EBERHARD KARLS UNIVERSITÄT TÜBINGEN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Deep Neural Networks

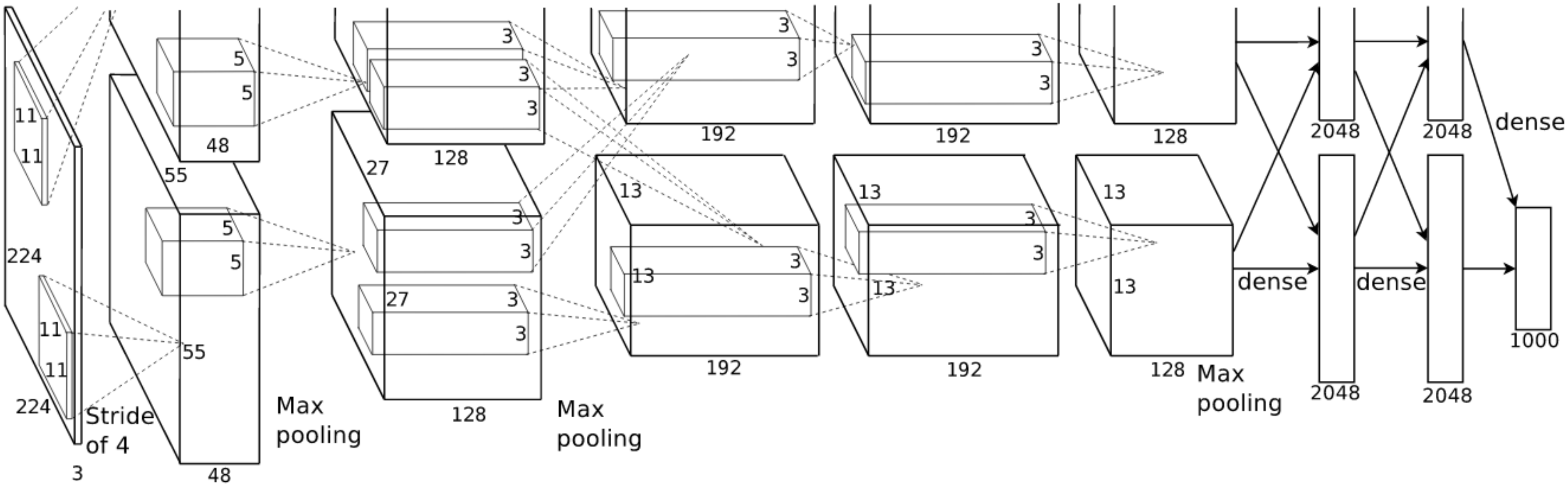
Chapter 12: DNN Applications

AlexNet [Krizhevsky et al. 2012]

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- AlexNet: The convolutional neural network that changed ML science and started the 3rd wave of neural networks
- Developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton.
- AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% vs. 26% error).
- The Network had a very similar architecture to LeNet by LeCun, but was deeper, bigger, and featured convolutional layers stacked on top of each other
- Previously it was common to only have a single conv. layer always immediately followed by a pooling layer.

AlexNet [Krizhevsky et al. 2012]



- Architecture:

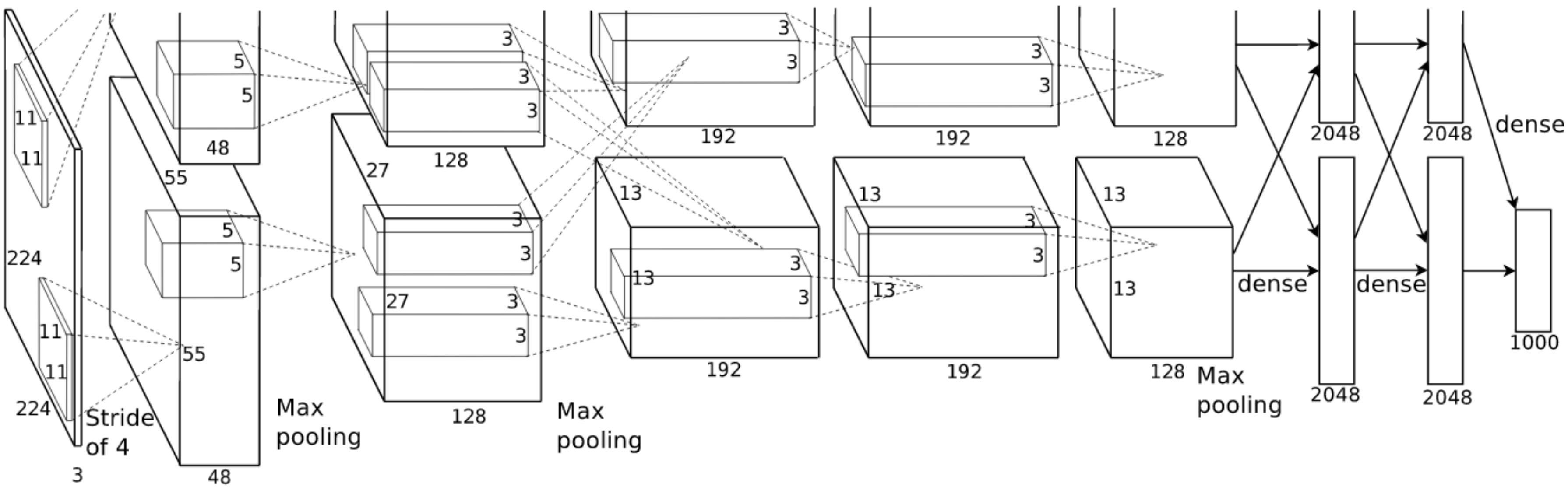
  CONV1 - MAX POOL1 - NORM1

  CONV2 - MAX POOL2 – NORM2

  CONV3 - CONV4 - CONV5 – MAX POOL3

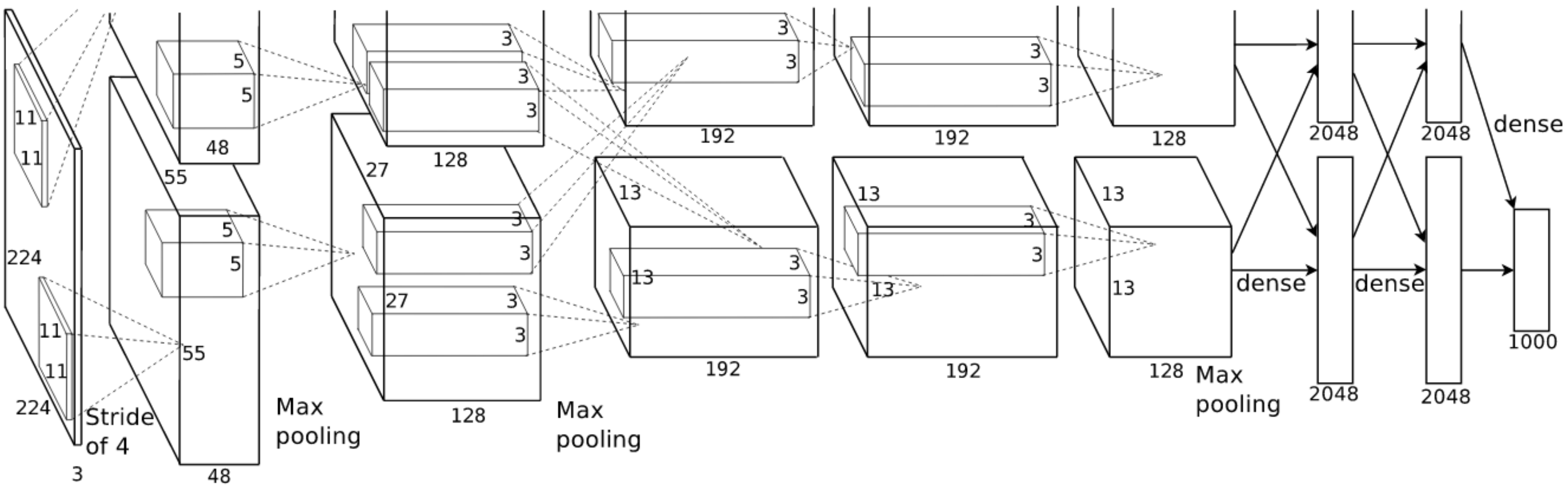
  FC6 -FC7 - FC8

AlexNet [Krizhevsky et al. 2012]



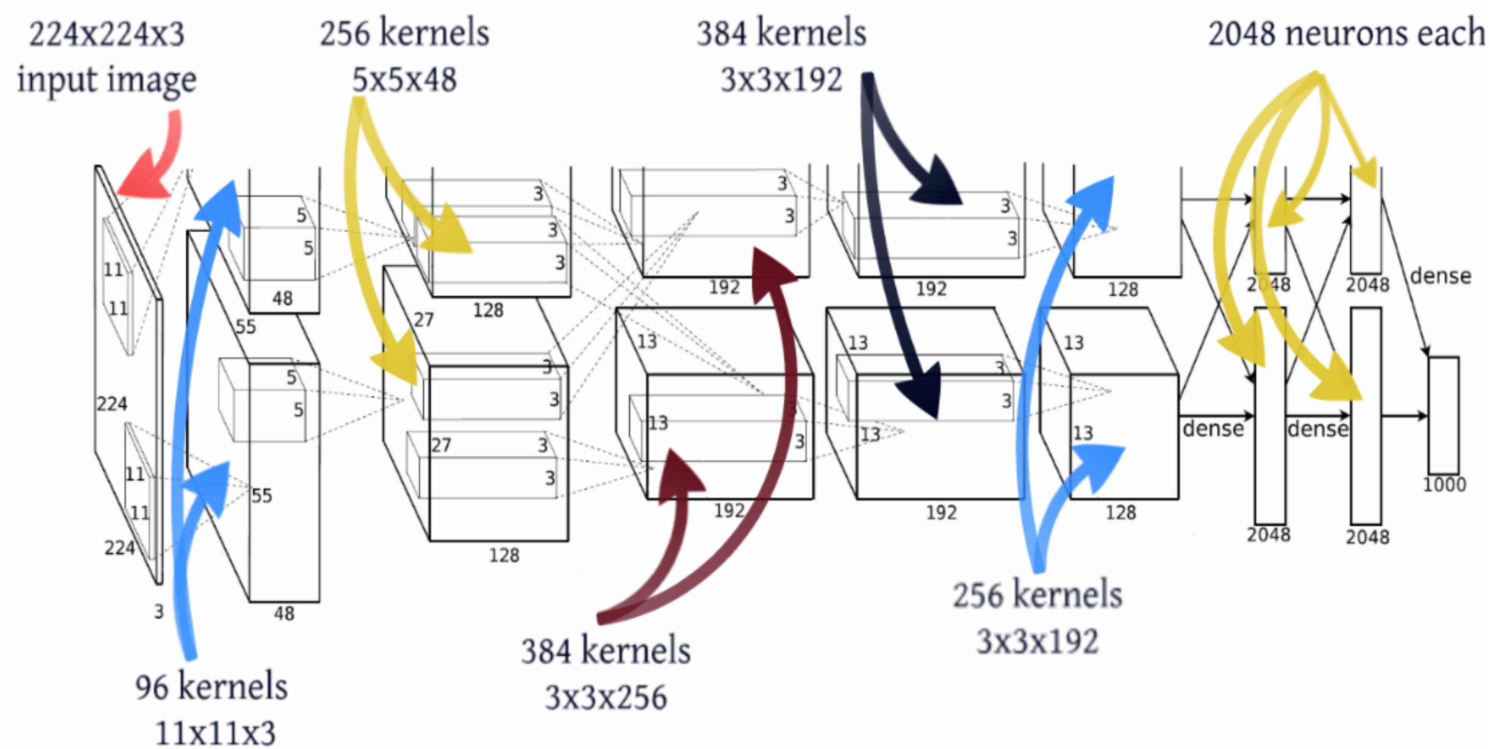
- Input: 224x224x3
- First layer (CONV1): 96 11x11 filters applied at stride 4
  - Output size: (227-11)/4+1=55
  - (11*11*3)*96 = 34848 parameters

AlexNet [Krizhevsky et al. 2012]



- Input: 224x224x3
- After CONV1: 55x55x96
- Second layer (POOL1): 3x3 filters applied at stride 2
  - Output size: (55-3)/2+1= 27

