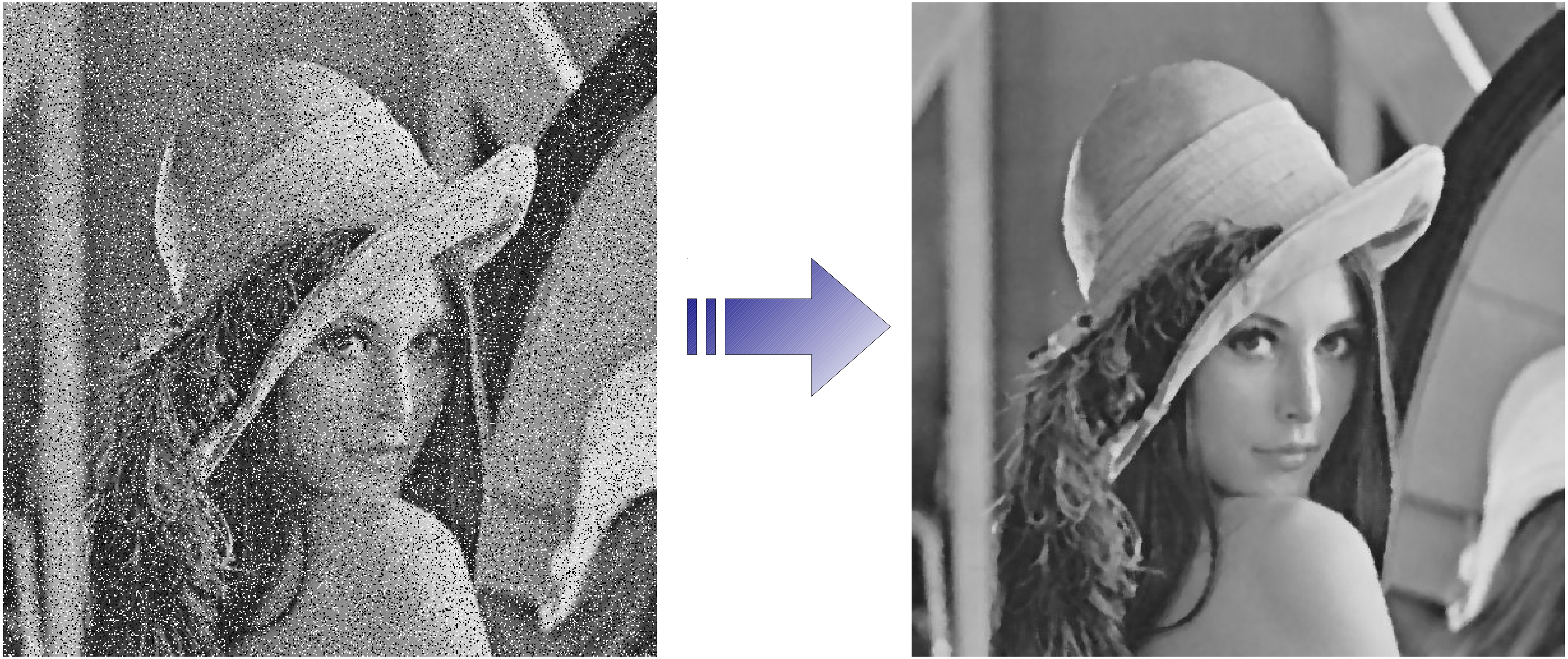
Digital Image Processing

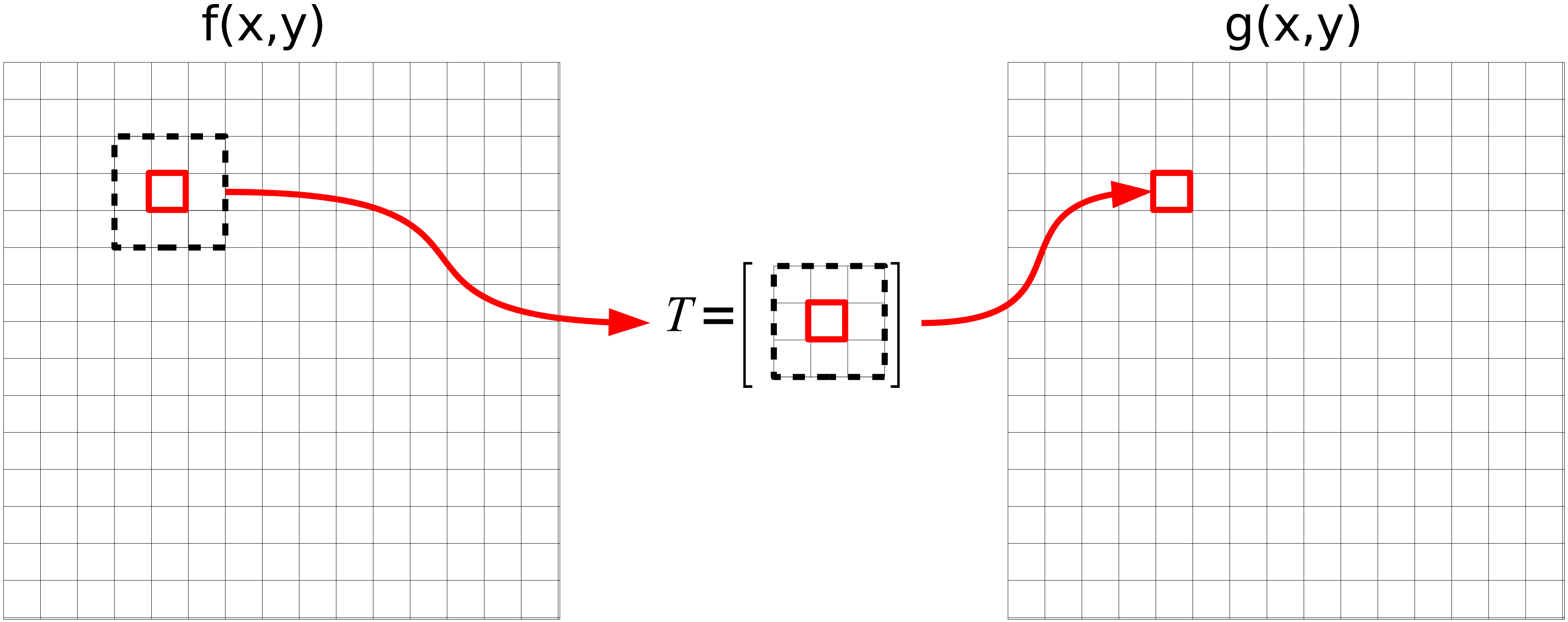
Berlin University of Technology (TUB),
Computer Vision and Remote Sensing Group
Berlin, Germany

Purpose of
Digital Image Processing

Image restoration: Improving *objective* image quality
e.g. noise suppression



Sliding Window

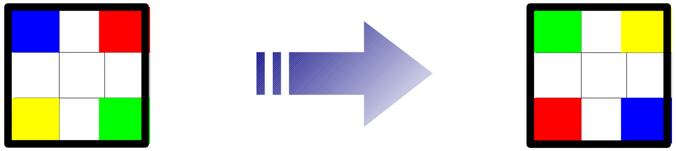


- Operator T takes into account only local information of f
- Result in g is based on pixel intensity and intensities of neighbours
