Open in Colab

Assignment 2

Authors: Vegard Iversen, Sondre Olimb, Sebastian Skogen Raa

Setup

In [ ]:
```
import numpy as np
import scipy.fftpack as fftpack
import matplotlib.pyplot as plt
import os
import cv2 as cv
from scipy import signal
import seaborn as sns

sns.set_theme()

sns.set_style("dark")
```

1 Temporal prediction

a)

Motion compensation is a method of predicting a frame in a video realtive to a referance frame. This is done by accounting for previous and future frames camera and/or object motion. This method of prediction between frames will in most cases give good compression efficiency.

b)

I- intra frames

These fraems ar coded without a referance to other fames. By coding the frames we achive a moderate amount of compression with spacial redundancy, however there is no temoiral redundancy. Since the frames are encoded with no refrance to other frames, there is no motion compensation. The compression can be both lossles and lossy. Relative to the other frames, it achvives the least amount of compression.[1[

P- Predictive frames

The predictive frames are encoded using motoion compesnastion from a previus I or P frame. It only stores the changes from previous I or P- frame, this migth be motion vectors or image data. P- frames reduces both the temporal and spatial redundancy and therfore achives a higer order of compression compared to I- frames. the higher order of compression will give a lower qaulity picture comapred to I-frames, but this i solved by using Group of Pictures (GOP). In a GOP the two ar combined to achive a higer quality image. (GOP also includes B-frames)[1]

B- Bidirectional

Bidirectional frames uses (as the name indicates) both past and future I- and P-frames. These are then used as a linear combination to estimate a sutable motionestimate. This Bidirectional encoding gives the higest order of compresion of the three frames. We can have forward, backward motion vectors. B-fames achives the lowest order of picture quality of the three frames, but as descirbed above, in a GOP high quality can be achived.[1]

c)

The transmission is ordered different from the display order in assosiation with the B-picture. The transmitted ordering is in done such that B-pictures appear after the past and future pictures. By doing this reordring it is ensured that the B-picture can always be constructed by referencing these past and future pictures from memory. This will of course introduce a delay dependant upon the number of consecutive B-pictures, which is 2 in mpeg 1 and 2 as shown in the figure below.

