course:
Retrieval of Multimedia Content on the Web (NDBI034)

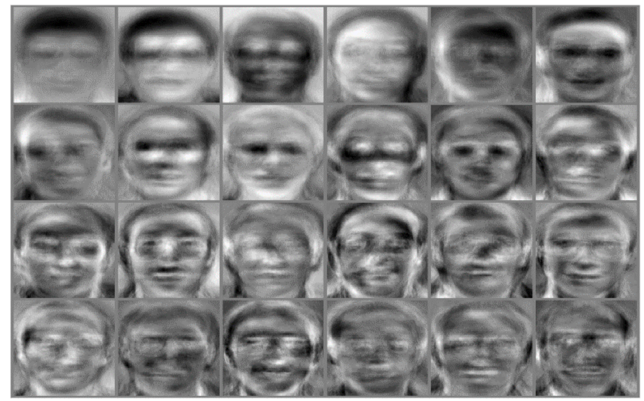
lecture 8:

Semantic descriptors – deep learning

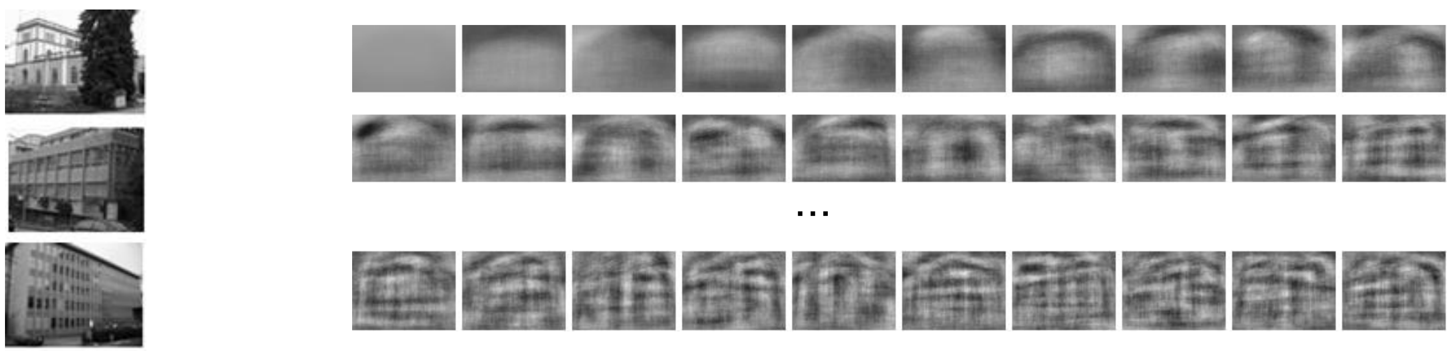
doc. RNDr. Tomáš Skopal, Ph.D.
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Semantic-visual words

- domain-specific high-level
  - eigenfaces (PCA)



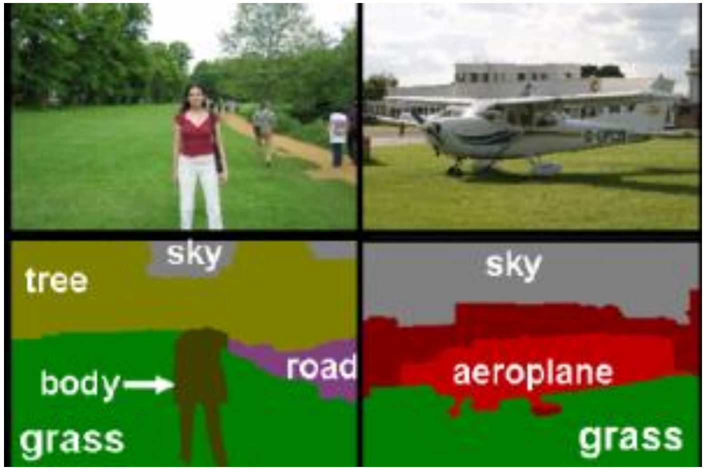
  - “eigenhouses” (SVD)



  - global match, restricted input (canonic form)

