Norwegian University of Science and Technology

TDT4195 : Visual Computing Fundamentals

Image Processing- Assignment 2

# Convolutional Neural Networks & Filtering in the Frequency Domain

*Group members*
Azad Md Abulkalam
Khan Md Ashiqul alam

*Group number:* 170

November 11, 2022 (Autumn 2022)

Comments:

https://www.ntnu.edu/studies/courses/TDT4195/2022/

NTNU
Norwegian University of
Science and Technology

## Task 1: Theory [1 pt]

### 1a) Determine padding [0.1pt]

To keep the output shape (Height×Width) of a convolutional layer to be equal to the input image, we can rearrange the provided equation for the asking padding as below.

$$P = \frac{(W_{out} - 1) * S - W_{in} + F}{2}$$

For example, we have an input image of shape 32x32, the given values (stride of 1, kernel size of $5 \times 5$, and 6 filters). Therefore, the padding should be,

$$P = \frac{(32 - 1) * 1 - 32 + 5}{2} = 2$$

So padding as 2 should be uses on each side.

### 1b) Determine Kernel size [0.2pt]

Again, we can rearrange the given equation for kernel height or width (as it is square shape) as follows,

$$F = (W_{in} + 2 * P - (W_{out} - 1) * S$$

Therefore, according to the given values (input as 512x512, feature map as 504x504, padding as 0 and stride as 1),

$$F = (512 + 2 * 0 - (504 - 1) * 1 = 9$$

So the kernel size is (Height) × (Width) = (9) x (9).

### 1c) Determine feature map size after pooling [0.2pt]

The equation for sub-sampling layer would be same as the given one except there is no padding value (means P=0).

$$W_{out} = (W_{in} - F)/S + 1$$

Therefore, according to the given values (input as 504x504, stride as 2, and filter as 2),

$$W_{out} = (504 - 2)/2 + 1 = 252$$

So the feature map size after the pooling layer is (Height) × (Width) = (252) x (252).

### 1d) Determine feature map size after second convolutional layer [0.2pt]

Based on the equation and given values from previous layers (input as 252x252, kernel as 3x3, padding as 0, and stride as 1), feature map size would be,

$$W_{out} = (W_{in} - F + 2 * P)/S + 1$$

$$W_{out} = (252 - 3 + 2 * 0)/1 + 1 = 250$$

So the feature map size after the second conv layer is (Height) × (Width) = (250) x (250).

### 1e) Number of parameters [0.3pt]

First conv layer (assume the number of input channel is 1),

$$No.\ of\ parameters = F_H * F_W * C_1 * C_2 + C_2 = 5 * 5 * 1 * 32 + 32 = 832$$

Second conv layer (number of input channel is 32),

$$No.\ of\ parameters = F_H * F_W * C_2 * C_3 + C_3 = 3 * 3 * 32 * 64 + 64 = 18496$$

Third conv layer (number of input channel is 64),

$$No.\ of\ parameters = F_H * F_W * C_3 * C_4 + C_4 = 3 * 3 * 64 * 128 + 128 = 73856$$

For the flatten layer, we need to compute the current feature map size from the beginning.

Input (32x32x1) $\rightarrow (1st\ conv) \rightarrow (32x32x32) \rightarrow (subsampling) \rightarrow (16x16x32) \rightarrow (2nd\ conv) \rightarrow (16x16x64) \rightarrow (subsampling) \rightarrow (8x8x64) \rightarrow (3rd\ conv) \rightarrow (8x8x128) \rightarrow (subsampling) \rightarrow (4x4x128) \rightarrow (flatten) \rightarrow (2048)$.

1st fully connected layer,

$$No.\ of\ parameters = no\ of\ inputs * no\ of\ neurons + no\ of\ neurons = 2048 * 64 + 64 = 131136$$

2nd fully connected (output) layer,

$$No.\ of\ parameters = no\ of\ inputs * no\ of\ neurons + no\ of\ neurons = 64 * 10 + 10 = 650$$

Therefore, the total number of parameters of the network,

$$832 + 18496 + 73856 + 131136 + 650 = 224970$$

## Task 2: Programming [3 points]

### 2a) Implementation of the given CNN [1.0pt]

The given CNN model is implemented using the task2.ipynb file. The model is trained for the given 5 epochs and therefore we got the final validation accuracy of 0.9876. Below you can see the training and validation loss during training.

