Introduction to Statistical Learning and Machine Learning

Chap 7 - Neural Network(Cont.)

Yanwei Fu
SDS, Fudan University

Applications by deep learning

人工智能领域的两股主流

在语音识别上的应用

音素（Phoneme）识别

2009年，Deep belief networks for phone recognition一文中，深度学习的错误率：23.0%

与之比较，不同GMM方法相应错误率：

- Maximum Likelihood Training (MLT)：25.6%,
- Sequence-Discriminative Training (SDT)： 21.7%

单词（Word）识别

2011年， Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition一文中，深度学习的错误率：30.4%

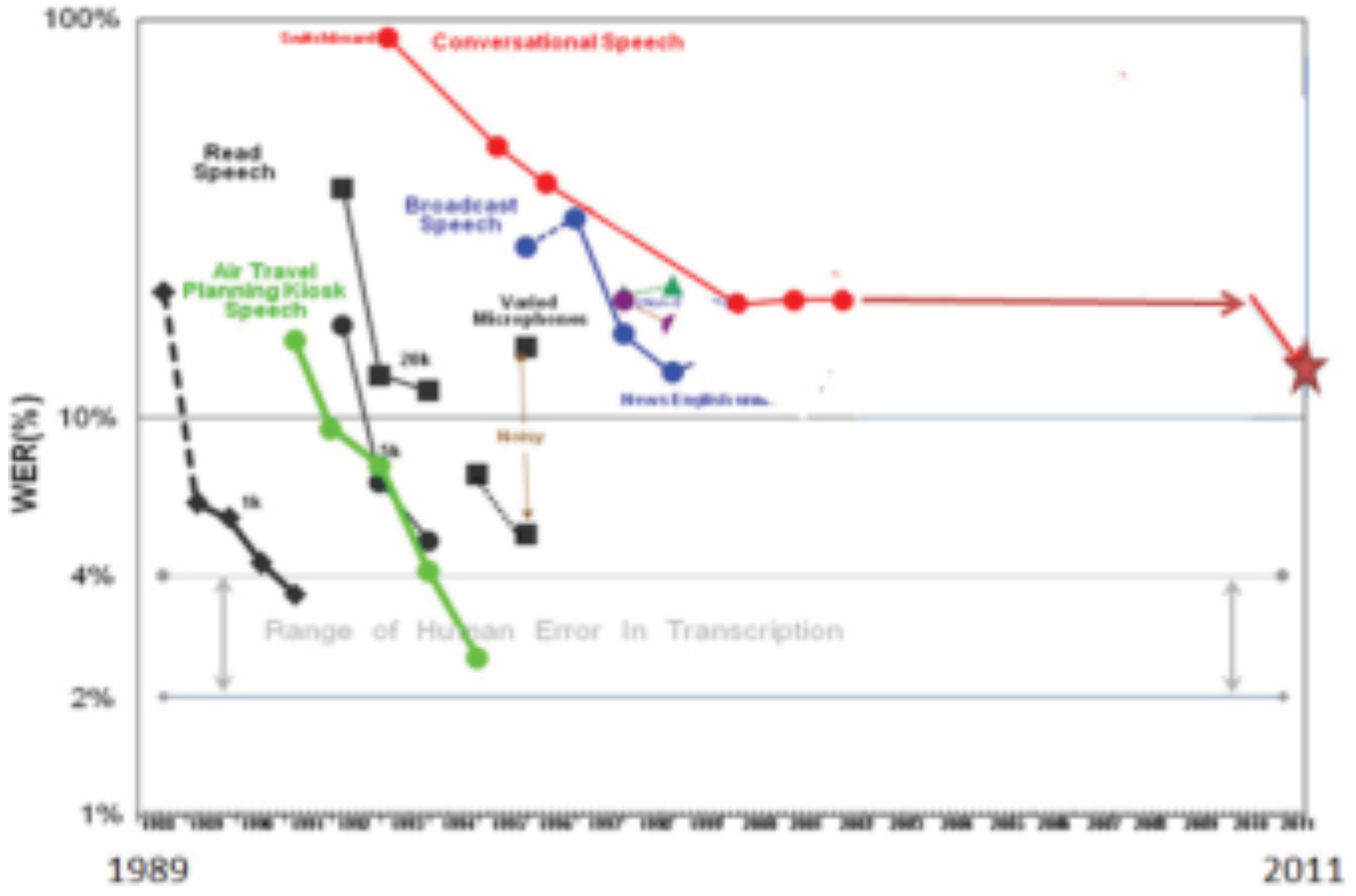
与之比较，不同GMM方法相应错误率：

- Maximum Likelihood Training (MLT)：39.6%,
- Sequence-Discriminative Training (SDT)： 36.2%

在语音识别上的应用

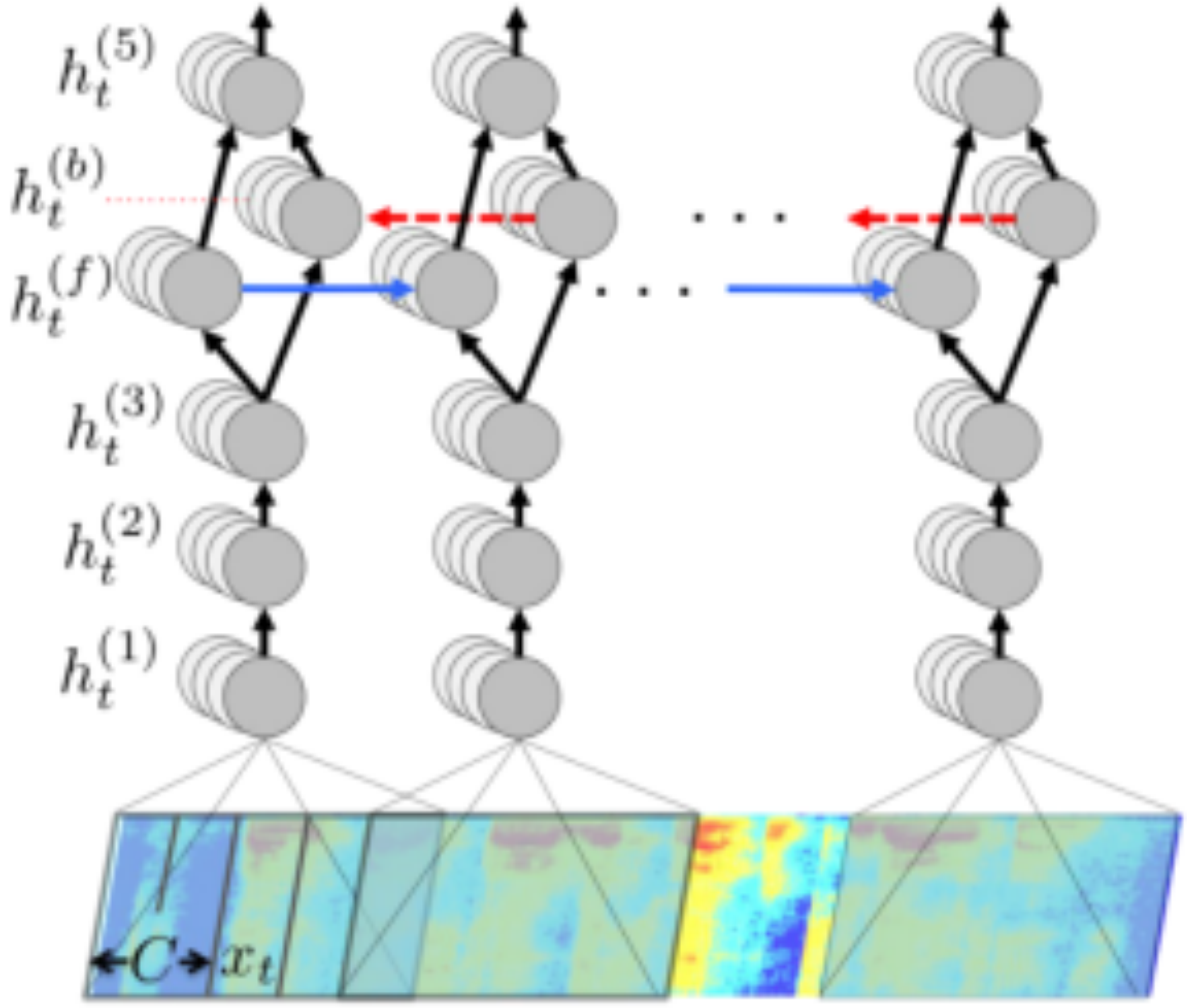
对话识别

2011年基于深度学习取得了十年来的重大突破



2014年百度推出基于RNN的DeepSpeech

在7380小时语音上叠加不同背景噪音生成10万小时级数据



Dataset	Type	Hours	Speakers
WSJ	read	80	280
Switchboard	conversational	300	4000
Fisher	conversational	2000	23000
Baidu	read	5000	9600

Model	SWB	CH	Full
Vesely et al. (GMM-HMM BMMI) [43]	18.6	33.0	25.8
Vesely et al. (DNN-HMM sMBR) [43]	12.6	24.1	18.4
Maas et al. (DNN-HMM SWB) [28]	14.6	26.3	20.5
Maas et al. (DNN-HMM FSH) [28]	16.0	23.7	19.9
Seide et al. (CD-DNN) [39]	16.1	n/a	n/a
Kingsbury et al. (DNN-HMM sMBR HF) [22]	13.3	n/a	n/a
Sainath et al. (CNN-HMM) [36]	**11.5**	n/a	n/a
DeepSpeech SWB	20.0	31.8	25.9
DeepSpeech SWB + FSH	13.1	**19.9**	**16.5**

在图像识别上的应用

IMAGENET大规模视觉识别挑战赛（ILSVRC 2014）

物体识别项目，15M图片，22K类

名称	时间	Top-5 Error
AlexNet	2012年	15.3%
OverFeat（New York University）	2013年	13.8%
VGG Net（Oxford）	2014年	7.3%
GoogLeNet（Google）	2014年	6.6%
人类	/	5.1%
Microsoft	2015年2月6日	4.94%
Google	2015年2月11日	4.82%
Microsoft	2015年12月10日	3.57%
Google	2015年12月11日	3.58%
Google	2016年2月23日	3.08%

在图像识别上的应用

人脸识别

LFW（5749个人，13233张人脸照片）

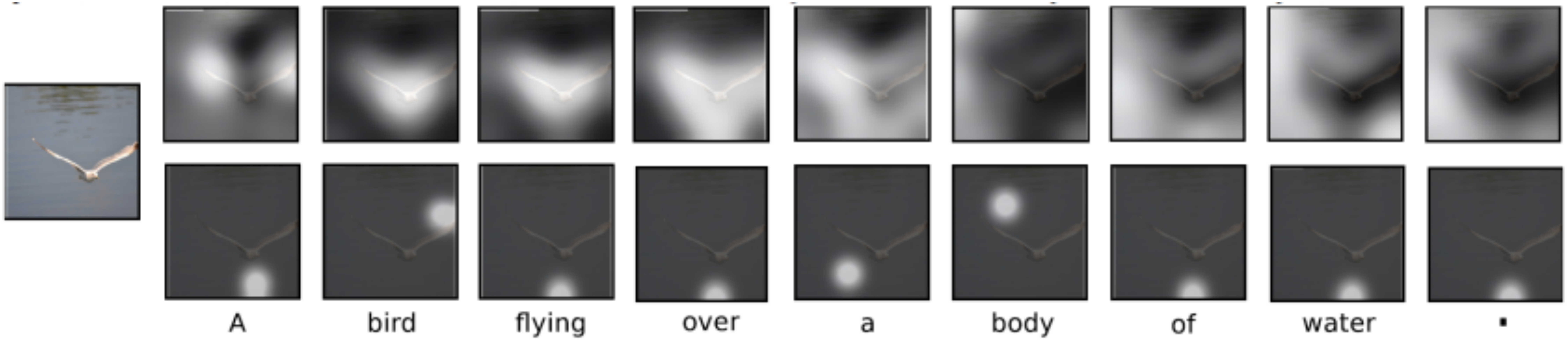
名称	时间	Top-1 Accuracy
传统方法	/	~96%
DeepFace（Facebook）	2014年	97.35%
人类	/	97.53%
GaussianFace（香港中文大学）	2014年	98.52%
DeepID3（香港中文大学）	2015年2月	99.53%
Facenet（Google）	2015年6月	99.63%
腾讯优图	2015年10月	99.65%
百度IDL	2015年10月	99.77%

Youtube Face DB（8M个人，200M张人脸照片）
FaceNet（Google）识别率可达95.12%（2015年）

在图像识别上的应用

关注度（Attention）

Yoshua Bengio团队，2016年



A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

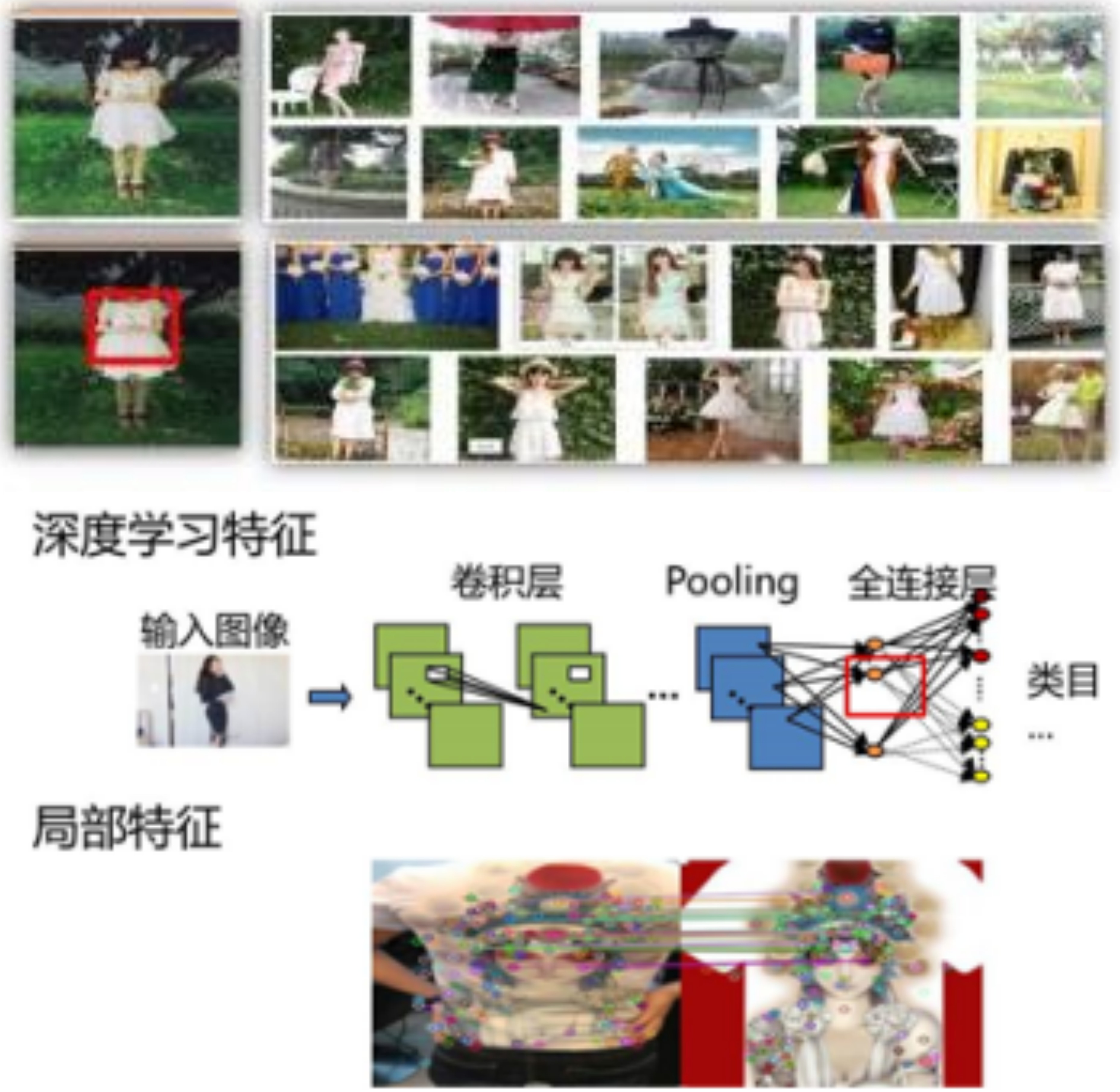
A giraffe standing in a forest with trees in the background.

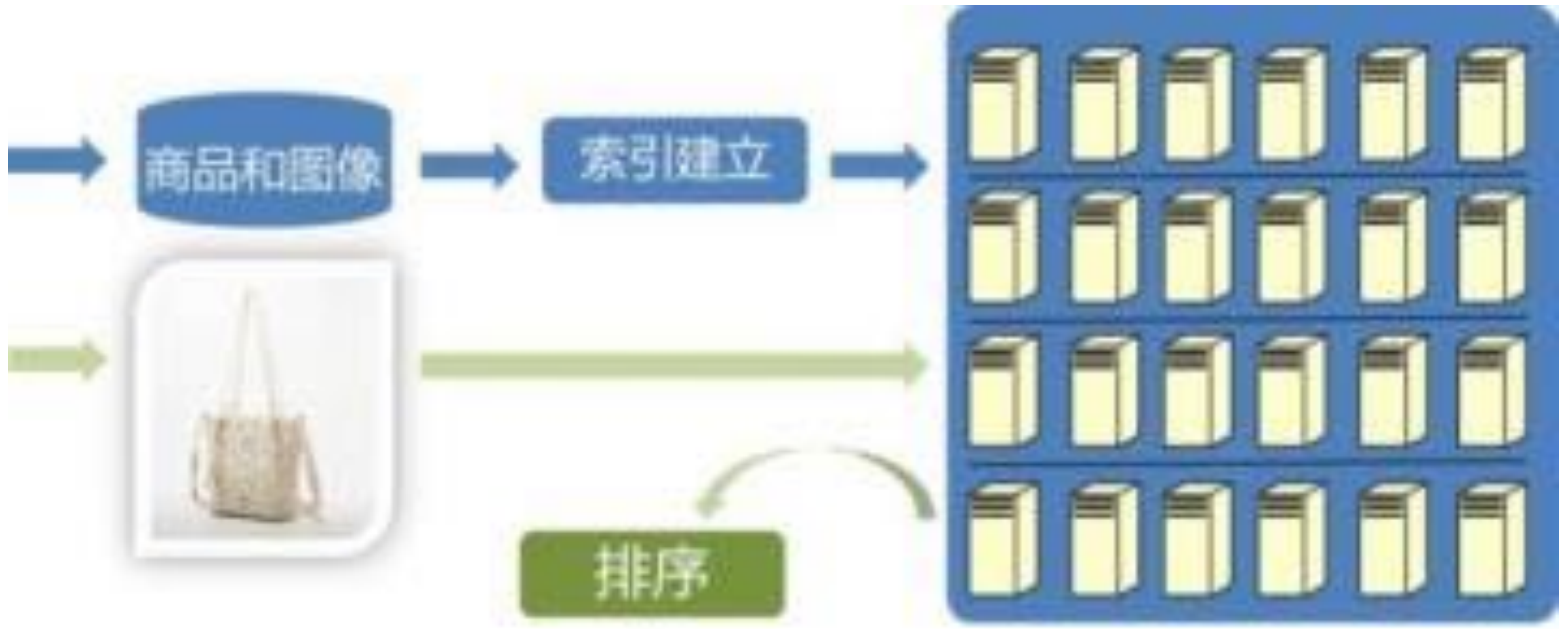
A woman is throwing a frisbee in a park.

在图像识别上的应用

海量图像的分类、识别

拍立淘

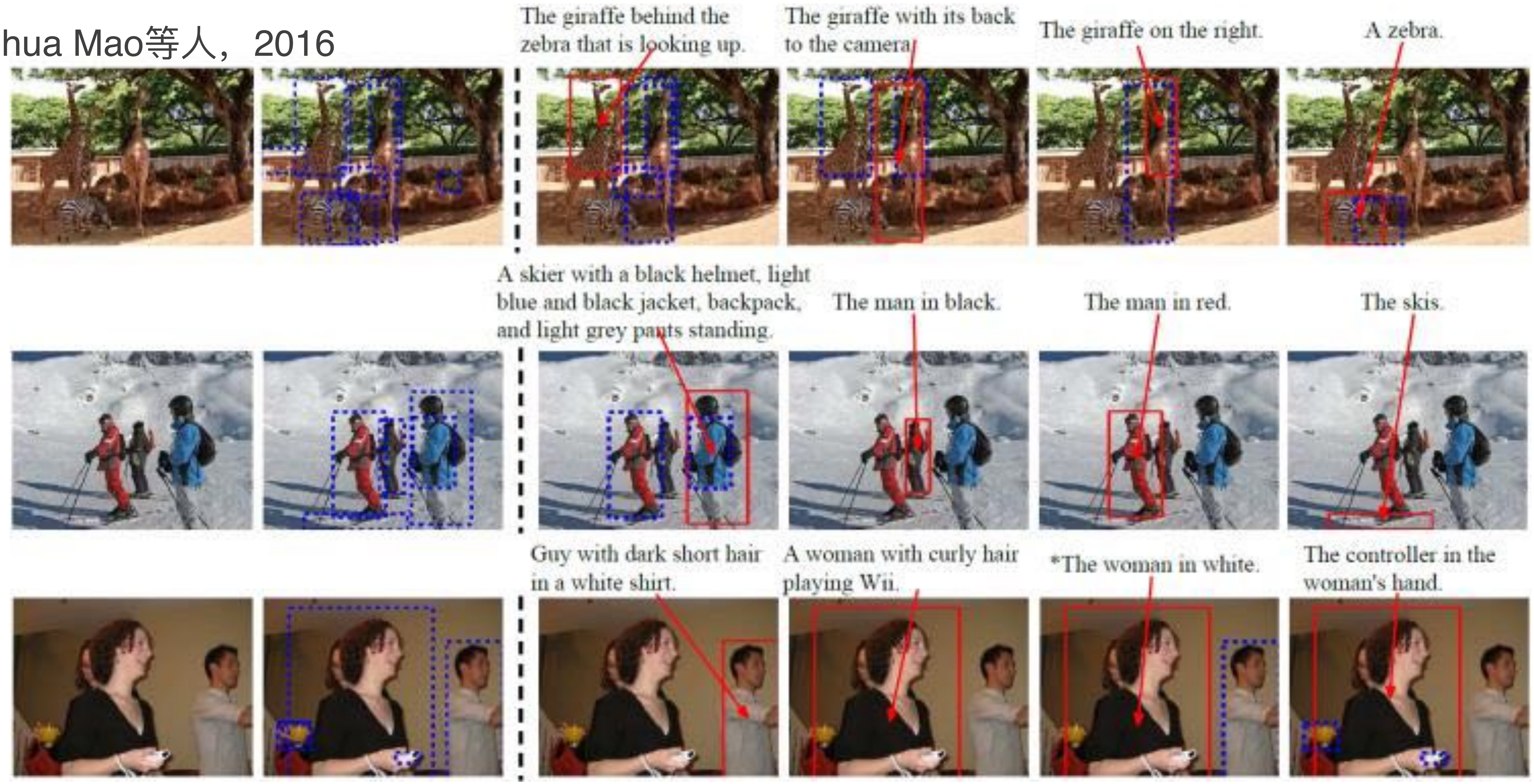




在图像识别上的应用

图像描述

Junhua Mao等人，2016

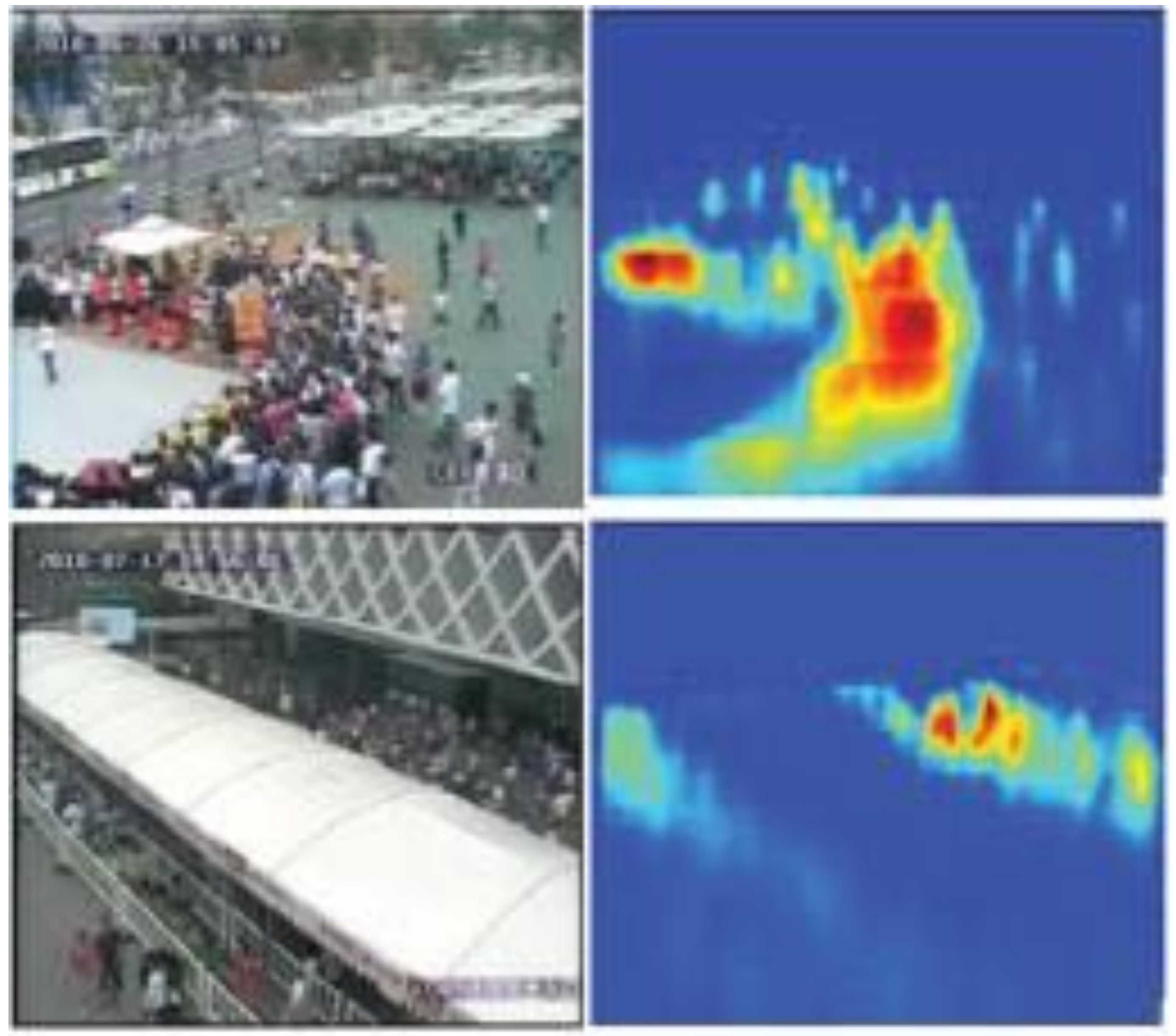


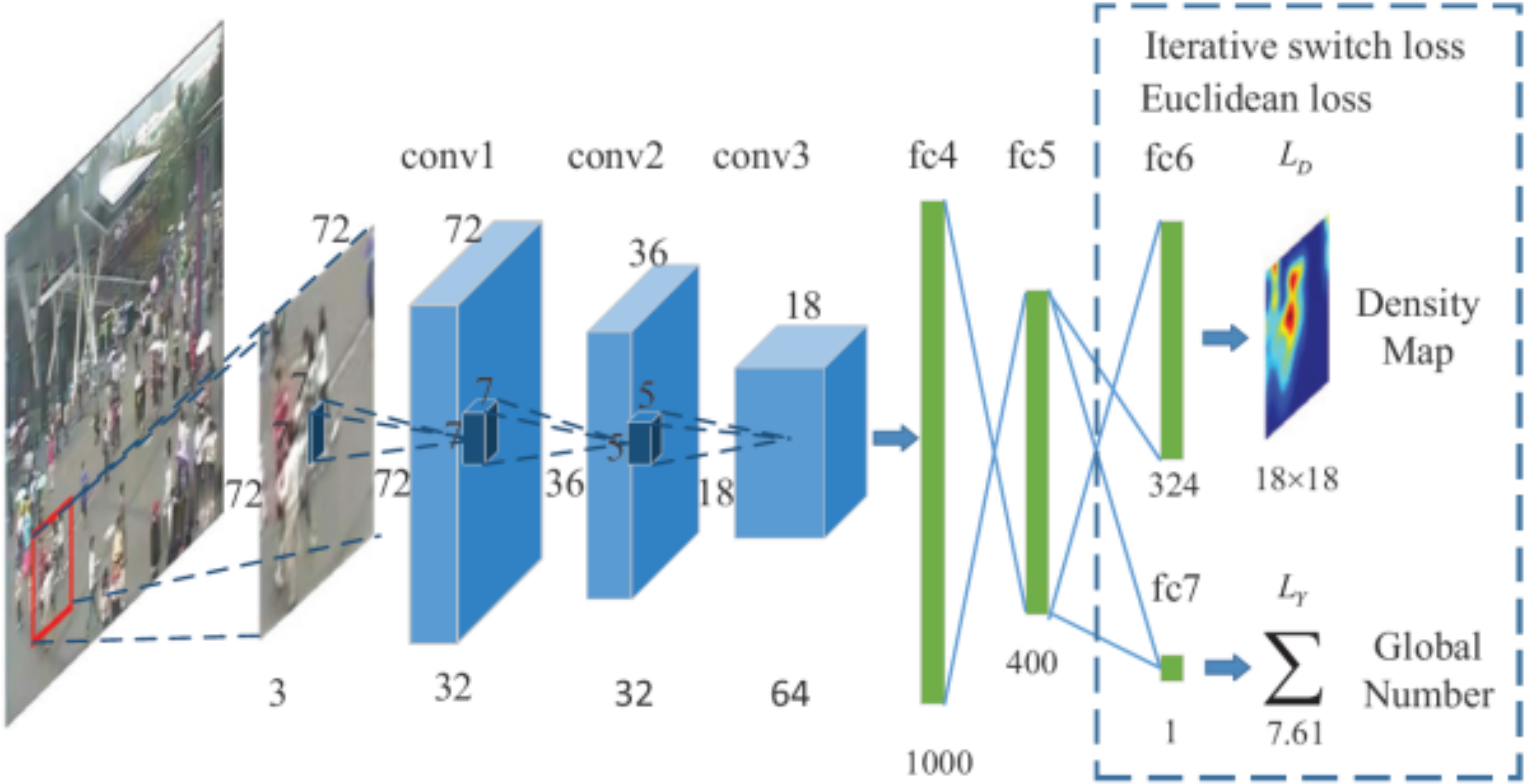
在图像识别上的应用

人群计数

Cong Zhang等人，2016

Method	Scene 1	Scene 2	Scene 3	Scene 4	Scene 5	Average
LBP+RR	13.6	58.9	37.1	21.8	23.4	31.0
Crowd CNN	10.0	15.4	15.3	25.6	4.1	14.1
Fine-tuned Crowd CNN	9.8	14.1	14.3	22.2	3.7	12.9
Luca Fiaschi et al. [7]	2.2	87.3	22.2	16.4	5.4	26.7
Ke et al. [6]	2.1	55.9	9.6	11.3	3.4	16.5
Crowd CNN+RR	2.0	29.5	9.7	9.3	3.1	10.7