Semantic-visual words

- often complex scene
  - unlimited camera settings, 3D projection, occlusion, abstract, sketch
- hierarchical decomposition
  - object-wise segmentation?
    - extremely difficult in general
    - good for 3D scene reconstruction, etc.
    - not a solution for retrieval



(source: http://www.cs.utexas.edu/~grauman/courses/fall2011/schedule.html)

Semantic-visual words

- where to get the high-level semantics?
  - human annotators → ground truth (data in classes)
  - how?
- let the neural networks do this dirty job “somehow”!
  - supervised learning (also unsupervised for some tasks)
  - a sort of perceptron neural network (NN)
    - back-propagation (gradient descent)
    - training image example + annotation provided by human user = the NN “magically” connects the semantics with visual features!
    - well, not that easy ☺

Learning in information retrieval

- history = failure@large-scale
  - fully-connected layers
  - limited number of neurons per layer (n^2 parameters)
    - because of computation power
  - therefore, shallow neural networks
  - small training data = overfitting
- today = success@large-scale
  - convolutional and fully-connected networks
  - more neurons per layer (but below n^2 parameters!)
  - GPU technology enabling fast vector parallelism
  - therefore, deep networks
  - large training data
    - e.g., ImageNet, 1.2M training images, 1000 classes

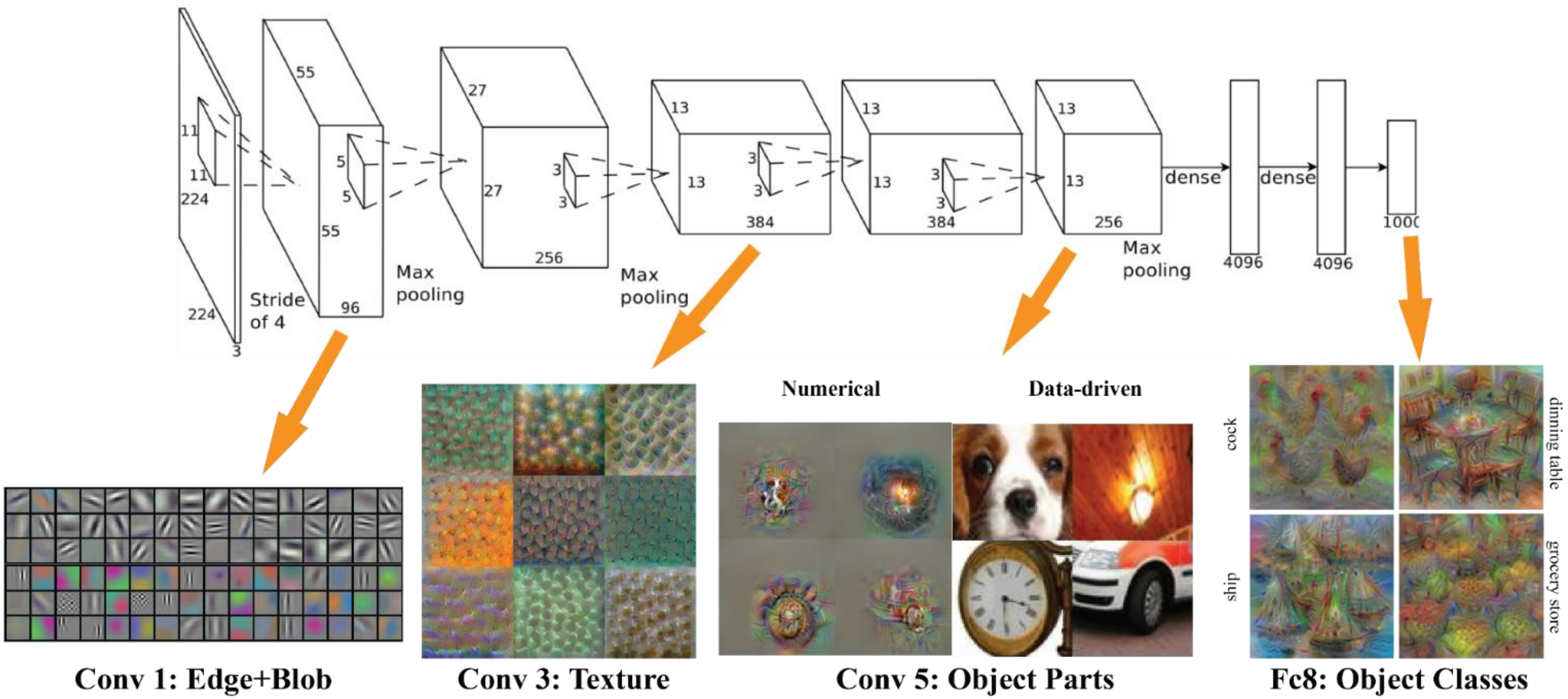
Semantic-visual words in CNN

- convolutional deep neural networks (CNN)
  - by LeCun 1989
  - convolution is hard-wired mechanism in human cognition (visual cortex)
    - where cells in retina ("pixels") is the input layer
  - inspired by this, CNN mimic the visual cortex
    - deep architecture: convolutional layers + fully connected layers
    - training example: raw image (no preprocessing, just scaling) + multiclass annotation
  - hierarchy of visual words
    - from low-level to high-level
    - fully connected layers on the top lead to semantic classes (as the output)

Semantic-visual words in CNN

- convolutional deep neural networks (CNN), cont.
  - the visual words (coded in neuron connection weights) arranged in the CNN such that they are activated regardless of
    - locality
    - scale
  - the last layer = semantic annotation
  - a (last – k)$_{th}$ layer = semantic-visual descriptor
    - the smaller k, the more semantic/less visual
    - the greater k, the more visual/less semantic

Convolutional deep network

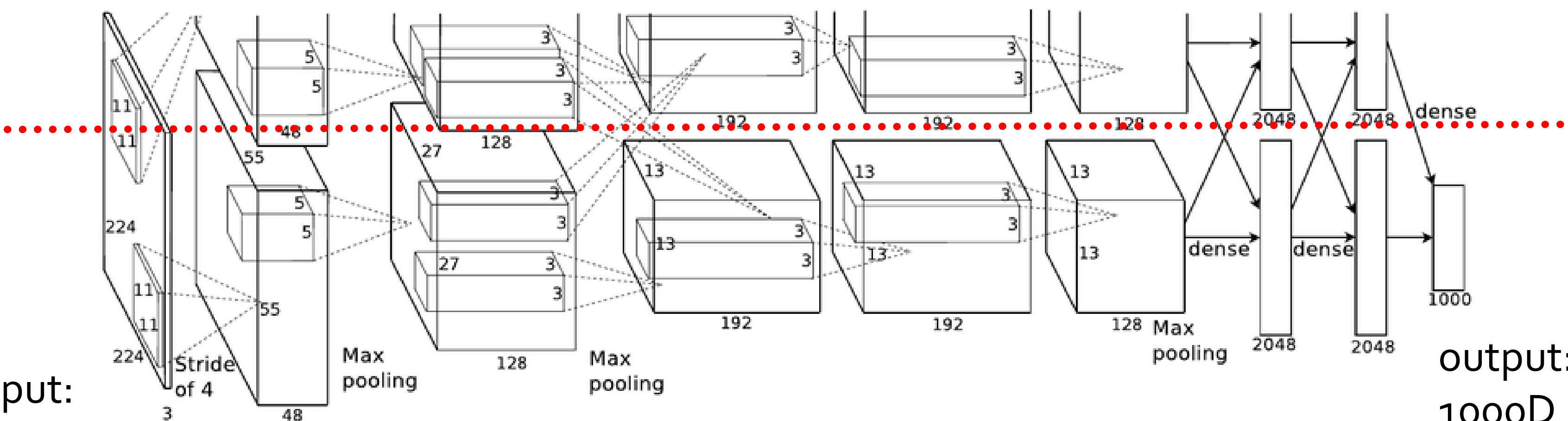


[Fig source: http://vision03.csail.mit.edu/cnn_art/index.html]

