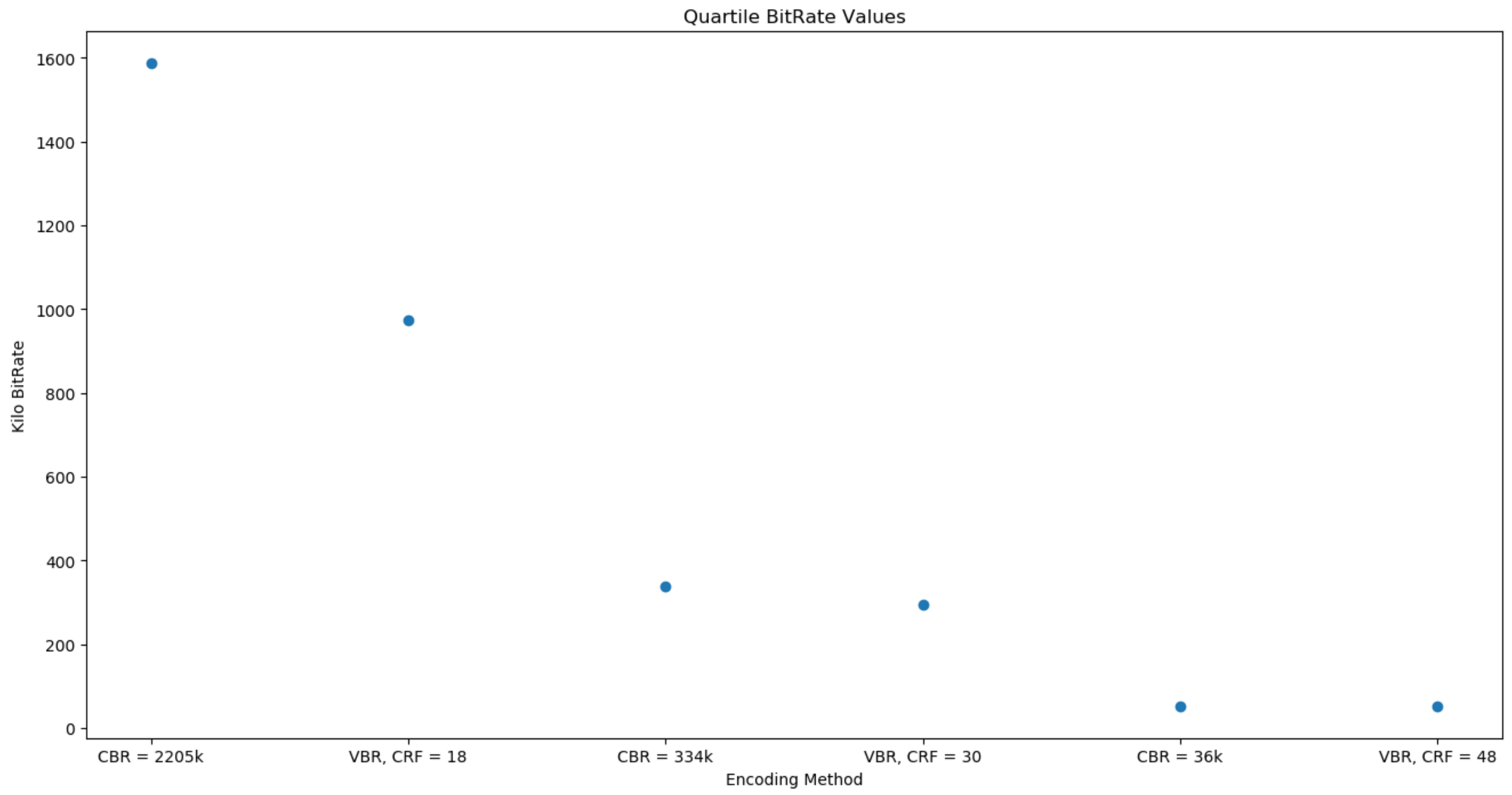
5(d) Compare CBR and VBR based on the results from subtask (b) and (c). Additionally, use box plots to visualize median and quartiles of SSIM and bitrate. Interpret your results!

