TDT4195 — Image Processing 2

Torjus Iveland & Aleksander Karlsson

October 2019

1 Convolutional Neural Networks

1.1 Theory

Task A

To have the output shape be equal to the input shape, we want to be able to slide the kernel over *all* the cells in the image. To be able to do this with a 5×5 filter, we need to add 2 units to each side, so that the center of the kernel can be put on all the image cells. *Thus, we need 2 units of padding in both the width and height dimensions.* The number of filters we use does not affect the padding we need to do on the image; however, it does affect the number of channels in the output.

We can check that this is correct using the formula 1 from the assignment.

$$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1 = \frac{W_1 - 5 + 2 * 2}{1} + 1 = W_1$$

$$H_2 = \frac{H_1 - F_H + 2P_H}{S_H} + 1 = \frac{H_1 - 5 + 2 * 2}{1} + 1 = H_1$$

We see that using 2 units of padding, the image retains its shape.

Task B

In this task, we start with the following formula.

$$W_2 = \frac{W_1 - F_w + 2P_W}{S_W} + 1$$

We have that $W_1 = 512$ and $W_2 = 504$. Therefore, we can eliminate the padding term from this formula, and the stride is 1, which gives us the following simplified formula.

$$\begin{gathered} W_2 = W_1 - F_{W_1} + 1 \\ 504 = 512 - F_{W_1} + 1 \\ F_{W_1} = 512 + 1 - 504 = 9 \end{gathered}$$

All the kernels on the first layer are of the same size. Since the kernels are square, the 12 kernels are all 9×9 kernels.

Task C

The spatial dimensions of the first layer output is of size 504×504. The kernel first covers the first 2×2 block of values and computes one output value using these four values. Then, it moves over by two units and repeats the process. After processing all 2×2 grids, the overall effect is that the size of our image is halved, and the final dimensions are $\frac{504}{2} \times \frac{504}{2} = 252 \times 252$.

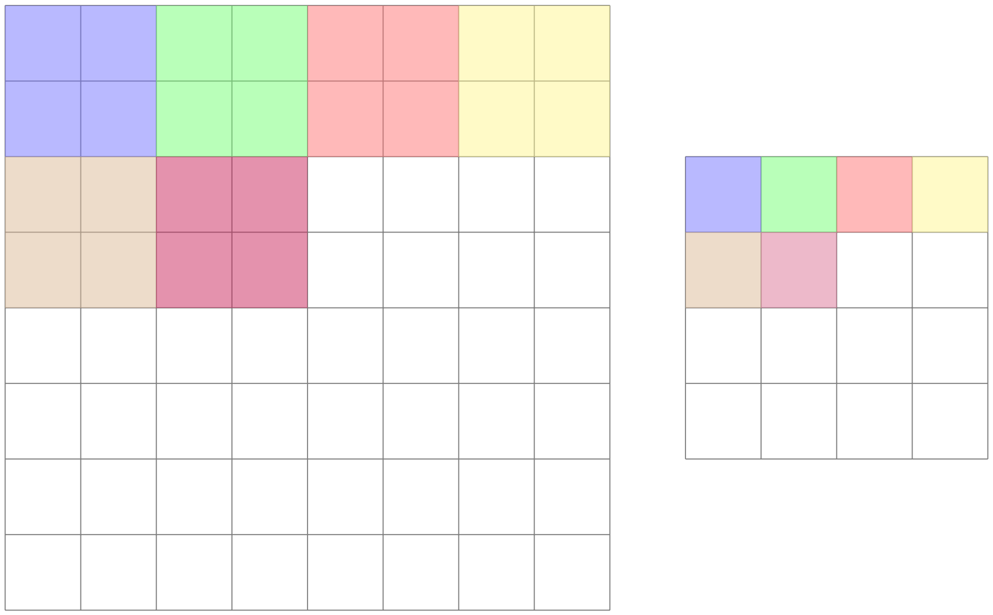

Figure 1: Illustration depicting the reduction from a size 8×8 original image on the left overlapped with kernels of size 2×2 with stride of 2 to the pooled image with size 4×4. Each block on the right contains the max value of the blocks with the same color on the left.

