Automated Feature Extraction with Machine Learning and Image Processing

Prof. Dr. Stefan Bosse

University of Siegen - Dept. Maschinenbau
University of Koblenz - Dept. Computer Science

Image Feature Extraction and Marking

In this module we learn the basic workflow of image processing consisting of:

1. Image feature marking: Image → Image
    - Cellular Automata
    - Multiagent Systems
    - Convolution and Convolutional Filtering
    - CNN Pixel Classifier
2. Image feature clustering (Point list → List of point lists)
    - Density-based Clustering
3. Geometric processing of point clouds (Point list → Point list)
    - Hull Computation

Cellular Automata and Image Processing

How can we apply algorithms to image data?

Cellular Automata and Image Processing

How can we apply algorithms to image data?

Relation between Images and Cellular Automata - local versa global state processing

Cellular Automata and Image Processing

How can we apply algorithms to image data?

Relation between Images and Cellular Automata - local versa global state processing

From local to global state using Point Clustering and Polygon Hulls

Cellular Automata and Image Processing

How can we apply algorithms to image data?

Relation between Images and Cellular Automata - local versa global state processing

From local to global state using Point Clustering and Polygon Hulls

Kernel-based Transformations and Convolutional Neural Networks!

Cellular Automaton

- A cellular automaton (CA) is basically an active (functional) matrix
- Applications:
  - Analysis and exploration of complex space-time dynamics and of the behavior of dynamical systems
  - Image transformations (state-less single-step and state-based/iterative)
  - Feature Analysis (Extraction, Amplification)
  - Parallel computation model!

Cellular Automaton

- Basic assumption:

> Complex phenomena are the result of the collective dynamics of a very large number of parts obeying simple rules

Cellular Automaton

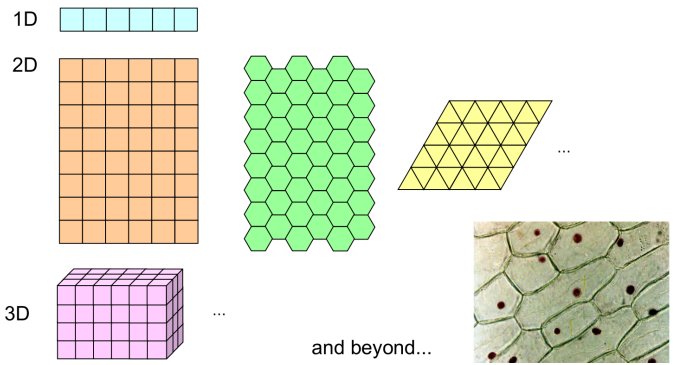
We want to define the simplest nontrivial model of a cellular system. We base our model on the following concepts:

- Cell and cellular space
- Neighborhood (local interaction)
- Cell state
- Transition rule

- We do not model all the details and characteristics of biological multicellular organisms but we obtain simple models where many interesting global and local phenomena can be observed.

Cellular Space

- The cellular space constructs the cell network featuring
  - Dimension
  - Size
  - Neighbourhood



[Floreano et al.]

Cellular Automaton

- A cellular automaton (CA) consists of a set of state-based cells $CA=\{c_1,c_2,..,c_n\}$.
- Each cell *i* has a number of cell variables $\{x_1,..,x_k\}_i$
- The variables define the data state *S*: $S(z_i)=S_i=\{x_1,..,x_k\}$
- The next state of the cells is calculated via an *activity* function *Act*.
- The state transition of a cell usually takes place in the *activity* function, optionally prepared by *before* (*pre*) or *after* (*post*) functions.

Scheduling

- A CA is a parallel computational architecture. All cell state transitions can be computed in parallel at the same time.
- But CA processing by generic computers requires a (semi-) sequential algorithm to perform cell state transitions ⇒ **CA simulation** with a scheduler
- In a parallel system the following formula is valid for *each cell* (①right hand side, ②left hand side):

$$S_i(t+1) = Act\big(S_i(t), \{S_j(t) | j \neq i \quad \text{and} \quad |j| \leq R\}\big)$$

- Sequential computation would modify right hand side input before used!
- Therefore, the old state of cells must be saved before the new state is computed!

