Object Recognition

Chapter 9: Deep Neural Networks for Object Recognition

Prof. Dr. Johannes Maucher

HdM CSM

Version 1.2
26.06.2017

Document History

Version Nr.	Date	Changes
1.0		Initial Version
1.1	19.06.2017	Added Slides for Gradient Descent Learning
1.2	26.06.2017	Added Slides for Deconvolution

Chapter 9: Deep Neural Networks for Object Recognition

Goal of Deep Learning

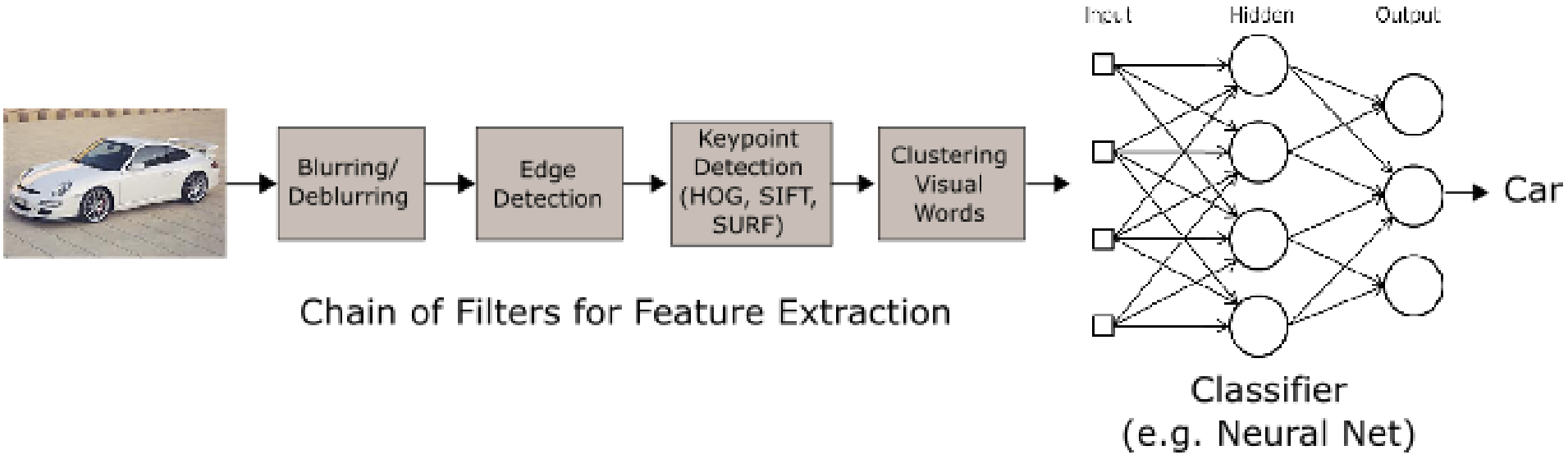
Goal as stated in [Bengio, 2009]

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. Automatically learning features at multiple levels of abstraction allow a system to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features.

Object Recognition: How to extract abstract features from images?

Humans recognise objects based on higher-level (abstract) features

Conventional Approach for feature extraction:

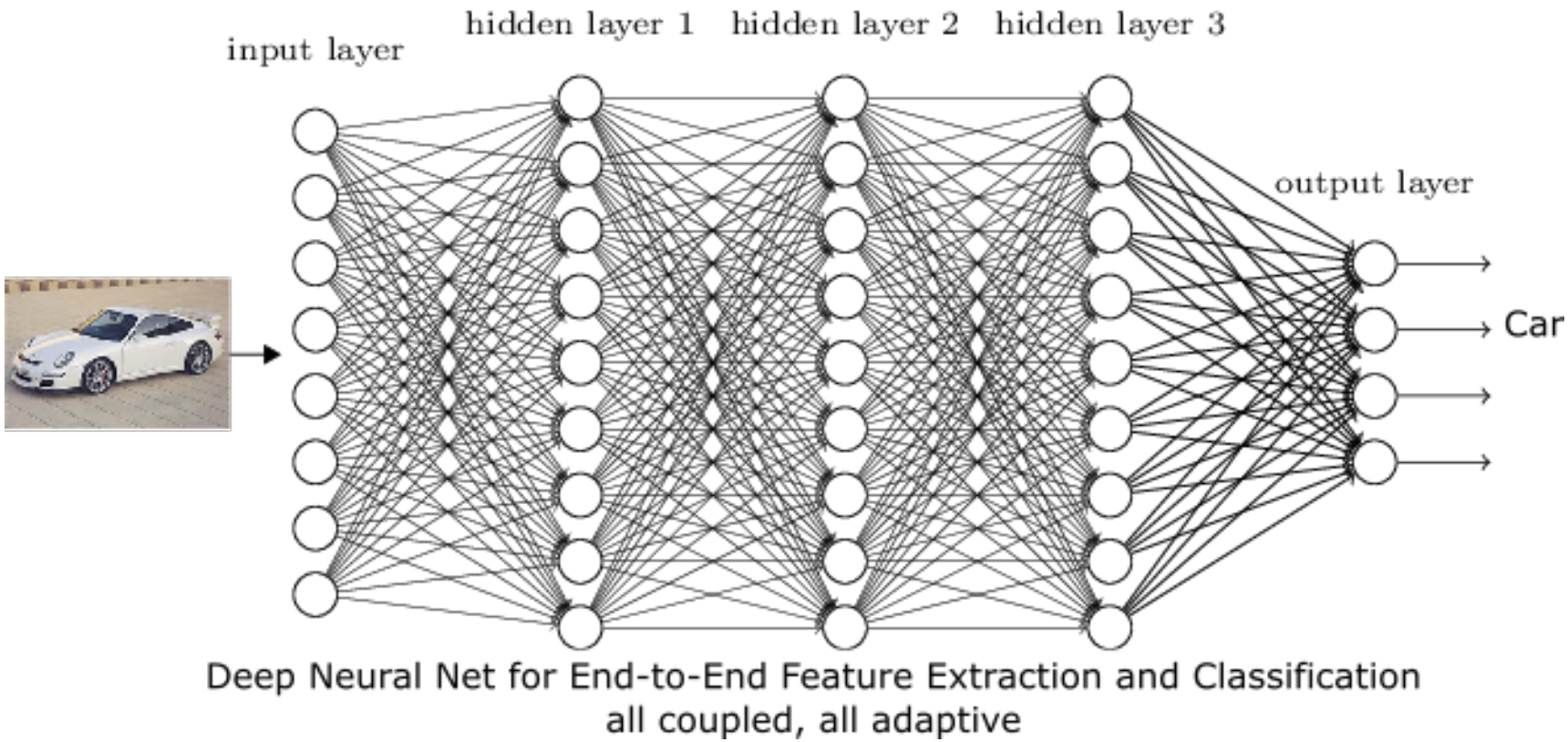


Drawbacks of conventional feature extraction

- Requires expert knowledge - which filters are suitable?
- Strongly application dependend
- All modules are decoupled

Object Recognition: How to extract abstract features from images?

Why not like the human brain?



Reduce Number of Parameters

Drawback of MLP and SLP

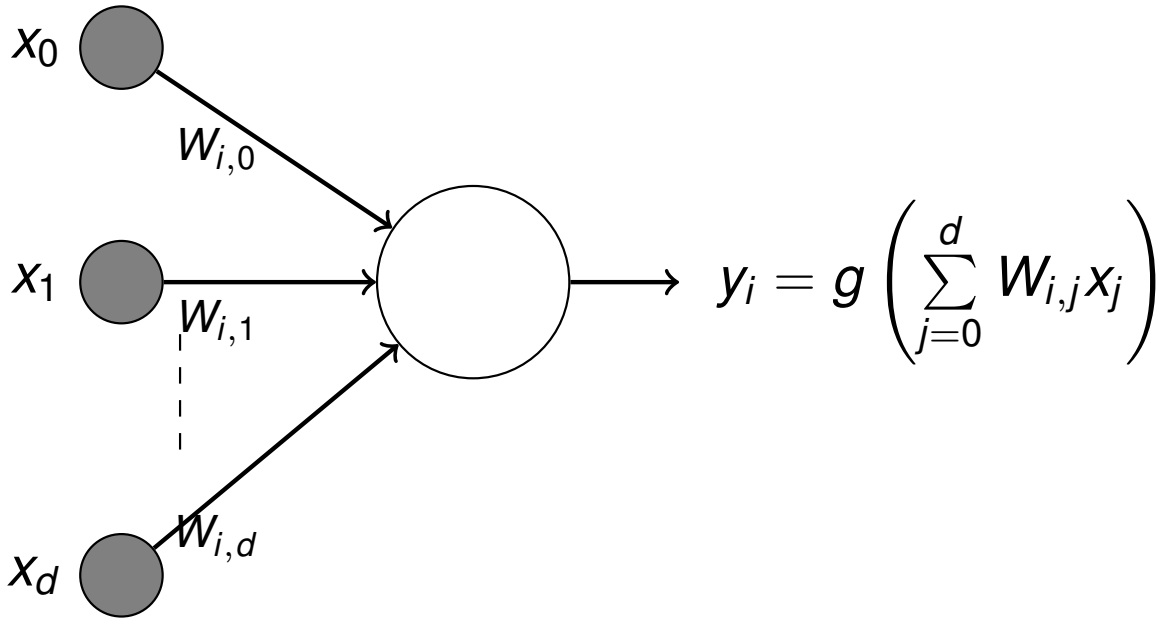
- Full connection implies large amount of free parameters
- Example: SLP linear classification of images with 100×100 pixels: 10^4 parameters
- Number of required training samples increases with the number of free parameters

Idea:

- Connect neurons only with a few neurons in the previous layer.
- Problem: Which neurons shall be connected?
- Answer: Depends on the domain and what shall be done in this domain.
- Common Approach: Contiguous neurons in one layer are connected to a common neuron in the successive layer.
- Examples:
  - Image domain: neurons which belong to the same spatial subregion of the image are contiguous.
  - Audio domain: neurons which belong to a common time-window are contiguous

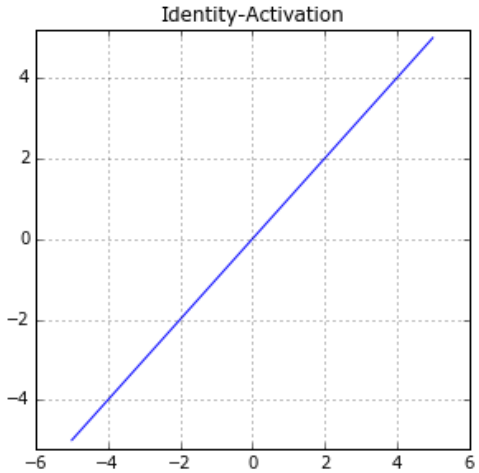
The first and still the prime domain of Convolutional Neural Networks is image processing/classification.

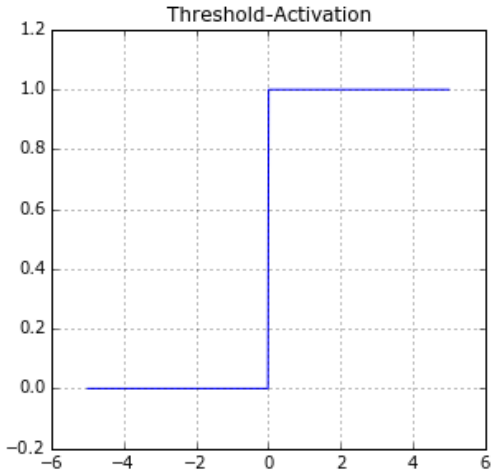
Model of Artificial Neuron

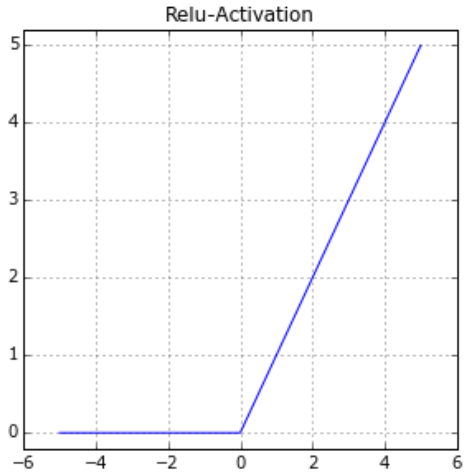


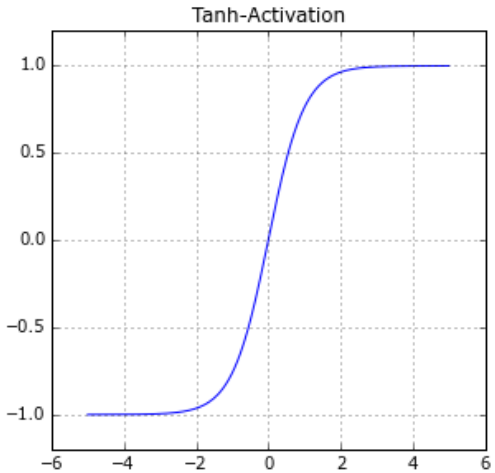
- Weights $W_{i,j}$ represent *connectivity* between neuron j in previous layer and neuron i in current layer.
- Activity function $g(in_i)$ maps the sum $in_i = \sum_{j=0}^{d} W_{i,j} x_j$ at the input of the neuron to an output value.
- Activity function is chosen in dependence of task (classifikation, regression), net-topology and learning algorithm.

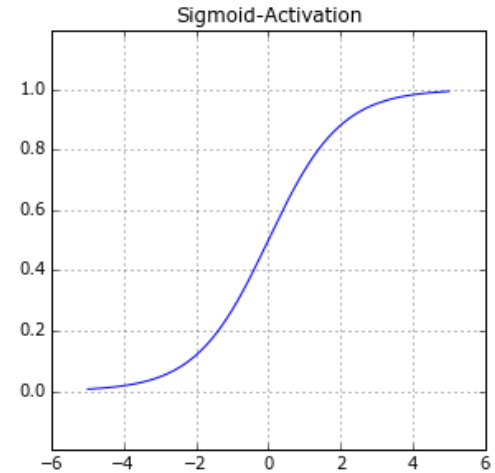
Activity functions





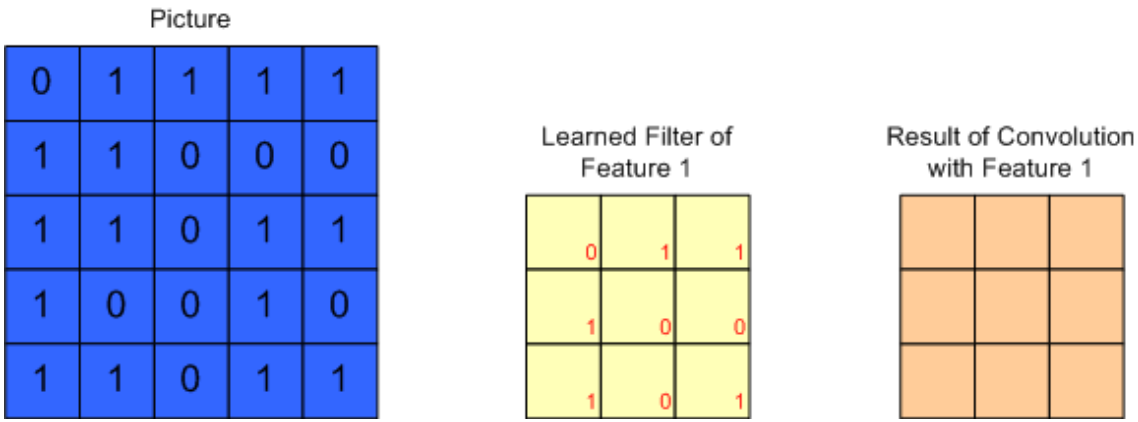




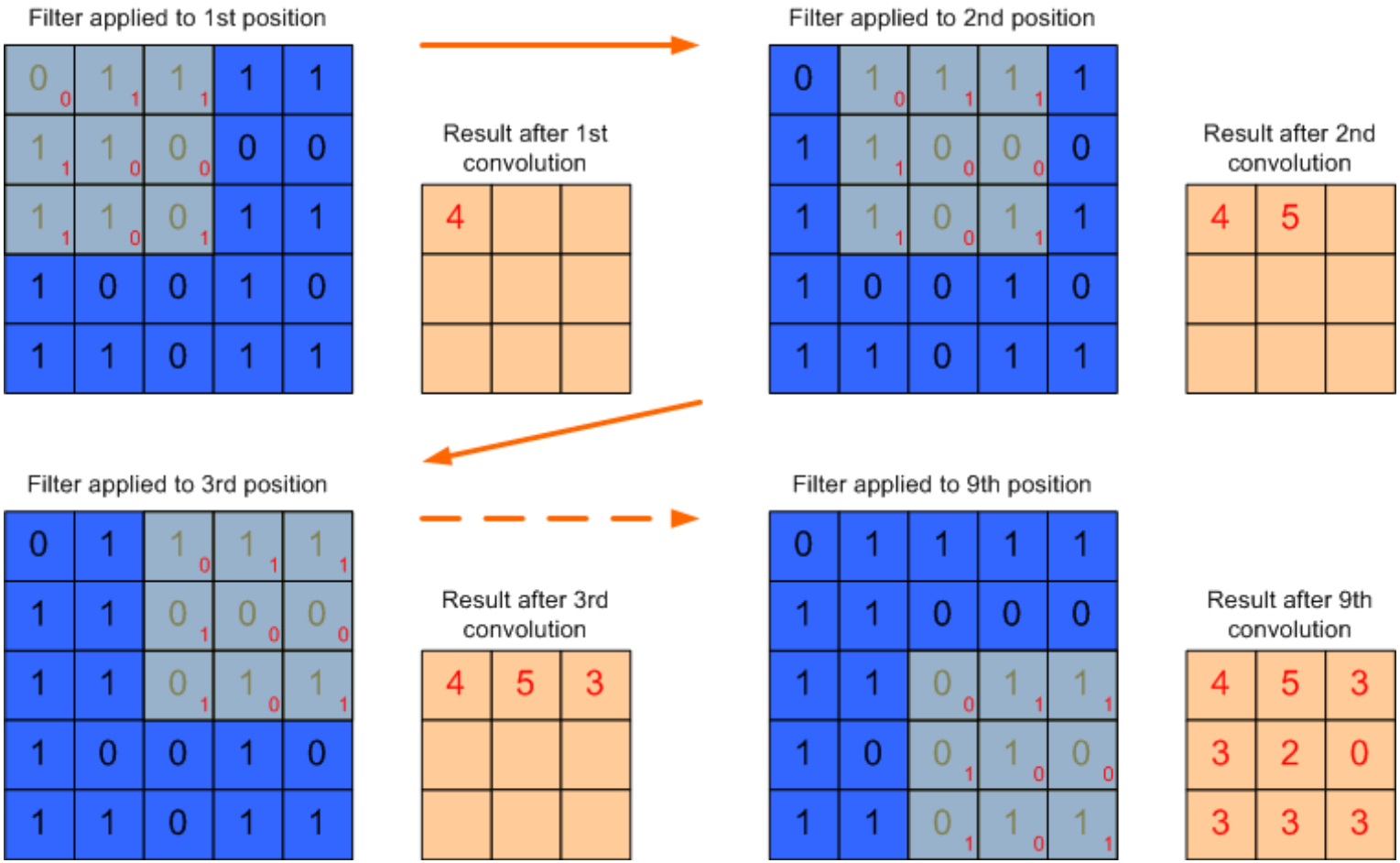


Concept of Convolution

1. Let I be the the input image with r rows and c pixels per rows.
2. Define the filter size: a is the number of rows and b is the number of pixels per row in the filter.
3. Shift a filter of size (a, b) for a specific feature f_k over all possible subregion positions of the Image I. The subregion-size is equal to the filter-size. The number of possible subregion-positions is $(r - a + 1) \cdot (c - b + 1)$.
4. At a given subregion position the filter output is an indicator for the presence of feature f_i in this subregion.

