TBMI26
Neural Networks and Learning Systems
Lecture 6
Deep Networks

Michael Felsberg
Computer Vision Laboratory
Department of Electrical Engineering
Linköping University, Sweden

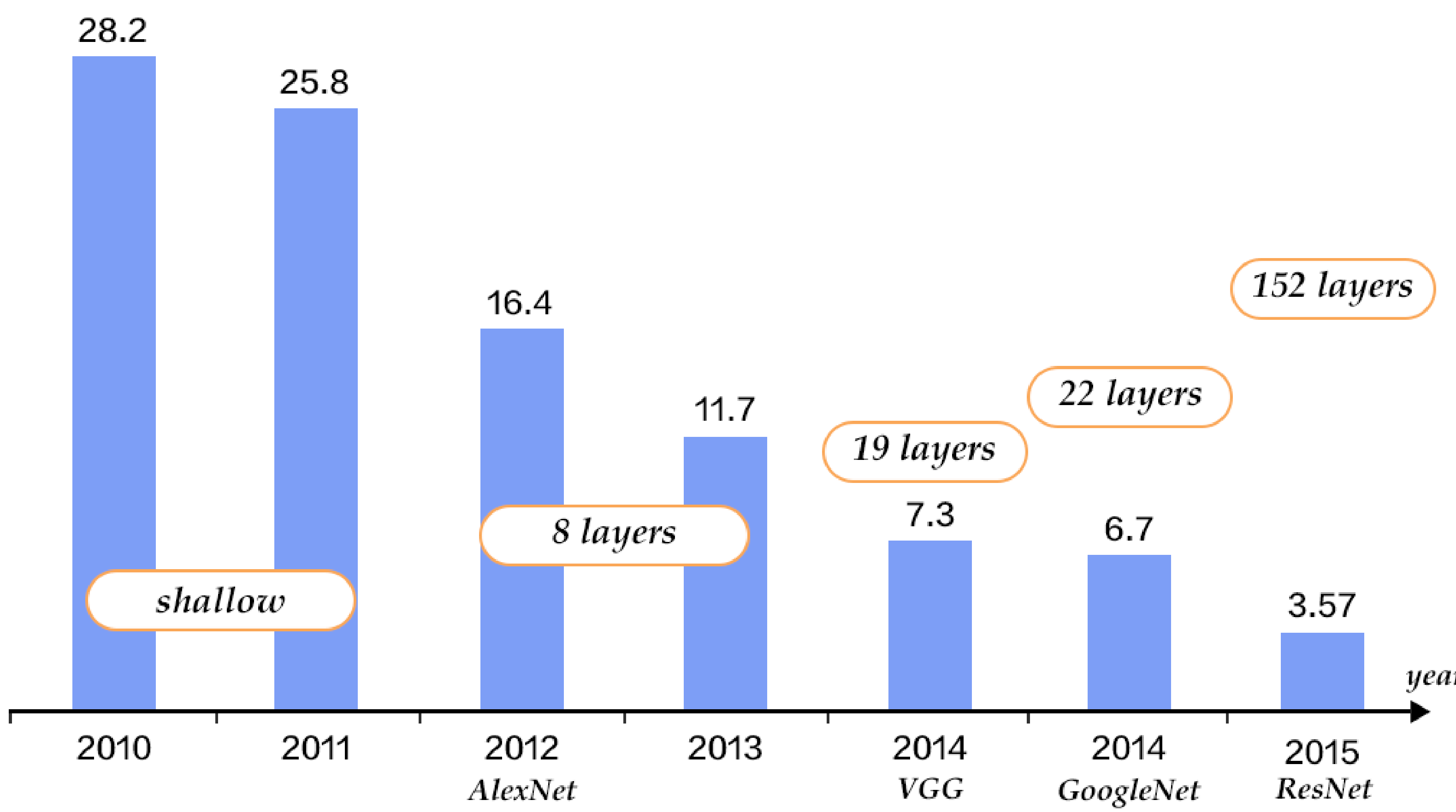
LINKÖPINGS
UNIVERSITET

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.)

LINKÖPINGS UNIVERSITET

Example: maximum likelihood estimation

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

$$\begin{aligned}\theta_{\mathrm{ML}} &= \arg\max_\theta \sum_{i=1}^{N} \log P(X = x^{(i)}; \theta) \\ &= \arg\min_\theta \sum_{i=1}^{N} (x^{(i)} - \theta)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N} x^{(i)}\end{aligned}$$

- Note:
  this example
  is parametric

Conditional log-likelihood

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

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log P\left(\boldsymbol{y}^{(i)} \,\middle|\, \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

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

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

Cross-entropy cost function

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

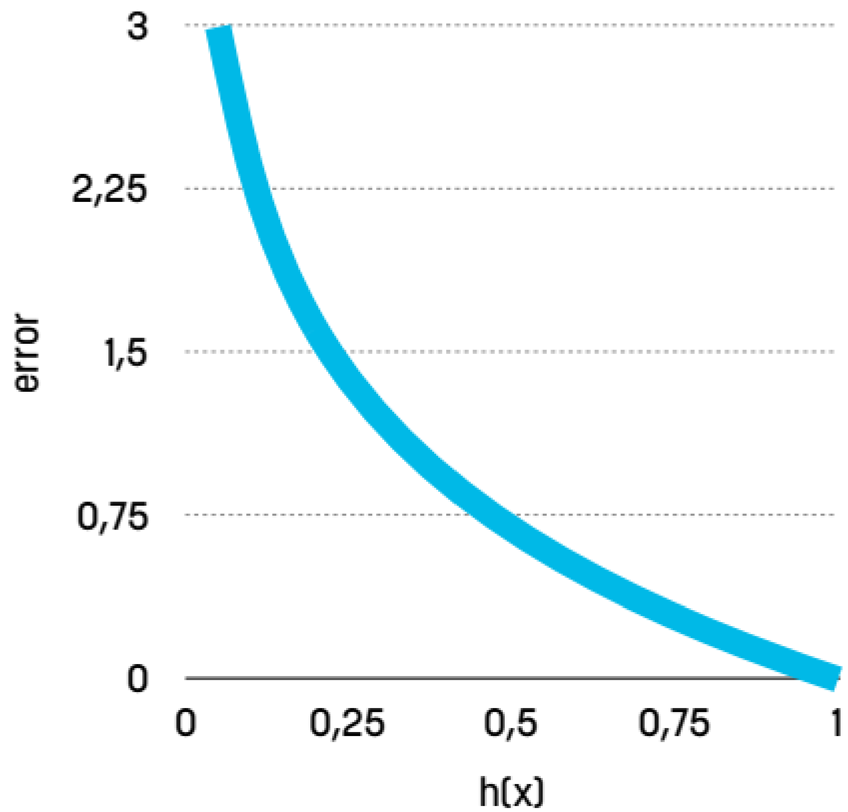
  interpretation: conditional probability $P(y_i = 1 \mid \boldsymbol{x})$ for binary random variable y_i. $P(y_i = 0 \mid \boldsymbol{x}) = 1 - P(y_i = 1 \mid \boldsymbol{x})$
- The cross-entropy is the ”natural” error function:

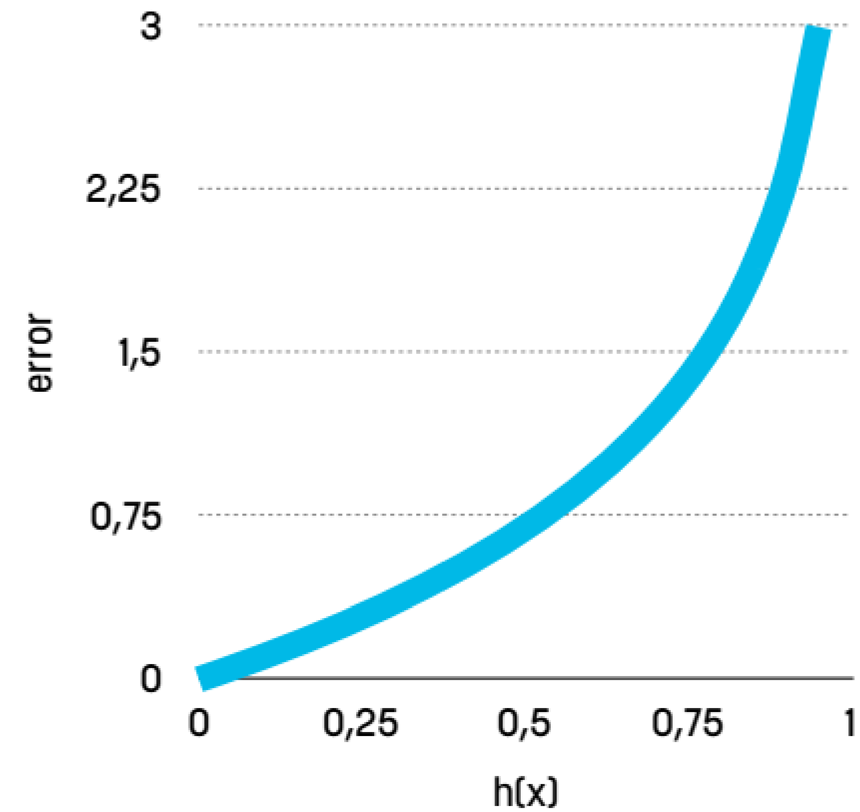
$$C = \begin{cases} -\ln h(\mathbf{x}_i) & \text{if } y_i = 1 \\ -\ln(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases}$$

- 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





$y = 1$

$y = 0$

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

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

Cross-entropy cost function

- Back to soft-max activation: $h_k(\mathbf{x}_i) = f_k(\mathbf{W}\mathbf{x}_i) = \dfrac{e^{\mathbf{w}_k \cdot \mathbf{x}_i}}{\sum_l e^{\mathbf{w}_l \cdot \mathbf{x}_i}}$

  interpretation: conditional probability
  $P(\boldsymbol{y}_i = (0, 1, 0, \ldots) \mid \boldsymbol{x})$ for one-hot random vector $\boldsymbol{y}_i$.

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

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

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

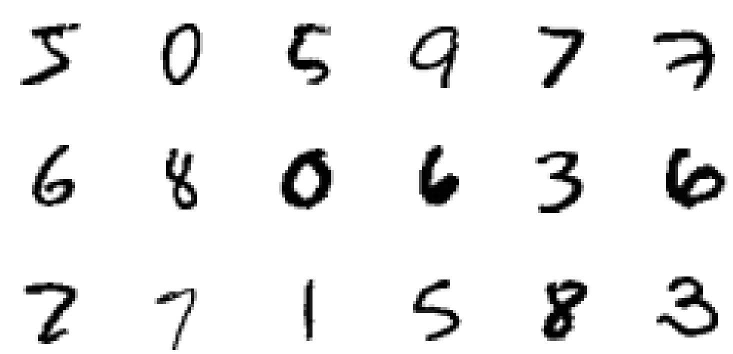
- Often the soft-max is (wrongly) called to be the cost