1.1.1 Task D

We know that the convolutional layer outputs 12 feature maps. The max pooling is applied individually to each of these 12 feature maps. Thus, we get shrinked feature maps, but the amount of feature maps after pooling stays the same. Therefore, the depth of the pooled feature maps is 12.

1.1.2 Task E

After the first convolutional layer, the feature maps have size 504×504. We assume the pooling discussed in the previous task also affects this task. After pooling, the feature maps have size 252×252. Next, these feature maps are fed into a convolutional layer with kernels of size 3×3, no padding and a stride of 1. Next, we have

$$W_3 = \frac{W_2 - F_{W_2} + 2P_W}{S_{W_2}} + 1 = \frac{252 - 3 + 2 * 0}{1} + 1 = 250$$
$$H_3 = \frac{H_2 - F_{H_2} + 2P_H}{S_{H_2}} + 1 = \frac{252 - 3 + 2 * 0}{1} + 1 = 250$$

which means that the sizes of the feature maps in the second layer will be 250×250. We assume that no pooling is used in this layer.

1.1.3 Task F

We want to compute the total number of weights and parameters in this network. The max pooling and flattening layers have no parameters to learn, but the convolutional and fully connected layers do.

For the first convolutional layer, we have that there are $5 * 5 * 3 * 32 + 32 = 2432$ parameters, because the kernels have size 5×5, there are 3 colors in the incoming images, there are 32 filters, and these 32 filters have one bias each.

For the second layer we have that there are $5 * 5 * 32 * 64 + 64 = 51264$ parameters, since the kernels are the same size, but the input depth is now 32 and there are 64 filters being learned.

We again use the same logic for the third convolutional layer. Here, we have that there are $5*5*64*128+128 = 204928$ parameters.

Next, we want to find the number of parameters in the fully connected layers. Each node has one weight for each input from the flattening layer, as well as one bias for each node. However, we do not know the number of nodes coming into the first fully-connected layer. To know this, we need to figure out the output size of the last pooling layer. And, to do that, we need to know the size of all the previous layers all the way up to the input images.

The incoming images have shape $[32, 32, 3]$. After passing the images through the first convolutional layer, the output shape is $[32 - 5 + 1, 32 - 5 + 1, 32] = [32, 32, 32]$ — the padding ensures the spatial dimensions remain the same. After passing the images through the pooling layer, this changes to $[16, 16, 32]$, since the spatial dimensions are halved.

Next, we pass this into the second convolutional layer. Again, the spatial dimensions remain the same, so the new output shape is $[16, 16, 64]$. Then, we again perform pooling, which gives $[8, 8, 64]$ as the output shape of layer 2. Next, we pass this into the third convolutional layer, and get $[8, 8, 128]$ as the shape. Lastly, we perform max pooling on this and get $[4, 4, 128]$ as the output shape from the final convolutional layer. After flattening, this means that the first fully-connected layer has $4 * 4 * 128 = 2048$ inputs.

We now know that the first fully-connected layer has 2048 inputs. This means that the first fully-connected layer has $64 * 2048 + 64 = 131136$ parameters.

The second fully-connected layer has 10 nodes. Each node is connected to all the 64 nodes on the previous layer, and each node has a bias. This means that the second fully-connected layer has $10 * 64 + 10 = 650$ parameters.

In total, the network has

$$2432 + 51264 + 204928 + 131136 + 650 = 390410$$

parameters.

1.2 Programming

1.2.1 Task A

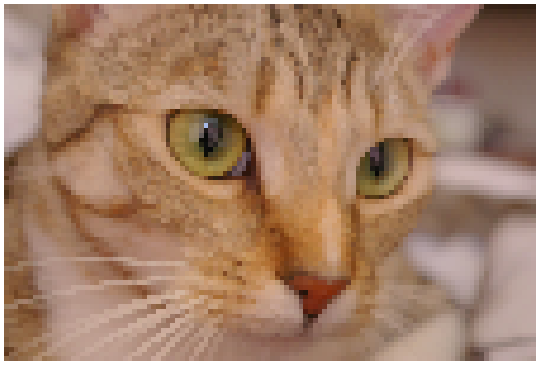

Figure 2: The cat image where max pooling with a kernel and stride of 4 has been applied. Note that this image is scaled.