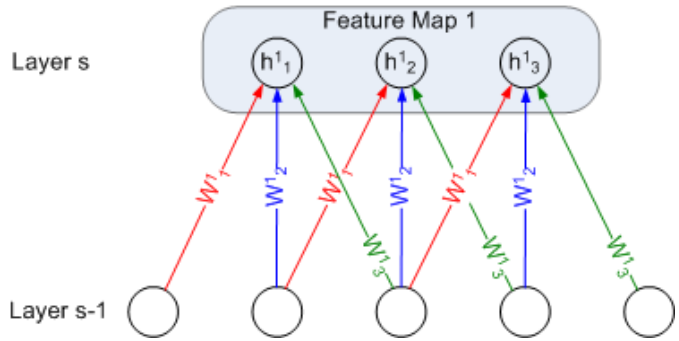


Concept of Convolution



Features, Feature Maps, Local Receptive Fields

- Filter coefficients of k.th feature f_k are the weights W^k between one neuron in the hidden layer and its local receptive field
- The local receptive field of a neuron are all neurons of the previous layer, that are connected to this neuron.
- For each feature f_k in the hidden layer there exists a feature map.
- The feature map contains as much hidden neurons as there are possible shifts of the filter across the input.
- Shared Weights: All neurons of a feature map have the same weight matrix W^k.
- Therefore: Low number of free parameters



- h_i^k is the $i.th$ output of the $k.th$ feature map in the first hidden layer
- $\mathbf{x}_i$ is the subregion of the input, which constitutes the local receptive field of h_i^k
- Output of corresponding hidden neuron:

$$h_i^k = g(W^k \cdot \mathbf{x}_i + b^k). \quad (1)$$

Single Feature Map at first hidden layer

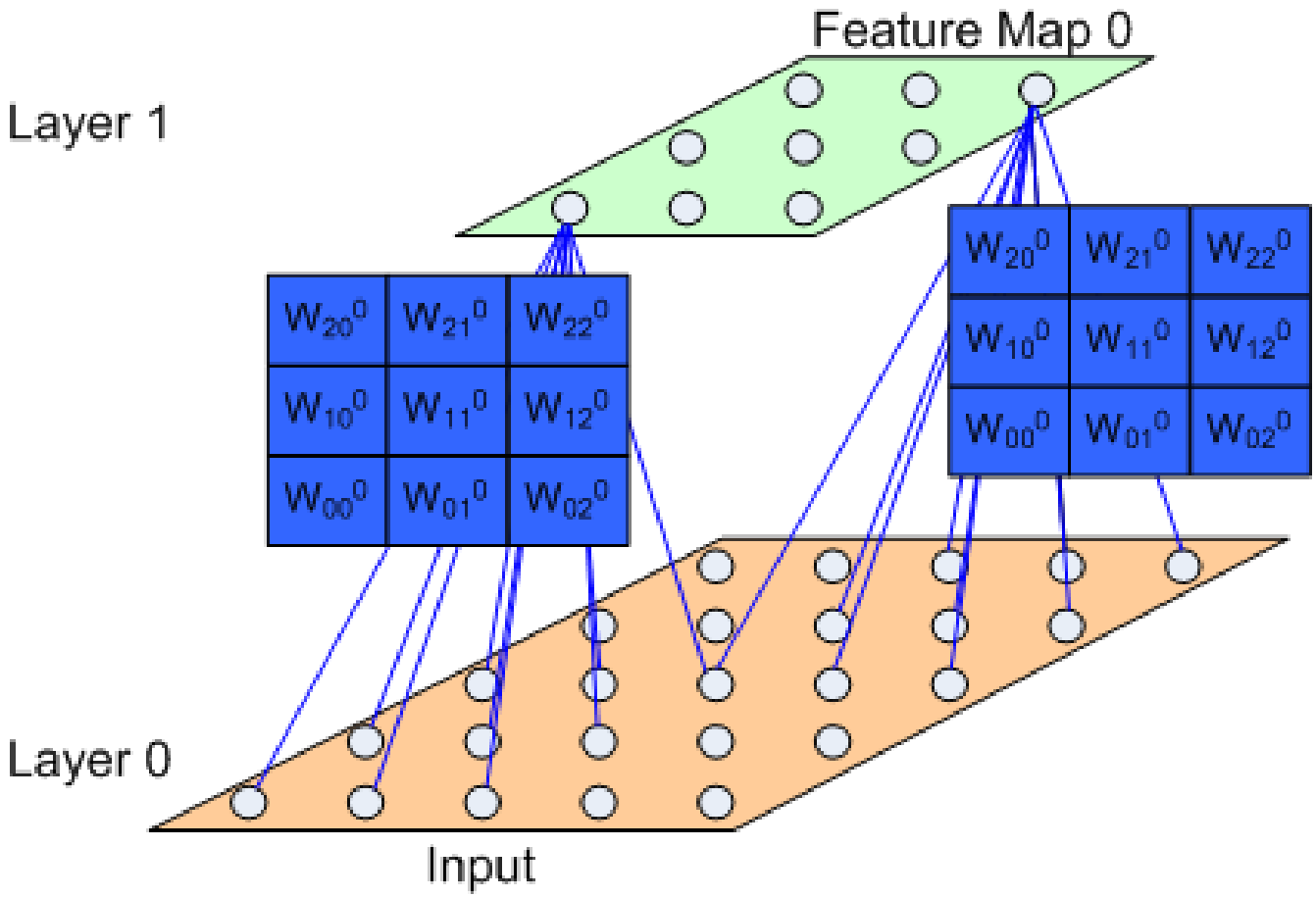


Figure: Input layer and first hidden layer of convolutional neural network corresponding to previous example. Only 2 out of the 9 hidden neurons and corresponding subregion connections are drawn. All hidden neurons in the feature map share the same weight matrix.

Multiple Feature Maps at arbitrary hidden layer

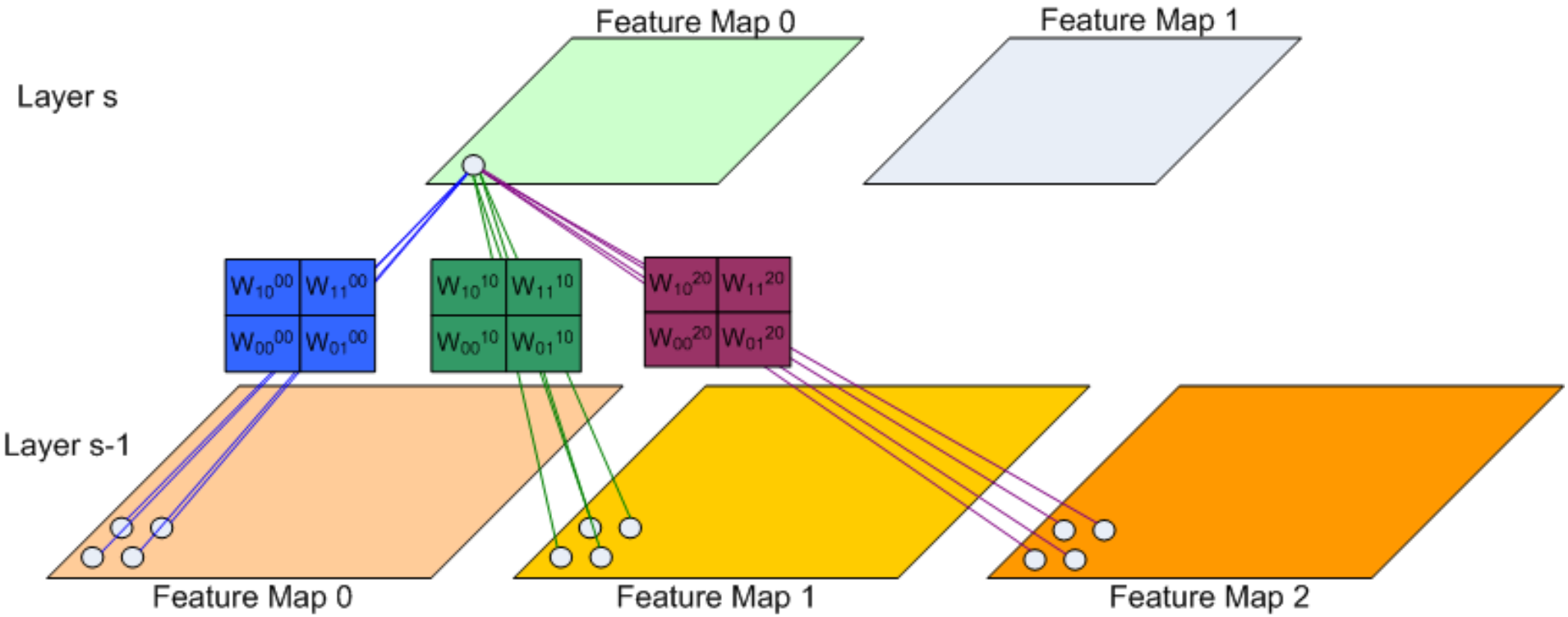


Figure: Excerpt of weights between the feature maps of two successive layers

Motivation for Pooling

Complexity

- Example: Classification of images
  - image size: 120×80
  - filter size: 10×10
  - number of features: 200
  - number of hidden layers: 1
  - $\Rightarrow$ 1.576200 inputs to the classifier

Invariance

- Image classification usually requires some degree of translational invariance, i.e. the classifier output shall be independent of small translational shifts in the image.

Pooling Operation

- Partition the entire region of N neurons in a feature map into a set of P non-overlapping subregions, each containing M neurons.
- Subsampling: For each of these subregions only one value is calculated from the M neurons in the convolutional layer:
  - Max-Pooling: calculate maximum of subregion
  - Min-Pooling: calculate minimum
  - Mean-Pooling: calculate mean

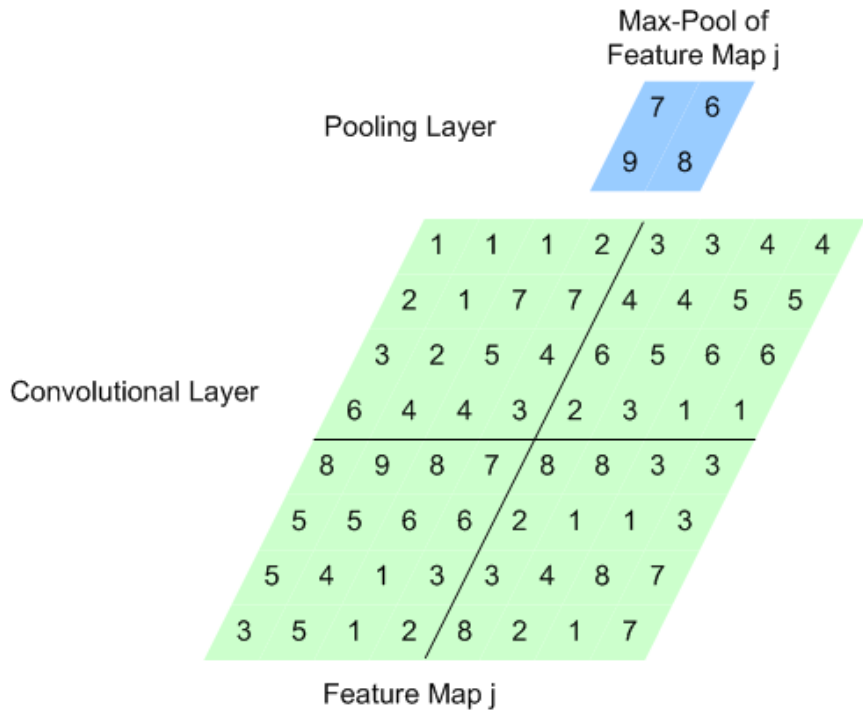


Figure: Max-Pooling of a feature map: Subsambling factor $= 16$

Convolution, Pooling and Classification in a CNN

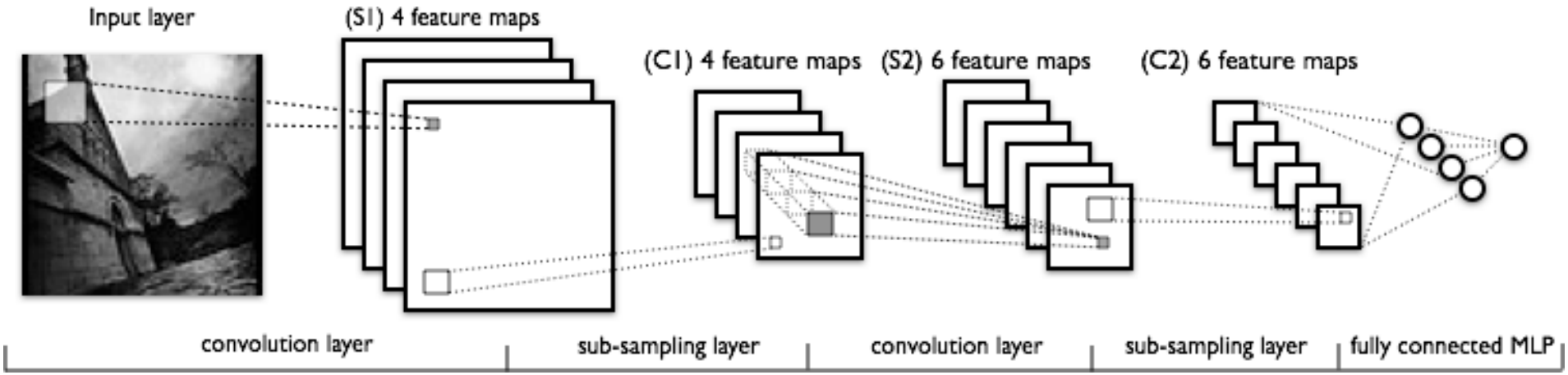


Figure: Combination of convolutional and pooling layers according to [LeCun et al., 2001] (Picture source [lab, 2011])

Configuration:

- Increase number of features with layer index, since number of neurons per feature map decreases.
- Pooling: Subregions of size (2×2) or (4×4)
- Convolution: Filter size depends on image size. E.g. (5×5) filter for (28×28) MNIST images.
- Training: Gradient descent such as backpropagation is efficient due to shared weights.

For pattern recognition CNNs belong to the best performing techniques. Strong Prior

Idea of Gradient Descent Learning

- Define Lossfunction $E(\mathbf{w})$, which somehow measures the difference between the target-output (r_p) and the current network output (y_p) for the given training input.

Idea of Gradient Descent Learning

- Define Lossfunction $E(\mathbf{w})$, which somehow measures the difference between the target-output (r_p) and the current network output (y_p) for the given training input.
- Loss-function for classification: Cross-Entropie:

$$E(\mathbf{w}_i) = -\sum_{p=1}^{N}\sum_{i=1}^{K} r_{p,i} \log y_{p,i}$$

- Best weights $\mathbf{w}$ are the ones which minimize the loss function.

Idea of Gradient Descent Learning

- Define Lossfunction $E(\mathbf{w})$, which somehow measures the difference between the target-output (r_p) and the current network output (y_p) for the given training input.
- Loss-function for classification: Cross-Entropie:

$$E(\mathbf{w}_i) = -\sum_{p=1}^{N}\sum_{i=1}^{K} r_{p,i} \log y_{p,i}$$

- Best weights $\mathbf{w}$ are the ones which minimize the loss function.
- Gradient of Loss-Function

$$\nabla E(\mathbf{w}) = \begin{pmatrix} \frac{\partial E}{\partial w_0} \\ \vdots \\ \frac{\partial E}{\partial w_d} \end{pmatrix}$$

points towards the steepest ascent.

Idea of Gradient Descent Learning

- Define Lossfunction $E(\mathbf{w})$, which somehow measures the difference between the target-output (r_p) and the current network output (y_p) for the given training input.
- Loss-function for classification: Cross-Entropie:

$$E(\mathbf{w}_i) = -\sum_{p=1}^{N}\sum_{i=1}^{K} r_{p,i} \log y_{p,i}$$

- Best weights $\mathbf{w}$ are the ones which minimize the loss function.
- Gradient of Loss-Function

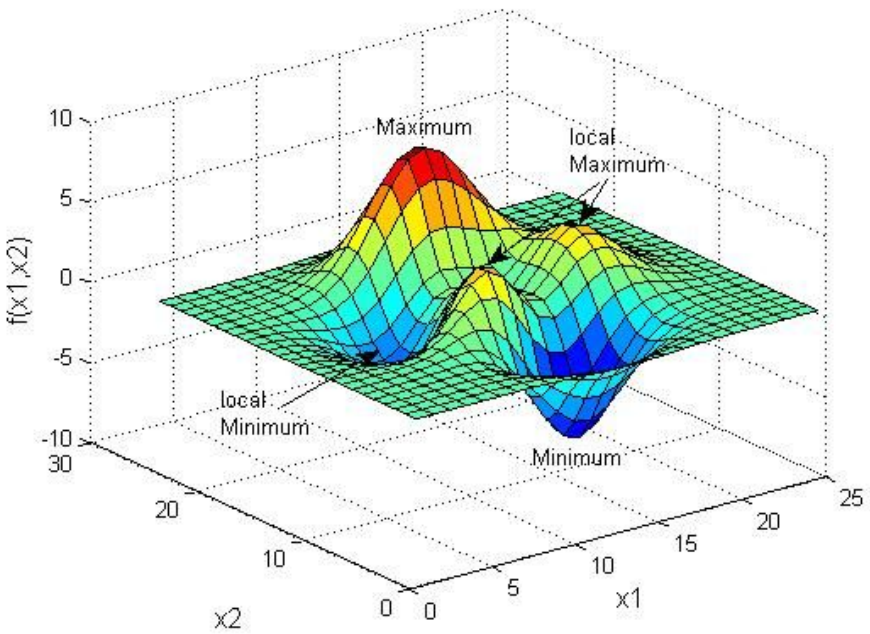
$$\nabla E(\mathbf{w}) = \begin{pmatrix} \frac{\partial E}{\partial w_0} \\ \vdots \\ \frac{\partial E}{\partial w_d} \end{pmatrix}$$

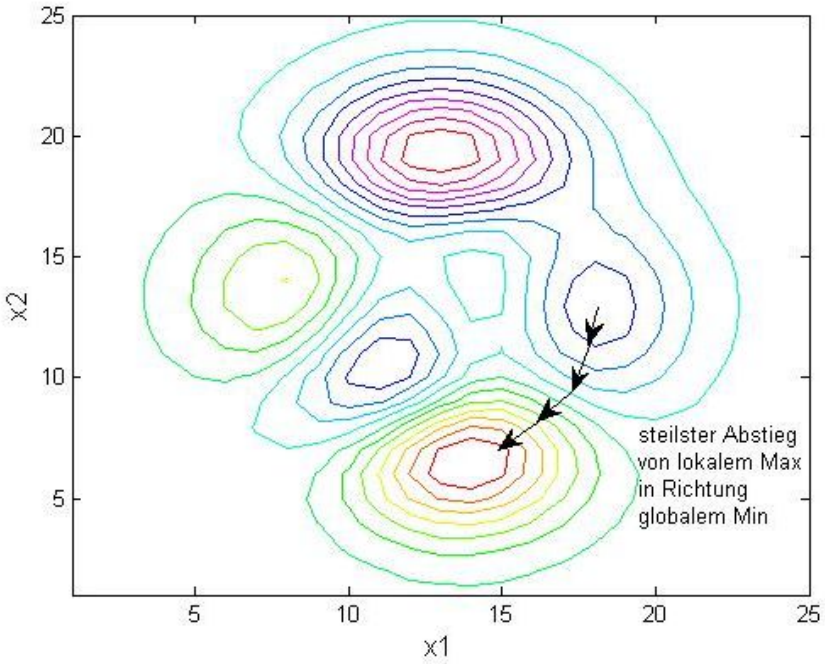
  points towards the steepest ascent.
- $-\nabla E(\mathbf{w})$ points towards steepest descent.
- Iteratively adapt weights w_j proportional to partial derivative of E w.r.t. w_j:

$$w_j = w_j + \Delta w_j = w_j + \eta \cdot -\frac{\partial E}{\partial w_j}$$

- Learningrate η controls step-size of weight-adaptations.

