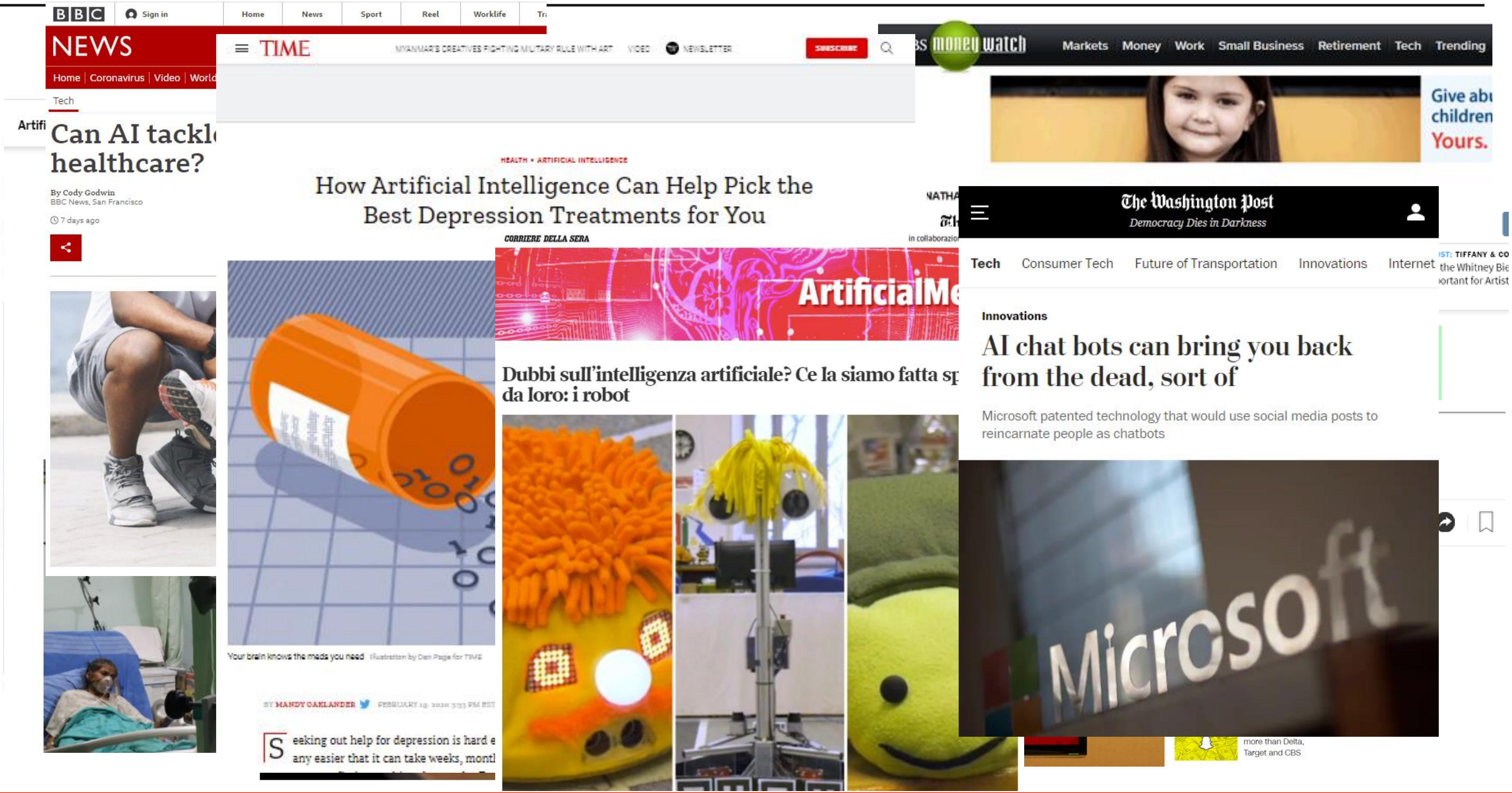
Development of intelligent systems (RInS)

Object recognition with Convolutional Neural Networks

Danijel Skočaj
University of Ljubljana
Faculty of Computer and Information Science

Academic year: 2022/23

Media hype

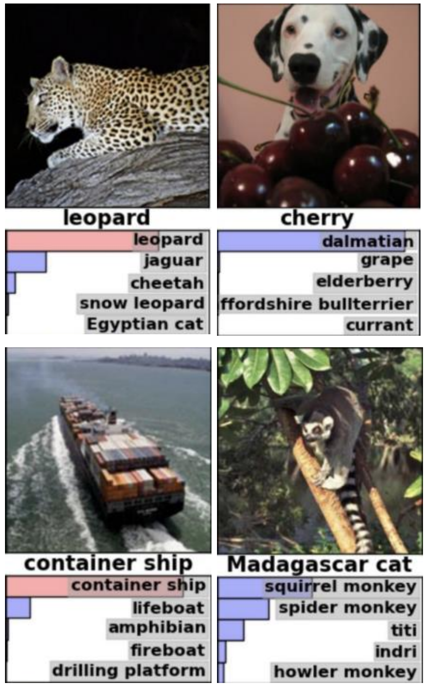


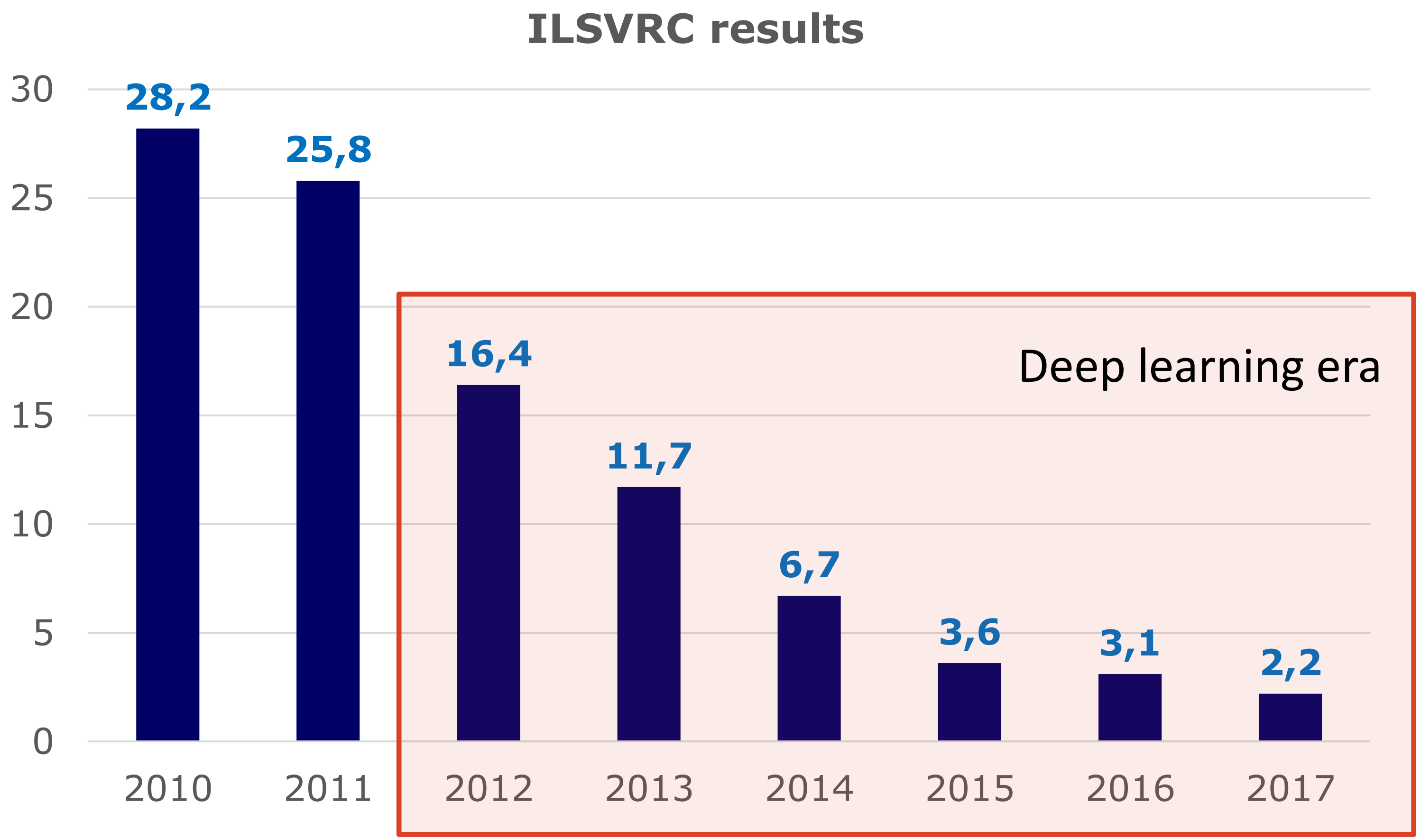
BBC
Sign in
Home
News
Sport
Reel
Worklife
NEWS
Home | Coronavirus | Video | World
Tech
Can AI tackl
healthcare?
By Cody Godwin
BBC News, San Francisco
7 days ago
TIME
HEALTH • ARTIFICIAL INTELLIGENCE
How Artificial Intelligence Can Help Pick the Best Depression Treatments for You
Your brain knows the meds you need
BY MANDY OAKLANDER
eeking out help for depression is hard e
any easier that it can take weeks, mont
CORRIERE DELLA SERA
ArtificialMe
Dubbi sull'intelligenza artificiale? Ce la siamo fatta sp
da loro: i robot
money watch
Markets
Money
Work
Small Business
Retirement
Tech
Trending
Give abu
children
Yours.
The Washington Post
Democracy Dies in Darkness
Tech
Consumer Tech
Future of Transportation
Innovations
Internet
Innovations
AI chat bots can bring you back from the dead, sort of
Microsoft patented technology that would use social media posts to reincarnate people as chatbots
Microsoft

Superior perfomance

IMAGENET

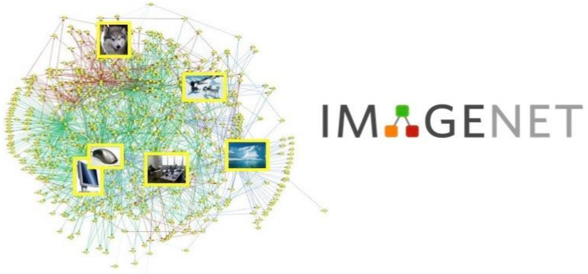
1k categories
1,3M images
Top5 classification





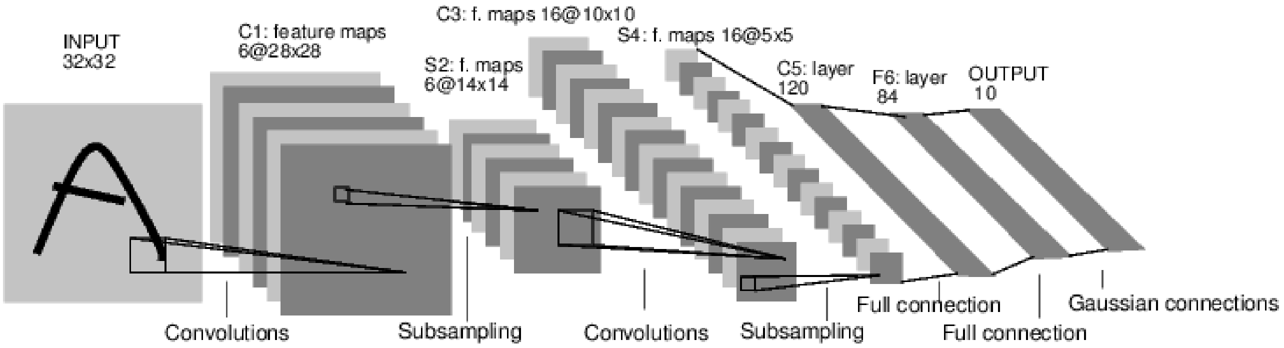
New deep learning era

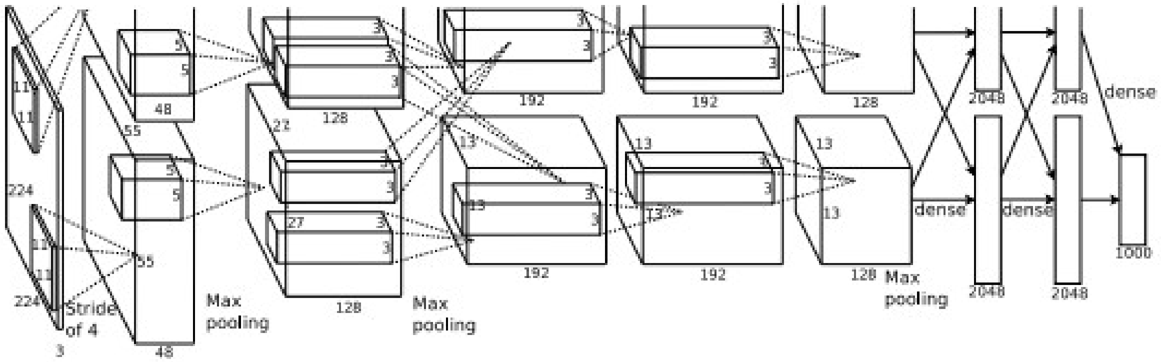
- More data!



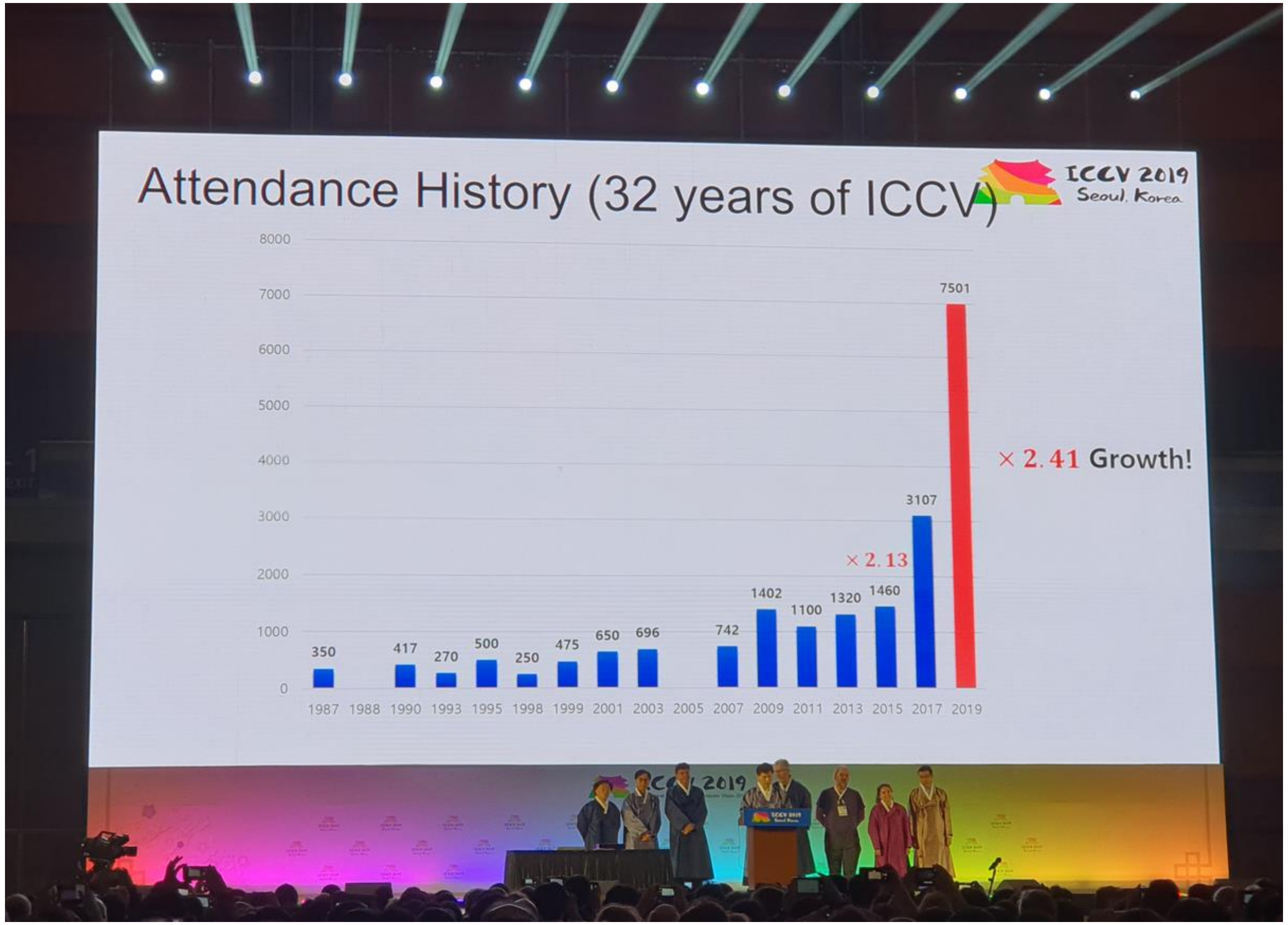
- More computing power - GPU!

- Better learning algorithms!





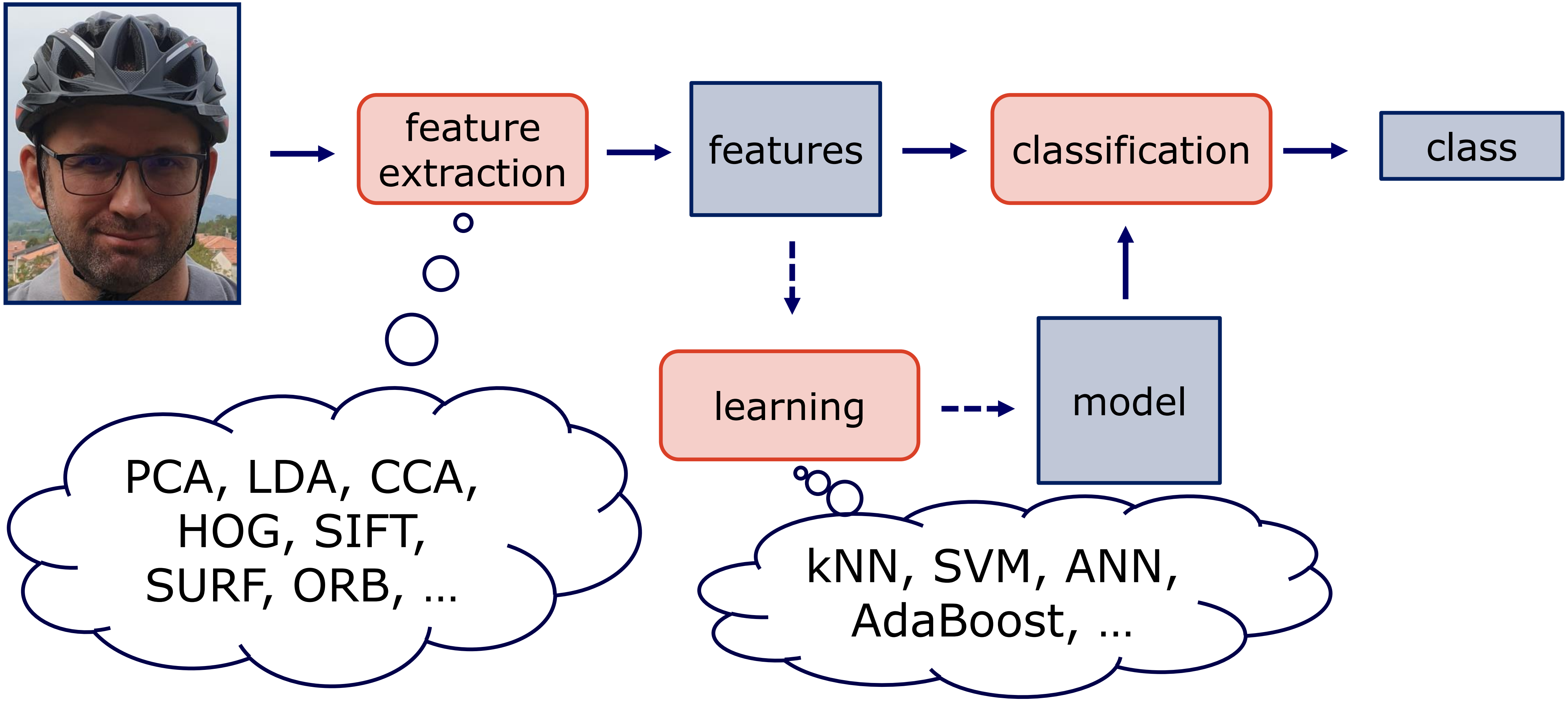
New deep learning era



ICCV 2019, Seoul, Korea, 27. 10. - 2. 11. 2019

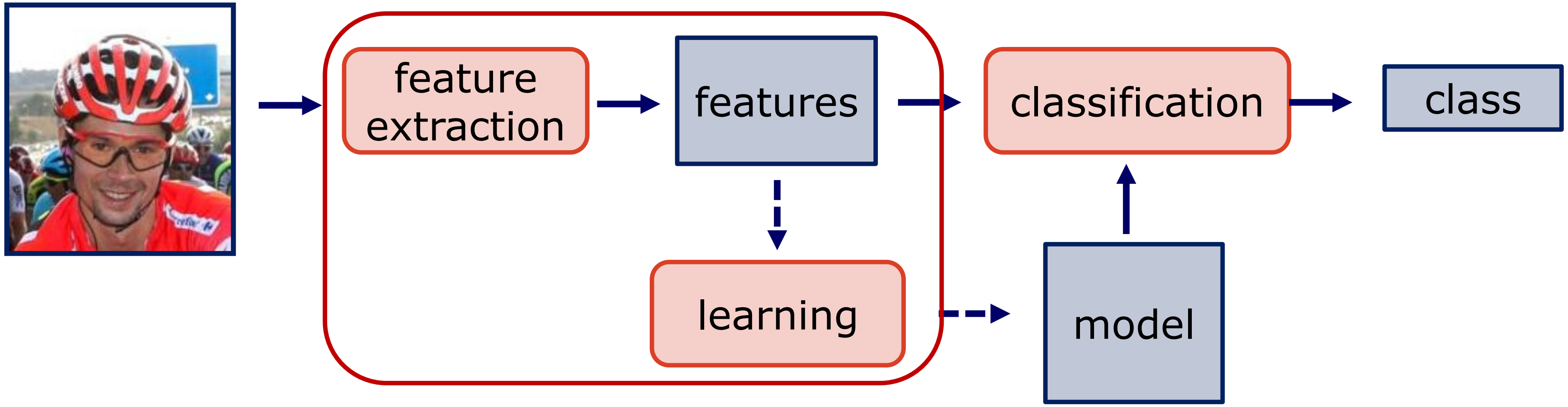
Machine learning in computer vision

- Conventional approach

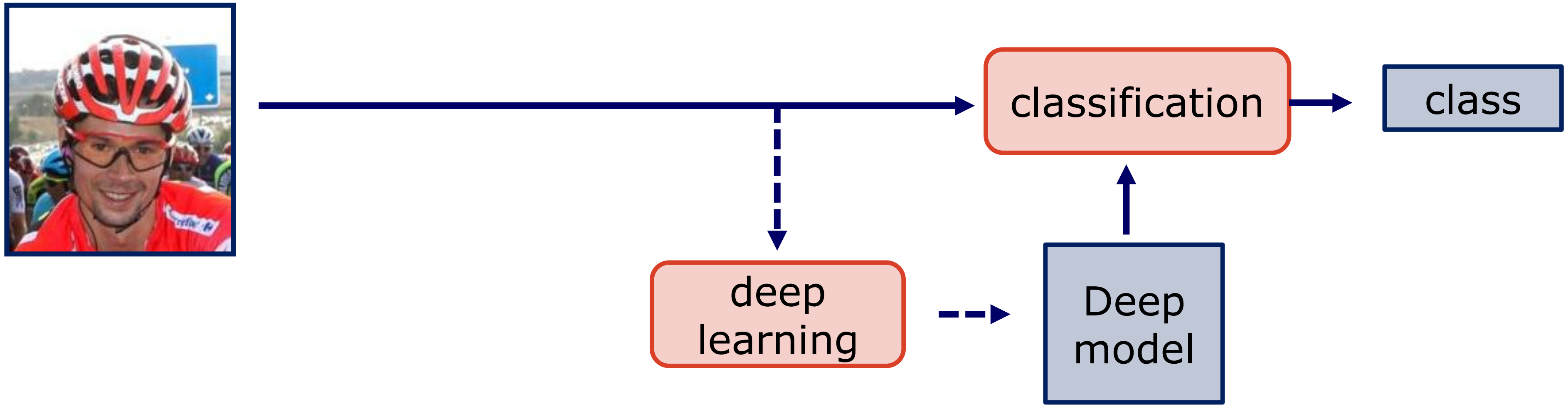


Deep learning in computer vision

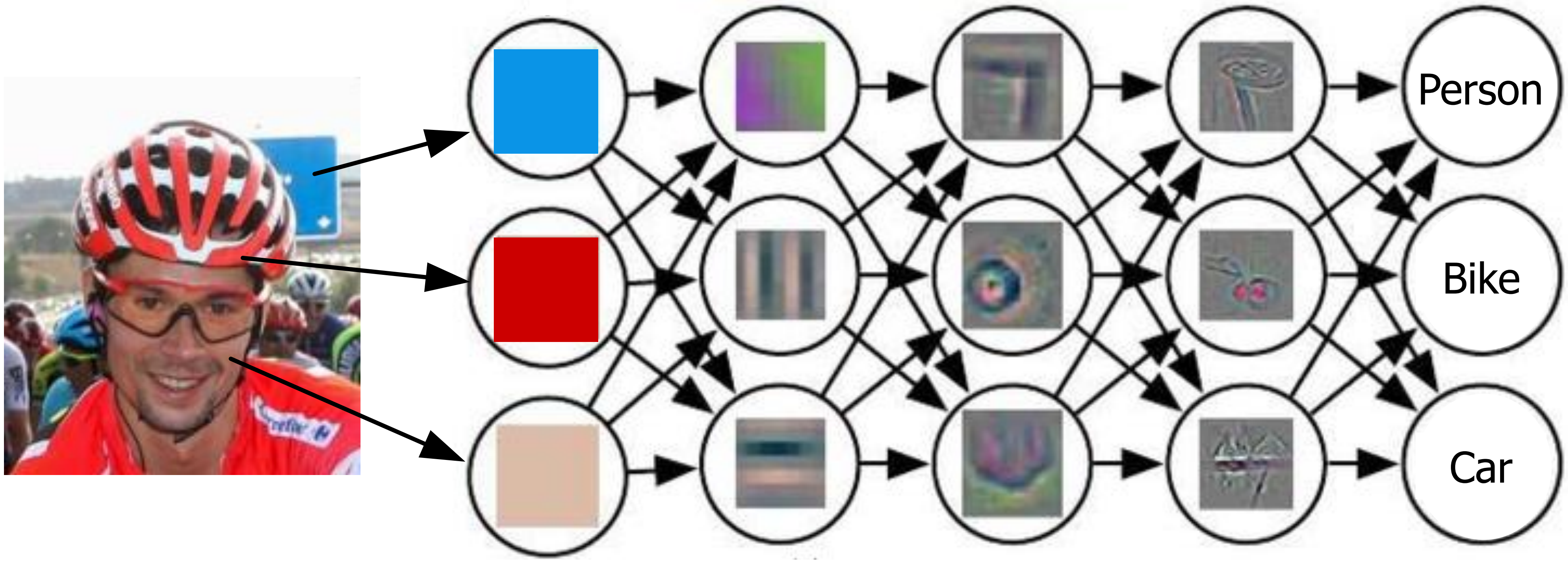
- Conventional machine learning approach in computer vision



- Deep learing approach

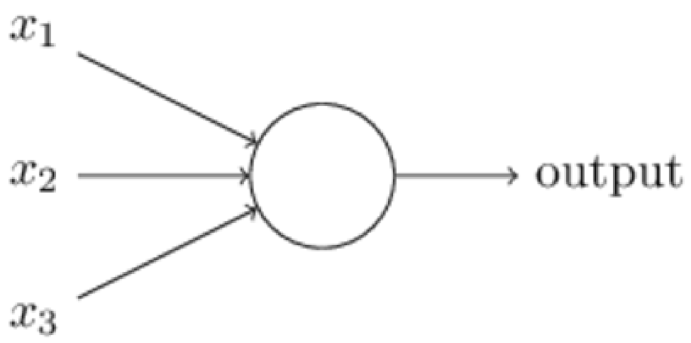


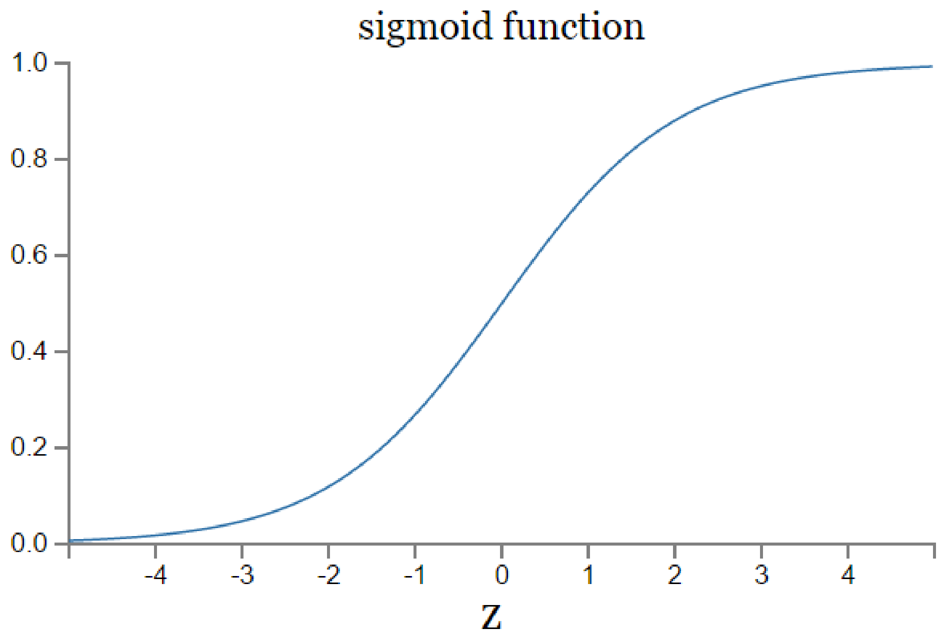
Deep learning – the main concept



Sigmoid neurons

- Real inputs and outputs from interval [0,1]





- Activation function: sgimoid function

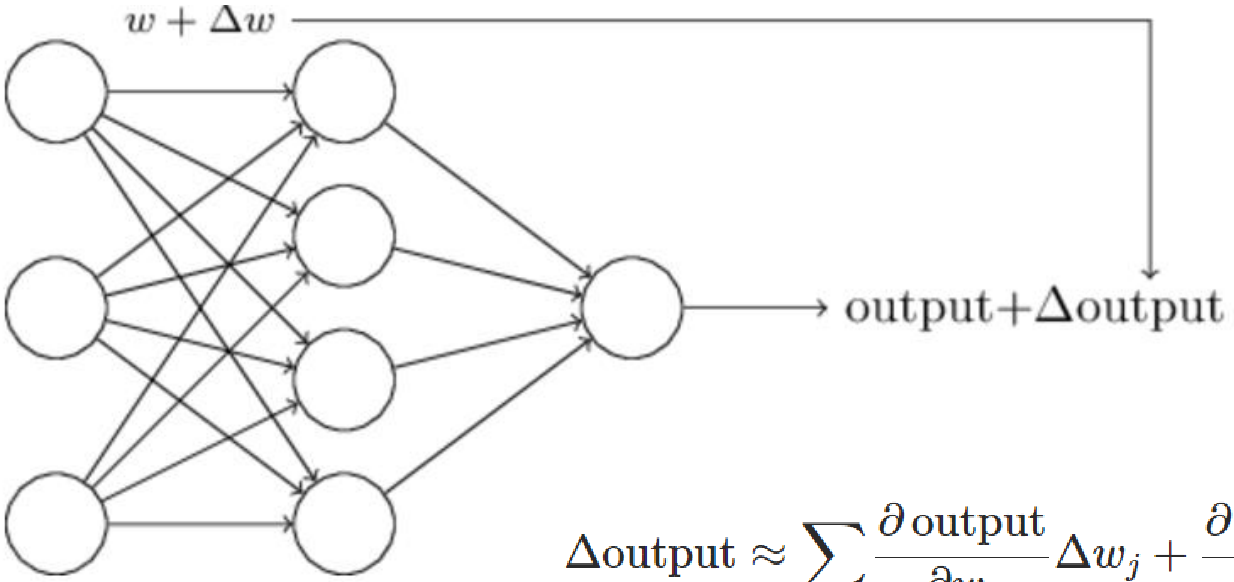
- *output* = $\dfrac{1}{1+\exp(-\sum_j w_j x_j - b)}$

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

$$\sigma(w \cdot x + b)$$

Sigmoid neurons

- Small changes in weights and biases causes small change in output

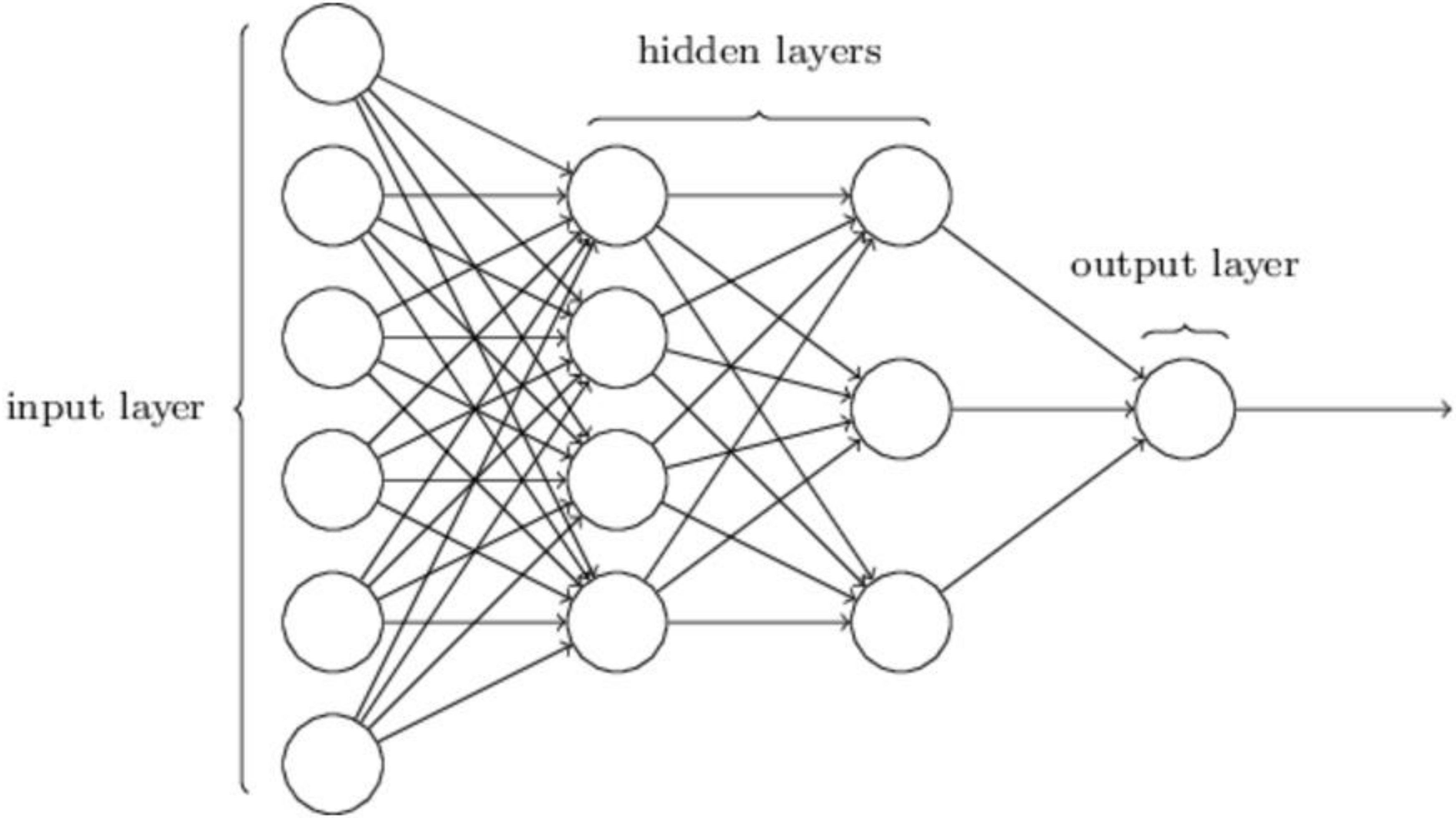


$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b$$

- Enables learning!

