Exercise 2: Image filtering

This exercise will give you some experience in filtering images using OpenCV. You will implement filtering operations yourself first, and then use OpenCV to do the same. This should give you both an in-depth understanding of the algorithms, but also show the benefits of using quality optimized code.

- Load and display a one of the noisy images for this exercise
- Implement a linear filter (correlation or convolution) which takes an input image (cv::Mat) and a kernel matrix (cv::Mat) as input. Use either 3x3 or variable size (2k+1) x (2k+1) size kernel matrix. The filter should also allow an option for padding the image.
- Implement a median filter
- Apply your two filters to an image with Gaussian noise. Which method performs best? What is the signal to noise ratio?
- Apply your two filters to an image with salt and pepper noise. Which method performs best?
- Optional: Construct a Gaussian kernel with variable standard deviation. Use the kernel in your implementation of a linear filter. Benchmark it against OpenCV's implementation `cv::GaussianBlur()`.

Notes on OpenCV types

- Many times it is advantageous to be explicit when you define a matrix, fx.

```
cv::Mat gaussianNoise(src.rows, src.cols, CV_32FC1);
cv::Mat saltpepper_noise = cv::Mat::zeros(src.rows, src.cols, CV_8U);
cv::Mat src = cv::imread(baseName + "." + ext, cv::IMREAD_GRAYSCALE);
```

Gaussian noise with zero mean will have both negative and floating point values, so CV_32F is appropriate. Only one channel is needed so I might as well be explicit and define the type as CV_32FC1

- Check that your images are the type you think they are

```
if(src.type() != CV_8UC1){
    std::cout << "Error: Image loaded in wrong format" << std::endl;
    return 0;
}
```

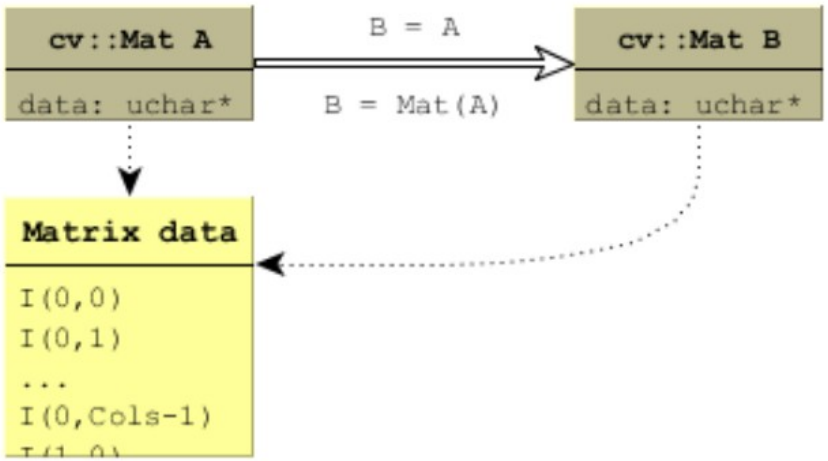
- Make sure to access the data using the appropriate type. Also make sure that the given type can contain the data

src = 8-bit uchar
gaussianNoise = 32-bit float
gaussianAndSrc = 8-bit uchar

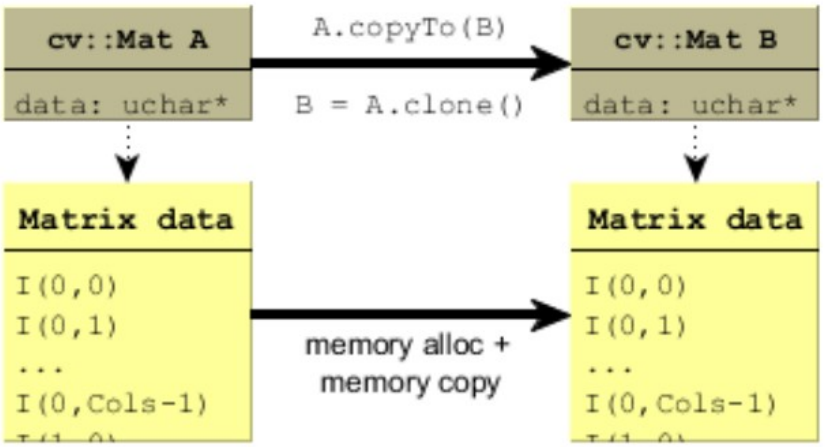
sum has values outside [0,255] range of uchar, so a saturate_cast<uchar> is needed

```
for(int i=0; i<src.rows; i++){
    for(int j=0; j<src.cols; j++){
        gaussianAndSrc.at<uchar>(i,j) = cv::saturate_cast<uchar>(src.at<uchar>(i,j) + gaussianNoise.at<float>(i,j));
    }
}
```

