Project 02. Image Processing
Gaussian and bilateral filters

Team:

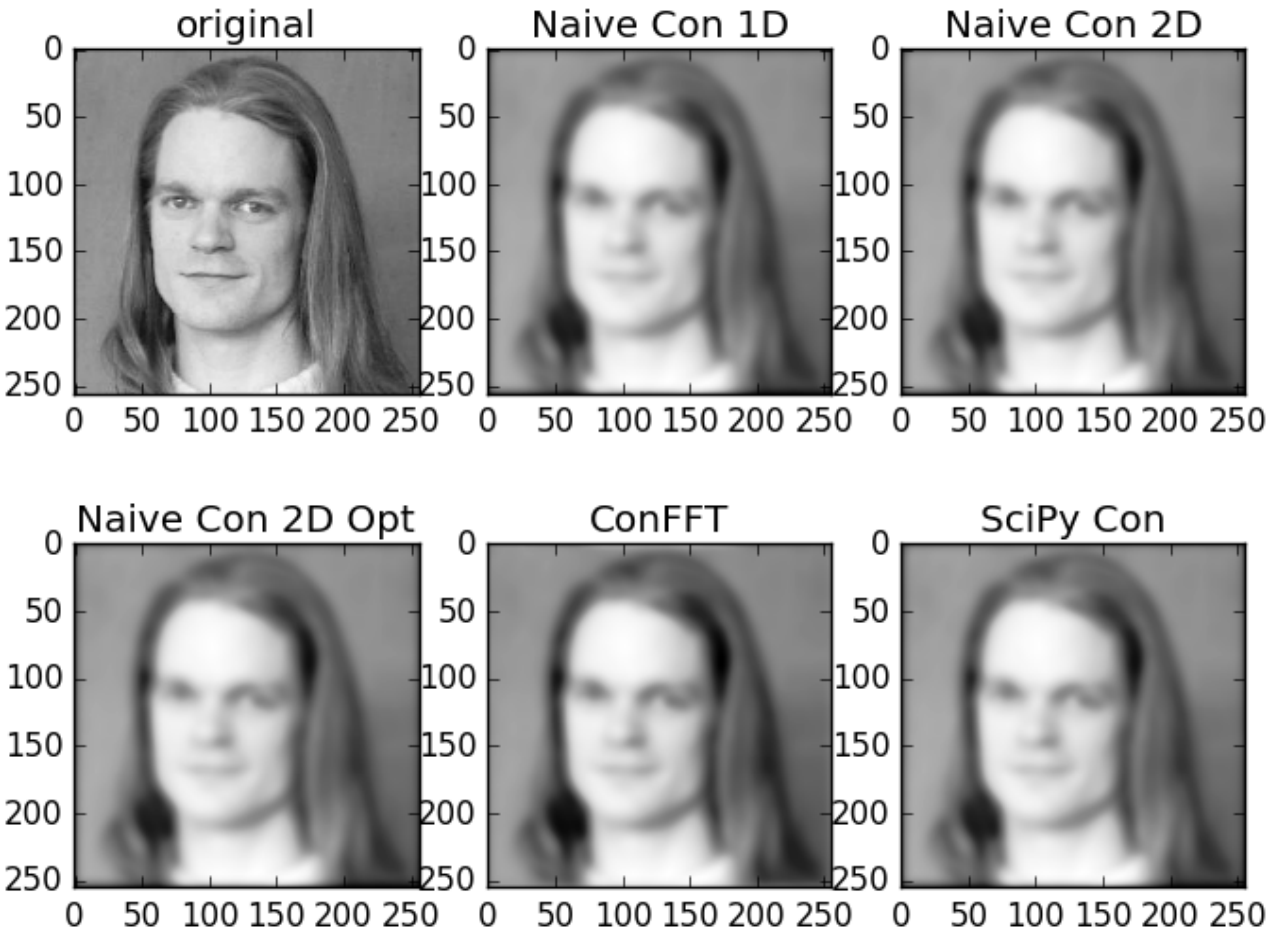
- Ahmad Zakor
- Mubashir Hanif
- Omar Gutiérrez

Task 2.1. Smoothing operations

At the right the different filters used to smooth an image.

It does exist a relationship between the size of filter and sigma.

We deal with the borders padding the images.