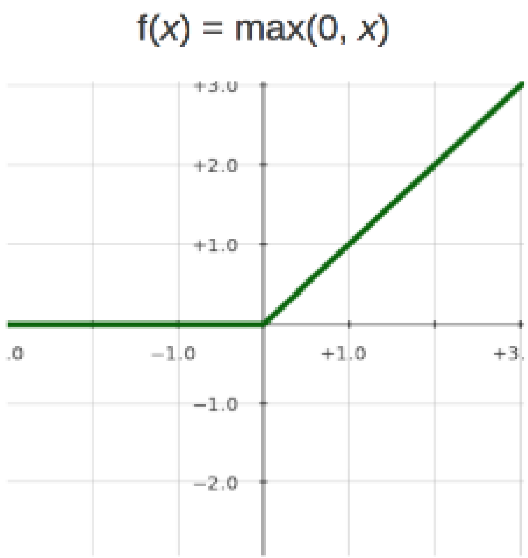
AlexNet [Krizhevsky et al. 2012]

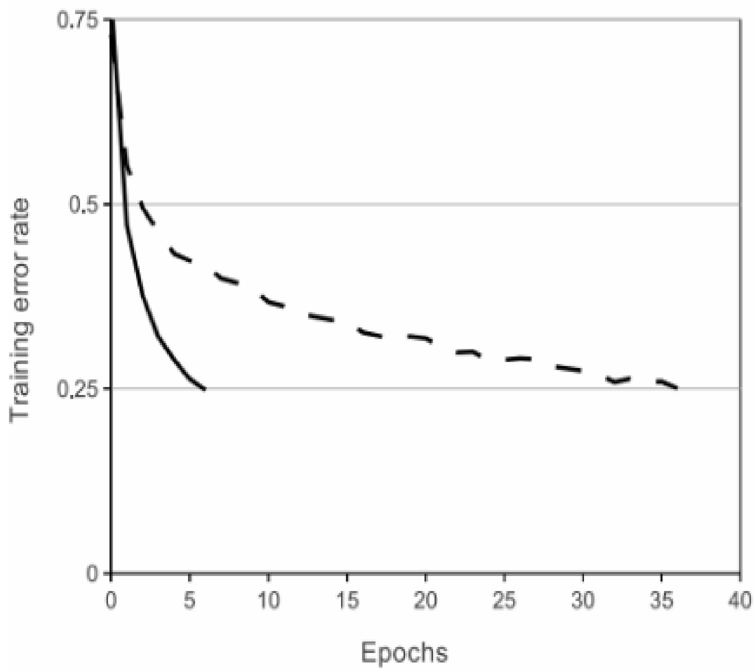


- 7 hidden “weight” layers
- 650K neurons
- 60M parameters
- 630M connections

AlexNet [Krizhevsky et al. 2012]

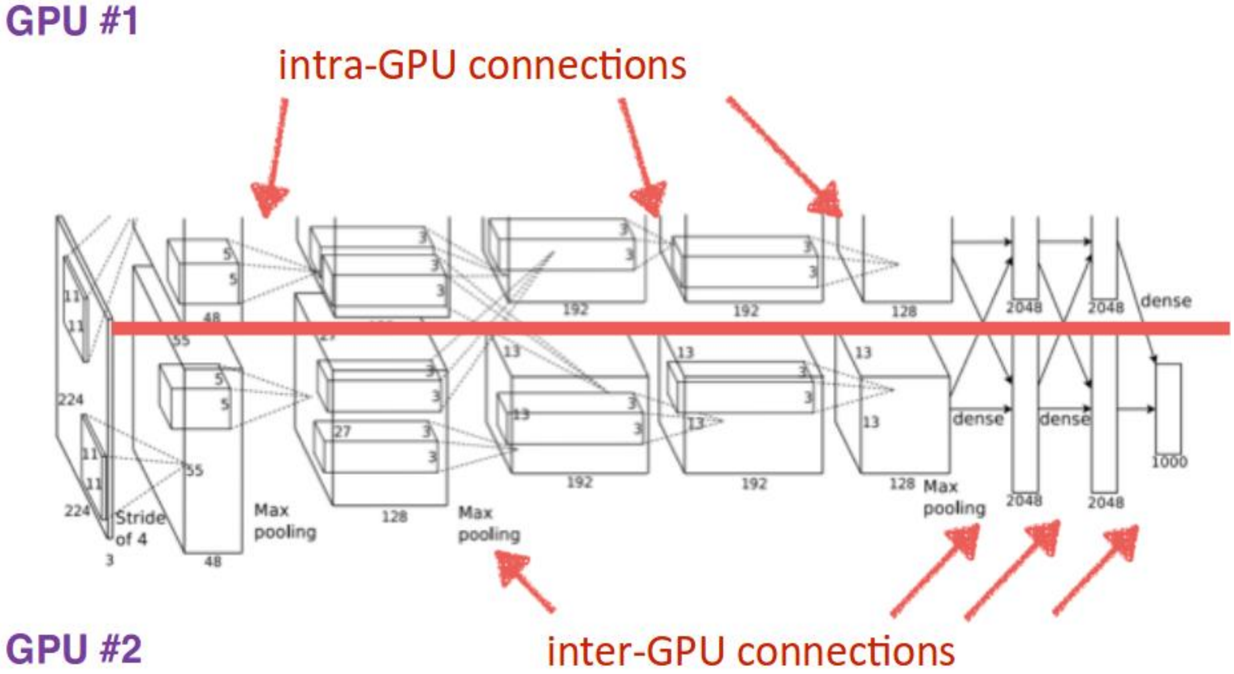
- **First use of ReLu:**
  - Non-saturating nonlinearity
  - Quicker training: A 4 layer CNN with ReLUs (solid line) converges six times faster than an equivalent network with tanh neurons (dashed line) on CIFAR-10 dataset





AlexNet [Krizhevsky et al. 2012]

- **Training on multiple GPUs:**
  - A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it.
  - Spread the net across two GPUs: The parallelization scheme employed essentially puts half of the kernels (or neurons) on each GPU, with one additional trick: the GPUs communicate only in certain layers.



AlexNet [Krizhevsky et al. 2012]

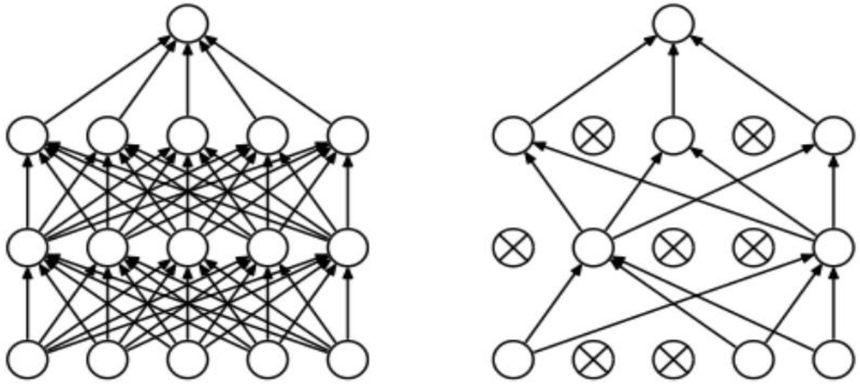
- **Local response normalization:** (not common any more)
  - take adjacent channels and renormalize their response. Channels are ordered arbitrarily.
  - Introducing some competition between neighboring neurons. Such lateral inhibition is observed in the brain, and you can also think of it as helping sharpening the response.
- **Overlapping pooling:**
  - top-1 and top-5 error rates decrease by 0.4% and 0.3%, respectively, compared to the non-overlapping scheme (2X2/2).

AlexNet [Krizhevsky et al. 2012]

- Fighting against overfitting: **Data augmentation:**
- At test time, average the predictions on the 10 patches.
- Altering the intensities of the RGB channels in training images: perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, add rescale the found principal components.
- This scheme approximately captures an important property of natural images, namely that object identity is invariant to changes in the intensity and color of the illumination. This scheme reduces the top-1 error rate by over 1%.

AlexNet [Krizhevsky et al. 2012]

- Fighting against overfitting: **Dropout:**
- Combining the predictions of many different models is helpful to reduce test error, but it is too expensive for big neural networks.
- Dropout consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are dropped out in this way do not contribute to the forward pass and do not participate in back-propagation.



Standard Neural Net   After applying dropout.

AlexNet [Krizhevsky et al. 2012]

- Results:
- ILSVRC 2010 (test set)

Model	Top-1	Top-5
Sparse coding [2]	47.1%	28.2%
SIFT + FVs [24]	45.7%	25.7%
CNN	**37.5%**	**17.0%**

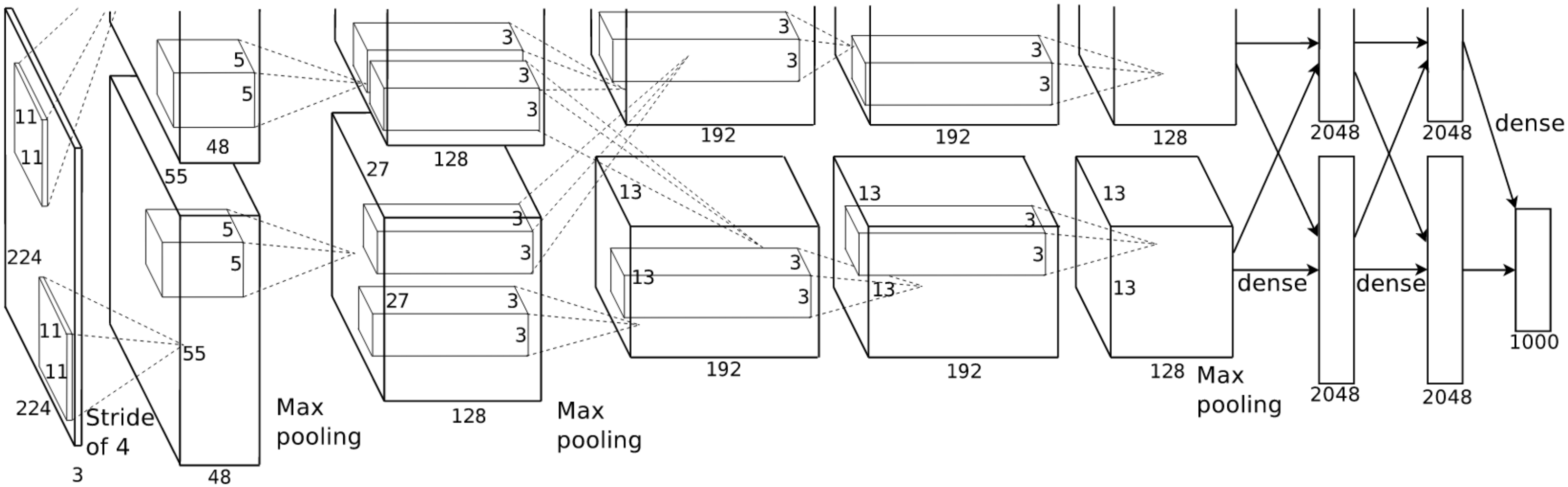
- ILSVRC 2012 (Models with an * were "pre trained" to classify the entire ImageNet 2011 Fall release)

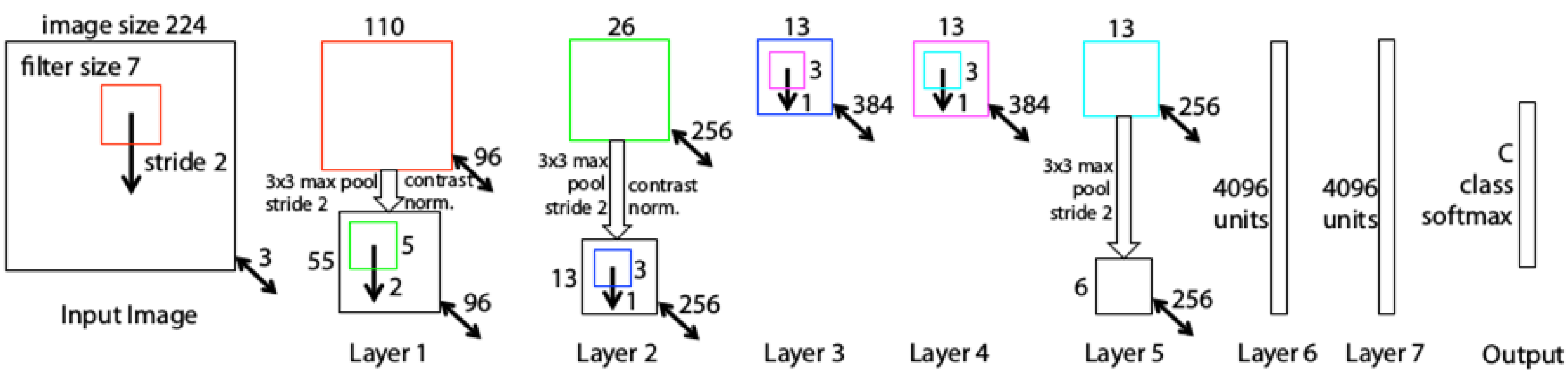
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	26.2%
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Can we go deeper?

