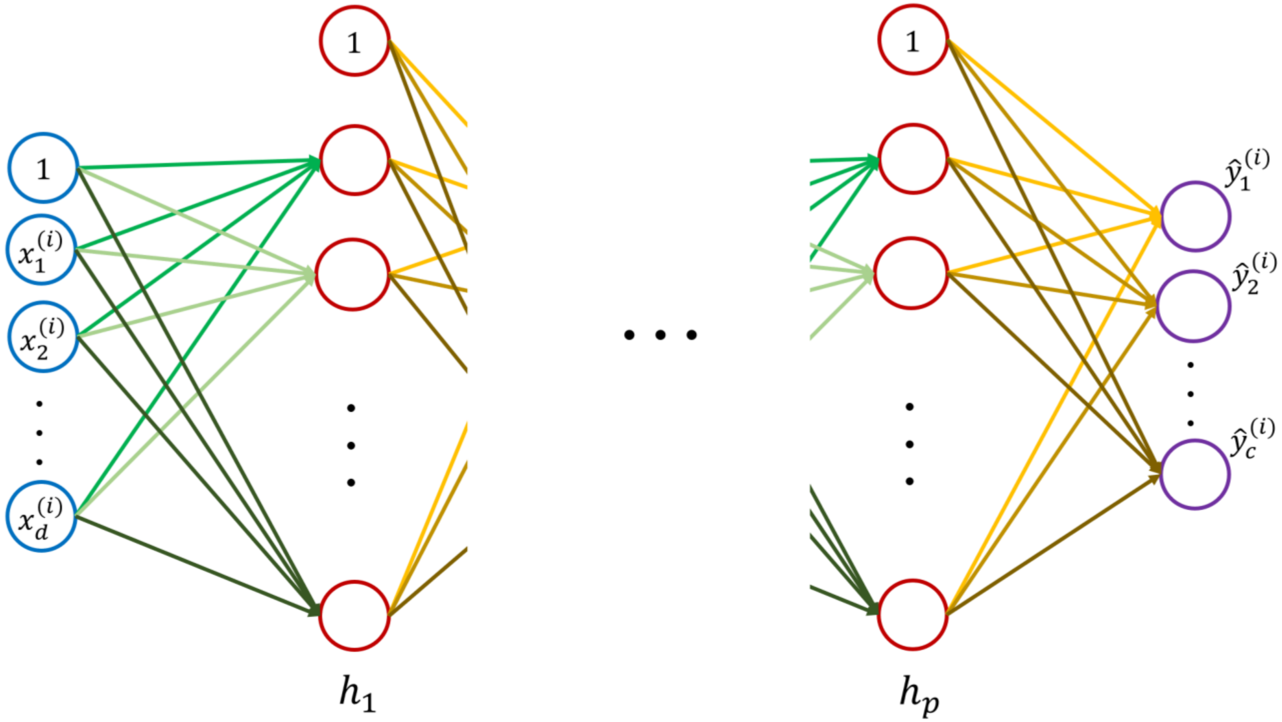
Avisek Gupta (avisek003@gmail.com)

Machine Learning

# 15 – Neural Networks and Deep Learning

November 04, 2022

# Multi-Layered Perceptrons



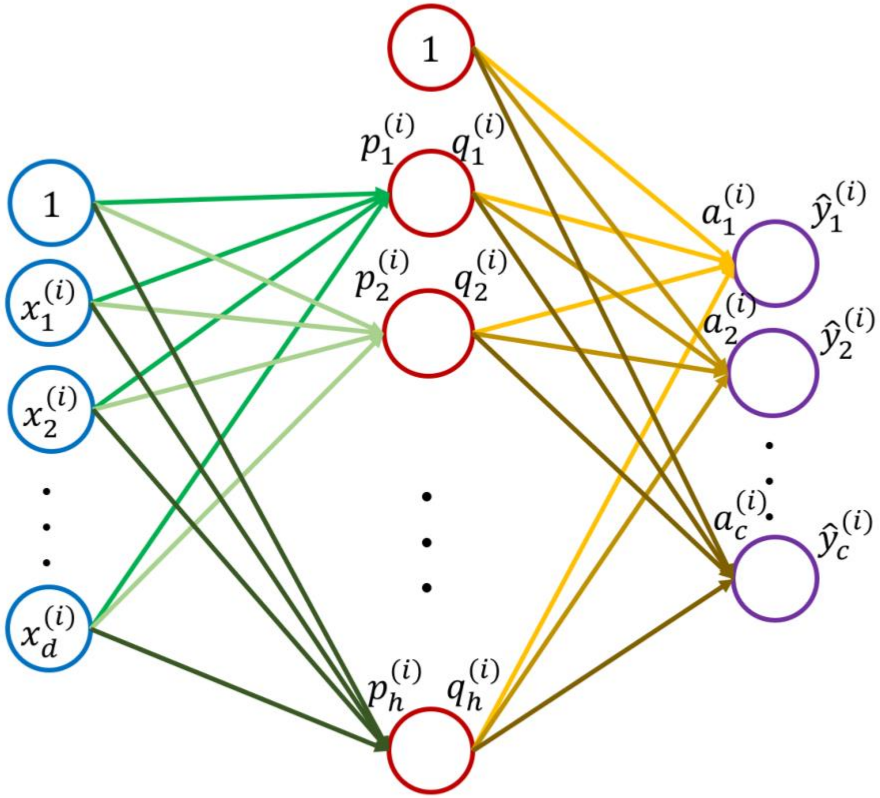

A Multi-Layered Perceptron with $p$ number of hidden layers can be represented as,

$$\hat{\mathbf{y}} = f_{p+1}(\ldots f_2(f_1(\mathbf{x})))$$

where each layer is an affine transformation followed by a possible non-linear transformation $\sigma$,

$$f_i(\mathbf{z}) = \sigma((W^i)^T \mathbf{z} + b^i)$$

# Motivation: Deep Neural Networks




A single hidden layer network can be represented as,

$$\mathbf{q} = f_1(\mathbf{x}), \quad \hat{\mathbf{y}} = f_2(\mathbf{q})$$

or,

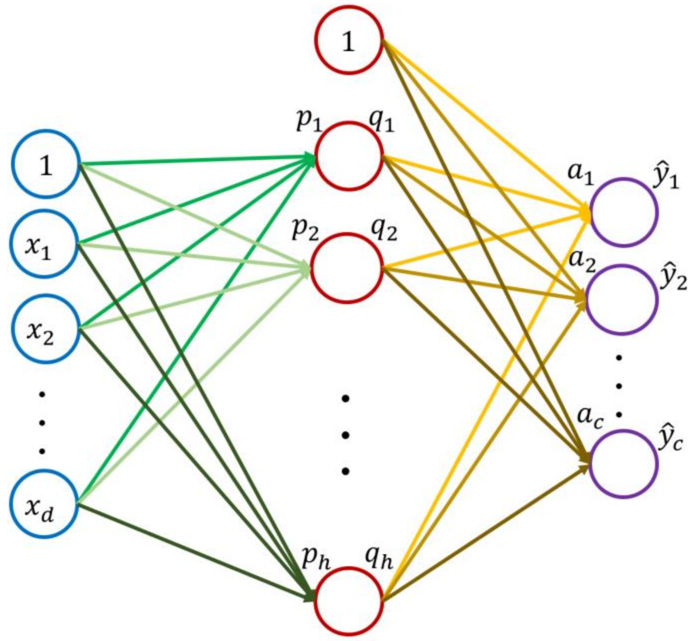
$$\hat{\mathbf{y}} = f_2(f_1(\mathbf{x}))$$

# Multi-Layered Perceptron

Feedforward:

$$p_j = \sum_{l=1}^{d} W_{lj}^{(1)} x_l + b_j^{(1)}, \; q_j = \sigma(p_j)$$

$$a_k = \sum_{l'=1}^{h} W_{l'k}^{(2)} q_{l'} + b_k^{(2)}, \; \hat{y}_k = \sigma(a_k), \; J = \frac{1}{2}\sum_{k=1}^{c}(t_k - \hat{y}_k)^2$$




---

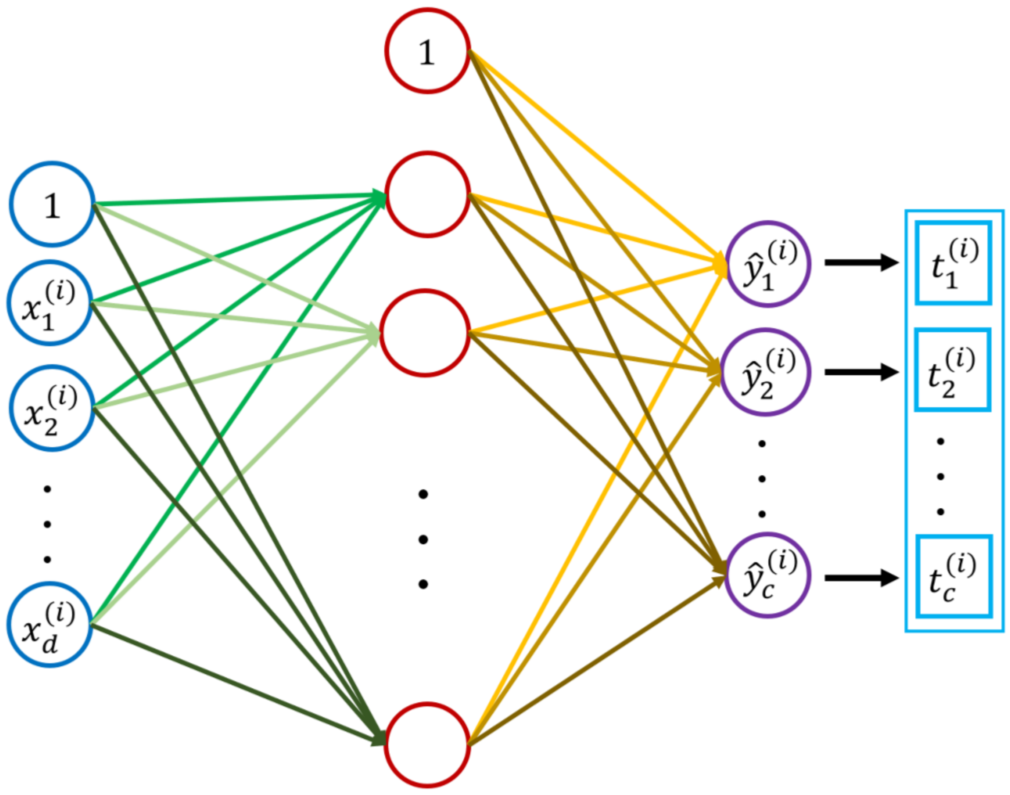
Backpropagation:

$$\frac{\partial}{\partial W_{l'k}^{(2)}} J = -(t_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k) q_{l'}, \quad \frac{\partial}{\partial b_k^{(2)}} J = -(t_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k),$$

$$\frac{\partial}{\partial W_{lj}^{(1)}} J = \sum_{k=1}^{c} \left[ -(t_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k) W_{jk}^{(2)} \right] q_j(1 - q_j) x_l,$$

$$\frac{\partial}{\partial b_j^{(1)}} J = \sum_{k=1}^{c} \left[ -(t_k - \hat{y}_k)\hat{y}_k(1 - \hat{y}_k) W_{jk}^{(2)} \right] q_j(1 - q_j).$$

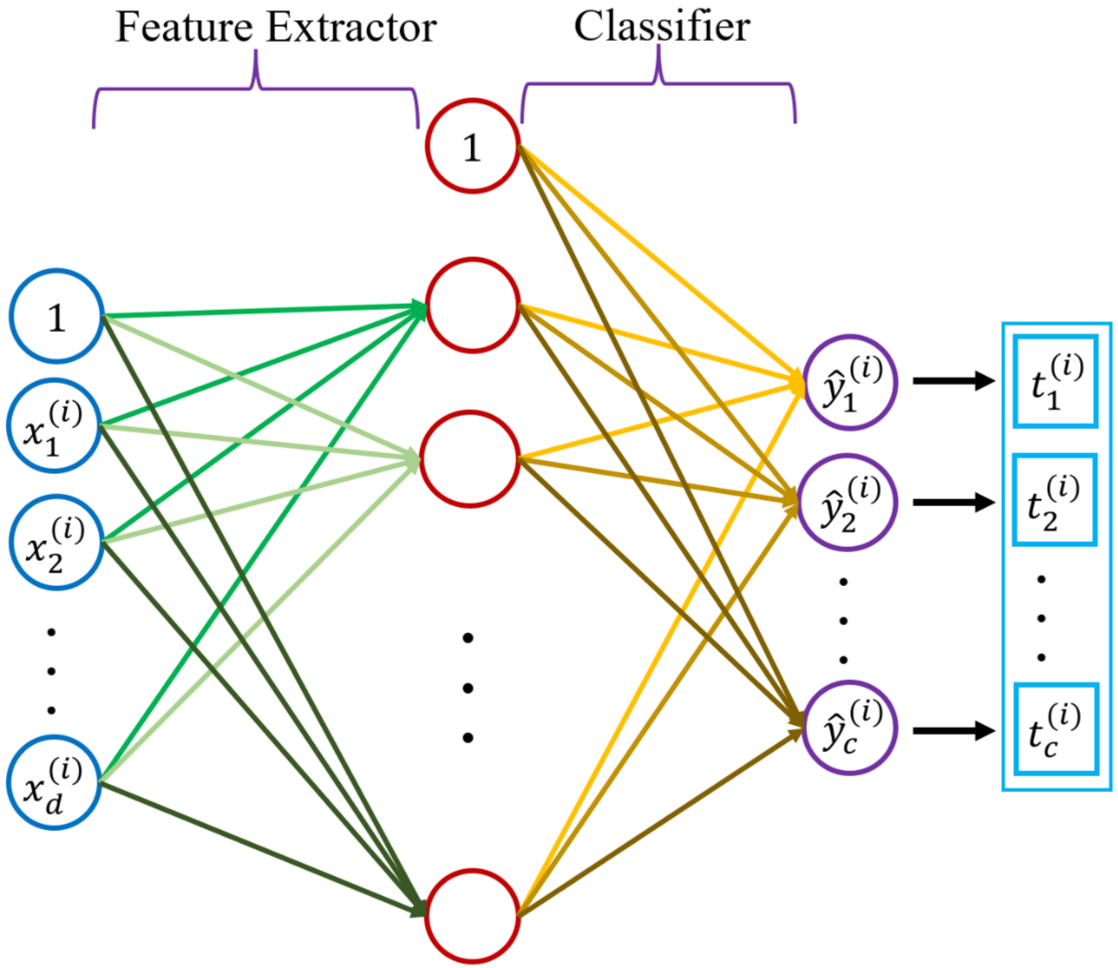
# Multi-Layered Perceptron for Classification




For each data instance $\mathbf{x}^{(i)}$, the MLP estimates a vector $\hat{\mathbf{y}}^{(i)}$, and compares it with the ground truth one-hot vector $\mathbf{t}^{(i)}$.

Several loss functions are available that can be used: MSE, CE, ...

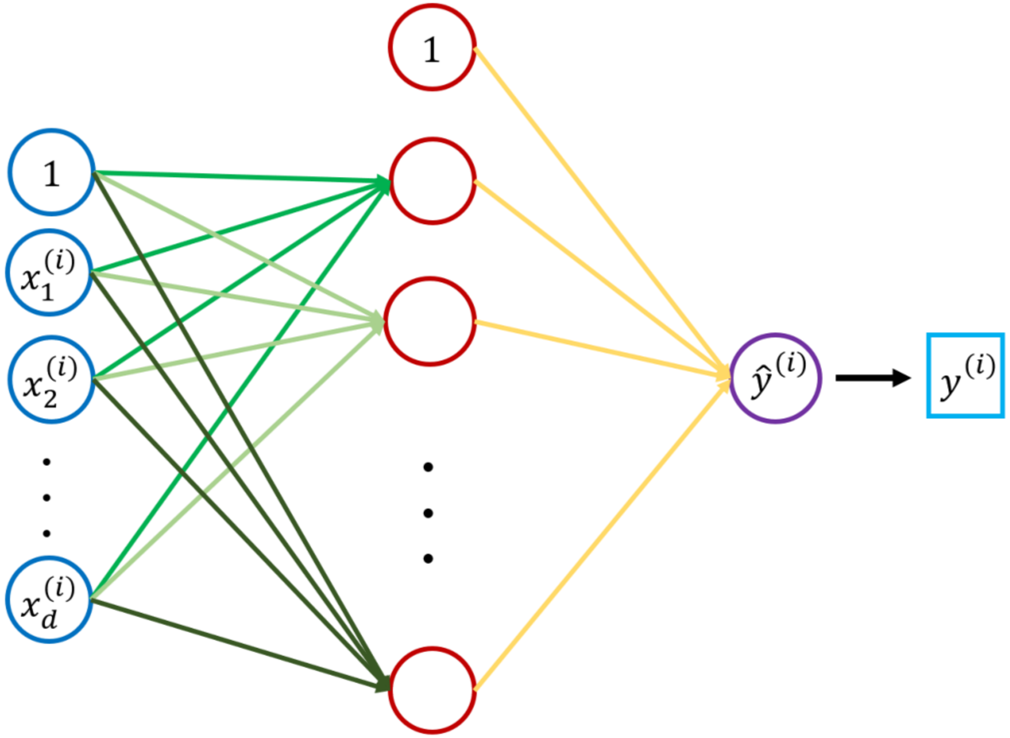
# Multi-Layered Perceptron for Classification




For each data instance $\mathbf{x}^{(i)}$, the MLP estimates a vector $\hat{\mathbf{y}}^{(i)}$, and compares it with the ground truth one-hot vector $\mathbf{t}^{(i)}$.