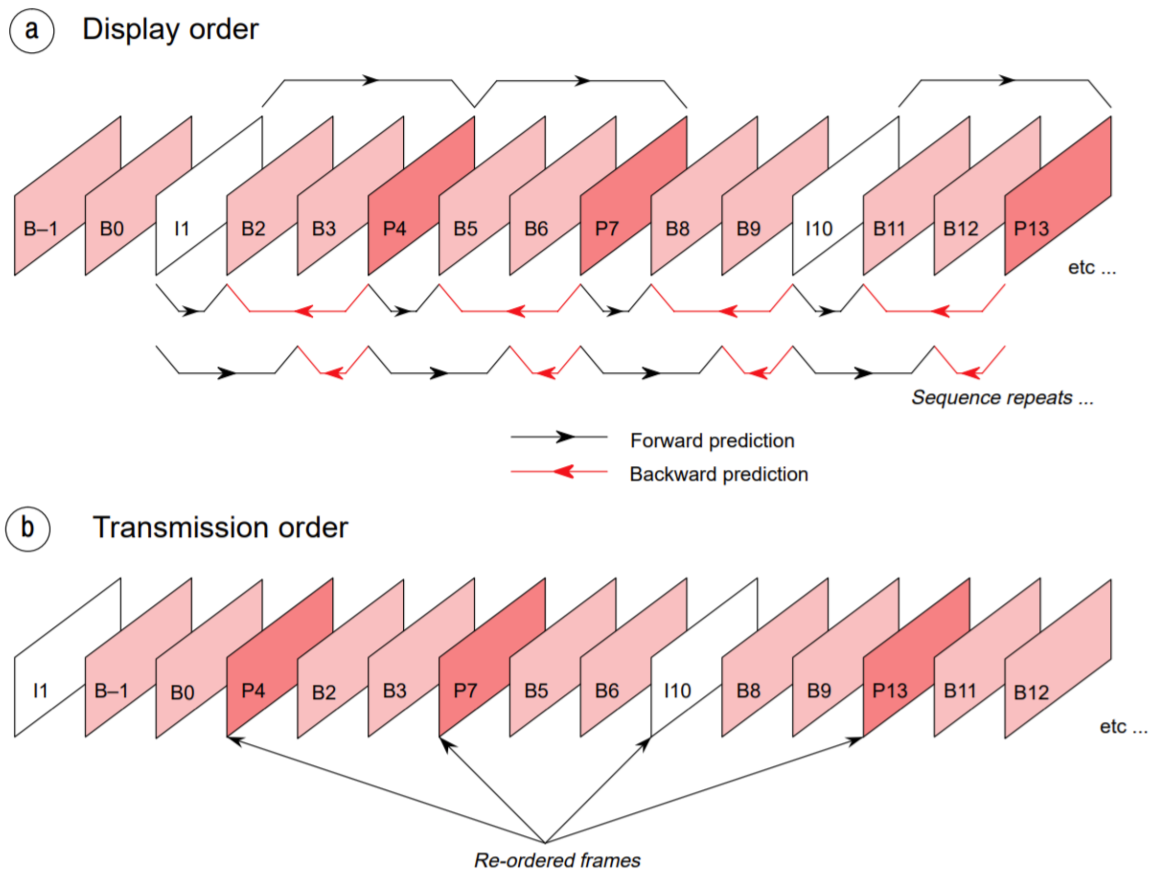


Figure of trasmission vs display order. [1]

d)

Main differences

- Variable block-size motion compensation with small block sizes
- Quarter-sample-accurate motion compensation
- Extention of the B pictures
- Multiple reference picture motion compensation

Others

- Motion vectors over picture boundaries
- Decoupling of referencing order from display order
- Decoupling of picture representation methods from picture referencing capability:
- Weighted prediction:
- Improved “skipped” and “direct” motion inference:
- Directional spatial prediction for intra coding
- In-the-loop deblocking filtering

2 Decomposition/Transform

a)

The transfrom that usually have been used in classic video coders are discret cosine transformers (8 × 8 for MPEG-1). For H.264 uses integer DCT (4 × 4).

b)

Block-transform-based video/still image coders usually exhibit so-called blocking artifacts. How is this effect minimized in H.264? (10 points)

In H.264 a deblocking filter is used to minimize the blocking noise from the DCT. In H.264 the deblocking filter is standard in difference to MPEG-1/2/4 where it was an optional additional feature in the decoder.

3 Video coding

We have that $\sigma_x = 1$

$\sigma_z^2 = \sigma_x^2(1-\rho)^2$

$\sigma_z^2 = 1 \cdot (1-0.9^2)$

$\sigma_z^2 = 0.19$

\ First for every second pixel

$\sigma_e^2 = E[(Z[n] - Z'[n])^2] = E[(Z[n] - a \times Z''[n+1])^2]$

To minimize we want to derivate wrt a and set equal to 0.\
$\frac{\delta}{\delta a}\sigma_e^2 = \frac{\delta}{\delta a}E[(Z[n] - a \times Z''[n+1])^2]$\ We can write out the expression. And use the information that $E[Z[n]] = 1$ and $E[Z[n]Z[n+k]] = 0.9^{|k|}$, and that we can approximate $Z[n] = Z'[n]$
$E[(Z[n] - a \times Z''[n+1])^2] = E[Z[n]^2 - 2 \times a \times Z[n]Z[n+1] + a^2 \times Z[n+1]^2] = 1 - 2 \times \epsilon$
\ $\frac{\delta}{\delta a}\sigma_e^2 = 0$ gives\ $\frac{\delta}{\delta a}(1 - 2 \times a \times 0.9 + a^2) = 0$\ $-2 \times 0.9 + 2a = 0$\ $a = 0.9$ \

$\sigma_e^2 = E[(Z[n] - 0.9 \times Z''[n+1])^2]$ \
$\sigma_e^2 = E[Z[n]^2 - 2 \times Z[n] \times 0.9 \times Z''[n+1] + 0.9^2 \times Z''[n+1]^2]$

$\sigma_e^2 = E[Z[n]^2] - 2 \times 0.9 \times E[Z[n] \times [Z''[n+1]] + 0.9^2 E[Z''[n+1]]$

if $Z[n] = Z'[n]$ then $Z''[n] = Z[n]$ $\sigma_e^2 = 1 - 2 \times 0.9 \times 0.9 + 0.9^2$ \ $\sigma_e^2 = 0.19$

$D_{even} = \frac{\pi e}{6} \times 0.19 \times 2^{-2R1}$\ Where R1 is the coded bits