Example: Handwritten digit recognition

Handwritten digit recognition

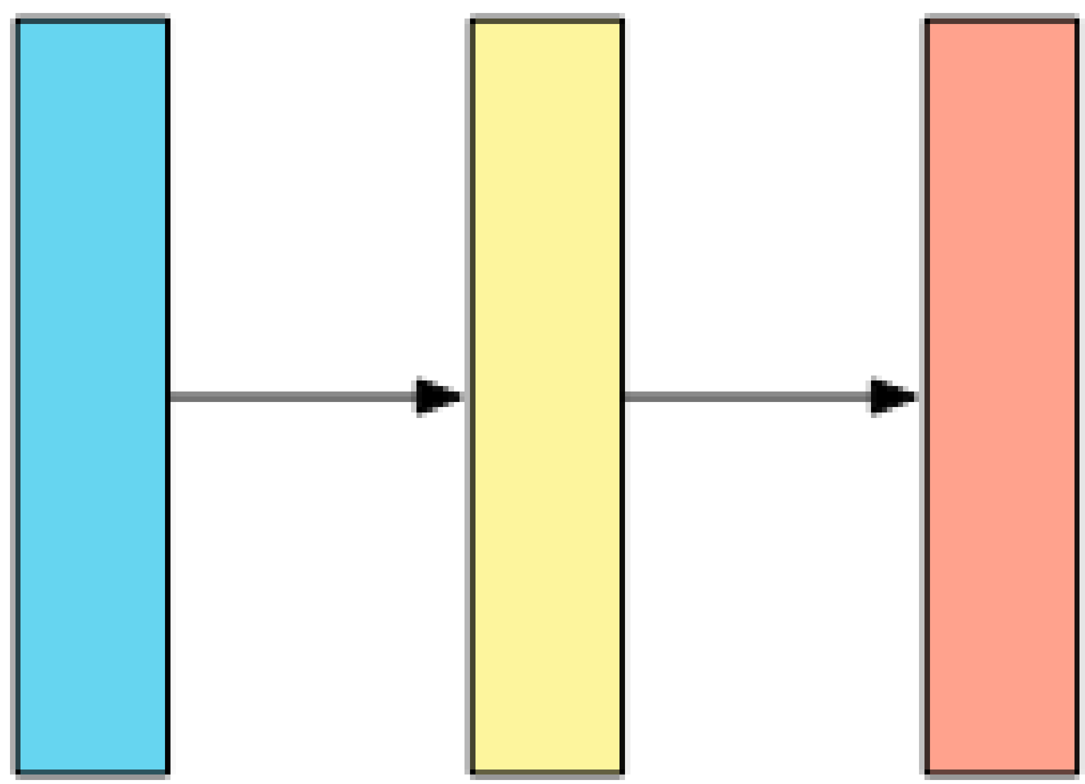
You are to build a feedforward net that takes in a greyscale image of a handwritten digit and outputs the digit (an integer).

supervised learning



Basic network architecture

one neuron for each pixel



one neuron for each digit

$1 + 28 \times 28$? 10

How to use the network

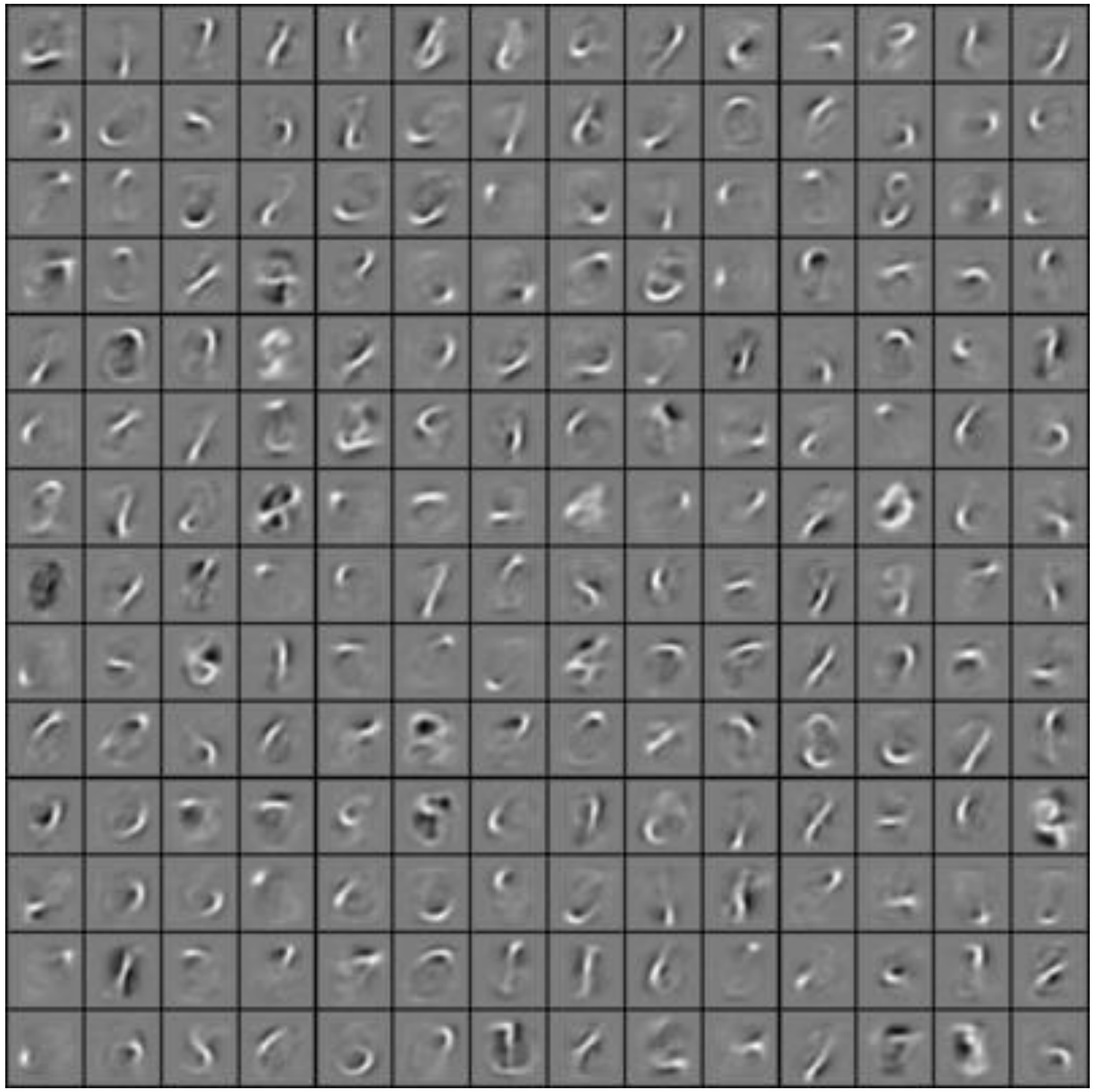
- Translate each image to a vector $\boldsymbol{x}$ with $1 + 28 \times 28$ components, where component x_i is the greyscale value for pixel i in the image.

  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 $\boldsymbol{y}$ 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.