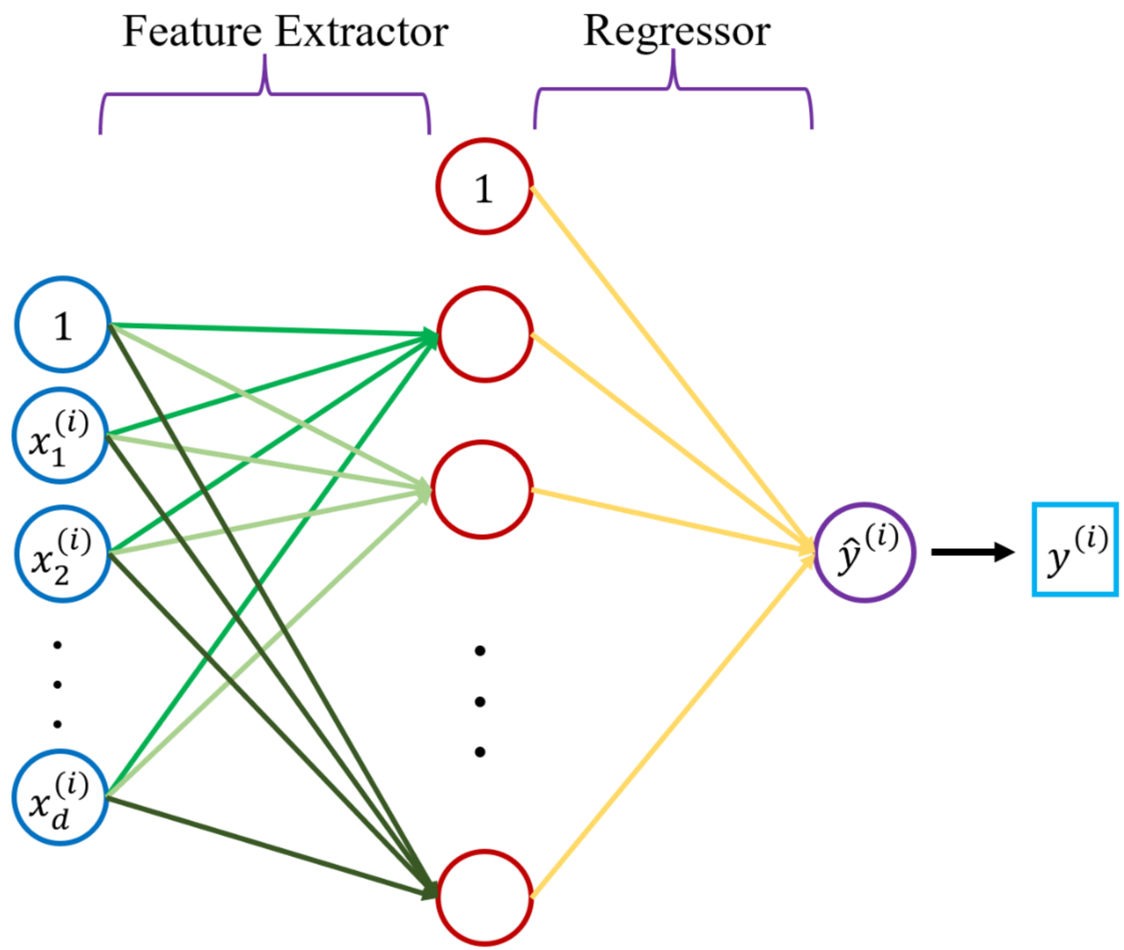
Several loss functions are available that can be used: MSE, CE, ...

# Multi-Layered Perceptron for Regression



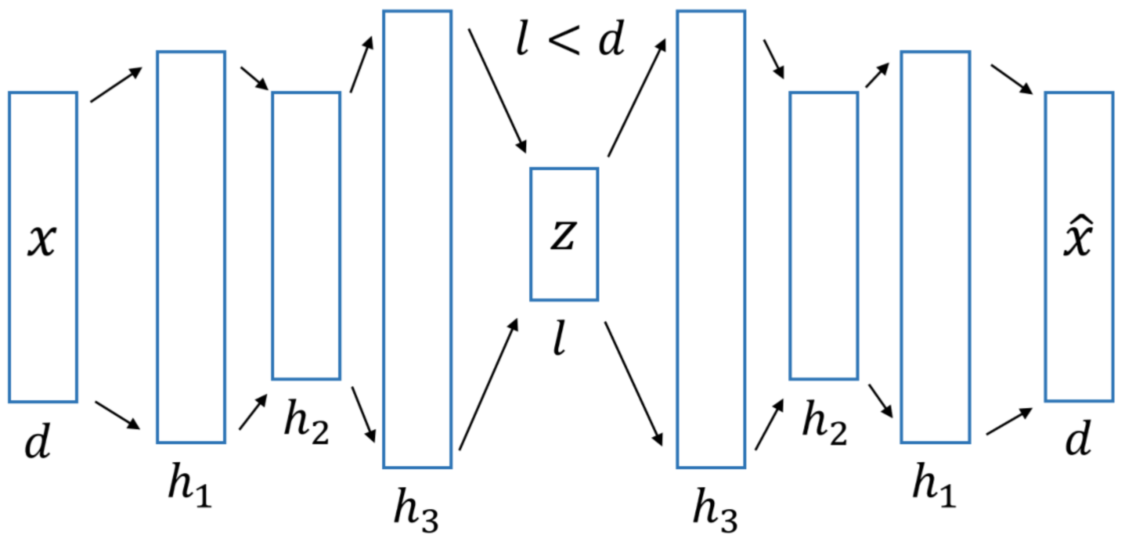

For each data instance $\mathbf{x}^{(i)}$, the MLP estimates a target value $\hat{y}^{(i)}$, and compares it with the ground truth target $y^{(i)}$.

# Multi-Layered Perceptron for Regression




For each data instance $\mathbf{x}^{(i)}$, the MLP estimates a target value $\hat{y}^{(i)}$, and compares it with the ground truth target $y^{(i)}$.

# Multi-Layered Perceptron for Dimension Reduction: Autoencoders

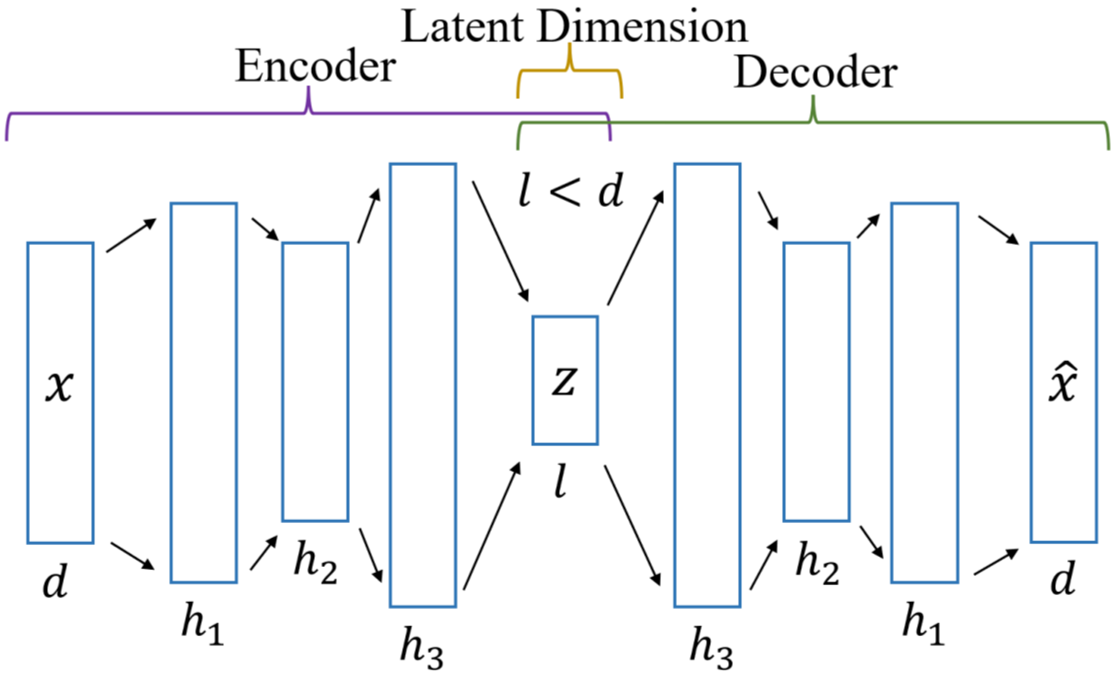



Each data instance $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is mapped by an *encoder network* to a lower dimensional latent vector $\mathbf{z}^{(i)} \in \mathbb{R}^l$, $l < d$. A *decoder network* maps the latent vector $\mathbf{z}^{(i)}$ back to the original dimension $\hat{\mathbf{x}}^{(i)}$. [MSE / CE loss can be used]

Reproducing the data instances makes the autoencoder network learn to:

- Map each data instance $\hat{\mathbf{x}}^{(i)}$ to a unique $\mathbf{z}^{(i)}$.
- Map similar data instances close to each other in the latent space.

# Multi-Layered Perceptron for Dimension Reduction: Autoencoders

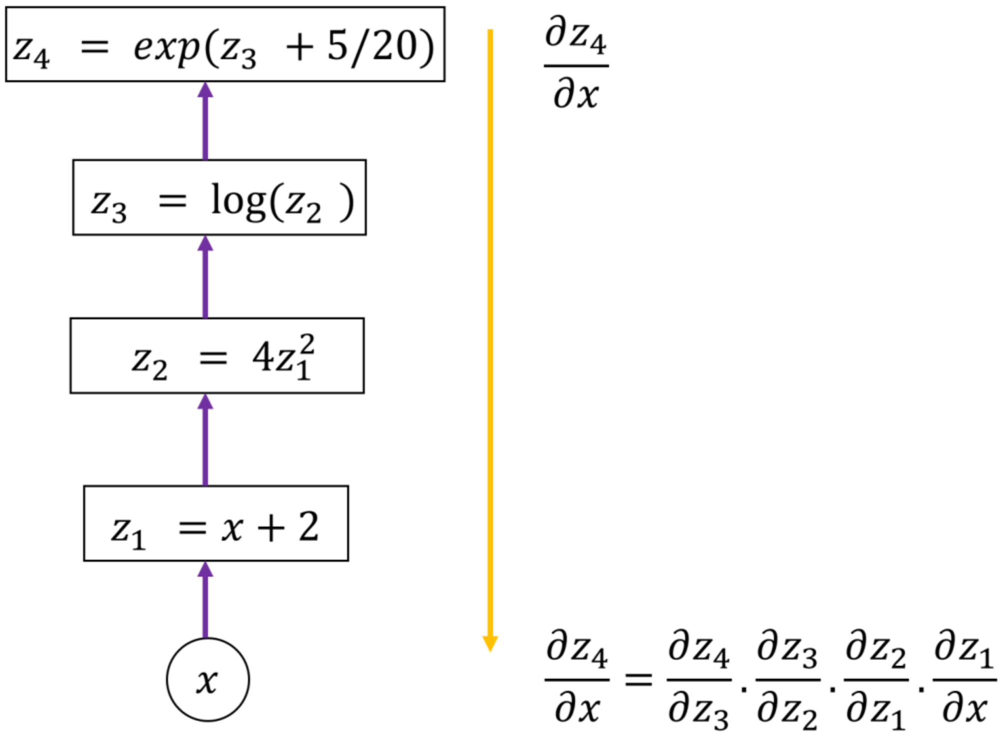



Each data instance $\mathbf{x}^{(i)} \in \mathbb{R}^d$ is mapped by an *encoder network* to a lower dimensional latent vector $\mathbf{z}^{(i)} \in \mathbb{R}^l$, $l < d$. A *decoder network* maps the latent vector $\mathbf{z}^{(i)}$ back to the original dimension $\hat{\mathbf{x}}^{(i)}$. [MSE / CE loss can be used]

Reproducing the data instances makes the autoencoder network learn to:

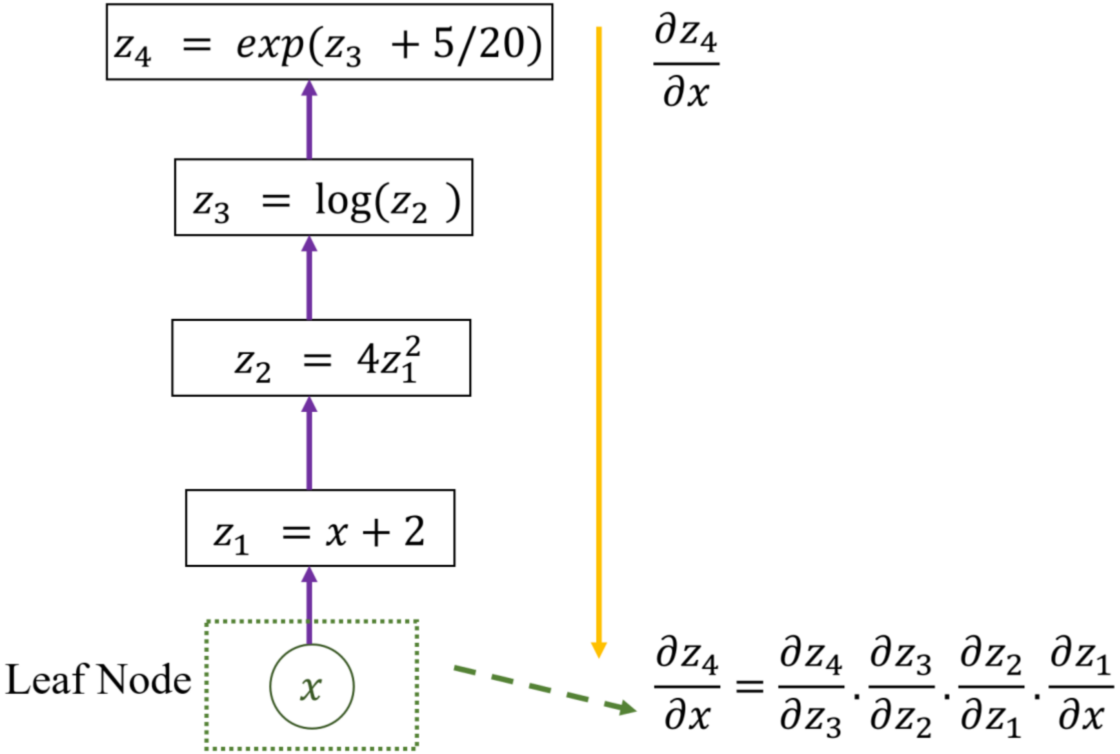
- Map each data instance $\hat{\mathbf{x}}^{(i)}$ to a unique $\mathbf{z}^{(i)}$.
- Map similar data instances close to each other in the latent space.

# Implementing Neural Networks: Automatic Differentiation



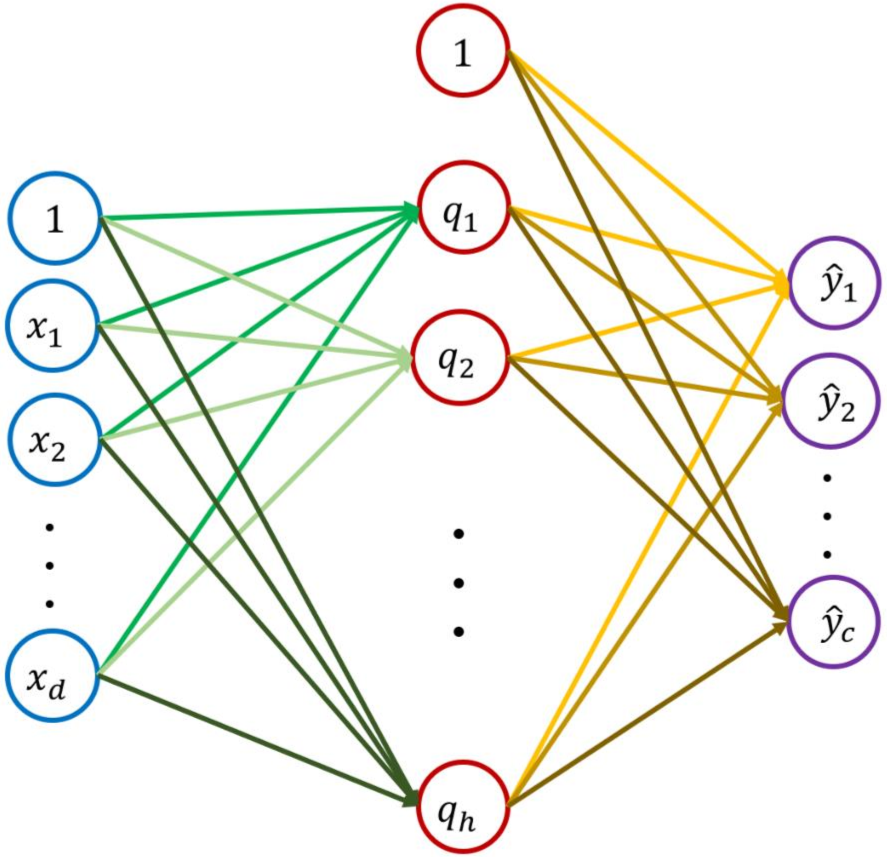

- Deep Learning libraries keep track of a series of operations that occur on a variable in the form of a **computational graph**.
- At the end of the series of operations, the gradient with respect to the initial variable can be automatically computed.

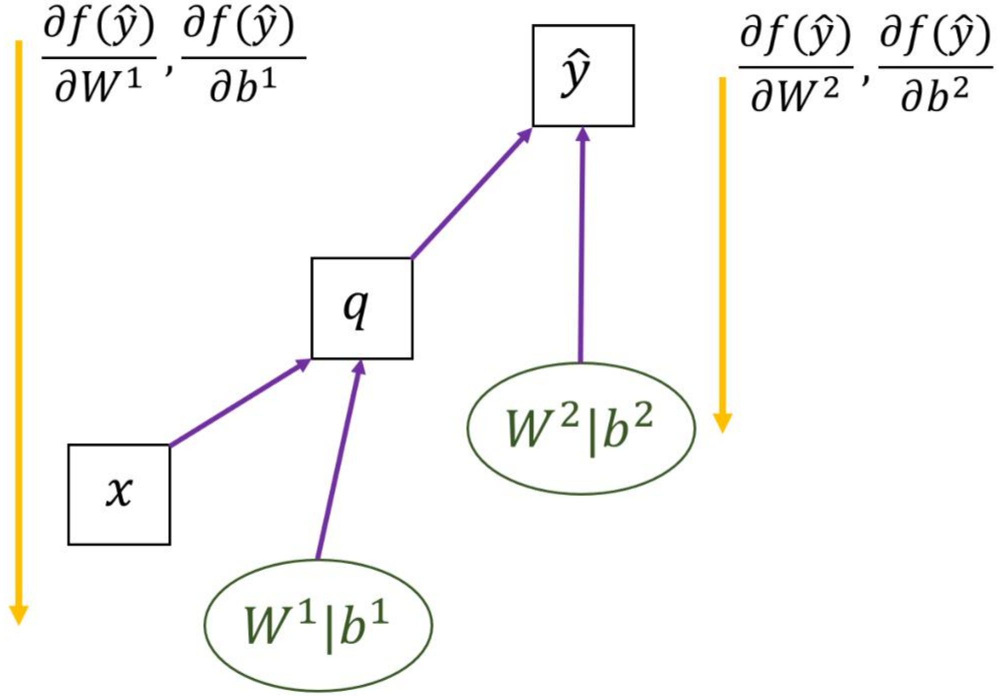
# Implementing Neural Networks: Automatic Differentiation




- Deep Learning libraries keep track of all operations that occur on a variable in the form of a **computational graph**.
- At the end of the series of operations, the gradient with respect to **any leaf variable** of the computational graph can be automatically computed.