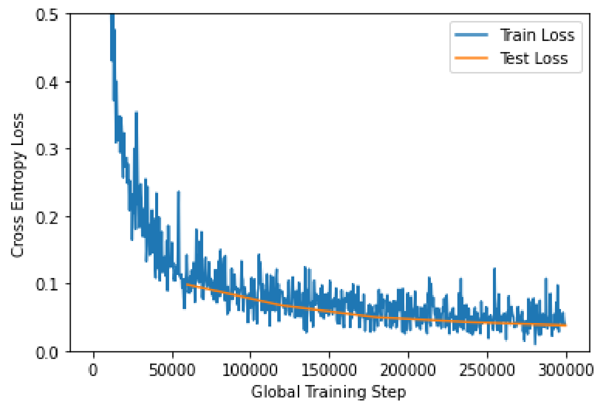

Figure 1: Train and test losses during the training steps.

To understand the overfitting of the model, we run the model on our train dataset and observe the loss (0.037672009672854646) and the accuracy (0.9876). It indicates that the model performs equally well on both train and test datasets which refers that the model is not overfitted. However, we could have tested this overfitting deeply if we have more unseen data.

## 2b) CNN model with Adam optimizer [0.5pt]

We change the optimizer from SGD to Adam along with the learning rate as 0.001. Afterwards, we train the model from scratch for 5 epochs. See the diagram below for train/validation loss of the both models together.

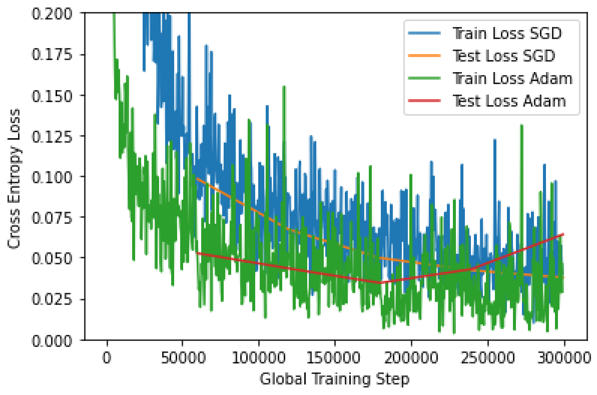

Figure 2: Train and test losses during the training steps of the both models.

## 2c) Visualize filters and activations [0.8pt]

We implement the solution using task2c.ipynb file. The visualization of filters and activations of the corresponding indices ([5,8,19,22,34]) are included in the below diagram.

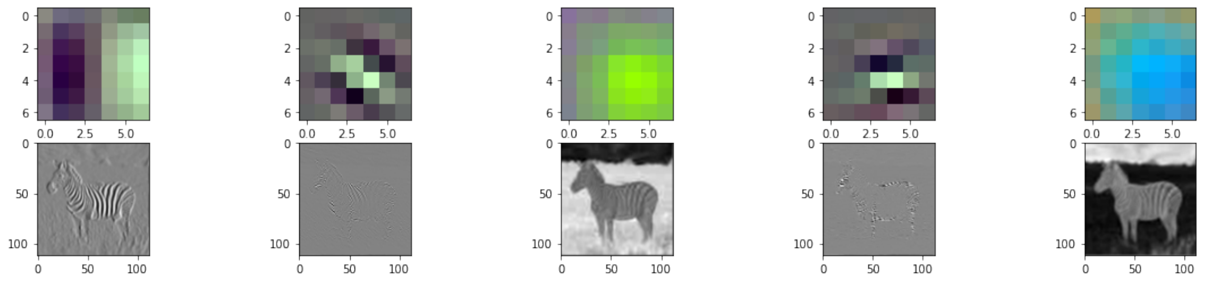

Figure 3: Visualization of filters (weights) and activations from the first layer of the pretrained ResNet50 model.