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

$$\eta_k = (1-\beta)\eta_0 + \beta\eta_\tau$$

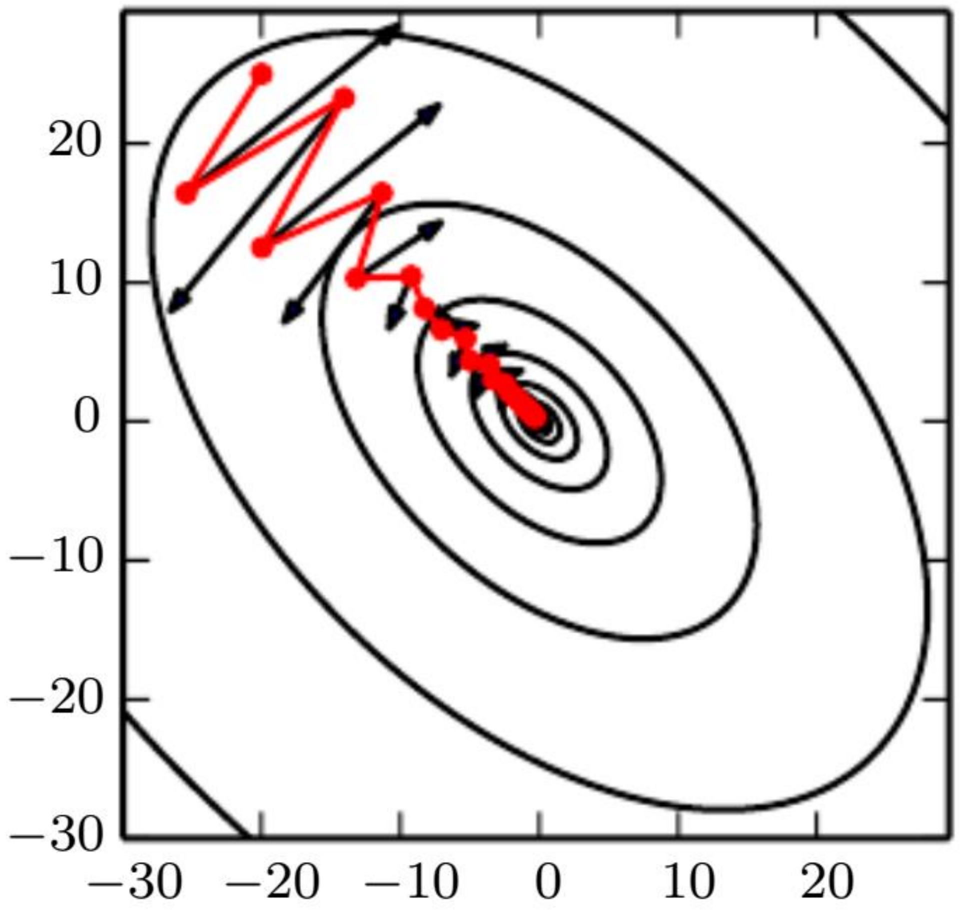
- Rule of thumb:
  - Initial rate larger than what initial results suggests
  - Final rate at about 1%

Momentum

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

$$\begin{aligned} \Delta\mathbf{w} &\leftarrow \alpha\Delta\mathbf{w} - (1-\alpha)\nabla_{\mathbf{w}}\varepsilon \\ \mathbf{w} &\leftarrow \mathbf{w} + \eta\Delta\mathbf{w} \end{aligned}$$

- Hyperparameter of 0.9 gives about factor 10 speed-up



Adam [Kingma & Ba, 2015]

- Based on RMSProp (adaptive gradient weight)

$$\begin{aligned}\mathbf{n} &\leftarrow \nu\mathbf{n} + (1-\nu)\nabla_{\mathbf{w}}\varepsilon \odot \nabla_{\mathbf{w}}\varepsilon \\ \mathbf{w} &\leftarrow \mathbf{w} - \eta\nabla_{\mathbf{w}}\varepsilon/(\sqrt{\mathbf{n}}+\lambda)\end{aligned}$$

element-wise operations

- Combined with momentum and bias correction

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

Initialization and Normalization

LINKÖPINGS UNIVERSITET

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

$$\mathrm{W}_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

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

Batch normalization

- In deep networks, the simultaneous update of layers will have second, third, … order effects
- To avoid this, $\boldsymbol{H}$=$\boldsymbol{XW}$+$\boldsymbol{b}$ is replaced during learning

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$\boldsymbol{\mu} = \frac{1}{m} \sum_i \boldsymbol{H}_{i,:}$$

mean and variance are computed over the (mini)batch

$$\boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m} \sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2}$$

- This *whitening* of activation is then transformed as $\gamma \boldsymbol{H}' + \boldsymbol{\beta}$ with suitable parameters

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

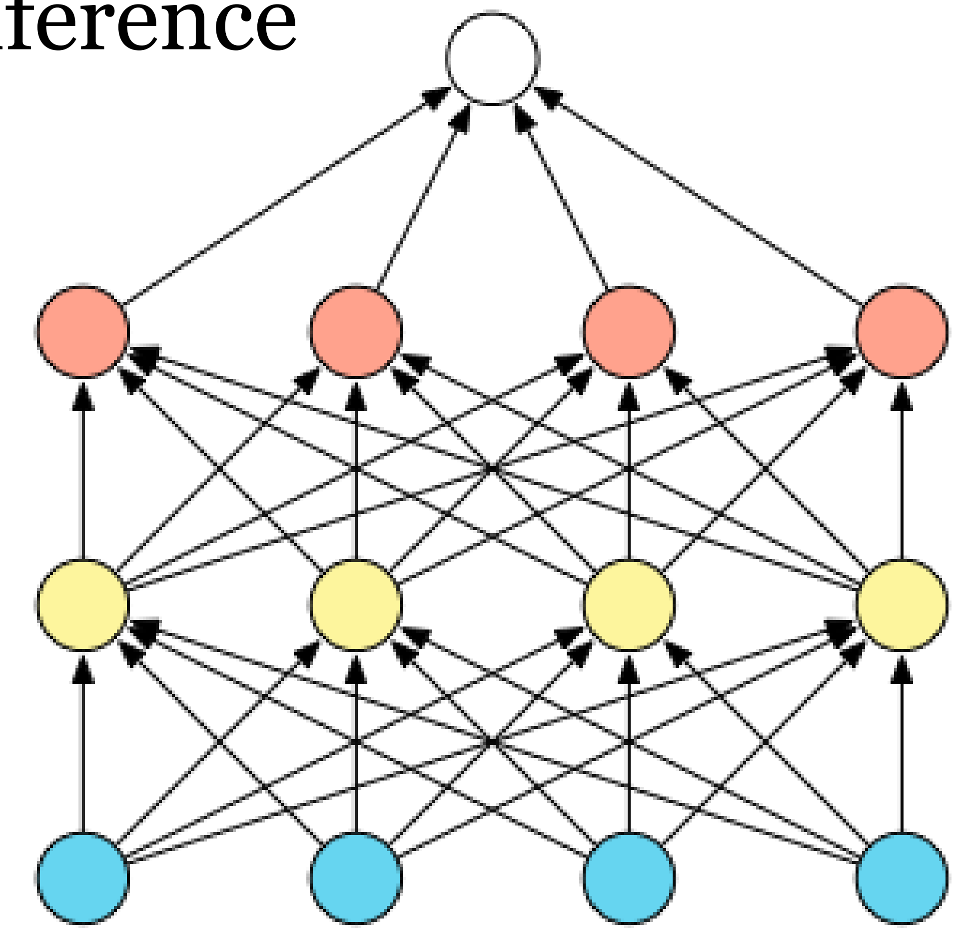
- *Dataset augmentation*. Generate new training data by systematically transforming the existing data.