  - '*Filter size*' refers to size of neighbourhood (e.g. 3x3 pixels)

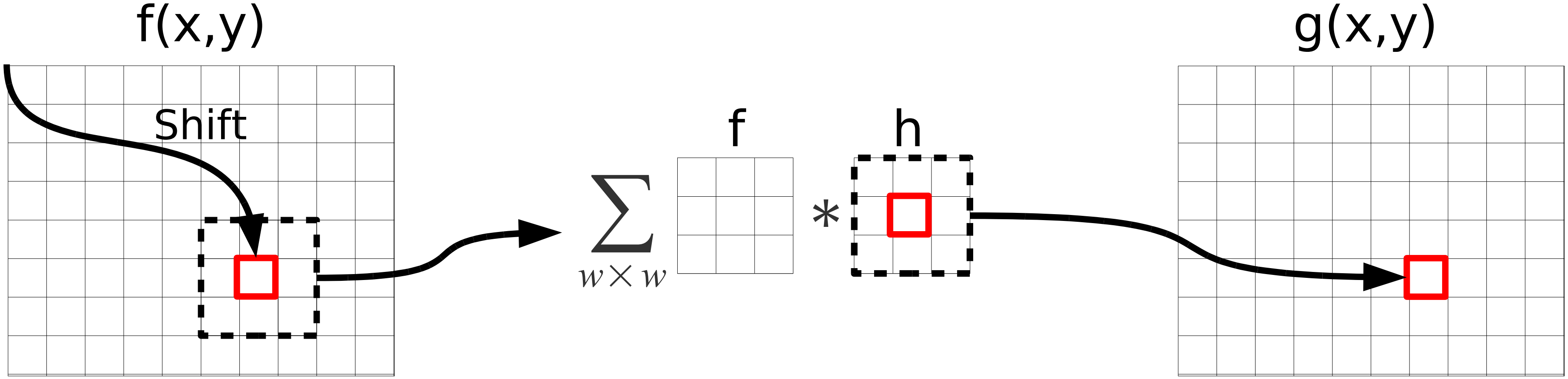
Convolution

$$g(\alpha,\beta)=\sum_{x=1}^{N}\sum_{y=1}^{M} f(x,y)\cdot h(x-\alpha,y-\beta)$$

1. **Flip filter kernel** (about the filter centre)



2. **Shift** (re-centre), **Multiply** and **Integrate**



Convolution

- Filter consists of coefficients and has a centre:

$$h(x-\alpha,y-\beta)=\begin{pmatrix} h(-1,-1) & h(0,-1) & h(1,-1) \\ h(-1,0) & \boxed{h(0,0)} & h(1,0) \\ h(-1,1) & h(0,1) & h(1,1) \end{pmatrix}$$

