Hochschule Hof
University of Applied Sciences

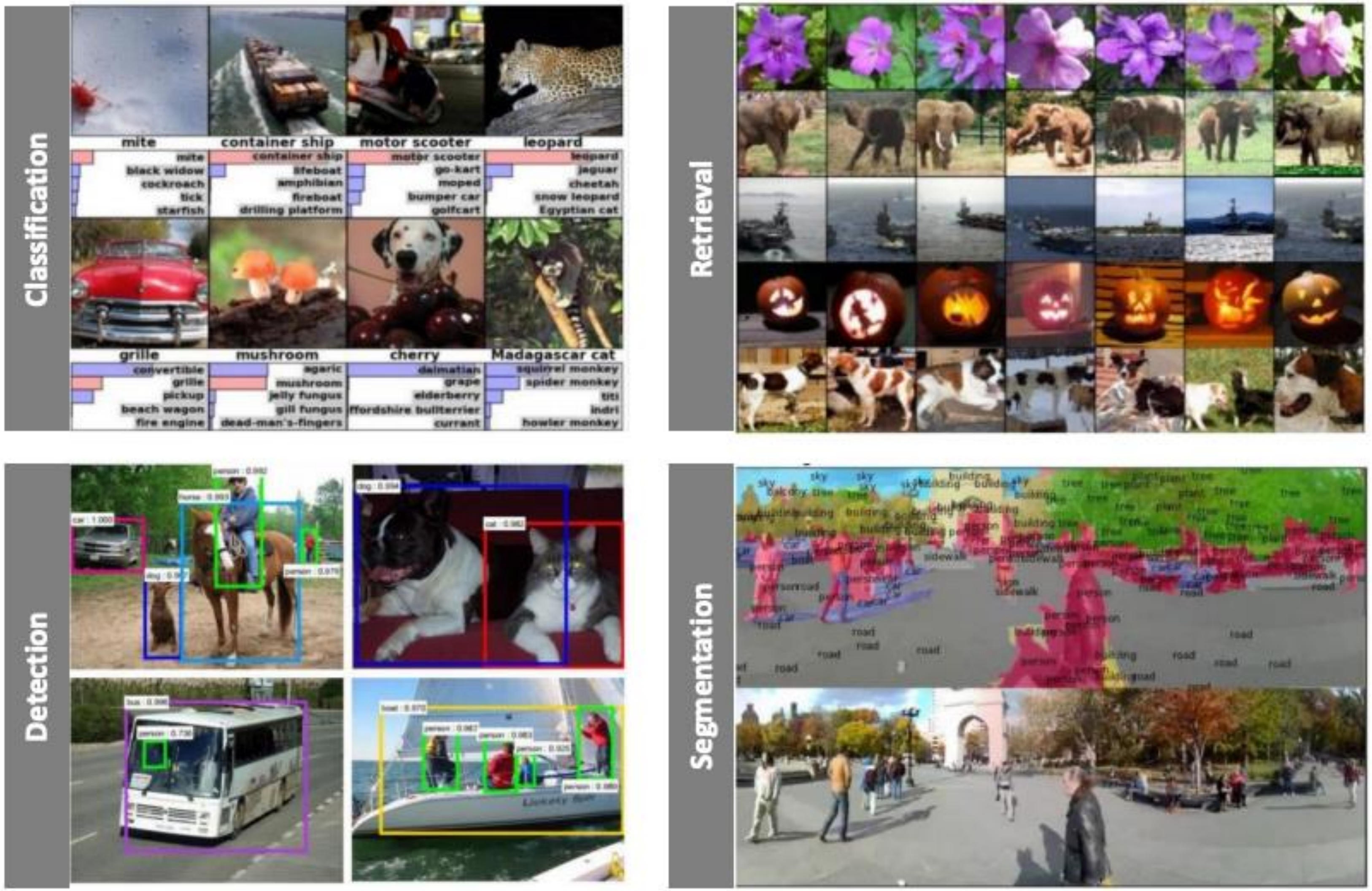
# Convolutional Neural Nets

## Applied Deep Learning

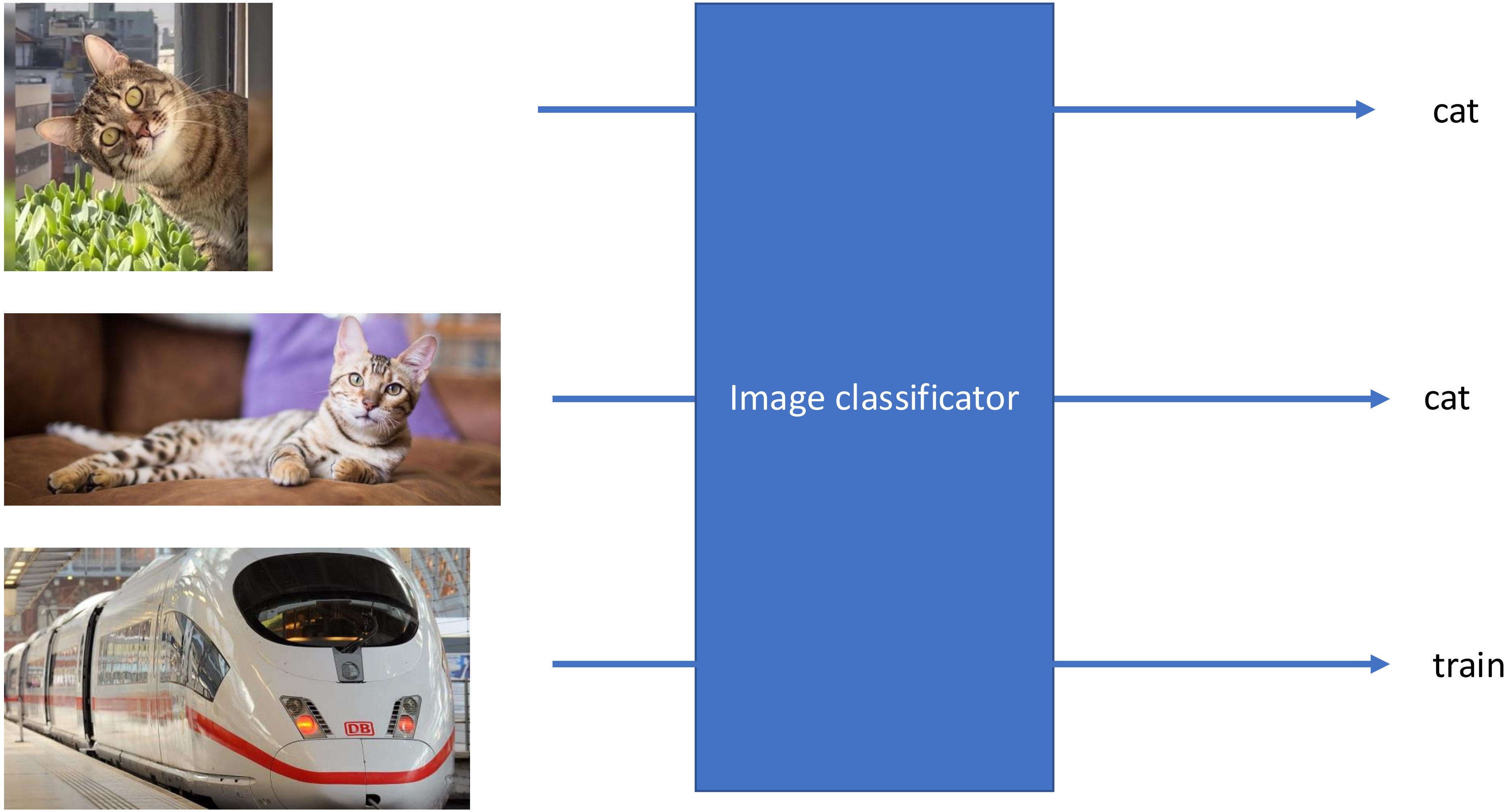
# Targets

- Know different image-based problems
- Understand the basic operation of CNNs
- Understand the components and concepts behind a CNN

# Motivation


Classification
mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
mushroom
cherry
Madagascar cat
convertible
grille
pickup
beach wagon
fire engine
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
dalmatian
grape
elderberry
ffordshire bullterrier
currant
squirrel monkey
spider monkey
titi
indri
howler monkey
Retrieval
Detection
Segmentation

# Motivation - classificator



# Image classification - Traditional

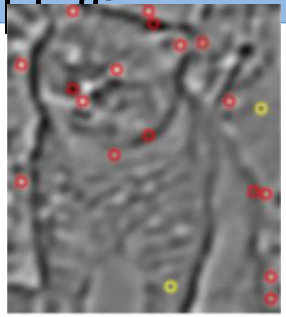
Hochschule Hof
University of Applied Sciences

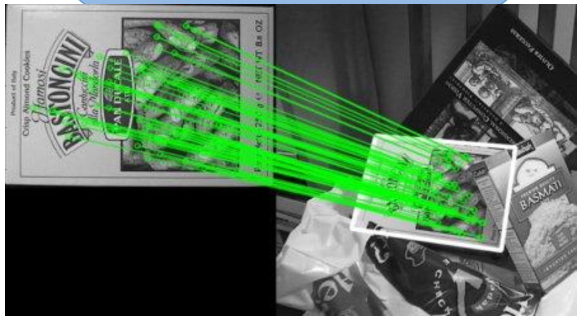
**Image acquisition**

**Preprocessing**
- Scaling
- Rectification
- Lighting
- Filter

**Feature Extraction**
- Sift/Surf
- Color
- Weight

**Classification**
- SVM
- kNN
- NN



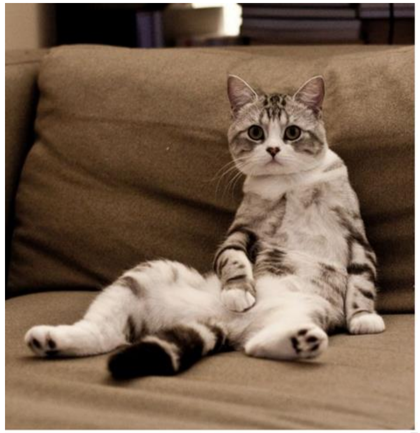


[2]