- A scheduler processes the individual calculation functions of all cells sequentially in three phases:

  1. *before*: ∀ cells do *pre*(*S*)
  2. *activity*: ∀ cells do *Act*(*S*)
  3. *after*: ∀ cells do *post*(*S*)

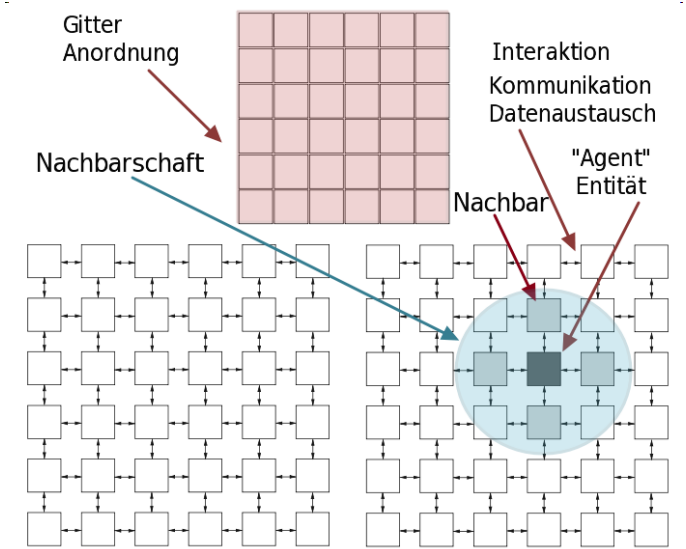
$$pre(S) : S_i \rightarrow S_{i,0}$$
$$Act(S) : S_{i,0} \times \Sigma \rightarrow S_i$$
$$post(S) : S_i \rightarrow S_i$$

- with Σ as the state of neighbourhood cells (or global state in general):

$$\Sigma(R)_i = \left\{ S_j | j \neq i \quad \text{and} \quad |j| \leq R \right\}$$

Architecture and basic principle

- The Cellular Automaton is a simulation tool that represents an artificial two-dimensional world of entities!
- For visualization, each cell gets a color from a defined color palette



Cell States

There is a

Data State
Defined by the set of all cell states (variables), visible

Control State
The cell activation (update) sequence, hidden (Scheduler)

Typically, the data state can be compsoed of:

- Input space X, e.g., a mapping of image pixels on cells 1:1
- Intermediate hidden states T, i.e., auxiliary and temporary variables, e.g., $t{=}x;x{=}y,y{=}t$
- Output space Y, e.g., mapping of cell states on an output image (pixels)

Cell Neighbourhood

- Informally, it is the set of cells that can influence directly a given cell
- In homogeneous cellular models it has the same shape for all cells
- Neighbourhood defined some kind of communication (data transfer, but w/o synchronisation)

Moore

Moore neighbourhood is a squared region around a center cell position with a radius R.

Neumann

Neumann neighbourhood is an approximated "circular" region around a center cell position with a radius R. R=1 neighbourhood defines only the direct neighbours North, South, West, East.

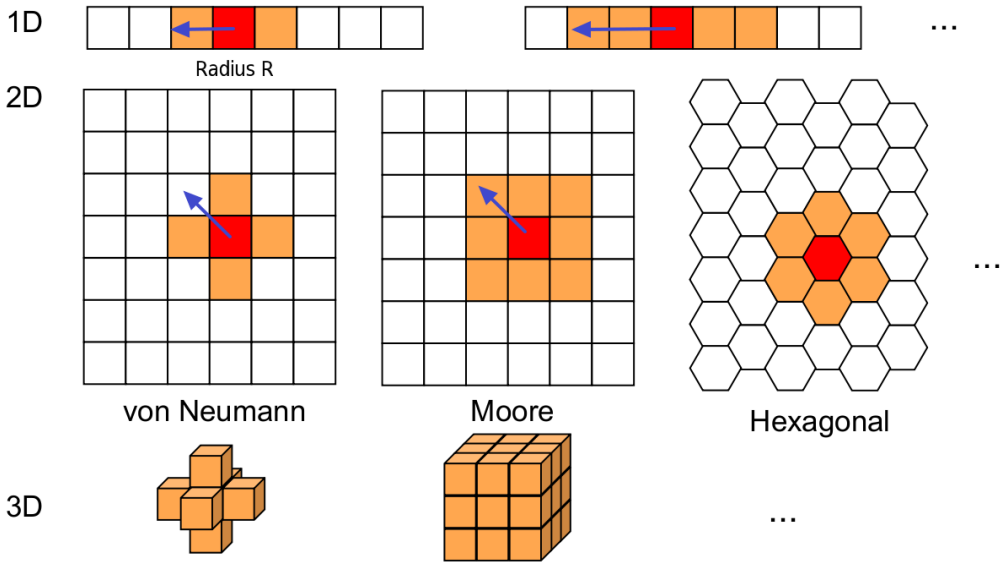


Fig. 2. Comparison of Moore and Neumann neighbourhood in different space dimensions

