TDT4195 Visual Computing Fundamentals, Image Processing Assignment 2

Task 1

a)

Given a single convolutional layer with a stride of 1, kernel size of 7×7, and 6 filters. If I want the output shape (Height×Width) of the convolutional layer to be equal to the input image, how much padding should I use on each side?

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

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

$$F_W = F_H = 7 \tag{3a}$$
$$S_W = S_H = 1 \tag{3b}$$

$$W_2 = W_1 = \frac{W_1 - F_W + 2P_W}{S_W} + 1 \tag{4a}$$
$$P_W = \frac{W_1 \cdot S_W - W_1 + F_W - 1}{2}, \quad S_W = 1, F_W = 7 \tag{4b}$$
$$P_W = \frac{W_1 - W_1 + 7 - 1}{2} \tag{4c}$$
$$P_W = 3 \tag{4d}$$

Since $F_W = F_H = 7$ and $S_W = S_H = 1$, $P_W = P_H = 3$ s.t. a padding of 3 on both sides on each dimension will give an output of the same shape as the input image.

---

b)

You are told that the spatial dimensions of the feature maps in the first layer are 506×506, and that there are 12 feature maps in the first layer. Assuming that no padding is used, the stride is 1, and the kernel used are square, and of an odd size, what are the spatial dimensions of these kernels? Give the answer as (Height) $\times$ (Width).

$$W_1 = H_1 = 512 \tag{5a}$$
$$W_2 = H_2 = 506 \tag{5b}$$
$$P_W = P_H = 0 \tag{5c}$$
$$S_W = S_H = 1 \tag{5d}$$
$$F_W = F_H \tag{5e}$$

Equation 1 gives:

$$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1 \tag{6a}$$
$$F_W = W_1 + 2P_W + 1 - W_2 \cdot S_W \tag{6b}$$
$$= 512 + 2 \cdot (0) + 1 - 506 \cdot 1 \tag{6c}$$
$$= 7 \tag{6d}$$

Because $W_1 = H_1$, $W_2 = H_2$, $P_W = P_H$, $S_W = S_H$ and $F_W = F_H = 7$, the size of the kernel is 7×7.

---

c)

If subsampling is done using neighborhoods of size 2×2, with a stride of 2, what are the spatial dimensions of the pooled feature maps in the first layer? (assume the input has a shape of 506×506). Give the answer as (Height) $\times$ (Width).

$$W_2 = \frac{W_1 - F_W}{S_W} + 1 \tag{7a}$$
$$= \frac{506 - 2}{2} + 1 \tag{7b}$$
$$= 253 \tag{7c}$$

$W_2 = H_2$ s.t. the dimensions of the pooled feature maps are 253×253.

---

d)

The spatial dimensions of the convolution kernels in the second layer are 3×3. Assuming no padding and a stride of 1, what are the sizes of the feature maps in the second layer? (assume the input shape is the answer from the last task). Give the answer as (Height) $\times$ (Width).

$$W_1 = H_1 = 253 \tag{8a}$$
$$P_W = P_H = 0 \tag{8b}$$
$$S_W = S_H = 1 \tag{8c}$$
$$F_W = F_H = 3 \tag{8d}$$

$$W_2 = \frac{W_1 - F_W + 2P_W}{S_W} + 1 \tag{9a}$$
$$= 253 - 3 + 1 \tag{9b}$$
$$= 251 \tag{9c}$$

$H_2 = W_2$ giving a feature map shape of 251×251.

---

e)

Table 1 shows a simple CNN. How many parameters are there in the network? In this network, the number of parameters is the number of weights + the number of biases. Assume the network takes in an 32×32 image.

For each convolutional layer the number of parameters are

$$n_{p,i} = F_H \cdot F_W \cdot C_1 \cdot C_2 + C_2 \tag{10}$$

where i refers to the layer.

For each fully-connected layer the number of parameters are

$$n_{p,i} = C_1 \cdot C_2 + C_2 \tag{11}$$

where i refers to the layer.

Layer 1

The layer has 32 filters each with a 5×5 kernel s.t. $C_2 = 32$ and $F_H = F_W = 5$.

$$n_{p,1} = 5 \cdot 5 \cdot 1 \cdot 32 + 32 = 832 \tag{12}$$

Layer 2

$$C_1 = 32 \tag{13a}$$
$$F_H = F_W = 3 \tag{13b}$$
$$C_2 = 64 \tag{13c}$$

$$n_{p,2} = 3 \cdot 3 \cdot 32 \cdot 64 + 64 = 18496 \tag{14}$$

Layer 3

$$C_1 = 64 \tag{15a}$$
$$F_H = F_W = 3 \tag{15b}$$
$$C_2 = 128 \tag{15c}$$

$$n_{p,3} = 3 \cdot 3 \cdot 64 \cdot 128 + 128 = 73856 \tag{16}$$

Layer 4

Layer 4 is a fully-connected layer.