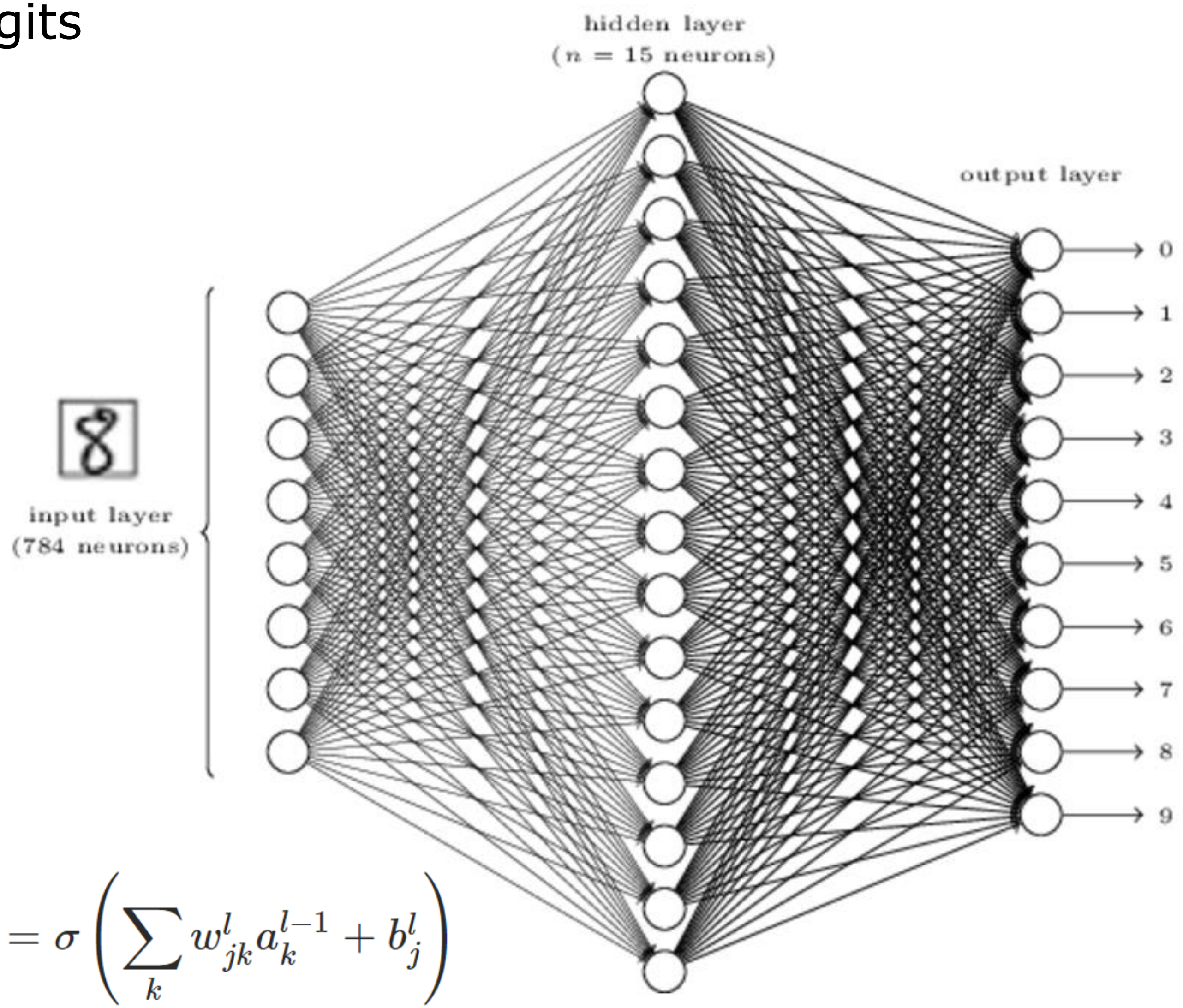
Feedfoward neural networks

- Network architecture:



Example: recognizing digits

- MNIST database of handwritten digits
  - 28x28 pixes (=784 input neurons)
  - 10 digits
  - 50.000 training images
  - 10.000 validation images
  - 10.000 test images



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Example code: Feedforward

- Code from https://github.com/mnielsen/neural-networks-and-deep-learning/archive/master.zip or *https://github.com/mnielsen/neural-networks-and-deep-learning*
  git clone https://github.com/mnielsen/neural-networks-and-deep-learning.git
- or *https://github.com/chengfx/neural-networks-and-deep-learning-for-python3* (for Python 3)

```
class Network(object):

    def __init__(self, sizes):
        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]

    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a


def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

```
net = network.Network([784, 30, 10])
net.SGD(training_data, 5, 10, 3.0, test_data=test_data)
```

```
In [55]: x,y=test_data[0]

In [56]: net.feedforward(x)
Out[56]:
array([[  1.83408119e-03],
       [  5.94472468e-08],
       [  1.84785949e-03],
       [  6.85718810e-04],
       [  1.41399919e-05],
       [  5.40491233e-06],
       [  4.74332685e-09],
       [  9.97920007e-01],
       [  8.19370561e-05],
       [  6.65086583e-05]])

In [57]: y
Out[57]: 7
```

Loss function

- Given:

$y\Big($  $\Big) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ for all training images

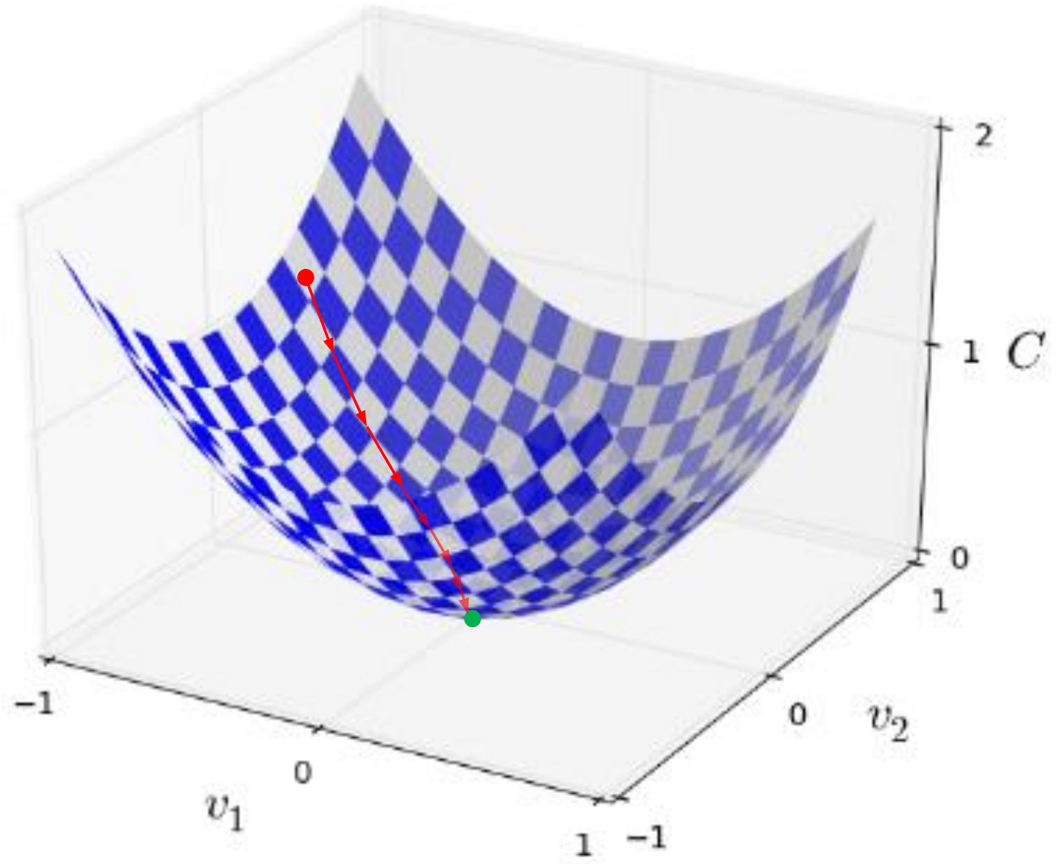
- Loss function: $C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$
  - (mean sqare error – quadratic loss function)

- Find weigths w and biases b that for given input x produce output a that minimizes Loss function C

Gradient descend

- Find minimum of $C(v_1, v_2)$
- Change of C: $\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \nabla C \cdot \Delta v \quad = -\eta \|\nabla C\|^2$
- Gradient of C: $\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$
- *Change v in the opposite direction of the gradient:* $\Delta v = -\eta \nabla C$

  Learning rate
- Algorithm:
  - Initialize v
  - Until stopping criterium riched
    - Apply udate rule $v \rightarrow v' = v - \eta \nabla C.$



Gradient descend in neural networks

- Loss function $C(w, b)$

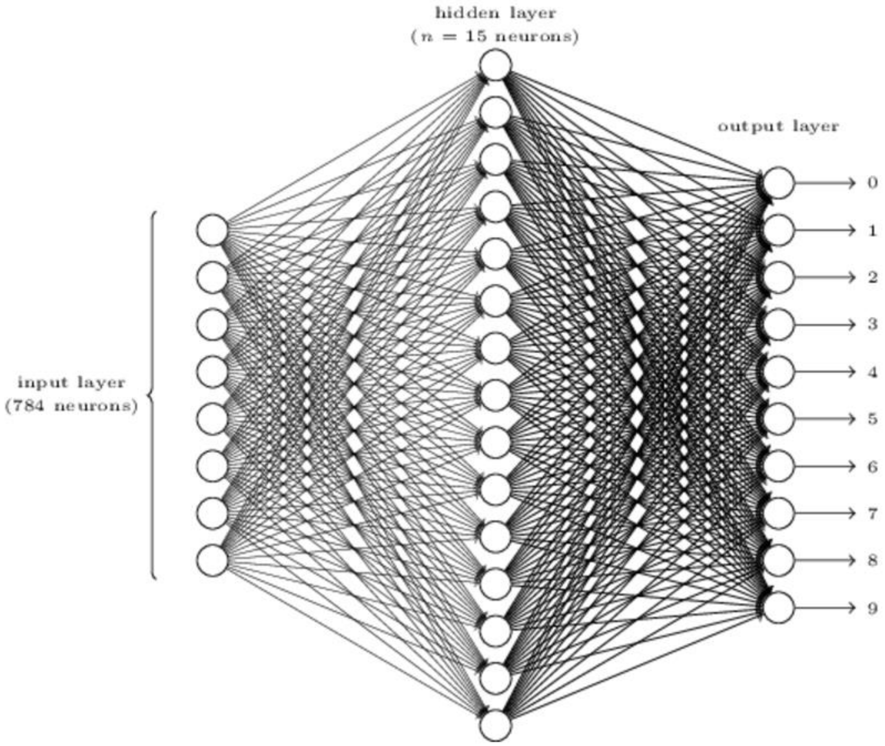
- Update rules:

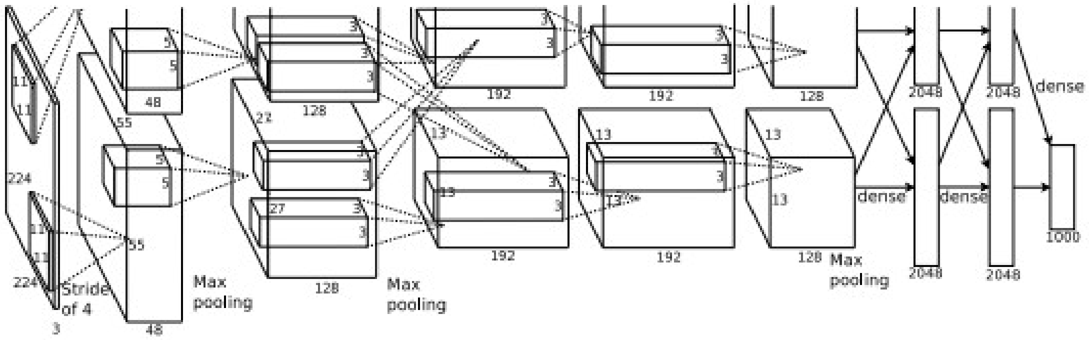
$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}$$

- Consider all training samples
- Very many parameters
  => computationaly very expensive

- Use Stochastic gradient descend instead





Example code: SGD

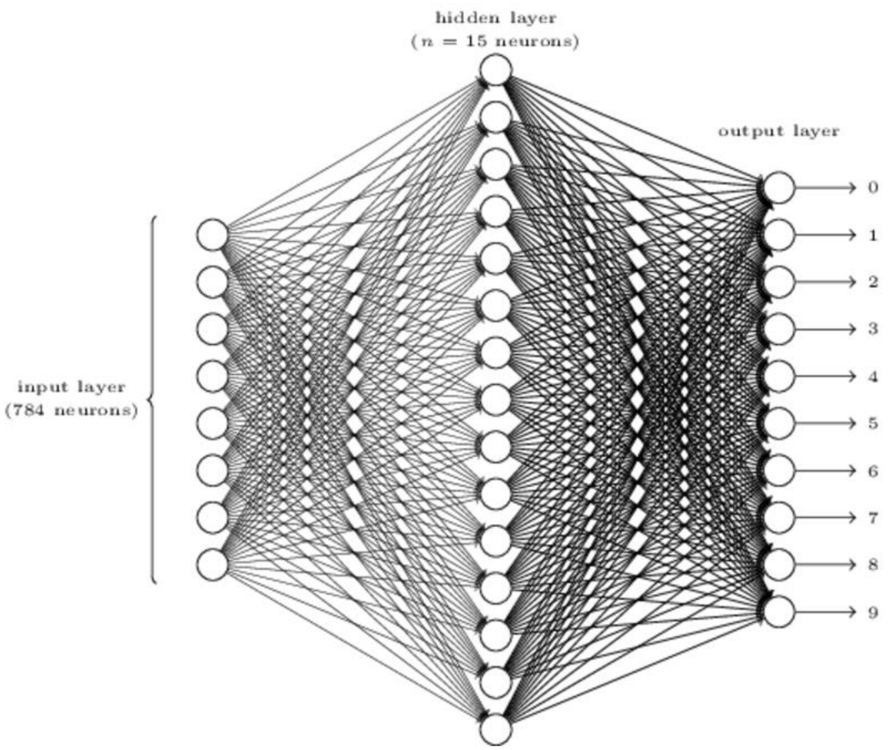
```
def SGD(self, training_data, epochs, mini_batch_size, eta):
    n = len(training_data)
    for j in xrange(epochs):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k+mini_batch_size]
            for k in xrange(0, n, mini_batch_size)]
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, eta)
```

```
def update_mini_batch(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

Backpropagation

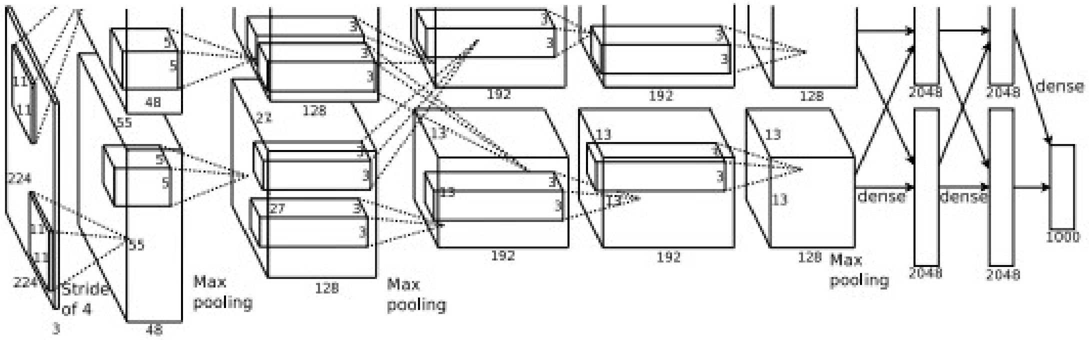
- All we need is gradient of loss function ∇C
  - Rate of change of C wrt. to change in any weigt
  - Rate of change of C wrt. to change in any biase

$$\frac{\partial C}{\partial b^l_j} \qquad \frac{\partial C}{\partial w^l_{jk}}$$



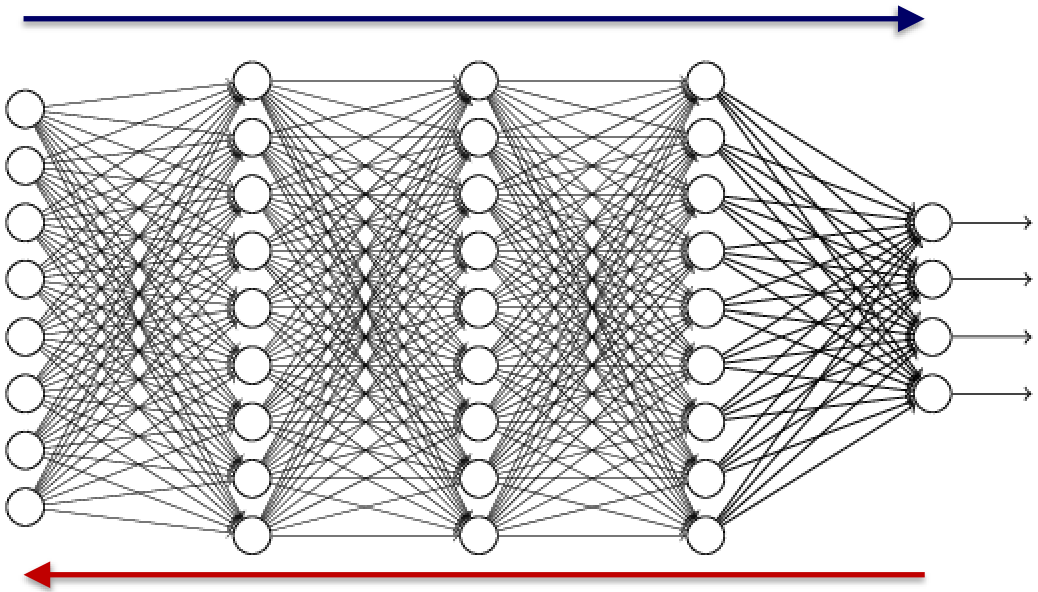
- How to compute gradient?
  - Numericaly
    - Simple, approximate, extremely slow ☹
  - Analyticaly for entire C
    - Fast, exact, nontractable ☹
  - Chain individual parts of netwok
    - Fast, exact, doable ☺

Backpropagation!



Main principle

- We need the gradient of the Loss function ∇C $\frac{\partial C}{\partial b_j^l}$ $\frac{\partial C}{\partial w_{jk}^l}$
- Two phases:
  - Forward pass; propagation: the input sample is propagated through the network and the error at the final layer is obtained



  - Backward pass; weight update: the error is backpropagated to the individual levels, the contribution of the individual neuron to the error is calculated and the weights are updated accordingly

Learning strategy

- To obtain the gradient of the Loss function ∇C : $\frac{\partial C}{\partial b_j^l}$ $\frac{\partial C}{\partial w_{jk}^l}$
  - For every neuron in the network calculate error of this neuron

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

    - This error propagates through the netwok causing the final error
    - Backpropagate the final error to get all δ_j^l
  - Obtain all $\frac{\partial C}{\partial b_j^l}$ and $\frac{\partial C}{\partial w_{jk}^l}$ from δ_j^l

Equations of backpropagation

- BP1: Error in the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \qquad\qquad \delta^L = \nabla_a C \odot \sigma'(z^L)$$

- BP2: Error in terms of the error in the next layer:

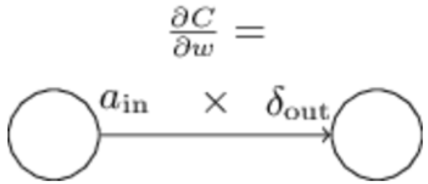
$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l) \qquad\qquad \delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l)$$

- BP3: Rate of change of the cost wrt. to any bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad\qquad \frac{\partial C}{\partial b} = \delta$$

- BP4: Rate of change of the cost wrt. to any weight:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \qquad\qquad \frac{\partial C}{\partial w} = a_{\text{in}}\delta_{\text{out}}$$



Backpropagation and SGD

For a number of **epochs**

Until all training images are used

Select a **mini-batch** of m training samples

For each training sample x in the mini-batch

Input: set the corresponding activation $a^{x,1}$

Feedforward: for each $l = 2, 3, \ldots, L$

compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$

Output error: compute $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$

Backpropagation: for each $l = L-1, L-2, \ldots, 2$

compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$

Gradient descend: for each $l = L, L-1, \ldots, 2$ and x update:

$$w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$$

$$b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

Example code: Backpropagation

```python
def backprop(self, x, y):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        # feedforward
        activation = x
        activations = [x] # list to store all the activations, layer by layer
        zs = [] # list to store all the z vectors, layer by layer
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)
        # backward pass
        delta = self.cost_derivative(activations[-1], y) * \
            sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())
        for l in xrange(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
        return (nabla_b, nabla_w)
```

```python
def cost_derivative(self, output_activations, y):
    return (output_activations-y)
```

```python
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))
```

```python
def sigmoid_prime(z):
    return sigmoid(z)*(1-sigmoid(z))
```

Activation and loss functions

Activation function	Loss function
Linear $$a_j^L = z_j^L$$	Quadratic $$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$
Sigmoid $$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$	Cross-entropy $$C = -\frac{1}{n} \sum_x \sum_j \left[y_j \ln a_j^L + (1-y_j) \ln(1-a_j^L) \right]$$
Softmax $$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$	Categorical Cross-entropy $$C = -\frac{1}{n} \sum_x \sum_j y_j \ln a_j^L$$
Other	Custom

Activation functions

Method	Papers
ReLU	8096
Sigmoid Activation	5363
GELU Gaussian Error Linear Units (GELUs)	5285
Tanh Activation	4936
Leaky ReLU	915
GLU Language Modeling with Gated Convolutional Networks	372
Swish Searching for Activation Functions	254
Softplus	204
Mish	183
SELU Self-Normalizing Neural Networks	178
PReLU Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification	86
ReLU6 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications	58
Hard Swish Searching for MobileNetV3	54
Maxout Maxout Networks	45
ELU Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)	34

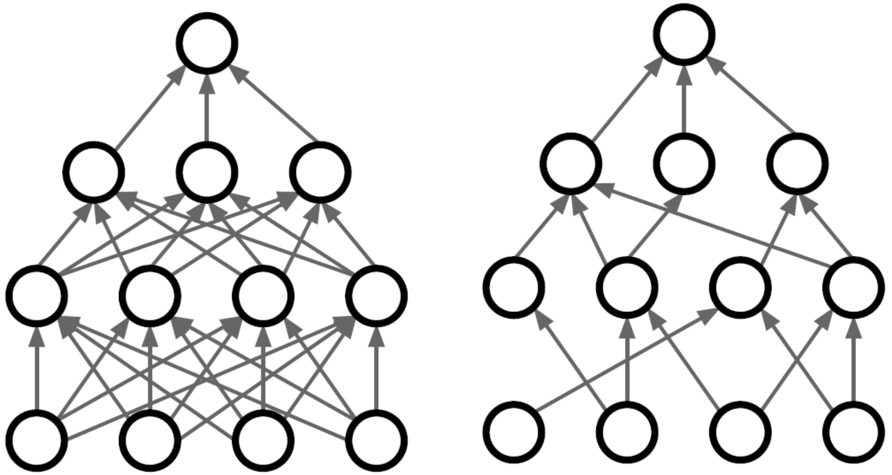
[https://paperswithcode.com]

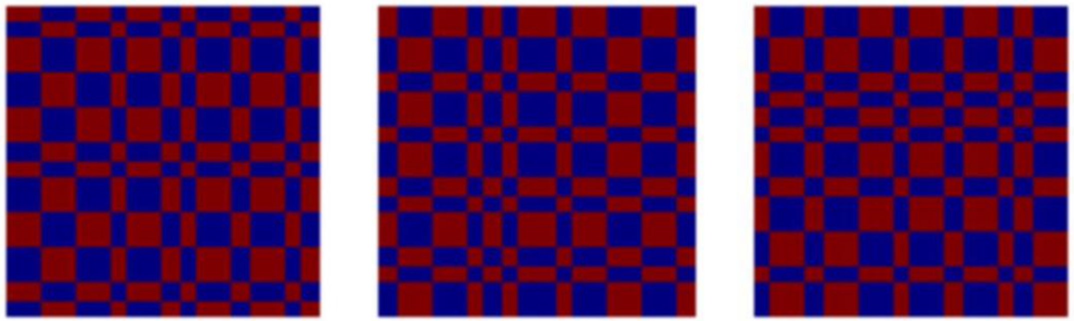
Regularisation

- How to avoid overfitting:
  - Increase the number of training images ☹
  - Decrease the number of parameters ☹
  - Regularization ☺

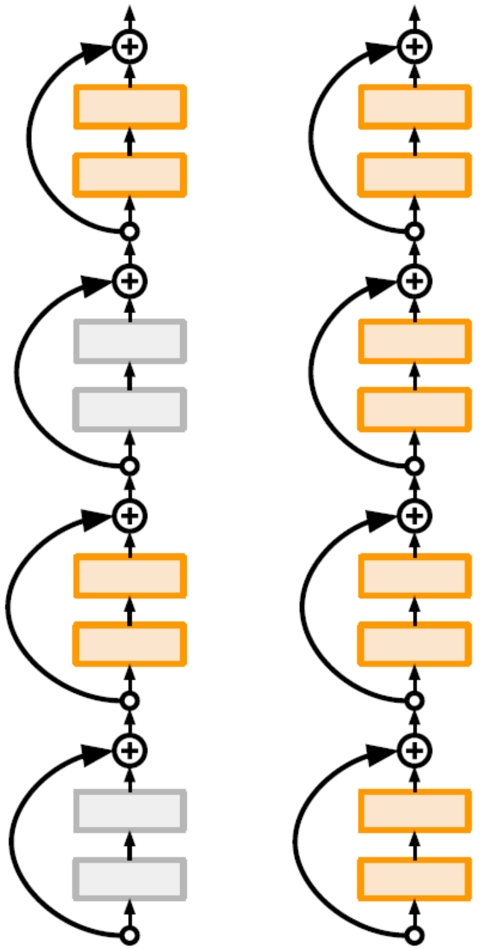
- Data Augmentation
- L1 regularisation
- L2 regularisation
- Dropout
- Batch Normalization
- DropConnect
- Fractional Max Pooling
- Stochastic Depth
- Cutout / Random Crop
- Mixup

[Wan et al. 2013]





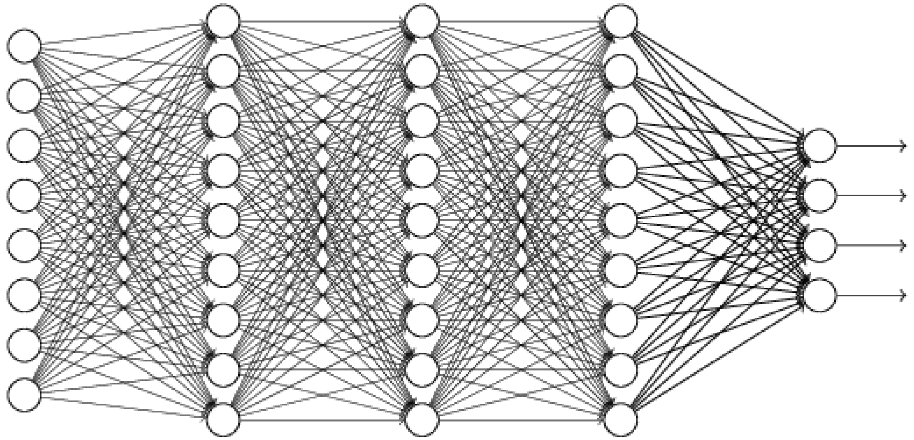
[Graham, 2014]



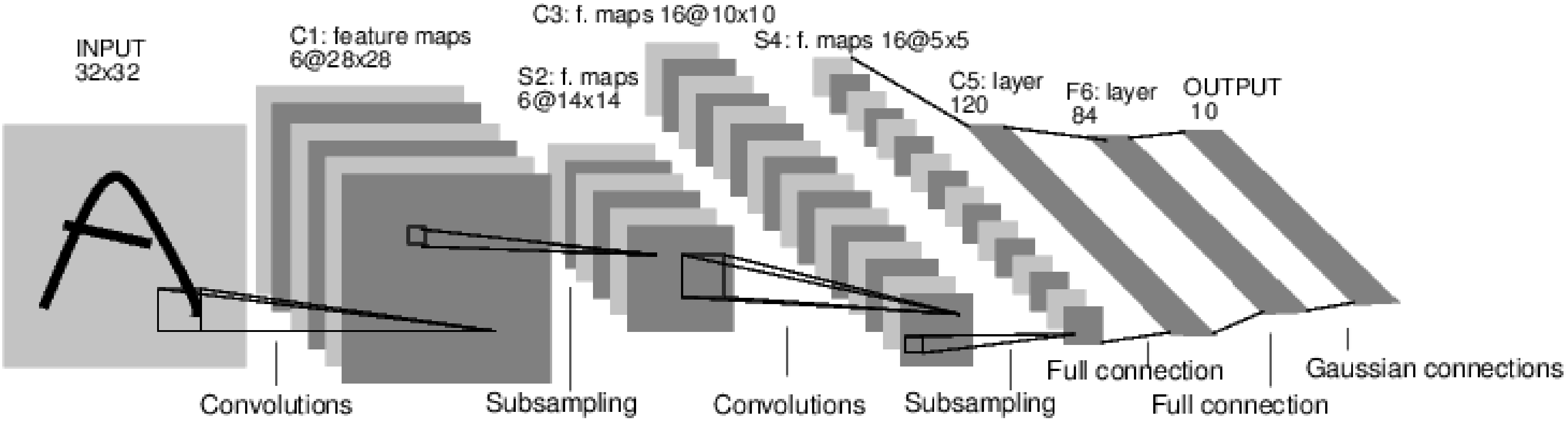
[Huang et al. 2016]