- + Image information is reduced to (relevant) features
- + Classifier can be trained with little data

# Image classification

[1]



 [3]

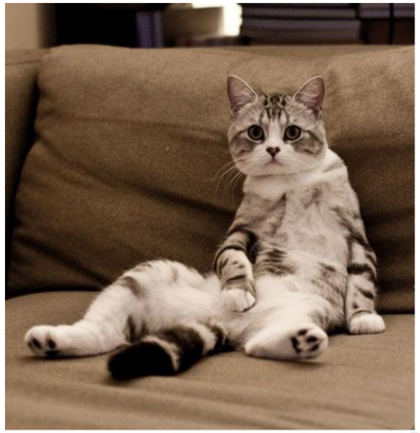
 [3]

 [3]

- Problems:
  - possible loss of information
  - Feature engineering complex
- Idea:
  - Use neural networks
  - Use every pixel of an image as a potential feature
  - Use hidden layers to learn more complex mappings

# Image classification

[1]





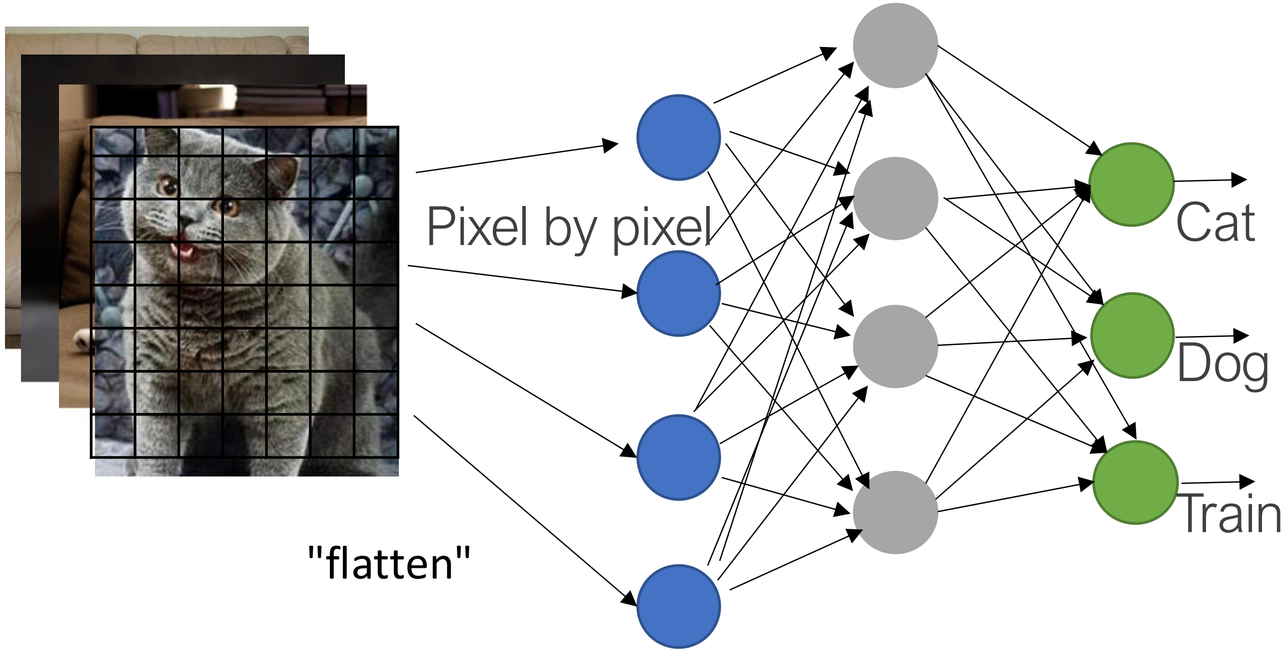
[3]



[3]



[3]



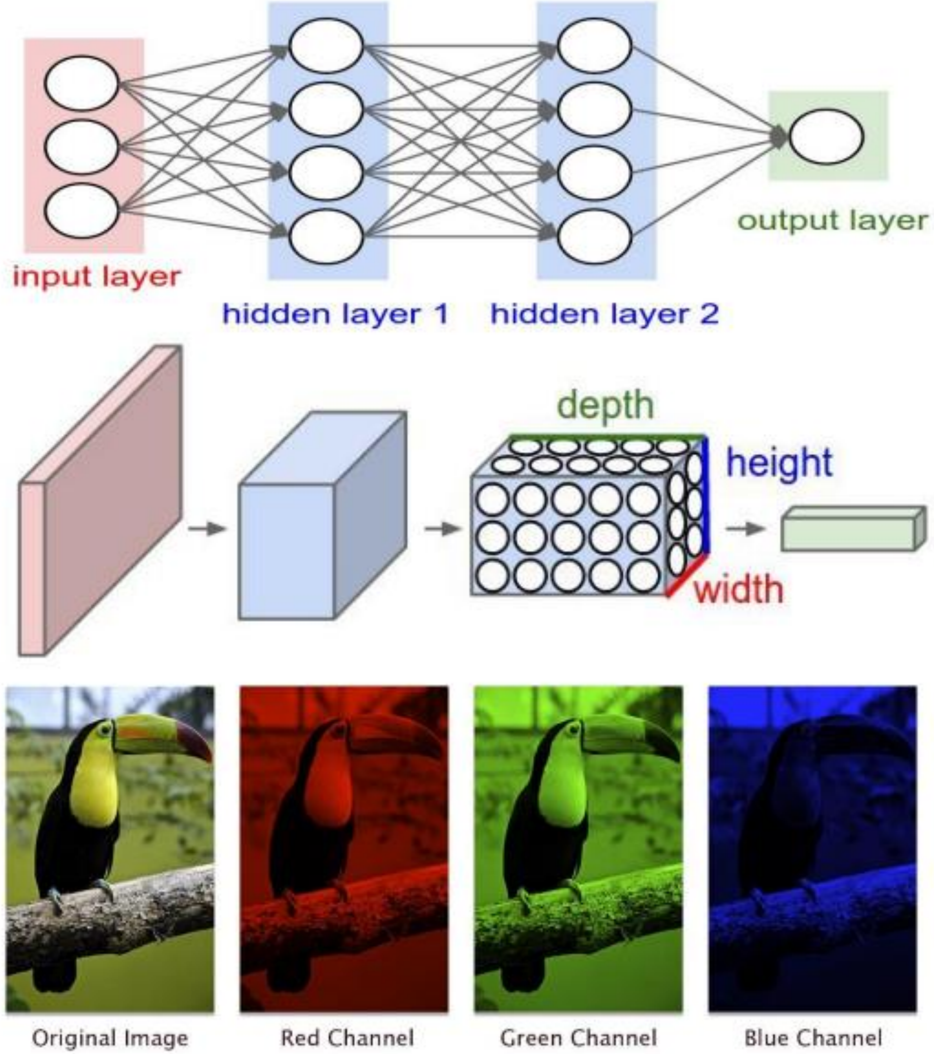

MNIST Example:

- 10 classes
- Images are 32x32 pixels
- → ca 10x32x32 = 30720 parameters
- → Very much training data necessary
- → Spatial context resolved

# CNNs - input



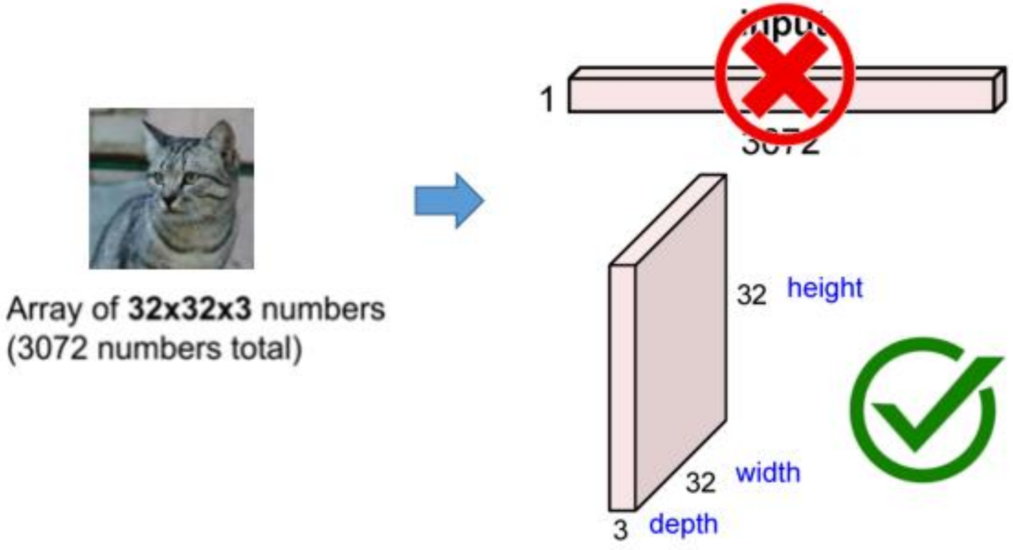
- Problem: Flattening destroys neighborhoods
- Idea:
  - directly use the images as input
  - Process the images as layers of a cube
  - Each layer corresponds to a (color) channel

