Automated Feature Extraction with Machine Learning and Image Processing

PD Stefan Bosse

University of Siegen - Dept. Maschinenbau
University of Bremen - Dept. Mathematics and Computer Science

Machine Learning in Image Processing

Feature Classes

> Machine Learning is commonly a data-driven approximation of a functional model $f(x): x \rightarrow y$, with x as input and +y* as output (target) features.

There are basically two different ML tasks:

1. Classification ⇒ Symbolic / categorical and discrete target feature variables
2. Regression ⇒ Numeric and continuous target feature variables

Feature Classes

Examples for categorical features:

- Damage (boolean decision, classification of damages like cracks, delaminations, and so on)
- Quality assessment (boolean decision, grade levels, class A/B/C, and so on)
- Geometrical objects (shapes, like lines, circles, ellipses)

Examples for numerical features:

- Material density, average pore size and/or density, crack length and/or density, and so on
- Mechanical properties (stiffness, homogeneity, and so on)
- Predictive lifetime
- Statistical aggregates (noise, average inhomogeneity, average size or density of impurities)

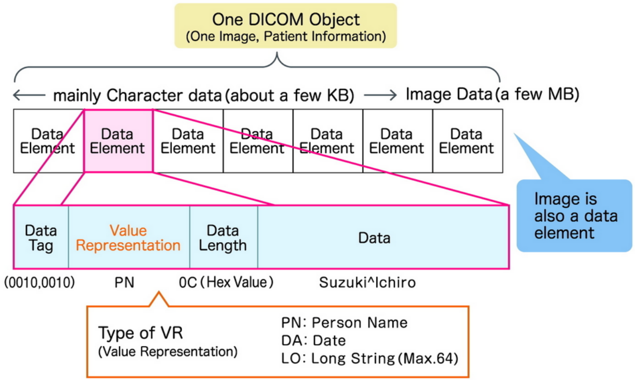
Image Classes

1. Two-dimensional intensity (photography) images (intensity represents surface reflection or material transmission)
    - Gray-level or multi-channel color (RGB, red, green, and blue channels);
    - Typical dimension 1000×1000 pixels;
    - Typical intensity resolution 8 bit (256 levels, gray or RGB), high-quality 16 bit (65536 levels)
    - Volume dimension: $height \times width \times channels$
    - Common data file formats: PNG, BMP, TIFF (not JPEG: irreversible compression creating image artifacts)

Image Classes

2. Three-dimensional tomography images (intensity represents material density)
    - Gray-level
    - Sliced image stack
    - Typical dimension 1000 × 1000 pixels × 1000 (100) pixels;
    - Volume dimension: *height* × *width* × *depth*
    - Common data file formats: numpy, ZIP of tiff files, Vol3, RAW, DICOM (medicine) ..

Image Classes



[https://www.matsusada.com/column/ct-tech2.html]

Fig. 1. DICOM CT scan data file format merging meta and raw image data

Image Feature Extraction

Target output features can be predicted by the data-driven model basically in two ways:

1. Using raw image input data;
2. Using computed (intermediate) image features.

Examples of computed image features:

- Image intensity distribution, inhomogeneity
- Detection and characterisation of geometrical object shapes (e.g., circles)
- Image transformations, i.e.,
  - wave-number frequency transformation,
  - gradient amplification using convolutional filter operations,
  - binarisation using a threshold

