

TBMI26

# Neural Networks and Learning Systems

## Lecture 6

### Deep Networks

**Michael Felsberg**

Computer Vision Laboratory

Department of Electrical Engineering

Linköping University, Sweden

# Introduction

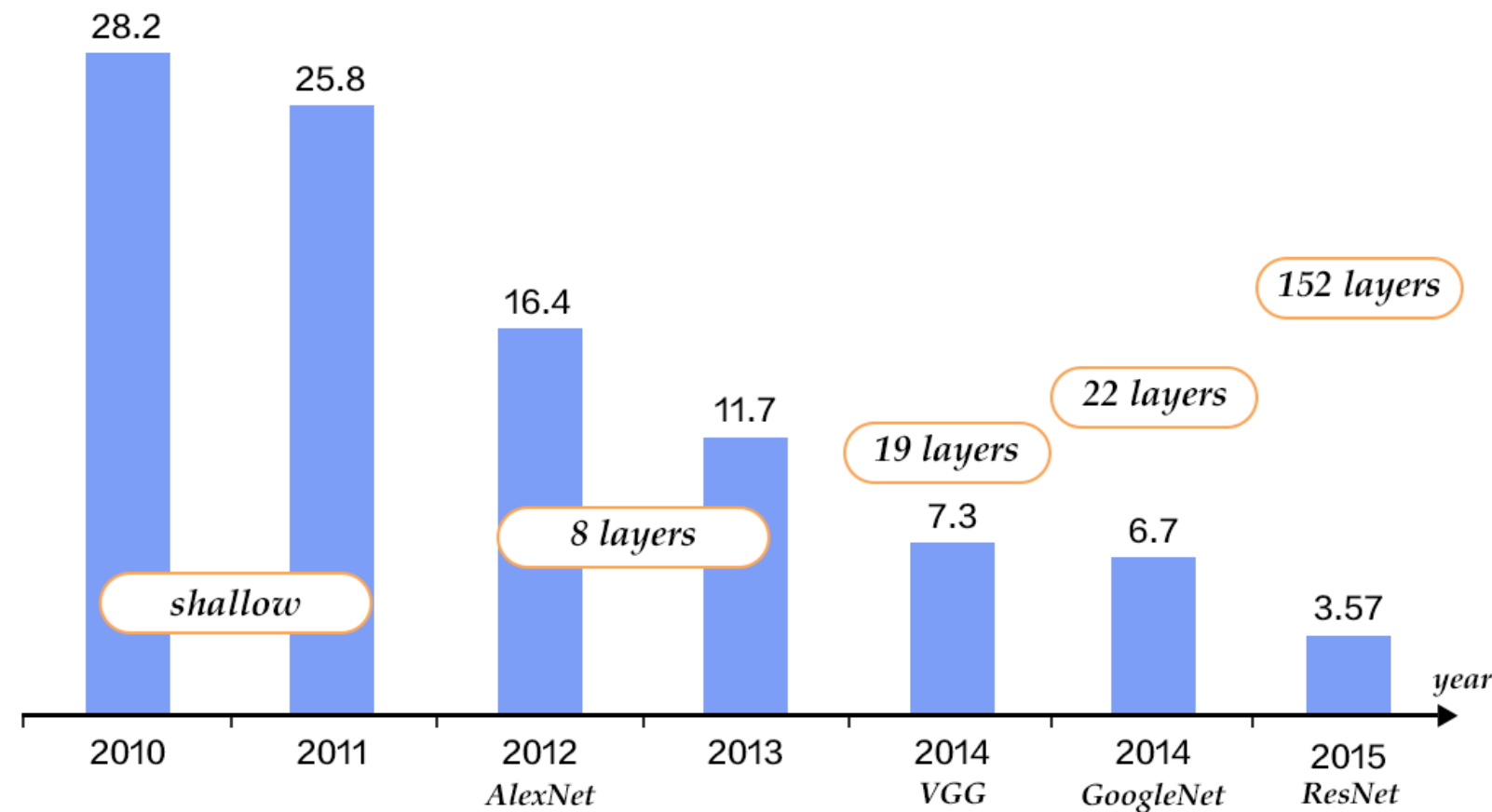
# TEDx-talk Yoshua Bengio



# ILSVRC-2010

- ImageNet Large Scale Visual Recognition Challenge [Deng et al. 2009]

- more than 14 million images
- More than 10 million images annotated
- More than 1 million images with bounding box



# Loss functions (cont.)

# Motivation: maximum likelihood estimation

- Probability distribution  $P(X=x; \theta) = C \exp(-(x - \theta)^2 / (2\sigma^2))$
- The maximum likelihood estimate for  $\theta$  is (assuming independence and i.i.d.)

$$\theta_{\text{ML}} = \arg \max_{\theta} \sum_{i=1}^N \log P(X = x^{(i)}; \theta)$$

$$= \arg \min_{\theta} \sum_{i=1}^N (x^{(i)} - \theta)^2$$

- Note:  
this example  
is parametric

$$= \frac{1}{N} \sum_{i=1}^N x^{(i)}$$

# Conditional log-likelihood

- *Supervised learning*: learn a conditional probability distribution over target values  $\mathbf{y}$ , given features  $\mathbf{x}$ .
- The assumption that the samples are i.i.d. yields

$$\boldsymbol{\theta}_{\text{ML}} = \arg \max_{\boldsymbol{\theta}} \sum_{i=1}^N \log P(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta})$$

- Same as minimizing *cross-entropy*  $E_X[-\log P(Y|X)]$
- Principled way to derive the cost function (incl. L2)

$$\text{cost}(\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}) = -\frac{1}{N} \sum_{i=1}^N \log P(\mathbf{y}^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta})$$

# Cross-entropy cost function

- Back to logistic activation:  $h(\mathbf{x}_i) = f(\theta \cdot \mathbf{x}_i) = \frac{1}{1 + e^{-\theta \cdot \mathbf{x}_i}}$

interpretation: conditional probability  $P(y_i = 1 \mid \mathbf{x})$  for binary random variable  $y_i$ .  $P(y_i = 0 \mid \mathbf{x}) = 1 - P(y_i = 1 \mid \mathbf{x})$

- The cross-entropy is the "natural" error function:

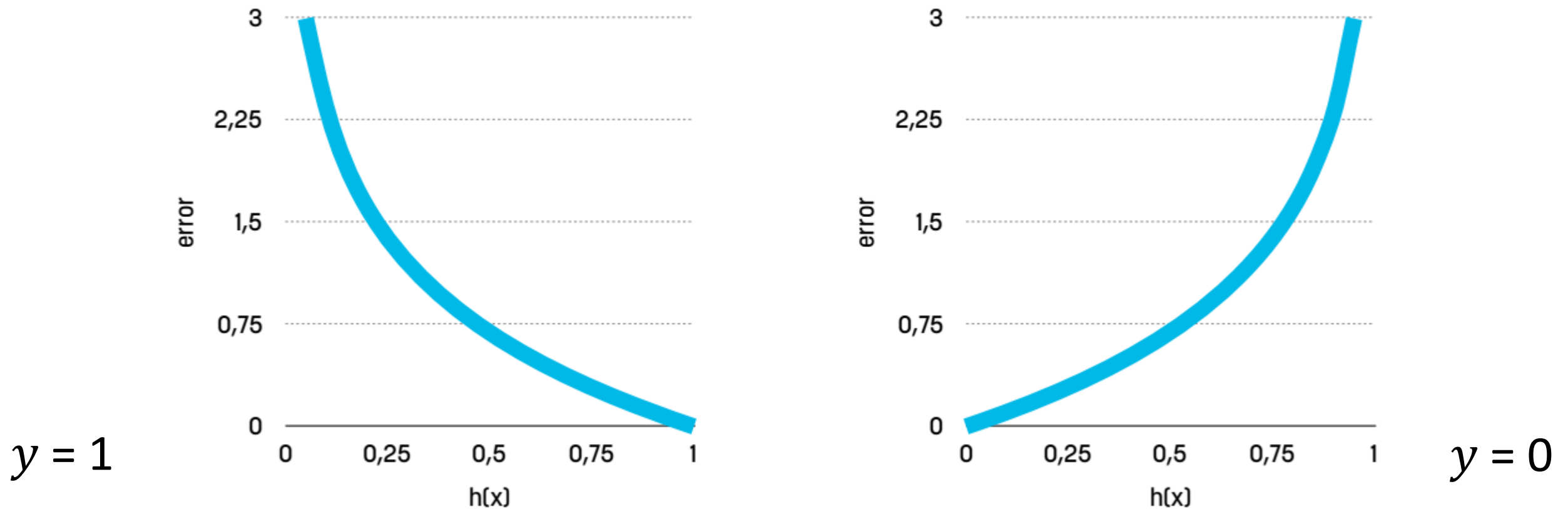
$$C = \begin{cases} -\ln h(\mathbf{x}_i) & \text{if } y_i = 1 \\ -\ln(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases}$$

- This is usually written as

$$C = -(y_i \ln h(\mathbf{x}_i) + (1 - y_i) \ln(1 - h(\mathbf{x}_i)))$$



# Sigmoid and cross-entropy balance each other



$$\frac{\partial}{\partial z} \ln f(z) = \frac{f'(z)}{f(z)} = 4 \frac{e^{-z}}{1 + e^{-z}} = 1 - f(z) \quad \frac{\partial}{\partial z} \ln(1 - f(z)) = -\frac{f'(z)}{1 - f(z)} = -\frac{1}{1 + e^{-z}} = -f(z)$$

$$\frac{\partial C}{\partial z} = -y_i(1 - f(z)) + (1 - y_i)f(z) = f(z) - y_i$$

# Cross-entropy cost function

- Back to soft-max activation:  $h_k(\mathbf{x}_i) = f_k(\boldsymbol{\theta}\mathbf{x}_i) = \frac{e^{\theta_k \cdot \mathbf{x}_i}}{\sum_l e^{\theta_l \cdot \mathbf{x}_i}}$

interpretation: conditional probability

$P(\mathbf{y}_i = (0, 1, 0, \dots) \mid \mathbf{x})$  for one-hot random vector  $\mathbf{y}_i$ .

$$P(\mathbf{y}|\mathbf{x}) = \prod_k h_k(\mathbf{x})^{y_k} = h_{k:y_k=1}(\mathbf{x})$$

- The cross-entropy is the "natural" error function:

$$-\ln P(\mathbf{y}_n|\mathbf{x}_n; n = 1 \dots N) = -\sum_n \sum_k y_{kn} \ln h_k(\mathbf{x}_n)$$

- Often the soft-max is (wrongly) called to be the cost

# Surrogate loss function

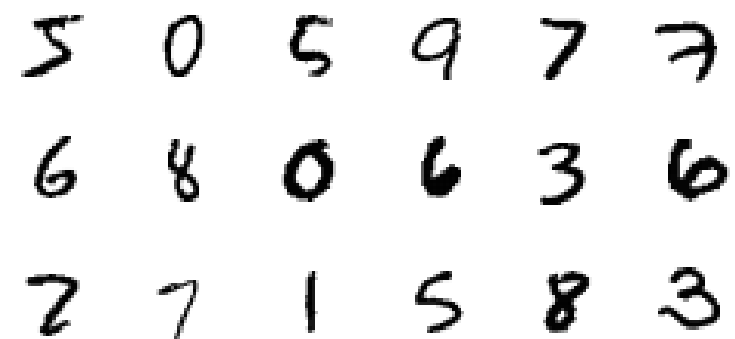
- Example: 0-1 loss for class membership (one-hot coding) surrogated / replaced with log-likelihood (cross-entropy)
  - Motivation 1: Gradient descent does not allow for 0-1 loss (discontinuous loss) – cross-entropy is continuously differentiable
  - Motivation 2: Test error with 0-1 loss might be lower for training with cross-entropy than 0-1 loss – cross-entropy pushes classes apart even if 0-1 loss on training set is zero

# Example: Handwritten digit recognition

# Handwritten digit recognition

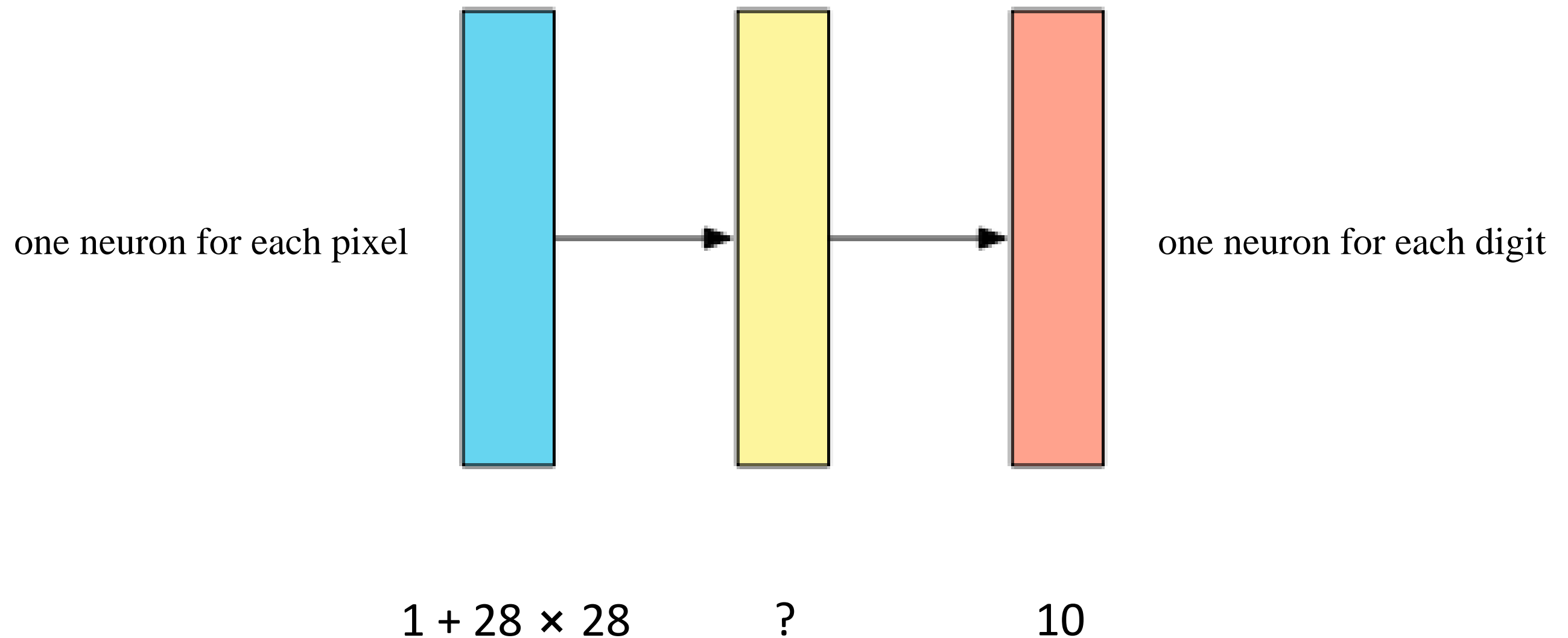
You are to build a feedforward net that takes in a greyscale image of a handwritten digit and outputs the digit (an integer).