Convolutional deep network

example: AlexNet

GPU1



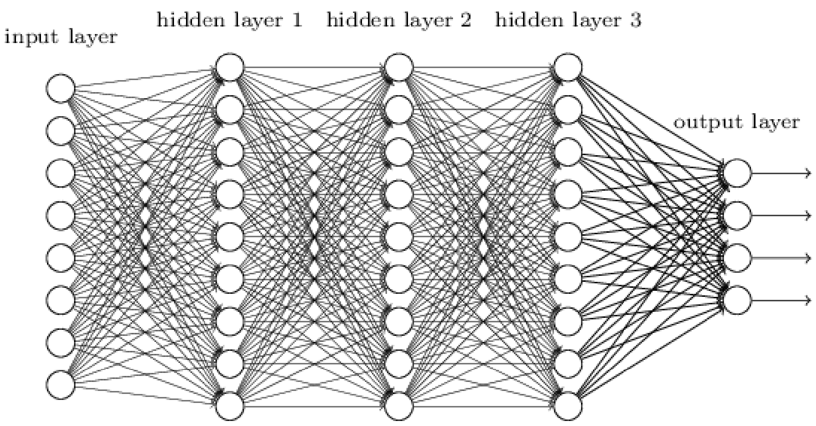
GPU2

input: **color image** scaled to 224x224

output: 1000D vector over **semantic classes**

Convolutional deep network

- why not traditional fully-connected perceptron layers?
  - not fit to image recognition needs
  - too many weights (parameters) vs. too few neurons (feature detectors)
    - prone to overfitting
    - bad recognition effectiveness/resolution
    - computationally expensive
  - global vs. local receptive field
    - not translation-invariant



- convolutional network
  - **actually IS the traditional multi-layer perceptron network!**
  - but constrained and designed for image recognition (annotation) task
    - not fully-connected, receptive fields introduced, shared weight banks (filters), etc.

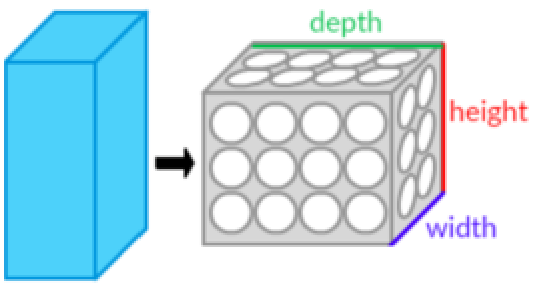
Convolutional deep network

- back-propagation (learning)
  - looks for the minimum of the error (loss) function in weight space using the method of gradient descent
    - applying partial derivative of the loss function w.r.t. w_i or bias
  - requires differentiable activation functions
    - like sigmoid, or treating the indifferentiable points (ReLU)
  - finds local extremes
  - note that in convolutional layers the convolution filters are learned!
    - compare to static convolution filters used for edge detection, etc.

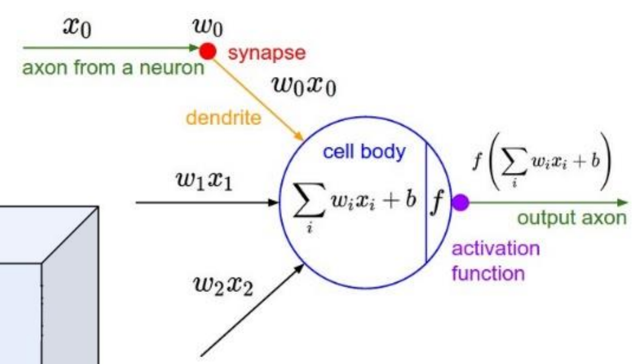
Convolutional deep network

- types of layers
  - data layers (input, output)
  - vision layers
    - convolutional layers, pooling layers
  - activation layers
    - ReLU, Sigmoid, …
  - common layers
    - inner product (fully connected layer), …
  - loss layers
    - softmax, Euclidean, …

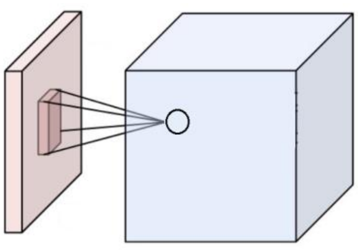
Convolutional layer



- neurons arranged in 3D block (3D layer)
  - width x height x depth
- each neuron is classic perceptron
  - connected to a set of neurons in the previous layer arranged in receptive field (3D block of neurons)