Idea of Gradient Descent





Additional Parameters for Gradient Descent

Minibatches: Calculate Loss-function $E(\mathbf{w})$ and adapt weights not over entire set of training data (Batch-Learning), but per minibatches (partitions of entire training data).

Learnrate Decay γ: Decrease Learning Rate with increasing time t:

$$\eta^t = \gamma \cdot \eta^{t-1}$$

Momentum α: Avoid Ping-Pong-Effects by adding previous weight-adaptation:

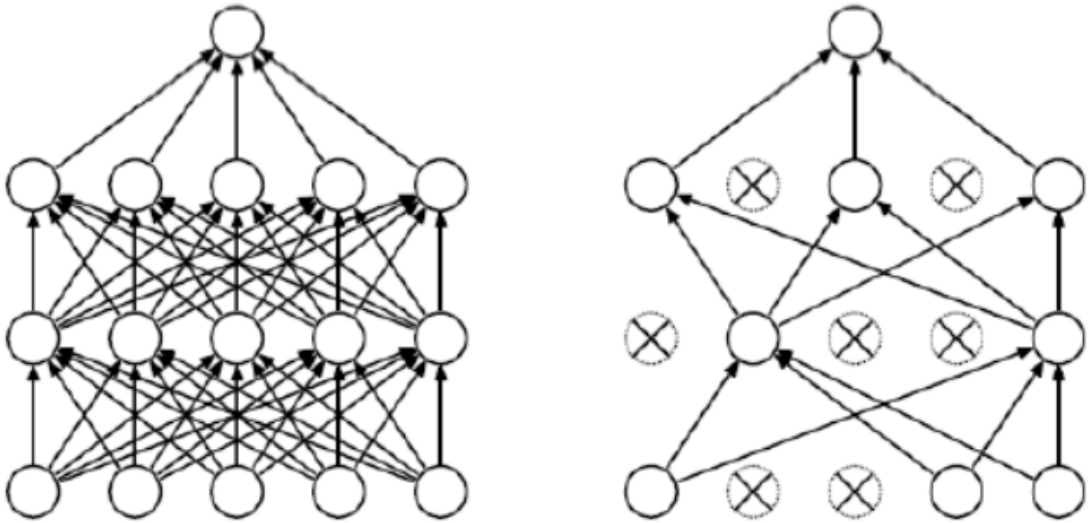
$$\Delta w_{j,i}^t = -\eta \cdot \frac{\partial E}{\partial w_{j,i}} + \alpha \Delta w_{j,i}^{t-1}$$

Weight Decay β: Simultanously minimize loss-function E and weights $\mathbf{w}$ by adding Regularisation Term to $E(\mathbf{w})$:

$$\beta \cdot \frac{1}{2} \sum_{w \in W} w^2$$

Yields better gerneralisation and less overfitting.

Avoid Overfitting by applying dropout in training

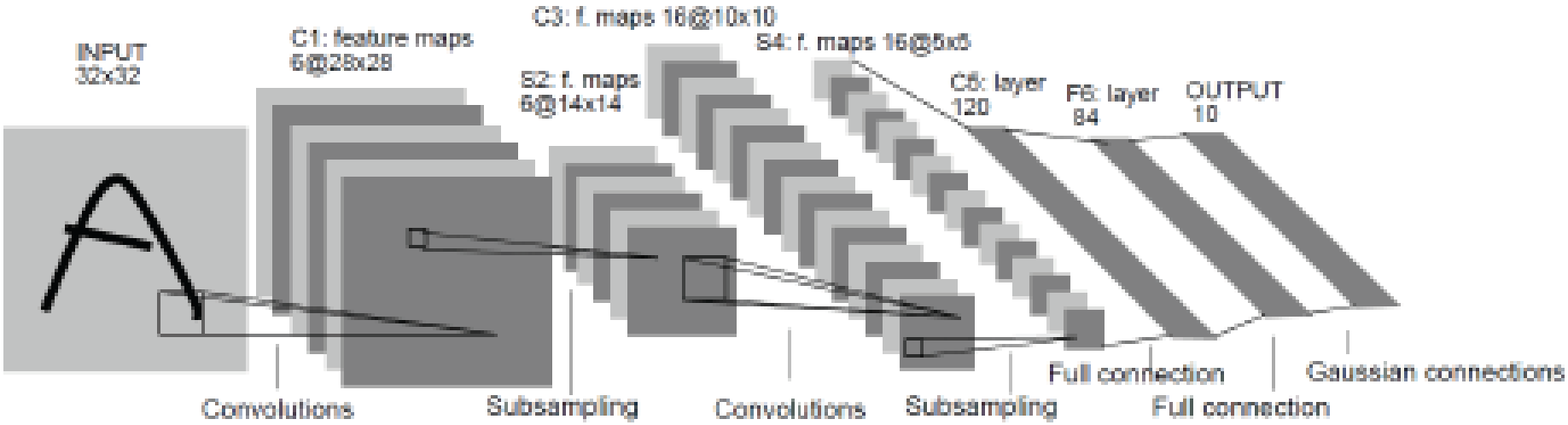


(a) Standard Neural Net

(b) After applying dropout.

- In each Iteration weights of randomly selected connections are temporarily set to zero.
- After the iterartion the previous weights are applied
- Reduces overfitting.

Example Architectures: LeNet-5

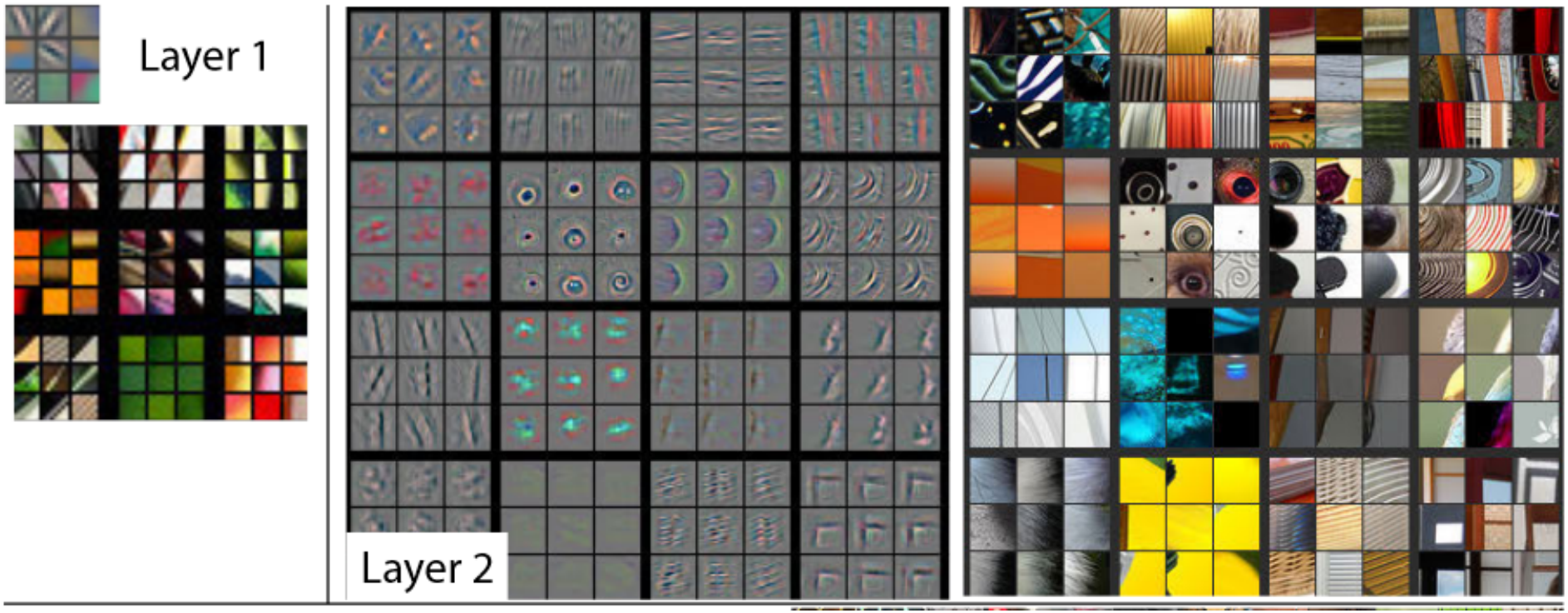


Architecture Notation:

$$\text{5-1Conv6} \rightarrow \text{2-2AvgPool} \rightarrow \text{5-1Conv16} \rightarrow \text{5-5AvgPool} \rightarrow \text{FC120} \rightarrow \text{FC 84} \rightarrow \text{FC10}$$

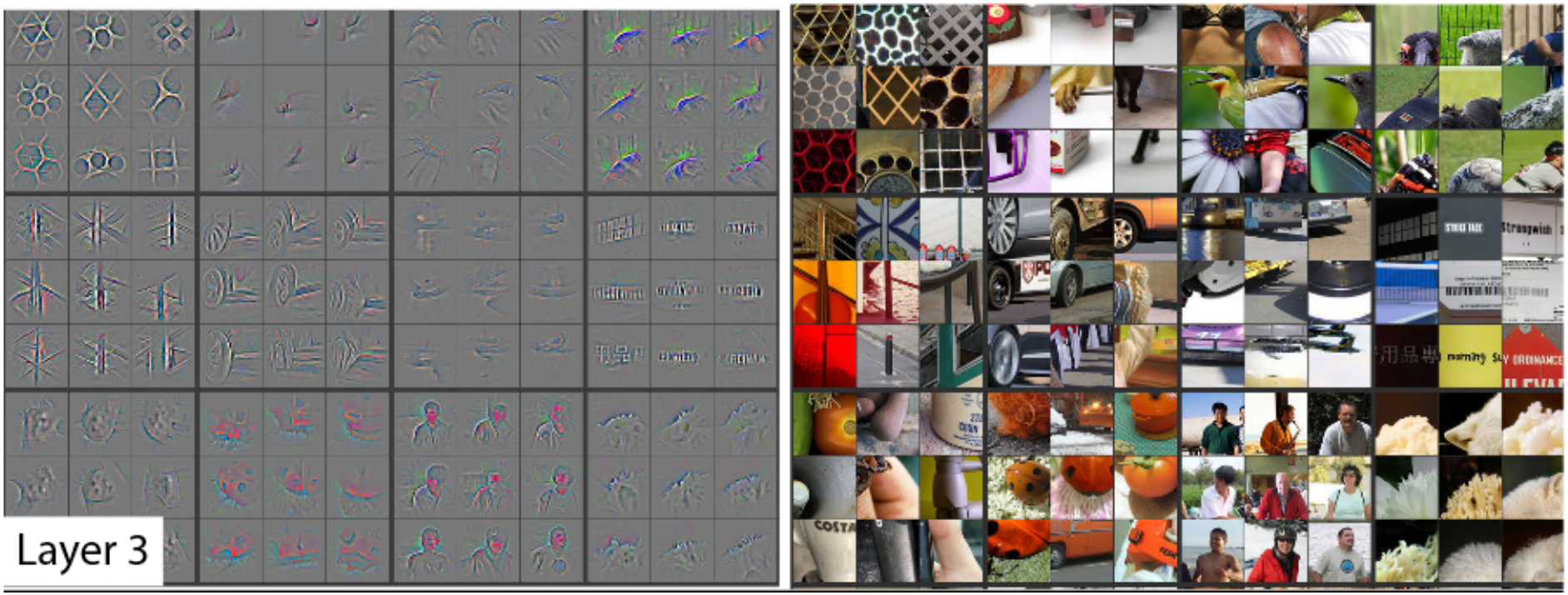
- *X-YConvZ*: Convolutional Layer with filter size *XxX*, stepwidth *Y* and *Z* feature maps
- *X-XAvgPoolY*: Average Pooling of size *XxX* and *Y* outputs.
- *FCX*: Fully connected layer with *X* outputs.

Feature representation in CNN layers



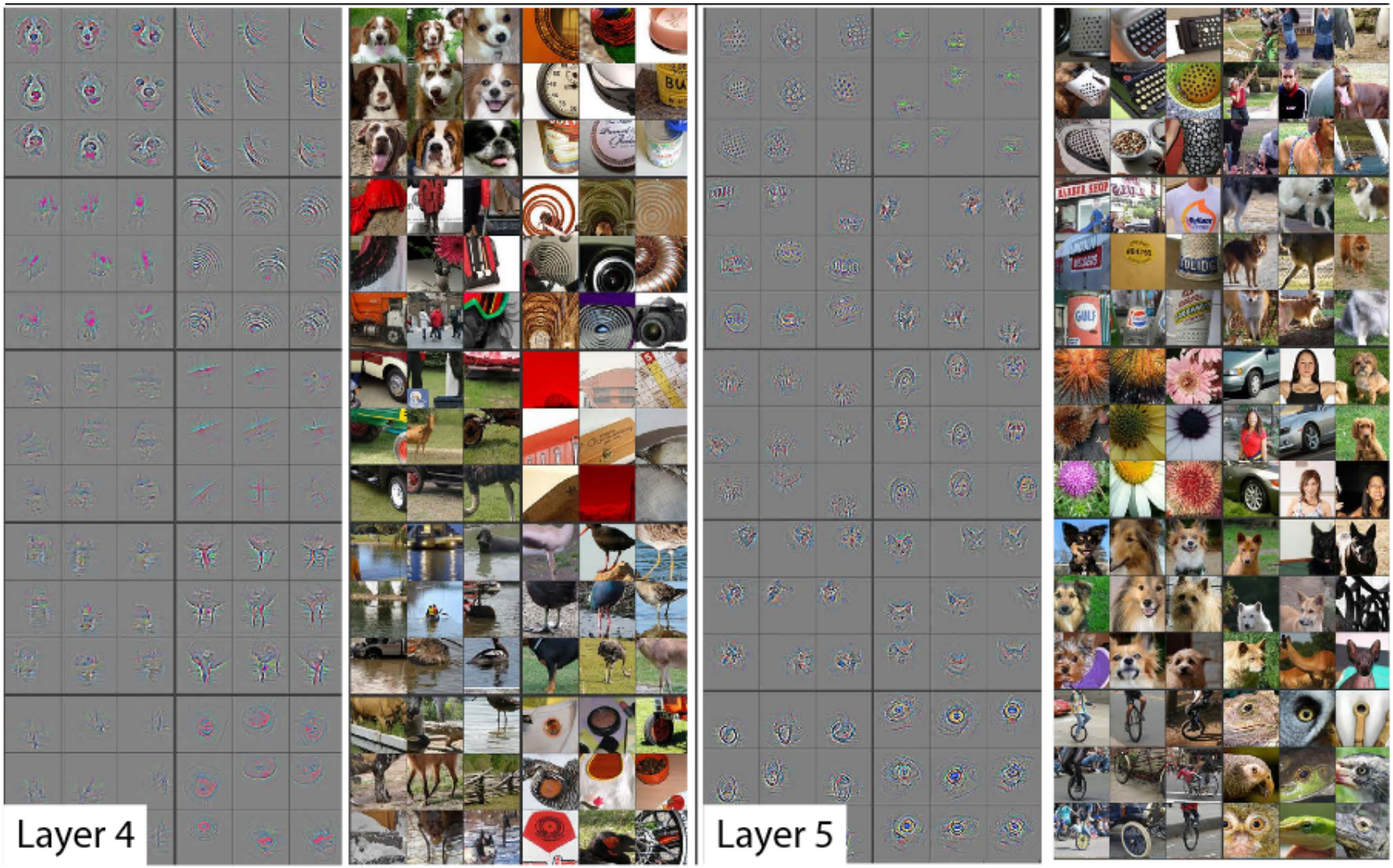
Quelle: [Matthew D. Zeiler, 2014]
http://www.matthewzeiler.com/pubs/arxive2013/arxive2013.pdf

Feature representation in CNN layers



Quelle: [Matthew D. Zeiler, 2014]
http://www.matthewzeiler.com/pubs/arxive2013/arxive2013.pdf

Feature representation in CNN layers



Quelle: [Matthew D. Zeiler, 2014]
http://www.matthewzeiler.com/pubs/arxive2013/arxive2013.pdf

