Deep Neural Networks

Evgeny Burnaev

Skoltech, Moscow, Russia

Outline

1. Introduction
2. First generation of Neural Networks
3. Modern Neural Networks
4. Convolutional Neural Networks
5. Regularizing Neural Networks

1 Introduction

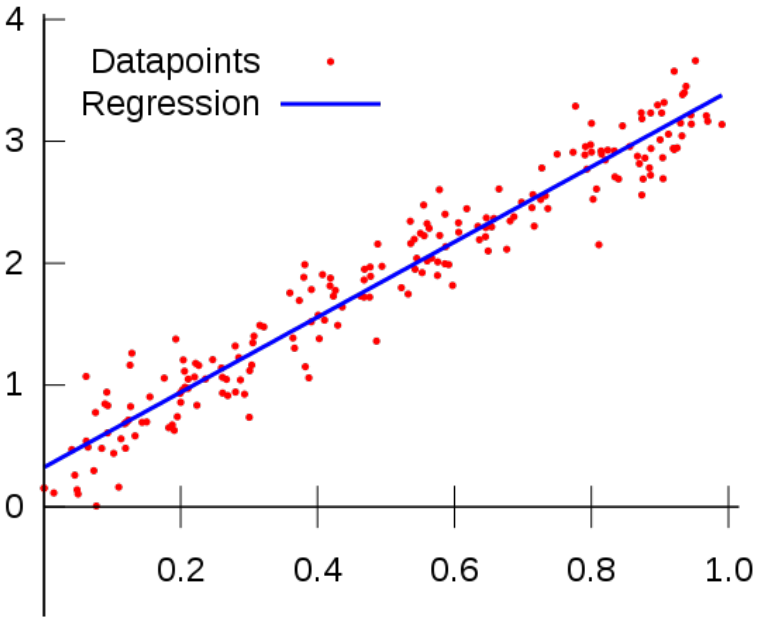
2 First generation of Neural Networks

3 Modern Neural Networks

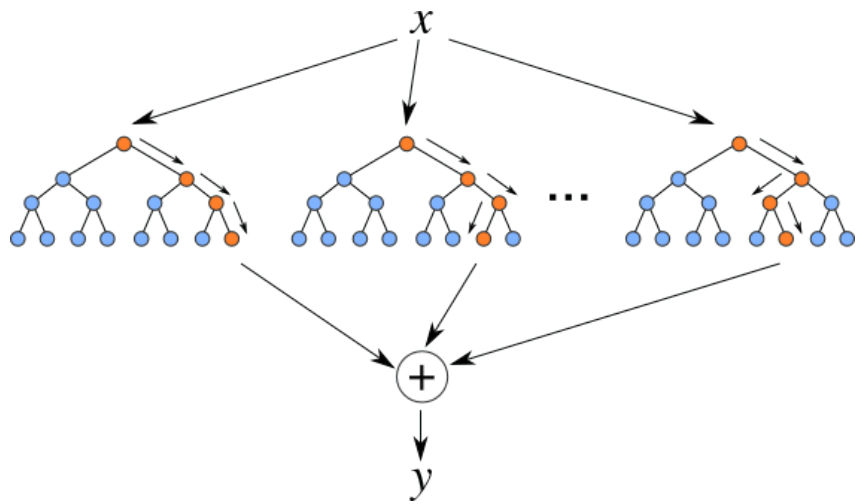
4 Convolutional Neural Networks

5 Regularizing Neural Networks

Linear models



Decision trees and ensembles

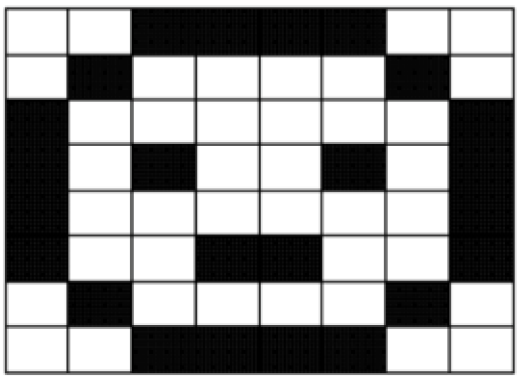


Machine Learning Problem

- U is a set of objects
- $X(U)$ is a set of features
- Y is a set of labels (classes, categories, etc.)
- For a new X we should define a corresponding class Y

Machine Learning Problem: Example

U $X(U)$ Y



Person

Bird

House

2006: Asirra competition

- 2006: accuracy 60%, $0.6^{12} = 0.00217$

2014: Asirra competition

Completed • Swag

Dogs vs. Cats

Wed 25 Sep 2013 – Sat 1 Feb 2014 (2 years ago)

Dashboard

Private Leaderboard - Dogs vs. Cats

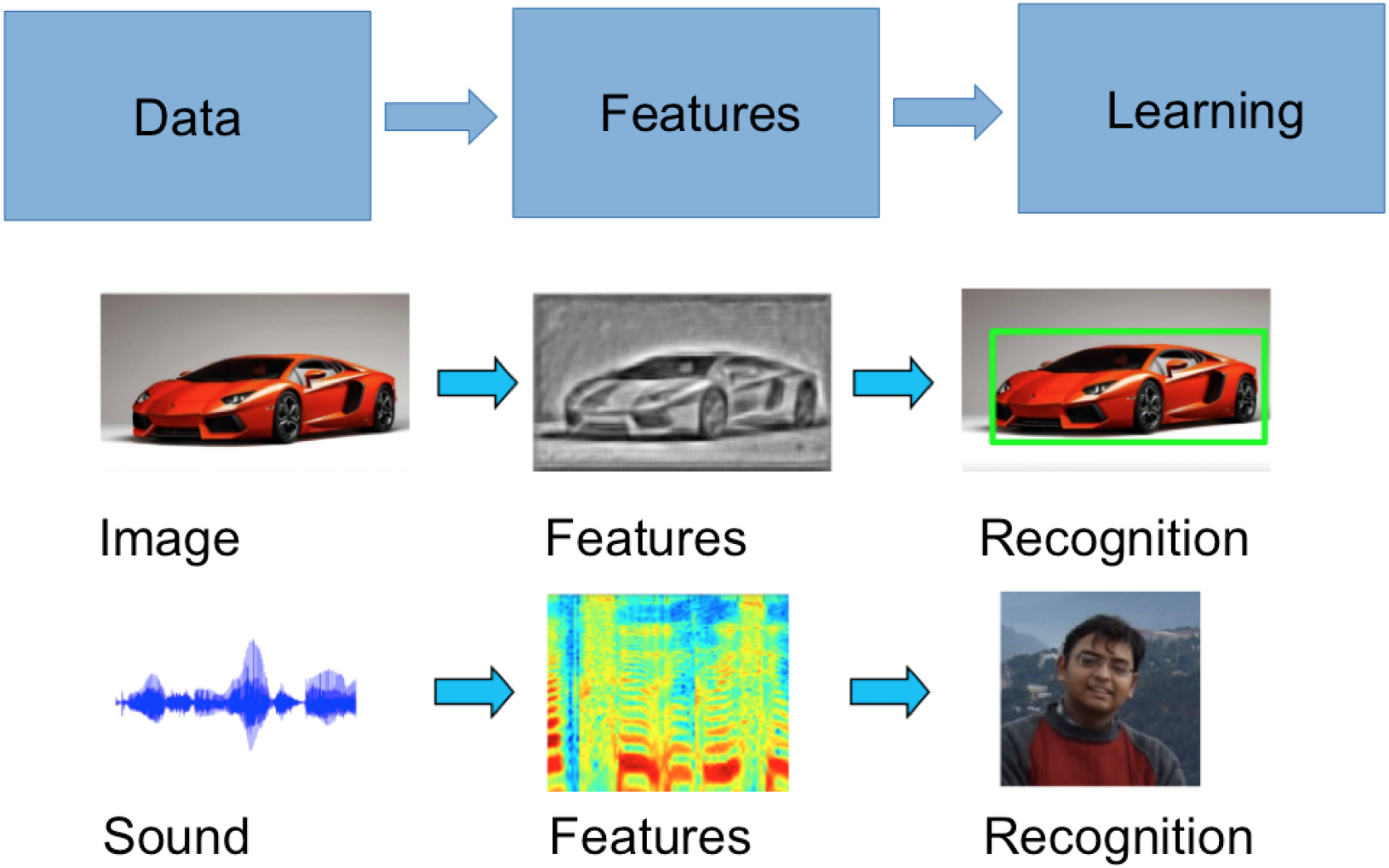
This competition has completed. This leaderboard reflects the final standings.

See someone using multiple accounts?
Let us know.

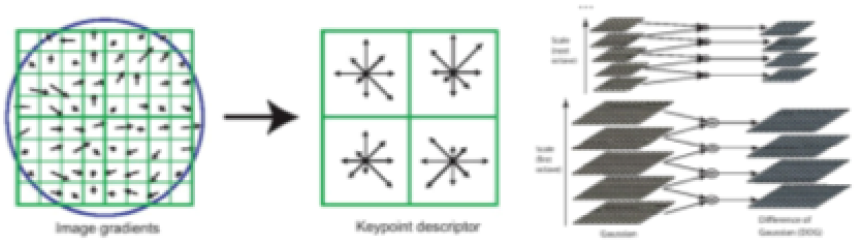
#	Δrank	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	—	Pierre Sermanet *	0.98914	5	Sat, 01 Feb 2014 21:43:19 (-1.2h)
2	↑4	orchid *	0.98309	17	Sat, 01 Feb 2014 23:52:30
3	—	Owen	0.98171	15	Sat, 01 Feb 2014 17:04:40 (-1.3h)
4	—	Paul Covington	0.98171	3	Sat, 01 Feb 2014 23:05:20
5	↓3	Maxim Milakov	0.98137	24	Sat, 01 Feb 2014 18:20:58
6	↓1	we've been in KAIST	0.98103	8	Sat, 01 Feb 2014 21:15:30

- $0.989^{12} = 0.875$

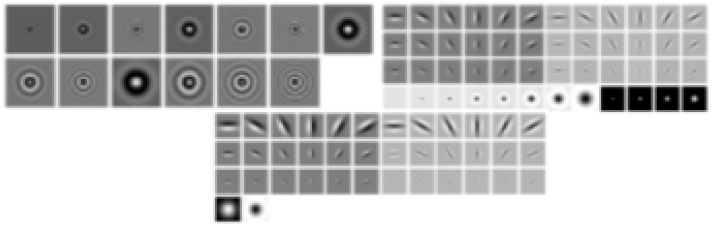
Old-school approach



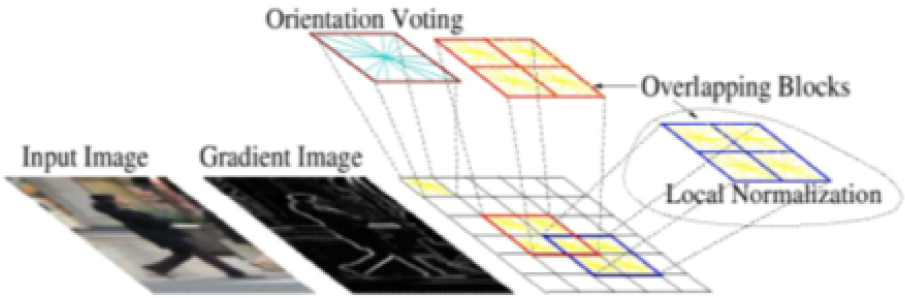
Secret magic: feature construction



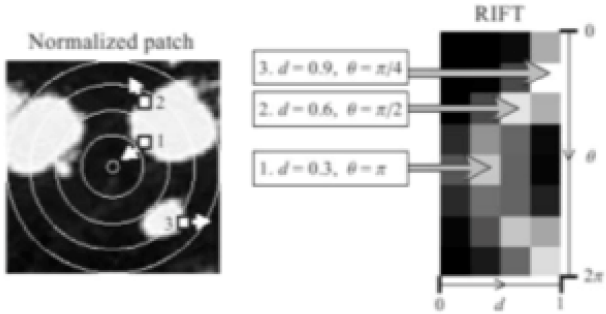
SIFT



Textons

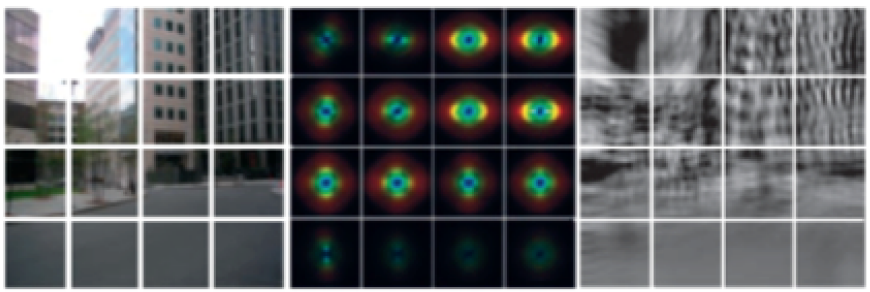


HoG

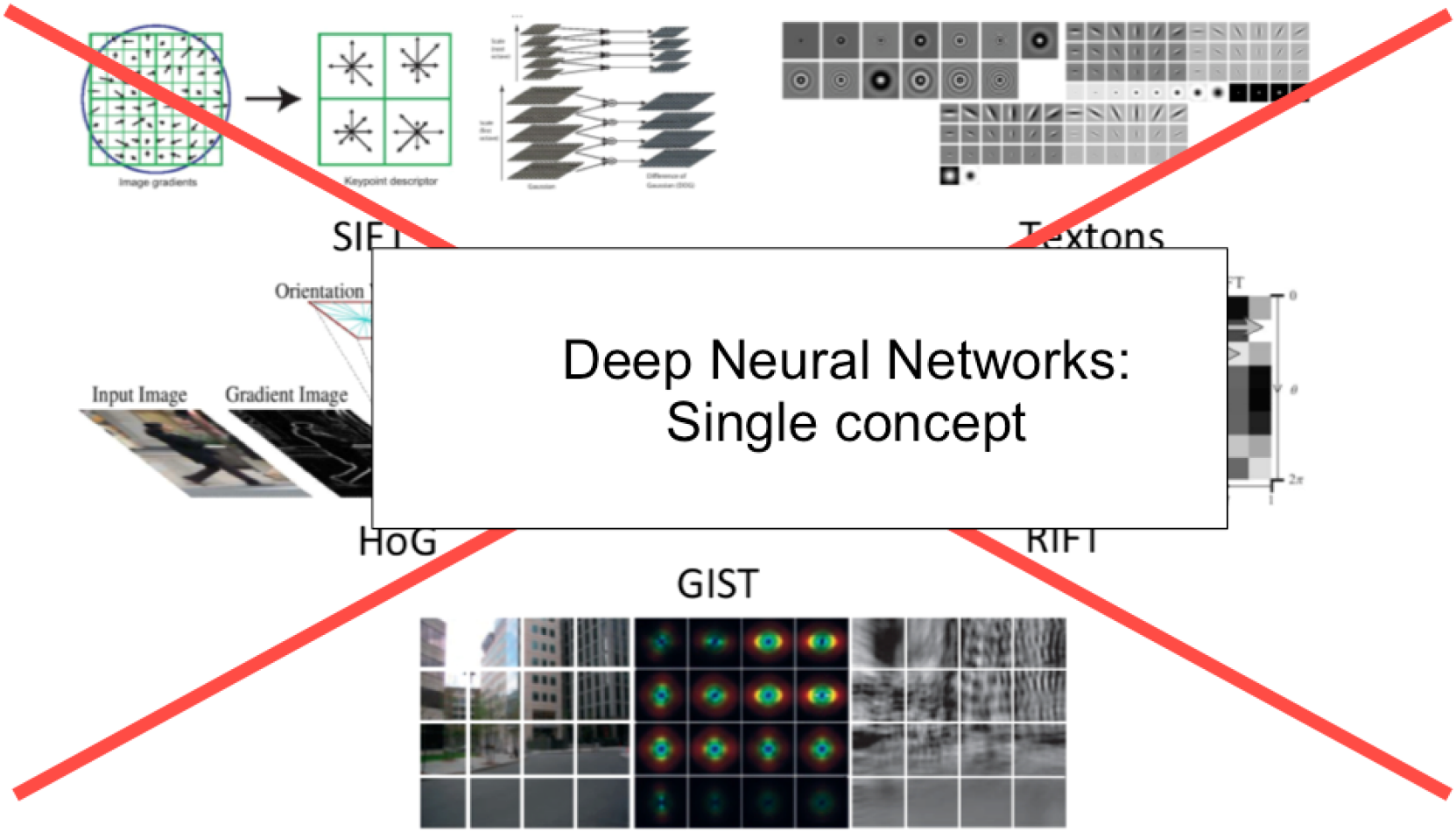


RIFT

GIST



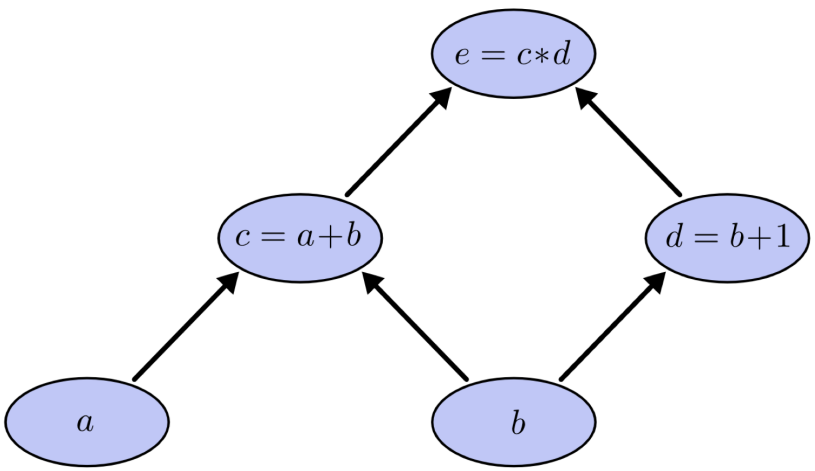
Secret magic: feature construction



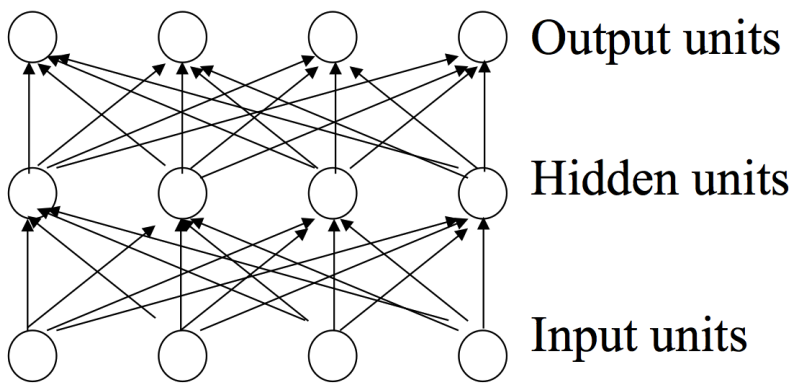
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Computational graph



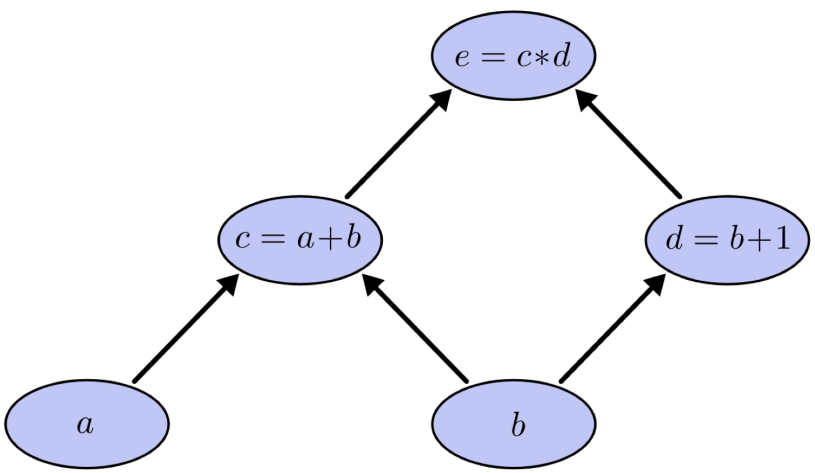
Fully-connected layered feed-forward network



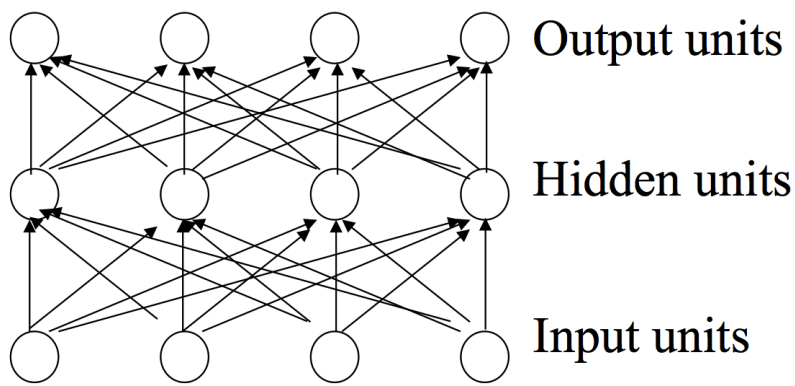
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Computational graph



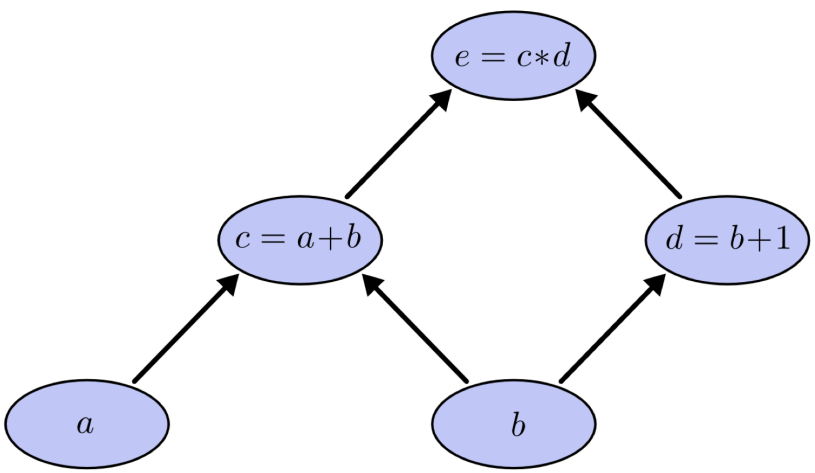
Fully-connected layered feed-forward network



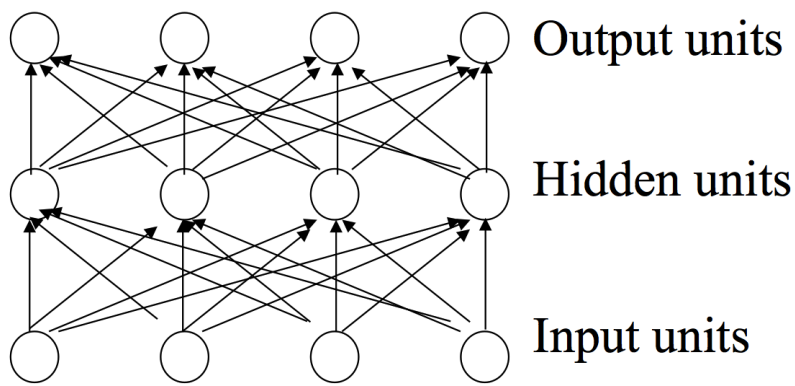
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Computational graph



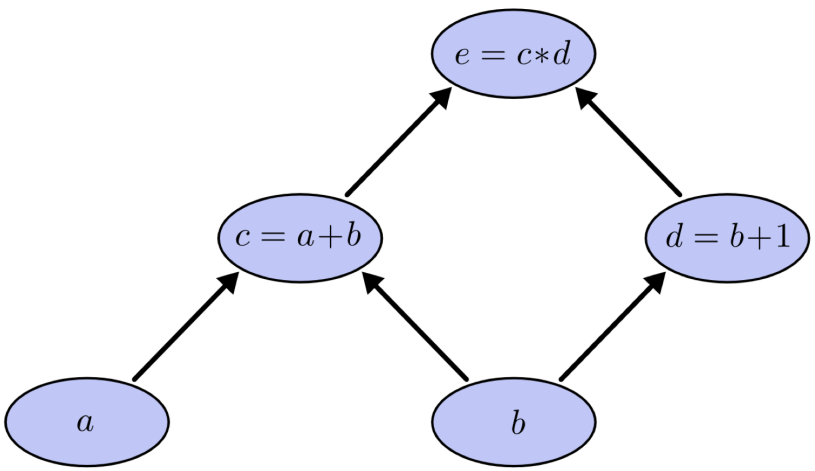
Fully-connected layered feed-forward network



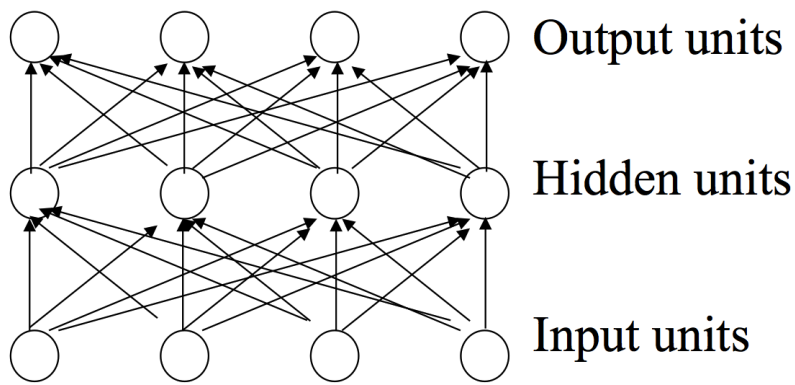
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Computational graph