Task 2.1. Naïve 2D Convolution

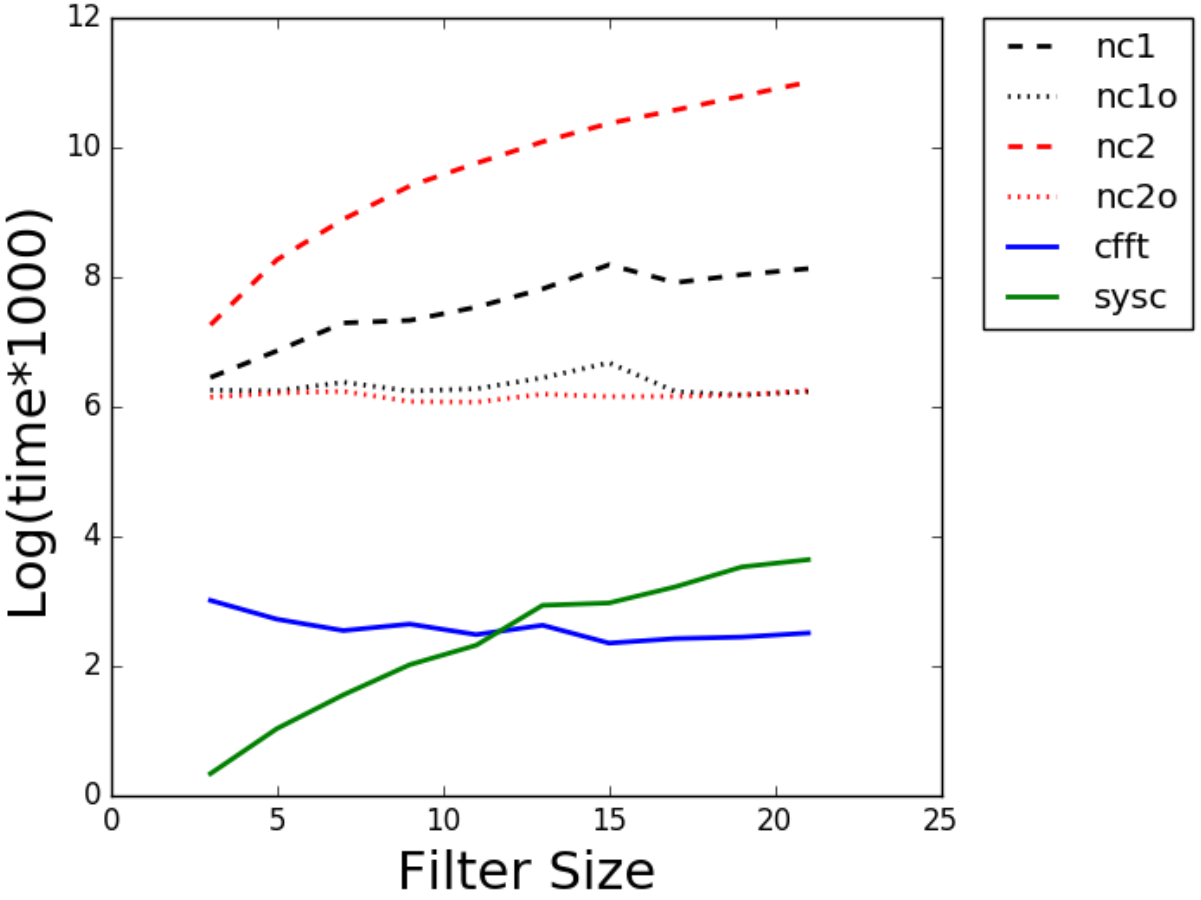
- Delete loops. We can do the next optimization

```
summe = 0.0
for i in range( 0, m ):
    xdash = int( x + ( i - bound ) )
    if( 0 > xdash or w <= xdash ):
        summe += 0.0
    else:
        summe += img[ xdash, y ] * filter1d[ i ]
new_image[ x, y ] = summe
```

```
summe = 0.0
extract = padded_img[x:(m+x), y]
mul = np.multiply(extract, filter1d)
summe = mul.sum()
new_img[x,y] = summe
```