For the rest of the pixels \
$\sigma_e^2 = E[(Z[n] - Z'[n])^2] = E[(Z[n] - b \times Z''[n-1] - c \times Z''[n+1])^2]$

minimizing as earlier. x is b or c.\ $\frac{\delta}{\delta x}\sigma_e^2 = \frac{\delta}{\delta x}E[(Z[n] - b \times Z''[n-1] - c \times Z''[n+1])^2]$\
$\frac{\delta}{\delta x}\sigma_e^2 = \frac{\delta}{\delta x}E[Z[n]^2 - 2 \times b \times Z[n]Z[n-1] - 2 \times cZ[n]Z[n+1] + b^2 Z[n-1]^2 + 2 \times b \times c \times$
\ Using that $Z[n-1]Z[n+1] = Z[n]Z[n+2]$\
$\frac{\delta}{\delta x}\sigma_e^2 = \frac{\delta}{\delta x}(1 - 2 \times b \times 0.9 - 2 \times c \times 0.9 + b^2 + 2 \times b \times c \times 0.9^2 + c^2)$\ derivate with respect to b gives\ $\frac{\delta}{\delta b}\sigma_e^2 = \frac{\delta}{\delta b}(1 - 2 \times b \times 0.9 - 2 \times c \times 0.9 + b^2 + 2 \times b \times c \times 0.9^2 + c^2)$\
$\frac{\delta}{\delta b}\sigma_e^2 = -2 \times 0.9 + 2b + 2 \times c \times 0.9^2$\ set equal 0\ $-2 \times 0.9 + 2b + 2 \times c \times 0.9^2 = 0$\
$b = -c \times 0.81 + 0.9$\ wrt c\
$\frac{\delta}{\delta c}\sigma_e^2 = \frac{\delta}{\delta c}(1 - 2 \times b \times 0.9 - 2 \times c \times 0.9 + b^2 + 2 \times b \times c \times 0.9^2 + c^2)$\
$\frac{\delta}{\delta c}\sigma_e^2 = -2 \times 0.9 + 2 \times b \times 0.9^2 + 2c$\ set equal 0 \

$2 \times 0.9 + 2b + 2 \times b \times 0.9^2 = 0$\ $c = -b \times 0.81 - 0.9$\ this gives c=b\
$-2 \times 0.9 + 2 \times b \times 0.9^2 + 2b = 0$\ $b = \frac{0.9}{0.81+1} = 0.4972 = c$ \

putting this into σ_e^2\
$\sigma_e^2 = 1 - 2 \times 0.4972 \times 0.9 - 2 \times 0.4972 \times 0.9 + 0.4972^2 + 2 \times 0.4972 \times 0.4972 \times 0.9^2 + 0.4$
\ $D_{odd} = \frac{\pi e}{6} \times \sigma_e^2 \times 2^{-2R2}$\ Where we can assume that $Z'[n] = Z[n]$\
$D_{odd} = \frac{\pi e}{6} \times 0.10497 \times 2^{-2R2}$\ Where R1 is the coded bits

Average distortion \ $D = \frac{\pi e}{6} \times (0.19 \times 2^{-2R1} + 0.10497 \times 2^{-2R2}$\ $R = \frac{1}{2}(R1 + R2)$

$R2 = R1 - \frac{1}{2}log_2\frac{0.19}{0.10497}$\ $D = \frac{\pi e}{6} \times (0.19 \times 2^{-2R1} + 0.10497 \times 2^{-2R2})$\
$D = \frac{\pi e}{6} \times (\sqrt{0.19 \times 0.10497} \times 2^{-2R})$\ $= \frac{\pi e}{6} \times 0.1412 \times 2^{-2R}$\ from the hint:

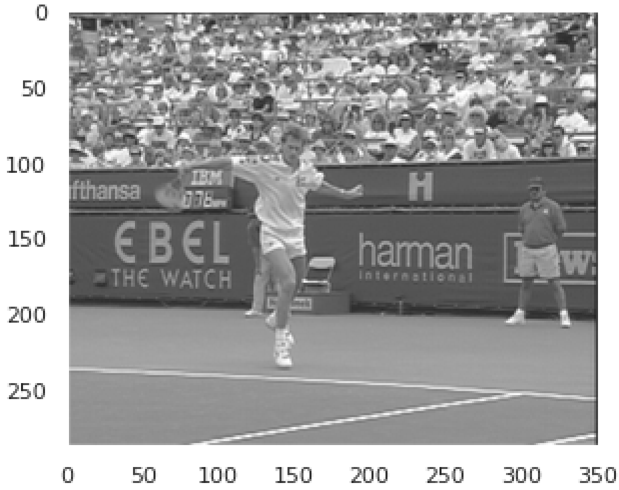
$D_{even} = \frac{\pi e}{6} \times \sigma_e^2 \times 2^{-2R1}$\ Where we can assume that $Z'[n] = Z[n]$