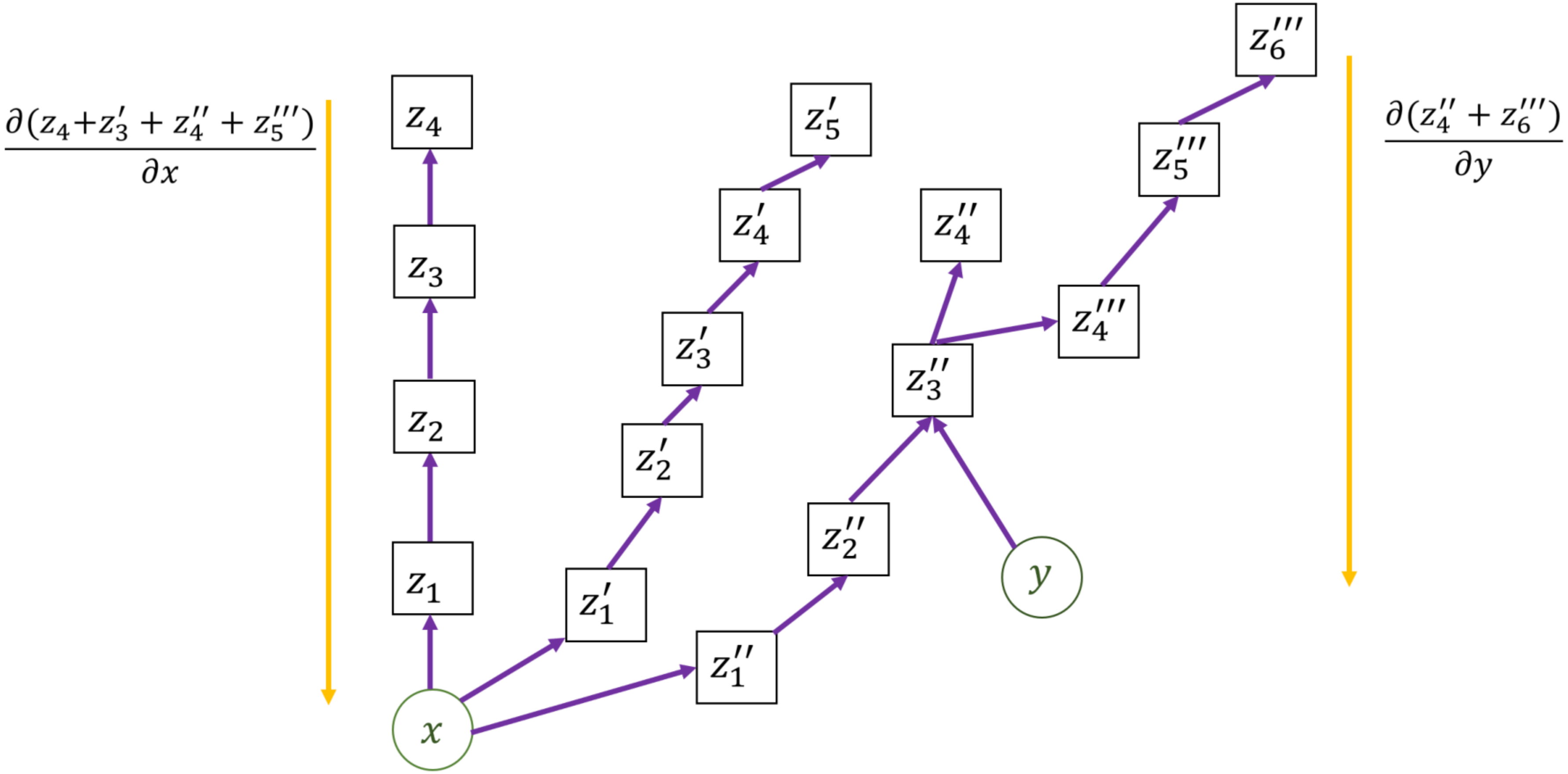
# Automatic Differentiation for MLPs



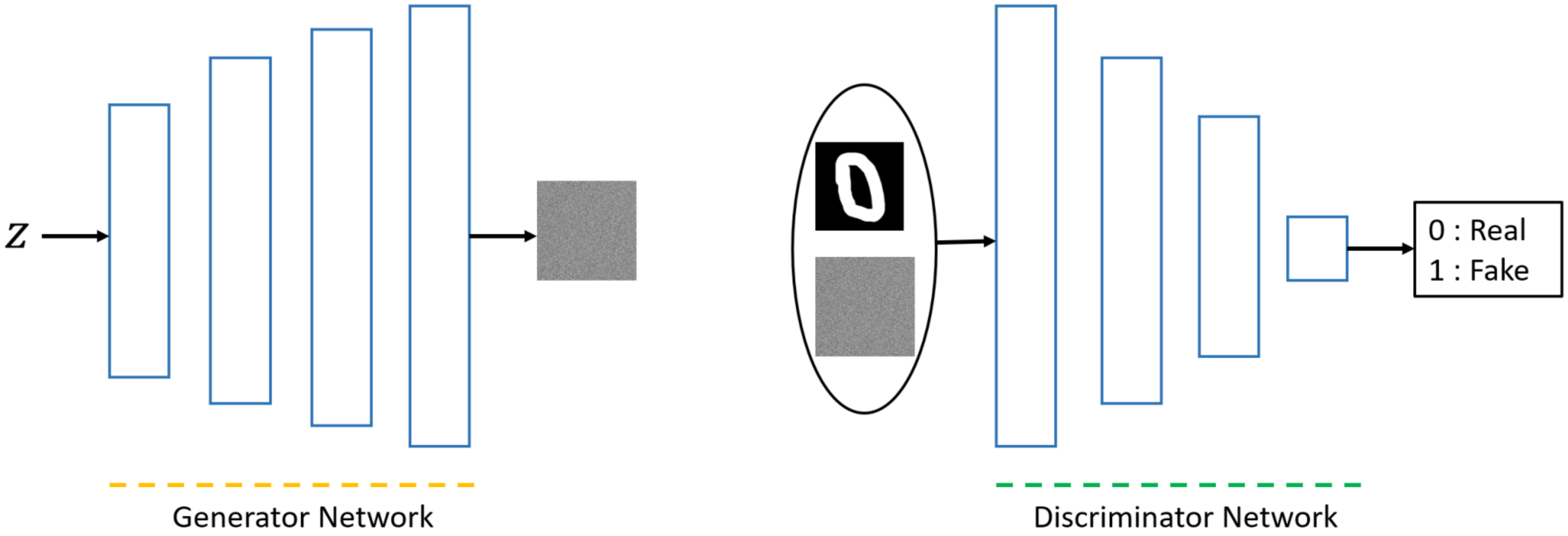

- The leaves of the computational graph are the network variables $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$.
- The gradients of the leaves with respect to any function of $\hat{\mathbf{y}}$ (such as any loss function) can be automatically computed.

# Automatic Differentiation



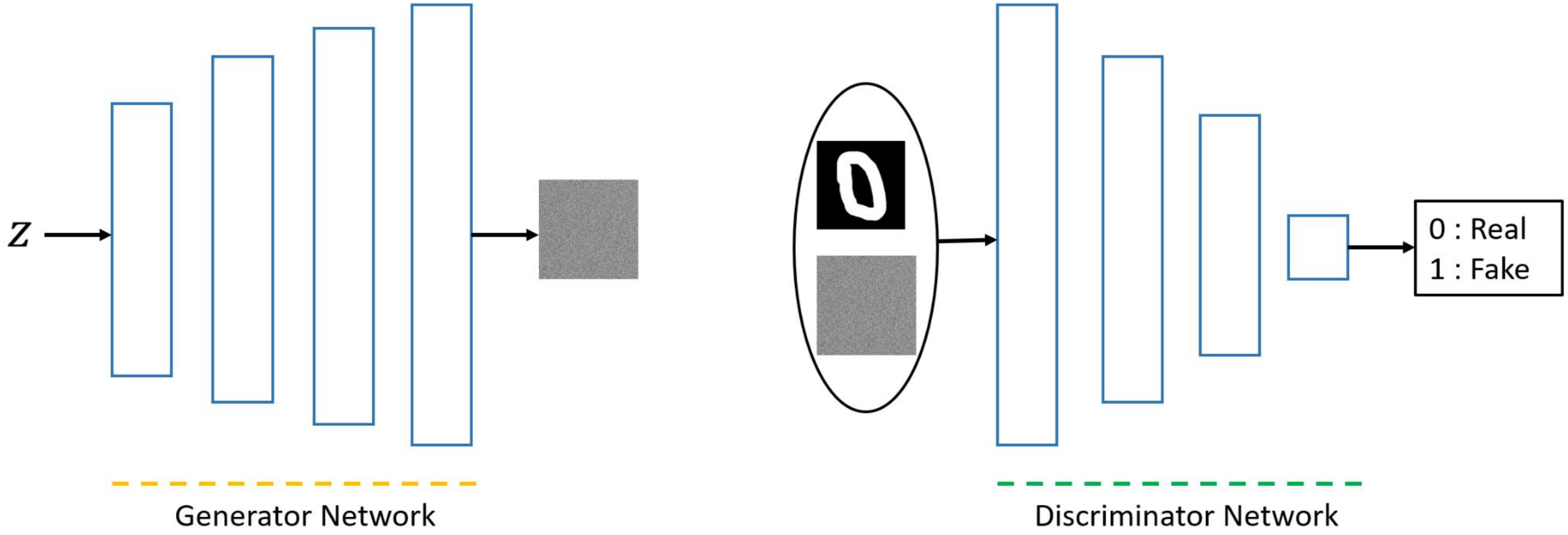

- In general, automatic differentiation can be done for any number of leaves, along any number of paths.
- The gradients along only some of the paths can also be computed.

# Estimating data distributions: Generative Adversarial Networks



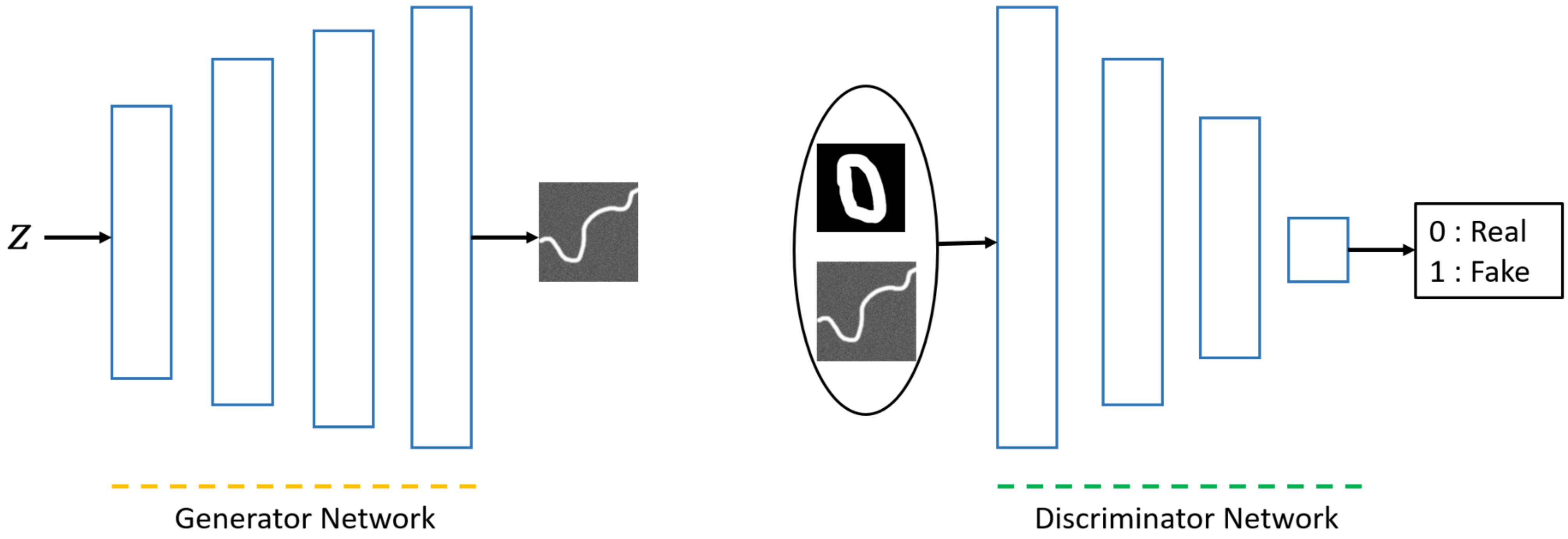

- We wish to estimate the distribution of the data $p_{data}$ using a Generator Network which follows a distribution $p_G$; the learning task is to obtain $p_G \approx p_{data}$.
- The Generator learns the data distribution by mapping $z \in \mathbb{R}^l$ to the data space.
- A Discriminator network is trained to discriminated between the (fake) data generated by the Generator, and data from a real data set.

## Generative Adversarial Networks



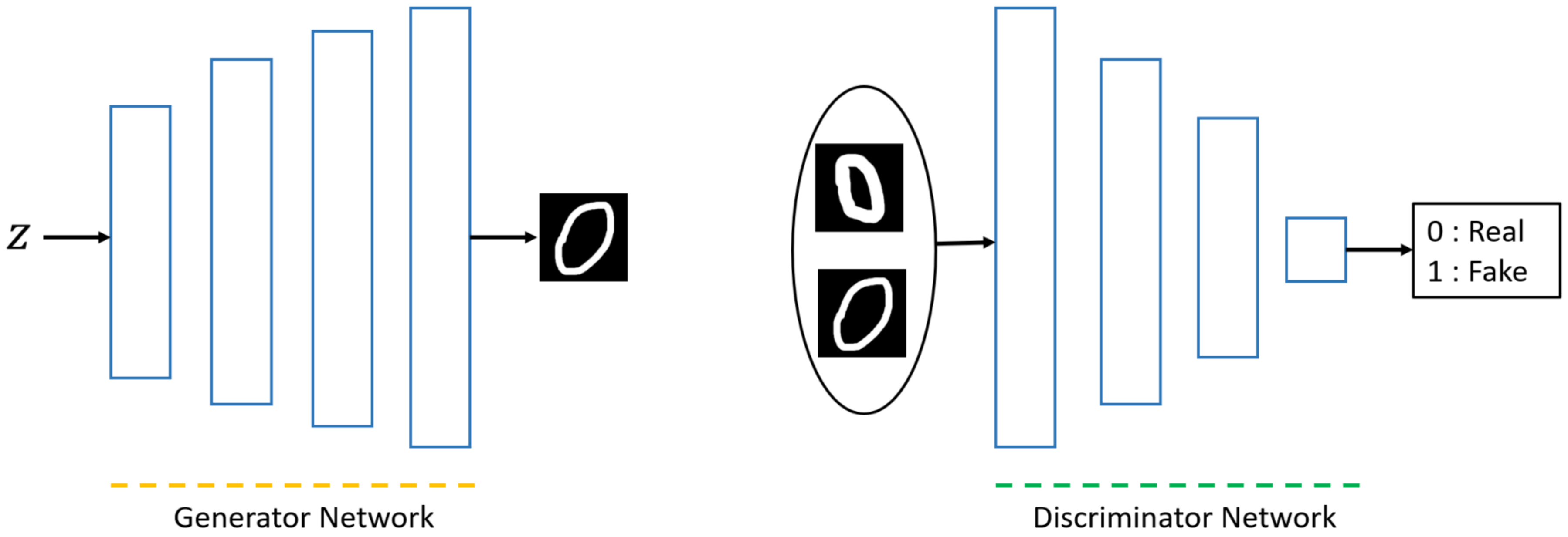

- The Discriminator is trained for $k_1$ iterations, so that it learns to properly discriminate between real and generated images.

# Generative Adversarial Networks




- The Discriminator is trained for $k_1$ iterations, so that it learns to properly discriminate between real and generated images.
- The Generator is trained for $k_2$ iterations, so that it learns to fool the Discriminator.

# Generative Adversarial Networks



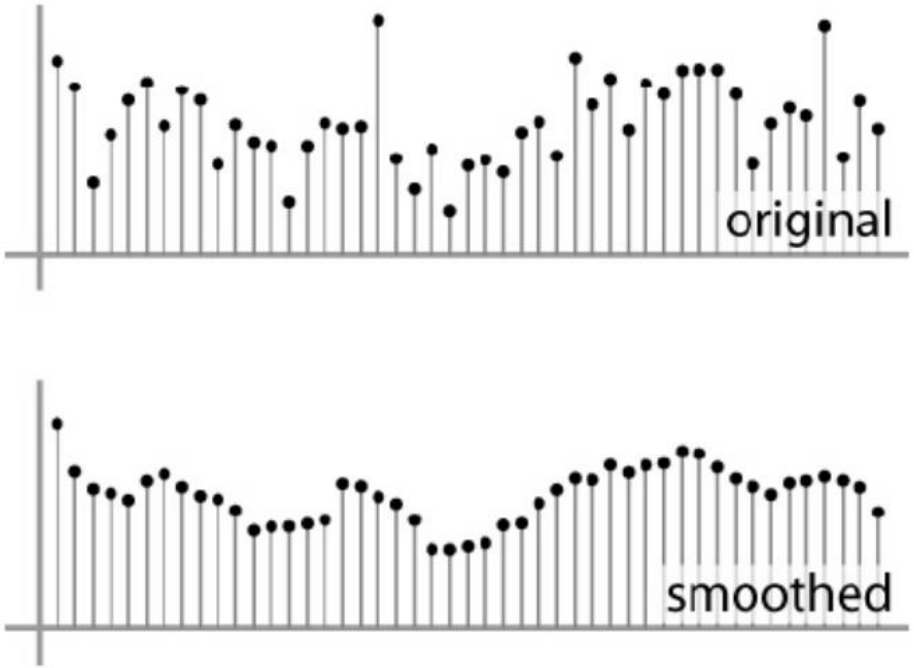

- The Discriminator is trained for $k_1$ iterations, so that it learns to properly discriminate between real and generated images.
- The Generator is trained for $k_2$ iterations, so that it learns to fool the Discriminator.
- The Discriminator $D$ and the Generator $G$ are alternately trained to optimize a joint objective function:

$$\min_G \max_D \; \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

# A short view on Deep Learning

# A Signal Processing Perspective of kernels

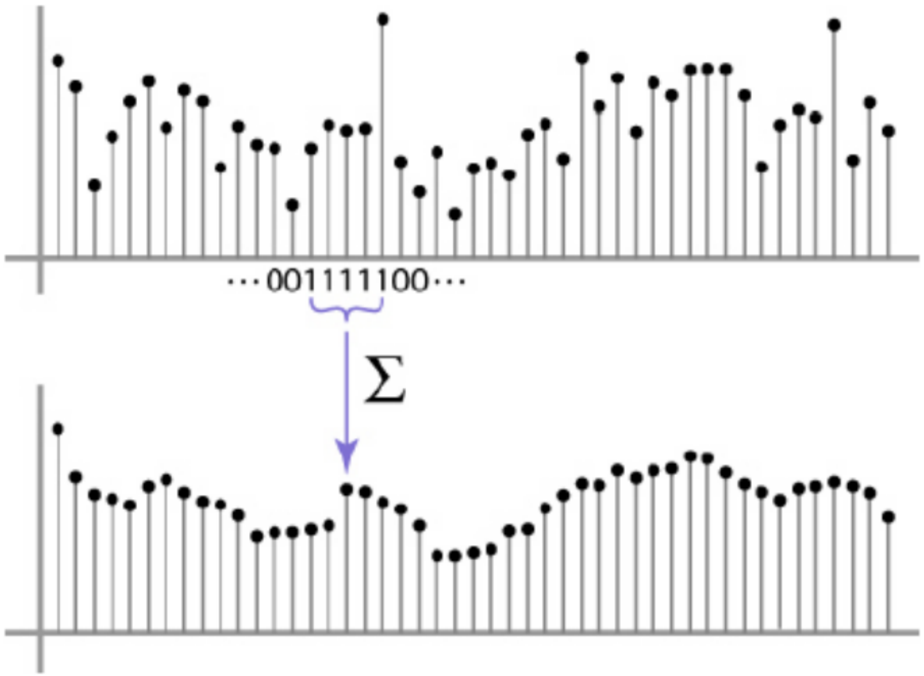
- How can we smooth a quantized signal?