The above image shows the result of applying the max pooling on the cat Chelsea.

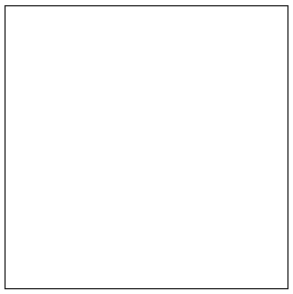

Figure 3: The checkerboard image after applying max pooling, with a border shown around the image since it is otherwise invisible against the background. Note that this image is scaled.

The result of applying max pooling to this image is an image half as large as the original image. It is also completely white. This is because the image is constructed so that each 2×2 cell in the image contains white, which is the largest-value color possible. The result is that all output pixels are white.

1.2.2 Task B

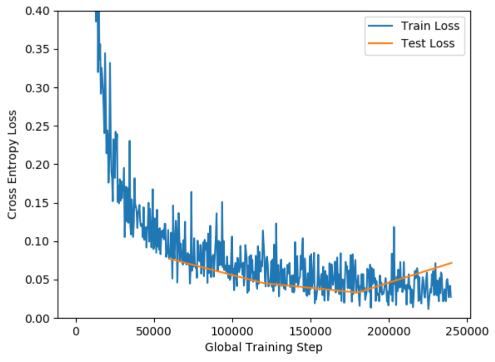

Figure 4: Plot of training and validation loss during training.

The final validation loss are approximately 0.072 and 0.974, respectively.

We can observe that the loss function on the validation data rises slightly towards the end of the graph while the training loss continues to improve. If the testing loss starts to rise while the training loss continues to shrink, that is a possible sign of overfitting. Thus, this is a sign of possible overfitting. One reason for this might be that the model is overly complex and learns the training set too well to generalize.

However, 4 seems like a small amount of epochs for a net of this size, and we have not yet confirmed that this behaviour continues for multiple epochs, since the upwards spike seems to only affect the last epoch. Thus, we do not believe there is conclusive proof of overfitting present in this chart.

1.3 Task C

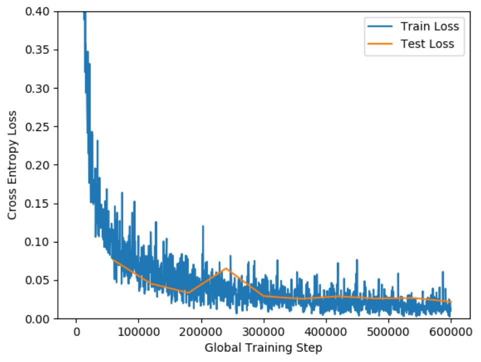

Figure 5: Plot of loss after increasing the amount of epochs.

Here, the final validation loss is approximately 0.02 and the final validation accuracy 0.9928.

We see that the spike in the previous task was indeed a local phenomenon, as the test loss continued to decrease after it.

From looking at the rest of the plot, we see that the test loss is beginning to flatten out. Towards the end, we see the training loss decreasing further while the testing loss remains approximately flat. Thus, we are beginning to see some very slight overfitting towards the very end of this loss.