## 2d) Determine the features [0.7pt]

In CNNs, filters are not defined but the value of each filter is learned during the training process. Therefore, we have to assume the filter type and features it extracts by looking at the corresponding filter and its activation output. By looking at the first filter and its activation, seems like a sobel filter extracts the edges well from the original image as the edges are clearly visible. For second and fourth filters and activations, it is hard to assume which kind of features they extracted but more or less edge detection as well based on image gradients. The 3rd and the last are segmentation where the zebra was one segment on both but the background was segmented into two different segmentations. First, the ground part of the background is colored completely white and the sky is black and in the latter they are interchanged but zebra remains the same. In summery, it would be contrast separability.

## Task 3: Theory [1 point]

### 3(a)

Lets give the reason first before pairing:

- The dots will be in the direction where there is periodic change of pixels
- The dots will be further away with lower time period(higher number of lines)

Based on that:

| Spatial Domain | Corresponding Frequency Domain |
|---|---|
| 1a | 2e |
| 1b | 2c |
| 1c | 2f |
| 1d | 2b |
| 1e | 2d |
| 1f | 2a |

### 3(b)[0.1pt] What are high-pass and low-pass filters?

- High-pass Filter: High pass filters eliminates any frequency that are lower than a threshold value. That means it only keeps higher frequency than a given threshold frequency.
- Low-pass Filter: Low pass filters eliminates any frequency that are higher than a threshold value. That means it only keeps lower frequency than a given threshold frequency.

### 3(c) [0.3pt] The amplitude $|F\{g\}|$ of two commonly used convolution kernels can be seen in Figure 4. For each kernel (a, and b), figure out what kind of kernel it is (high- or low-pass). Shortly explain your reasoning.

**Solution:** We know the the FFT image is center shifted. That means low frequencies are at the middle and higher frequencies are further away from center.

- In figure, 4a the filter has zero value around the center. So no low frequency can pass. That means its a High-pass filter.
- In figure, 4b the filter has zero value outside the periphery of the inner white circle at the center. So only low frequencies around the center and inside the circle can pass. That means its a Low-pass filter.

We can see for Figure 4a the filter has zero value around the center

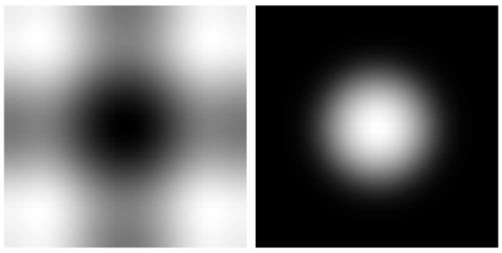
(a) (b)

Figure 4: (a)High-pass Filter, (b)Low-pass Filter

## Task 4: Programming [3 points]

**4(a) [0.8pt] Implement a function that takes an grayscale image, and a kernel in the frequency domain, and applies the convolution theorem (seen in Equation 4). Try it out on a low-pass filter and a high-pass filter on the grayscale image "camera man"(im = skimage.data.camera()). Include in your report the filtered images and the before/after amplitude $|F\{f\}|$ of the transform. Make sure to shift the zero-frequency component to the center before displaying the amplitude. Implement this in the function convolve_im in task4a.py/task4a.ipynb. The high-pass and low-pass filter is already defined in the starter code.**

**Sulotion:** The code is with comments to understand the process. Thus I will not describe every steps here. But for the ringing effect with low pass filter:
The ringing effect is created due to the hard edges of the low pass filter. To realize hard edges it requires a huge amount of sinusoidal(infinite). Thus the ringing effect is present in reconstructed image as hard edges need large number of summation of this.