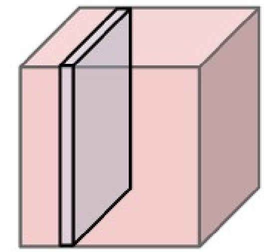


    - receptive field depth = previous layer depth
    - weights w_i of the connections = 3D block (filter) of the same volume as the receptive field

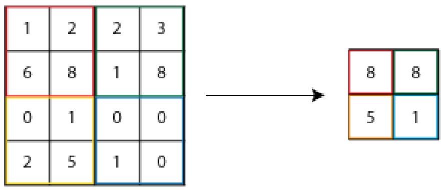


  - neuron is activated as usual
    - computing dot-product of activations x_i of neurons from the receptive field and the filter

Convolutional layer

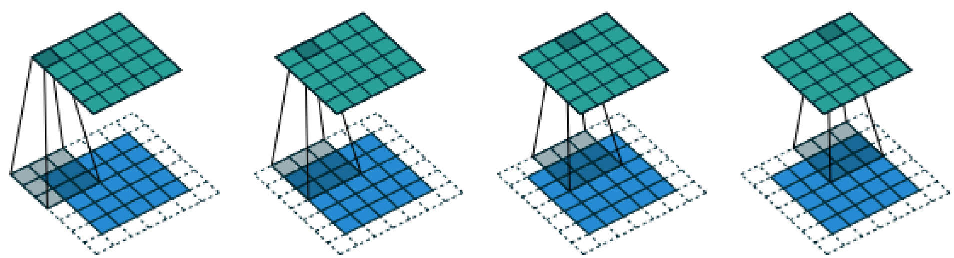


- depth slice
  - neurons in the same depth of layer
    - share the same filter
      - assuming same features may appear regardless of the position in image
  - for each neuron the receptive field slides (in width, height)
    - stride = parameter of sliding
      - 1 is by 1 pixel, 2 is by 2 pixels, so smaller output



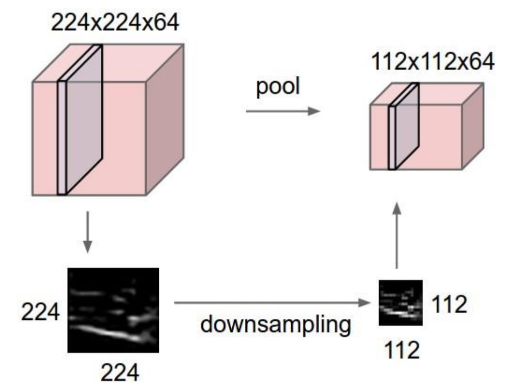
    - zero-padding = thickness of borders filled with zeros
      - extends the sliding region

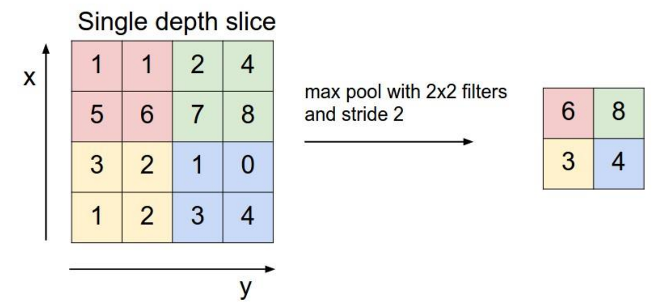
0	0	0	0	0	0	0	0	0	0
0	77	80	82	78	70	82	82	140	0
0	83	78	80	83	82	77	94	151	0
0	87	82	81	80	74	75	112	152	0
0	87	87	85	77	66	99	151	167	0
0	84	79	77	78	76	107	162	160	0
0	86	72	70	72	81	151	166	151	0
0	78	72	73	73	107	166	170	148	0
0	76	76	77	84	147	180	168	142	0
0	0	0	0	0	0	0	0	0	0