$$C_1 = 128 \tag{17a}$$
$$C_2 = 64 \tag{17b}$$

$$n_{p,4} = 128 \cdot 64 + 64 = 8256 \tag{18}$$

Layer 5

Layer 5 is a fully-connected layer.

$$C_1 = 64 \tag{19a}$$
$$C_2 = 10 \tag{19b}$$

$$n_{p,5} = 64 \cdot 10 + 10 = 650 \tag{20}$$

All layers

The total number of parameters is then

$$n_p = n_{p,1} + n_{p,2} + n_{p,3} + n_{p,4} + n_{p,5} \tag{21a}$$
$$n_p = 832 + 18496 + 73856 + 8256 + 650 \tag{21b}$$
$$n_p = 102090 \tag{21c}$$

Task 2

a)

Implement the network in Table 1. Report the final accuracy on the validation set for the trained network. Include a plot of the training and validation loss during training.
By looking at the final train/validation loss/accuracy, do you see any evidence of overfitting? Shortly summarize your reasoning.
`torch.nn.Linear` takes in an `in_features` parameter. Therefore the shape of the output of layer 3 needs to be calculated. The width and height of each output is the same. This yields:

Output layer 1

Feature map output size:

$$W = \frac{32 - 5 + 2 \cdot 2}{1} + 1 = 32 \tag{22}$$

Pooled feature map output size:

$$W = \frac{32 - 2}{2} + 1 = 16 \tag{23}$$

Output layer 2

Feature map output size:

$$W = \frac{16 - 3 + 2 \cdot 1}{1} + 1 = 16 \tag{24}$$

Pooled feature map output size:

$$W = \frac{16 - 2}{2} + 1 = 8 \tag{25}$$

Output layer 3

Feature map output size:

$$W = \frac{8 - 3 + 2 \cdot 1}{1} + 1 = 8 \tag{26}$$

Pooled feature map output size:

$$W = \frac{8 - 2}{2} + 1 = 4 \tag{27}$$

This gives an output shape of 4×4. The output of layer 3 is then sent through `torch.nn.Flatten`. Layer 3 has 128 channels giving the input argument `in_features=128*4*4` to `torch.nn.Linear`.

The final test loss is at 0.038 and the final test accuracy is at 0.988. With a low test loss and a high accuracy the model seems to perform well on new data which points to good generalisation. This does thus not show any signs of overfitting.

Listing 1: Creation of network model to be trained

```
    def create_model():
    """
        Initializes the mode. Edit the code below if you would ...
            like to change the model.
    """
    model = nn.Sequential(
        nn.Conv2d(in_channels=1, out_channels=32,
                  kernel_size=5, stride=1, padding=2),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        #
        nn.Conv2d(in_channels=32, out_channels=64,
                  kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        #
        nn.Conv2d(in_channels=64, out_channels=128,
                  kernel_size=3, stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2, stride=2),
        #
        nn.Flatten(),
        nn.Linear(128*4*4, 64), # Output shape of last
                                # MaxPool2d is 4x4 with 128 channels
        nn.ReLU(),
        nn.Linear(64, 10),
        # No need to include softmax, as this is already ...
            combined in the loss function
    )
    # Transfer model to GPU memory if a GPU is available
    model = utils.to_cuda(model)
    return model
```

