Image Processing - Assignment 2

Group 3
Martin Eek Gerhardsen

October 25

Department of Engineering Cybernetics

Contents

1 Convolutional Neural Networks **1**

1.1 Task 1: Theory 1

1.2 Task 2: Programming 6

2 Filtering in the Frequency Domain **11**

2.1 Task 3: Theory 11

2.2 Task 4: Programming 13

1 Convolutional Neural Networks

1.1 Task 1: Theory

a)

Listing the equations for calculating the resulting height and width:

$$W_{i+1} = (W_i - F_W + 2P_W)/S_W + 1 \tag{1}$$
$$H_{i+1} = (H_i - F_H + 2P_H)/S_H + 1 \tag{2}$$

Now defining:

$$\begin{aligned} S_W &= S_H = 1 \\ F_W &= F_H = 5 \\ H_2 &= H_1 = H \\ W_2 &= W_1 = W \end{aligned}$$

Then we get for the width:

$$\begin{aligned} W_2 &= (W_1 - F_W + 2P_W)/S_W + 1 \\ W - 1 &= W - 5 + 2P_W \\ 2P_W &= 4 \\ P_W &= 2 \end{aligned}$$

And for height:

$$\begin{aligned} H_2 &= (H_1 - F_H + 2P_H)/S_H + 1 \\ H - 1 &= H - 5 + 2P_H \\ 2P_H &= 4 \\ P_H &= 2 \end{aligned}$$

So we should therefore be padding with $P_H = 2$ and $P_W = 2$.

c)

Now defining:

$$\begin{aligned} S_H &= S_W = 1 \\ P_H &= P_W = 0 \\ H_1 &= W_1 = 512 \\ H_2 &= W_2 = 504 \\ F_H &= ? \\ F_W &= ? \end{aligned}$$

Then using the same equations as above, we calculate:

$$\begin{aligned}
H_2 &= (H_1 - F_H + 2 * P_H)/S_H + 1 \\
504 &= (512 - F_H + 2 * 0)/1 + 1 \\
F_H &= 512 + 1 - 504 = 9 \\
\\
W_2 &= (W_1 - F_W + 2 * P_W)/S_W + 1 \\
504 &= (512 - F_W + 2 * 0)/1 + 1 \\
F_W &= 512 + 1 - 504 = 9
\end{aligned}$$

We also knew from the task that the kernel was supposed to be square, and its dimensions odd, which we can confirm from the result. The kernel must therefore be of the size $9x9$.

d)

Starting with subsampling with neighbourhoods, defining:

$$\begin{aligned}
H_1 &= W_1 = 512 \\
S_H &= S_W = 2 \\
P_H &= P_W = 0 \\
F_H &= F_W = 2 \\
H_2 &= ? \\
W_2 &= ?
\end{aligned}$$

Then we get:

$$\begin{aligned}
H_2 &= (H_1 - F_H + 2 * P_H)/S_H + 1 \\
&= (512 - 2 + 2 * 0)/2 + 1 = 255 \\
W_2 &= (W_1 - F_W + 2 * P_W)/S_W + 1 \\
&= (512 - 2 + 2 * 0)/2 + 1 = 255
\end{aligned}$$

Then, using the same information utilized above, we define:

$$\begin{aligned}
H_1 &= W_1 = 255 \\
S_H &= S_W = 1 \\
P_H &= P_W = 0 \\
F_H &= F_W = 9 \\
H_2 &= ? \\
W_2 &= ?
\end{aligned}$$

Then we calculate:

$$\begin{aligned} H_2 &= (H_1 - F_H + 2 * P_H)/S_H + 1 \\ &= (255 - 9 + 2 * 0)/1 + 1 = 247 \\ W_2 &= (W_1 - F_W + 2 * P_W)/S_W + 1 \\ &= (255 - 9 + 2 * 0)/1 + 1 = 247 \end{aligned}$$

Then the spatial dimensions of the pooled feature maps in the first layer would be $247x247$.

e)

The number of pooled feature maps would be the same as the number of feature maps, which is 12.

f)

Defining:

$$\begin{aligned} H_1 &= W_1 = 247 \\ S_H &= S_W = 1 \\ P_H &= P_W = 0 \\ F_H &= F_W = 3 \\ H_2 &=? \\ W_2 &=? \end{aligned}$$