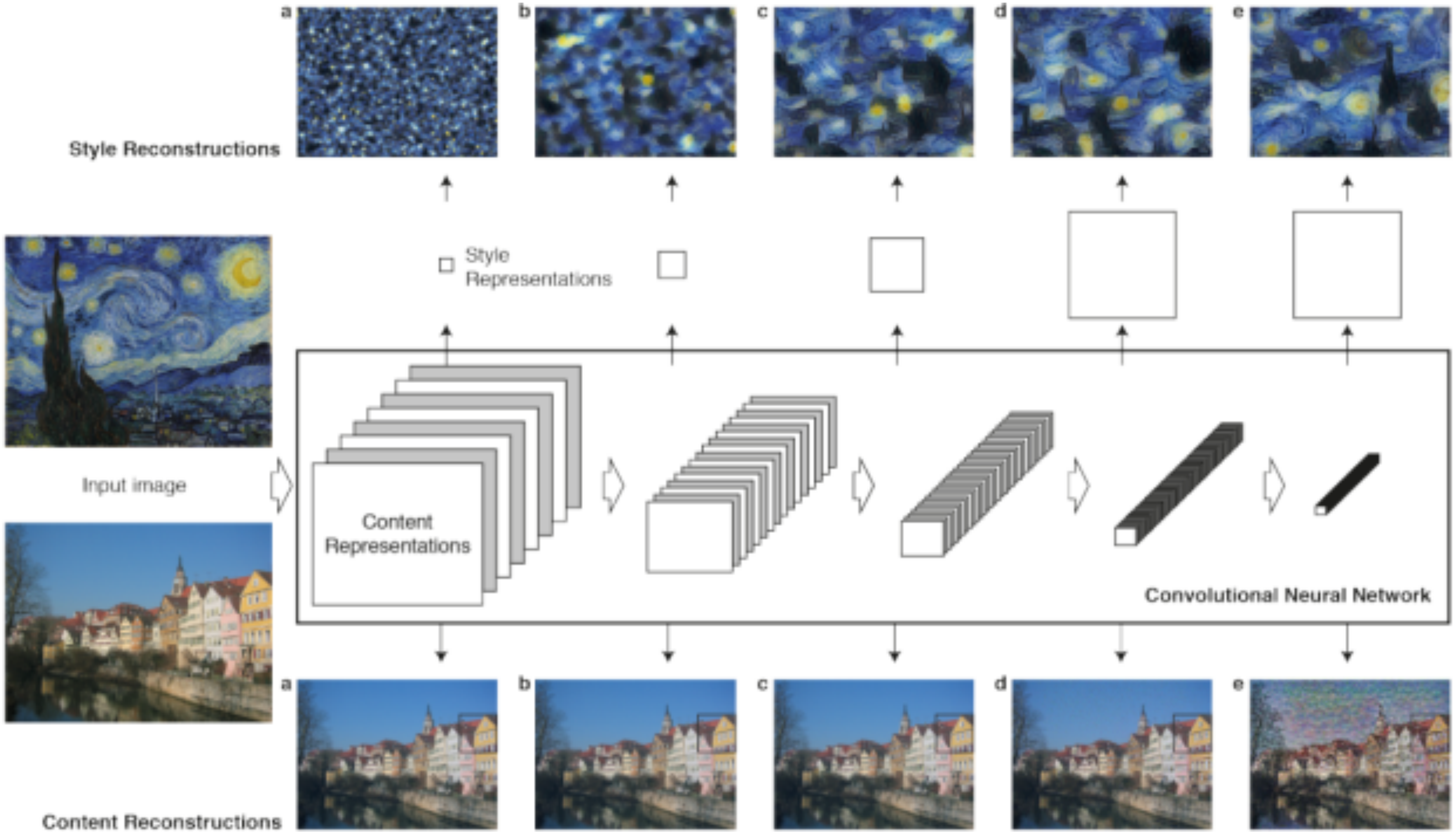




在图像处理上的应用

绘画风格变换

Leon A. Gatys等人，2015

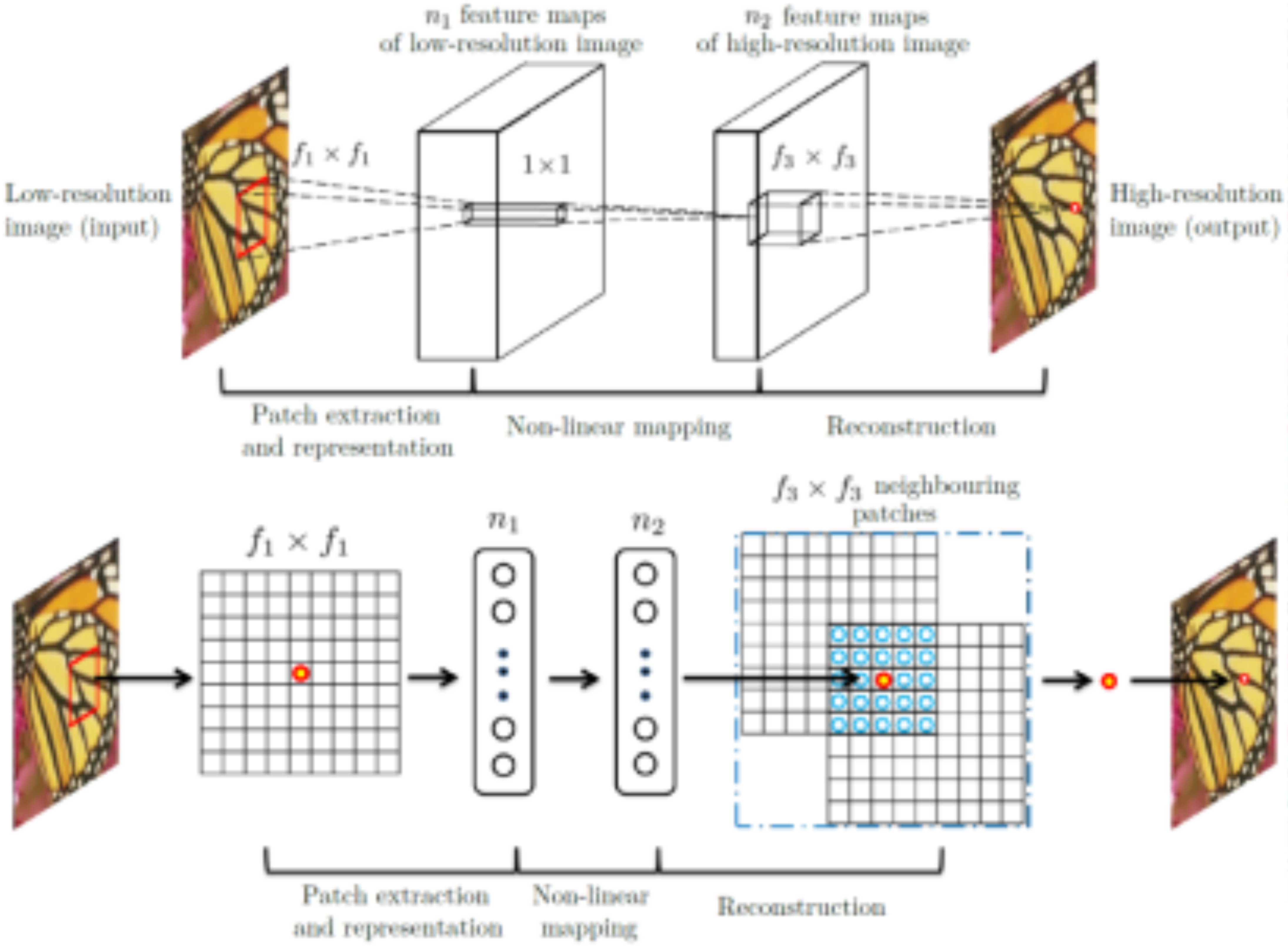


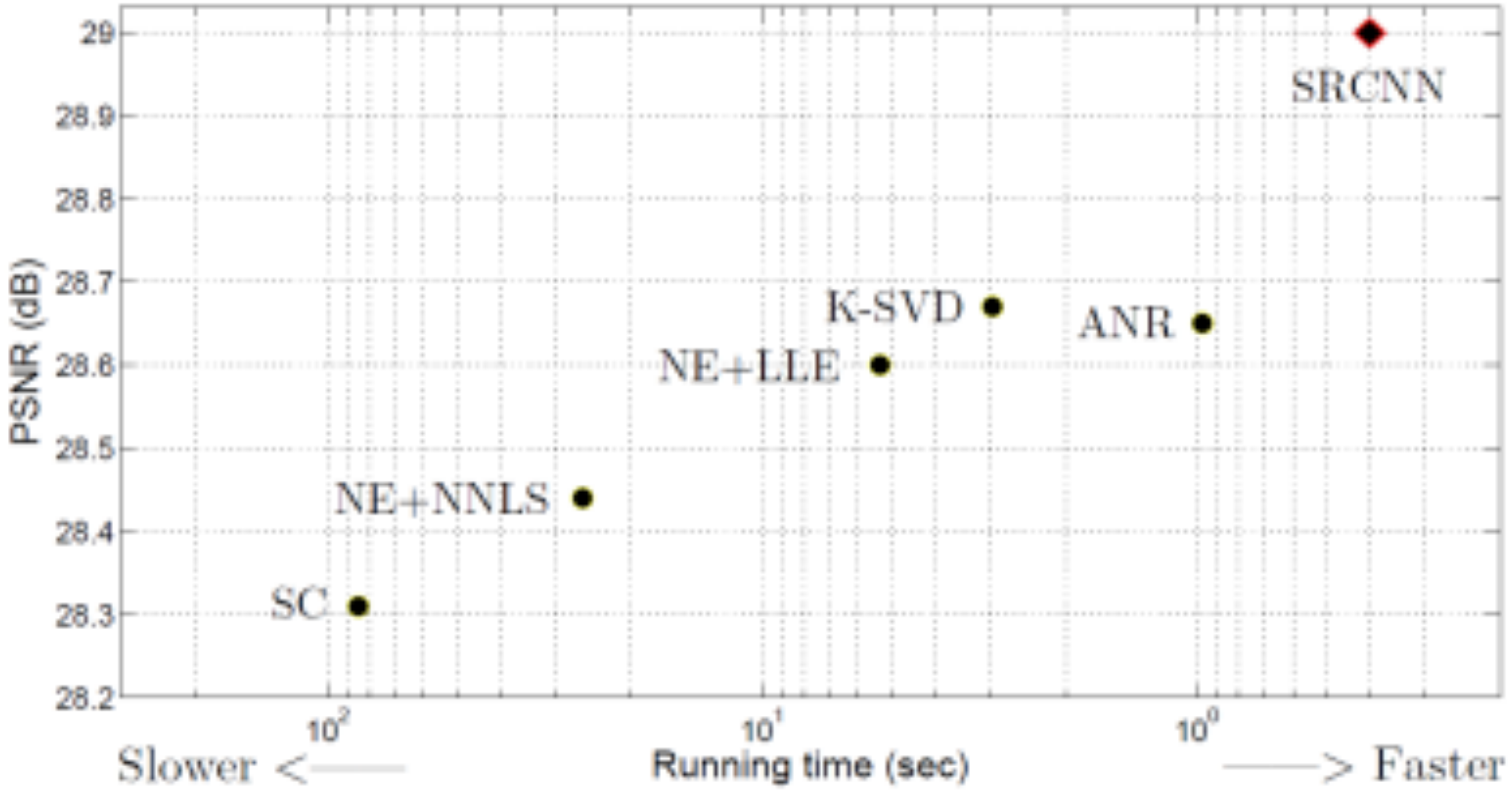
在图像处理上的应用

超分辨率

2014年Xiaoou Tang等人的工作

信噪比高、速度快



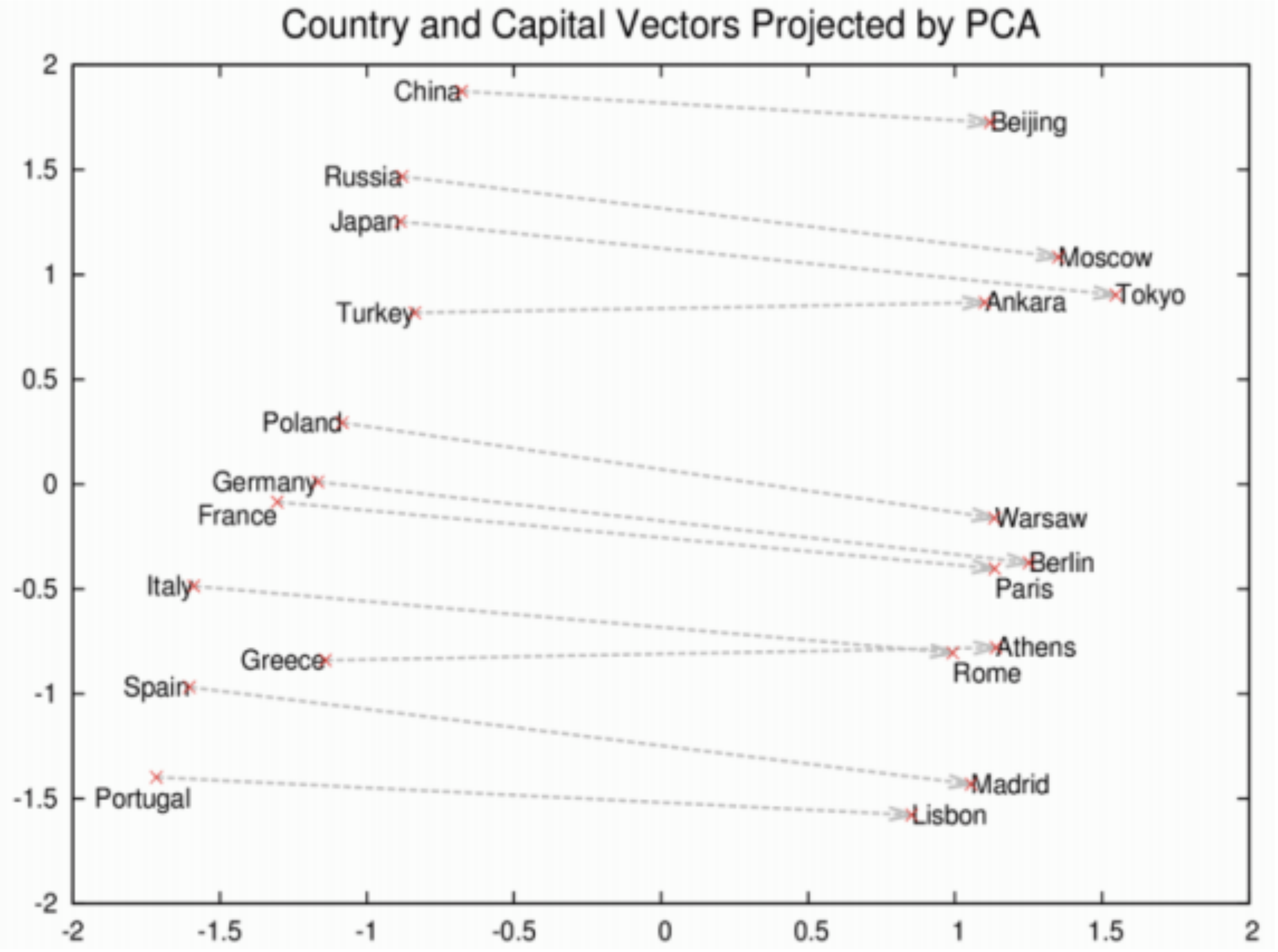


在自然语言理解上的应用

Word2Vec的适时出现

词语获得了更稠密的向量表示
词语的相关性更容易计算（余弦距离）
深度学习具备了重要的输入

输入：计算机	
自动化	0.674172
应用	0.614087
自动化系	0.611133
材料科学	0.607891
集成电路	0.600370
技术	0.597519
电子学	0.591316
建模	0.577239
工程学	0.572856
微电子	0.570087



在自然语言理解上的应用

定制化的NLP应用

将过去统计机器翻译的成熟成果迁移到神经网络模型上
基于深度学习的情感分析
利用神经网络模型检测小说中的人物关系

从文本理解到文本生成

新闻、专利、百科词条、论文的生成
智能人机对话系统

大规模知识图谱的构建与应用

阅读理解、机器翻译、文档摘要
新概念、新知识的自动学习
基于知识图谱实现智能推理

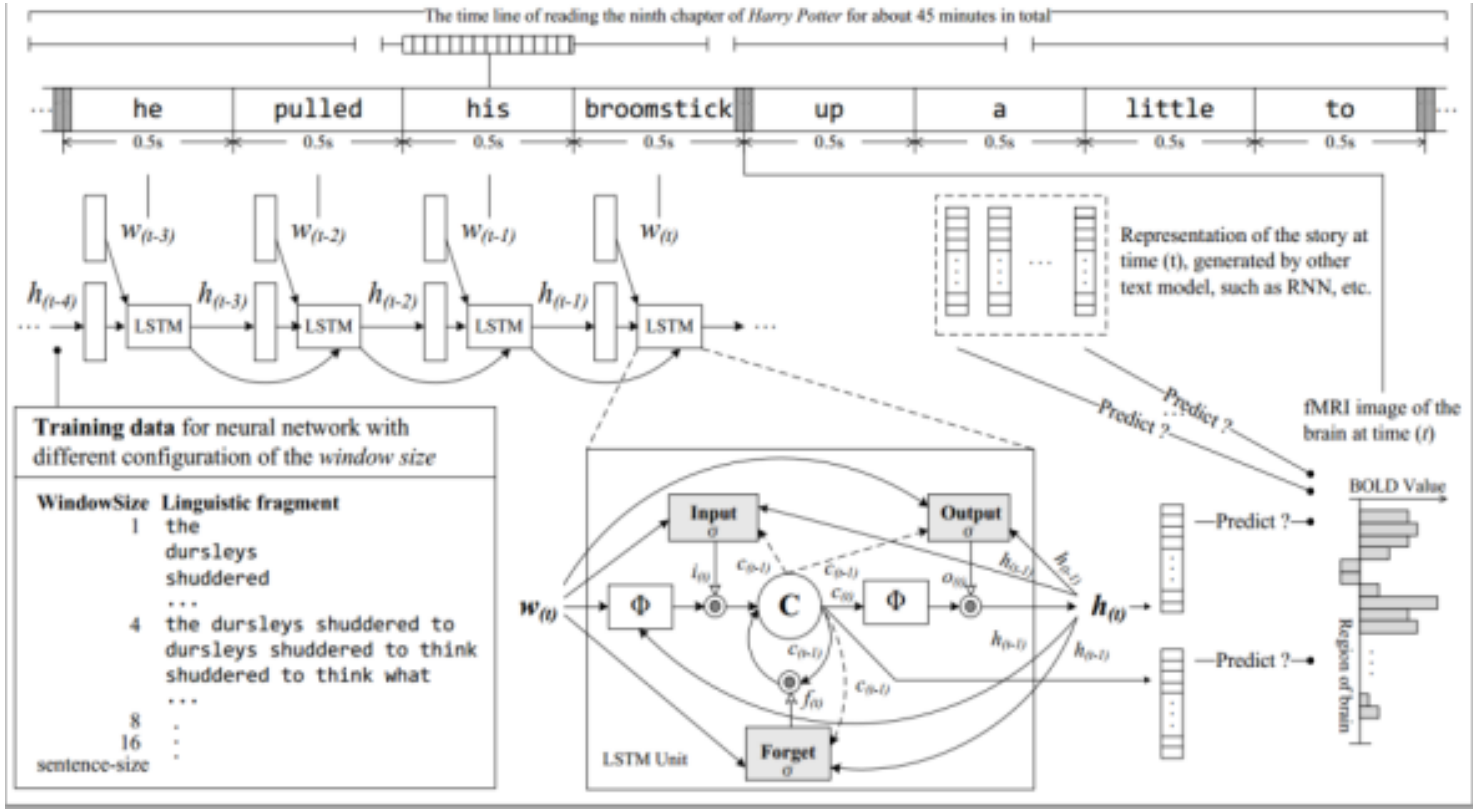
在自然语言理解上的应用

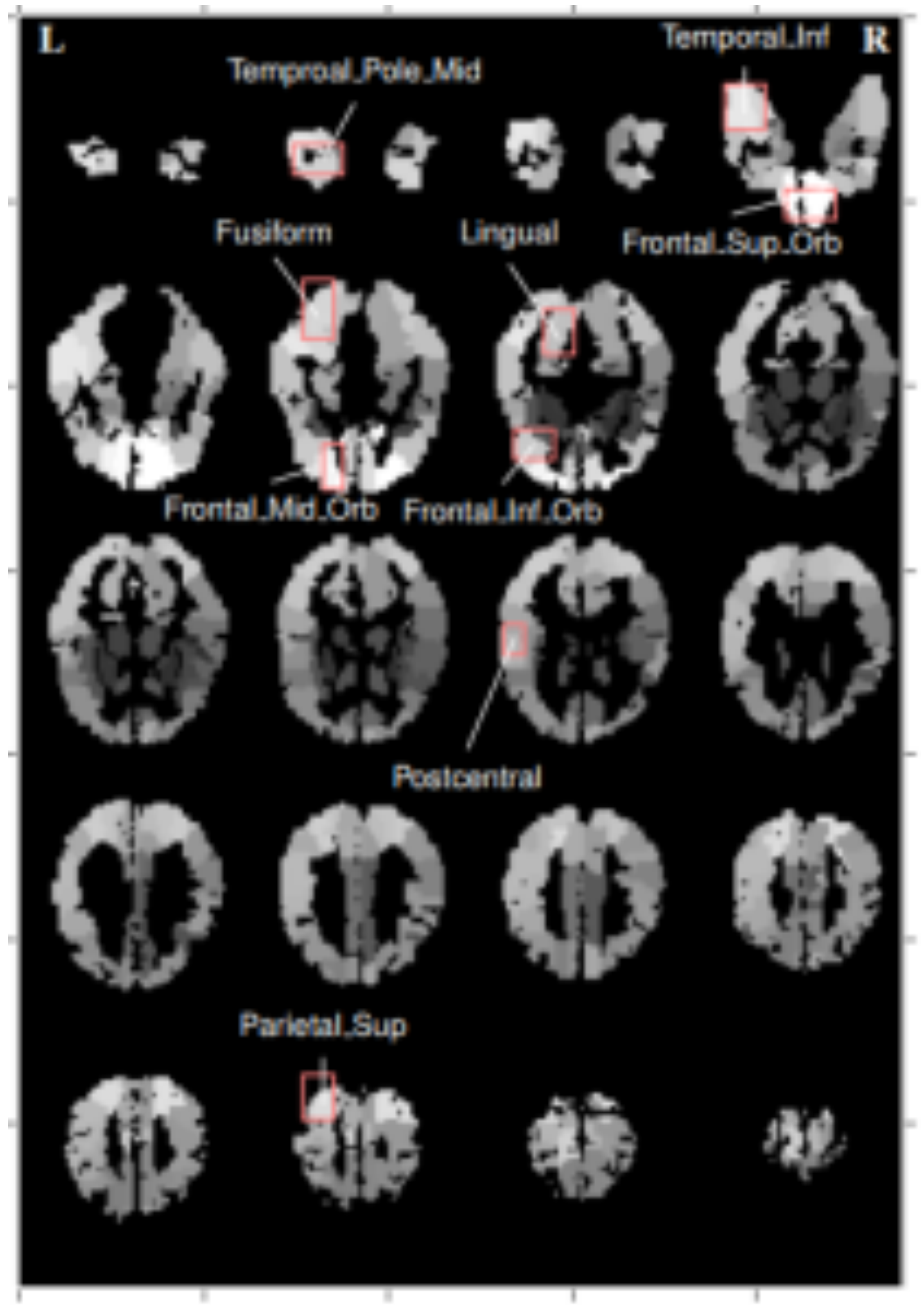
LSTM架构的认知解释

人阅读和机器阅读时的神经元活动是否可以相互预测?

