Imperial College London

Deep Learning within Computer Vision

a whirlwind tour of the key principles

Tae-Kyun (T-K) Kim
Senior Lecturer
https://labicvl.github.io/

Slide material acknowledgement: Toby Breckon (Durham University), at BMVA computer vision summer school 2018

The move from **shallow** to **deep** …

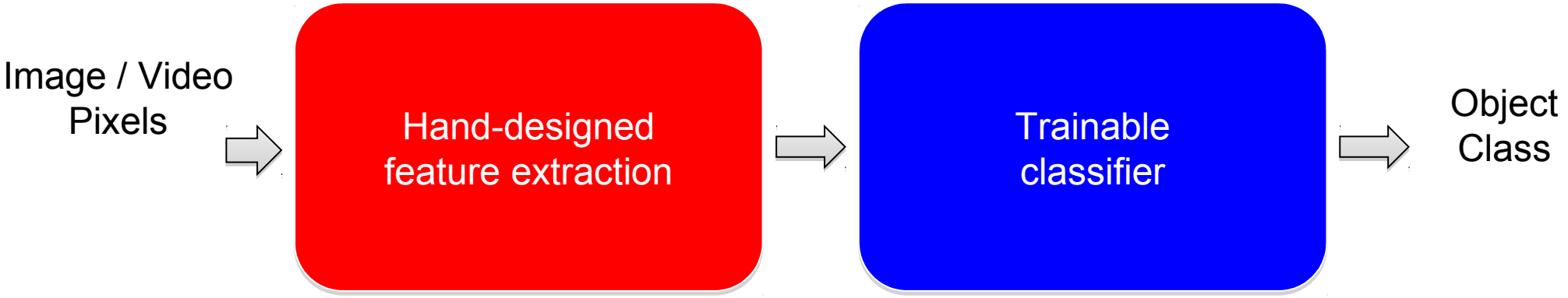
(< ~2013)

pre deep learning / *pre abyssi*

(> ~2013)

post deep learning / *post abyssi*

Traditional (shallow) Approaches



- Features are not learned
  - vectors of shape measures, edge distributions, colours distributions, feature points, HOG, visual words, ... etc.
    - … i.e. calculated **summary** "numerical" **descriptors**

- Trainable classifier is often generic

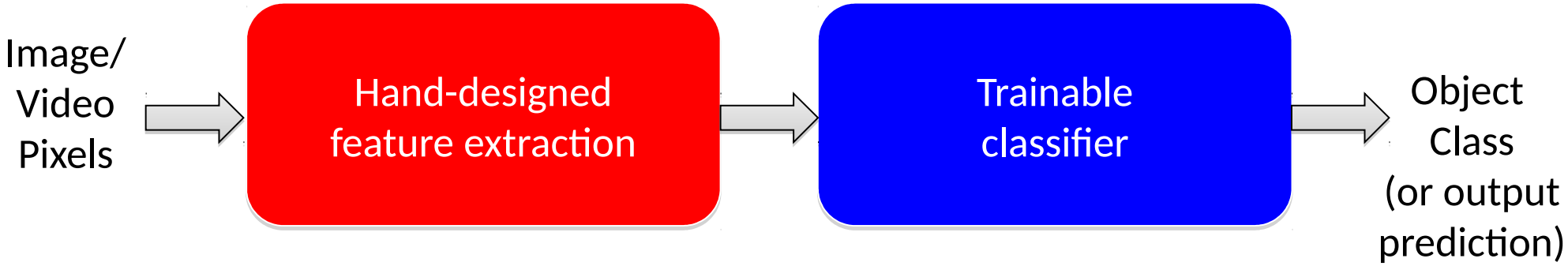
(e.g. SVM kernel, Decision Forest)

Deep Learning – end to end approaches

- Learn a *feature hierarchy* all the way from pixels (or voxels) to classifier
- Each layer extracts *"features"* from the output of previous layer
- Layers have similar structure, performing varying functions
- Train (i.e. optimize) all layers jointly