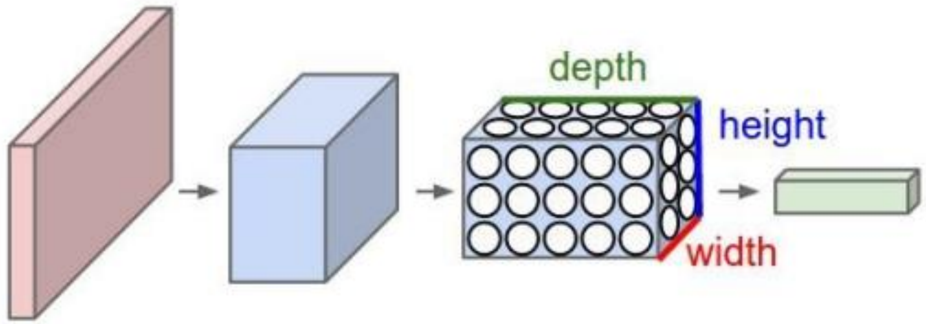


# CNNs - input

Hochschule Hof
University of Applied Sciences

- Problem: Flattening destroys neighborhoods
- Idea:
  - directly use the images as input
  - Process the images as layers of a cube
  - Each layer corresponds to a (color) channel

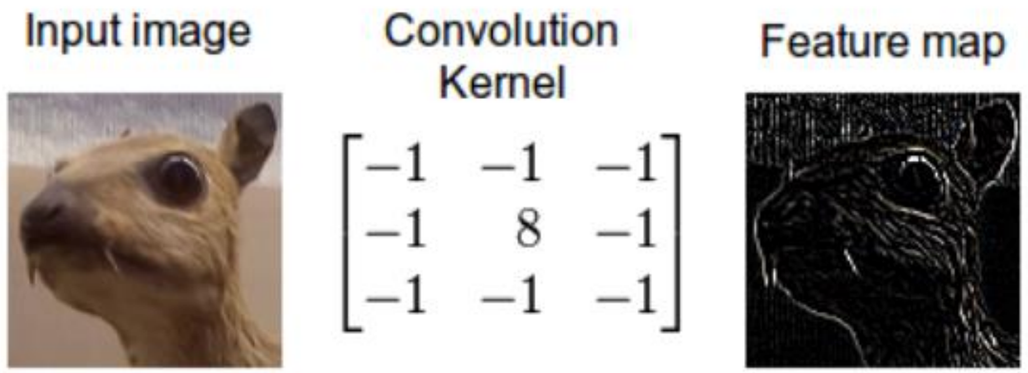


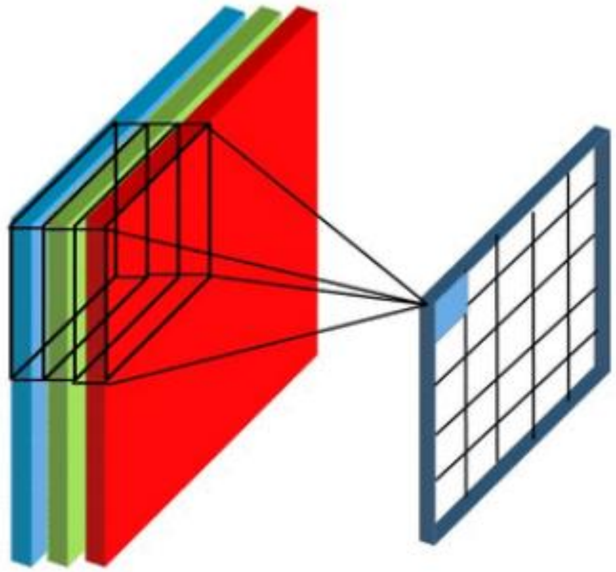



# CNNs - Filters / Convolutions

Hochschule Hof
University of Applied Sciences

- Calculation of potentially relevant features (corners, edges, gradients,...)
- For this: use a set of filters (convolution kernel)
- A filter is a matrix (usually n by n) that is applied to an input to obtain a new image
- Mathematically: a convolution i * k of a function i with a kernel function k



# CNNs - Filters / Convolutions

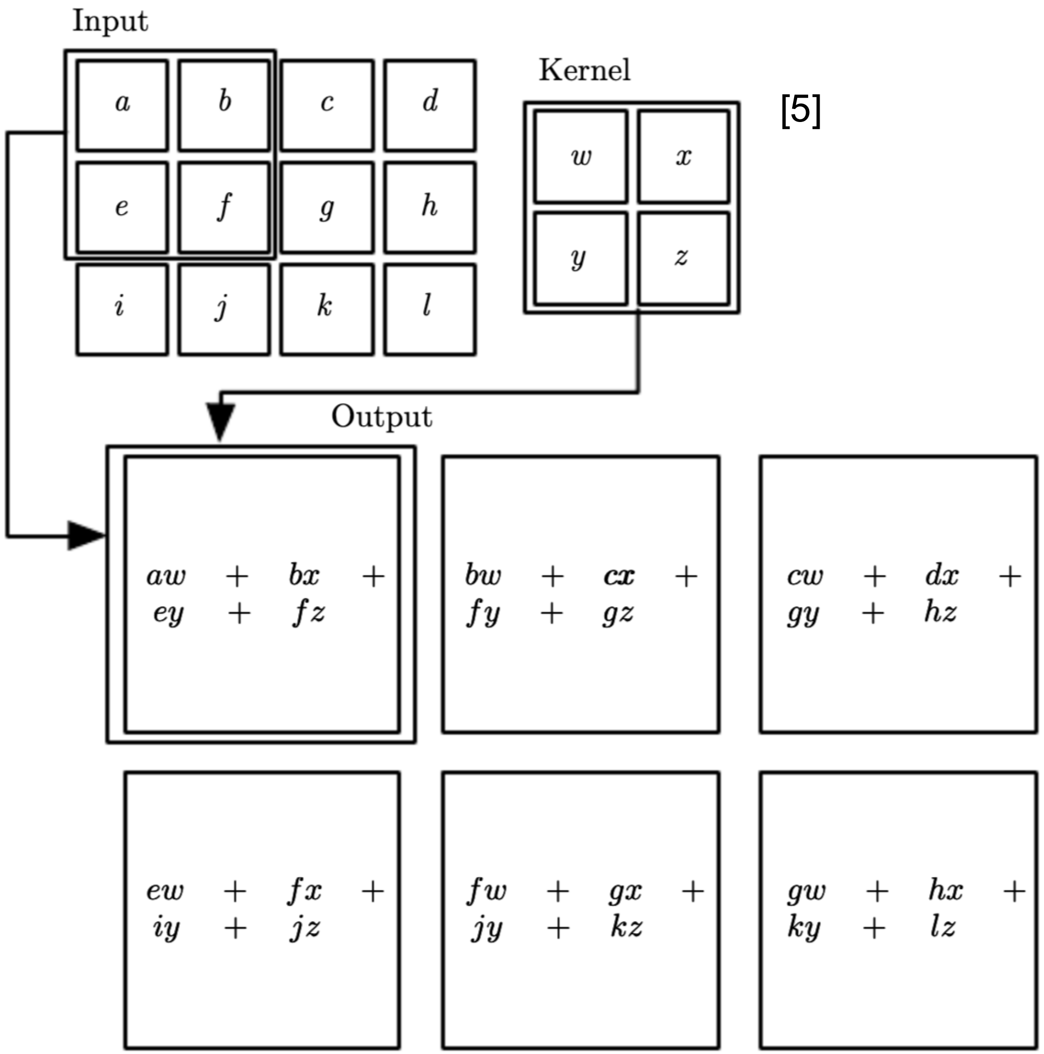
- Definition:

s(t)=

- Sliding Window:

$$s(t) = \sum_{-\infty}^{\infty} x(a)\, w(t-a)$$

- Two dimensions:

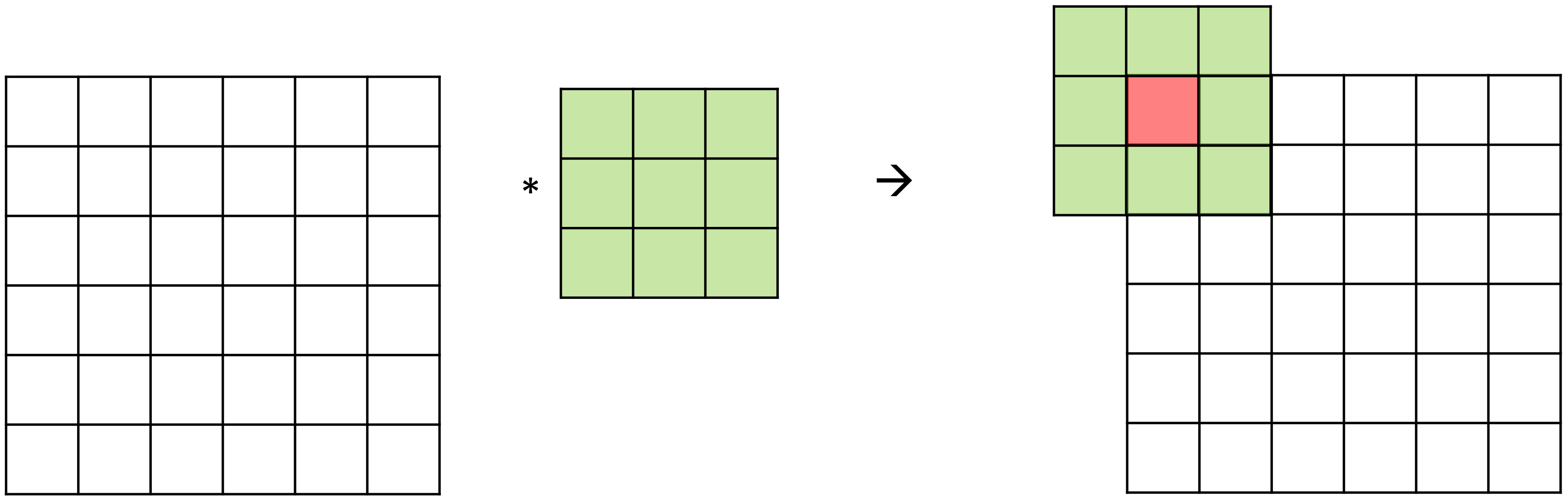



[5]