- 8 Layers for AlexNet (2012)
- ZF-Net (2013) also 8 Layers, but improves accuracy
- Evidence that depth is important
- But: Deep Nets are difficult to train:
  - A lot of parameters to train
  - Training is sensitive to initialization
  - Vanishing gradient problem: small gradients in early layers since backpropagation has to go through many layers

Review of AlexNet and ZF-Net





ZF Net Architecture

VGG-Net‘s approach

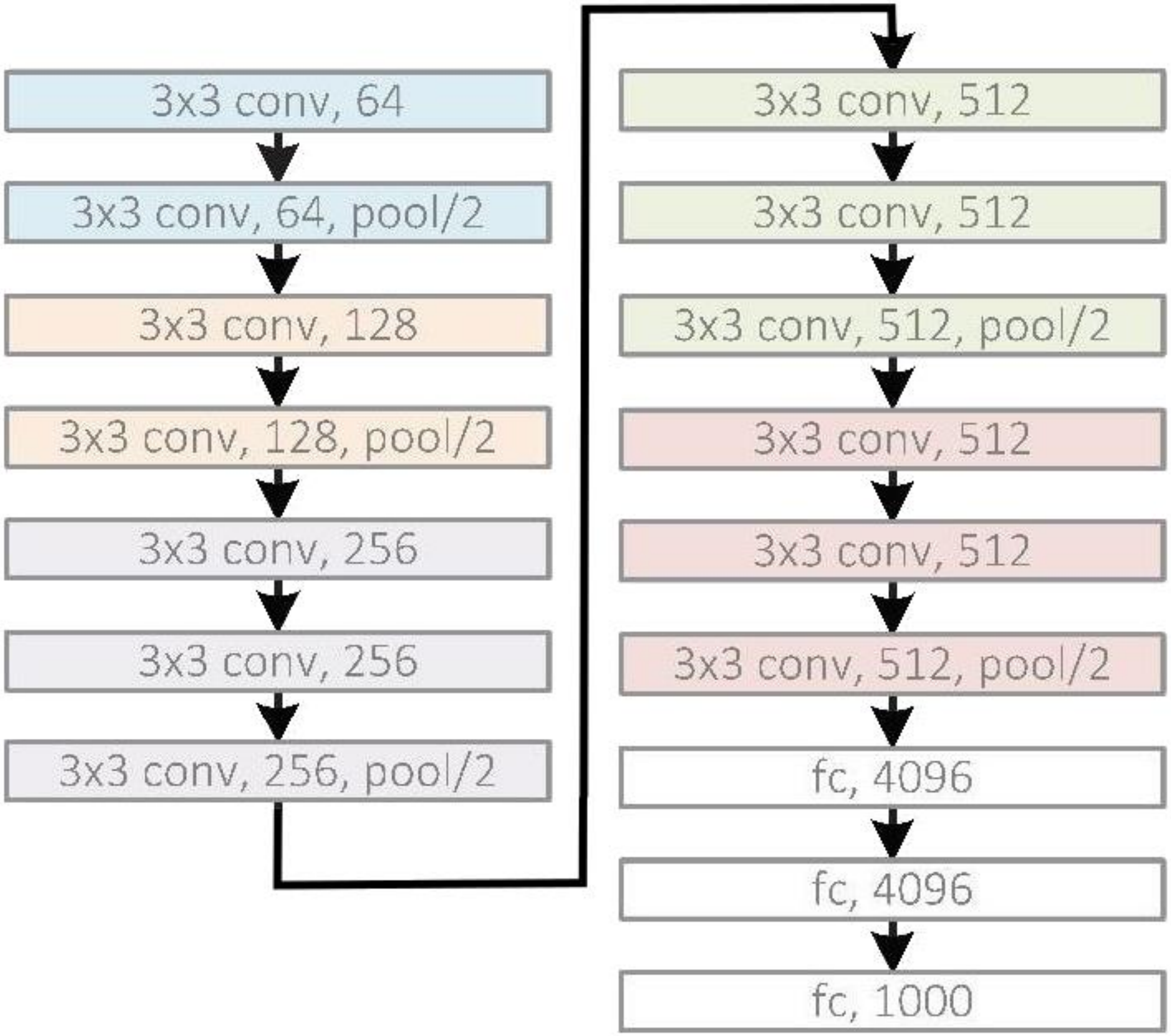
- Use small (3x3) convolutional filters all throughout the network
  - Replacing one large convolutional filter by several small ones allows the network to learn more complicated features (with the same receptive field) early in the network
- Use more convolutions between pooling layers
- To overcome initialization problem, start by training smaller networks
  - Use trained weights in the smaller network to initalize most layers in the larger network

Different VGG-Net architectures

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					

- Architectures D (VGG16) and E (VGG19) still commonly used as a basis for training other neural networks today
- LRN = Local Response Normalization (from AlexNet) did not improve the network

VGG16



3x3 conv, 64
3x3 conv, 64, pool/2
3x3 conv, 128
3x3 conv, 128, pool/2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256, pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512, pool/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512, pool/2
fc, 4096
fc, 4096
fc, 1000

VGG-Net results

Table 3: **ConvNet performance at a 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**

- Image scale: rescaling of training/test images before cropping
- Multiple values: scale randomly chosen in interval to achieve scale invariance

VGG-Net results

Table 6: **Multiple ConvNet fusion results.**

Combined ConvNet models	Error		
	top-1 val	top-5 val	top-5 test
ILSVRC submission			
(D/256/224,256,288), (D/384/352,384,416), (D/[256;512]/256,384,512) (C/256/224,256,288), (C/384/352,384,416) (E/256/224,256,288), (E/384/352,384,416)	24.7	7.5	7.3
post-submission			
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), dense eval.	24.0	7.1	7.0
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop	23.9	7.2	-
(D/[256;512]/256,384,512), (E/[256;512]/256,384,512), multi-crop & dense eval.	**23.7**	**6.8**	**6.8**

7.3% test error using an ensemble of 7 models. After the submission, we decreased the error rate to 6.8% using an ensemble of 2 models.

- Dense eval.: turn fully connected layers into convolutions and evaluate on bigger parts of the image
- Multi-crop: evaluate the network on several crops of the image and average the results

Weight initialization

- Can‘t set weights to zero: need to differentiate neurons
- Random initalization
- Since we use non-saturating activation function, activations could grow very large
- Try to tune initialization to keep magnitude of activations approximately constant
- Depends on how many neurons in the previous layer contribute to an activation (n_i)
- Random initialization should have $\sigma_i = \frac{1}{\sqrt{n_i}}$

Variance Normalized initialization

- Very good for forward pass, but can still create problems for backpropagation (vanishing gradients) so it makes learning difficult
- For backpropagation, $\sigma_i = \frac{1}{\sqrt{n_{i+1}}}$ would be optimal
- Compromise, due to Glorot and Bengio (2010):

$$\sigma_i = \sqrt{\frac{2}{n_i + n_{i+1}}}$$

- This initialization allows even deeper VGG-Nets to be trained without initialization from prior models.

Better Optimizers

- Stochastic Gradient Descent (SGD) works well for optimizing DNNs, but has a few problems:
  - Difficulties when the Hessian matrix is poorly conditioned, i.e. moving back and forth in a narrow valley without making progress along the valley
  - Variance in the gradient due to stochasticity creates noise close to the minimum
  - A constant learning rate is inefficient: a small learning rate doesn‘t make progress at the beginning, a large learning rate doesn‘t converge as well at the end

SGD with momentum

- Addresses the first two of those problems
- Motivated pysically: ball rolling down a surface (with friction)
- Keep exponential moving average of the gradient with a velocity parameter

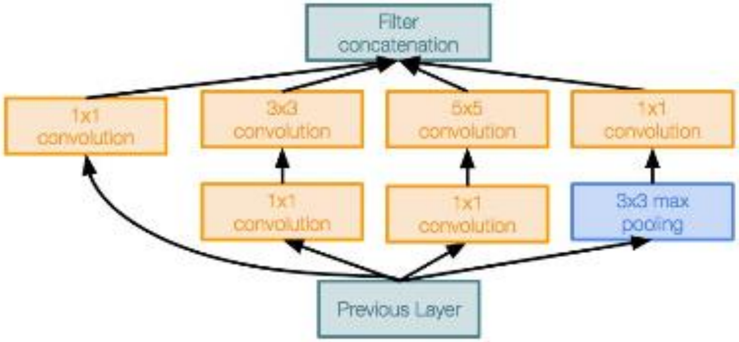
$$v \leftarrow \alpha v - \epsilon \nabla_\theta \mathcal{L}(\theta; x^{(batch)}, y^{(batch)})$$

$$\theta \leftarrow \theta + v$$

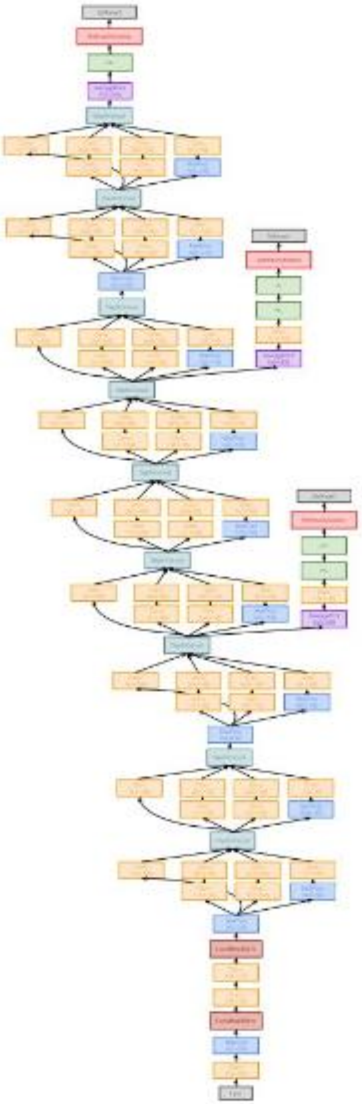
- Alpha is the momentum parameter, i.e. how slowly the velocity decays on its own, epsilon is the learning rate
- The learning rate problem is usually solved by manually decreasing the learning rate at intervals during training, other optimizers have an adaptive learning rate built in.