1.3.1 Task D

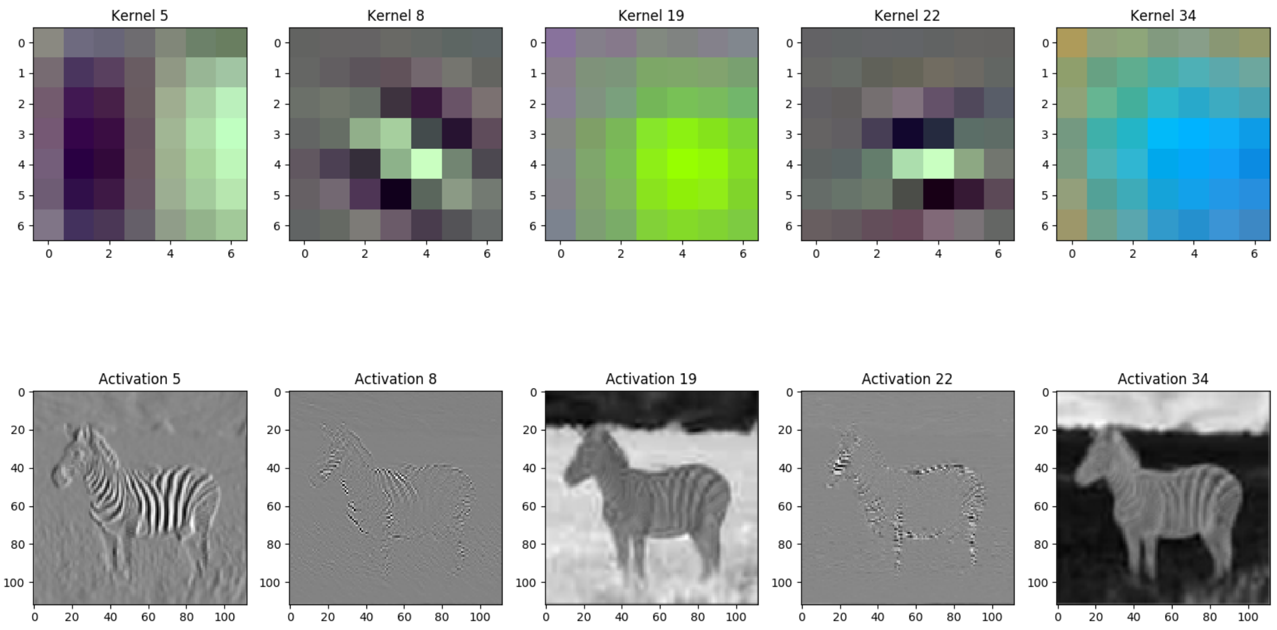

Figure 6: The resulting visualization of the kernels and activations.

1.3.2 Task E

In this task, we describe the filters shown in the task above.

Kernel 5 is a vertical edge detection kernel. It detects rapid changes in color from a dark color to a lighter color. As we can see from the activation the zebra has clear vertical stripes. The sky is indistinguishable as a result from the fact that the kernel only looks for vertical changes, not horizontally.

Kernel 8 is a diagonal line detector. It works much in the same way as kernel 5 but instead of looking for vertical edges it detects diagonal edges from top left to bottom right. However, it looks for thinner line than kernel 5. Additionally, due to the presence of an extra dark line, it looks for rapid changes going from darker colors, to lighter colors, and back to dark colors again.

Kernel 19 detects changes in color over a larger area. Looking at the activation it detects the grass-plain, and the contrast with the sky. It also detects the zebra but does not perform nearly as well on the zebra-stripes as the vertical edge detection in kernel 5.

Kernel 22 is a horizontal line detection kernel. It works in a similar way to kernel 8. It detects thin lines of rapid change of color in the horizontal space. In the activation we can see that it detects the vertical stripes of the zebra (mainly found on the legs and face). As with kernel 8 it does not however detect the change from the ground to sky as this is a large edge, not a thin line.

Kernel 34 is similar to kernel 19, just with different colors. The activation produced looks like (although not exactly) the negative to the activation of kernel 19.

2 Theory

2.1 Task A

To convolve using the convolution theorem, we first transform both the kernel and the image into the frequency domain. If we perform pointwise multiplication on these two frequency domain images, we get the frequency domain representation of the convolution. To get the convolution, we can perform the inverse Fourier transformation. Mathematically, this can be represented as

$$\mathscr{F}(\mathscr{F} * g) = \mathscr{F}(f) \cdot \mathscr{F}(g)$$

$$\mathscr{F}^{-1}[\mathscr{F}(f * g)] = f * g = \mathscr{F}^{-1}[\mathscr{F}(f) \cdot \mathscr{F}(g)]$$

Note that for pointwise multiplication to make sense, the kernel and image must be the same size. Thus, we might need to pad the kernel so that it is the same size as the image.