Then:

$$\begin{aligned} H_2 &= (H_1 - F_H + 2 * P_H)/S_H + 1 \\ &= (247 - 3 + 2 * 0)/1 + 1 = 245 \\ W_2 &= (W_1 - F_W + 2 * P_W)/S_W + 1 \\ &= (247 - 3 + 2 * 0)/1 + 1 = 245 \end{aligned}$$

Therefore, the sizes of the feature map of the second layer would be $245x245$.

g)

Defining the 0-layer as the input layer, so that the other layers can be defined incrementally as well as being the given layer in the table. Furthermore, superscript will be used to specify the layer type. I will use C for Conv2D, M for MaxPool2D

$$
\begin{aligned}
H_0 &= W_0 = 32 \\
C_0 &= 3 \\
S_H^C &= S_W^C = 1 \\
P_H^C &= P_W^C = 2 \\
F_H^C &= F_W^C = 5 \\
S_H^M &= S_W^M = 2 \\
P_H^M &= P_W^M = 0 \\
F_H^M &= F_W^M = 2 \\
C_1 &= 32 \\
C_2 &= 64 \\
C_3 &= 128
\end{aligned}
$$

The flattening-layer will then convert the $4x4x128$ result from layer 3 to a vector of size $4 * 4 * 128 = 2048$. Then this is mapped with to 64 hidden units, and finally to the 10, representing the ten different classes (digits).

$$n_{i+1} = F_H^C * F_W^C * C_i * C_{i+1} + C_{i+1} = 25 * C_i * C_{i+1} + C_{i+1} \tag{3}$$

To calculate the number of parameters, we look at each filters individually. Using eq. (3), we can calculate the number of parameters for each layer:

$$
\begin{aligned}
n_1 &= 25 * 3 * 32 + 32 = 2432 \\
n_2 &= 25 * 32 * 64 + 64 = 51264 \\
n_3 &= 25 * 64 * 128 + 128 = 204928 \\
n_4 &= 4 * 4 * 128 * 64 + 1 = 131073 \\
n_5 &= 64 * 10 + 1 = 641 \\
n &= \sum_{i=1}^{5} n_i = 390338
\end{aligned}
$$

I also calculate the spatial dimensions of the layers. This wasn't strictly needed to calculate the number of parameters, but was useful for task 2:

$$\begin{aligned}
H_1^C &= (H_0 - F_H^C + 2 * P_H^C)/S_H^C + 1 = (32 - 5 + 2 * 2)/1 + 1 = 32 \\
H_1^M &= (H_1^C - F_H^M + 2 * P_H^M)/S_H^M + 1 = (32 - 2 + 2 * 0)/2 + 1 = 16 \\
H_2^C &= (H_1^M - F_H^C + 2 * P_H^C)/S_H^C + 1 = (16 - 5 + 2 * 2)/1 + 1 = 16 \\
H_2^M &= (H_2^C - F_H^M + 2 * P_H^M)/S_H^M + 1 = (16 - 2 + 2 * 0)/2 + 1 = 8 \\
H_3^C &= (H_2^M - F_H^C + 2 * P_H^C)/S_H^C + 1 = (8 - 5 + 2 * 2)/1 + 1 = 8 \\
H_3^M &= (H_3^C - F_H^M + 2 * P_H^M)/S_H^M + 1 = (8 - 2 + 2 * 0)/2 + 1 = 4 \\
W_1^C &= (W_0 - F_W^C + 2 * P_W^C)/S_W^C + 1 = (32 - 5 + 2 * 2)/1 + 1 = 32 \\
W_1^M &= (W_1^C - F_W^M + 2 * P_W^M)/S_W^M + 1 = (32 - 2 + 2 * 0)/2 + 1 = 16 \\
W_2^C &= (W_1^M - F_W^C + 2 * P_W^C)/S_W^C + 1 = (16 - 5 + 2 * 2)/1 + 1 = 16 \\
W_2^M &= (W_2^C - F_W^M + 2 * P_W^M)/S_W^M + 1 = (16 - 2 + 2 * 0)/2 + 1 = 8 \\
W_3^C &= (W_2^M - F_W^C + 2 * P_W^C)/S_W^C + 1 = (8 - 5 + 2 * 2)/1 + 1 = 8 \\
W_3^M &= (W_3^C - F_W^M + 2 * P_W^M)/S_W^M + 1 = (8 - 2 + 2 * 0)/2 + 1 = 4
\end{aligned}$$