GoogLeNet [Szegedy et al., 2014]

- Deeper networks with computational efficiency:
  - 22 layers
  - Efficient “Inception” module
  - No FC layers
  - Only 5 million parameters
    (12x less than AlexNet)
  - ILSVRC’14 classification winner

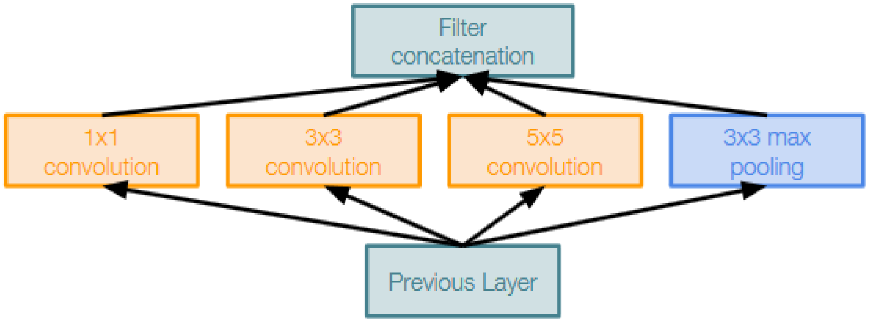


Inception module



GoogLeNet [Szegedy et al., 2014]

- “Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other
- Apply parallel filter operations on the input from previous layer:
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise

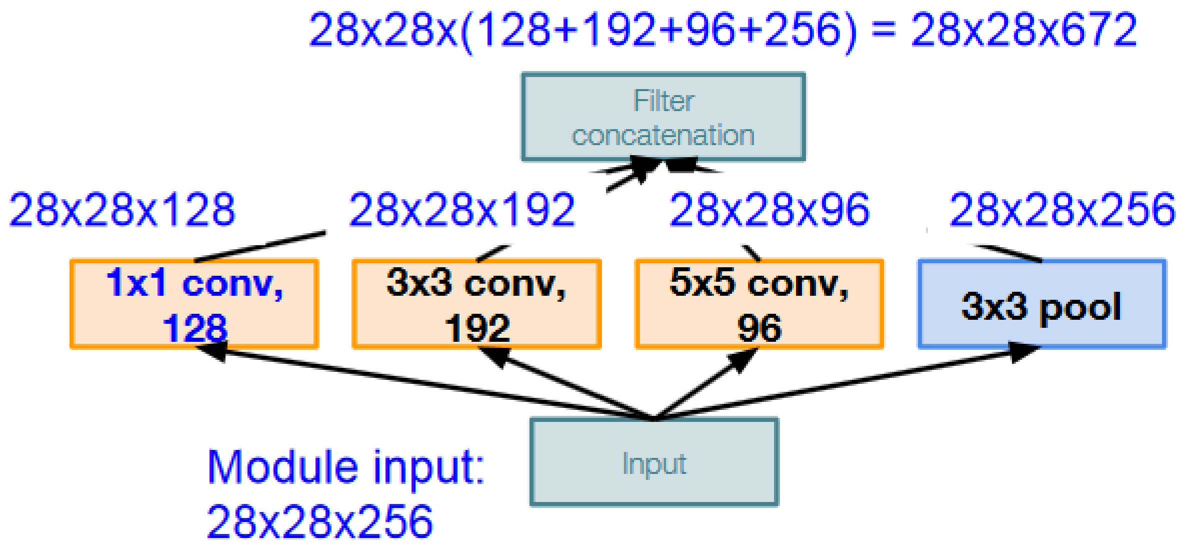


Naive Inception module

GoogLeNet [Szegedy et al., 2014]

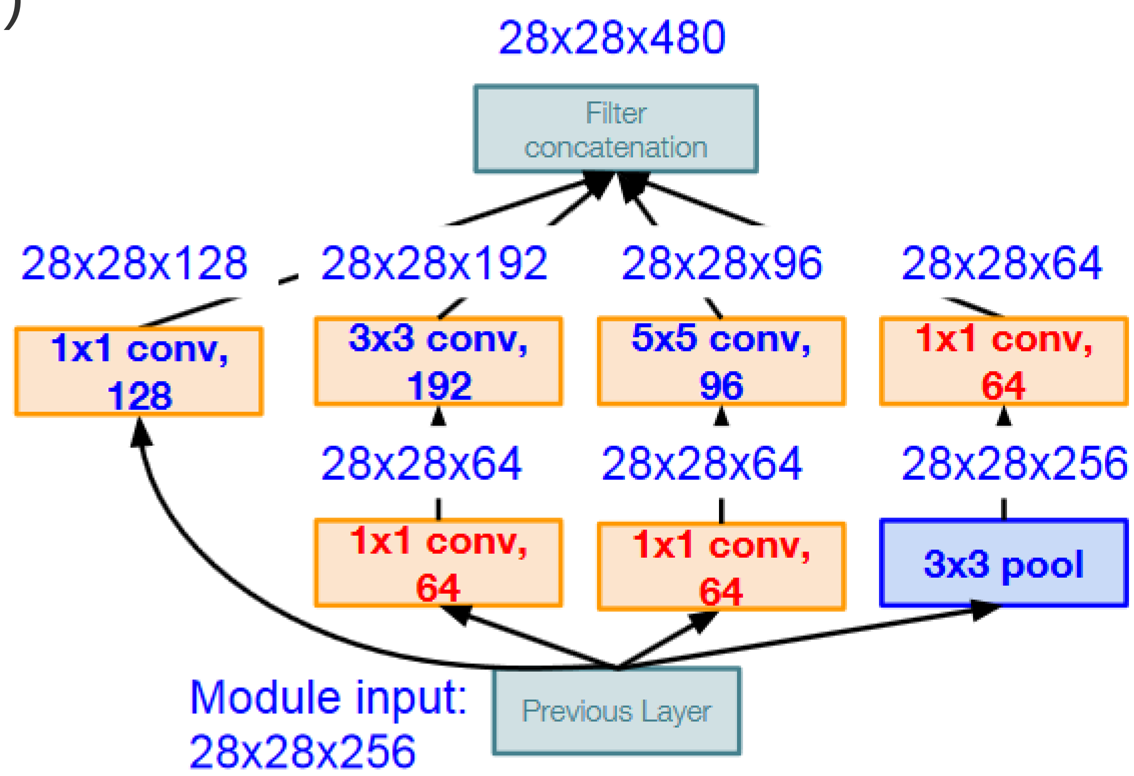
- Conv Ops:
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x256
  - [5x5 conv, 96] 28x28x96x5x5x256

  Total: 854M ops
- # of channels grows at every layer!

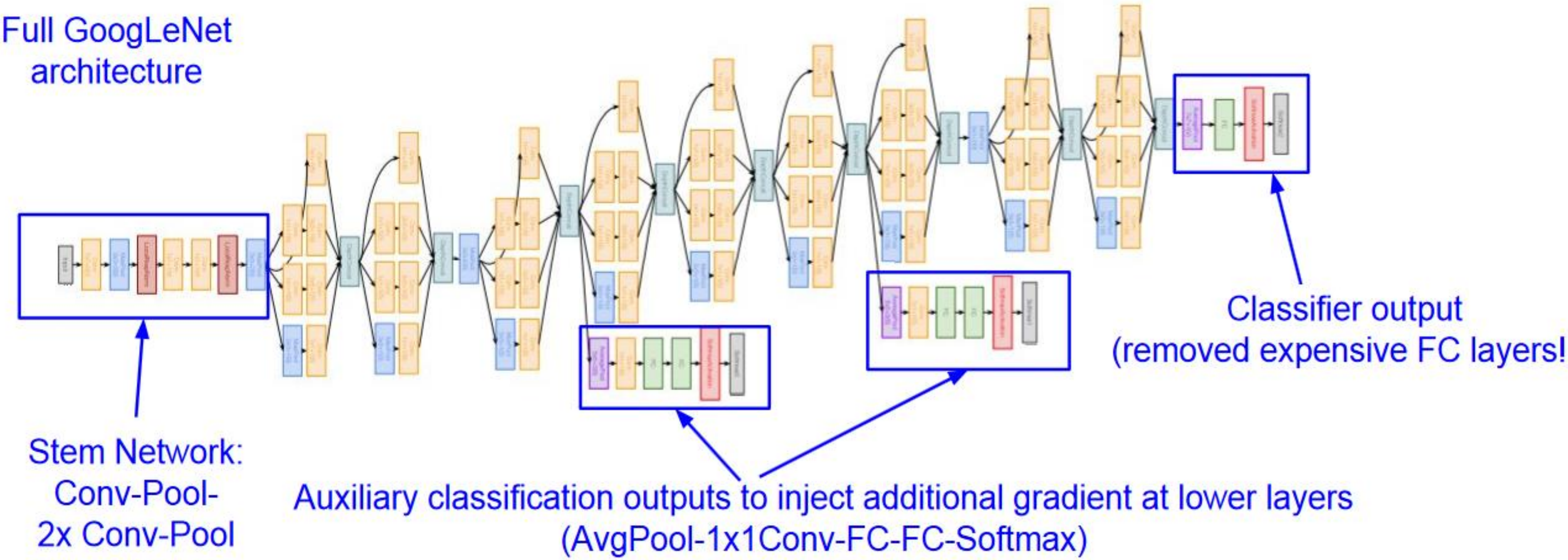


GoogLeNet [Szegedy et al., 2014]

- Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth
- Conv Ops: Total: 358M ops (vs 854M ops for naive version)



GoogLeNet [Szegedy et al., 2014]



22 total layers with weights (including each parallel layer in an Inception module)

GoogLeNet [Szegedy et al., 2014]

- Results

Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Number of models	Number of Crops	Cost	Top-5 error
1	1	1	10.07%
1	10	10	9.15%
1	144	144	7.89%
7	1	7	8.09%
7	10	70	7.62%
7	144	1008	6.67%

Batch Normalization

- Internal Covariate Shift: the change in the distribution of network activations due to the change in network parameters during training.
- Improve the training speed by fixing the distribution of the layer inputs as the training progresses.
- “you want unit gaussian activations? just make them so.” [Ioffe and Szegedy, 2015]
- Simplifications:
  - Normalize each scalar feature independently, by making it have the mean of 0 and the variance of 1.
  - Use mini-batches in stochastic gradient training, each mini-batch produces estimates of the mean and variance of each activation.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization
- Reduces the need for dropout

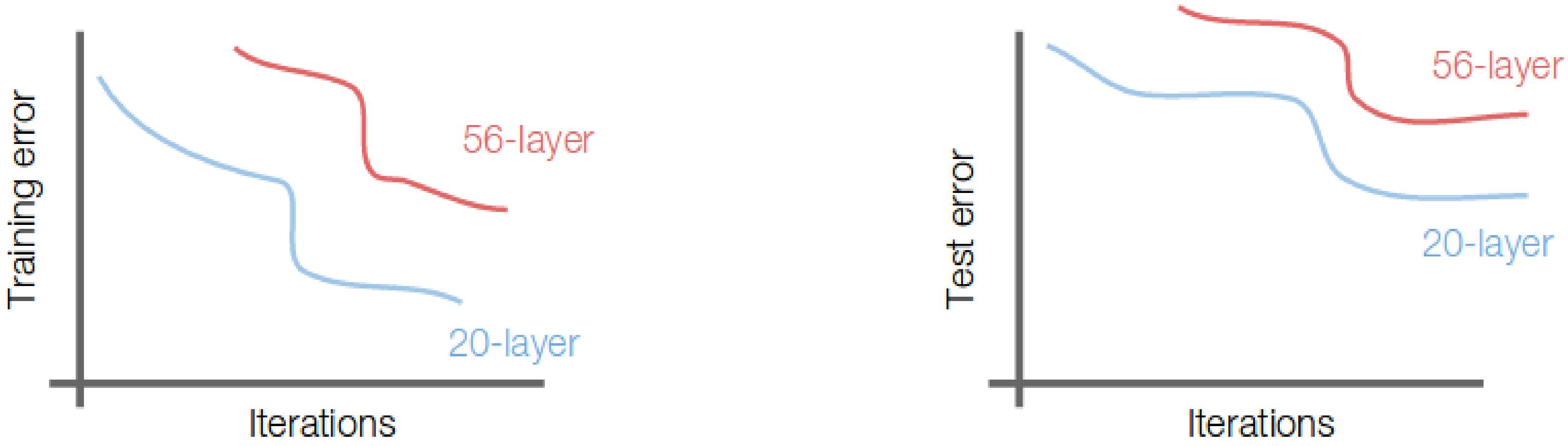
Batch Normalization

- Inception: GoogLeNet trained with initial learning rate of 0.0015.
- BN-Baseline: Same as Inception with Batch Normalization before each nonlinearity.
- BN-x5: Inception with Batch Normalization and no dropout. The initial learning rate was increased by a factor of 5, to 0.0075.
- BN-x30: Like BN-x5, but with the initial learning rate 0.045 (30 times that of Inception).
- BN-x5-Sigmoid: Like BN-x5, but with sigmoid non-linearity

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

ResNet [He et al. 2015]

- Is learning better networks as easy as stacking more layers?
- With increasing network depth, accuracy gets saturated and then degrades rapidly.
- Such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.

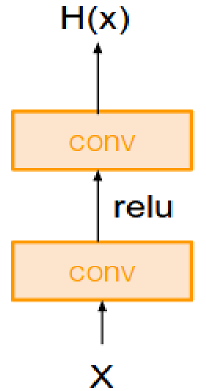


ResNet [He et al. 2015]

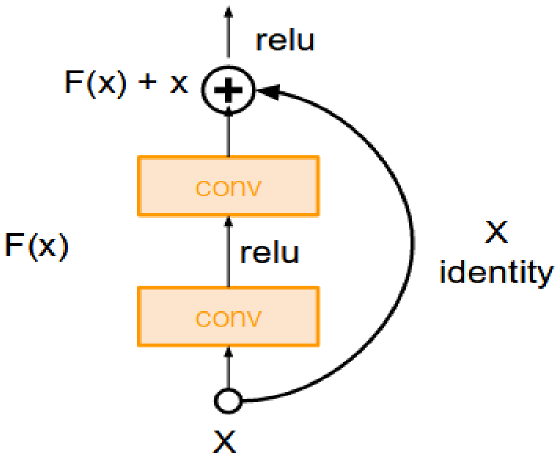
- The degradation problem is addressed by introducing a deep residual learning framework. Instead of hoping each few stacked layers directly fit a desired underlying mapping, let these layers explicitly fit a residual mapping.
- Formally, denoting the desired underlying mapping as $H(x)$, we let the stacked nonlinear layers fit another mapping of $F(x) := H(x) - x$.
- Hypothesis: it is easier to optimize the residual mapping than to optimize the original mapping. This reformulation is motivated by the counterintuitive phenomena about the degradation problem.

