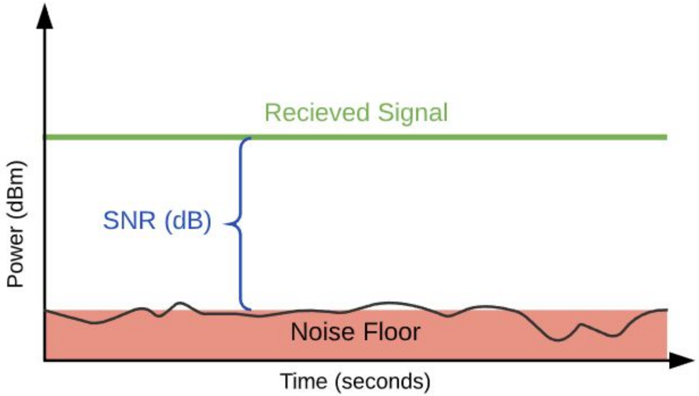
# Computer Vision

## Noise and Smoothing

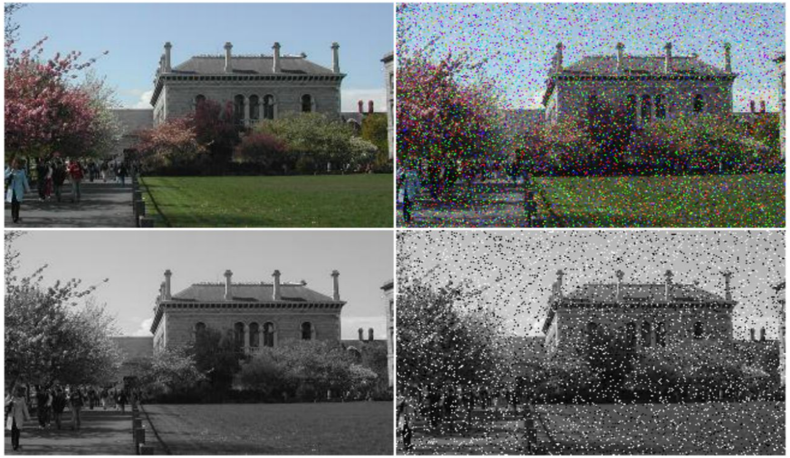
# Noise

- Affect most images
- Degrades the image
- Can cause problems with processing
- What can cause noise?
    - Measurement noise



# Salt and Pepper Noise on Image

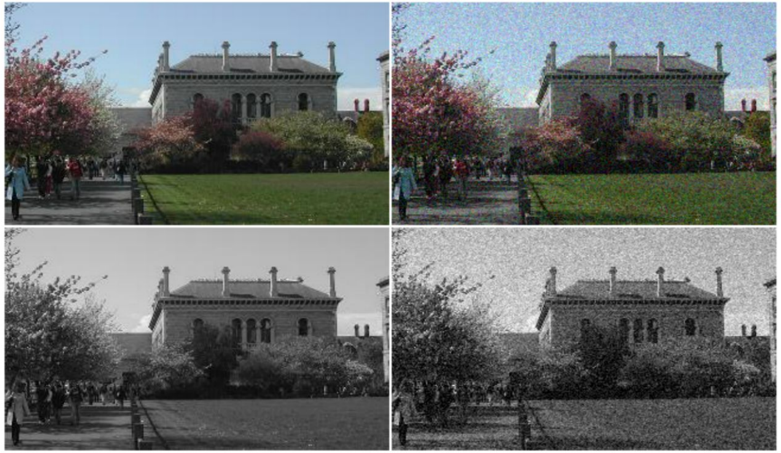
- Impulse noise - Defects in Camera Pixels
- Noise is either max or min values



Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

# Gaussian Noise in Image

- Most real noise is like Gaussian Noise
- Distribution is Gaussian
  - Mean
  - Standard Deviation