LSTM架构在认知角度是否合理?

Model	Cosine Dist.	Similarity
Random	-0.128	0.436
BoW(tf-idf)	0.184	0.592
AveEmbedding	0.634	0.817
RNN_hidden	0.016	0.508
LSTM_hidden	0.224	0.612
LSTM_memory	**0.724**	**0.862**





在围棋上的应用

AlphaGo

目前在GoRating上已经超越柯洁、李世石等人排名世界第一。
在目标确定、规则明确的任务中，（弱）人工智能击败人类是必然的

在……省电上的应用

Google DeepMind

用于操控计算机服务器和相关设备（例如冷却系统）来管理部分数据中心，从而减少**15%**能耗

2014年总能耗
4,402,836兆瓦时

366,903个美国家庭x1年

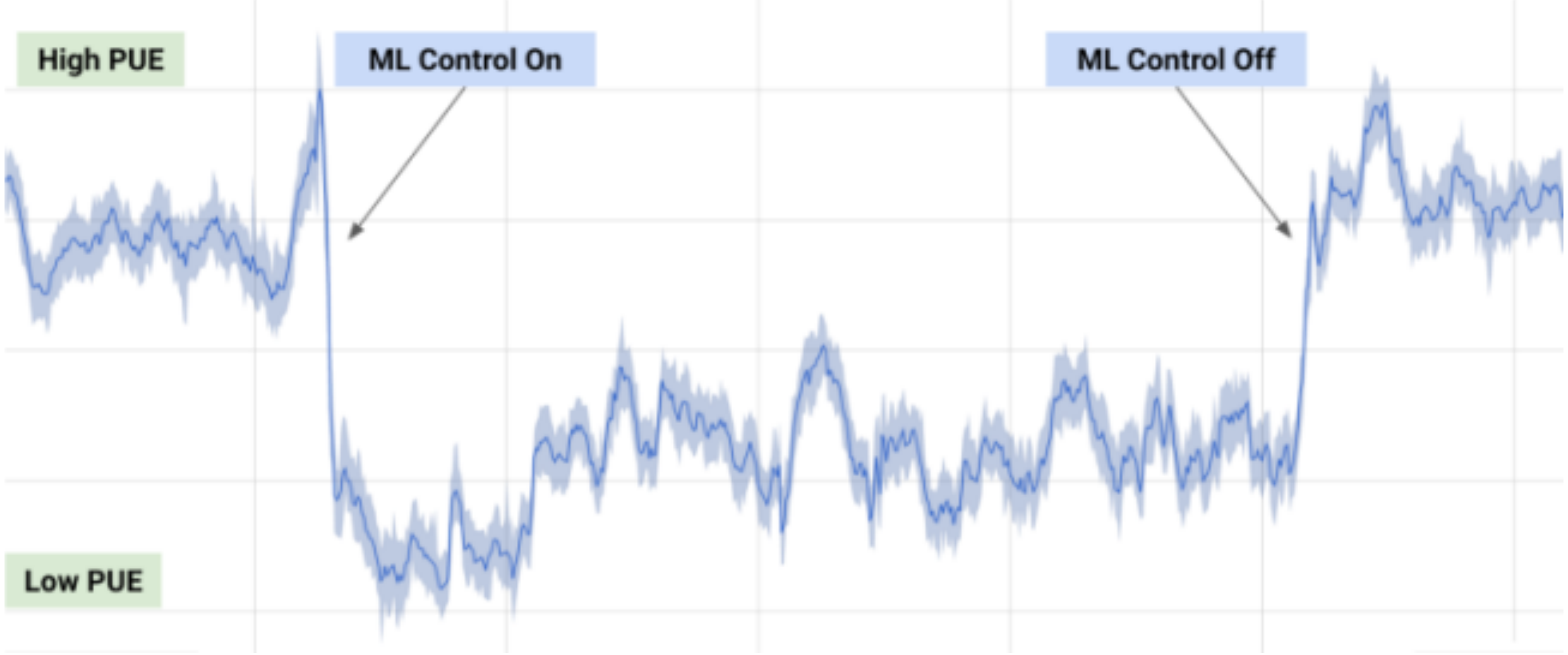
商用电价
25美元至40美元/兆瓦时

总计可节省16,500,000-26,500,000美元/年

在……省电上的应用

Google DeepMind

用于操控计算机服务器和相关设备（例如冷却系统）来管理部分数据中心，从而减少**15%**能耗

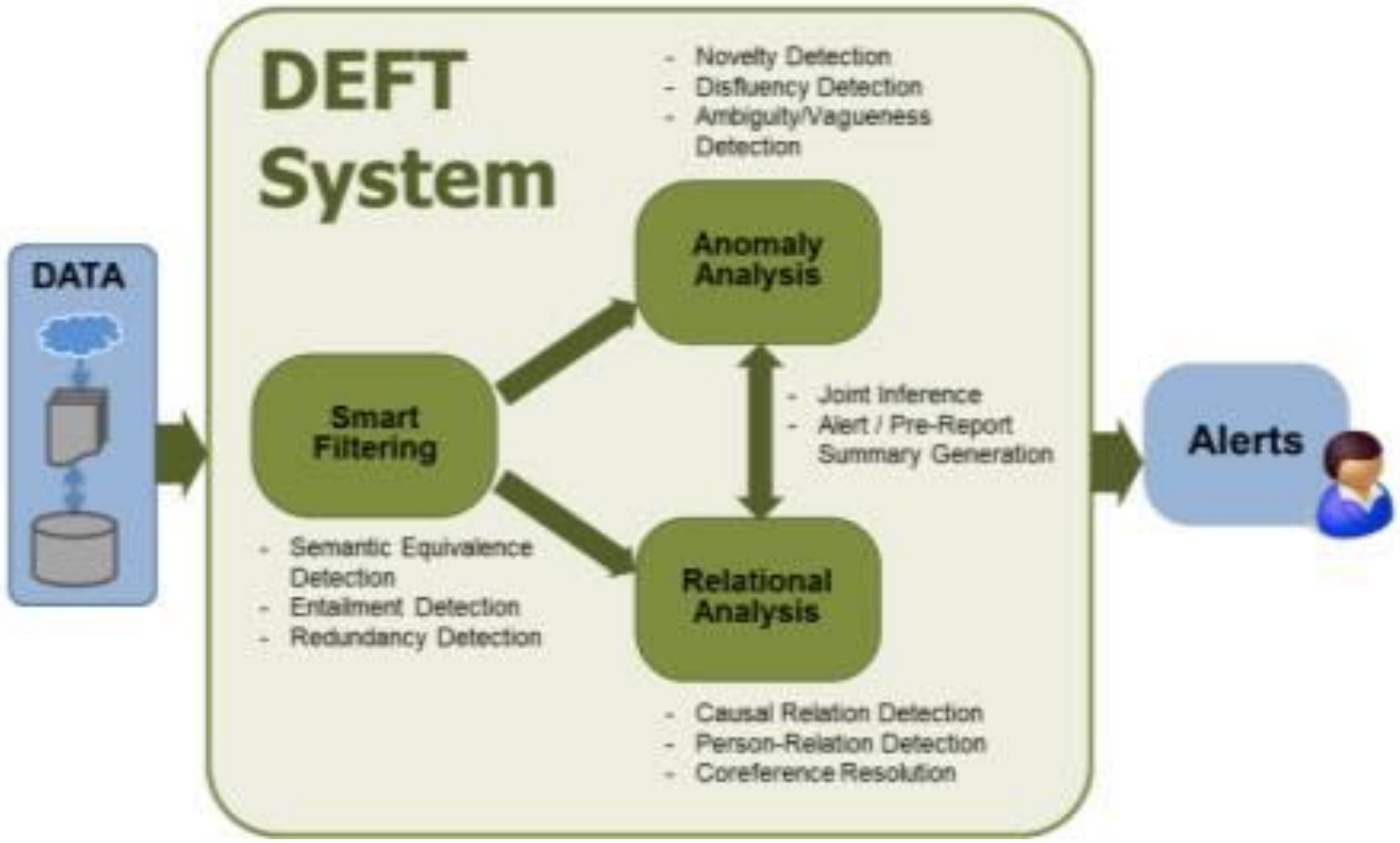


在军事领域的应用

美国军方早已开展相关研究与应用

2009年DARPA已着手撰写关于深度学习的报告，2010年起开始资助相关项目

2012年资助DEFT项目（Deep Exploration and Filtering of Text），对海量文本数据进行分析



2015年资助TRACE项目（Target Recognition and Adaption in Contested Environments），对图像中的目标进行识别

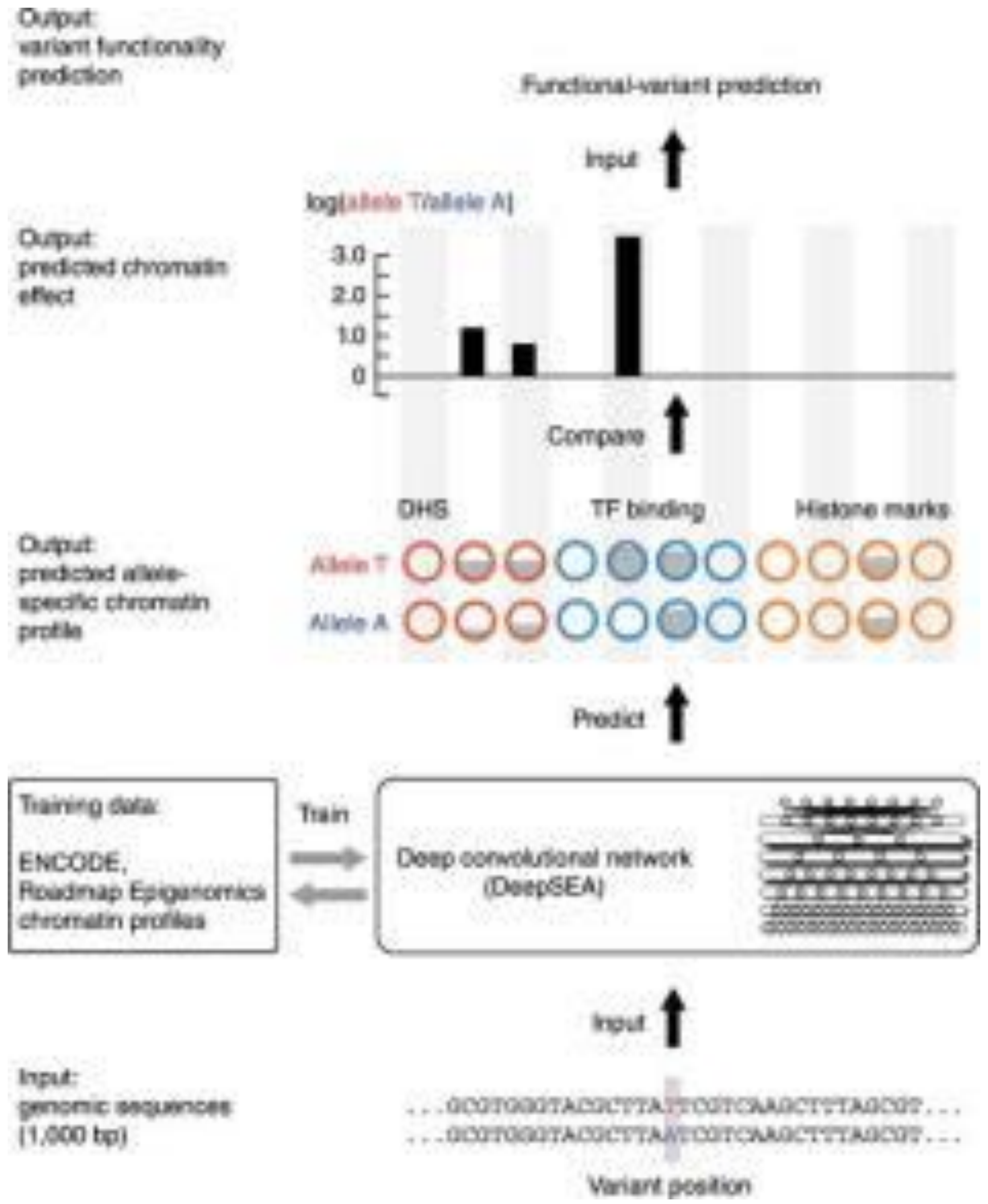
在医疗领域的应用

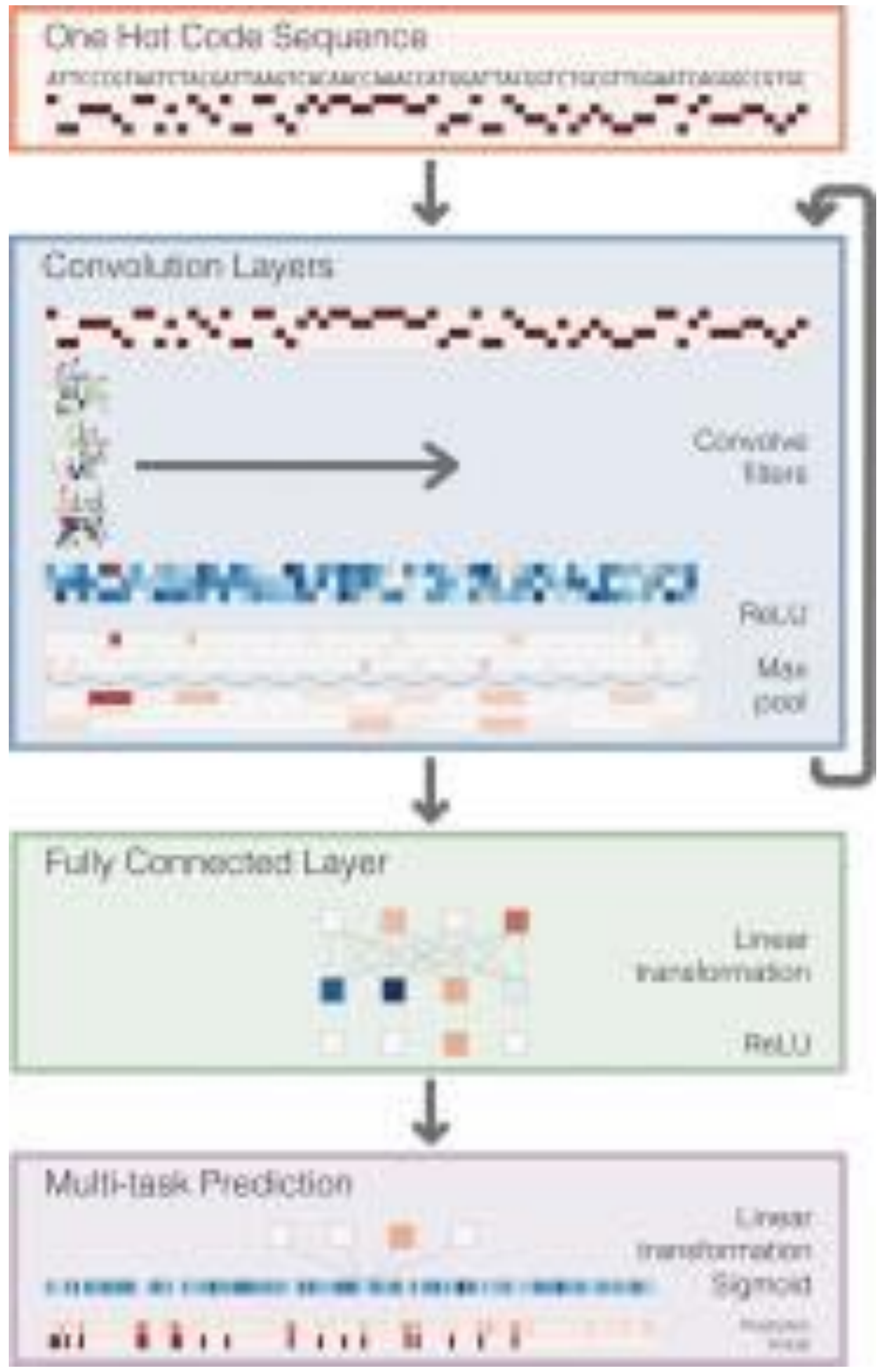
多种分析技术已经在DNA分析、癌症预测等方面产生影响

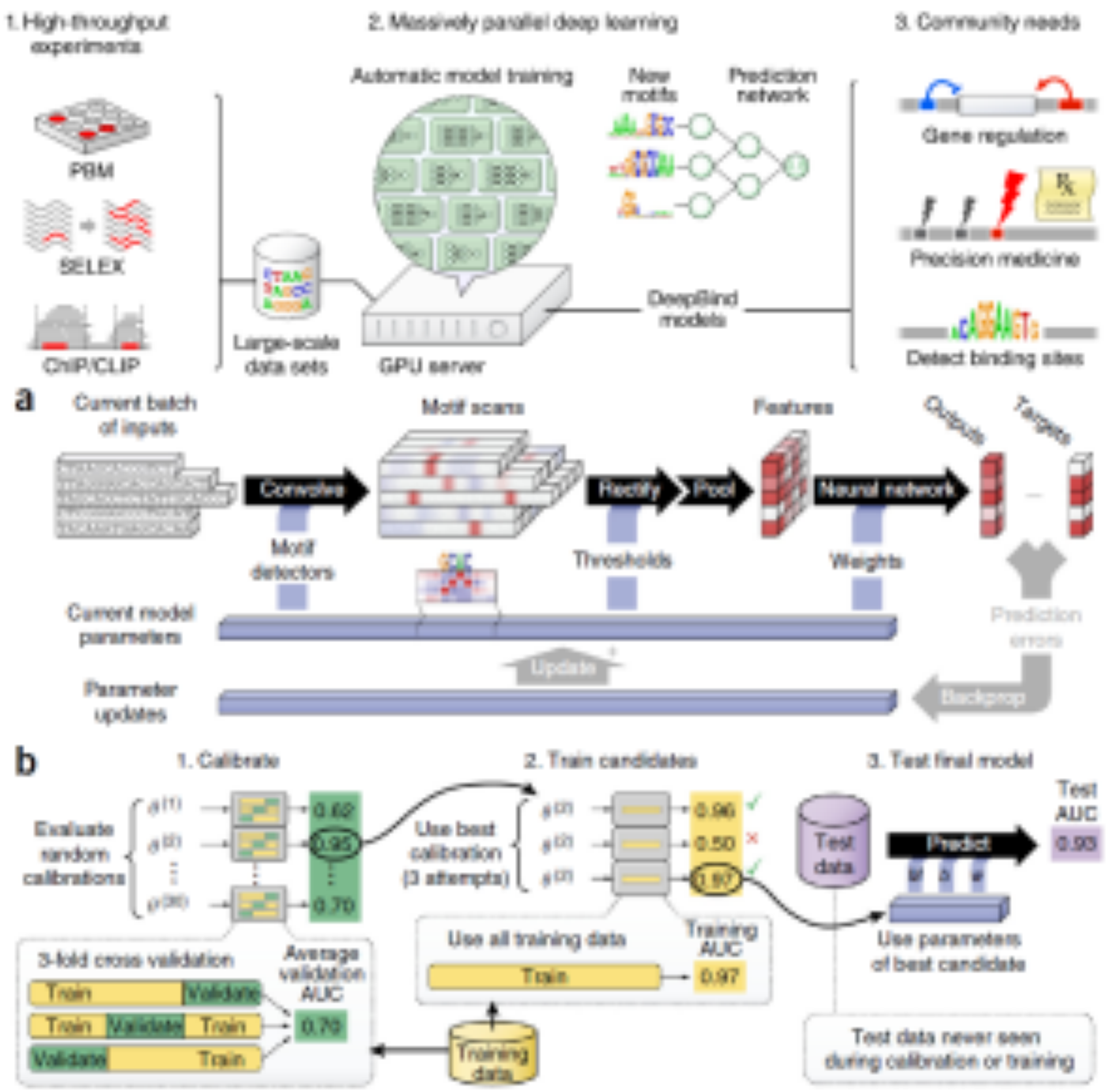
Princeton大学的DeepSEA可预测重要调控位点对单核苷酸变异的影响

Harvard大学的Basset可预测单核苷酸多态性对染色质可接近性的影响

Toronto大学的DeepBind 能发现RNA与DNA上的蛋白结合位点，预测突变的影响



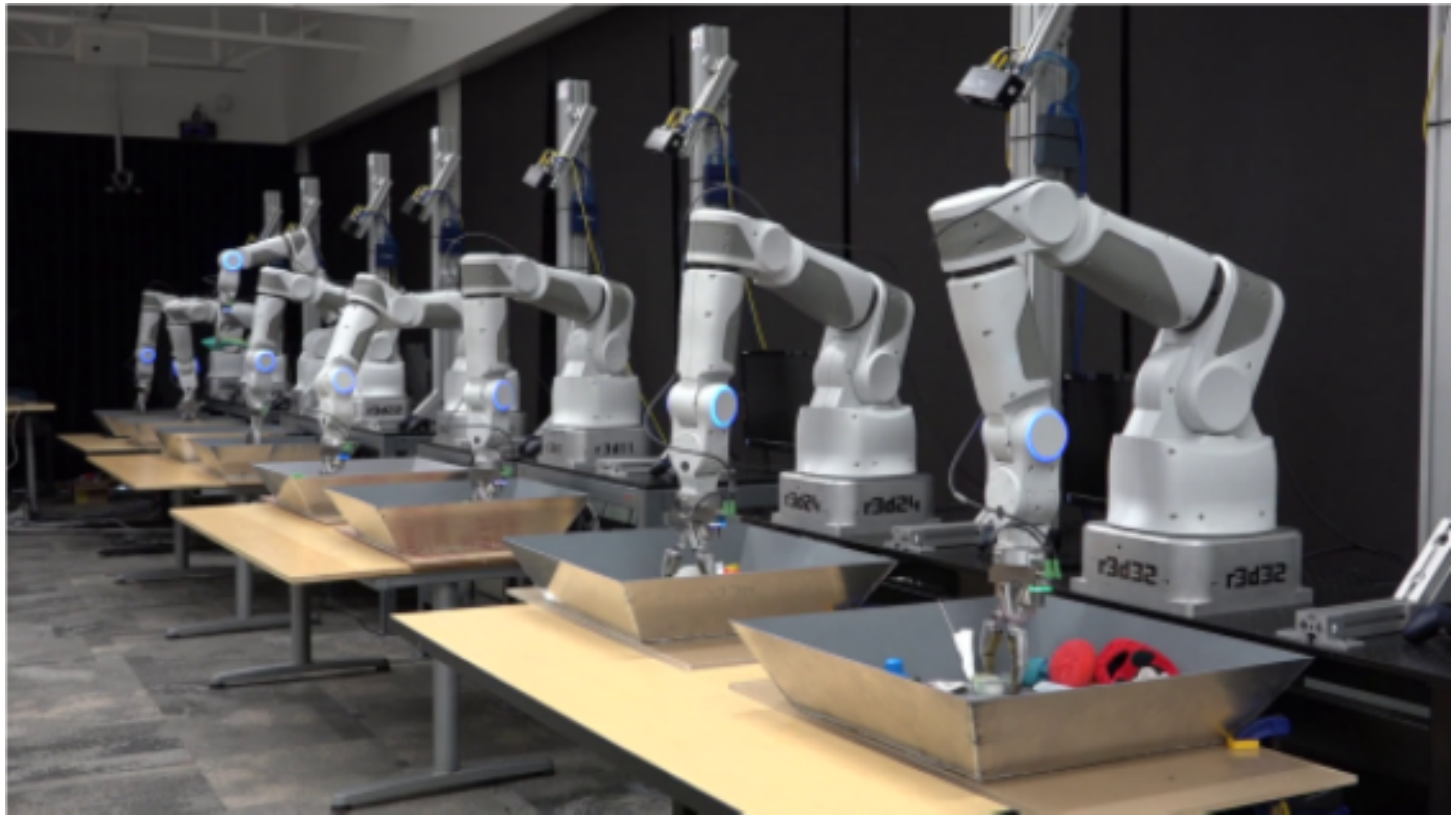


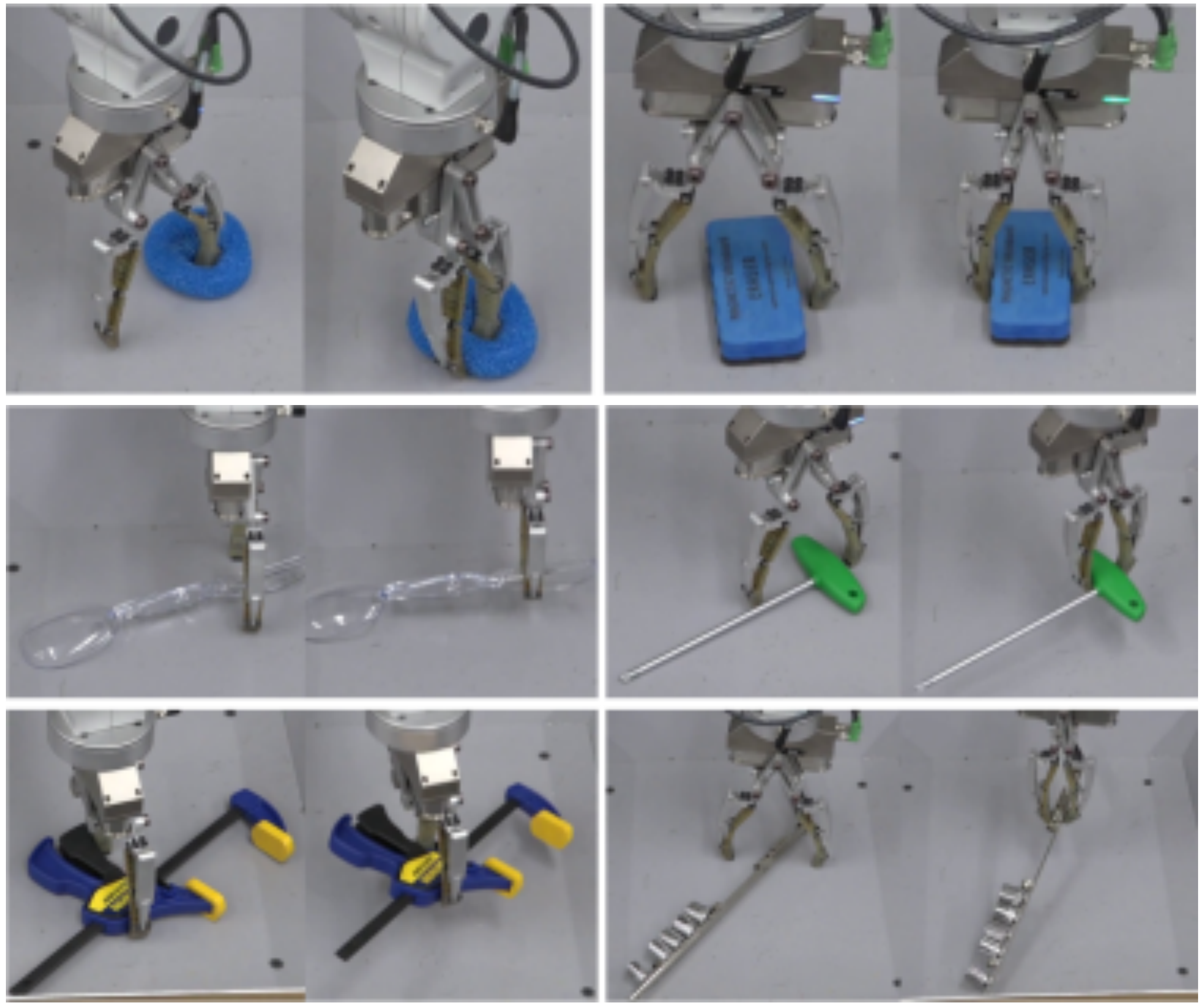


在智能制造领域的应用

Google在制造领域的工作（2016年）

14台机械臂，80万次抓取作为训练，可实现对未见过的软硬材质、透明、不同重量、异形等多样化物件的精准抓取





深度学习后续发展可能

局部最优

梯度弥散问题
局部极值问题

人工设计的可能性

在初始化时引入人工是否有意义?