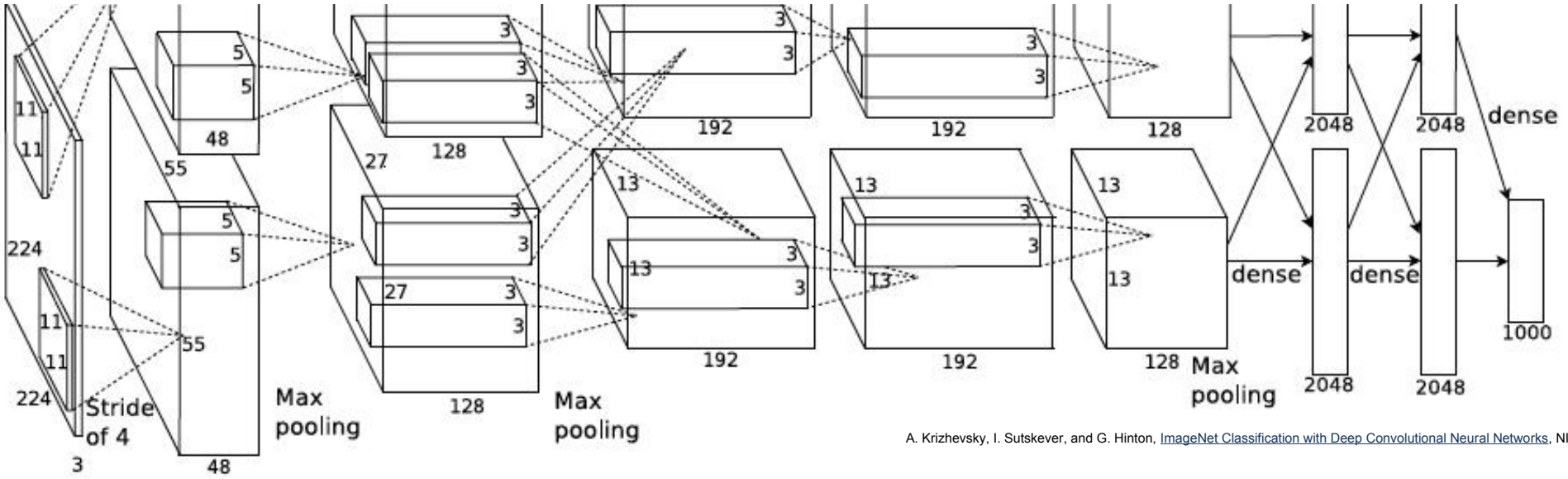
“Shallow” vs. “deep” architectures

Traditional recognition: **“Shallow”** architecture



(modern) Deep learning: **“Deep”** architecture

A *typical* end-to-end deep learning **convolutional neural network** ….