# Padding



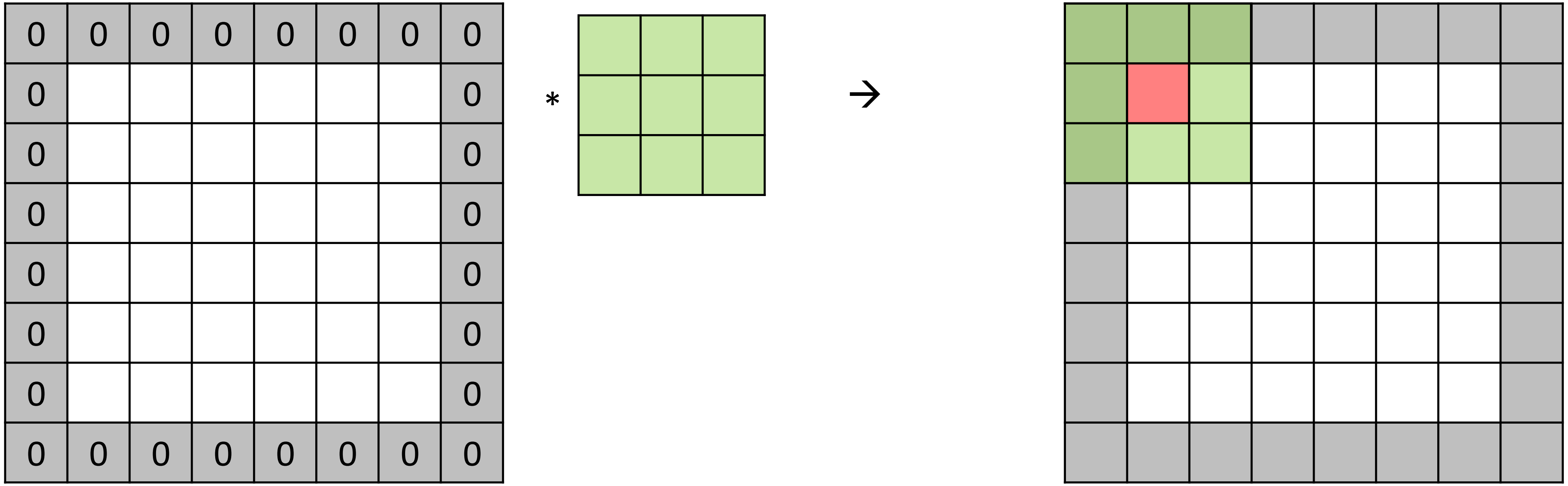
- What about the edge pixels?
- Without padding



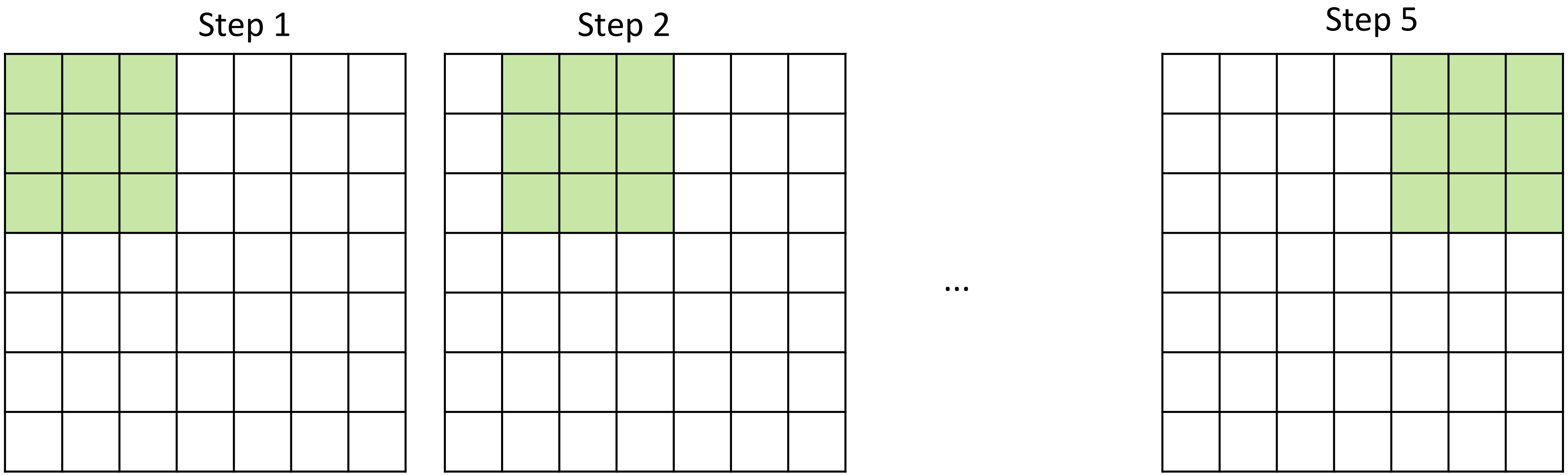
# Padding

Hochschule Hof
University of Applied Sciences

- What about the edge pixels?
- With zero padding (parameter corresponds to number of edge pixels)