ResNet [He et al. 2015]

- The formulation of F(x)+x can be realized by feedforward neural networks with shortcut connections. Shortcut connections simply perform Identity mapping, and their outputs are added to the outputs of the stacked layers.
- Identity shortcut connections add neither extra parameter nor computational complexity. The entire network can be trained by SGD with backpropagation.



"Plain" layers

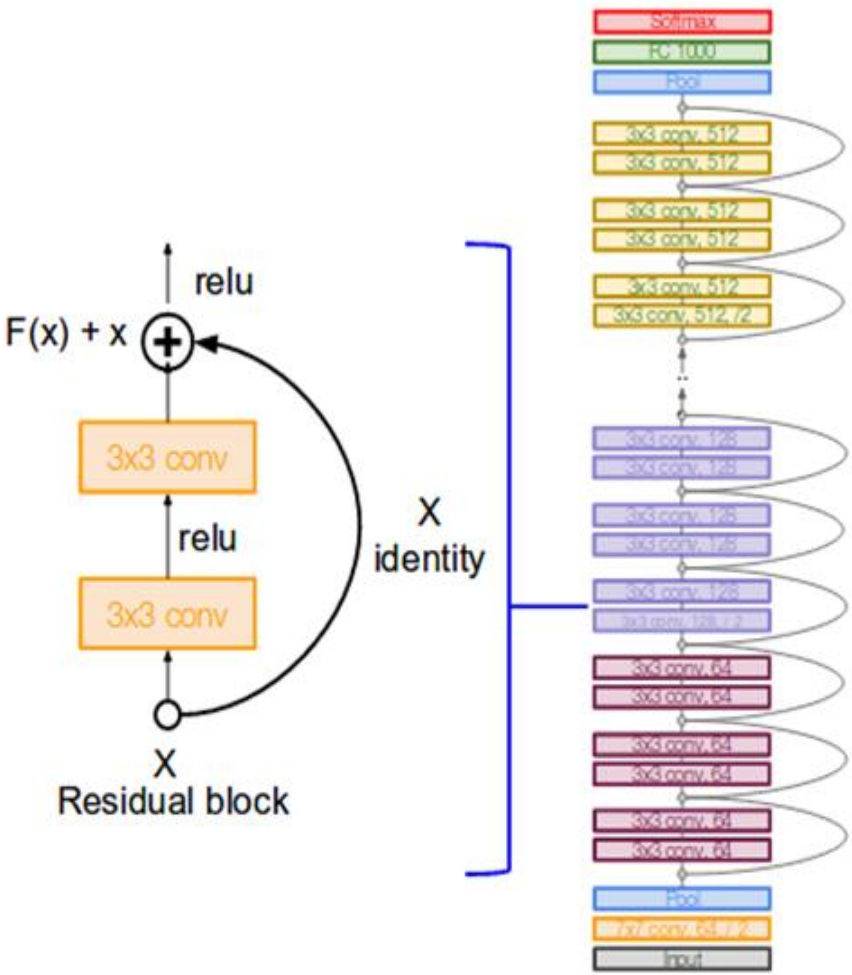


Residual block

ResNet [He et al. 2015]

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional convolutional layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)
- Total depths of 34, 50, 101, or 152 layers for ImageNet



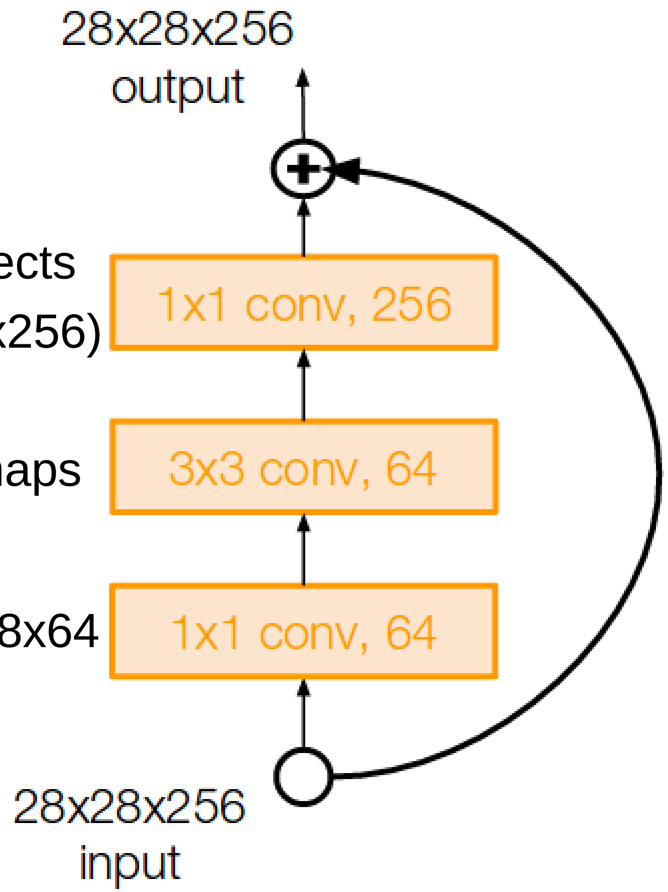
ResNet [He et al. 2015]

- For deeper networks (ResNet-50+):
  add “bottleneck” layer to improve efficiency.

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64

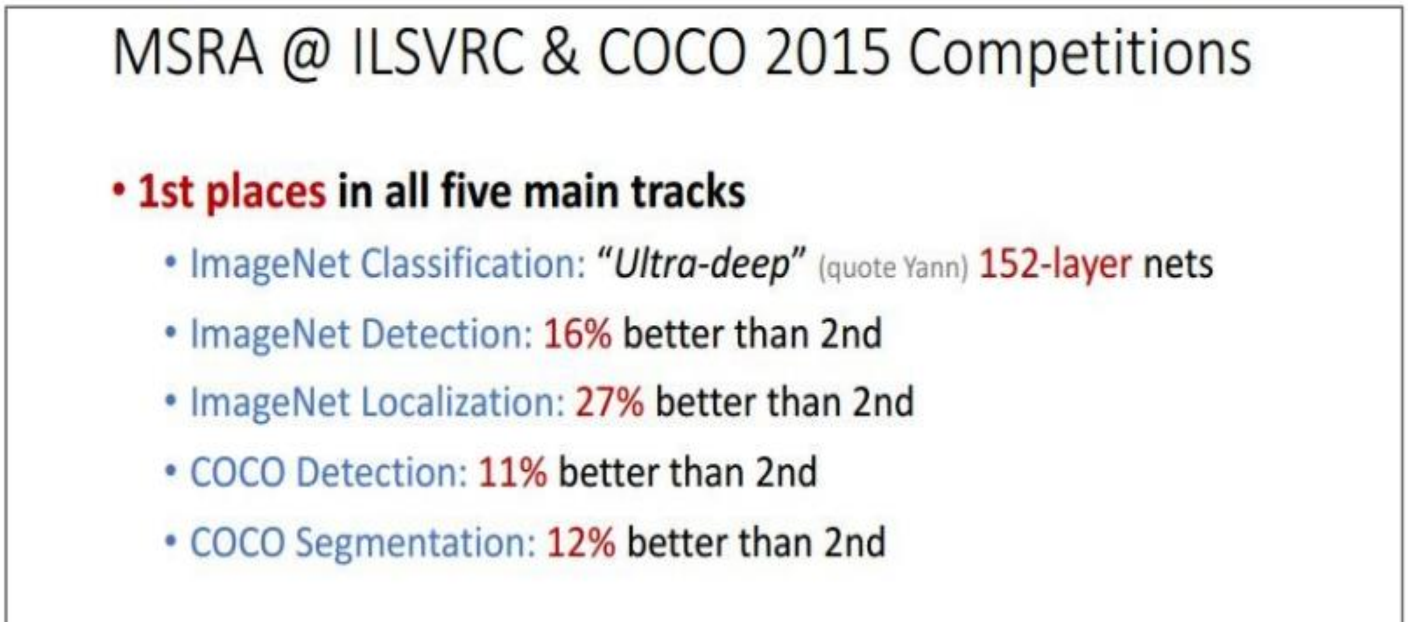


ResNet [He et al. 2015]

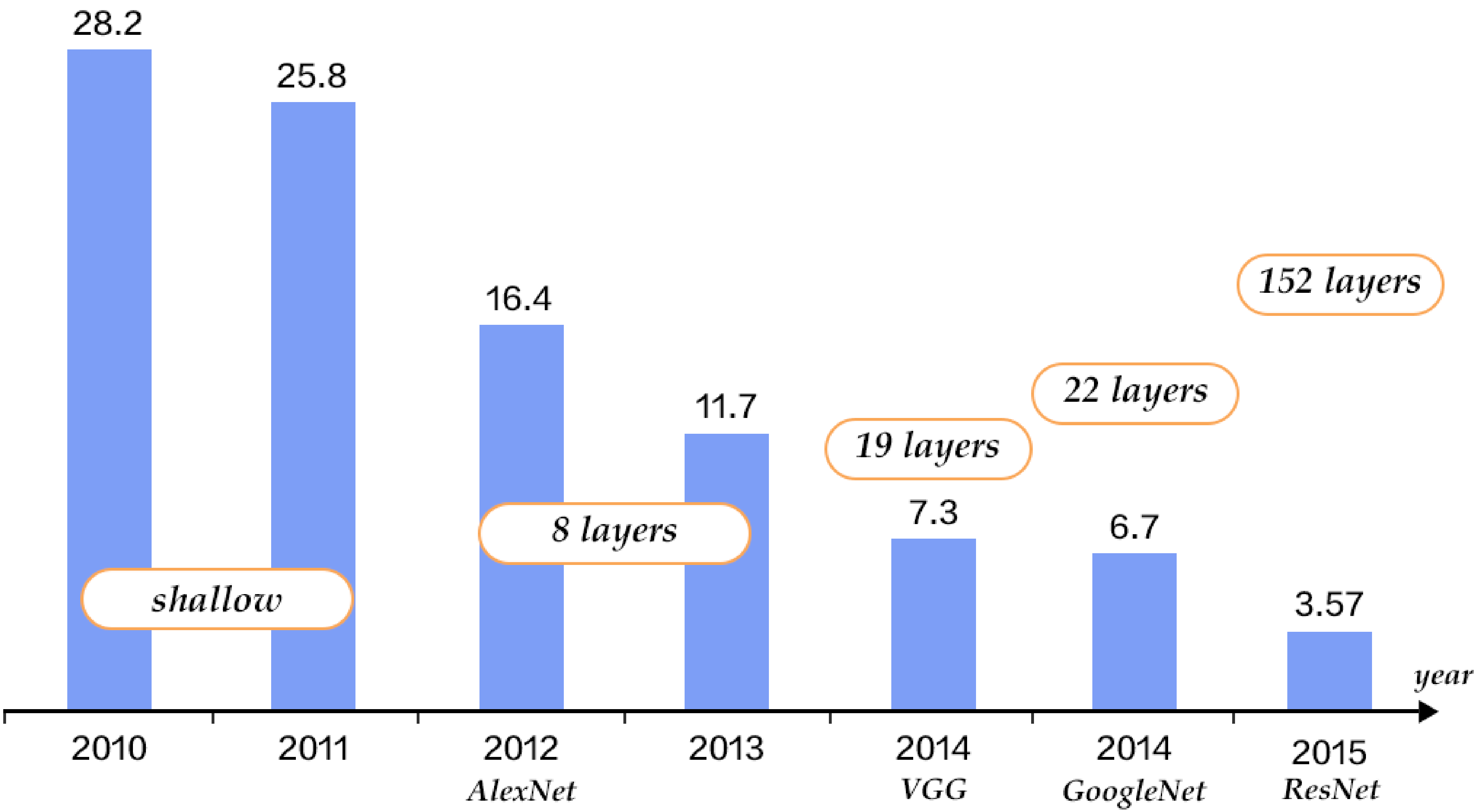
- Training ResNet in practice:
  - Batch Normalization after every convolutional layer
  - Xavier/2 initialization from He et al.
  - SGD + Momentum (0.9)
  - Learning rate: 0.1, divided by 10 when validation error plateaus
  - Mini-batch size 256
  - Weight decay of 1e-5
  - No dropout used