  example: rotating and scaling images

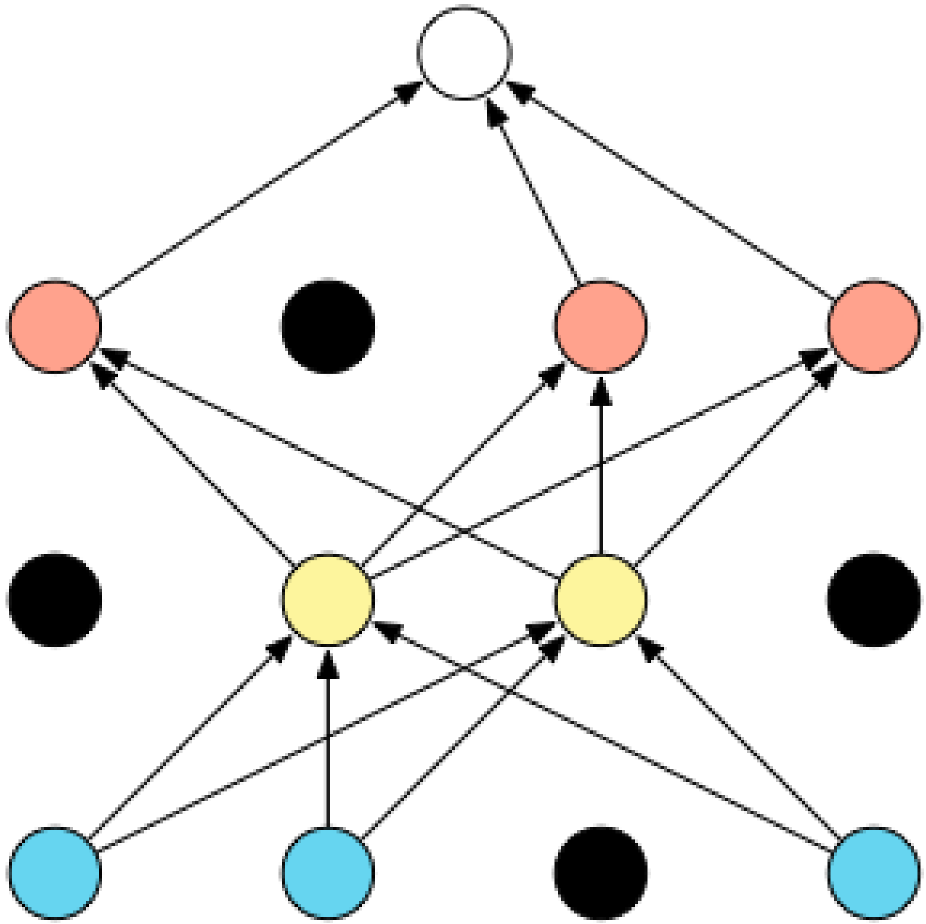
- *Early stopping*. Stop the training when the validation set error goes up and backtrack to the previous set of parameters.
- *Bagging / ensemble* methods. Train several different models separately, then have all of the models vote on the output.