计算复杂度

永远存在复杂度的问题

代价函数的设计优化

重构误差的考虑、引入惩罚项

人脑机理模拟

是否人脑的机制是最合适的?

整个网络的设计优化

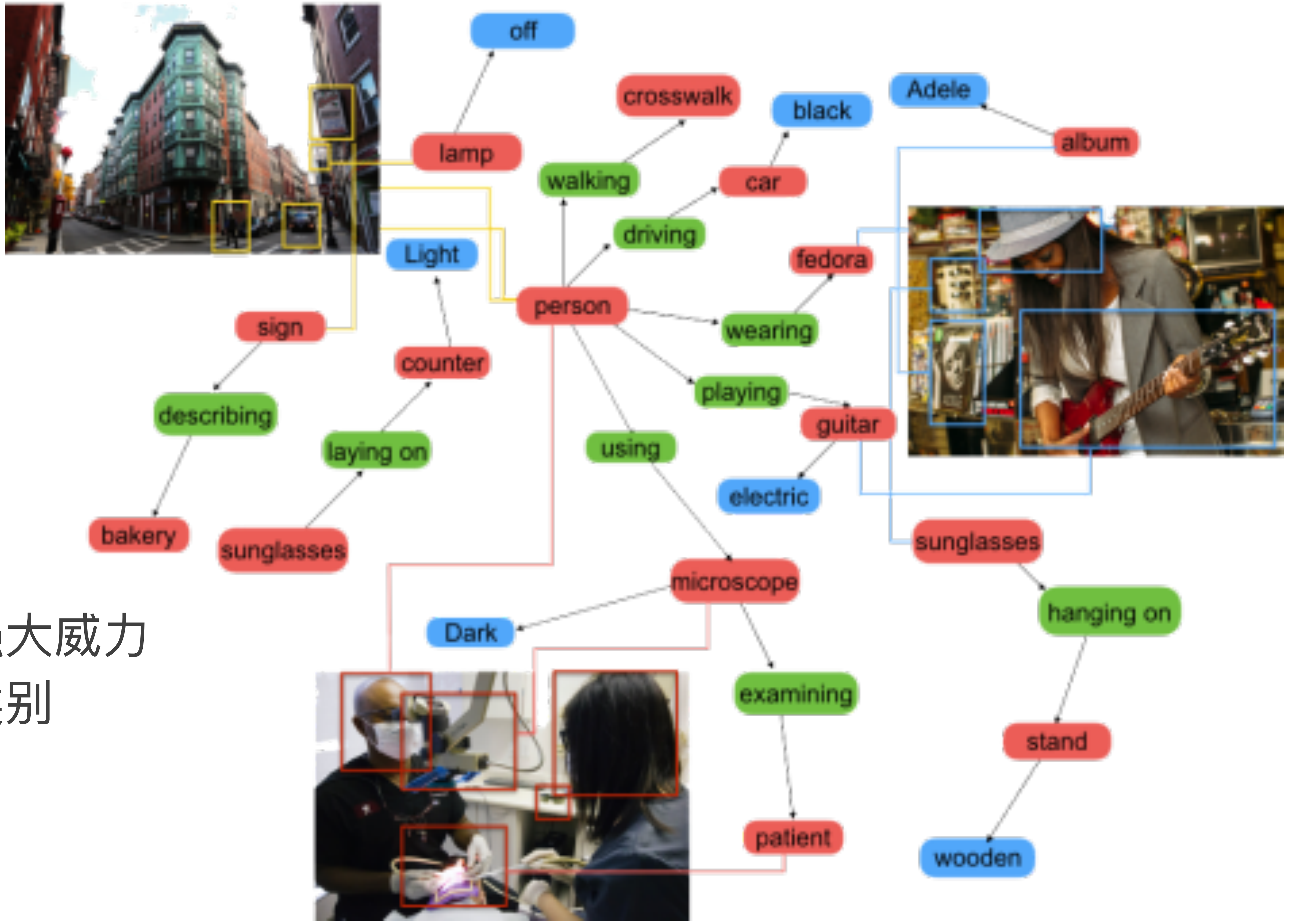
DeconvNet，DeepPose……

深度学习后续发展可能

数据集

更多种类、更大规模的数据集可能出现，如Feifei Li目前在推动的视觉基因组（Visual Genome）

- 108,249张图像
- 4.2M个区域描述
- 1.7M个视觉问题问答答案
- 2.1M个实体概念
- 1.8M个属性描述
- 1.8M个关系描述



One-Shot Learning

深度学习利用需要借助大量训练数据才能实现其强大威力
人类却能仅通过有限样例就能学习到新的概念和类别

深度学习后续发展可能

分布式框架软件

发挥CPU+GPU的混合性能

指令集与计算芯片

针对深度学习优化的新架构

寒武纪处理芯片

体系结构顶级会议ISCA 2016中

- 9篇与深度学习相关（共57篇）
- 1篇为评分最高论文

专用处理芯片

以FPGA为主的解决方案

降低成本、降低功耗

更多类型的新处理芯片?

Tensor Processing Unit（TPU）？

智能的三种类型

感知智能

对视觉、听觉、触觉等感知能力的模拟

认知智能

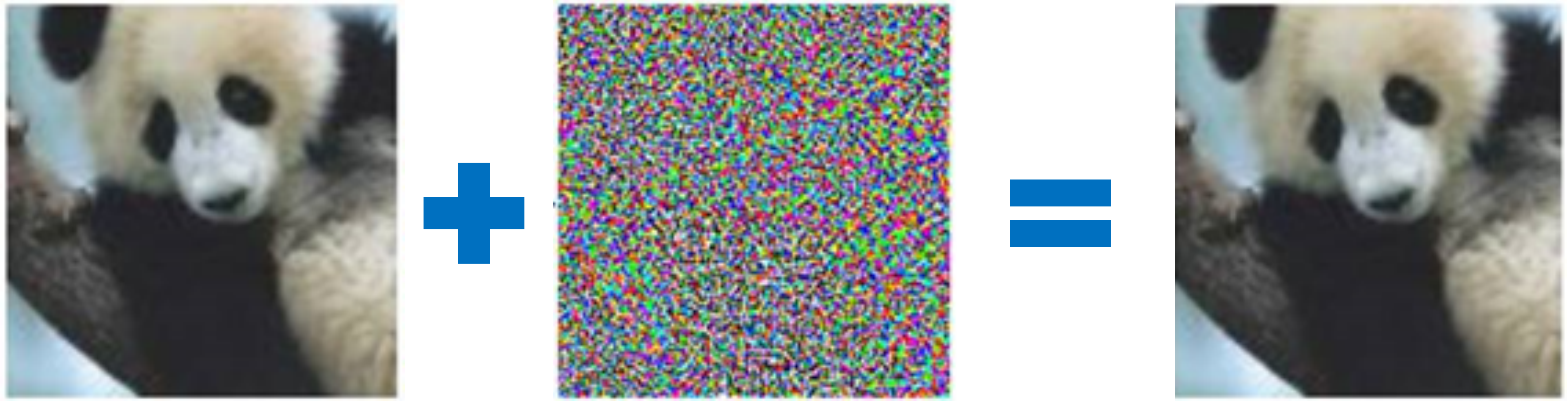
对推理、规划、决策、学习等认知能力的模拟

创造性智能

对灵感、顿悟等能力的模拟

深度学习已经解决一切了吗?

深度学习已经解决一切了吗?



A. 机判为熊猫
(正确)

小噪声扰动

B.机判为猿猴
(错误)

深度学习已经解决一切了吗?

Chap 7 - Neural Network

- Recap
- Regularization
- Batch Normalization

Back-Propagation

- Back-propagation is “just the chain rule” of calculus

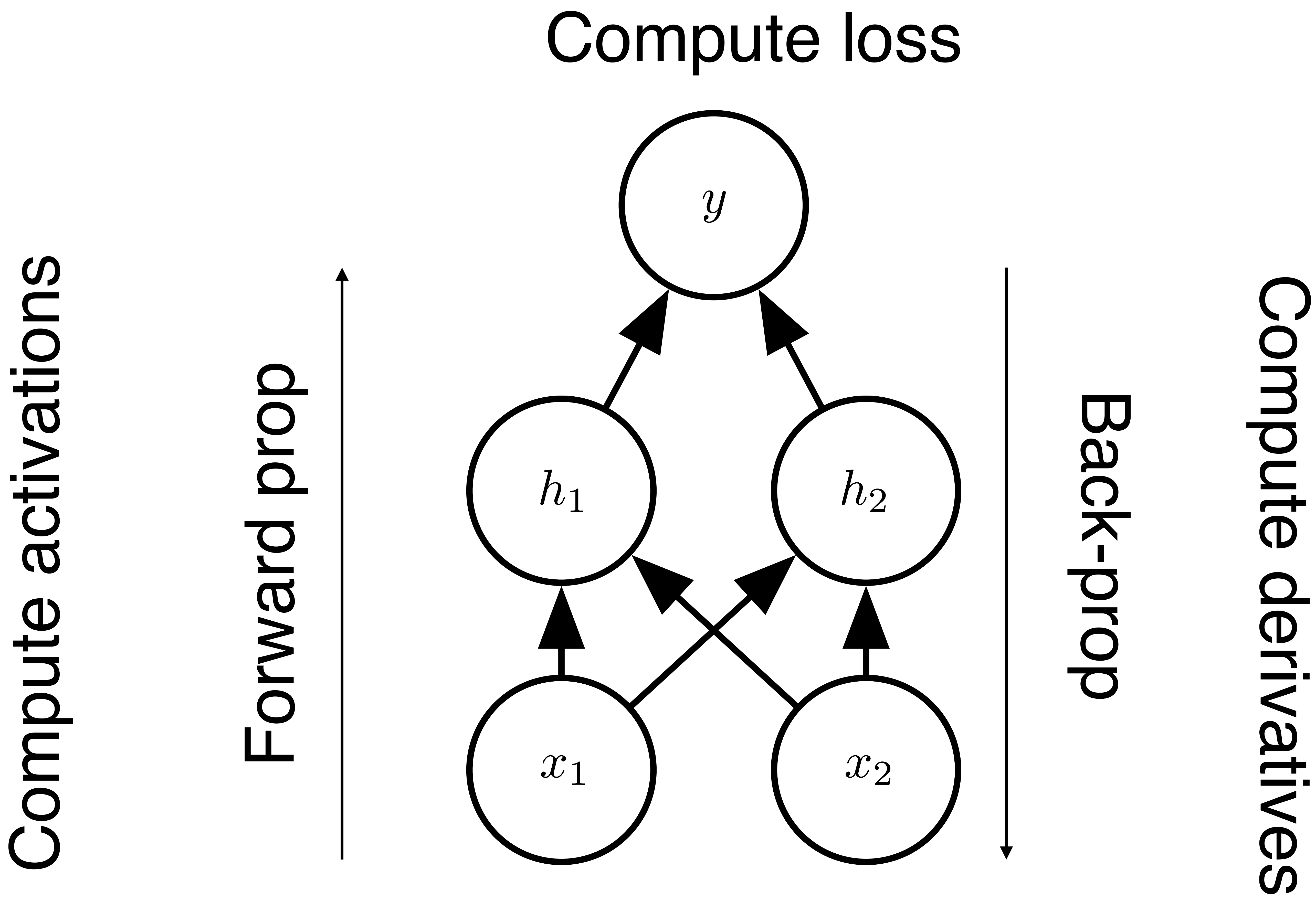
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{6.44}$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^\top \nabla_{\boldsymbol{y}} z, \tag{6.46}$$

- But it’s a particular implementation of the chain rule
  - Uses dynamic programming (table filling)
  - Avoids recomputing repeated subexpressions
  - Speed vs memory tradeoff

(Goodfellow 2017)

Simple Back-Prop Example



(Goodfellow 2017)

Computation Graphs

Multiplication

Logistic regression

ReLU layer

Linear regression and weight decay

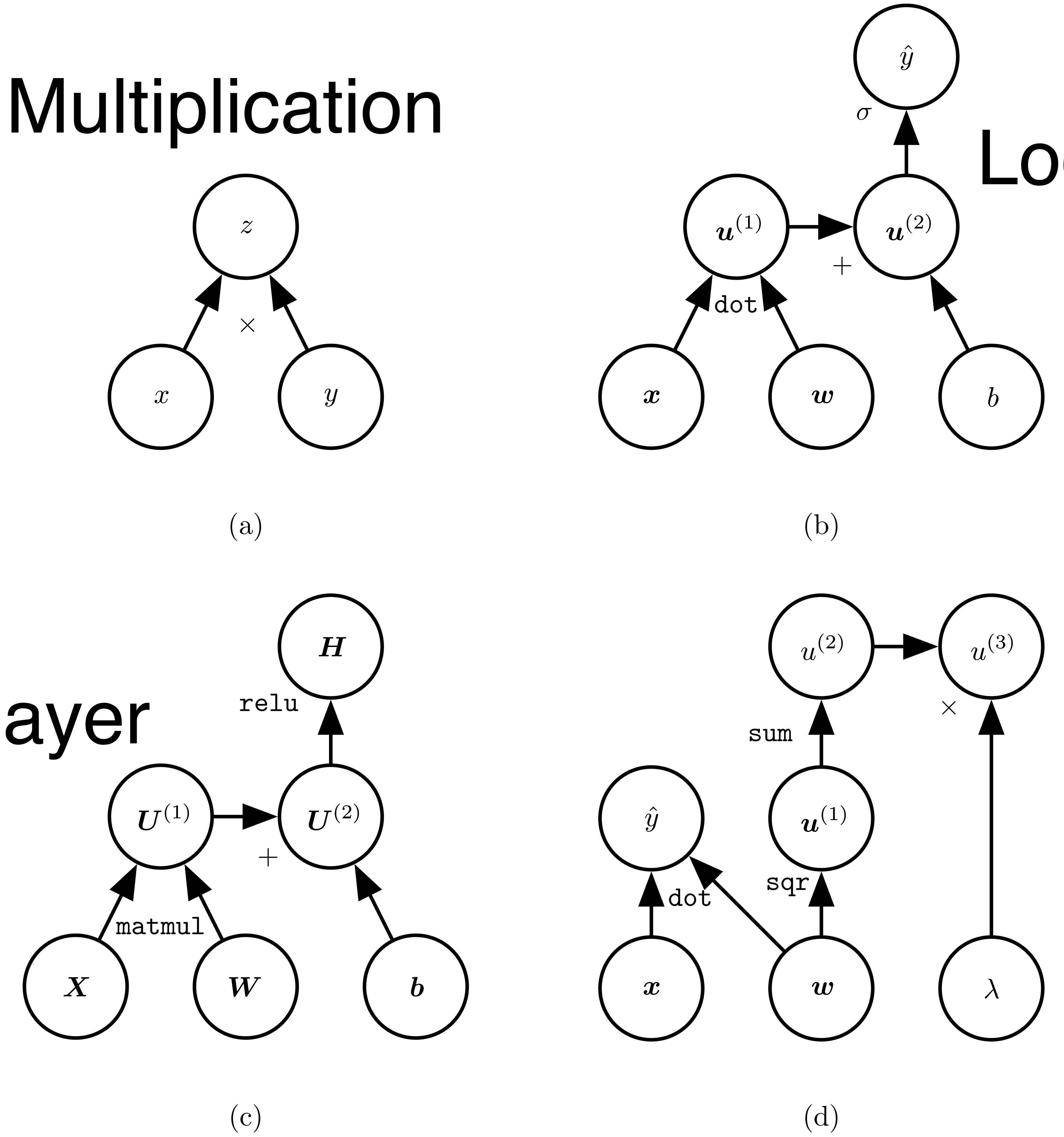
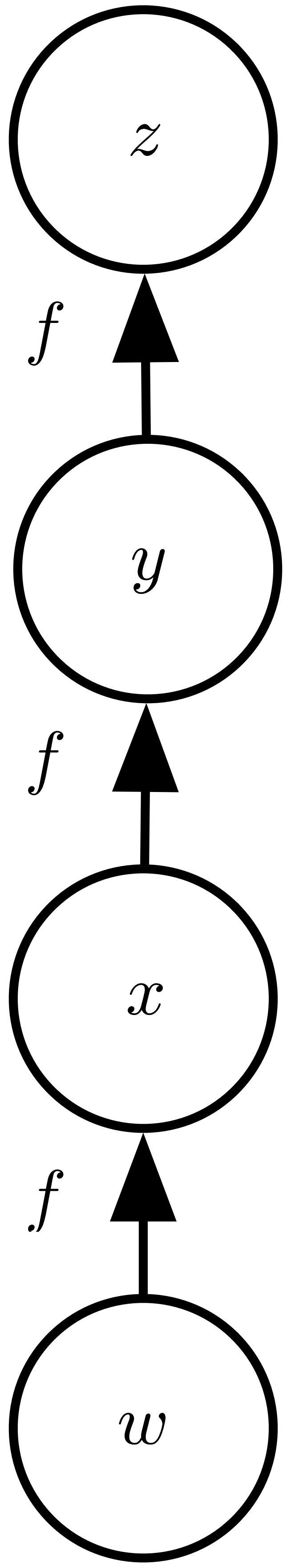


Figure 6.8

(Goodfellow 2017)

Repeated Subexpressions



$$\frac{\partial z}{\partial w} \tag{6.50}$$

$$= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \tag{6.51}$$

$$= f'(y) f'(x) f'(w) \tag{6.52}$$

$$= f'(f(f(w))) f'(f(w)) f'(w) \tag{6.53}$$

Back-prop avoids computing this twice

Figure 6.9

(Goodfellow 2017)

Regularization for Deep Learning

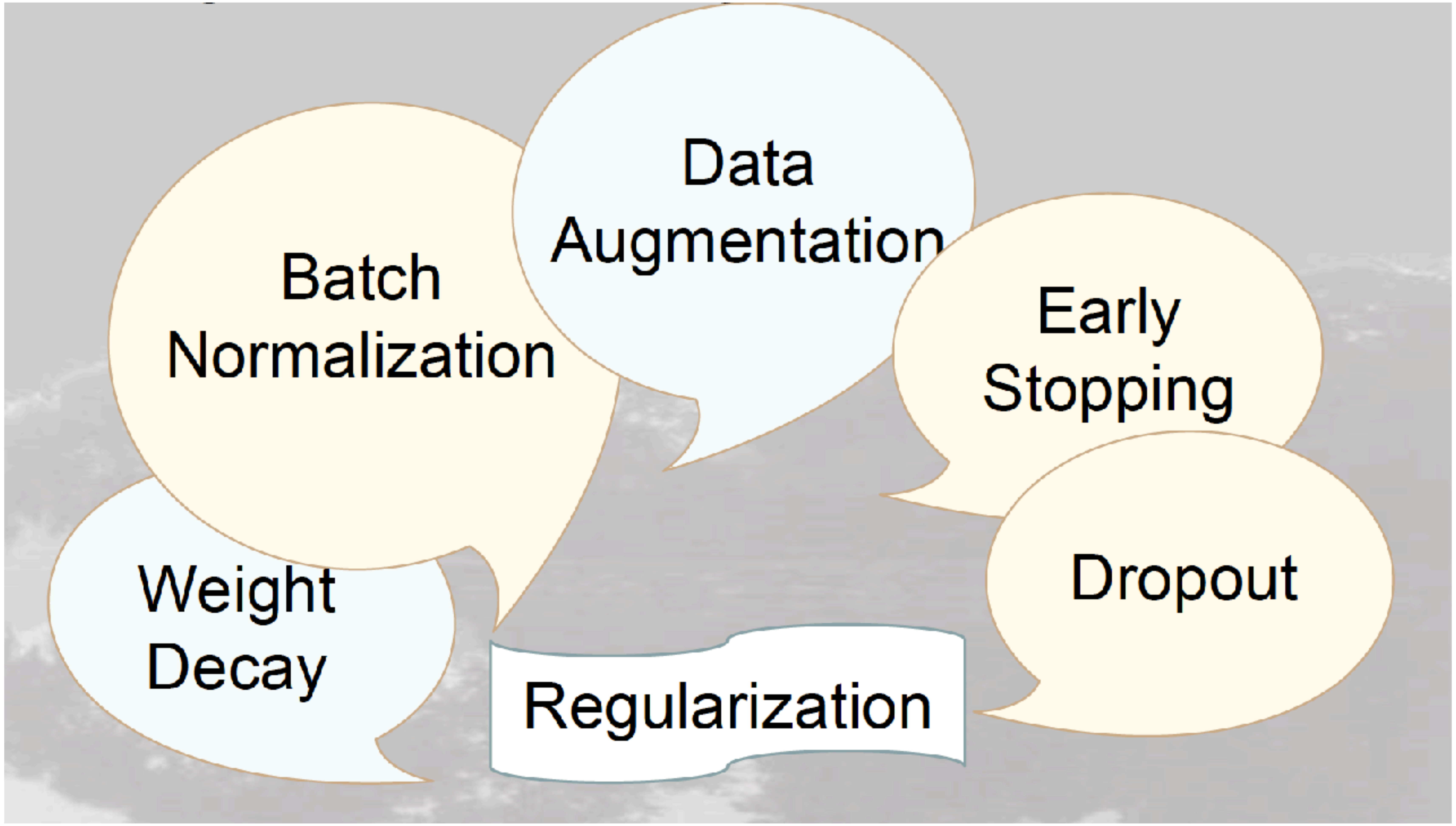
Definition of Regularization

Optional subtitle

“Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

To avoid overfitting, and improve generalization performance

Optional subtitle



Some Observations of Deep Nets

- # of parameters >> # of data, hence easy to fit data
- Without regularization, deep nets also have benign generalization
- For random label or random feature, deep nets converge with 0 training error but without any generalization

Weight Decay as Constrained Optimization

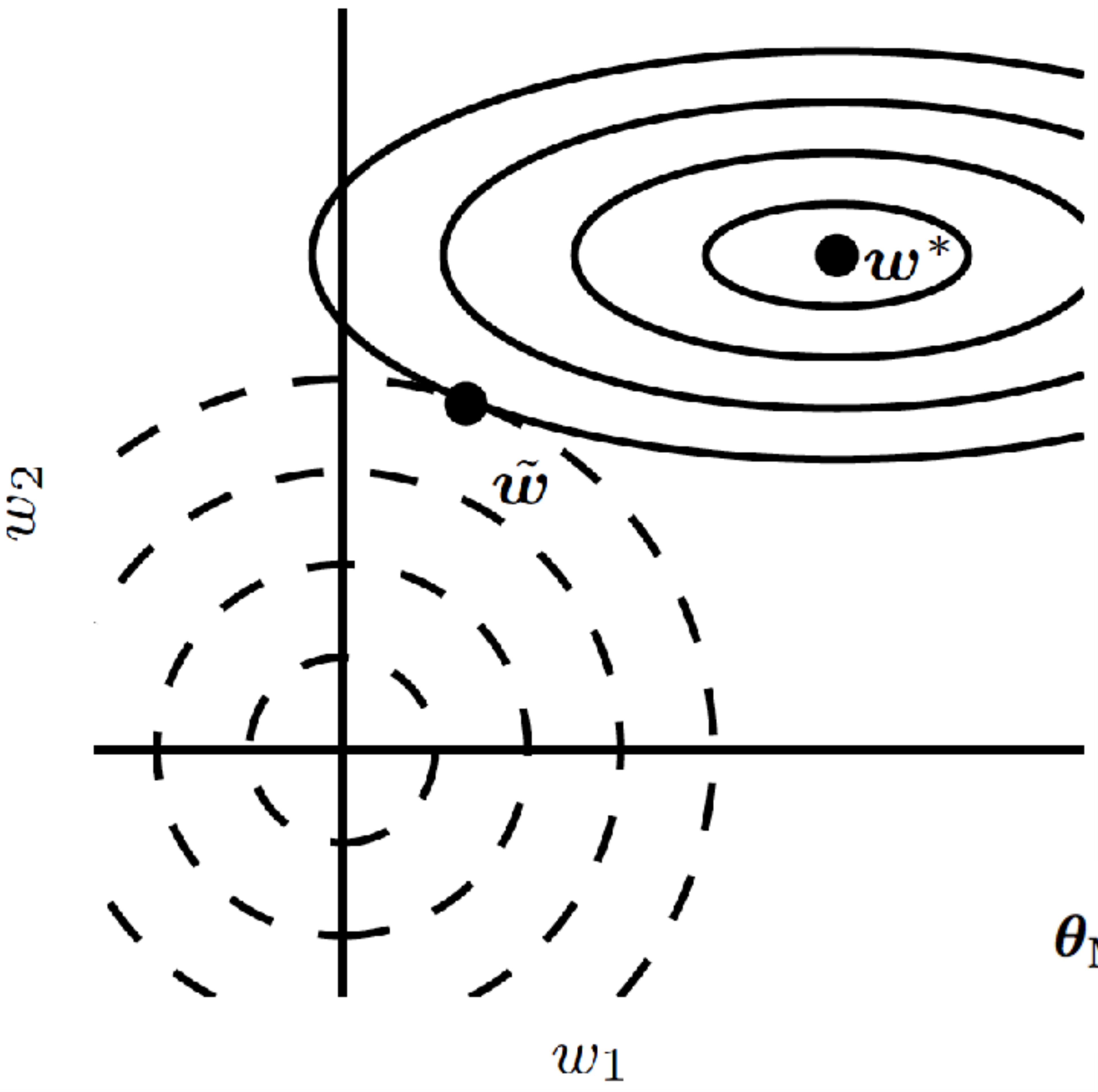


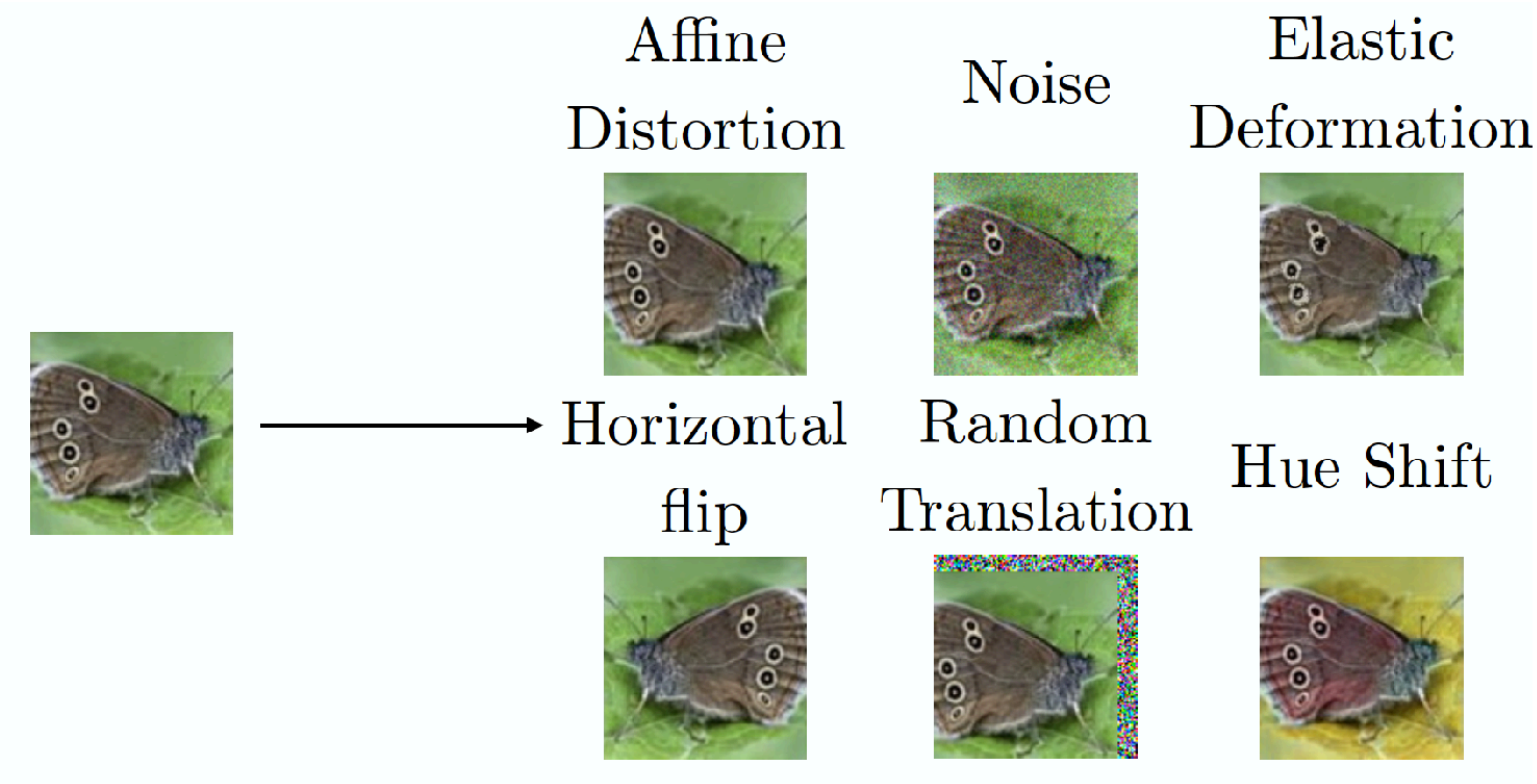
Figure 7.1

- L1: Encourages sparsity, equivalent to MAP Bayesian estimation with Laplace prior
- Squared L2: Encourages small weights, equivalent to MAP Bayesian estimation with Gaussian prior

$$\boldsymbol{\theta}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}).$$

Dataset Augmentation

Optional subtitle



(Goodfellow 2016)