ResNet [He et al. 2015]

- Able to train very deep networks without degrading
- (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- 1st place in all ILSVRC and COCO 2015 competitions



ResNet [He et al. 2015]



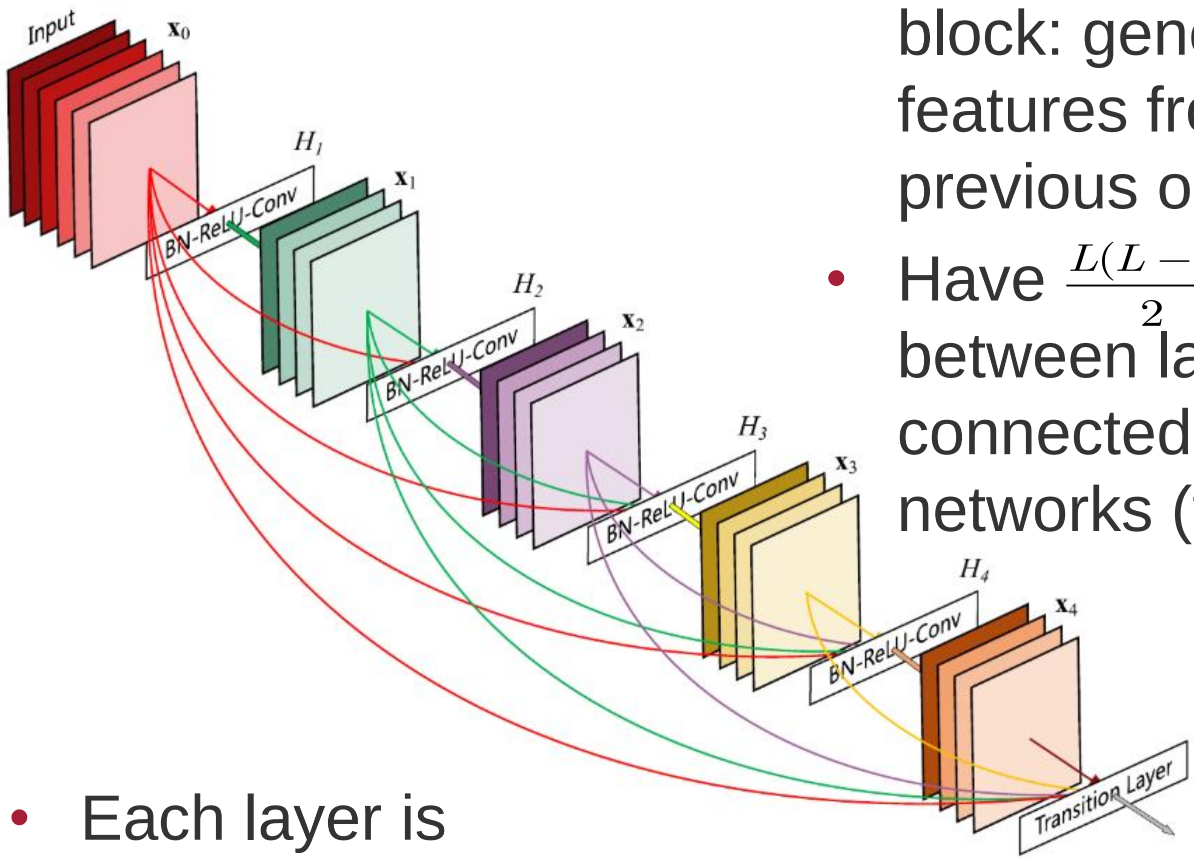
Key insights from ResNet

- Residual blocks:
  - Use identity connections to preserve gradient flow
  - Connect many elements into a block and then apply downsampling
- Can stack many residual blocks to get very deep networks
- Can still train these networks since early layers get gradients from the identity connections

How to go further?

- Idea of residual block maybe not well enough motivated
- Instead of adding the identity transform, keep all the features from earlier layers and use them as input in subsequent layers
- Make blocks where the number of filters keeps increasing

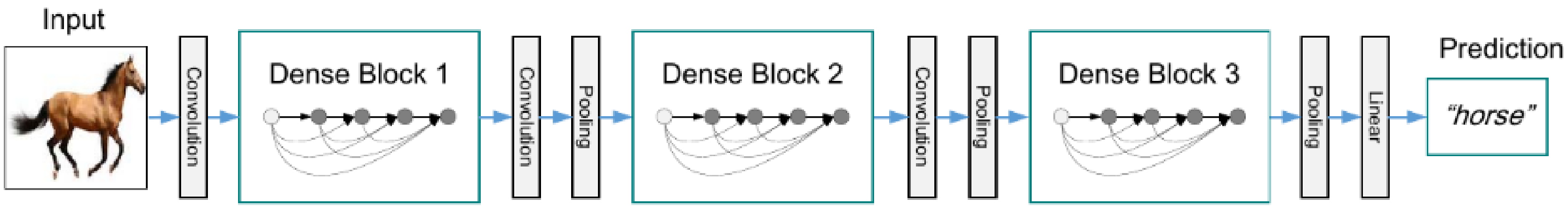
DenseNet [Huang et al. 2016]



- Central idea of a dense block: generate a few new features from all the previous ones in each layer
- Have $\frac{L(L-1)}{2}$ connections between layers => densely connected convolutional networks (title of the paper)

- Each layer is BatchNorm-ReLU-Conv

DenseNet [Huang et al. 2016]



- After each dense block there is a transition layer that applies a max pooling to reduce spatial size
- Can choose the growth rate (number of features added by each layer, called k in the paper) of the network to fit requirements in terms of input complexity and computational effort
- Initial convolution to reduce size
- Global average pool and a single FC layer at the end for classification

DenseNet [Huang et al. 2016]

- Bottleneck layers:
  - Since each layer only generates a relatively small number of features (k = 12 to 48), probably not all input features matter
  - Apply 1x1 convolutions before 3x3 convolutions to save parameters and computation time as in Inception and ResNet
  - Typically have 4*k features after the 1x1 convolution

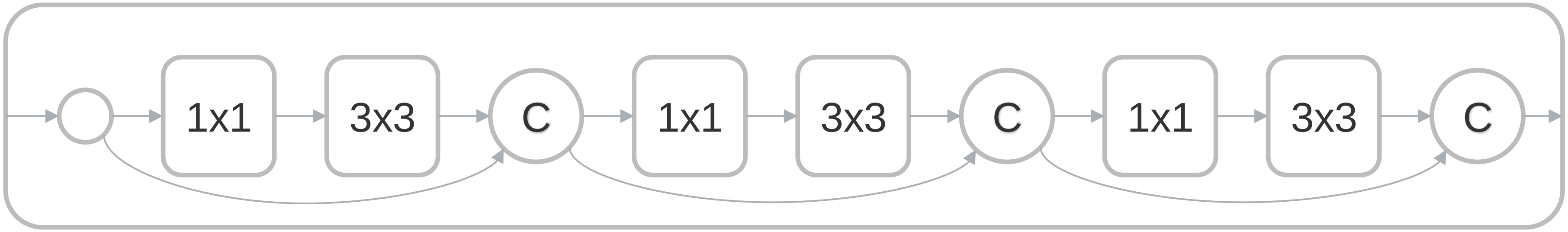
DenseNet [Huang et al. 2016]

- DenseNets usually have a lot of layers (>100)
- Compression:
  - With many layers, number of features grows quickly
  - Not all features may be relevant in later layers
  - Have some points where information is discarded
  - When using compression, add a 1x1 convolution to the transition layer that reduces number of features by a set factor θ
  - Typically, $\theta = 0.5$

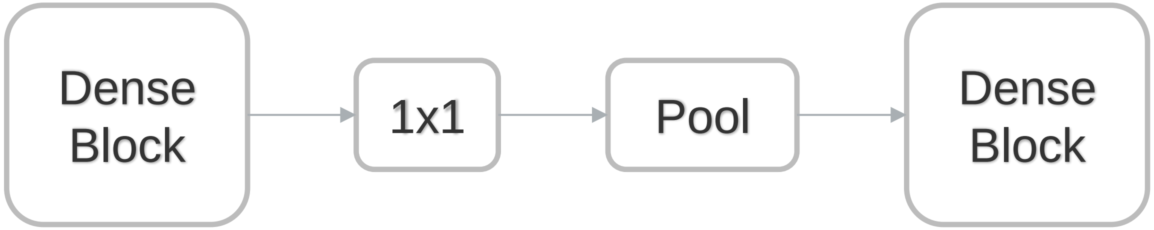
DenseNet [Huang et al. 2016]

- Example: DenseNet-161 (using Bottlenecks and Compression) for ImageNet classification
- Beginning and End basically the same as ResNet: 7x7 (stride 2) convolution and pooling (3x3, stride 2) at the start to reduce size, global average pool and a single FC layer for classification
- $k = 48$
- 4 dense blocks with 6, 12, 36 and 24 layers, with each layer being: BN-ReLU-Conv(1x1)-BN-ReLU-Conv(3x3)
- Compression Layers between dense blocks ($\theta = 0.5$) and maxpooling (2x2, stride 2)

DenseNet [Huang et al. 2016]