Pooling layer

- reduces the spatial size of input
  - reason: performance, overfitting
  - placed in between convolutional layers
  - could be omitted
- for every depth slice
  - usually 2x2 filter applied and stride 2
  - usually max pooling (max value taken), could be also avg, L2-norm
- backpropagation
  - routing the gradient to input with the highest value





ReLU (Rectified Linear Unit) layer

- elementwise activation function $f(x) = max(0,x)$
- most popular activation function for deep networks
  - computationally cheap
  - scale-invariant
  - efficient gradient propagation
  - biologically plausible

Fully connected and loss layers

- fully connected layers
  - as in regular networks, neurons connected to all neurons in previous layer
  - activation computed as inner product
- loss layer
  - usually the last layer (in very deep CNNs also in the middle)
  - determines the penalization of predicted and true labels
    - softmax – predicting one class out of *k* (sums to 1)
    - sigmoid cross-entropy – *k* independent probabilities in [0,1]

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j} \qquad f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

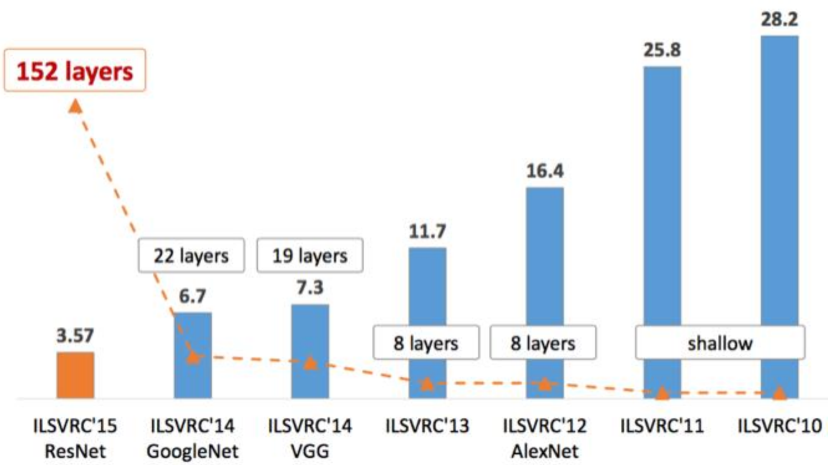
Basic use case – classification

vector of weights/
probabilities to classes
(loss layer)

```
---------- Prediction for ../../../examples/images/bike.jpg ----------
0.5061 - "n03792782 mountain bike, all-terrain bike, off-roader"
0.4462 - "n09193705 alp"
0.0104 - "n09468604 valley, vale"
0.0079 - "n09246464 cliff, drop, drop-off"
0.0065 - "n09472597 volcano"
```

Basic use case – classification

- models (trained CNN) rapidly evolving
  - ILSVRC (ImageNet Large Scale Visual Recognition Challenge)
  - going deeper…