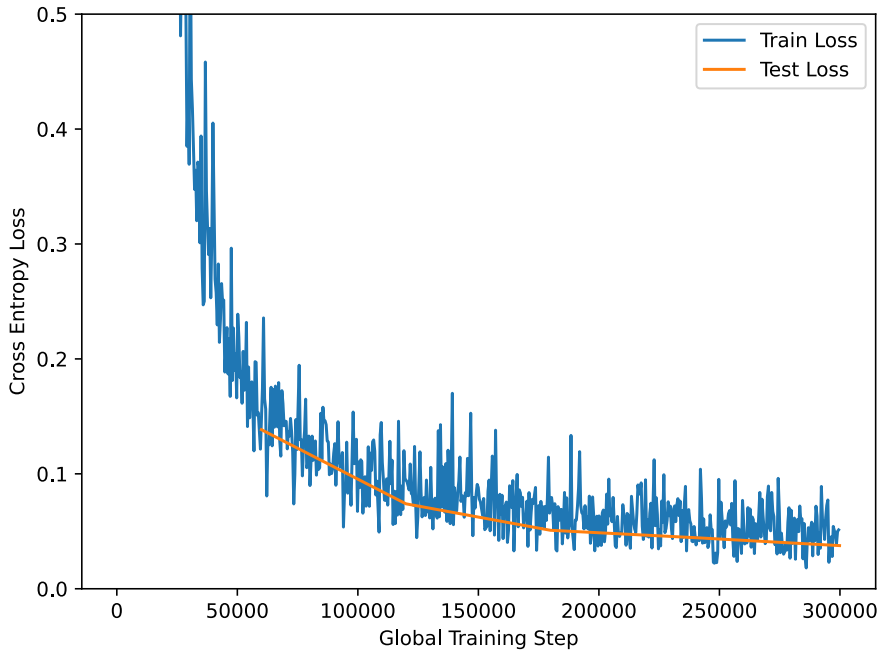


Figure 1: Plot of training and validation loss for the neural network

---

b)

Change the SGD optimizer to Adam (use `torch.optim.Adam` instead of `torch.optim.SGD`), and train your model from scratch.
Use a learning rate of 0.001.
Plot the training/validation loss from both models (the model with Adam and the one with SGD) in the same graph and include this in your report.
To train the networks, both the same learning rate of 0.001 for the two optimisation algorithms and a learning rate of 0.02 for SGD and a learning rate of 0.001 for Adam was used.

Using the same learning rate of 0.001 for both optimisation algorithms gave a final test loss of 2.08 and a test accuracy of 0.530 with SGD and a final test loss of 0.0287 and a test accuracy of 0.990 with Adam.

Using a learning rate of 0.02 for SGD and a learning rate of 0.001 for Adam gave a final test loss of 0.0375 and a test accuracy of 0.988 with `torch.optim.SGD` and a final test loss of 0.0287 and a test accuracy of 0.990 with `torch.optim.Adam`.

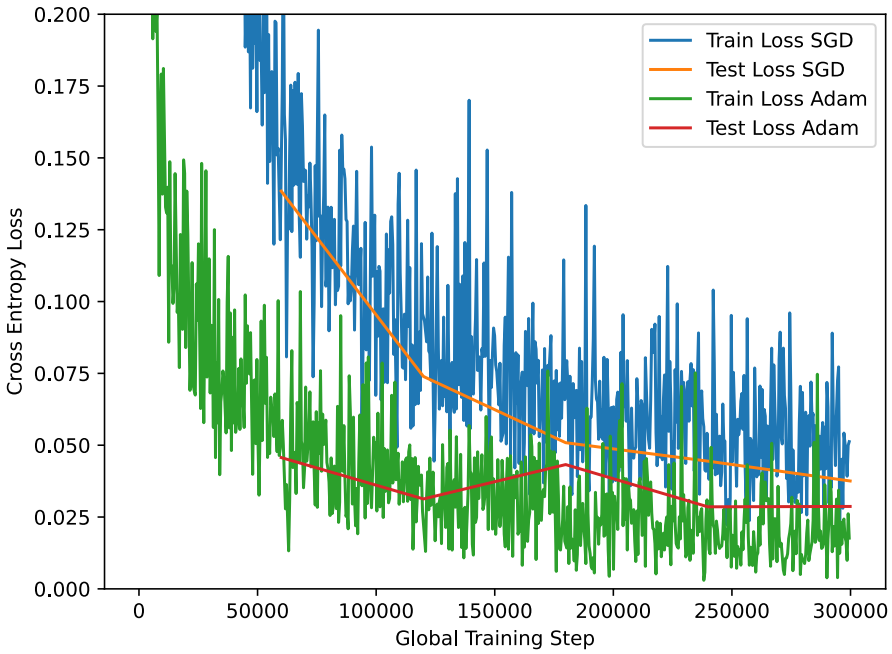


Figure 2: Plot of training and validation loss using a learning rate of 0.02 for the Stochastic Gradient Decent optimiser and a learning rate of 0.001 for the Adam optimiser