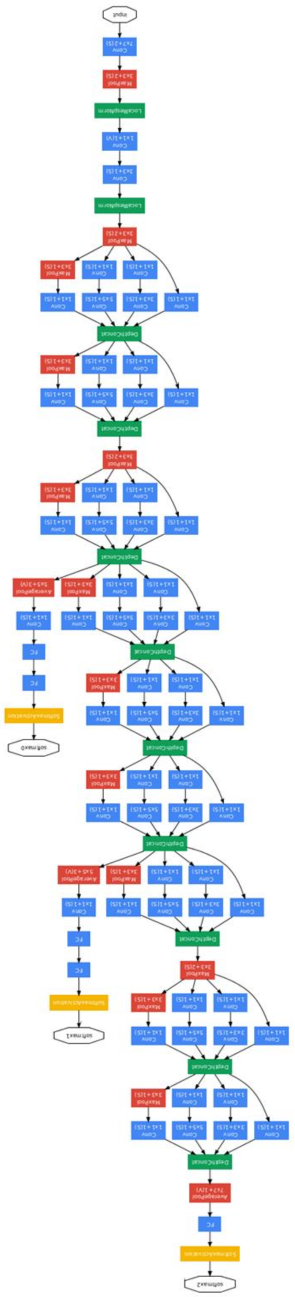
Convolutional neural networks

- From feedforward fully-connected neural networks



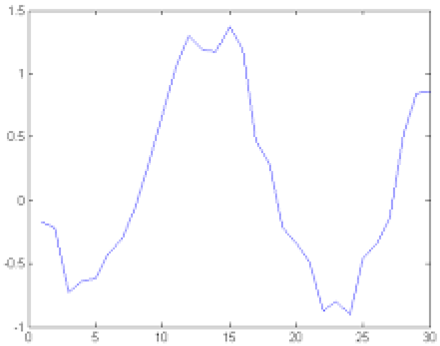
- To convolutional neural networks

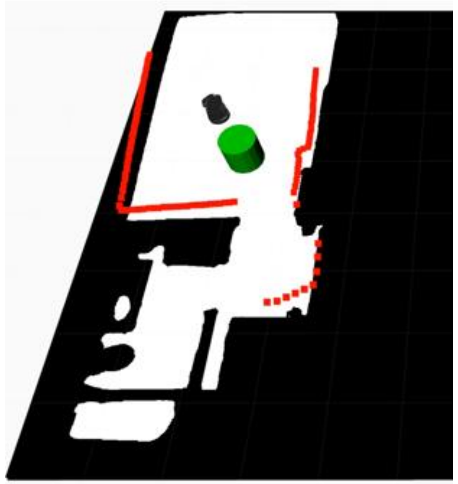


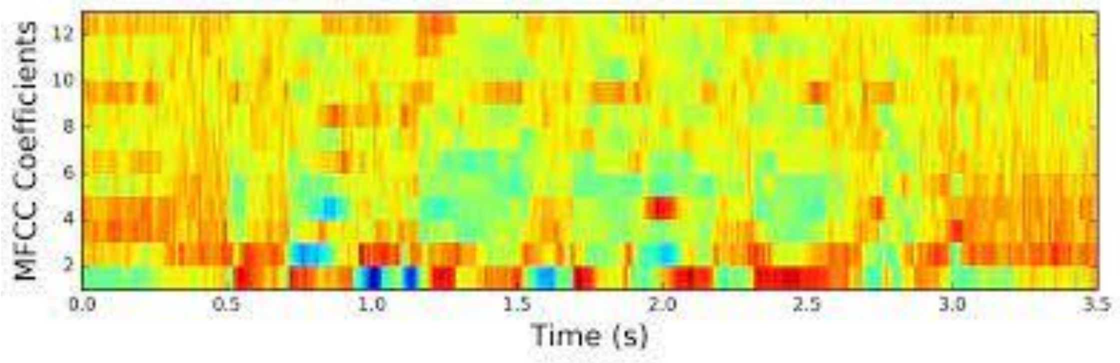


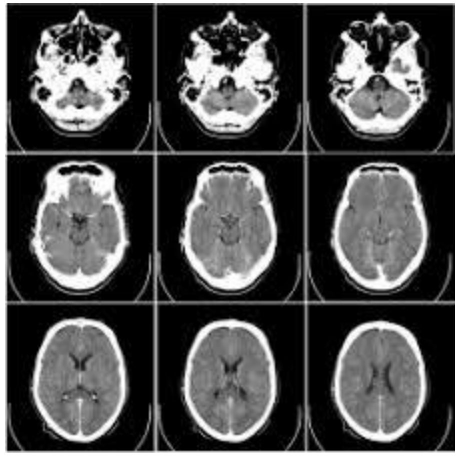
Convolutional neural networks

- Data in vectors, matrices, tensors
- Neigbourhood, spatial arrangement
- 2D: Images,time-fequency representations



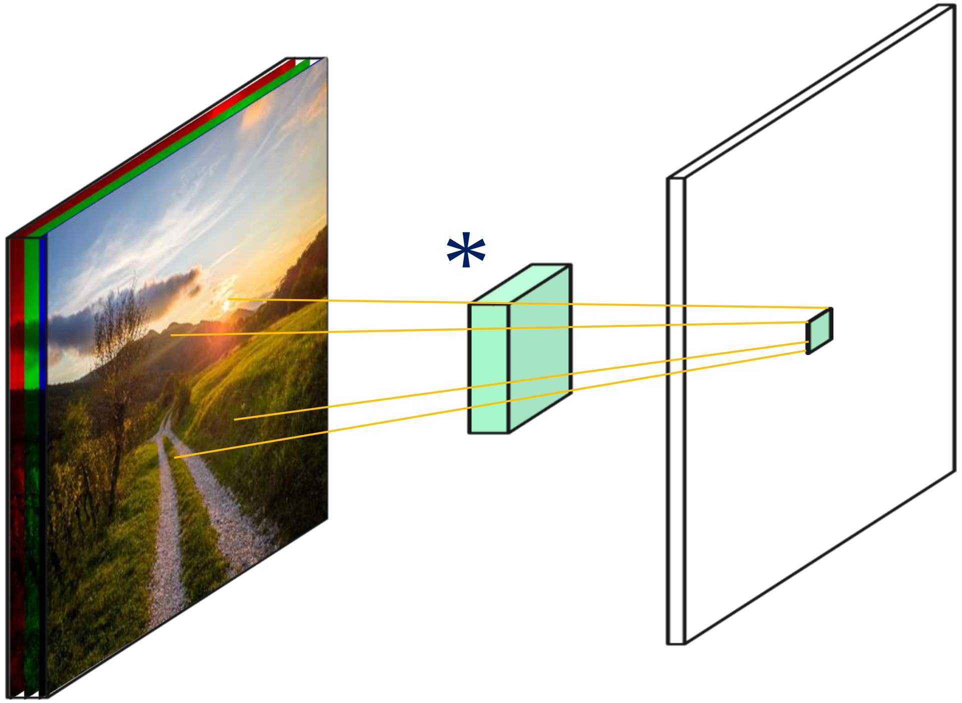




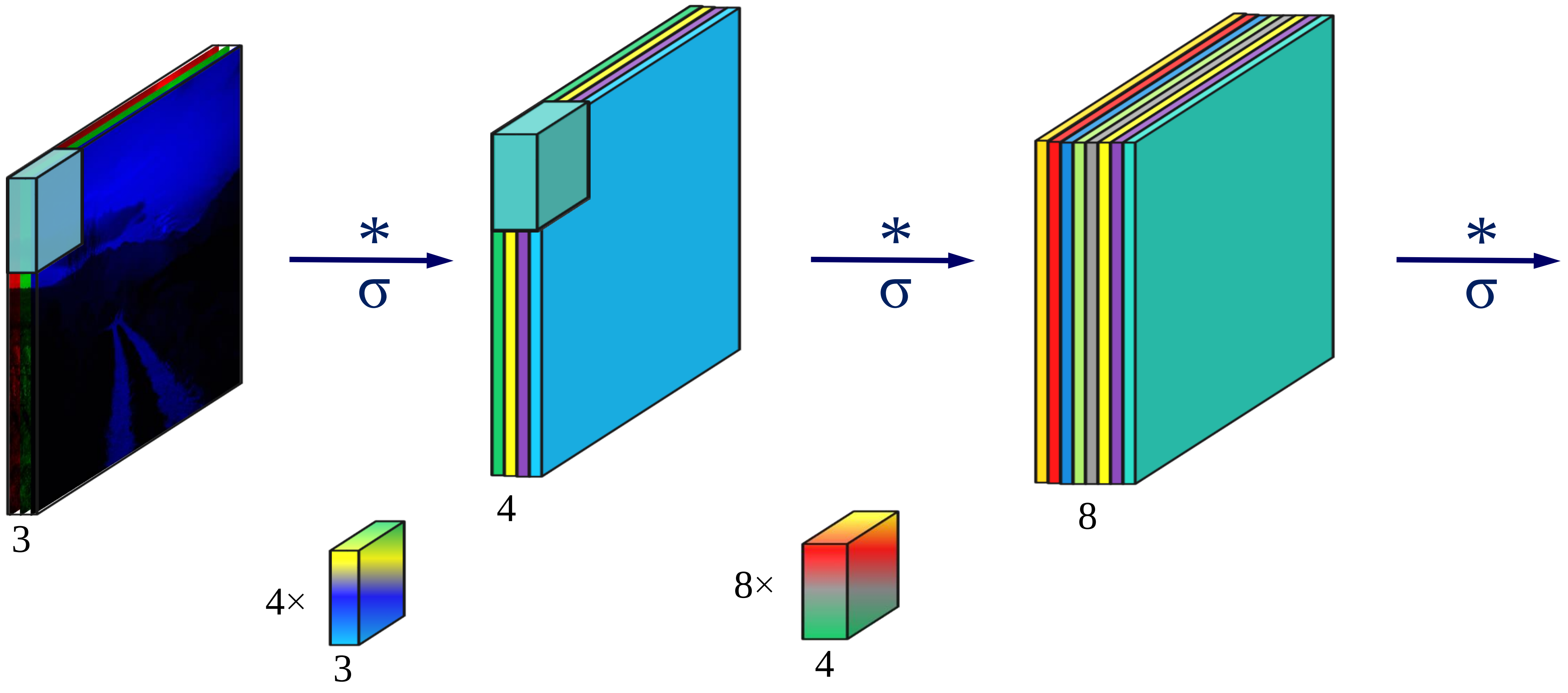


- 1D: sequential signals, text, audio, speech, time series,…
- 3D: volumetric images, video, 3D grids

Convolution layer

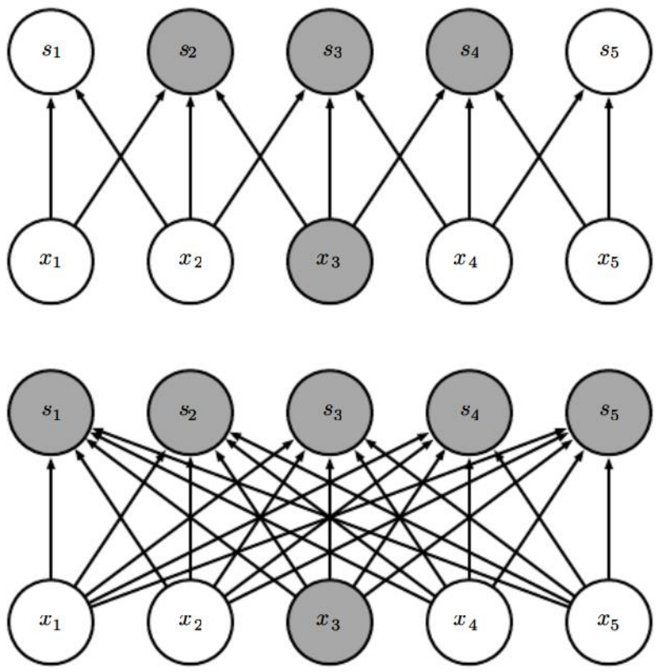


Convolution layer

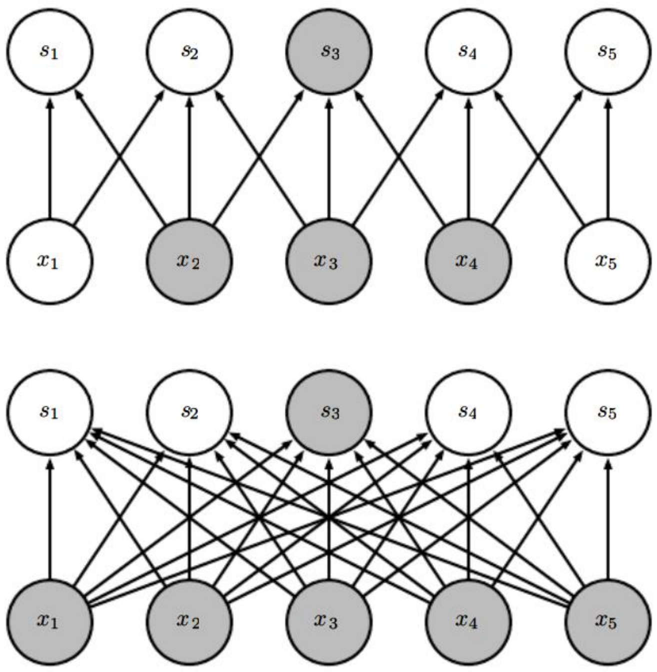


Sparse connectivity

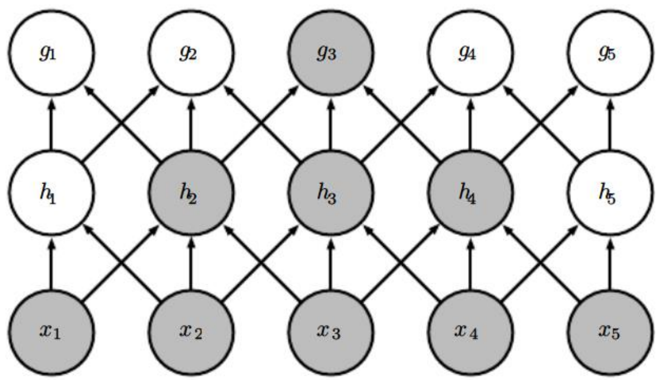
- Local connectivity – neurons are only locally connected (**receptive field**)
  - Reduces memory requirements
  - Improves statistical efficiency
  - Requires fewer operations



from below



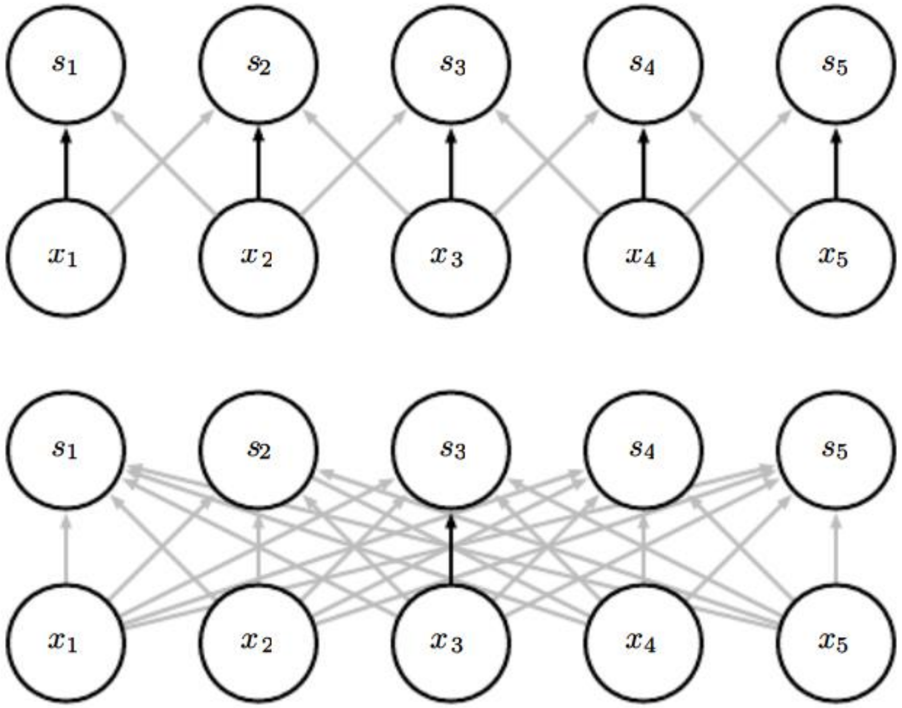
from above



The receptive field of the units in the deeper layers is large
=> Indirect connections!

Parameter sharing

- **Neurons share weights!**
  - Tied weights
- Every element of the kernel is used at every position of the input
- All the neurons at the same level detect the same feature (everywhere in the input)
- Greatly reduces the number of parameters!



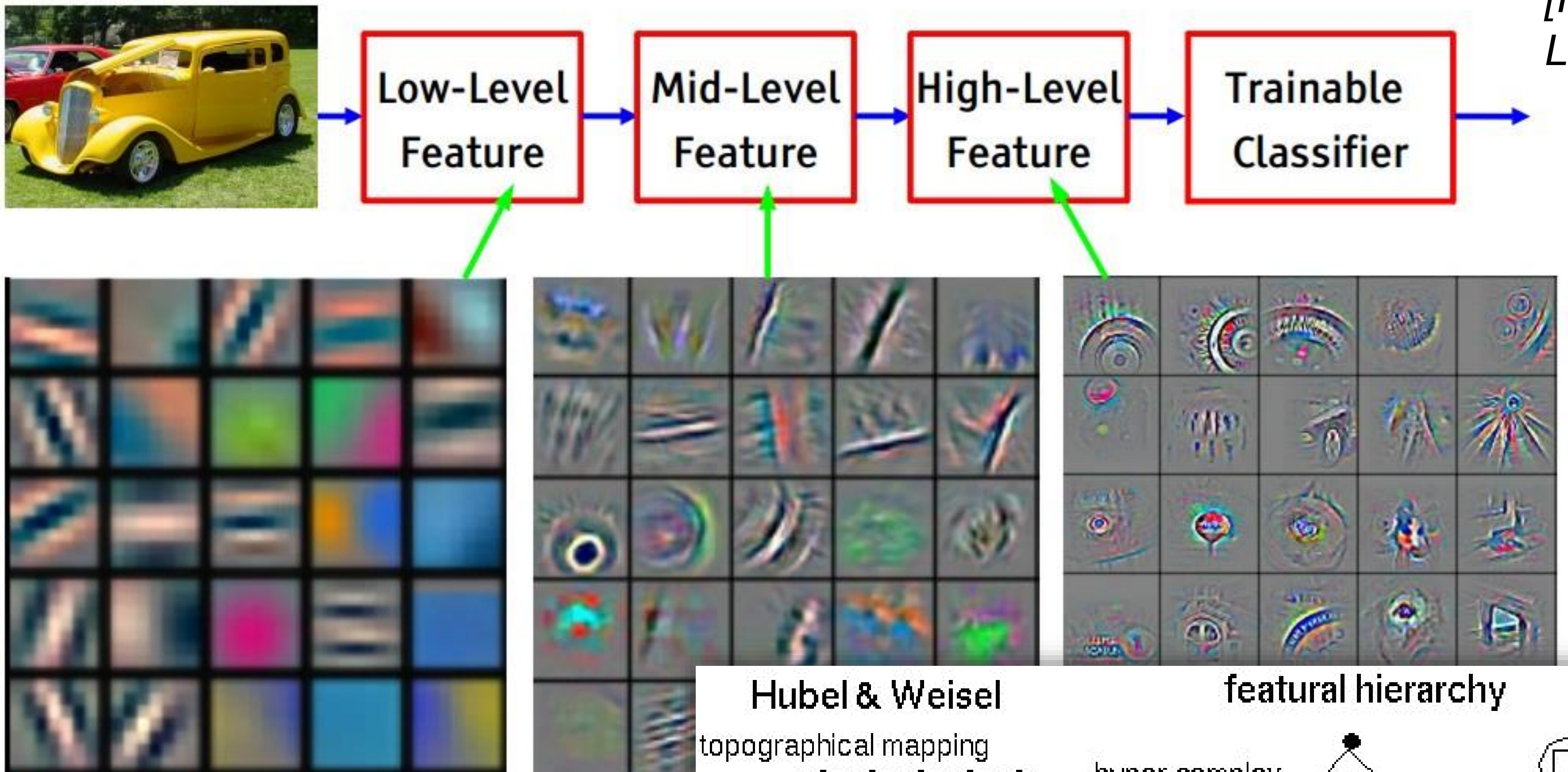
- **Equivariance to translation**
  - Shift, convolution = convolution, shift
  - Object moves => representation moves

- Fully connected network with an infinitively strong prior over its weights
  - Tied weights
  - Weights are zero outside the kernel region

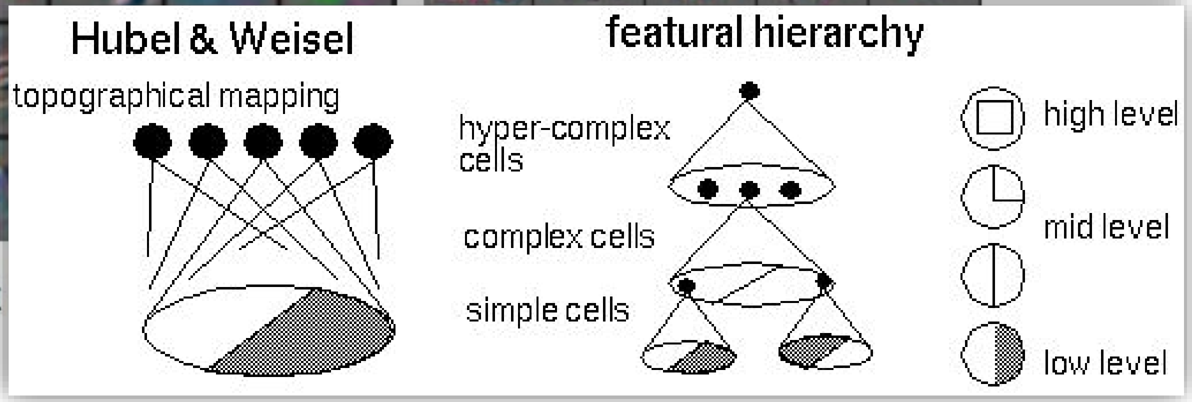
  => learns only local interactions and is equivariant to translations

Convolutional neural network

[From recent Yann LeCun slides]

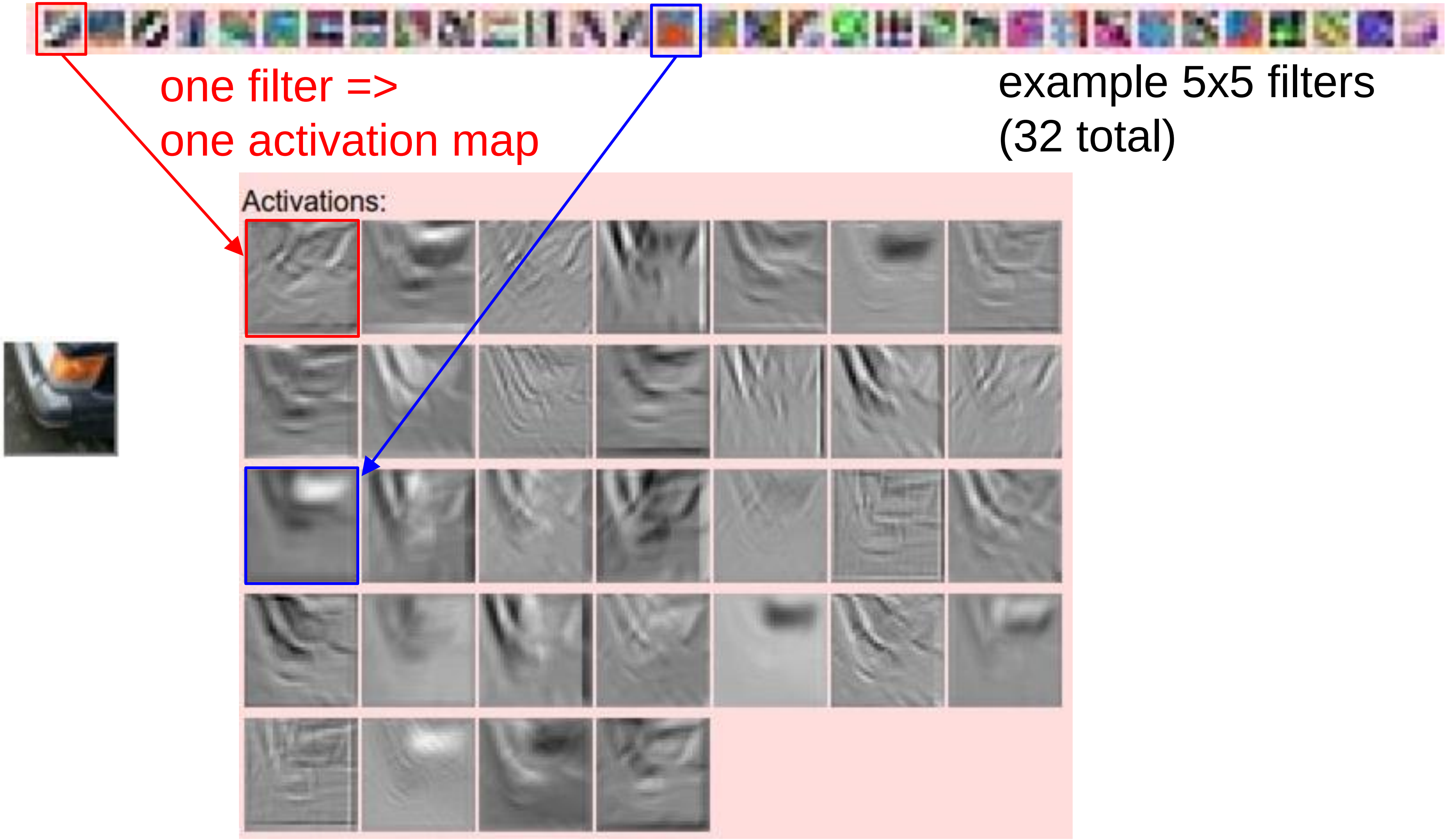


Feature visualization of convolutional net



Slide credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

Convolutional neural network

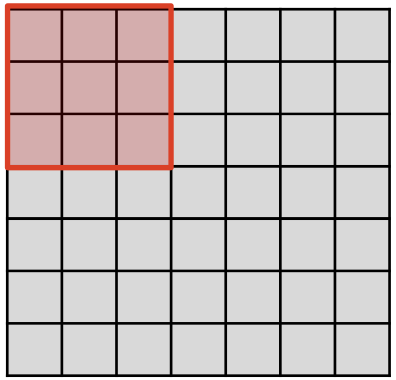


input image:

Slide credit: Fei-Fei Li, Andrej Karpathy, Justin Johnson

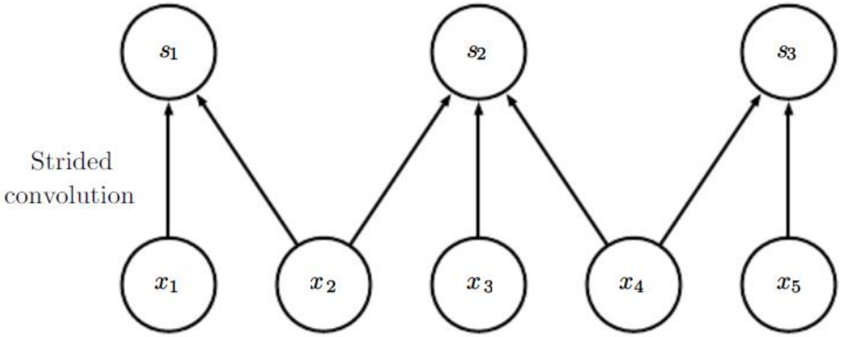
Stride

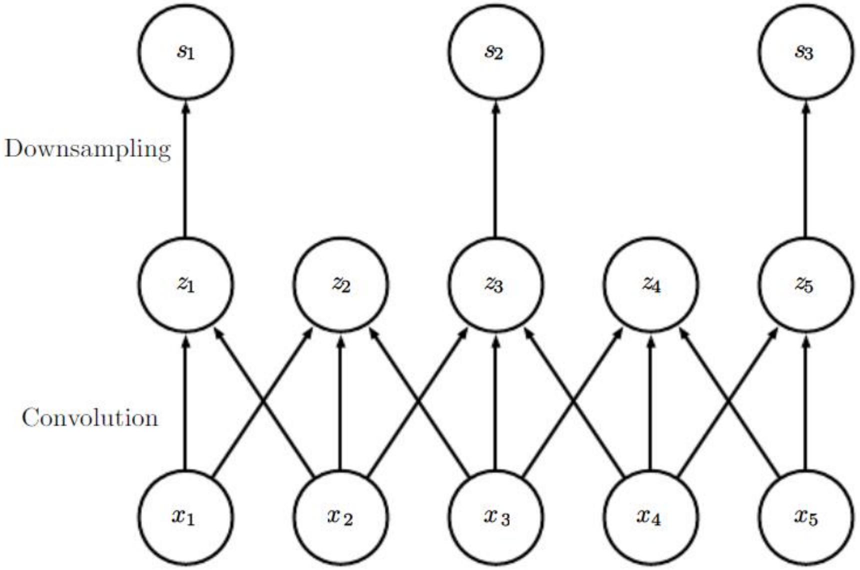
- Step for convolution filter



Stride=1
Stride=2

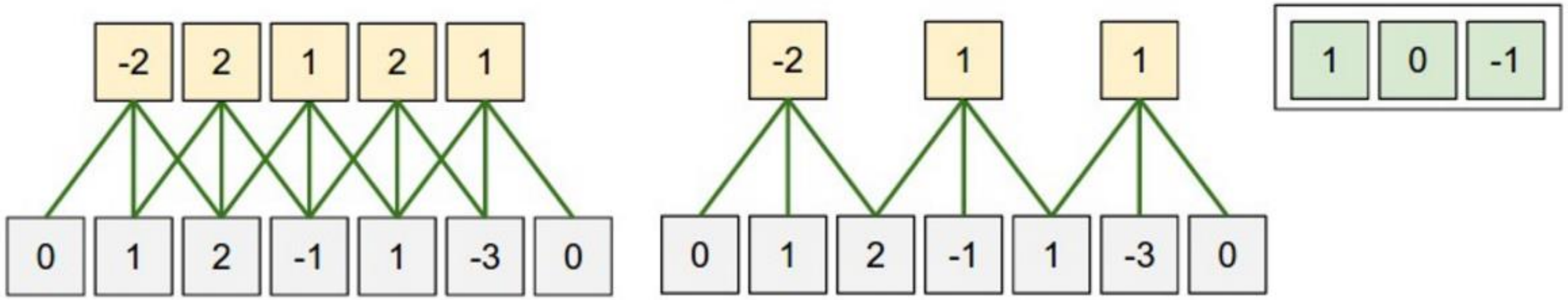
Convolution with stride>1
is equivalent to
convolution + downsampling





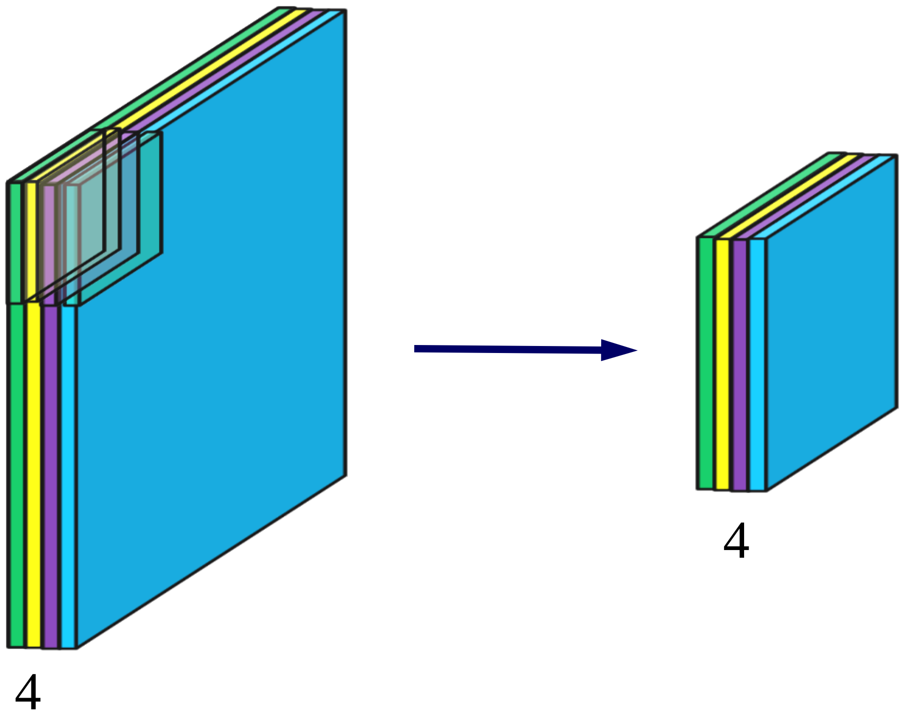
- Output size:
$$\frac{N-F}{S}+1$$

- Example:

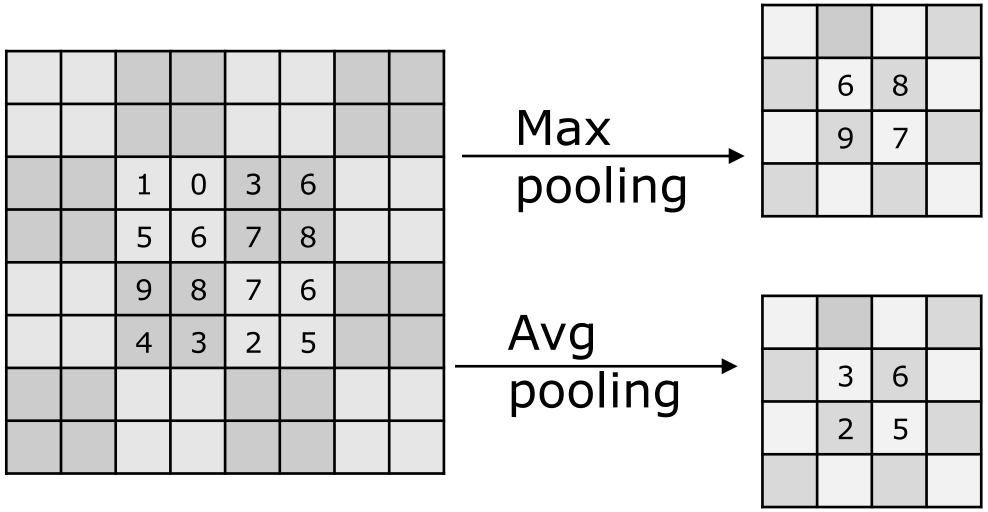


Pooling layer

- Downsampling – reduces the volume size (width and height)
- Process each activation map independently – keeps the volume depth unchanged