- Linear filters are applied by *convolution*:

$$g(x,y)=f(x,y)*h(x,\alpha,y,\beta)=$$

$$\sum_{3\times 3}\begin{pmatrix} f(x-1,y-1)h(1,1) & f(x,y-1)h(0,1) & f(x+1,y-1)h(-1,1) \\ f(x-1,y)h(1,0) & f(x,y)h(0,0) & f(x+1,y)h(-1,0) \\ f(x-1,y+1)h(1,-1) & f(x,y+1)h(0,-1) & f(x+1,y+1)h(-1,-1) \end{pmatrix}$$

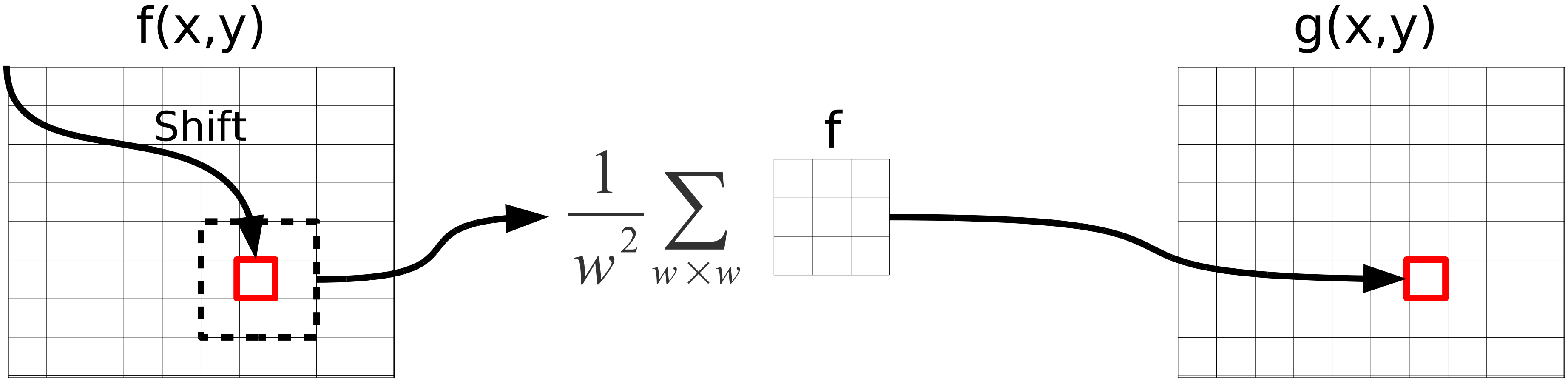
Filter Techniques

Example: Noise Suppression by Moving Average Filter

$$h(x,y)=\frac{1}{w^2}\begin{pmatrix}1 & 1 & 1 & \cdots\\ 1 & 1 & 1 & \cdots\\ 1 & 1 & 1 & \cdots\\ \vdots & \vdots & \vdots & \ddots\end{pmatrix}$$

(w x w Filter Kernel)

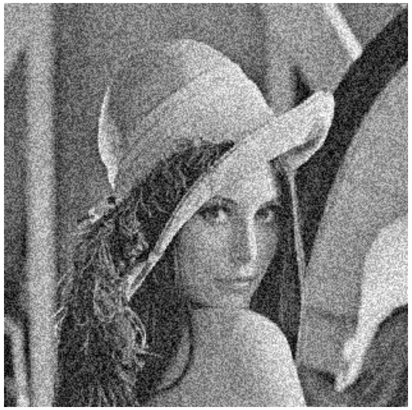
- Each pixel intensity is replaced by the local average...



