2. Exercise – Noise Suppression

→ Part I - Theoretical

→ Part II - Practical

- Moving Average Filter
- Median Filter
- Bilateral Filtering

- Optional: Non-Local Means

2. Exercise – Theory

1. When should the median filter be applied to an image and when the moving average filter?

Hint: Think about type of noise and type of degradation present in the image.

2. **Explain** your answer to question 1.

3. Is there a **general** better choice than the moving average filter?

Hint: Revisit slides 54 and following in lecture 3.

4. **Explain** your answer to question 3.

2. Exercise - Given Functions

FILE: main.cpp
int main(int argc, char argv)**

- Main function
  - Usage:
    - dip2 generate path_to_original
      - Calls Dip2::generateNoiseImages(...)
      - Generates and saves noisy images
    - dip2 restorate
      - Uncomment Dip2::test(...) for basic testing!
      - Calls Dip2::run(...) for noise reduction

FILE: Dip2.cpp/h
void Dip2::generateNoisyImages(string fname)

- Applies two noise models to original image
- Saves both images (noiseType_1.jpg and noiseType_2.jpg)

2. Exercise - Given Functions

void Dip2::run(string fname)

- Loads both noisy images
- Tries to reduce noise by different methods
  - Calls Dip2::noiseReduction(..)
  - **TO DO:**
    Set parameters of Dip2::noiseReduction(..) according to the type of noise you observe in the two images
- Saves result images

FILE: Dip2.cpp

```
void Dip2::run(void){
    ...
    Mat noise1 = imread("noiseType_1.jpg", 0);
    Mat noise2 = imread("noiseType_2.jpg", 0);
    ...
    Mat restorated1 = noiseReduction(noise1, "", 1);
    Mat restorated2 = noiseReduction(noise2, "", 1);
    ...
}
```

2. Exercise - Given Functions

```
Mat Dip2::noiseReduction(Mat& src, string method,
                         int kSize, double param)
```

- Parameter:
  - **src** : noisy source image
  - **method** : defines method to be used
    - median, average, adaptive, bilateral
  - **kSize** : Kernel size
  - **param** : adaptive smoothing: threshold
    bilateral filter: std-dev of radiometric kernel
  - **return** : noised reduced output image

- Calls
  - averageFilter(...), medianFilter(...), bilateralFilter(...), or nlmFilter(...)

2. Exercise – To Do

`Mat Dip2::spatialConvolution(Mat& src, Mat& kernel)`

- Parameter:
  - **`src`** : source image
  - **`kernel`** : kernel of the co
  - **`return`** : output image
- Applies convolution in spati
- One method of border hand
- Do **NOT** use convolution fu

```
Mat Dip2::spatialConvolution(Mat& src, Mat& kernel){

   // init output
   Mat dst = Mat::zeros(src.rows, src.cols, CV_32FC1);

   // some indices and temp. variables
   int x,y;
   double val;

   // loop through image
   for(int i=0; i<src.cols; i++){
      for(int j=0; j<src.rows; j++){
               //((double*)(out->imageData + j * out.widthStep))[i] = 0;
         val = 0;
         // loop through local window
         for(int r=-kernel.cols/2; r<=kernel.cols/2; r++){
            for(int s=-kernel.rows/2; s<=kernel.rows/2; s++){
               // global image position based on local window position
               x = i+r;
               y = j+s;
               // border check
               float value = getValue(src, y, x);
               // accumulate values
               val += value * kernel.at<float>(-s+kernel.cols/2, -r+kernel.rows/2);
               //((double*)(out->imageData + j * out->widthStep))[i] += ((uchar*)(in
            }
         }
         // set final value
         dst.at<float>(j,i) = val;
      }
   }
   return dst;
}
```

2. Exercise – To Do

`Mat Dip2::averageFilter(Mat& src, int kSize)`

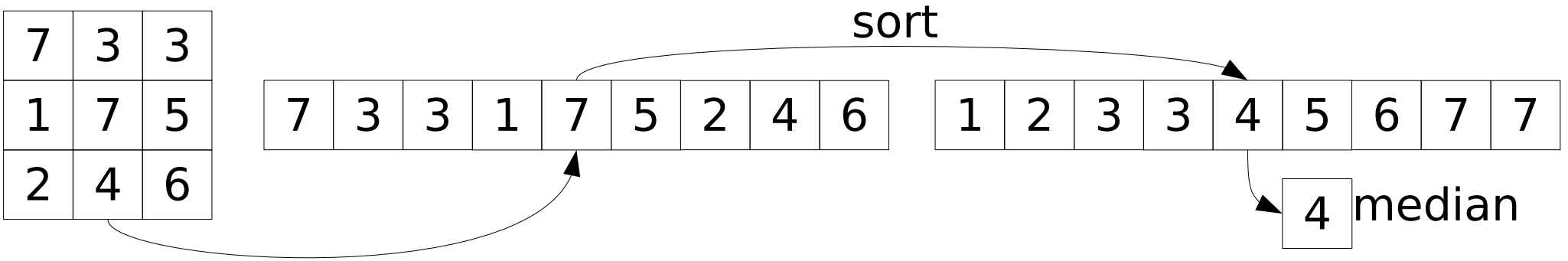
- Parameter:
  - **`src`** : noisy source image
  - **`kSize`** : Kern
  - **`return`** : outp
- Uses convolution
- Calls spatialConv

```
Mat Dip2::averageFilter(Mat& src, int kSize){

    // define kernel
    Mat kernel(kSize, kSize, CV_32FC1, Scalar(1./(kSize*kSize)));

    // use convolution
    return spatialConvolution(src, kernel);
}
```

`Mat Dip2::medianFilter(Mat& src, int kSize)`

- Parameter:
  - **`src`** : noisy source image
  - **`kSize`** : Kernel size
  - **`return`** : output image
- Applies local median filtering

2. Exercise - Median




```
#include <iostream>
#include <algorithm>

int main() {
  int array[] = { 23, 5, -10, 0, 0, 321, 1, 2, 99, 30 };
  int elements = sizeof(array) / sizeof(array[0]);
  std::sort(array, array + elements);
  for (int i = 0; i < elements; ++i)
     std::cout << array[i] << ' ';
}
```