Cell State Transitions

- The value of the state of each cell belong to a finite set, whose elements are commonly numbers.
- The value of the state is often represented by cell colors. A *color* function maps cell states on colors (from a palette)
- There can be a special quiescent state s0.
- The transition rule is the fundamental element of the CA.
- It must specify the new state corresponding to each possible configuration of states of the cells in the neighborhood.

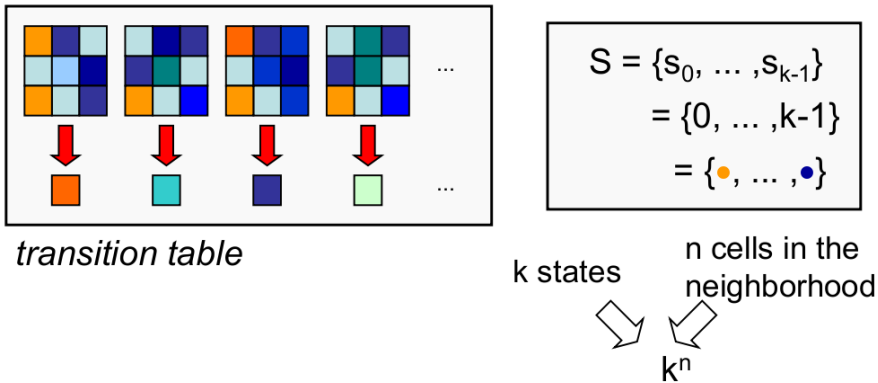


Fig. 3. A transition rule maps states of neighbouring cells on new cell states. Cell states are visualized by colors.

Cell updates by transition rules modify the cell states over (discrete) time. In order to start with the updating of the cells of the CA we must specify the initial state of the cells (initial conditions or seed)

[Floreano et al.]

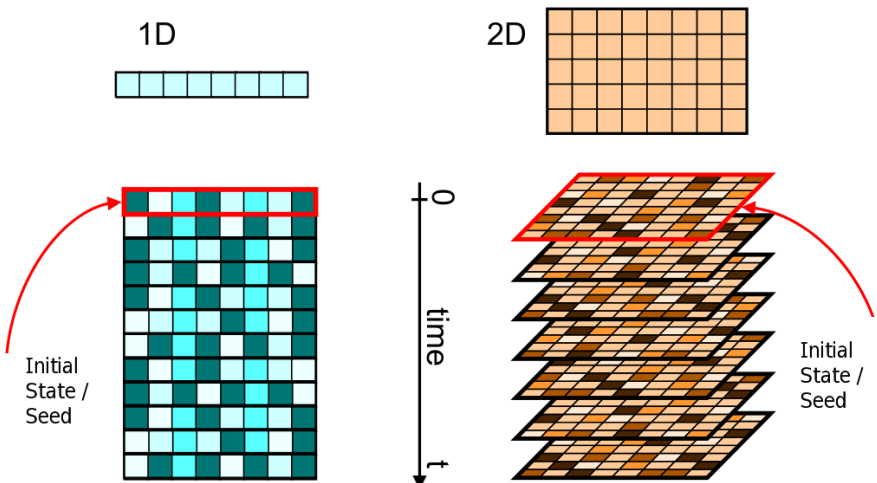


Image processing with CA

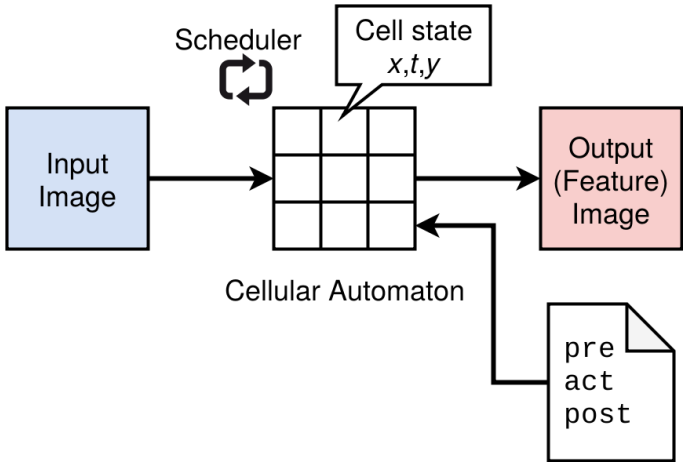


Fig. 4. An input image is mapped 1:1 on a cell matrix. The CA activation (iteratively) modifies the state of the cells, finally mapped 1:1 on an output (feature) image.