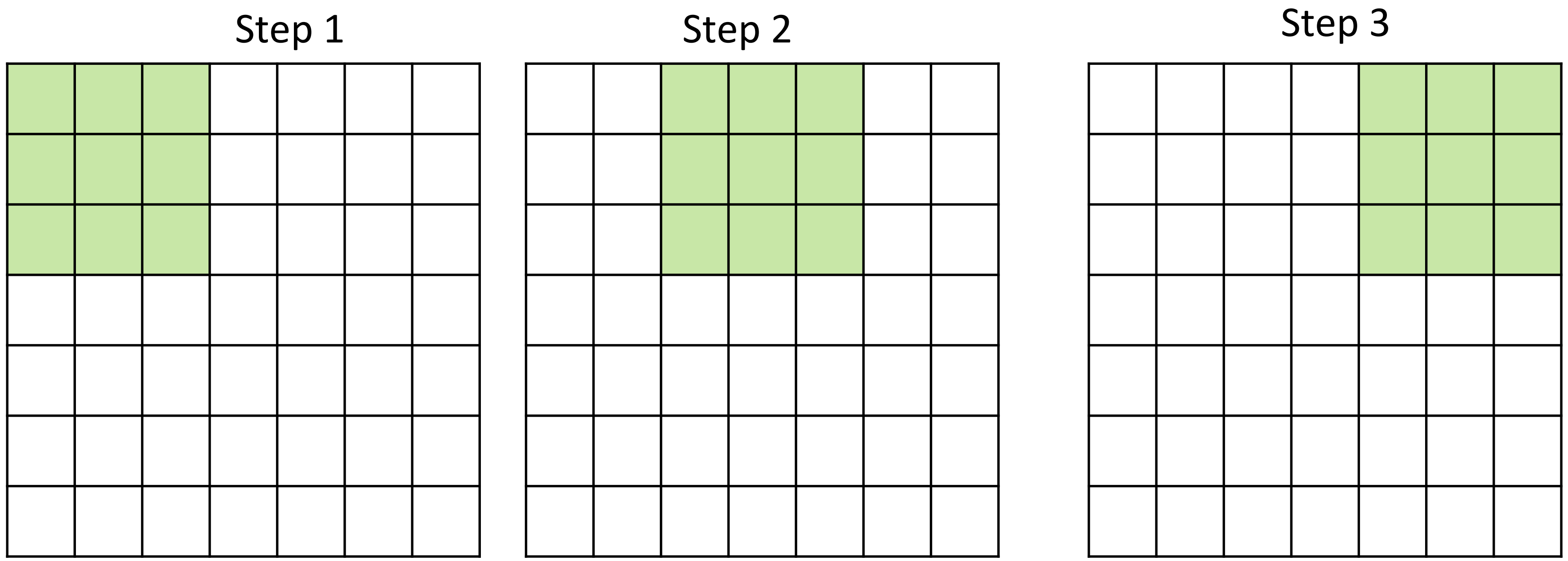
# Step size - Stride




7x7 input
3x3 filter
Stride 1

➔ Output 5x5

# Step size - Stride



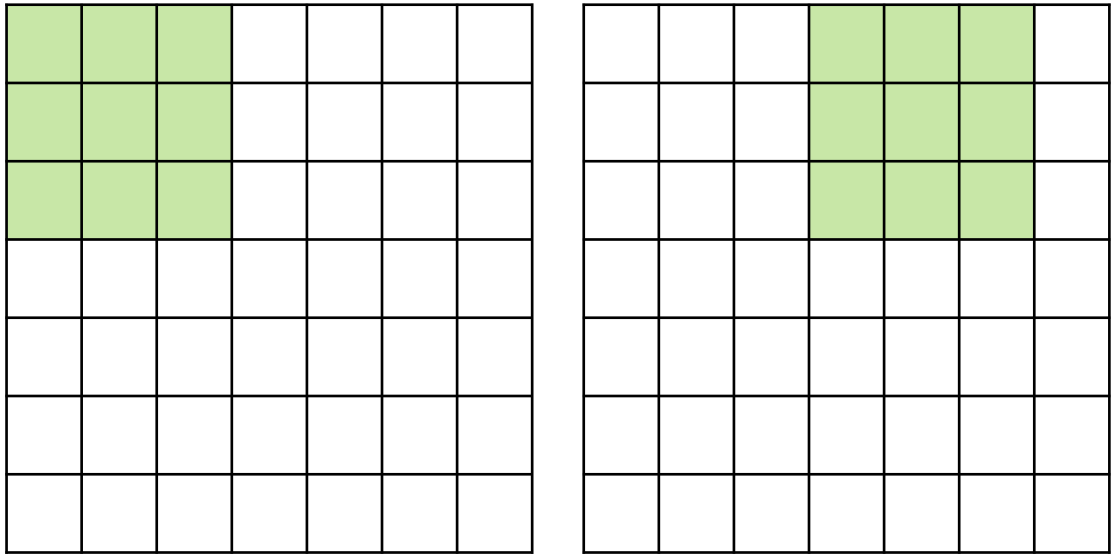

7x7 input
3x3 filter
Stride 2

➔ Output 3x3

# Step size - Stride



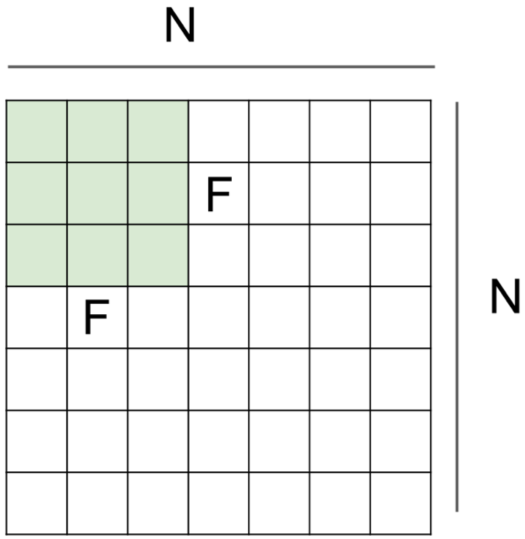
Step 1 Step 2



7x7 input
3x3 filter
Stride 3

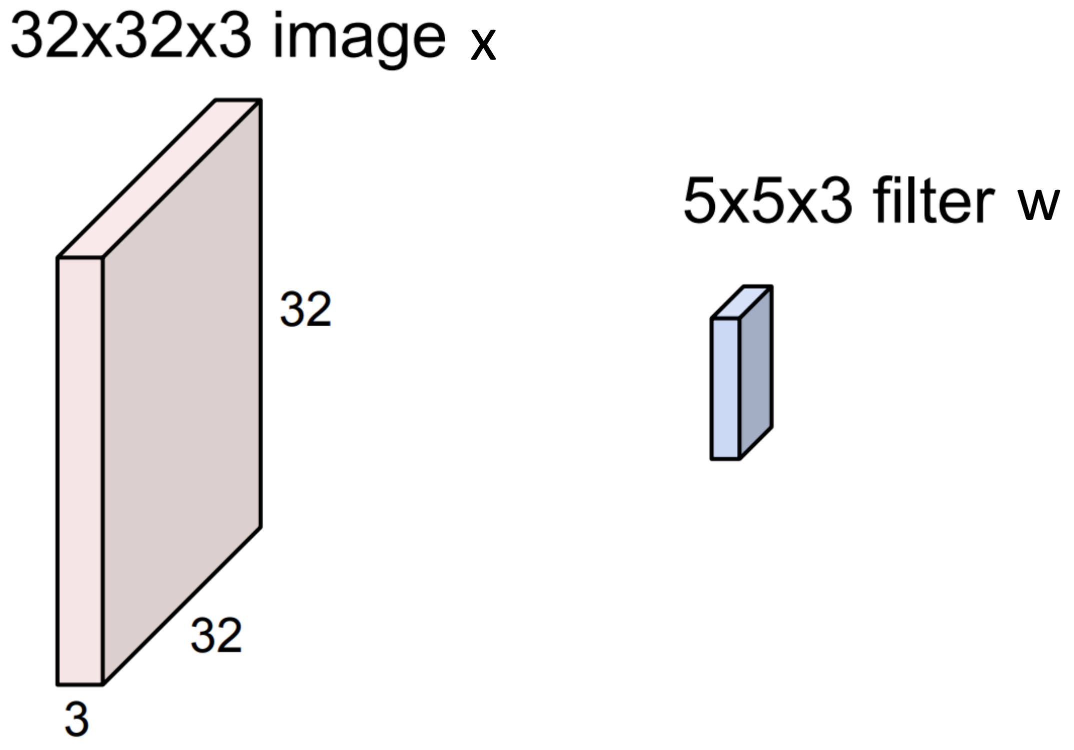
➔ Don't go!

Context:
**Output size: (N - F) / stride + 1**
Stride 1 => (7-3)/1 +1 = 5
Stride 2 => (7-3)/2 +1 = 3
Stride 3 => (7-3)/3 +1 = 2.33



# CNN's

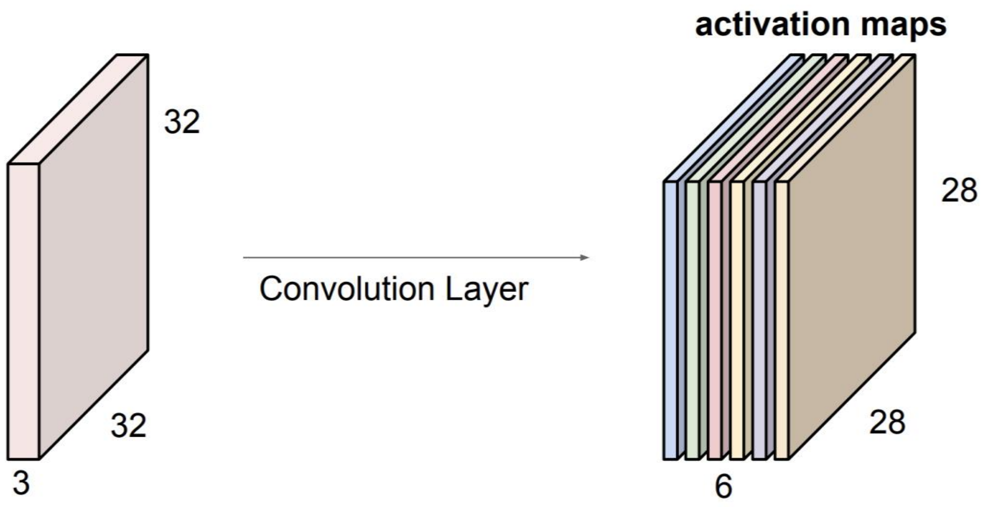
- Move the filter over the image
- Filter depth always corresponds to the image depth (here 3)
- Result 5*5*3 dimensional vector product $w^T x + b$



# CNNs - Layer output

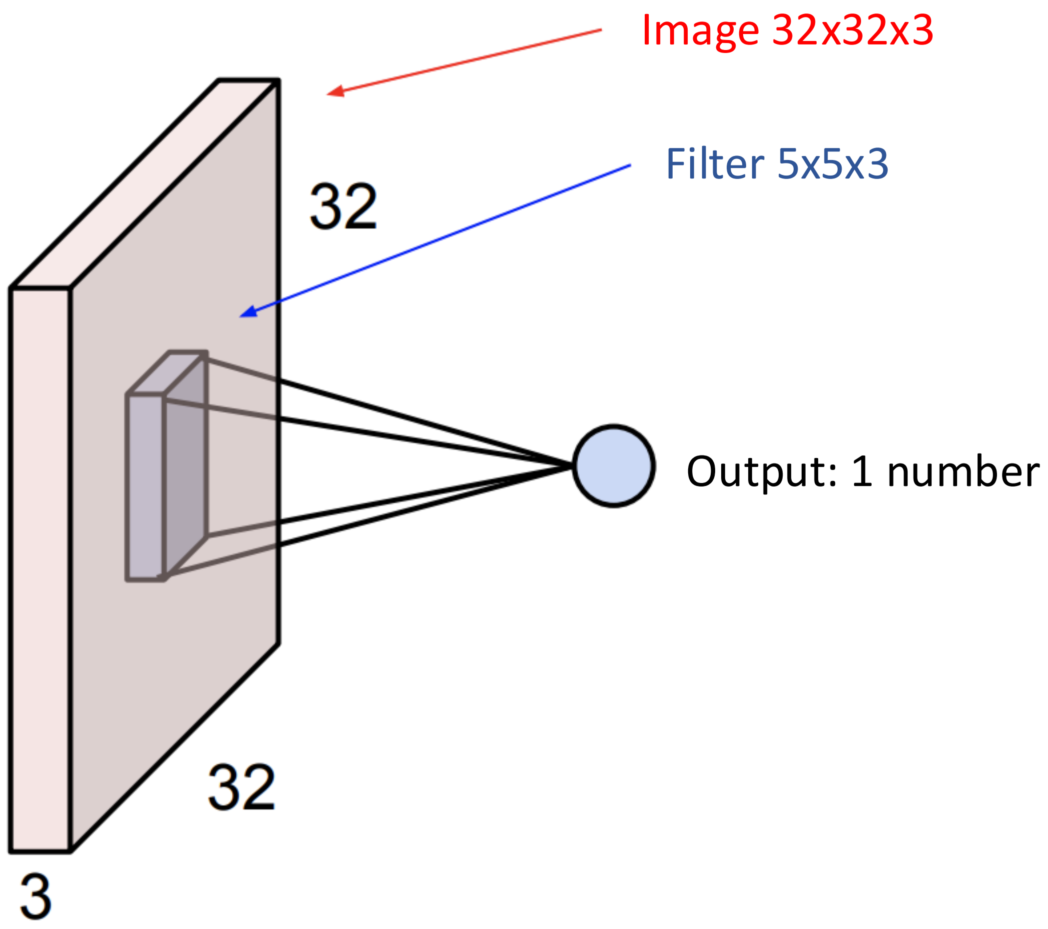
- Output of the convolution layer determined by
  - Number of filters
  - Padding
  - Stride

