Computer Vision Lab 1
Basic image operations

David Van Hamme

1 OpenCV

In this course, we will use OpenCV, an open source computer vision library originally developed by Intel. The library can be used free of charge for all purposes under a BSD license. All important basic image processing functionality is present: reading and writing of video and images, filter operations, color transformations, feature detection, segmentation, stereo vision, calibration, ...

OpenCV is cross-platform and supports Linux, Windows and OS X. The library has bindings for Python, C++ and C, but Python is strongly recommended for ease of use.

You can use OpenCV in the development environment that you choose yourself. In fact, a text editor and command line is all you need. However, your work flow may be more efficient if you use an IDE like Spyder. Some students may be familiar with Jupyter notebooks and prefer to use that. Use whatever tools you like.

Installation

You can find out how to install OpenCV for the language and the operating system of your choice in the links on `opencv.org`.

Linux hints

For those with limited programming experience, we recommend using Python under the Ubuntu operating system. In Ubuntu you can easily install the Python package manager " pip " via the command line:

```
sudo apt install python-pip
```

Then you can install the entire OpenCV library with

```
sudo pip install opencv-contrib-python
```

For your convenience you can also install the Spyder IDE:

```
sudo apt install spyder
```

Windows hints

One of the easiest ways to get python and OpenCV up and running in Windows is by downloading Anaconda from `https://www.anaconda.com/distribution/`. After installing anaconda, you can open an anaconda command prompt and install opencv with

```
pip install opencv-python
```

The anaconda distribution already includes the Spyder IDE for autocompletion and convenient debugging.

Documentation

You can find the documentation on `http://docs.opencv.org/trunk/index.html`. When functions or data structures are suggested in these lab assignments, look them up in the documentation so that you understand what they are and how to use them. You will also find sample programs and tutorials here.

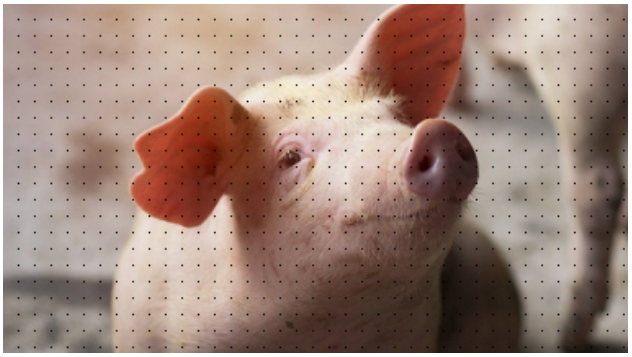
2 Reading, manipulating and writing pixel data

Images are arrays of pixel intensities. These arrays can be encoded and compressed in various ways to store them on disk, most commonly in png or jpeg format. OpenCV takes care of the decoding and decompressing when you read images from file, so you have direct access to the pixel data once they are read. Manipulating image arrays is a core competence for computer vision.

2.1 Exercise 1

Write a simple program that reads **clouds.jpg** (download from Ufora) and displays it on screen. The image is a multi-dimensional numpy data array, and you can use array indexing to manipulate this array. The assignments below can each be solved in one line of code.

Question 1 What do the dimensions of the image array represent?
Assignment 1 Crop the image so it becomes square by chopping off the a part on the right side.
Assignment 2 Discolor the image by reducing the intensity of the red value of every pixel by half.
Assignment 3 Discolor the image by doubling the intensity of the red value of every pixel. You may have to handle an overflow problem (and use two more lines of code).
Assignment 4 Make a regular grid of black dots on the image so that the dots are 10 pixels apart vertically and horizontally, like in the image below.



Useful functions: **imread, imwrite, namedWindow, imshow, waitKey, destroyAllWindows**.

3 Thresholding

Thresholding is a method to segment grayscale images. It can be used to find objects of interest in images. Pixel intensity values are compared to a threshold value and classified according to whether they are higher or lower than this value. Finding the correct threshold value is often not trivial.

3.1 Exercise 2

Write a simple program to perform basic image thresholding on **clouds.jpg**. On simple images like this, a fixed threshold can be effective to separate foreground from background. The goal in this case is to achieve a binary image where the clouds are white and the empty sky is black.

Assignment 5 Convert the image to a grayscale image.
Assignment 6 Threshold the grayscale image at 50% of the maximum value for this datatype.
Assignment 7 Threshold the grayscale image at the ideal threshold determined by Otsu's method.

Useful functions: **cvtColor, threshold**

3.2 Exercise 3

Try your thresholding from the previous exercise on **painting2.jpg**. Clearly, a single threshold will not work to isolate the painting due to the intensity gradient present in the background. This can be addressed by adaptive thresholding: an appropriate threshold for each pixel can be determined based on the mean intensity in an area around the pixels.

Assignment 8 Adaptively threshold the grayscale version of **painting2.jpg** so you get a similar result to the one below, where the background is uniformly white and you can cut out the painting along black lines.

Useful functions: **adaptiveThreshold**

4 Filtering

Filtering is often the first step in image analysis. Filters can be used to blur or sharpen an image, to remove noise, or to compute features from the image for further analysis.

A filter kernel is a window that is moved over the image and for each window position, a single new pixel value is calculated from the entire window. In the case of linear filters, the kernel is a matrix of weights by which the surrounding pixels are multiplied before summing them in the central pixel.

4.1 Exercise 4

A Gaussian filter replaces each pixel with a weighted average of the surrounding pixels. The weights in the kernel are determined by a 2D normal distribution around the central pixel, so nearby pixels have more influence than slightly more distant pixels. This type of filter is often used to remove *white noise* from the image. White noise is a form of noise in which each pixel has undergone a random deviation from its original value.

Assignment 9 Remove the white noise from **whitenoise.png** by Gaussian filtering. Find parameters for the Gaussian kernel that you find strike a good balance between noise level and blurriness of the result. This is subjective, but experiment with it!
Question 2 Can you choose the kernel size and sigma of the distribution independent of each other?

Useful functions: **GaussianBlur**.

4.2 Exercise 5

Salt and pepper noise is a form of noise in which some pixels turn black or white. This type of noise is common for some types of medical imaging (notably MRI). It can be alleviated by median filtering, which replaces every pixel by the median of the surrounding pixels. As a result, the outliers have no influence anymore, in contrast to a Gaussian filter, where the influence is only reduced.

Assignment 10 Test the Gaussian filter on **saltandpeppernoise.png**.
Assignment 11 Apply median filtering on the same image.
Question 3 Which result is preferable and why?

Useful functions: **medianBlur**.

4.3 Exercise 6

Unsharp masking is a technique to sharpen images by accentuating contrasts. The workflow is as follows:

- blur the image;
- subtract the blurred version from the original to obtain a difference image. This difference image should be able to contain negative values, I suggest using 32-bit floating point because it is supported by the most functions in opencv;
- amplify the difference by multiplying it with a factor;
- add this amplified difference image to the original image.

Assignment 12 Implement unsharp masking to sharpen **unsharp.png**. Make sure you do not get overflow in your datatype! Your goal is to achieve something similar to the image below.
Useful functions: **addWeighted**

4.4 Exercise 7

You can make custom filters to realize specific image adaptations, e.g. blurring in one particular direction.

Assignment 13 Write a program that blurs **blots.png** diagonally with the kernel below (mind the multiplication factor in front).

$$1/7 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Useful functions: **filter2D**.

5 Reporting

Write a brief report about this lab session in which you answer each question and provide the output image of each assignment. Append your source code.