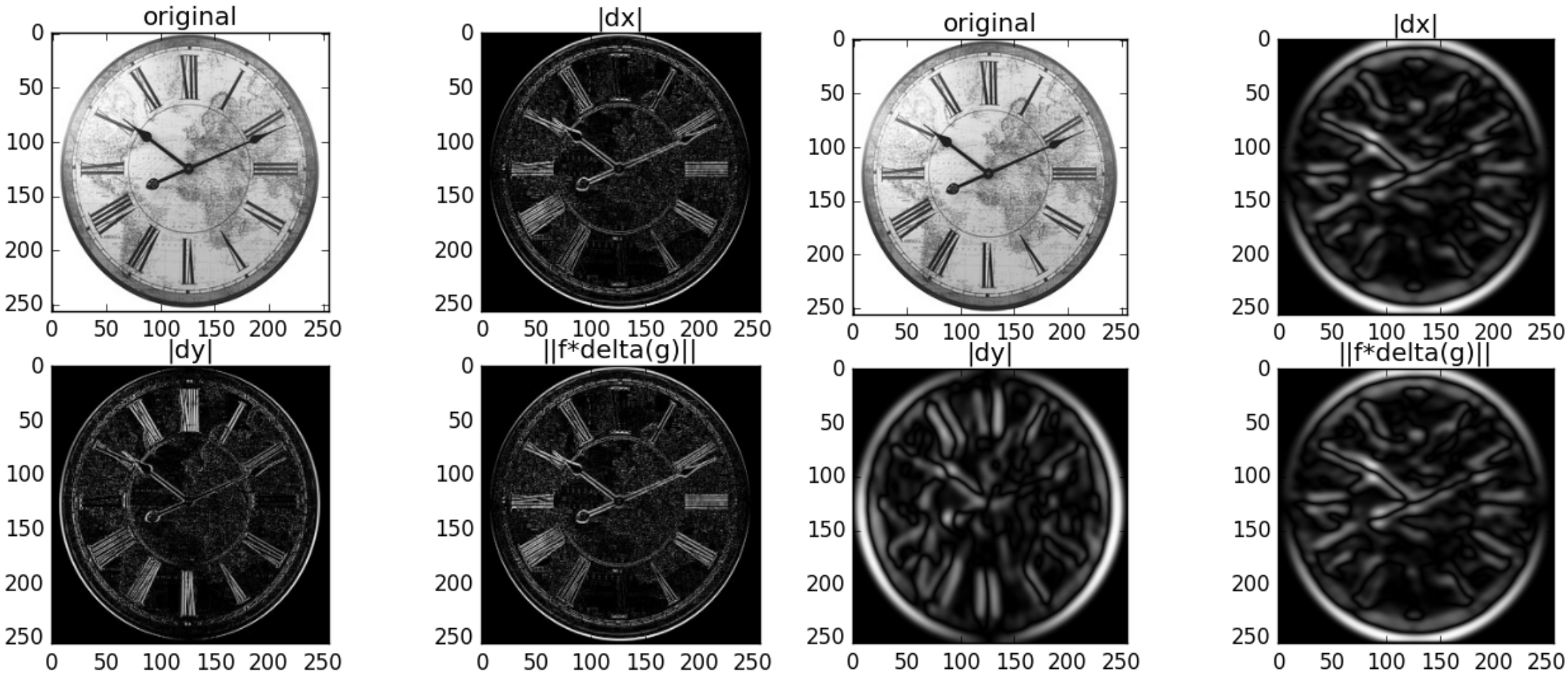
Task 2.1. Timing

- Notice the intersection between the green and blue line.
- The previous optimization makes difference!

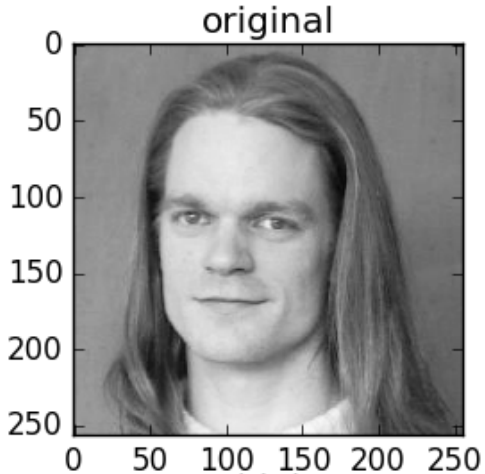


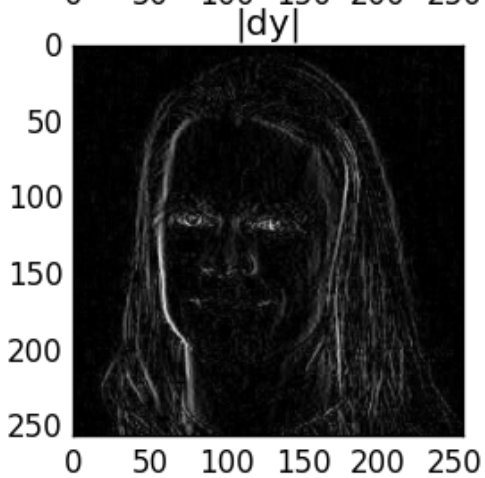
Task 2.2. Gradient images

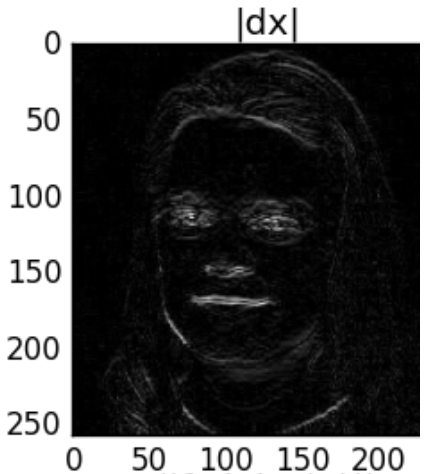
- The gradient operator is applied to an smoothed version.