Kernel-based Transformations

- Convolution or folding is a linear operation applied to each CA cell → Linear and state-less CA transition rules
- Common kernel-based transformations:
  - Denoising
  - Intensity gradient
  - Edge detection / amplification
- A kernel-based transformation produces always a linear output:

$$I^{\star}(x,y) = \sum_i \sum_j I(x-i+a, y-j+b) k(i,j)$$

- with *k*: kernal/folding matrix, *a*,*b*: index of center point of folding matrix *k*

Effect	Kernel
Average	$\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$
Sharpening	$\begin{pmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{pmatrix}$
Relief	$\begin{pmatrix}-2&-1&0\\-1&1&1\\0&1&2\end{pmatrix}$

Effect	Kernel
Edge, Laplace	$\begin{pmatrix}0&1&0\\1&4&1\\0&1&0\end{pmatrix}$
Edge, Sobel	$\begin{pmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{pmatrix}$
Noise?, Bosse	$\begin{pmatrix}-1&-1&0\\0&0&1\\1&0&0\end{pmatrix}$

Most kernel-based edge and gradient computations are sensitive in a specific direction (axis).

- E.g., there are two Sobel edge filter kernel, one for the x-, and one for the y-axis sensitive, finally combined:

$$S^x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

$$S^y = S^{x^T} = \begin{pmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

$$z_{x,y} = \sqrt{\left(\sum_{k=1}^{3} S^x_{k,l} z_{x+k-1,y+l-1}\right)^2 + \left(\sum_{l=1}^{3} S^y_{k,l} z_{x+k-1,y+l-1}\right)^2}$$

Iterative Transformations

- Kernel-based transformations usually terminates after one step, except if the input is the output of the previous update phase!
- In iterative transformations, the input of the next update phase is the output from the last (or the initial seed image data)

Edge Detection with CA

For edge detection of intensity images, some approaches first convert the intensity images into binary ones, and then evolve two-state cellular automata using specific state transition rules to determine edge pixels, while the others directly update pixel states based on the relationship of the central pixel with its neighbourhood, mostly a 1-ring von Neumann or Moore neighbourhood.

Popovici et al. proposed an edge detection approach based on the state differences between the central pixel and the pixels in its von Neumann neighbourhood. If all the absolute state differences are less than a threshold ε, then the state of the central pixel becomes 0, otherwise it remains unaltered.

Edge Detection with CA

- The rule can be formulated as:

$$s_i^+ = \begin{cases} 0 & |s_j - s_i| < \delta, \forall j \in N_i \\ s_i & \text{otherwise} \end{cases}$$

- where s and s^+ are the current and the updated states of the central cell i, N_i is the von Neumann neighbourhood of the cell c, and s_j is the current state value of the cell j in N_i.

- In the case of feature marking, the conditional expression || < δ can be applied to other input variables (input image), and the state s represents a marked feature count only. But in this case it is a one-step update.

Edge Detection with CA

Wongthanavasu and Sadananda proposed a conditional rule to update the cellular state as:

$$s_i^+ = \begin{cases} s_i & s_i < (s_{\max} - s_{\min}) \\ s_{\max} - s_{\min} & \text{otherwise} \end{cases}$$

$$s_{\max} = \max\left\{s_j | j \in N_i\right\}, s_{\min} = \min\left\{s_j | j \in N_i\right\}$$

- where $s_{\max}$ and $s_{\min}$ are the maximum and minimum states, respectively, in the von Neumann neighbourhood N_{i} of the central cell i.

Agent-based Image Processing: Feature Marking

Lui proposed a similar approach of neighbourhood exploration for feature marking in mages using mobile agents.

- In contrast to Ca cells, the state of an agent is not fixed to a specific position.
- There are explorations agents with the following behaviour:
  - **Replication**. If the agent finds a feature at the current position, it stays (or increasing the feature count), then replicate a specific number *r* of child agents that migrate to a neighboruing cell (randomly chosen).
  - **Diffusion**. If the agent do not find a feature, it migrates to neighboruing cell (randomly chosen).
  - The number of diffusions and hops is limited.
  - At each new place the feature detection starts again.