Adversarial Examples

Optional subtitle

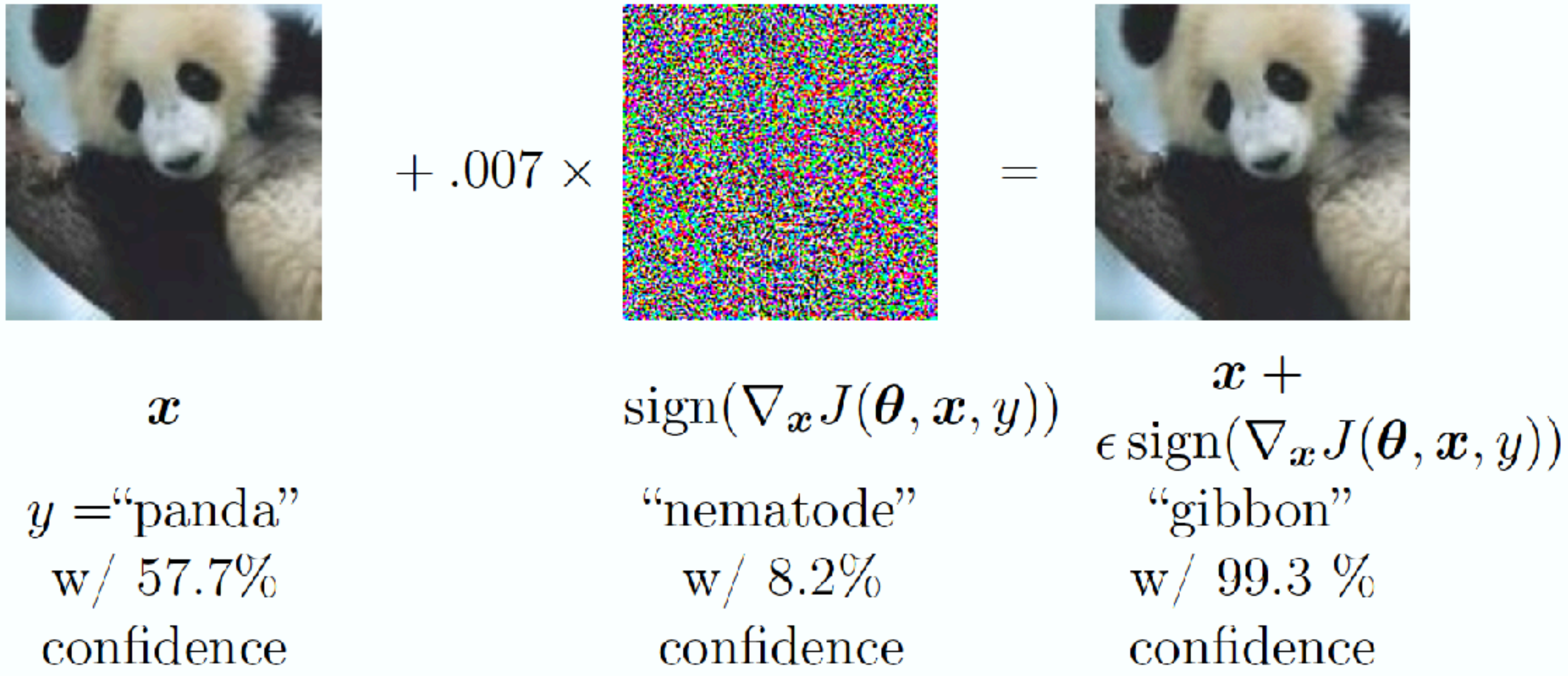


Figure 7.8

Training on adversarial examples is mostly intended to improve security, but can sometimes provide generic regularization.

ADVERSARIAL MANIPULATION OF DEEP REPRESENTATIONS

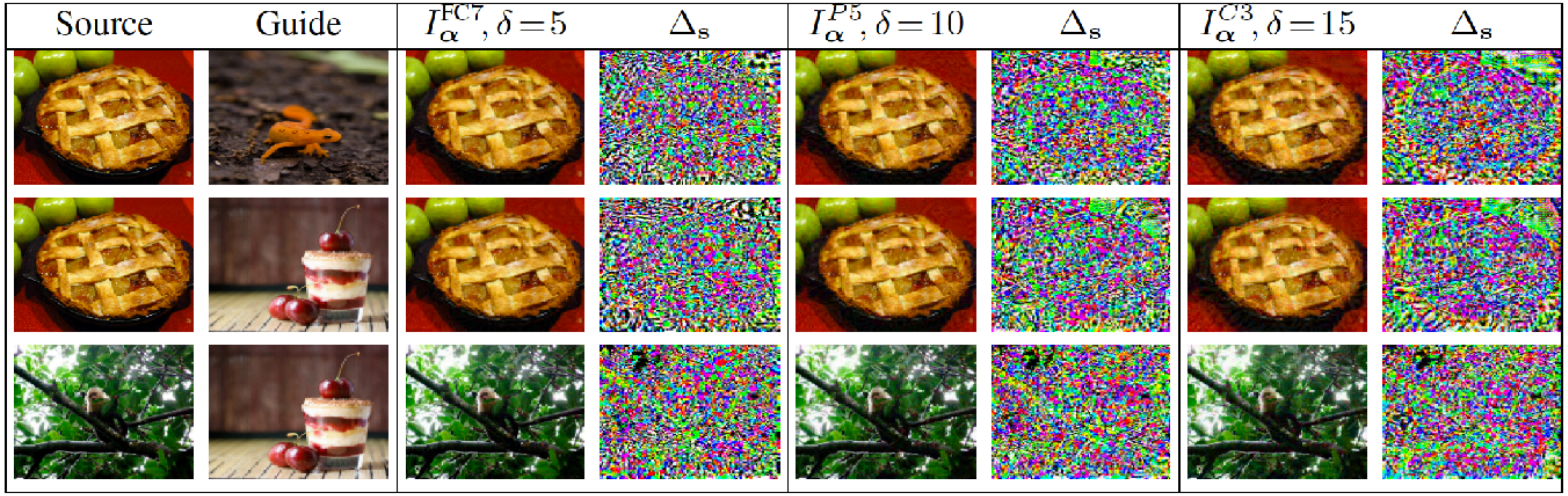


Figure 1: Each row shows examples of adversarial images, optimized using different layers of Caffenet (FC7, P5, and C3), and different values of $\delta = (5, 10, 15)$. Beside each adversarial image is the difference between its corresponding source image.

Let I_s and I_g denote the *source* and *guide* images. Let ϕ_k be the mapping from an image to its internal DNN representation at layer k. Our goal is to find a new image, I_α, such that the Euclidian distance between $\phi_k(I_\alpha)$ and $\phi_k(I_g)$ is as small as possible, while I_α remains close to the source I_s. More precisely, I_α is defined to be the solution to a constrained optimization problem:

$$I_\alpha = \arg\min_{I} \| \phi_k(I) - \phi_k(I_g) \|_2^2 \tag{1}$$

$$\text{subject to} \quad \|I - I_s\|_\infty < \delta \tag{2}$$

(David Fleet's Group, ICLR 2016)

Learning Curves

Optional subtitle

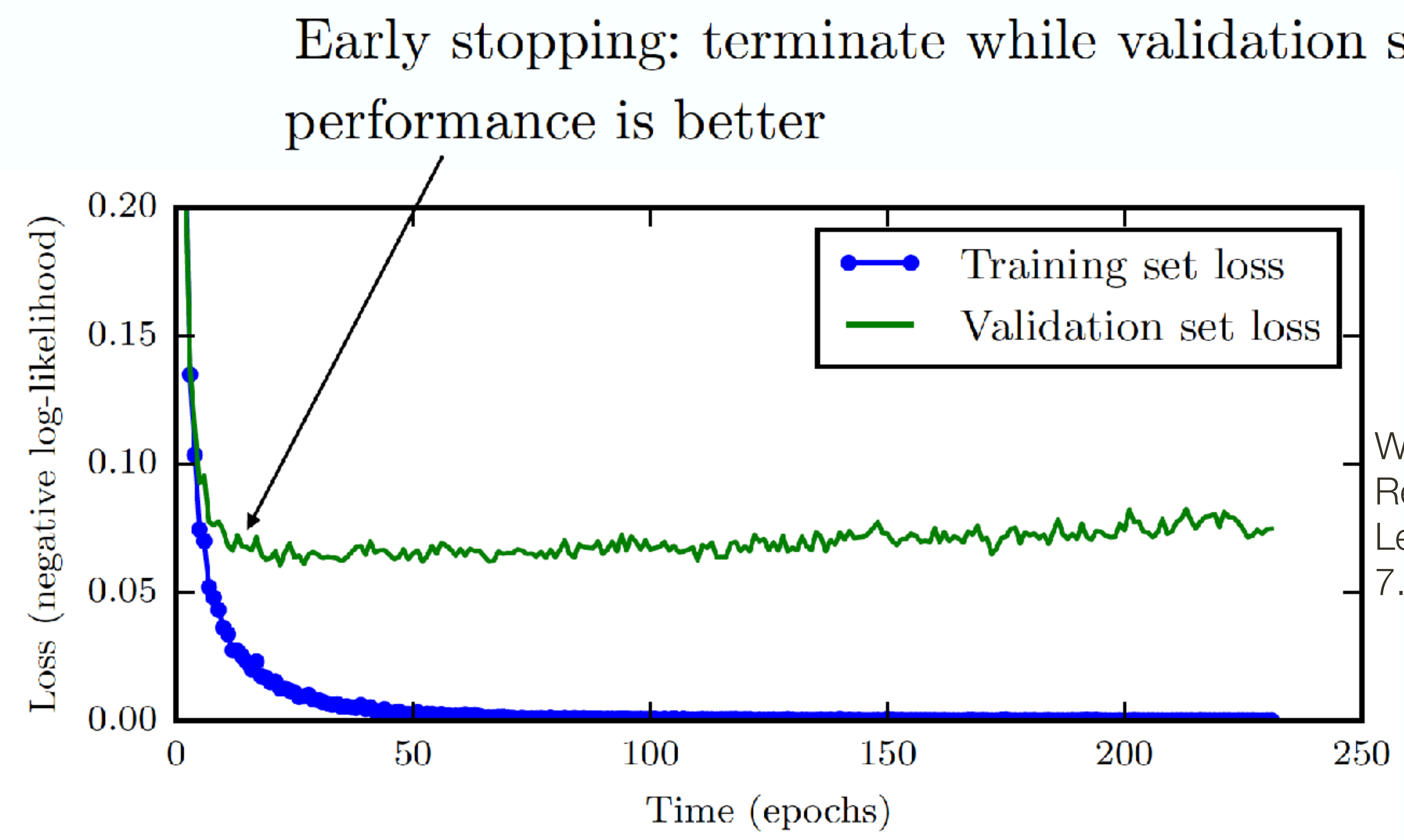


Figure 7.3

Why it works? Refer to “Deep Learning” book, Chap 7.8.

(Goodfellow 2016)

Bagging

Optional subtitle

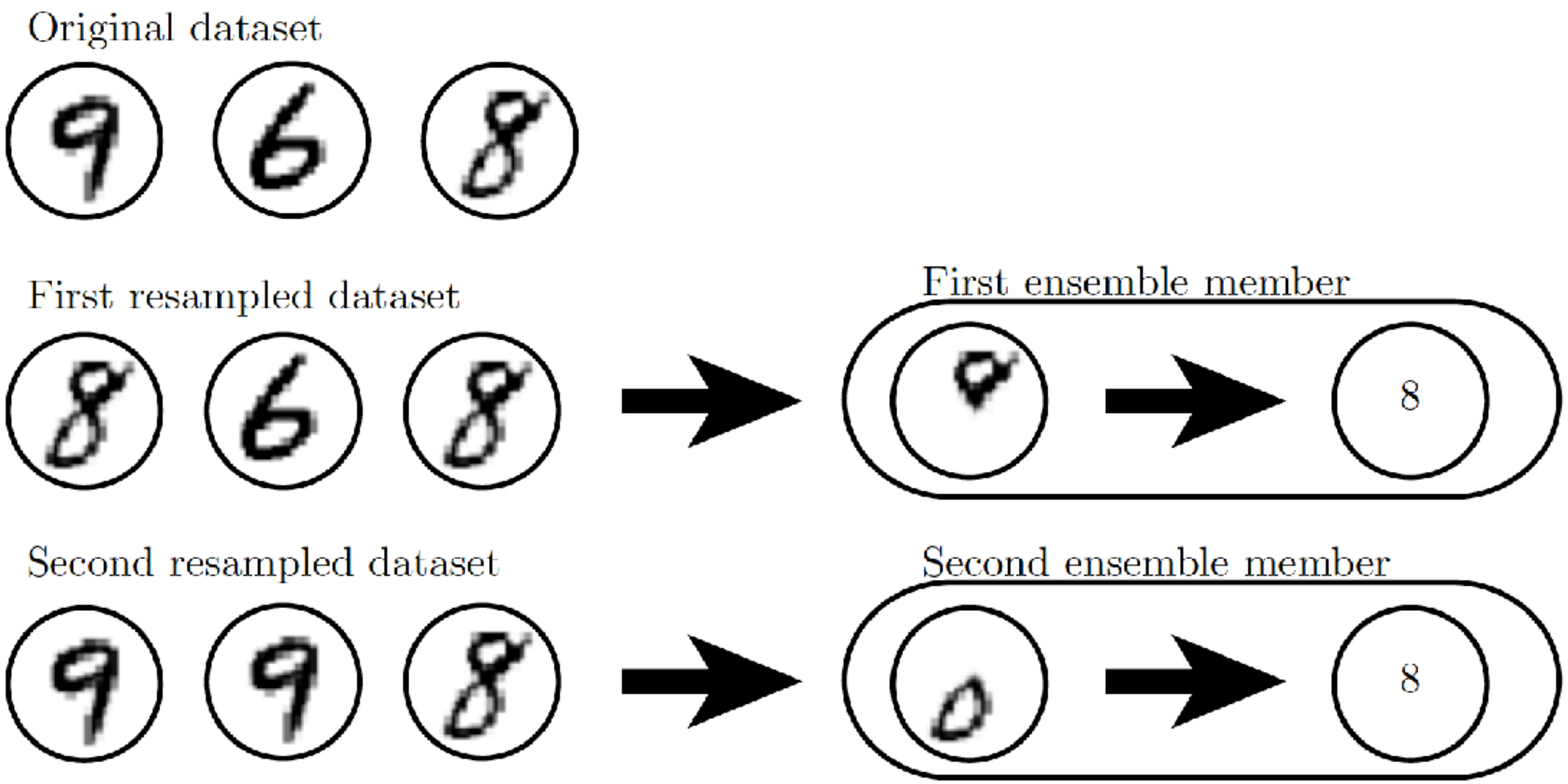


Figure 7.5

(Goodfellow 2016)

Batch Normalization

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Ioffe and Szegedy 2015

Batch Normalization

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{W}$$

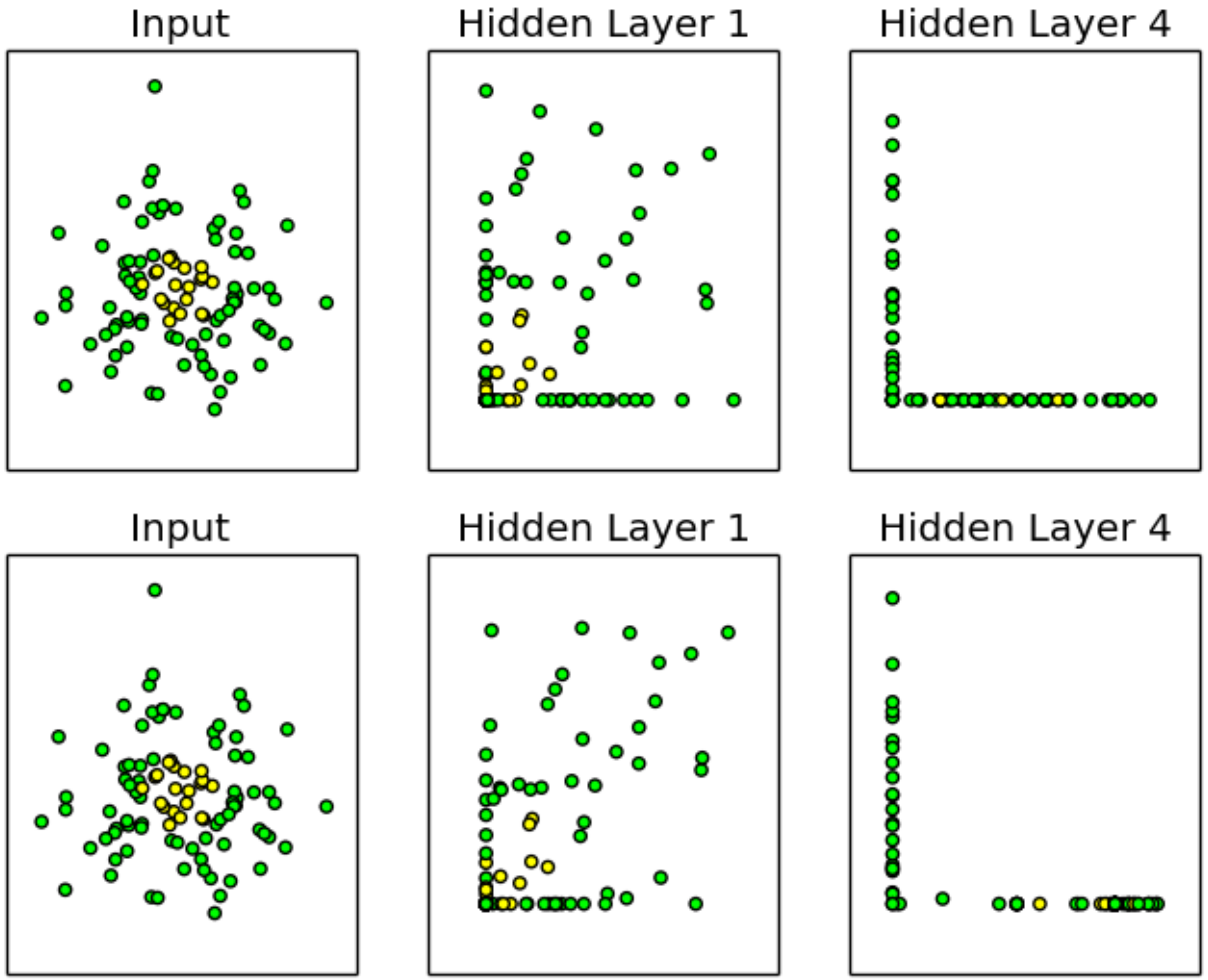
$$\tilde{\boldsymbol{Z}} = \boldsymbol{Z} - \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{Z}_{i,:}$$

$$\hat{\boldsymbol{Z}} = \frac{\tilde{\boldsymbol{Z}}}{\sqrt{\epsilon + \frac{1}{m}\sum_{i=1}^{m} \tilde{\boldsymbol{Z}}_{i,:}^{2}}}$$

$$\boldsymbol{H} = \max\{0, \boldsymbol{\gamma}\hat{\boldsymbol{Z}} + \boldsymbol{\beta}\}$$

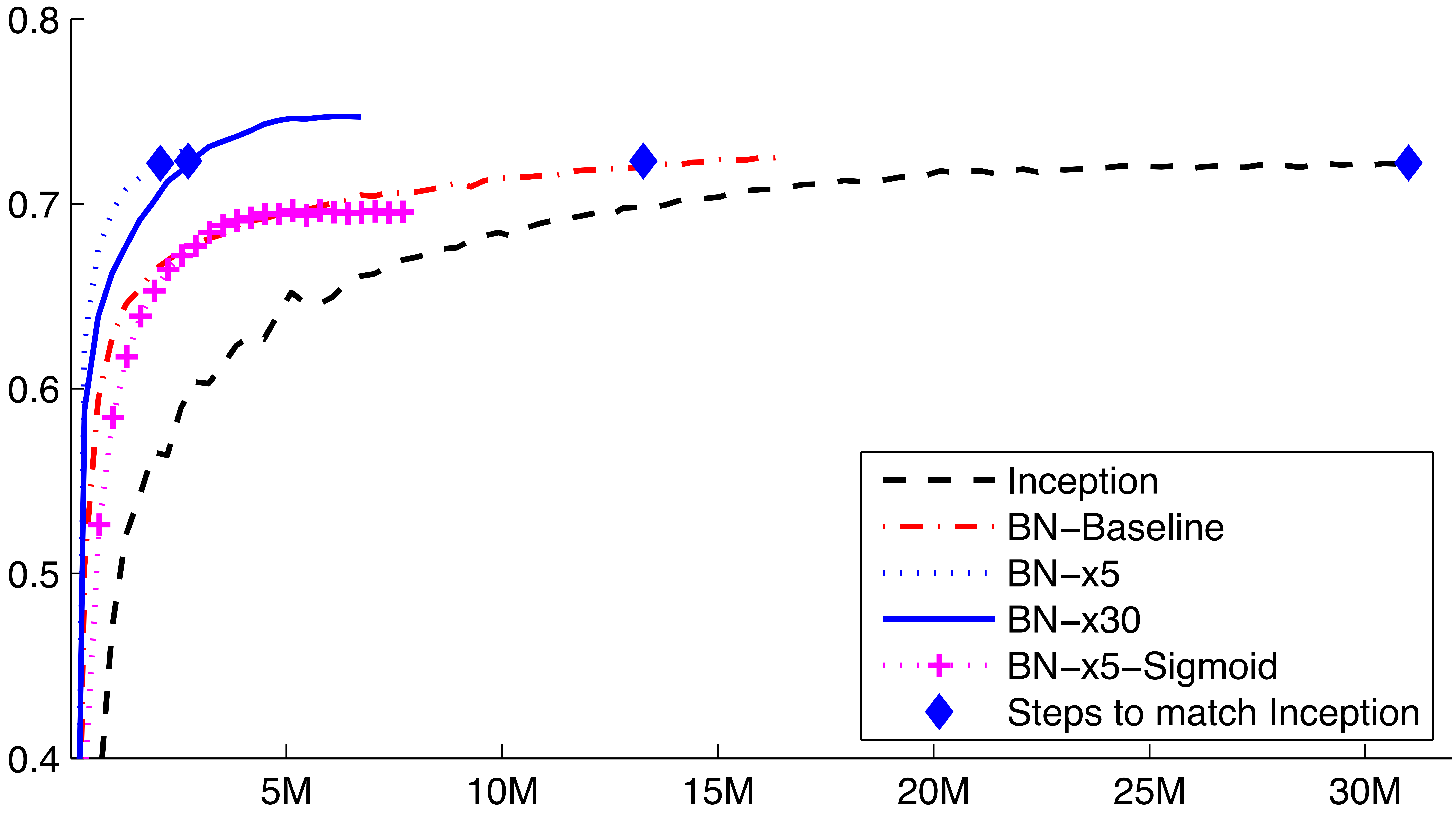
"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Ioffe and Szegedy 2015

Before SGD step



After SGD step

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Ioffe and Szegedy 2015



“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Ioffe and Szegedy 2015

Recap

1

Deep Learning Building Blocks

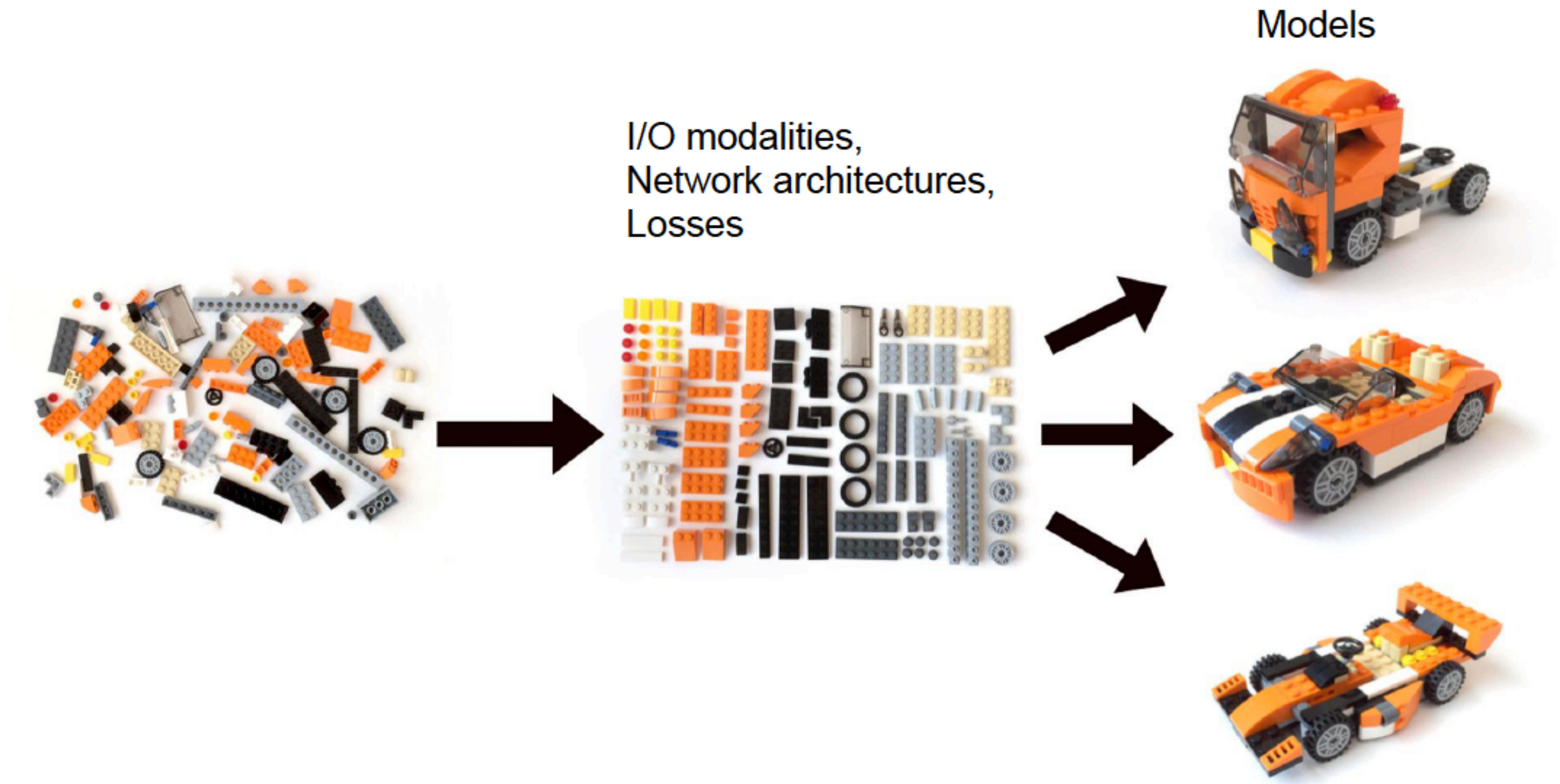


Image: Nagel, Wolfram. Multiscreen UX Design: Developing for a Multitude of Devices. Morgan Kaufmann, 2015.

