Exercise Sheet 2: Fourier Transformation

Due on 16.05.2025, 10:00

Kolja Bauer (kolja.bauer@lmu.de)

Task 1: Fourier Transform (10P)

In this exercise we learn basic principles of Discrete Fourier Transformation(DFT) and try out libraries offering a suitable toolbox.

1. Start from loading the images *stripes_vert.gif* and *stripes_diag.gif*. Closely inspect images. Please note that we have edge and compression artifacts. Remove them by cropping only central part of an image, i.e. in range `[10:-10, 10:-10]` and additionally threshold values at value 1.1 (Initial value range after loading is [0, 2]). Now plot those images. These preprocessing steps help to obtain cleaner DFT images.

2. Transform images to Fourier domain. Use functions `scipy.fftpack.fft2` and `np.fft.fftshift` from the `scipy` and `numpy` libraries respectively. First function computes the DFT of an image and second function shifts zero-frequencies to the center of the spectrum, merely for the convenience of representation. Now plot both magnitude and phase using functions `np.abs` and `np.angle` to get respective components of the spectrum. What do you observe? You should obtain a very sparse image of DFT magnitudes, especially for the vertical stripes.

   Hint: If you can't see anything in the magnitude image apply *log* function to transformed images. Make sure you do not apply *log* function to 0 values.

3. Now lets work with real images. Load the image *saturn.png*, transform it to grayscale and plot.

4. Apply Fourier transformation and plot magnitude and phase of its DFT.

5. Now change the image in the original pixel domain by:

   - rotating the image
   - changing the contrast
   - flipping horizontally

   for every transformation compute Fourier transform and plot it. How have the aforementioned transformations of the pixel space affected the DFT domain?

6. Now let's study the inverse Fourier transformation. Invert Fourier domain back to pixel domain and plot image using `scipy.fftpack.ifft2` function, do not forget to shift phase again if necessary. Compare it to the original image.

7. DFT spectrum has a nice intuitive explanation: central part of an image encodes low-frequency components that contains "majority" of the semantic information. We can verify this observation by applying low pass filter to DFT domain and plot the result of the inverse Fourier transform. Zero-out all values outside the central area of the DFT representation, for example leave only values in the central rectangle of size $150px \times 150px$ but you can experiment with the size. Now inverse transform this image and plot alongside, original image, inverted image after low-pass filter and a pixel-level difference between those two. You should see small differences between the images.

8. In the previous exercises we explored the effect of the low-pass filtering on the image reconstruction. Now we can turn this process around and apply high pass filter for the task of image denoising. If there is a lot of noise(especially regular noise) it leaves traces in the fourier domain. We can manually remove this noise in the Fourier domain and obtain a cleaner image.

    - Load image *moonlanding.png* and apply Fourier transformation. In the frequency space find an optimal threshold t for frequencies to keep only low-frequency components. After thresholding apply inverse Fourier transform plot your denoised version of the image.
    - Compare this method with a more naive denoising approach: blur the image by applying a gaussian filter to pixel domain. Compare results of both methods and plot them side by side.

Task 2: Image Sub-sampling and Anti-aliasing (10P)

Aliasing is a phenomenon that occurs when a continuous signal is sampled at a rate that is too low, resulting in distortion or loss of information in the sampled signal. Specifically in computer vision domain, this happens quite often when we sub-sample the image, namely change the number of pixels in the image. When you change the size of an image, the pixels in the original image need to be scaled up or down to fit the new size. This can cause problems such as aliasing, which is when the pixels in the new image don't accurately represent the details in the original image.

To avoid aliasing, it is important to sample the signal at a rate that is at least twice the maximum frequency component in the signal. This is known as the Nyquist-Shannon sampling theorem[1], while in practice, anti-aliasing filters are often used to remove high frequency components from the signal before sampling. In this exercise we will learn how we can achieve this.

1. **Make sure you have the Pillow package installed. In this exercise we will be mainly using it**, as many other packages will have the anti-aliasing enabled by default and it would be hard to spot it.

2. Load the provided image `alias_sample.png` and plot the image in grayscale.

3. Now subsample your image by a factor of 2 using the built-in function `Image.resize` with `NEAREST` and `LANCZOS` for resampling filter. Plot these two images, and you should see the aliasing effect in the resized image with `NEAREST` filter.

   Hint: Because Lanczos enabled windowed sinc resampling filter by default, which provides good anti-aliasing, while the other not.

4. Now we will try to build our own filters learned from the lecture. Recall the Gaussian filter and the 2D convolution we implemented in Exercise 1, and use them as a low-pass filter to remove the high frequency part of the original image. *Take care of the value range and the dimension*

5. Now convert the gaussian-filtered image back to PIL image and resize it with `NEAREST` as you've done in **step 3**. Plot the image and compare it with the one you obtained before. Is the aliasing effect still there? (try to change the size and var of the gaussian filter if aliasing is still there).

6. Recall the Fourier Transform we practiced in Task 1, now use it as a low-pass filter to process the original image. Repeat the same resizing process with `NEAREST`. Plot and check the result.

7. Visualize and align the 4 resized images you obtained from plain `NEAREST` and `LANCZOS` and the two filtered one you obtained from **step 5 and 6** with correct title. Explain in a few sentences:

   - Why do they look differently?
   - What are the similarities and differences between the two low-pass filter we used in step 4 to 6?

[1] https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

---

Important Note: Submit exactly one ZIP file via Moodle before the deadline. Please submit your solution as a Jupyter Notebook. The ZIP file should contain your notebook and the images you are loading. Make sure that it runs on different operating systems and uses relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The notebook file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends, and captions. Missing plots will result in point reduction.