2.2 Task B

Low-pass filters are filters which pass frequencies lower than some threshold, while eliminating all frequencies higher than the threshold. High-pass filters work similarly: they pass all frequencies higher than some threshold, while eliminating all frequencies lower than the threshold.

2.3 Task C

Kernel A is a high-pass filter. Since convolution in the frequency domain is reduced to pointwise kernel, this kernel filters out the low frequencies in the middle, and passes high frequencies at the top and bottom of the image. However, some of the high frequencies are filtered as well. This could for example correspond to a vertical edge filter of sorts.

Kernel B is a high-pass filter. This kernel filters out the low frequencies in the middle and passes the higher frequencies outside of the middle.

Kernel C is a low-pass filter. The frequencies in the middle are passed on. Higher frequencies are weighted less and less, until they eventually are not passed on at all.

3 Programming

3.1 Task A

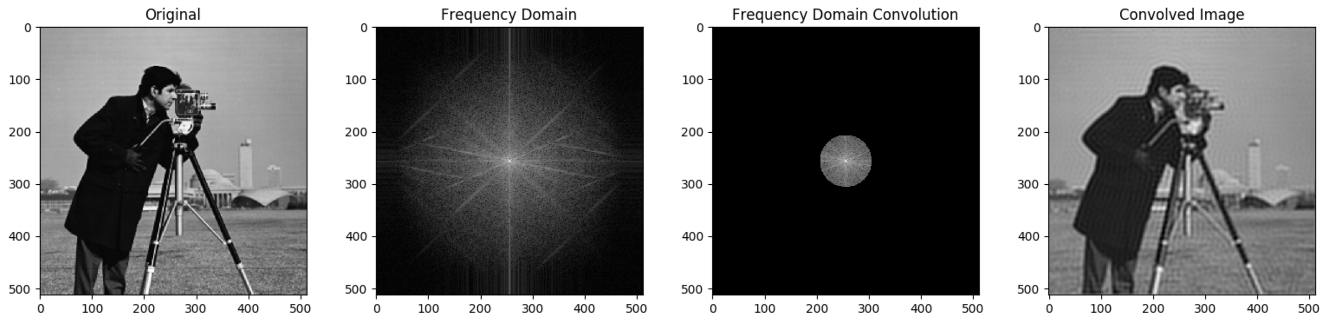

Figure 7: Results from applying the low-pass filter. As we can see from the frequency domain convolution, low frequencies are kept.

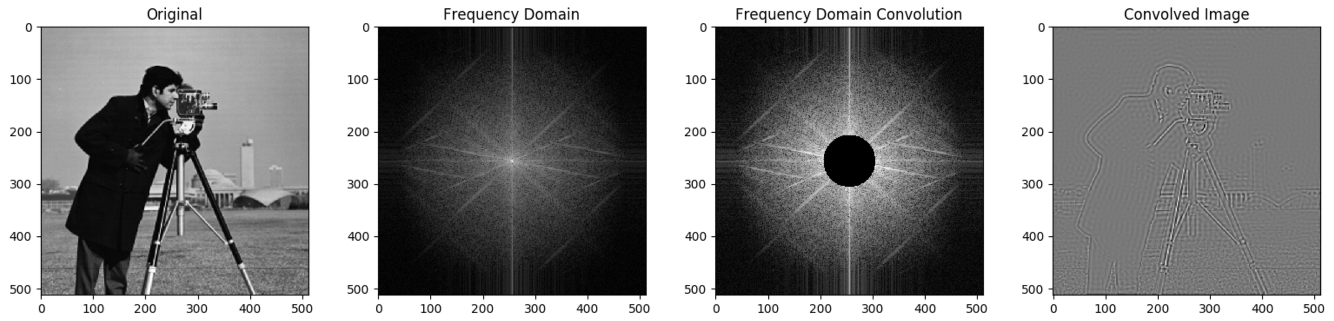

Figure 8: Results from applying the high-pass filter. As we can see from the frequency domain convolution, high frequencies are kept.

The frequency domain images here have all been scaled according to the formula $\log(1 + \mathscr{F}(x))$, where $\mathscr{F}(x)$ is the original frequency domain image. This scaling formula yields the same results as in the lecture slides, which is why we chose it.