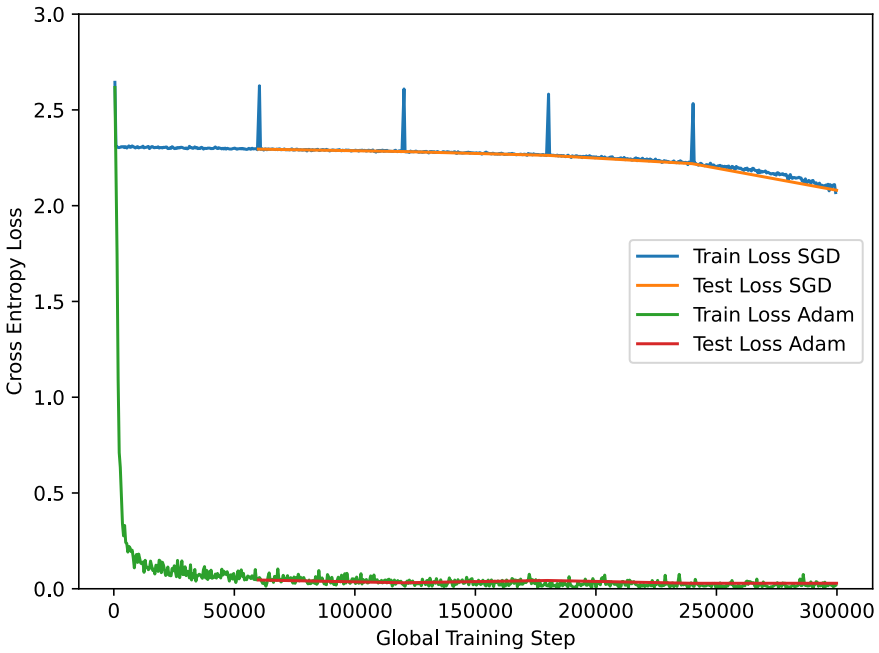


Figure 3: Plot of training and validation loss using a learning rate of 0.001 for both the Stochastic Gradient Decent optimiser and the Adam optimiser

---

c)

Run the image `zebra.jpg` through the first layer of the ResNet50 network. Visualize the filter, and the grayscale activation of a the filter, by plotting them side by side. Use the pre-trained network ResNet50 and visualize the convolution filters with indices $[5, 8, 19, 22, 34]$.
Include the visualized filters and activations in your report.

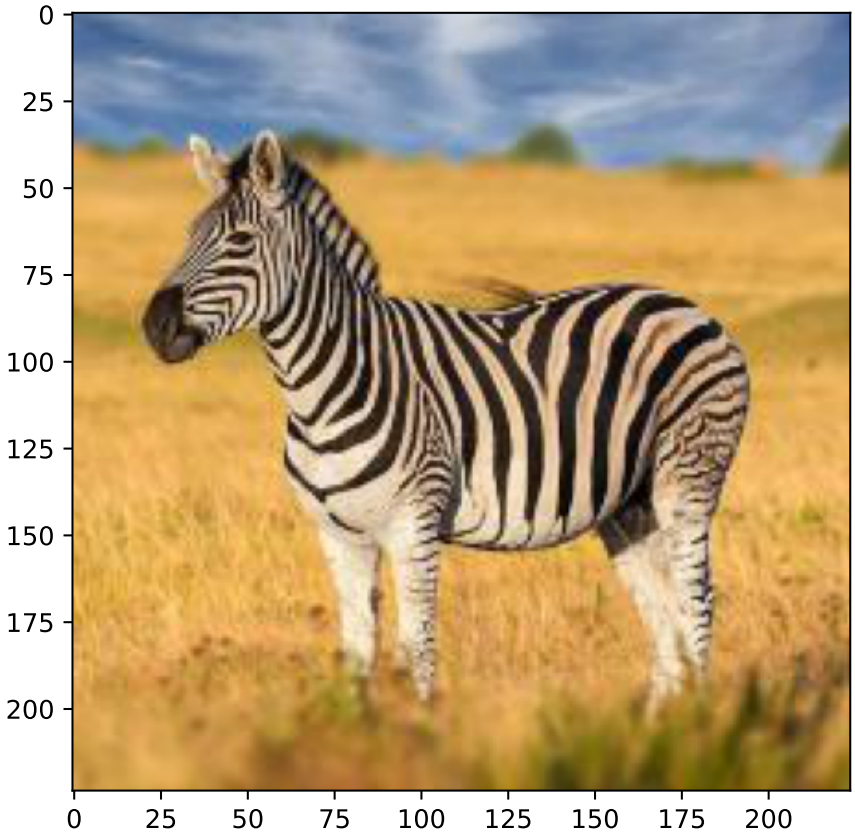


Figure 4: Image of zebra

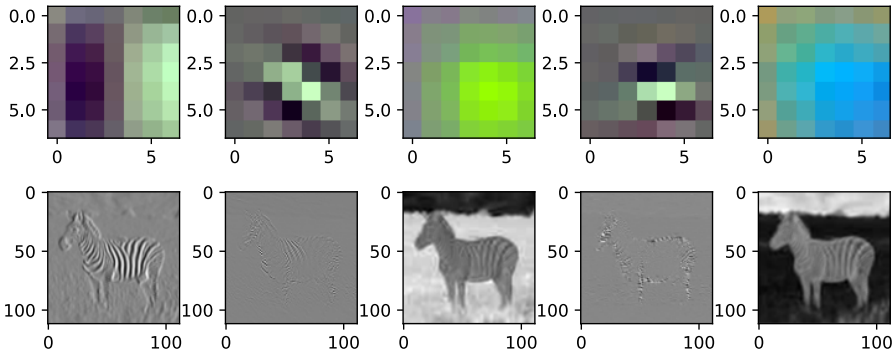


Figure 5: Filters and activations on the zebra image

d)

Looking at the visualized filter, and its corresponding activation on the zebra image, describe what kind of feature each filter extracts. Explain your reasoning.