With 6 5x5 filters you get 6 separate outputs



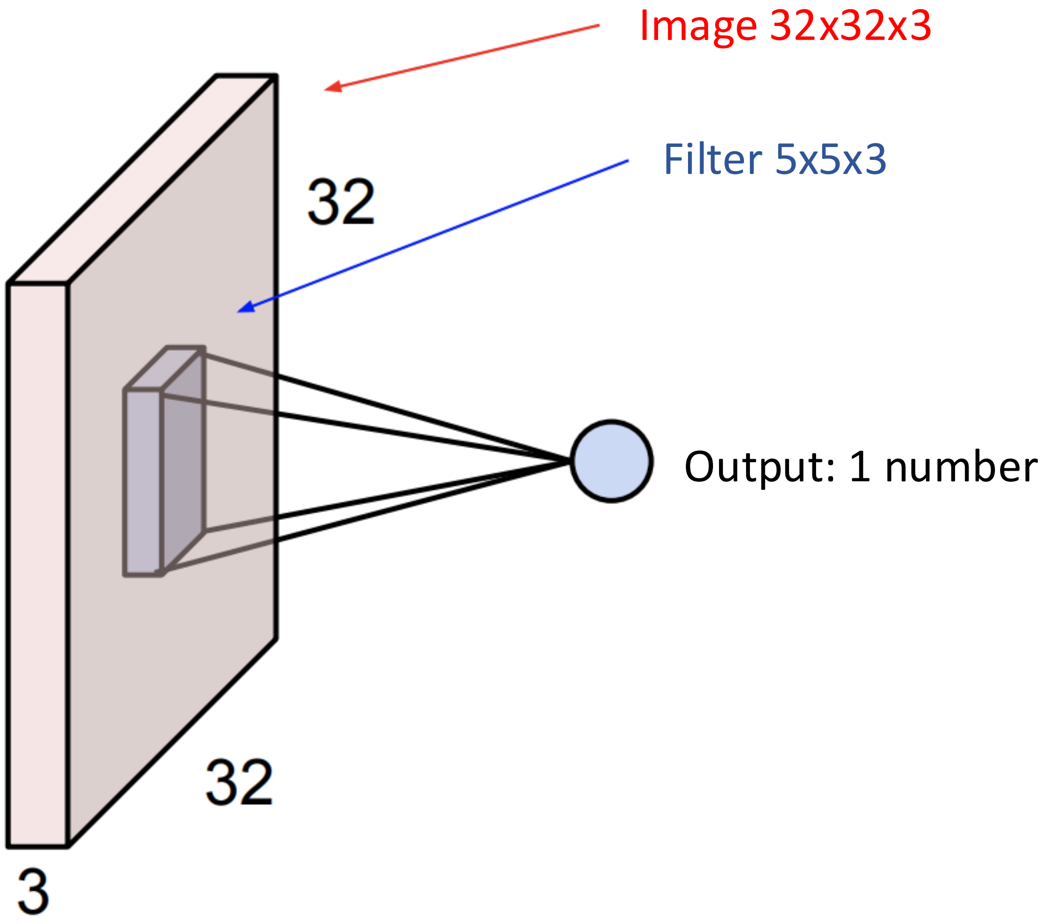

These are combined to form a new cube

# CNNs - folding




- Instead of a fully networked network or layer, we use a convolutional layer to learn important features
- One neuron corresponds to one iteration of a complete convolution

# CNNs - folding




- How many weights need to be learned?

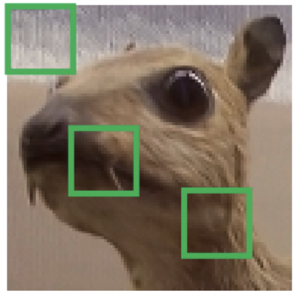
  F x F x depth + 1

- Example here: 5 x 5 x 3 + 1 = 76
- Comparison ANN: 3072 weights
- But why?

# CNNs - folding

- Assumption:
  - Our filter learns to recognize certain features (edge,...)
  - For the recognition it is irrelevant which image or at which position the feature is.

Input image



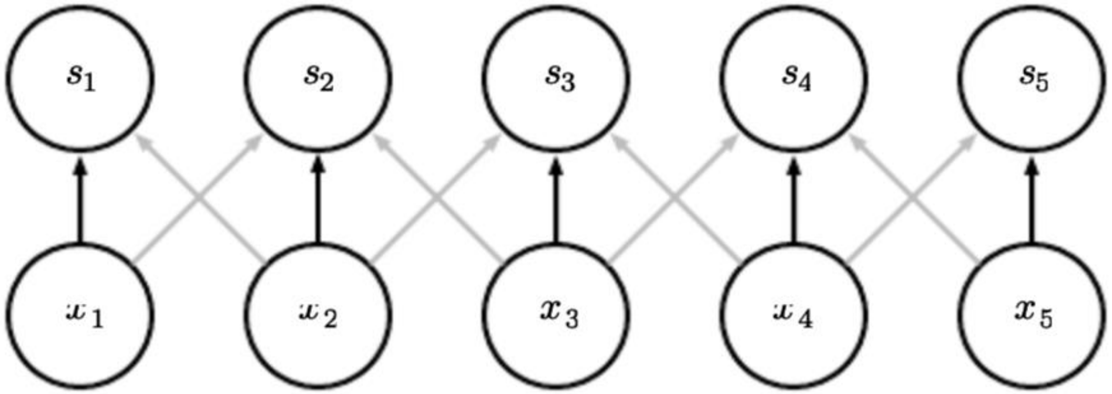
Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

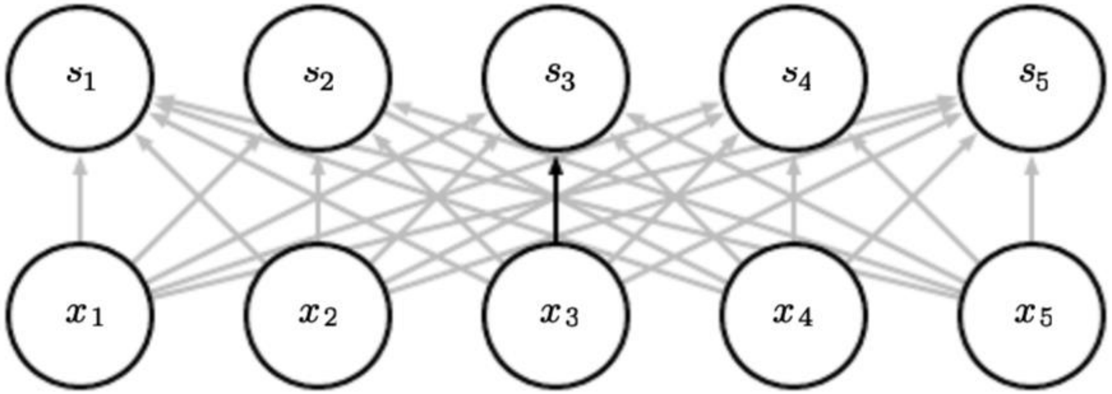
Feature map

- The weights of the filter always remain the same
- We therefore need to train the weights only once
- Other neurons with the same filter share the weights
- The number of parameters is significantly reduced

