Exercise 4 - Image Filtering

The purpose of this exercise is to illustrate different image filtering techniques.

Learning Objectives

After completing this exercise, the student should be able to do the following:

1. Compute the correlation between an image and a filter using the `scipy.ndimage.correlate` function.
2. Use different border handling strategies when using filtering an image, including `constant` and `reflection`.
3. Implement and apply a mean filter to an image.
4. Implement and apply a median filter to an image (`skimage.filters.median`).
5. Implement and apply a Gaussian filter to an image (`skimage.filters.gaussian`)
6. Describe the effects of applying the mean, the Gaussian and the median filter to images containing Gaussian and outlier noise.
7. Describe the concept on an image edge.
8. Describe the concept of image gradients.
9. Use the Prewitt filter to extract horizontal and vertical edges and their combined magnitude (`skimage.filters.prewitt_h`, `skimage.filters.prewitt_v`, `skimage.filters.prewitt`).
10. Estimate a threshold in an edge image to create a binary image reflecting the significant edges in an image.
11. Implement, test, adapt and evaluate a function that can automatically detect important edges in an image.
12. Implement and test a program that apply filters to a video stream.
13. Test the impact of a video processing frame rate when applying different filters to the video stream.

Installing Python packages

In this exercise, we will be using both scikit-image, OpenCV and SciPy. You should have these libraries installed, else instructions can be found in the previous exercises.

We will use the virtual environment from the previous exercise (`course02502`).

Exercise data and material

The data and material needed for this exercise can be found here: (https://github.com/RasmusRPaulsen/DTUImageAnalysis/blob/main/exercises/ex4-ImageFiltering/data/)

Filtering using Python

scikit-image and SciPy contain a large number of image filtering functions. In this exercise, we will explore some of the fundamental functions and touch upon more advanced filters as well.

Filtering using correlation

We will start by exploring the basic correlation operator from SciPy. Start by importing:

```
from scipy.ndimage import correlate
```

Now create a small and simple image:

```
input_img = np.arange(25).reshape(5, 5)
print(input_img)
```

and a simple filter:

```
weights = [[0, 1, 0],
           [1, 2, 1],
           [0, 1, 0]]
```

Now we can correlate the image with the weights:

```
res_img = correlate(input_img, weights)
```

Exercise 1

Print the value in position (3, 3) in `res_img`. Explain the value?

Border handling

When the value of an output pixel at the boundary of the image is computed, a portion of the filter is usually outside the edge of the input image. One way to handle this, is to assume that the value of the *off-the-edge pixels* of the image are 0. This is called zero padding. Since 0 is the value of a black pixel, the output image will have a dark edge. Another approach is to *reflect* the actual pixel values of the image to the *off-the-edge-pixel*. This is the default behaviour of `correlate`. We can also set the *off-the-edge-pixel* to have a constant value (for example 10) by:

```
res_img = correlate(input_img, weights, mode="constant", cval=10)
```

Exercise 2

Compare the output images when using `reflection` and `constant` for the border. Where and why do you see the differences.

Mean filtering

Now we will try some filters on an artificial image with different types of noise starting with the mean filter.

Exercise 3

Read and show the image **Gaussian.png** from the exercise material.

Create a mean filter with normalized weights:

```
size = 5
# Two dimensional filter filled with 1
weights = np.ones([size, size])
# Normalize weights
weights = weights / np.sum(weights)
```

Use `correlate` with the **Gaussian.png** image and the mean filter. Show the resulting image together with the input image. What do you observe?

Try to change the size of the filter to 10, 20, 40 etc.. What do you see?

What happens to the noise and what happens to the places in image where there are transitions from light to dark areas?

Median filtering

The median filter belongs to the group of *rank filters* where the pixel values in a given area are sorted by value and then one of the values are picked. Here the median value of the sorted values.

Start by importing the filter:

```
from skimage.filters import median
```

We can create a *footprint* which marks the size of the median filter and do the filtering like this:

```
size = 5
footprint = np.ones([size, size])
med_img = median(im_org, footprint)
```

Exercise 4

Filter the **Gaussian.png** image with the median filter with different size (5, 10, 20...). What do you observe? What happens with the noise and with the lighth-dark transitions?

Comparing mean and median filtering

Try to load and show the **SaltPepper.png** image. This image has noise consist of very dark or very light pixels.