Image Feature Extraction

Example: Pore characterisation in Microsclices of AM parts

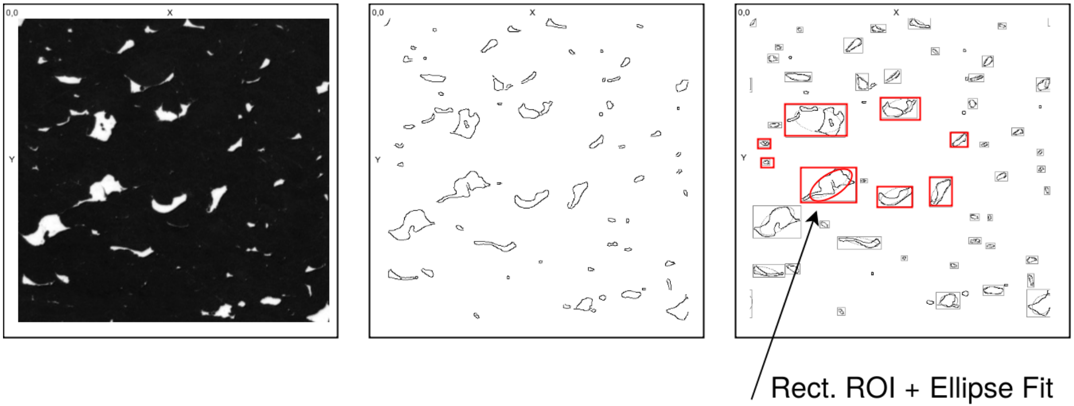


Fig. 2. (Left) Micrograph image of a material slice with pores/impurities (Middle) Edge detection using a Canny filter (Right) Shape characterisation by ellipse fitting

Image Feature Extraction

Example: Pore characterisation in X-ray images from die casted plates

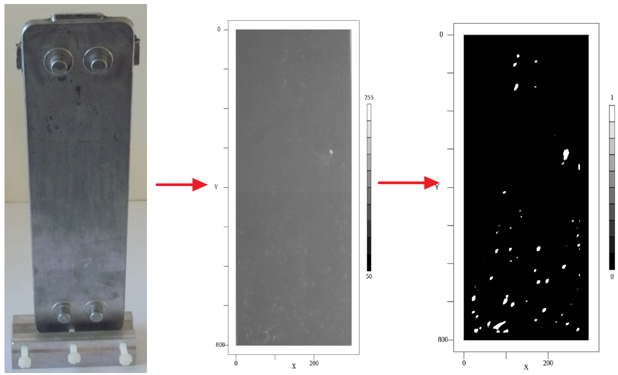


Fig. 3. (left) Die casted aluminum plate with pores (Center) Single projection X-ray image of plate (Right) Pore marking by semantic pixel classifier (white=pore feature)

Workflow for Object Feature Detection

Data: Micrograph images of material slices from rectangular probes proced with Additive Manufacturing technologies (metal powder laser melting).

Object Features: Elliptical pores (impurities) characterised by varying size (axis lengths of ellipse), orientation, density, and spatial distribution (inhomogeneity)

Target Features: Statistical and geometrical characterisation of pores, material density, distribution of defects

> ⚠ Feature extraction should be scale, intensity, and position invariant! I.e., object detection should be possible for objects of different sizes, orientation, and position within the images and different image exposures.

Workflow for Object Feature Detection

1. Threshold binarisation of micrograph images
2. Application of Canny Filter to extract pore edges / boundaries (Parameter selection!)
3. Creation of a linear point list (coordinates of marked boundary points of pores)
4. Density-based Clustering (DBSCAN) to get groups of points belonging to one pore object
5. ROI bounding box approximation for each point cluster group (iterative expansion and shrinking)
6. Ellipse fitting (direct algebraic method), feature calculation (axis lengths, orientation, area)
7. Statistical characterisation

Image Binarization

$$I_b^1(x,y) = \begin{cases} 1 & I(x,y) \geq I_{thr} \\ 0 & I(x,y) < I_{thr} \end{cases}$$

$$I_b^0(x,y) = \begin{cases} 1 & I(x,y) \leq I_{thr} \\ 0 & I(x,y) > I_{thr} \end{cases}$$

Image Binarization

```
i1 = load.image('http://edu-9.de/uploads/assets/pores_microsl
m1 = as.matrix(i1,mode='uint8')
print(minMax(m1[100:200,100:200]))
m1.binary = matrix(255,nrow(m1),ncol(m1),mode='uint8')
m1.binary[m1>100]=0
plot(m1.binary)
```