Deep Learning: Zooming Out

Platforms

Microsoft Azure

aws

Google Cloud

Frameworks

scikit learn

Caffe

K Keras

Microsoft CNTK

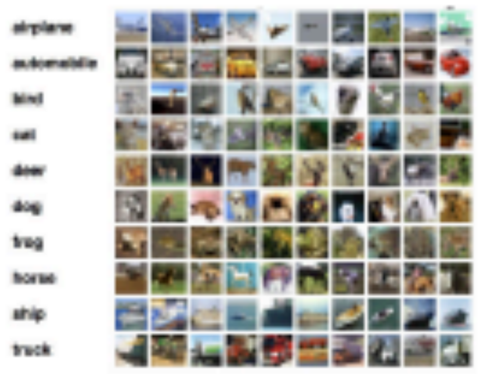
mxnet

theano

PYTORCH

Caffe2

Datasets



Caltech 101

IMAGENET

LDC Linguistic Data Consortium

UCI Machine Learning Repository

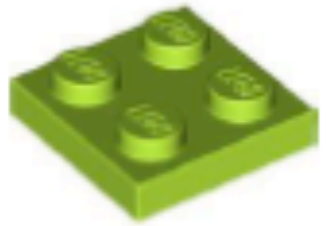
WMT Workshop 2014

Deep Learning: Zooming Out

Non-Linearities

Relu
Sigmoid
Tanh
GRU
LSTM
Linear
...

Connectivity Pattern



Fully connected
Convolutional
Dilated
Recurrent
Recursive
Skip / Residual
Random
...

Optimizer

SGD
Momentum
RMSProp
Adagrad
Adam
Second Order (KFac)
...

Loss

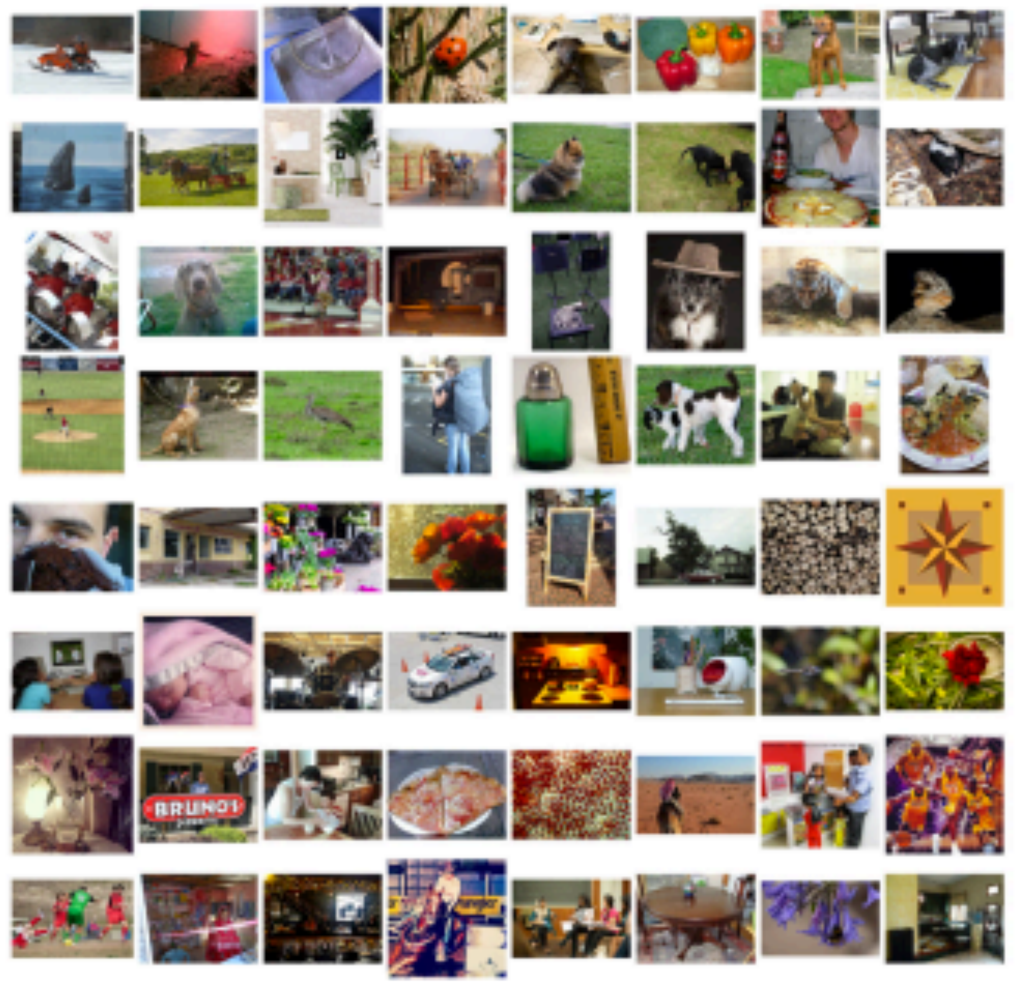
Cross Entropy
Adversarial
Variational
Max. Likelihood
Sparse
L2 Reg
REINFORCE
...

Hyper Parameters

Learning Rate
Decay
Layer Size
Batch Size
Dropout Rate
Weight init
Data augmentation
Gradient Clipping
Beta
Momentum

Images

- Structured
- Classification
- Segmentation
- Medical images
- Generative Models
- Art



http://www.image-net.org/

Progressive GANs, Karras et al, 2017

A Neural Algorithm of Artistic Style

Gatys et al, 2015

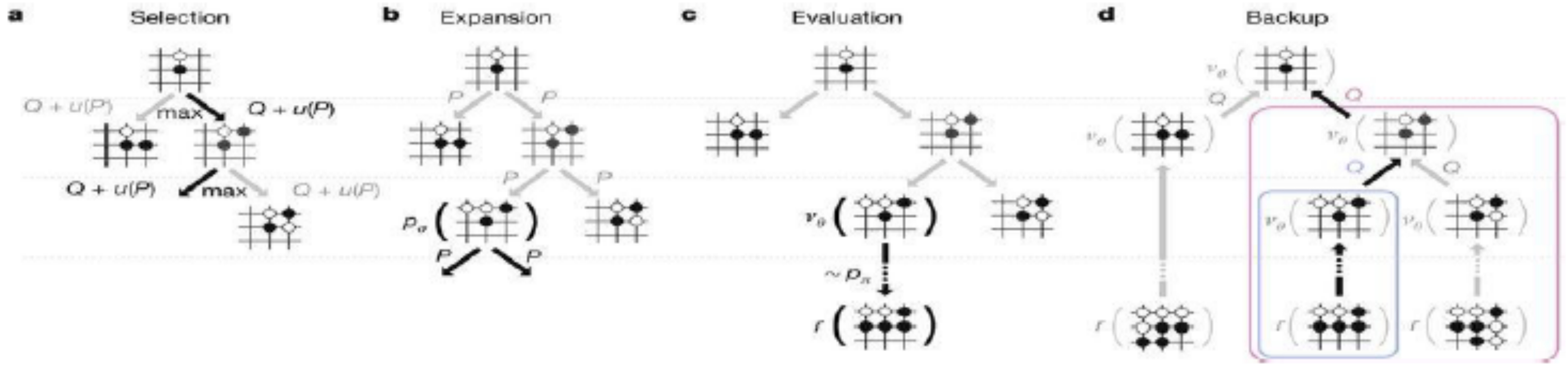
Sequences

> *50 years ago, the fathers of artificial intelligence convinced everybody that logic was the key to intelligence. Somehow we had to get computers to do logical reasoning. The alternative approach, which they thought was crazy, was to forget logic and try and understand how networks of brain cells learn things. Curiously, two people who rejected the logic based approach to AI were Turing and Von Neumann. If either of them had lived I think things would have turned out differently... now neural networks are everywhere and the crazy approach is winning.*

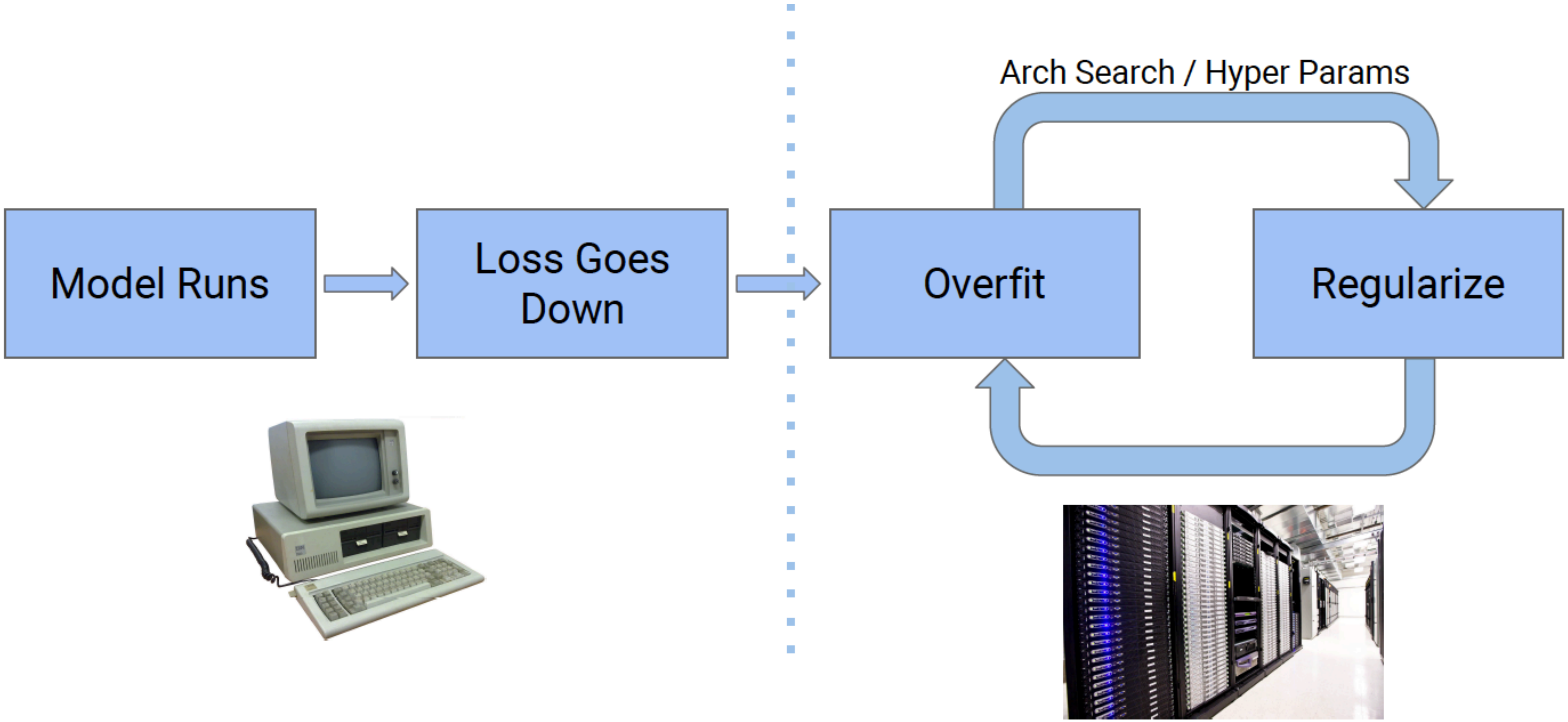
- Words, Letters
- Speech
- Images, Videos, Touch
- Programs

```
while (*d++ = *s++);
```

- Sequential Decision Making (RL)



Deep Learning Vicious Cycle



Challenges of training very deep ConvNets

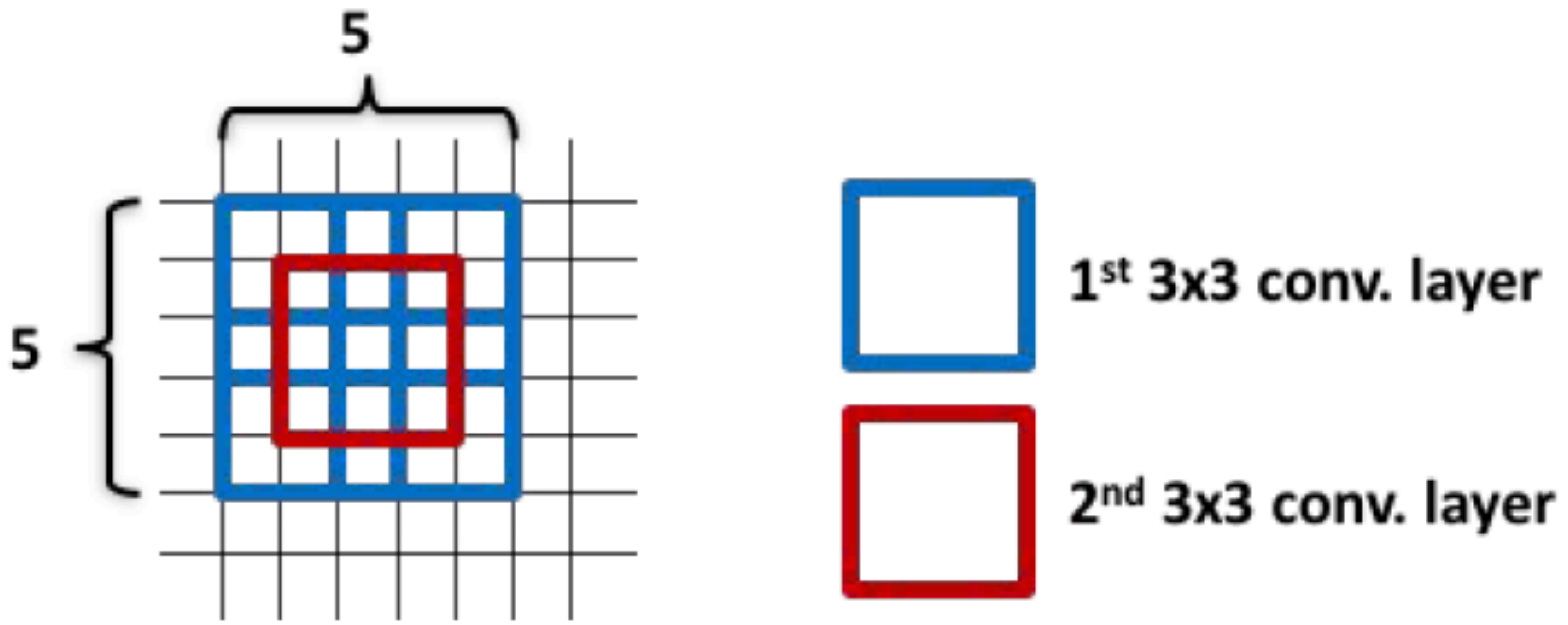
- We have seen that depth is important
- Why not to keep adding layers?

Two main reasons:

- computational complexity
  - ConvNet will be too slow to train and evaluate
- optimisation
  - we won’t be able to train such nets

Building Very Deep ConvNets

- Use stacks of small (3×3) conv. layers
  - in most cases, the only kernel size you need
  - a cheap way of building a deep ConvNet
- Stacks have a large receptive field
  - two 3×3 layers – 5×5 field
  - three 3×3 layers – 7×7 field
- Less parameters than a single layer with a large kernel

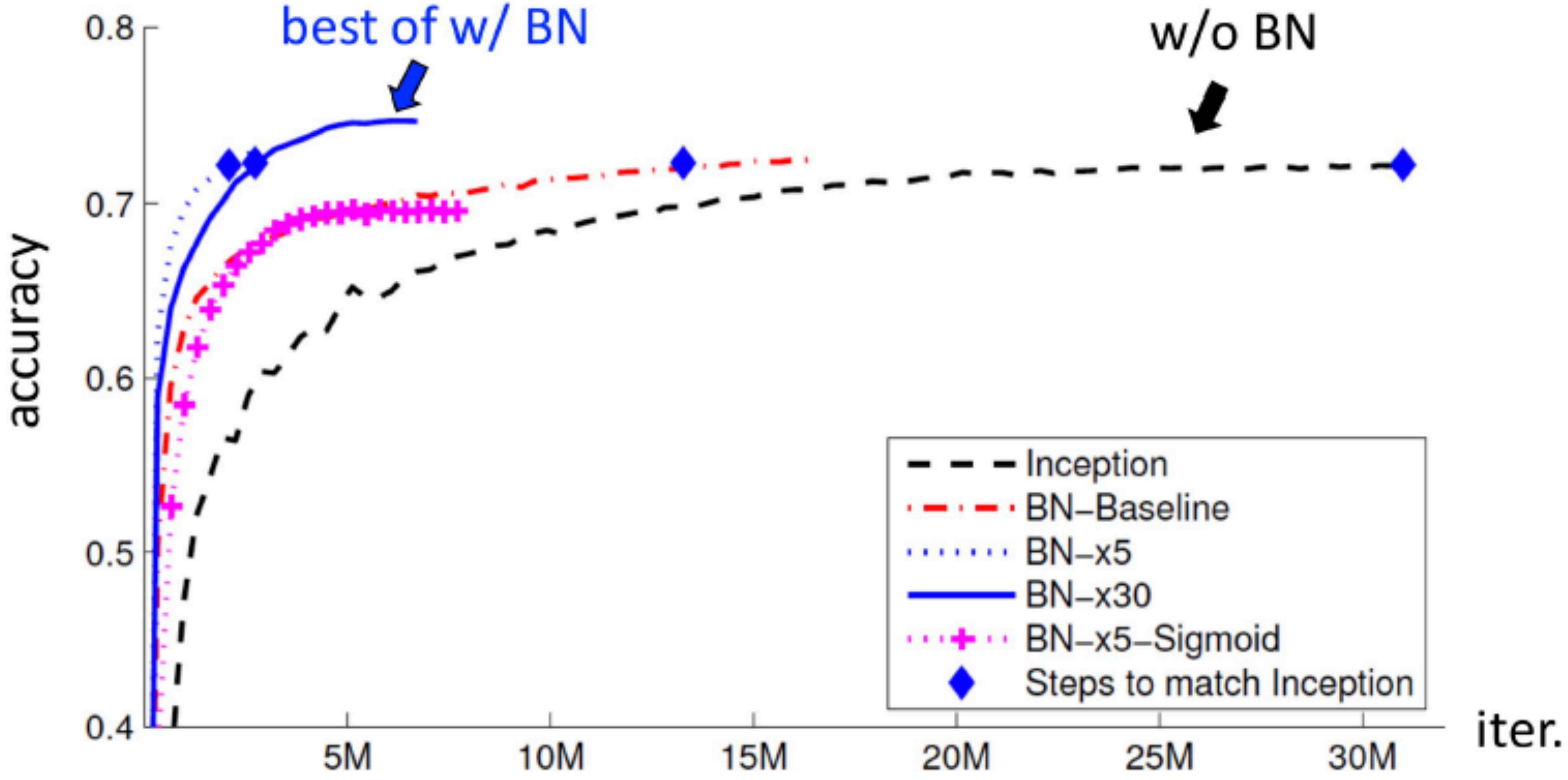


(Some) Tricks of the Training Networks

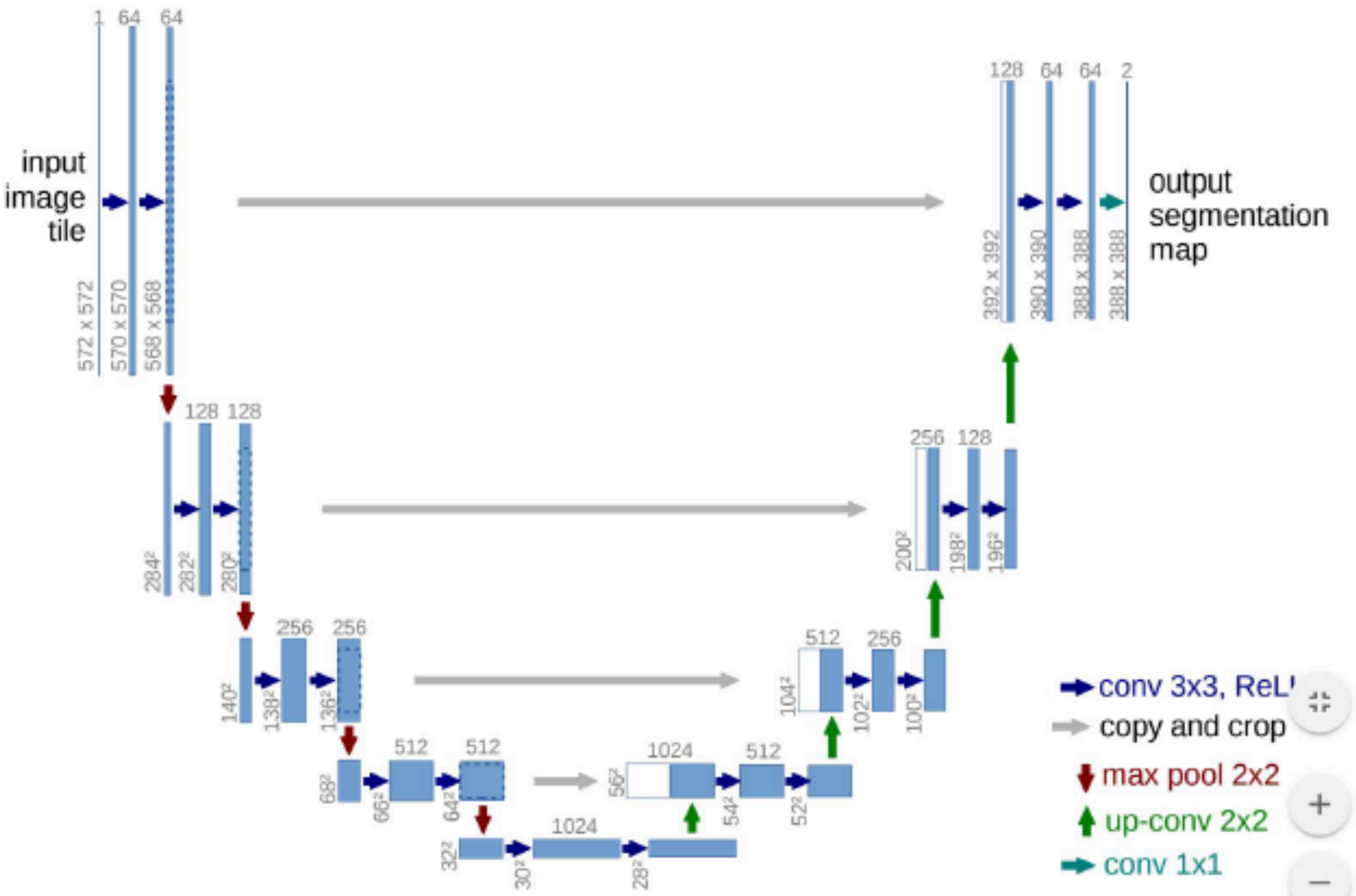
- Optimization
  - SGD with momentum – typical choice for ConvNets
  - Batch Norm
- Initialization
  - Weight init: start from the weights which lead to stable training
  - Sample from zero-mean normal distribution w/ small variance 0.01
    - Adaptively choose variance for each layer
      - preserve gradient magnitude [Glorot & Bengio, 2010]: 1/sqrt(fan_in)
      - works fine for VGGNets (up to 20 layers), but not sufficient for deeper nets
- Model
  - Stacking 3x3 convolutions
  - Inception
  - ResNet adds modules which ensure that the gradient doesn't vanish

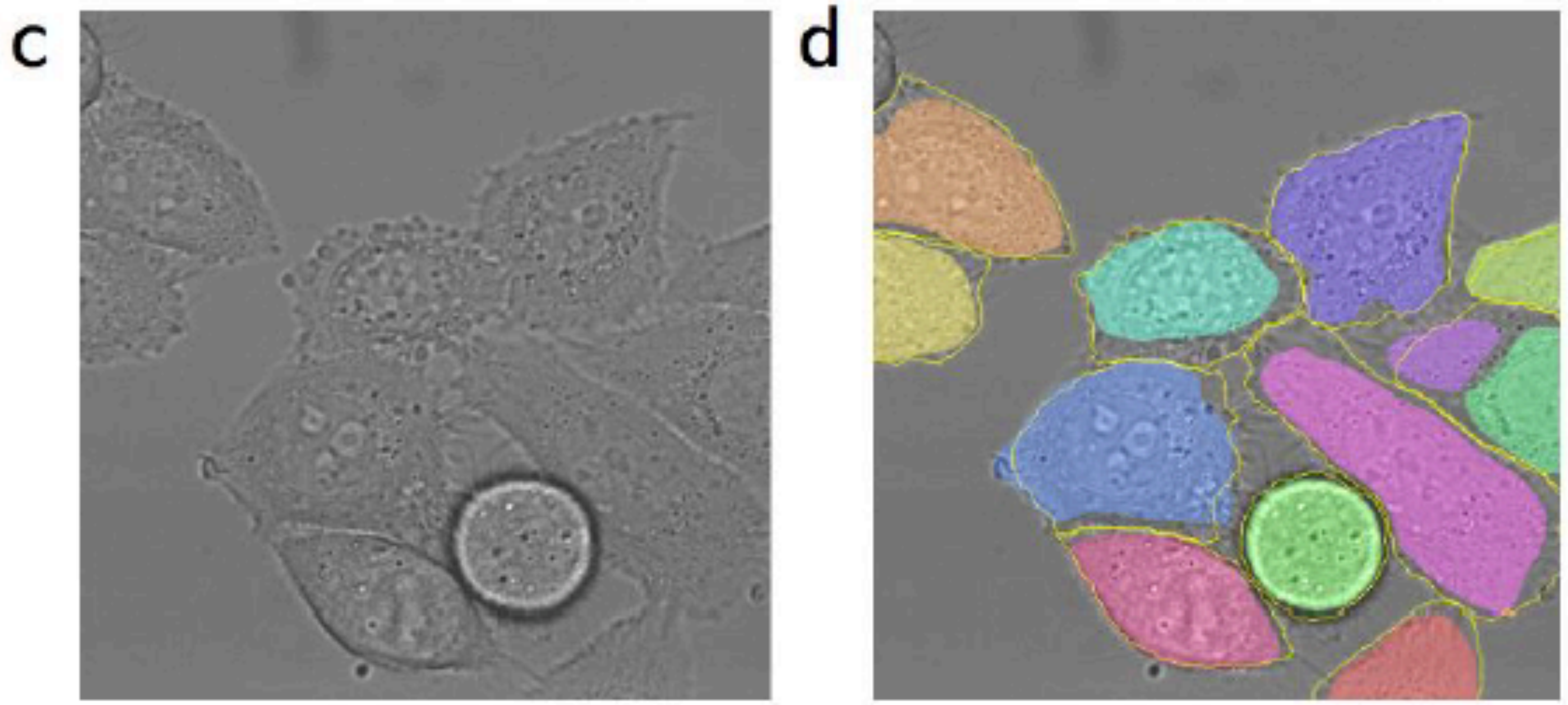
Inception Net v2: Importance of Batch Norm