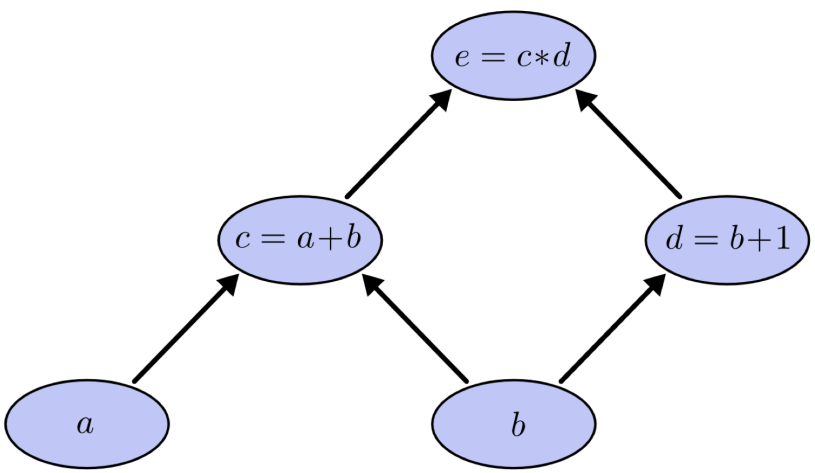
Fully-connected layered feed-forward network



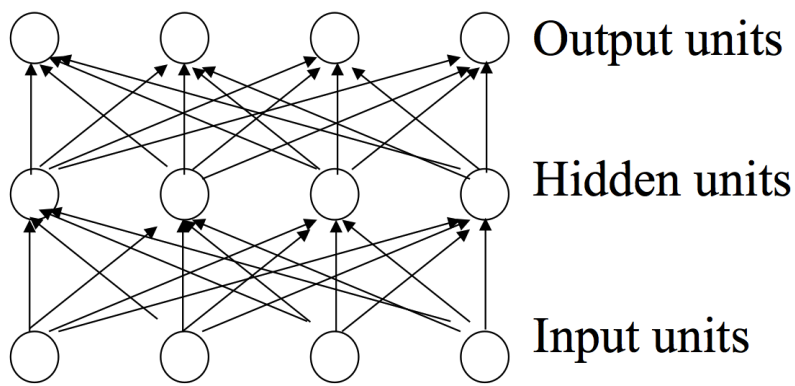
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Computational graph



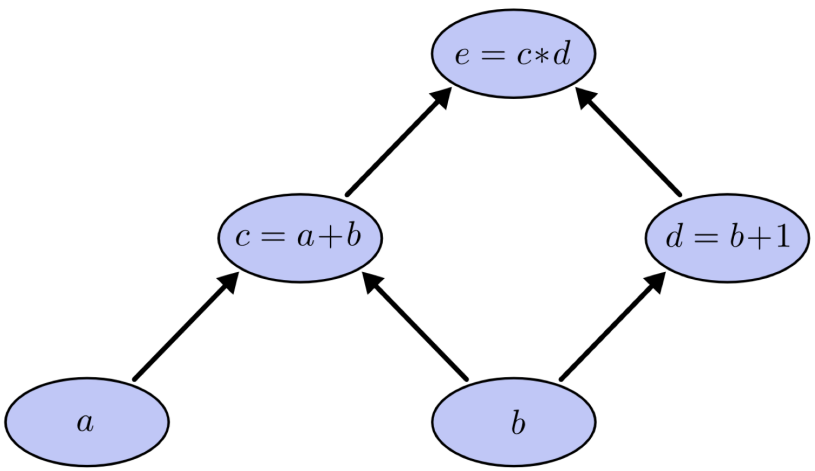
Fully-connected layered feed-forward network



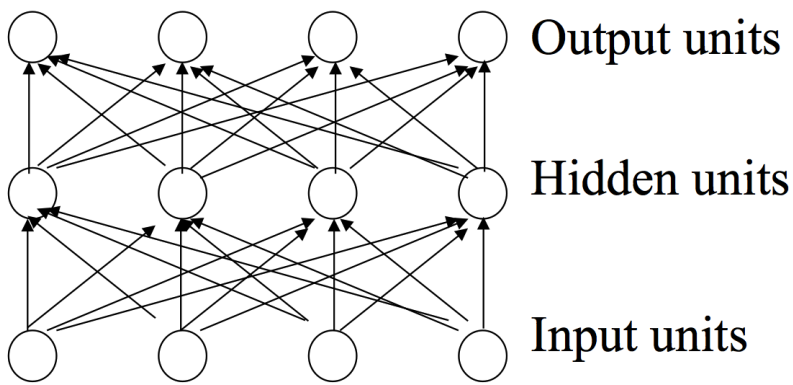
Neural Network

- Mathematically a neural network is a combination of simple transformations (linear/non-linear)
- Depth of an architecture
- Types of layers (convolutional, full-connected, etc.)
- Neural Networks are made up of **nodes** or **units**, connected by **links**, i.e. it is a kind of a computational graph
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

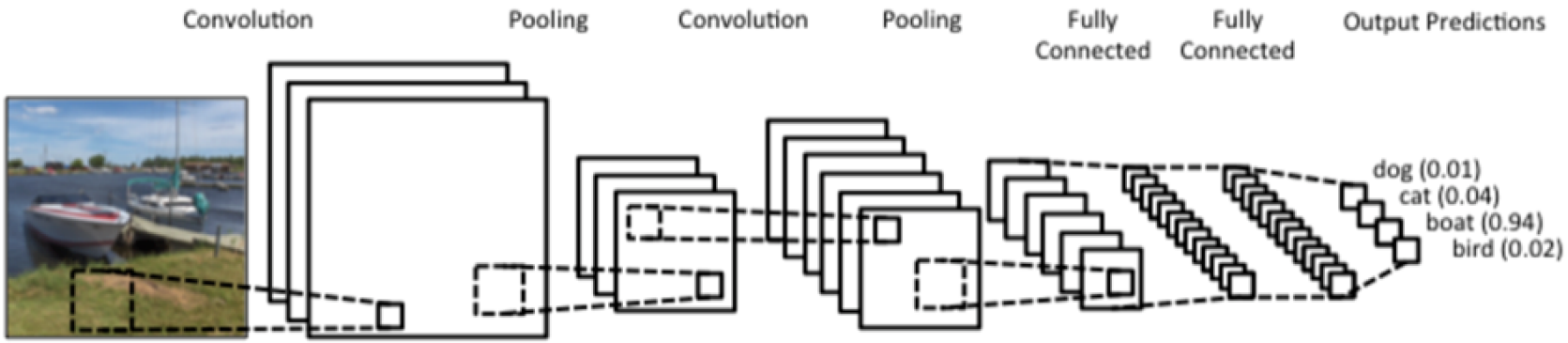
Computational graph



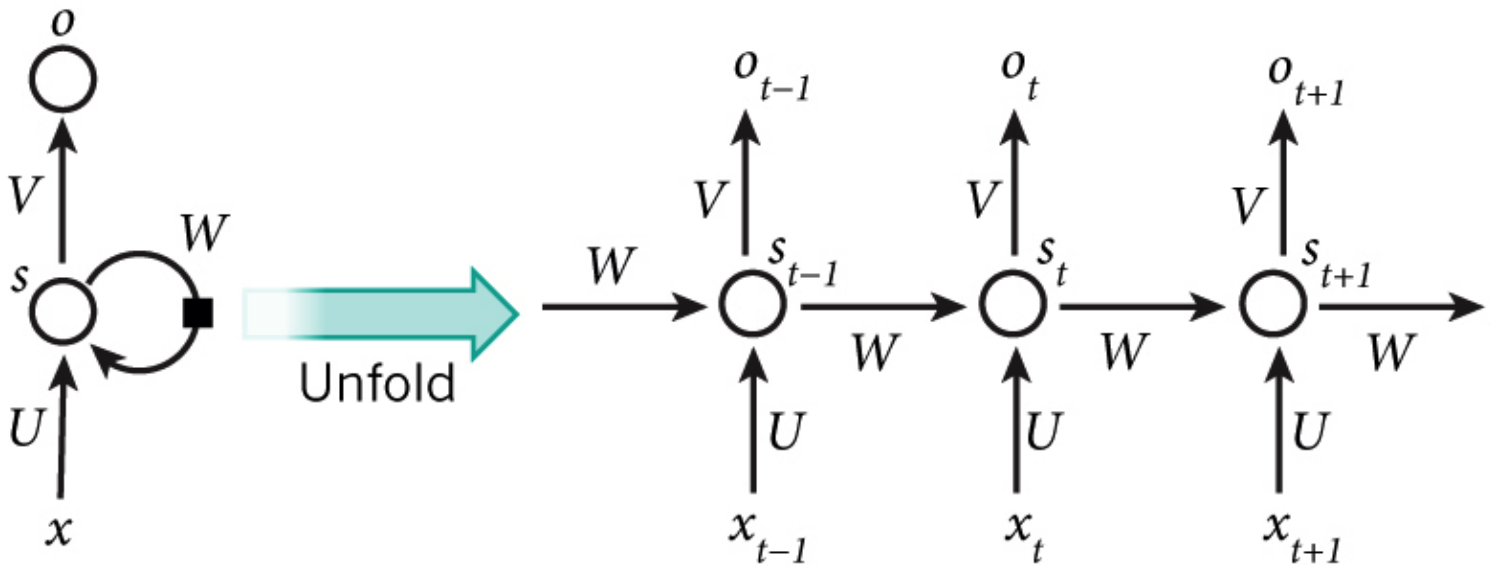
Fully-connected layered feed-forward network



Convolutional network

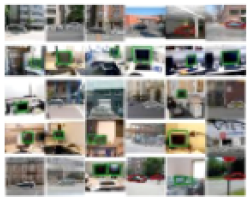


Recurrent neural network



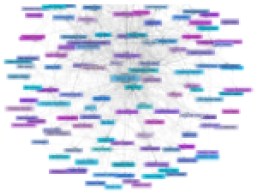
Main Application domains

Images, video

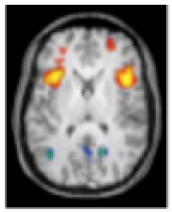


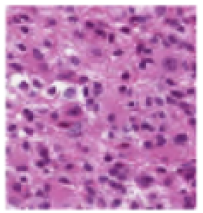
Text

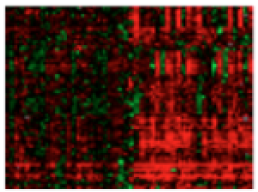
Social networks



Genomics, biology



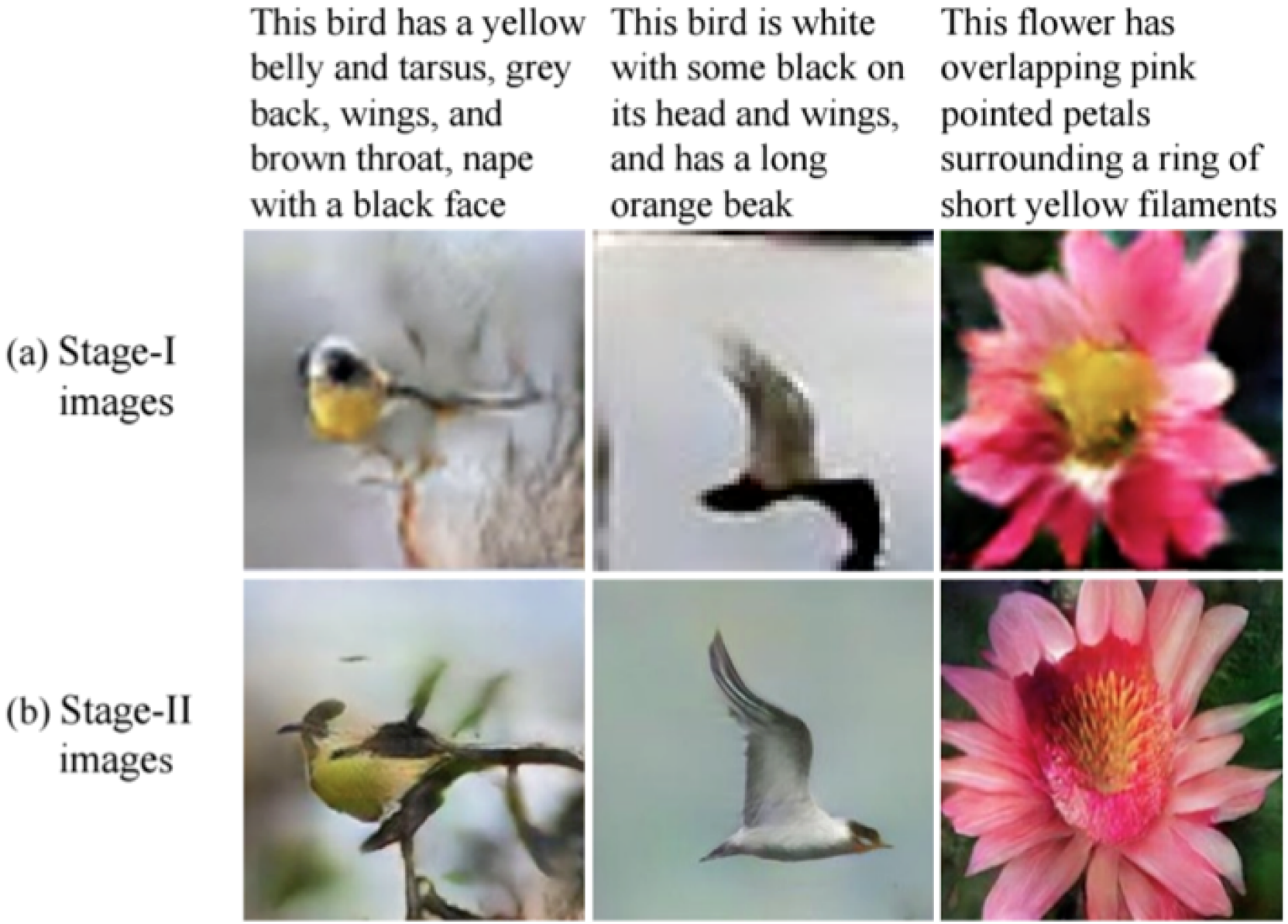




Recommendation systems

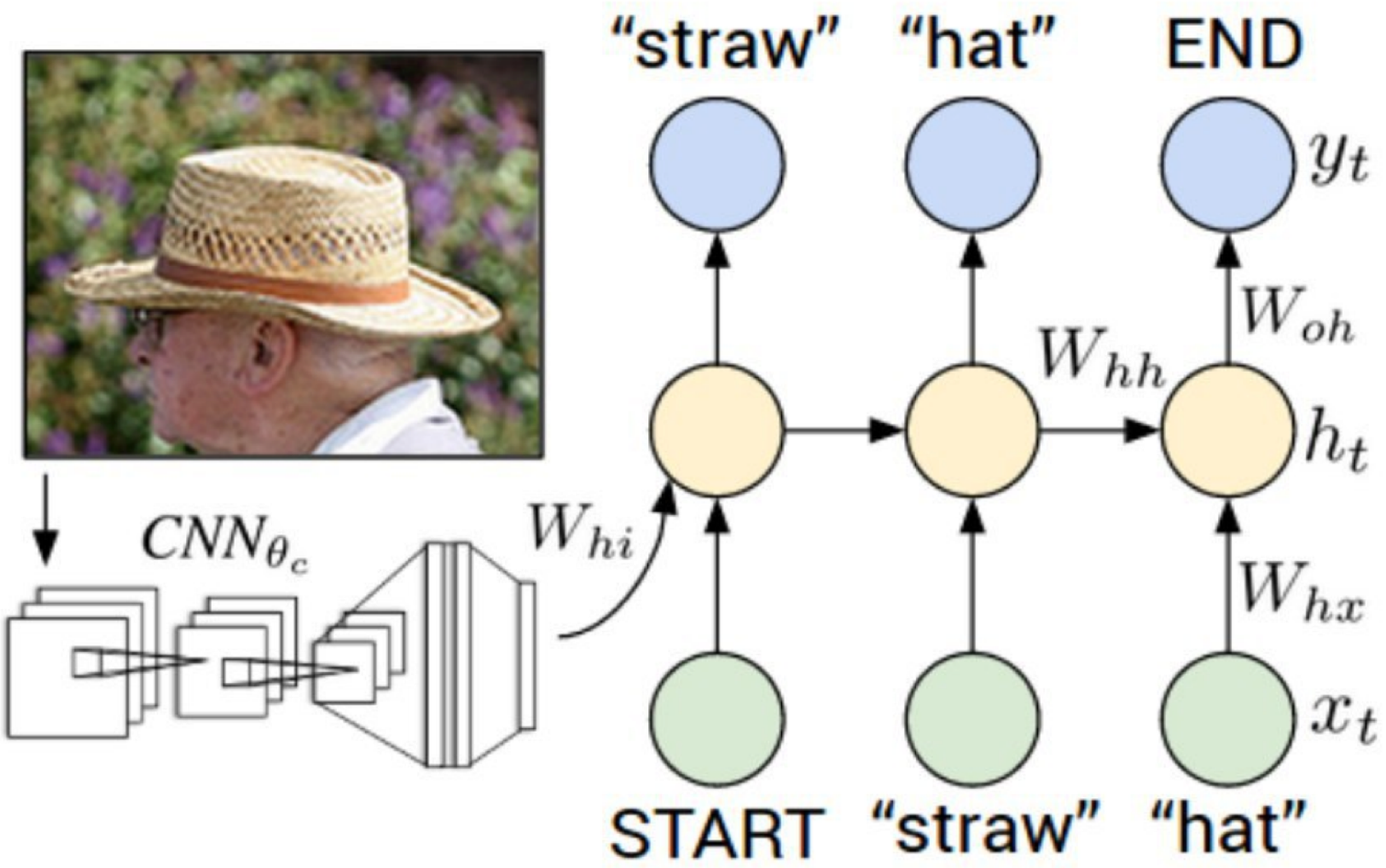
Example of application

- Image captioning



Example of application

- Image captioning



ImageNet

- Princeton, Stanford, Google
- More then 10 millions of images, manually labeled using Amazon Mechanical Turk

Simplicity of available tools

- E.g. Tensorflow is an open framework for deep neural networks (and not only!)
- More than 15 Google engineers develop it full-time
- ~ 700 thousands of code, ~ 600 participants of external open-source development
- Example: playground.tensorflow.org

Simplicity of available tools

- E.g. Tensorflow is an open framework for deep neural networks (and not only!)
- More than 15 Google engineers develop it full-time
- ~ 700 thousands of code, ~ 600 participants of external open-source development
- Example: playground.tensorflow.org

Simplicity of available tools

- E.g. Tensorflow is an open framework for deep neural networks (and not only!)
- More than 15 Google engineers develop it full-time
- ~ 700 thousands of code, ~ 600 participants of external open-source development
- Example: playground.tensorflow.org

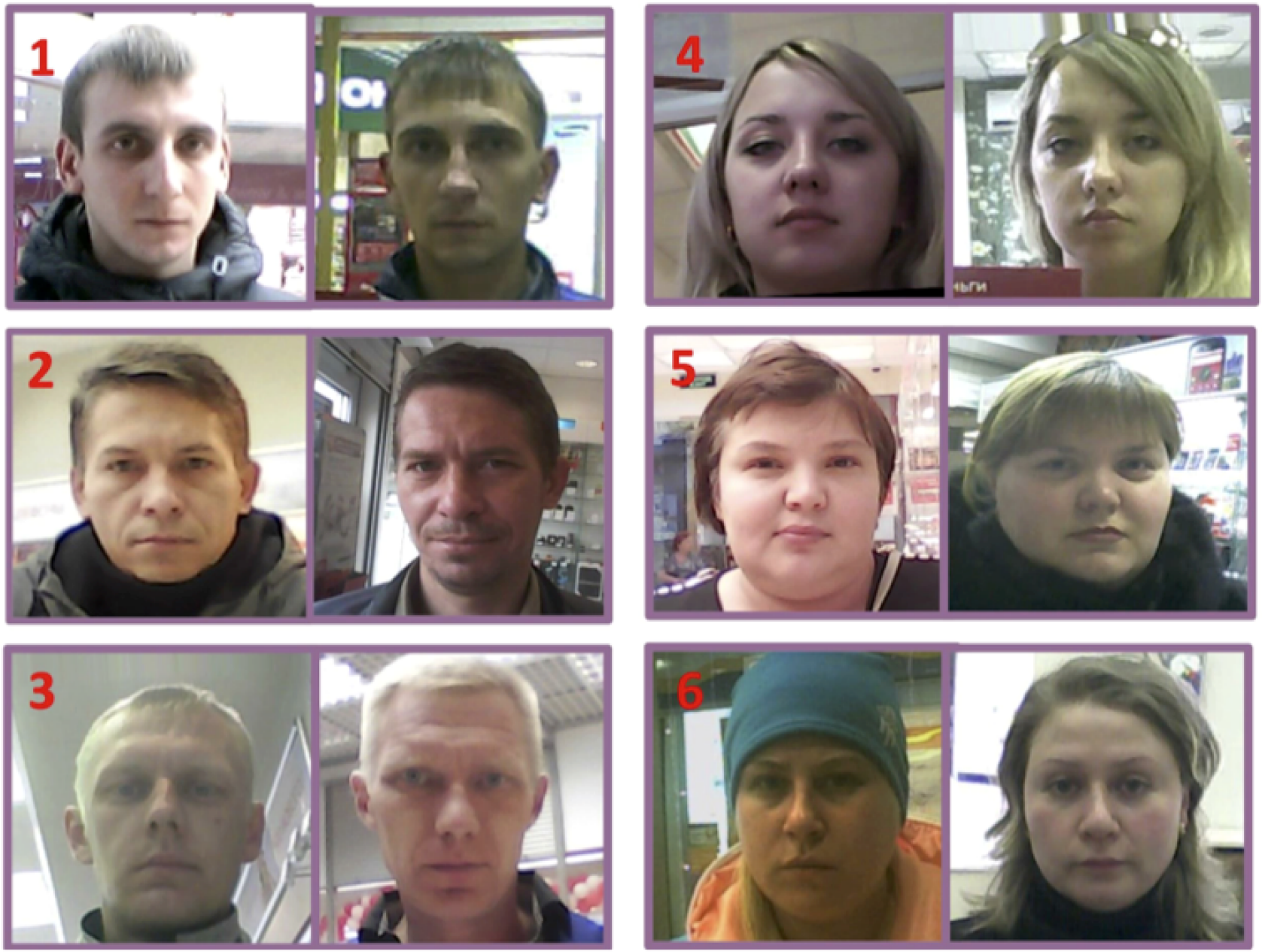
Simplicity of available tools

- E.g. Tensorflow is an open framework for deep neural networks (and not only!)
- More than 15 Google engineers develop it full-time
- ~ 700 thousands of code, ~ 600 participants of external open-source development
- Example: playground.tensorflow.org

Applications: VisionLabs (Russia)

- Face recognition and identification from camera images
- Several-fold advantage from using Deep Learning (requirement to the number of "bad" loans — not more then 1 for 10 millions)
- Integration of the system in top Russian banks

Are persons different? (from Khanin's presentation)

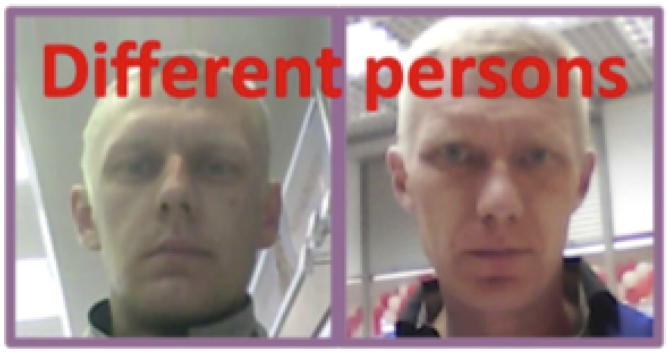


Answer

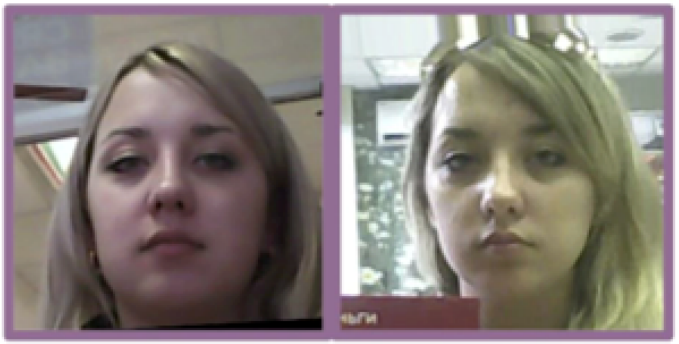
1

2

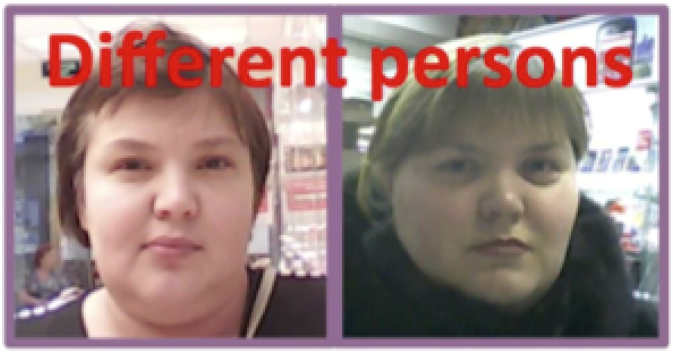
3



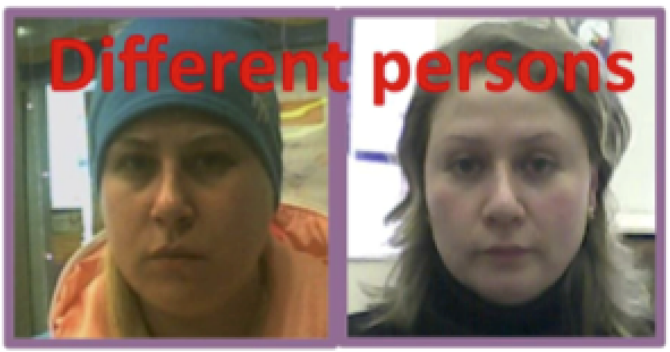
4



5



6



Vocord (Russia)