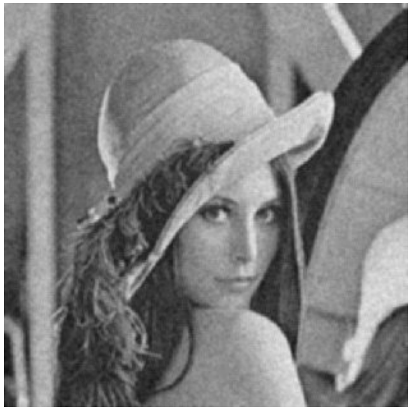
Filter Techniques

Example: Noise Suppression by Moving Average Filter

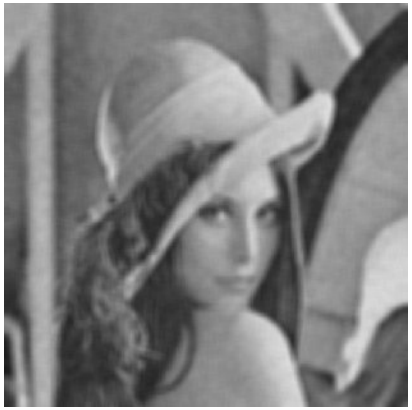
Gaussian Noise



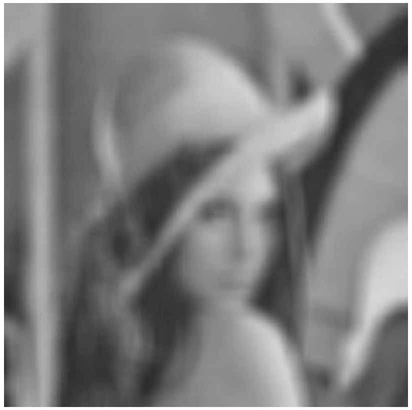
f



w=5

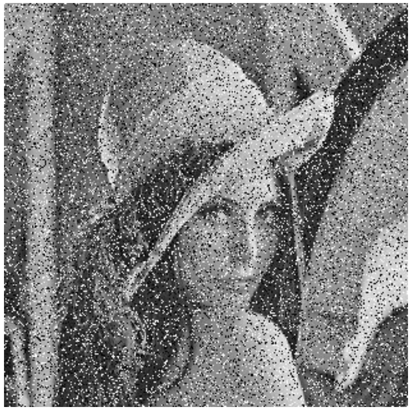


w=11

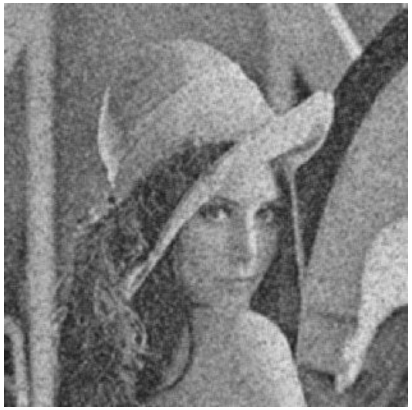


w=25

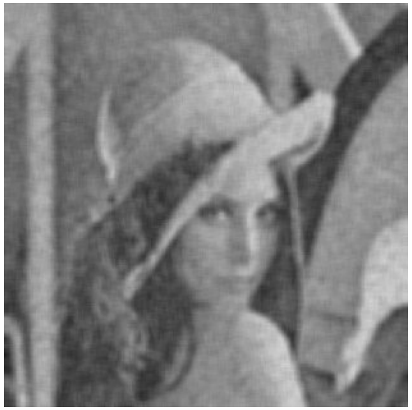
Shot Noise



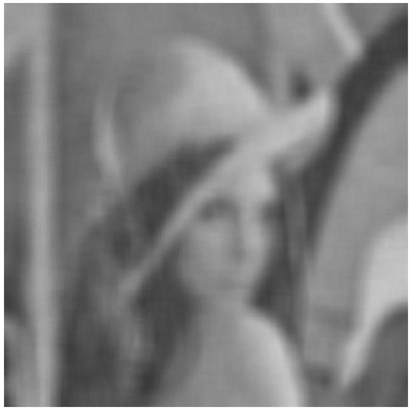
f



w=5



w=11



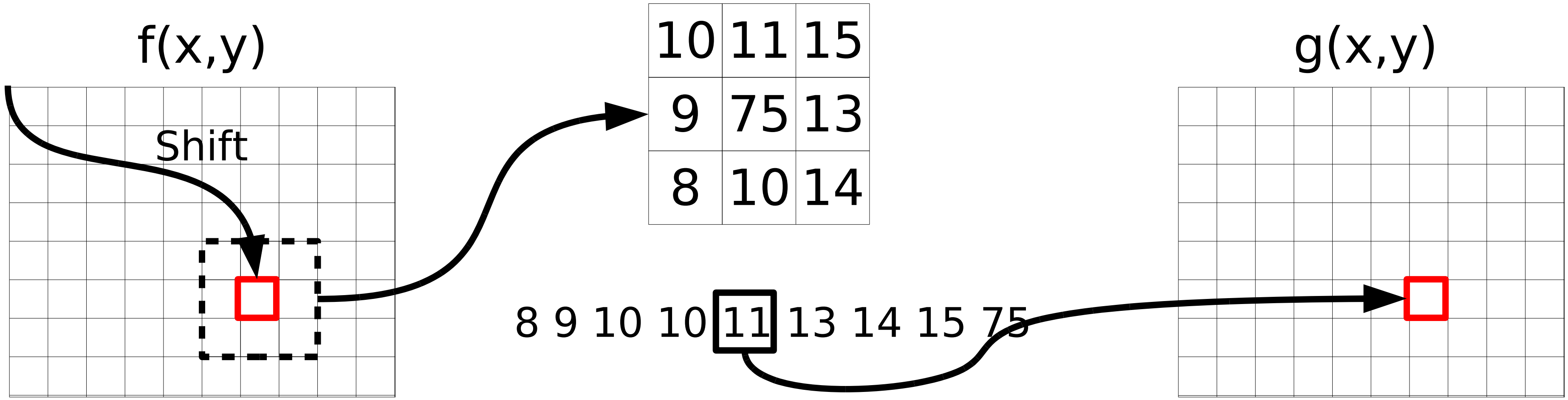
w=25

Filter Techniques

Example: Noise Suppression by Median Filter (NOTE: No convolution)

1. Consider intensities in a local NxN window
2. Sort intensities
3. Select middle value (median) as result

- Each pixel intensity is replaced by the local median...