Higher accuracy and faster training with batch-norm



U-Net architecture

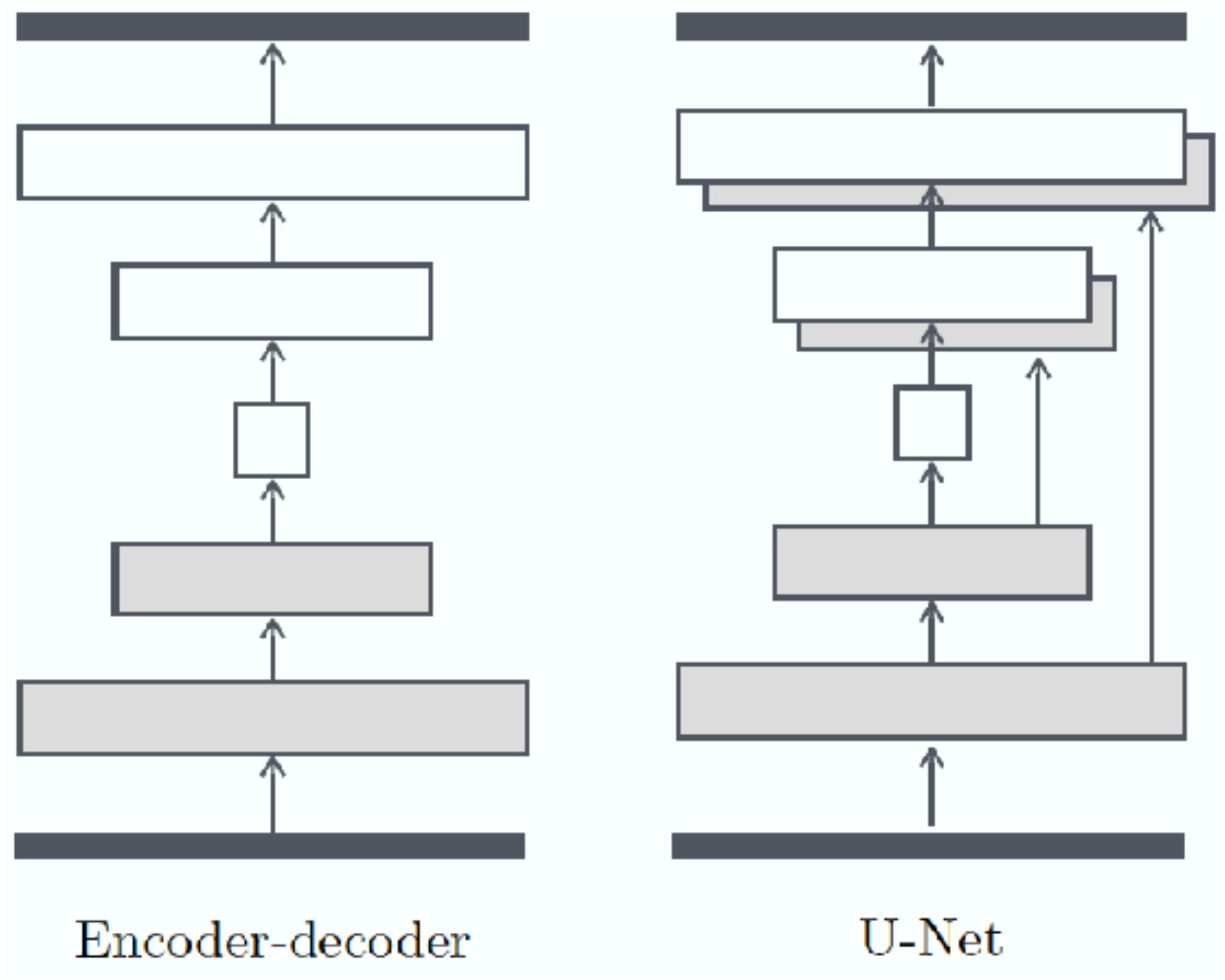




1. Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, Cham, 2015.

CycleGAN

Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks



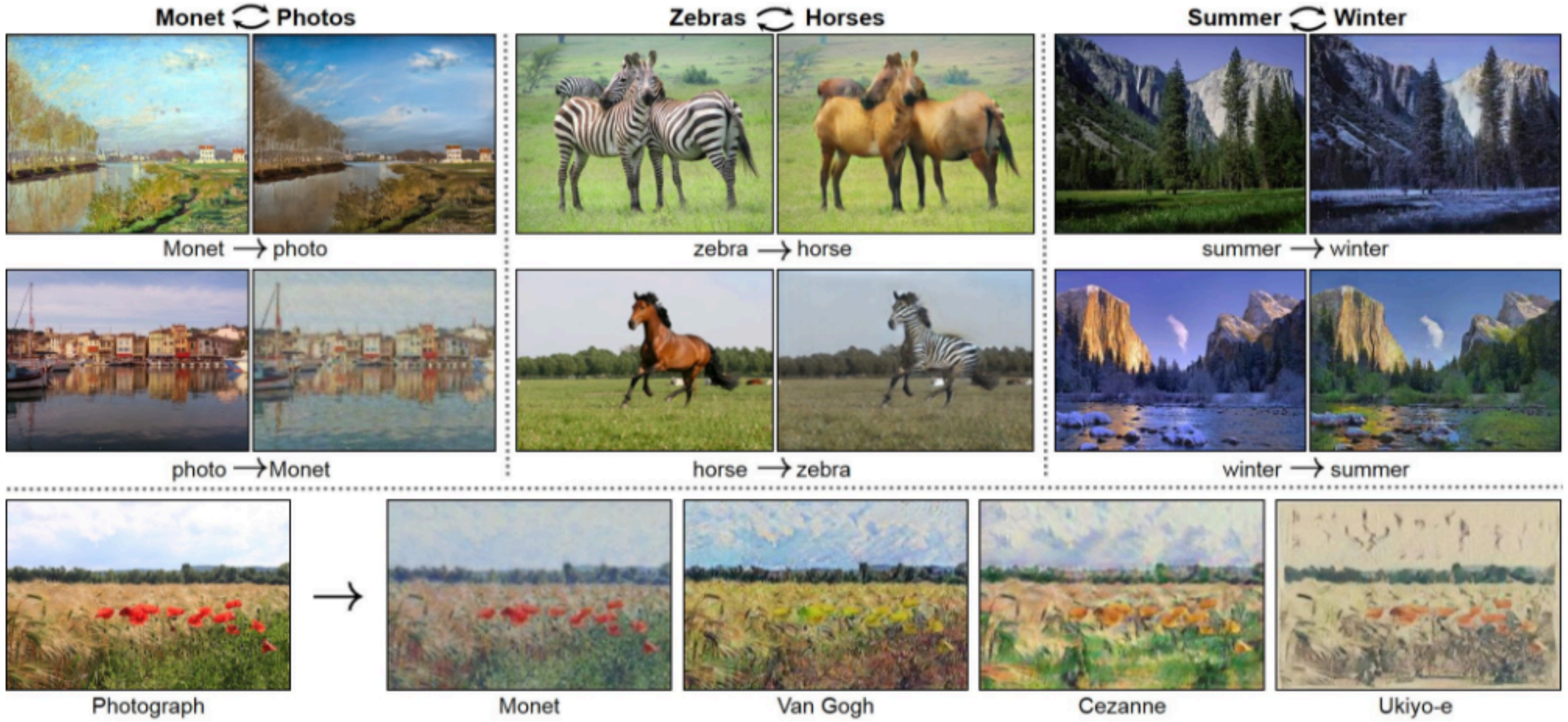
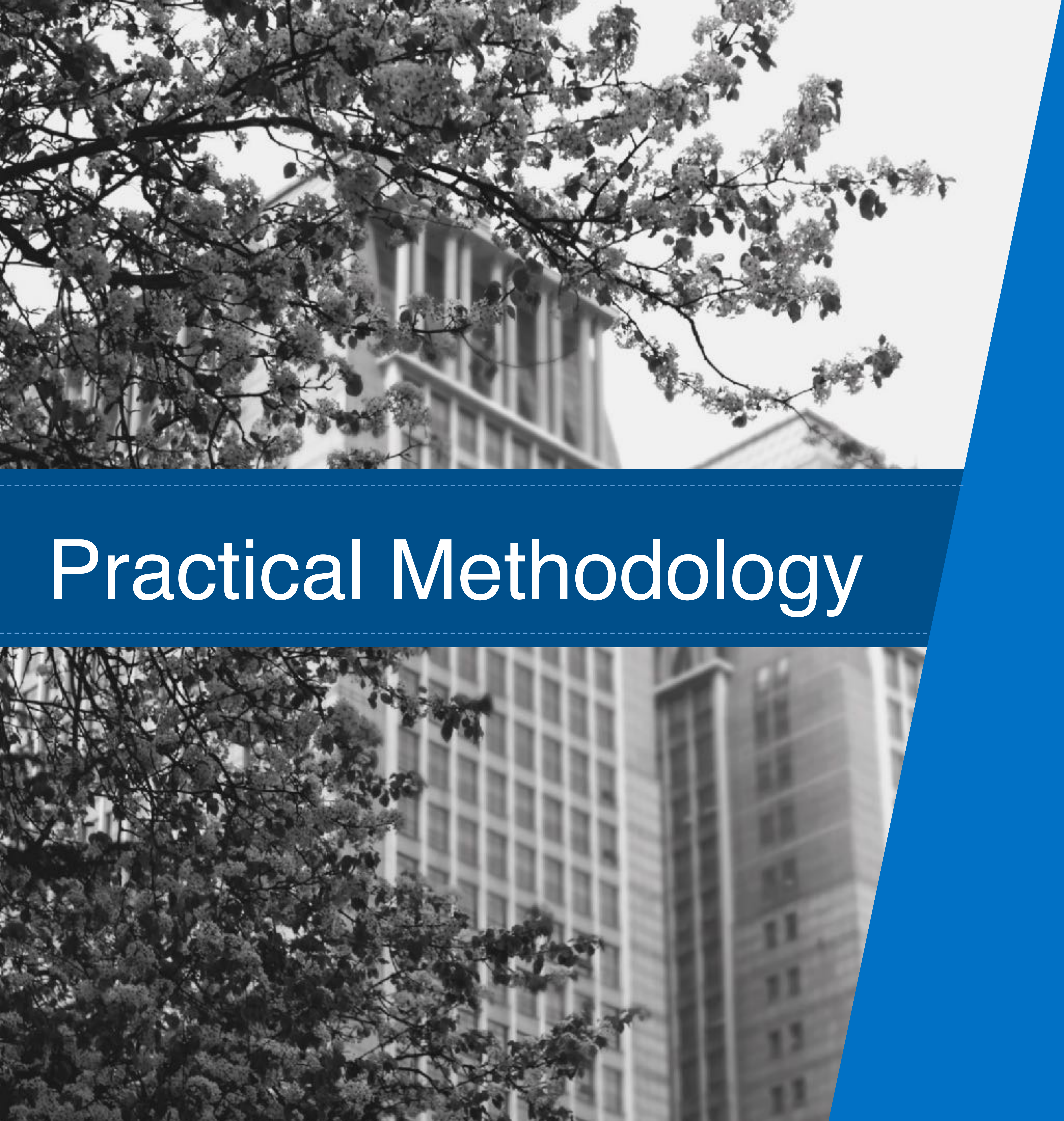


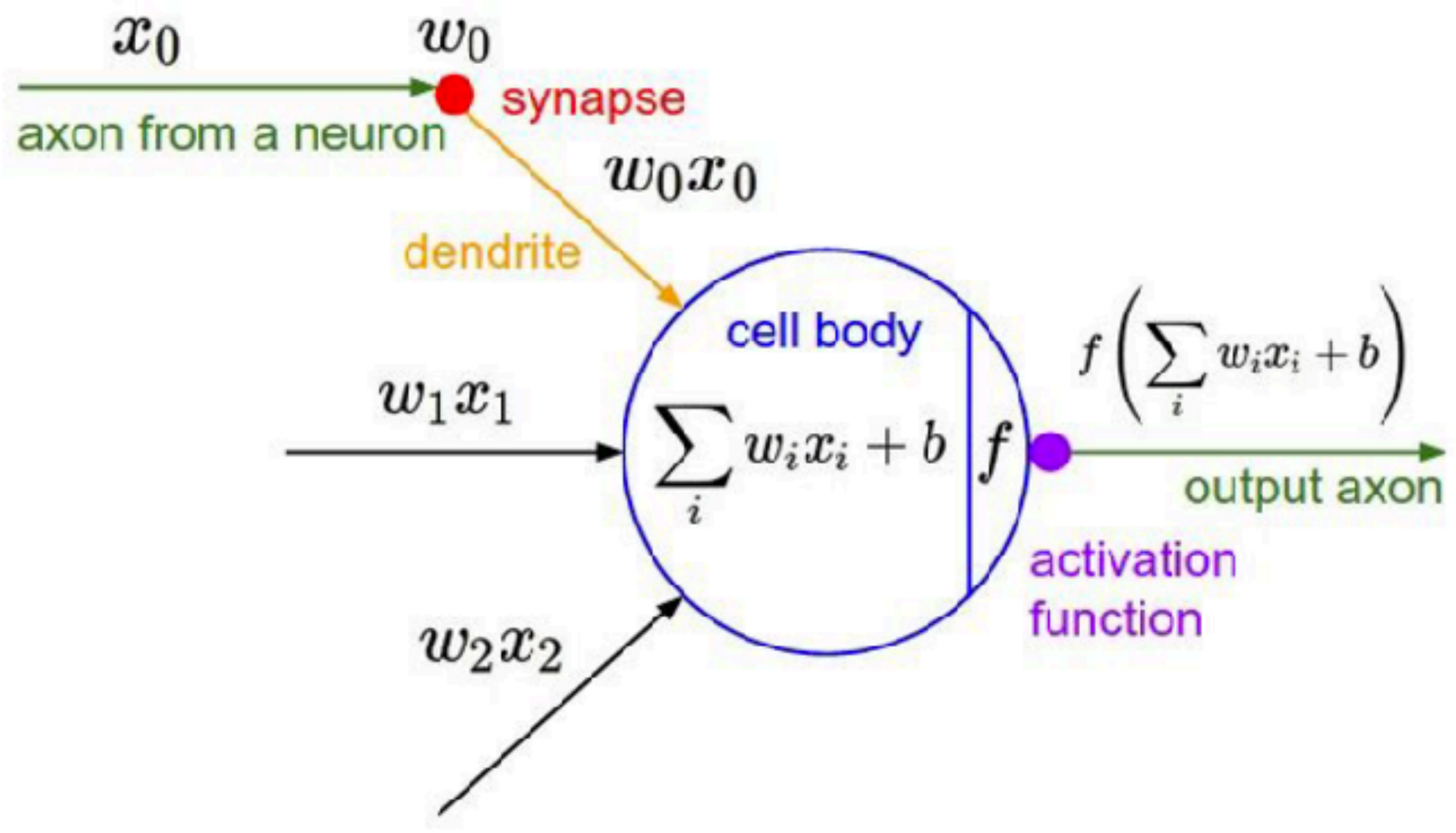
Figure 3: Two choices for the architecture of the generator. The "U-Net" [34] is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.



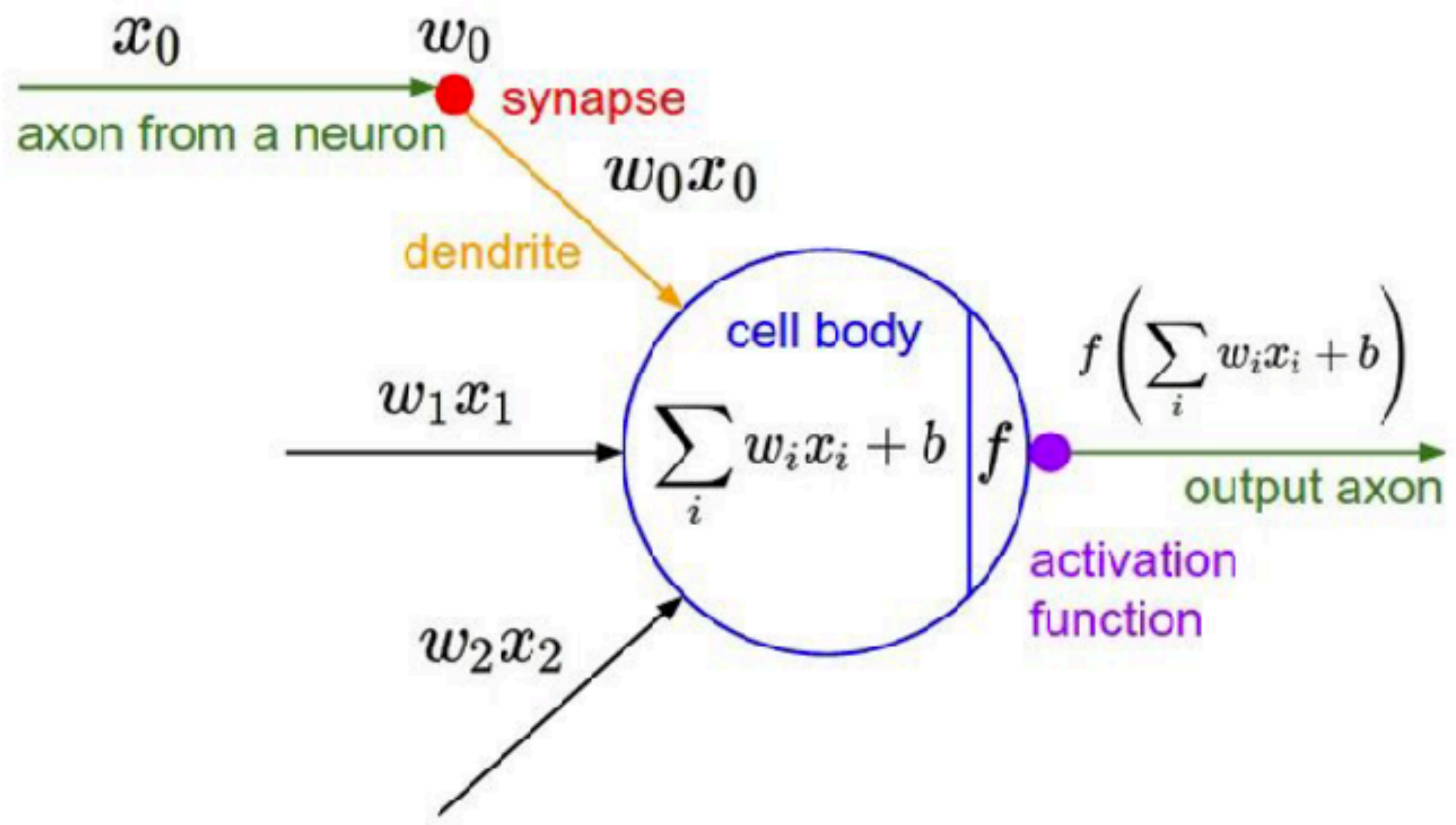
2

Practical Methodology

Activation Functions

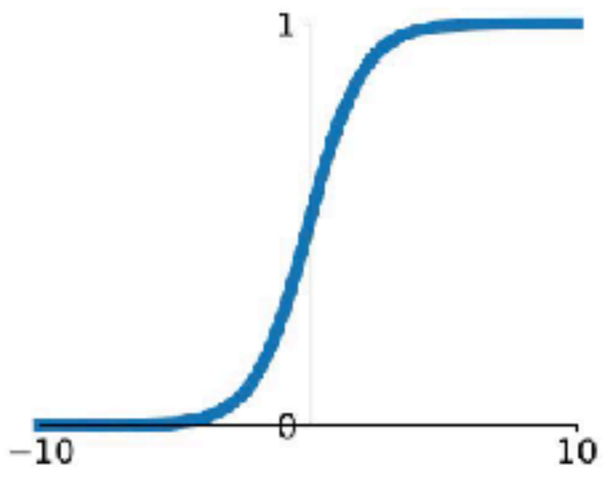


Activation Functions



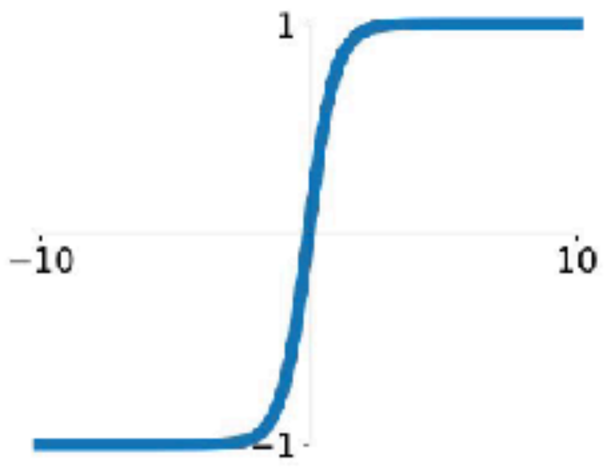
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



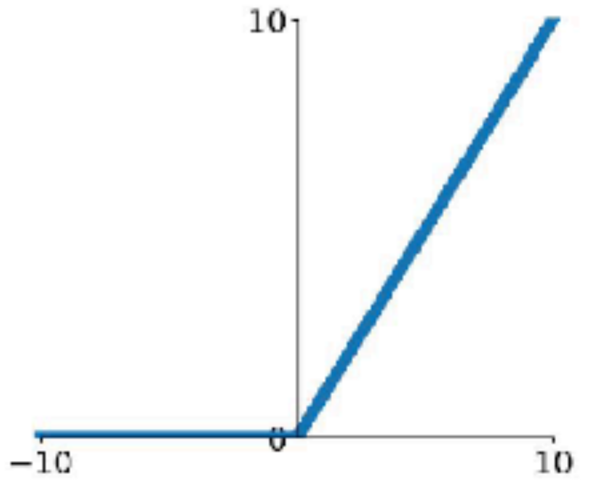
tanh

$\tanh(x)$

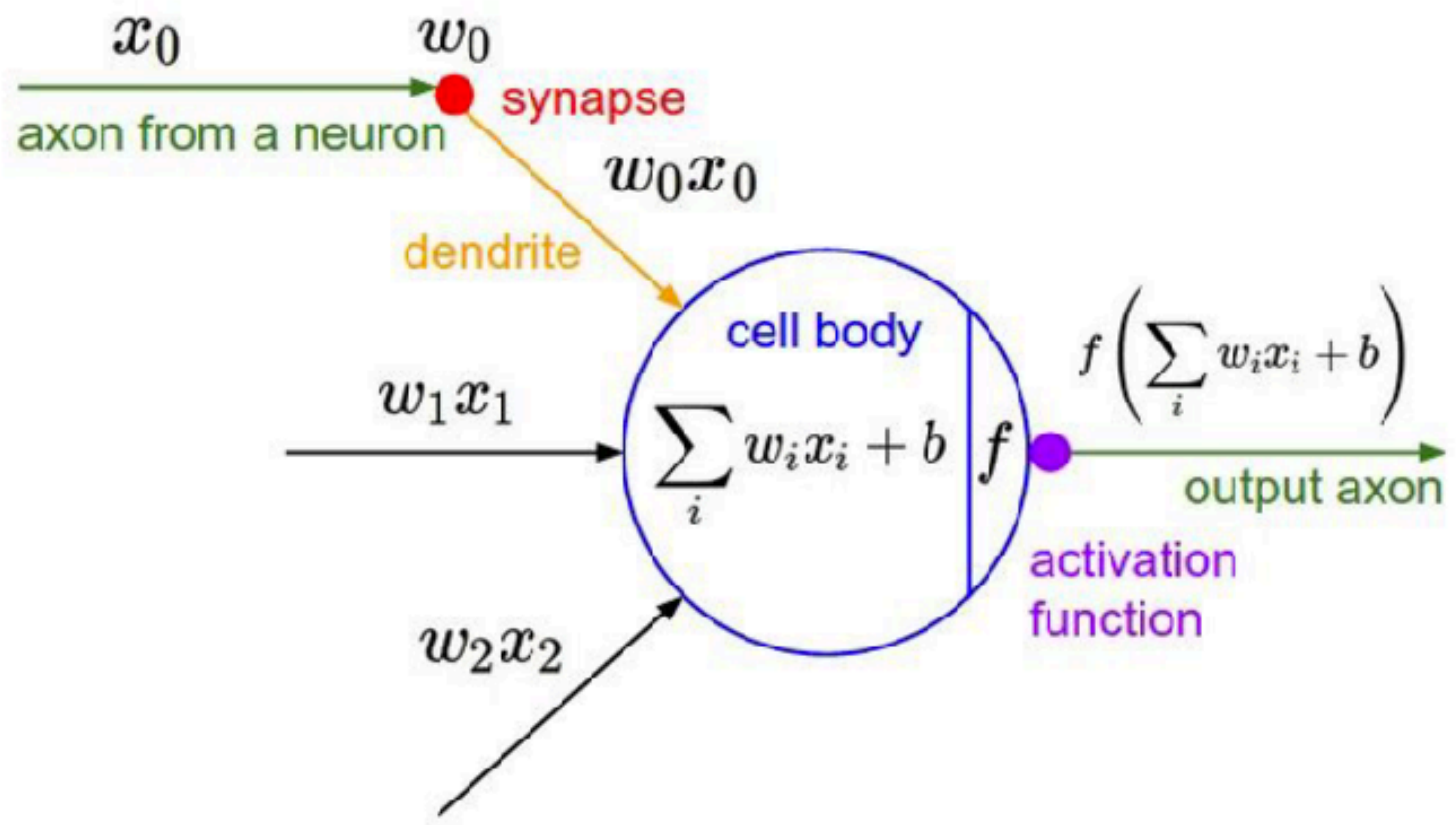


ReLU

$\max(0, x)$

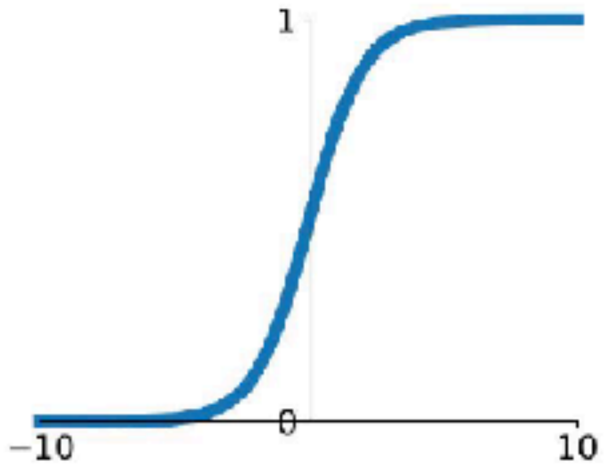


Activation Functions



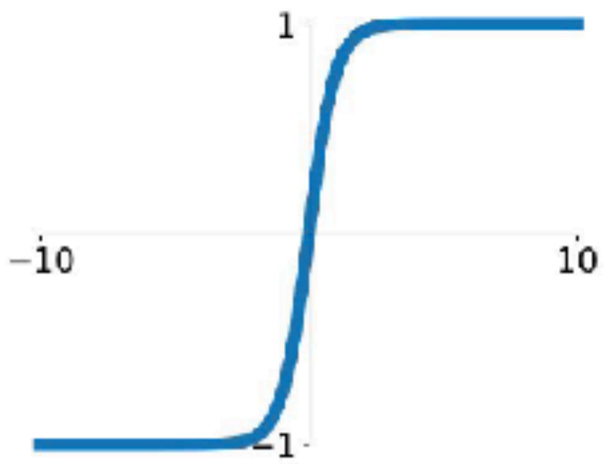
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