- Conditional expression for feature detection:

$$s_i^+ = s_i + \begin{cases} 1 & \left(\sum |v_j - v_i| < \delta, \forall j \in N_i\right) \in [\epsilon_1, \epsilon_2] \\ 0 & \text{otherwise} \end{cases}$$

- with *v* as the input pixel intensity and *s* as the feature marking count.

CA Implementation of FM-MAS

```
state : { input:0, output:0, weight:1, deltaweight:0 },
activity : function (x,y) {
 // neighbors: ordering [NW,N,NW,W,E,SW,S,SE]
 var countdn = this.ask(this.neighbors, 'absdiffcount', 'input',this.delta,'<='),
     mark  = countdn>=this.eps1 && countdn <= this.eps2;
 if (mark) {
   this.output += this.weight;
   // simulate new agent replications on heighbor nodes by increasing weight
   for (var i=0;i<this.replicate*this.weight;i++) {
     var delta = [0,0];
     while (delta[0]==0 && delta[1]==0)
       delta=[Math.random.select([-1,0,1]),Math.random.select([-1,0,1])];
     this.set(delta,'deltaweight',this.get(delta,'deltaweight')+1);
   }
  } else if (this.weight>0) {
    this.deltaweight--;
    // simulate aagent diffusion to heighbor node by increasing weight
    var delta = [0,0];
    while (delta[0]==0 && delta[1]==0)
      delta=[Math.random.select([-1,0,1]),Math.random.select([-1,0,1])];
    this.set(delta,'deltaweight',this.get(delta,'deltaweight')+1);
  }
},
after : function (x,y) {
  this.weight += this.deltaweight;
  this.deltaweight=0;
},
```

Learning CA

Which transition rules leads to a specific (global) emergent behaviour, e.g., edge detection?

Learning CA

Which transition rules leads to a specific (global) emergent behaviour, e.g., edge detection?

Besides classical folding kernels, the answer is unknown.

Learning CA

Which transition rules leads to a specific (global) emergent behaviour, e.g., edge detection?

Besides classical folding kernels, the answer is unknown.

Solution: Learn the rules, i.e., the kernel matrix! E.g., by using evolutinary and genetic algorithms.

Density-based Pointcloud Clustering DBSCAN

- The output of the CA is an image with pixels representing point-wise features
- But point clouds are still high-dimensioal
- Dimensionality reduction by finding groups of points close together ⇒ **Clustering**

Density-based spatial clustering of applications with noise (DBSCAN) is a well-known data clustering algorithm that is commonly used in data mining and machine learning.

Algorithm

- Based on a set of points (let's think in a bidimensional space as exemplified in the figure), DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points.
- It also marks as outliers the points that are in low-density regions.

Input

List of all point coordinates, can be pre-clustered in independent lists if the point belong to different classes

Output

List of point index lists with respect to the original input list. Each sub-list is a cluster group.

[https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html]

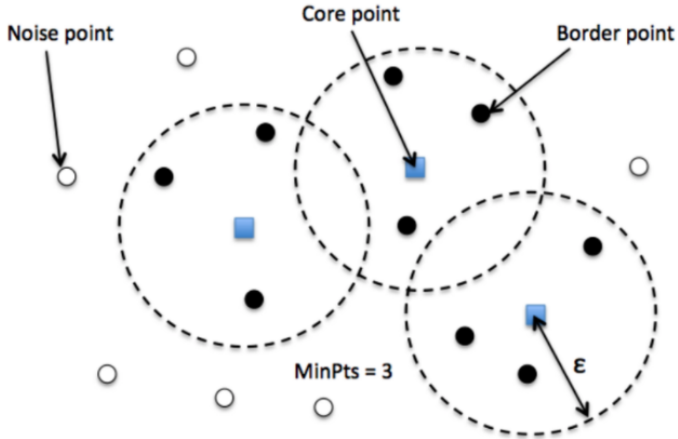


Fig. 5. Points: Core - This is a point that has at least m points within distance n from itself. Border - This is a point that has at least one core point at a distance n. Noise - This is a point that is neither a Core nor a Border. And it has less than m points within distance n from itself.

1. The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).

2. If there are at least ‘minPoint’ points within a radius of ‘ε’ to the point then we consider all these points to be part of the same cluster.