Dropout

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



unmodified neural net



net after applying dropout

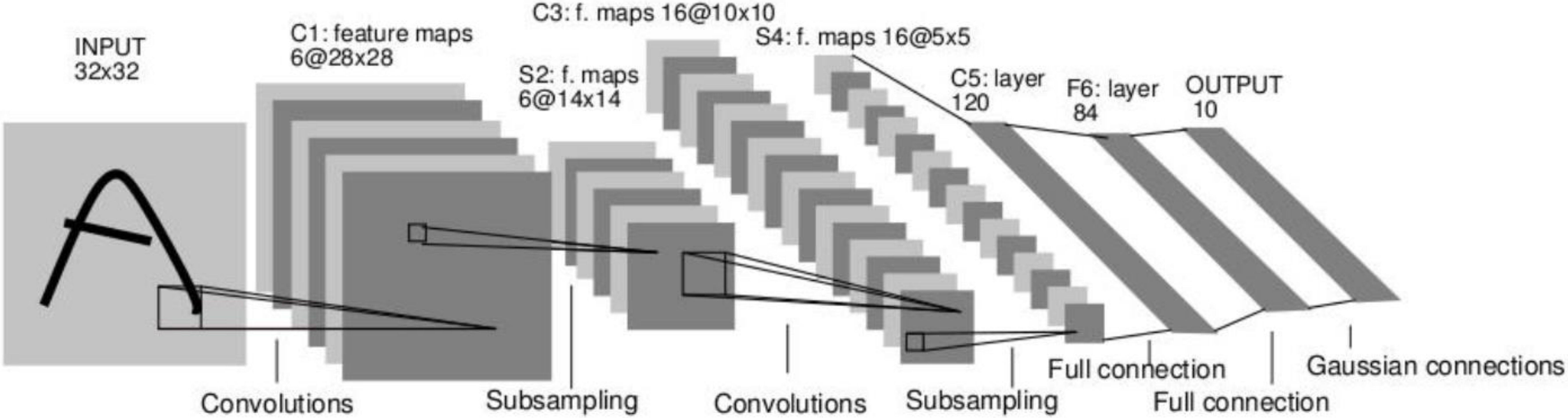
Discussion: Local minima

- 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

Network Architectures

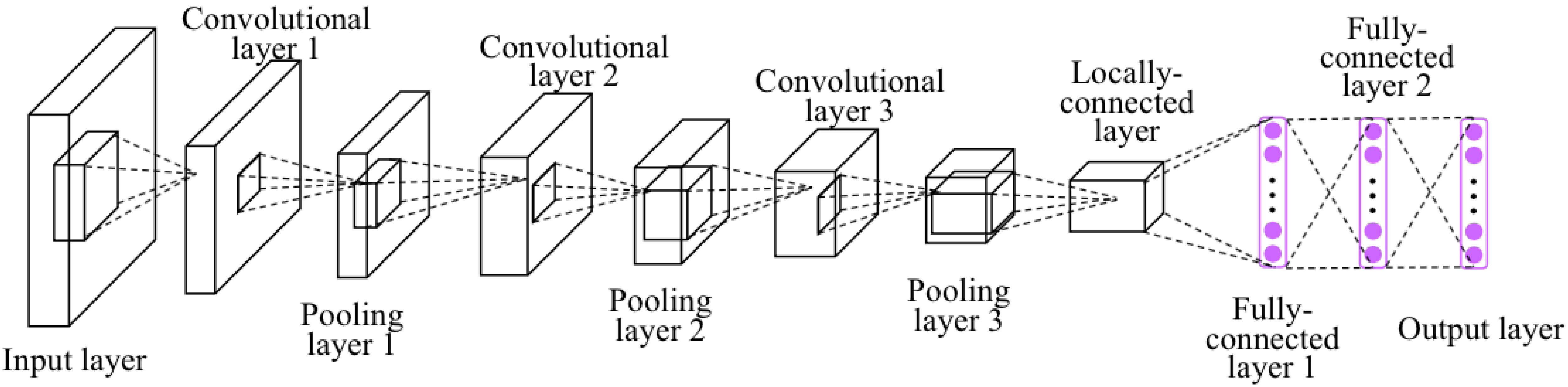
LINKÖPINGS UNIVERSITET

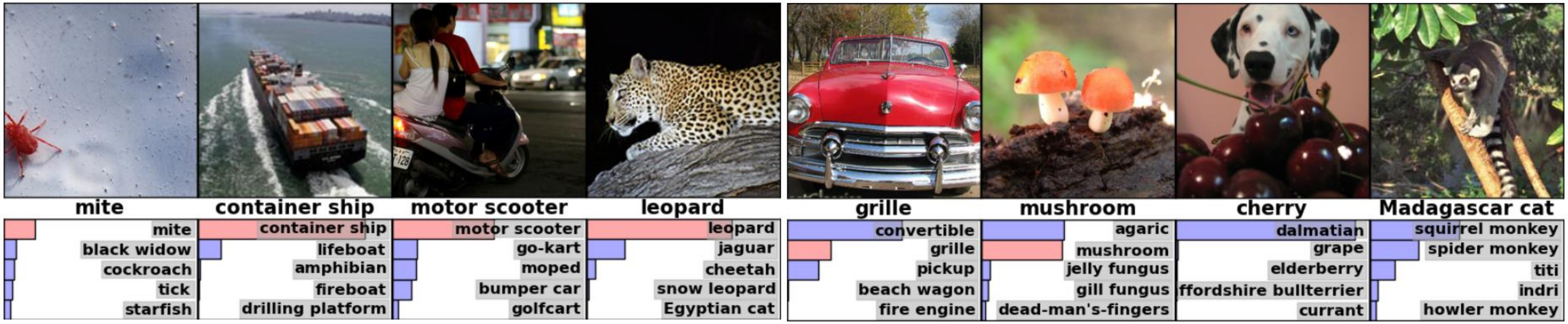
LeNet 5 [LeCun et al. 1998]



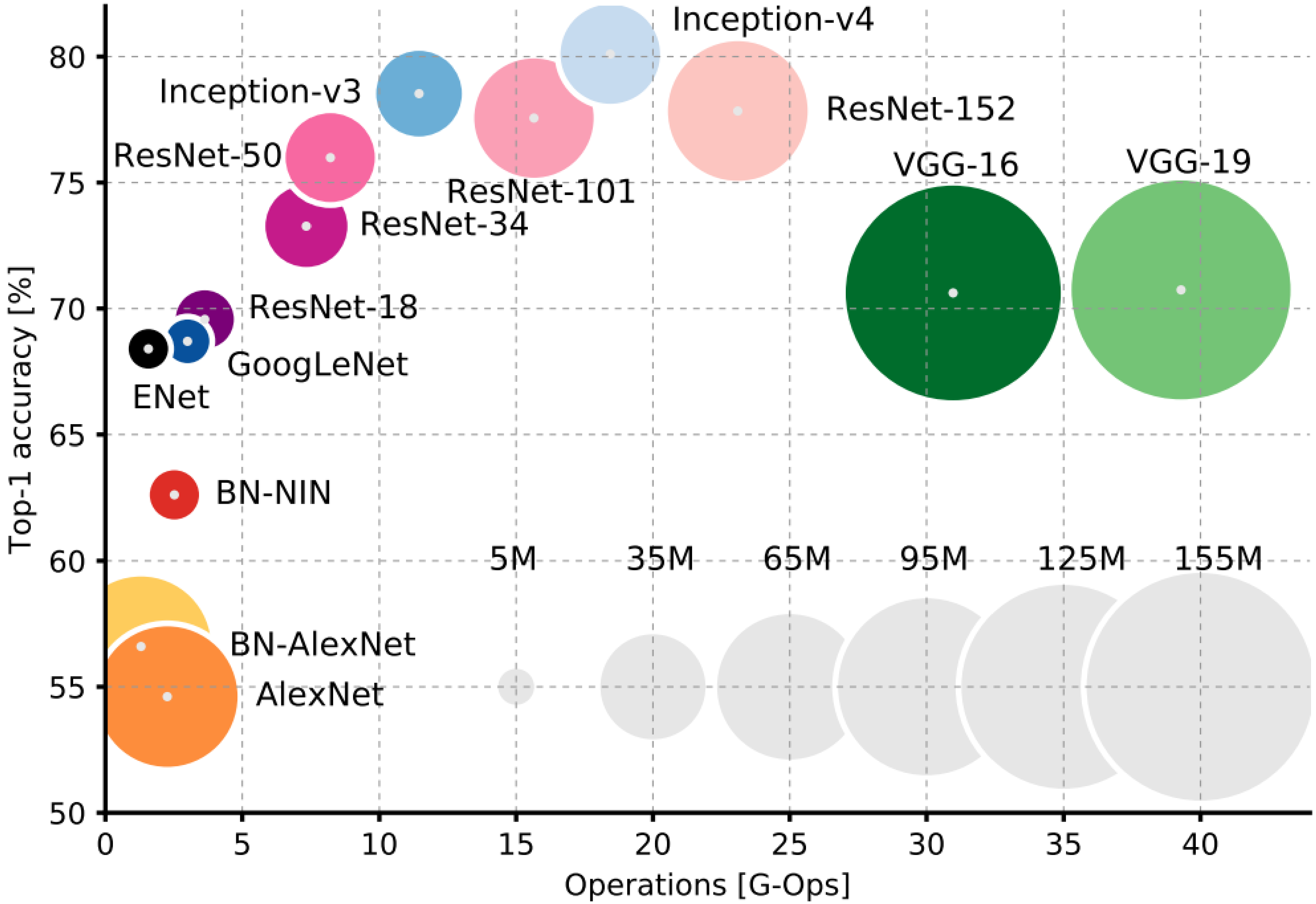
- Average pooling
- Sigmoid/tanh activation
- 60K training samples, 10 classes (MNIST)