Source: S. Marschner

# A Signal Processing Perspective of kernels

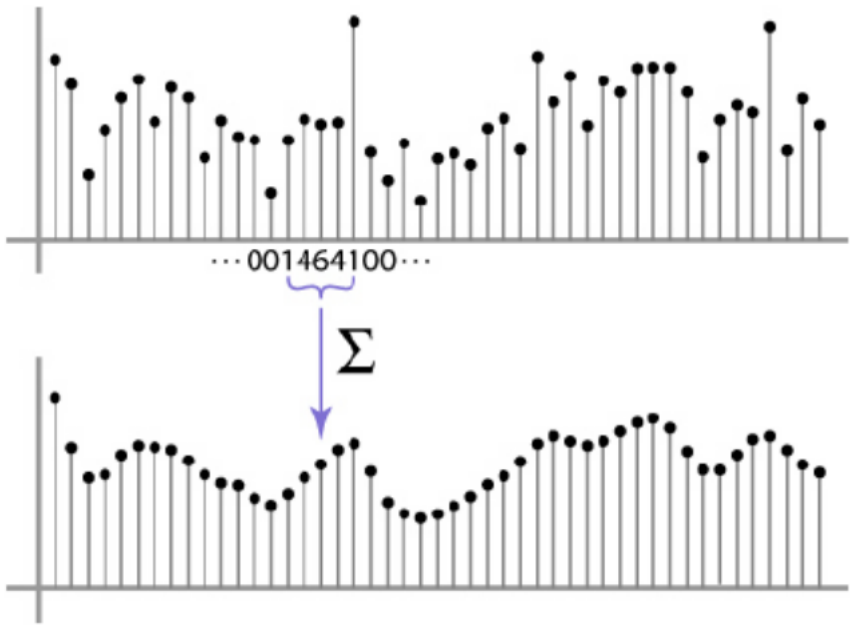
- How can we smooth a quantized signal?
  - ➢ Moving Average




Source: S. Marschner

# A Signal Processing Perspective of kernels

- How can we smooth a quantized signal?
  - *Weighted* Moving Average

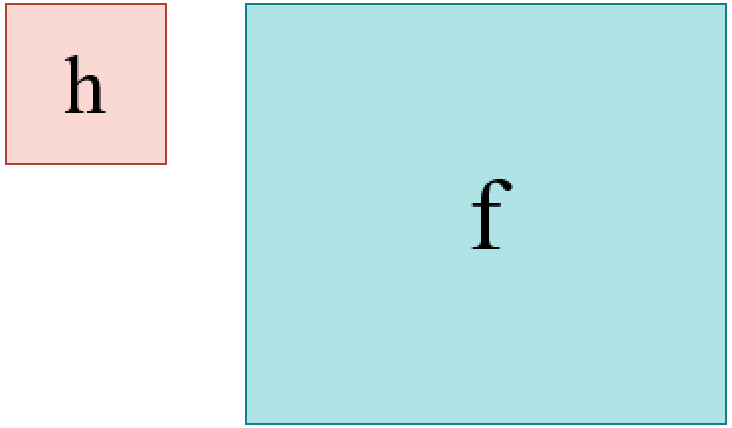



Source: S. Marschner

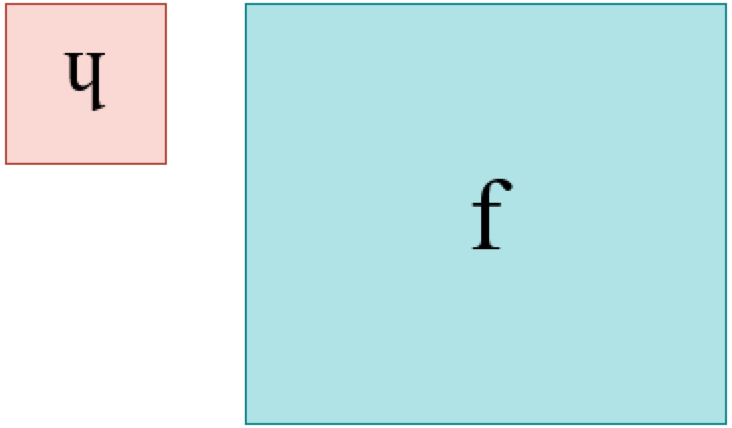
# Correlation & Convolution Operators

Correlation: $$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

Convolution: $$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$




Correlation




Convolution

# Weighted averages of 2D signals

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Source: S. Seitz

# Weighted averages of 2D signals

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Source: S. Seitz

# Weighted averages of 2D signals

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Source: S. Seitz

# Weighted averages of 2D signals

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Source: S. Seitz

# Weighted averages of 2D signals

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

Source: S. Seitz

# Weighted averages of 2D signals

$$F[x, y]$$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

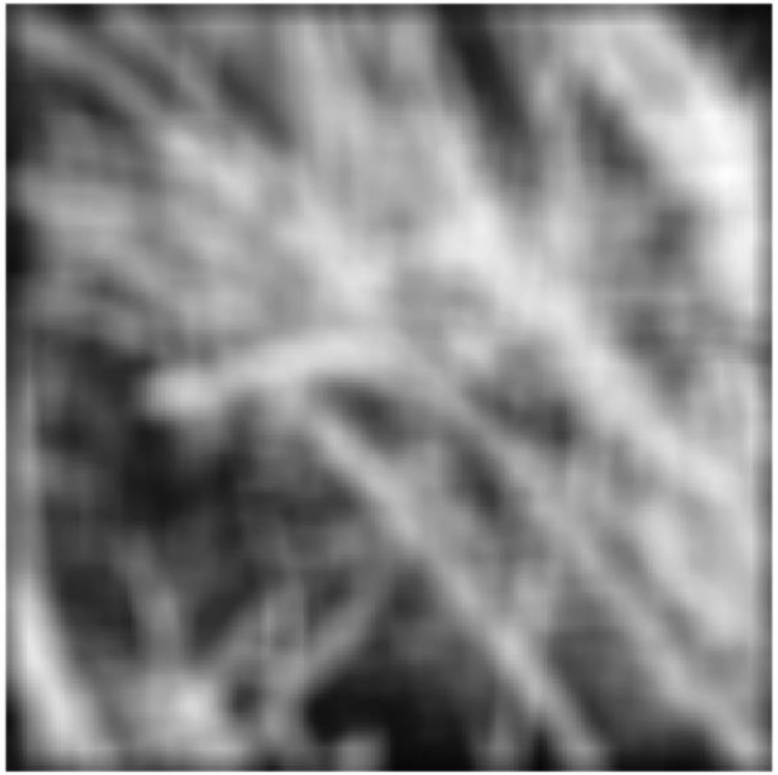
$$G[x, y]$$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 | |
| | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 | |
| | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 | |
| | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 | |
| | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | |

Source: S. Seitz

# Effect of Convolutions: A Smoothed Signal

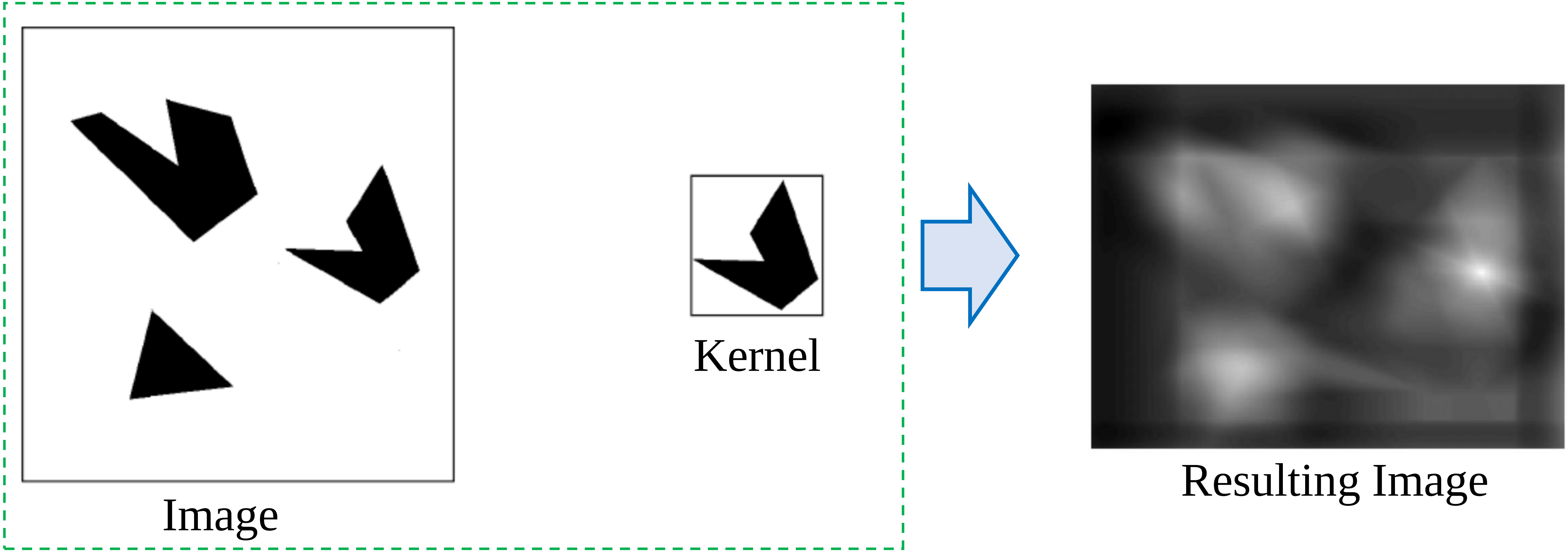
original



filtered

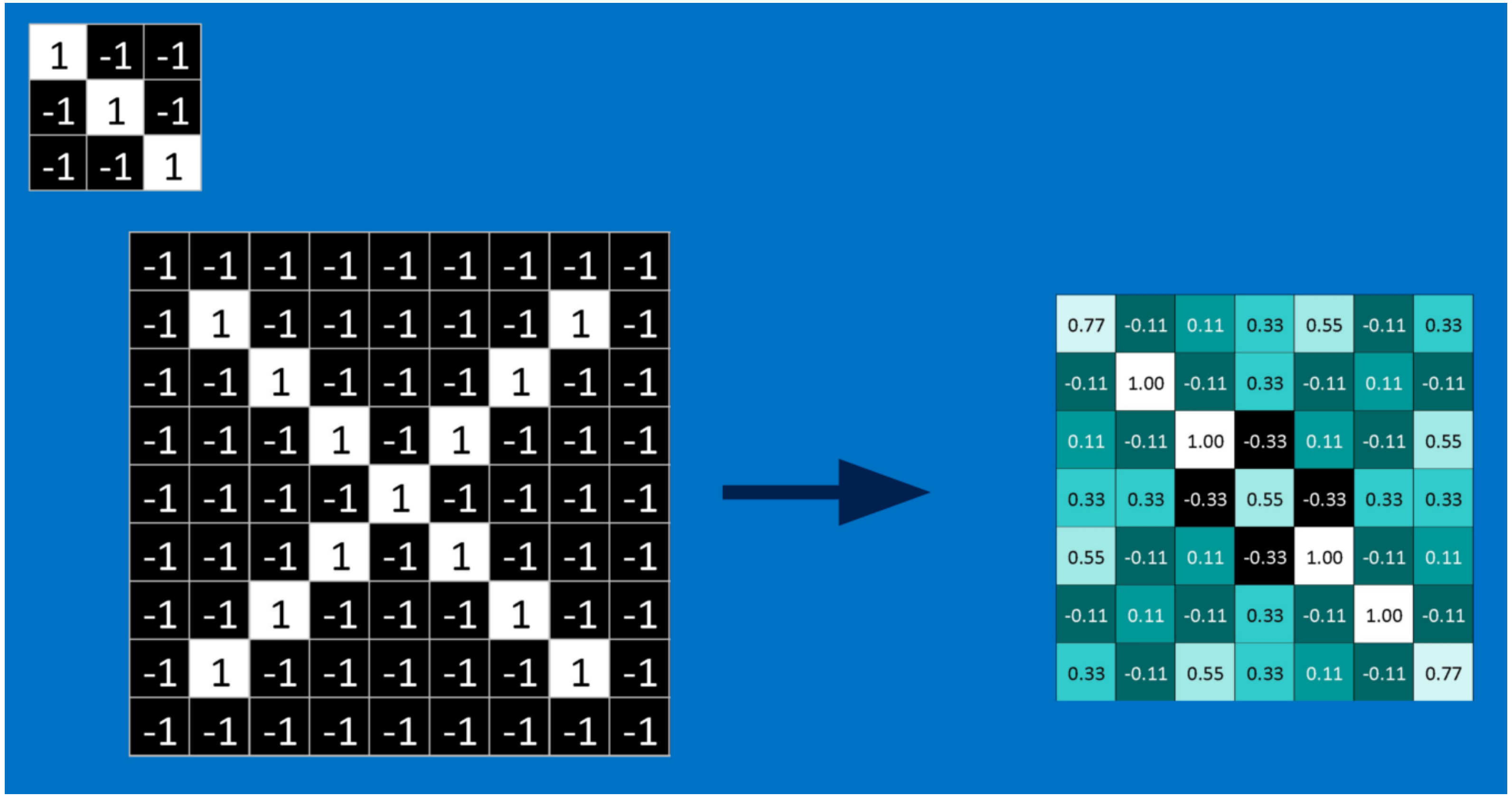
Source: K. Graumann

# Normalized Correlation for Template Matching

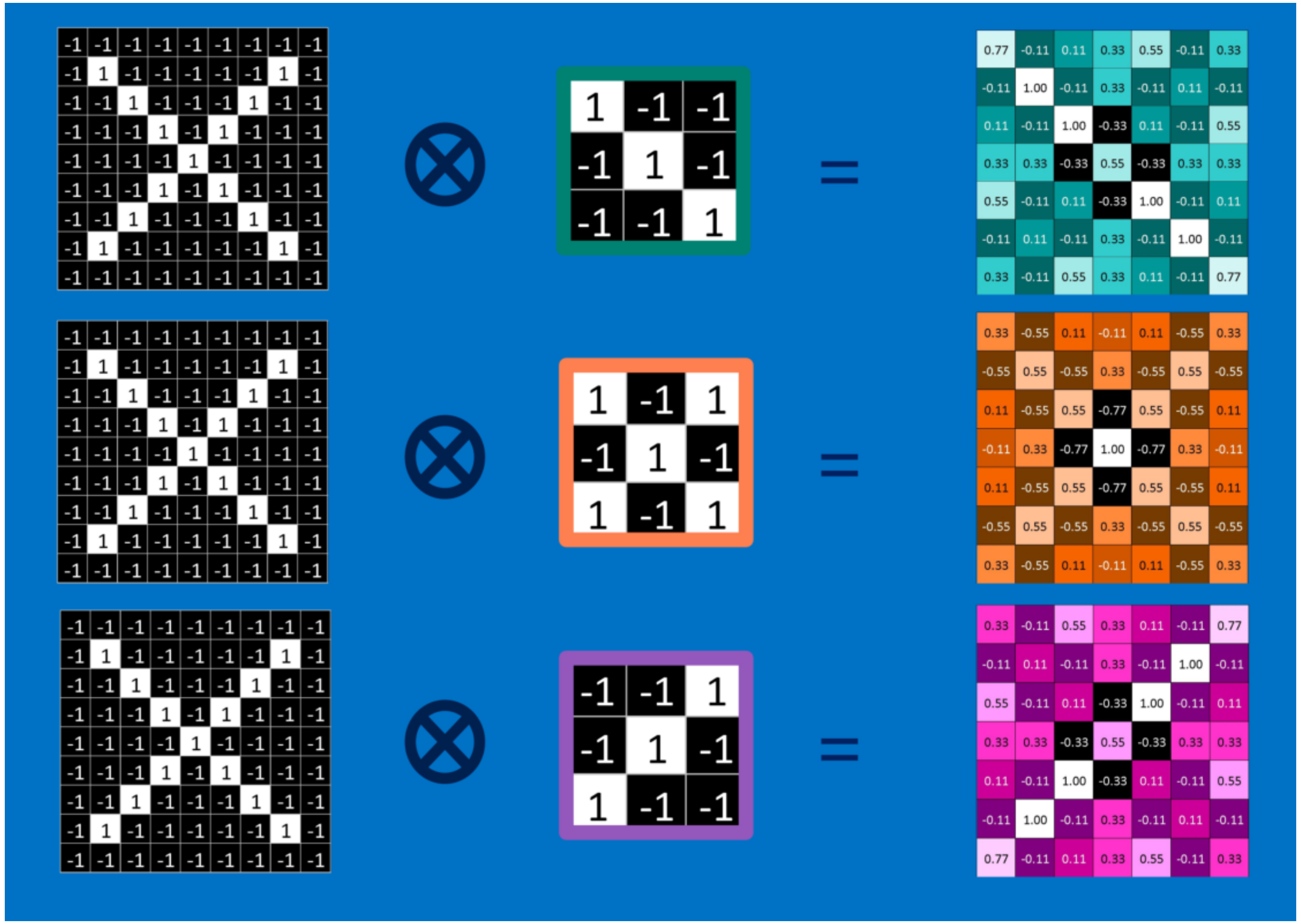



Source: K. Graumann

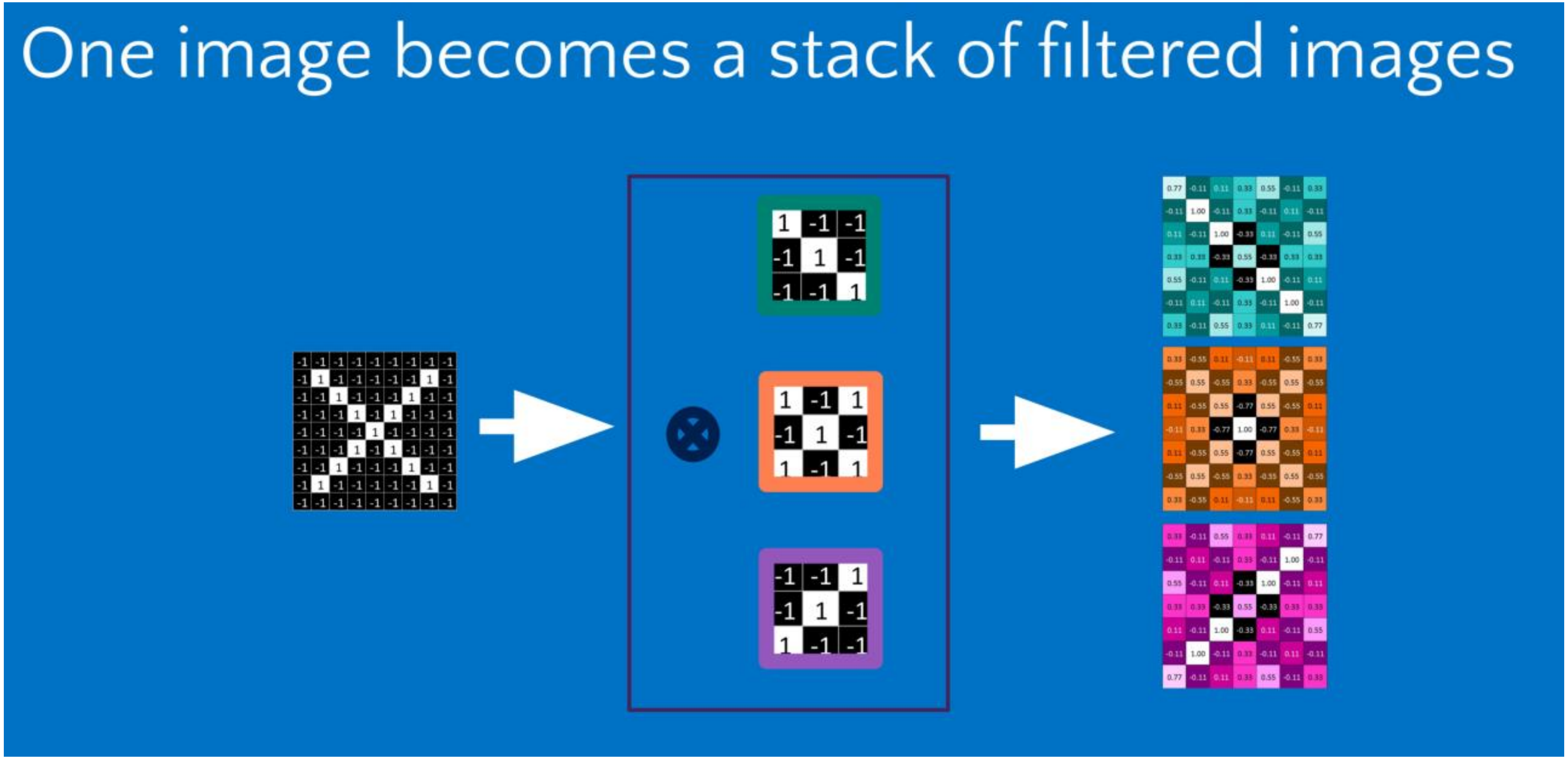
# Convolution operation



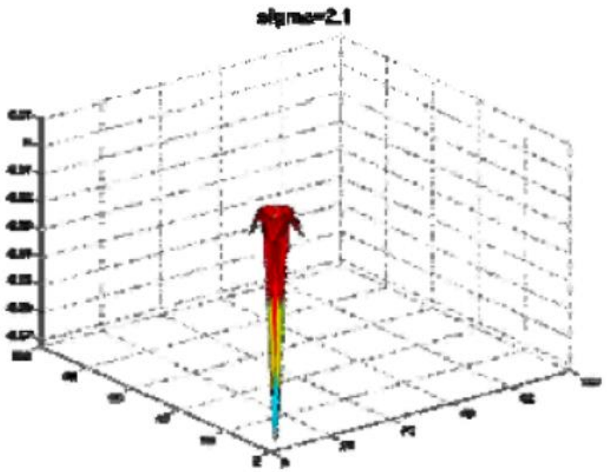
# Convolution operation

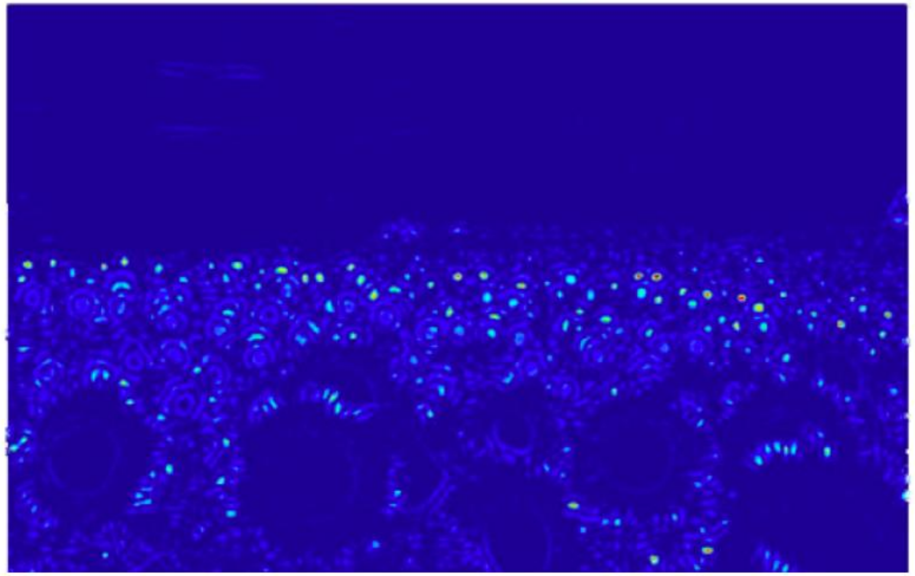


# Convolution operation



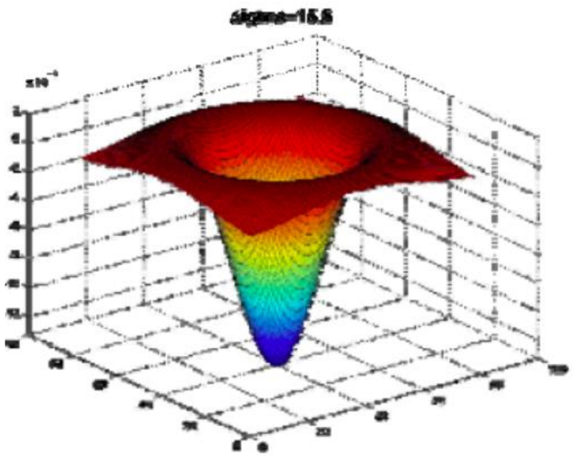
# Different features may exist at different scales

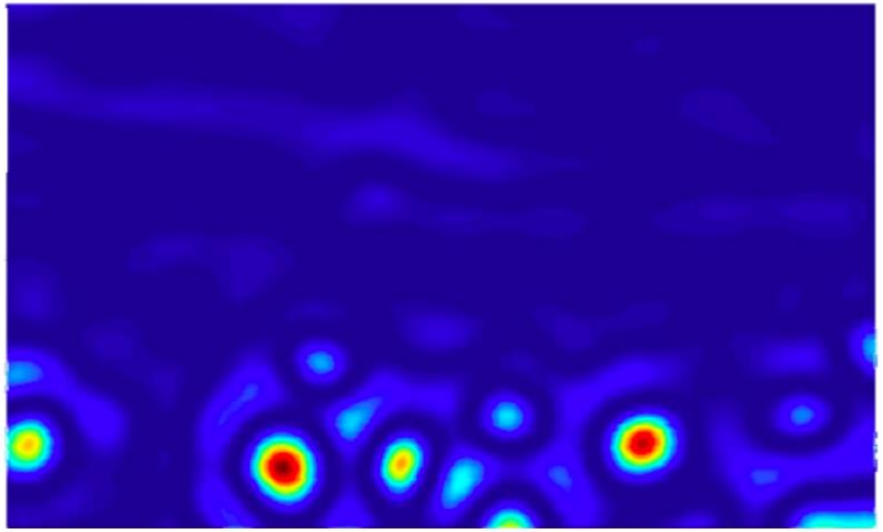


Source: K. Graumann

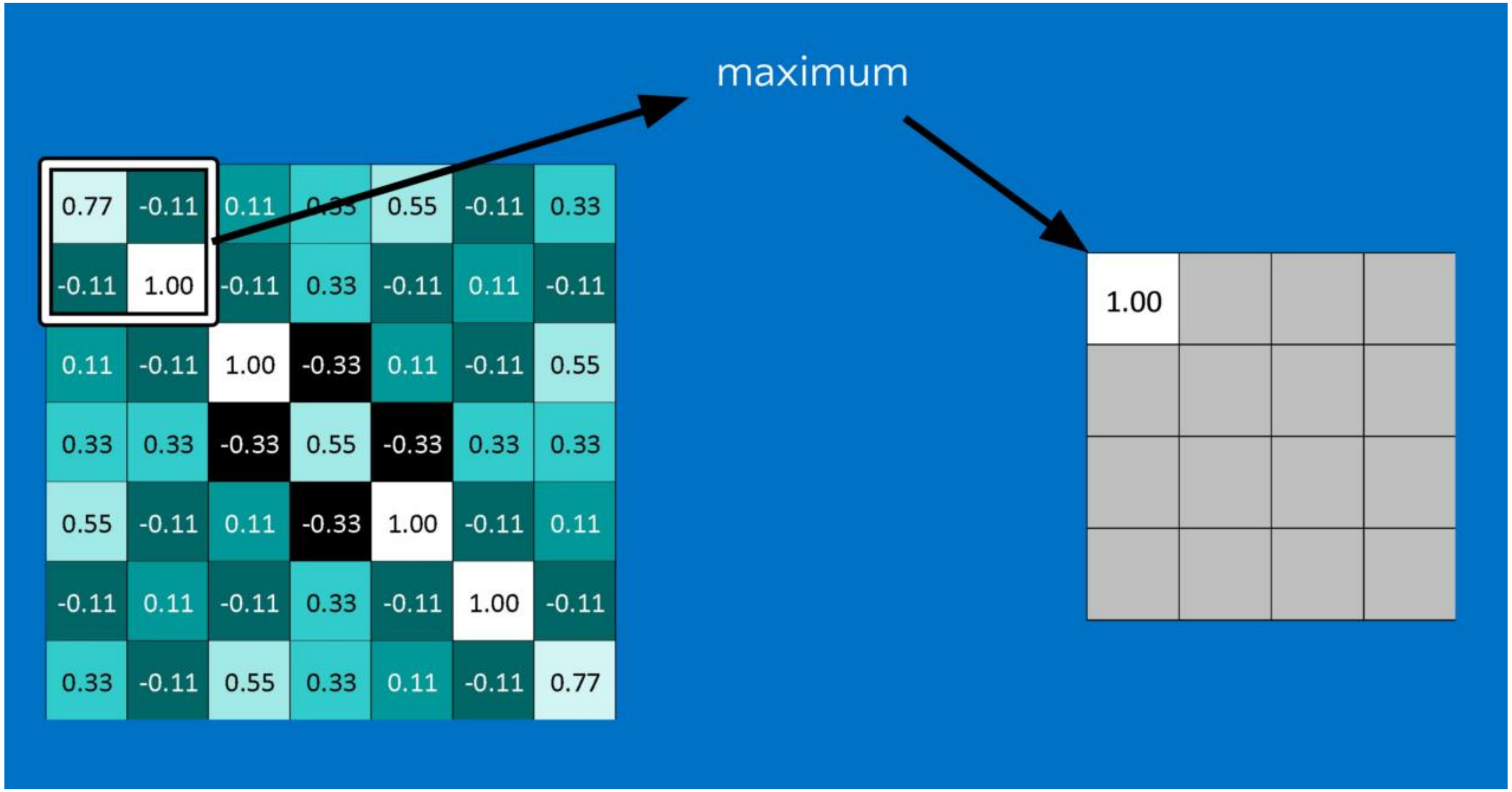
# Different features may exist at different scales