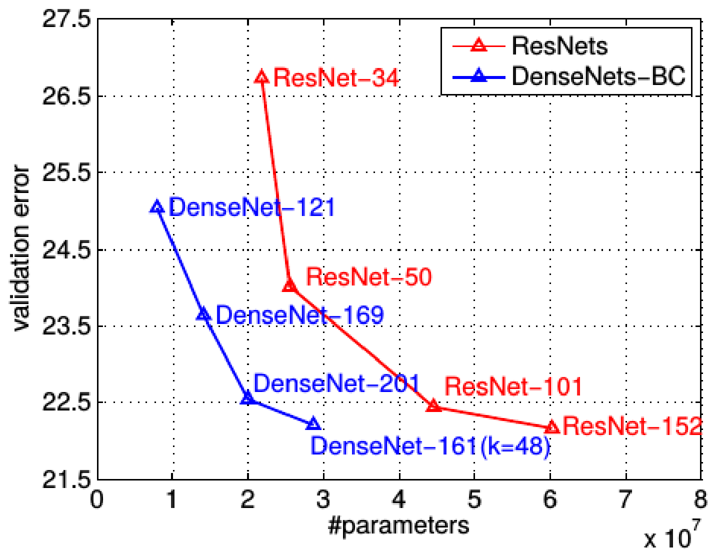


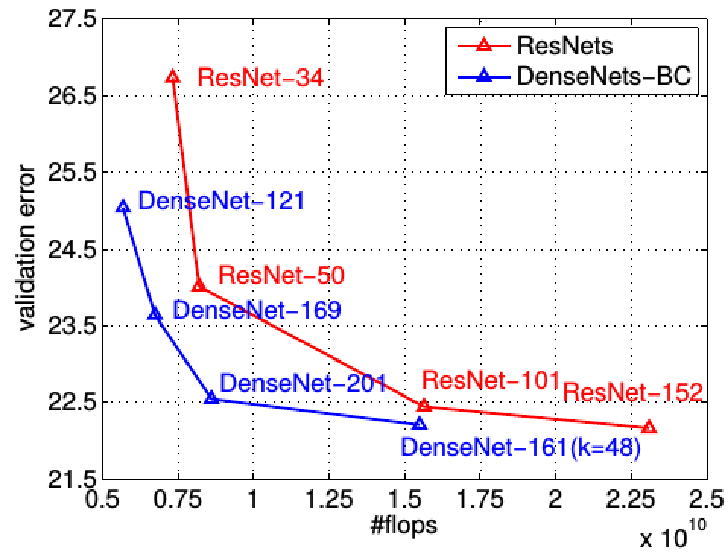
Example dense block
with bottlenecks



Example transition layer
with compression

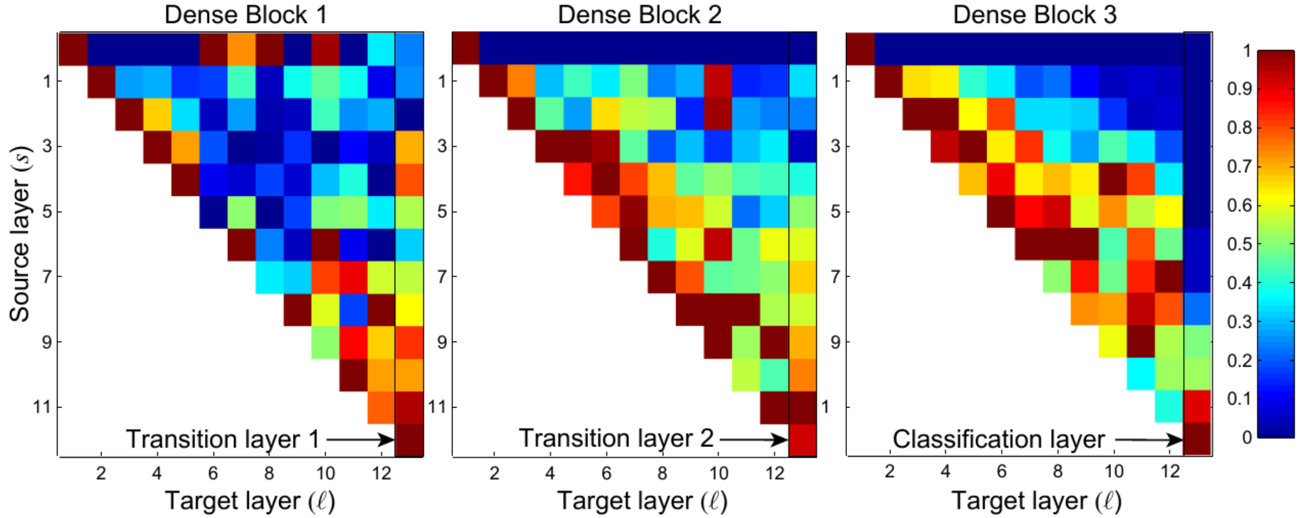
DenseNet [Huang et al. 2016]





- Comparison between DenseNet-BC and ResNet (Top-1 error)
- Consistently uses fewer parameters and less computation than ResNet to achieve (at least) the same accuracy

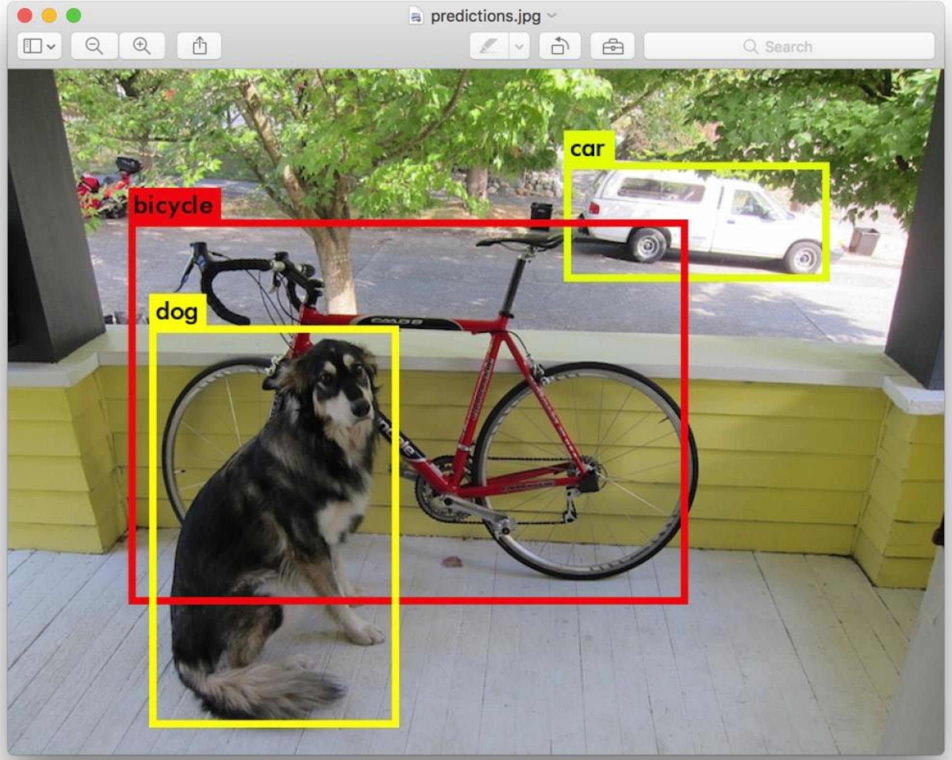
DenseNet [Huang et al. 2016]



- Validity of dense connections: typically, features from many different layers are used
- If the network did not have the shortcut connections, the information would have to move through layers another way, wasting connections

Object Detection

- Image classification is nice, but for real world applications we would like to have some data about where an object is, not just whether a picture contains a object
- One popular approach is to have the algorithm output bounding boxes annotated with an object class
- But how do we find these bounding boxes?



Object Detection approaches

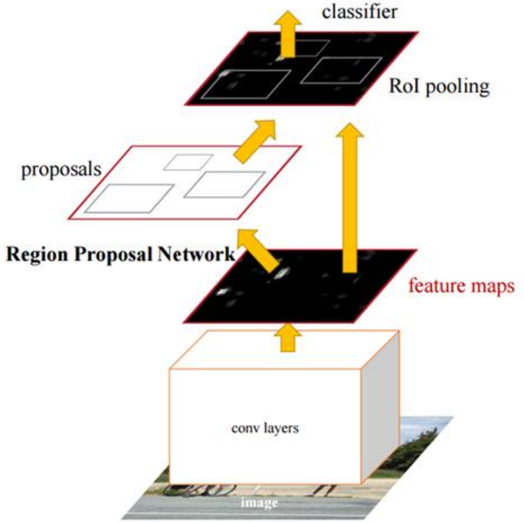
- Approach 1: Sliding window
  - Slide a window of predefined size across the image and at each step apply a classifier.
  - Keep only the positions with the highest confidence of a class
  - Can be used with almost any classifier
- Disadvantages:
  - Very (!) slow, since the classifier has to be run at each location (at every pixel or at least every n pixels)
  - Fixed bounding box size (though you could train some parameters to adjust the bounding box size)

Object Detection: R-CNN

- Approach 2: R-CNN (Region-based CNN)
  - Instead of checking every position at a single size (or even every position at multiple sizes), generate regions of interest (RoI) ahead of time and only check those positions
  - Take those regions and feed them into the classifier (warp the region if necessary to fit the classifier)
  - Much fewer classifications than the sliding window
- Disadvantages:
  - Need an additional algorithm for generating RoI (R-CNN paper uses Selective Search)
  - Still many overlapping patches being processed (~2000)

Object Detection: R-CNN

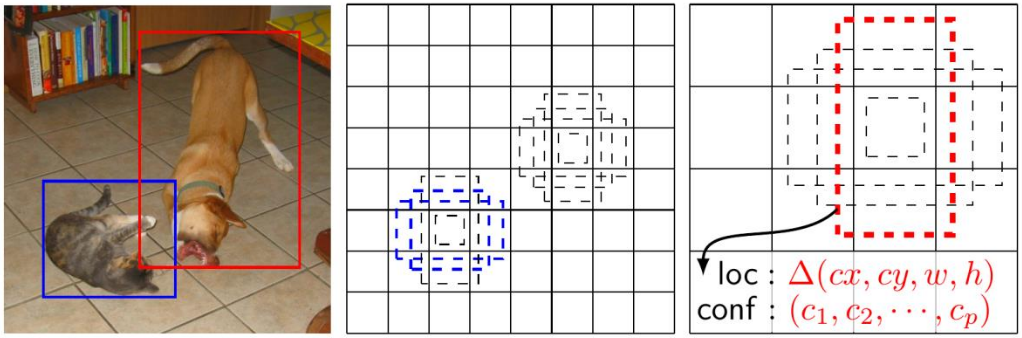
- Fast R-CNN:
  - Feed the whole image through a CNN to generate features
  - Collect the features from an RoI and use them to classify that proposal
  - Only uses the expensive pass through the CNN once
- Faster R-CNN:
  - Use a CNN to generate RoIs with the same features used for classification
  - Much better reuse of work
  - Can be trained end-to-end



Faster R-CNN workflow

Object Detection: YOLO/SSD

- Approach 3: YOLO/SSD
  - Have a set of default bounding boxes distributed throughout the image and of different sizes
  - For each of those default bounding boxes generate class confidence scores and scale/offset corrections
  - This can be done in a single pass in a CNN
  - Hence the names „You Only Look Once“ and „Single Shot Detector“
  - Much faster (>10 x) than Faster R-CNN, but only slightly less accurate



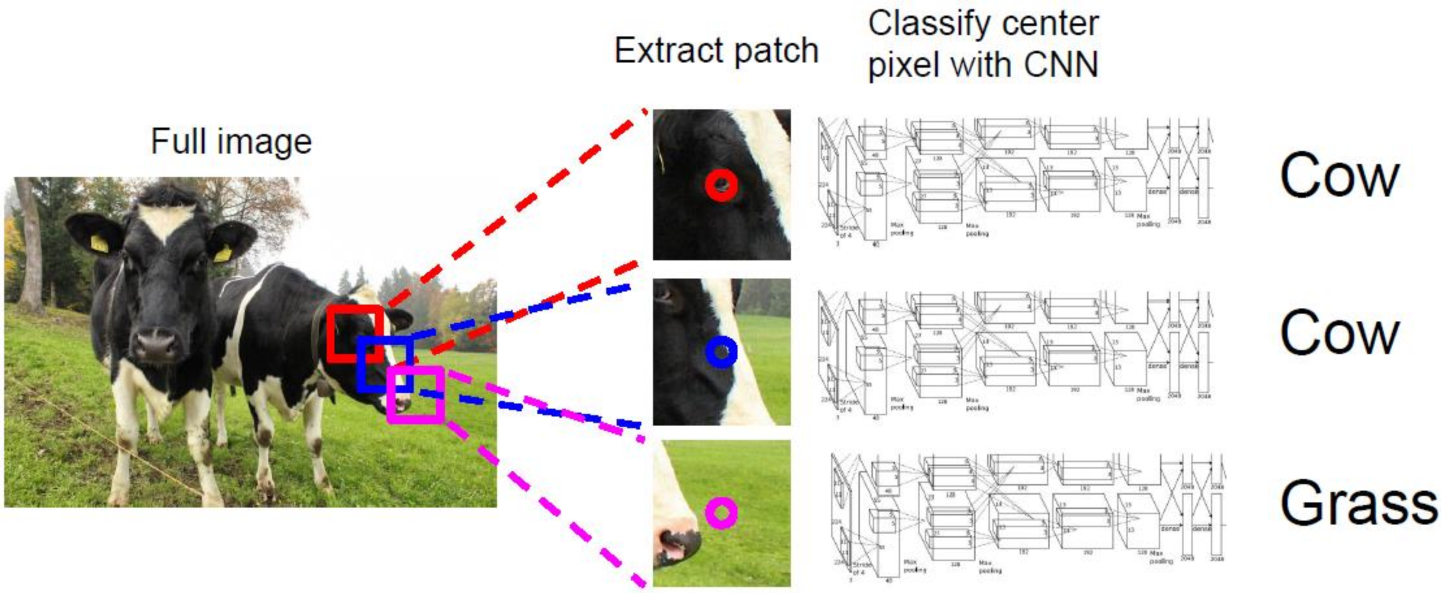
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Semantic Segmentation

- Prediction at pixel level: next step in the progression from coarse to fine inference
- Label each pixel in the image with a category label
- Don’t differentiate instances, only care about pixels
- Transfer of recent success in classification to dense prediction by reinterpreting classification nets and fine-tuning their learned representations.