Deep Neural Networks (simple)



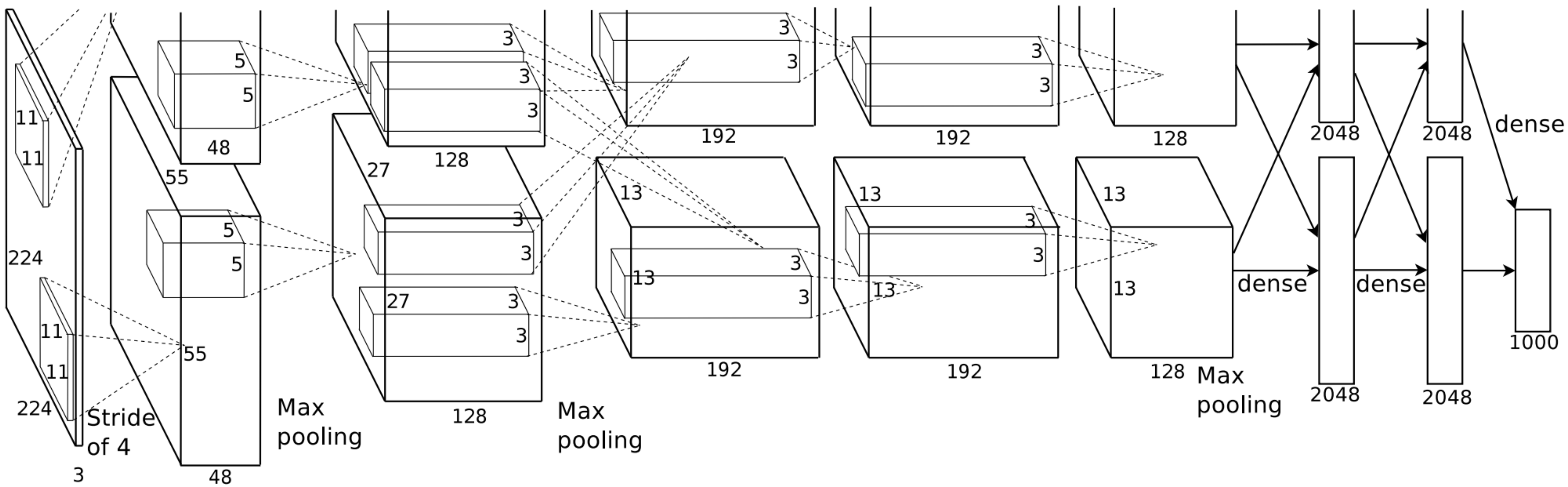


Which architecture?



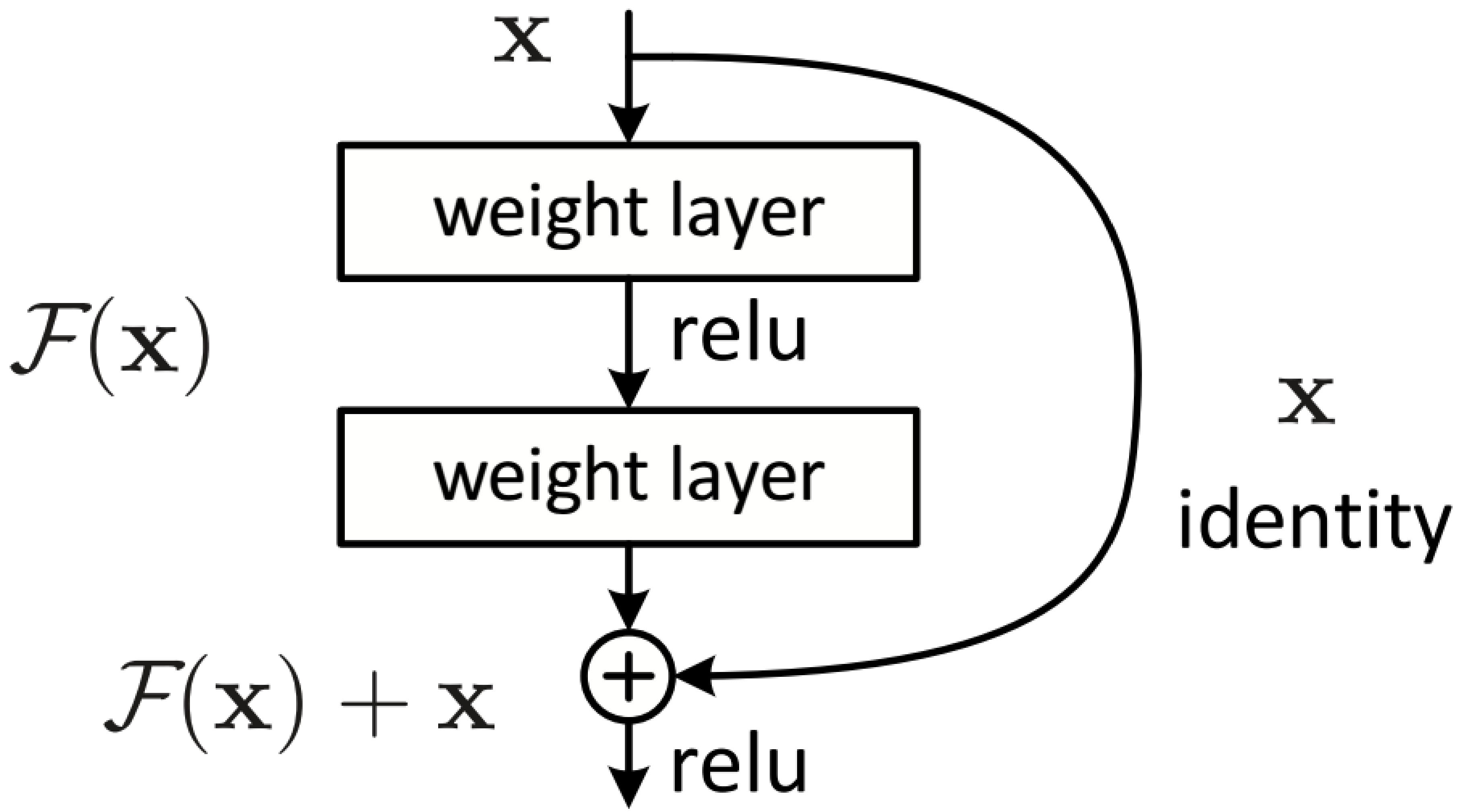
Canziani, Paszke, Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. arXiv:1605.07678