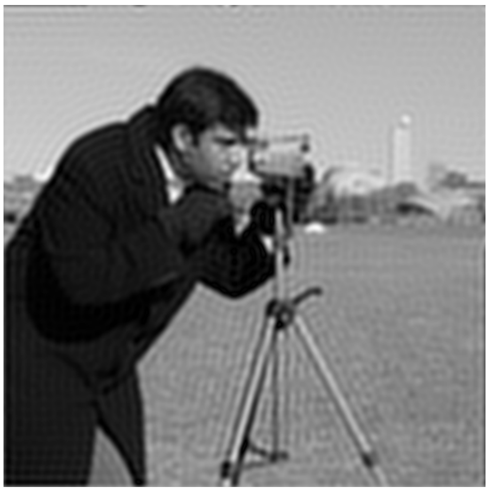
Figure 5: Low-pass filtering ringing effect

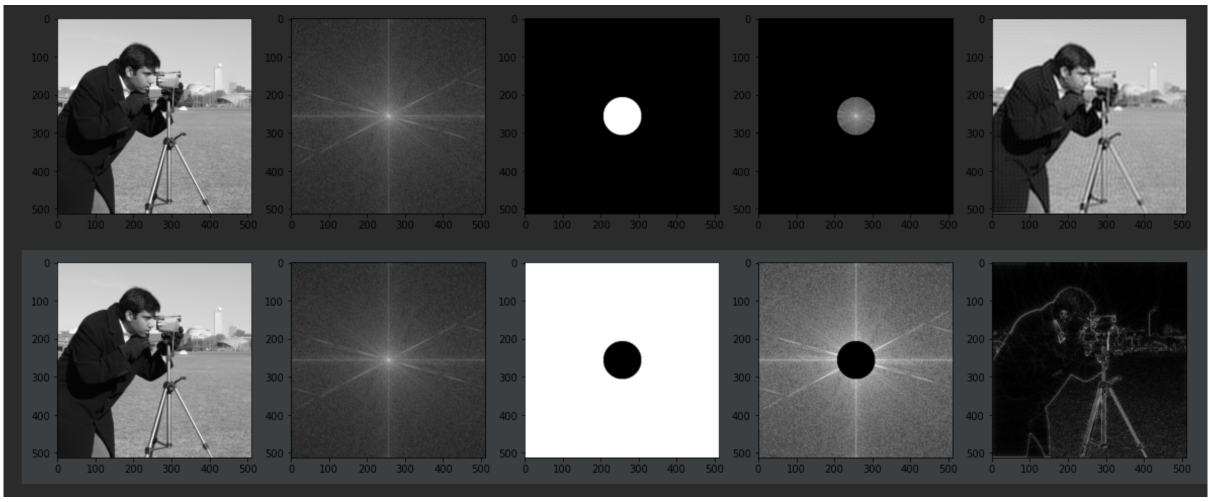

Figure 6: Top image: Low-pass filtering, Bottom image: High-pass filtering

**4(b) [0.4pt] Implement a function that takes an grayscale image, and a kernel in the spatial domain, and applies the convolution theorem. Try it out on the gaussian kernel given in assignment 1, and a horizontal sobel filter (Gx). Include in your report the filtered images and the before/after amplitude $|F\{f\}|$ of the transform. Make sure to shift the zero-frequency component to the center before displaying the amplitude. Implement this in the function convolve_im in task4b.py/task4b.ipynb. The gaussian and sobel filter are already defined in the starter code.**

Here we have the whole process of filtering through image visualization: The edge detected image we have used our own complex imshow function. Thus it is not gray rather black. But the edges can be clearly seen in black and white also. And the Gaussian filtering smooths the image