supervised learning



5	0	5	9	7	7
6	8	0	6	3	6
2	7	1	5	8	3

# Basic network architecture



# How to use the network

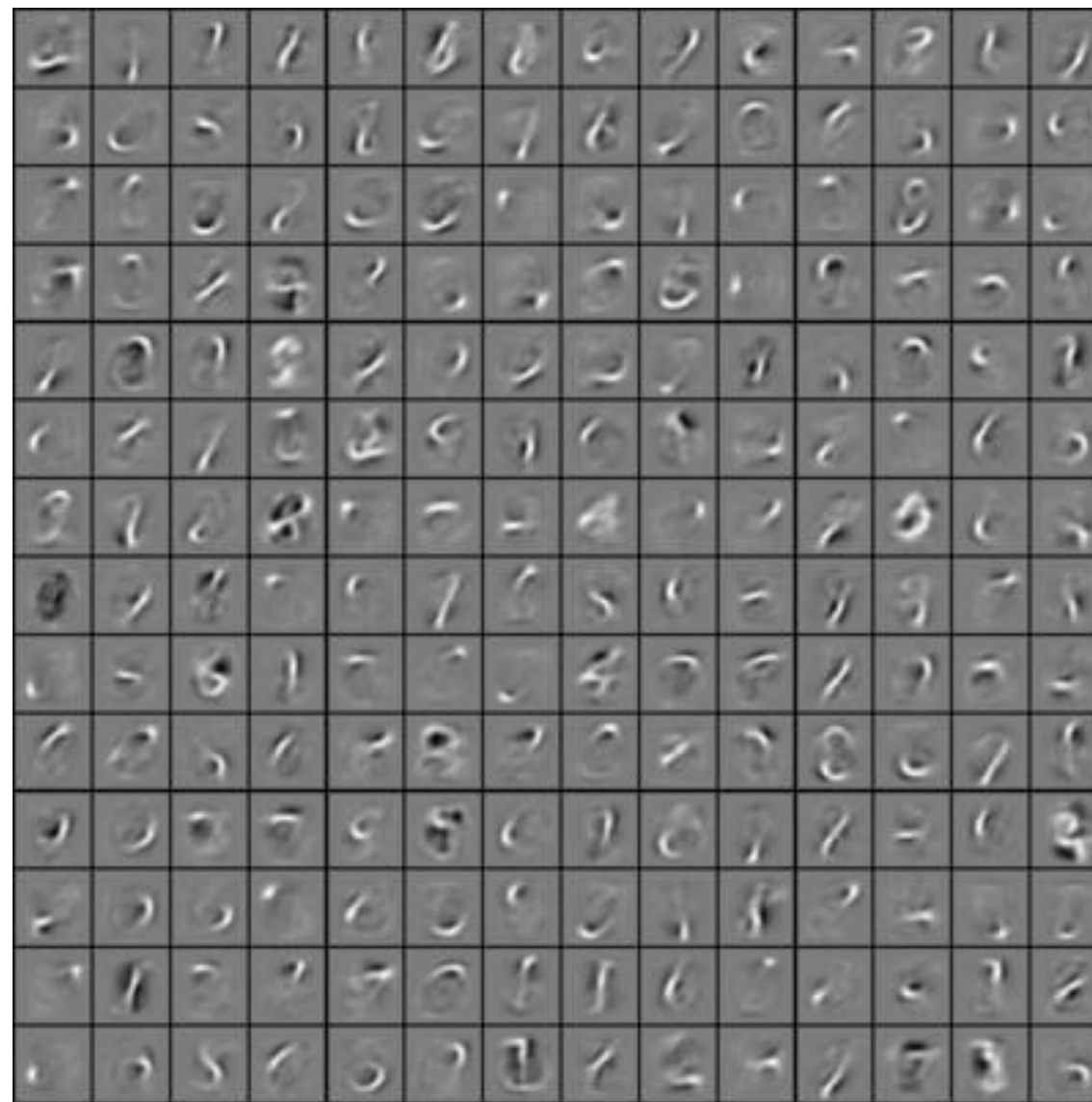
- Translate each image to a vector  $\mathbf{x}$  with  $1 + 28 \times 28$  components, where component  $x_i$  is the greyscale value for pixel  $i$  in the image.

The greyscale value is a fraction  $k/255$  between 0 (black) and 1 (white).

- Feed the image to the network.
- Find the neuron  $y_i$  in the output layer that has the highest activation and predict the digit  $i$ .

Softmax layer

# What does the net learn?



Source: [Kylin-Xu](#)



# How to train the network

- To train the network we use the MNIST database, which consists of 70,000 handwritten labelled digits.
- Each target is translated into a vector  $\mathbf{y}$  with 10 components, where  $y_i$  is 1 if the target equals  $i$  and 0 otherwise (one-hot).

Example: If the target is 3 then  $y_3 = 1$ , and all other components zero.