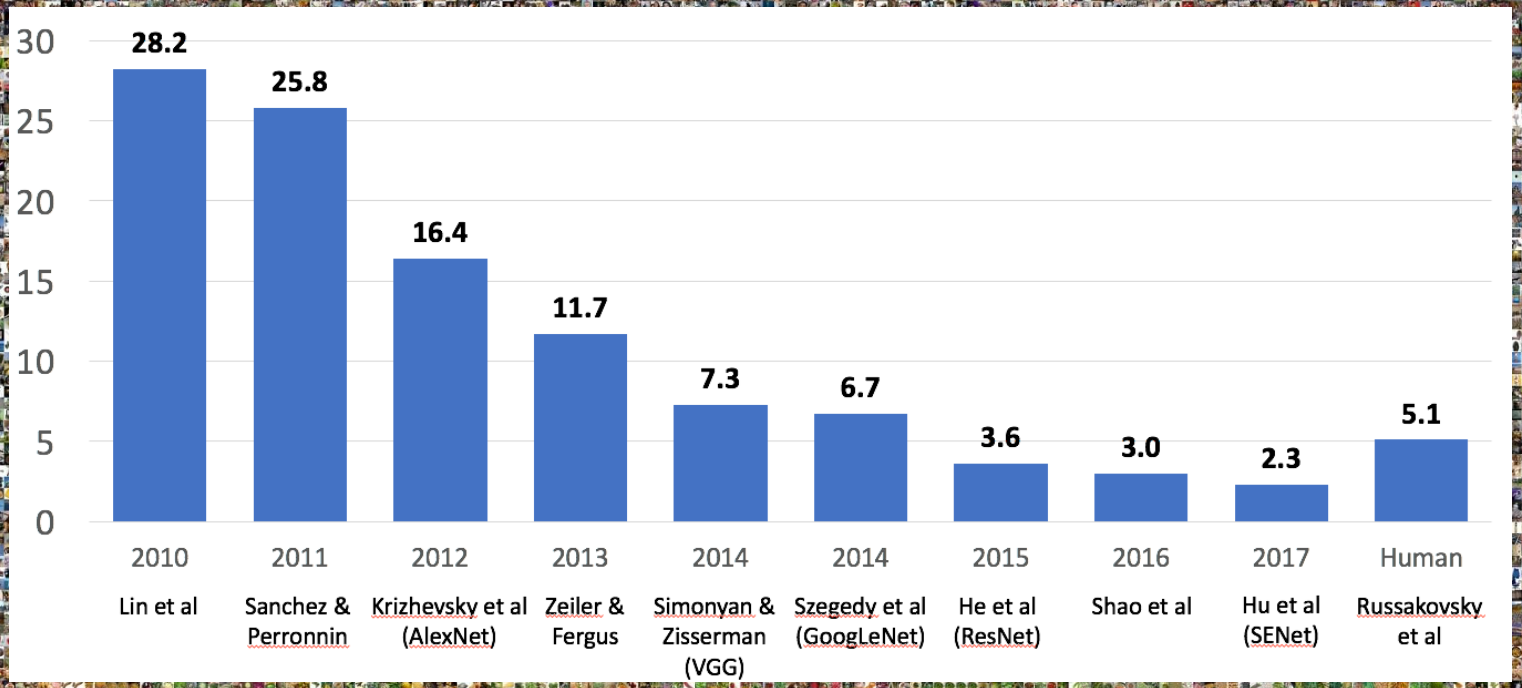


A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

- **"AlexNet":** seminal ImageNet Challenge winner (2012)
  - **Bigger** model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More **data** (10^6 vs. 10^3 images)
  - **GPU** implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - **Algorithms:** better regularization for training (DropOut)

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:
1,000 object classes
1,431,167 images



[Human - Russakovsky et al. IJCV 2015]

[This slide – from: Fei-Fei Li & Justin Johnson & Serena Yeung]

AlexNet's performance on this benchmark task was the research event ("discovery") that led to the **shallow** to **deep** transformation in computer vision ...*

* although CNN were not created overnight, and have their origins in [LeCun, 1998] among others

Key question – **why does this stuff work so well ?**

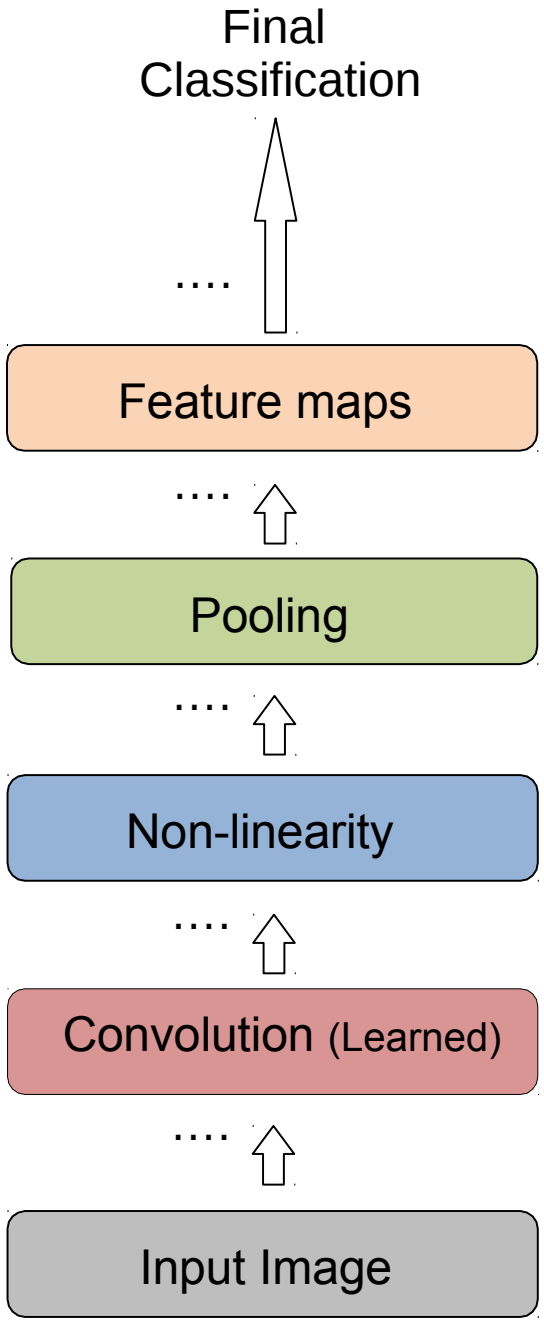
(let's examine some of the fundamentals)

Key **Principle**

Each layer in a **deep network performs a different transformation (function)** to map from input to output – these vary from {convolution, pooling, sub-sampling, non-linear mapping (fully connected), ….}.