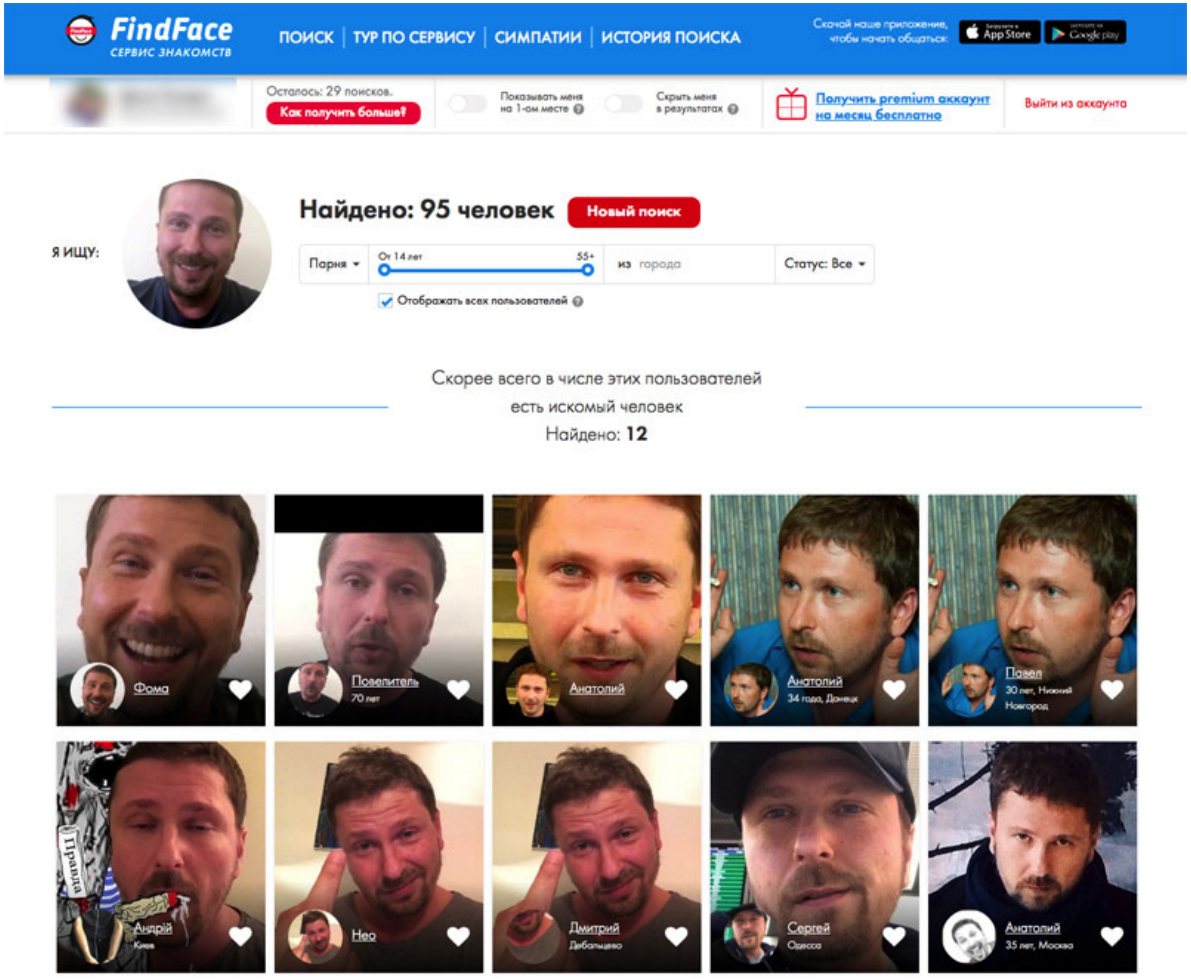
- www.vocord.ru
- Recognition of car number plates
- Record accuracy on large datasets

Atomwise (USA)

- Accelerates the search of new drugs
- Already discovered a new drug for multiple sclerosis and anti-virus tool aimed at slowing the spread of Ebola haemorrhagic fever

FindFace (Russia)

- Search by photos in social networks

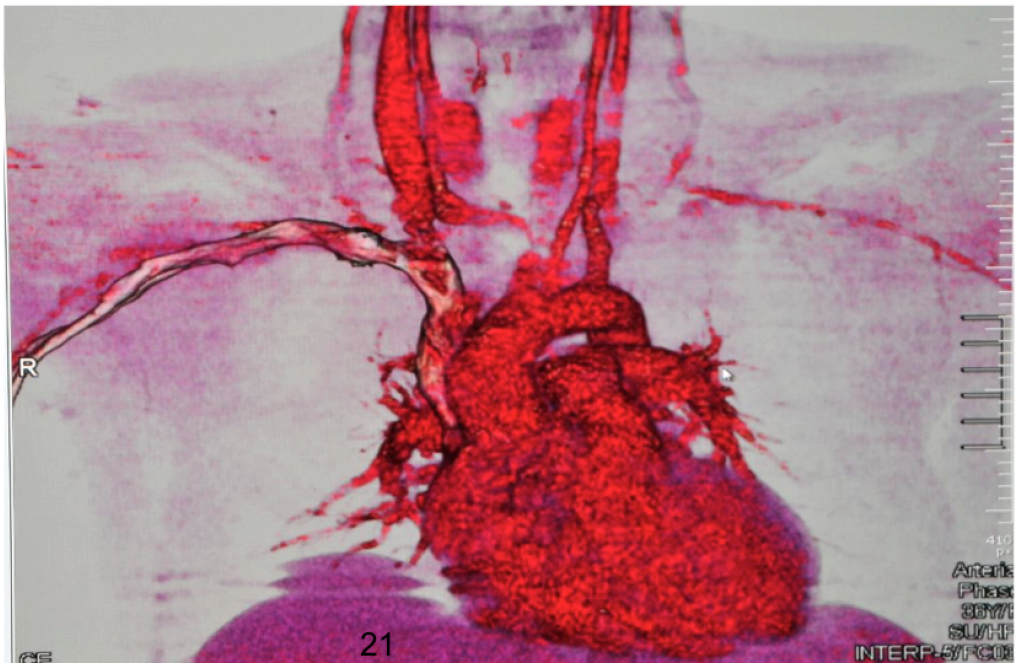


Prisma (Russia)

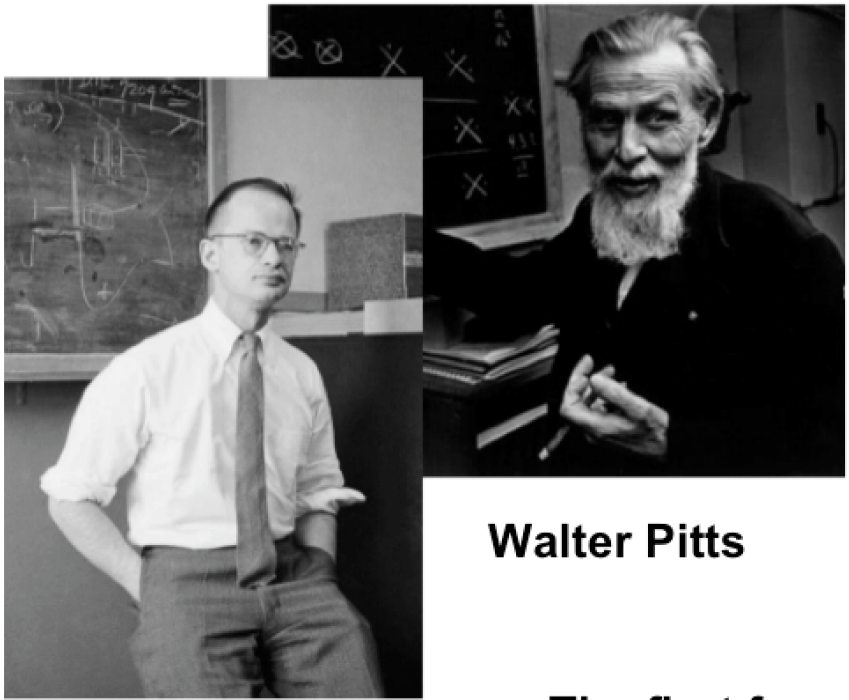
- Styling images

Arterys (USA)

- Startap Arterys, AI based on DL in the cloud
- The first FDA approval for the diagnosis of heart disease
- Accurate measurements of the volume of each ventricle allowing more precise assessment of health
- Precise quantification of blood flow through the heart Arterys takes an average of 15 seconds to produce a result for one case, which a professional human analyst would expect to spend between 30 minutes to an hour working on



Historical Background



Walter Pitts

Warren McCulloch

The first formal neuron - 1943

Frank Rosenblatt

The first artificial neural network - 1958

- What did cause the breakthrough?

Causes of the breakthrough

The reasons for the breakthrough in the combination of factors!

- The increase in power and availability of hadrware (GPU)
- Big volumes of open, labeled data
- Available frameworks: Tensorflow (Google), Torch (Facebook, Google), Theano (open-source), Caffee (Berkley), Veles (Samsung)
- The openness of research and of source code, realizing the most widely used frameworks
- Significant investments from big IT companies (Google, Facebook, Amazon, IBM, etc.)

As a result, the point "focusing" of research on a "limited" topic (> 50 research groups, > 1000 researchers)

1 Introduction

2 First generation of Neural Networks

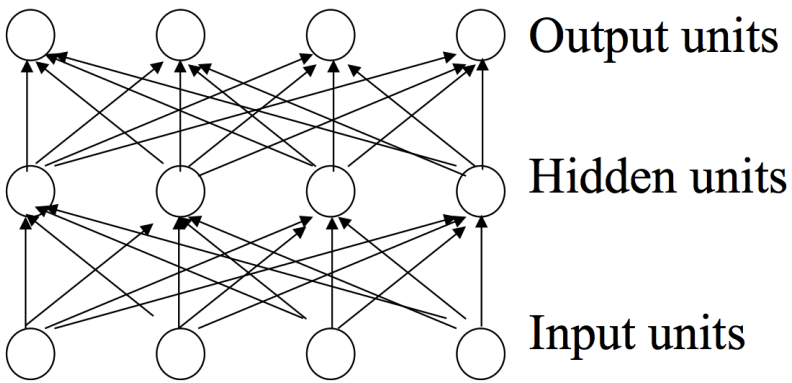
3 Modern Neural Networks

4 Convolutional Neural Networks

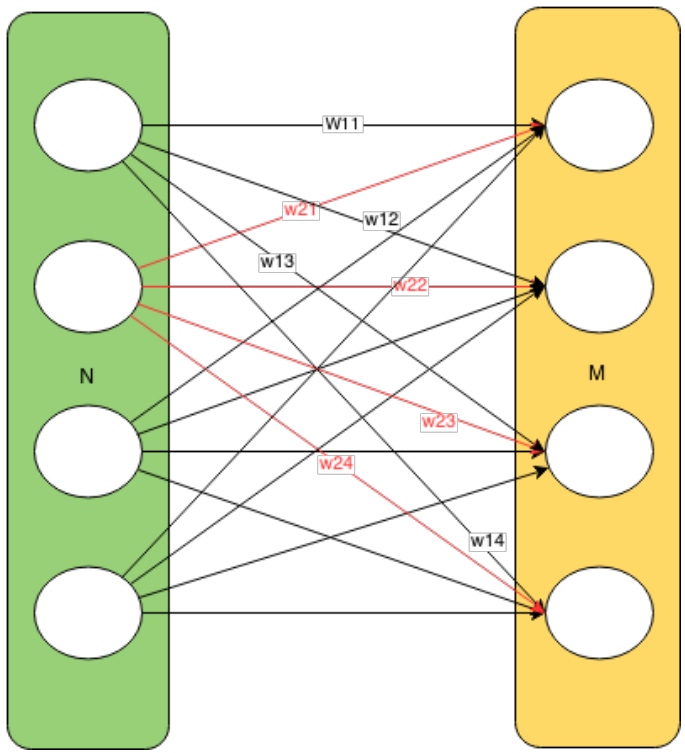
5 Regularizing Neural Networks

- Initially the most popular networks were fully-connected layered feed-forward networks

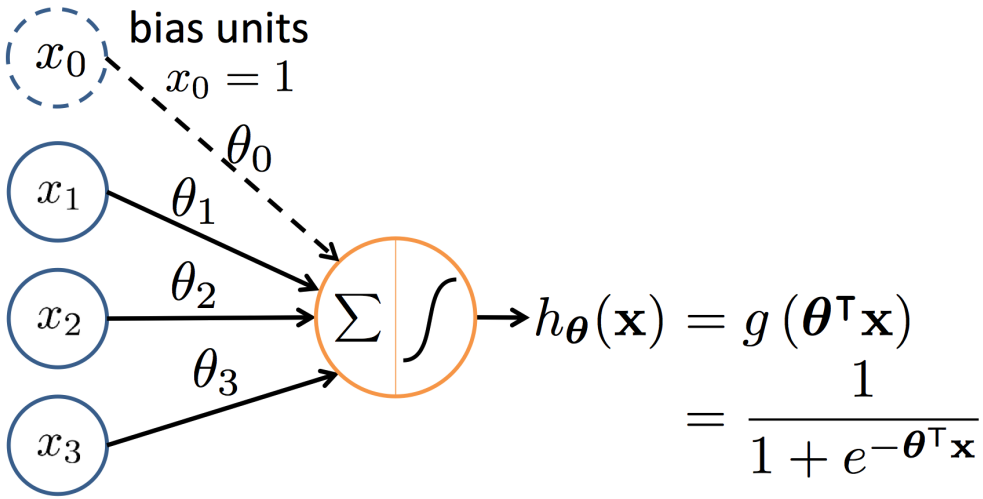
Fully-connected layered feed-forward network



Fully connected layer

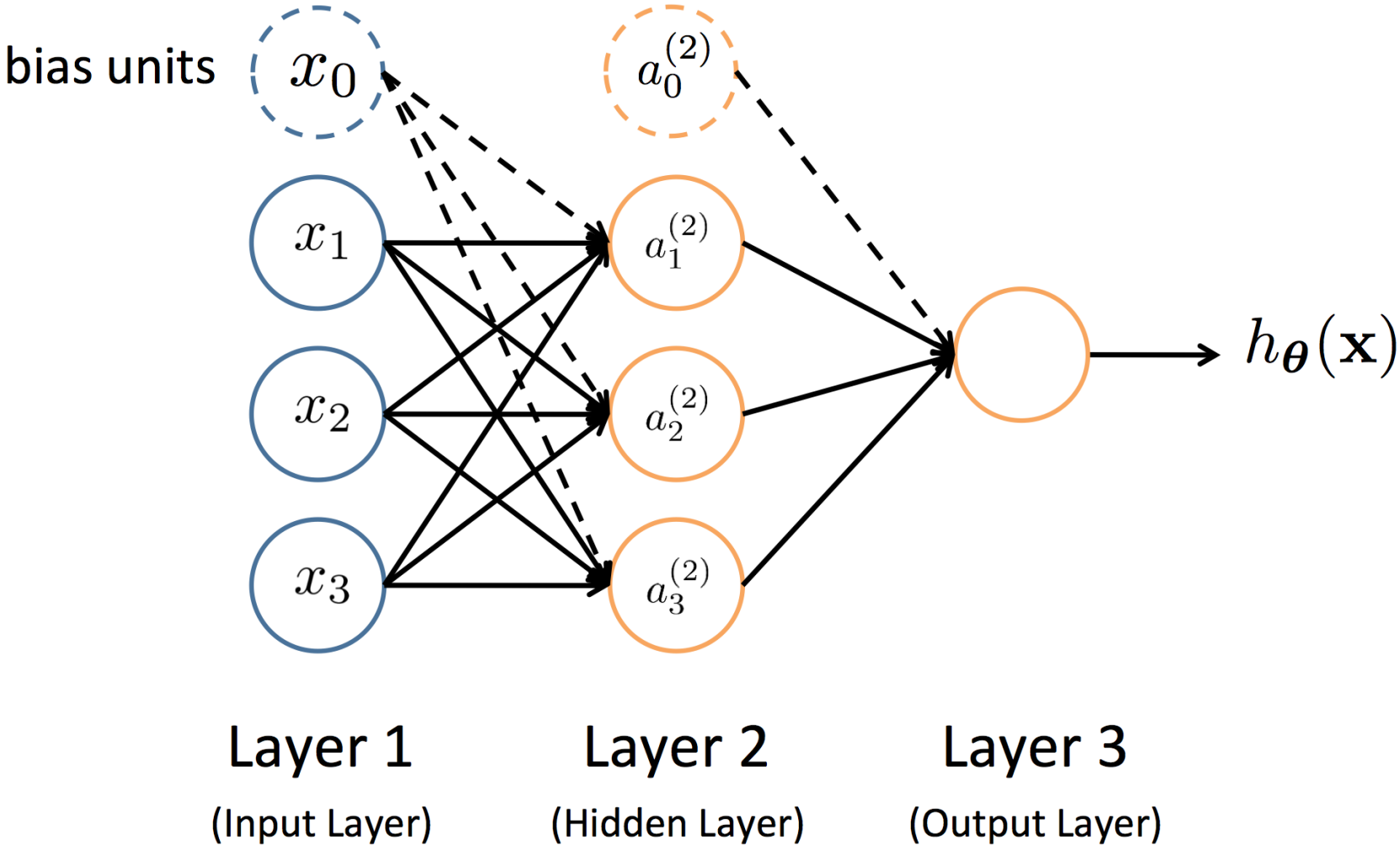


- Let us define $\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$



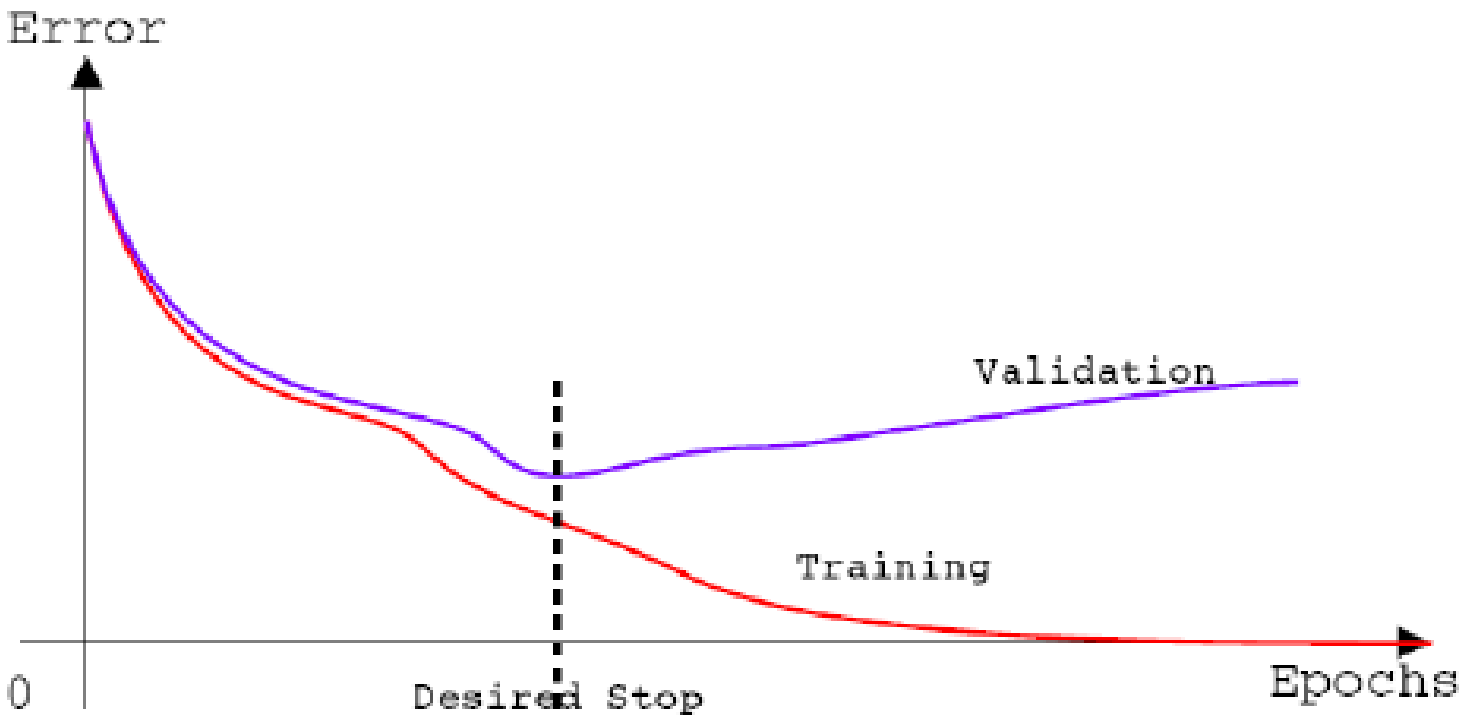
- Sigmoid (logistic) activation function $g(z) = \frac{1}{1+e^{-z}}$

Multi-Layered Neural Network



Applied Issues

- Backpropagation for training: gradient descent + efficient computations (based on graph-like representation)
- Local minima
- Random initialization
- Random division into train and validation sets
- Second order optimization methods
- Early stopping



1 Introduction

2 First generation of Neural Networks

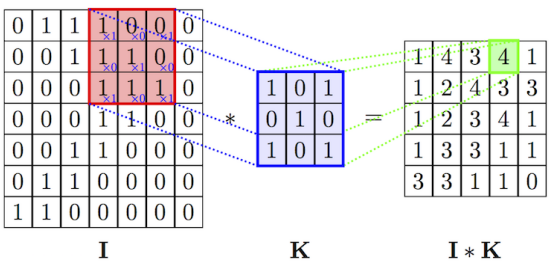
3 Modern Neural Networks

4 Convolutional Neural Networks

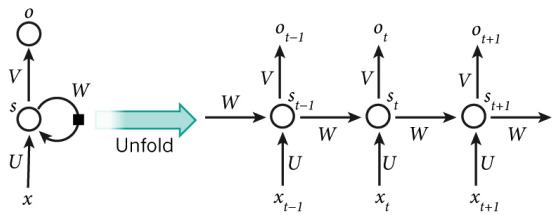
5 Regularizing Neural Networks

Different architectures

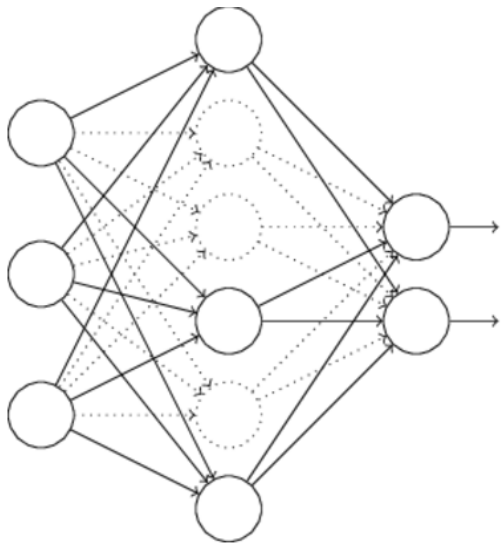
Convolutional layer



Recurrent layer



Dropout layer



Computation graphs

- State-of-the-art quality for many computer vision and NLP problems
- Sometimes even superhuman performance!
- Key components are:
  - Computational power and GPUs
  - Lots of data (or transfer learning)
  - Stochastic optimization
  - Regularization techniques

Let $\phi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous function on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer p, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $\mathbf{w}_i \in \mathbb{R}^m$, where $i = 1, \ldots, p$, such that we may define:

$$F(\mathbf{x}) = \sum_{i=1}^{p} v_i \phi(\mathbf{w}_i^{\mathrm{T}} \mathbf{x} + b_i)$$

as an approximate realization of the function f, where f is independent of ϕ; that is,

$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

for all $\mathbf{x} \in I_m$. In other words, functions of the form $F(\mathbf{x})$ are dense in $C(I_m)$.