# Regularization

# Norm-based regularisation

- We can regularise the training of a neural network by adding an additional term to the error function.
- L2-regularisation: Give preference to parameter vectors with smaller Euclidean norms ('lengths'):

$$\|\boldsymbol{\theta}\|_2^2 = \boldsymbol{\theta}^\top \boldsymbol{\theta}$$

- L1-regularisation: Give preference to parameter vectors with smaller absolute-value norms:

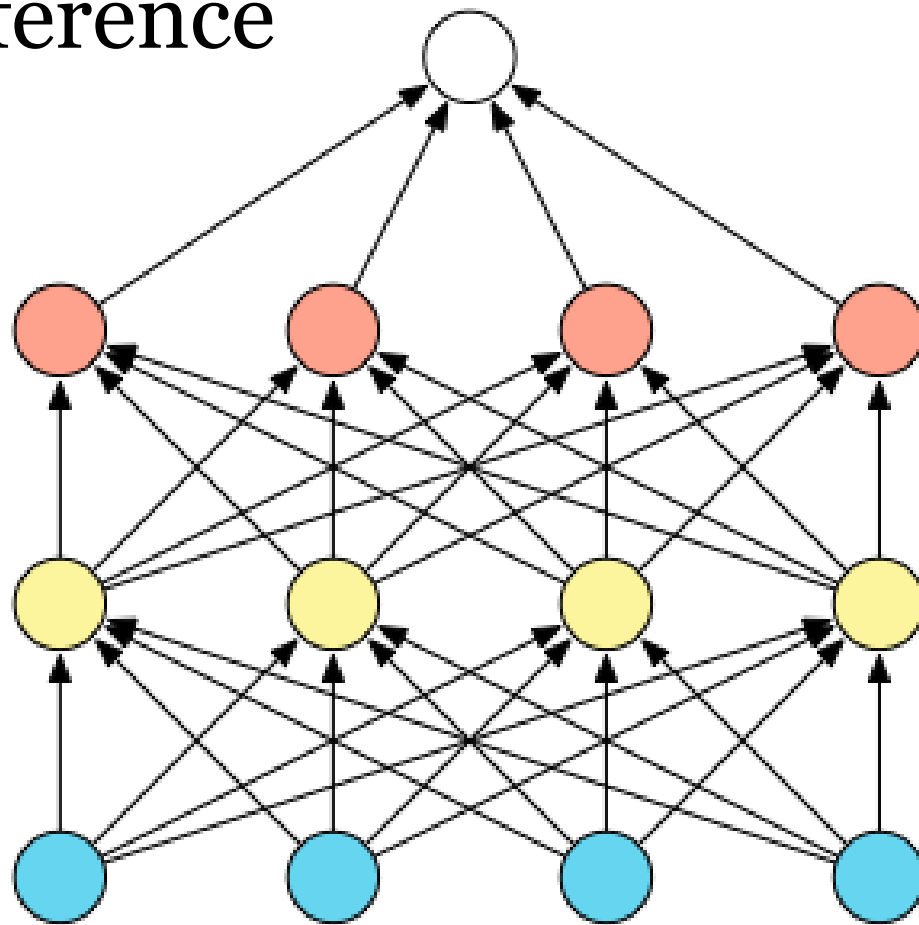
$$\|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i|$$

# Selected regularization techniques

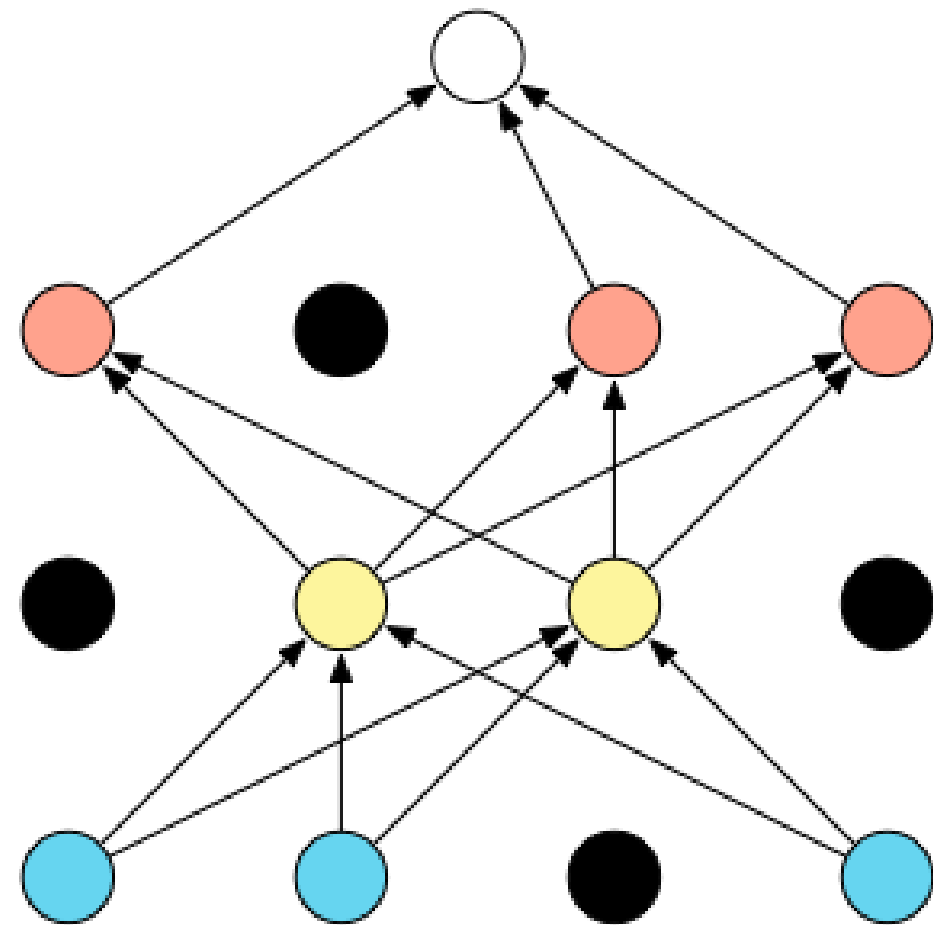
- *Dataset augmentation*. Generate new training data by systematically transforming the existing data.  
example: rotating and scaling images
- *Early stopping*. Stop the training when the validation set error goes up and backtrack to the previous set of parameters.
- *Bagging / ensemble* methods. Train several different models separately, then have all of the models vote on the output.

# Dropout

- Randomly set a fraction of units to zero during training / inference



unmodified neural net



net after applying dropout

# Discussion: Local minima

- Deep networks always lead to local minima (model identifiability problem)
- Not necessarily a problem: mostly similarly low cost
- *True global minimum not relevant*
- Test: plot norm of gradient over time
  - if local minima are the problem, the norm needs to shrink close to zero
- In high dimensions, saddle points are much more likely than local minima; local minima have low costs

# Initialization and Normalization

# Initialization

- Gradient-based learning might lead to wrong directions or long trajectories
- Initialization in an area connected to a solution results in faster learning
- Initial weights *may not* be symmetric or identical, as this will just lead to redundancy
- Initial values should be large to *break symmetry*
- Initial values should not be too large to avoid numerical issues



# Initialization

- Determining initial values is costly
- Good heuristic: draw weights from Gaussian or uniform distribution
- Normalized (Xavier) initialization for  $m$  inputs,  $n$  outputs (tanh; add factor 4 for sigmoid):

$$\theta_{ij} \sim U \left( -\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}} \right)$$

- Other approaches: random orthogonal matrices, sparse initialization, ...