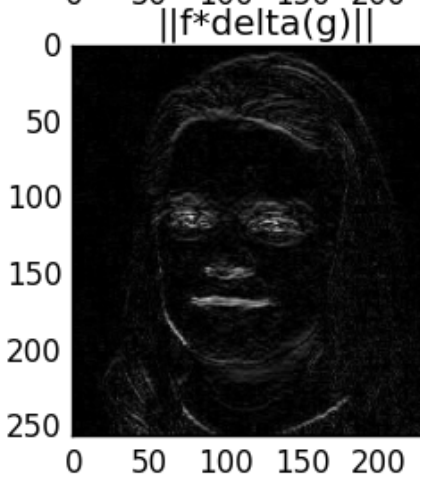


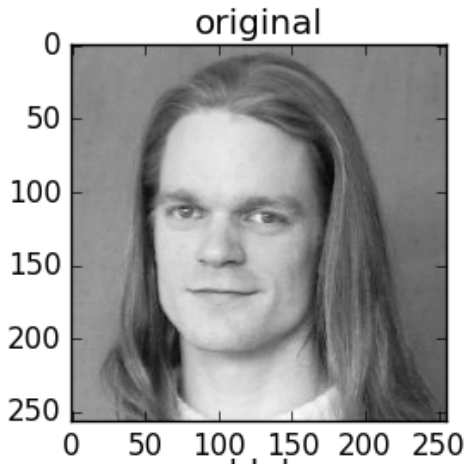
Task 2.2. Gradient images

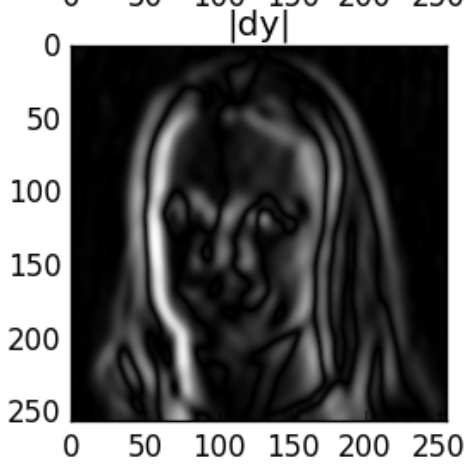


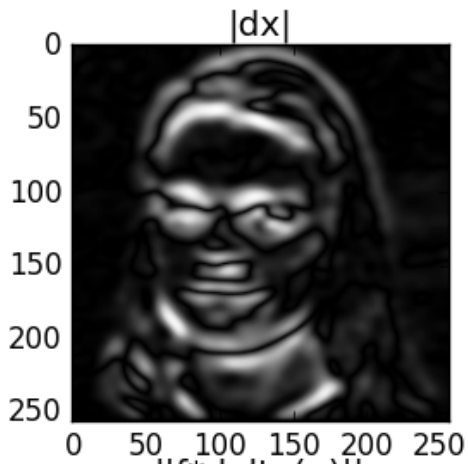


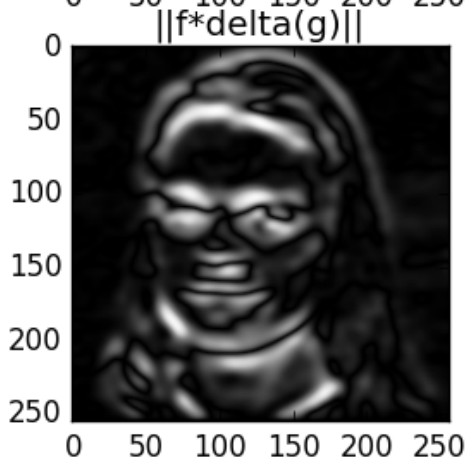












universitätbonn

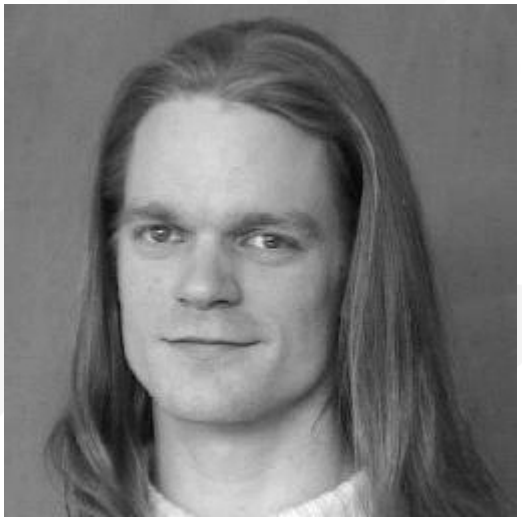
b-it

Task 2.3. Gaussian smoothing using recursive filter

causal