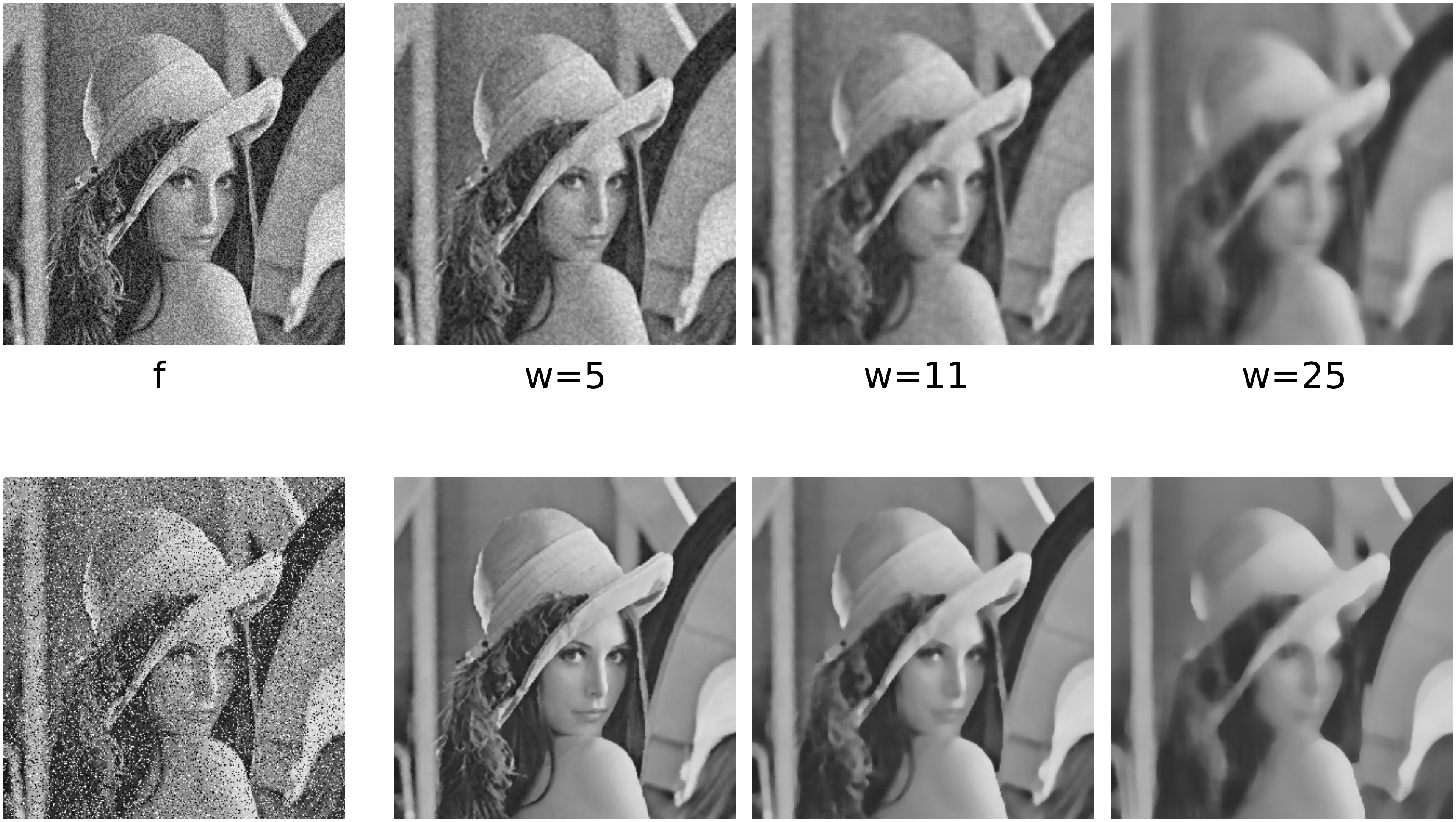
10	11	15
9	75	13
8	10	14

8 9 10 10 11 13 14 15 75

- Effectively removes outliers
- Preserves sufficiently large (>> wxw) image structures