3. The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point

Parameters

The DBSCAN algorithm basically requires 2 parameters:

eps [https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80]

specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.

minPoints

the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

Parameter estimation

The parameter estimation is a problem for every data mining task. To choose good parameters we need to understand how they are used and have at least a basic previous knowledge about the data set that will be used.

eps

if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

minPoints

As a general rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints ≥ D + 1. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.

More information:

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996, August). A density-based algorithm for discovering clusters in large spatial databases with noise. In Kdd (Vol. 96, №34, pp. 226–231).

Polygon Hulls

- Up to here, we have still point clouds, although clustered.
- For, e.g., area or perimeter computation of a cluster, a (concave) polygon hull approximation is needed (simplifying, averaging, and denoising point cloud)
- Well known algorithms and software:
  - Graham Scan
  - Polylidar

The hull may not be limited to monotonic hulls without holes and caves (mixing of convex and concave hull segments)!

- A hull can be described by a list of ordered boundary points
- A hull can be described by a list of pice-wise linear lines (between boundary points)

[https://gis.stackexchange.com/questions/143821/how-to-find-the-concave-hull-for-a-cloud-of-points-in-3d-space]

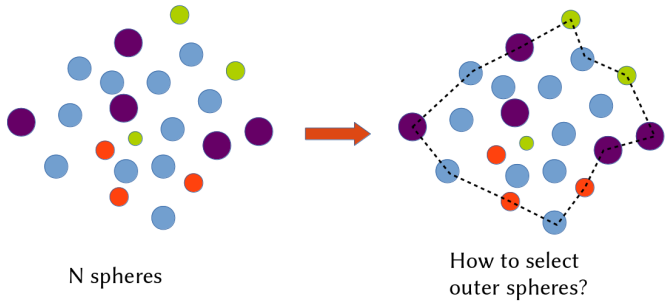


Fig. 6. The outline of a point cloud as a concave hull

Further readings:

1. Implementation of a fast and efficient concave hull algorithm, http://www.it.uu.se/edu/course/homepage/projektTDB/ht13/project10/Project-10-report.pdf

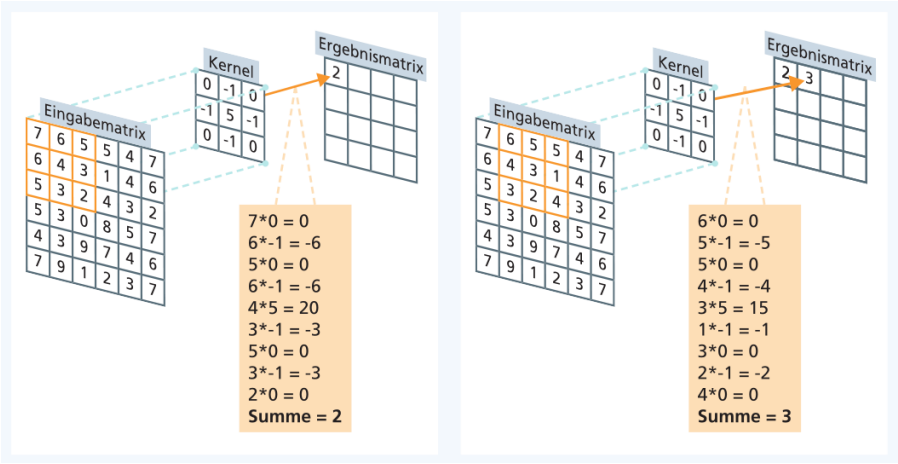
Convolutional Neural Networks

- CNN are suitable for classifying different structural features in the data regardless of location and orientation
- A CNN is based on matrix algebra with convolution operations

For further reading: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529?gi=521747216671; G. Paaß, Artificial Intelligence, What is behind the technology of the future?, Jumper

- Well-known software framework for the browser: convnet.js https://cs.stanford.edu/people/karpathy/convnetjs