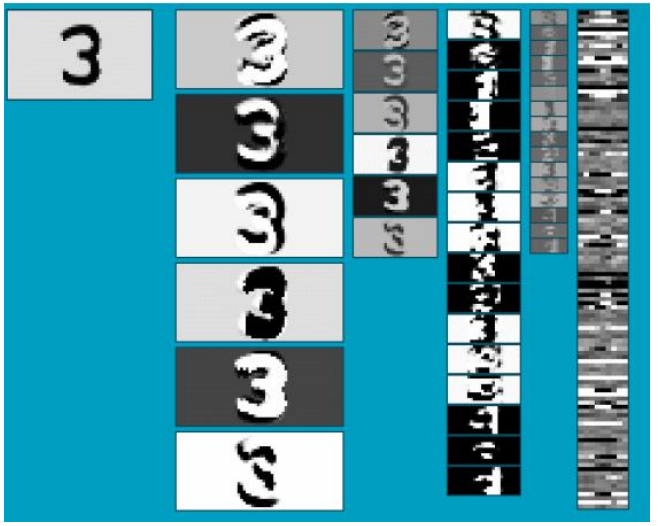
Within a **traditional (shallow) Neural Network**, the perceptron **activation functions are all the same** (but we vary the weights).

This provides the network with a **larger parameterization space to represent (complex) input to output relationships.**

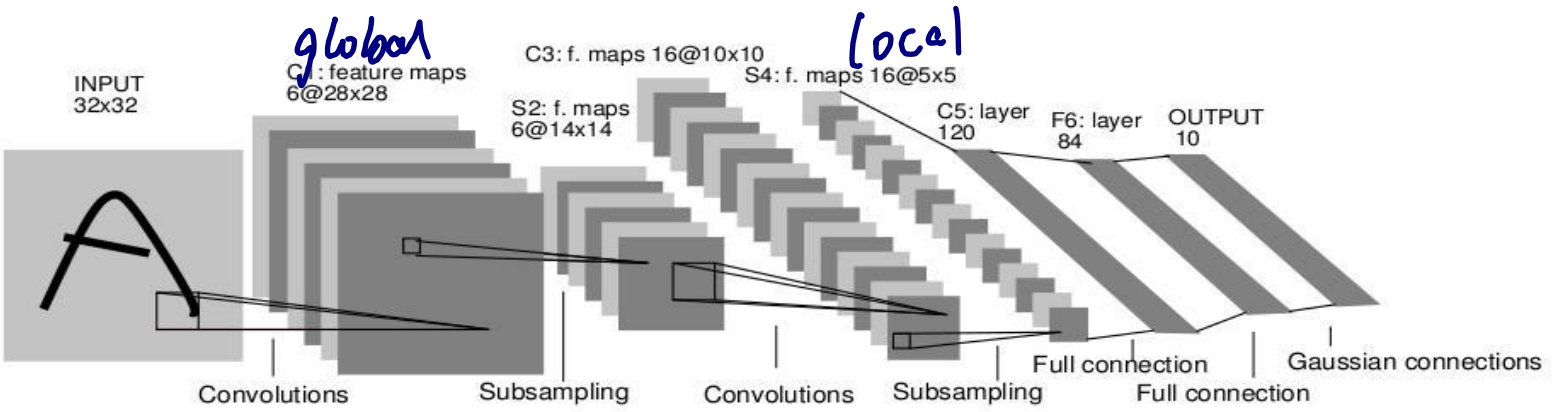


Convolutional Neural Networks (CNN)

- Multi-layer Neural network with:
  - **Local** connectivity
  - **Shared** weight parameters across spatial positions
- Stack **multiple stages of feature extractors** operating **directly on the image**
- Higher **stages compute more global, more invariant feature representation**
- **Final classification** layer at the end



Task: digit recognition



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86(11): 2278–2324, 1998.

Clarification:

Convolutional Neural Networks (CNN)
(i.e. networks using convolution layers)

are **a subset of** the **generalized**
Deep Learning extension to
Neural Networks

(i.e. **Deep Neural Networks**)

CNN are a specific to images (or similar densely sampled signals)**

A deep multi-layer architecture ...