Filter Techniques

Example: Noise Suppression by Median Filter



Gaussian Noise

f

w=5

w=11

w=25

Shot Noise

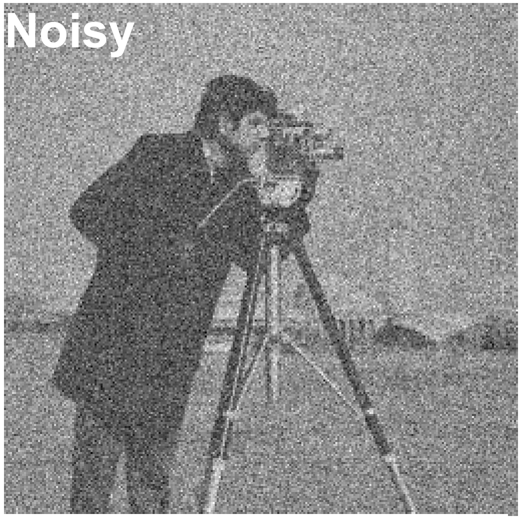
f

w=5

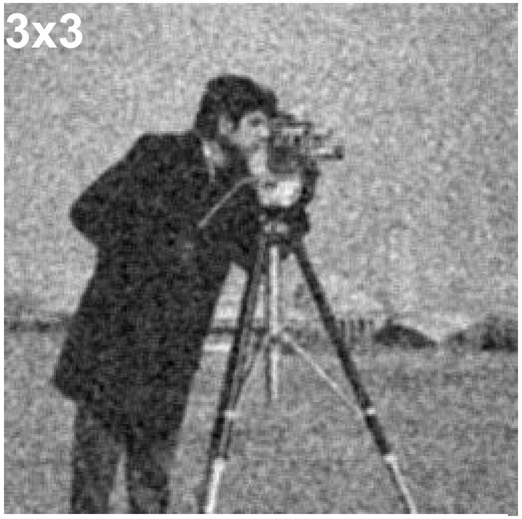
w=11

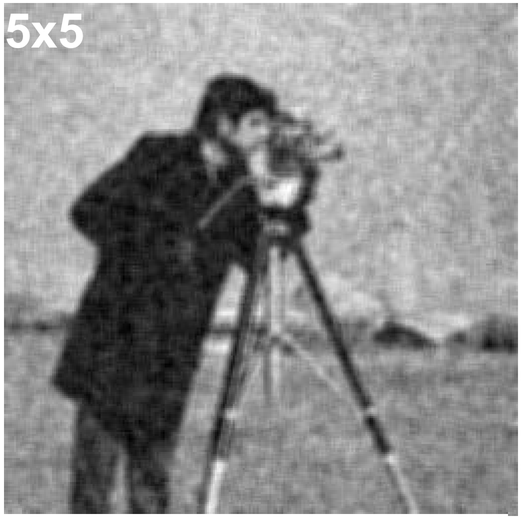
w=25

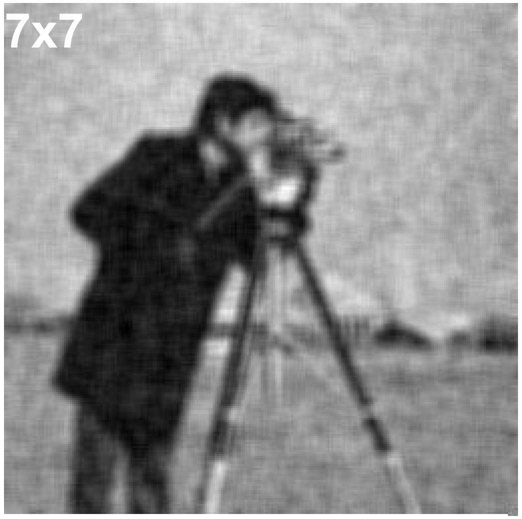
Noise Suppression vs. Resolution



Moving average filtering







Adaptive Smoothing

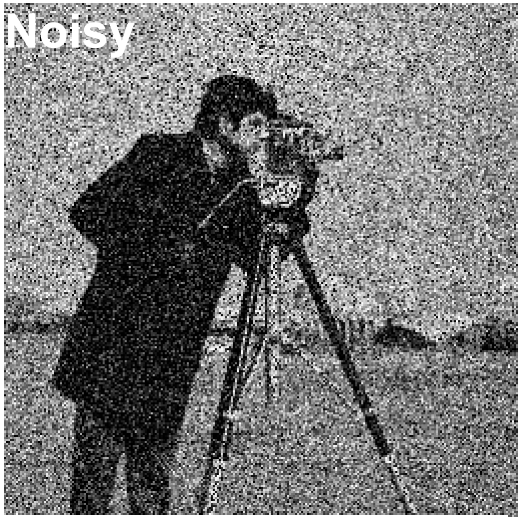
$$m_n(x,y)=\begin{cases}1/N^2 & -N/2\leq x,y<N/2\\ 0 & .\end{cases}$$

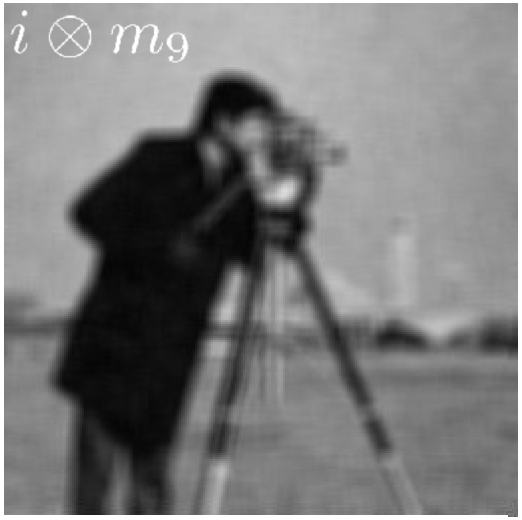
$$o_n(x,y)=\begin{cases}i\otimes m_n & |i\otimes m_3-i\otimes m_n|\leq T\\ i & .\end{cases}$$

- Average unless filtered version departs too far from original
  - Largest discrepancies expected near strong edges
  - Threshold T and size N must be specified by the user!