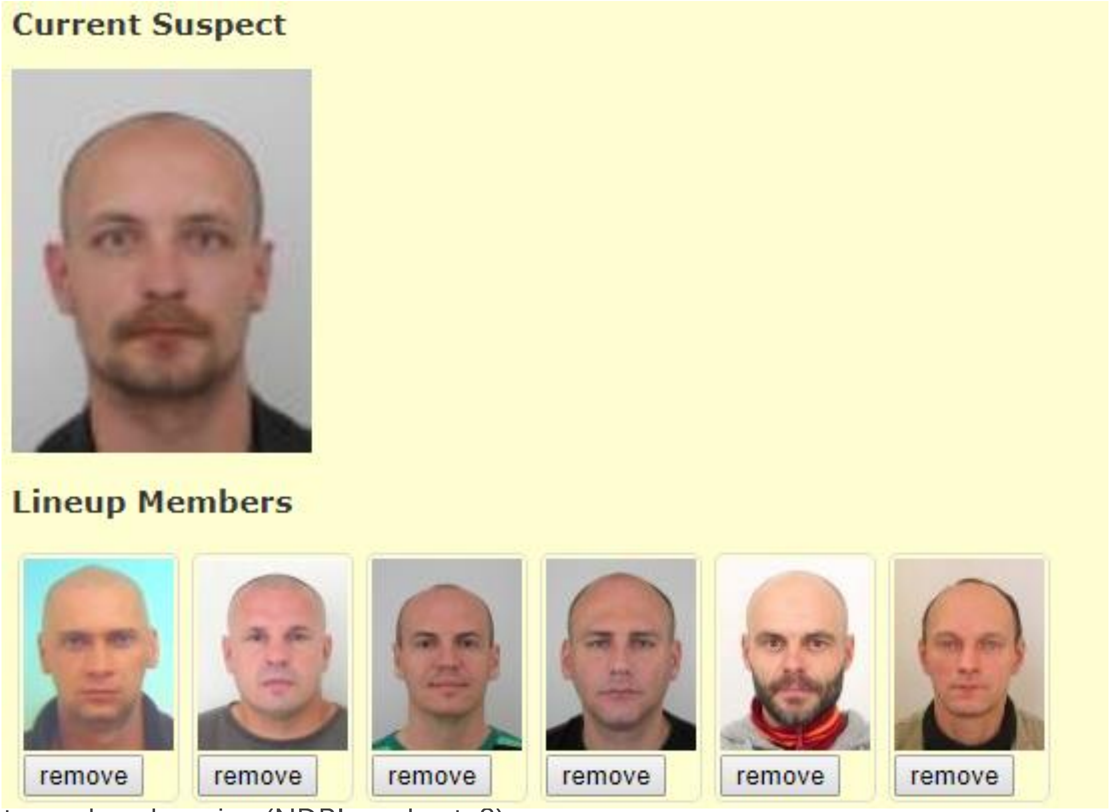


Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

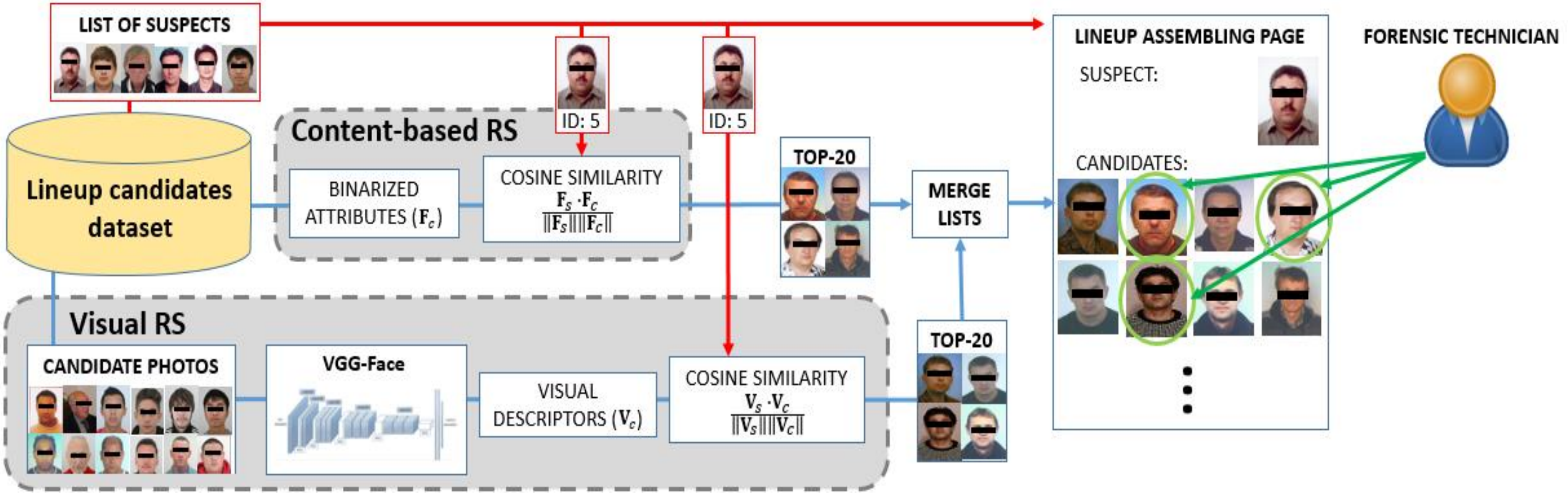
Basic use case – similarity

- Use results of (some layer of) CNN to calculate similarity of objects

- Recommend
  for police lineups

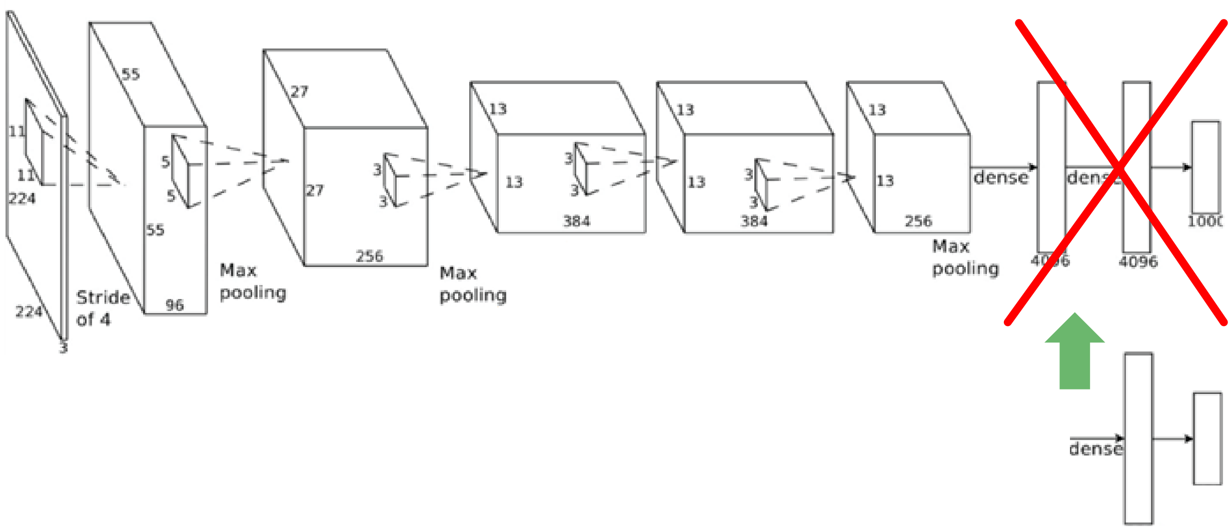


Basic use case – similarity

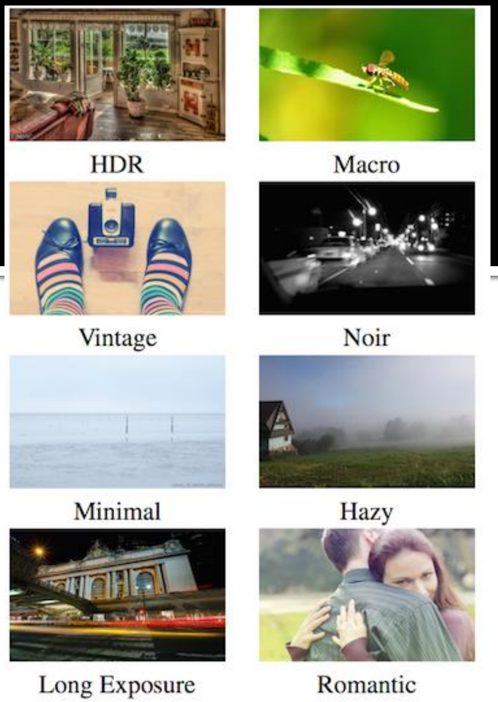


Transfer learning in CNN

- light-weighed transfer learning
  - fine-tuned CNN - taking an already learned model, adapting the architecture, and resuming training from the already learned model weights



Transfer learning in CNN



- example Flickr style fine-tuning
  - take AlexNet model (trained on 1.3M generic images)
  - replace classification layer (1000 class neuron) by another (20 style class neurons) + random init.
  - train (fine-tune) by smaller training data (80K Flickr images, 20 style classes)