ImageNet Large Scale Visual Recognition Challenge

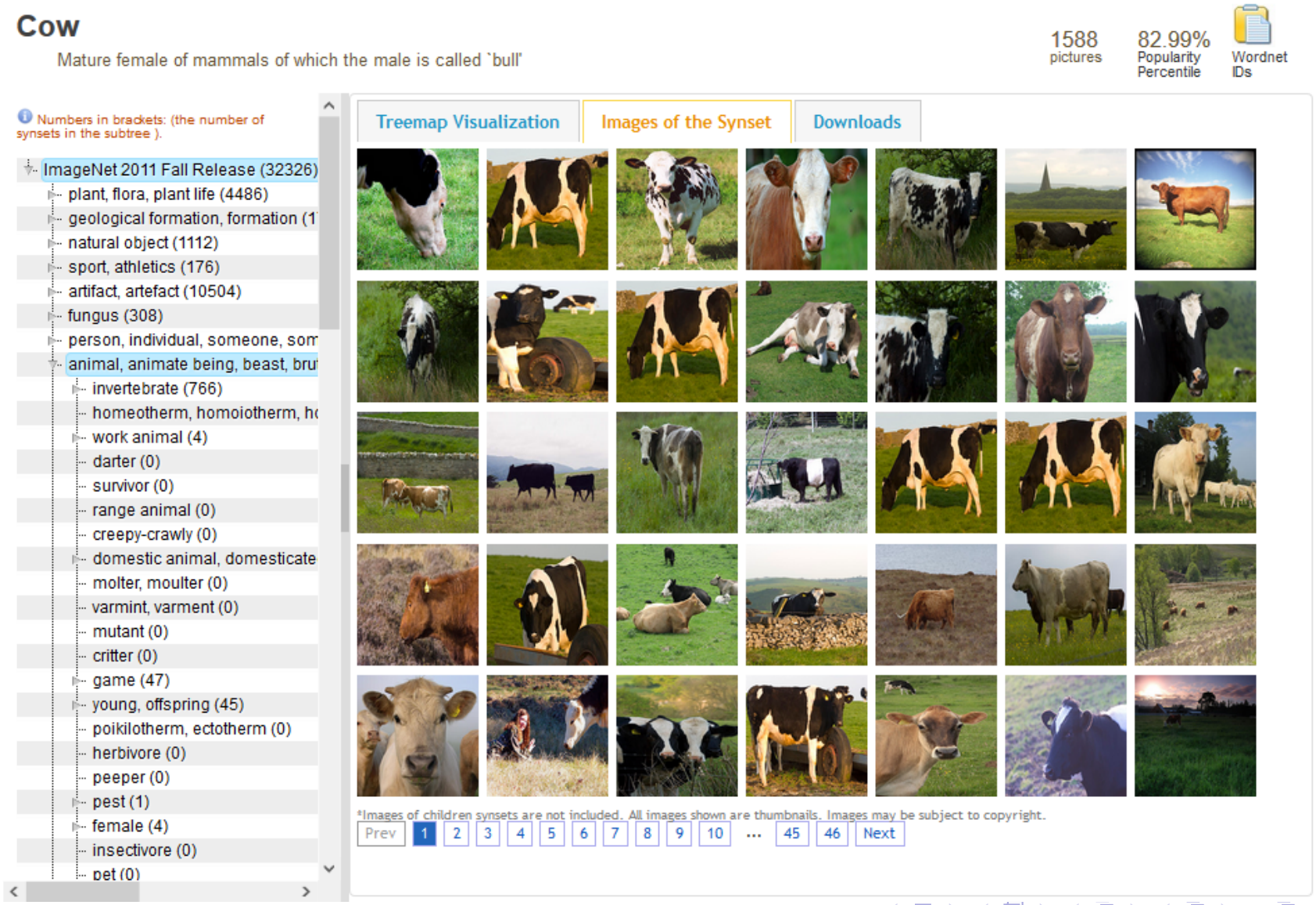
- Evaluates algorithms for object detection and image classification at large scale
  - Detection challenge on fully labeled data for 200 categories of objects
  - Image classification challenge with 1000 categories
  - Image classification plus object localization challenge with 1000 categories
- Training data is a subset of ImageNet data-
- Started in 2010

Example: ImageNet-Net Taxonomy

ImageNet is an image database organized according to the WordNet hierarchy, in which each node of the hierarchy is depicted by hundreds and thousands of images. (15 million images in 22000 categories).

- S: (n) Eskimo dog, **husky** (breed of heavy-coated Arctic sled dog)
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) working dog (any of several breeds of usually large powerful dogs bred to work as draft animals and guard and guide dogs)
      - S: (n) dog, domestic dog, Canis familiaris (a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds) *"the dog barked all night"*
        - S: (n) canine, canid (any of various fissiped mammals with nonretractile claws and typically long muzzles)
          - S: (n) carnivore (a terrestrial or aquatic flesh-eating mammal) *"terrestrial carnivores have four or five clawed digits on each limb"*
            - S: (n) placental, placental mammal, eutherian, eutherian mammal (mammals having a placenta; all mammals except monotremes and marsupials)
              - S: (n) mammal, mammalian (any warm-blooded vertebrate having the skin more or less covered with hair; young are born alive except for the small subclass of monotremes and nourished with milk)
                - S: (n) vertebrate, craniate (animals having a bony or cartilaginous skeleton with a segmented spinal column and a large brain enclosed in a skull or cranium)
                  - S: (n) chordate (any animal of the phylum Chordata having a notochord or spinal column)
                    - S: (n) animal, animate being, beast, brute, creature, fauna (a living organism characterized by voluntary movement)
                      - S: (n) organism, being (a living thing that has (or can develop) the ability to act or function independently)
                        - S: (n) living thing, animate thing (a living (or once living) entity)
                          - S: (n) whole, unit (an assemblage of parts that is regarded as a single entity) *"how big is that part compared to the whole?"; "the team is a unit"*
                            - S: (n) object, physical object (a tangible and visible entity; an entity that can cast a shadow) *"it was full of rackets, balls and other objects"*
                              - S: (n) physical entity (an entity that has physical existence)
                                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or nonliving))

Example: ImageNet-Net result for query cow

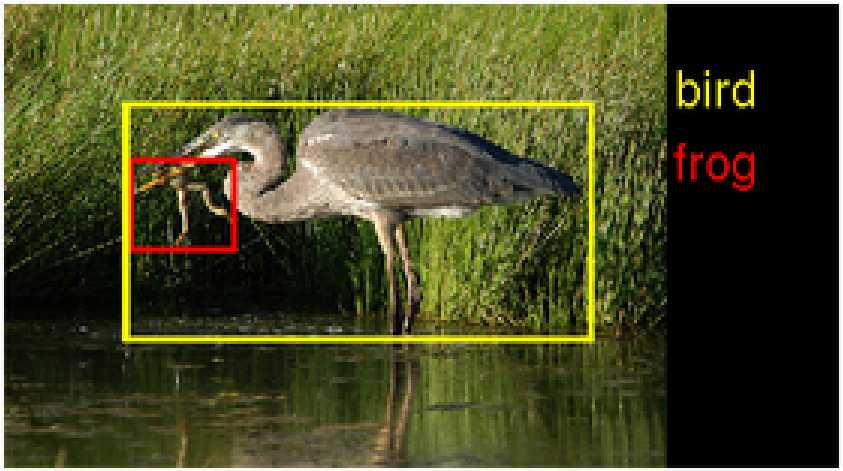


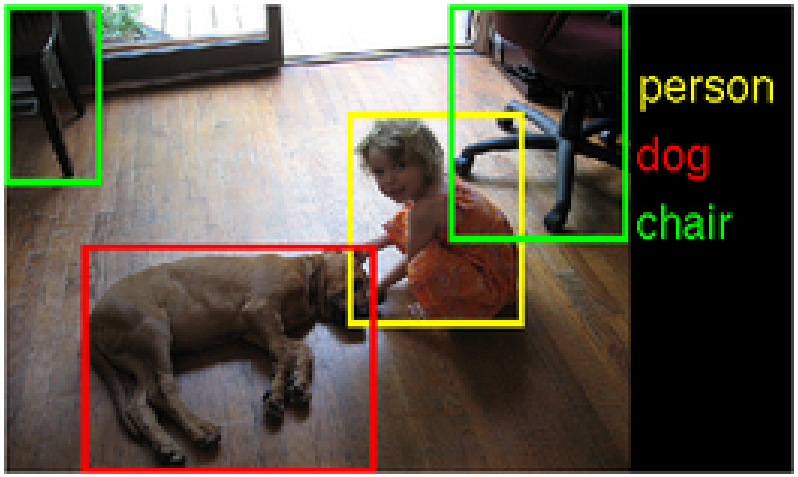
Detection Challenge

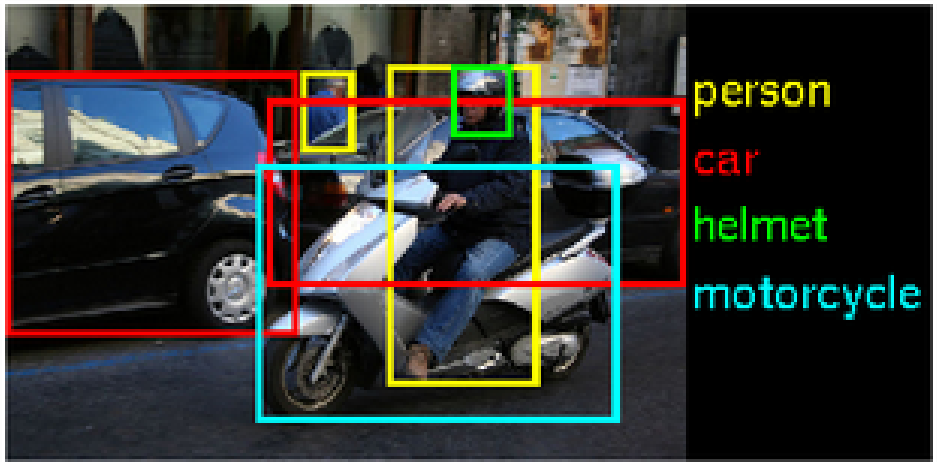
As specified in ILSVRC 2014

- 200 object categories
- Training: 456567 images with 478807 objects from ImageNet image-net.org/
- Validation: 20121 images with 55502 objects from flickr and other search engines
- Test: 40152 images from flickr and other search engines
- Average image resolution in validation data: 482×415 pixels
- Task: For each image, algorithms will produce a set of annotations (c_i, b_i, s_i) of class labels c_i, bounding boxes b_i and confidence scores s_i.

Detection Challenge: Example Images







Source: http://image-net.org/challenges/LSVRC/2014/index

Classification and localisation challenge

- 1000 object categories
- Training 1.2 Million images from ImageNet image-net.org/
- Validation: 50 000 images from flickr and other search engines
- Test: 100 000 images from flickr and other search engines
- Task: For each image, algorithms will produce 5 class labels c_i, $i \in \{1, \ldots, 5\}$ in decreasing order of confidence and 5 bounding boxes b_i, $i \in \{1, \ldots, 5\}$, one for each class label.
- Top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.
- In the localisation-task object-category and bounding boxes must match. Bounding boxes are defined to match, if they have an overlap of $> 50\%$.

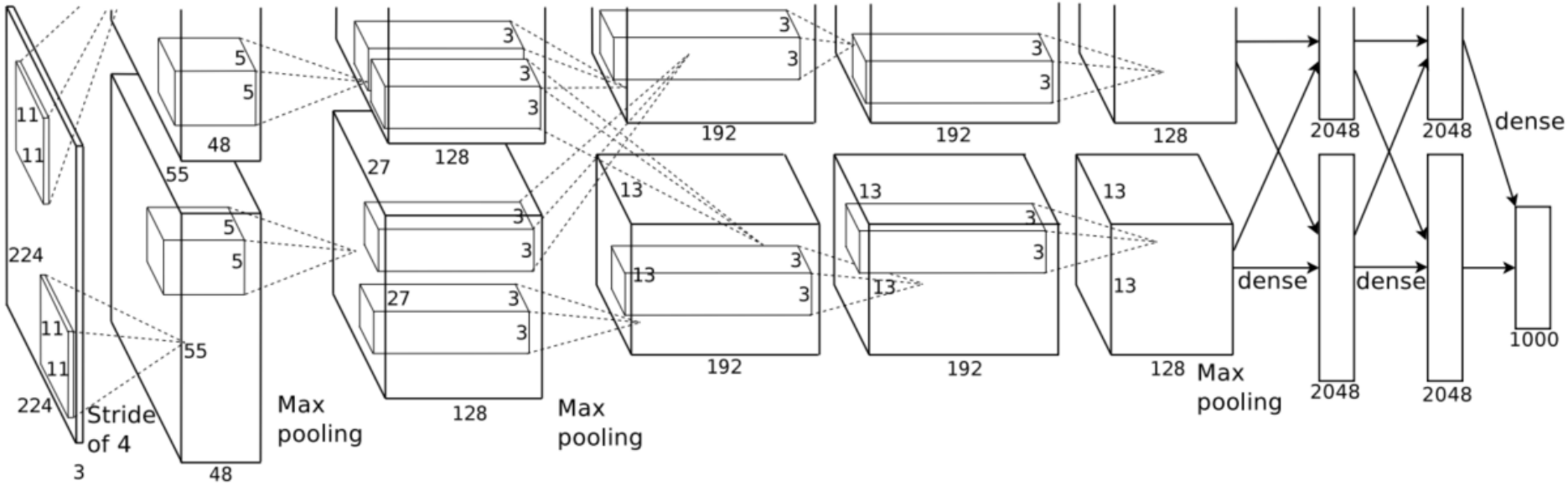
AlexNet

- Introduced in [Krizhevsky et al.,] (One of the milestone-papers in Object Recognition)
- Won ILSVRC 2012 classification- and localisation task
- Top-5 classification error of 15.4% (next best achieved 26.2%).
- Key elements:
  - ReLu activations decrease training time
  - Local Response Normalisation after ReLu
  - Overlapped Pooling regions
  - Reduce overfitting by
    - Data augmentation
    - Dropout
  - Training: 15 million annotated images; duration 6 days; two GTX 580 3GB GPUs

Cropping and preprocessing of training data

- ImageNet consists of variable-resolution images
- AlexNet requires images of size 256×256.
- rectangular image is first rescaled such that shorter side is of length 256
- Then the central 224×224 patch is cropped from the resulting image.
- Only preprocessing routine is subtraction of the mean value over the training set from each pixel.

Architecture



Source: [Krizhevsky et al.,]

11-4Conv96 (ReLU) → Norm → 3-2MaxPool → 5-1-2Conv256 (ReLU) → Norm → 3-2MaxPool → 3-1-1Conv384 (ReLU) → 3-1-1Conv384 (ReLU) → 3-1-1Conv256 (ReLU) → 3-2MaxPool → FC4096 (Dropout, ReLU) → FC4096 (Dropout, ReLU) → FC1000

Architecture

- Overall architecture is distributed to 2 GPUs
- Feature maps in the 2nd, 4th and 5th conv-layer are connected only to the feature maps of the previous layer on the same GPU.
- ReLu-Activation in all conv- and fc-layers.
- Response-Normalisation after ReLU in 1st and 2nd conv-layer
- Overlapping 3×3 Max-Pooling with stride length 2 after 1st, 2nd and 5th layer.

Reduce Overfitting by Data Augmentaion

- Artifically enlarge training dataset by label-preserving transformations
- Transformations are processed in Python code on CPU. Artificially generated images need not be stored.
  1. Image translations and horizontal reflections:
     - Random extraction of 224 $\times$ 224 - patches and their horizontal flips from 256 $\times$ 256 images
       $\Rightarrow$ Increases number of training images by a factor of 2048
  2. Altering intensities of RGB channels in training images:
     - Provides more robustness w.r.t. color changes

Reduce Overfitting by Dropout

- During training the output of each hidden neuron is temporarily (for the current iteration) set to zero with a propability of 0.5. Such dropped out neurons are not considered in the iteration's forward- and backward-pass.
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons
- Provides more robust features
- In AlexNet dropout is applied in the first 2 FC-layers.
- Increases training convergence time by a factor of 2.

Training

- Optimisation of multinomial logistic regression by mini-batch gradient descent (backpropagation).
- Mini batch size: 128
- Momentum: 0.9
- Regularisation: 0.0005 on L_2
- Dropout regularisation on first 2 fully-connected layers with dropout-rate 0.5.
- learning rate initally: 0.01.
- Initial weights: Drawn from zero-mean Gaussian distribution and standard-deviation of 0.01.
- Learning stopped after 90 epochs.

Testing

- From each test - image 10 versions are created:
  - the four 224 $\times$ 224 corner crops and the center crop
  - from each of the 5 crops above the horizontal flip
- For the 10 versions the corresponding output of the softmax-layer is calculated.
- Select class which yields highest average output over the 10 versions.
- No multi-scale classification as in *OverFeat* [Sermanet et al.,]

Feature quality and classification examples

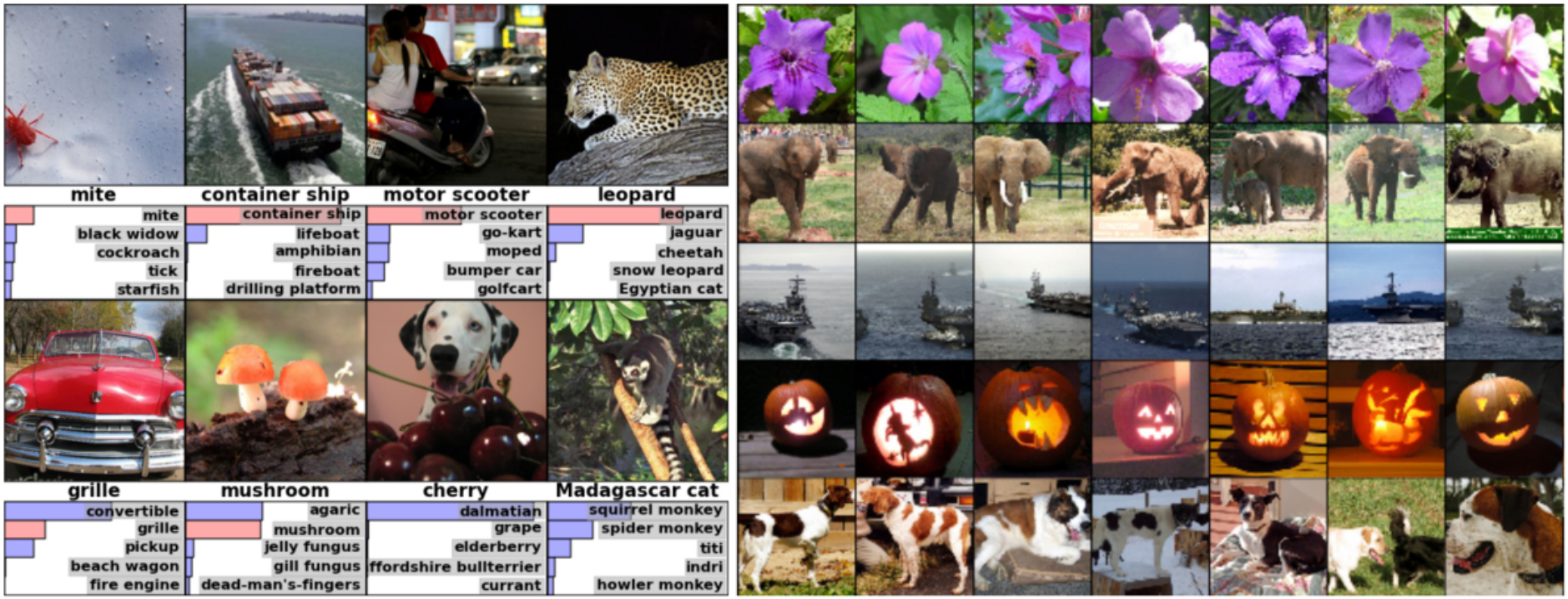


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Source: [Krizhevsky et al.,]

OverFeat

- Introduced in [Sermanet et al.,]
- Integrated Recognition, Localisation and Detection using CNNs
- Won ILSVRC 2013 localisation task and obtained top-results on classification and on detection task.
- Mult-task learning using a single shared network.
- Multi-scale classification

Architecture

In training this architecture is applied in a non-spatial manner. I.e. the 1000 outputs predict the class-presence, but no spatial information. However, in the classification phase spatial outputs are produced.