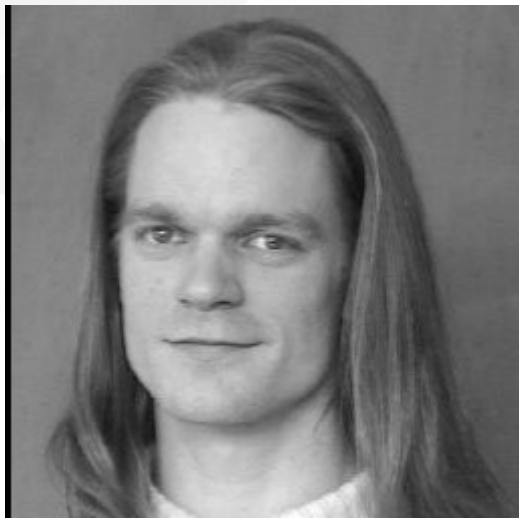
sigma= 0.666

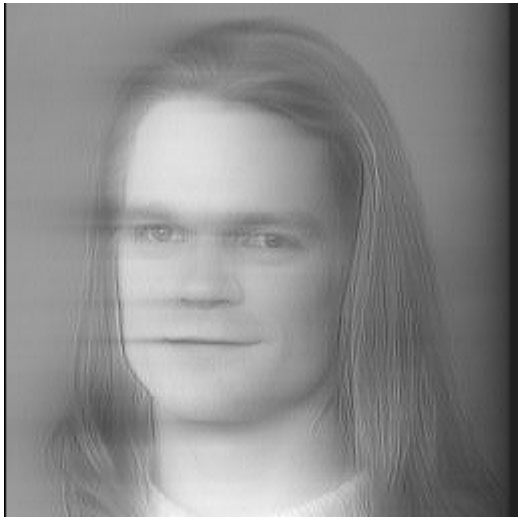
sigma= 3.666



original

anti-causal





result for
sigma = 3.666

Task 2.3. Gaussian smoothing using recursive filter

- **Causal** filter we do the recursion from ***right to left***.

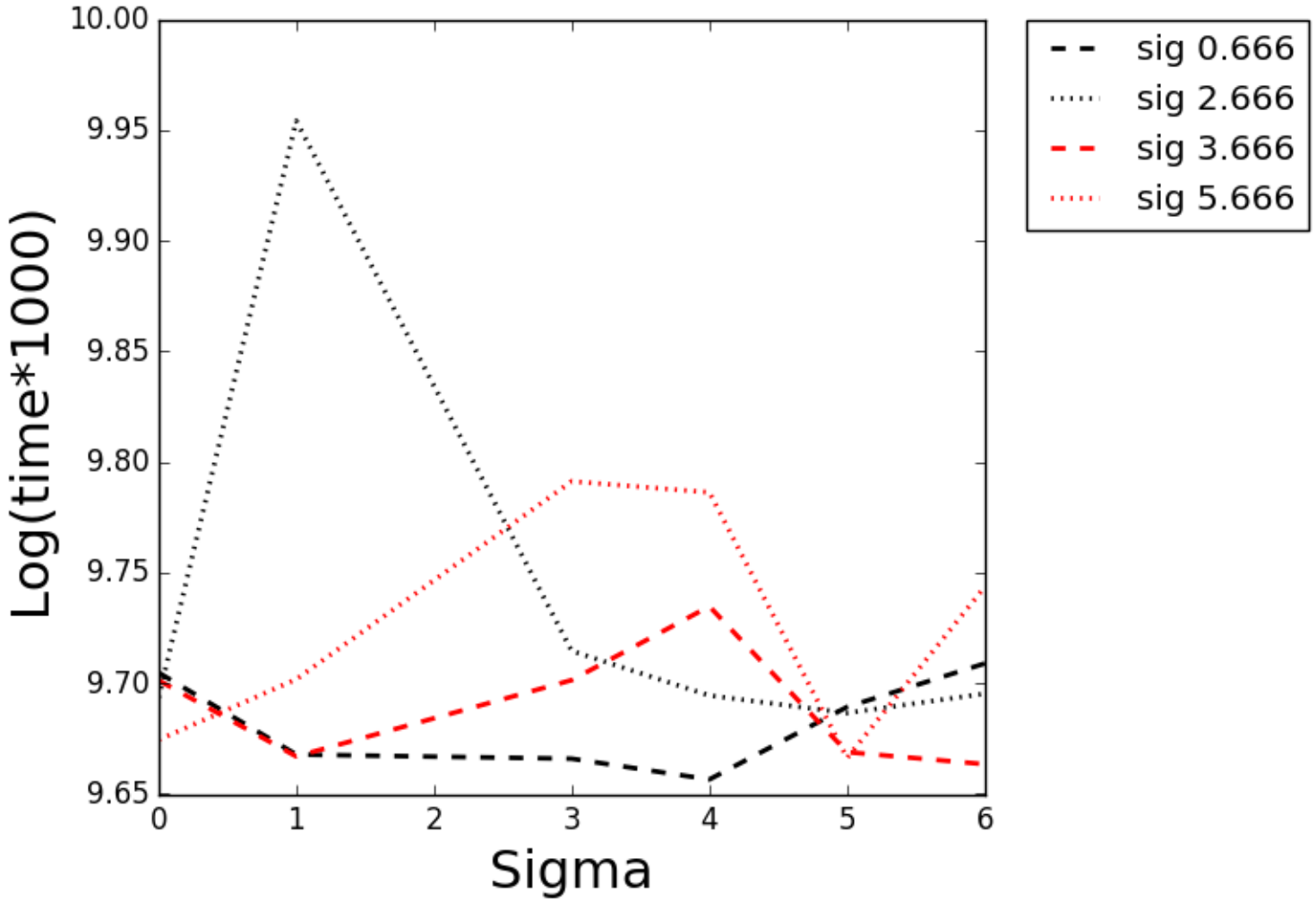
$$\sum_{m=0}^{3} a_m^+ x[n-m] - \sum_{m=1}^{4} b_m^+ y^+[n-m]$$