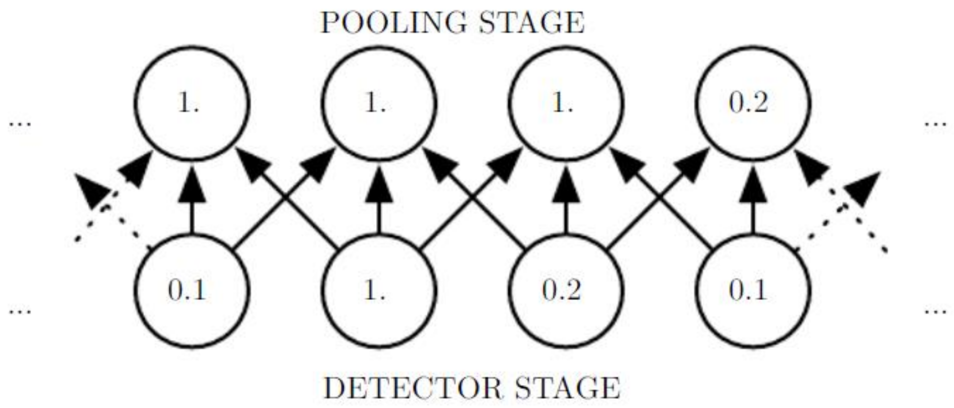


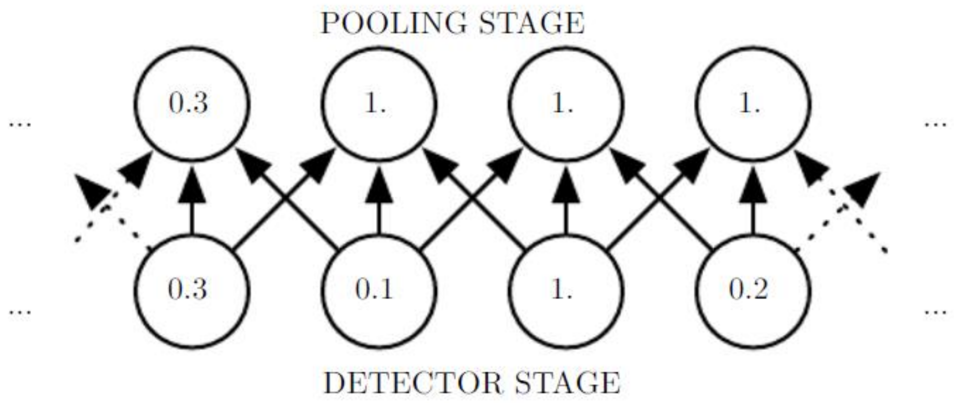
- Example with
  - F=2
  - S=2



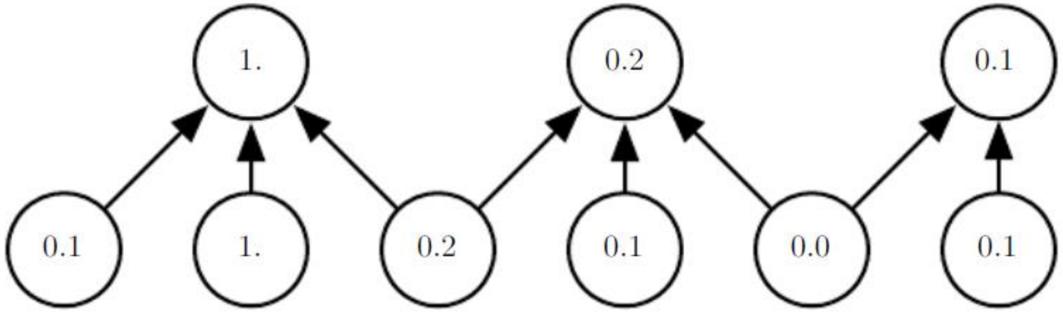
Pooling

- Max pooling introduces translation invariance

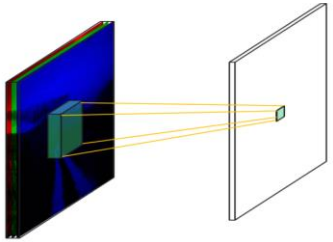
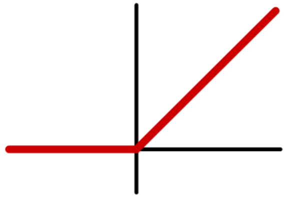
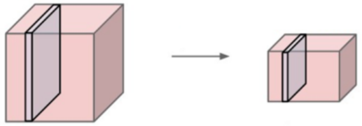
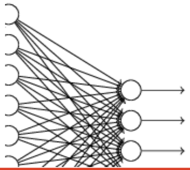


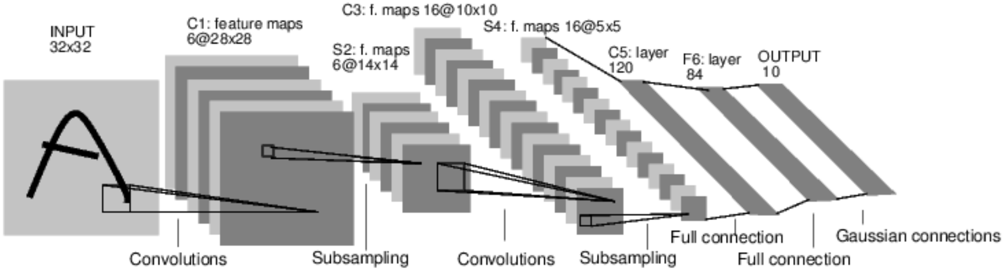


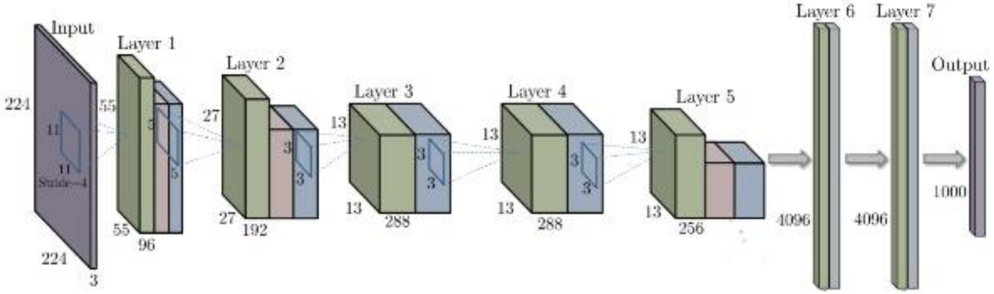
- Pooling with downsampling
  - Reduces the representation size
  - Reduces computational cost
  - Increases statistical efficiency

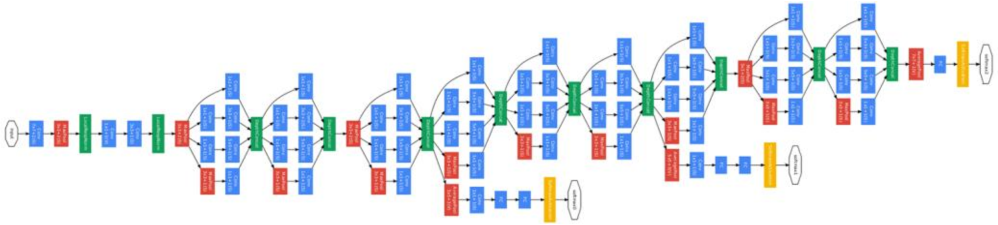


CNN layers

- Layers used to build ConvNets:
  - INPUT:
    raw pixel values
    
  - CONV:
    convolutional layer
    
  - (BN: batch nornalisation)
  - (ReLU:)
    introducing nonlinearity
    
  - POOL:
    downsampling
    
  - FC:
    for computing class scores
    
  - SoftMax

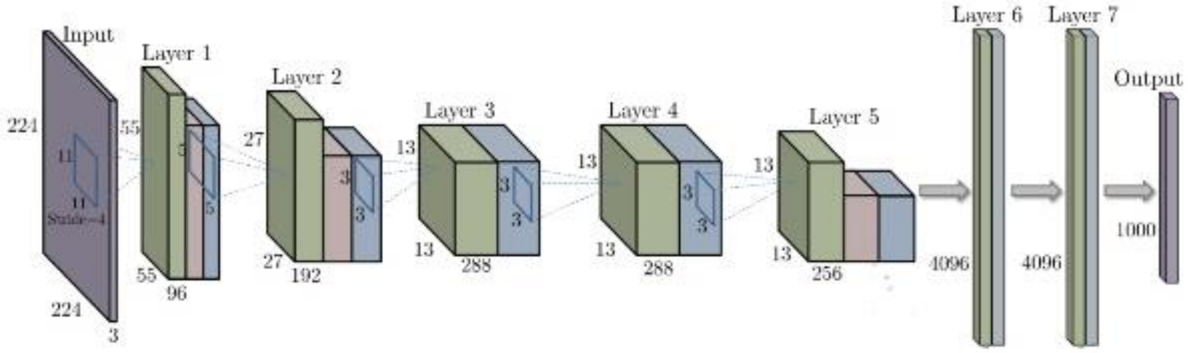




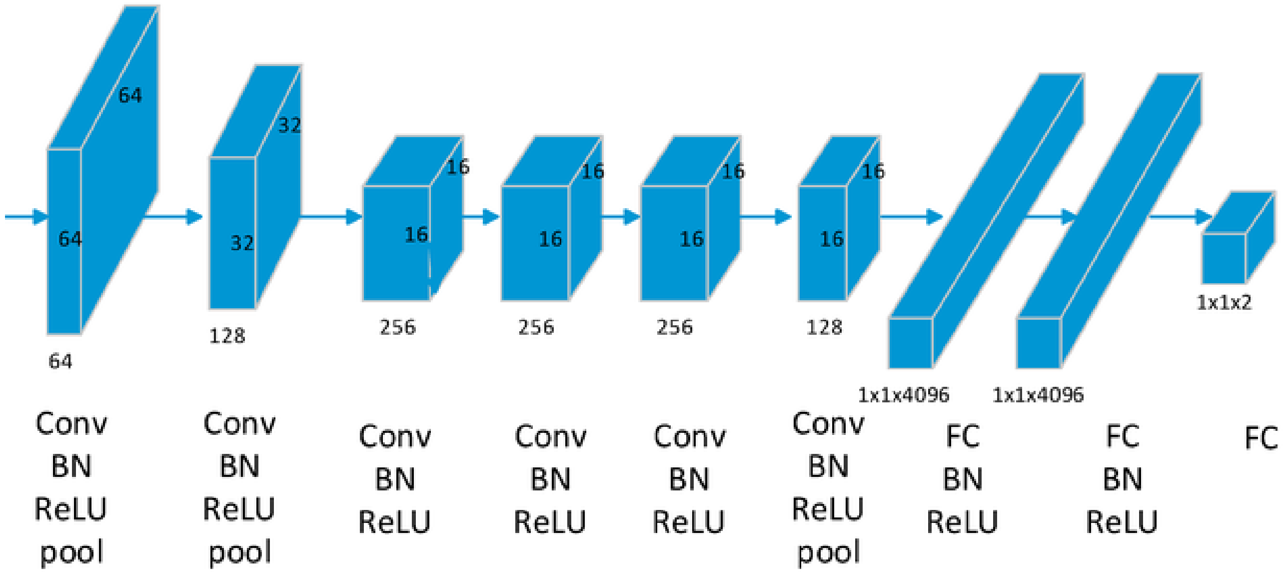


CNN architecture

- Stack the layers in an appropriate order



Babenko et. al.



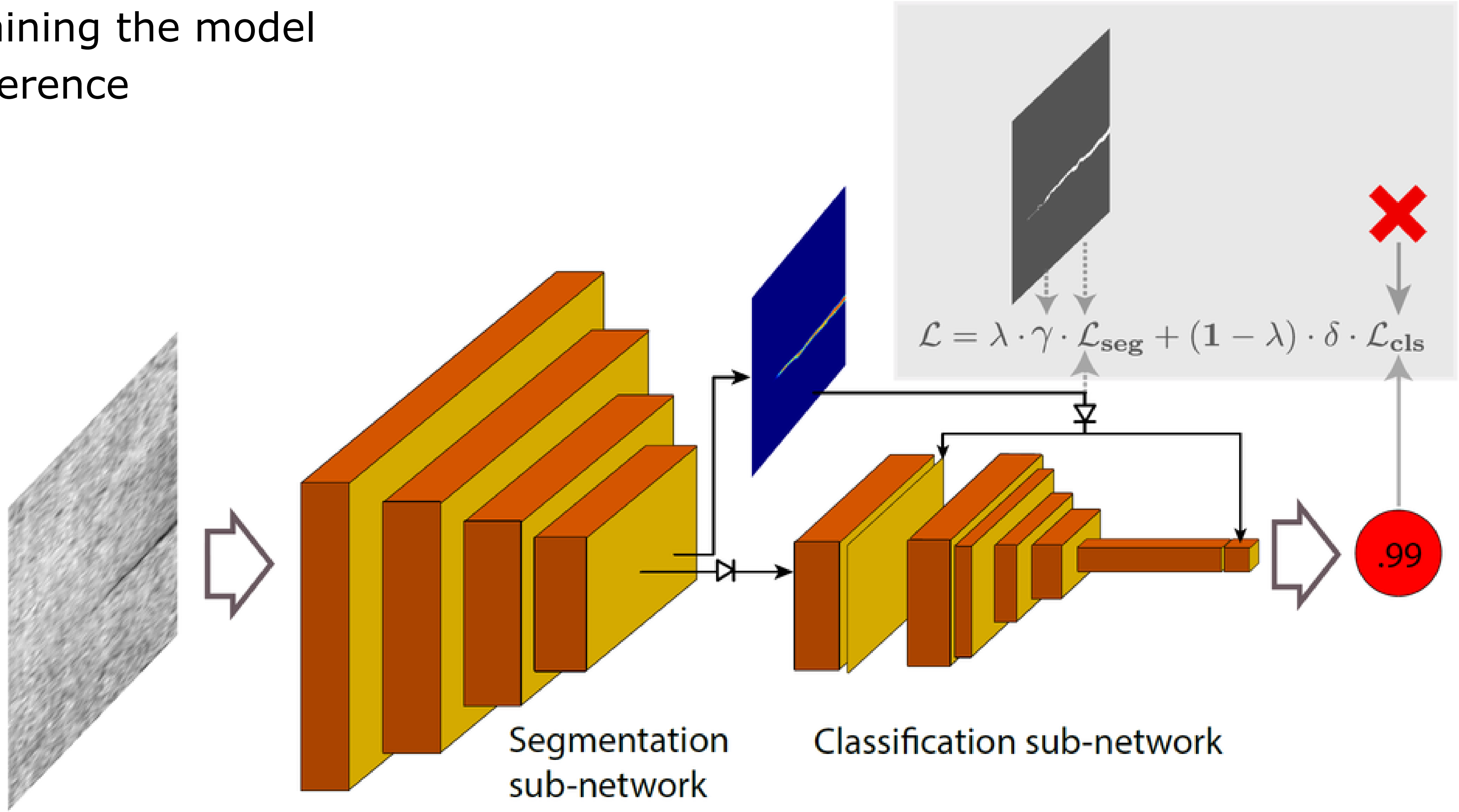
Hu et. al.

Typical solution

Korak 1: Zajem podatkov

Network architecture

- Training the model
- Inference



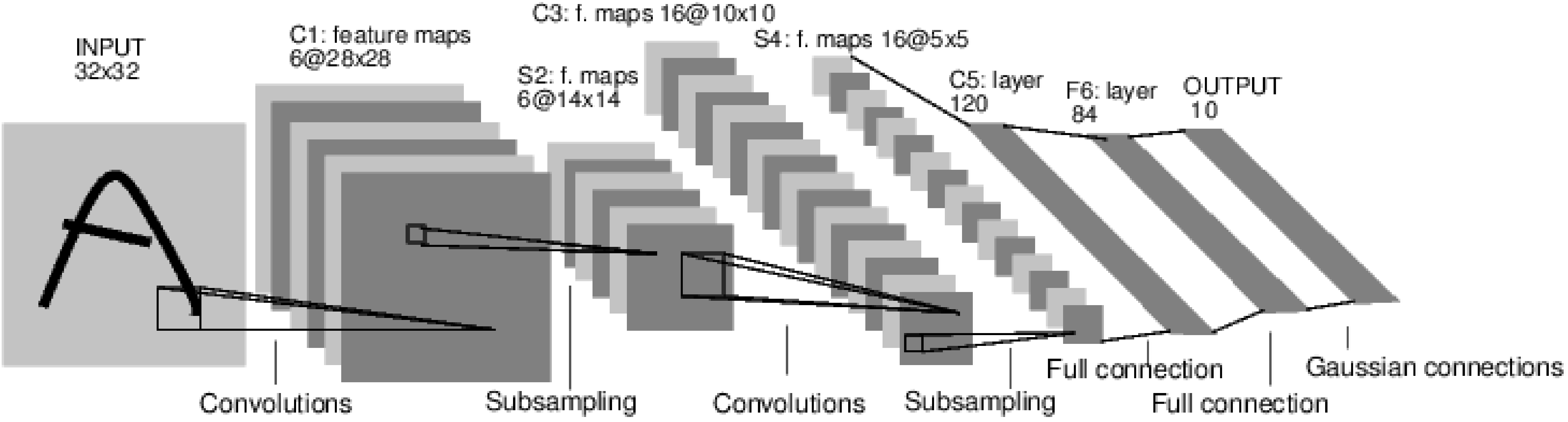
Example implementation in TensorFlow

```python
with variable_scope.variable_scope(scope, 'SegDecNet', [inputs]) as sc:
    end_points_collection = sc.original_name_scope + '_end_points'
    # Collect outputs for conv2d, max_pool2d
    with arg_scope([layers.conv2d, layers.fully_connected, layers_lib.max_pool2d, layers.batch_norm],
                   outputs_collections=end_points_collection):
        # Apply specific parameters to all conv2d layers (to use batch norm and relu - relu is by default)
        with arg_scope([layers.conv2d, layers.fully_connected],
                       weights_initializer= lambda shape,dtype=tf.float32, partition_info=None: tf.random_normal(shape, mean=0,stddev=0.01, dtype=dtype),
                       biases_initializer=None,
                       normalizer_fn=layers.batch_norm,
                       normalizer_params={'center': True,'scale': True, 'decay': self.BATCHNORM_MOVING_AVERAGE_DECAY,'epsilon': 0.001}):

            net = layers_lib.repeat(inputs, 2, layers.conv2d, 32, [5, 5], scope='conv1')
            net = layers_lib.max_pool2d(net, [2, 2], scope='pool1')
            net = layers_lib.repeat(net, 3, layers.conv2d, 64, [5, 5], scope='conv2')
            net = layers_lib.max_pool2d(net, [2, 2], scope='pool2')
            net = layers_lib.repeat(net, 4, layers.conv2d, 64, [5, 5], scope='conv3')
            net = layers_lib.max_pool2d(net, [2, 2], scope='pool3')
            net = layers.conv2d(net, 1024, [15, 15], padding='SAME', scope='conv4')
            net_prob_mat = layers.conv2d(net, 1, [1, 1], scope='conv5', activation_fn=None)

            with tf.name_scope('decision'):
                net_prob_mat = tf.nn.relu(net_prob_mat)
                decision_net = tf.concat([net, net_prob_mat],axis=3)
                decision_net = layers_lib.max_pool2d(decision_net, [2, 2], scope='decision/pool4')
                decision_net = layers.conv2d(decision_net, 8, [5, 5], padding='SAME', scope='decision/conv6')
                decision_net = layers_lib.max_pool2d(decision_net, [2, 2], scope='decision/pool5')
                decision_net = layers.conv2d(decision_net, 16, [5, 5], padding='SAME', scope='decision/conv7')
                decision_net = layers_lib.max_pool2d(decision_net, [2, 2], scope='decision/pool6')
                decision_net = layers.conv2d(decision_net, 32, [5, 5], scope='decision/conv8')

                with tf.name_scope('decision/global_avg_pool'):
                    avg_decision_net = keras.layers.GlobalAveragePooling2D()(decision_net)
                with tf.name_scope('decision/global_max_pool'):
                    max_decision_net = keras.layers.GlobalMaxPooling2D()(decision_net)
                with tf.name_scope('decision/global_avg_pool'):
                    avg_prob_net = keras.layers.GlobalAveragePooling2D()(net_prob_mat)
                with tf.name_scope('decision/global_max_pool'):
                    max_prob_net = keras.layers.GlobalMaxPooling2D()(net_prob_mat)

                # adding avg_prob_net and max_prob_net may not be needed, but it doesen't hurt
                decision_net = tf.concat([avg_decision_net, max_decision_net, avg_prob_net, max_prob_net], axis=1)
                decision_net = layers.fully_connected(decision_net, 1, scope='decision/FC9',normalizer_fn=None,
                                                      biases_initializer=tf.constant_initializer(0), activation_fn=None)
    return decision_net
```

Segmentation network

Classification network

LeNet-5



CONV
5x5

POOL
F=2, S=2

CONV
5x5

POOL
F=2, S=2

FC

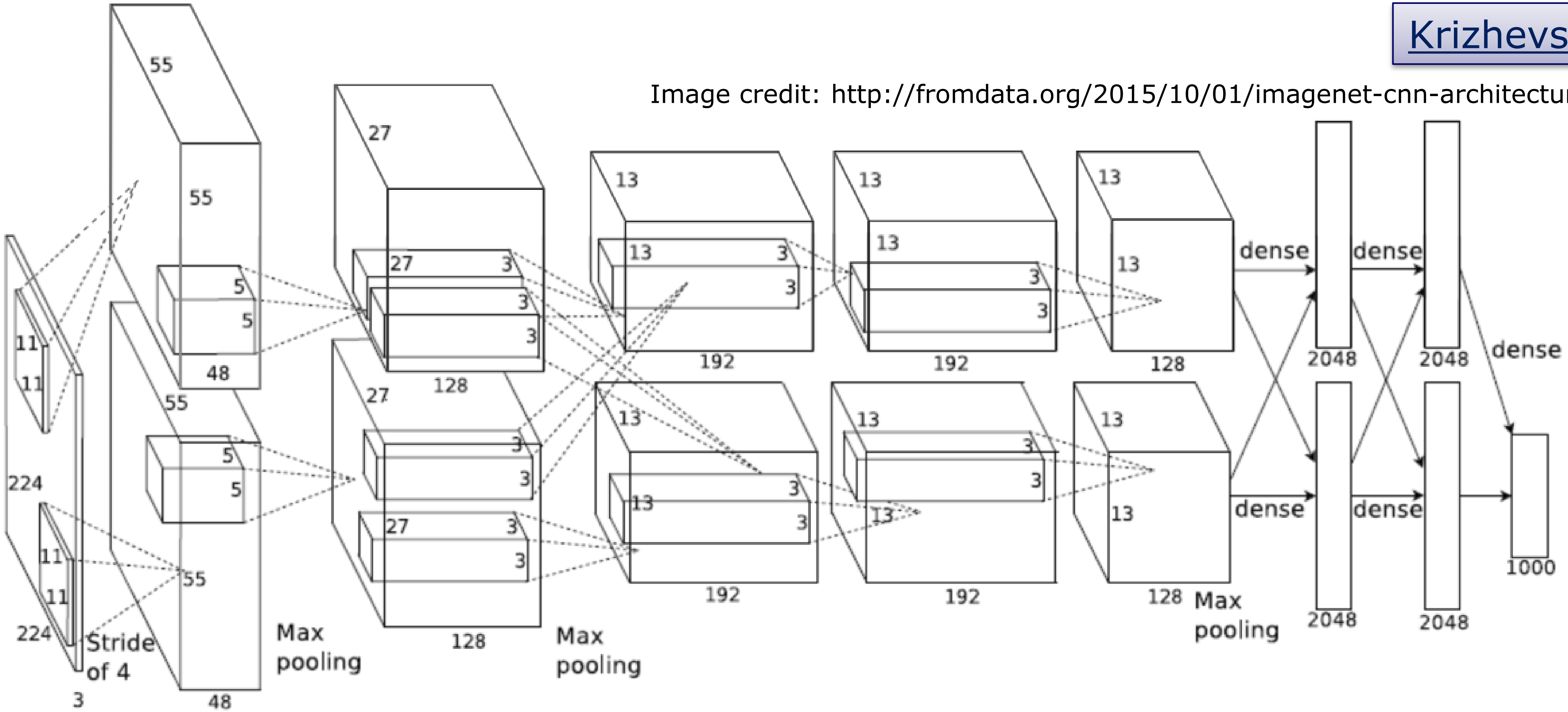
FC

LeCun et al., 1998

AlexNet

Krizhevsky, 2012

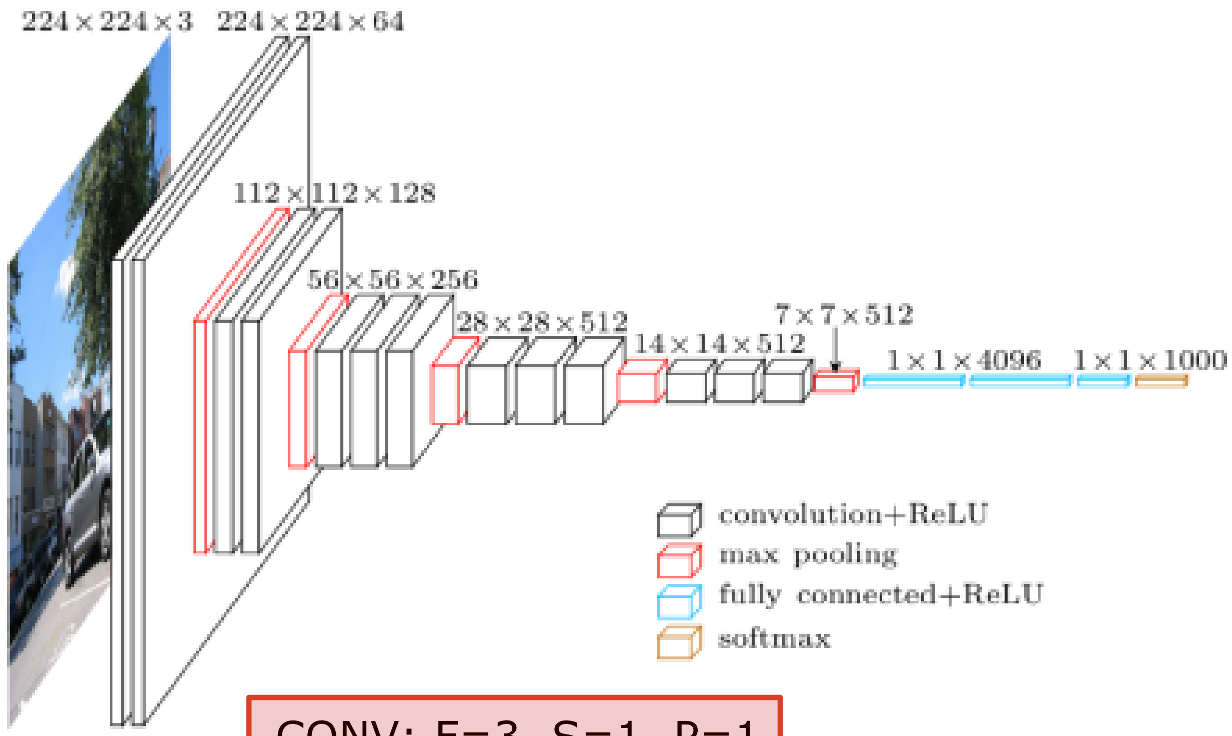
Image credit: http://fromdata.org/2015/10/01/imagenet-cnn-architecture-image/



CONV1	POOL	CONV2	POOL	CONV3	CONV4	CONV5	POOL	FC6	FC7	FC8
F=11	F=3	F=5	F=3	F=3	F=3	F=3	F=3	4096	4096	1000
S=4	S=2	S=1	S=2	S=1	S=1	S=1	S=2			
		P=2		P=1	P=1	P=1				

- ReLU, data augmentation, Dropout, Momentum, L2 regularisation

VGG



CONV: F=3, S=1, P=1
POOL: F=2, S=2

- Classical CNN backbone shape
- VGG16, VGG19

[Simonyan & Zisserman, 2014]

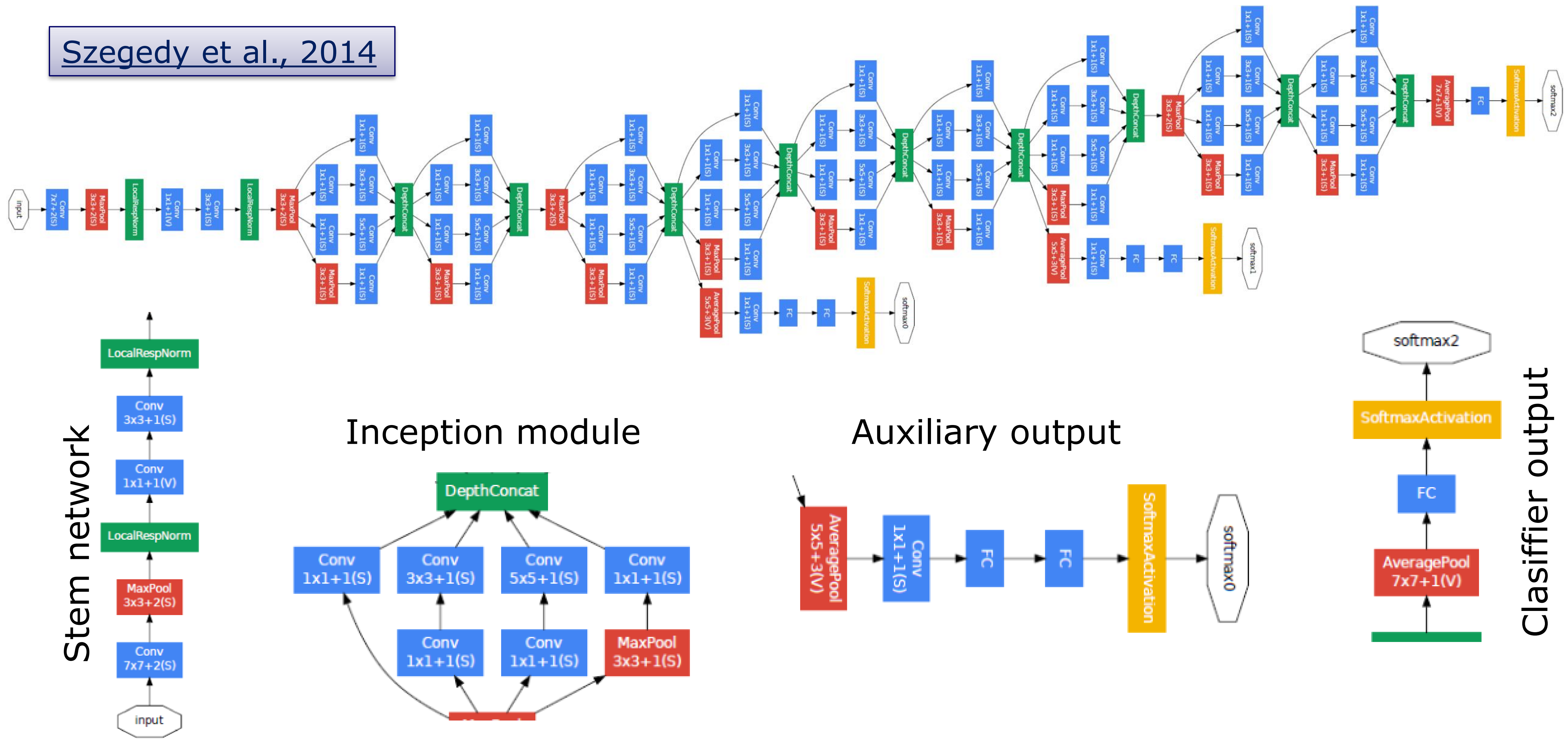
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

GoogLeNet / Inception

Szegedy et al., 2014



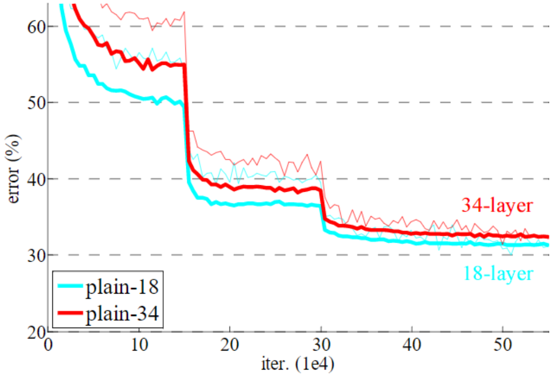
ResNet

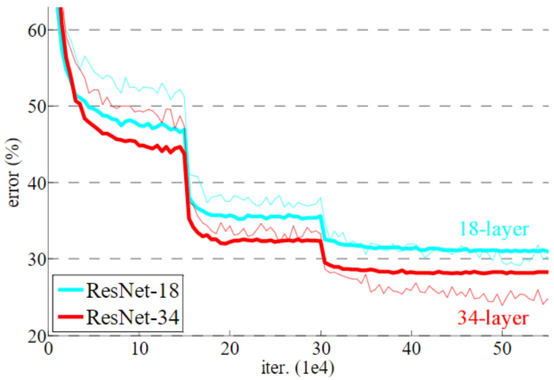
- Going deeper!
- Plain deep networks do not work