The images of the filter kernels have been normalised between zero and one. This means that darker pixels represent negative or low values in the actual kernels. The pixels in the images of the activation will be brighter if the correlation between the kernels and the image is high.
The first kernel from the left resembles a sobel kernel. This seems to extract the vertical edges in the image.
The second kernel and the fourth kernel seem to extract edges slanted in opposite directions. Each pixel in the activation image is the result of the sum of the product of the kernel value and the pixels overlapping with the kernel. The diagonal of the second kernel is bright. If the pixel this kernel operates on also is bright while the pixels on the sub- and superdiagonal are dark, the resulting pixel in the image will be bright. This is because the products are mostly positive. An analogous explanation can be applied to kernel four.
The third kernel appears to extract the field around the zebra. The colours of the field in the image is mostly yellow and green. The kernel is mostly green/greenyellow which means the red and green channels have high values. Yellow colours have high red and green channel values. This means the kernel filters out most of what isn't the field.
The fifth kernel seems to extract the sky. The kernel is mostly blue with high blue channel values. It also extracts the white parts of the zebra because white clearly also contains high blue channel values.

Task 3

a)

Given the images in the spatial and frequency domain in Figure 3, pair each image in the spatial domain (first row) with a single image in the frequency domain (second row). Explain your reasoning.

Lower frequencies show up in the centre of the DFT-transformed image. Horizontal changes in the original image show up horizontally in the transformed image. Therefore the dots that are the furthest apart show the highest changes in the original image. This gives the relation given in Table 1.

1a	1b	1c	1e	1f	1d
2e	2c	2f	2d	2a	2b

Table 1: Relation between spatial domain images and frequency domain images. Pairs are stacked vertically.

---

b)

What are high-pass and low-pass filters?

High-pass filters allow high frequencies to pass, while low-pass filters allow low frequencies to pass. This means that the filters will either filter out high or low frequencies.

---

c)

The amplitude $|\mathcal{F}\{g\}|$ 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.

Kernel a is brighter in the parts of the image that correspond to higher frequencies, while kernel b is brighter in the parts that correspond to lower frequencies. Convolution is the same as multiplication in the frequency domain. As such the higher frequencies convolved with the kernel in b will be multiplied with lower numbers or zero. This means the output image will either have a lower amplitude in these regions or no amplitude at all. The opposite is true for the kernel a.
Therefore kernel a is a high-pass filter, while kernel b is a low-pass filter.

Task 4

a)

Implement a function that takes an grayscale image, and a kernel in the frequency domain, and applies the convolution theorem. 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 $|\mathcal{F}\{f\}|$ of the transform.
You will observe a "ringing" effect in the filtered image. What is the cause of this?

The ringing effect is due to removing high-frequency noise with a sharp low-pass filter. This leads to oscillations in the output image.

Listing 2: Convolution function

```
1   def convolve_im(im: np.array,
2                   fft_kernel: np.array,
3                   verbose=True):
4       """ Convolves the image (im) with the frequency kernel ...
            (fft_kernel),
5           and returns the resulting image.
6
7           "verbose" can be used for turning on/off visualization
8           convolution
9
10      Args:
11          im: np.array of shape [H, W]
12          fft_kernel: np.array of shape [H, W]
13          verbose: bool
14      Returns:
15          im: np.array of shape [H, W]
16      """
17      # START YOUR CODE HERE ### (You can change anything inside ...
            this block)
18
19      # Fourier transform image, shift the frequencies to centre
20      im_fft = np.fft.fft2(im)
21      im_fft = np.fft.fftshift(im_fft)
22
23      # Shift the frequencies of the kernel
24      fft_kernel = np.fft.fftshift(fft_kernel)
25
26
27      # Convolve the image with the kernel in the frequency domain
28      conv_result_fft = np.multiply(im_fft, fft_kernel)
```

```
    # Shift the frequencies back before inverse Fourier transforming
    conv_result_fft = np.fft.fftshift(conv_result_fft)
    conv_result = np.fft.ifft2(conv_result_fft).real
    # Shift frequencies s.t. zero-frequency is in centre again
    conv_result_fft = np.fft.fftshift(conv_result_fft)

    # Get magnitude of images in frequency domain for visualisation.
    im_fft          = np.log(np.abs(im_fft) + 1)
    fft_kernel      = np.log(np.abs(fft_kernel) + 1)
    conv_result_fft = np.log(np.abs(conv_result_fft) + 1)

    if verbose:
        # Use plt.subplot to place two or more images beside ...
            eachother
        plt.figure(figsize=(20, 4))
        # plt.subplot(num_rows, num_cols, position (1-indexed))

        plt.subplot(1, 5, 1)
        plt.imshow(im, cmap="gray")

        plt.subplot(1, 5, 2)
        # Visualize FFT
        plt.imshow(im_fft, cmap="gray")

        plt.subplot(1, 5, 3)
        # Visualize FFT kernel
        plt.imshow(fft_kernel, cmap="gray")

        plt.subplot(1, 5, 4)
        # Visualize filtered FFT image
        plt.imshow(conv_result_fft, cmap="gray")

        plt.subplot(1, 5, 5)
        # Visualize filtered spatial image
        plt.imshow(conv_result, cmap="gray")
    ### END YOUR CODE HERE ###
    return conv_result
```