3.2 Task B

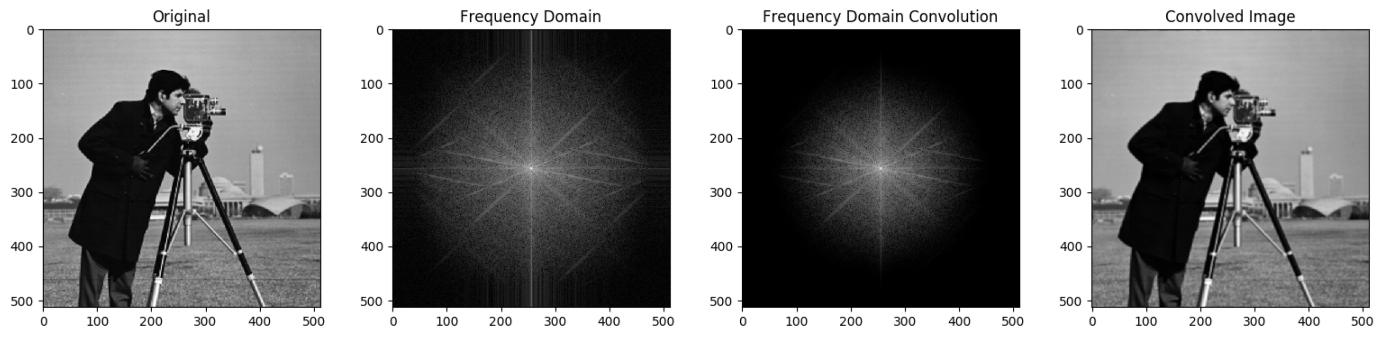

Figure 9: Camera man convolved with a Gaussian kernel.

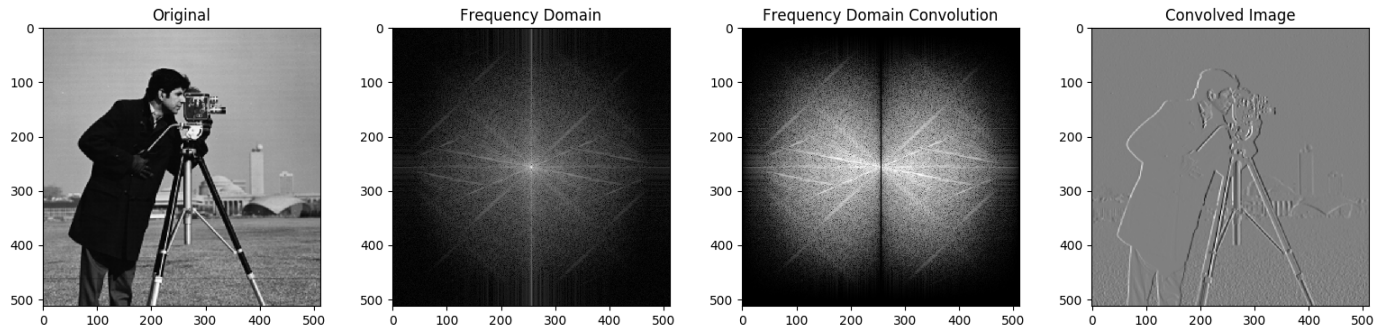

Figure 10: Camera man convolved with the horizontal Sobel kernel.