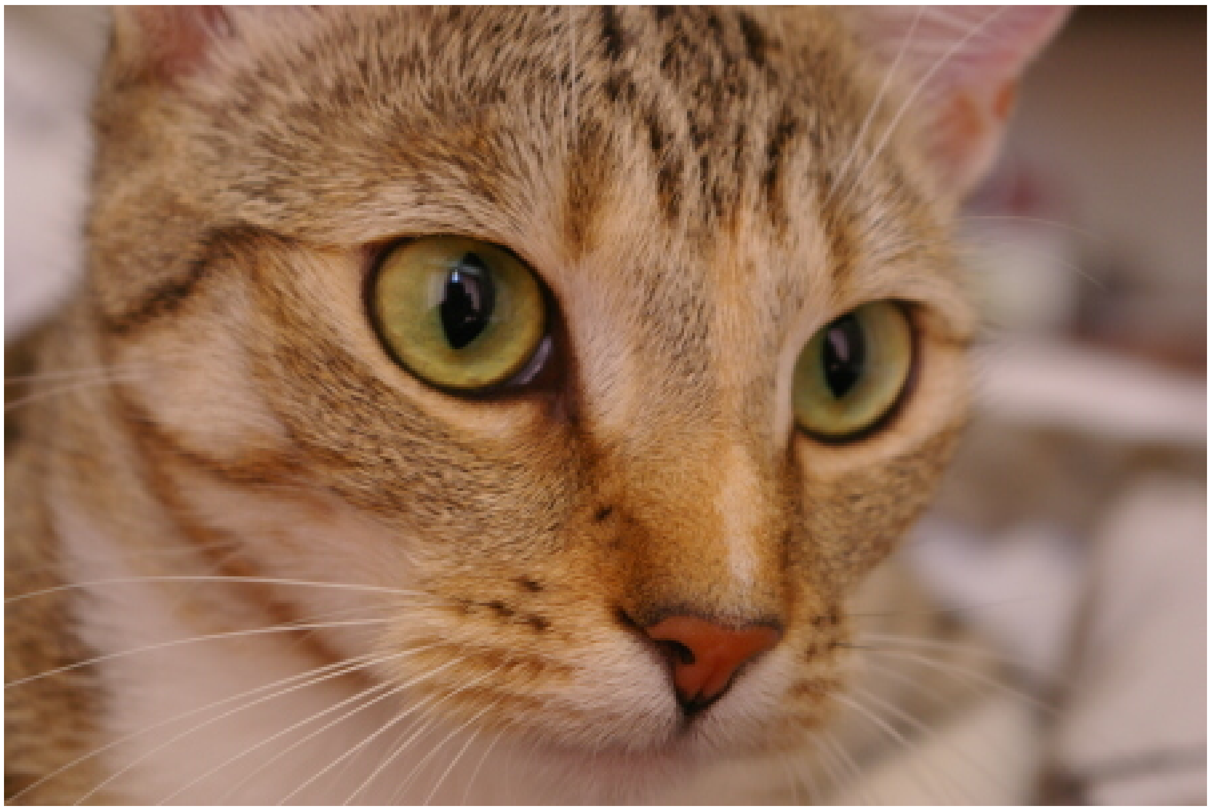

Figure 1: Chelsea without maxpooling

1.2 Task 2: Programming

a)

For Chelsea, see fig. 1, and for the maxpooled version see fig. 2. For the checkerboard, see fig. 3, and the maxpooled version see fig. 4. Of course, as the checkerboard has very quick changes in intensity, and as we maxpool, this change happens too quick to be expressed in the maxpooled version. Simply, inside the kernel only the white colour is chosen, and we can no longer see the black part of the checkerboard.

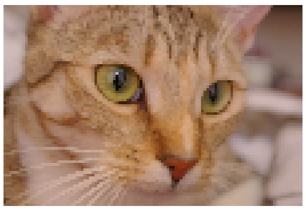
Figure 2: Chelsea with maxpooling

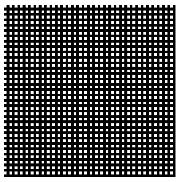
Figure 3: Checkerboard without maxpooling

b)

See fig. 5 for the plot of the cross entropy loss for the test and training data. The final validation loss was calculated as 0.06908821107620955, and the final validation accuracy was calculated as 0.9744. The final training loss was calculated as 0.06584324309183384 and the final training accuracy was calculated as 0.9777166666666667.