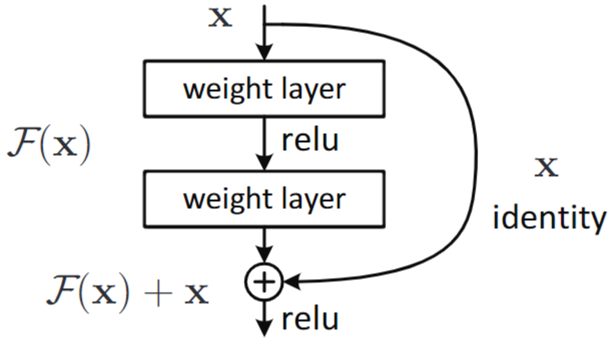
- Shortcut connections!
  - Figth vanishing gradient problem
- Learn residual functions

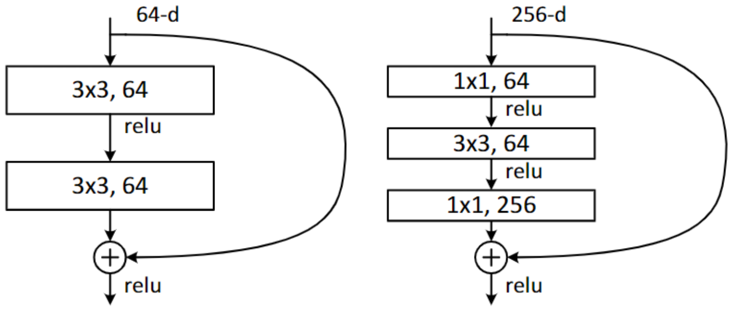
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}$$

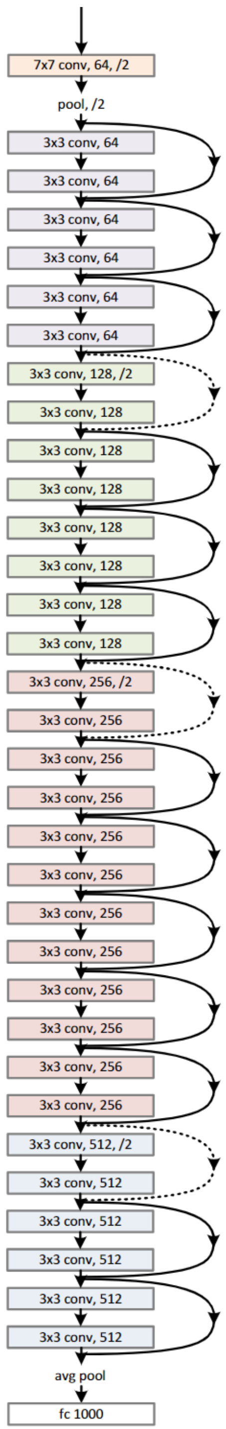
- Bottleneck building blocks
- Very deep networks:
  - 152, 101, 50, 34, 18

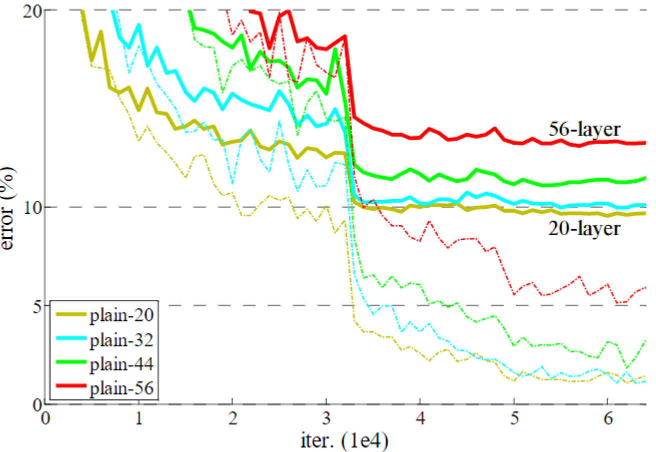


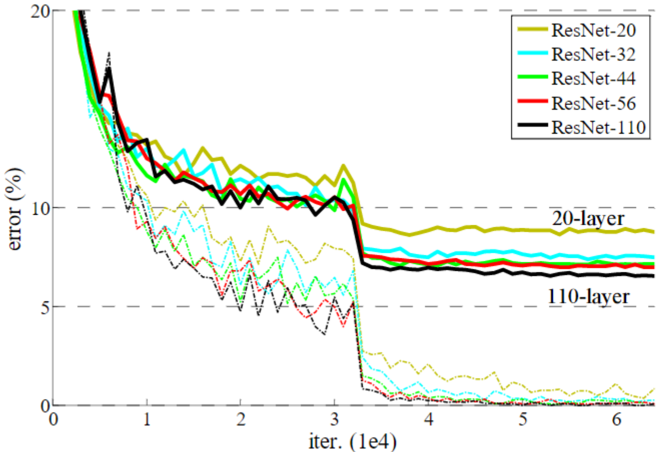


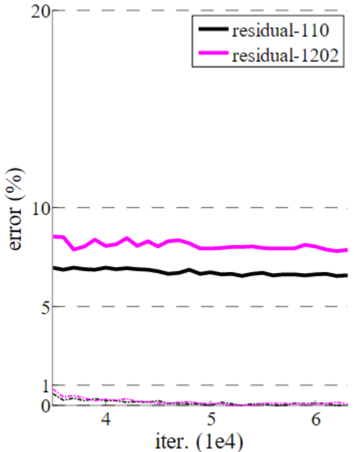










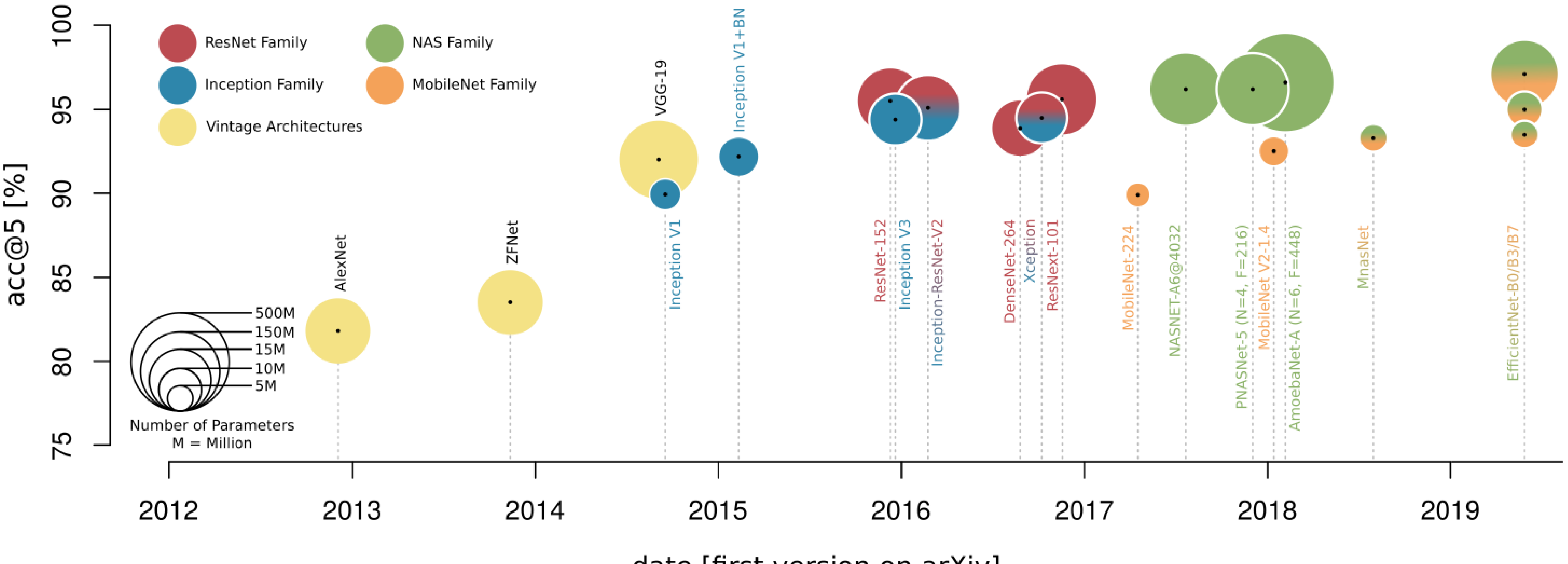


	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

He et al., 2015

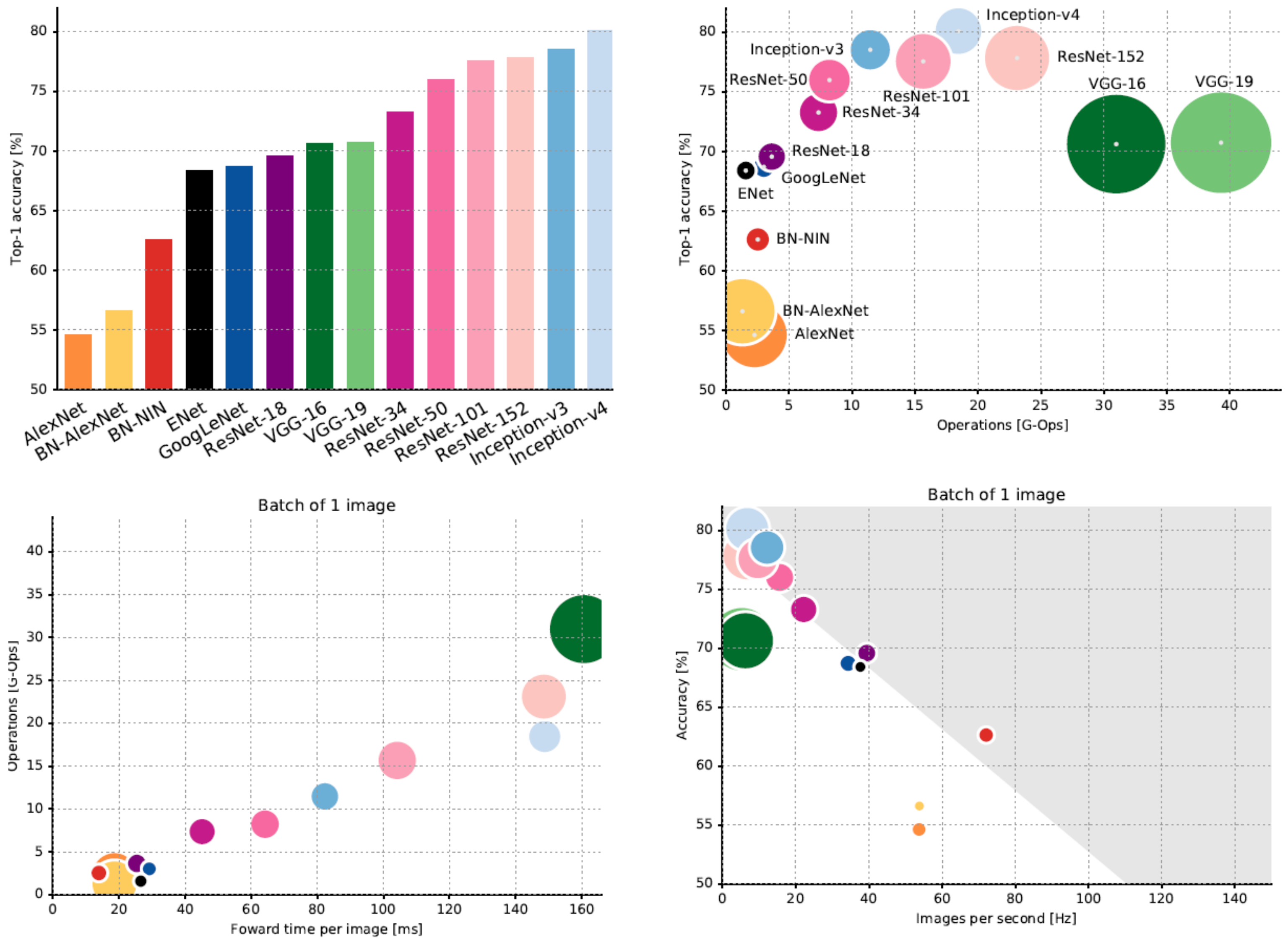
Architectures overview

- Date of publication, main type



[Hoeser and Kuenzer, 2020]

Analysis of DNN models

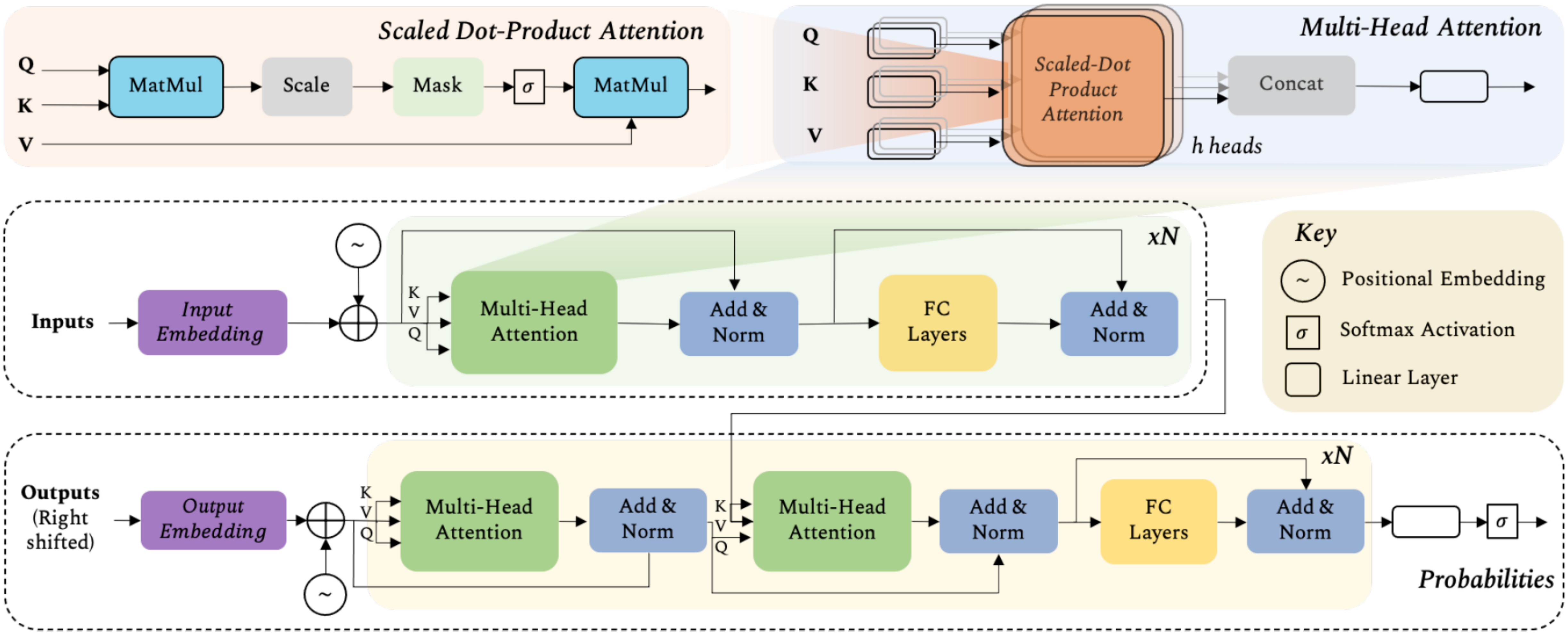


[Canziani et al., 2017]