Layer	1	2	3	4	5	6	7	Output 8
Stage	conv + max	conv + max	conv	conv	conv + max	full	full	full
# channels	96	256	512	1024	1024	3072	4096	1000
Filter size	11x11	5x5	3x3	3x3	3x3	-	-	-
Conv. stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling size	2x2	2x2	-	-	2x2	-	-	-
Pooling stride	2x2	2x2	-	-	2x2	-	-	-
Zero-Padding size	-	-	1x1x1x1	1x1x1x1	1x1x1x1	-	-	-
Spatial input size	231x231	24x24	12x12	12x12	12x12	6x6	1x1	1x1

Source: [Sermanet et al.,]

Differences to AlexNet:

- No normalization after ReLu
- Pooling regions do not overlap
- Larger feature maps in layer 3, 4 and 5

Training

- Training is done in the same way as in AlexNet ([Krizhevsky et al.,]), with slightly different values for learning-rate, weight-decay and momentum.

General concept of Sliding Window implementation in CNNs

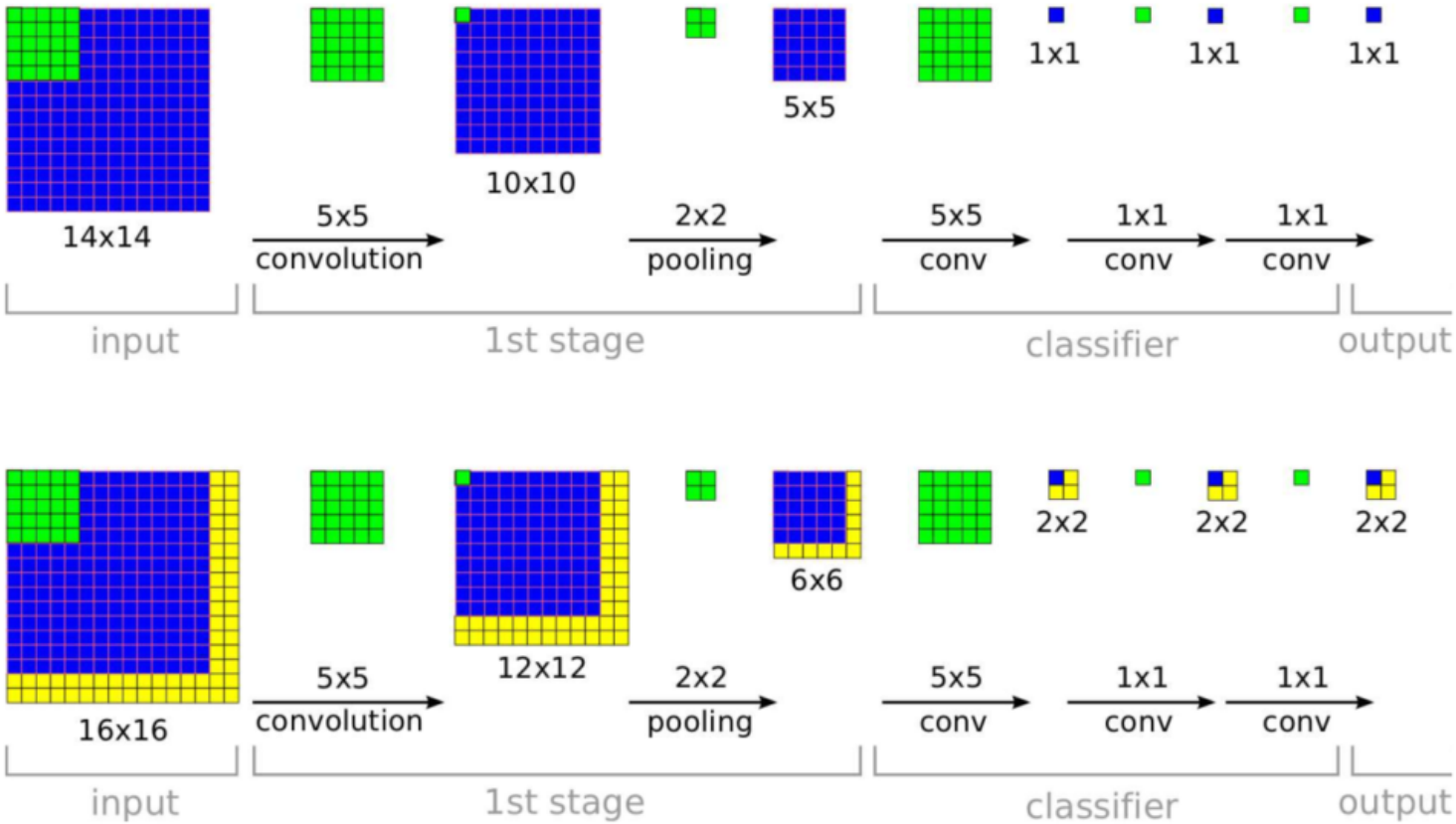


Figure 5: **The efficiency of ConvNets for detection.** During training, a ConvNet produces only a single spatial output (top). But when applied at test time over a larger image, it produces a spatial output map, e.g. 2x2 (bottom). Since all layers are applied convolutionally, the extra computation required for the larger image is limited to the yellow regions. This diagram omits the feature dimension for simplicity.

Source: [Sermanet et al.,]

Scales applied in the classification-phase

Scale	Input size	Layer 5 pre-pool	Layer 5 post-pool	Classifier map (pre-reshape)	Classifier map size
1	245x245	17x17	(5x5)x(3x3)	(1x1)x(3x3)xC	3x3xC
2	281x317	20x23	(6x7)x(3x3)	(2x3)x(3x3)xC	6x9xC
3	317x389	23x29	(7x9)x(3x3)	(3x5)x(3x3)xC	9x15xC
4	389x461	29x35	(9x11)x(3x3)	(5x7)x(3x3)xC	15x21xC
5	425x497	32x35	(10x11)x(3x3)	(6x7)x(3x3)xC	18x24xC
6	461x569	35x44	(11x14)x(3x3)	(7x10)x(3x3)xC	21x30xC

Table 5: **Spatial dimensions of our multi-scale approach**. 6 different sizes of input images are used, resulting in layer 5 unpooled feature maps of differing spatial resolution (although not indicated in the table, all have 256 feature channels). The (3x3) results from our dense pooling operation with $(\Delta_x, \Delta_y) = \{0, 1, 2\}$. See text and Fig. 3 for details for how these are converted into output maps.

Source: [Sermanet et al.,]

Layer-5 Pooling and generation of 3D-output map in classification phase

General Idea

- For an image of arbitrary scale apply conv-layers on entire image
- Resolution (number of neurons) in the conv-layers varies with varying resolution of input image
- In the last conv-layer apply a specific pooling in the classification phase
- The classifier (i.e. the fully connected layers) is convolved over the pooled-output of the last conv-layer.
- The output is a 3D-map, whose first 2 dimensions indicate position and the 3rd dimension indicates class-probabilities. I.e. the number of elements in the 3rd dimension is the number of classes C.

Convolutionalisation / Fully Convolutional Network (FCN)

- The process of applying the trained CNN on larger images and convolving the classifier (FC-layers) over the (pooled) output of the last conv-layer is called Convolutionalisation
- The corresponding net is often called Fully Convolutional Network ([Shelhamer et al., 2016])

Layer-5 Pooling and generation of 3D-output map in classification phase

Concrete: from [Sermanet et al.,]

1. For a single image, at a given scale, start with the unpooled layer 5 feature maps
2. Each of the unpooled maps undergoes a 3×3 max-pooling operation (non-overlapping regions), repeated 3×3 times for (Δ_x, Δ_y) pixel offsets of $\{0, 1, 2\}$.
3. This produces a set of pooled feature maps, replicated (3×3) times for different (Δ_x, Δ_y) combinations.
4. The classifier (layers 6,7,8) has a fixed input size of 5×5 and produces a C-dimensional output vector for each location within the pooled maps. The classifier is applied in sliding-window fashion to the pooled maps, yielding C-dimensional output maps (for a given (Δ_x, Δ_y) combination).
5. The output maps for different (Δ_x, Δ_y) combinations are reshaped into a single 3D output map (two spatial dimensions $\times C$ classes).

1-D demo of layer-5 pooling applied in the classification phase

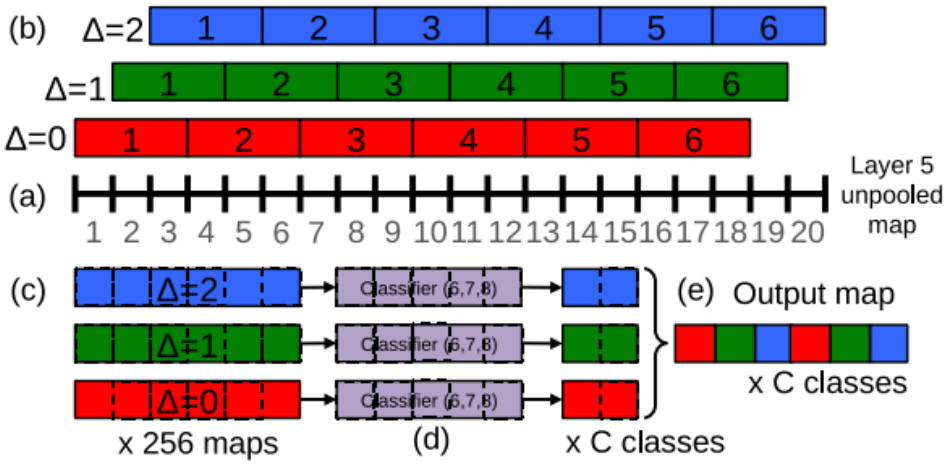


Figure 3: 1D illustration (to scale) of output map computation for classification, using y-dimension from scale 2 as an example (see Table 5). (a): 20 pixel unpooled layer 5 feature map. (b): max pooling over non-overlapping 3 pixel groups, using offsets of $\Delta = \{0, 1, 2\}$ pixels (red, green, blue respectively). (c): The resulting 6 pixel pooled maps, for different Δ. (d): 5 pixel classifier (layers 6,7) is applied in sliding window fashion to pooled maps, yielding 2 pixel by C maps for each Δ. (e): reshaped into 6 pixel by C output maps.

Source: [Sermanet et al.,]

Calculate class predictions

1. Calculate 3D-Output map for each scale and their horizontally flipped versions as described above.
2. For each scale and flip: Take spatial max for each class
3. Calculate average C-dimensional vector over all scales and flips
4. Take the top-1 or top-5 elements of this vector as the predicted classes

Localization

- Start from the CNN trained on the classification task
- Replace the FC-layers by new FC-layers for regression
- Train regression FC-layers to predict the object bounding boxes and keep conv-layers frozen.
- Combine the regression predictions along with the classification predictions at each location and scale.

Localization: Training of FC-layers for regression

- Input: Pooled feature maps from layer 5.
- As with classification, there are 3×3 copies, resulting from the (Δ_x, Δ_y) shifts.
- 2 FC-layers of size 4096 and 1024, respectively.
- Output: 4 units which specify the coordinates of the bounding-box
- Regression layers are trained by minimizing L_2-loss between predicted and true bounding box.
- The final regressor layer is class-specific, having 1000 different versions, one for each class.
- Regressors are trained in a mult-scale manner

Bounding-Box Regression ([Sermanet et al.,])

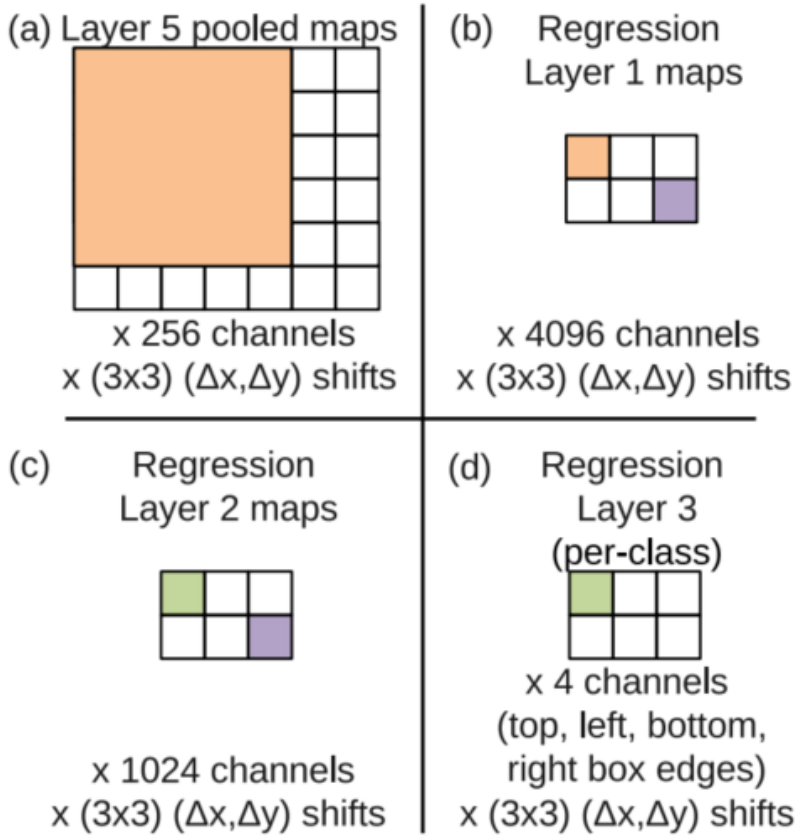


Figure 8: Application of the regression network to layer 5 features, at scale 2, for example. (a) The input to the regressor at this scale are 6x7 pixels spatially by 256 channels for each of the (3x3) Δ_x, Δ_y shifts. (b) Each unit in the 1st layer of the regression net is connected to a 5x5 spatial neighborhood in the layer 5 maps, as well as all 256 channels. Shifting the 5x5 neighborhood around results in a map of 2x3 spatial extent, for each of the 4096 channels in the layer, and for each of the (3x3) Δ_x, Δ_y shifts. (c) The 2nd regression layer has 1024 units and is fully connected (i.e. the purple element only connects to the purple element in (b), across all 4096 channels). (d) The output of the regression network is a 4-vector (specifying the edges of the bounding box) for each location in the 2x3 map, and for each of the (3x3) Δ_x, Δ_y shifts.

Combine Class- and Bounding-Box predictions

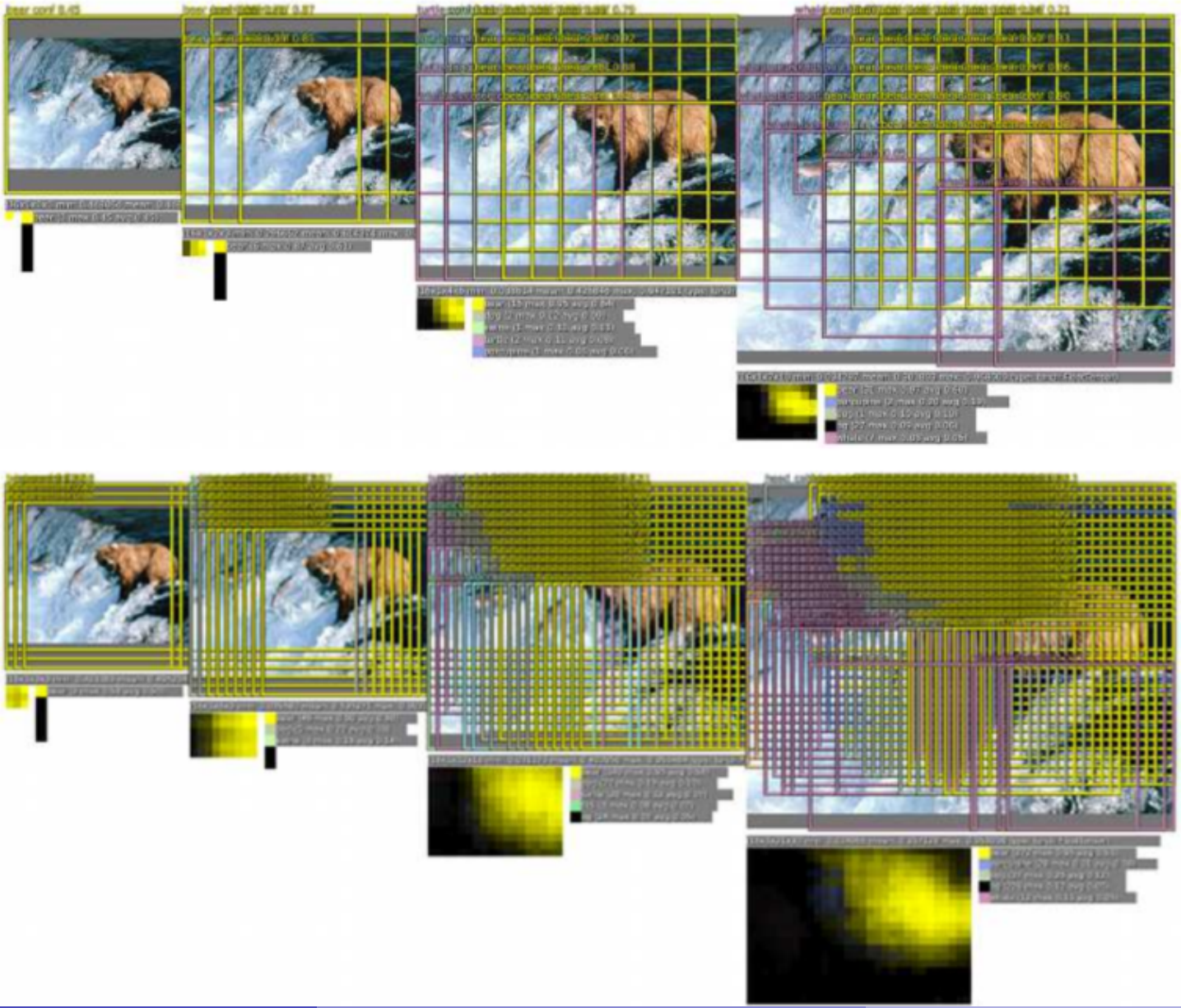
1. Assign to C_s the set of classes in the top k for each scale $s \in \{1, \ldots, 6\}$ found by taking the maximum detection class outputs across spatial locations for that scale
2. Assign to B_s the set of bounding boxes predicted by the regressor network for each class in C_s, across all spatial locations at scale s
3. Set $B = \cup_s B_s$
4. Repeat merging until done:
   1. $(b_1^*, b_2^*) = argmin_{b_1 \neq b_2 \in B} matchScore(b_1, b_2)$
   2. If $matchScore(b_1^*, b_2^*) > t \Rightarrow stop$
   3. Otherwise: Set $B \setminus \{b_1^*, b_2^*\} \cup boxMerge(b_1^*, b_2^*)$