We can notice overfitting when training loss is a lot smaller than validation loss. Both losses were, as we can see, pretty similar, and we shouldn't conclude that there is any overfitting here.

c)

See fig. 6 for the plot of the cross entropy loss for the test and training data. The final validation loss was calculated as 0.023527828954214854, and the final validation accuracy was calculated as 0.9917. The final training loss was calculated as 0.011397973218215578, and the final training accuracy was calculated as 0.9968666666666667.

Again, as above, we can notice overfitting when training loss is a lot smaller than validation loss. In this case, there seems to be evidence of overfitting, as the validation loss seems to be about double the training loss. This may not be significant enough, depending on the application of this neural network, but there is still more evidence of overfitting in this task

Figure 4: Checkerboard with maxpooling

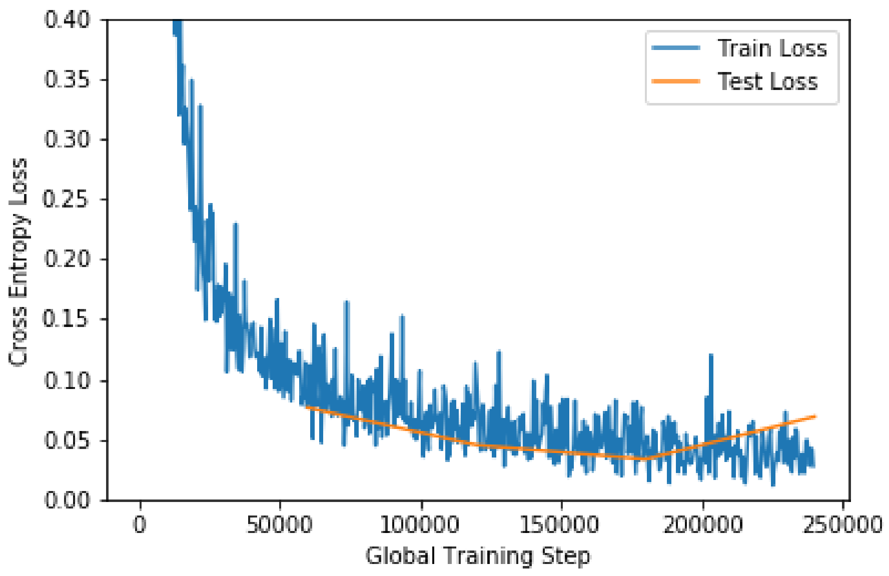

Figure 5: Cross entropy loss for training and test. 4 epochs

than the previous.

d)

See the result from fig. 7.

e)

Kernel 5: This kernel is very similar to a Sobel x-kernel, with *darker* weights on the left and *lighter* weights on the right. Sobel x is used for edge detection/enhancement in the x-direction, and we can notice this in the image, as especially the stripes on the zebras stomach is highlighted, while the stripes on its face and neck are more diffuse, as these are more in the y-direction. The kernel seems to be more purple on the left, which indicates more red and blue, and more green/white on the right, which indicates more red, green and blue with a focus on green. What we may note, is that the focus seems to be on green, as the largest difference seems to be between these (no green on the left, mostly green on the right).

Kernel 8: This kernel highlights the centre/diagonal, while the diagonal above and below are darker. This seems to emphasise the whiter diagonals of the image, but it also seems like this kernel is removing certain diagonals, as seen from the resulting image.