Ex. 1. R+ microslice image pre-processing: Normalization and Binarization (0/255)

Edge Detection with Canny Filter

[DOI:10.1109/ICSMC.2009.5346873]

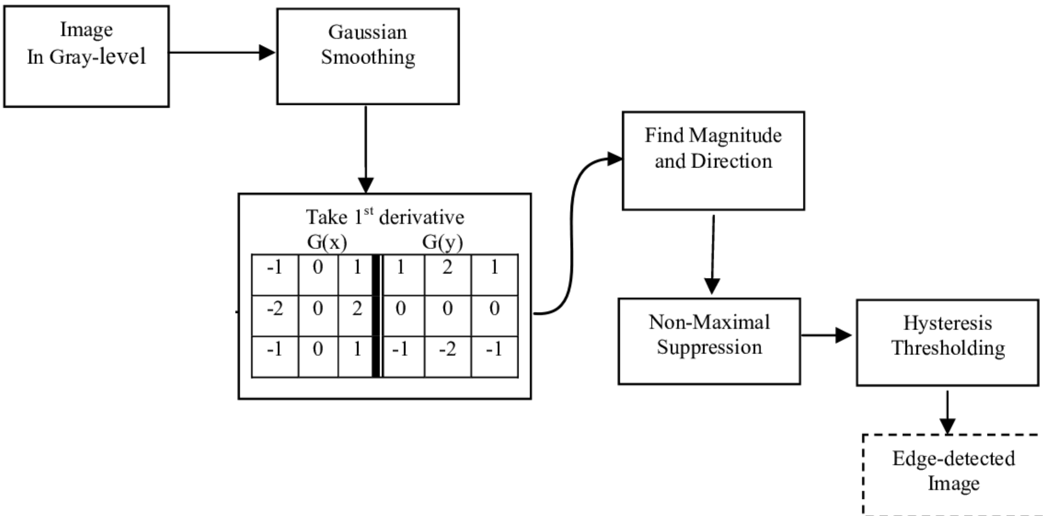


Fig. 4. Canny edge detection algorithm

Edge Detection with Canny Filter

```
use math,imager,plot

i1 = load.image('http://edu-9.de/uploads/assets/pores_microsl
m1 = as.matrix(i1,mode='uint8')
m1.stats = minMax(m1)
m1.stats$mean = mean(m1)
m1 = (m1 - m1.stats$min)
m1[m1>100] = 255
print(m1.stats)
plot(m1)
```

Ex. 2. R+ microslice image pre-processing: Normalization and Binarization (0/255)

Edge Detection with Canny Filter

```
use math,imager,plot

m1.edges=cannyEdges(m1,t1=50)
plot(m1.edges)
```

Ex. 3. Canny edge filter in R+

Point Clustering using DBSCAN

- Try to group points from Canny edges to define a ROI marking a pore candidate

[https://medium.com/@agarwalvibhor84/lets-cluster-data-points-using-dbscan-278c5459bee5]

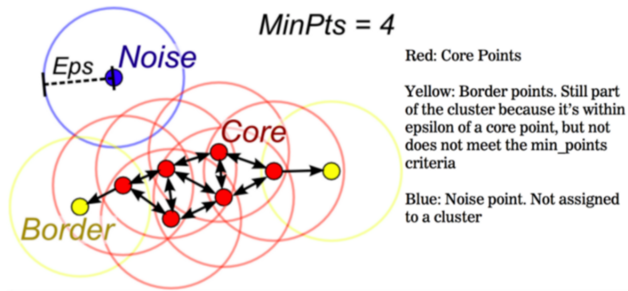


Fig. 5. Three point classes: Core, Border, Noise

Core point
: A selected point is a core point if it has at least minimum number of points (*MinPts*) including itself within its epsilon-neighborhood. In figure 1, red points are core points that have at least *MinPts*=4 in their neighborhood. If we've a core point, it means it is a dense region.