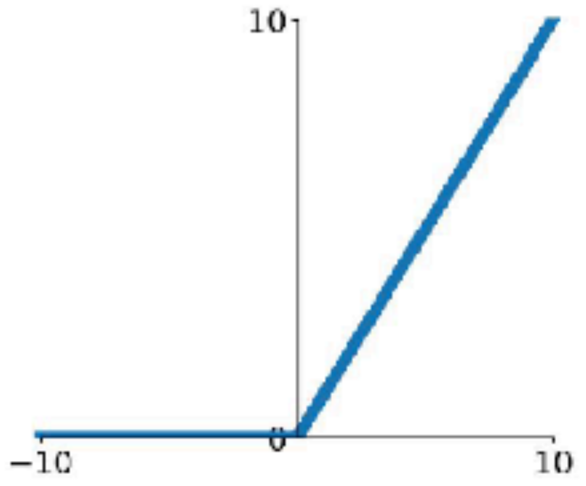
tanh

$\tanh(x)$



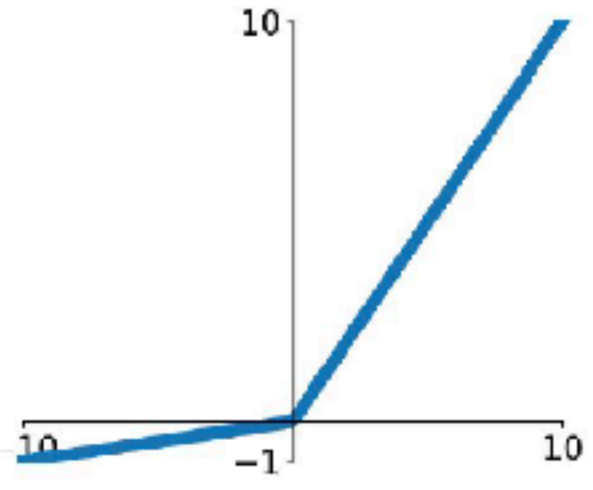
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

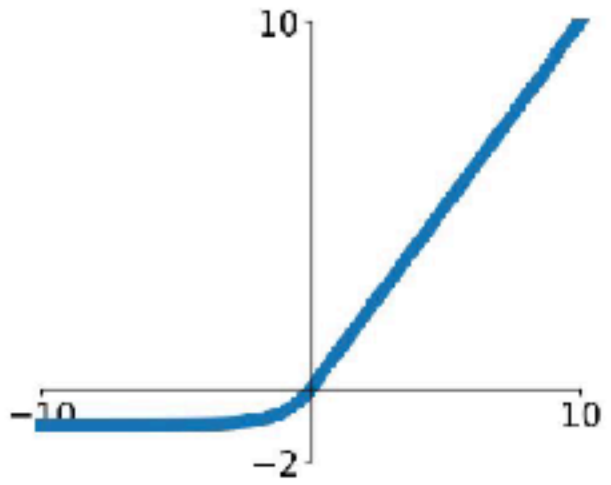


Maxout

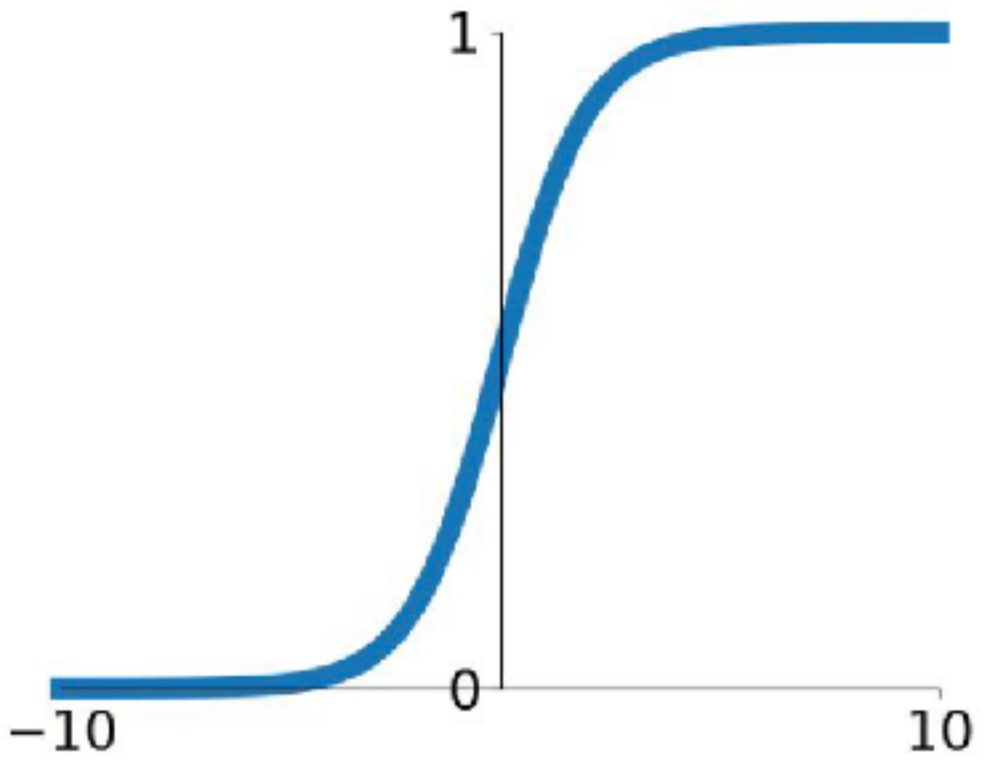
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions



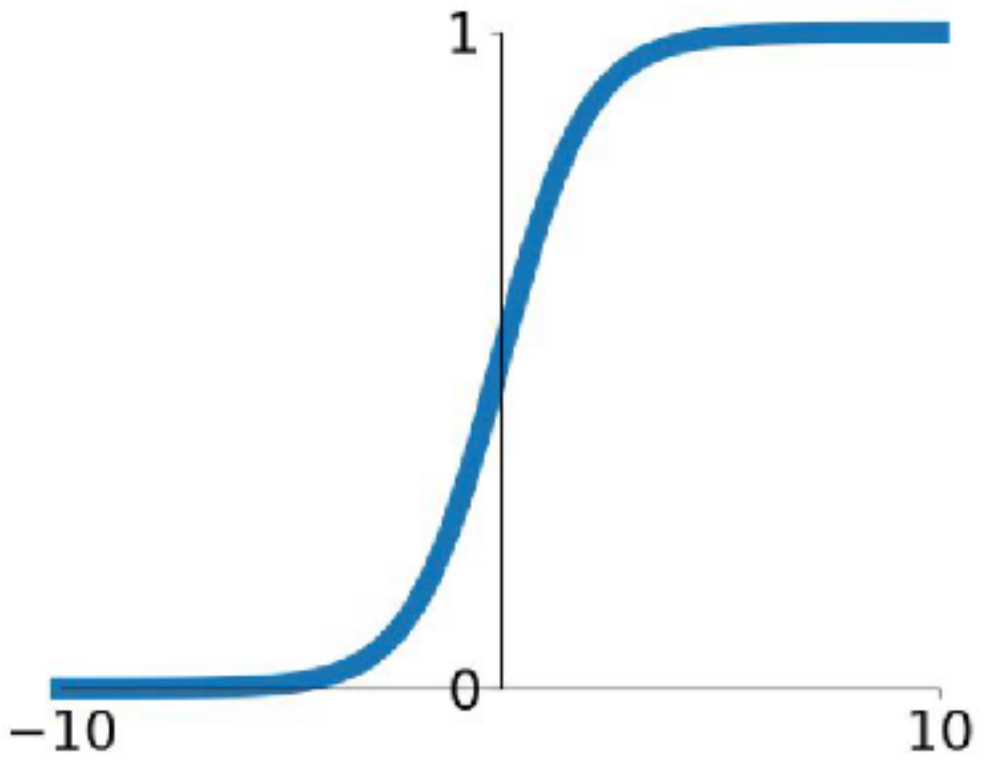
Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive

Activation Functions

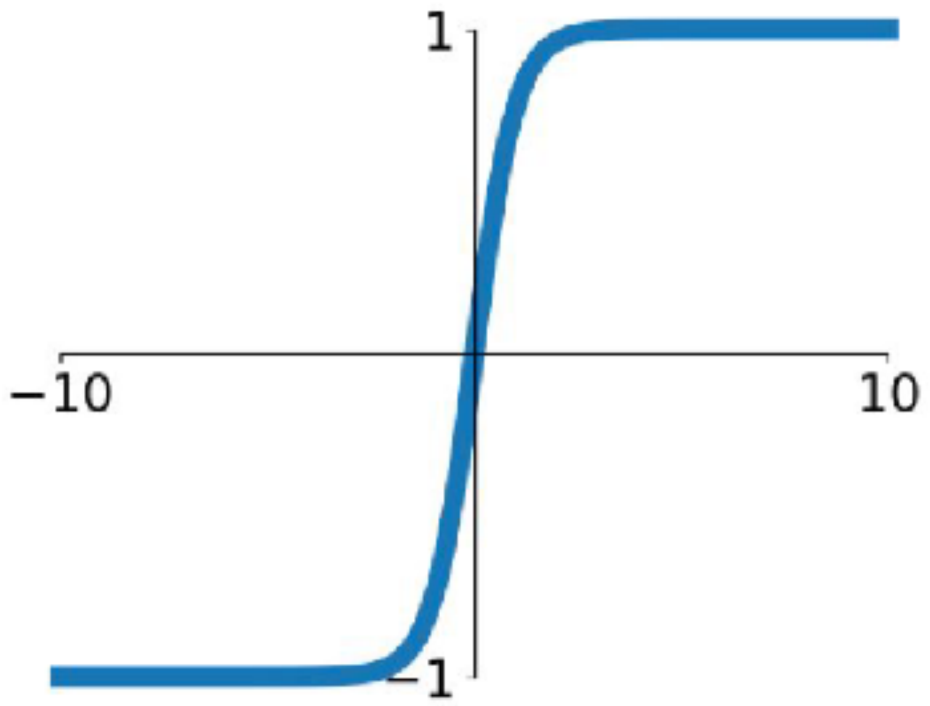


Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating "firing rate" of a neuron

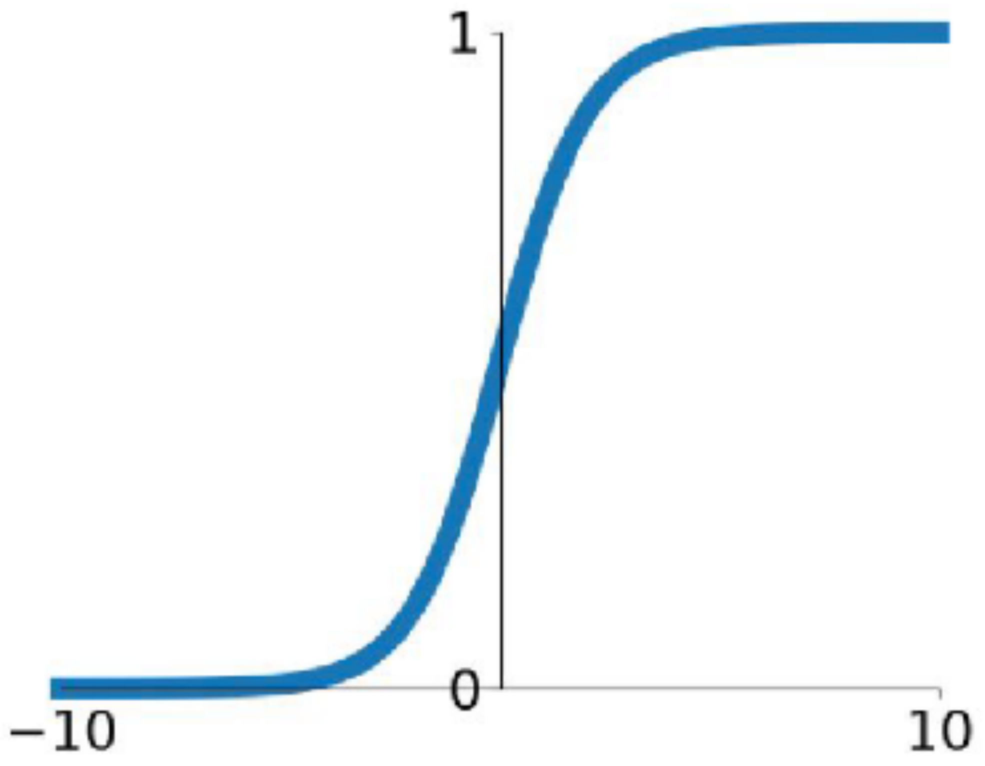
1. Saturated neurons "kill" the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive



tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

Activation Functions

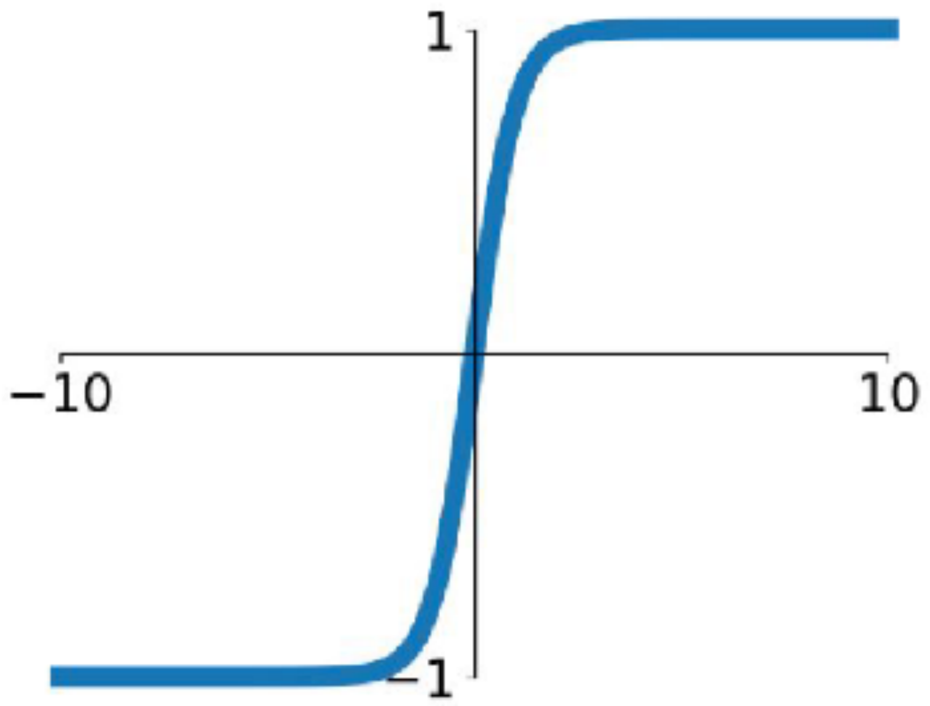


Sigmoid

$$\sigma(x) = 1/(1+e^{-x})$$

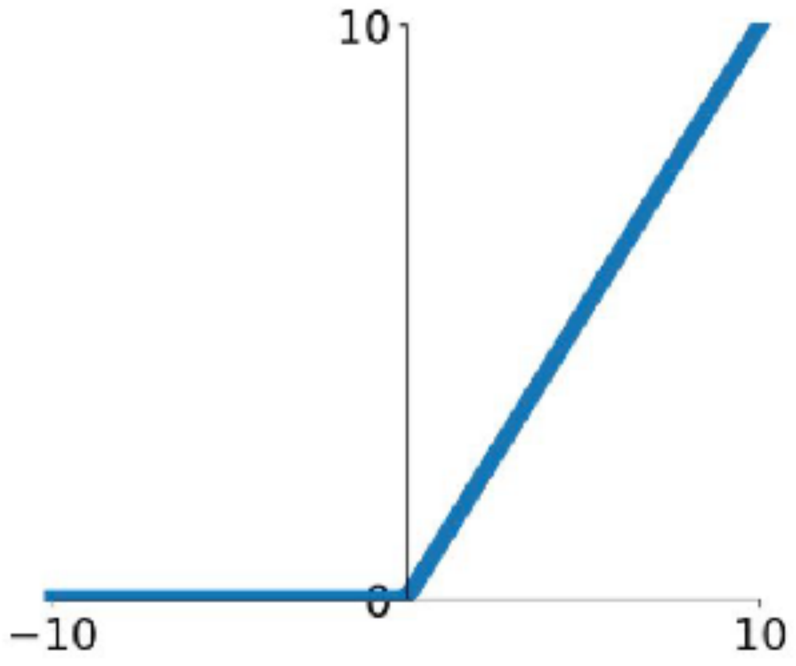
- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. exp() is a bit compute expensive



tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(



ReLU
(Rectified Linear Unit)

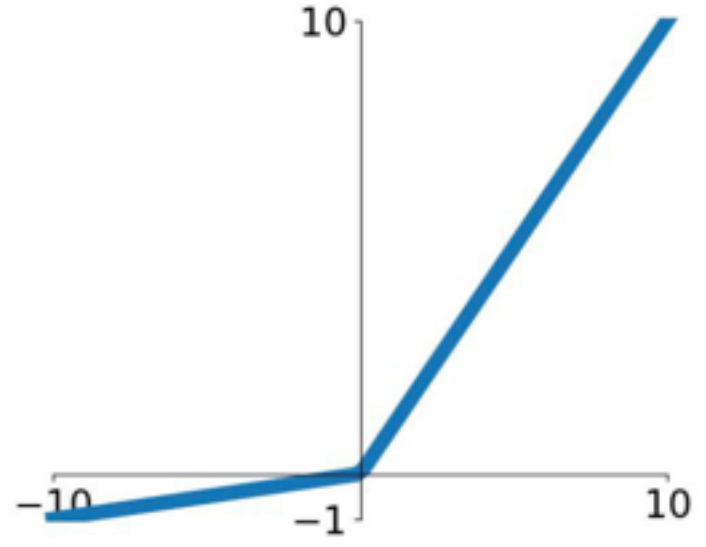
- Computes **f(x) = max(0,x)**

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

- Not zero-centered output
- An annoyance:

  dead ReLU
  will never activate
  => never update

Activation Functions



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x
- **will not “die”.**

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

Maxout “Neuron”

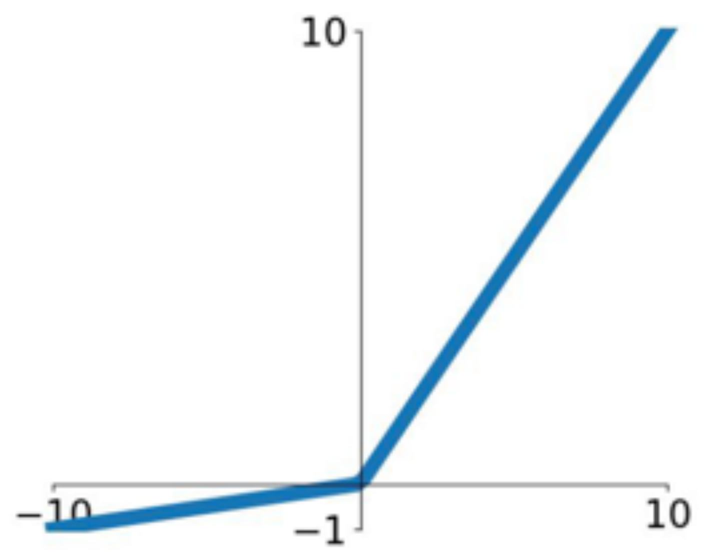
[Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters/neuron :(

Activation Functions



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x
- **will not "die".**

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

Maxout "Neuron"

[Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

Problem: doubles the number of parameters/neuron :(

In practice:

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don't expect much
- Don't use sigmoid

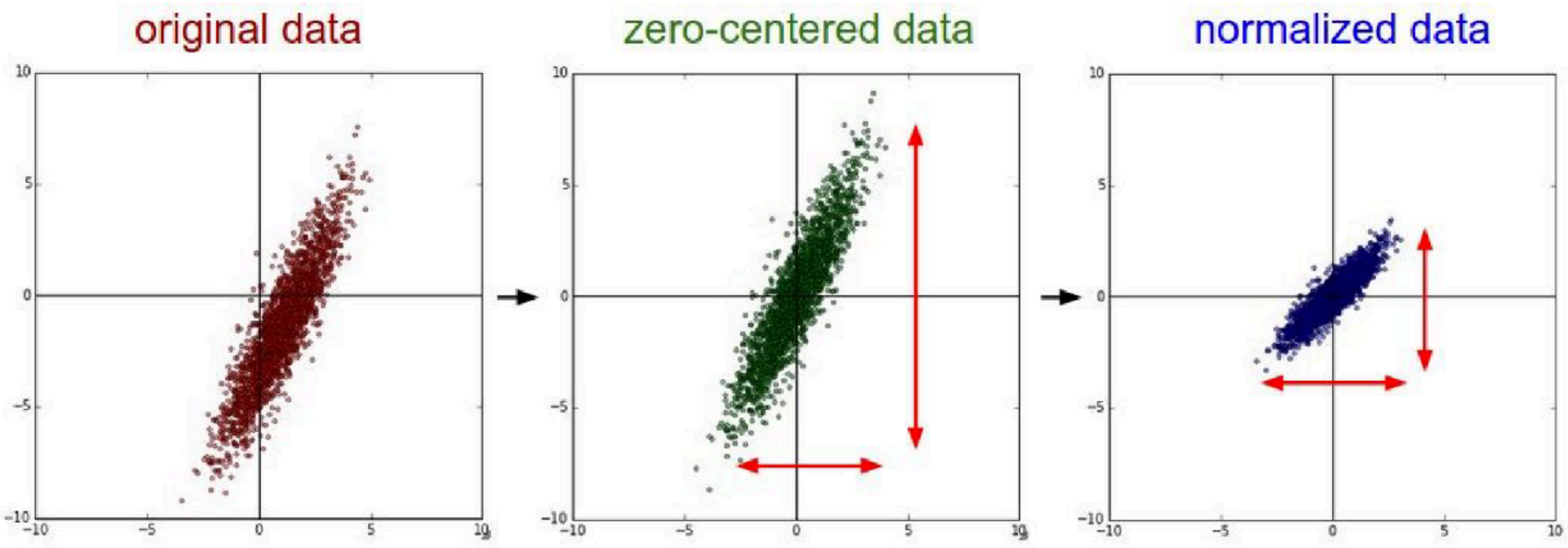
Babysitting the Learning Process

Three Step Process

- Use needs to define metric-based goals
- Build an end-to-end system
- Data-driven refinement

How to prepare the data?

Step 1: Preprocess the data



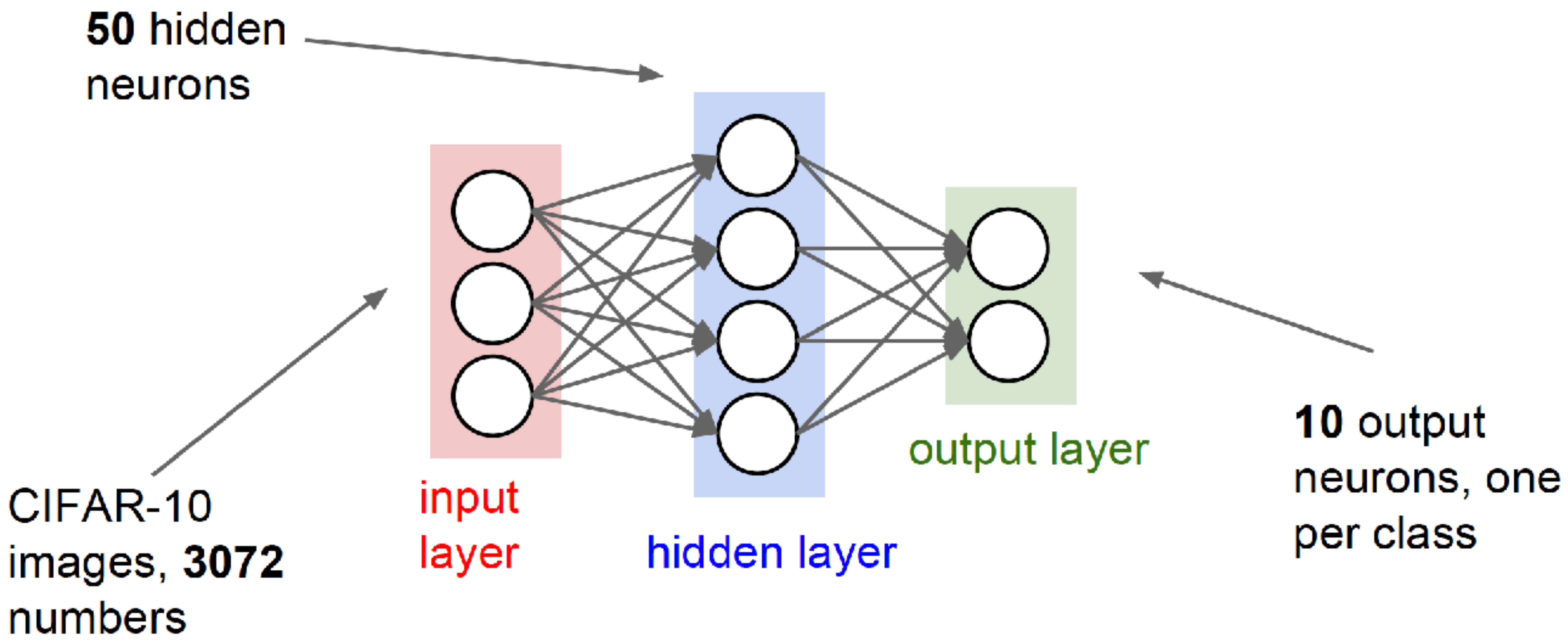

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Which model to use?

Step 2: Choose the architecture:
say we start with one hidden layer of 50 neurons:



Choose Metrics

- Accuracy? (% of examples correct)
- Coverage? (% of examples processed)
- Precision? (% of detections that are right)
- Recall? (% of objects detected)
- Amount of error? (For regression problems)

Is the Loss Reasonable?(1)

Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0)
print loss
```

```
2.30261216167
```

disable regularization

loss ~2.3.
"correct " for
10 classes

returns the loss and the
gradient for all parameters

Is the Loss Reasonable?(2)

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
print loss
```

crank up regularization

```
3.06859716482
```

loss went up, good. (sanity check)

Then, let's try to train it.

Lets try to train now…

Tip: Make sure that you can overfit very small portion of the training data

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

The above code:

- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla 'sgd'

Lets try to train now…

Tip: Make sure that you can overfit very small portion of the training data

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03
```

```
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

Very small loss, train accuracy 1.00, nice!

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing

Notice train/val accuracy goes to 20%

Now let's try learning rate 1e6

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low
loss exploding:
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

```
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
```

```
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

cost: NaN almost always means high learning rate...

Lets try to train now…

Start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low
loss exploding:
learning rate too high

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

3e-3 is still too high. Cost explodes….

=> Rough range for learning rate we should be cross-validating is somewhere [1e-3 … 1e-5]

How to do Cross Validation?

coarse -> fine cross-validation in stages

First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
… (repeat as necessary)

Tip for detecting explosions in the solver:
If the cost is ever > 3 * original cost, break out early

For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=5, reg=reg,
                                   update='momentum', learning_rate_decay=0.9,
                                   sample_batches = True, batch_size = 100,
                                   learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.