- *matchScore* computes the sum of the distance between centers of the two bounding boxes and the intersection area of the boxes
- *boxMerge* computes the average of the bounding boxes' coordinates

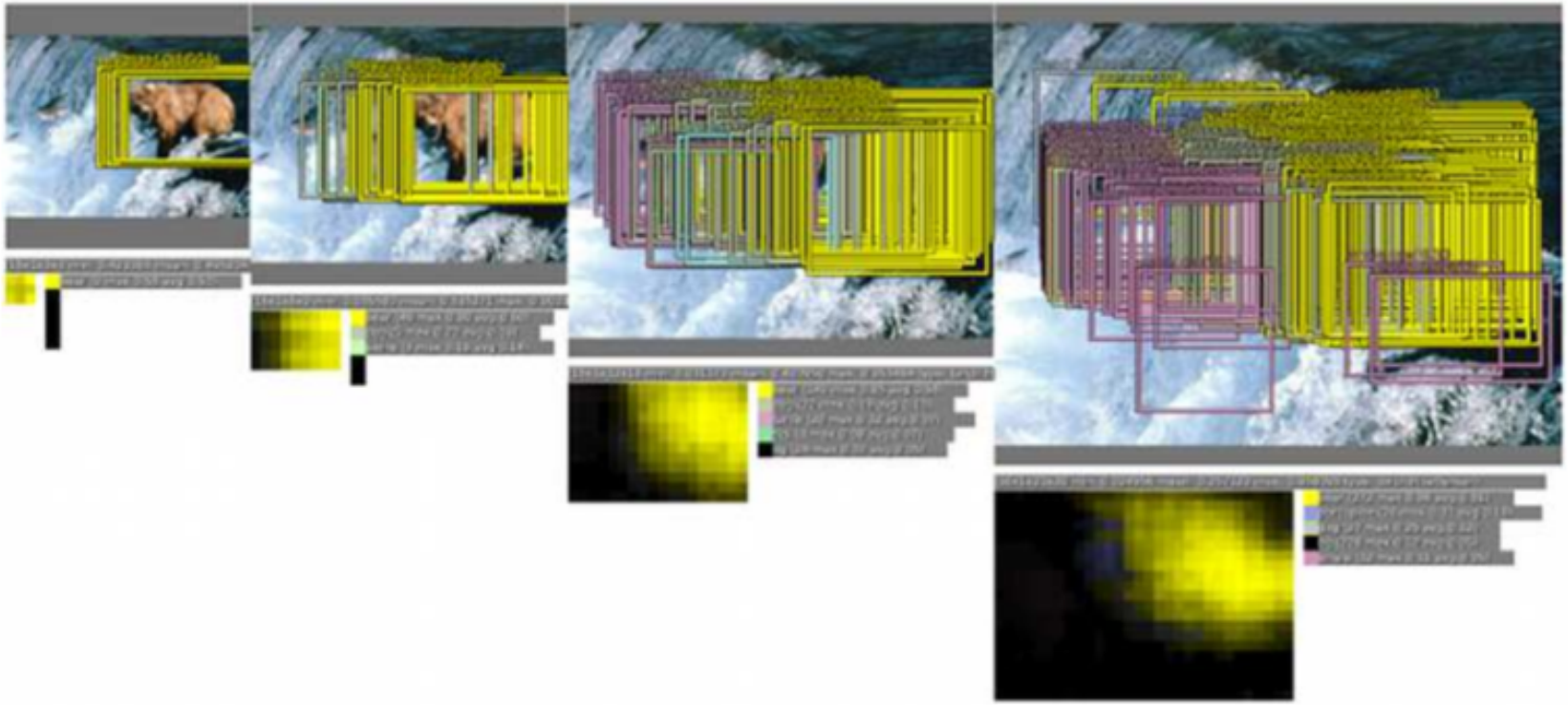
Final Prediction

- Final prediction is given by taking the merged bounding boxes with maximum class scores.
- This is computed by cumulatively adding the detection class outputs associated with the input windows from which each bounding box was predicted

Output of the classifier (fine and coarse resolution) ([Sermanet et al.,])



Prediction and merging of bounding boxes ([Sermanet et al.,])



Detection Task

- Detection training is similar to classification training but in a spatial manner.
- Multiple location of an image may be trained simultaneously.
- The main difference with the localization task, is the necessity to predict a background class when no object is present.

VGG Net

- Visual Geometry Group (VGG) introduced VGG Net in [Karen Simonyan, 2014].
- Main Goal: Investigate influence of depth in CNNs
- Won ILSVRC 2014 on localisation- and 2*nd* on classification task
- 13-layer version:

VGG Net Architecture

- 224×224 RGB images at the input
- Preprocessing: Subtract mean-RGB image computed on trainings set.
- Small receptive fields of 3×3 in filters, particularly in the first conv-layers.
- Stride: 1
- Padding: 1 for 3×3-filters
- 2×2-Max-Pooling with stride 2
- ReLu activation in all hidden layers
- Feature Maps: 64 in the first layer and increasing by a factor of 2 after each pooling layer.

VGG: Investigated Configurations

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					

Number of parameters in millions:

- A, A-LRN: 133
- B: 133
- C: 134
- D: 138
- E: 144

Convolutional filters with small receptive fields in VGG Net

- Stack of two conv-layers with 3×3 receptive field has effective receptive field of 5×5 and stack of three conv-layers with 3×3 receptive field has effective receptive field of 7×7.
- Stack of three conv-layers with 3×3 versus single conv-layer with 7×7 layers:
  - Stacked version has more ReLu-nonlinearities, which enables more discriminative decision function
  - Stacked version has less parameters: $3 \cdot 3^2 C^2 = 27C^2$ versus $7^2 C^2 = 49C^2$, where C is the number of channels (feature maps in the layer). Less parameters impose regularisation.

Training

- Optimisation of multinomial logistic regression by mini-batch gradient descent (backpropagation).
- Mini batch size: 256
- Momentum: 0.9
- Regularisation: 0.0005 on L_2
- Dropout regularisation on first 2 fully-connected layers with dropout-rate 0.5.
- learning rate initally: 0.01.
- Learning stopped after 370K iterations (74 epochs). Compared to other CNNs (AlexNet) less epochs, due to
  - implicit regularisation imposed by greater depth and smaller convolution filters
  - pre-initialisation of some layers: weights of deeper VGG versions (E) are initialised with learned weights of shallower versions (A).

Training Image Size

- Input to the CNN is of size 224×224
- This input is cropped from a training image, which is isotropically rescaled.
- The smallest side of the rescaled training image is called training scale S.
- Two approaches for setting S:
  - Use constant S (single-scale training), e.g. $S = 256$ or $S = 384$.
  - multi-scale training: randomly sample S from a range e.g. $[256, 512]$ (training-set augmentation).

Test-(Classification)- Phase

- Similar as the classification in *OverFeat* [Sermanet et al.,].

Performance at single test scale

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Source: [Karen Simonyan, 2014]

Performance at multiple test scales

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Source: [Karen Simonyan, 2014]

Overview

- Introduced 2015 in [He et al., 2016]
- Research Goal:
  - Recent work has shown that deeper networks perform better, e.g. VGG ([Karen Simonyan, 2014]).
  - Question: *Is learning better networks as easy as stacking more layers?*
- Research Result:
  - Stacking more and more layers together yields degading performance if conventional approach is applied.
  - However, with the new concept of residual nets, performance increases with increasing depth.
- Very Deep Net with 152 layers
- Won ILSVRC 2015 classification task with 3.57% top-5 error rate
- Won several other competitions on other dataset $\Rightarrow$ shows generic applicability of the concept

Deeper Networks are harder to train

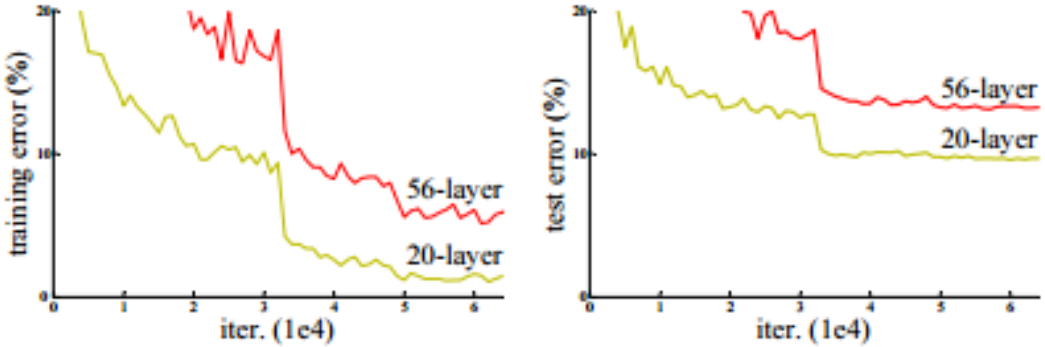


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Source: [He et al., 2016]

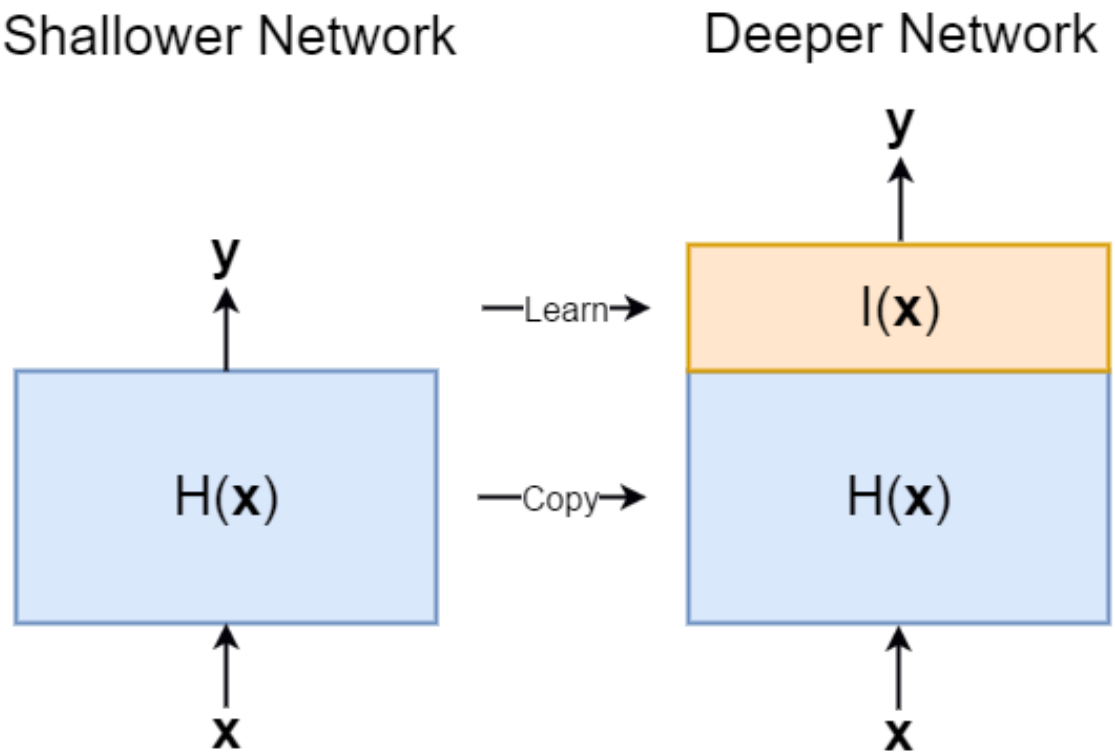
- Experiments with even deeper networks have shown degrading performance
- These networks already integrated techniques to mitigate the *vanishing gradient problem*, e.g. Batch Normalization ([Ioffe and Szegedy, 2015]).
- Particularly interesting is, that the training error degrades. I.e. the weak performance is not due to overfitting.

Are Shallower Networks Subsets of Deeper Networks?

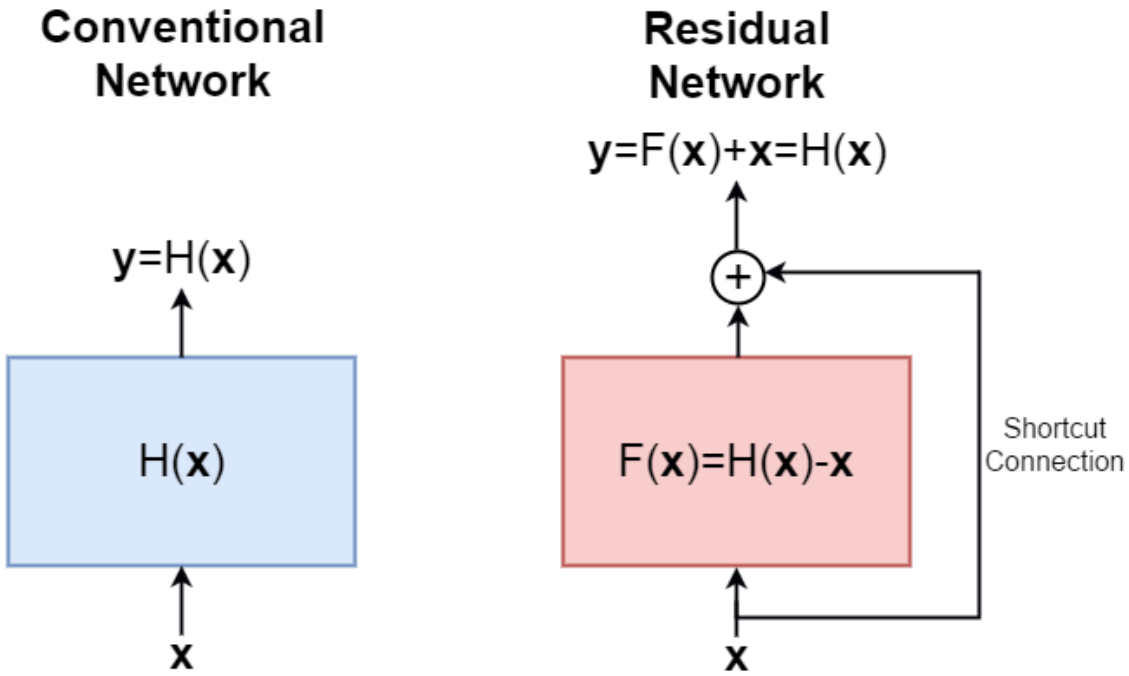
- How can it be, that a deeper network performs worse than the shallower?
- Shallower networks should be subsets of deeper networks.
- A deeper network constructed from a shallower one by (see next slide)
  - Copying the weights from the shallower network,
  - learning the identity in the remaining layers

  should not perform worse than the shallow network!?
- Degradation Problem: Since the deeper network constructed in this way actually performs worse, one can hypothesize, that it's not that easy to learn the identity mapping with several layers.

Are Shallower Networks Subsets of Deeper Networks?

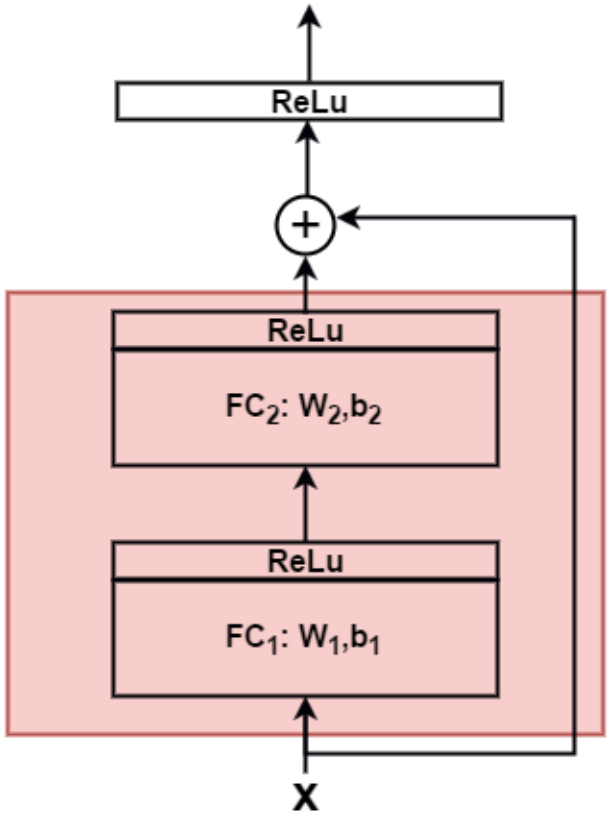


Concept of Residual Net



- Is it easier to learn the target mapping $H(\mathbf{x})$ or the residual mapping $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$?
- Shortcut-connections have no parameters
- SGD and backpropagation can be applied for residual nets in the same way as for conventional nets.

Residual Block



Residual Block:

$F(\mathbf{x})=W_2(\mathrm{ReLu}(W_1\mathbf{x}+b_1)+b_2$

Residual block can consist of an arbitrary number of layers (2 or 3 layers are convenient) of arbitrary type (FC,Conv) and arbitrary activation functions

Building Block in a deep residual net

- In general a single building block in a residual net calculates **y** from it's input **x** by:

$$\mathbf{y} = F(\mathbf{x}, \{W_i, b_i\}) + \mathbf{x}, \tag{2}$$

  where W_i and b_i are the weights-matrix and the bias-vector in the i.th layer of this block.
- In this case the dimensions of the output **y** and the input **x** must be the same.
- If the output and input shall have different dimensions, the input can be transformed by W_s:

$$\mathbf{y} = F(\mathbf{x}, \{W_i, b_i\}) + W_s\mathbf{x}, \tag{3}$$

Options to obtain higher dimension in output than in input

Option A: Use zero-padding shortcuts for increasing dimensions

Option B: Projection shortcuts are used for increasing dimensions and identity for the others

Option C: All shortcuts are projections

In the ResNet architecture (next slide) such dimension-increasing shortcuts are represented by dotted lines.