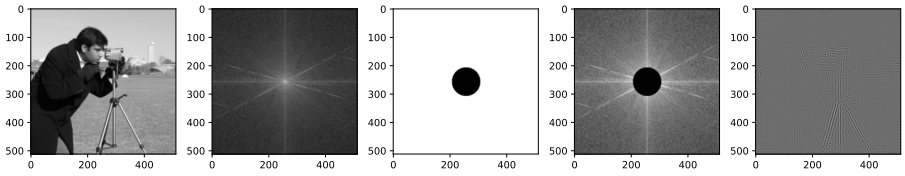


Figure 6: High-pass filter applied to *camera man*. The figure shows (from the left): the input image, Fourier transformed image, Fourier transformed high-pass kernel, convolution result in the frequency domain, convolution result in the spatial domain.

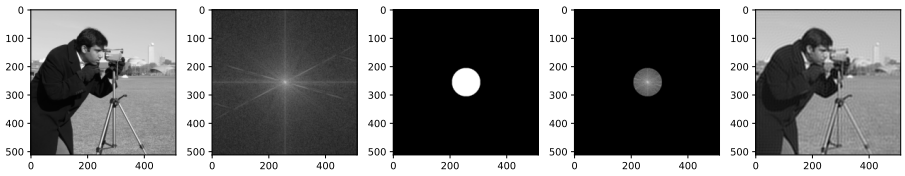


Figure 7: Low-pass filter applied to *camera man*. The figure shows (from the left): the input image, Fourier transformed image, Fourier transformed low-pass kernel, convolution result in the frequency domain, convolution result in the spatial domain.

---

b)

Implement a function that takes a 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 (G_x).
Include in your report the filtered images and the before/after amplitude $|\mathcal{F}\{f\}|$ of the transform.

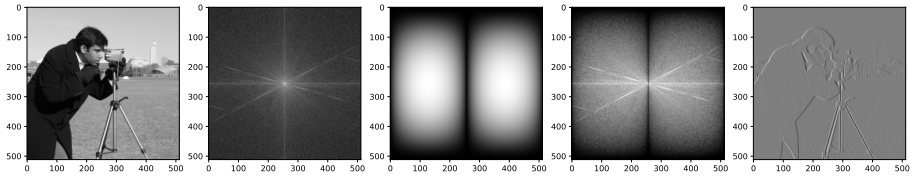


Figure 8: Sobel kernel applied to *camera man*. The figure shows (from the left): the input image, Fourier transformed image, Fourier transformed Sobel kernel, convolution result in the frequency domain, convolution result in the spatial domain.

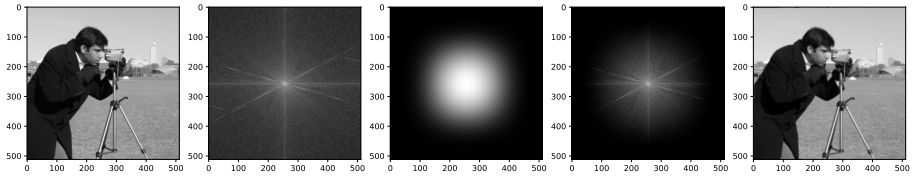
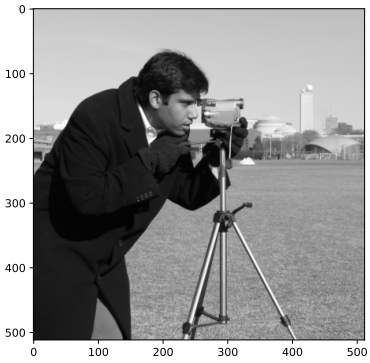
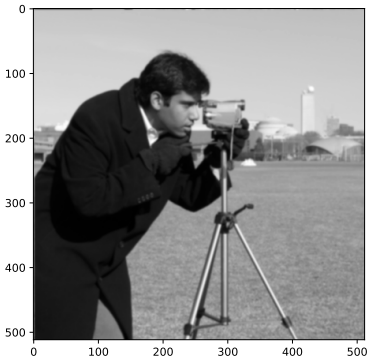