3.3 Task C

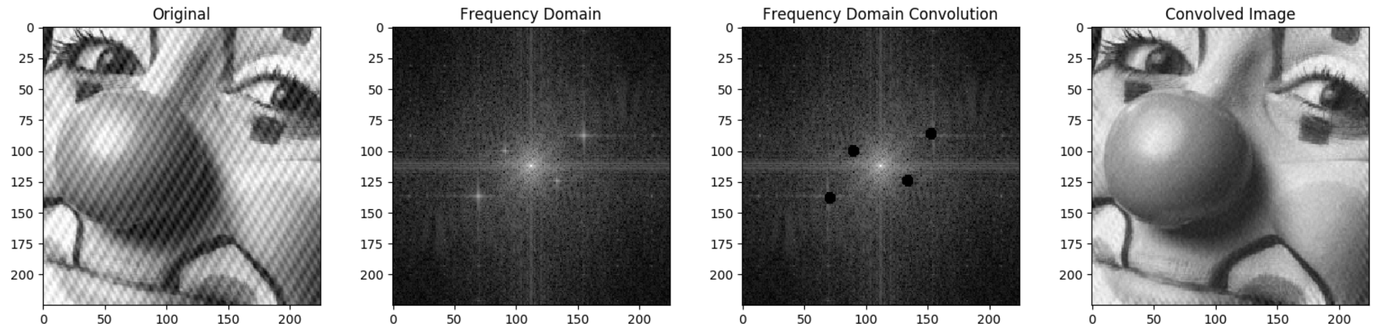

Figure 11: Clown convolved with the defined kernel.

This filter is called a notch filter. It removes specific frequencies from the image. As we can see in the images above, the filter has removed 4 areas in the frequency domain image which correspond to the periodic noise in the original image.

3.4 Task D

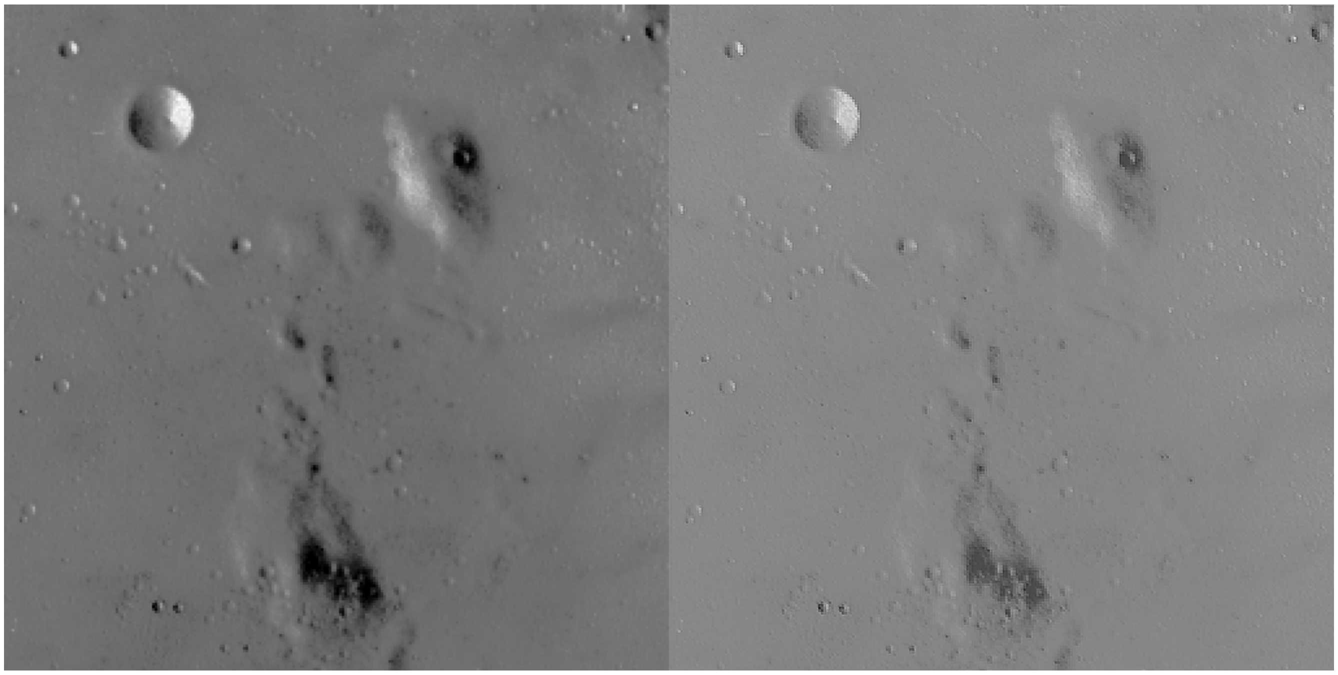

Figure 12: Sharpened moon image using the Laplacian operator.