Folding Operation

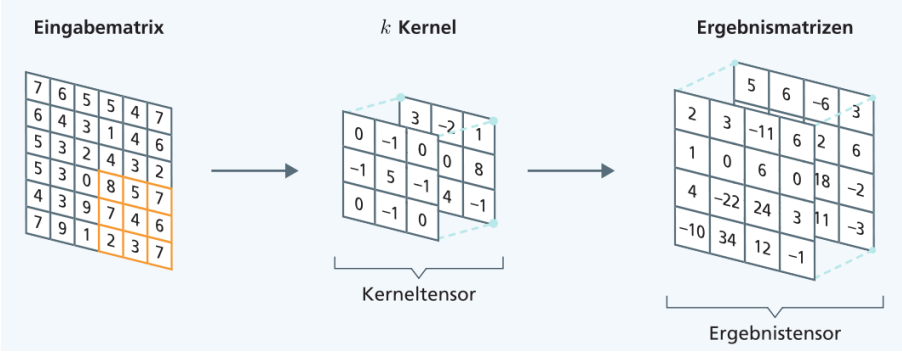


[20]

Fig. 7. First calculation step (left) and second calculation step (right) in the convolution layer for a shifted small area of the input matrix. The kernel is "pushed" successively over the entire input matrix and the result matrix is filled

Convolutional Neural Networks

- A CNN is composed of different layers:
  - Convolution layers
  - Merging (pooling) layers
  - Output by classification layers (softmax)



[20]

Fig. 8. A convolution layer contains k kernels and k result matrices, which are each combined into tensors

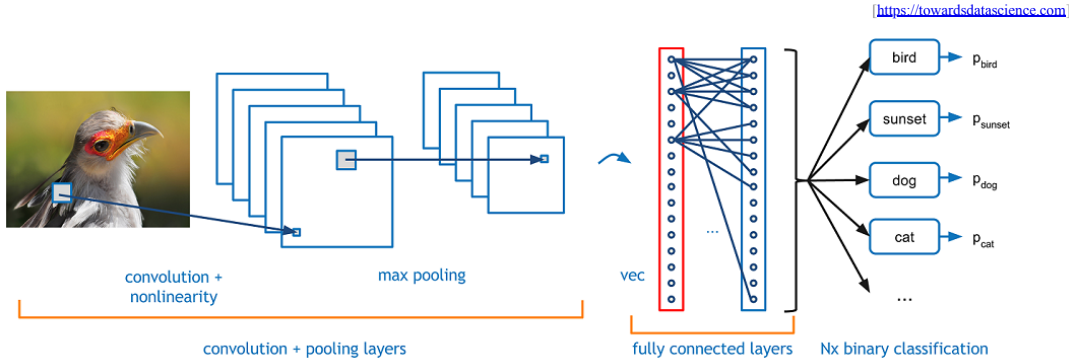


Fig. 9. General construction of a CNN with alternating layers of convolutions, merging, and finally binary classification

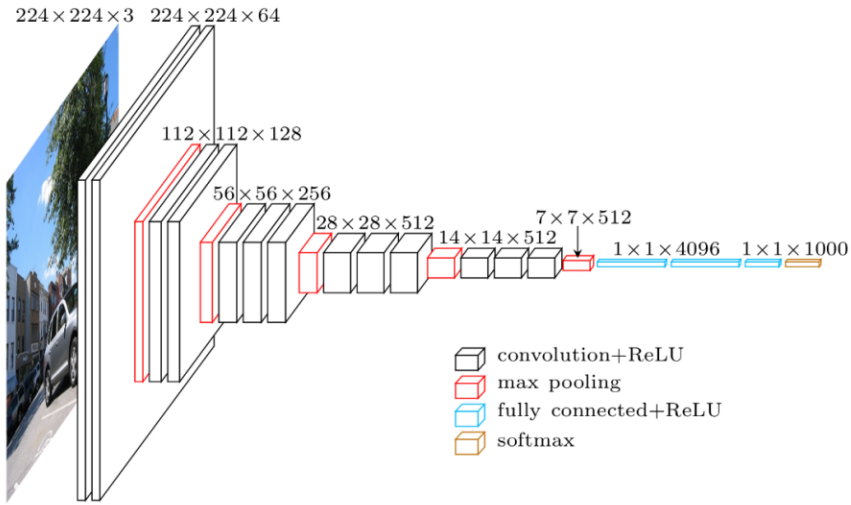


Fig. 10. Depending on the number of structural features, there may be a large number of the following folding and merging layers

CNN Point Classifier

- As with the CA approach, a CNN can be used to predict feature output images ⇒ **CNN Image Feature Marking**
- The idea: A CNN is used to predict the feature class (e.g., marking of pixel belonging to a crack shape) with a small moving segment window.
  - The central pixel of this segment (e.g., (10,10) in a 20 × 20 pixel segment) is the pixel for which the CNN should predict the feature classification