Figure 9: Gaussian kernel applied to *camera man*. The figure shows (from the left): the input image, Fourier transformed image, Fourier transformed Gaussian kernel, convolution result in the frequency domain, convolution result in the spatial domain.



(a) Input image



(b) Gaussian kernel applied to the input image

Figure 10: Input and output image of convolution with a Gaussian kernel.

---

c)

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.
Include the filtered result in your report.

Periodic vertical lines like these will show up horizontally in the Fourier transformed image. The frequency spectrum of the image shows dots scattered over a horizontal line. By designing a filter that cancels out these dots, it is possible to filter out the vertical lines from the original image.

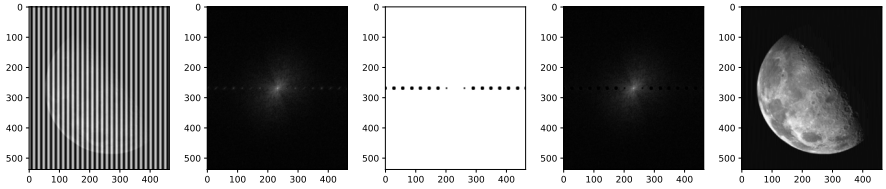


Figure 11: Notch filter applied to a picture of the Moon. The figure shows (from the left): the input image, Fourier transformed image, Fourier transformed Notch kernel, convolution result in the frequency domain, convolution result in the spatial domain.

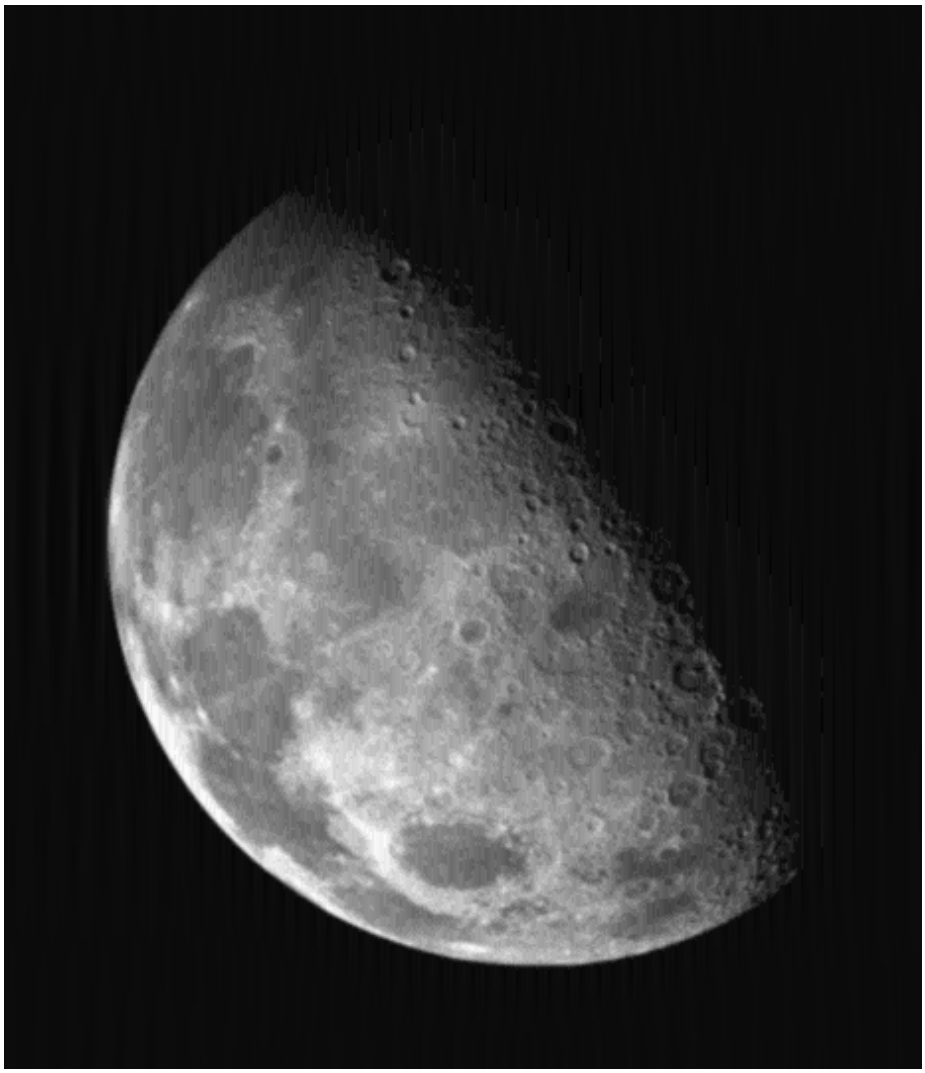


Figure 12: Filtered image of the Moon

---

d)