4 Motion estimation

a)

In [ ]:

```
im25 = cv.imread('frame_25.png', cv.IMREAD_GRAYSCALE).astype(np.int16)
im26 = cv.imread('frame_26.png', cv.IMREAD_GRAYSCALE).astype(np.int16)
im27 = cv.imread('frame_27.png', cv.IMREAD_GRAYSCALE).astype(np.int16)

plt.imshow(im25, cmap='gray')
plt.show()
```



In [ ]:

```python
im25
```

Out[ ]:

```
array([[ 33,  33,  34, ...,  32,  32,  34],
       [ 33,  33,  32, ...,  34,  34,  34],
       [ 26,  27,  24, ...,  41,  35,  37],
       ...,
       [144, 143, 144, ...,  66,  29,  37],
       [143, 142, 144, ...,  63,  28,  37],
       [143, 144, 145, ...,  60,  28,  36]], dtype=int16)
```

In [ ]:

```python
def reshape_split(image: np.ndarray, kernel_size: tuple):

    img_height, img_width = image.shape
    try:
      tile_height, tile_width, channels = kernel_size
    except:
      tile_height, tile_width = kernel_size
      channels = 1
    if channels == 1:
      tiled_array = image.reshape(img_height // tile_height,
                                  tile_height,
                                  img_width // tile_width,
                                  tile_width
                                  )
    else:
      tiled_array = image.reshape(img_height // tile_height,
                                  tile_height,
                                  img_width // tile_width,
                                  tile_width
                                  ,channels)
    tiled_array = tiled_array.swapaxes(1, 2)
    return tiled_array
```

In [ ]:

```python
images = [im25, im26, im27]
```

In [ ]:

```python
def cost_function(x1, x2):
  diff = x1 - x2
  # print(type(diff))
  return np.mean(np.square(diff), axis=(-1, -2))
```

In [ ]:

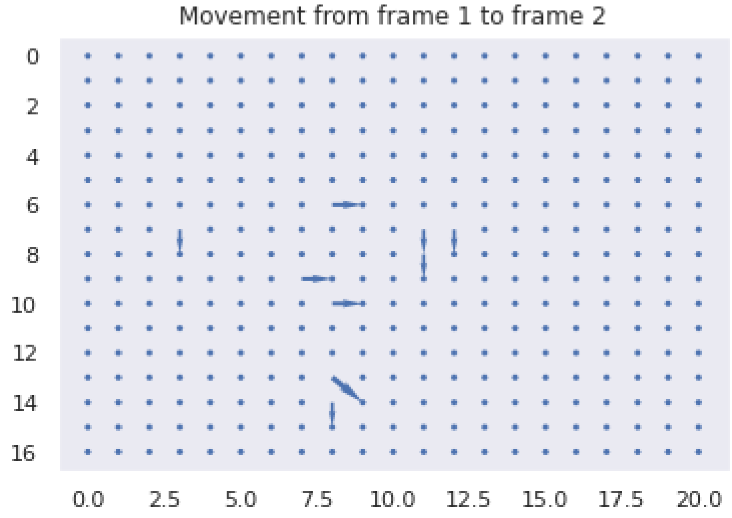
```python
from numpy.lib.index_tricks import AxisConcatenator
def gen_disp_vectors(images, block_size, search_area_size, plot_as_img=False):
```

```
    """
    Input:
    images: list of two dimentional images
    block_size: size of macro blocks used
    search_area_size: number of pixels to search for match in horizontal and vertical

    Output:
    disp_vectors: list of vectors corresponding to frames N-1 to N
    """

    search_area_subdivs = search_area_size//block_size
    print("Num of macroblocks in search area:", search_area_subdivs)
    offset_lim = search_area_subdivs-1 # makes sure search area does not go out of bou
    disp_vectors = [] # list of finnal displacement vectors
    for idx_im in range(1, len(images)):
      prev_image = images[idx_im-1]
      current_image = images[idx_im]

      prev_macro_blocks = reshape_split(prev_image, (block_size, block_size))
      current_macro_blocks = reshape_split(current_image, (block_size, block_size))
      block_shape = prev_macro_blocks.shape

      vector_shape = (block_shape[0]-offset_lim, block_shape[1]-offset_lim)
      vector_macro_blocks = np.full((vector_shape + (3,)), 0) # 3dr dim added to be ea

      for macro_block_vidx in range(block_shape[0]-offset_lim):
        for macro_block_hidx in range(block_shape[1]-offset_lim):

          search_results = np.ones((search_area_subdivs, search_area_subdivs))

          prev_macro_block = prev_macro_blocks[macro_block_vidx, macro_block_hidx]
          current_search_area = current_macro_blocks[macro_block_vidx:macro_block_vidx

          search_results = cost_function(current_search_area, prev_macro_block)

          diff_min_vec = np.array(np.unravel_index(np.argmin(search_results, axis=None

          vector_macro_blocks[macro_block_vidx, macro_block_hidx, 0:2] = diff_min_vec

      disp_vectors.append(vector_macro_blocks)

      fig, ax = plt.subplots()
      ax.set_title('Movement from frame '+str(idx_im)+' to frame '+ str(idx_im+1))

      print("image number:", idx_im)
      if plot_as_img:
        ax.imshow(vector_macro_blocks*250)
      else:
        x, y = np.meshgrid(np.arange(vector_shape[1]), np.arange(vector_shape[0]))
        u = vector_macro_blocks[:, :, 0]
        v = vector_macro_blocks[:, :, 1]

        ax.quiver(x, y, u, v,
                  color="C0", angles='xy',
                  scale_units='xy', scale=1, width=.01,
                  headwidth=2, minshaft=1.5)
        plt.gca().invert_yaxis()

      plt.show()

    return disp_vectors
```