# Batch normalization

- In deep networks, the simultaneous update of layers will have second, third, ... order effects
- To reduce this effect,  $\mathbf{z}$  is replaced during learning

$$z'_i = \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}} \quad \mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m z_i \quad \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (z_i - \mu_{\mathcal{B}})^2$$

where the empirical mean and variance are computed over the (mini)batch of size  $m$

- This normalized activation is transformed  $z''_i = \gamma z'_i + \beta$  with moving-average parameters

# Optimization Algorithm

# Minibatch methods

- *Batch* – deterministic: whole training set
- *Stochastic gradient descent*: single training samples
- *Minibatch*: subsets from training set
  - Large enough to exploit multicore architecture
  - Size coincides with accuracy of e.g. gradients
  - Small enough to fit into memory
  - Often power of 2: 32 .. 256
  - Number of minibatches coincides with regularization
  - Repeated drawing: *epochs*

# Learning rate

- Stochastic methods (SGD, MB) require decaying learning rate
- Conditions for convergence:
  - Sum of learning rates goes to infinity
  - Sum of squared learning rates is bounded
- Often applied: linear combination of initial and final learning rate

$$\alpha_k = (1 - \lambda)\alpha_0 + \lambda\alpha_\tau \quad \lambda = \frac{k}{\tau}$$

- Rule of thumb:
  - Initial rate larger than what initial results suggests
  - Final rate at about 1%

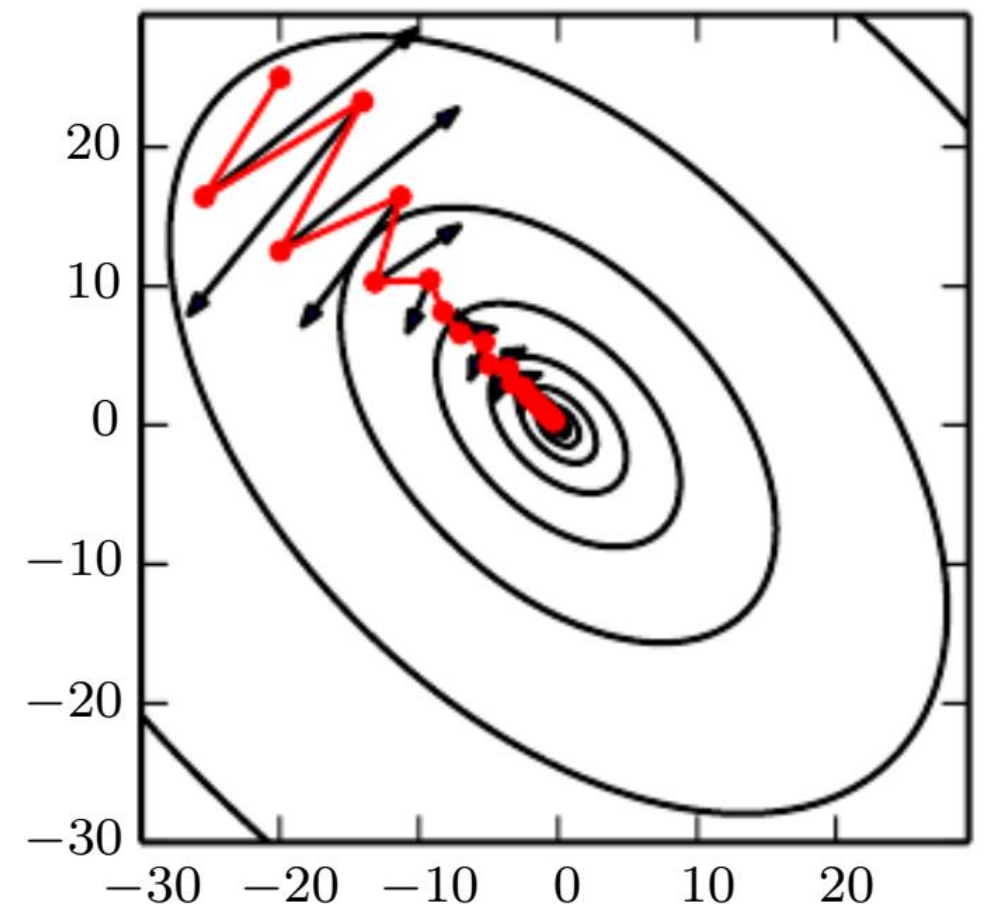
# Momentum

- Gradient is problematic due to curvature, small scale, or noise
- Introduce velocity as exponentially decaying moving average of gradients

$$\Delta\theta \leftarrow \lambda\Delta\theta - (1 - \lambda)\nabla_{\theta}J$$

$$\theta \leftarrow \theta + \alpha\Delta\theta$$

- Hyperparameter of 0.9 gives about factor 10 speed-up



# Adam [Kingma & Ba, 2015]

- Based on RMSProp (adaptive gradient weight)

$$\begin{aligned}\mathbf{n} &\leftarrow \nu \mathbf{n} + (1 - \nu) \nabla_{\boldsymbol{\theta}} J \odot \nabla_{\boldsymbol{\theta}} J \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J / (\sqrt{\mathbf{n}} + \varepsilon)\end{aligned}$$

element-wise operations

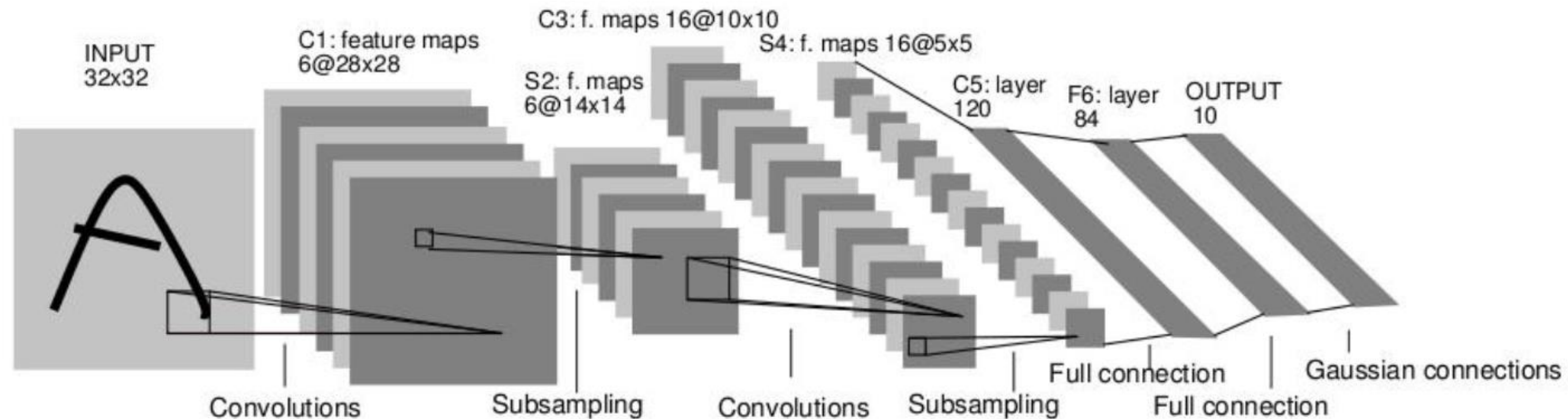
- Combined with momentum and **bias correction**