Deep architectures are better

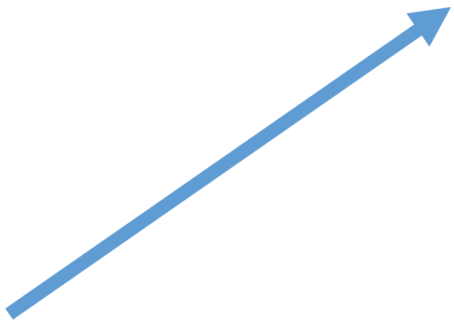
- Deep nets can be trained and regularized effectively
- Deep nets can achieve better performance with same number of parameters
- Deep architectures are more intuitive and can be interpreted as a sequence of simple data transformations

Training computational graph

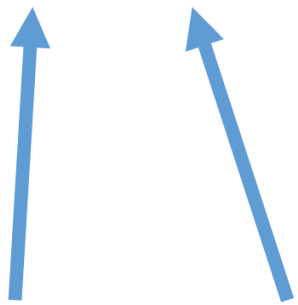
- $f_j(\mathbf{x}, \mathbf{w}_j)$ is a j-th layer of the network
- $v_j(\mathbf{x}) = f_j(v_{j-1}(\mathbf{x}), \mathbf{w}_j)$
- $a(\mathbf{x}, \mathbf{w}) = v_n(\mathbf{x})$ is an output
- Optimization problem

$$Q(\mathbf{w}) = \sum_{i=1}^{l} L(y_i, a(\mathbf{x}_i, \mathbf{w})) \to \min_{\mathbf{w}}$$

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial f_i} \frac{\partial f_i}{\partial w_i}$$



Comes from previous steps

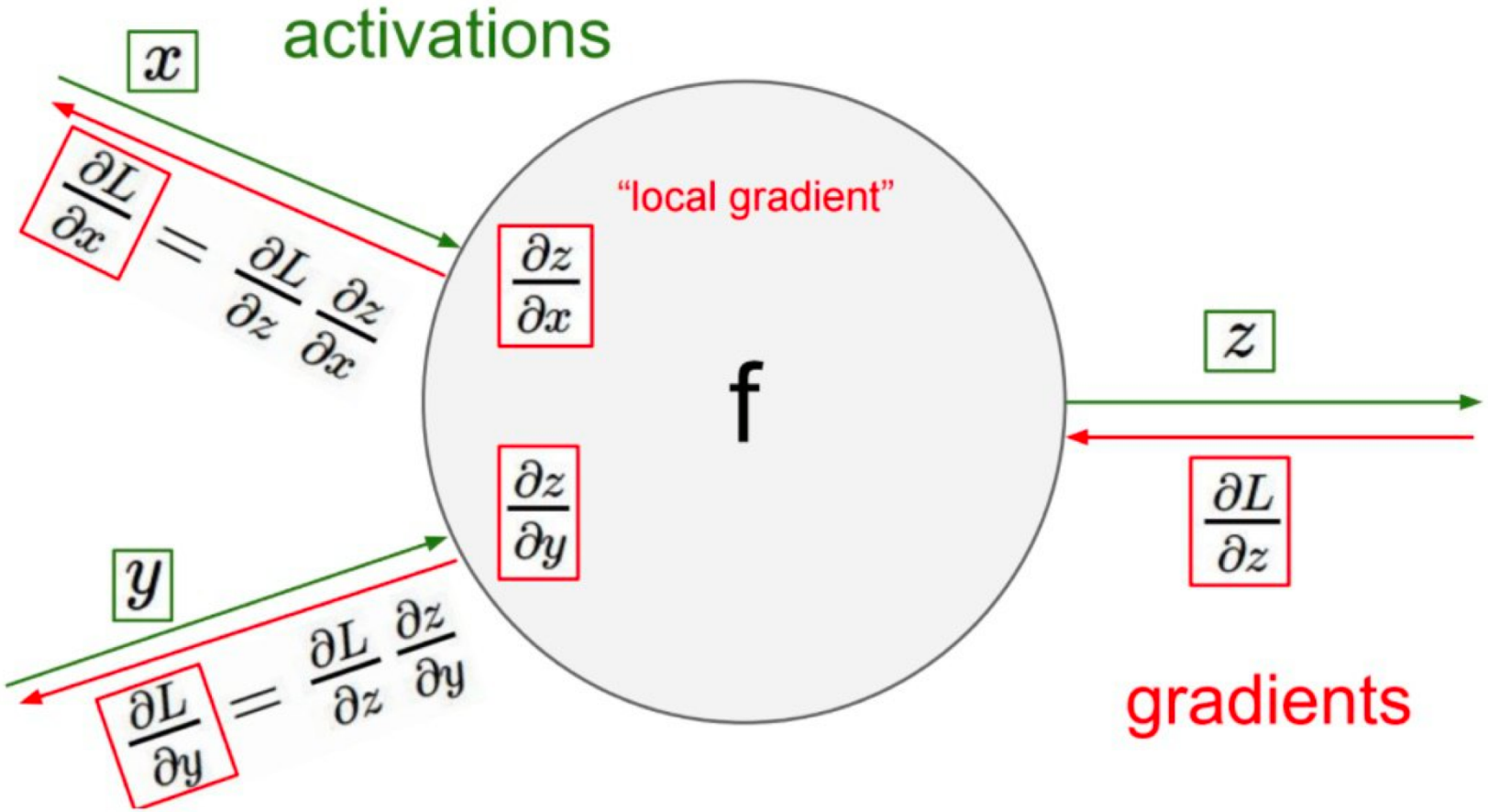


Easy for differentiable layers

$$\frac{\partial Q}{\partial f_{ij}} = \sum_k \frac{\partial Q}{\partial f_{i+1,k}} \frac{\partial f_{i+1,k}}{\partial f_{ij}}$$

$$\frac{\partial Q}{\partial w_{ij}} = \sum_k \frac{\partial Q}{\partial f_{ik}} \frac{\partial f_{ik}}{\partial w_{ij}}$$

Backpropagation



1 Introduction

2 First generation of Neural Networks

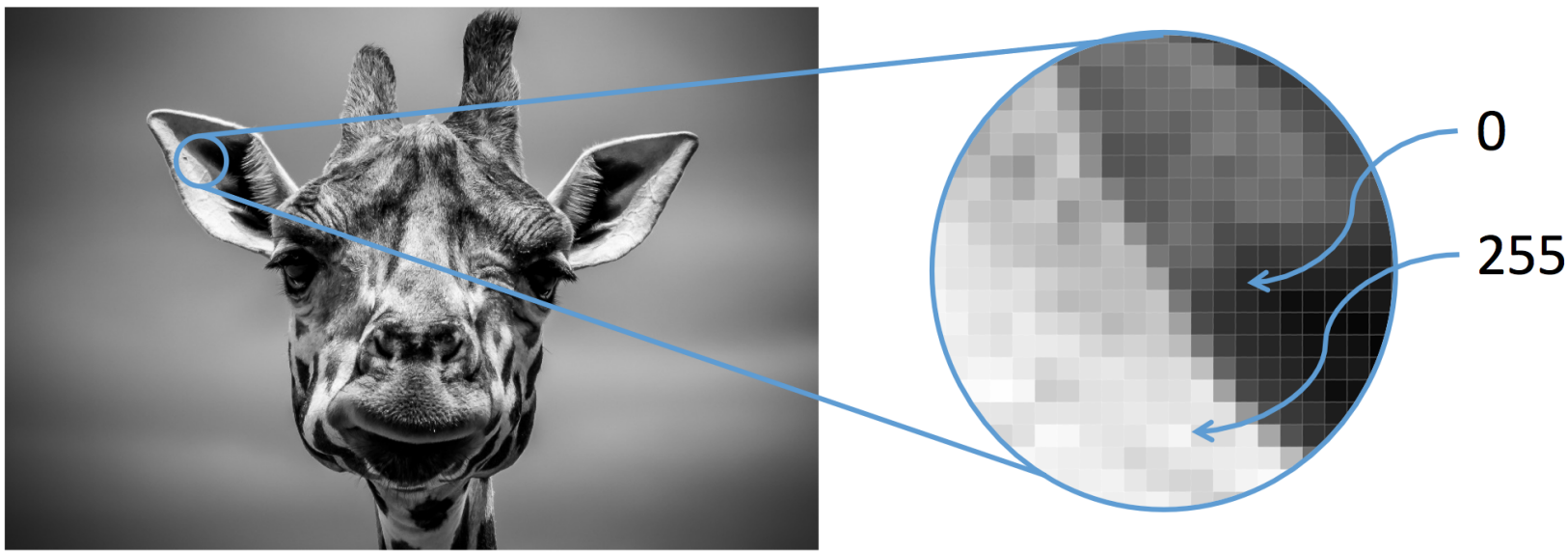
3 Modern Neural Networks

4 Convolutional Neural Networks

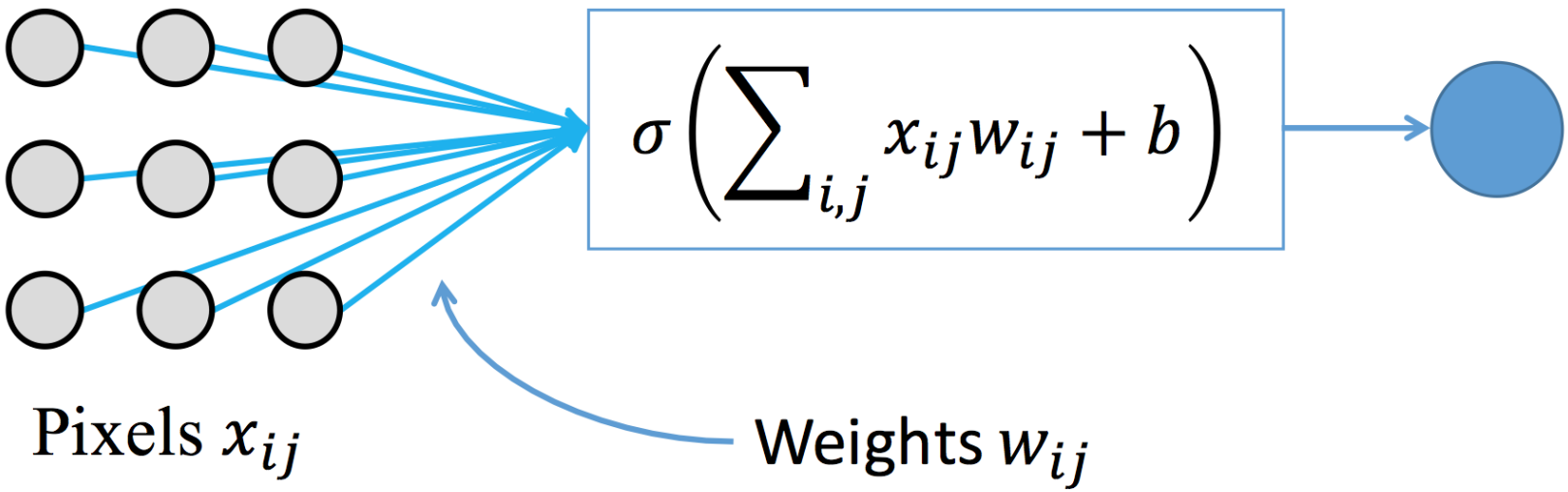
5 Regularizing Neural Networks

Image Representation

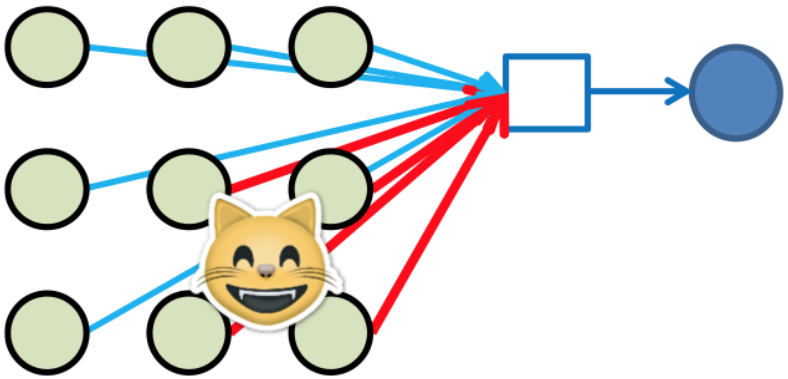
- Each pixel stores its brightness (or **intensity**) ranging from 0 to 255, 0 intensity corresponds to black color
- Color images store pixel intensities for 3 channels: **red**, **green** and **blue**



- Normalize input pixels: $\mathbf{x}_{\text{norm}} = \frac{\mathbf{x}}{255} - 0.5$
- Maybe MLP will work?

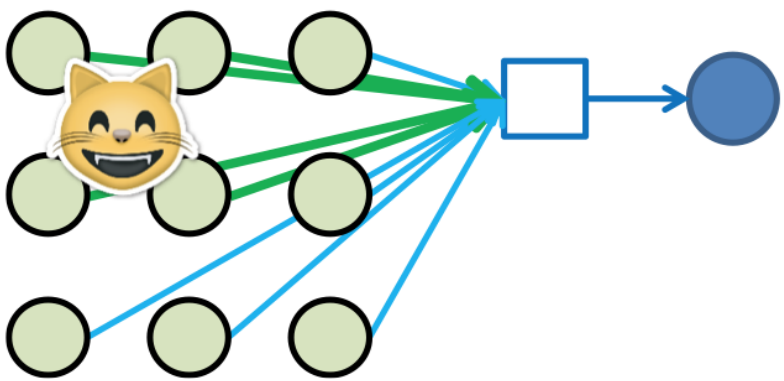


Fully-connected layered feed-forward network



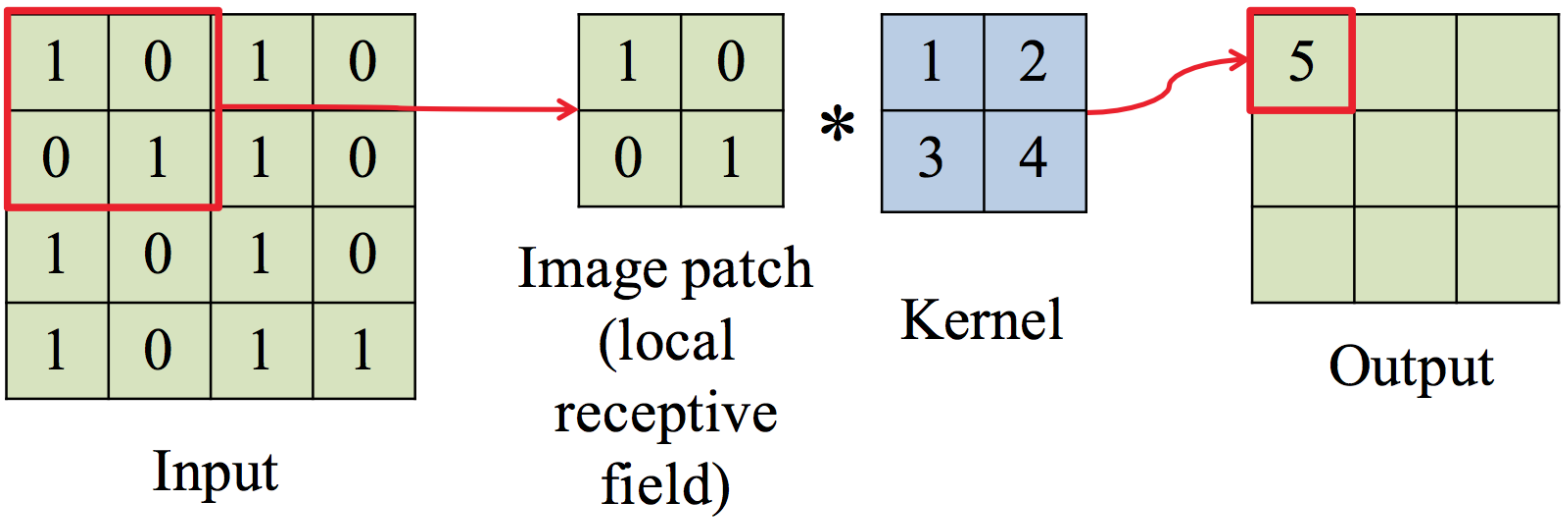
On this training image red weights w_{ij} will change a little bit to better detect a cat

Fully-connected layered feed-forward network

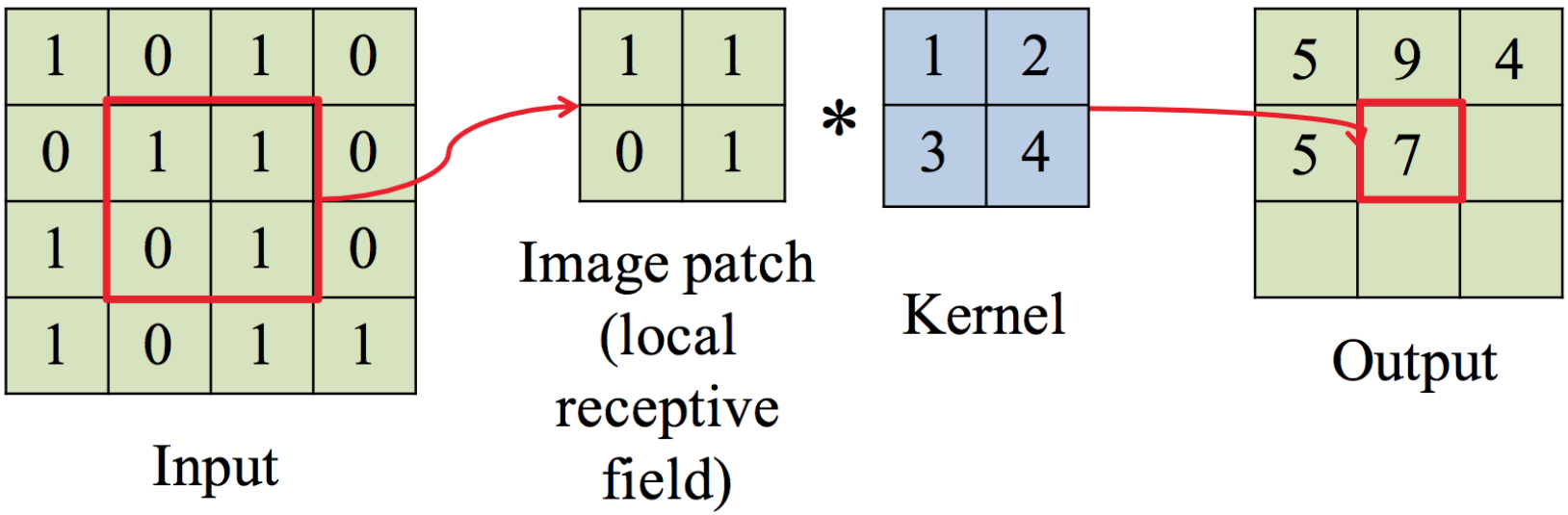


On this training image green weights w_{ij} will change a little bit to better detect a cat

- Convolution is a dot product of a **kernel** (or filter) and a patch of an image (**local receptive field**) of the same size



- Convolution is a dot product of a **kernel** (or filter) and a patch of an image (**local receptive field**) of the same size



Convolutions have been used for a while

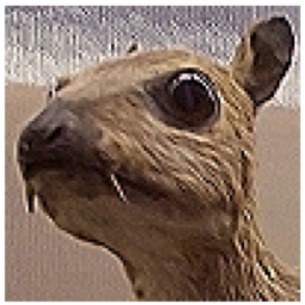
Original image

Kernel

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

Edge detection

$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

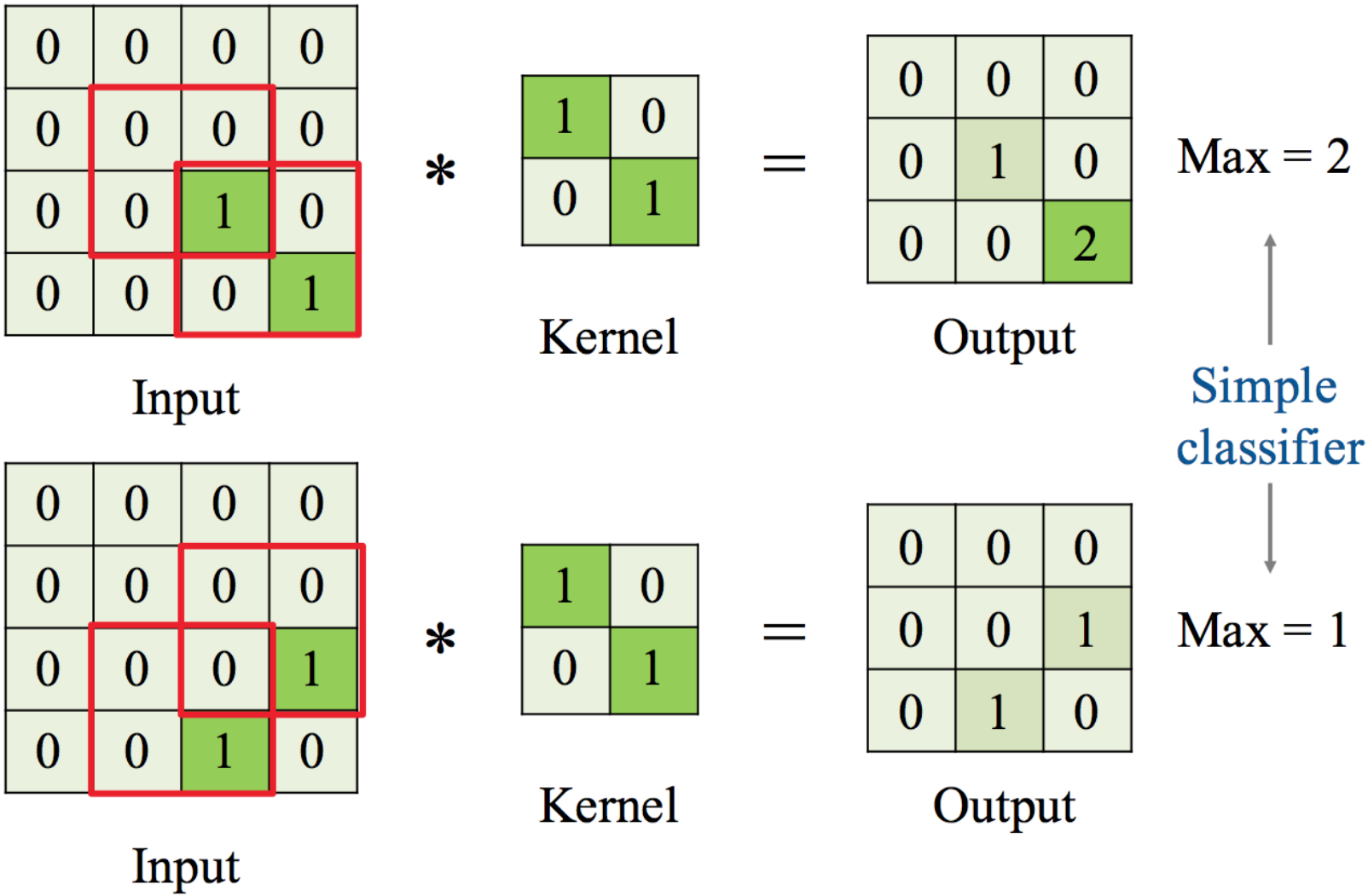


Sharpening

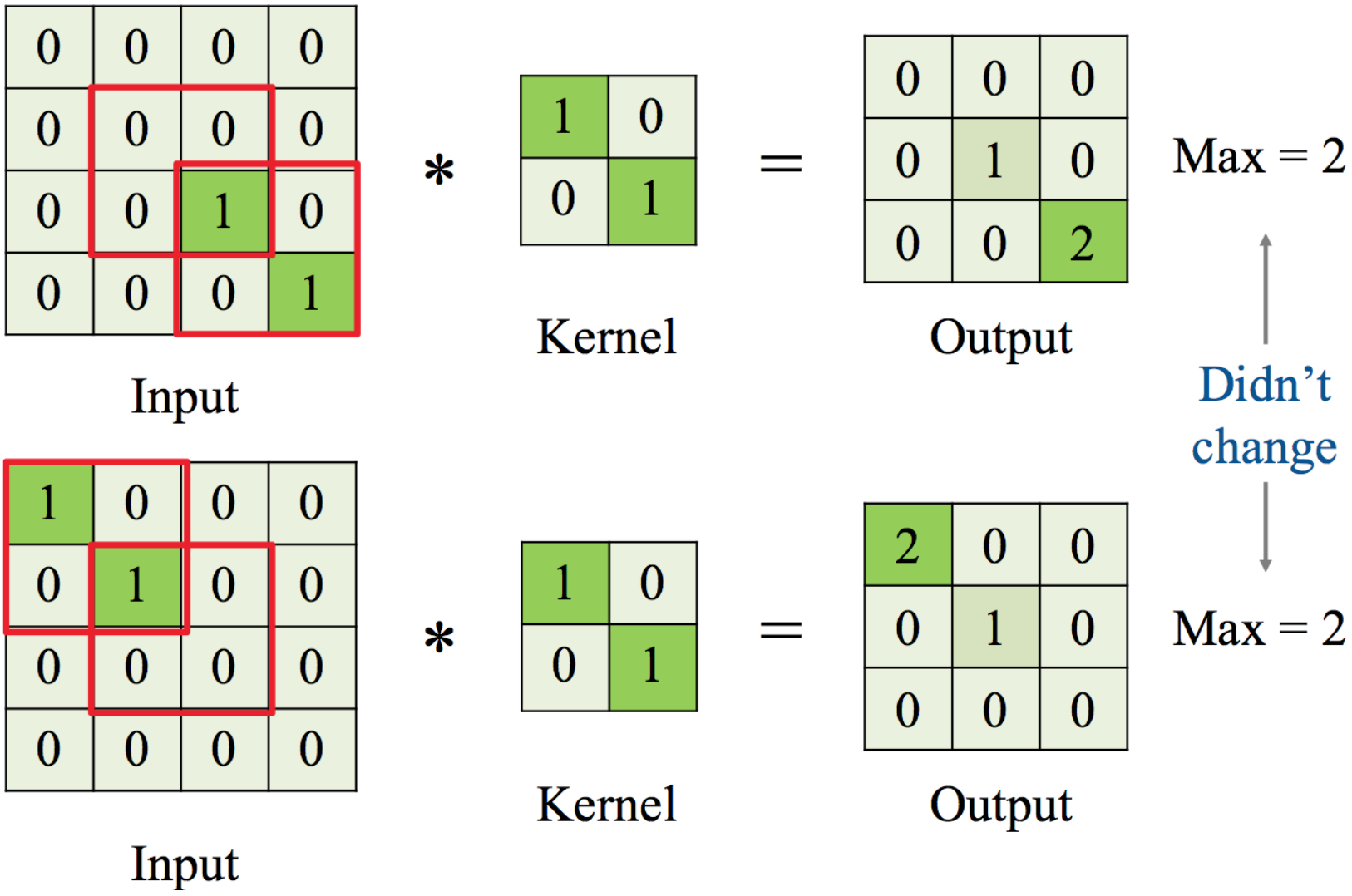
$$* \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