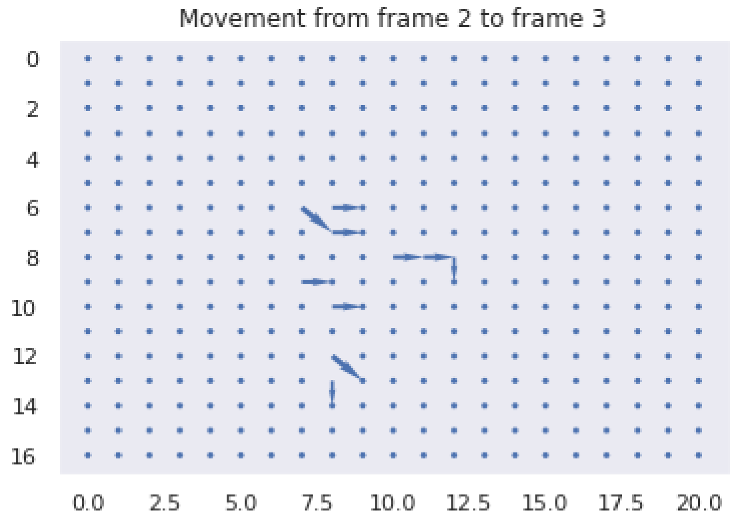
In [ ]:

```
vecs16 = gen_disp_vectors(images, block_size=16, search_area_size=32, plot_as_img=Fa
```

```
Num of macroblocks in search area: 2
image number: 1
```




```
image number: 2
```

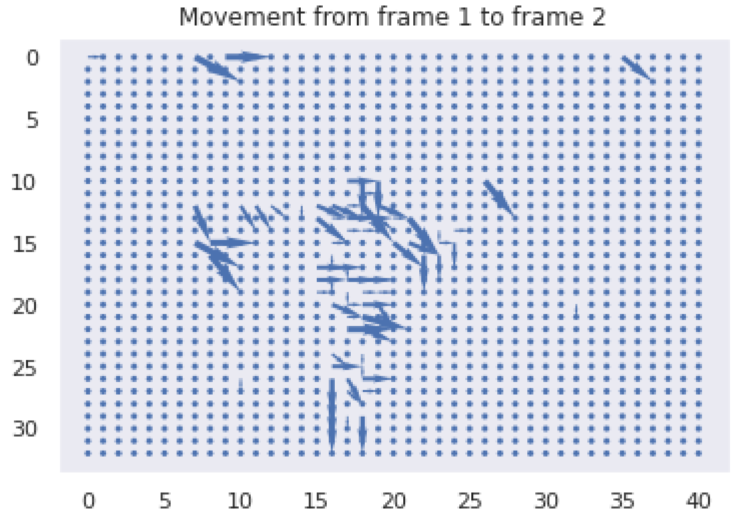


b)