$$\begin{aligned}\Delta \boldsymbol{\theta} &\leftarrow \lambda \Delta \boldsymbol{\theta} - (1 - \lambda) \nabla_{\boldsymbol{\theta}} J \\ \widehat{\Delta \boldsymbol{\theta}} &\leftarrow \Delta \boldsymbol{\theta} / (1 - \lambda^t) \\ \mathbf{n} &\leftarrow \nu \mathbf{n} + (1 - \nu) \nabla_{\boldsymbol{\theta}} J \odot \nabla_{\boldsymbol{\theta}} J \\ \hat{\mathbf{n}} &\leftarrow \mathbf{n} / (1 - \nu^t) \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \widehat{\Delta \boldsymbol{\theta}} / (\sqrt{\hat{\mathbf{n}}} + \varepsilon)\end{aligned}$$

# Network Architectures

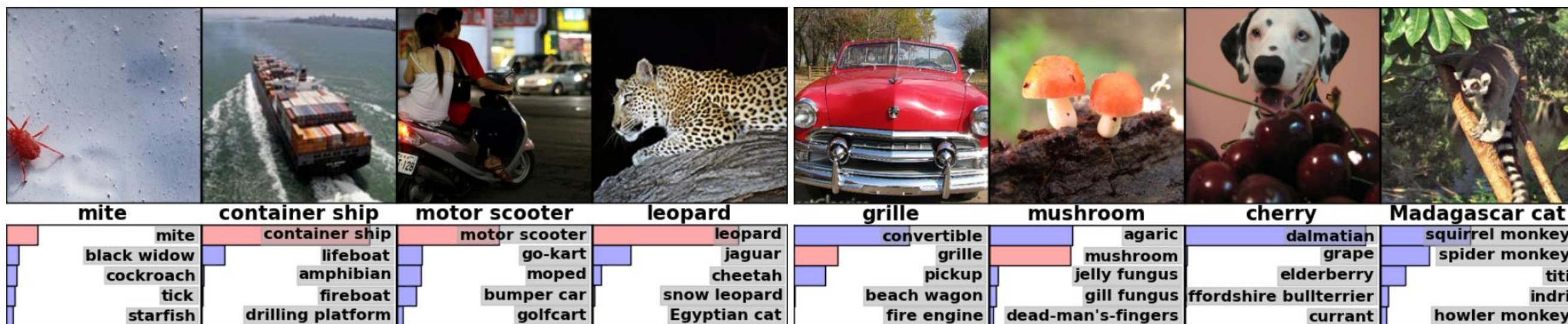
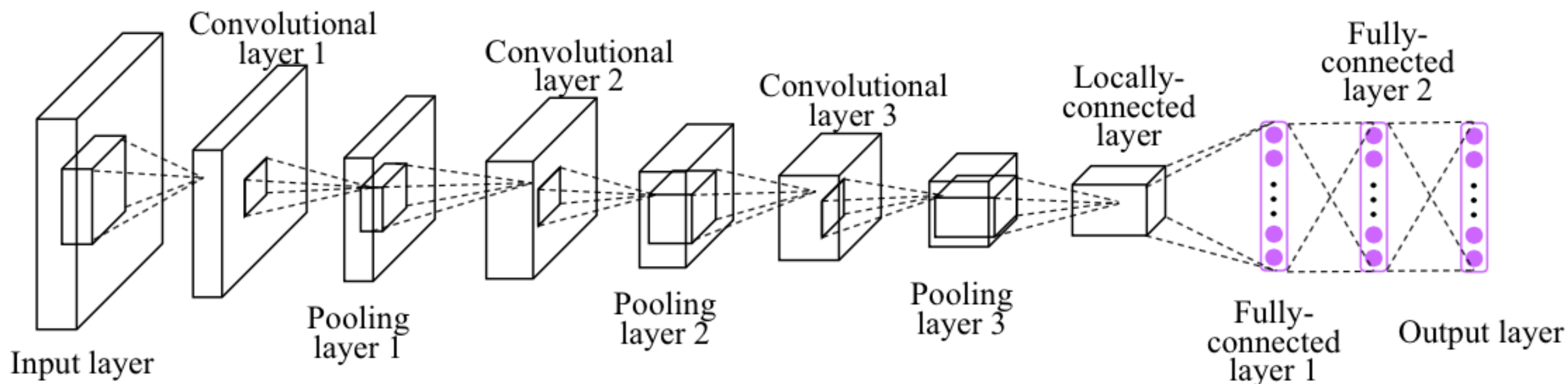