# Parameter Sharing



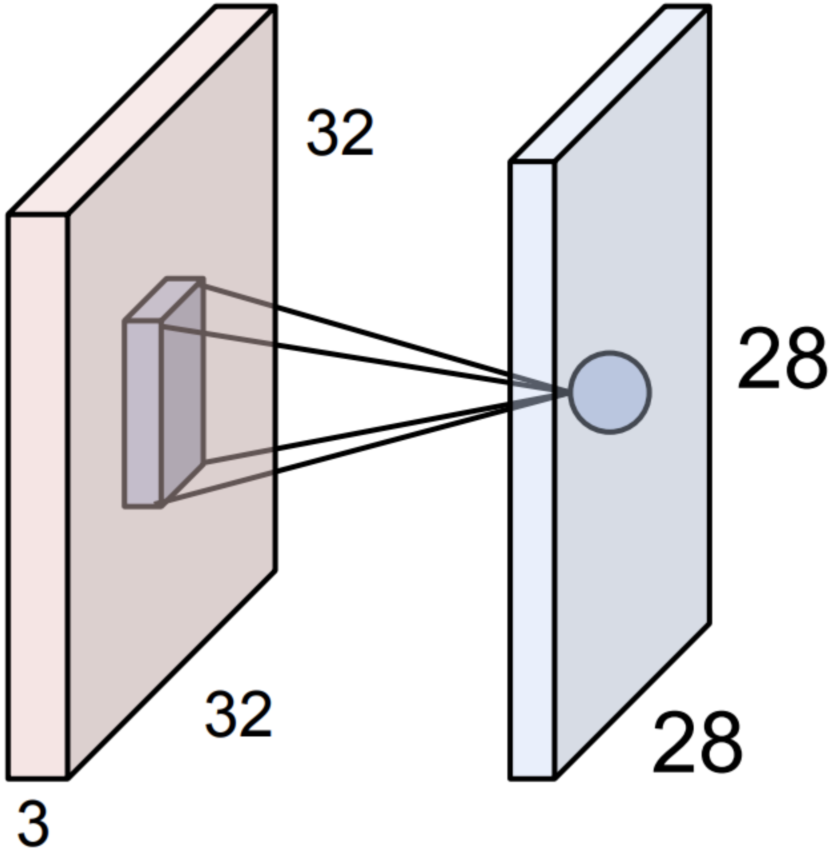

Parameter Sharing
Black weights Multiple used




Fully connected

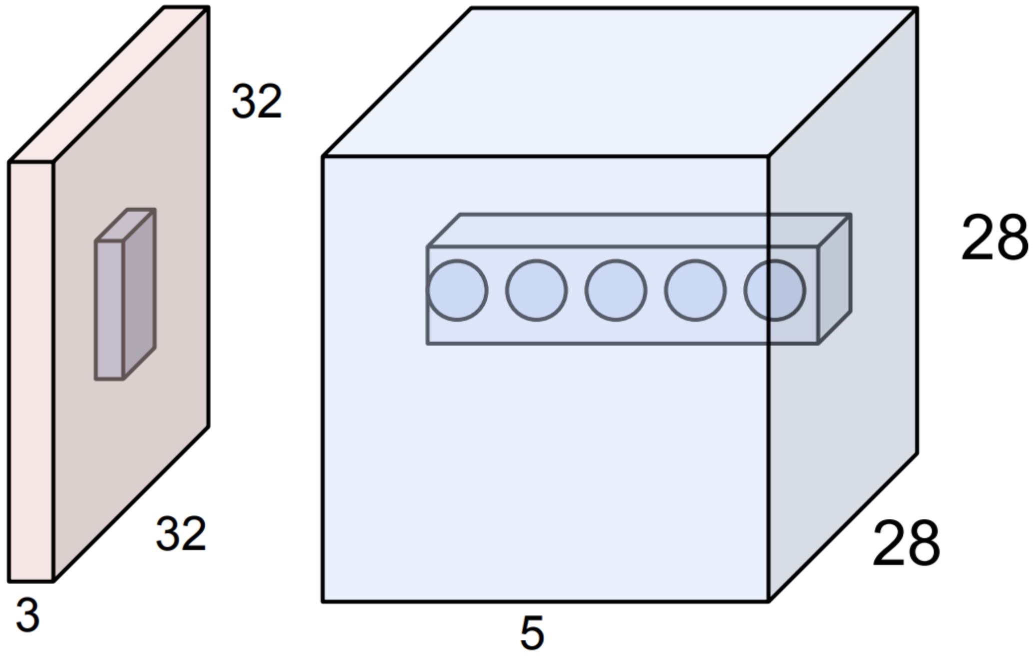
# CNNs - Structure






- The activation map here is a 28x28 area of neuron outputs
- All share parameters
- Each neuron is connected to a small region of the input
- 5x5 filter → 5x5 receptive field of a neuron

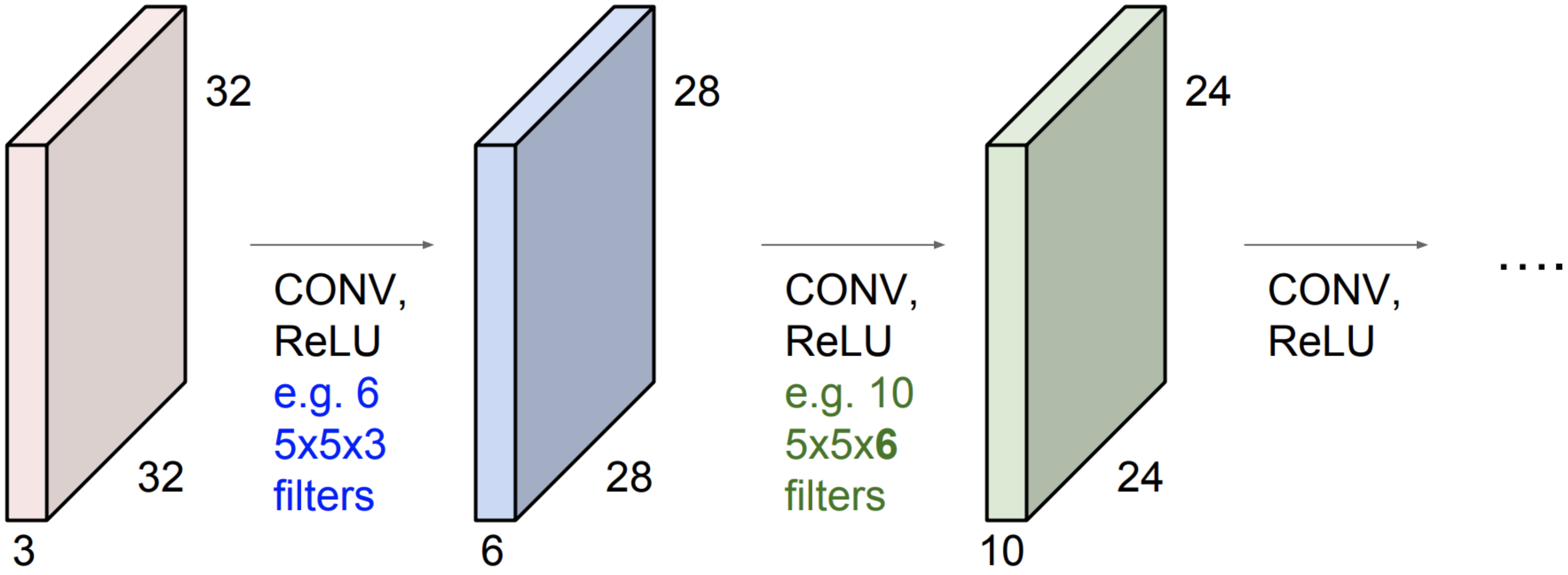
# CNNs - Structure




- With 5 filters it results in a 28 x 28 x 5 grid of neurons
- → 5 different neurons see the same input

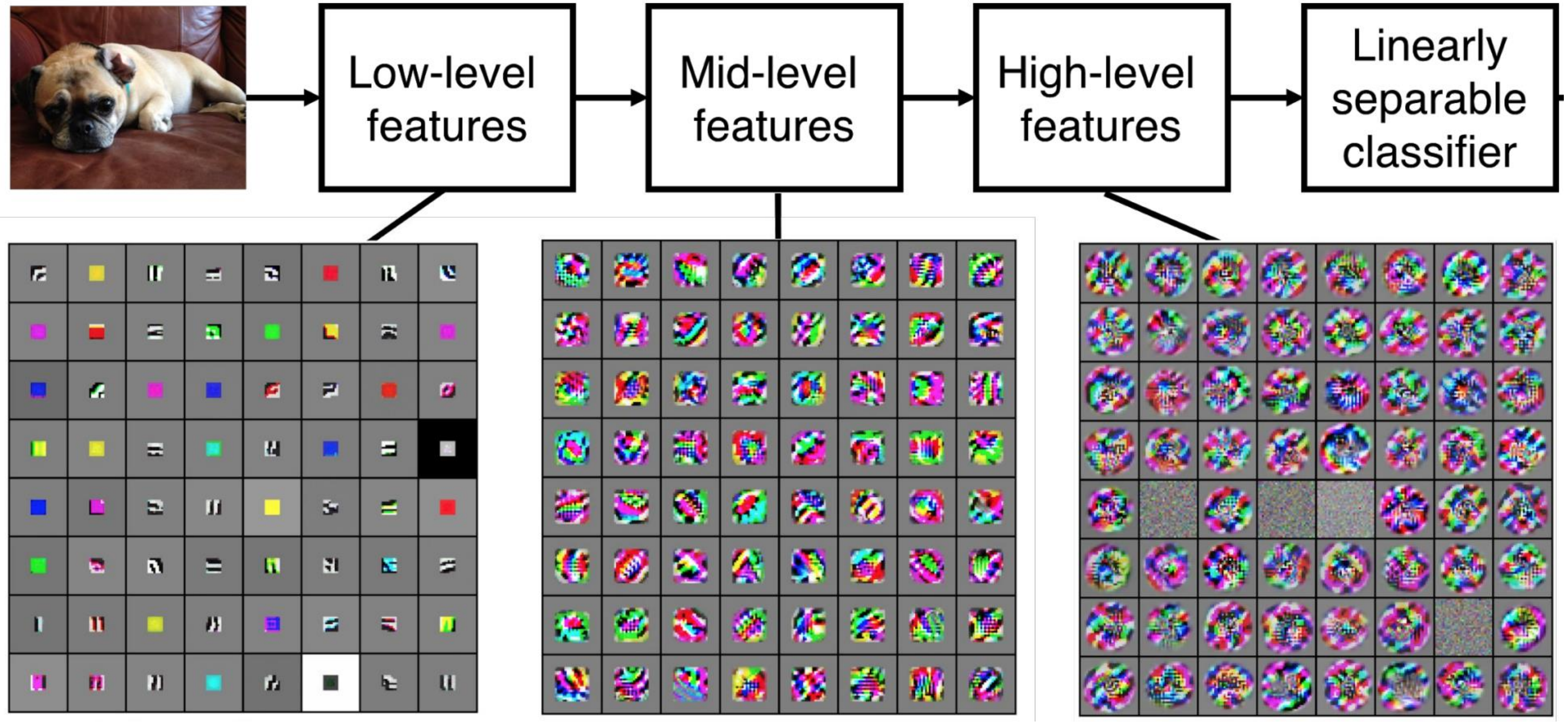
# CNNs - Structure






- Activation function mostly ReLu
- Repeated application of a 5x5 filter shrinks input
- Fast shrinkage works poorly

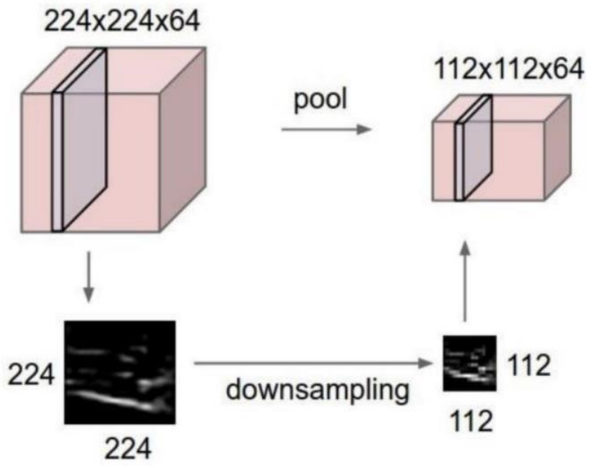
# CNNs - Structure




- To the right:
  - Number of filters increases
  - Complexity of features increases

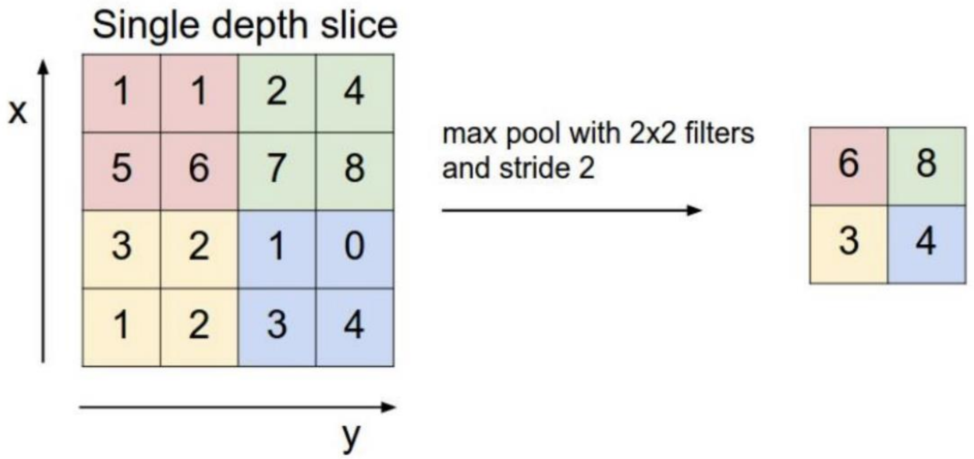
# CNNs - Pooling

- No clear regulation how many / which CONV - layer are needed
- Architectures see appendix
- Problem of overfitting but given
- Therefore pooling or dropout to manipulate CONV outputs.