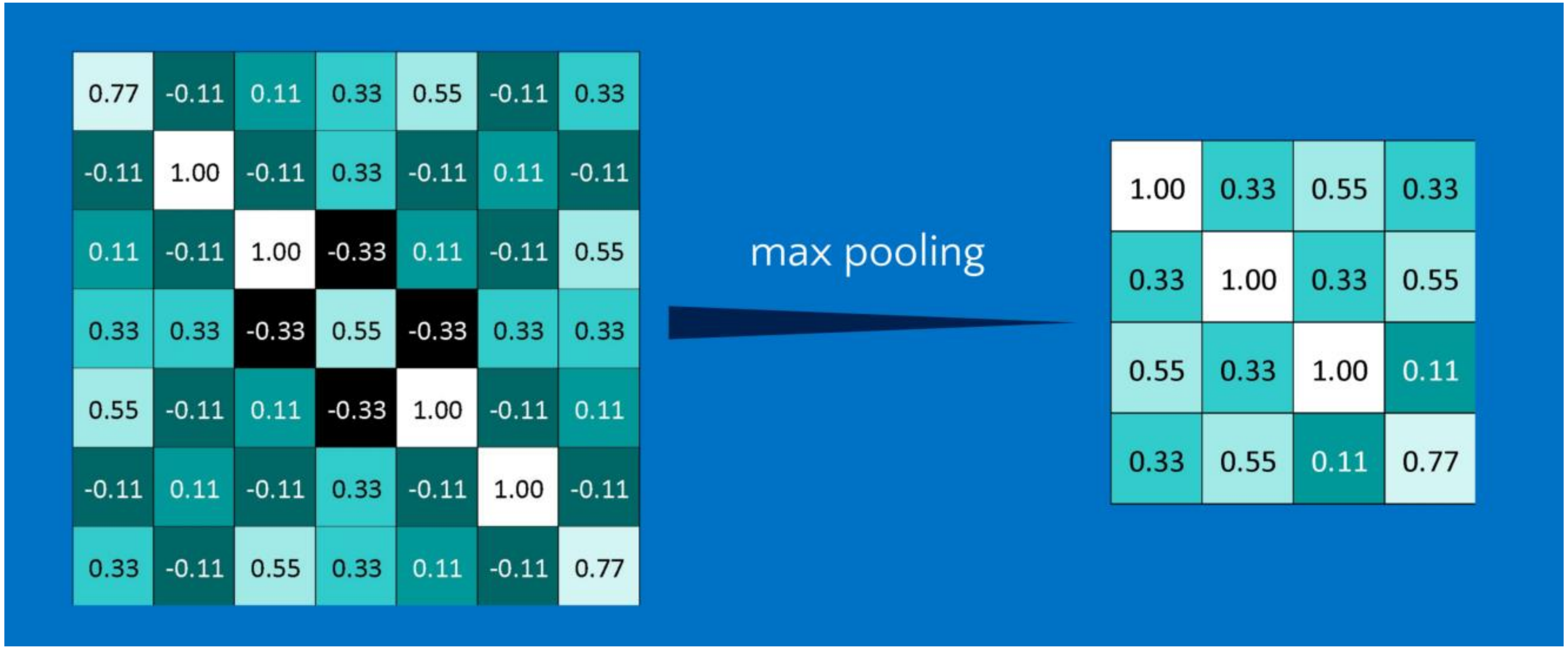


Source: K. Graumann

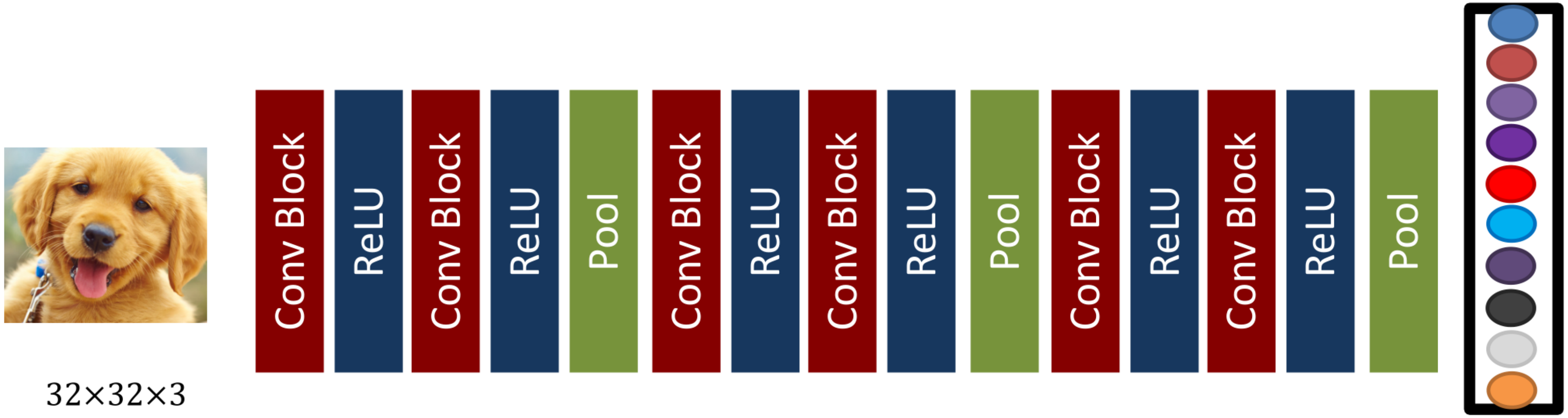
# Max Pool operation



# Max Pool operation



# Convolution Neural Networks




32×32×3
Input

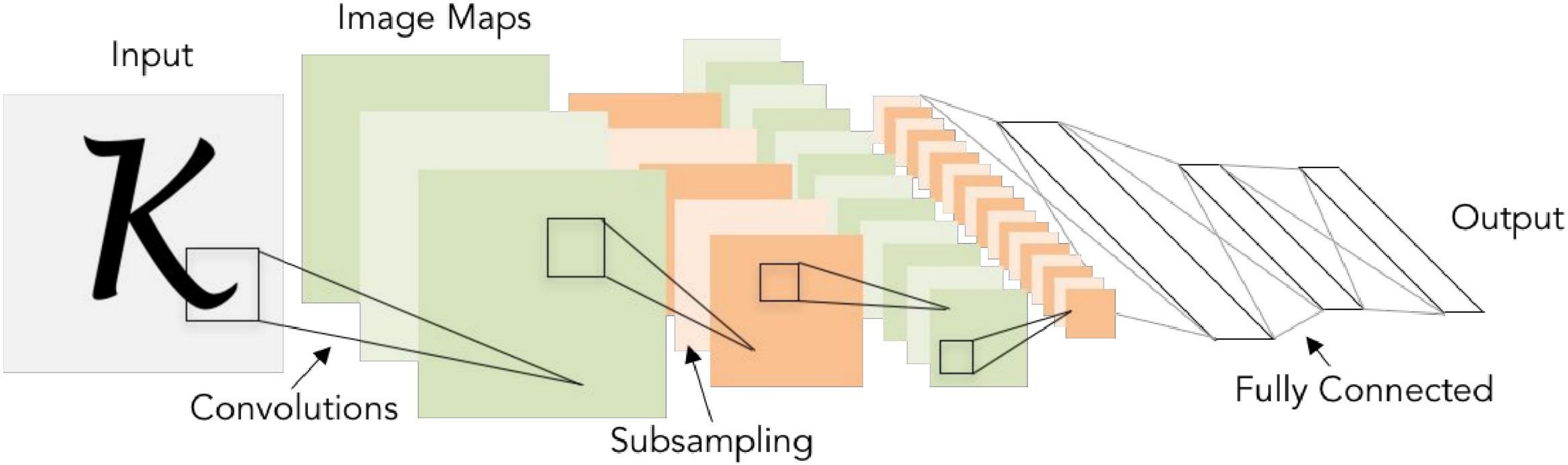
$10x1$
Output

Source: J. C. Niebles, R. Krishna

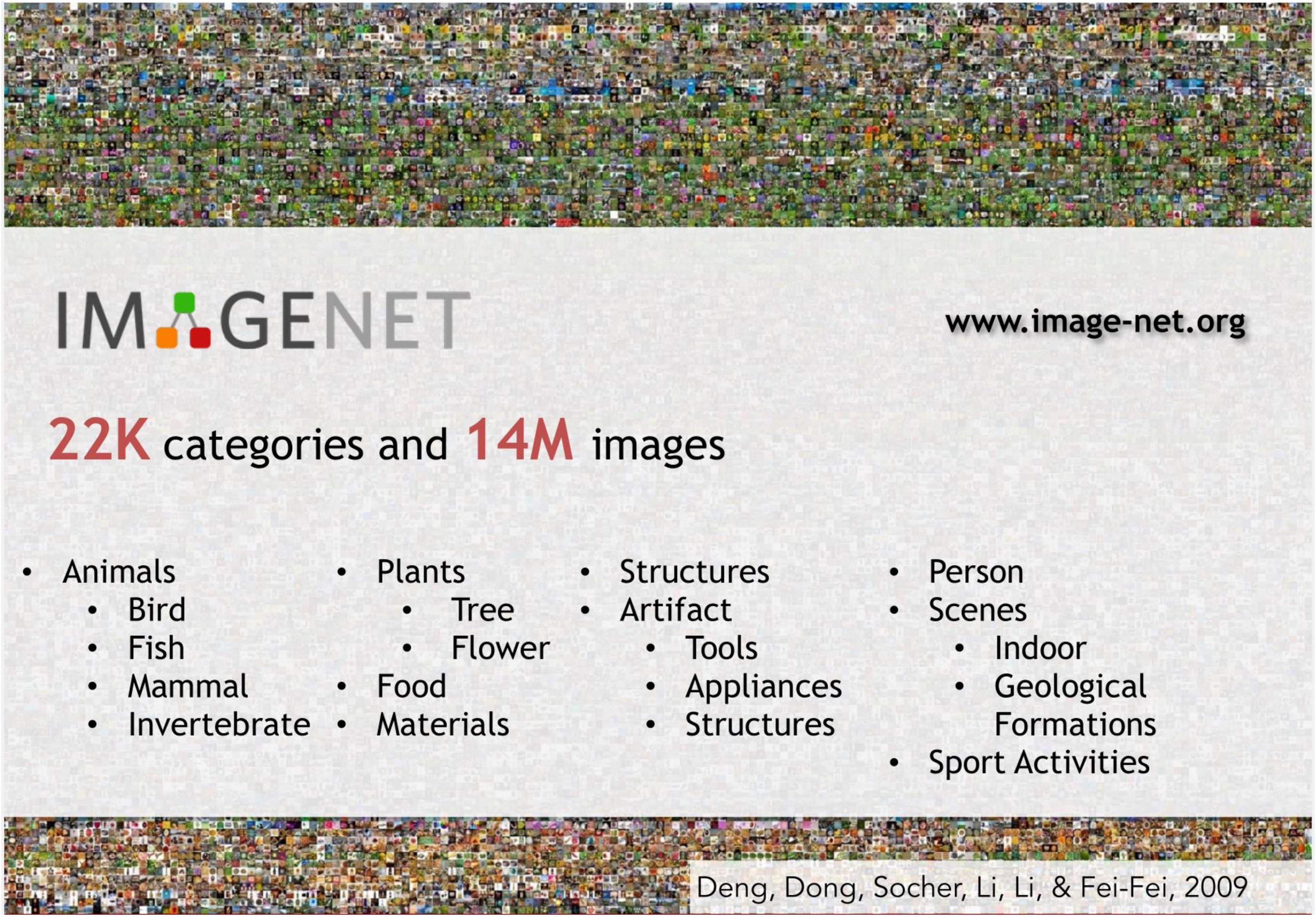
# Review: LeNet-5

[LeCun et al., 1998]

- 70% test accuracy on the MNIST dataset
- Used average pooling instead of max pooling

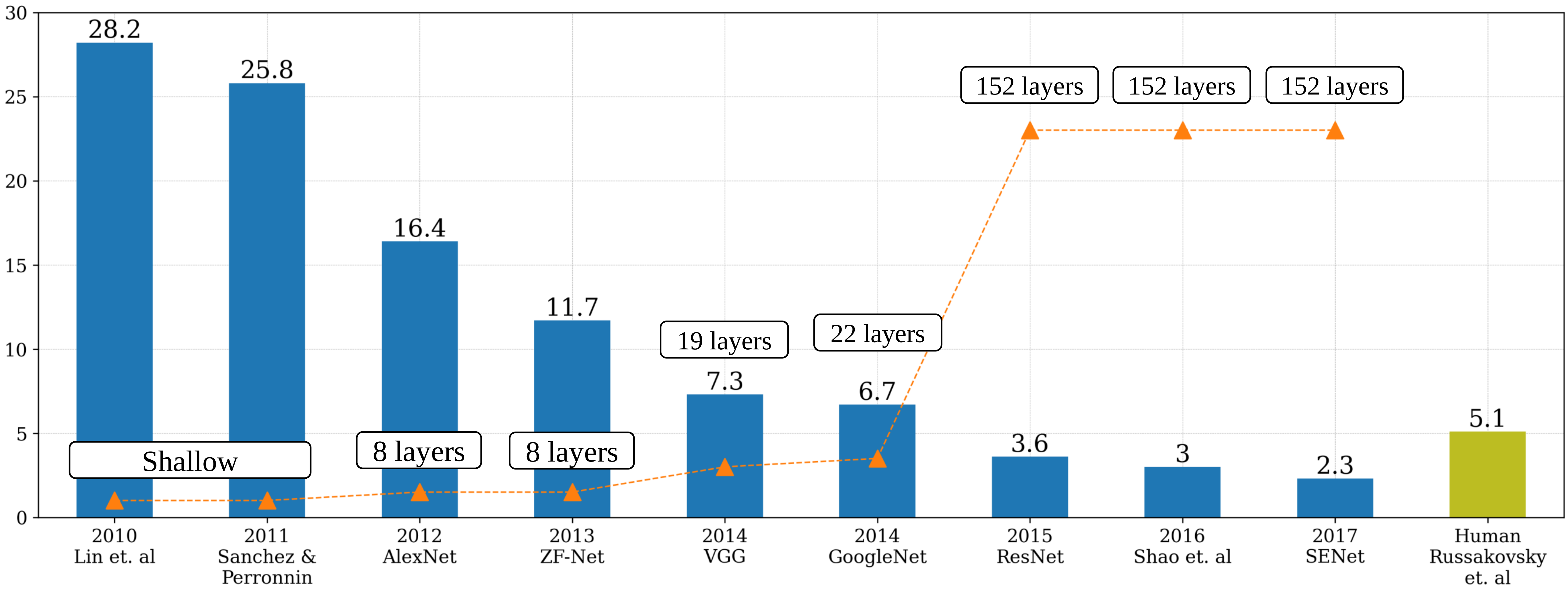



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

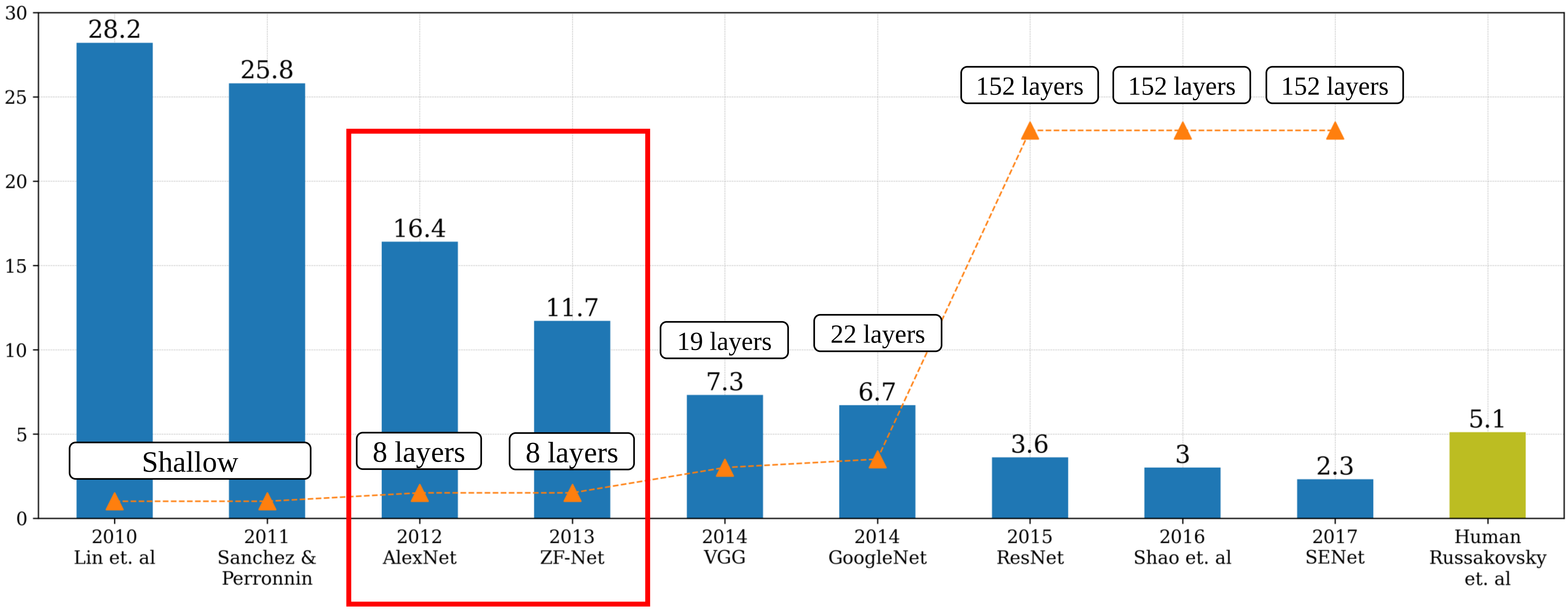



https://image-net.org/index.php
https://www.kaggle.com/c/imagenet-object-localization-challenge
https://pytorch.org/vision/stable/generated/torchvision.datasets.ImageNet.html

# ImageNet Challenge Error Rates over time

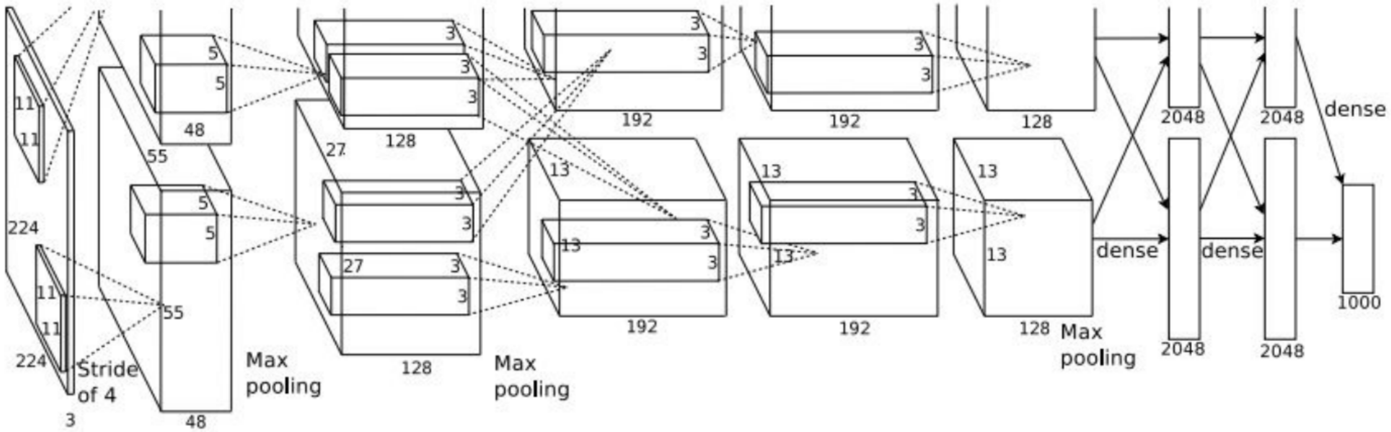


# ImageNet Challenge Error Rates over time



# Case Study: AlexNet

*[Krizhevsky et al. 2012]*

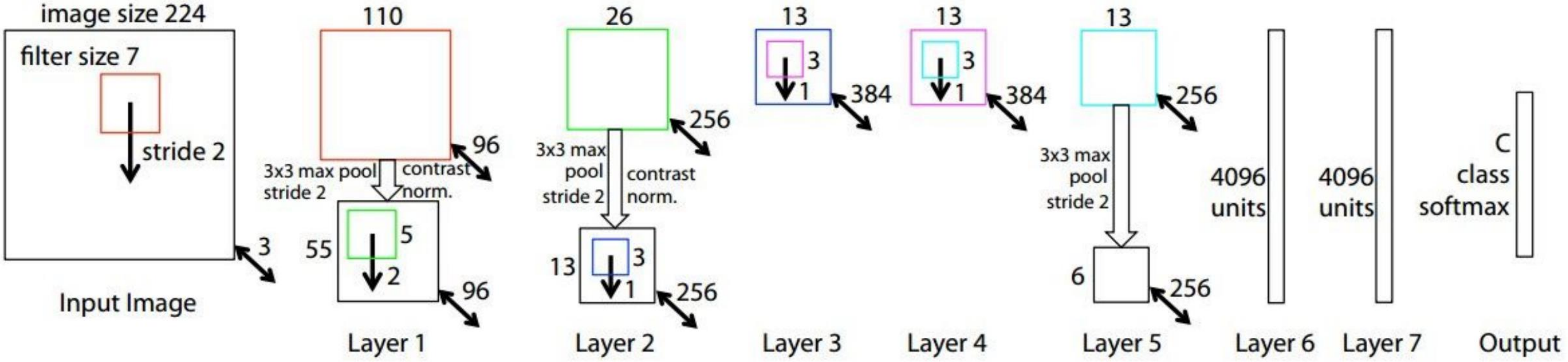



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256]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