# LeNet 5 [LeCun et al. 1998]

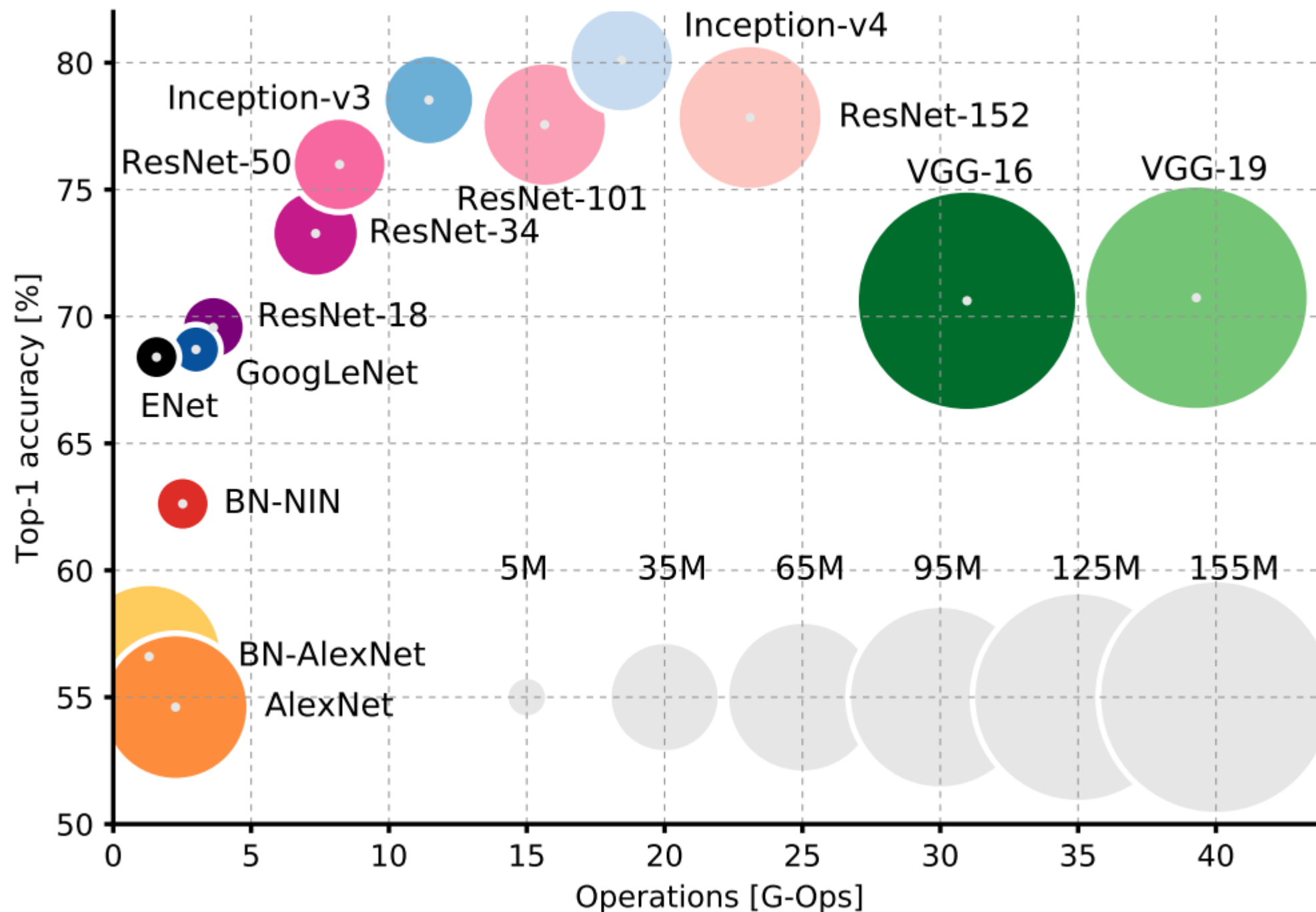


- Average pooling
- Sigmoid/tanh activation
- 60K training samples, 10 classes (MNIST)