Deep vs shallow copy



OpenCV Mat: Assignment operator or Copy Constructor



OpenCV Mat: copyTo() or clone() methods

Using shallow copies to edit Regions of Interest (ROI)

```
// selecting a ROI
Mat roi(img, Rect(10,10,100,100));
// fill the ROI with (0,255,0) (which is green in RGB space);
// the original 320x240 image will be modified
roi = Scalar(0,255,0);
```

A short Cmake guide

- CMake is used to configure a project
- A CMakeLists.txt file defines
  - the resource files
  - required libraries
  - target executables and libraries to be build
- All configuration and compiling is done through terminal
  - QTCreator is just the middle man

Compile-link process

- Compiling a program is a two-step process:
  - Compilation: “translates” each source file (.cpp/.c) into an object (.o) file
  - Linking: collects all object files in a single executable
    - Static libraries (.lib / .a) are copied into the resulting executable.
    - Dynamic libraries (.lib+.dll / .so) remain in their position and need to be found by the dynamic linker.

CMakeLists.txt

You choose the name of your project

Look if OpenCV is installed

Define executable. Choose name and list source files

Link libraries to the executable. Name of executable must match the name defined above

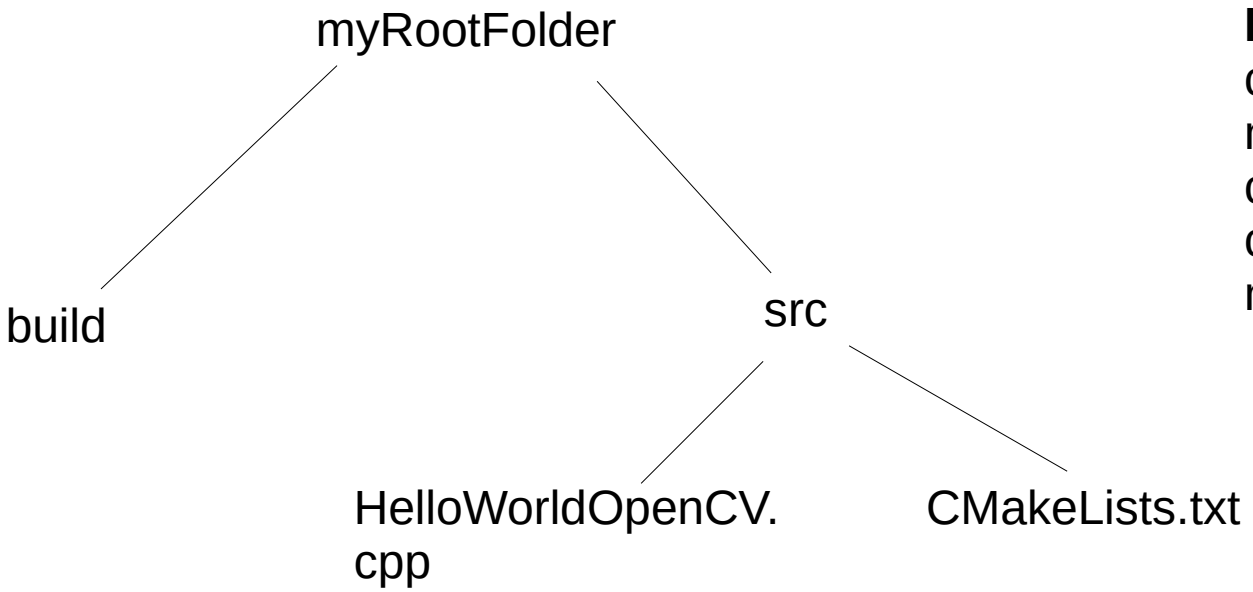
```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(HelloWorldOpenCV)

find_package(OpenCV REQUIRED)

add_executable(HelloWorldOpenCV HelloWorldOpenCV.cpp)

target_link_libraries(HelloWorldOpenCV ${OpenCV_LIBS})
```

Structure of a simple CMake project



Example:
cd ~/workspace/HelloWorldOpenCV
mkdir build
cd build
cmake ../src
make

```
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
project(HelloWorldOpenCV)

find_package(OpenCV REQUIRED)

add_executable(HelloWorldOpenCV HelloWorldOpenCV.cpp)

target_link_libraries(HelloWorldOpenCV ${OpenCV_LIBS})
```

Run executable:
./HelloWorldOpenCV