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

# ZFNet

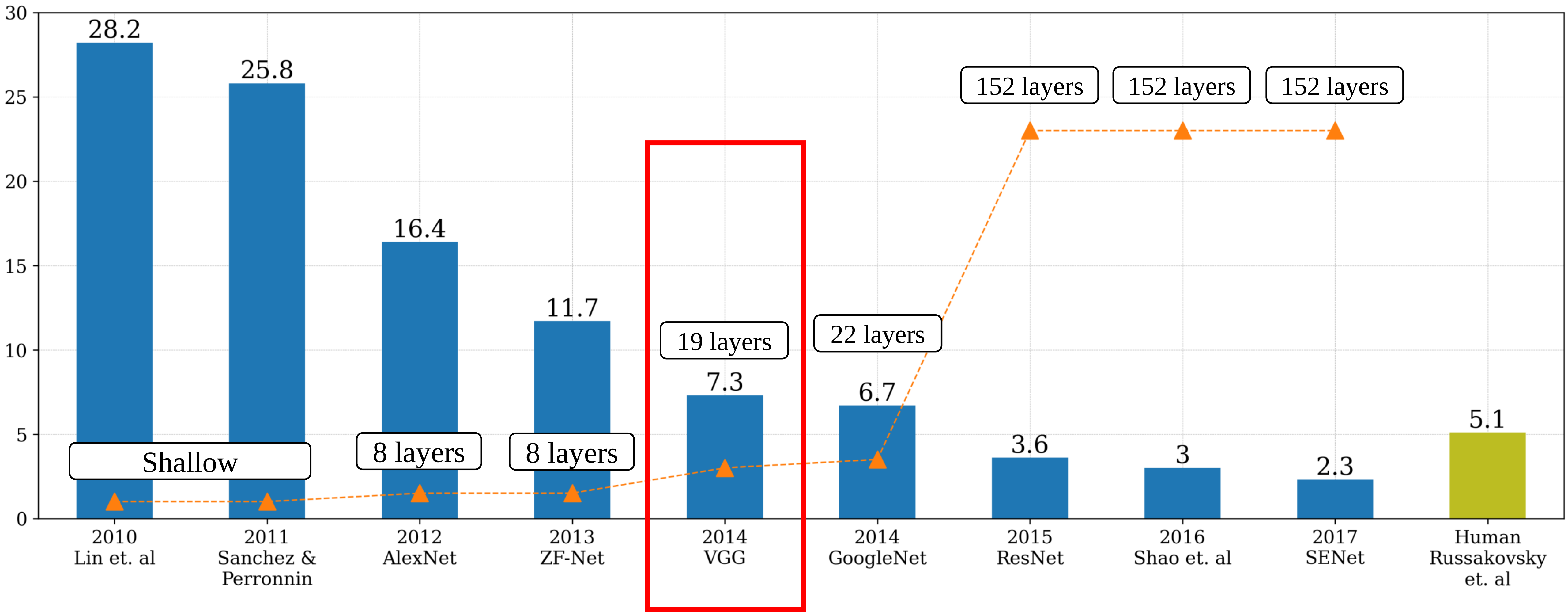
*[Zeiler and Fergus, 2013]*



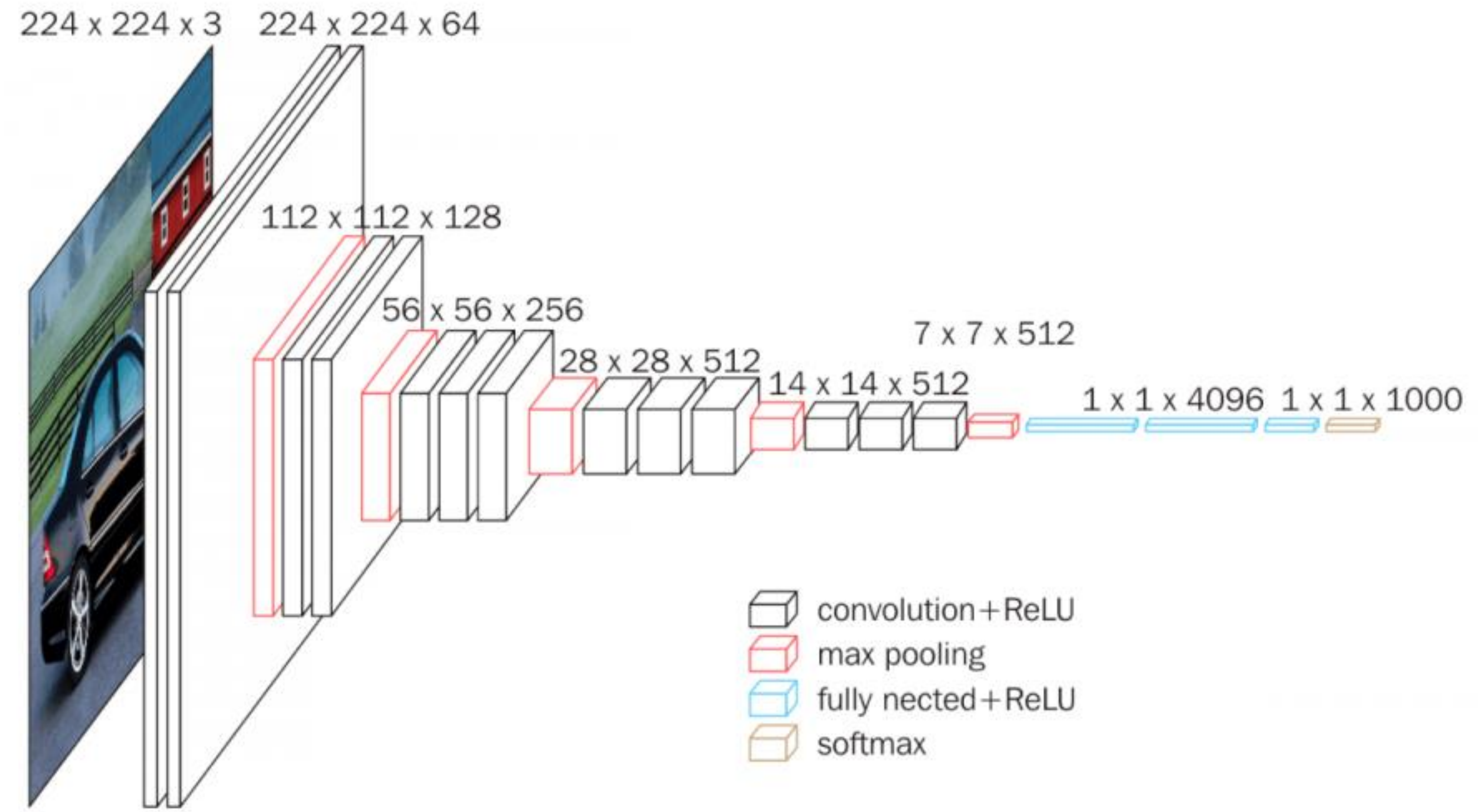

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

# ImageNet Challenge Error Rates over time

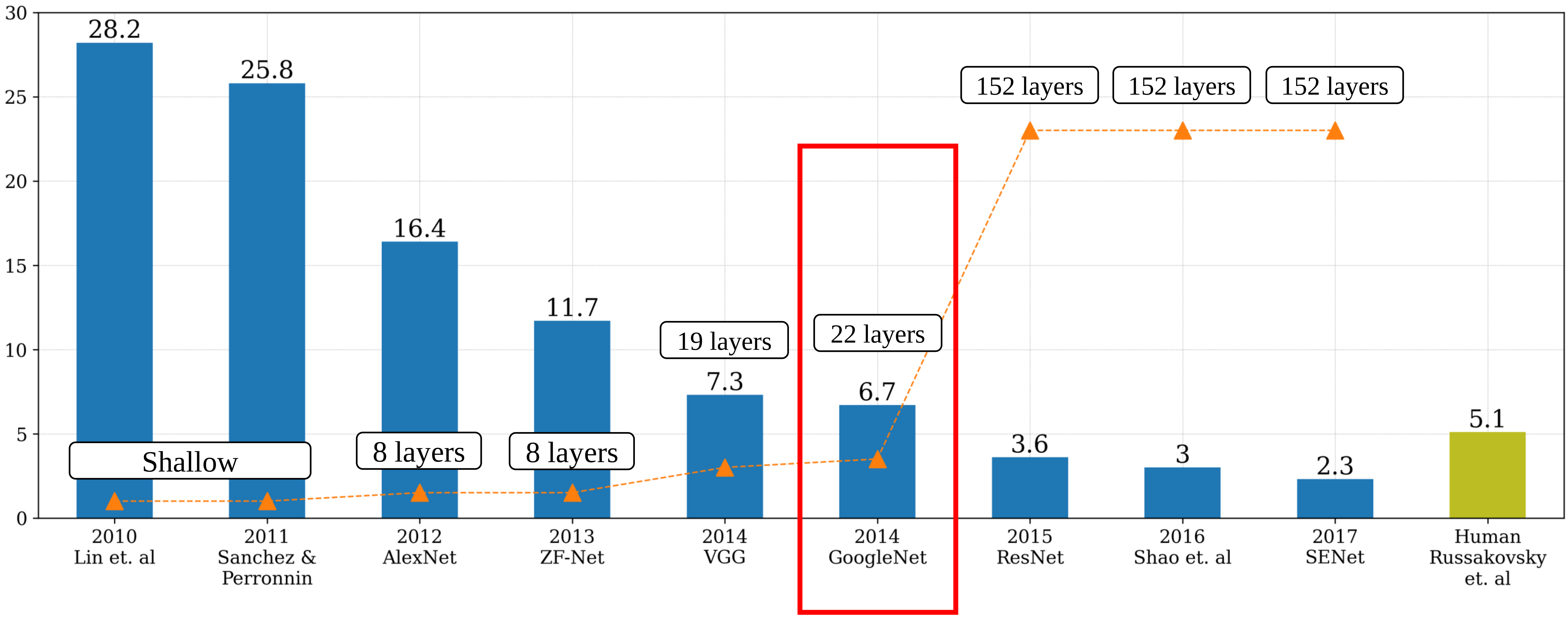


# VGG-Net 2014

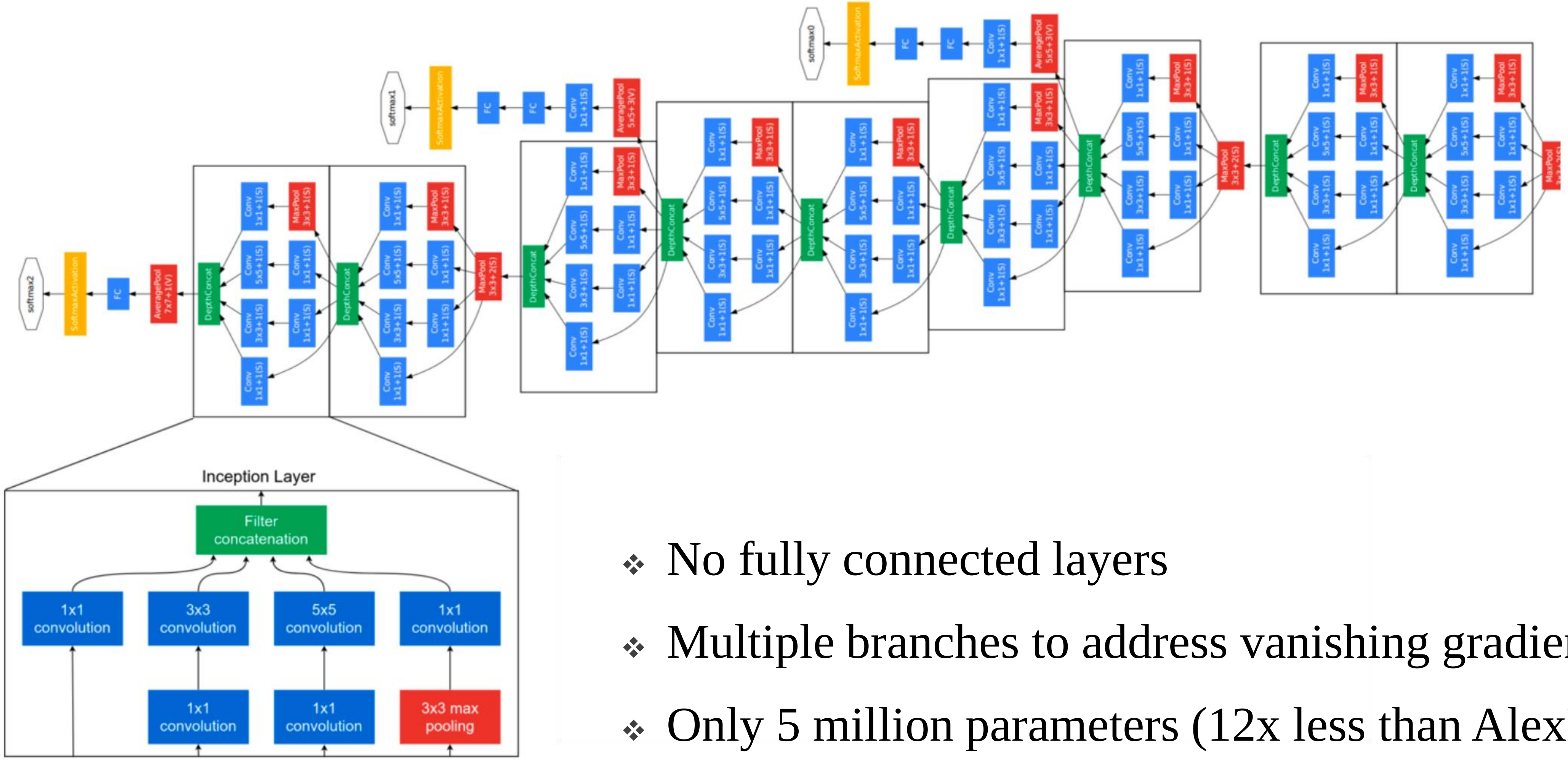



❖ Used 3×3 convolution kernels

# ImageNet Challenge Error Rates over time

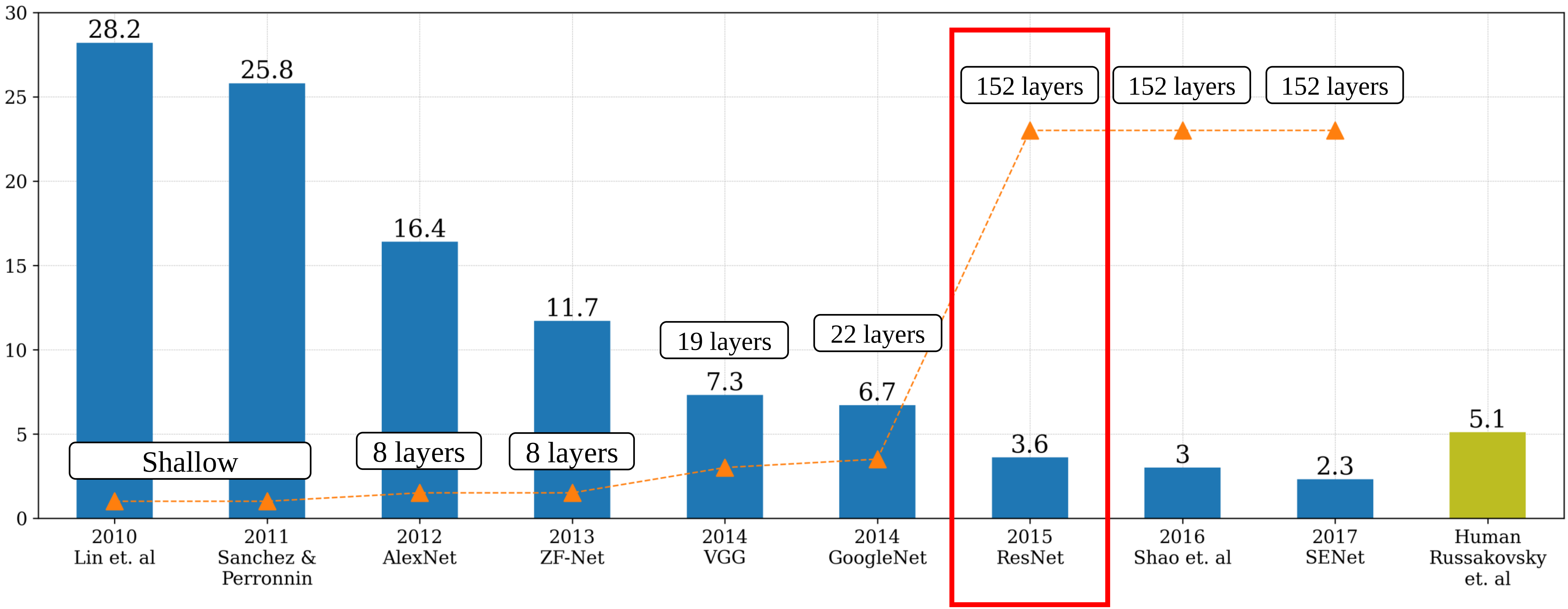


# GoogLeNet / InceptionNet v1 (and v2, v3) 2014

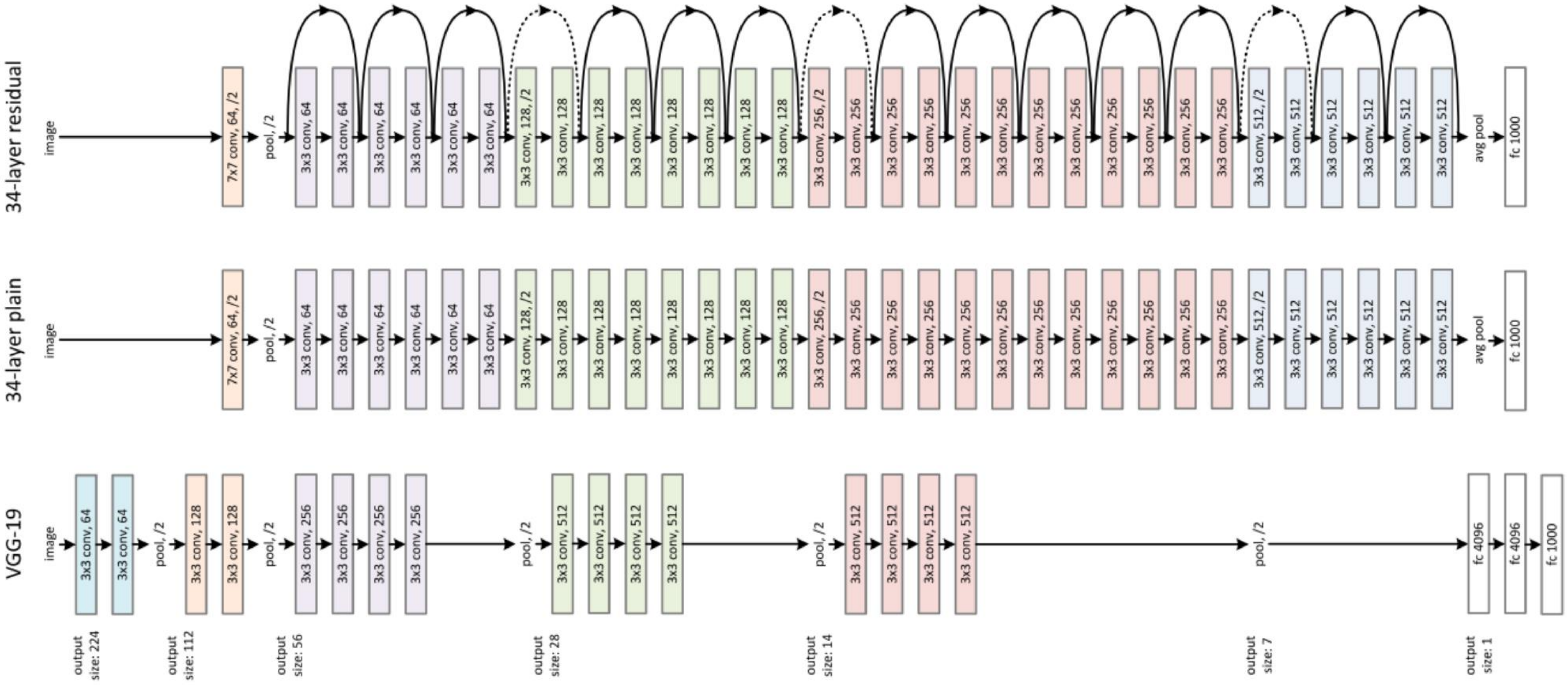



- No fully connected layers
- Multiple branches to address vanishing gradients
- Only 5 million parameters (12x less than AlexNet, 27x less than VGG-16)

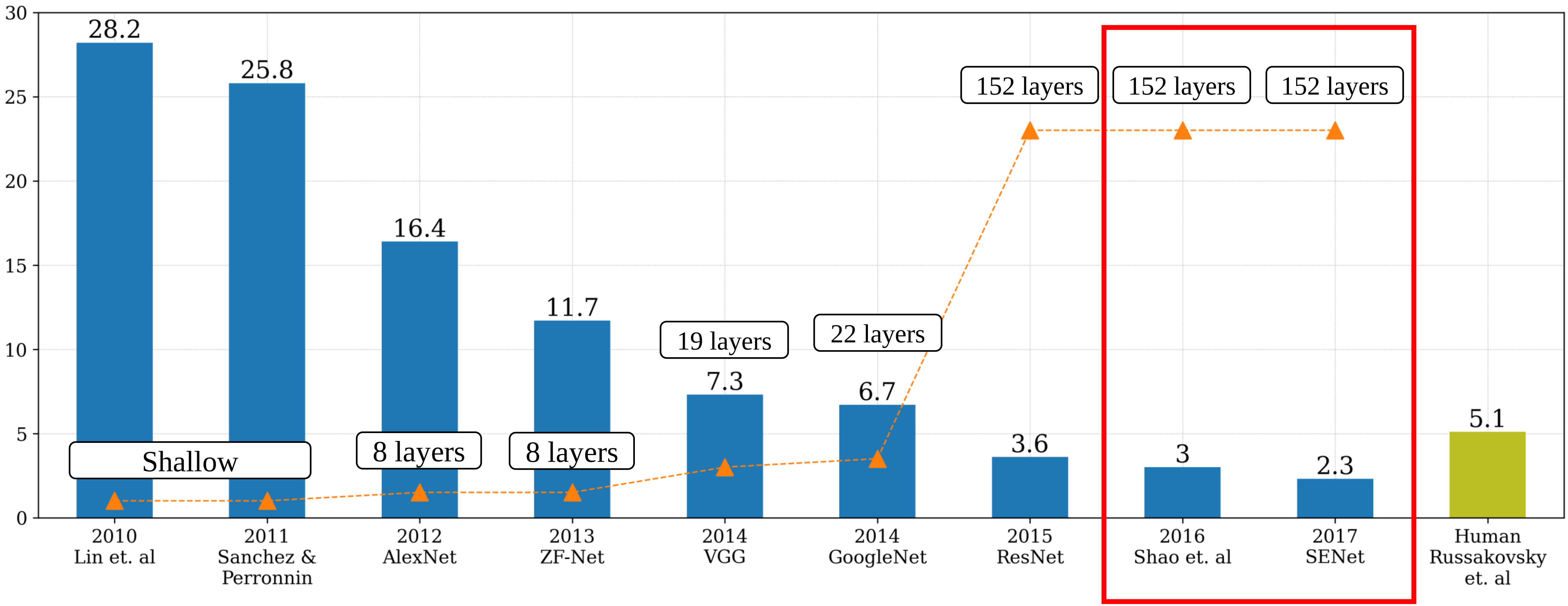
# ImageNet Challenge Error Rates over time



# He et al. 2015 – ResNet



# ImageNet Challenge Error Rates over time



Improving ResNets...