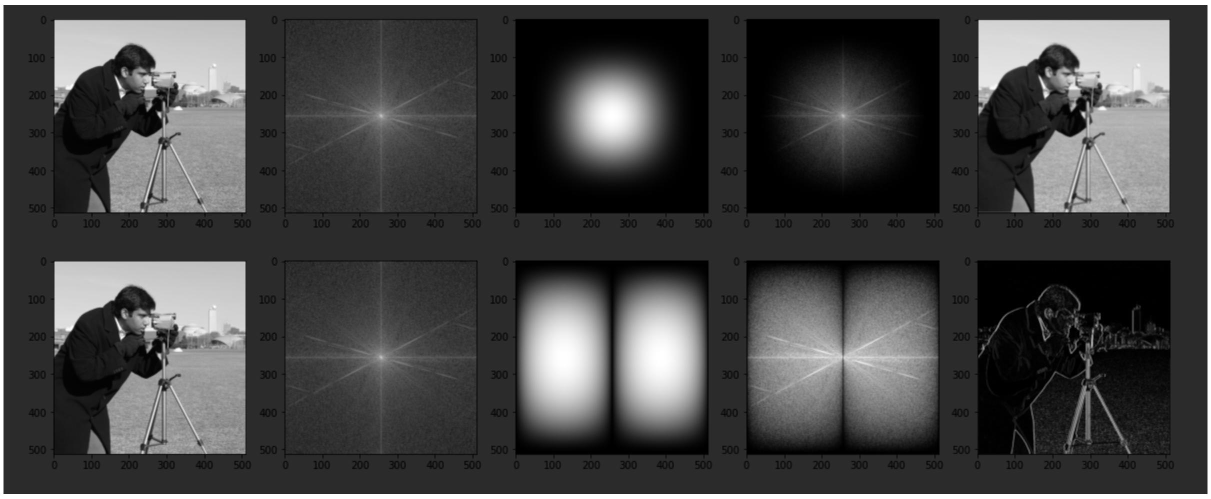

Figure 7: Top image: Gaussian filtering, Bottom image: Sobel x filtering [Left to right: input_image, fft_image, fft_kernel, fft_conv output_image]

**4(c) [1.0pt] Use what you've learned from the lectures and the recommended resources to remove the noise in the image seen in Figure 5a. Note that the noise is a periodic signal. Also, the result you should expect can be seen in Figure 5b Include the filtered result in your report. Implement this in the file task4c.py/task4c.ipynb. Hint: Try to inspect the image in the frequency domain and see if you see any abnormal spikes that might be the noise.**

First lets see the FFT of moon(centered):

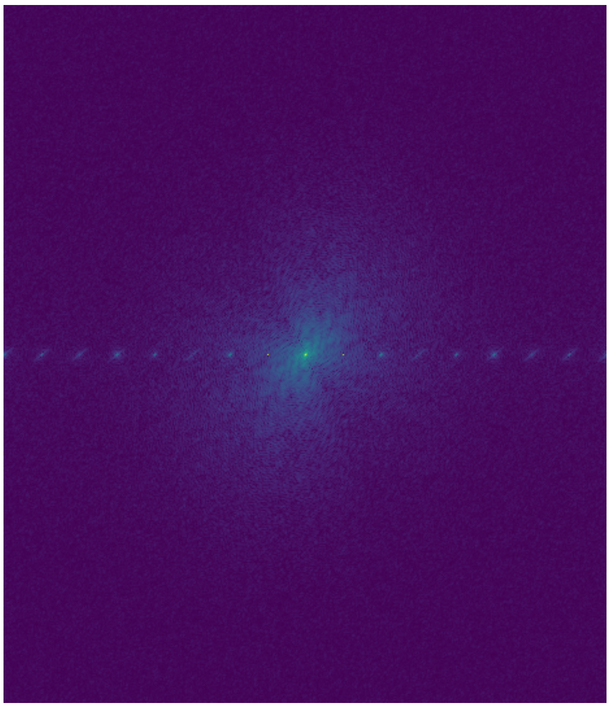

Figure 8: Center shifted FFT of the unfiltered Moon

We can see there are some spikes in horizontal direction. Now we will remove these bright spots except the center one. The algorithm of doing this and creating the filter is well commented in the code. Thus will not discuss it here.

So the filter(all pass except those bright spots) looks like:

Figure 9: FFT filter for the Moon

After applying convolution and inverse fft, the filtered moon looks like in spatial domain:

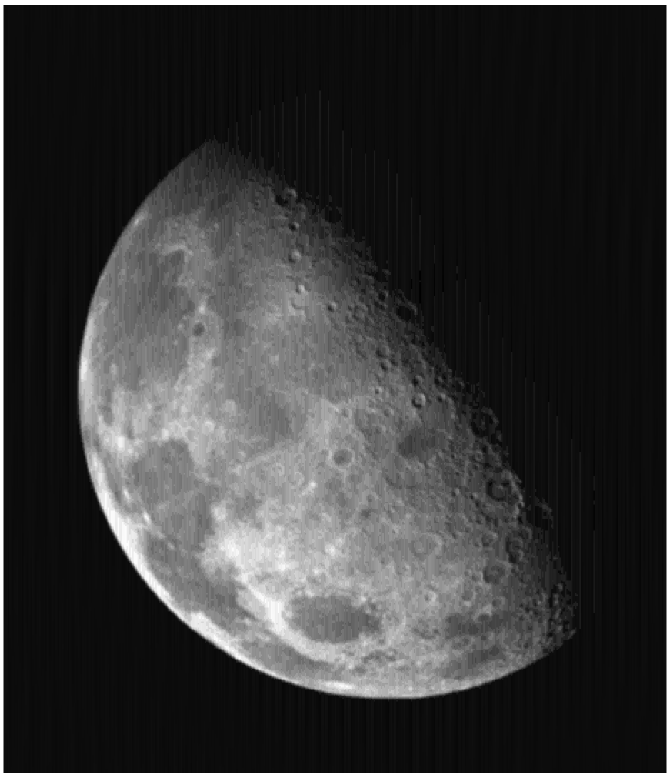

Figure 10: Filtered Moon

**4(d) [0.8pt] Now we will create a function to automatically find the rotation of scanned documents, such that we can align the text along the horizontal axis. You will use the frequency domain to extract a binary image which draws a rough line describing the rotation of each document. From this, we can use a hough transform to find a straight line intersecting most of the points in the binary image. When we have this line, we can easily find the rotation of the line and the document. Your task is to generate the binary image by using the frequency spectrum. See Figure 6 which shows you what to expect. We've implemented most of the code for you in this task; you only need to alter the function create_binary_image in task4d.py/task4.ipynb. Include the generated image in your report (similar to Figure 6). Hint: You can use a thresholding function to threshold the magnitude of the frequency domain to find your binary image (it's OK to hardcode the thresholding value).**

For this one: We created the binary image with hard coded threshold value of 200. This value was achieved though observing the min max value of the fft and a bit of trial and error.All the steps for creating the binary image is commented in the code.

The process and final output looks like:

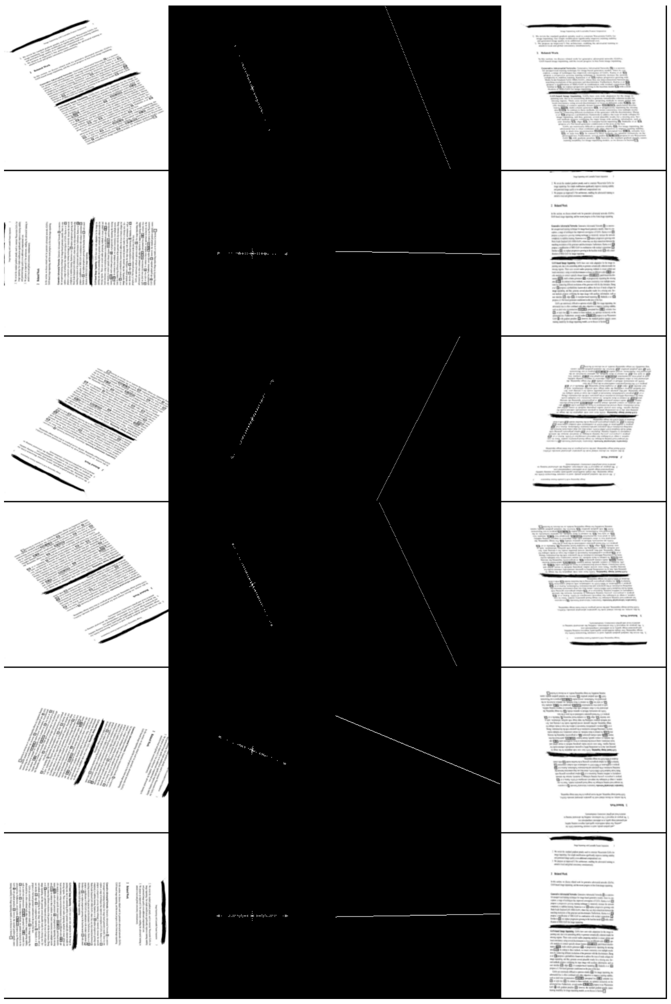

Figure 11: Text alignment