- **Anti-causal** filter we do the recursion from ***left to right***.

$$\sum_{m=1}^{4} a_m^- x[n+m] - \sum_{m=1}^{4} b_m^- y^-[n+m]$$

- Although, the recursive filter *should* be faster, using vector libraries we need to be careful with the implementation:
  - We can do the summations with element-wise operations.
  - In the recursive process we can avoid to call a function already computed.

Task 2.3. Timing



Task 2.4. Bilateral filters

- A **bilateral filter** is a non-linear, edge-preserving and noise-reducing smoothing filter for images. The intensity value at each pixel in an image is replaced by a weighted average of intensity values from nearby pixels.

$$\tilde{f}[x,y] = \gamma[x,y] \cdot \sum_{i=x-\frac{m}{2}}^{x+\frac{m}{2}} \sum_{j=y-\frac{n}{2}}^{y+\frac{n}{2}} g_\rho\left[f[x,y] - f[x+i,y+j]\right] G_\sigma[i,j] f[x+i,y+j]$$

Task 2.4. Bilateral filters

Python

2 Minutes

C++

5 Seconds

Task 2.4. Bilateral filter

$\rho = 10, \sigma = 1.5$

$\rho = 10, \sigma = 5$

$\rho = 10, \sigma = 10$

$\rho = 100, \sigma = 1.5$

$\rho = 100, \sigma = 5$

$\rho = 100, \sigma = 10$

Task 2.4. Bilateral filter

$\rho = 10, \sigma = 1.5$

$\rho = 10, \sigma = 5$

$\rho = 10, \sigma = 10$

$\rho = 100, \sigma = 1.5$

$\rho = 100, \sigma = 5$

$\rho = 100, \sigma = 10$

Task 2.4. Bilateral filters

$\rho = 10, \sigma = 1.5$

$\rho = 10, \sigma = 5$

$\rho = 10, \sigma = 10$

$\rho = 100, \sigma = 1.5$

$\rho = 100, \sigma = 5$

$\rho = 100, \sigma = 10$

Task 2.4. Bilateral filters

$\rho = 10, \sigma = 1.5$

$\rho = 10, \sigma = 5$