# “Good Practices for Deep Feature Fusion”

*[Shao et al. 2016]*

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

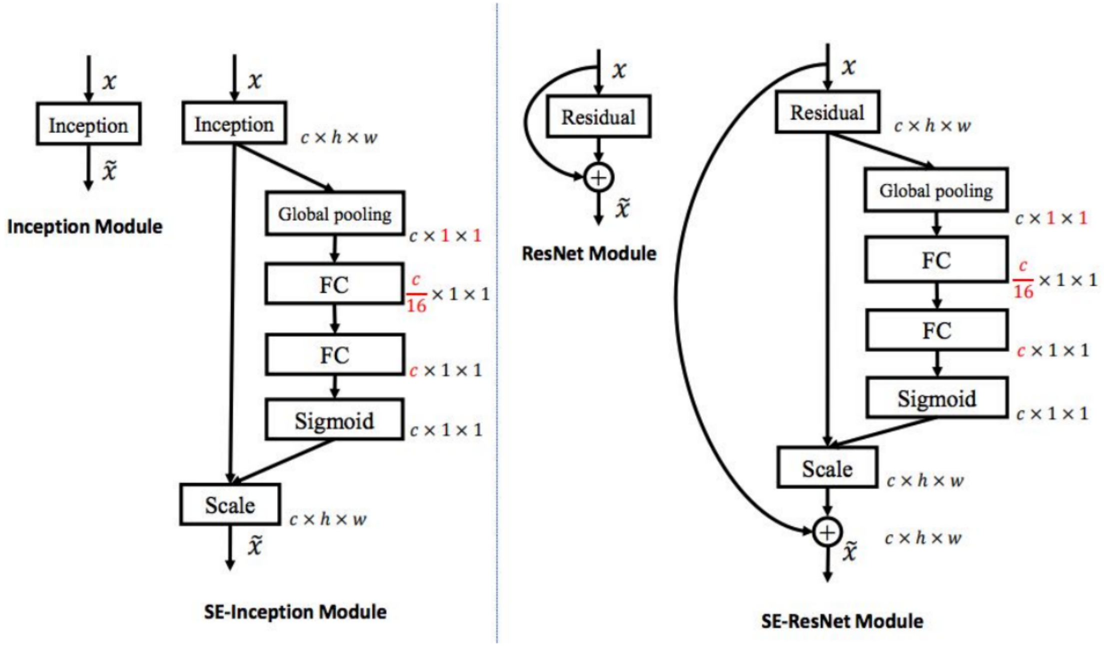
| | Inception-v3 | Inception-v4 | Inception-Resnet-v2 | Resnet-200 | Wrn-68-3 | Fusion（Val.） | Fusion（Test） |
|---|---|---|---|---|---|---|---|
| Err. (%) | 4.20 | 4.01 | 3.52 | 4.26 | 4.65 | 2.92 (-0.6) | 2.99 |

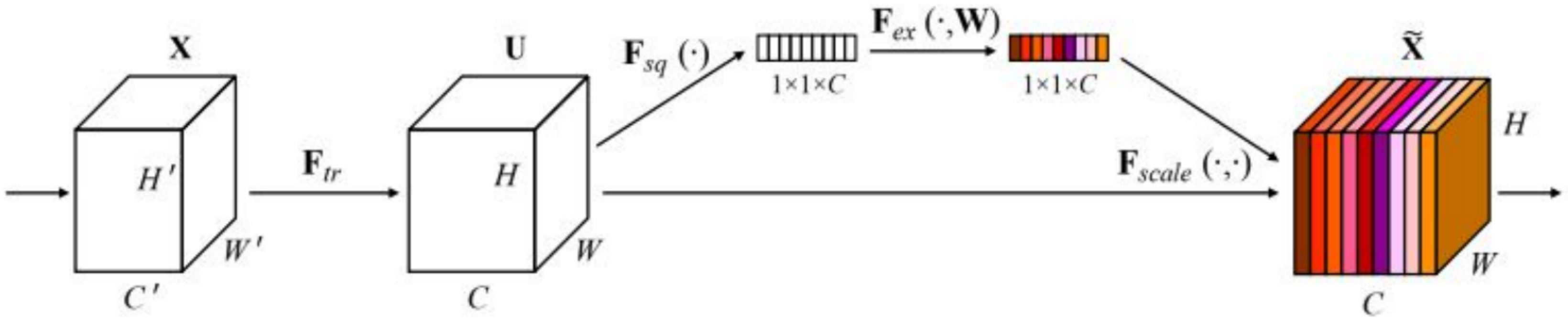
Improving ResNets...

# Squeeze-and-Excitation Networks (SENet)

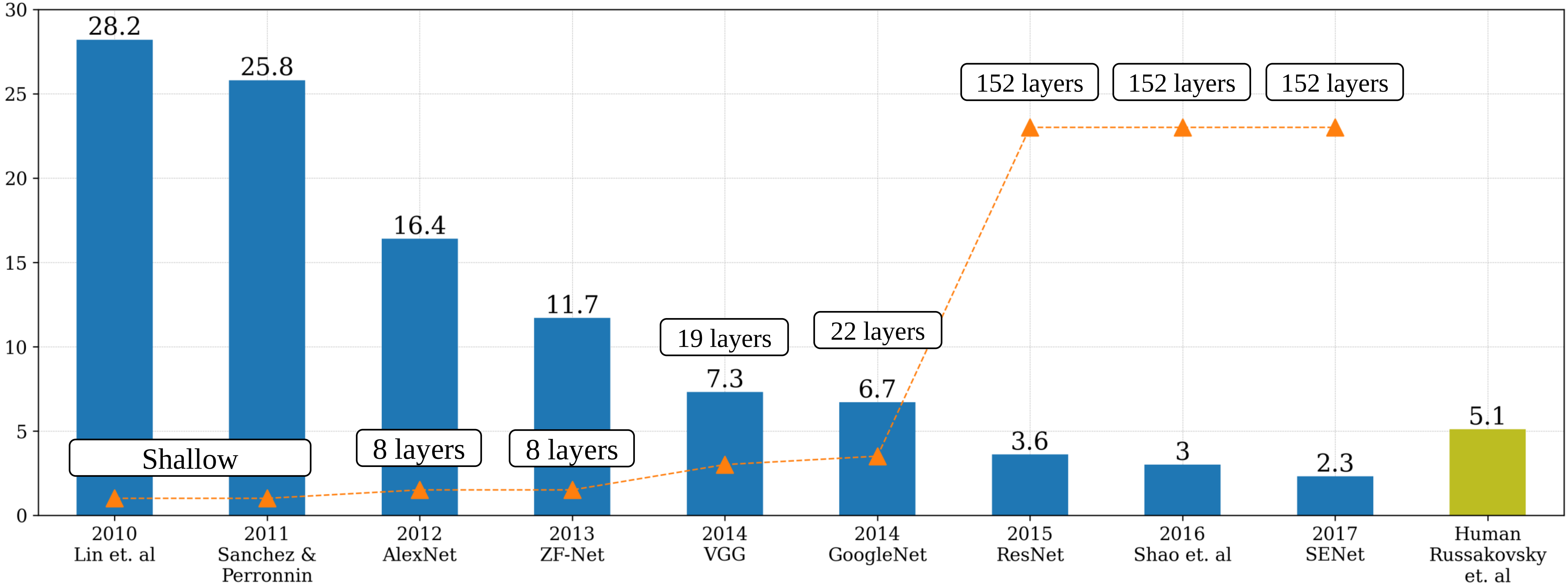
*[Hu et al. 2017]*

- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)



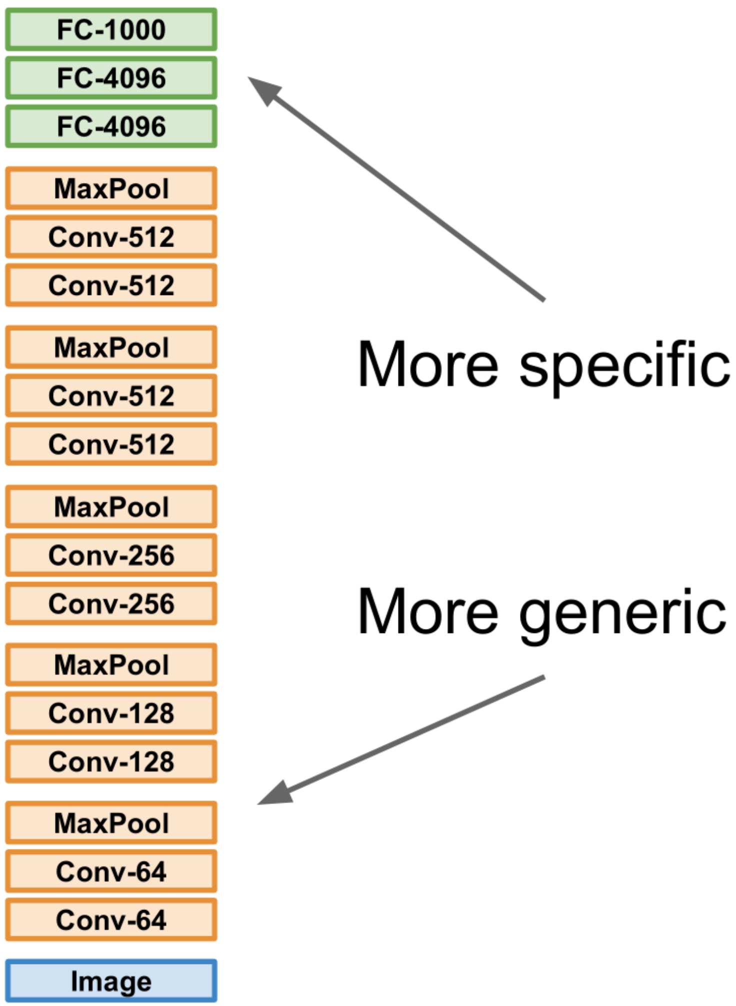



# ImageNet Challenge Error Rates over time




Completion of the challenge: Annual ImageNet competition no longer held after 2017
-> now moved to Kaggle.

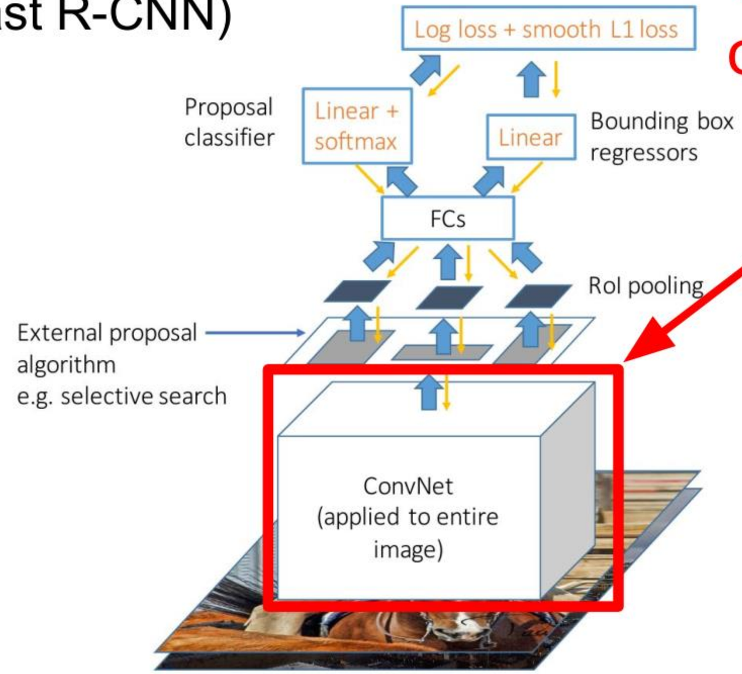
# Transfer Learning




| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer | You're in trouble… Try linear classifier from different stages |
| **quite a lot of data** | Finetune a few layers | Finetune a larger number of layers |

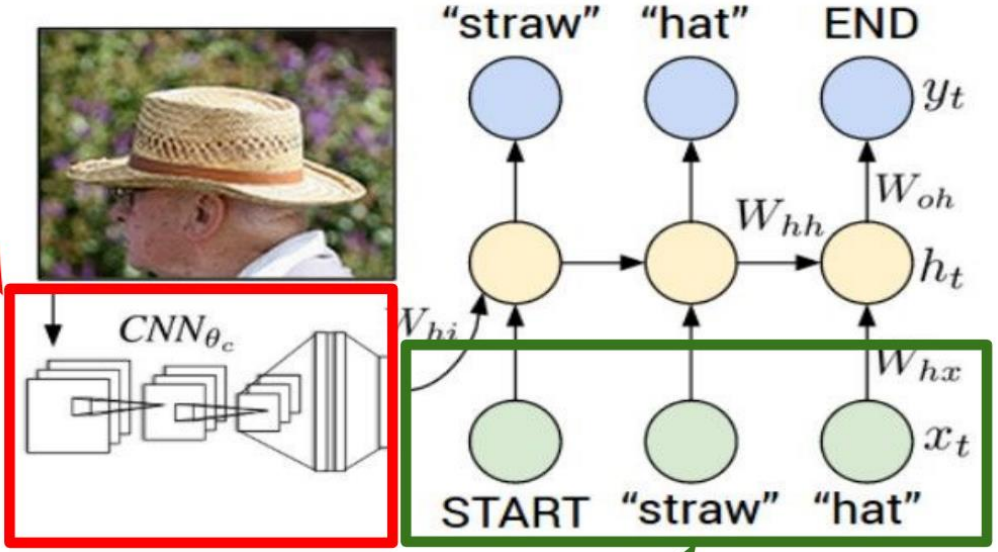
# Transfer learning with CNNs is pervasive…
## (it’s the norm, not an exception)

Object Detection
(Fast R-CNN)




CNN pretrained on ImageNet

Image Captioning: CNN + RNN

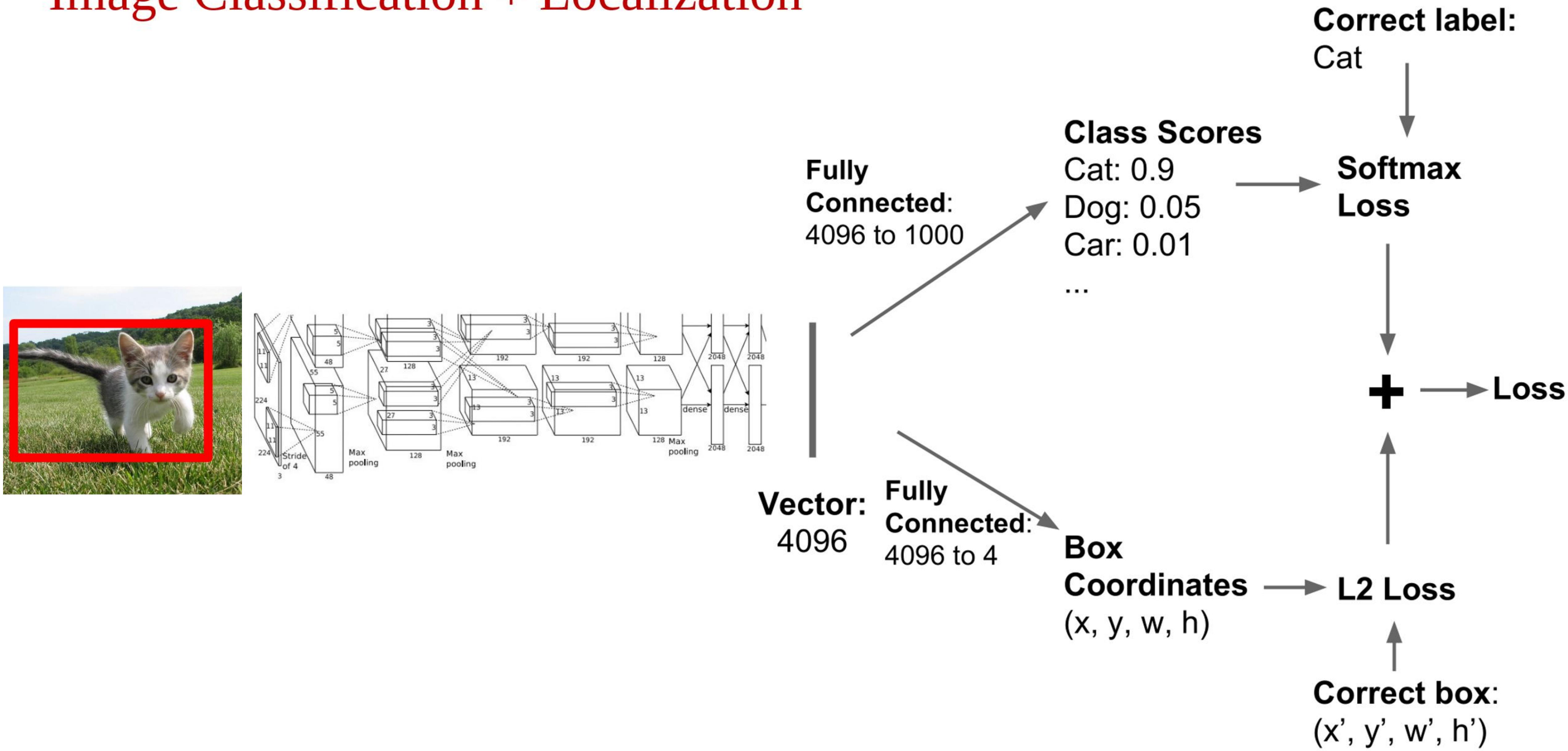



Word vectors pretrained with word2vec

Girshick, “Fast R-CNN”, ICCV 2015


Karpathy and Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions”, CVPR 2015

# Image Classification + Localization



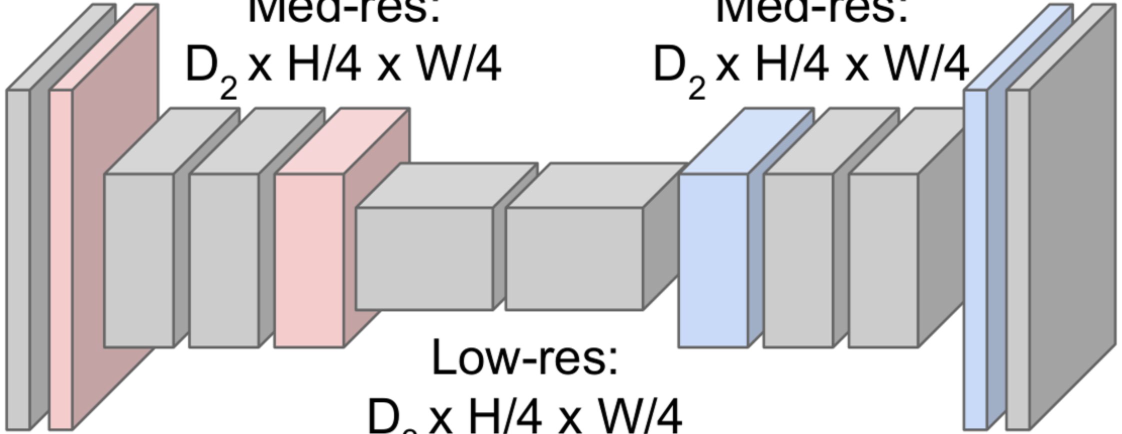
# Semantic Segmentation

**Downsampling**:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Upsampling**:
Unpooling or strided transposed convolution

Input:
$3 \times H \times W$




Predictions:
$H \times W$

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

# Deep Learning

- Typical Deep Learning networks formed using Convolutional and Max Pool layers

- Deep Learning models show higher generalization capabilities compared to classical Machine Learning methods

- Deep Learning models have been highly successful in problems across several domains - Computer Vision, Natural Language Processing, Information retrieval, Computational Biology and Chemistry, Astrophysics,…