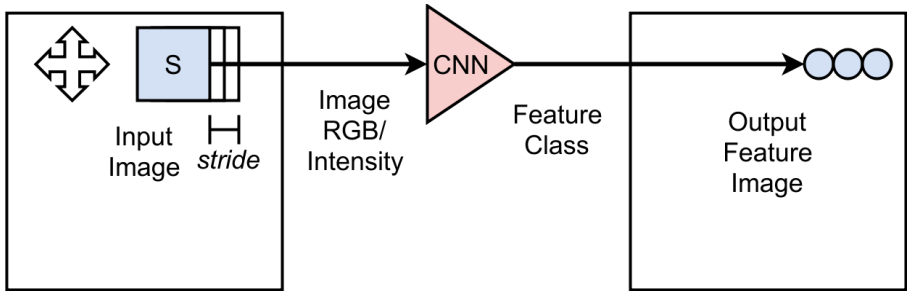


Fig. 11. Iterative image pixel feature marking by a moving segment window

- Training is supervised with labelled input data.
- Training and prediction poses high computational complexity.
- The CNN can be simple, e.g.: two convoultional layers with 8 and 16 filter, respectively, one soft-max layer. But even such a simple CNN has about 10000 parameters!

Training

We need a set of labelled images. The regions with a feature are described with closed polygon paths. For each input image of size $w \times h$, we get a feature map image of size $w \times h$ pixels. A randomly selected and shuffled segment set is used for training (created in advance). Segmentation can be strided (*stride*=Δx,Δy)

Prediction

The model is applied to all pixels of an input image with optional striding. The result is a feature map images of size $w/stride \times h/stride$ pixels.

Example

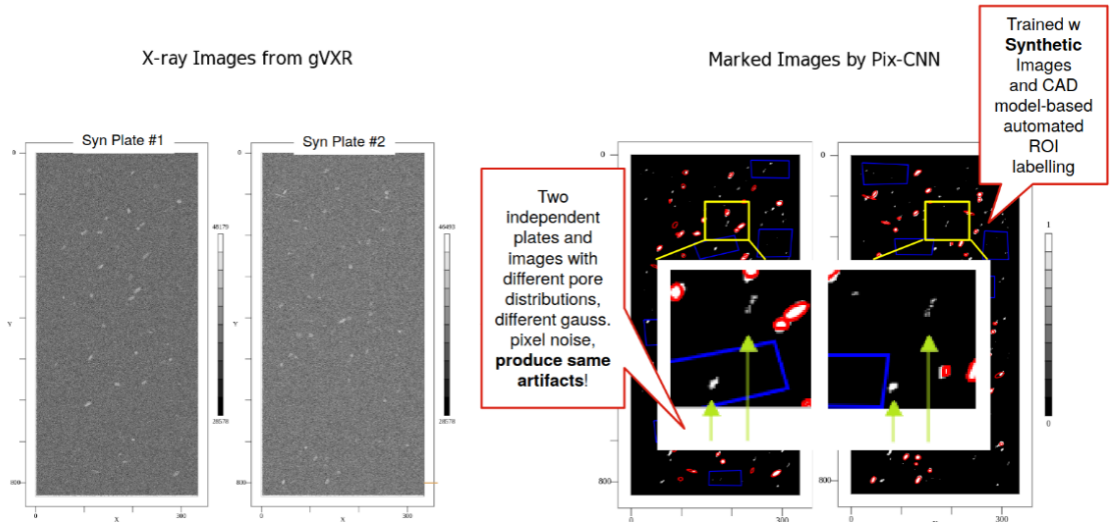


Fig. 12. Feature marking with a trained CNN pixel classifier of pores in X-ray images (Aluminum diecasted plates, here synthetic X-ray images simulated with gVXR)

Summary

In this module we learnrd the basic workflow of feature extraction with image processing consisting of:

1. Image feature marking: Image → Image
   - Cellular Automata
   - Multiagent Systems
   - Convolution and Convolutional Filtering
   - CNN Pixel Classifier
2. Image feature clustering (Point list → List of point lists)
   - Density-based Clustering
3. Geometric processing of point clouds (Point list → Point list)
   - Hull Computation
4. Finally we can compute aggregate paramters (from hull):
   - Mass of Center (MoC)
   - Geometric model fit (e.g., ellipse or ellipsoid fit), ellipse parameters
   - Area, Volume

Further Reading

1. Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies by Dario Floreano and Claudio Mattiussi, MIT Press