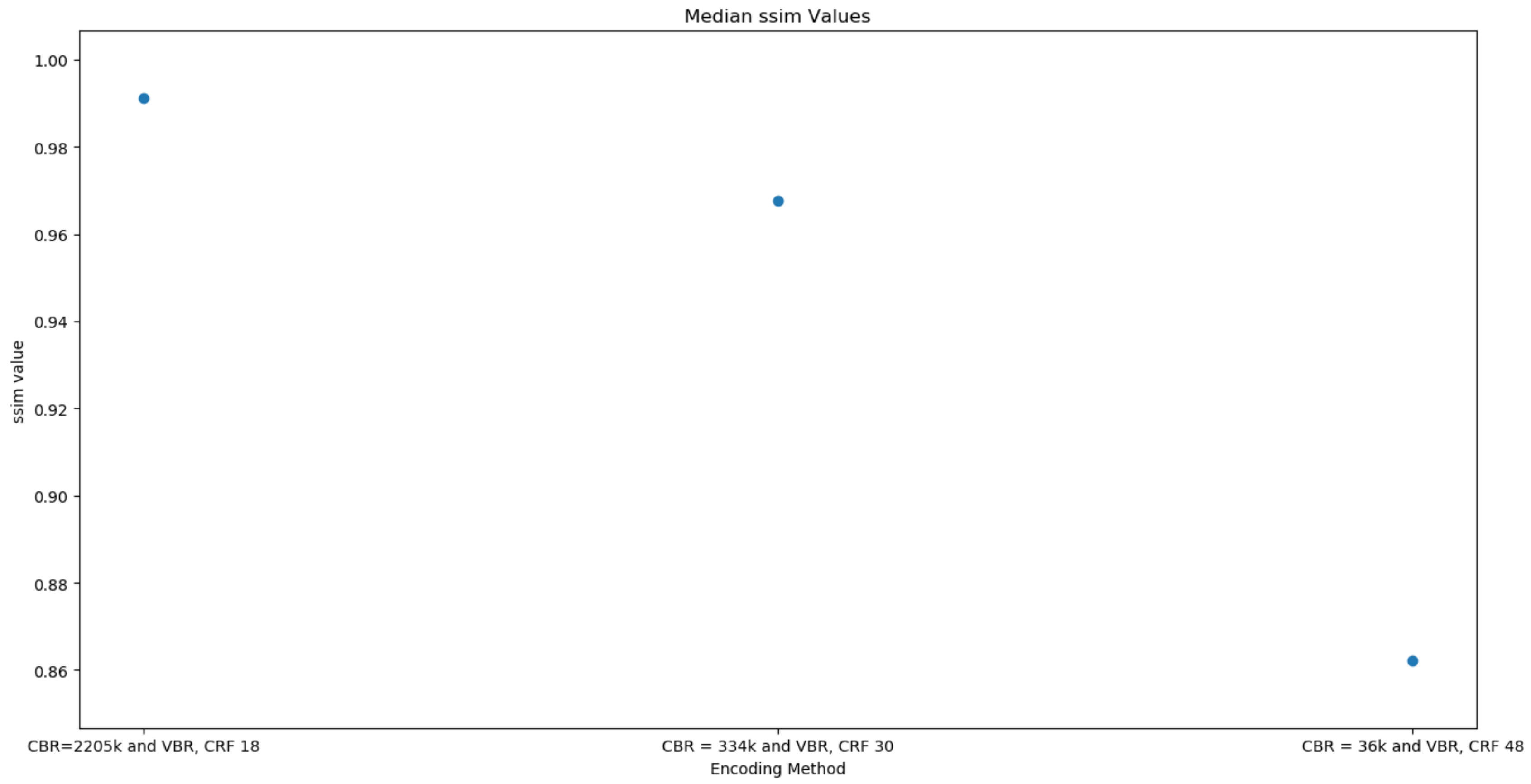
Compression:

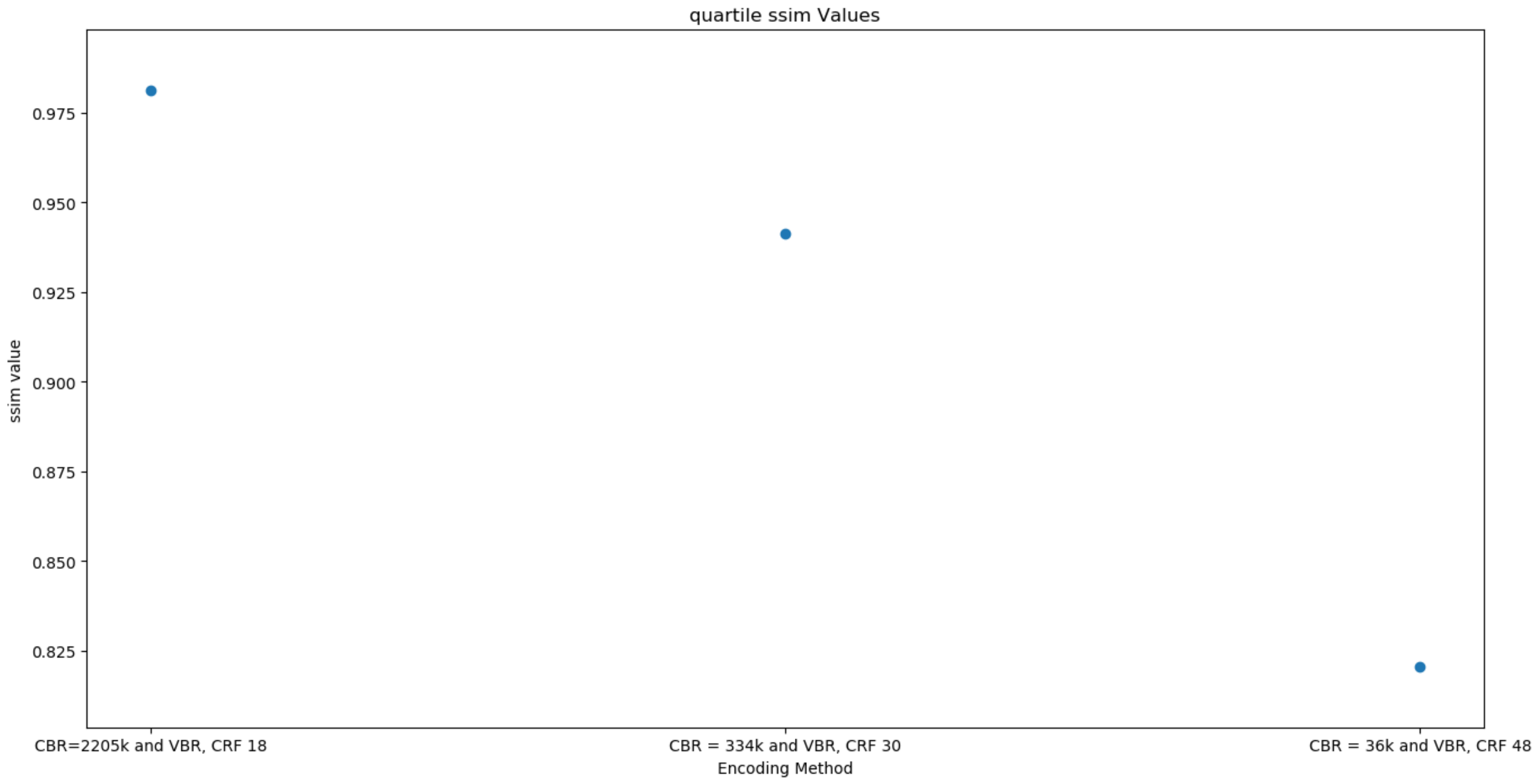
From the **bitrate compression**, we can see that VBR 'spends' more bits to encode certain scenes – the more complex ones, and preserves bits when it does not required – simple scenes.

From the **ssim compression** we see that the highest predicted perceived visual difference is predicted for those same complex scenes, where there is a different in the amount of bits used, as expected.


Median BitRate Values
Kilo BitRate
0
500
1000
1500
2000
2500
3000
3500
CBR = 2205k
VBR, CRF = 18
CBR = 334k
VBR, CRF = 30
CBR = 36k
VBR, CRF = 48
Encoding Method


Quartile BitRate Values
Kilo BitRate
1600
1400
1200
1000
800
600
400
200
0
CBR = 2205k
VBR, CRF = 18
CBR = 334k
VBR, CRF = 30
CBR = 36k
VBR, CRF = 48
Encoding Method


Median ssim Values
ssim value
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
CBR=2205k and VBR, CRF 18
CBR = 334k and VBR, CRF 30
CBR = 36k and VBR, CRF 48
Encoding Method



<u>Interepting the results:</u>

Bitrates: the median and quartile bitrates values are higher for the CBR encoding in the best quality video, the different is getting less significant in te middle quality, and not percived for the worst quality.