Border point
: A selected point that is within a neighborhood of a core point but it itself cannot be a core point. In the figure 1, yellow points are identified as border points. If we've a border point, it means the point is in a vicinity or at the border of dense region.

Noise point
: A selected point that is neither a core point nor a border point. It means these points are outliers that are not associated with any dense clusters. In the figure 1, blue point is identified as noise point.

DBSCAN Algorithm

Initially, the algorithm begins by selecting a point randomly uniformly from the set of data points. Checks if the selected point is a core point. Again, a point is a core point if it contains at least *MinPoints* number of minimum points in its epsilon-neighborhood.

Then, finds the connected components of all the core points, ignoring non-core points.

Assign each non-core point to the nearest cluster if the cluster is its epsilon-neighbor. Otherwise, assign it to noise.

The algorithm stops when it explores all the points one by one and classifies them as either core, border or noise point.

Alg. 1. DBSCAN

ROI Bounding Box Approximation from point list

- Output from DBSCAN: List of point groups from canny edge filter
- Calculate rectangular (not rotated) average bounding boxes (pore boundary point group)

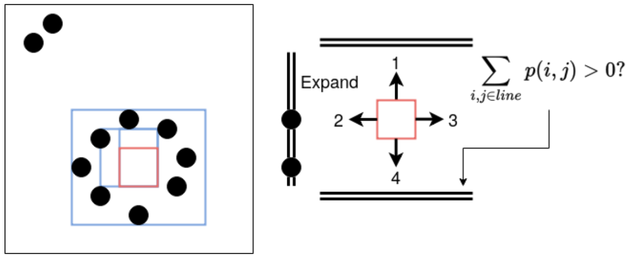


Fig. 6. An initial ROI (red) positioned at the center of mass of a point cluster is expanded iteratively by increasing one side and computing the point sum along this side. If the line is empty, the next side is expanded.

ROI Bounding Box Approximation from point list

- Bounding boxes can be computed from a pixel set list, i.e., a list of coordinate vectors

```
use math, imager, plot
pxs = {
  [1,2],
  [3,9],
  [4,7]
}
pxs.bbox = bbox(pxs)
print(pxs.bbox)
```

Ex. 4. Example of a bbox calculation from a pixel set list

Ellipse Fitting from point list

Problem: Calculate the parameters of an ellipse equation for a set of boundary points.

General Ellipse Equation

[Halíř, FLusser, Numerically stable direct least squares fitting of an ellipse]

$$F(x,y) = ax^2 + bxy + cy^2 + dx + ey + f = 0$$
$$b^2 - 4ac < 0$$

with a,b,c,d,e, and f coefficients.

$$\vec{a} = [a,b,c,d,e,f]^T$$
$$\vec{x} = \left[x^2, xy, y^2, x, y, 1\right]$$

Minimization Problem

The ellipse-specific fitting problem for a set of points ***p*** can be reformulated as:

$$\min_{a} \left|\left|\hat{D}\vec{a}\right|\right|^2, \vec{a}^T \hat{C}\vec{a} = 1$$

with ***D*** as the design matrix containing expanded ellipse equation terms, one row for each point:

$$\hat{D} = \begin{pmatrix} x_1^2 & x_1 y_1 & y_1^2 & x_1 & y_1 & 1 \\ .. & .. & .. & .. & .. & .. \\ x_n^2 & x_n y_n & y_n^2 & x_n & y_n & 1 \end{pmatrix}$$

and ***C*** is a 6 × 6 constraint matrix (independent from the number of points).

Solving an Eigenvalue Problem

- The minimization problem is ready to be solved by a quadratically constrained least squares minimization.