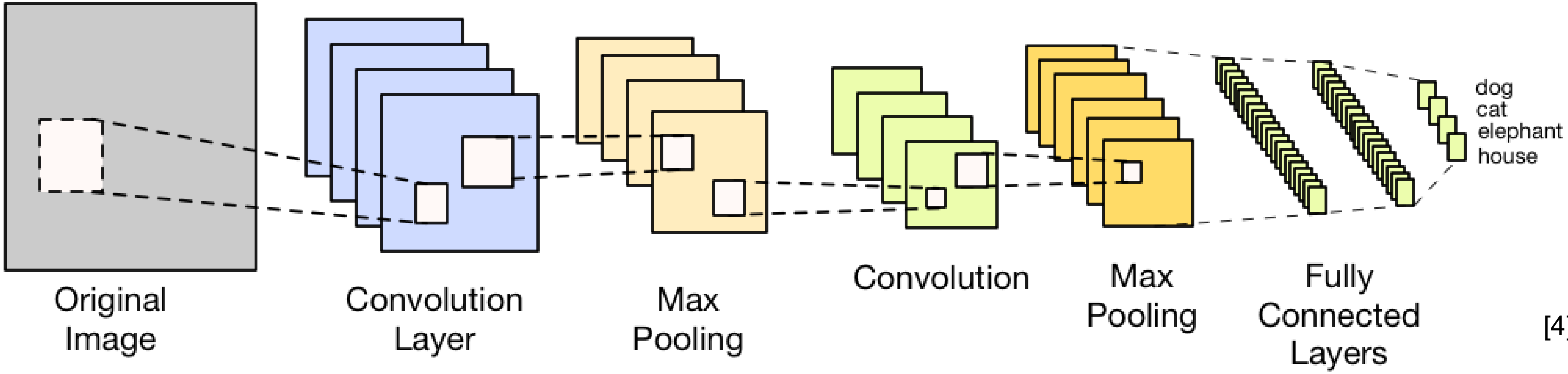

+ Reduction of parameters

+ Reduces overfitting



# CNN Architecture



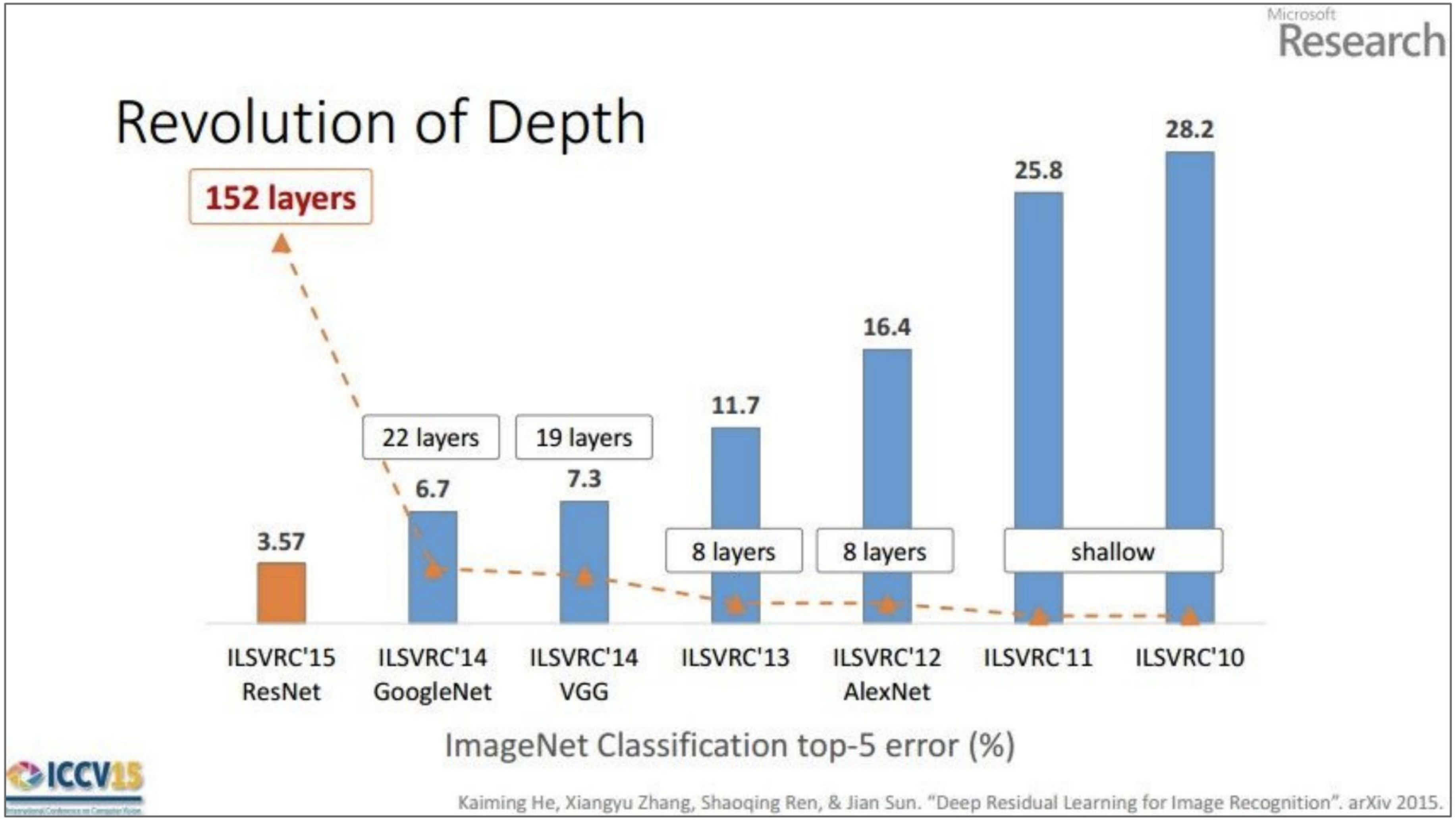



[4]

Input | Convolution stages | Neural net

# Development CNNs





# Implementation in Pytorch



in - channel

out - channel

```
nn.Conv2d(1, 16, 3, padding="same")
```

Kernel - size

Padding, ...

# Implementation in Pytorch

Hochschule Hof
University of Applied Sciences

```
nn.Conv2d(1, 16, 3, padding="same"),
nn.ReLU(),
nn.MaxPool2d(2),

nn.Conv2d(16, 32, 3, padding="same"),
nn.ReLU(),
nn.MaxPool2d(2),

nn.Conv2d(32, 64, 3, padding="same"),
nn.ReLU(),
nn.MaxPool2d(2),
nn.Flatten(),

nn.Linear(64*3*3, 128),
nn.ReLU(),
nn.Dropout(0.5),
nn.Linear(128, 10)
```

Number of classes

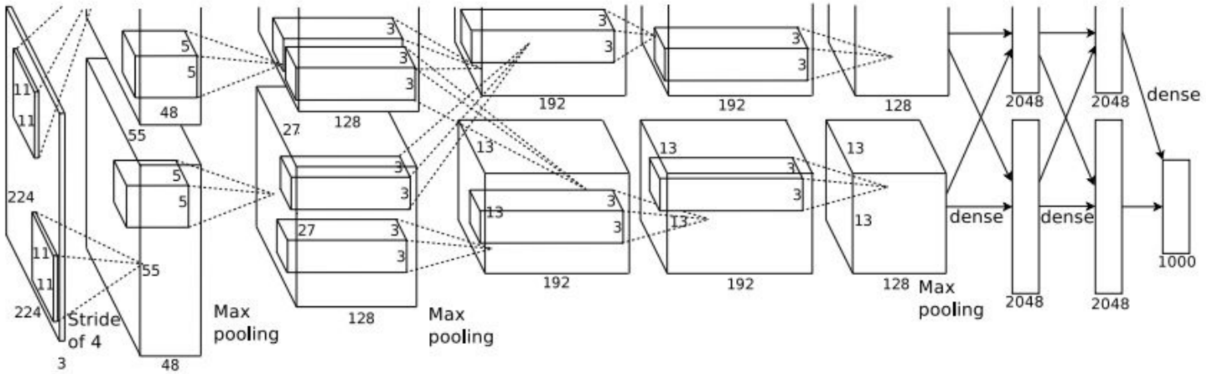
# Summary

- CNNs are special neural networks whose focus is on processing 2-dimensional data (images)
- Using the CNNs eliminates the need for manual feature engineering
- Building Blocks are CONV/RELU/POOL/FC
- Parameter sharing allows massive savings in the number of parameters

# Appendix

## Case Study: AlexNet

*[Krizhevsky et al. 2012]*




Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

**Details/Retrospectives:**
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%