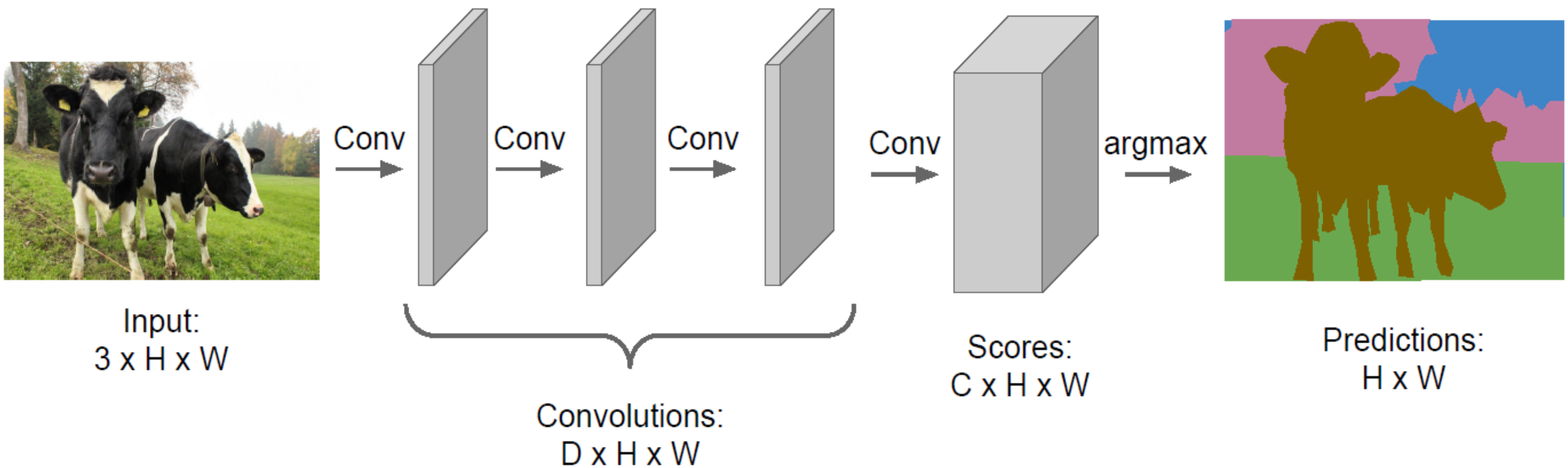
1. Attempt: Sliding Window

- Label exactly one pixel per iteration:
  - Consider a new patch
  - Classify the central pixel
- Problem:
  - Very inefficient and time consuming



2. Attempt: Fully Convolutional Nets

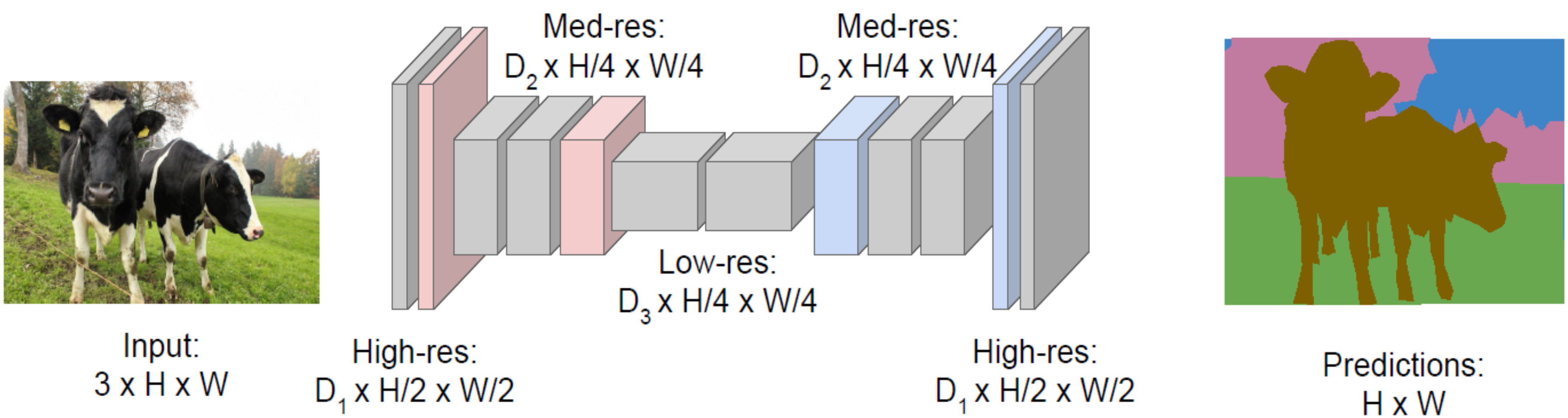
- Remove fully connected layers and change them with convolutional kernels to preserve the depth
- Merge the input feature maps of the last layer to predictions using again a 1x1 kernel per class
- Assign the class with the highest logit to the pixel
- Convolutions at original input resolution are expensive



3. Attempt: Mirror Networks

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network.
- Downsampling:
  - Max pooling, strided convolutions
- Upsampling: ?



In-Network upsampling: “Unpooling”

- Nearest neighbor: fill all cells with the same activation
- „Bed of Nails“: place the activation in the upper left corner of the 2x2 patch and fill the remaining cells with 0

Nearest Neighbor

1	2
3	4

Input: 2 x 2

1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

“Bed of Nails”

1	2
3	4

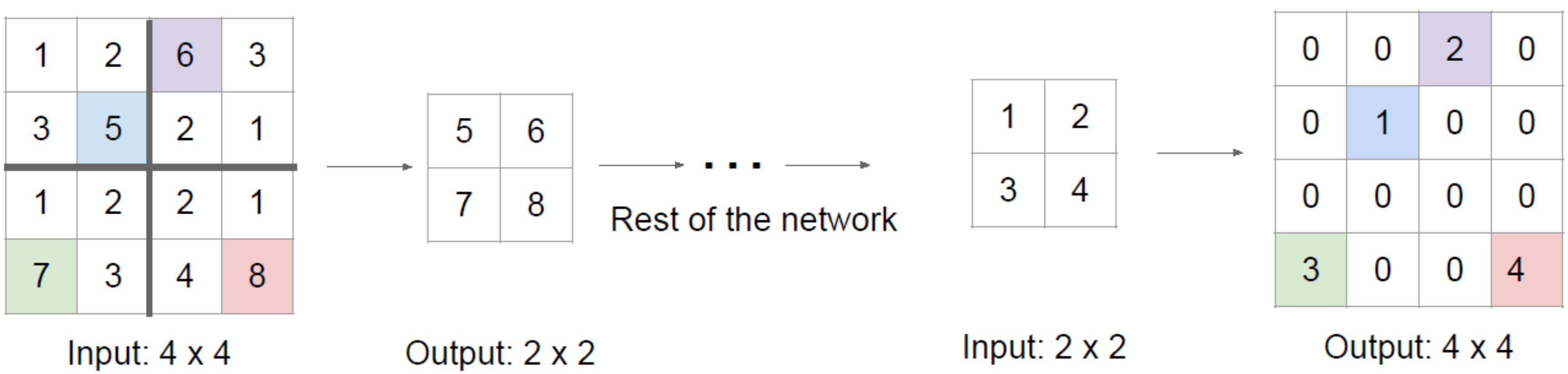
Input: 2 x 2

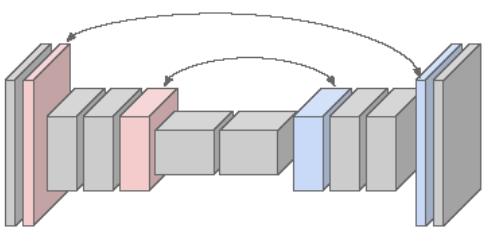
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

Max Unpooling

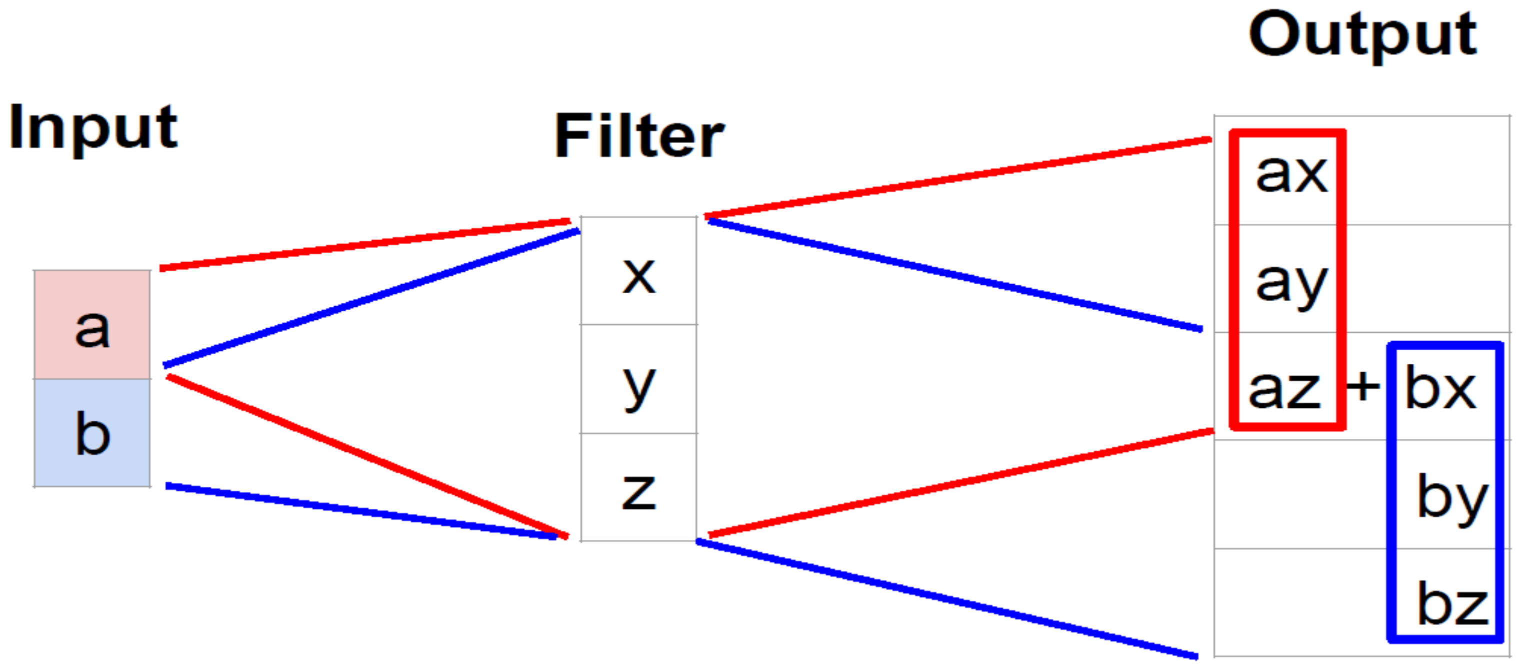
- Add connections between Corresponding pairs of downsampling and upsampling layers
- Remember positions of maxima in the downsampling path and use them in the upsampling path





Learnable Upsampling

- 3x3 transpose convolution, stride 2 pad 1:
  - Filter moves 2 pixels in the output for every one pixel in the input
- Output contains copies of the filter weighted by the input, summing at where at overlaps in the output



3 Nets for Semantic Segmentation

- Fully convolutional Network [Lang et al. 2014]
  - VGG-Net
  - Fully convolutional layer + learnable upsampling
  - Combine information from several layers
- DeconvNet [Noh et al. 2015]
  - VGG-Net
  - Fully convolutional layer + Max Unpooling
- Tiramisu [Jegou et al. 2016]
  - DenseNet
  - Skip connections
  - Learnable upsampling

Links of Interesting Papers

- AlexNet: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- Dropout: https://arxiv.org/pdf/1207.0580.pdf
- VGG-Net: https://arxiv.org/pdf/1409.1556.pdf
- Initialialization: https://arxiv.org/pdf/1502.01852.pdf
- GoogLeNet: https://arxiv.org/pdf/1409.4842.pdf
- Batch normalization: https://arxiv.org/pdf/1502.03167.pdf
- ResNet: https://arxiv.org/pdf/1512.03385v1.pdf
- DenseNet: https://arxiv.org/pdf/1608.06993.pdf

Links of Interesting Papers

- Overfeat: https://arxiv.org/pdf/1312.6229.pdf
- Faster R-CNN: https://arxiv.org/pdf/1506.01497.pdf
- Yolo: https://arxiv.org/pdf/1612.08242.pdf
- SSD: https://arxiv.org/pdf/1512.02325v2.pdf
- FCN: https://arxiv.org/pdf/1411.4038.pdf
- DeconvNet: https://arxiv.org/pdf/1505.04366.pdf
- SegNet: https://arxiv.org/pdf/1511.00561.pdf
- Tiramisu: https://arxiv.org/pdf/1611.09326.pdf