- We get a solution of the minimization problem by solving the Eigenvalue Problem, getting the Eigenvectors, and applying some filtering (only positive Eigenvalue are selected)

$$\min_a \left|\left|\hat{D}\vec{a}\right|\right|^2, \vec{a}^T \hat{D}^T \vec{a} = \lambda \vec{a}^T \hat{C} \vec{a} = \lambda$$

```
%% Pseudo Code!
function fit_ellipse(x, y) {
 D = [x.*x x.*y y.*y x y ones(size(x))]; % design matrix
 S = D' * D; % scatter matrix
 C(6, 6) = 0; C(1, 3) = 2; C(2, 2) = -1; C(3, 1) = 2; % constraint matrix
 [gevec, geval] = eig(inv(S) * C); % solve eigensystem
 [PosR, PosC] = find(geval > 0 &  ̃isinf(geval)); % find positive eigenvalue
 a = gevec(:, PosC); % corresponding eigenvector
 a
}
```

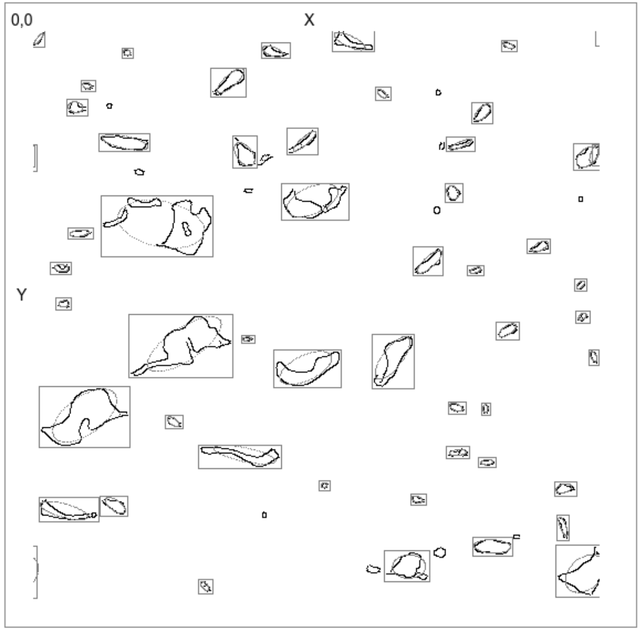


Fig. 7. Examples of results from the Canny-DBSCAN-ROI-ElliFit workflow for a ADM micrograph image

Ellipse Features

Solving the general ellipse equation delivers six polynomial parameters. But relevant for pore analysis are the following parameters:

1. Major and minor axis lengths w and h
2. Orientation angle of the major axis θ
3. Area A of the ellipse
4. Center coordinates

Ellipse Features

These parameters can be derived from the general equation parameters:

$$c_x = \frac{2cd - be}{b^2 - 4ac}$$

$$c_y = \frac{2ae - db}{b^2 - 4ac}$$

$$w = \sqrt{\frac{-4fac + cd^2 + ae^2}{4ac^2}}$$

$$h = \sqrt{\frac{-4fac + cd^2 + ae^2}{4a^2c}}$$

$$\theta = \tan^{-1}\left(2\frac{b}{a-c}\right)\frac{360}{4\pi}$$

Statistical Analysis

1. Average Density (from binarized image)
2. Average pore size (from ellipse fitting), aspect ratio w/h, variance of these features
3. Average pore orientation
4. Spatial distribution of pores

Data-driven Modelling and Machine Learning

[Khan et al., 2018]

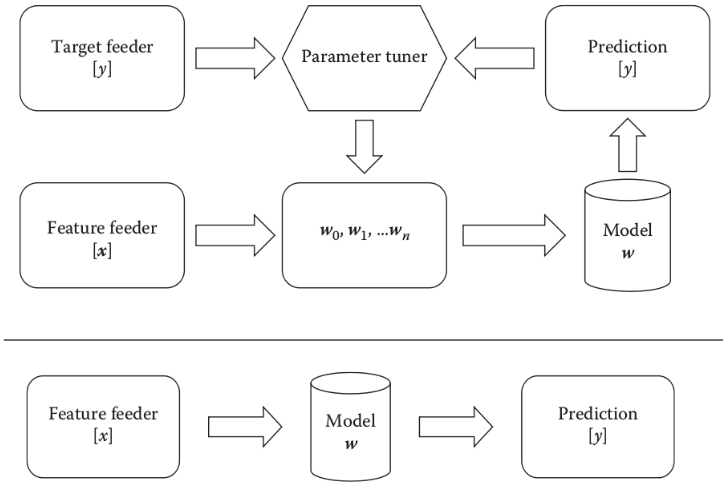


Fig. 8. (Top) Training of a data-driven machine model (Bottom) Application and inference