34-Layer ResNet Architecture for ImageNet

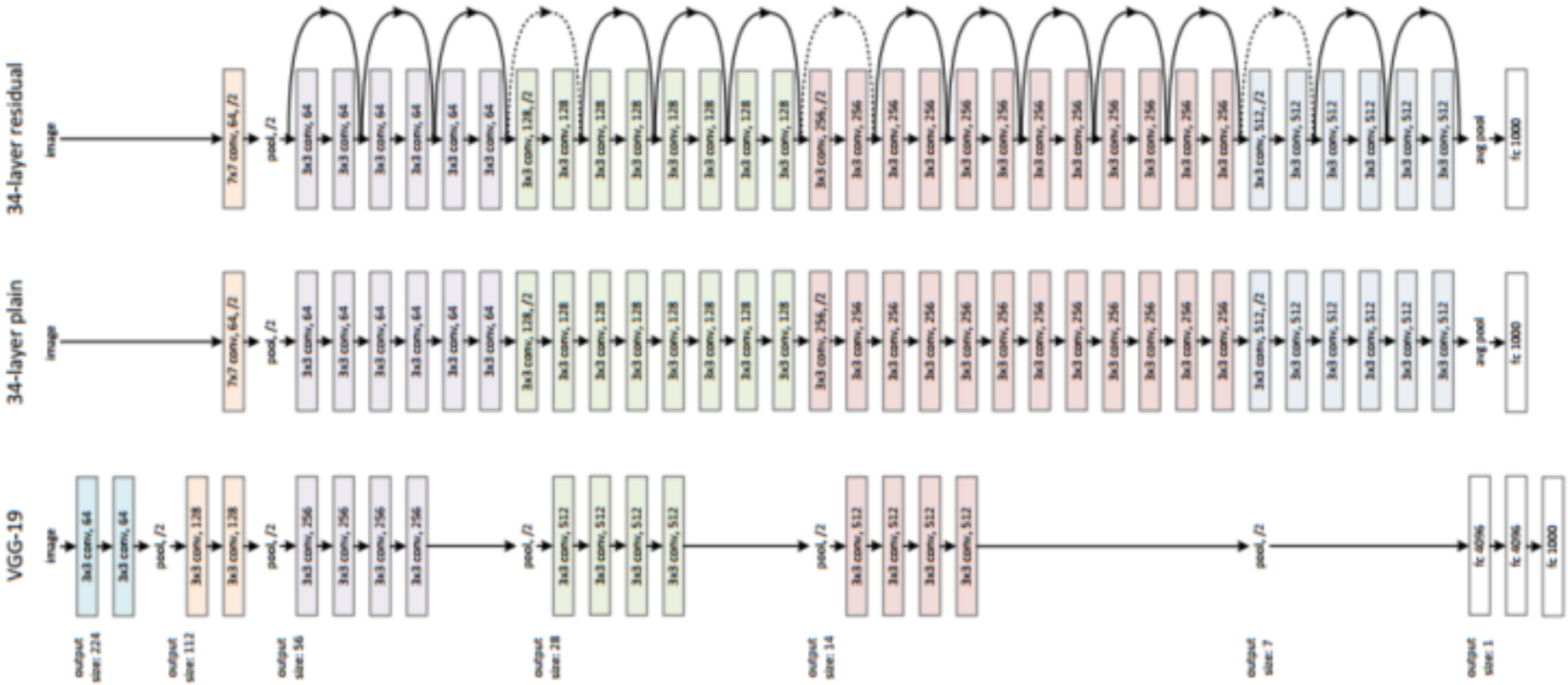


Figure: VGG-19 model (bottom), Plain 34-layer network and 34 layer Residual Network

Source: [He et al., 2016]

ResNet Architectures for ImageNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Source: [He et al., 2016]

Training

- Scale-augmentation similar as in VGG-Net: Resize image such that its shorter side is within $[256, 480]$ (randomly sampled).
- 224×224 - crop is randomly sampled from image or its horizontal flip.
- Color augmentation
- Batch normalization (according to [Ioffe and Szegedy, 2015]) after convolution and before activation.
- SGD training with mini-batch size of 256
- Learning rate starts from 0.1 and is divided by 10 if the error plateaus.
- Weight decay: 0.0001
- Momentum: 0.9
- No dropout

Testing

- 10-crop testing as in AlexNet: 5 views and their horizontal flips
- Fully-convolutional form as in VGG-Net
- Average scores at multiplie ranges: $[224, 256, 384, 480, 640]$

Comparison of Top-1 Error Rate in Plain- and Residual Net

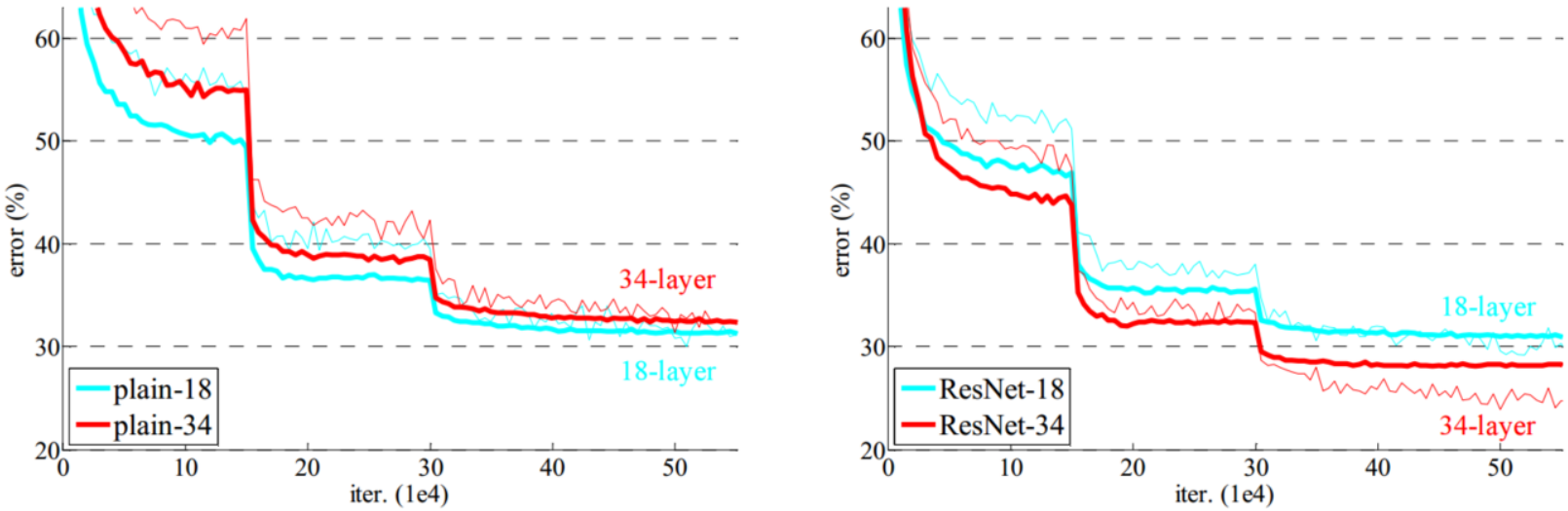


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Source: [He et al., 2016]

Comparison of different dimension increasing options

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

Source: [He et al., 2016]

Error Rates of Ensembles

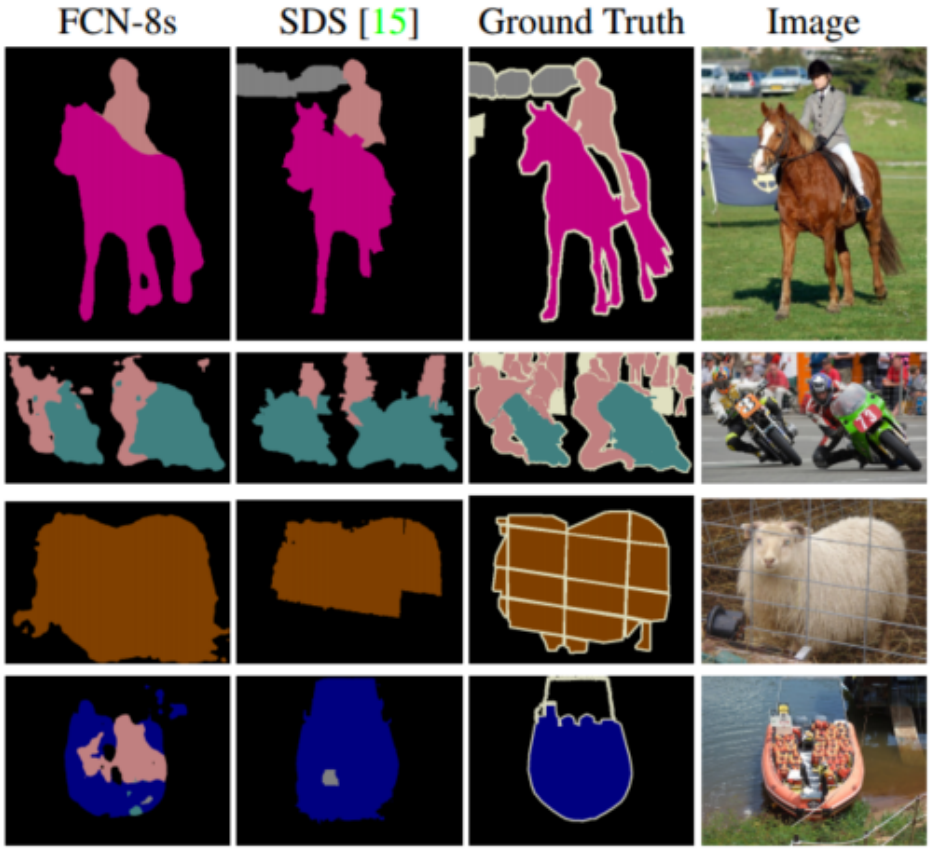
Combination of 6 models of different depth (only 2 of them of depth 152):

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

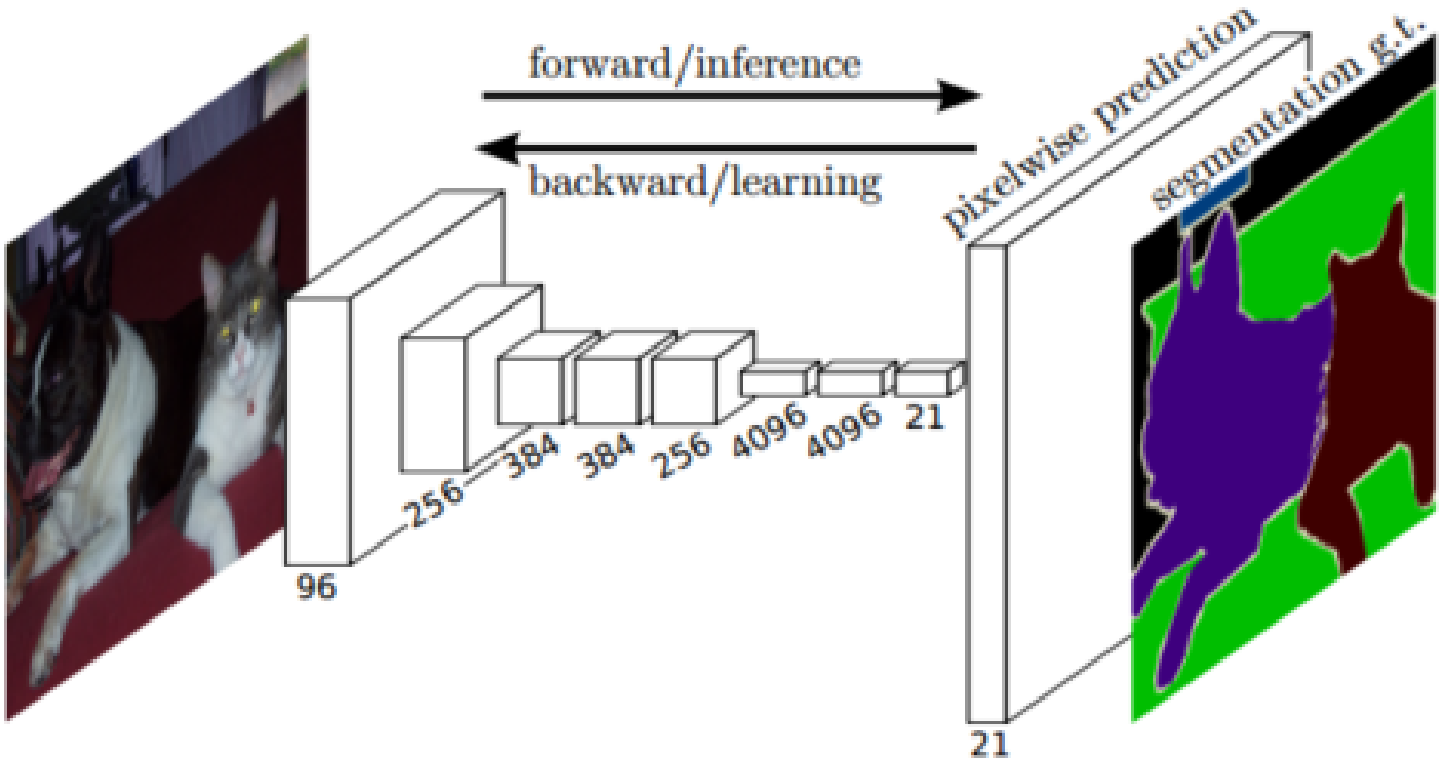
Source: [He et al., 2016]

Semantic Segmentation with CNNs [Shelhamer et al., 2016]



Source: [Shelhamer et al., 2016]

Semantic Segmentation with CNNs [Shelhamer et al., 2016]



Source: [Shelhamer et al., 2016]

Challenge of applying CNNs for semantic segmentation

- In typical CNNs for object recognition:
  - In deeper layers the problem of *What is contained in the picture?* is solved.
  - In deeper layers resuolution is strongly reduced, i.e. the question of *Where?* can not be solved sufficiently.
- Semantic segmentation requires accurate answers for *What* and *Where*.

Conventional approaches for segmentation (not semantic segmentation):

- Unsupervised clustering: Hierarchical/Ward, Gaussian Mixture Models, Mean-Shift-Clustering

Concept of CNN-based semantic segmentation as introduced in [Shelhamer et al., 2016]

- Apply trained CNN for object recognition, e.g. *AlexNet, VGG-Net*
- Convolutionalize (see OverFeat) FC-layers for larger images as applied in training
- From the 2-D Map of class-probability-vectors in the deepest layer apply deconvolution in order to reconstruct an image model
- Apply finer resolution information from the lower layers
- Combine information on What from deepest layer and information on Where from less deeper layers
- End-to-End per pixel training for
  - Learning deconvolution layers
  - Learning combination of information from different layers
  - fine-tune all weights
- Number of different classes incl. background in PASCAL VOC data: 21

Convolution as Matrix Multiplication

Input **X**

$$\begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}$$

Filter **W**

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix}$$

Convolution as Matrix Multiplication

The serialized output $\mathbf{Y_s}$ of a convolutional filtering can be calculated by matrix-multiplication

$$\mathbf{Y_s} = W_S X_S,$$

where $\mathbf{X_s}$ is the serialization of the input.

$$W_s^T = \begin{bmatrix} w_{0,0} & 0 & 0 & 0 \\ w_{0,1} & w_{0,0} & 0 & 0 \\ w_{0,2} & w_{0,1} & 0 & 0 \\ 0 & w_{0,2} & 0 & 0 \\ w_{1,0} & 0 & w_{0,0} & 0 \\ w_{1,1} & w_{1,0} & w_{0,1} & w_{0,0} \\ w_{1,2} & w_{1,1} & w_{0,2} & w_{0,1} \\ 0 & w_{1,2} & 0 & w_{0,2} \\ w_{2,0} & 0 & w_{1,0} & 0 \\ w_{2,1} & w_{2,0} & w_{1,1} & w_{1,0} \\ w_{2,2} & w_{2,1} & w_{1,2} & w_{1,1} \\ 0 & w_{2,2} & 0 & w_{1,2} \\ 0 & 0 & w_{2,0} & 0 \\ 0 & 0 & w_{2,1} & w_{2,0} \\ 0 & 0 & w_{2,2} & w_{2,1} \\ 0 & 0 & 0 & w_{2,2} \end{bmatrix} \quad \mathbf{X_s} = \begin{bmatrix} X_{0,0} \\ X_{0,1} \\ X_{0,2} \\ X_{0,3} \\ X_{1,0} \\ X_{1,1} \\ X_{1,2} \\ X_{1,3} \\ X_{2,0} \\ X_{2,1} \\ X_{2,2} \\ X_{2,3} \\ X_{3,0} \\ X_{3,1} \\ X_{3,2} \\ X_{3,3} \end{bmatrix}$$

The de-serialized result $\mathbf{Y}$ is a 2×2-matrix.
Note: Because of space limitation, here the transpose W_S^T is plotted. However in the convolution W_S is applied.

Deconvolution as Matrix Multiplication

- The serialized output $\mathbf{V_s}$ of a deconvolutional filtering can be calculated by matrix-multiplication

$$\mathbf{V_s} = W_S^T \mathbf{U_s},$$

  where $\mathbf{U_s}$ is the serialization of the input.
- Note that deconvolution (= transpose convolution) is not the inverse of convolution.
- For the given 3×3 - matrix W and an 2×2 input matrix $\mathbf{U_s}$, the de-serialized output is a 4×4 - matrix.

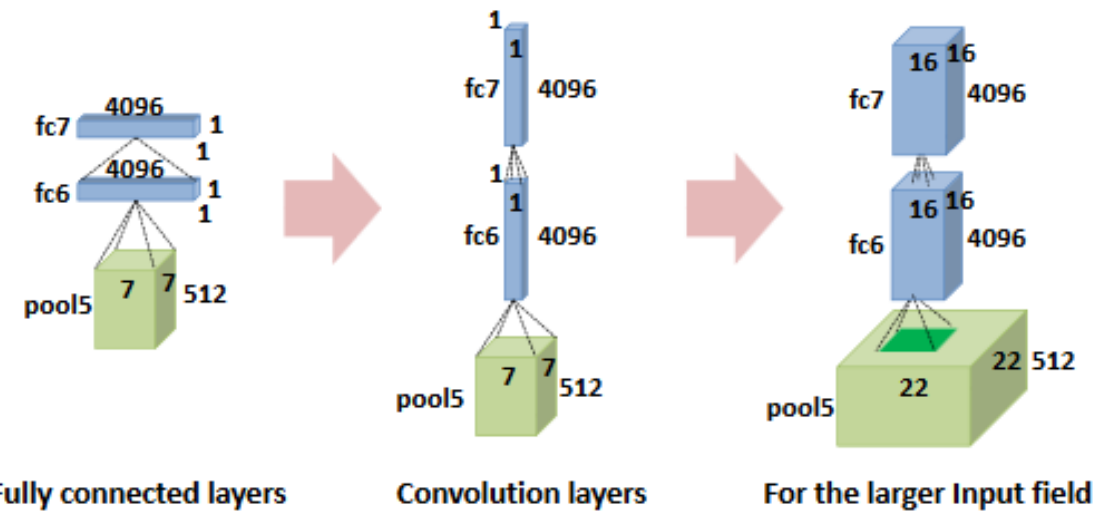
$\Rightarrow$ If convolutional filtering with W yields an output, which is smaller than the input, then deconvolutional filtering yields an output, which is - by the same factor - larger than the input.
$\Rightarrow$ The resizing increases with the stride-length S.