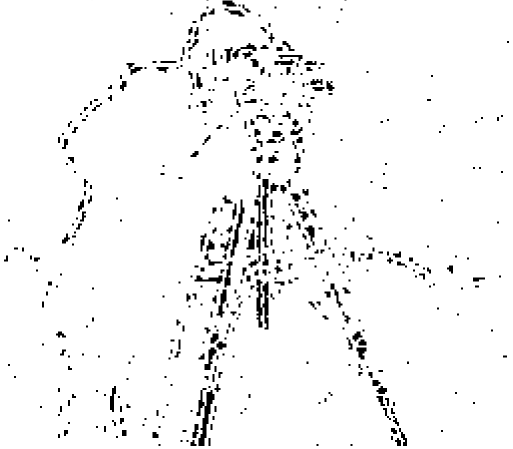
Edge Preservation (II)

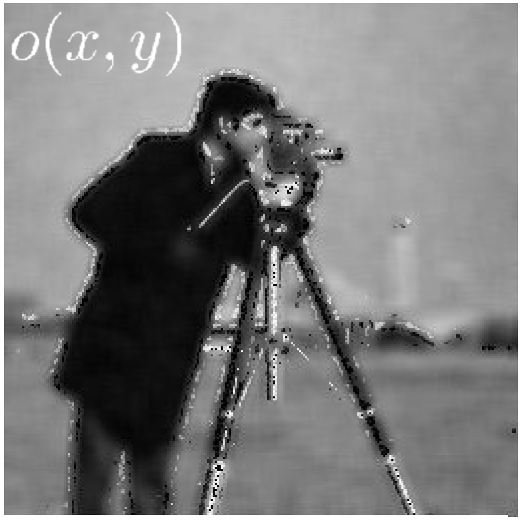
Original

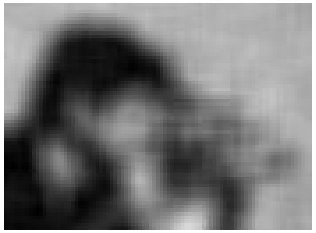

Noisy

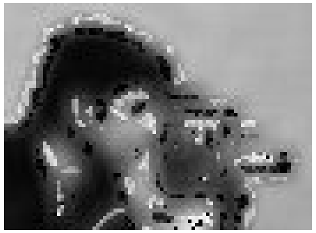

$i \otimes m_9$

$i \otimes (m_3 - m_9) \leq 40$

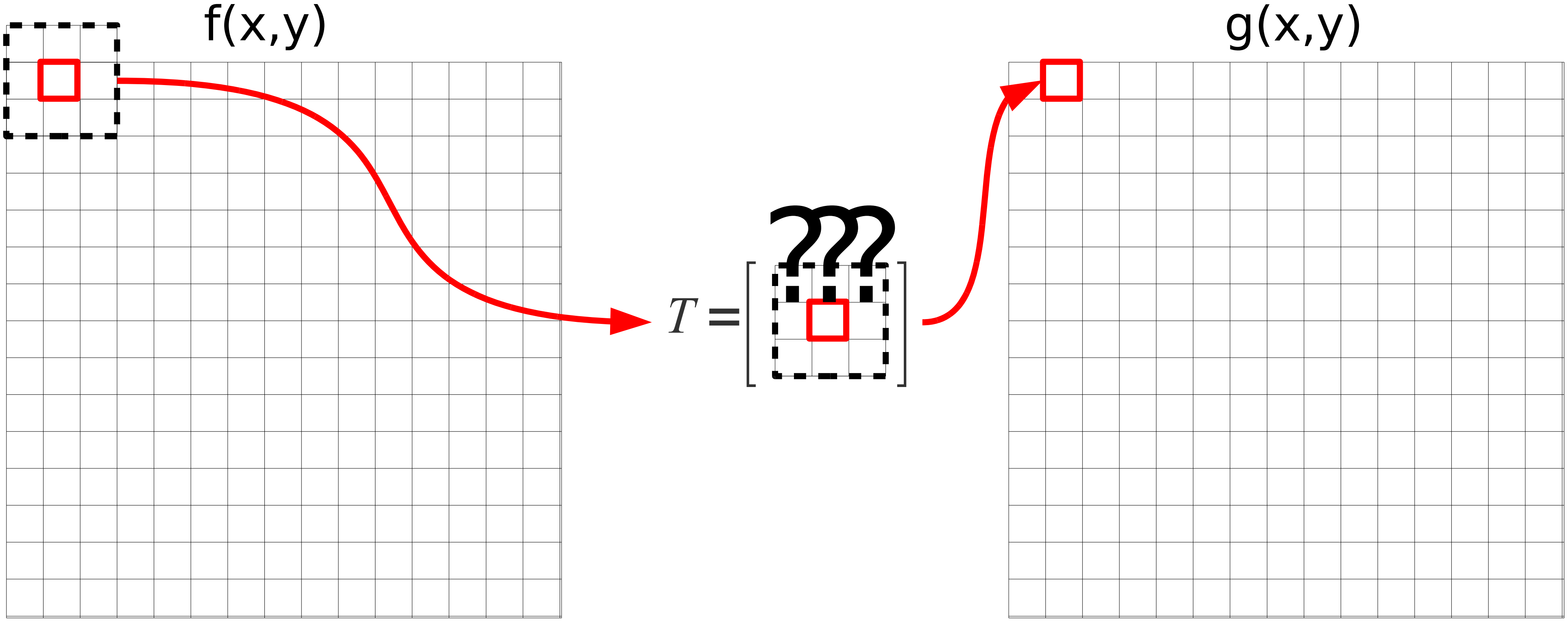



$o(x, y)$





Border handling



- Problem: Unknown image values beyond image borders
- Possible solution
  - “Shrink” output image using only available information
  - Adapt kernel shape
  - Use “default” values (0, 255)
  - Use other image information (e.g. mirroring)

2. Exercise - Given Functions

```
int main(int argc, char** argv)
```

- Main function
  - Declares variables
  - Loads original image
  - Generates and saves noisy versions
  - Tries to reduce noise by different methods
  - Usage:
    - dip2 generate path_to_original
      - Calls generateNoiseImages(...)
      - Generates and saves noisy images
    - dip2 restorate
      - Calls noiseReduction(...)

```
void generateNoisyImages(Mat& orig)
```

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

2. Exercise - Given Functions

```
void noiseReduction(Mat& src, Mat& dst,
const char* method, int kSize, int thresh)
```

- Parameter:
  - src : noisy source image
  - dst : noised reduced output image
  - method : defines method to be used
    - median, average, adaptive
  - kSize : Kernel size
  - thresh : threshold for adaptive smoothing
- Calls
  - averageFilter(...), medianFilter(...), adaptiveFilter(...)

2. Exercise – To Do

```
void spatialConvolution(Mat& src, Mat& dst, Mat& kernel)
```

- Parameter:
  - src : noisy source image
  - dst : output image