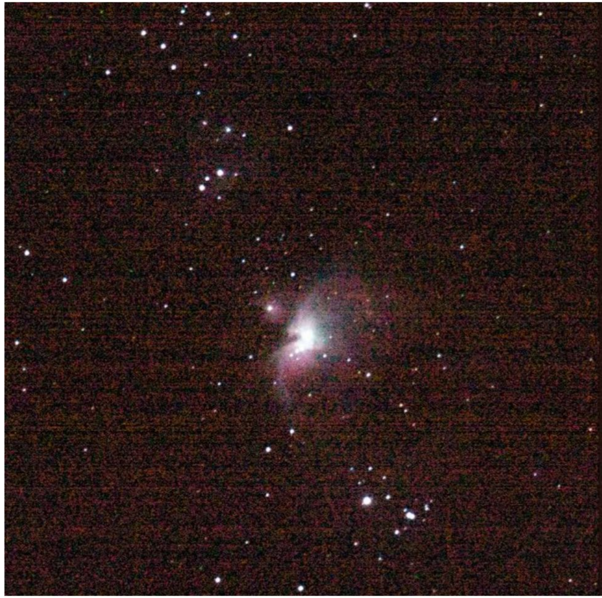


Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

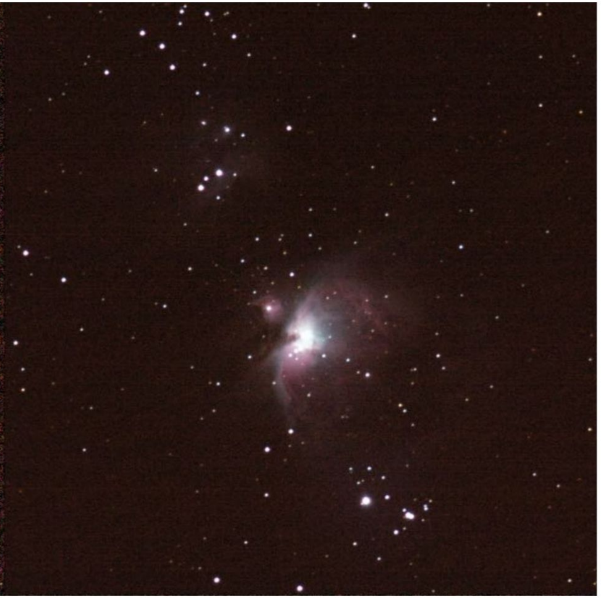
# Smoothing

- Remove or reduce noise on image
- Linear smoothing transformations
  - Image averaging
  - Local averaging
  - Gaussian smoothing
- Non-linear transformations
  - Rotating mask
  - Median filter

# Smoothing - Image Averaging



Single Image

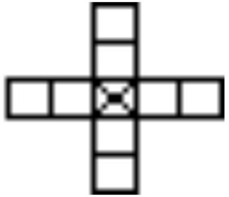


21-fold stacking

# Smoothing - Median Filter (non-linear)

- Uses the median value
- Not affected by noise
- Doesn’t blue edges much
- Damages thin lines and sharp corners

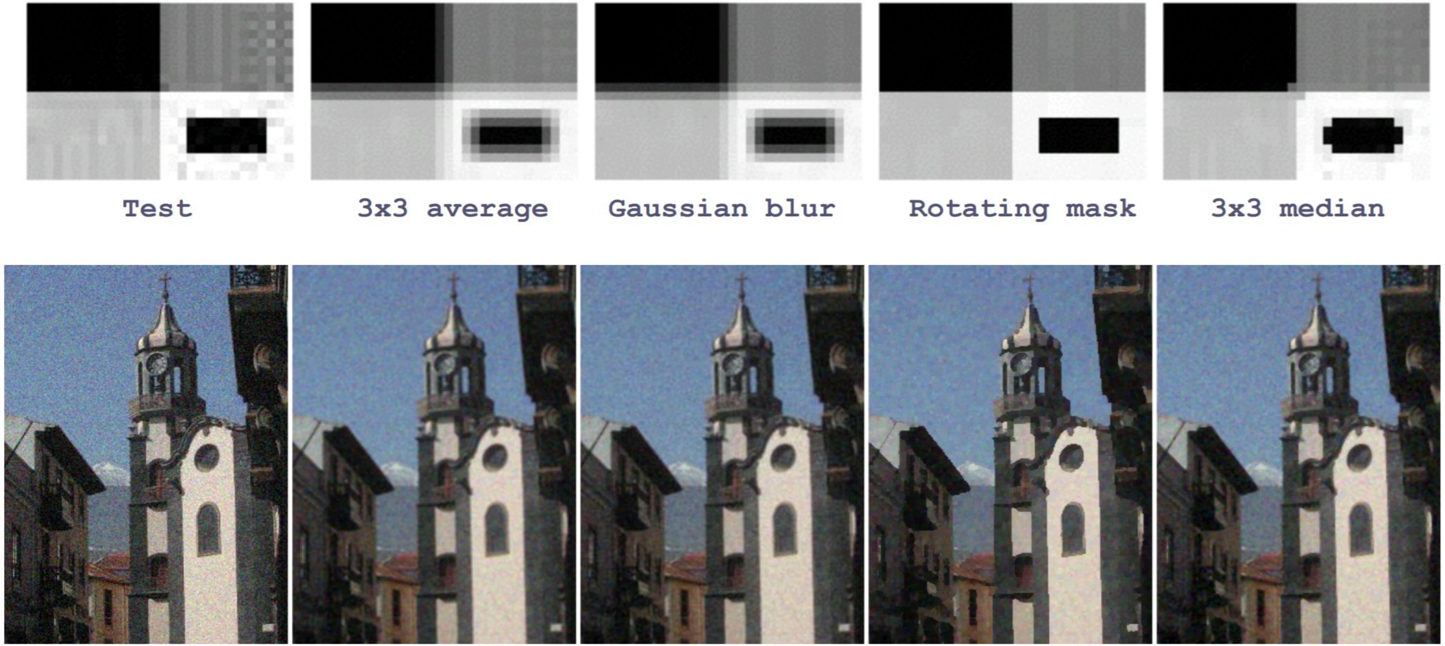
```
11   18   20   21   23   25   25   30   250
     Median = 23    Average = 47
```



```
cv::medianBlur(image, smoothed_image, 5);
```

Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

# Smoothing Examples



Test

3x3 average

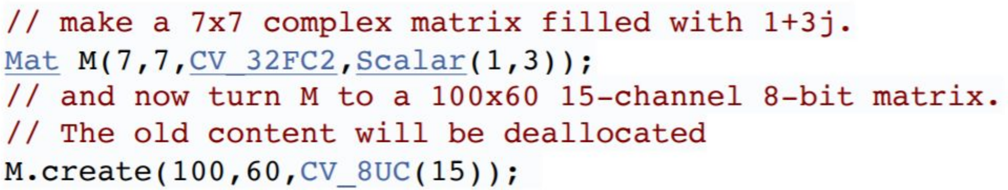
Gaussian blur

Rotating mask

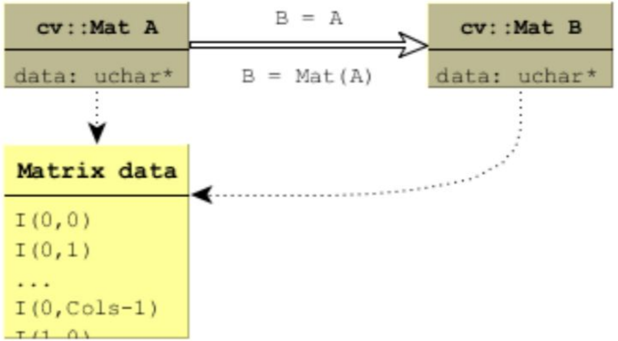
3x3 median

Based on *A Practical Introduction to Computer Vision with OpenCV* by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014

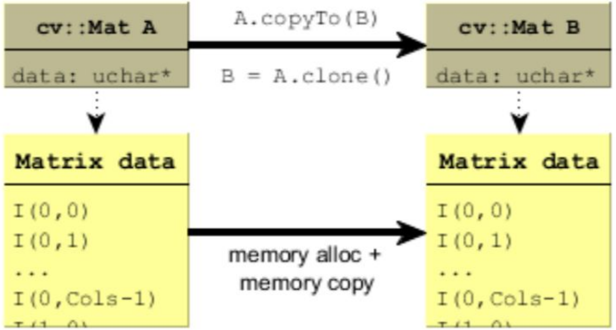
# OpenCV Hints - Matrix Creation



```
// make a 7x7 complex matrix filled with 1+3j.
Mat M(7,7,CV_32FC2,Scalar(1,3));
// and now turn M to a 100x60 15-channel 8-bit matrix.
// The old content will be deallocated
M.create(100,60,CV_8UC(15));
```