Blurring

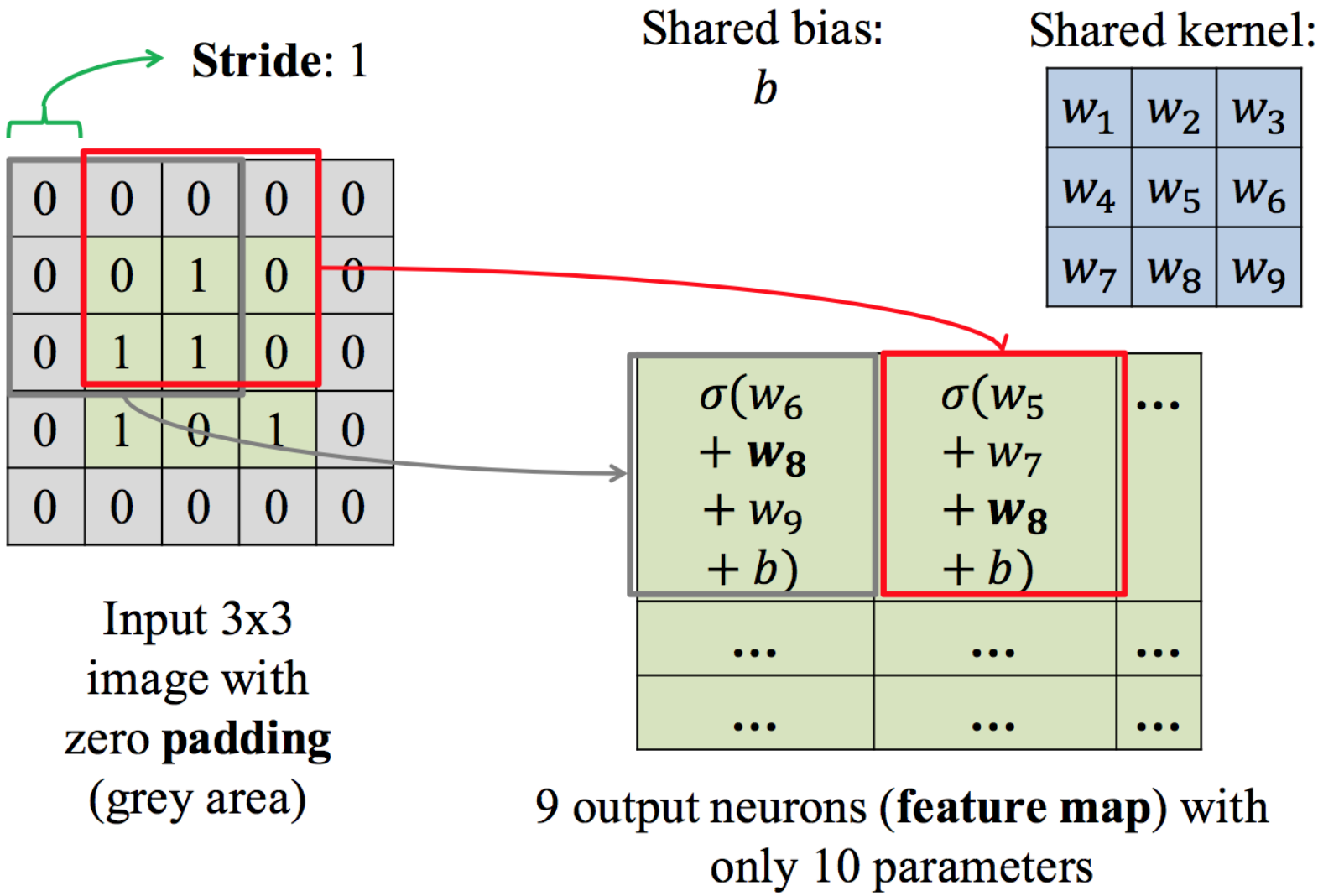
Convolution is similar to correlation



Convolution is translation invariant

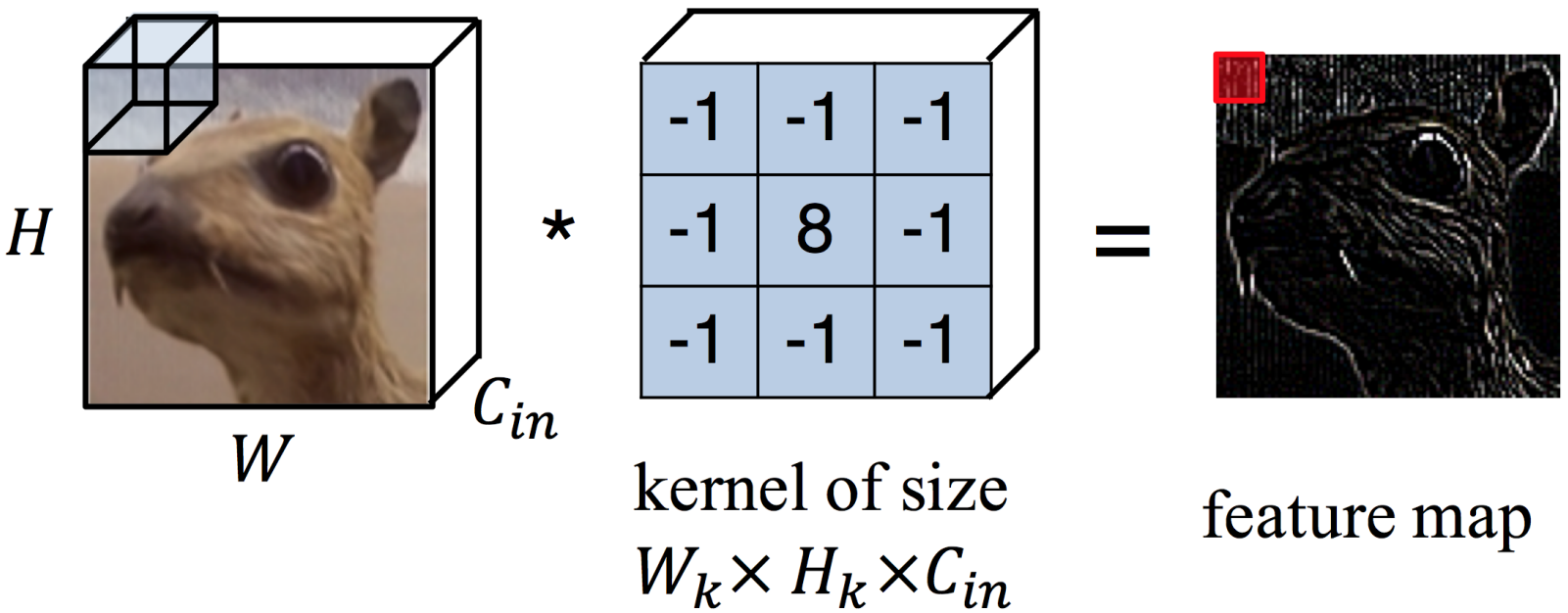


Convolutional layers



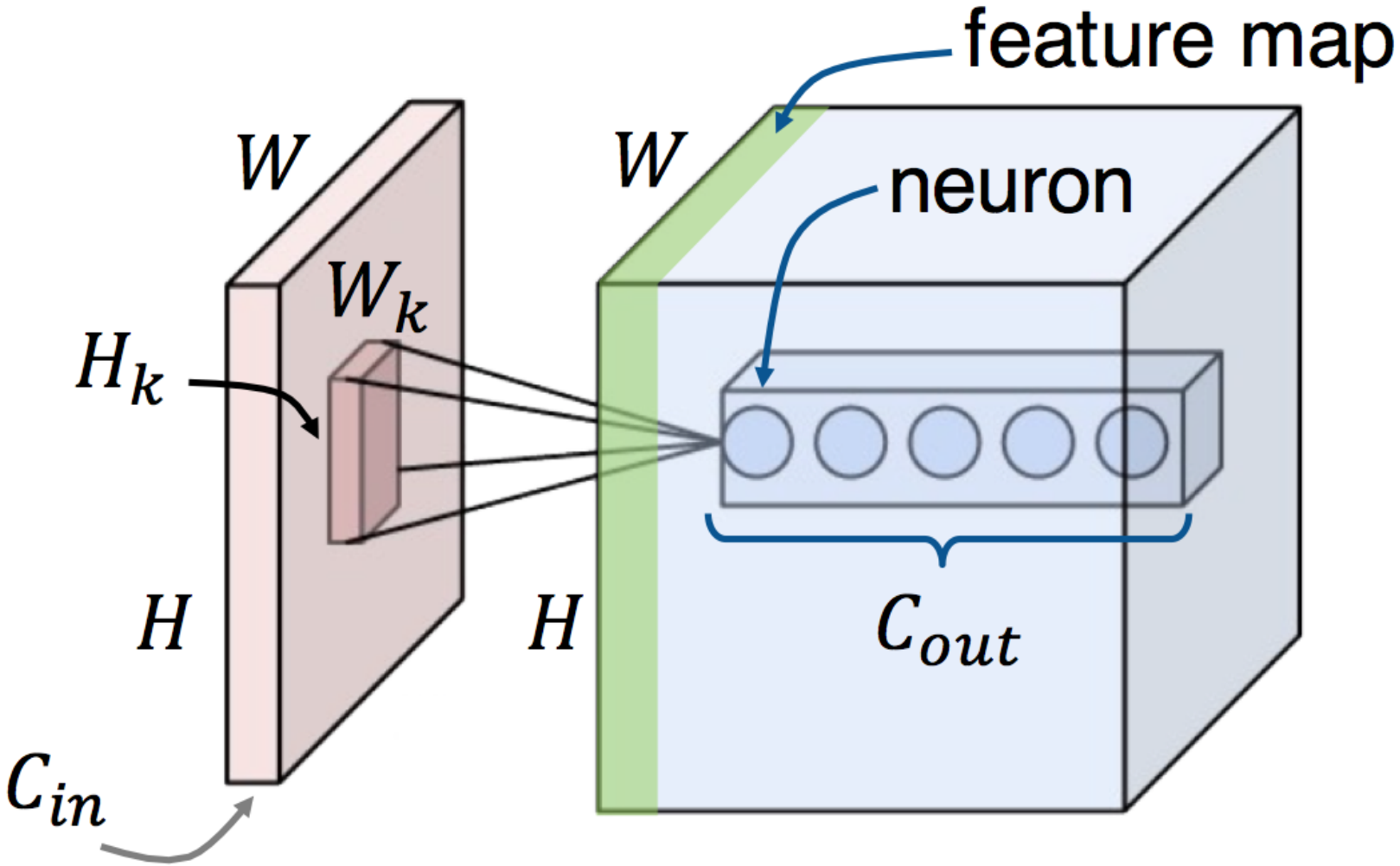
Let's say we have a color image as an input, which is $W \times H \times C_{in}$ **tensor** (multidimensional array), where

- W is an image width
- H is an image height
- C_{in} is a number of input channels (e.g. 3 RGB channels)



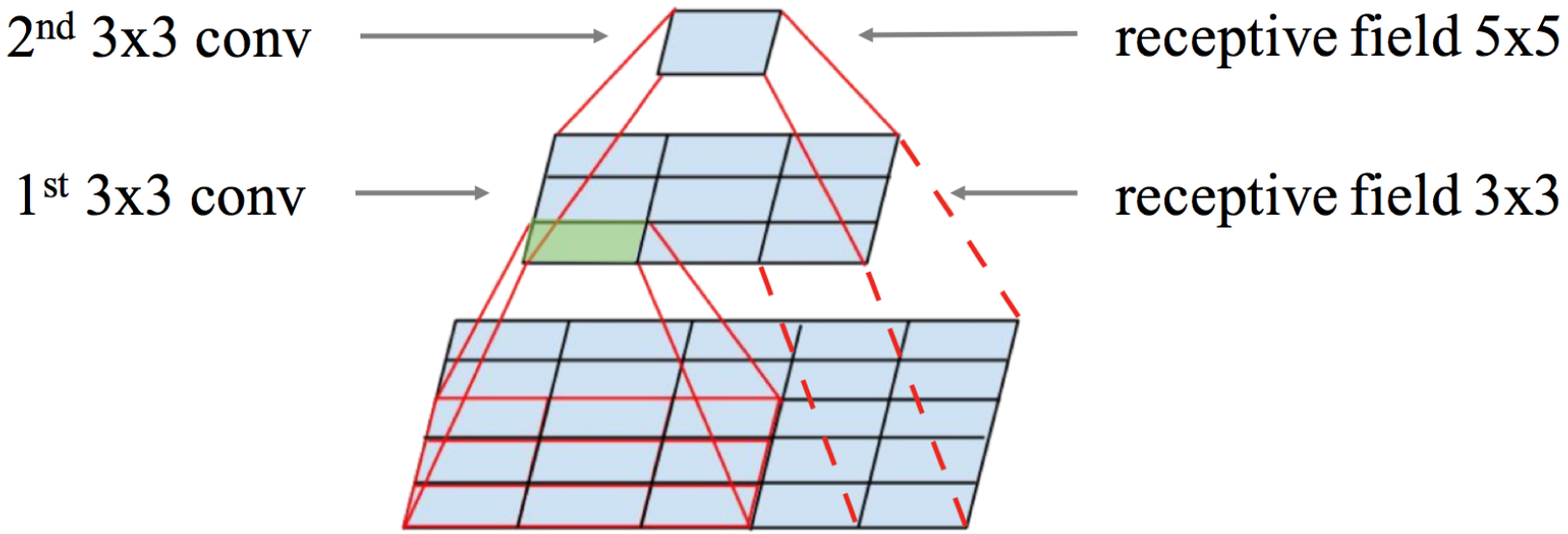
Convolutional layer

- We want to train C_{out} kernel of size $W_k \times H_k \times C_{in}$
- Having a stride of 1 and enough zero padding we can have $W \times H \times C_{out}$ output neurons



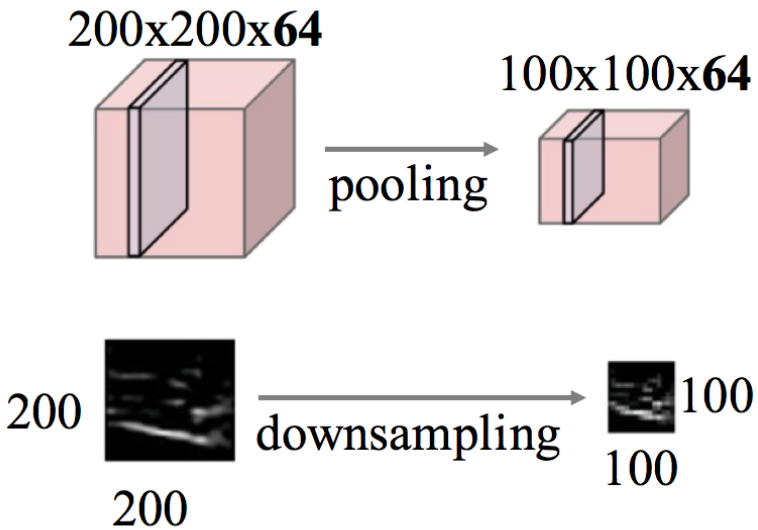
One convolutional layer is not enough!

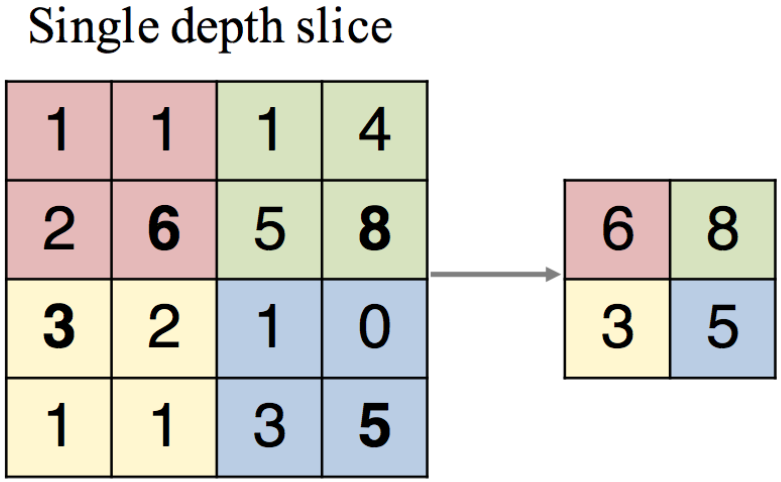
- Let's say neurons of the 1-st convolutional layer look at the patches of the image of size 3×3
- What if an object of interest is bigger than that?
- We need a 2-nd convolutional layer on top of the 1-st!



Pooling layer

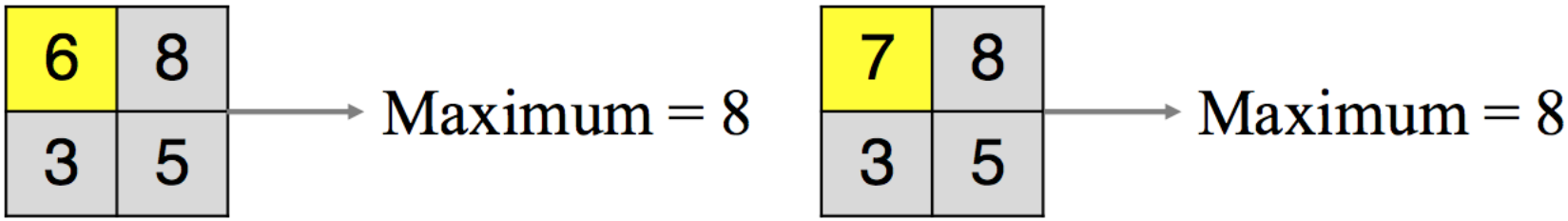
- This layer works like a convolutional layer but doesn't have kernel, instead it calculates **maximum** or **average** of input patch values



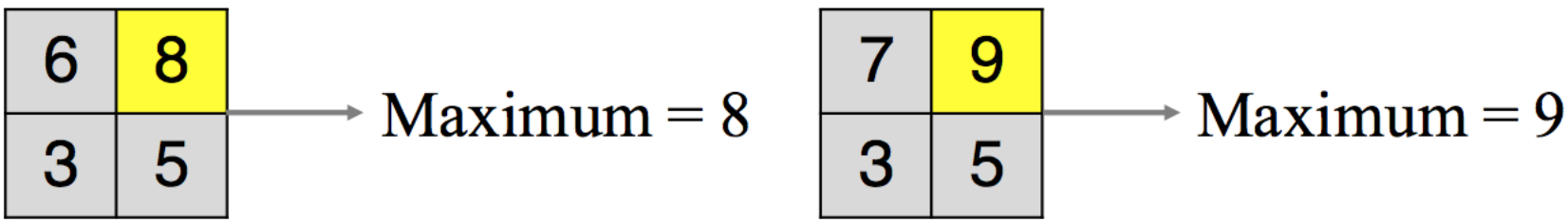


2x2 **max pooling** with stride 2

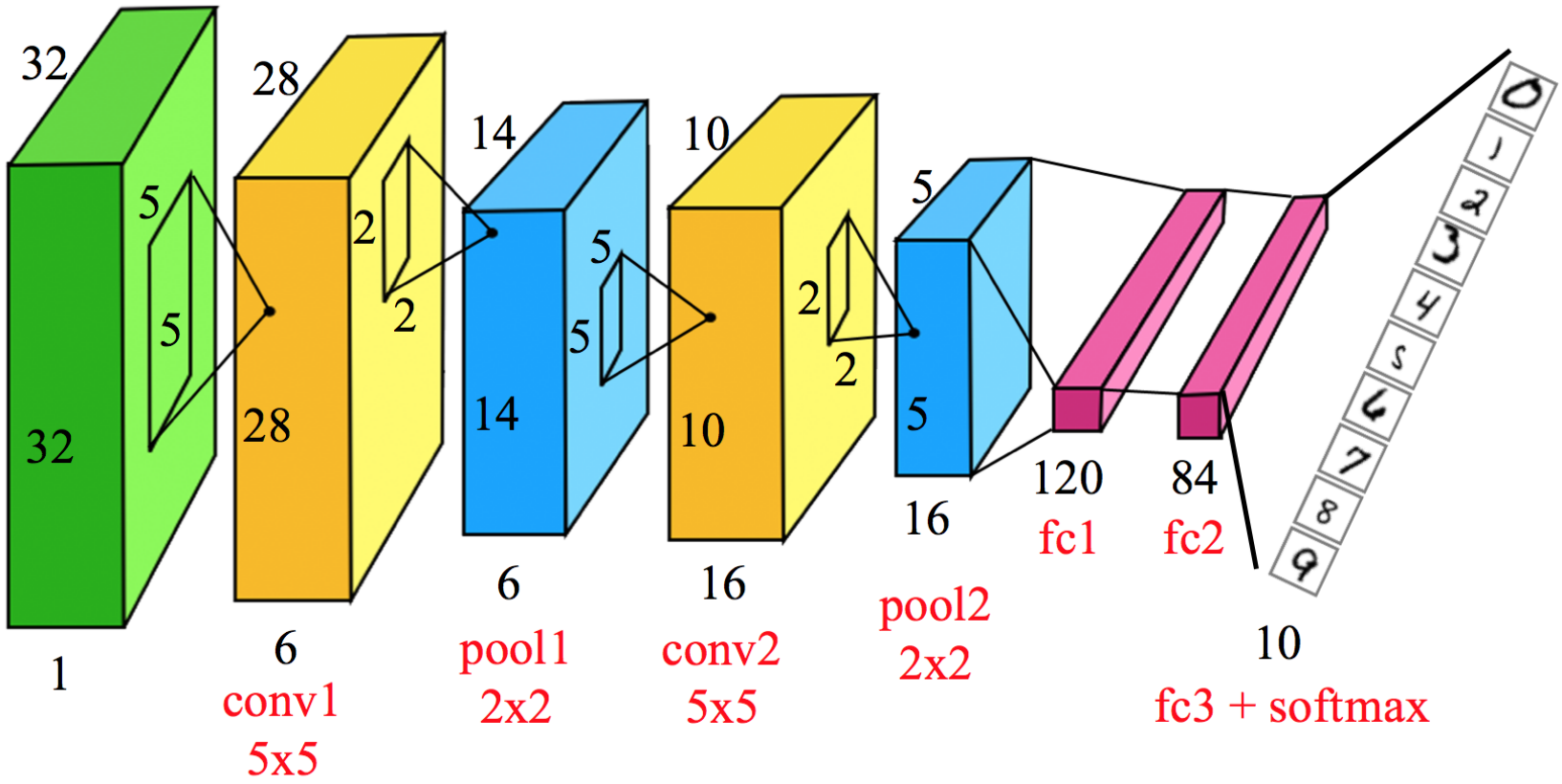
- Strictly speaking: maximum is not a differentiable function!



- There is no gradient with respect to non maximum patch neurons, since changing them slightly does not affect the output.



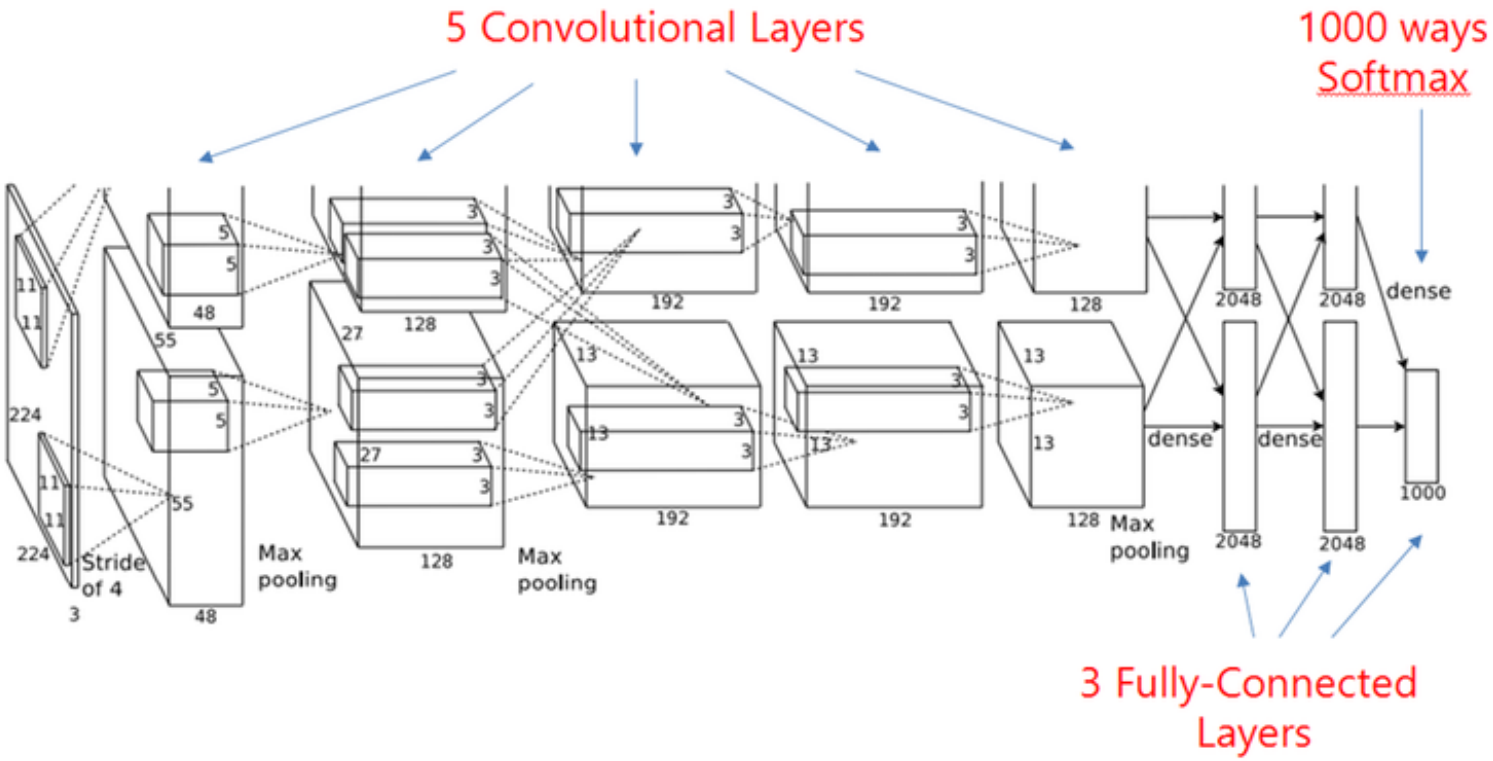
Convolutional network



http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

AlexNet (2012)