$\rho = 10, \sigma = 10$

$\rho = 100, \sigma = 1.5$

$\rho = 100, \sigma = 5$

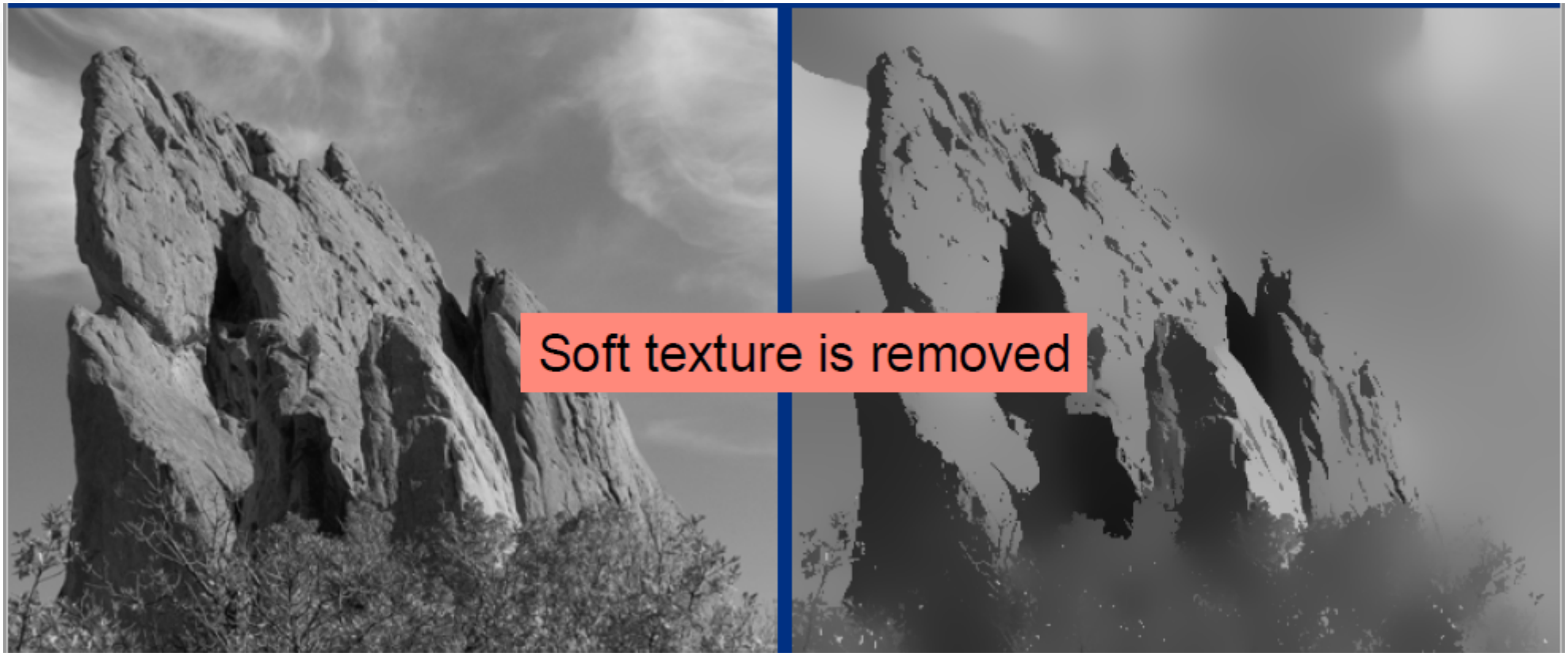
$\rho = 100, \sigma = 10$

universitätbonn

b-it

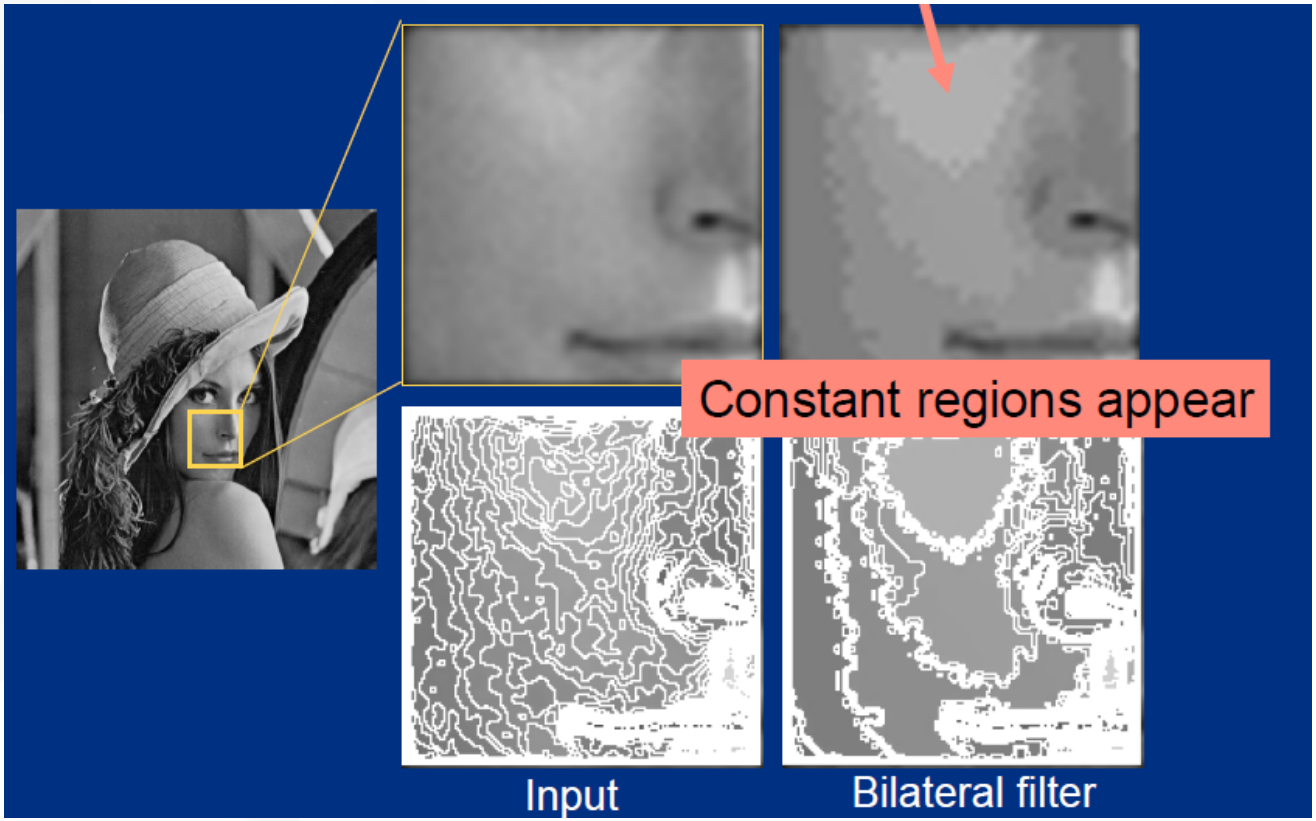
Task 2.4. Bilateral filters

- Limitations: Soft texture removed



Task 2.4. Bilateral filters

- Constant regions appear



Can this be an advantage?

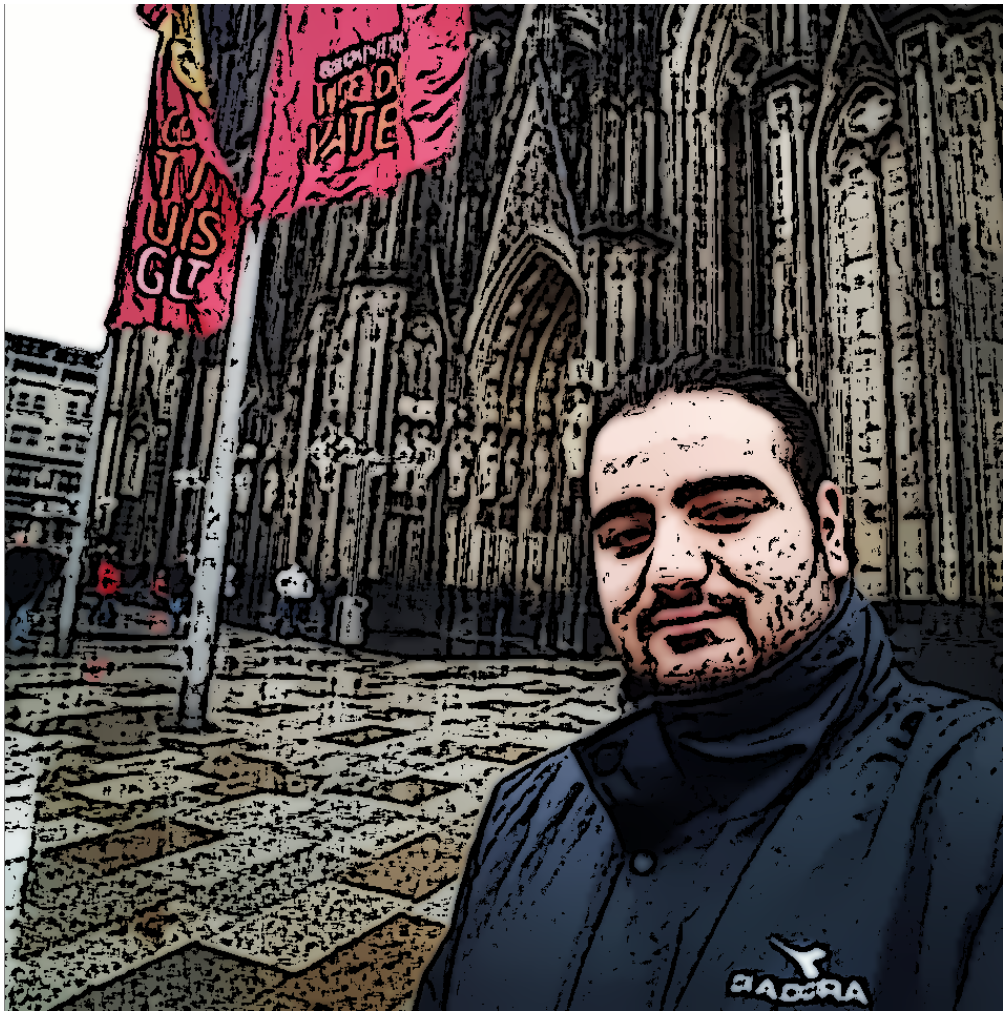
Task 2.4. Bilateral filters

Yes, we can make cartoons!

Task 2.4. Bilateral filters

- Cartoonize a photo:
  - Down sample image using Gaussian pyramid
  - Apply bilateral filter
  - Up sample image to original size
  - Convert to grayscale and apply median blur
  - Detect and enhance edges
  - Convert back to color image

Task 2.4. Bilateral filters



universitätbonn

b-it