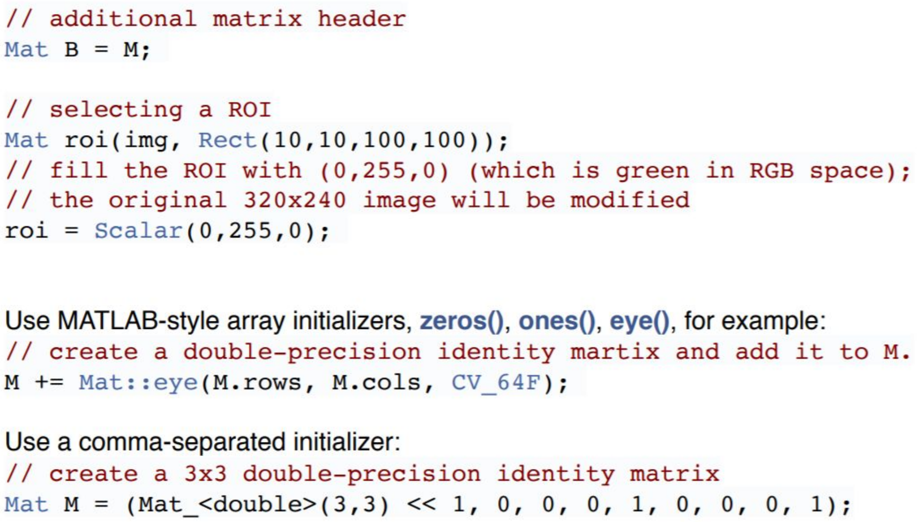
OpenCV Mat: Assignment operator or Copy Constructor



OpenCV Mat: copyTo() or clone() methods

© http://pklab.net

# OpenCV Hints - Matrix Initializers



```
// additional matrix header
Mat B = M;

// 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);
```

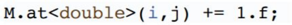
Use MATLAB-style array initializers, **zeros()**, **ones()**, **eye()**, for example:

```
// create a double-precision identity martix and add it to M.
M += Mat::eye(M.rows, M.cols, CV_64F);
```

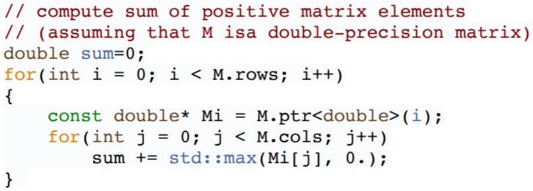
Use a comma-separated initializer:

```
// create a 3x3 double-precision identity matrix
Mat M = (Mat_<double>(3,3) << 1, 0, 0, 0, 1, 0, 0, 0, 1);
```

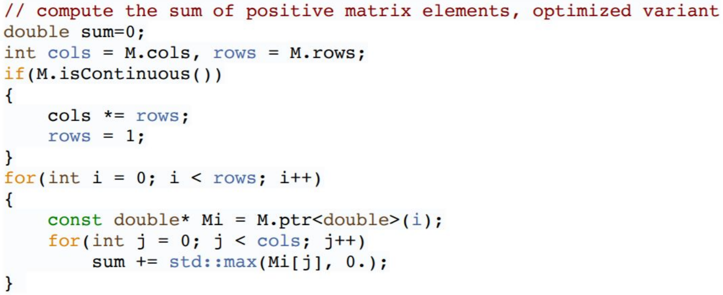
# OpenCV Hints - Access Element



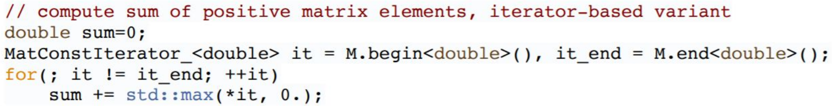
```
M.at<double>(i,j) += 1.f;
```



```
// compute sum of positive matrix elements
// (assuming that M isa double-precision matrix)
double sum=0;
for(int i = 0; i < M.rows; i++)
{
    const double* Mi = M.ptr<double>(i);
    for(int j = 0; j < M.cols; j++)
        sum += std::max(Mi[j], 0.);
}
```



```
// compute the sum of positive matrix elements, optimized variant
double sum=0;
int cols = M.cols, rows = M.rows;
if(M.isContinuous())
{
    cols *= rows;
    rows = 1;
}
for(int i = 0; i < rows; i++)
{
    const double* Mi = M.ptr<double>(i);
    for(int j = 0; j < cols; j++)
        sum += std::max(Mi[j], 0.);
}
```



```
// compute sum of positive matrix elements, iterator-based variant
double sum=0;
MatConstIterator_<double> it = M.begin<double>(), it_end = M.end<double>();
for(; it != it_end; ++it)
    sum += std::max(*it, 0.);
```