Kernel 19: There is a huge focus on green in this kernel, with a focus on

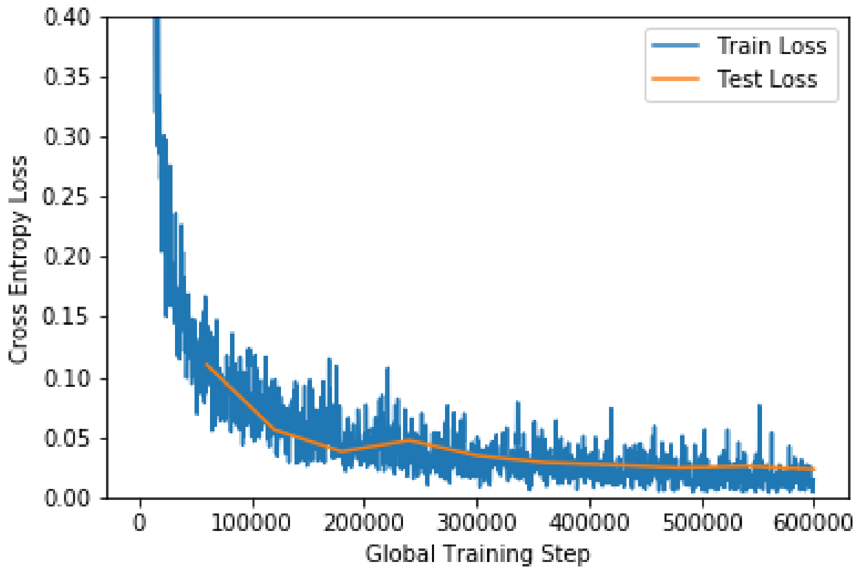

Figure 6: Cross entropy loss for training and test. 10 epochs

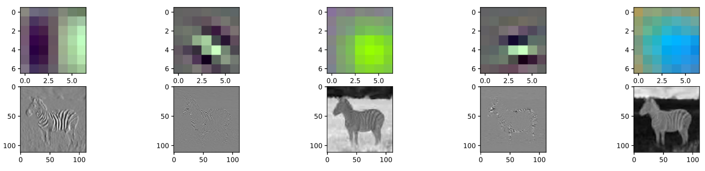

Figure 7: Visualization of convolution layers with indices [5, 8, 19, 22, 34]

the bottom right of the kernel. So this kernel seems to extract the ground / grass. As yellow is a combination of green and red, we can see in the image that the grass is highlighted.

Kernel 22: The kernel looks similar to a Sobel y-kernel, though it doesn't cover the entire kernel and more diagonalized. This, then, seems to be detecting edges around larger objects. As we can see from the image, it seems to be detecting the edges (primarily in the y-direction) of the zebra.

Kernel 34: This kernel is very similar to kernel 19, though with a focus on blue. From the image, we can see that the sky is highlighted, so this kernel seems to be extracting the sky, or other larger blue bodies (speculation: possibly water also?).

2 Filtering in the Frequency Domain

2.1 Task 3: Theory

a)

From the convolution theorem, we can see that the Fourier transform of a convolution of two signals is multiplication of the Fourier transforms of those individual signals. So a stepwise description would be:

- Ensure the signals are of the same size using padding (zero padding)
- Find the Fourier transform of the individual signals (i.e. using FFT)
- Pointwise multiply the transformed signals
- Use inverse Fourier transform to find the convolution of the original signals

b)

Low- and high-pass filters tell us which frequencies should be kept, low-pass allows low frequencies to pass, and high-pass allows high frequencies to pass. Both then removes the frequencies not allowed to pass.

This means that low-pass filters should be close to the filter gain (i.e. 1) for frequencies closer to the origin than the cut-off frequency (low frequencies), and should be close to zero for higher frequencies than the cut-off frequency. Depending on how we want the filter to behave, the filter can either be similar to a cylinder or a cone, where cylinder gives a much *harsher* cut-off frequency (no frequencies after cut-off) and cone gives a *smoother* cut-off frequency (allowing, but suppressing some frequencies after cut-off).

The high-pass filter is simply the $K - LPF$, where K is the filter gain (i.e. 1 again) and LPF is the low-pass filter. This implies that the high-pass filter follows the same pattern as the low-pass, but where the low-pass filter includes frequencies, the high-pass filter removes them, and vice versa where the high-pass includes the frequencies.

c)

As the images are shifted, we find the low frequencies in the centre and high around the image. Also using the white parts of the images as high amplitude, and black as low amplitude.

a) This is primarily a high-pass filter, as the lower frequencies are suppressed. This filter is primarily a high-pass filter along the y-axis (so high frequencies are only allowed given that they are along the y-axis), rather than both x and y.

b) This is a high-pass filter, the lower frequencies in the centre are largely suppressed, and the higher frequencies are multiplied with values closer to 1. Though, as the values closest to the x- and y-axes are slightly darker than the diagonals for high frequencies, the filter prefers the signal to have both high frequencies in x and y at the same time.

c) This is a low-pass filter, the lower frequencies in the centre are largely included, while the higher frequencies are suppressed.