Pretrained models

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
mobilenet_v3_large = models.mobilenet_v3_large(pretrained=True)
mobilenet_v3_small = models.mobilenet_v3_small(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
```

Transformers

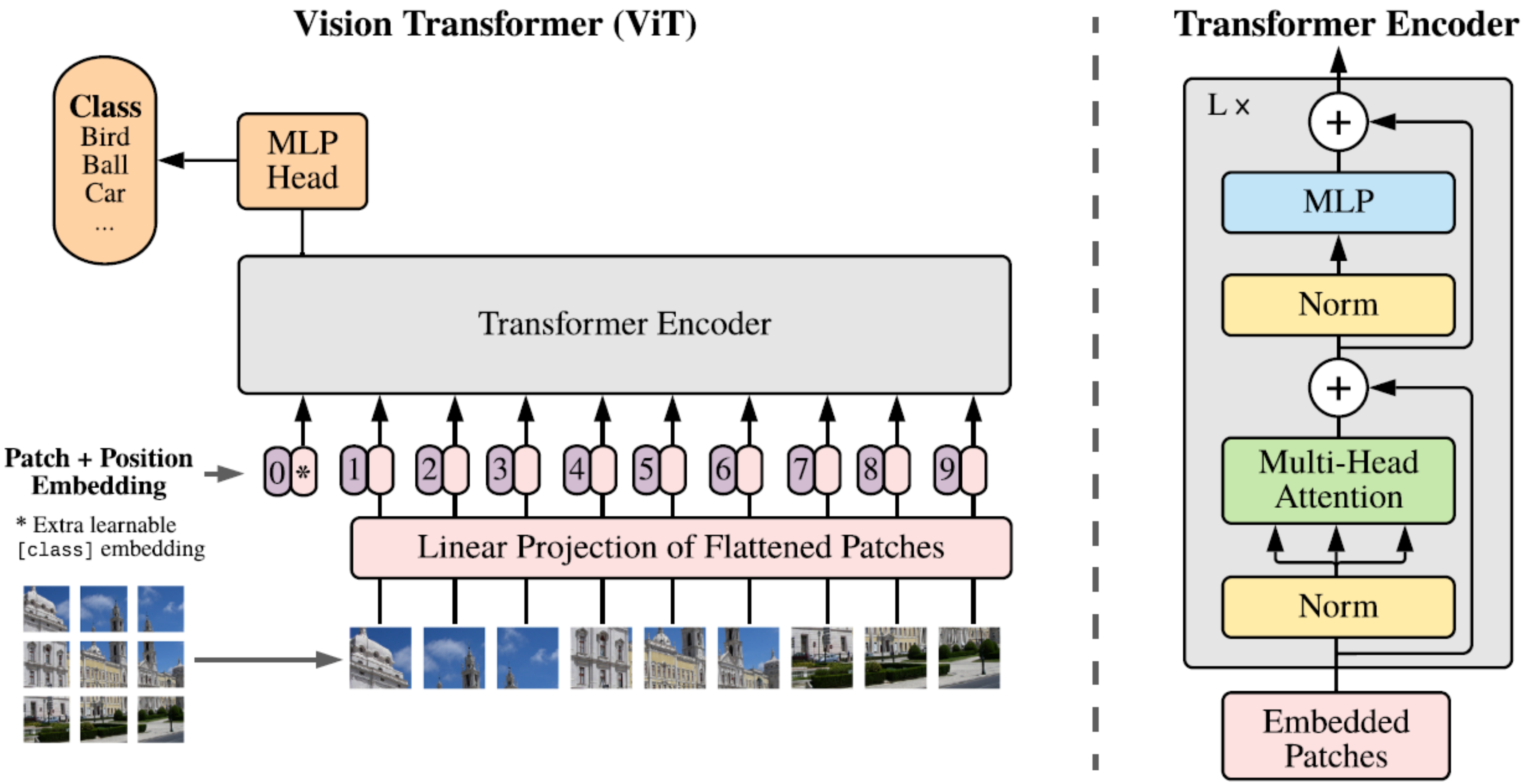


[Vaswani et.al, NIPS 2017] *[Khan et.al, 2021]*

ViT - Vision Transformer

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\cdots;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}$$
$$\mathbf{z'}_\ell = \mathrm{MSA}(\mathrm{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1},$$
$$\mathbf{z}_\ell = \mathrm{MLP}(\mathrm{LN}(\mathbf{z'}_\ell)) + \mathbf{z'}_\ell,$$
$$\mathbf{y} = \mathrm{LN}(\mathbf{z}_L^0)$$

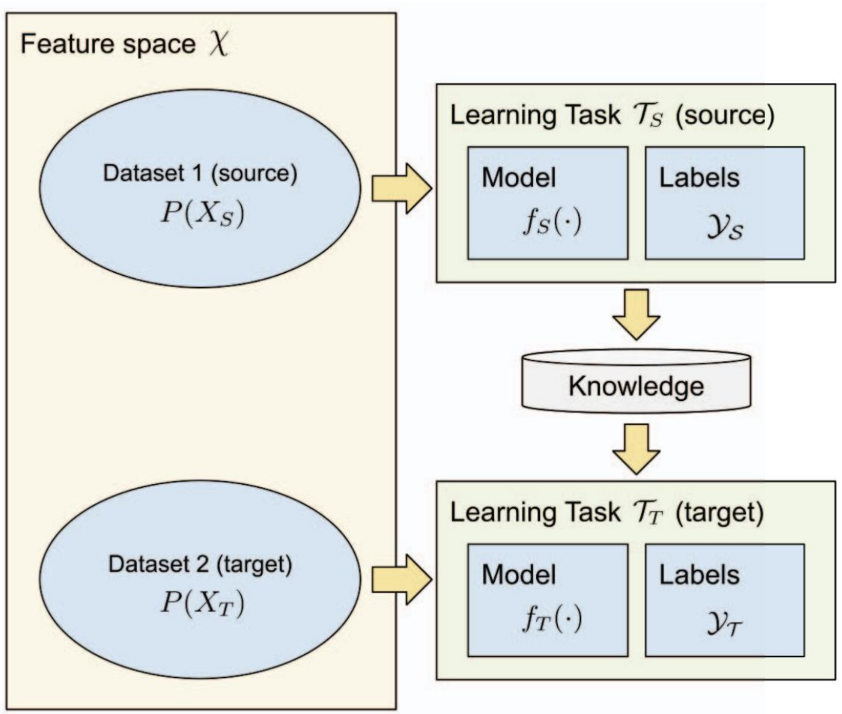
- AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

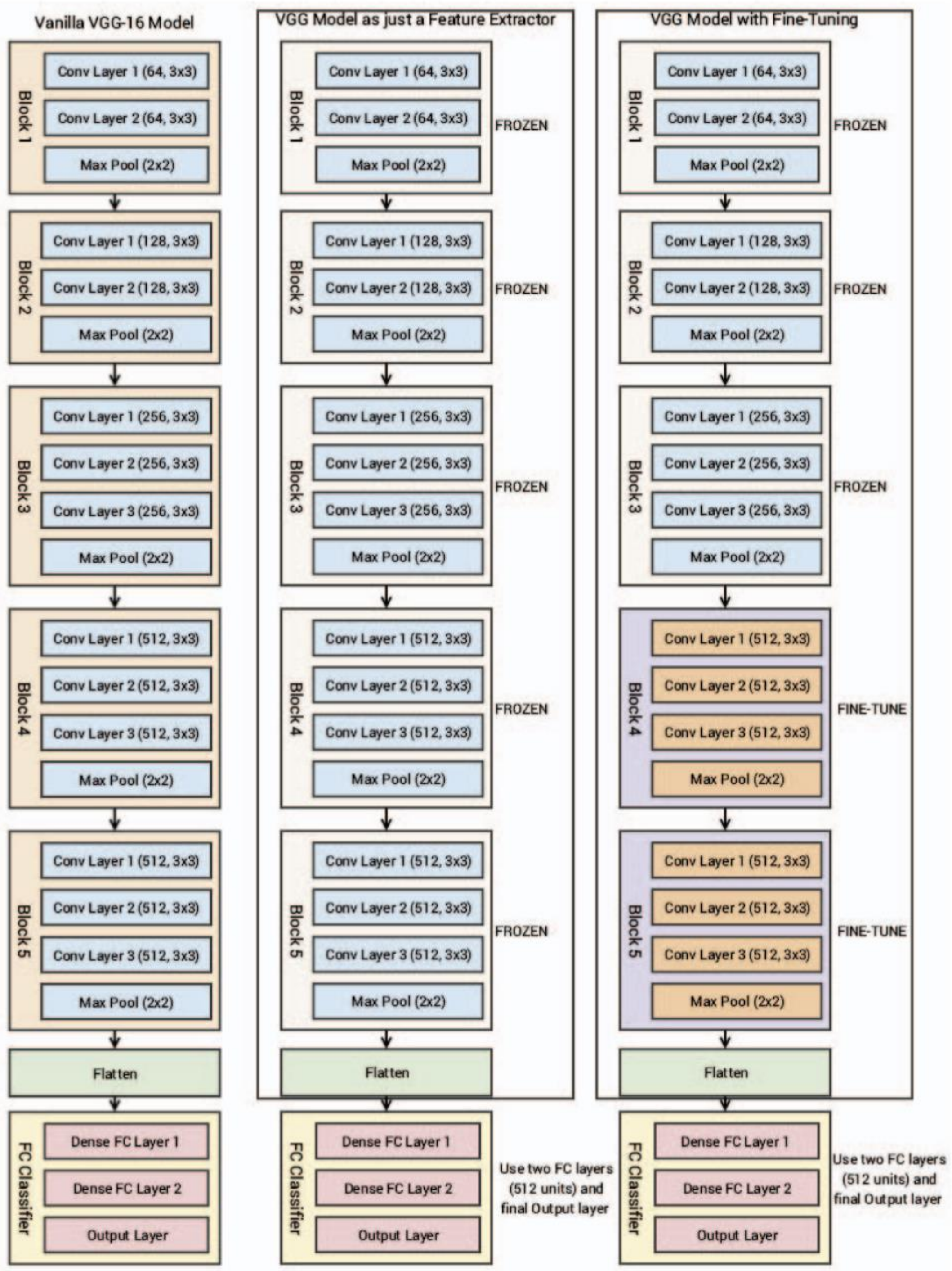


[Dosovitskiy et.al, Google, 2020, ICLR 2021]

Transfer learning

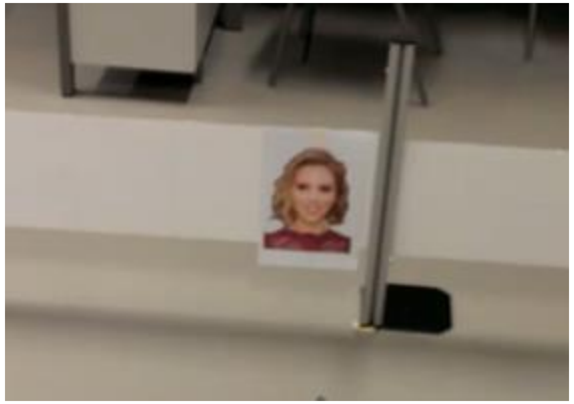
- Train on a large related dataset
- Fine-tune on the target dataset
- Heavily used





[Ribani & Marengoni 2019](#)

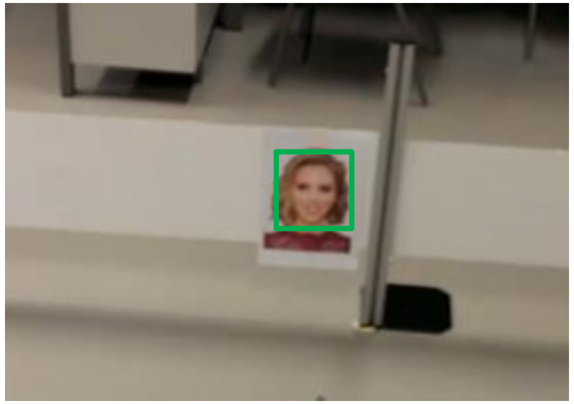
Two stage object detection and recognition

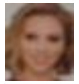


very fast
efficient

Face detection

HOG+SVM
AdaBoost
SSD





Face recognition

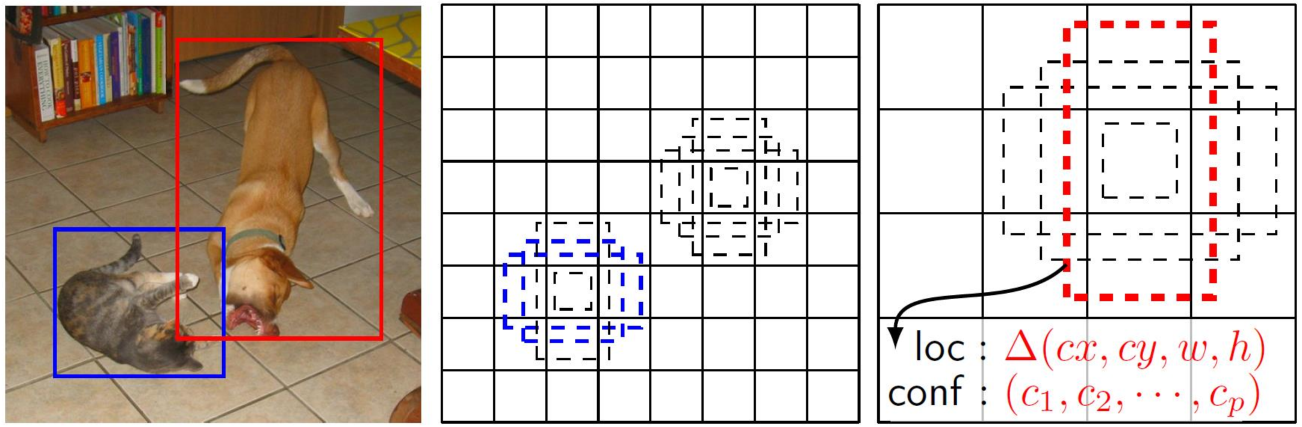
CNN
PCA/LDA

could be slower
computationally more complex

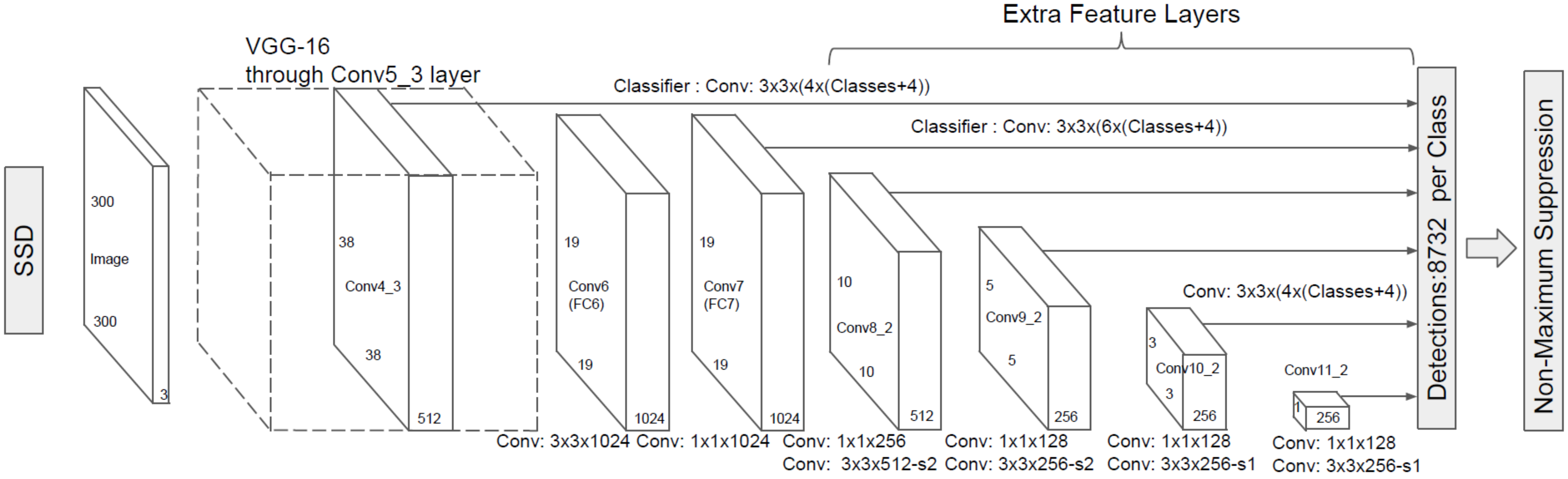
„Scarlet“

SSD: Single Shot MultiBox Detector

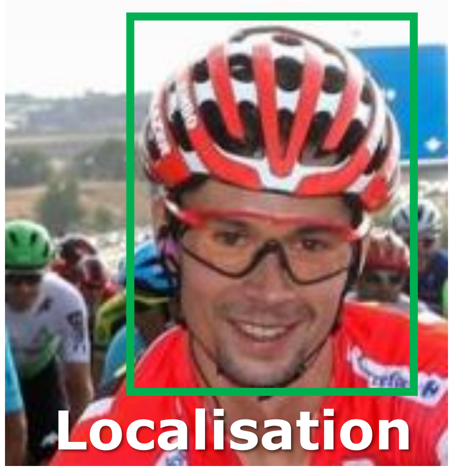
- Multi-scale feature maps for detection
- Convolutional predictors for detection
- Default boxes and aspect ratios
- Real time operation

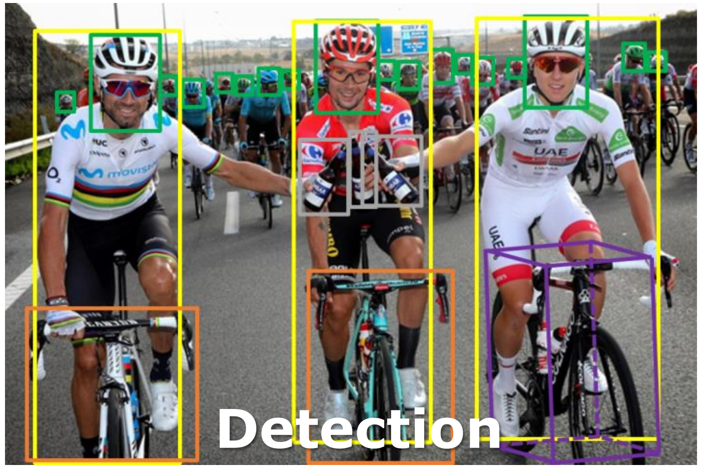


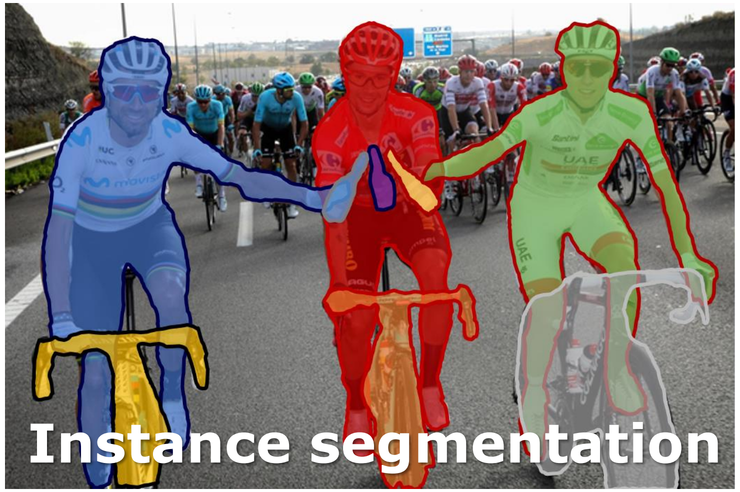
[Liu et al., ECCV 2016]

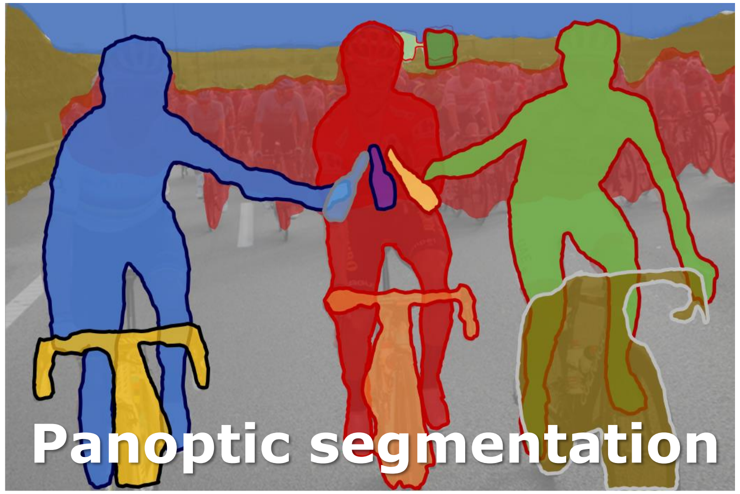


Main computer vision tasks

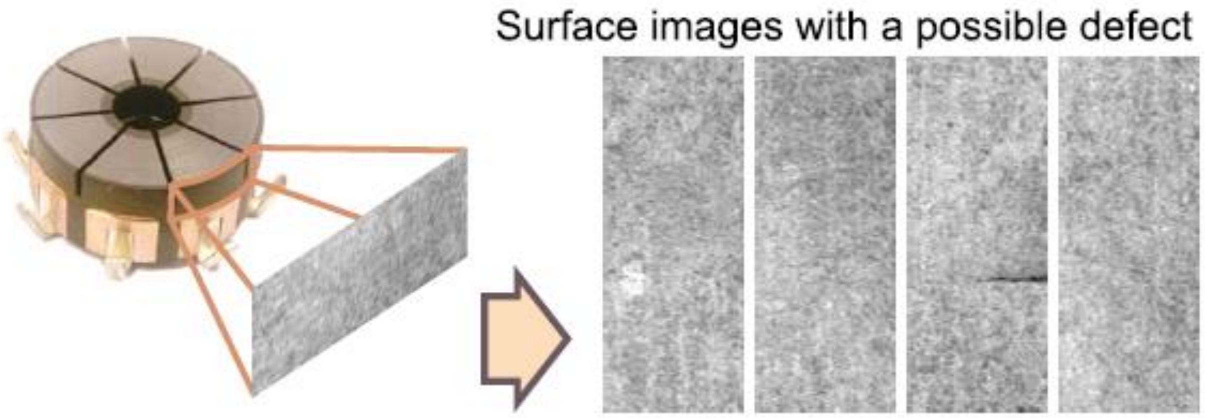


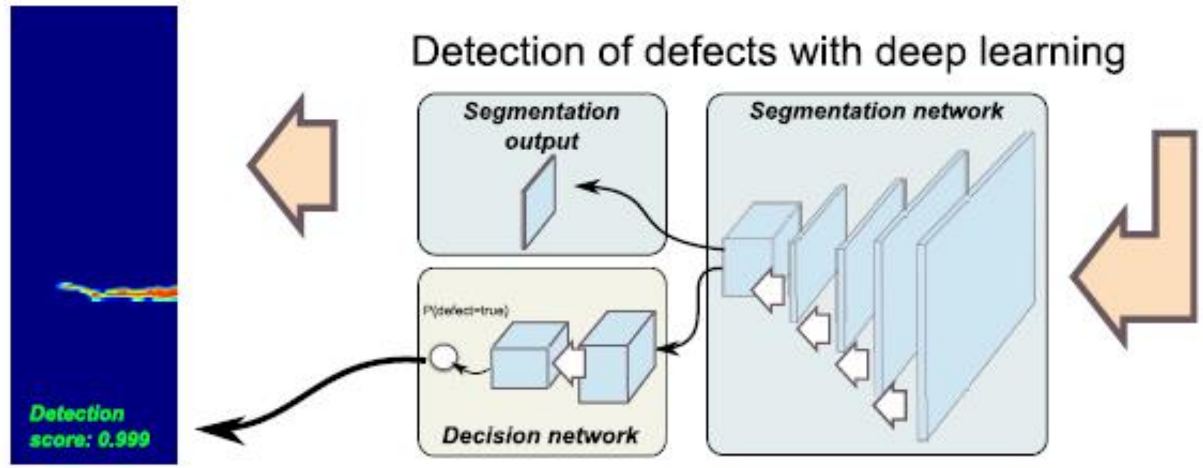


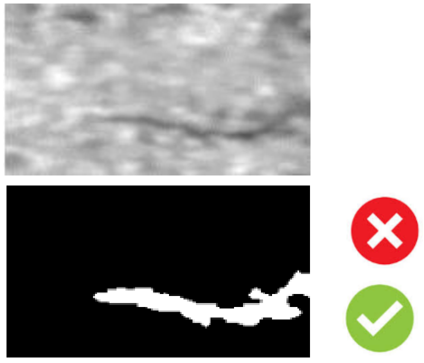


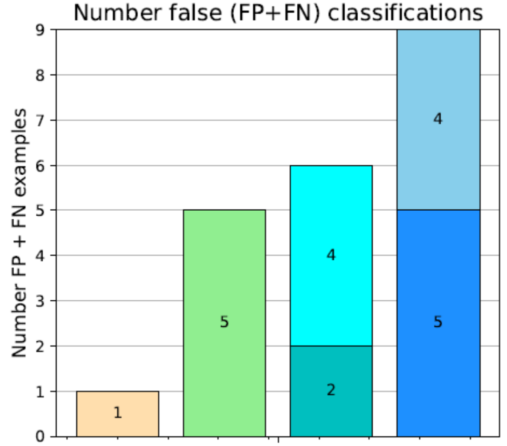


Surface-defect detection

