# OpenCV – Basics and some comments about „Machine Learning“

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

# Functionality - OpenCV

- „Open Source Computer Vision Library“
- Computer-Vision library with C++ and Python Interface
- Summary of modules
  - Image-Filters (Smoothing, Morphologic Filters, Histogram, Edge-Detection..)
  - Image/Video Interface (Capturing, Display, Save..)
  - Video analyses (Optical Flow, Kalman…)
  - 3D Framework (Calibration, Homography / Transformation / Projection..)
  - Classification and Clustering (Object-Detection, Feature Extraktion, GMM, K-NN, Kmeans, Trees, Neuronal Networks, SIFT/SURF…)
  - OpenCL / GPU Implementation of exhausting algorithms

# Generic Dataformat

- CV_<bit-depth>{U|S|F}C(<number_of_channels>
  - Unsigned | Signed | Float
- Example
  - unsigned char -> CV_8UC1
  - unsigned long -> CV_64UC1
  - float -> CV_32F
  - double -> CV_64F
  - 3 float tupel -> CV_32FC3
- Alternative
  - Usage of cv::DataType („trait“) class for generic C++ datatypes
  - „DataType<float>::type“ = CV_32F

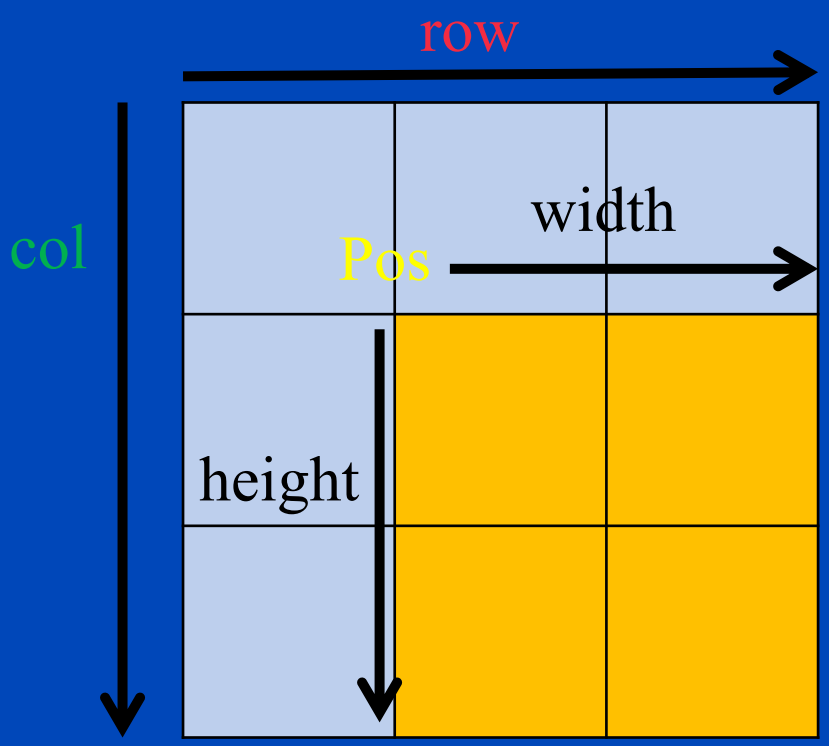
# Datastructure : Mat

```
cv::Mat matrix(row, col, CV_32FC1);
```

- Creates [ row X col ] matrix

```
matrix.at<float>(0,0) = 0.3;
```

- Access with at()-Function
  - Index starting at „0“
  - Generic C++ Datatypes as template argument




```
cv::Mat matrix2 = matrix(cv::Rect(c_pos, r_pos, width, height));
```

- Creates only header of ROI that references data of „matrix“
  -> NO „Deep-Copy“
- Deep Copy with „clone()“ or„copyTo()“
- matrix(roi) = matrix2(roi2);

```
matrix2(roi2).copyTo(matrix(roi));
```

# Datastructure : Mat (2)

- Supports STL Iteratoren -> STL Algorithms can be used
  - std::sort(mat.begin<double>(), mat.end<double>());
- Matrix Operations
  - Transpose, Matrix multiplication, Invert, Cross product
  - Determinant, Eigenvalues, Mean, StdDev
  - Element-wise operations
  - Min/Max search
  - Logic Operation
  - Cov. Matrix
  - Cartesian <-> Polar Coordinats
  - Discrete Fourier and Cos Transformation

# Further Functions

- PCA - compressPCA()
- Solve LGS - solve( X , Y, Res, Method)
  - X = | 3*a | 1* b |
        | 1.5 *a | 6 *b |

    Y = | 2 |
        | 3 |
  - Method – several algorithms for determined, over determined equations and Least-Square problem

- Solve polynomial equations
- solvePoly(X, Y, maxIter)
  - X = | $a * x^n$ | $b * x^{n-1}$ | … |
        | … | … | … |
  - -> Sorted by ascending order

# „Machine Learning“ - Overview

- Supported methods
  - Normal Bayes Classifier
  - Support Vector Machines (SVM)
  - Decision Trees , Boosting, Random Trees, Extremely Random Trees
  - Neuronal Networks
  - K-Nearest Neighbors (KNN)
  - Gaussian-Mixture-Models

# „Machine Learning“ – KNN

- K-Nearest Neighbors (KNN)
  - Number of max Nearest-Neighbors must be set befor training (Default = 32)
  - During classification the number of actually used Nearest-Neighbors can be chosen up to max NN
  - Problems with huge datasets or to high number of Nearest-Neighbors (~ 50 mio Observations á 4 Features, 15 NNs)

    -> Divide Data in several chunks an use multiple classification calls on trained KNN-Model

# „Machine Learning“ – GMM

- Gaussian-Mixture-Models
  - Is named „EM“ in OpenCV though complete GMM
  - For every Model the number of Gaussians („Clusters“) must be set for approximating the data
  - Start values for EM can be set by providing Means, Covar-Matrix and Weights
    - Weights must be column vectors – otherwise there can arise problems while intern normalization
    - Covar Matrix will only be used if Weights are also set
    - Only diagonal of Covar-Matrix is used (no correlation between dimensions)
    - Covar must be normalized (divided through number of samples)
    - Function returns index of most prob. Gaussian and log probability. Probabilities for the single Gaussians can be obtained optionally

# „Machine Learning“ – GMM

- Gaussian-Mixture-Models (2)
  - Probability of combined Gaussians has to be calculated manually. Just multiply and add up
    - Weights
    - Single Gaussian probabilities
  - It is not possible to use one GMM-Model for several classes when a class is represented by more than one Gaussian
    - Create one GMM Model per class
    - Calculate probability for combined Gaussians
    - Weight every combined Gaussians probability with a prior probability for that class
    - Choose class with highest probability (see “Bayes” classification)

# Questions, Comments, Discussion ?!