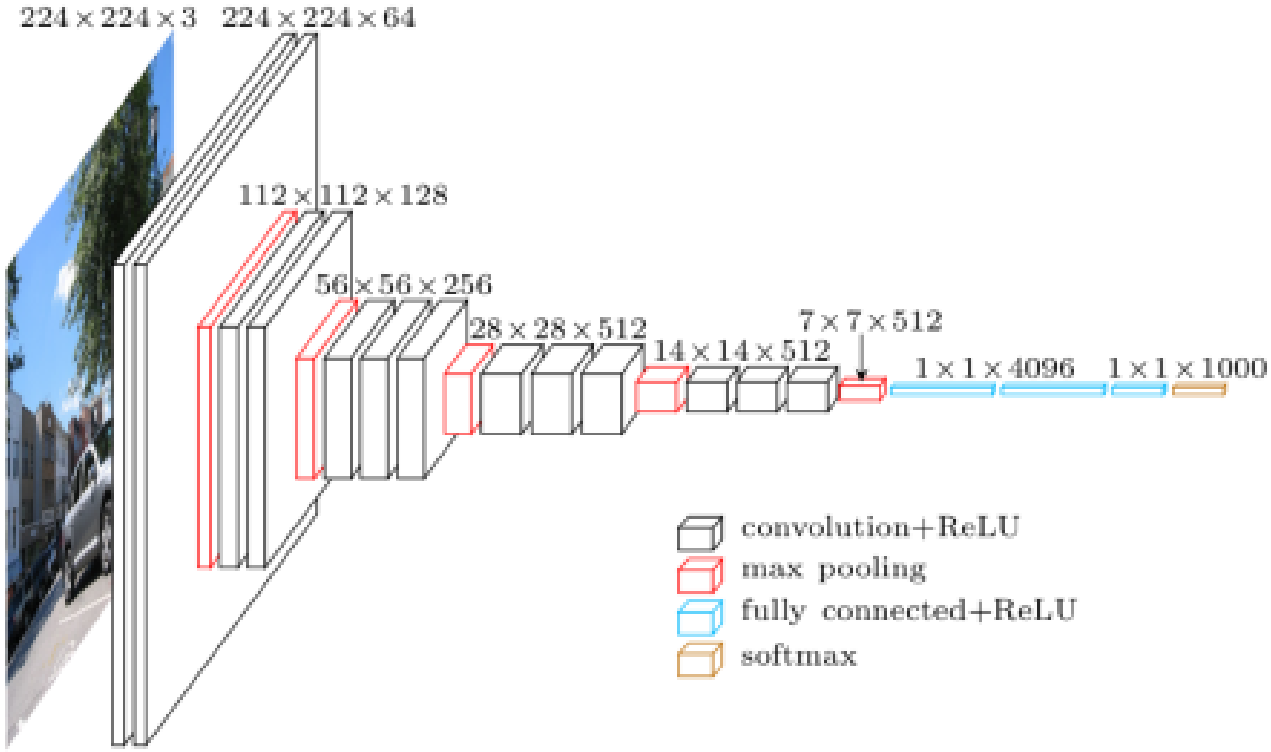
- First deep convolutional neural net for ImageNet
- Significantly reduced top 5 error from 26% to 15%



- 11×11, 5×5, 3×3 convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum
- 60 million parameters
- Trains on 2 GPUs for 6 days

VGG (2015)

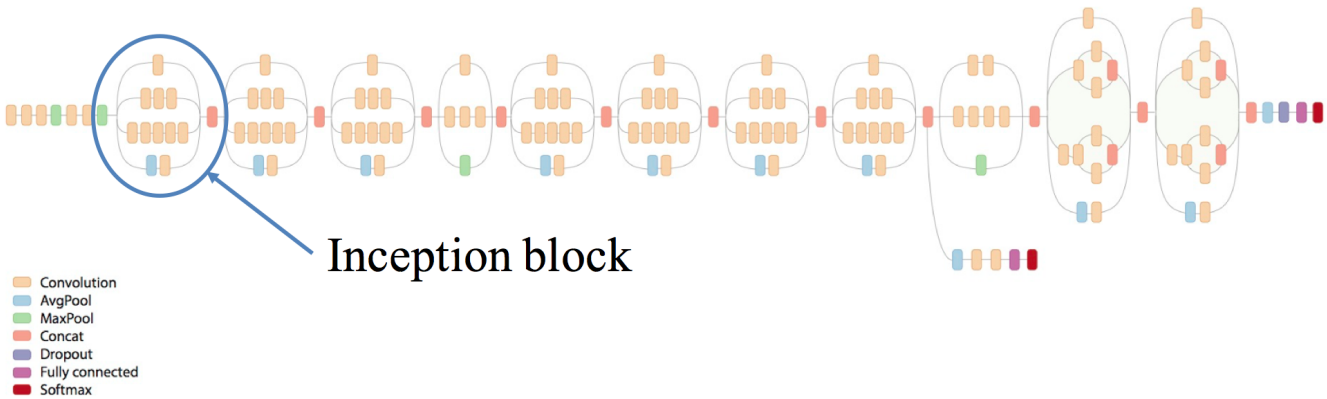
- Similar to AlexNet, only 3×3 convolutions, but lots of filters!
- ImageNet top 5 error: 8.0% (single model)



- Training similar to AlexNet with additional multi-scale cropping
- 138 million parameters
- Trains on 4 GPUs for $2-3$ weeks

Inception V3 (2015)

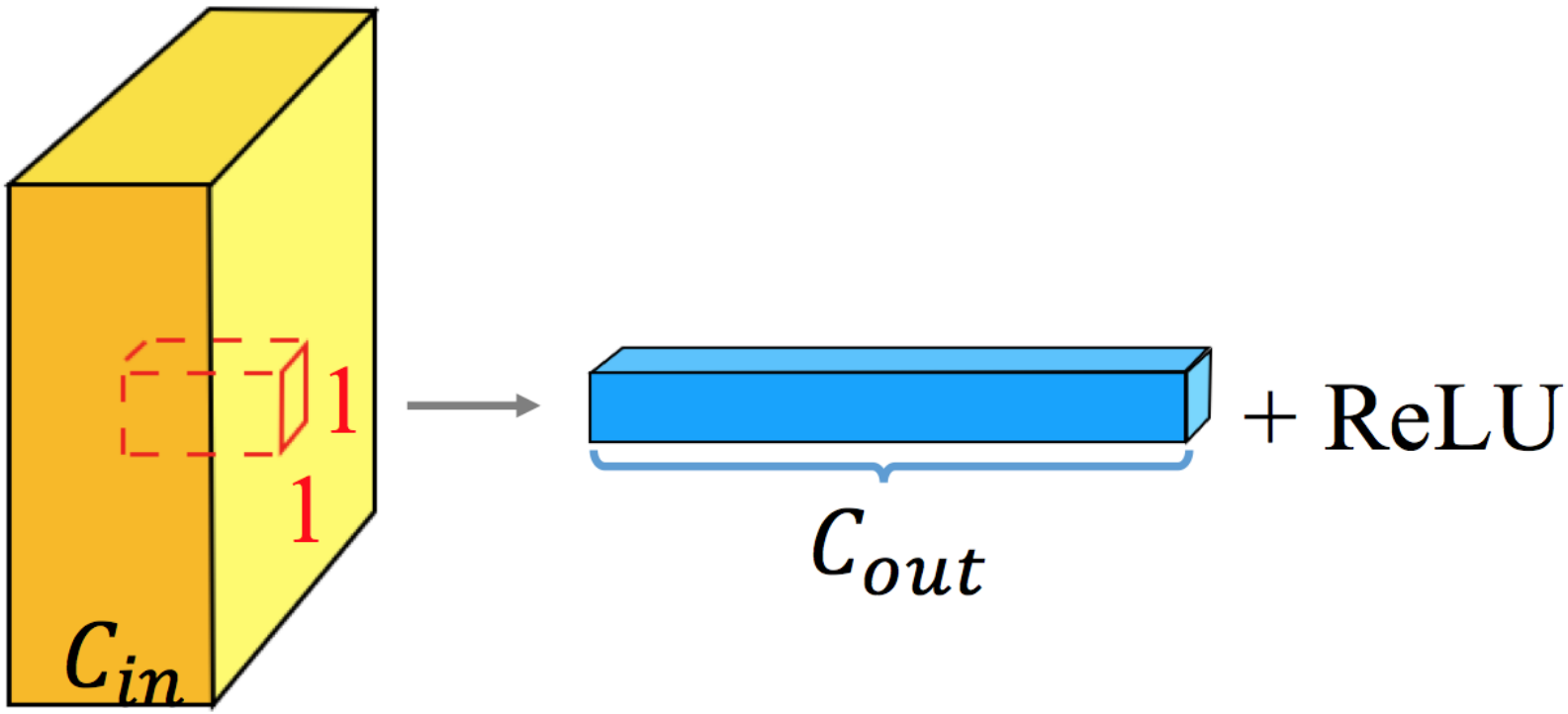
- Similar to AlexNet? Not quite, uses Inception block introduced in GoogLeNet (a.k.a. Inception V1)
- ImageNet top 5 error: 5.6% (single model), 3.6% (ensemble)



- Batch normalization, image distortions, RMSProp
- 25 million parameters!
- Trains on 8 GPUs for 2 weeks

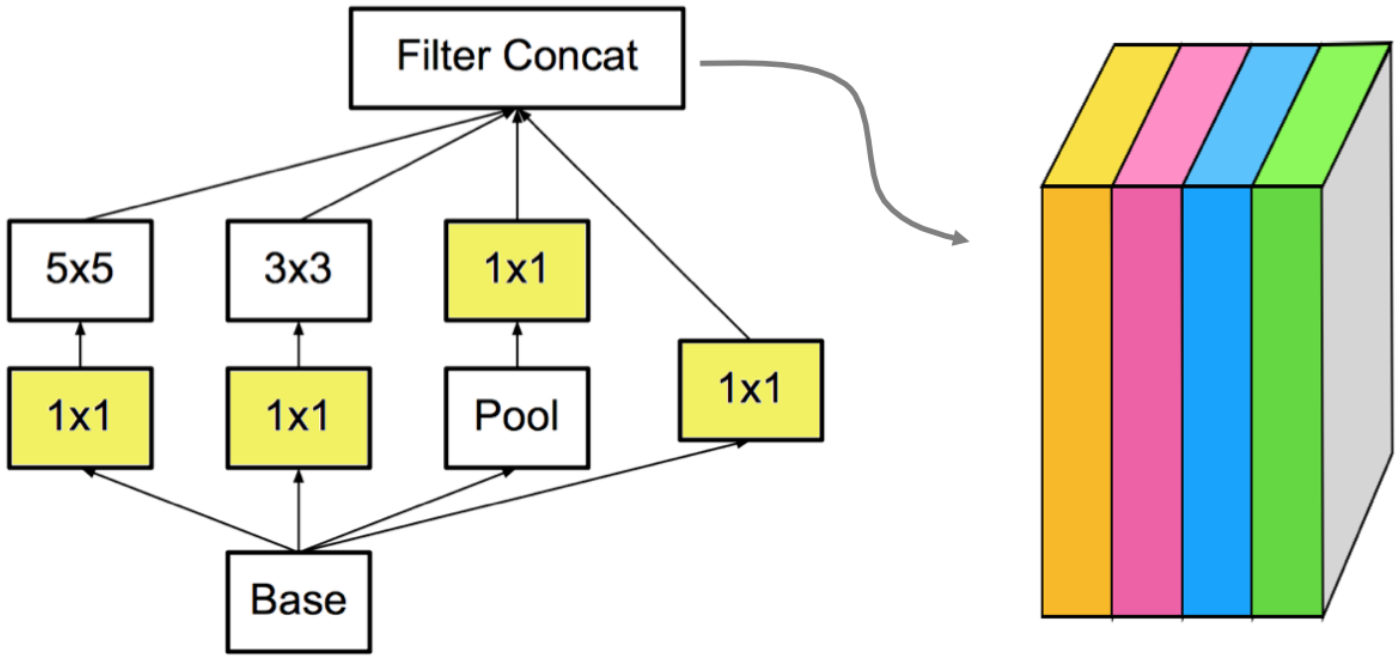
1×1 convolution

- Such convolutions capture interactions of input channels in one "pixel" of feature map
- They can reduce the number of channels not hurting the quality of the model, because different channels can correlate
- Dimensionality reduction with added ReLU activation



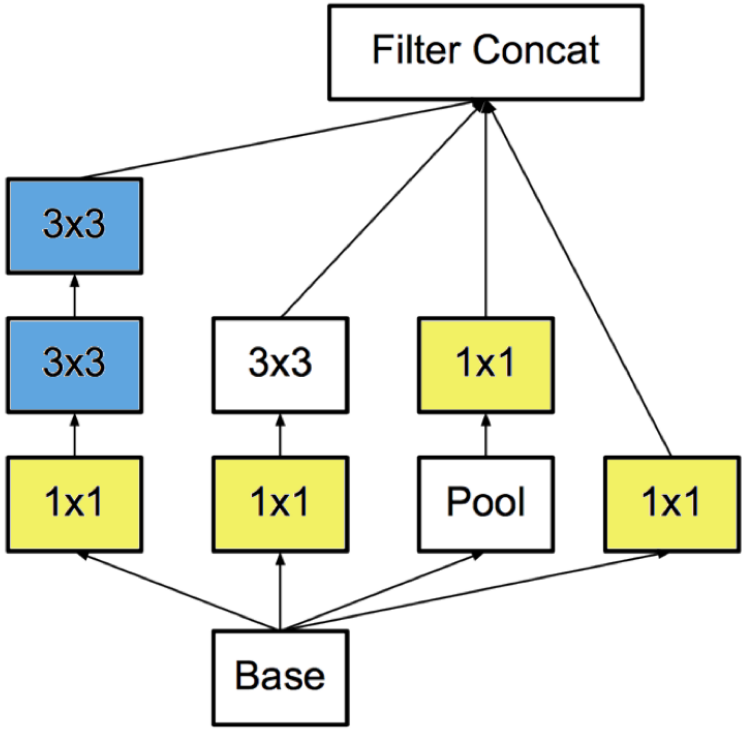
Inception block

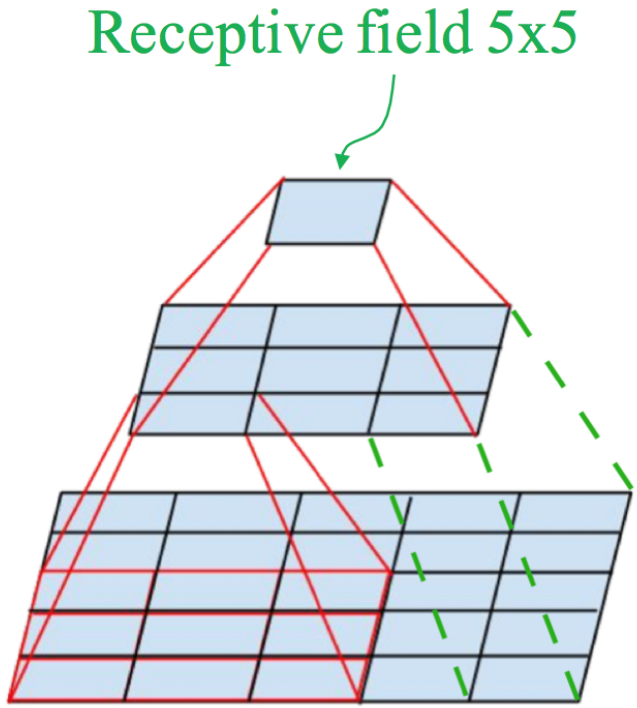
- All operations inside a block use stride 1 and enough padding to output the same spatial dimensions $(W \times H)$ of feature map.
- 4 different feature maps are concatenated on depth at the end



Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

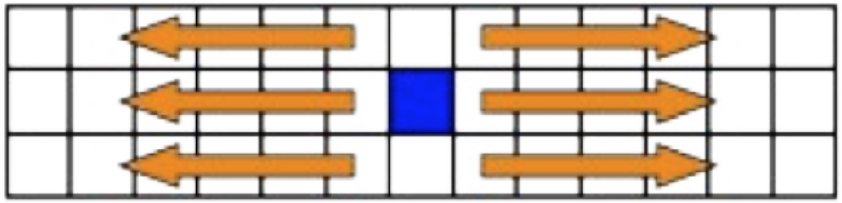
Replacing 5×5 convolutions



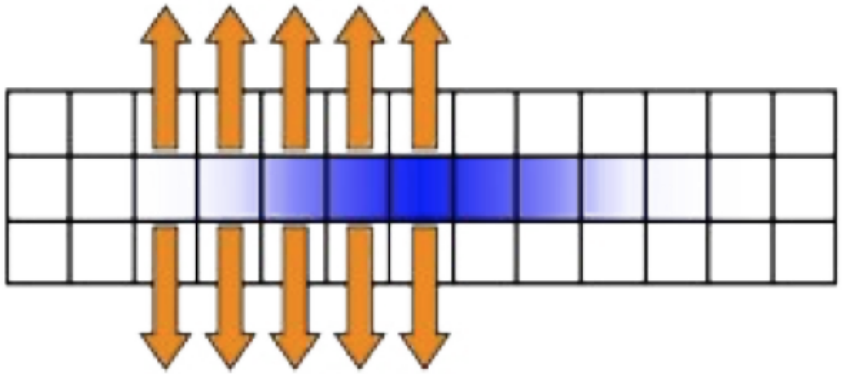


Christian Szegedy, https://arxiv.org/pdf/1512.00567.pdf

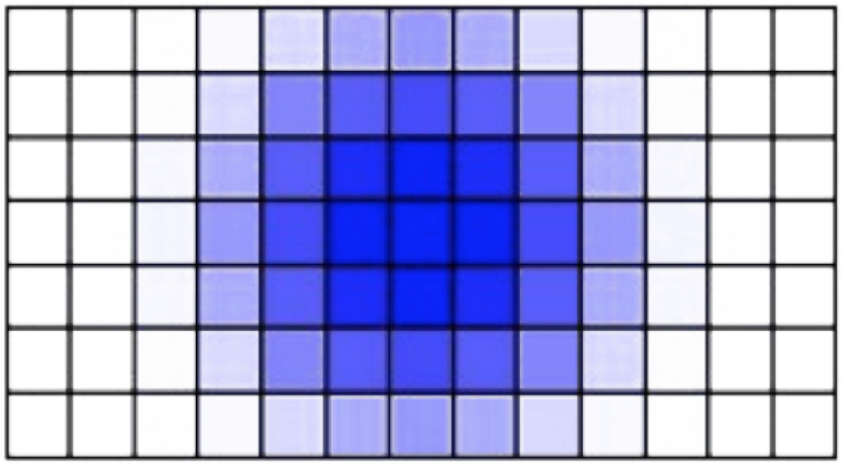
Filter decomposition



Blur the source horizontally

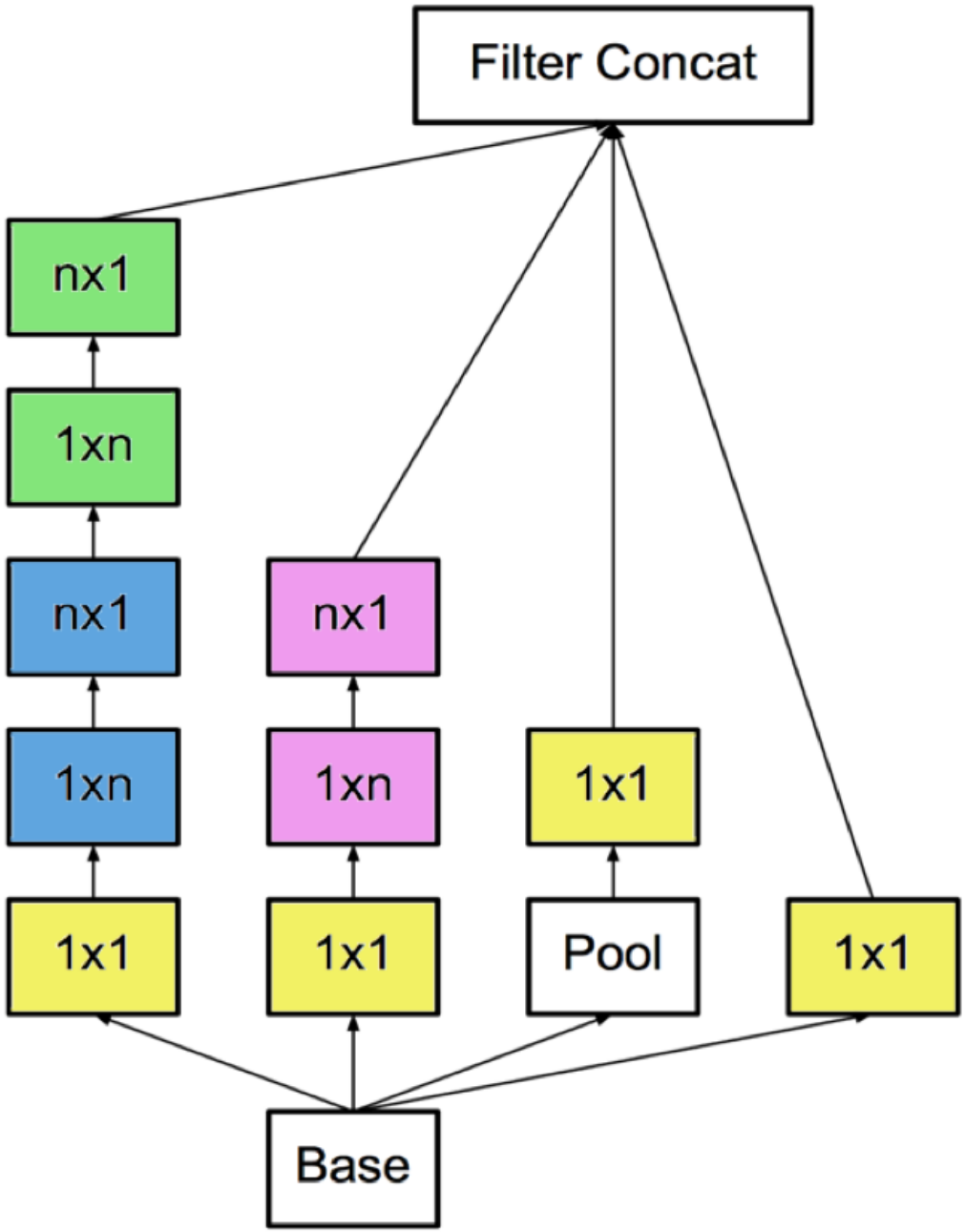


Blur the blur verticaly



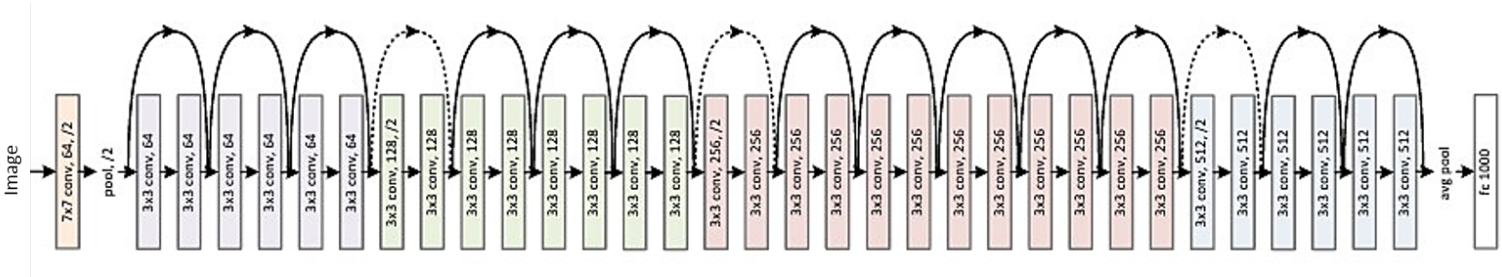
Result

Inception block



ResNet (2015)

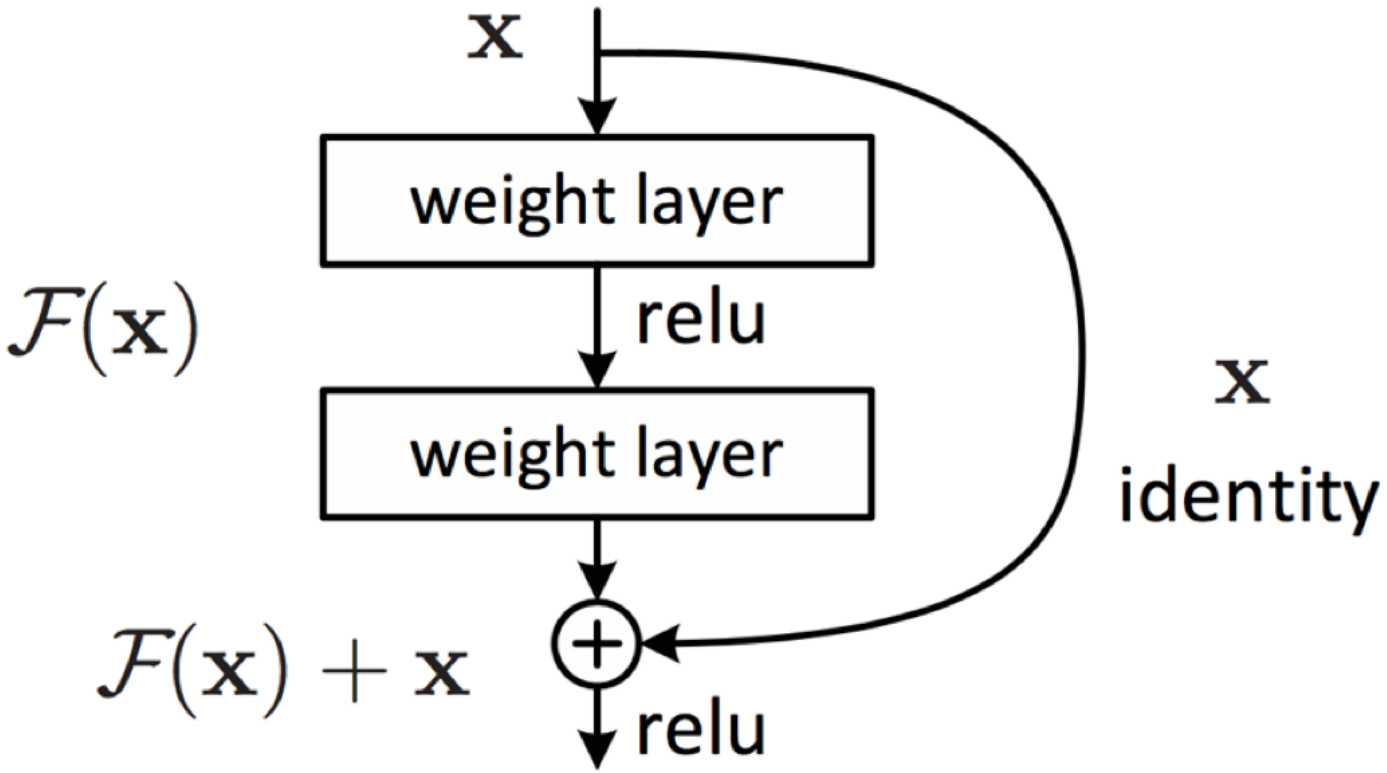
- Introduces residual connections
- ImageNet top 5 error: 4.5% (single model), 3.5% (ensemble)