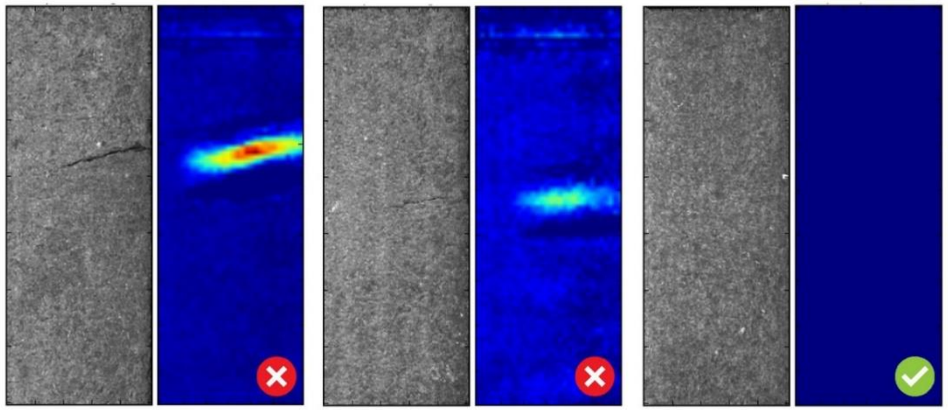




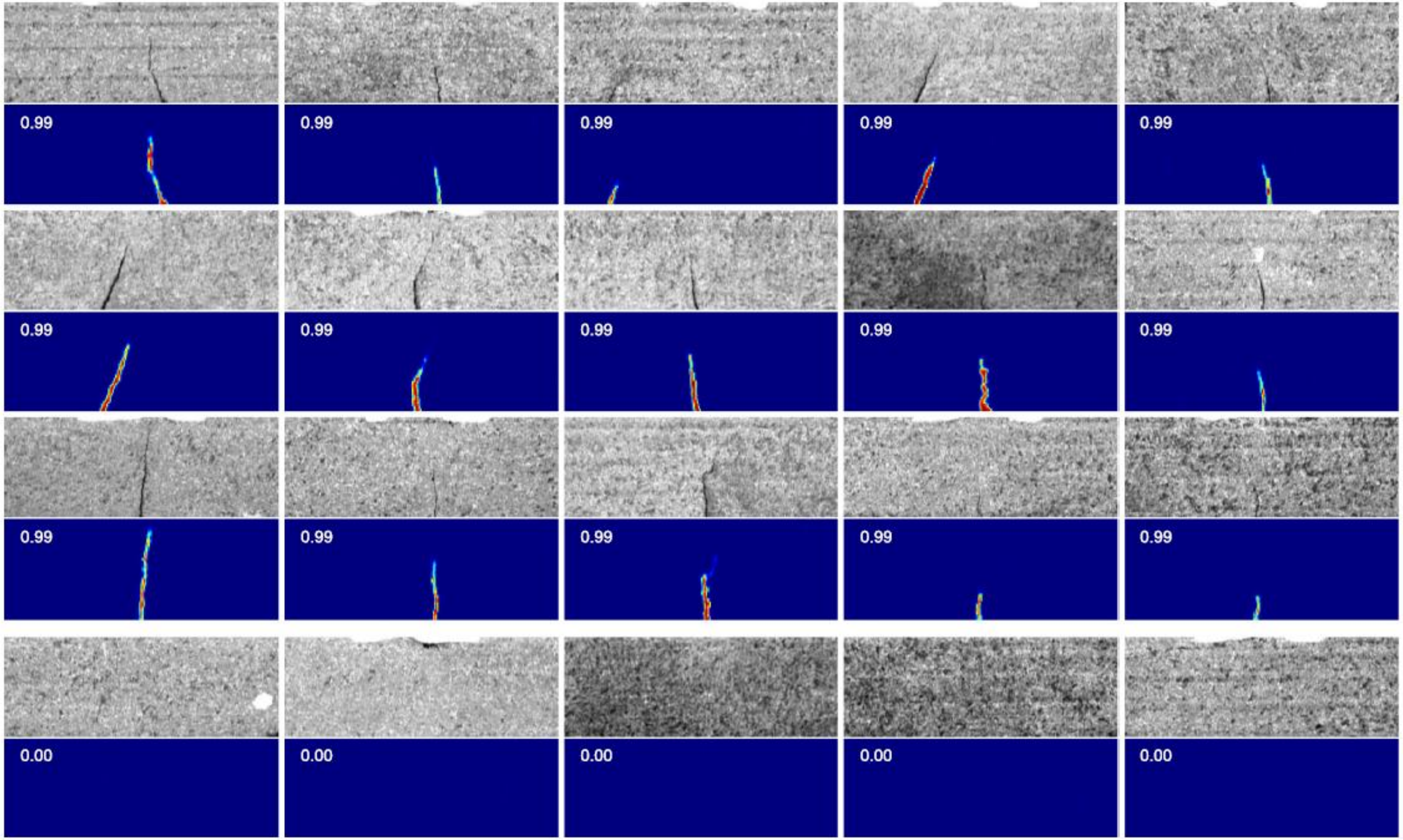




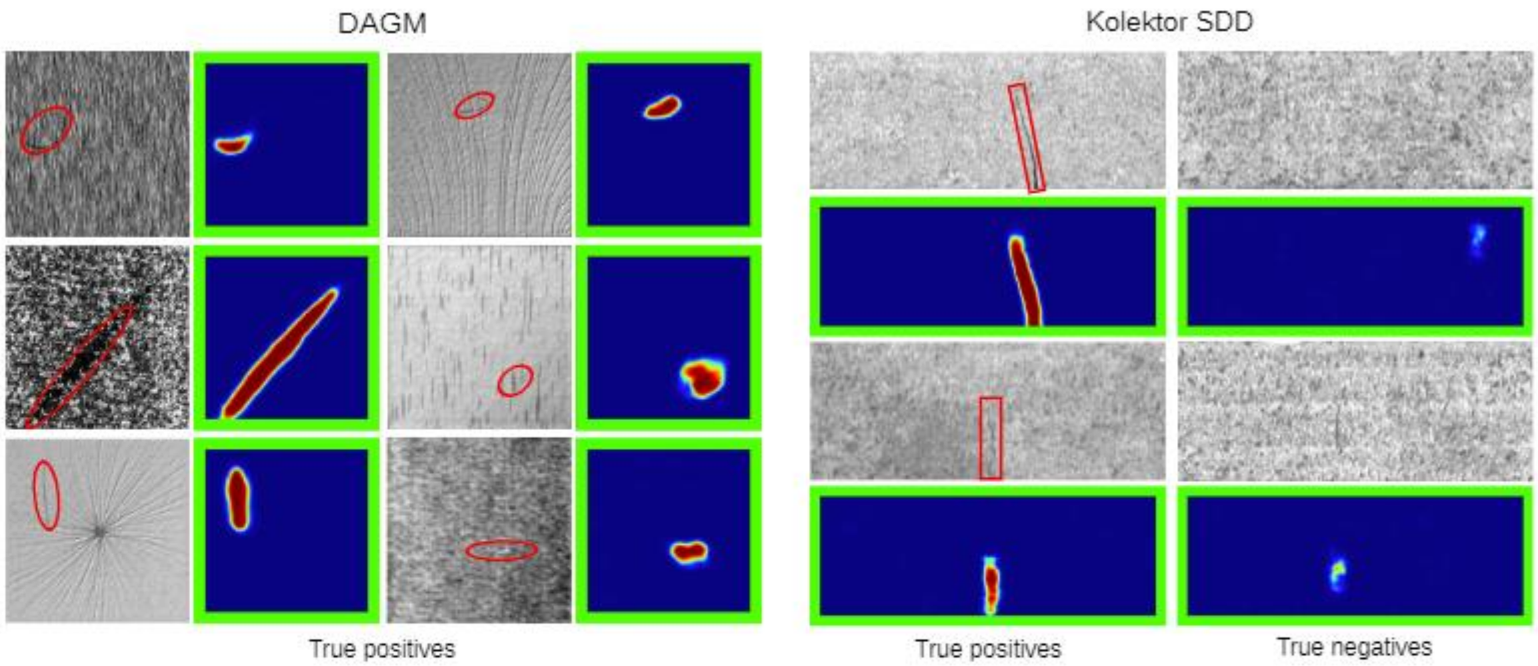
Segmantation-based data-driven surface-defect detection

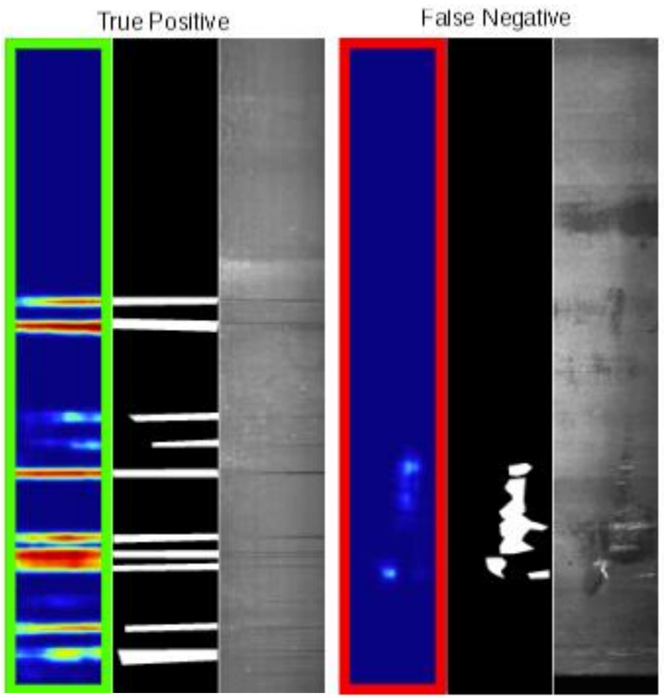


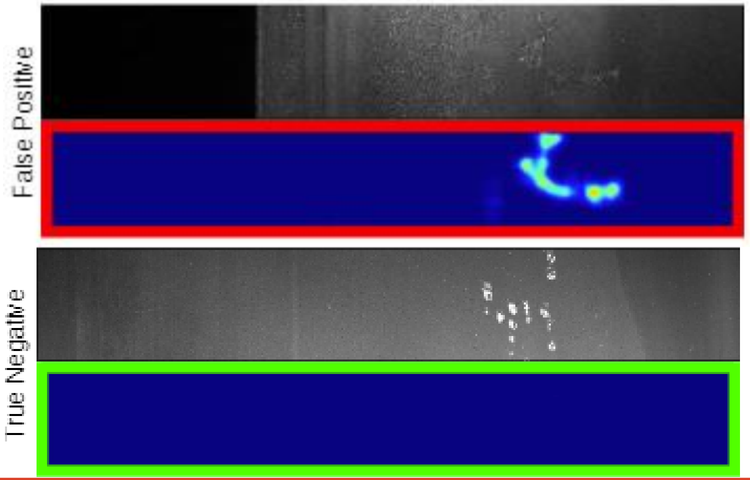
Surface-defect detection

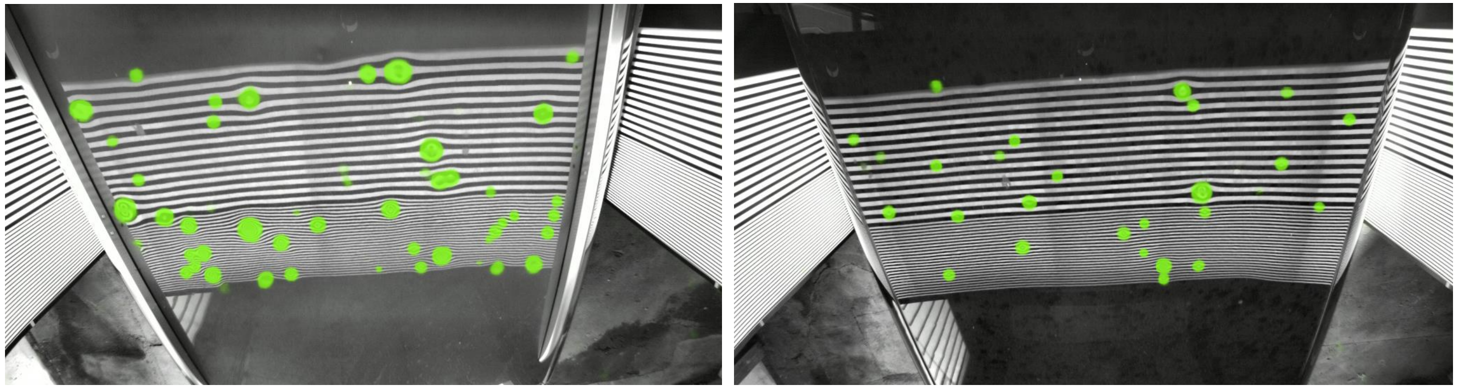


Surface-defect detection

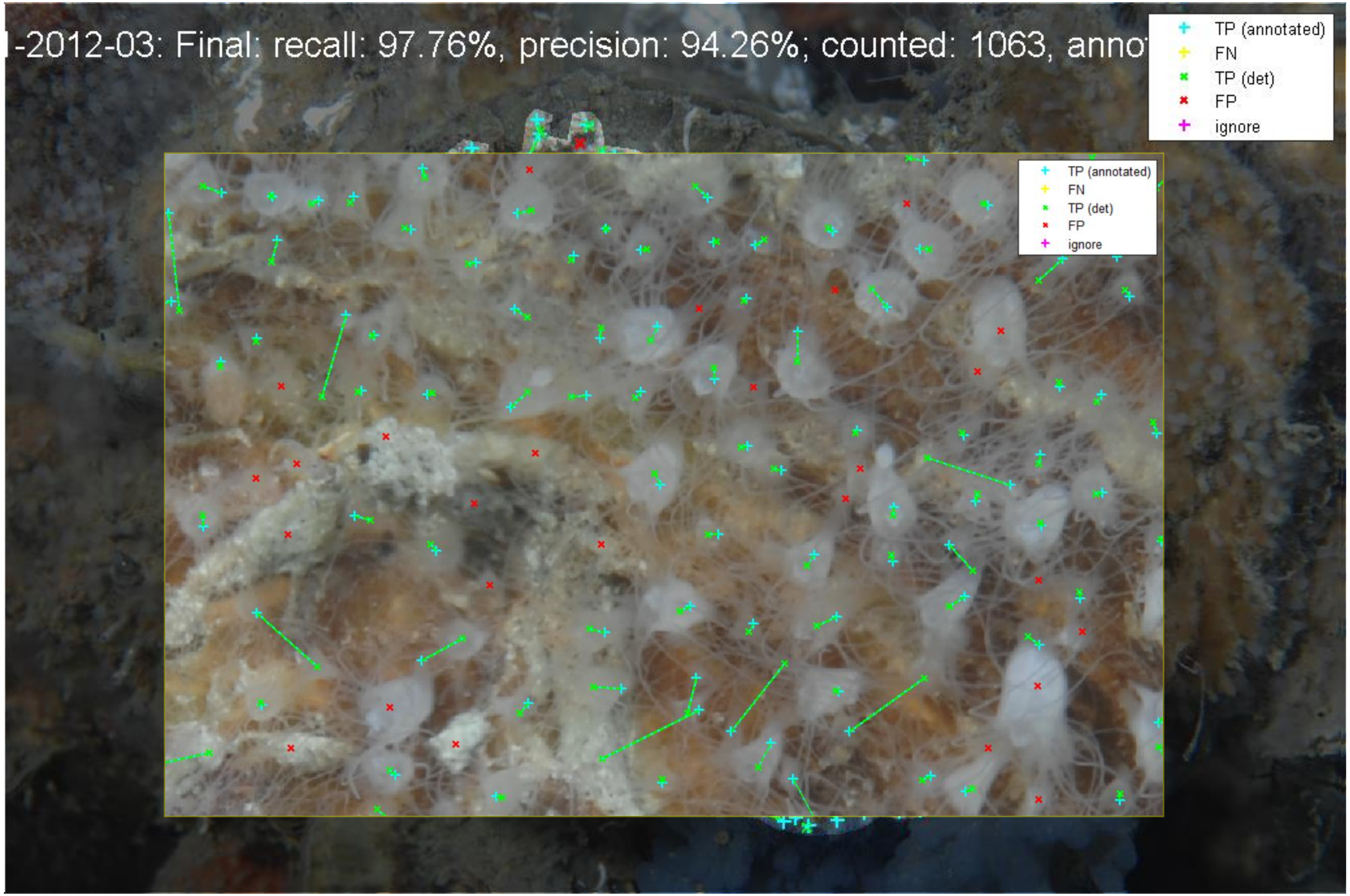




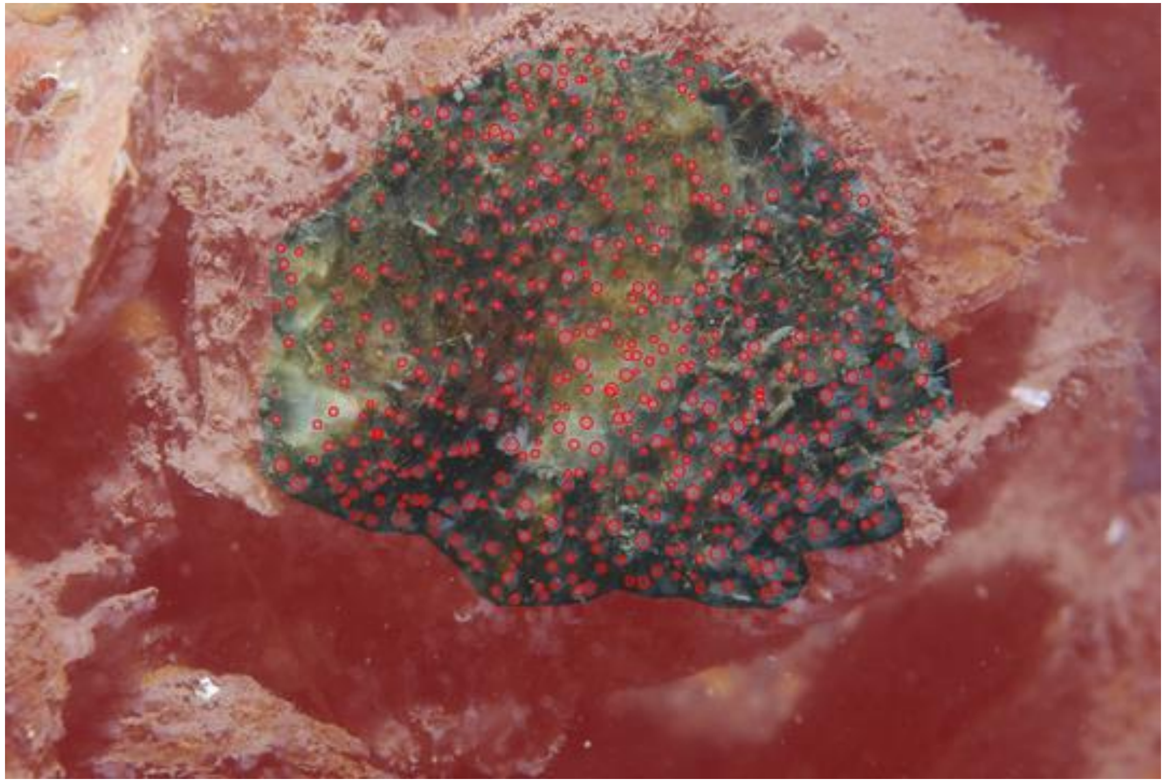


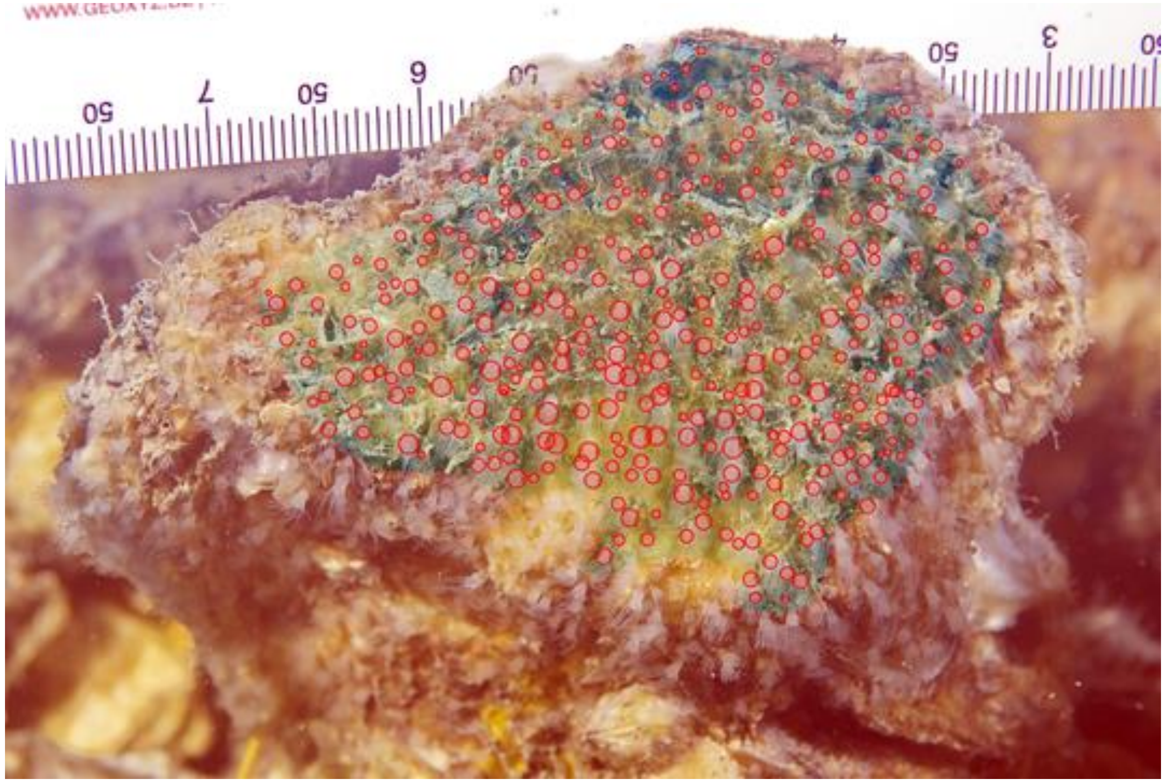


Polyp counting

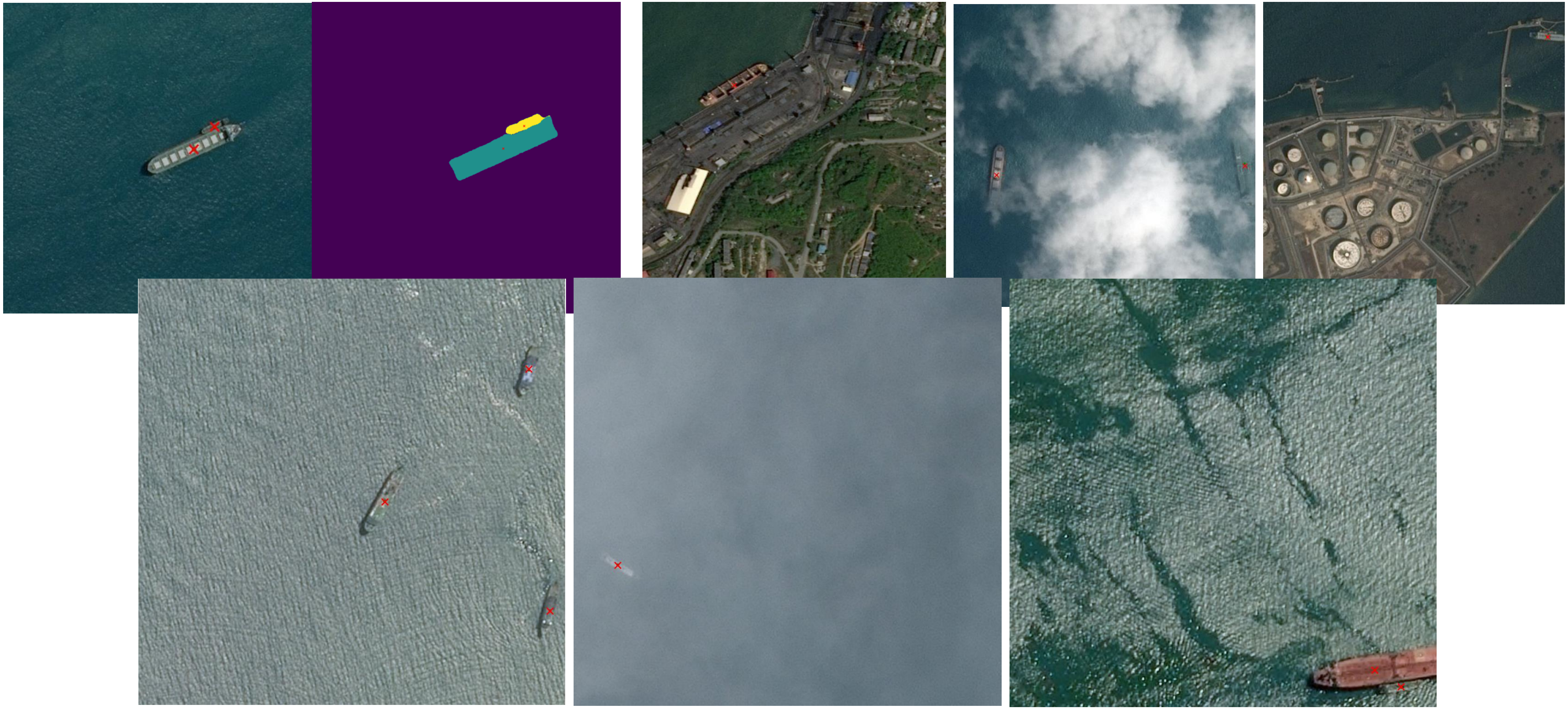


Polyp counting



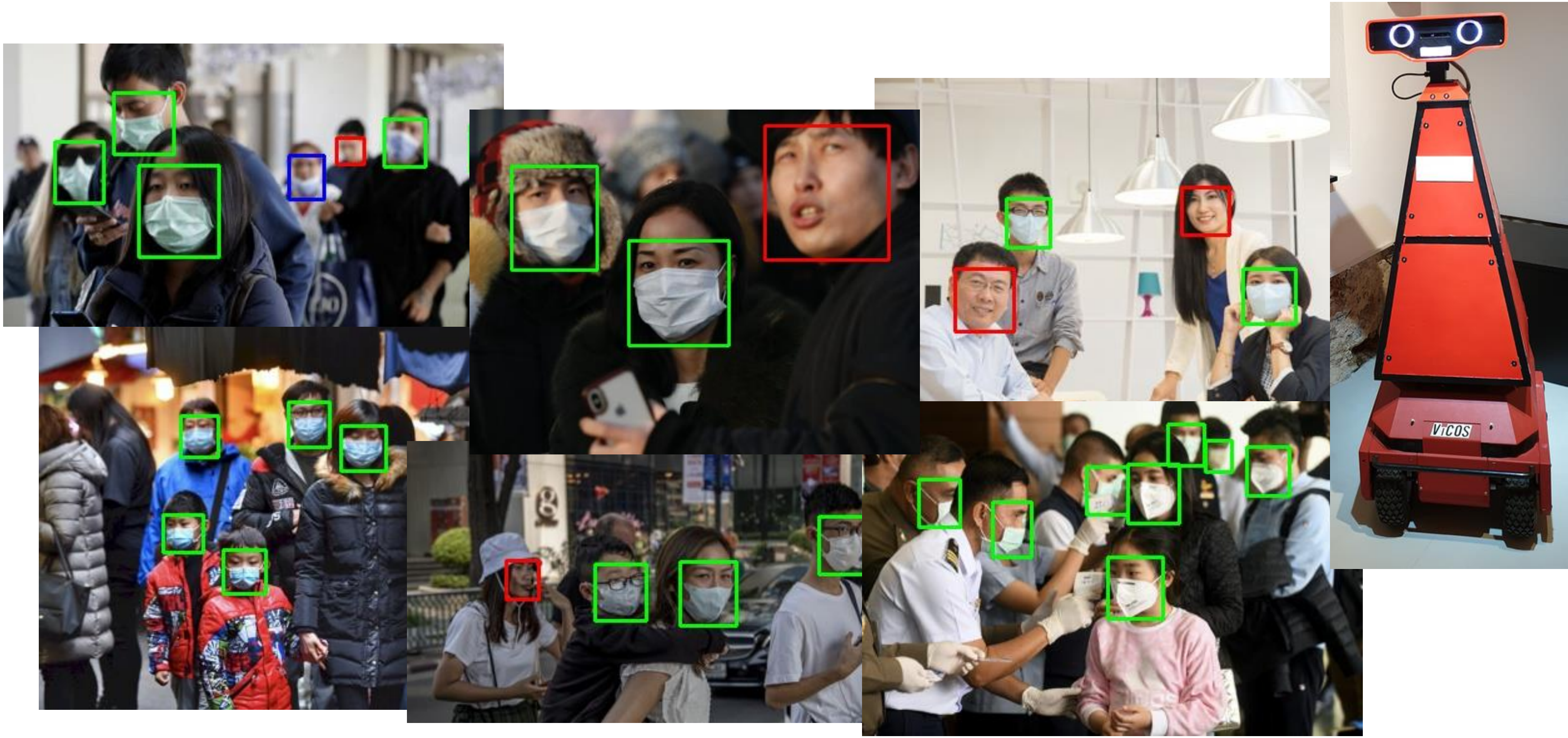


Ship detection



Face detection

Mask-wearing detection

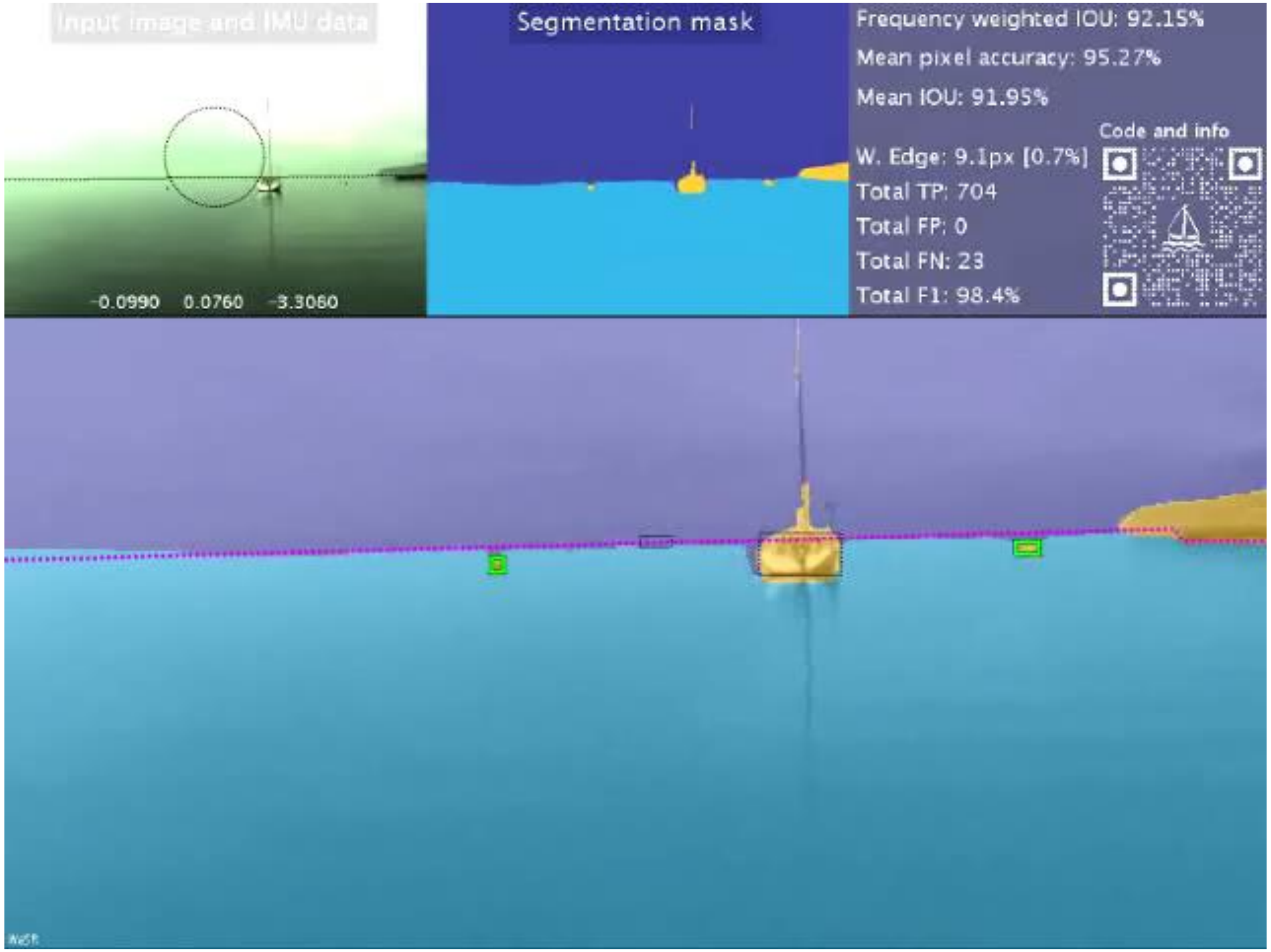


Obstacle detection on autonomous boat

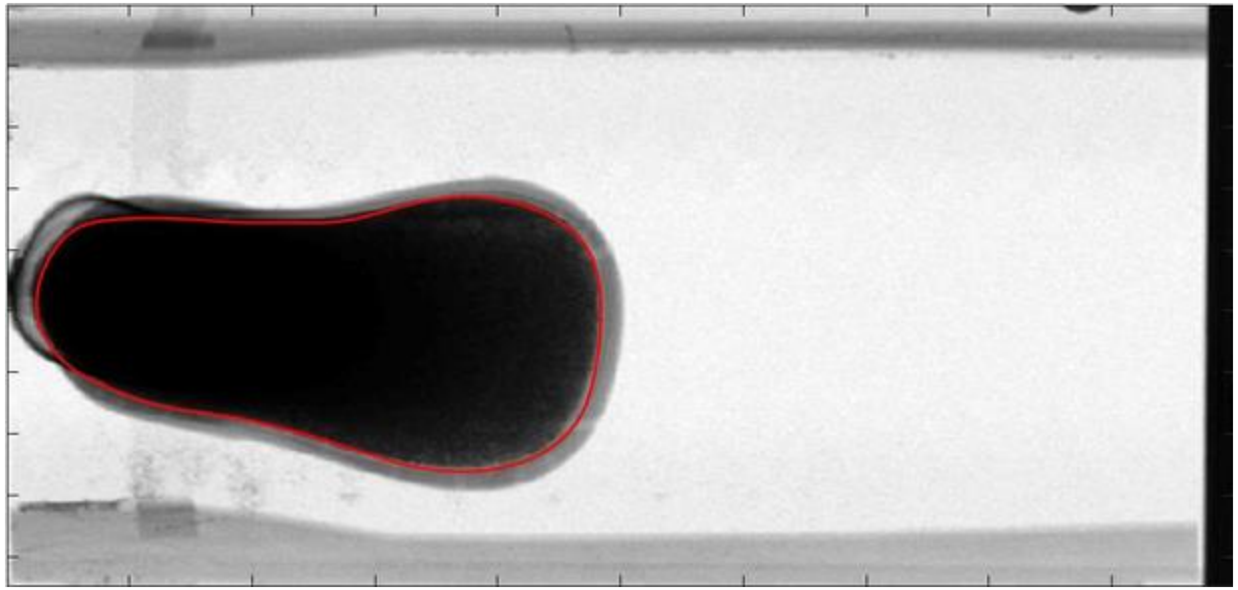
USV equipped with different sensors:

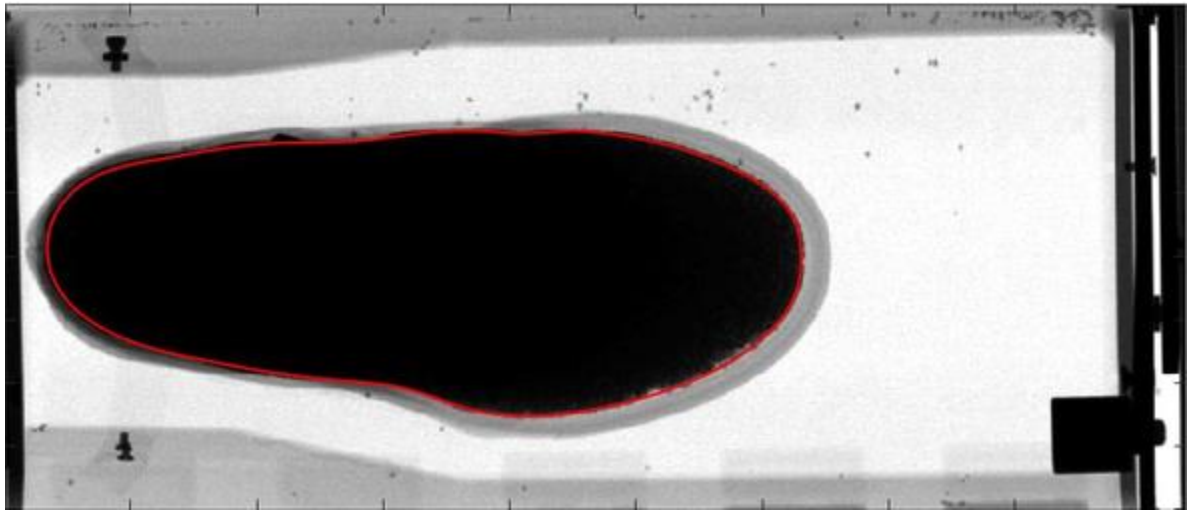
- stereo camera
- IMU
- GPS
- compass

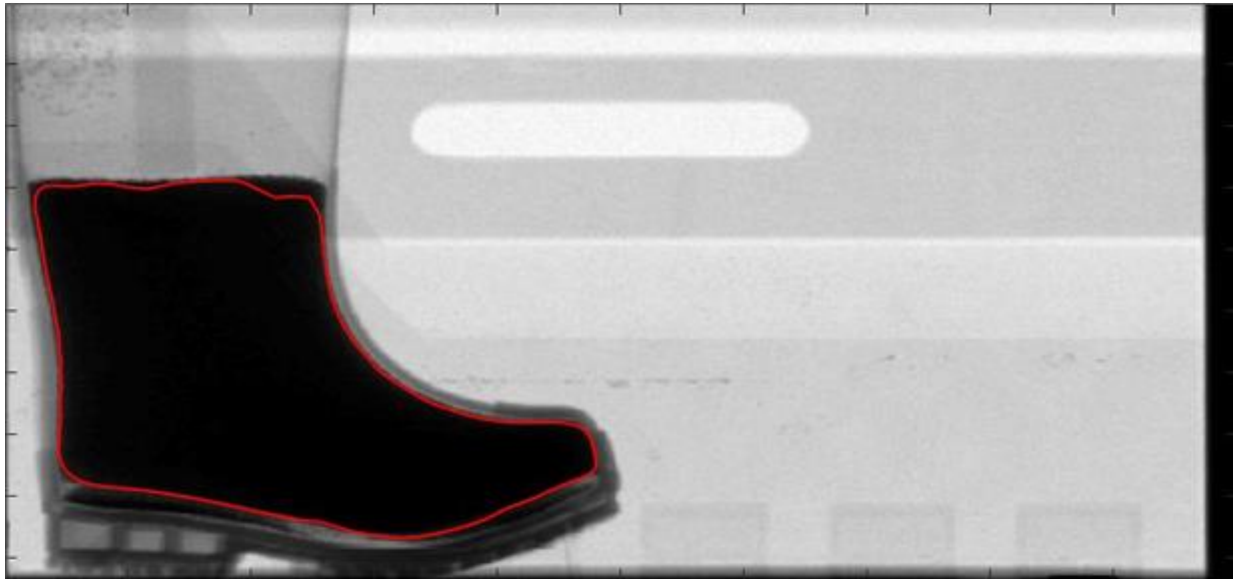
Segmentation based on RGB + IMU

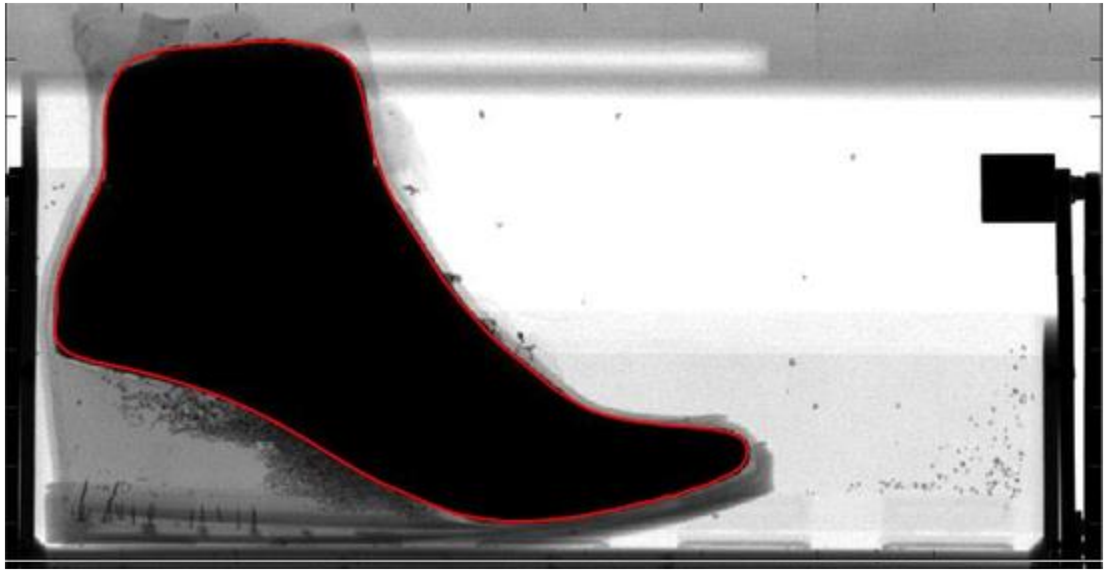


Semantic edge detection

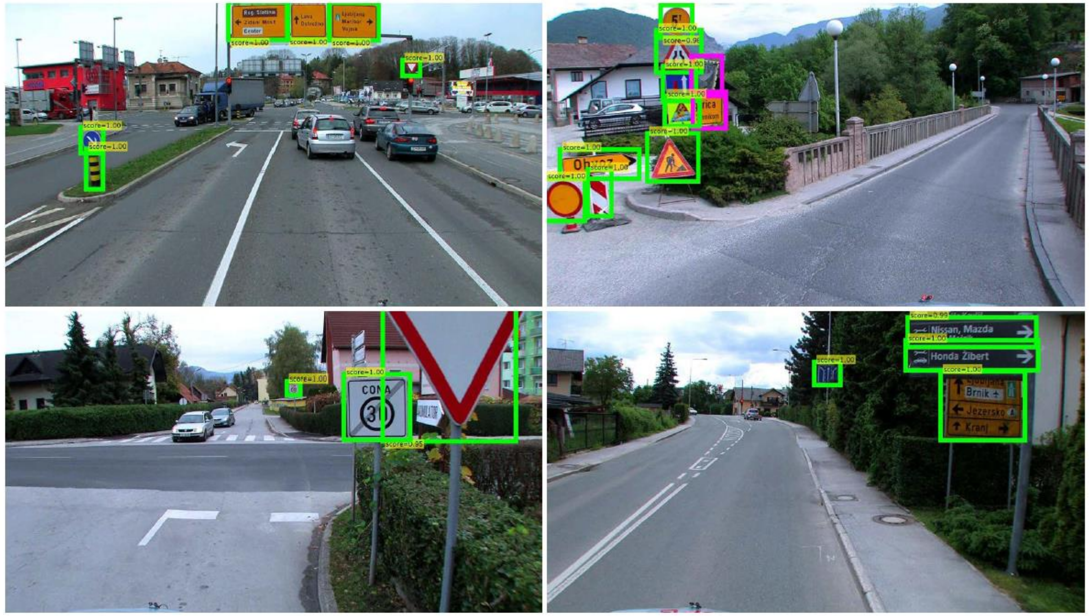


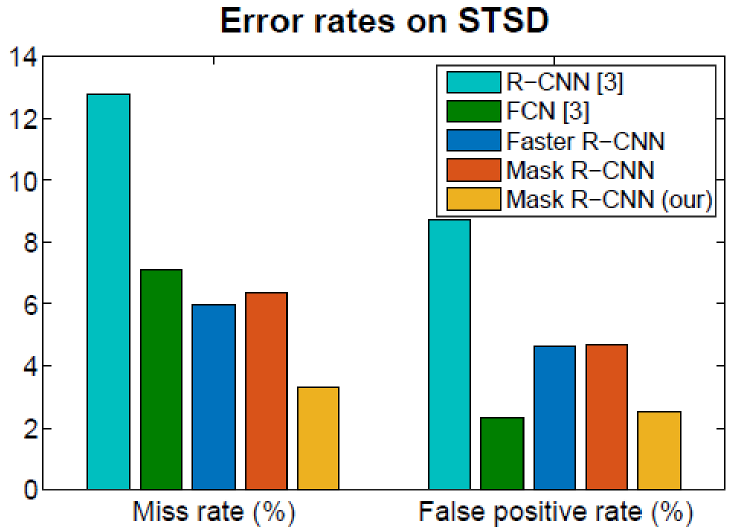






Object (traffic sign) detection

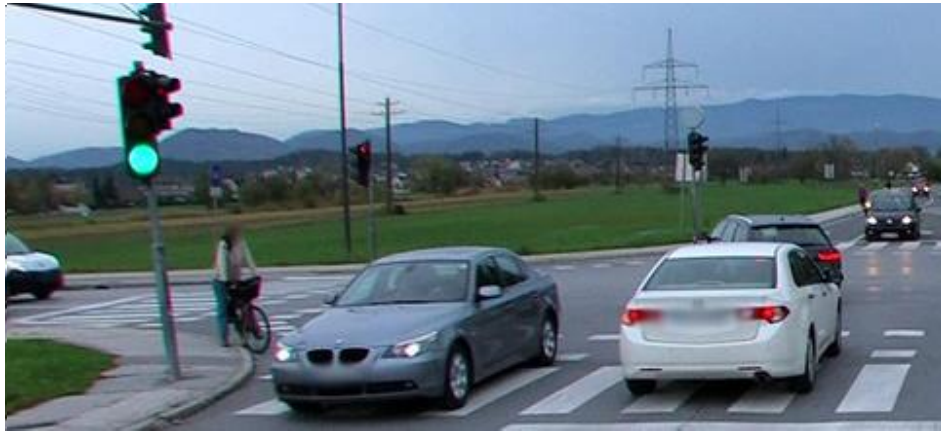




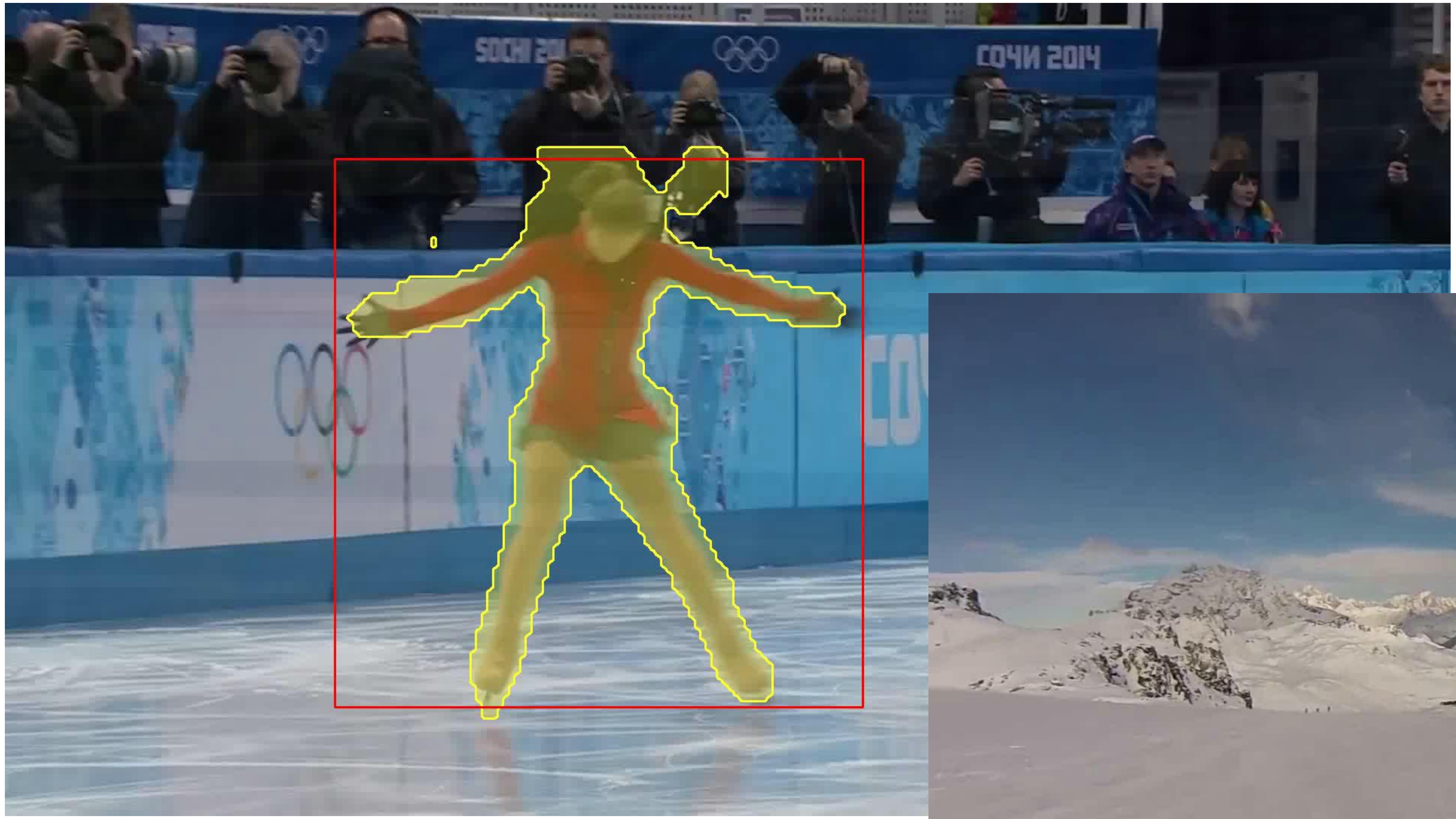
Object (traffic sign) detection

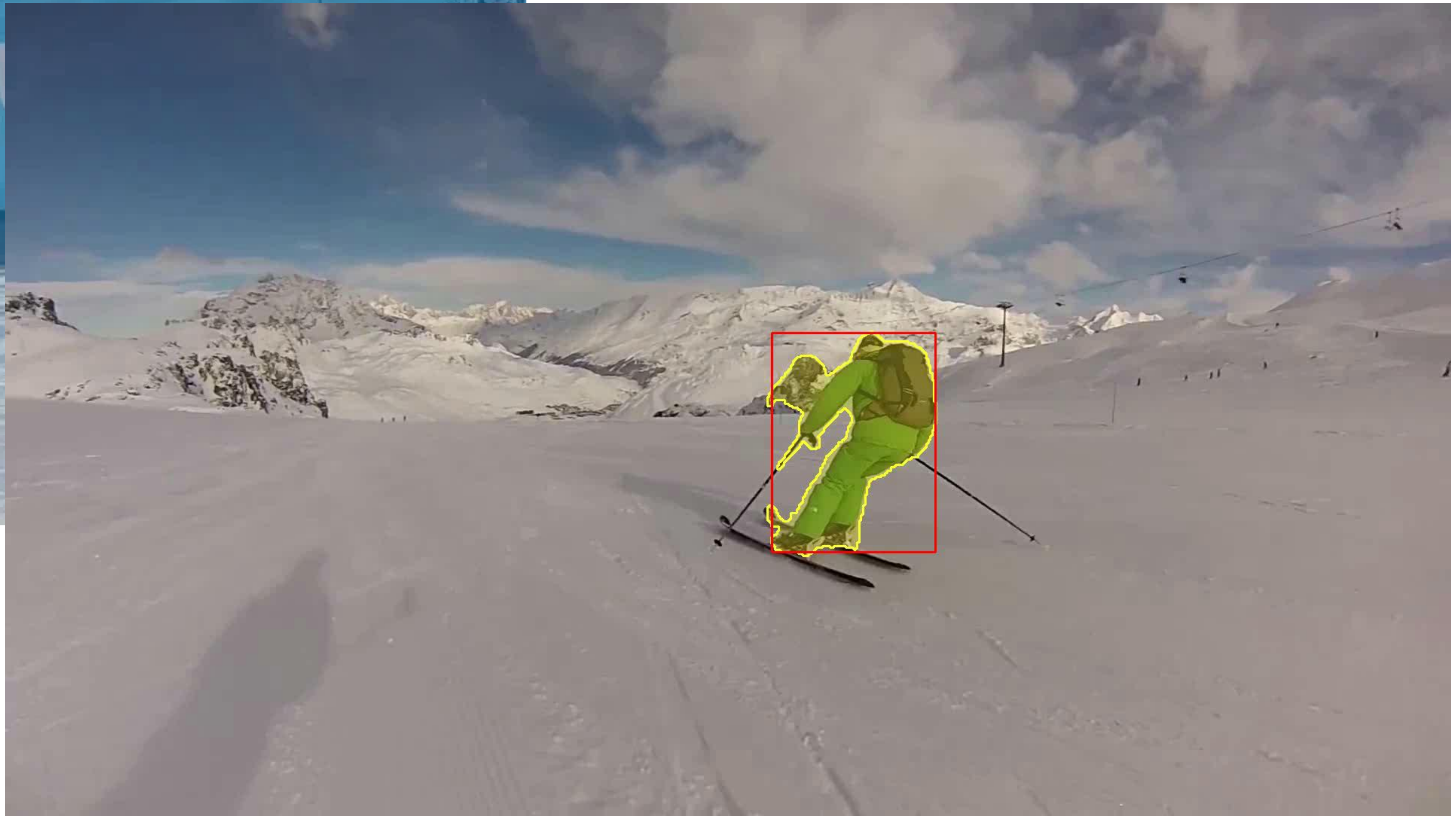
Image anonymisation

- Detection and anonimysation of car plates and faces

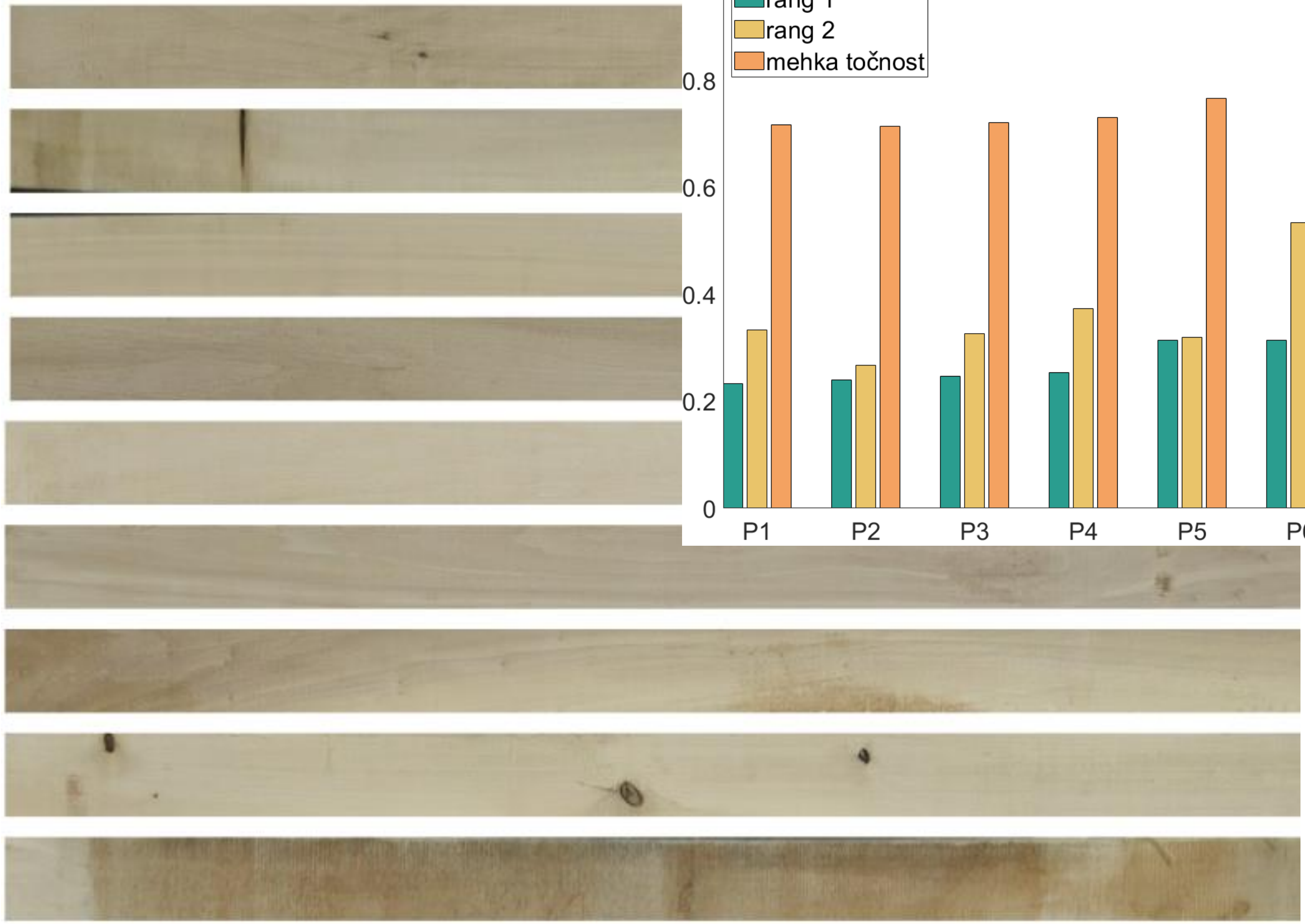


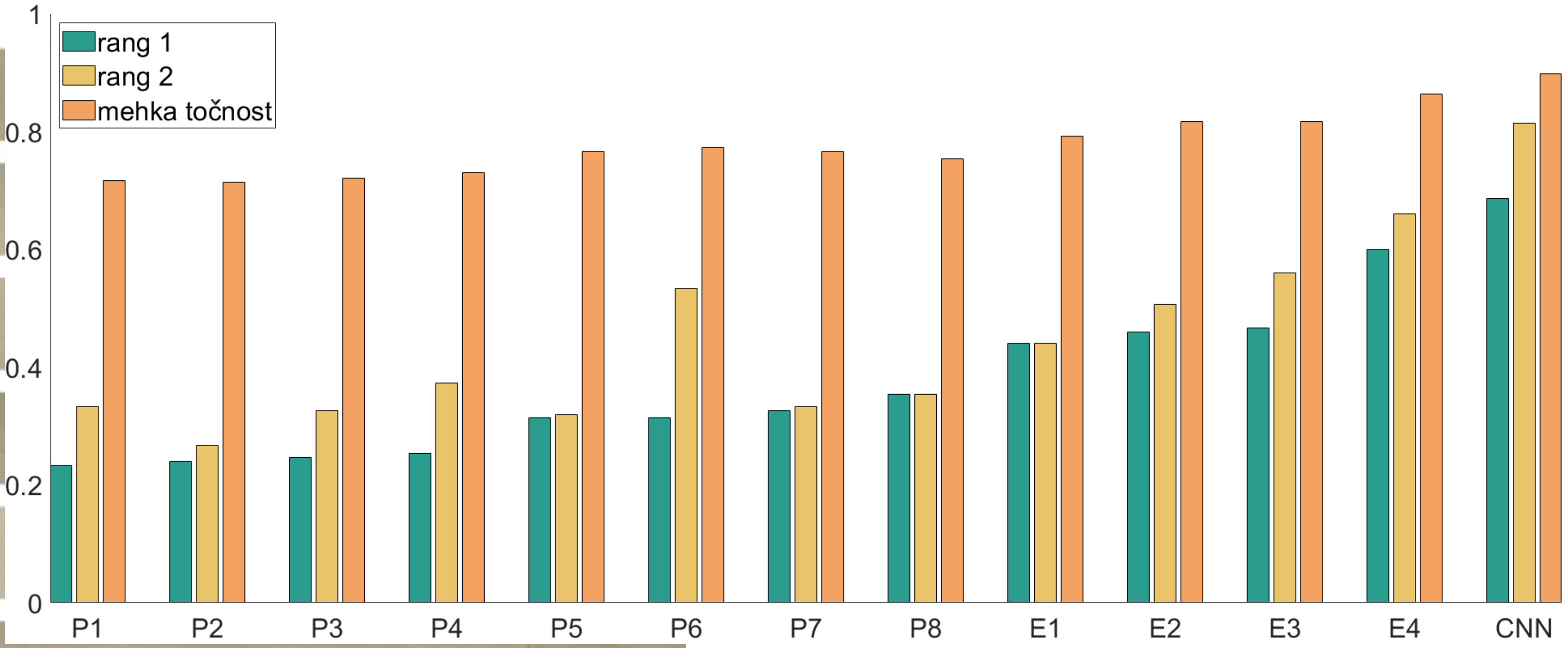
Visual tracking

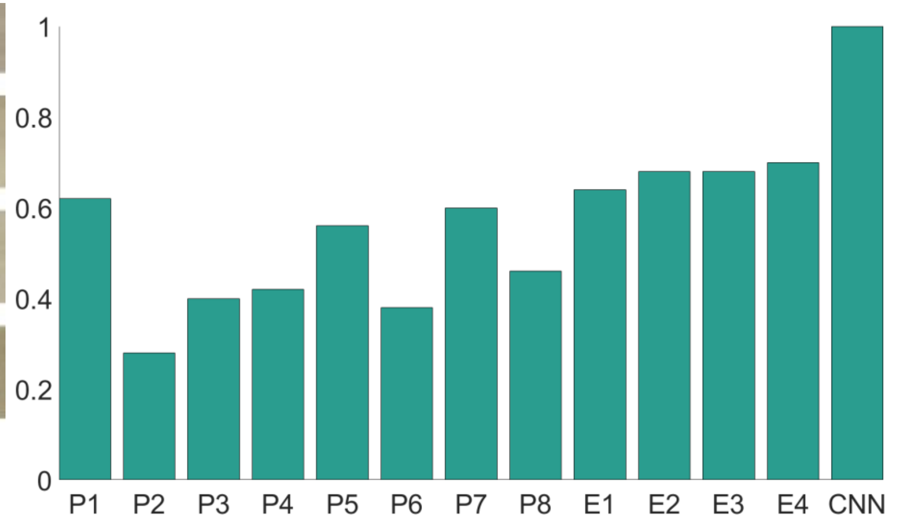




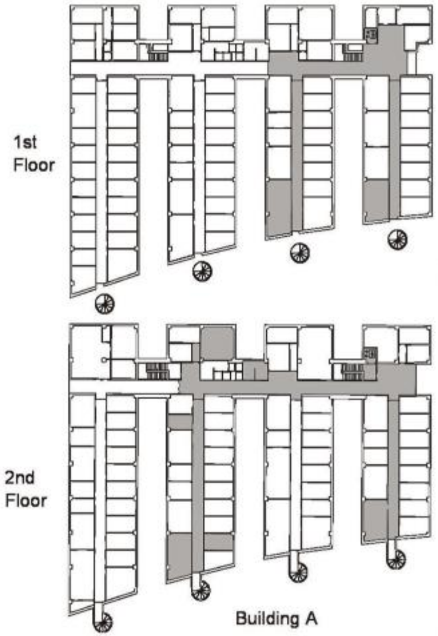
Plank classification







Place recognition



Semantic segmentation

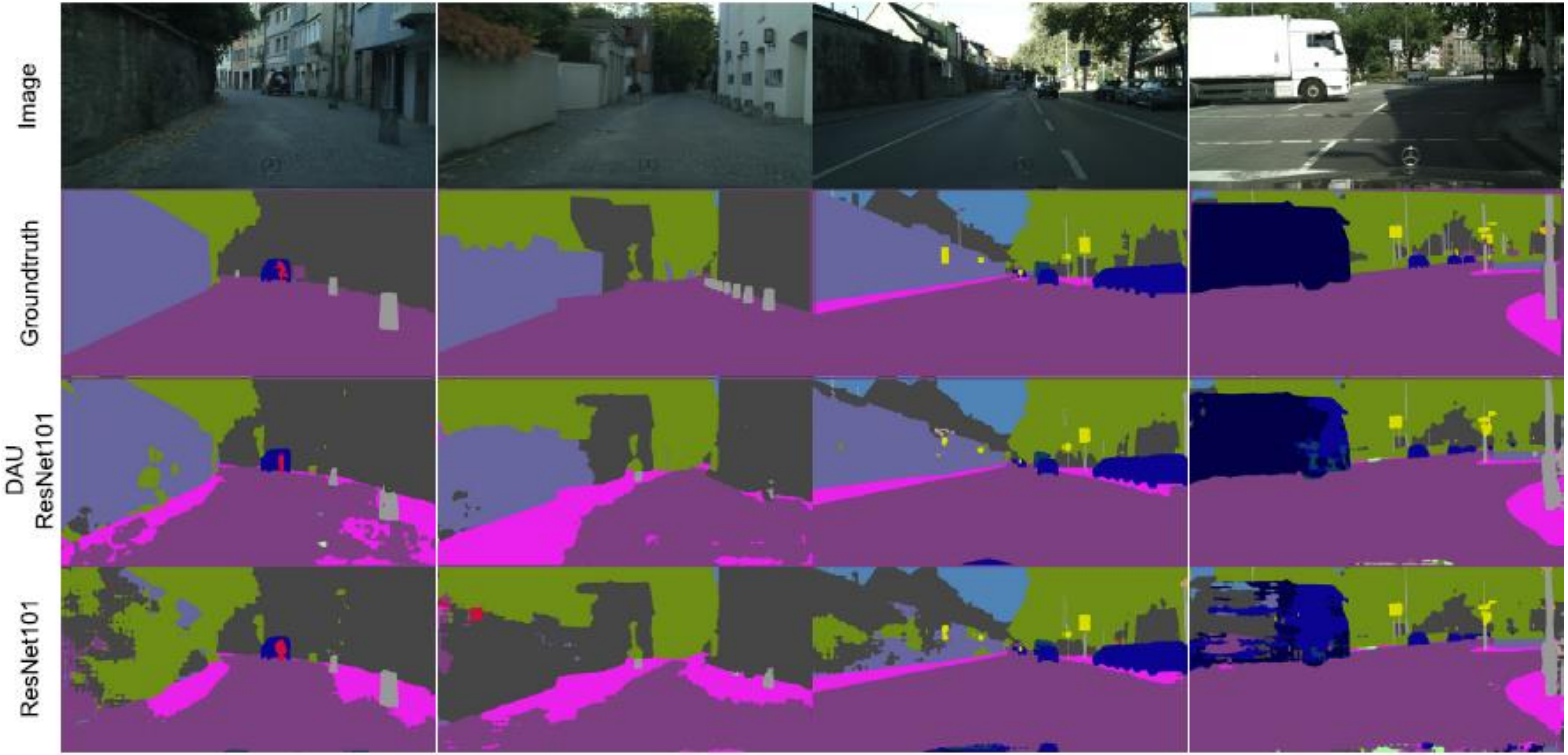


Image enhancement

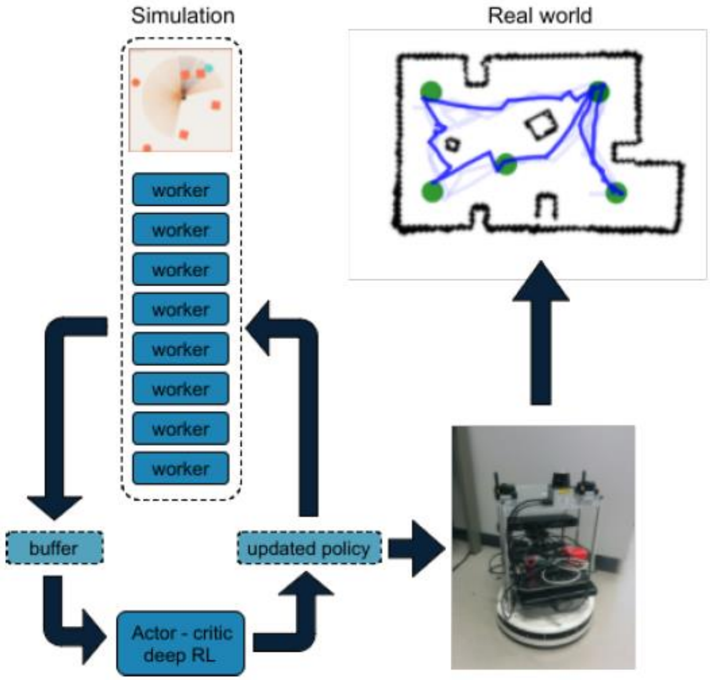
- Deblurring, super-resolution

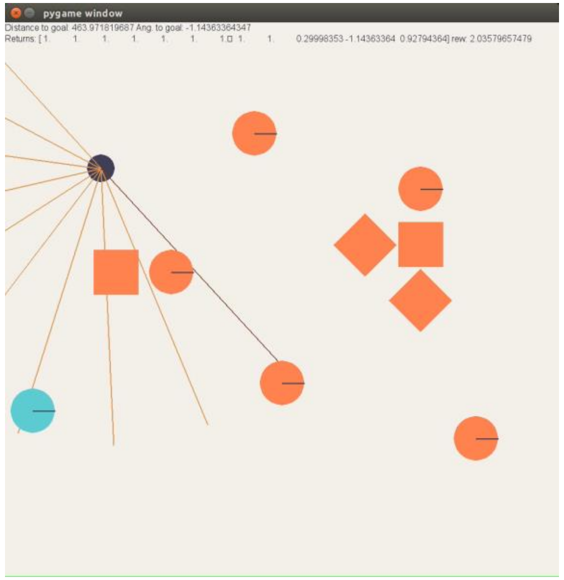

Original
Original
DAU-SNR-Deblur (our)
DAU-SNR-Deblur (our)
ABA

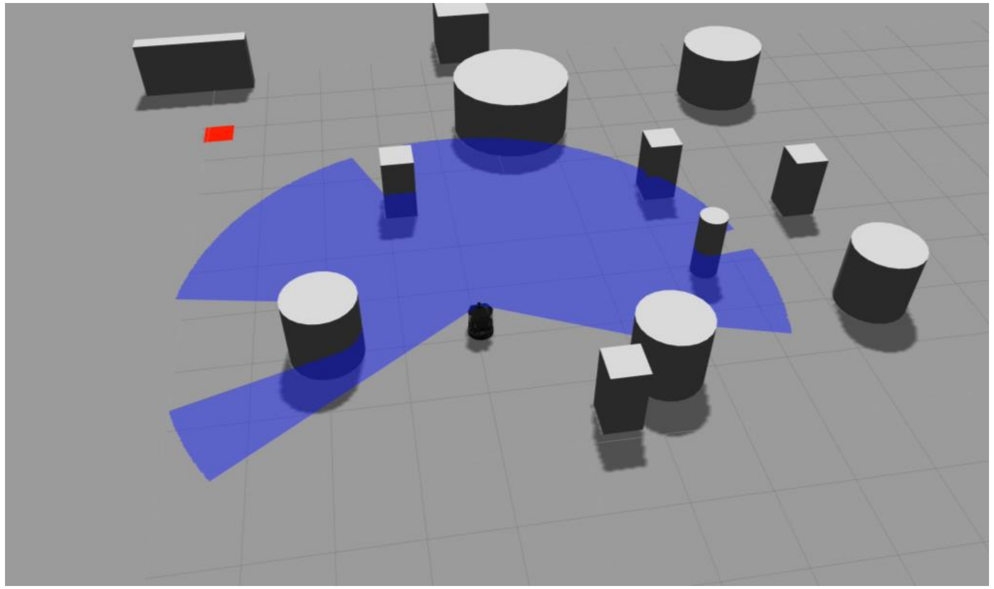

Original
Original
DAU-SNR-Deblur (our)
DAU-SNR-Deblur (our)

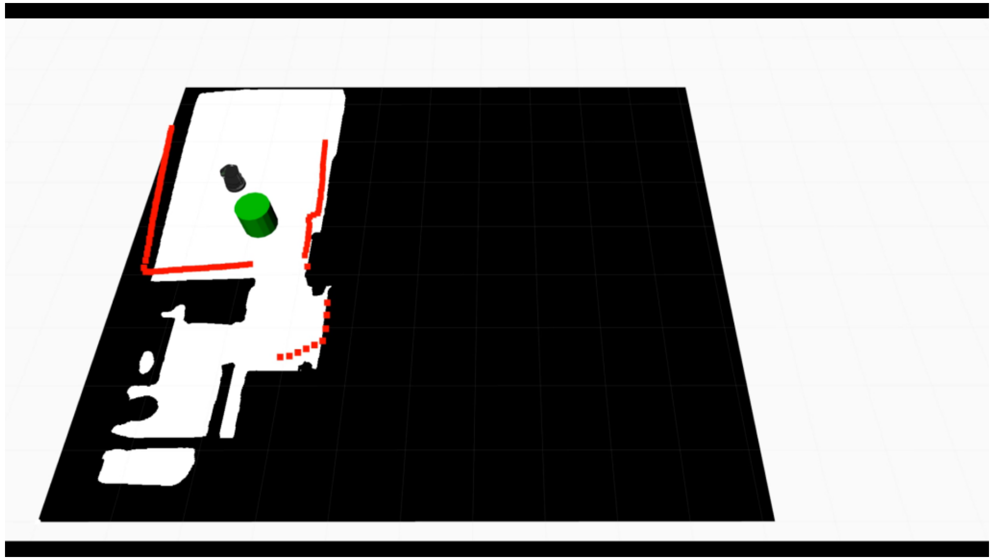
Deep reinforcement learning