Convolutional Neural Networks

Convolutional Neural Networks combine typically:

1. Multi-dimensional matrix convolution with arbitrarily sized filter kernels
   - Mapping of matrix data on matrix data (commonly dimensionality expansion)
2. Fully connected neural node layers
   - Mapping of vectors on scalar valies (dimensionality reduction)
3. Pooling layers
   - Data reduction (fusion)

CNN Architecture

[Ragav Venkatesan and Baoxin Li,2018]

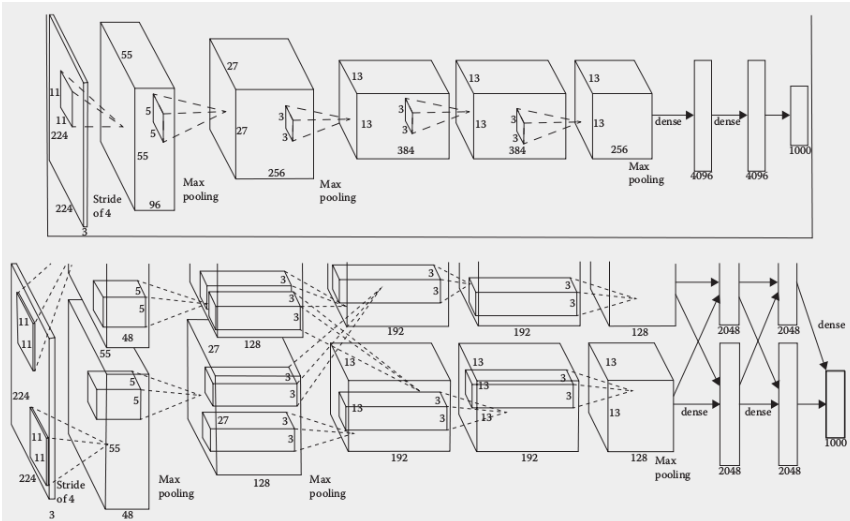


Fig. 9. Examples of CNN architecture consisting of interlacing different class layers

Convolutional Neural Networks

[Khan et al., 2018]

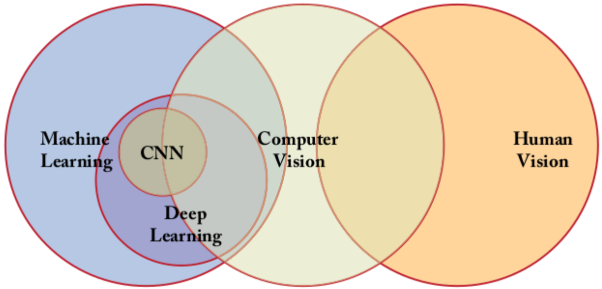


Fig. 10. The relation between human vision, computer vision, machine learning, deep learning, and CNNs.

Vector, Matrix, and Tensor Data

Vector

A vector is commonly a linear list of values (real or complex type):

$$\vec{x} = [v_1, v_2, .., v_n]$$

$$\vec{a} \odot \vec{b} = [c_1, c_2, c_i, .., c_n], c_i = a_i \cdot b_i, n = |a| = |b|$$

$$\vec{x} \cdot \vec{w} = \sum_{i=1}^{n} x_i w_i, n = |x| = |w|$$

Vector, Matrix, and Tensor Data

Matrix

$$\hat{m} = \begin{bmatrix} v_{1,1} & v_{1,2} & .. & v_{1,i} \\ .. & .. & .. & .. \\ v_{j,1} & v_{j,2} & .. & v_{j,i} \end{bmatrix}$$

$$\hat{a} \otimes \hat{b} = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{i,j} b_{j,i}, n = \mathrm{rows}(a), m = \mathrm{cols}(b)$$

Vector, Matrix, and Tensor Data

Tensor

A scalar is a level zero tensor.