What would be the size of the output for a 2×2 input matrix U and a 3×3 - filtermatrix W, if the stride-length is $S = 2$?

Transform FC-Layers to Convolution-Layers ⇒ Fully Convolutional Network

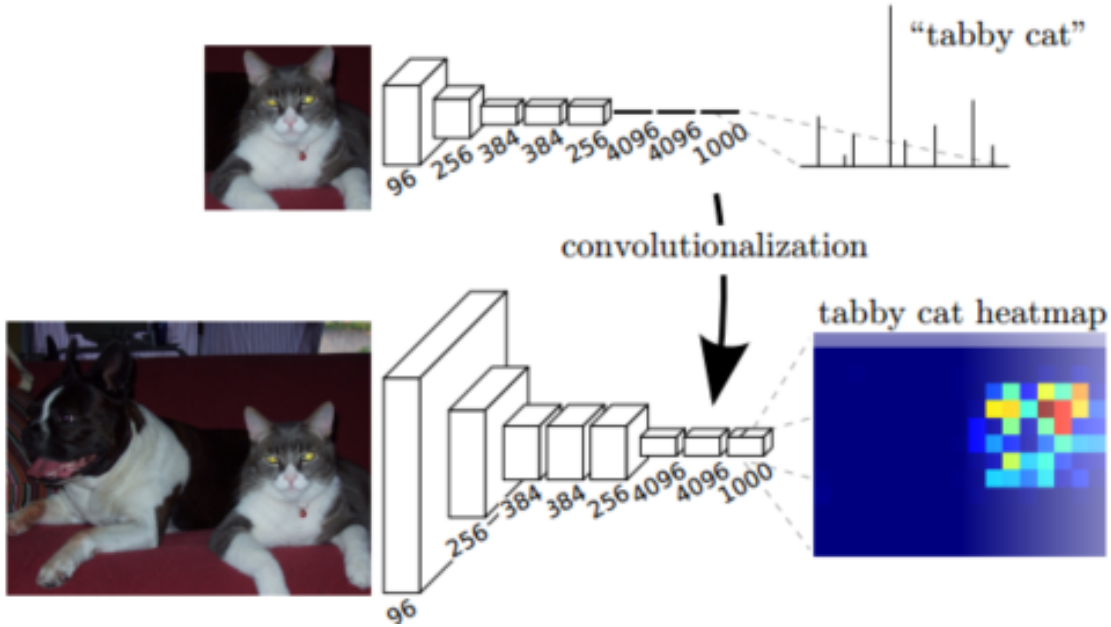
Conversion of FC-layers to convolutional layers:

- Has already been described in section OverFeat
- Each FC-Layer is interpreted as a conv-layer whose filter is as large as the feature-maps in the previous layer
- Number of the neurons in FC-layer is number of *Feature Maps* in the re-interpretation



Fully Convolutional Network (FCN)

Convolutionalisation:

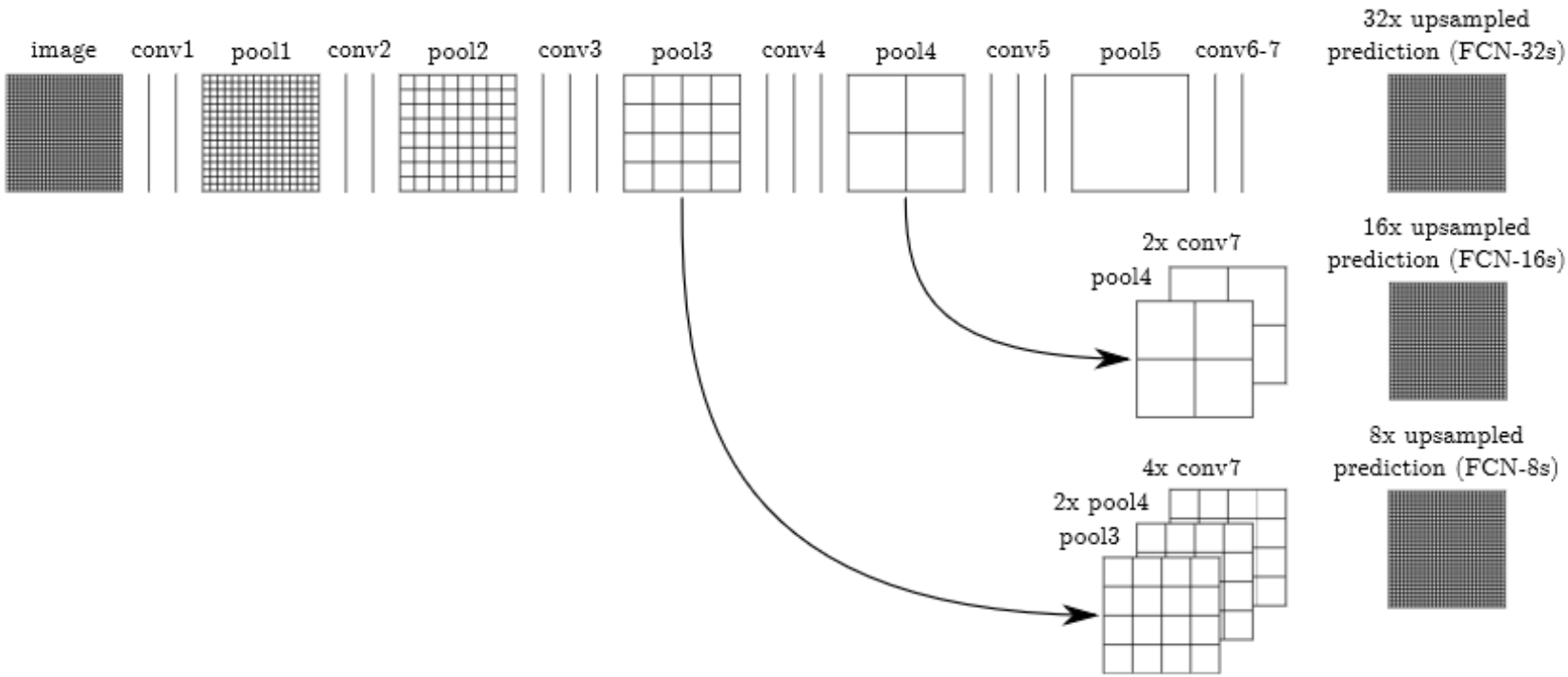


Source: [Shelhamer et al., 2016]

- Train CNN over *small images*. Output is vector of class probabilities
- Apply trained CNN on larger image. Output is a 2-D-map of class-probability-vectors

Combine finer resolution with coarser resolution information

Based on VGG-16:



Source: [Shelhamer et al., 2016]

Combining classification and localisation from different layers

- From the input to the final prediciton layer (conv-7) an image-sized map can be reconstructed by upsampling with a 32-pixel-stride deconvolution.
- The weights of this deconvolution-filter are initialized to bilinear interpolation and are then learned by pixelwise end-to-end training.
- The corresponding segmentation is very coarse
- Combining fine layers and coarse layers lets the model make local predictions that respect global structure.
  - add a 1×1-conv-layer on top of pool4 to produce additional class predictions.
  - fuse this output with the predictions computed on top of conv-7 by $2\times$ upsampling conv7-based prediction-map
    - $2\times$ upsampling conv7-based prediction-map
    - add this upsampled conv7-based prediction-map with the pool4-based prediction map
    - upsample the sum by a factor of 16 by applying stride-16 deconvolution
    - $\Rightarrow$ FCN-16
    - The weights of FCN-16 are initialized with the learned weights of the coarser version FCN-32 and are then learned by pixelwise end-to-end training.
- The same process is applied to obtain to integrate even finer resolution prediction from the output of pool3 $\Rightarrow$ FCN-8

Segmentation resolution of different FCNs

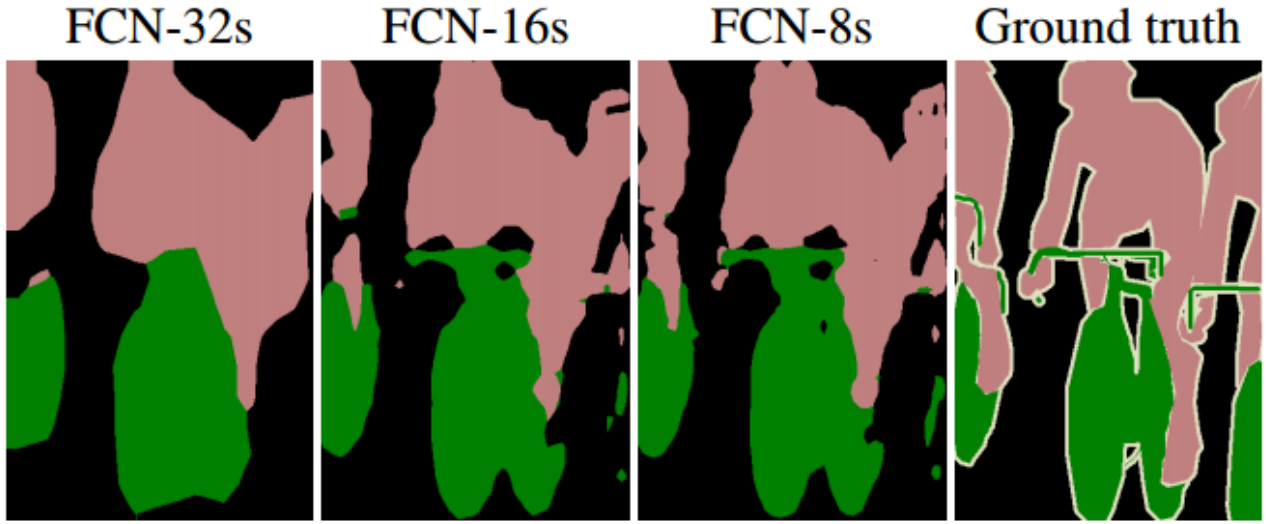


Figure 4. Refining fully convolutional nets by fusing information from layers with different strides improves segmentation detail. The first three images show the output from our 32, 16, and 8 pixel stride nets (see Figure 3).

Source: [Shelhamer et al., 2016]

Test Results

- n_{ij}: number of pixels of true class i, predicted as class j
- n_{cl}: number of classes
- t_i: total number of pixels of true class i: $t_i = \sum_j n_{ij}$
- Pixel Accuracy: $\sum_i n_{ii} / \sum_i t_i$
- Mean Accuracy: $(1/n_{cl}) \sum_i (n_{ii}/t_i)$
- Mean IU: $(1/n_{cl}) \sum_i (n_{ii}/(t_i + \sum_j n_{ji} - n_{ii}))$

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	**90.3**	**75.9**	**62.7**	**83.2**

Source: [Shelhamer et al., 2016]

- In FCN-32s-fixed only the last layer is fine-tuned
- For all others pixelwise end-to-end training is applied

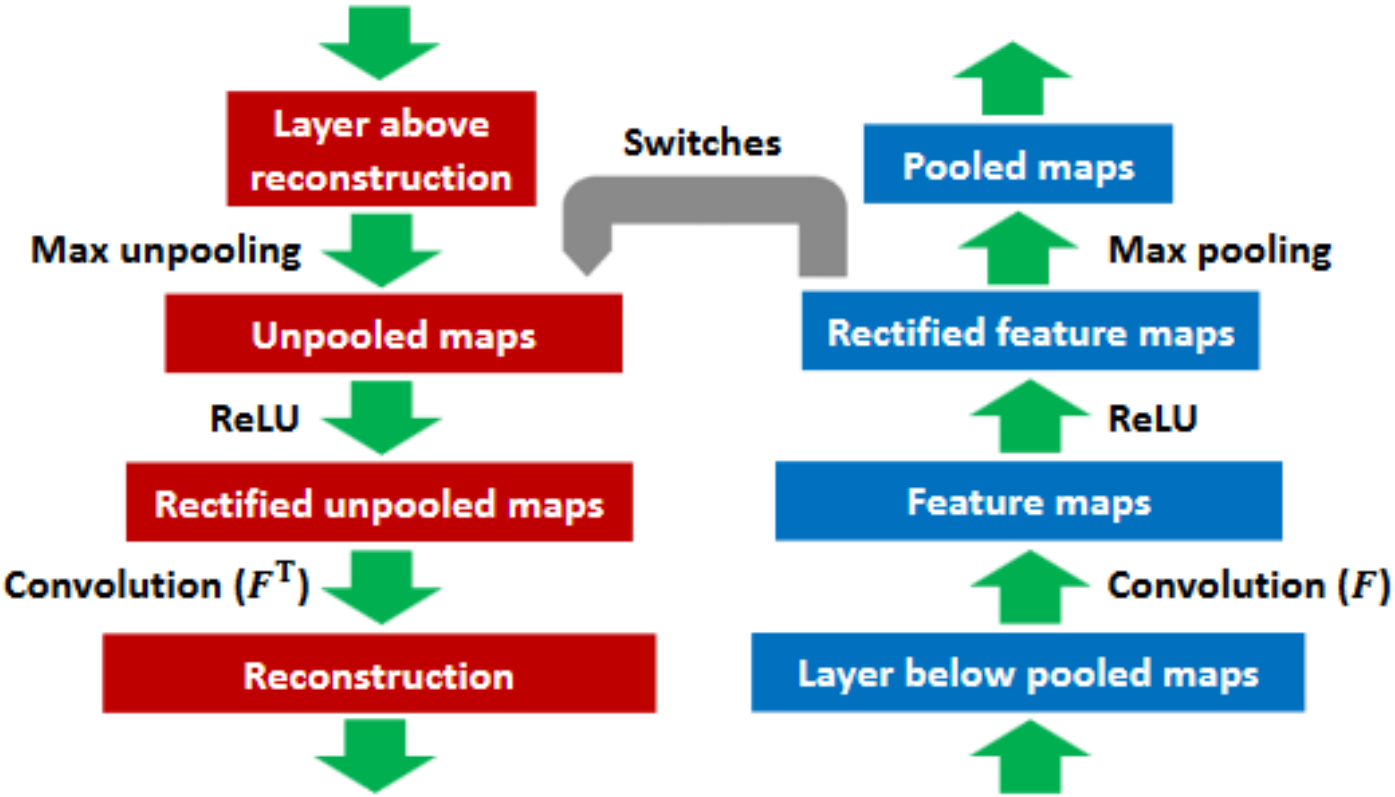
Deconvolution and Unpooling: Applications

- Semantic segmentation [Shelhamer et al., 2016]
- Visualization of features [Matthew D. Zeiler, 2014]
- Generative Models
- Encoder/Decoder-Models

Convolutional Arithmetic Tutorial:
http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html

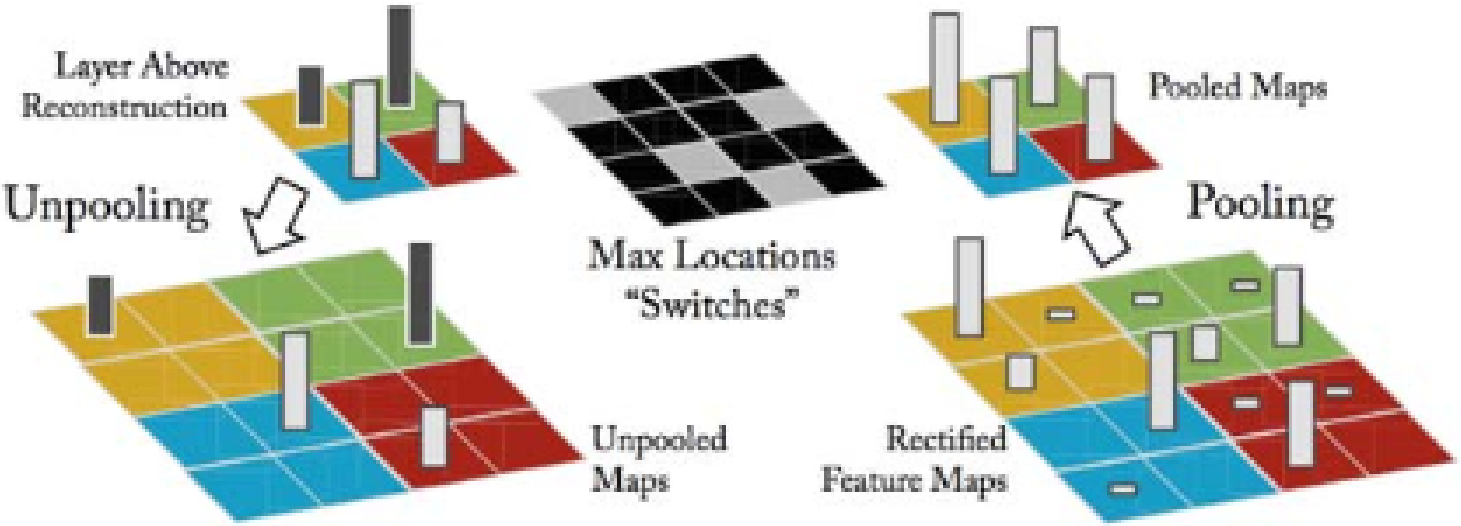
Jupyter Notebook: Convolution and Deconvolution Demo:
https://www.hdm-stuttgart.de/ maucher/ipnotebooks/ObjectRecognition/deconvolution.html

Deconvolution as applied in [Matthew D. Zeiler, 2014] for visualisation



Source: [Matthew D. Zeiler, 2014]

Deconvolution as applied in [Matthew D. Zeiler, 2014] for visualisation



Source: [Matthew D. Zeiler, 2014]

Unpooling:

- Max-Pooling is non-invertible
- Unpooling is just an approximate inverse
- *Switches* record the location of the maximas

References I

Bengio, Y. (2009).

Learning deep architectures for ai.

Foundations and Trends in Machine Learning, 2(1):1–127.

He, K., Zhang, X., Ren, S., and Sun, J. (2016).

Deep residual learning for image recognition.

In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Ioffe, S. and Szegedy, C. (2015).

Batch normalization: Accelerating deep network training by reducing internal covariate shift.

CoRR, abs/1502.03167.

Karen Simonyan, A. Z. (2014).

Very deep convolutional networks for large-scale image recognition.

Krizhevsky, A., Sutskever, I., and Hinton, G. E.

Imagenet classification with deep convolutional neural networks.

References II

lab, L. (2011).

Deep learning tutorials v0.1.

http://deeplearning.net/tutorial/.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2001).

Gradient-based learning applied to document recognition.

In *Intelligent Signal Processing*, pages 306–351. IEEE Press.

Matthew D. Zeiler, R. F. (2014).

Visualizing and understanding convolutional networks.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and Lecun, Y.

Overfeat: Integrated recognition, localization and detection using convolutional networks.

Shelhamer, E., Long, J., and Darrell, T. (2016).

Fully convolutional networks for semantic segmentation.

IEEE Transactions on Pattern Analysis and Machine Intelligence, PP(99):1.