- Automatic generation of learning examples
- Goal-driven map-less mobile robot navigation



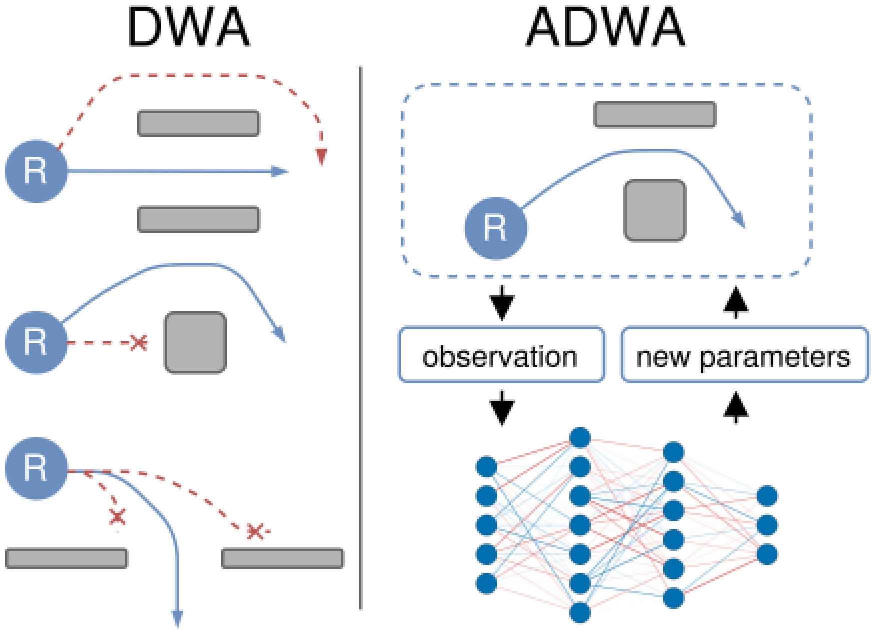


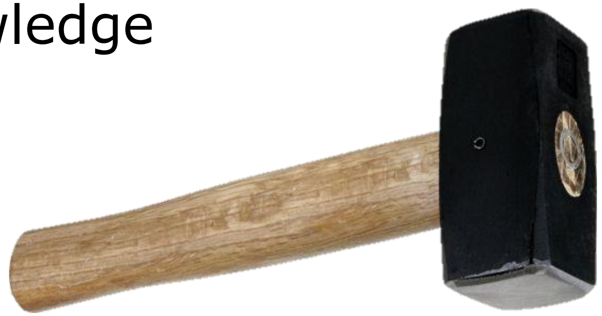


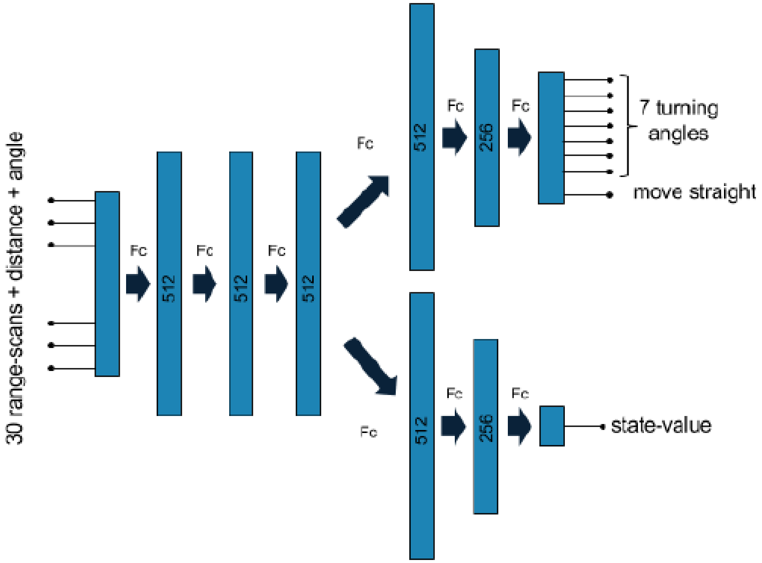


Innate and learned

- Goal-driven map-less mobile robot navigation
- Constraining the problem using a priory knowledge







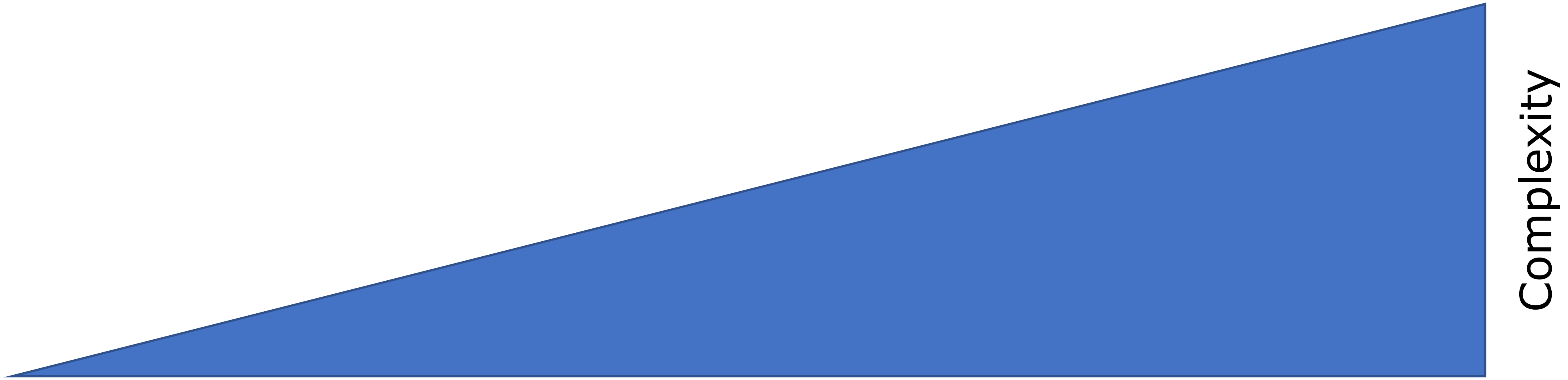
Engineering approach

Engineering approach + deep learning

Pure learning

Problem solving

- Different problem complexities



Simple,
well defined problems

Complex, vaguely
defined problems

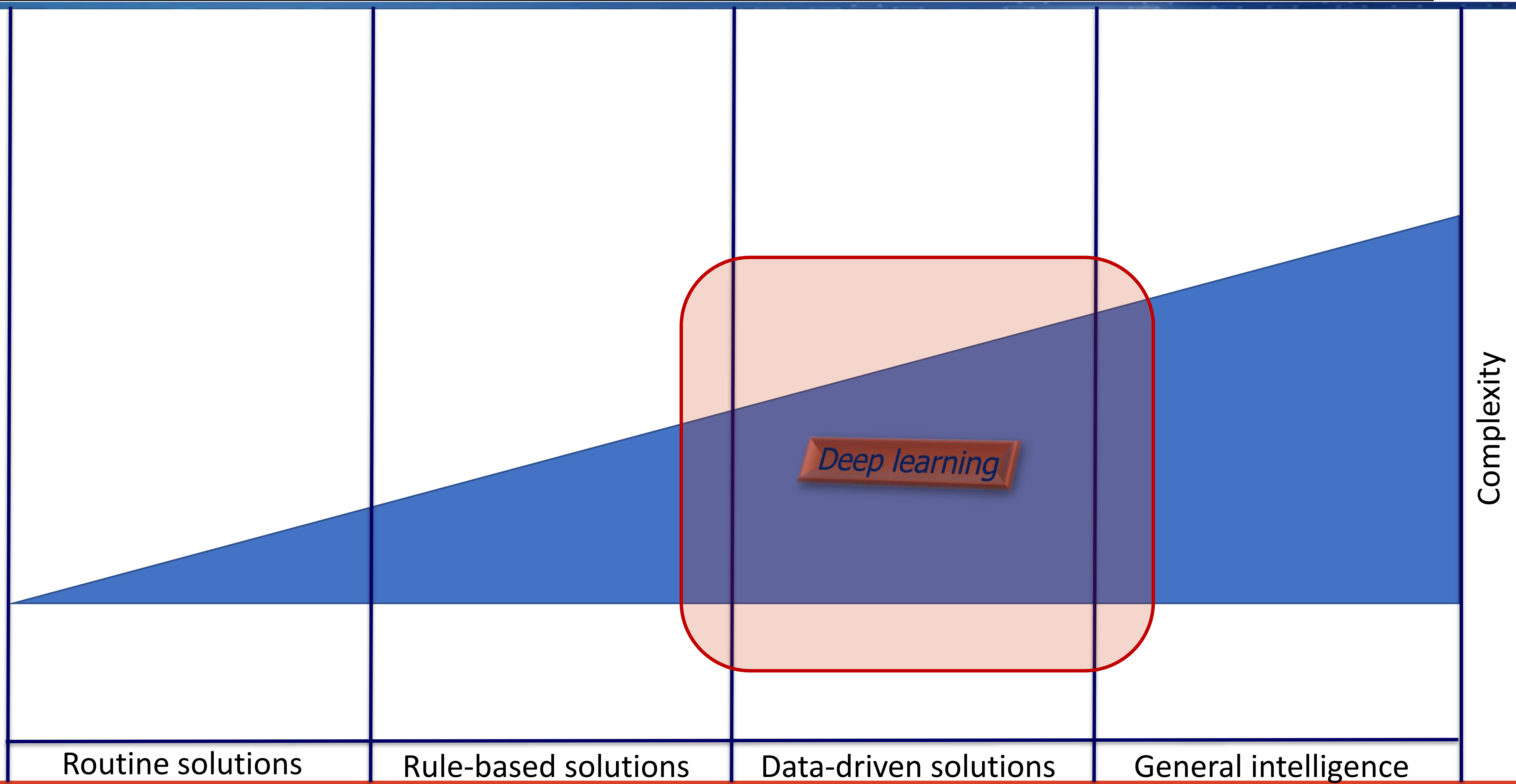
Rule-based
decision making

Data-driven
decision making

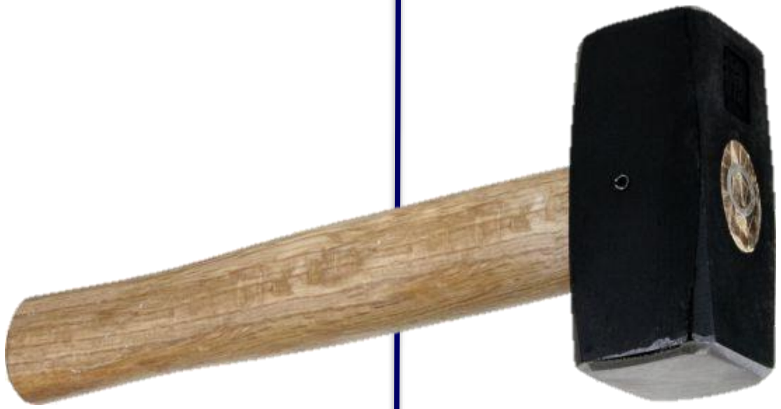
Programming

Machine learning

Problem solving



Adequate tools



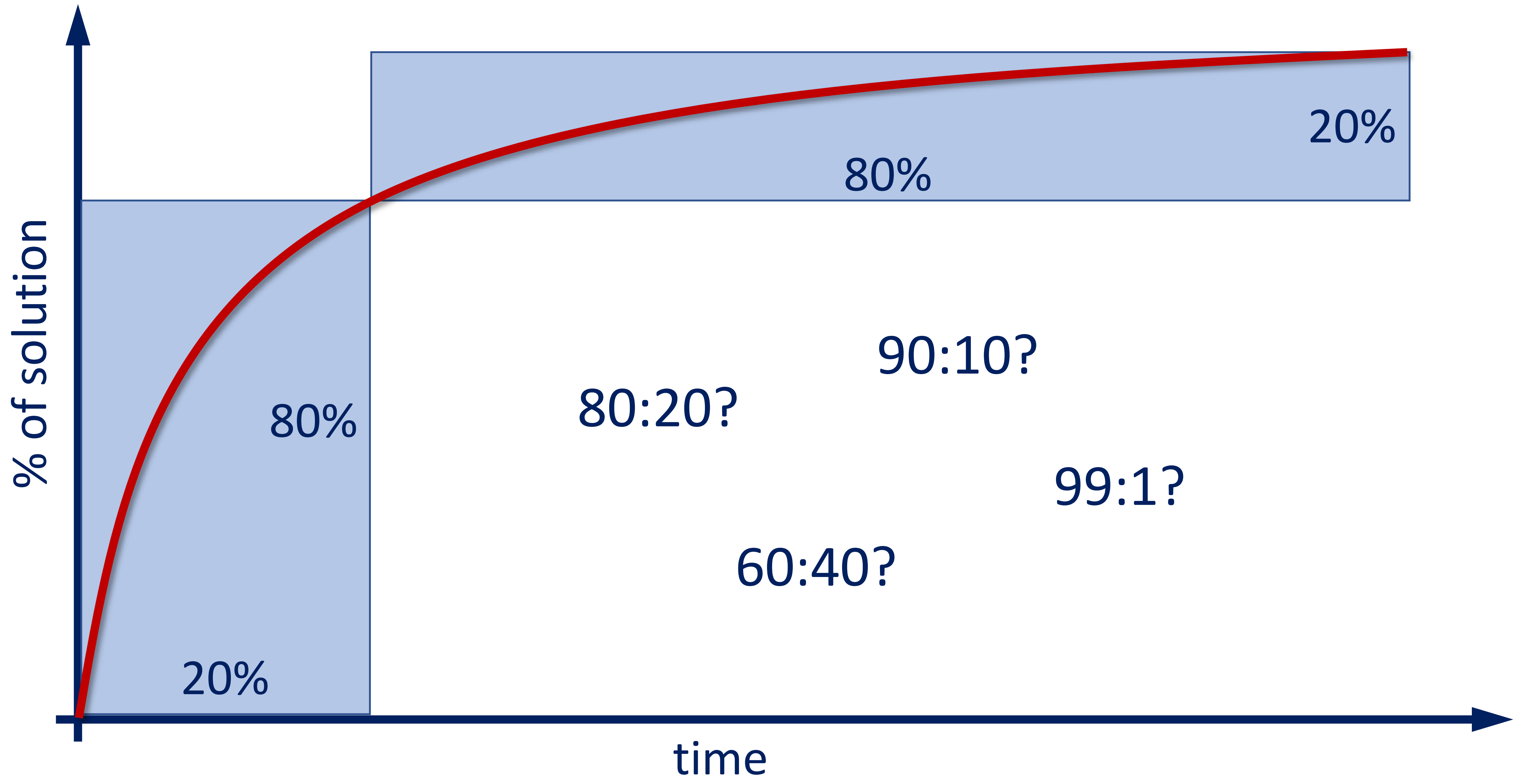
Deep learning

Routine solutions | Rule-based solutions | Data-driven solutions | General intelligence

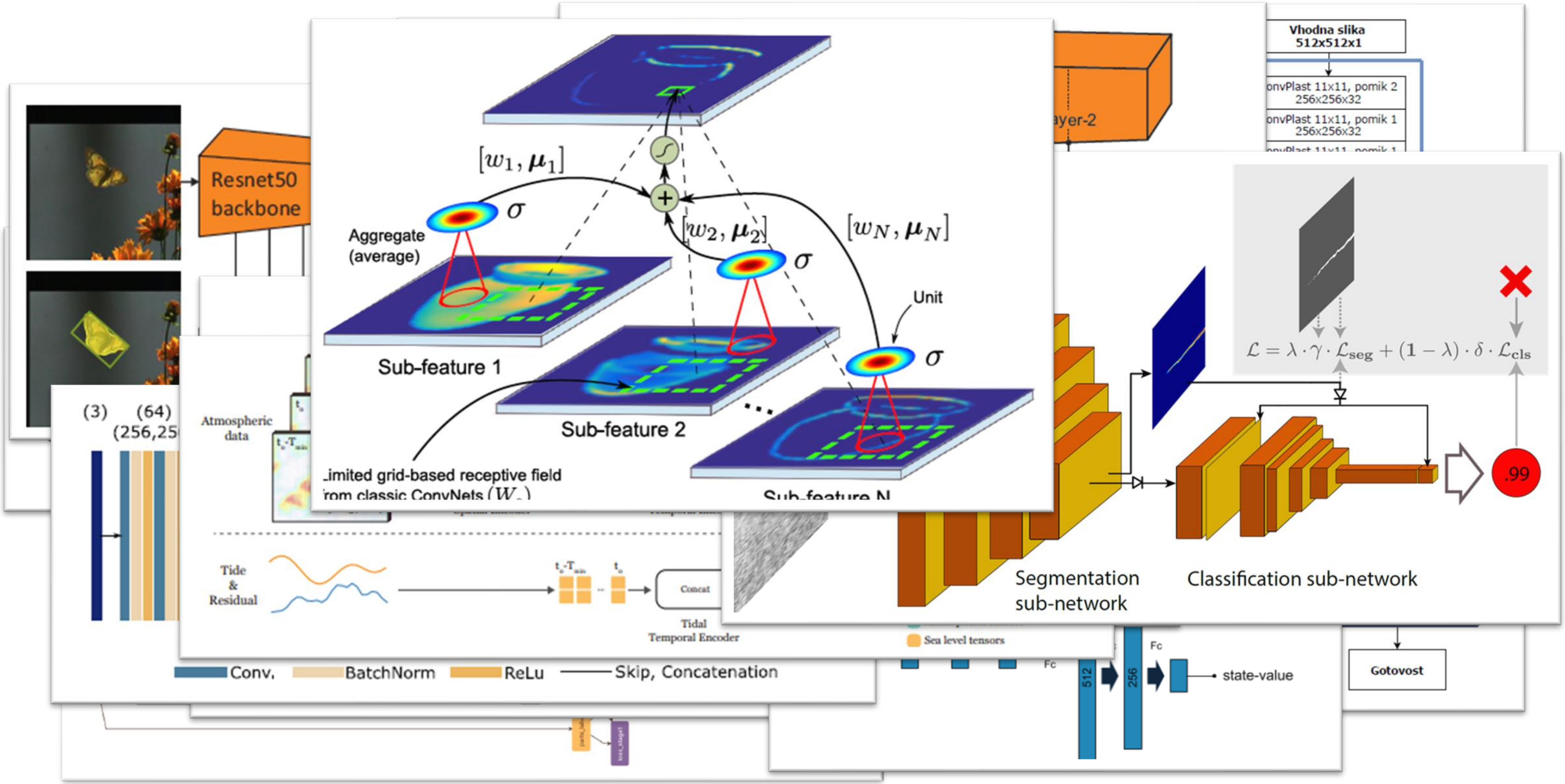
Development, deployement and maintainance

- Data, data, data!
  - Enough data, representative data
  - Correctly annotated data
- Appropriate deep architecture design
  - Proper backbone, architecture, loss function, …
  - Learning, parameter optimisation
- Efficient implementation
  - Execution speed
  - Integration
- Maintenance
  - Incremental improvement of the learned model
  - Reflecting to changes in the environment

Development of deep learning solutions



Knowledge and experience count



Software

- Neural networks in Python
- Convolutional neural networks using PyTorch or TensorFlow
- or other deep learning frameworks
- Optionally use Google Colab

Literature

- Michael A. Nielsen, Neural Networks and Deep learning, Determination Press, 2015 http://neuralnetworksanddeeplearning.com/index.html

- Ian Goodfellow and Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016 http://www.deeplearningbook.org/

- Fei-Fei Li, Andrej Karpathy, Justin Johnson, CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University, 2016 http://cs231n.stanford.edu/
- Papers