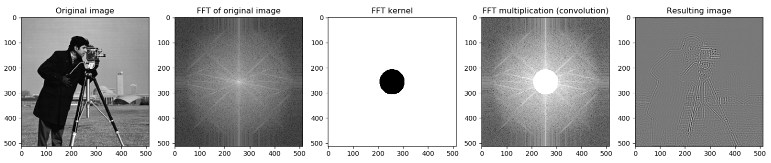

Figure 8: High-pass filtered camera man

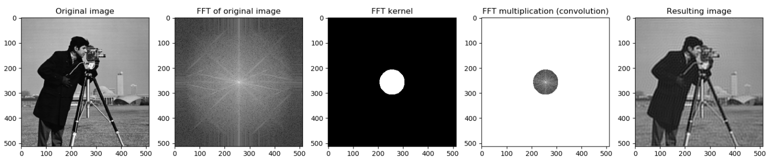

Figure 9: Low-pass filtered camera man

2.2 Task 4: Programming

a)

The high-pass filtered image, with the amplitude of the Fourier transforms, can be seen from fig. 8, and similarly for the low-pass filtered image from fig. 9.

b)

The image convolution process for the gaussian kernel can be seen from fig. 10, and the image convolution process for the horizontal sobel filter can be seen from fig. 11

c)

The image convolution process for the mystery-filtering can be seen from fig. 12. It is pretty simple to see that this filter removes certain filter-bands, which implies that this is a combination of a four notch filters, or inverse

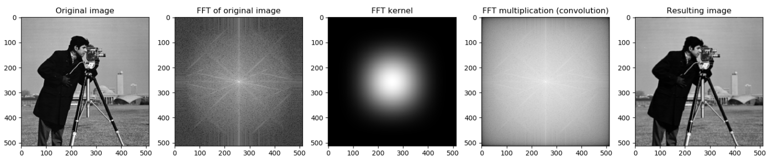

Figure 10: Gaussian filtered camera man

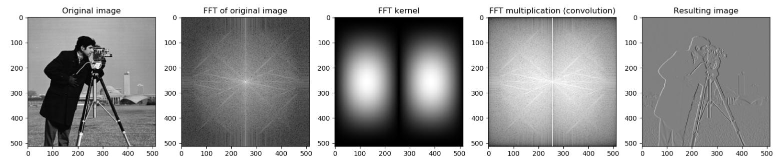

Figure 11: Horizontal sobel filtered camera man

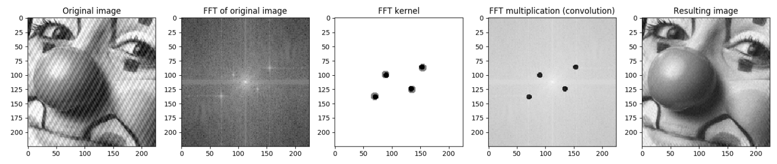

Figure 12: Mystery-filtered clown (notch filter)

bandpass filters. This then *removes* certain bands of frequencies along the x-axis and y-axis.

From the Fourier transform of the image, we can see that certain frequencies are very prevalent, and this filter removes those specific points. These were the primary causes for the picture being so wave-y before applying the filter.

To conclude, I'd call this filter a notch filter (or more accurately a combination of four notch filters).

d)

See the sharpened image from fig. 13.

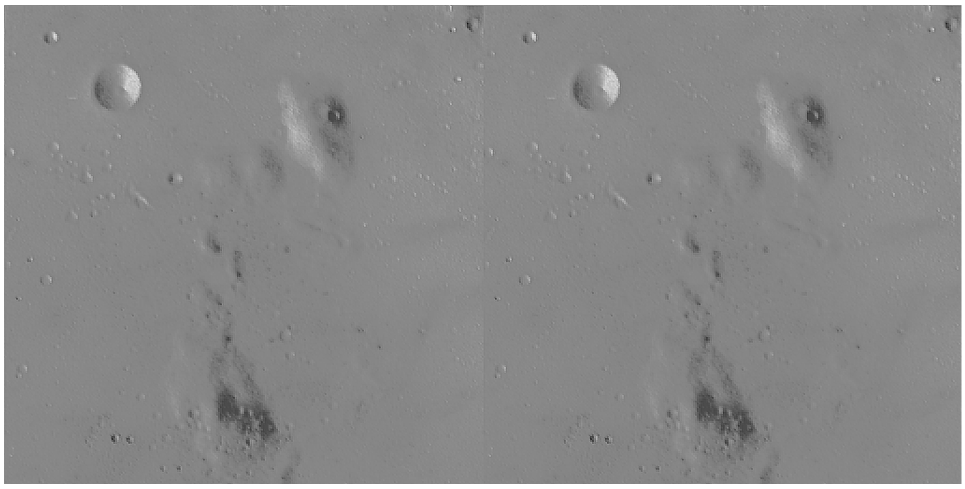
Figure 13: The original and the sharpened image