  - kernel : Kernel of the convolution

- Applies convolution in spatial domain
- Border handling
- Do **NOT** use convolution functions of OpenCV

2. Exercise – To Do

```
void averageFilter(Mat& src, Mat& dst, int kSize)
```

- Parameter:
  - src : noisy source image
  - dst : output image
  - kSize : Kernel size

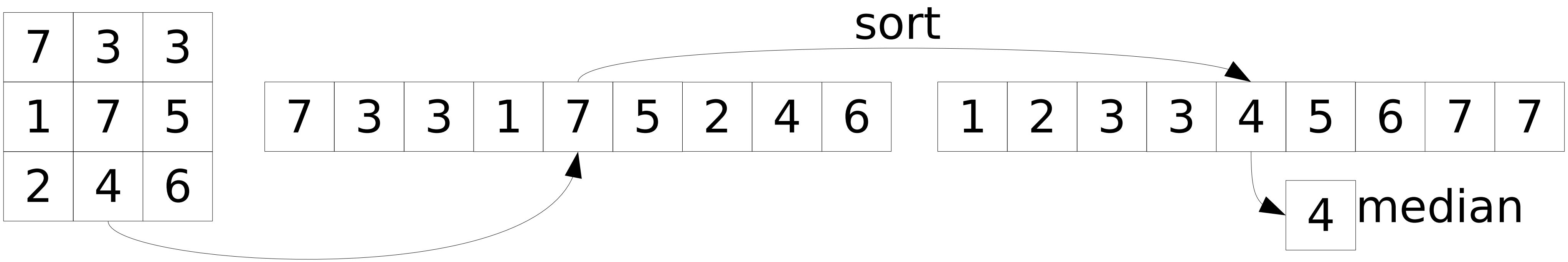
- Uses convolution to calculate local average
- Calls spatialConvolution(...)

```
void medianFilter(Mat& src, Mat& dst, int kSize)
```

- Parameter:
  - src : noisy source image
  - dst : output image
  - kSize : Kernel size

- Applies local median filtering

2. Exercise - Median



7	3	3
1	7	5
2	4	6

7	3	3	1	7	5	2	4	6

1	2	3	3	4	5	6	7	7

```
#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] << ' ';
}
```

2. Exercise – To Do

```
void adaptiveFilter(Mat& src, Mat& dst, int kSize,
double threshold);
```

- Parameter:
  - src : noisy source image
  - dst : output image
  - kSize : Kernel size
  - threshold : smooth only if difference is below this value

- Uses moving average filter, but preserves edges
- Calls averageFilter(...)

2. Exercise – To Do

- Deadline: 4th May

- **ONE** solution per group

- printout includes (red denotes mandatory material):
  → Cover stating **group id** and names
  → **Code** that was written or changed by you
  → Input, intermediate, and output images
  → Discussion of obtained results

- mail includes (red denotes mandatory material):
  → **Group id** within the mail (body or title)
  → **All program files** necessary to compile and run program
  → **Input**, intermediate, and **output images**
  → Printout as pdf-file