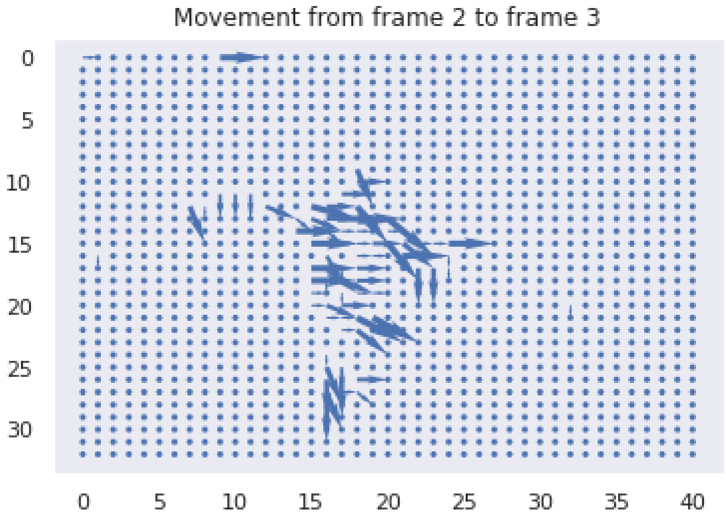
In [ ]:

```python
vecs8 = gen_disp_vectors(images, block_size=8, search_area_size=32, plot_as_img=Fals
vecs8 = gen_disp_vectors(images, block_size=8, search_area_size=32, plot_as_img=True
```

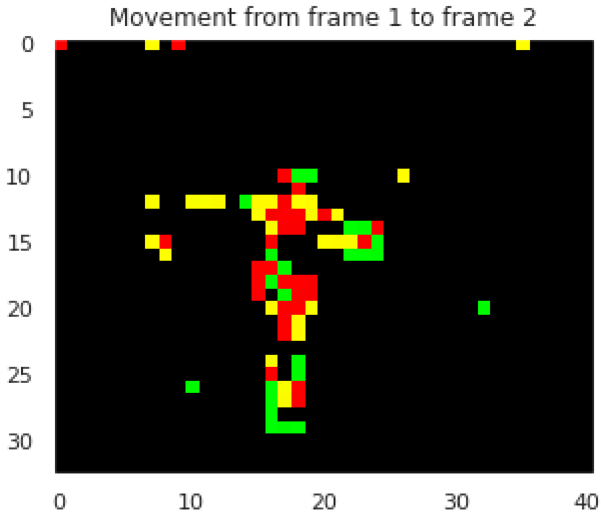
```
Num of macroblocks in search area: 4
image number: 1
```



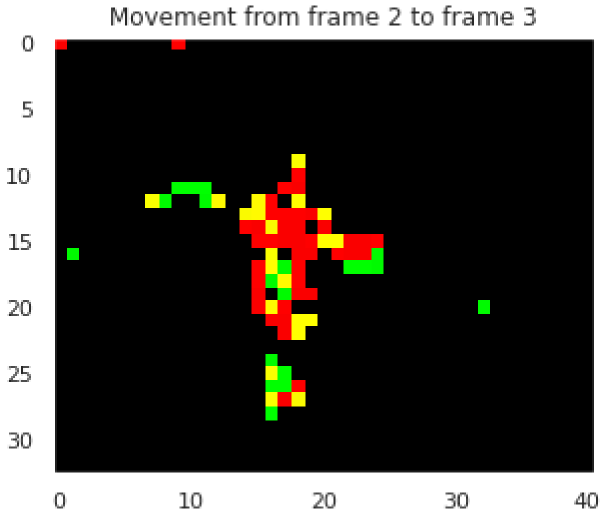
```
image number: 2
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats o
r [0..255] for integers).
Num of macroblocks in search area: 4
image number: 1
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats o
r [0..255] for integers).
image number: 2
```



Affect on number of operations

Going from 16x16 macro blocks to 8x8 blocks in an search area of 32x32 quadruple the number of possible blocks considered for movement bewteen frames. This also result in quadruple the amout of operations.

Referances

[1] MPEG video coding, Dr. S.R. Ely (BBC), EBU Technical Review Winter 1995