Exercise 5

Try to use your mean and median filter with different filter sizes on the **Salt-Pepper.png**. What do you observe? Can they remove the noise and what happens to the image?

Gaussian filter

Scikit-image contains many different filters.

The Gaussian filter is widely used in image processing. It is a smoothing filter that removes high frequencies from the image.

Exercise 6

Let us try the Gaussian filter on the **Gaussian.png** image. Start by importing the filter:

```
from skimage.filters import gaussian
```

and do the filtering:

```
sigma = 1
gauss_img = gaussian(im_org, sigma)
```

Try to change the `sigma` value and observe the result.

Exercise 7

Use one of your images (or use the **car.png** image) to try the above filters. Especially, try with large filter kernels (larger than 10) with the median and the Gaussian filter. Remember to transform your image into gray-scale before filtering.

What is the visual difference between in the output? Try to observe places where there is clear light-dark transition.

Edge filters

In image analysis, an *edge* is where there is a large transition from light pixels to dark pixels. It means that there is a *high pixel value gradient* at an edge. Since objects in an image are often of a different color than the background, the outline of the object can sometimes be found where there are edges in the image. It is therefore interesting to apply filters that can estimate the gradients in the image and using them to detect edges.

The **Prewitt filter** is a simple gradient estimation filter. The Python version of the Prewitt filter can estimate the horizontal gradient using the `prewitt_h` filter, the vertical gradient with the `prewitt_v` filter and the *magnitude of the edges* using the `prewitt` filter. The magnitude is computed as

$$V(x,y) = \sqrt{(P_v^2 + P_h^2)} \ ,$$

where P_v and P_h are the outputs of the vertical and horizontal Prewitt filters.

Start by importing the filter:

```
from skimage.filters import prewitt_h
from skimage.filters import prewitt_v
from skimage.filters import prewitt
```

Exercise 8

Try to filter the **donald_1.png** photo with the `prewitt_h` and `prewitt_v` filters and show the output without converting the output to unsigned byte. Notice that the output range is [-1, 1]. Try to explain what features of the image that gets high and low values when using the two filters?

Exercise 9

Use the `prewitt` filter on **donald_1.png**. What do you see?

Edge detection in medical images

The **ElbowCTSlice.png** image is one slice of a CT scan of an elbow from a person that climbed, wanted to show off, fell, landed on his arm and fractured a bone.

Exercise 10

The goal of this exercise is to detect the edges that seperates the bone from the soft tissue and the edges that separates the elbow from the background. Your detection algorithm should follow this outline:

- Read the CT image
- Filter the image using either a Gaussian filter or a median filter
- Compute the gradients in the filtered image using a Prewitt filter
- Use Otsu's thresholding method to compute a threshold, T, in the gradient image
- Apply the threshold, T, to the gradient image to create a binary image.

The final binary should contain the edges we are looking for. It will probably contain noise as well. We will explore methods to remove this noise later in the course.

You should experiment and find out:

- Does the median or Gaussian filter give the best result?
- Should you use both the median and the Gaussian filter?

- What filter size gives the best result?
- What sigma in the Gaussian filter gives the best result?

Tip: To get a better understanding of your output, uou can use the scaled visualization and colormapping that we explored in an earlier exercise:

```
min_val = edge_img.min()
max_val = edge_img.max()
io.imshow(edge_img, vmin=min_val, vmax=max_val, cmap="terrain")
```

Video filtering

Now try to make a small program, that acquires video from your webcam/telephone, filters it and shows the filtered output. In the exercise material there is a program that can be modified.

Exercise 11

Modify the `process_gray_image` function in the program so it performs a Prewitt filter on the input image.

Also try to make it perform the automatic edge-detection (Prewitt + Otsu) from exercise 10.

Exercise 12

Try to use a median filter with a size of 10 on the video stream. What happens with the frames-per-second? Why?

References

- sci-kit image filters
- rank filters
- scipy correlate

Differences between `matplotlib.pyplot.imshow()` and `skimage.io.imshow()`

You should also be aware of the differences in the default behaviour between scikit-image and matplotlib when displaying an image. The default behaviour of matplotlib.pyplot.imshow() is to use the dynamic range of the image values to plot the image. It is, if the image intensities range from [26, 173], the black colour is assigned to 26 and the white to 173. Meanwhile, skimage.io.imshow() displays the dynamic range according to the image type, [0, 255] for integer images and [0., 1.] for float images.

Feel free to use either package, as far as you know what you are doing.