# Deep Neural Networks (simple)

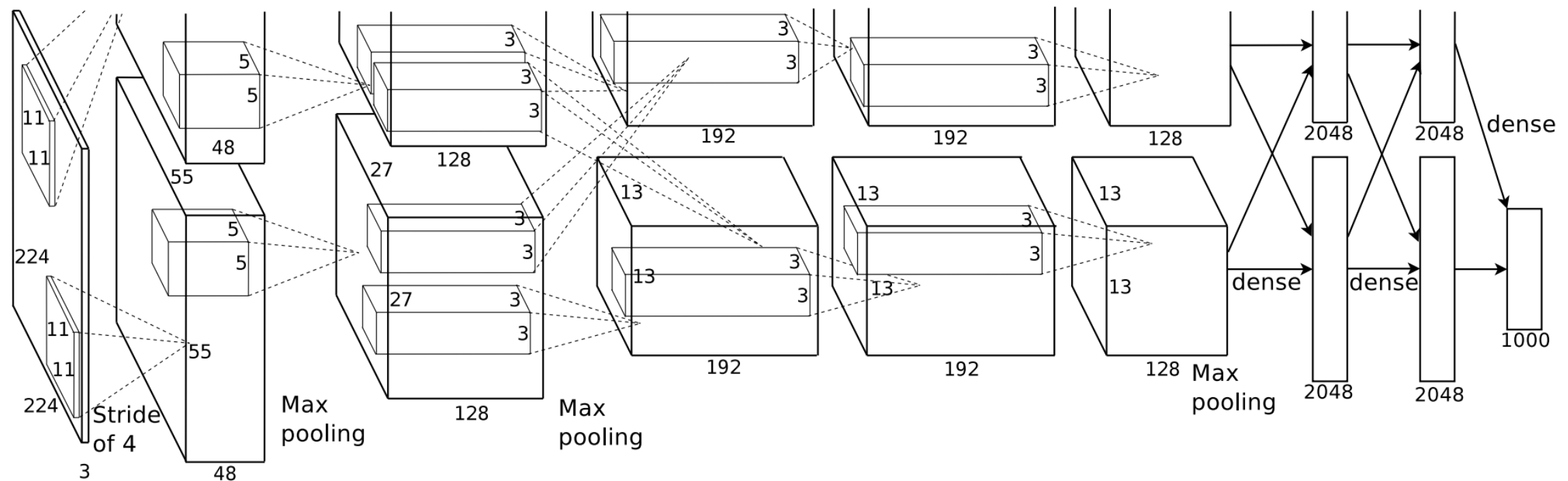


# Which architecture?



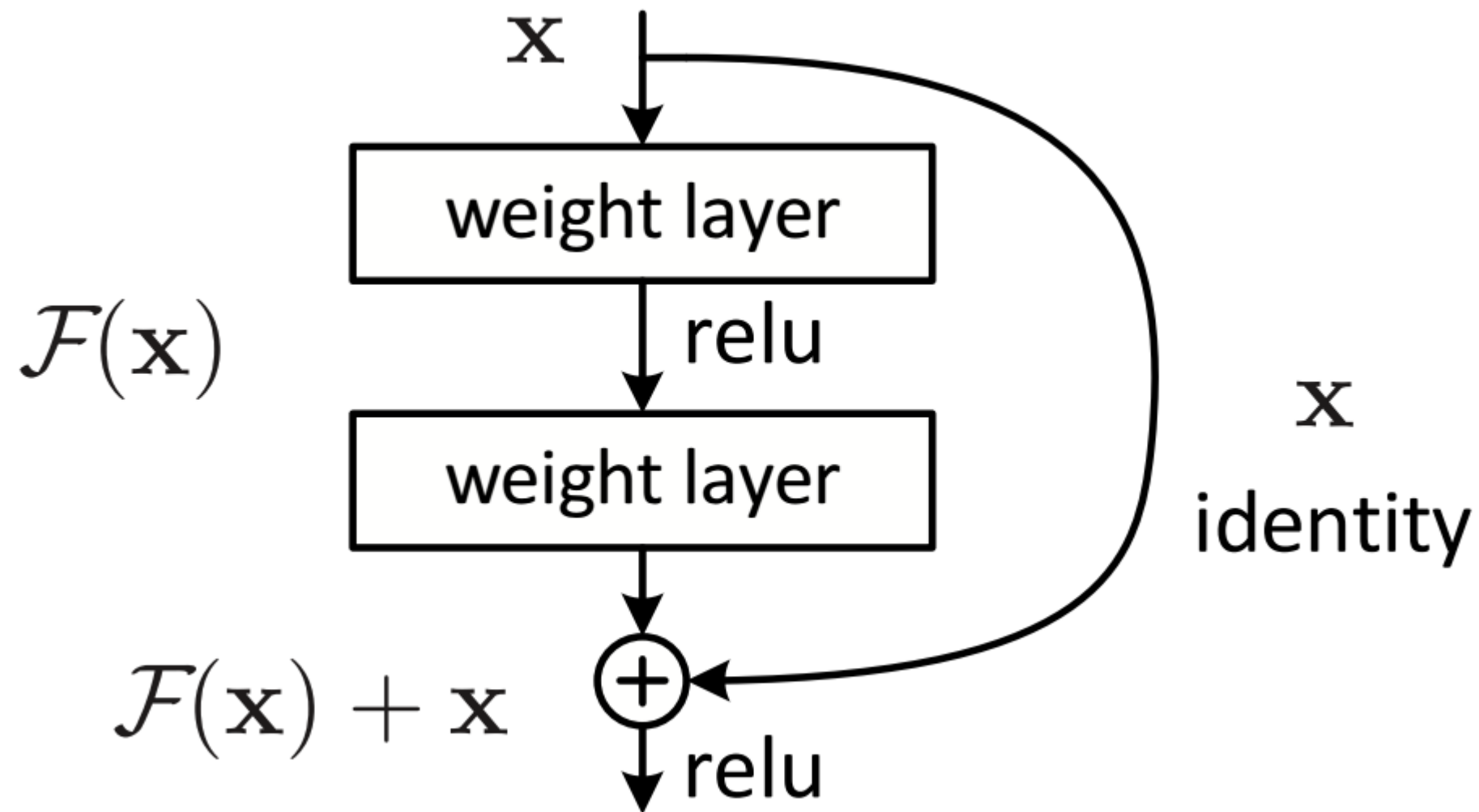
Canziani, Paszke, Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678

# AlexNet [Krizhevsky et al. 2012]



- Max pooling
- ReLU activation
- Dropout regularization
- 1.2M training samples, 1000 classes (ILSVRC)

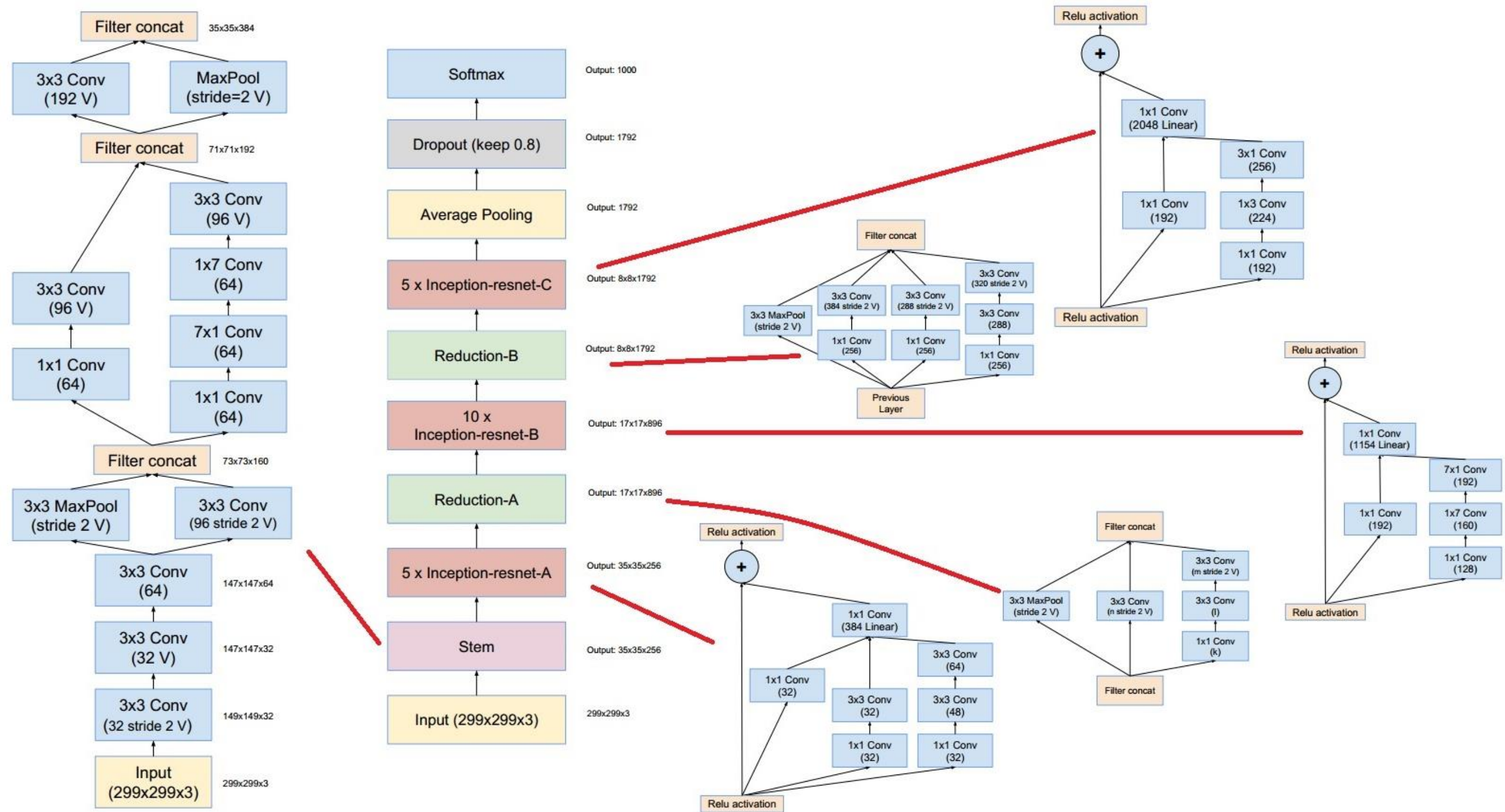
# Shortcuts / ResNet (152 layers!)



He et al. Deep Residual Learning for Image Recognition. CVPR, IEEE Press, 2016



# Inception v4



# Definition of Visual Tasks



dog	<input checked="" type="checkbox"/>
person	<input type="checkbox"/>
chair	<input type="checkbox"/>
sofa	<input checked="" type="checkbox"/>
bottle	<input type="checkbox"/>

## Classification Task:

Is there a dog in the image?

Is there a sofa in the image?

## Categorization Task:

Is this an indoor scene?



## Localization Task:

Where is the dog in the image?

## Detection Task:

Where are known objects (dog, sofa) in the image?

## Tracking Task (generic/specific):

Where is the object/dog from the first frame in all subsequent frames of the image sequence?

Michael Felsberg  
michael.felsberg@liu.se

[www.liu.se](http://www.liu.se)