- 152 layers, few 7×7 convolutional layers, the rest are 3×3, batch normalization, max and average pooling
- 60 million parameters
- Trains on 8 GPUs for $2 - 3$ weeks

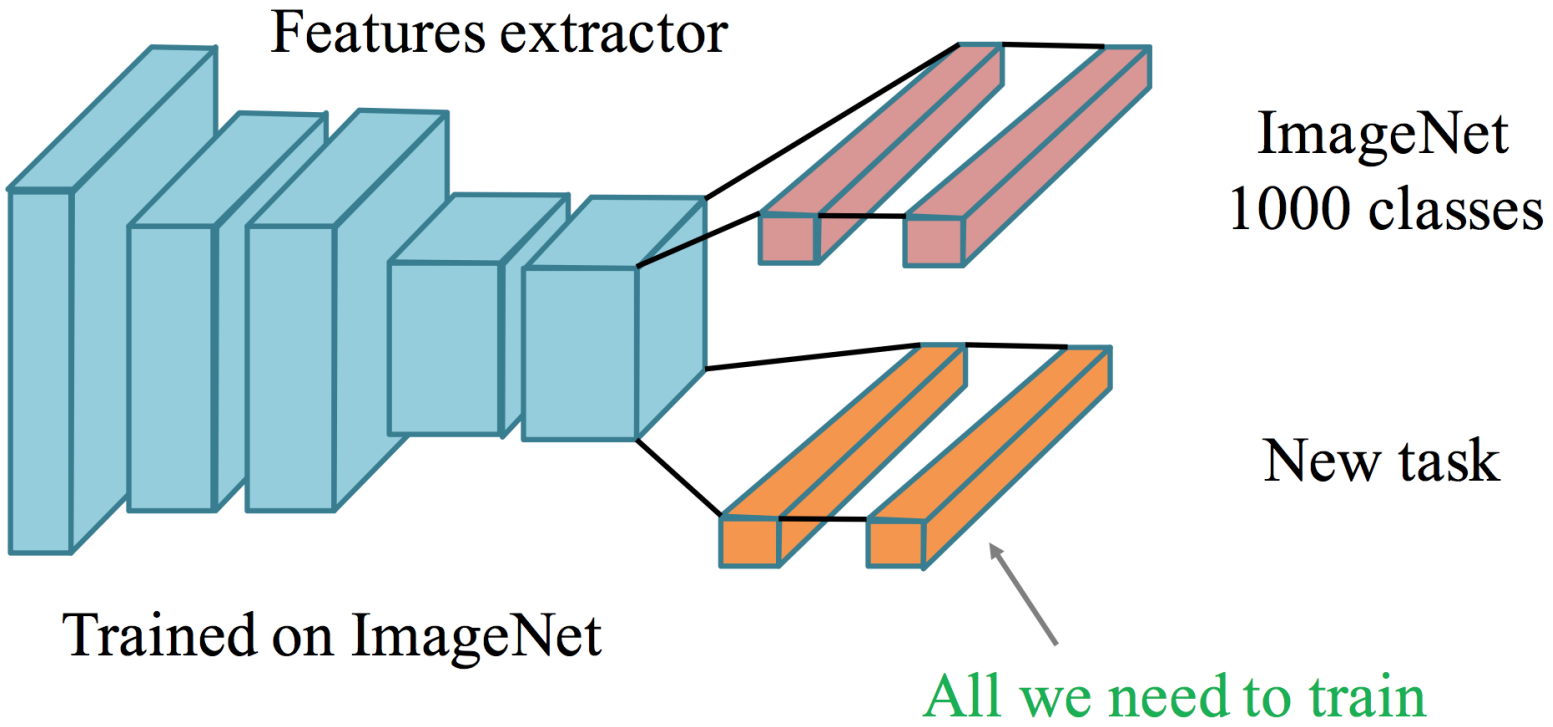
Residual connections

- We create output channels adding a small delta $F(\mathbf{x})$ to original input channels $\mathbf{x}$



- This way we can stack thousands of layers and gradients do not vanish thanks to residual connections

- Deep networks learn complex features extractor, but we need lots of data to train it from scratch!
- What if we can reuse an existing features extractor for a new task?

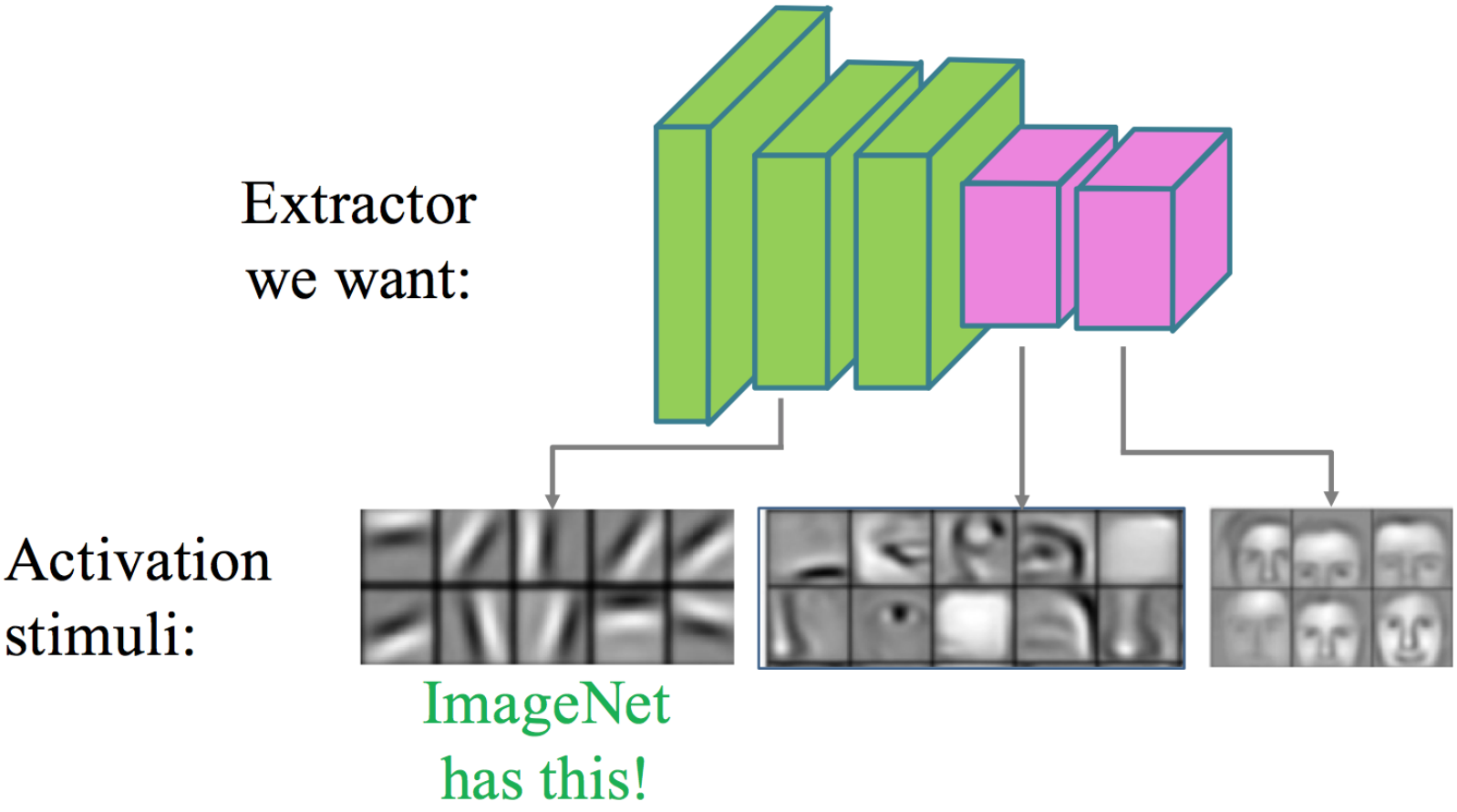


Transfer learning

- You need less data to train (for training only final MLP)
- It works if a domain of a new task is similar to ImageNet's
- E.g. won't work for human emotions classification, ImageNet doesn't have people faces in the dataset!

Transfer learning

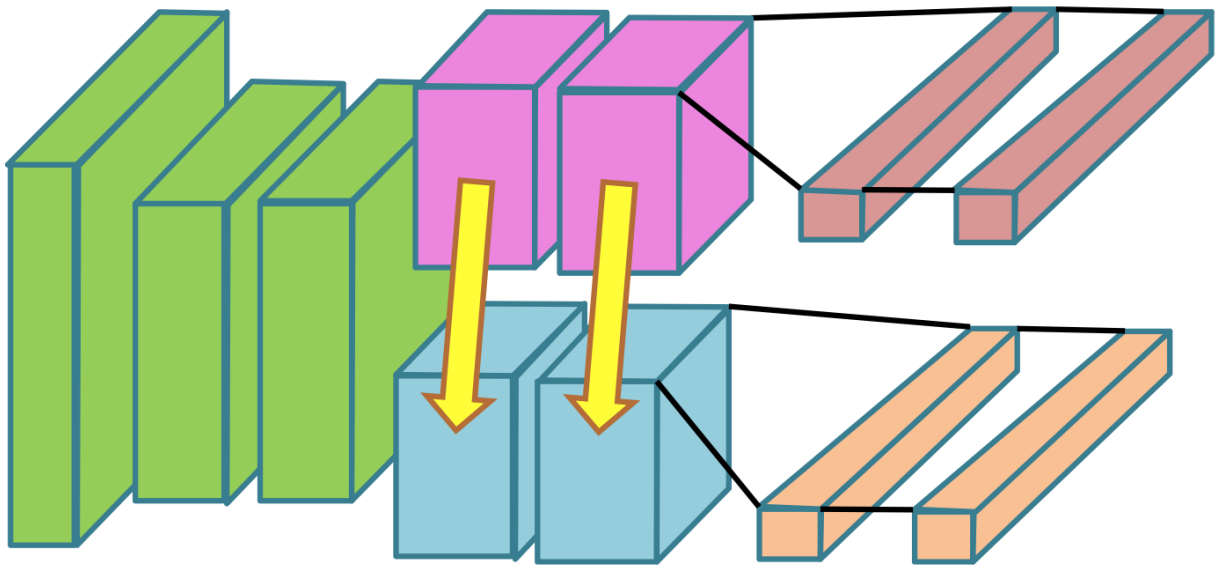
- But what if we need to classify human emotions?
- Maybe we can partially reuse ImageNet features extractor?



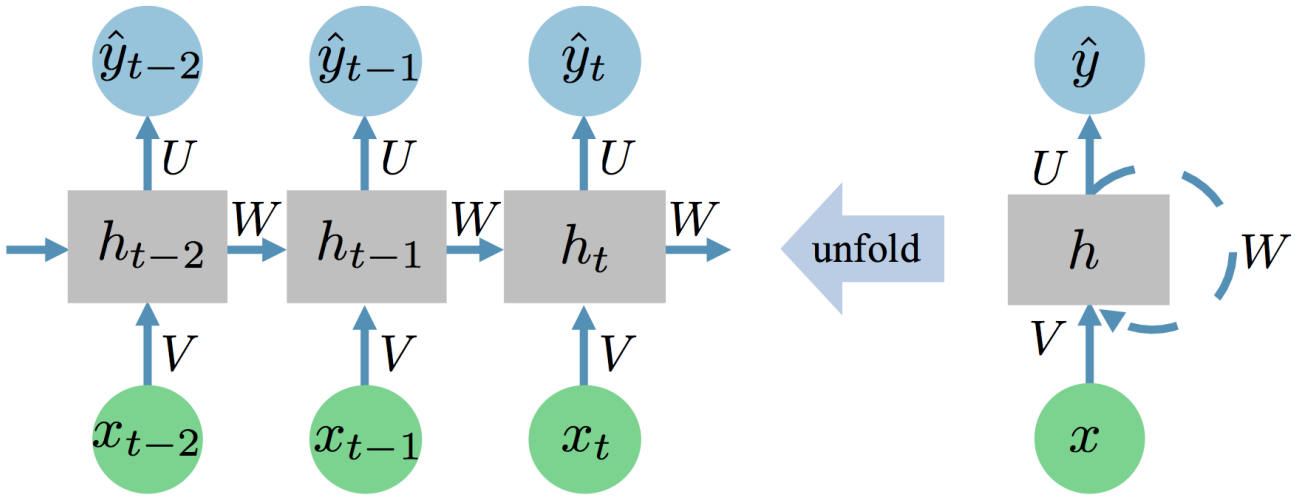
Honglak Lee, http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Transfer learning

- You can initialize deeper layers with values from ImageNet
- This is called **fine-tuning**, because you do not start with a random initialization
- Propagate all gradients with smaller learning rate



Recurrent neural networks



- $\mathbf{x}$ is an input
- $\widehat{y}$ is an output (prediction)
- h is a hidden state

- $h_t = f_h(V\mathbf{x}_t + Wh_{t-1} + b_h)$
- $\widehat{y}_t = f_y(Uh_t + b_y)$

Conclusions

- Simple differentiable computational blocks can be combined into graphs and still efficiently trained
- Convolutional architectures achieve superhuman performance in some vision problems
- Graphs trained on large datasets can be reused for other tasks
- Recurrent neural networks are quite efficient for sequential data

1 Introduction

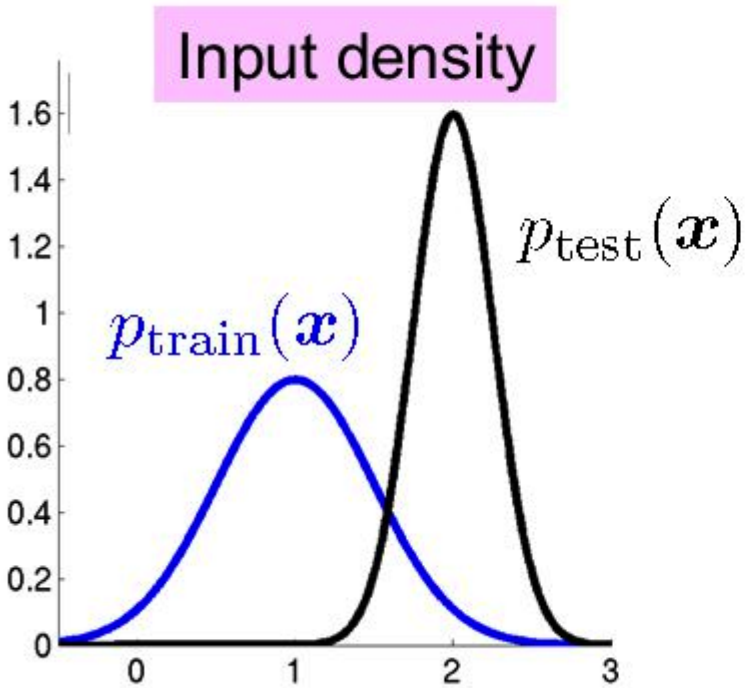
2 First generation of Neural Networks

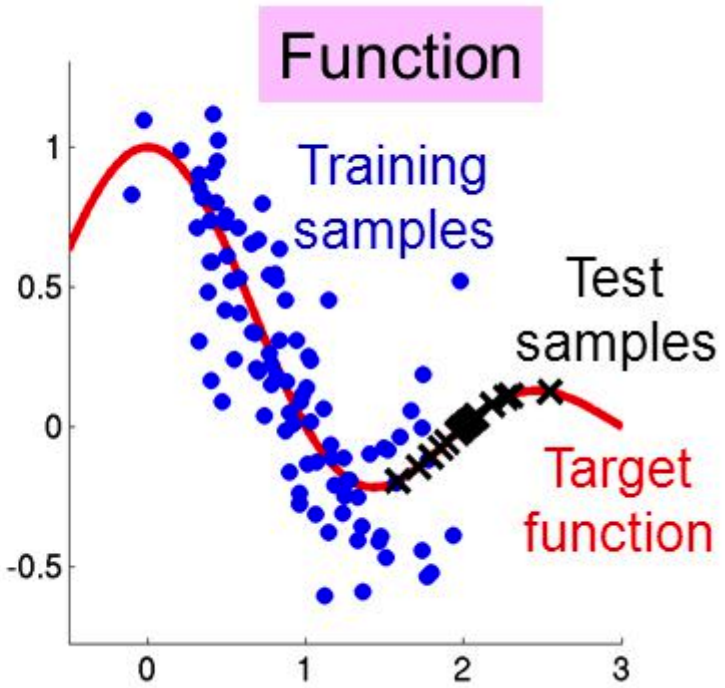
3 Modern Neural Networks

4 Convolutional Neural Networks

5 Regularizing Neural Networks

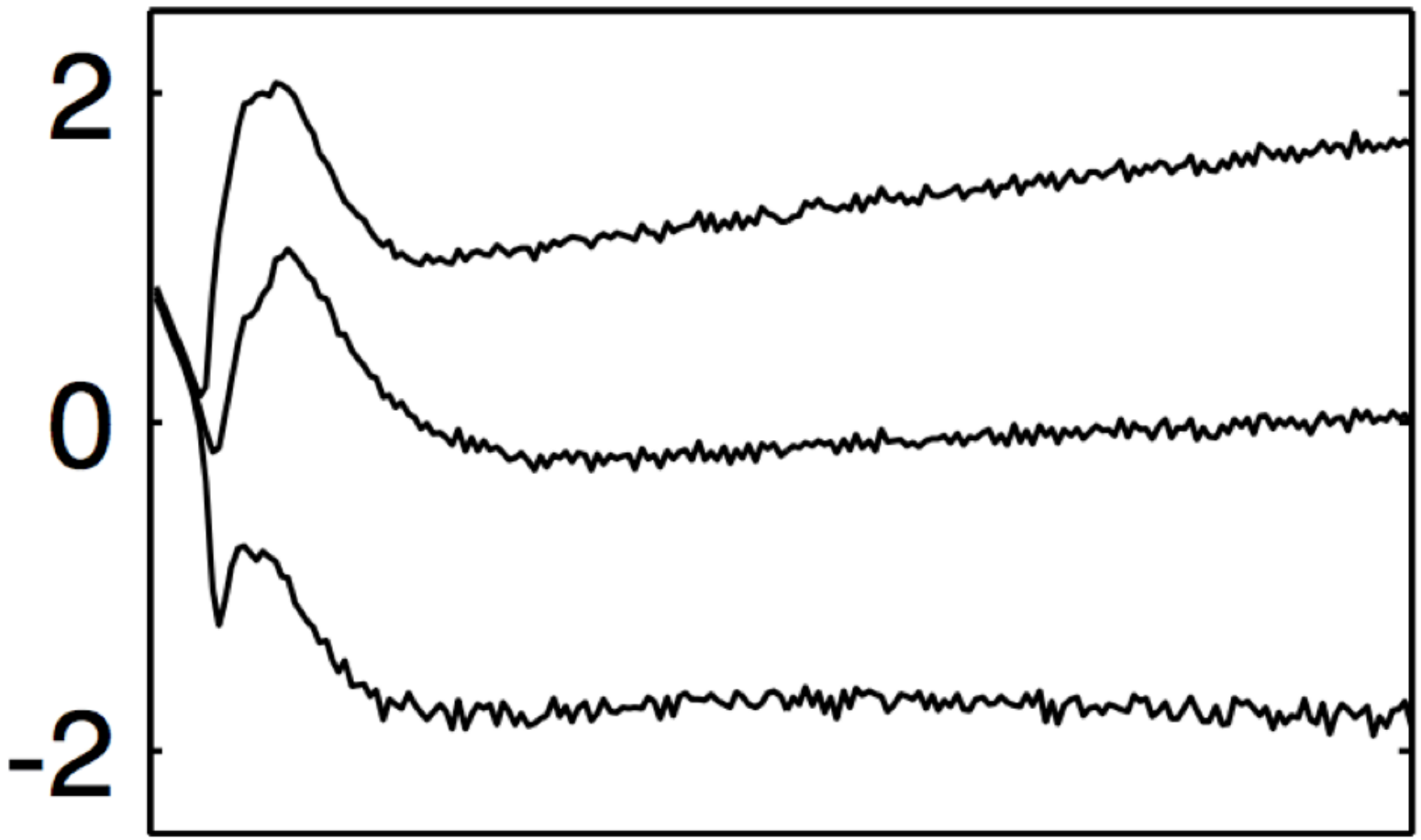
Covariate shift





Input distribution in neural network

- Input distribution changes over the course of training
- Example: 3 fully connected layers, 100 activations each
- Strong internal covariate shift



Simple normalization

- First idea: standardize output of a layer
- $f(\mathbf{x}; b) = \mathbf{x} + b - \mathbb{E}(\mathbf{x} + b)$
- If the backprop ignores dependence of sample average on b, then after step $b + g$:

$$f(\mathbf{x}; b) = \mathbf{x} + b + g - \mathbb{E}(\mathbf{x} + b + g) = \mathbf{x} + b - \mathbb{E}(\mathbf{x} + b)$$

- Parameter b can grow indefinitely
- Gradients should take normalization into account!

Full normalization

- Normalizations layer:
$$\widehat{\mathbf{x}} = \mathrm{Norm}(\mathbf{x}, \mathcal{X})$$
- We should be able to calculate gradients:
$$\frac{\partial \mathrm{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathbf{x}} \quad \text{and} \quad \frac{\partial \mathrm{Norm}(\mathbf{x}, \mathcal{X})}{\partial \mathcal{X}}$$
- Standartization requires inverse square root of covariance matrix:
$$\{\mathrm{Cov}(\mathbf{x})\}^{-1/2} (\mathbf{x} - \mathbb{E}\mathbf{x})$$

Simplifications

- Normalize each layer activation independently:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Estimate mean and variance of input based on current mini-batch

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

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

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

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

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

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

Gradients for BatchNorm layer

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

- Average mean and variance estimates over all training batches:

for $k = 1 \dots K$ **do**
// For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

- Use these new estimates for inference:

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

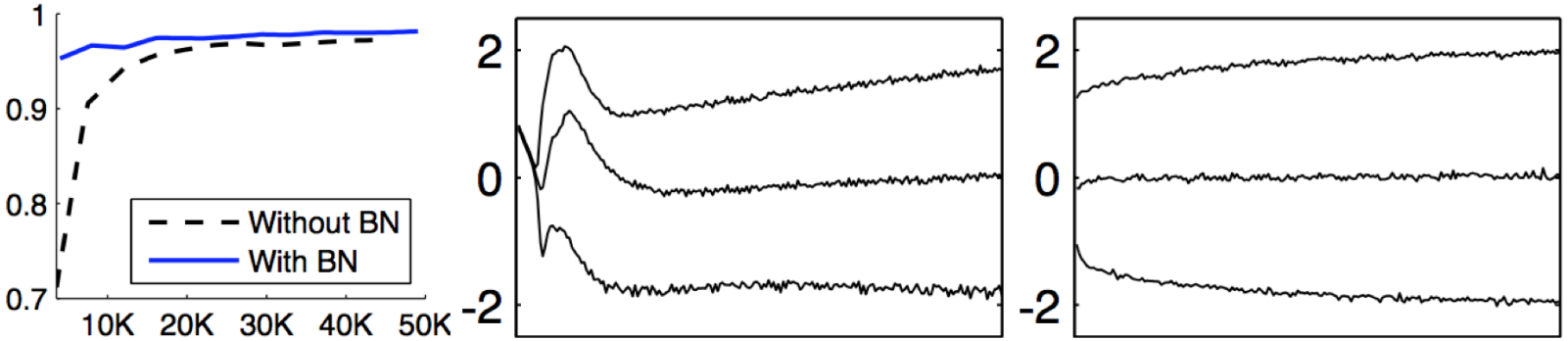
BatchNorm usage

- Usually BatchNorm is inserted before nonlinearity
- Learning rate can be increased
- Optimization becomes robust to parameter scale:

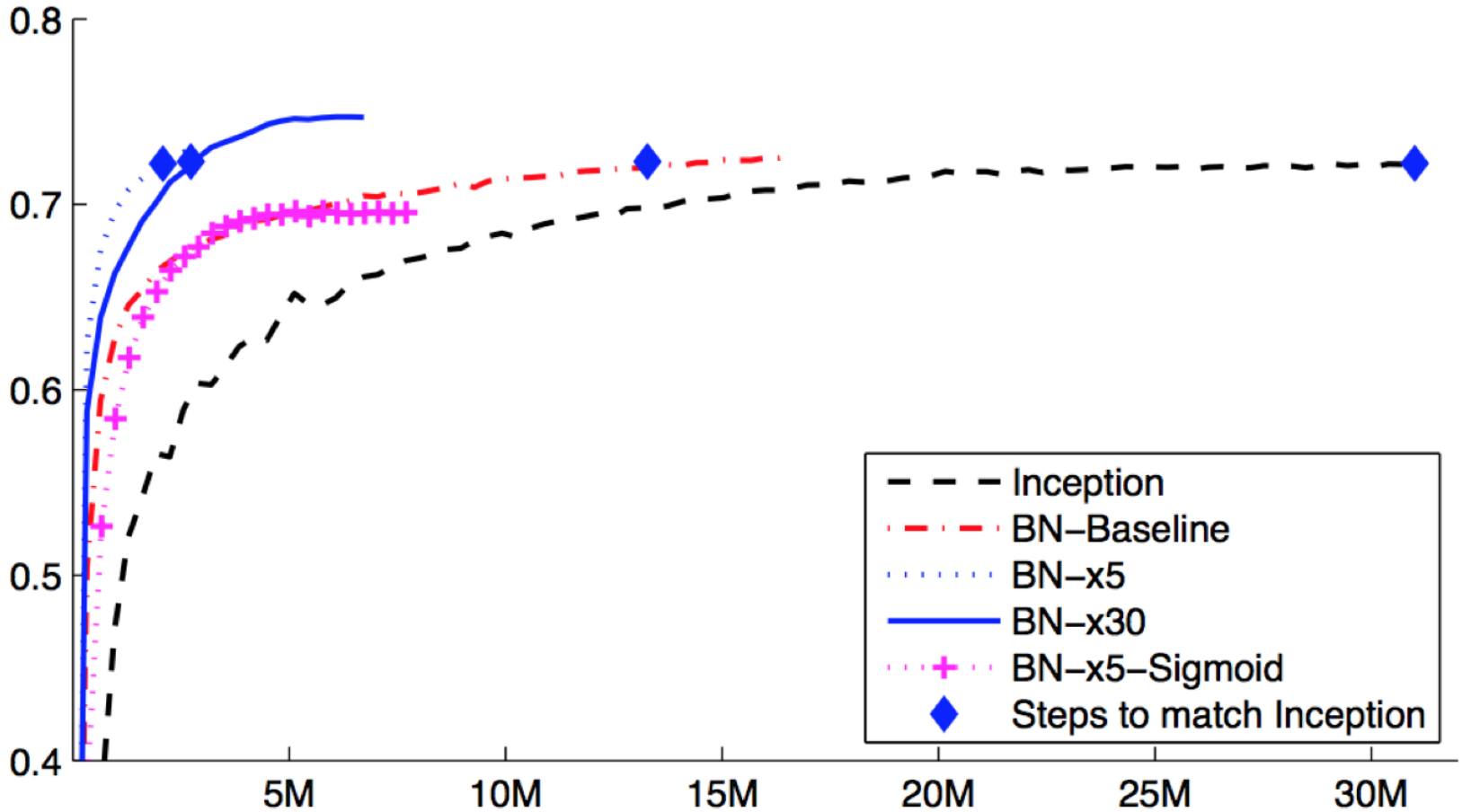
$$\frac{\partial \mathrm{BN}((aW)u)}{\partial u} = \frac{\partial \mathrm{BN}(Wu)}{\partial u}$$
$$\frac{\partial \mathrm{BN}((aW)u)}{\partial (aW)} = \frac{1}{a} \cdot \frac{\partial \mathrm{BN}(Wu)}{\partial W}$$

- Dropout layers can be removed
- Weight regularization can be reduced

BatchNorm example



BatchNorm example



- Weights reparametrization:

$$\mathbf{w} = \frac{g}{\|\mathbf{v}\|}\mathbf{v}$$

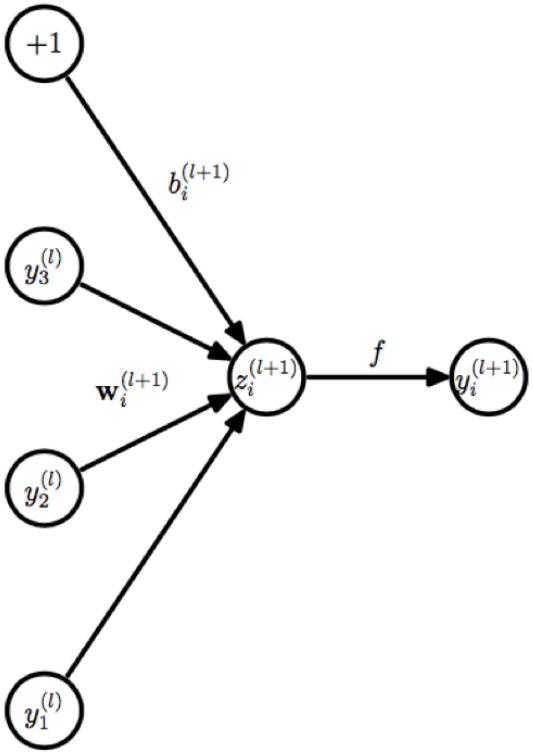
- Fixes the Euclidean norm of $\mathbf{w}$ to g
- Gradients:

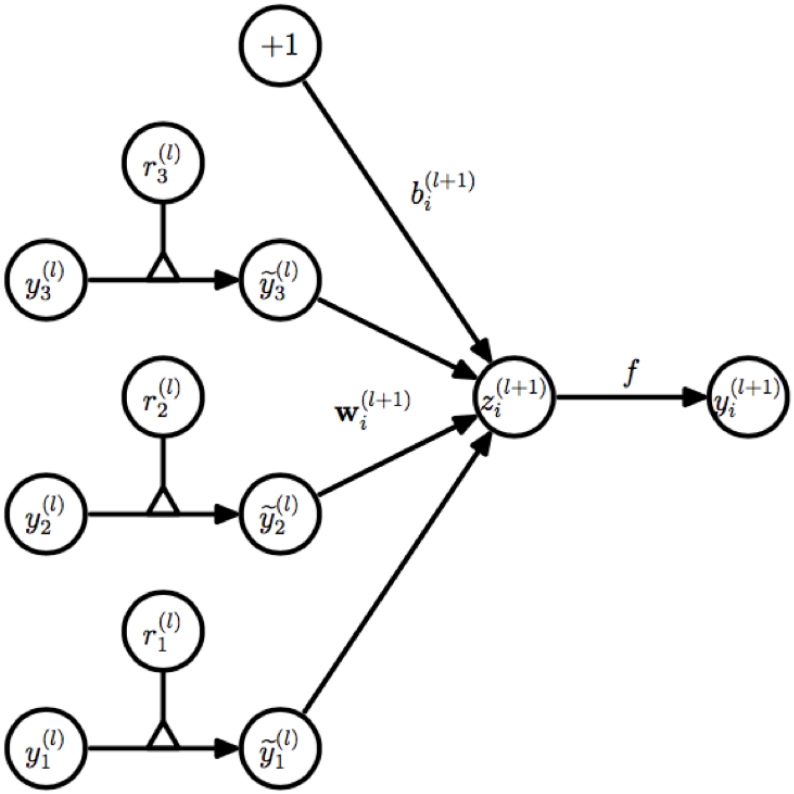
$$\nabla_g L = \frac{\nabla_{\mathbf{w}} L \cdot \mathbf{v}}{\|\mathbf{v}\|}, \quad \nabla_{\mathbf{v}} L = \frac{g}{\|\mathbf{v}\|}\nabla_{\mathbf{w}} L - \frac{g\nabla_g L}{\|\mathbf{v}\|^2}\mathbf{v}$$

WeightNorm results on CIFAR-10

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	**7.31%**

Dropout


+1
$b_i^{(l+1)}$
$y_3^{(l)}$
$\mathbf{w}_i^{(l+1)}$
$z_i^{(l+1)}$
f
$y_i^{(l+1)}$
$y_2^{(l)}$
$y_1^{(l)}$


+1
$r_3^{(l)}$
$b_i^{(l+1)}$
$y_3^{(l)}$
$\tilde{y}_3^{(l)}$
$r_2^{(l)}$
$\mathbf{w}_i^{(l+1)}$
$z_i^{(l+1)}$
f
$y_i^{(l+1)}$
$y_2^{(l)}$
$\tilde{y}_2^{(l)}$
$r_1^{(l)}$
$y_1^{(l)}$
$\tilde{y}_1^{(l)}$

Dropout

- Randomly disable outputs of a layer:

$$r_{jk} \sim \text{Bernoulli}(p)$$

$$v_{j+1}(\mathbf{x}) = f_{j+1}(v_j(\mathbf{x}) \otimes r_j; w_j)$$

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

Dropout

- Prevents co-adaptation of neurons and makes them more robust to random perturbations
- Similar to training 2^n models with shared weights
- During inference: multiply weights of dropout layer by p
- Works well with max-norm regularization
- Variational dropout can learn separate dropout weight for each node

Weight regularization

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

  "We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix"

Weight regularization

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

  "We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix"

Weight regularization

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

  "We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix"

Weight regularization

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

  "We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix"

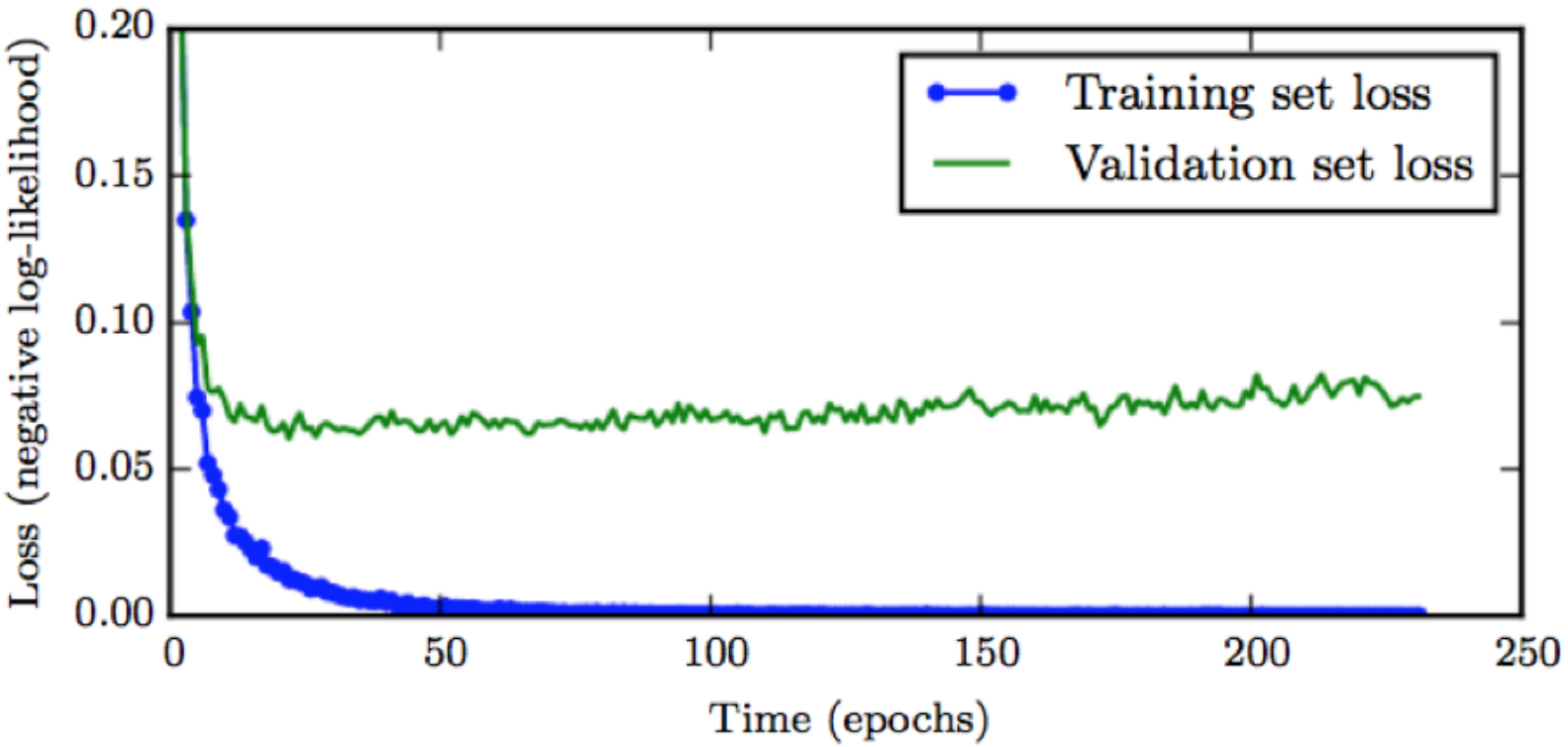
Weight regularization

- L_2 regularization: adds $\lambda\|\mathbf{w}\|^2$ to the loss function
- Penalizes peaky weight vectors
- Prevents neurons form focusing on one strong input
- Something similar to dropout
- Wager, Wang, Liang. Dropout Training as Adaptive Regularization:

  "We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix"

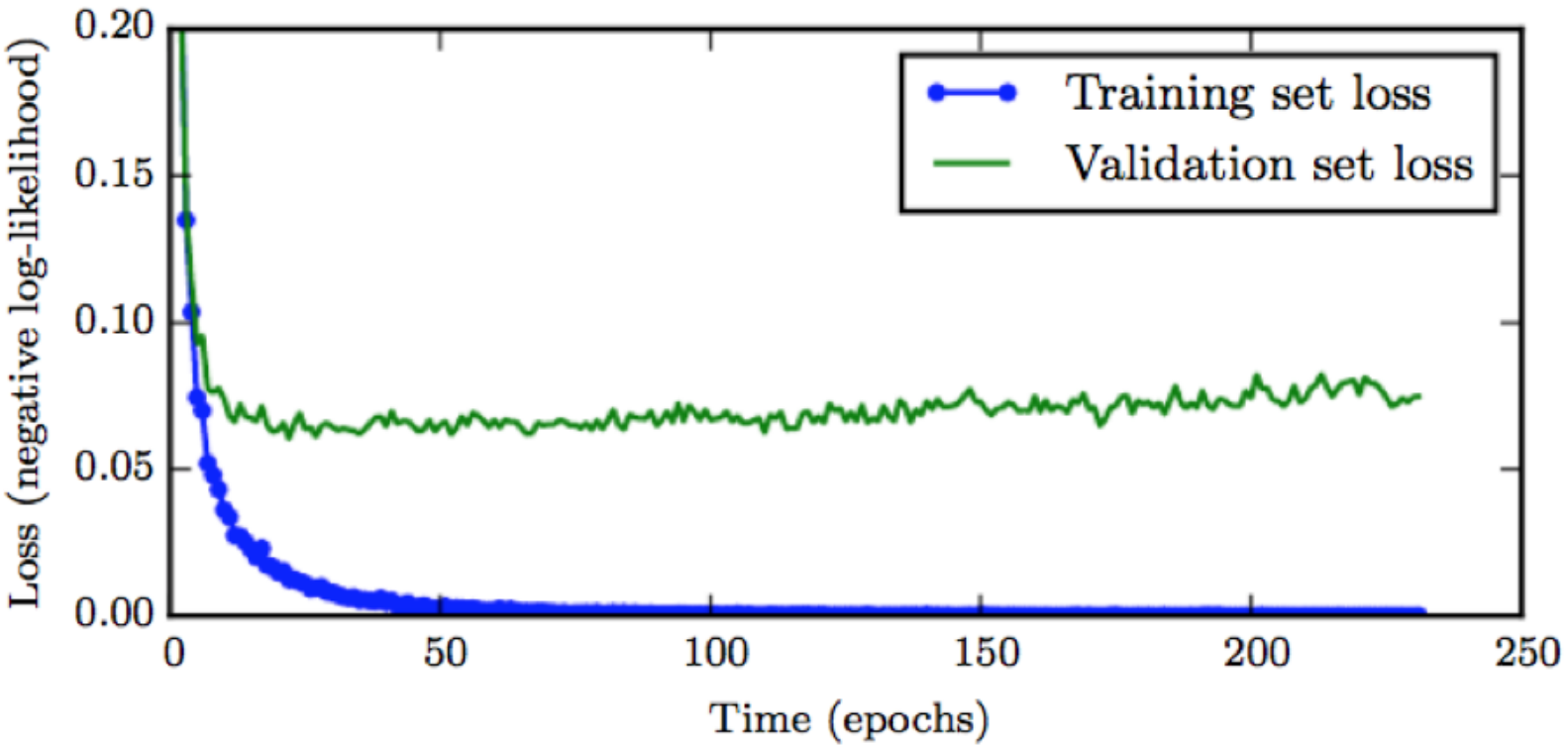
Early stopping

- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization



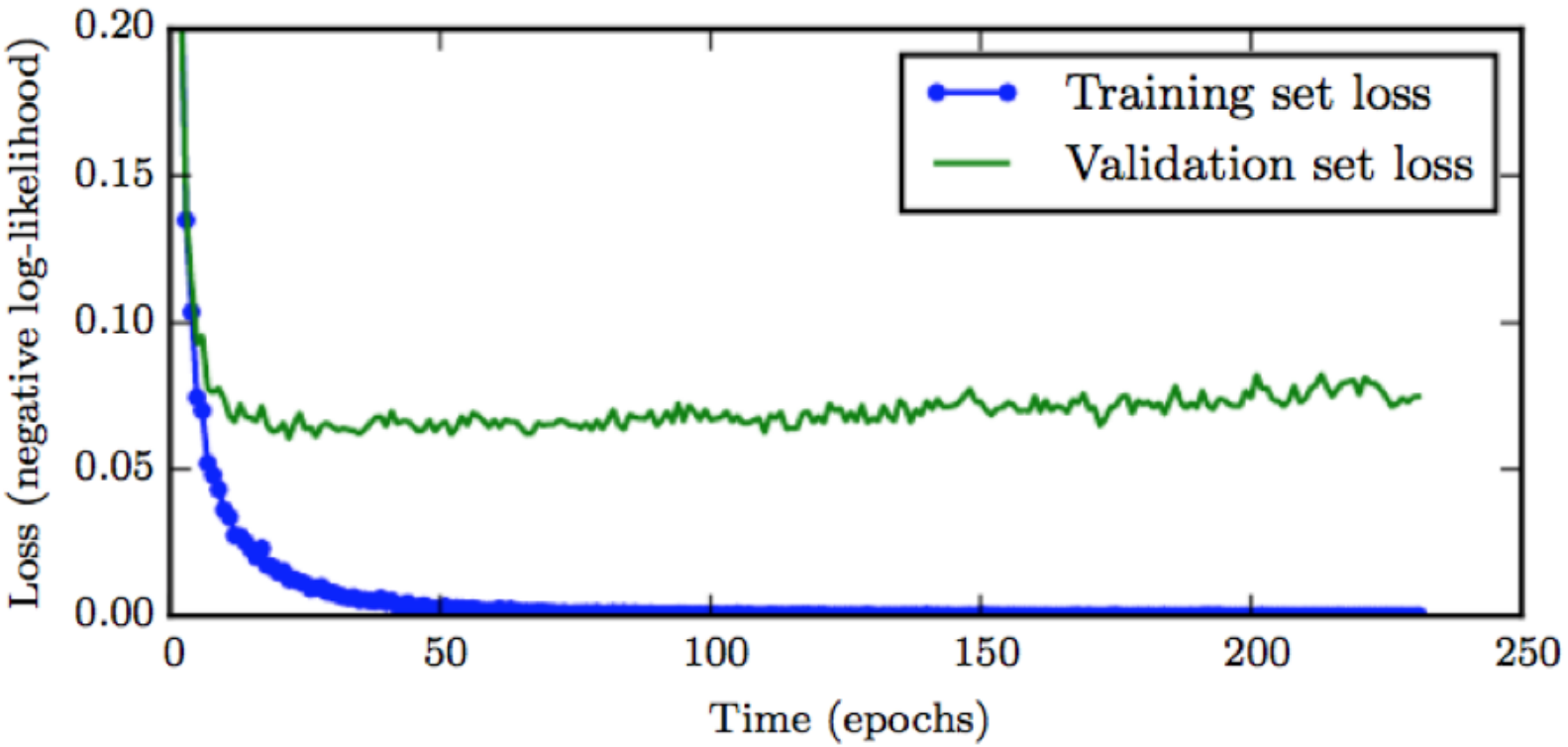
Early stopping

- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization

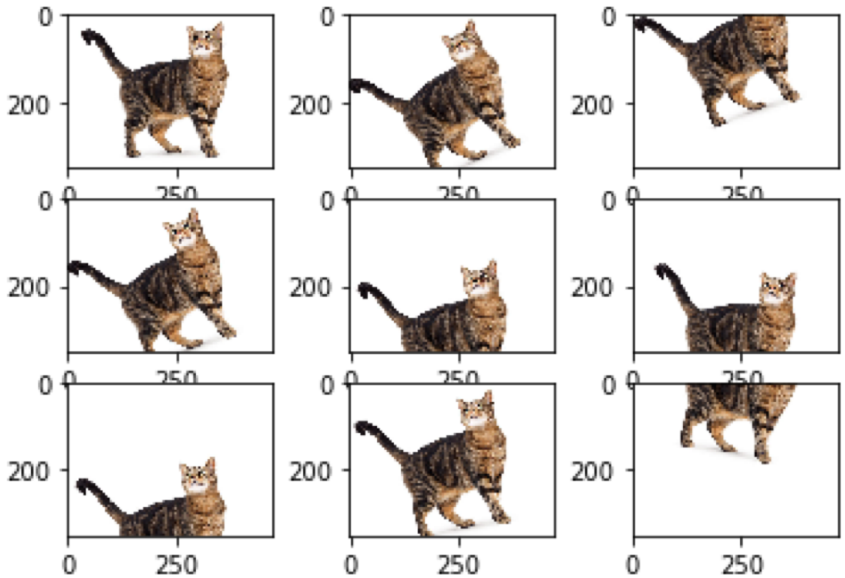


Early stopping

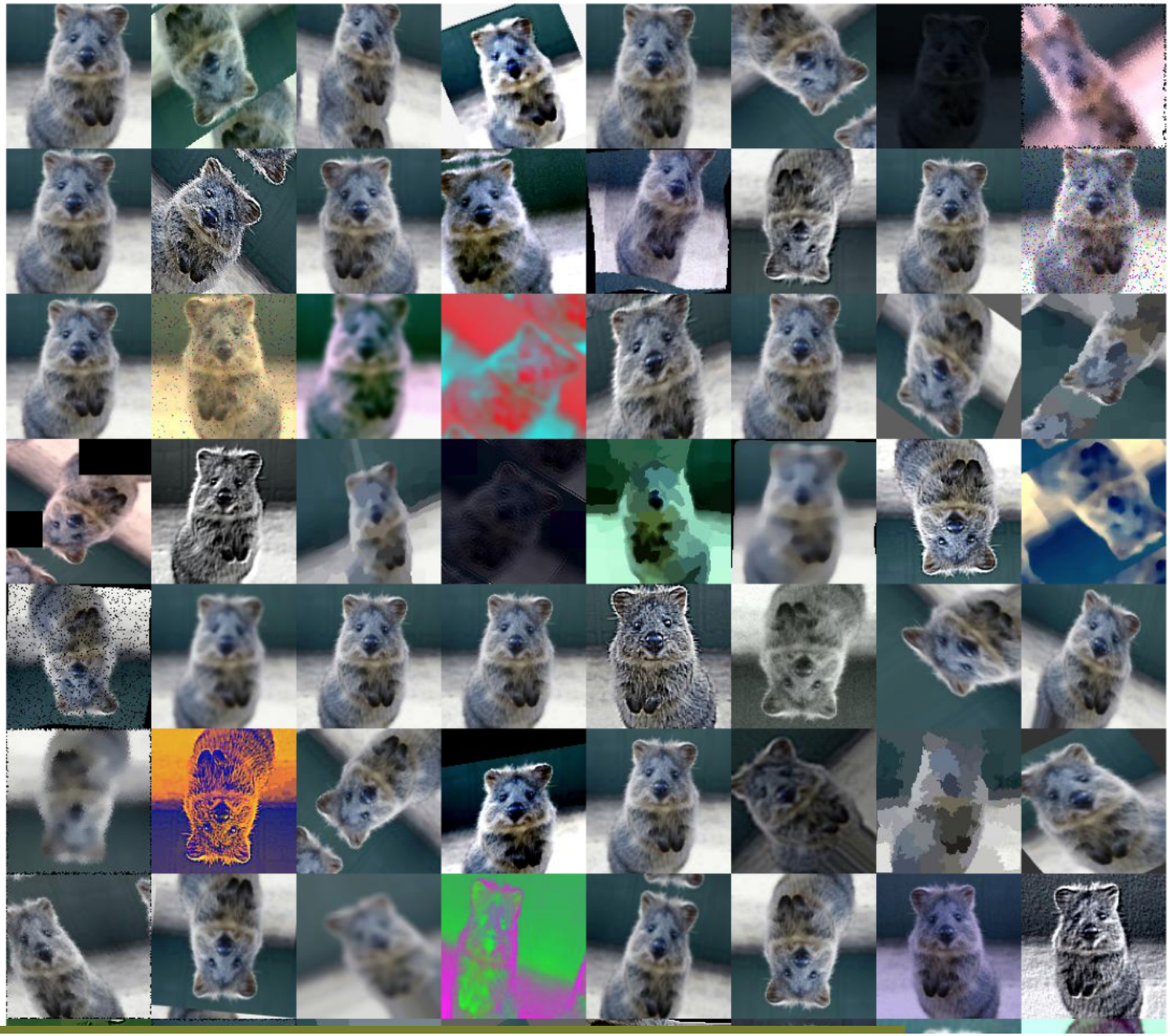
- Select number of optimization steps based on validation quality
- When the optimal number of steps is known, one can retrain the model both on training and validation data
- For a linear model with MSE loss and SGD optimizer, early stopping is equivalent to L_2 regularization



Data augmentation



Data augmentation



Data augmentation for images

- Shift
- Zooming in/out
- Rotation
- Flip
- Distortion
- Shade

Data augmentation

- You can even use style transfer!

Data augmentation for audio

- Time stretching
- Pitch shifting
- Dynamic Range Compression
- Background noise
- ...

Data augmentation

- Improves robustness of the model
- Useful for small samples
- Improves results on large datasets too

Conclusions

- Normalizations improve training both in terms speed and quality
- Regularizations (dropout, max-norm, L_2) prevent co-adaptation
- Data augmentation is useful to improve generalization
- Many other regularization techniques: multi-task learning, adversarial samples, sparsification etc.