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. Include the generated image in your report.

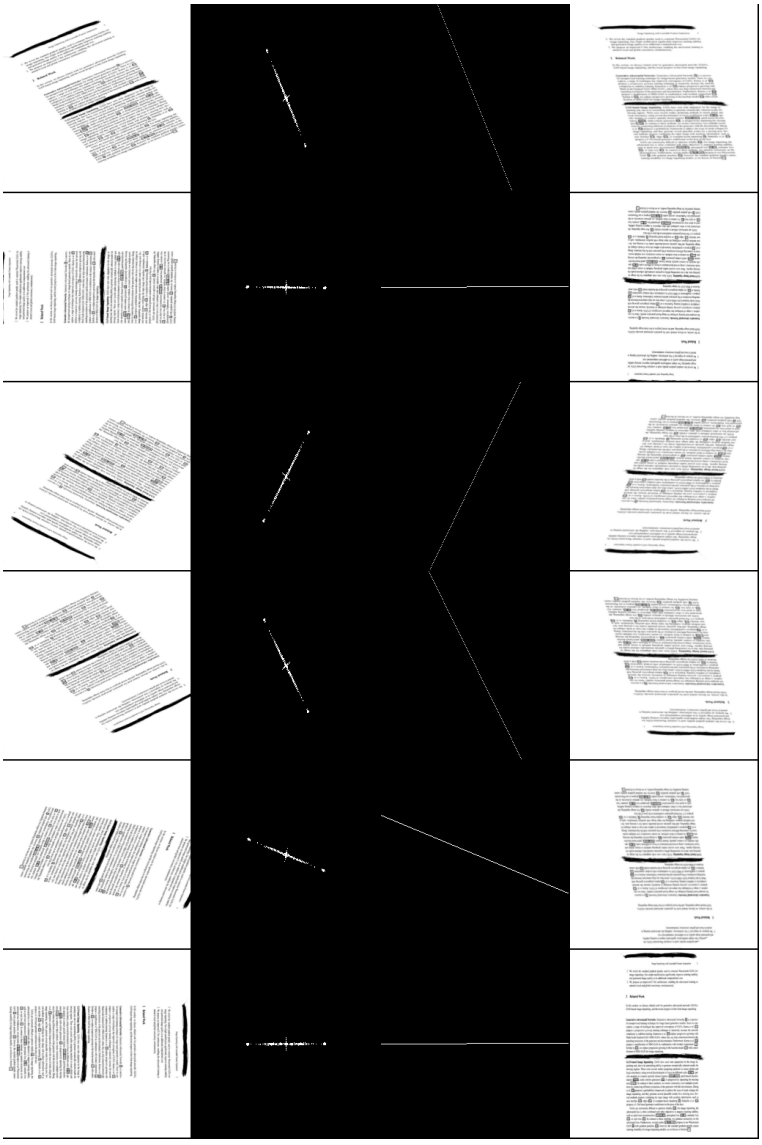


Figure 13: Original image, binary image in frequency domain, Hough line, rotated image

Listing 3: Binary image creation

```
    def create_binary_image(im):
    """Creates a binary image from a greyscale image "im"

    Args:
        im ([np.ndarray, np.float]): [An image of shape [H, W] ...
            in the range [0, 1]]

    Returns:
        [np.ndarray, np.bool]: [A binary image]
    """

    # START YOUR CODE HERE ### (You can change anything inside ...
        this block)
    # Fourier transform image, shift the frequencies to centre
    im_fft = np.fft.fft2(im)
    im_fft = np.fft.fftshift(im_fft)
    # Get magnitude, normalize
    im_fft = np.log(np.abs(im_fft) + 1)
    im_fft = utils.normalize(im_fft)

    threshold = 0.5
    binary_im = np.zeros_like(im)
    # Set to 1 where the normalized magnitude is above threshold
    binary_im = np.where(im_fft>threshold, 1, 0)
    binary_im = binary_im.astype(bool)

    ### END YOUR CODE HERE ###
    return binary_im
```