A vector is an array of numbers along an axis (level one tensor).

A matrix is an arrangement of numbers along two axes (level two tensor).

A tensor is an arrangement of numbers along n axes.

Vector, Matrix, and Tensor Data

Volumes

- Volumes are (here) three-dimensional data structures representing vectors, 2D matrix, and 3D tensor objects.
- **A volume is a packed linear array of values with a 321 Layout**:
  - First order (most significant) index dimension is depth (sz)
  - Second order index dimension is width (sx)
  - Third order(least significant) index height (sy)
- Input, intermediate, output and kernel data can be represented by volumes

Vector, Matrix, and Tensor Data

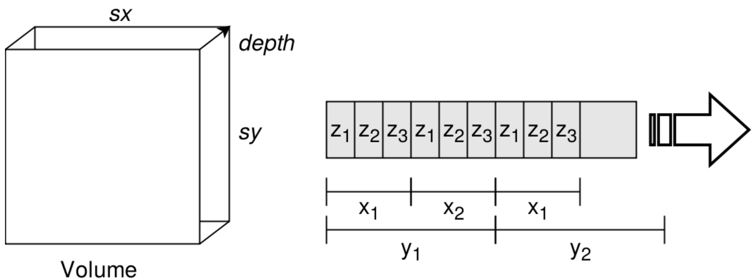


Fig. 11. Packed linear data model (memory layout) of 3D volumes

Vector, Matrix, and Tensor Data

Operations

- Addition (elementwise)
- Multiplication (elementwise)
- Multiplication and Addition (Dot Product)
- Convolution (Mapping)
- Transformation (including Fourier)

Convolutional Layer

- In contrast to kernel-based filtering operations using commonly 3 × 3 two-dimensional filters, convolution can be performed here with any kernel size and dimension.
- In contrast to kernel-based filtering operations, the kernel parameters (weights) are not pre-determined. They are evolved during the ML training process.

Convolutional Layer

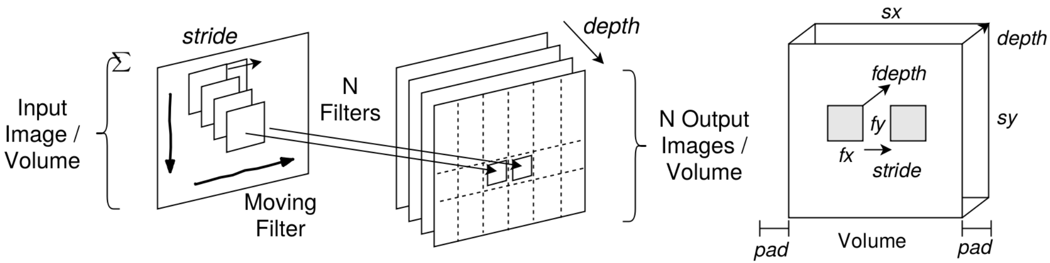


Fig. 12. Convolution with *N* filters applied to one input image (stride: shift of filter position in each dimension)

Example: Handwritten Digit Recognition

[Ragav Venkatesan and Baoxin Li,2018]

LAYER NUMBER	INPUT SHAPE	RECEPTIVE FIELD	NUMBER OF FEATURE MAPS	TYPE OF NEURON
1	$28 \times 28 \times 1$	5×5	20	Convolutional
2	$24 \times 24 \times 20$	2×2	–	Pooling
3	$12 \times 12 \times 20$	5×5	50	Convolutional
4	$8 \times 8 \times 50$	2×2	–	Pooling
5	800	–	500	Fully connected
6	500	–	10	Softmax

Fig. 13. CNN layers and configuration for handwritten digit recognition (using the MNIST data set consisting of $28 \times 28 \times 1$ images)