It can be explained by the VBR encoding conserving a lot of bits on simple scenes, hence having a lot of low range bitrates, and few high range bitrates, hence causing the median to be relatively low.

but as the quality is getting worst, the number of bits used per scene become closer: the amount of bits the CBR **can** use are closer to the amount VBR need to use, and the different becomes less and less.

same explanation go also to the quartile bitrate.

SSIM:

as the predicted quality degrades, there are more low values of ssim, and the median and quartile are droping.

5(e) you are the team leader of Netflix's video compression team and responsible for encoding videos. Netflix delivers its content on two ways: rental of DVDs, which are delivered via post, and providing the content on their servers, so that clients can stream the videos via the Internet.
How would you encode videos...
• for the DVD and
• for the online streaming service?
Explain your choices!

Solution 5(e):

Factors to take into consideration:

(a) DVDs has limited size

(b) Servers theoretically has (effective) unlimited capacity, but still we would like to preserve space, exhaustion and energy.

(c) The display quality of the end clients is varied due to factors like screen resolution and network bandwidth and QoS (quality of service) required.

(d) portability

Other factors: securing from unauthorized copying, availability of content in case of the server going down, caching.

Discussion:

(e) For both **DVDs and streaming**:

(i) We would like to choose widely spread codec for both DVD and streaming, to ensure support on all (or most) devices. If needed, several copies can be made with different codecs, to ensure portability on the end user device.

(ii) Technologies that prevents DVDs unauthorized copying could be achieved regardless the video encoding method (e.g. by encryption) and Portability can be achieved by using popular codec, and does not affected by the encoding method. Hence, we will focus on achieving maximum quality.

(f) In addition, for **DVDs:** our main concern will be to achieve maximum quality in the space we have, make sure he content will play on all (or at least most) devices, and to prevent unauthorized copying if possible.

In order to achieve the best quality possible, we would like to use our limited bits wisely: more bits for complex scenes, and less for simple once. Using CBR will defeat this purpose – all type of scenes will be encoded with the same bitrate. Hence, we rather use **VBR**, with the lowest CRF that can still be fitted into the DVD.

(g) For **Streaming:** availability and caching does not related directly to the encoding method. The main consideration regarding the encoding method is again the quality for the end user.

We would like to enable our customer to stream the optimized video file to his clients, depends on their device display capability and band width.

For this purpose, we can utilize the use of both **VBR** or **CBR**, depends on the situation.

When the end user capabilities are virtually unlimited and the QoS demands it, we would like to stream video with the best possible quality we can make. Still, due to server space, exhaustion and energy, there is no point in just wasting bits on ‘simple’ scenes, hence we will use **VBR** with CRF=0 (lossless) or close, to get the best quality with the minimal required bits.

Other situation that could arrive is an end user that has a low bandwidth. Streaming a VBR encoded video might be not practical for complex scenes, since the high bitrate will make the streaming too slow. To circumvent this, we can encode the video in a suitable constant bitrate, in order to ensure smooth (though low quality) streaming.

The goal will be to create multiple video files that will allow us to stream in the best possible quality in the given situation. if the limits are low we will choose high quality VBR video, if the stream is very limited, we will resort to suiting CBR video, and in-between we'll have to try and find the best encoding for each situation.

Conclusion:

The main concern regarding encoding will be ensuring quality.

For DVDs, we would like to use **VBR**, to get best quality from limited bits. Other considerations seems secondary to the quality consideration, and using **CBR** doesn't make sense – we will 'waste' bits on scenes which does not required as much bits.

For Streaming, we would like to create multiply video files, in order to be able to stream the video that will provide the best quality in given conditions, taking required QoS, device display capabilities and bandwidth into consideration.
To achieve this, we will use both VBR and CBR encoding methods.

Source: based in part on the technological blog of Netflix [https://medium.com/netflix-techblog]