AlexNet [Krizhevsky et al. 2012]



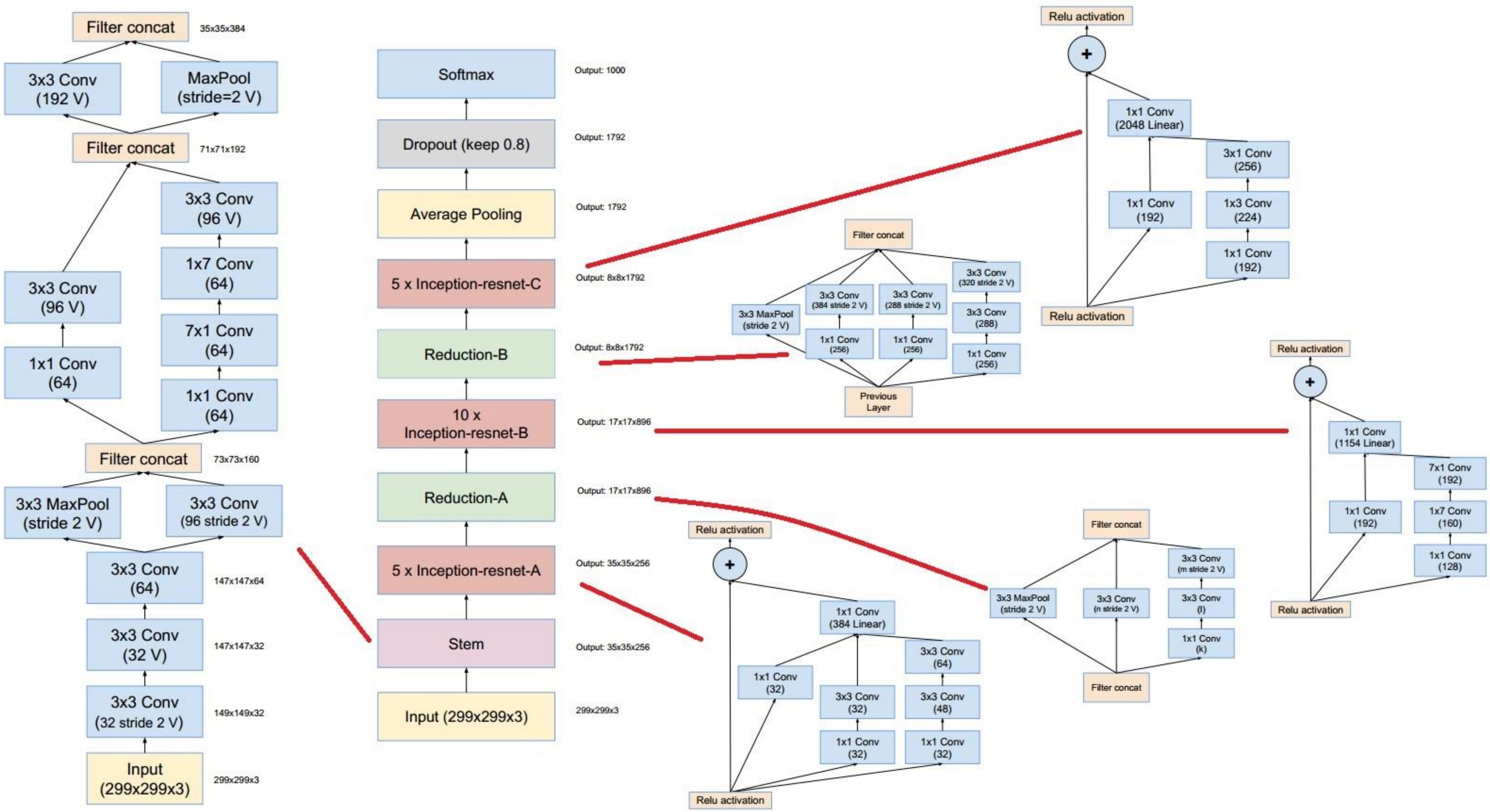
- Max pooling
- ReLU activation
- Dropout regularization
- 1.2M training samples, 1000 classes (ILSVRC)

Shortcuts / ResNet (152 layers!)



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

Inception v4



C. Szegedy et al., Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, arXiv 2016

Michael Felsberg
michael.felsberg@liu.se

LINKÖPINGS UNIVERSITET