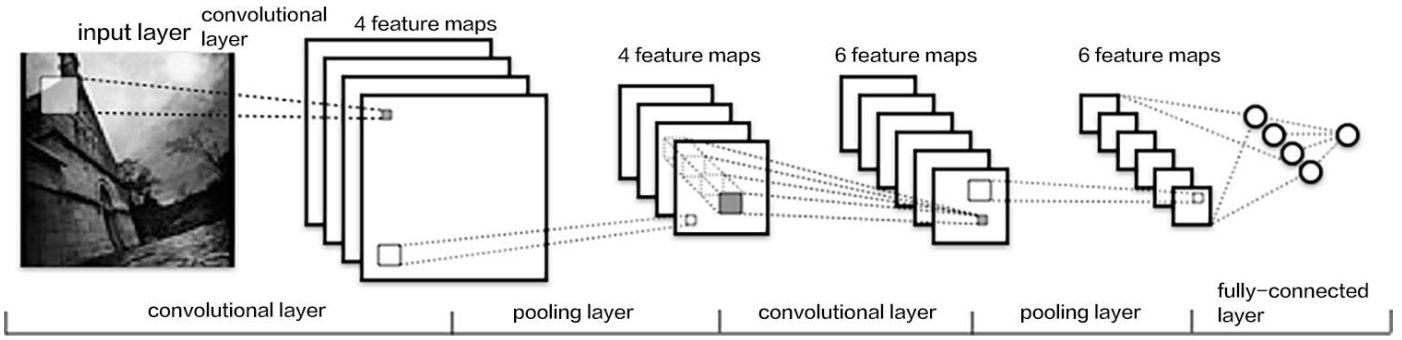
Convolutional Neural Networks (CNN)

- Feed-forward network:
  - Convolve input (feature extraction)
  - Non-linearity (rectified linear)
  - Pooling (local max)

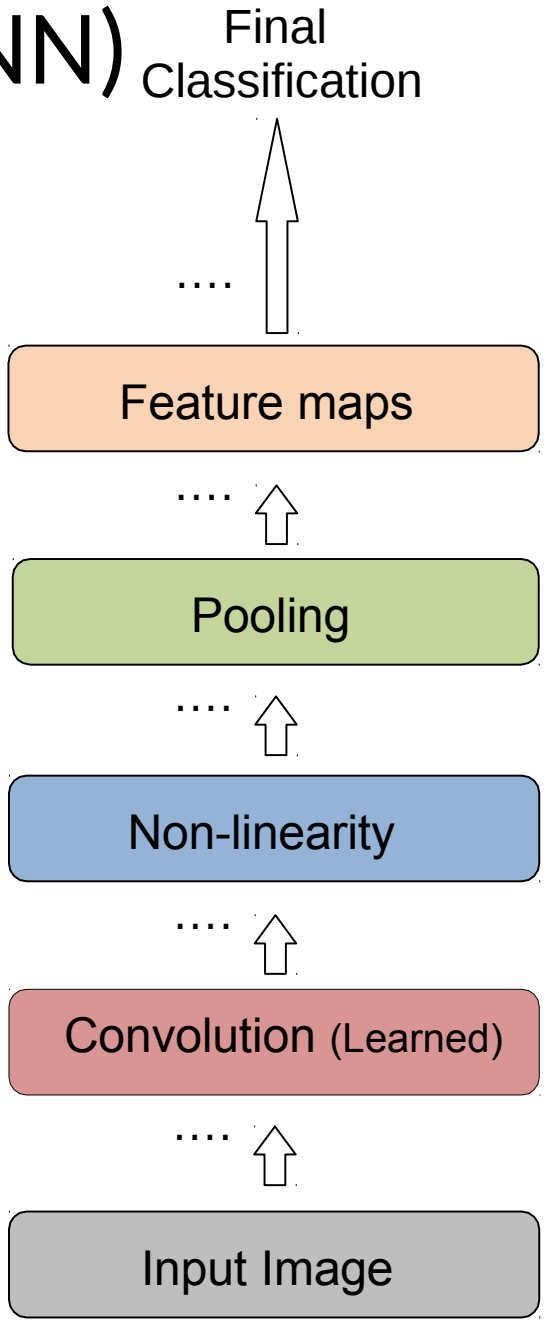
Pooling: down-sampling

Supervised learning (with labels)

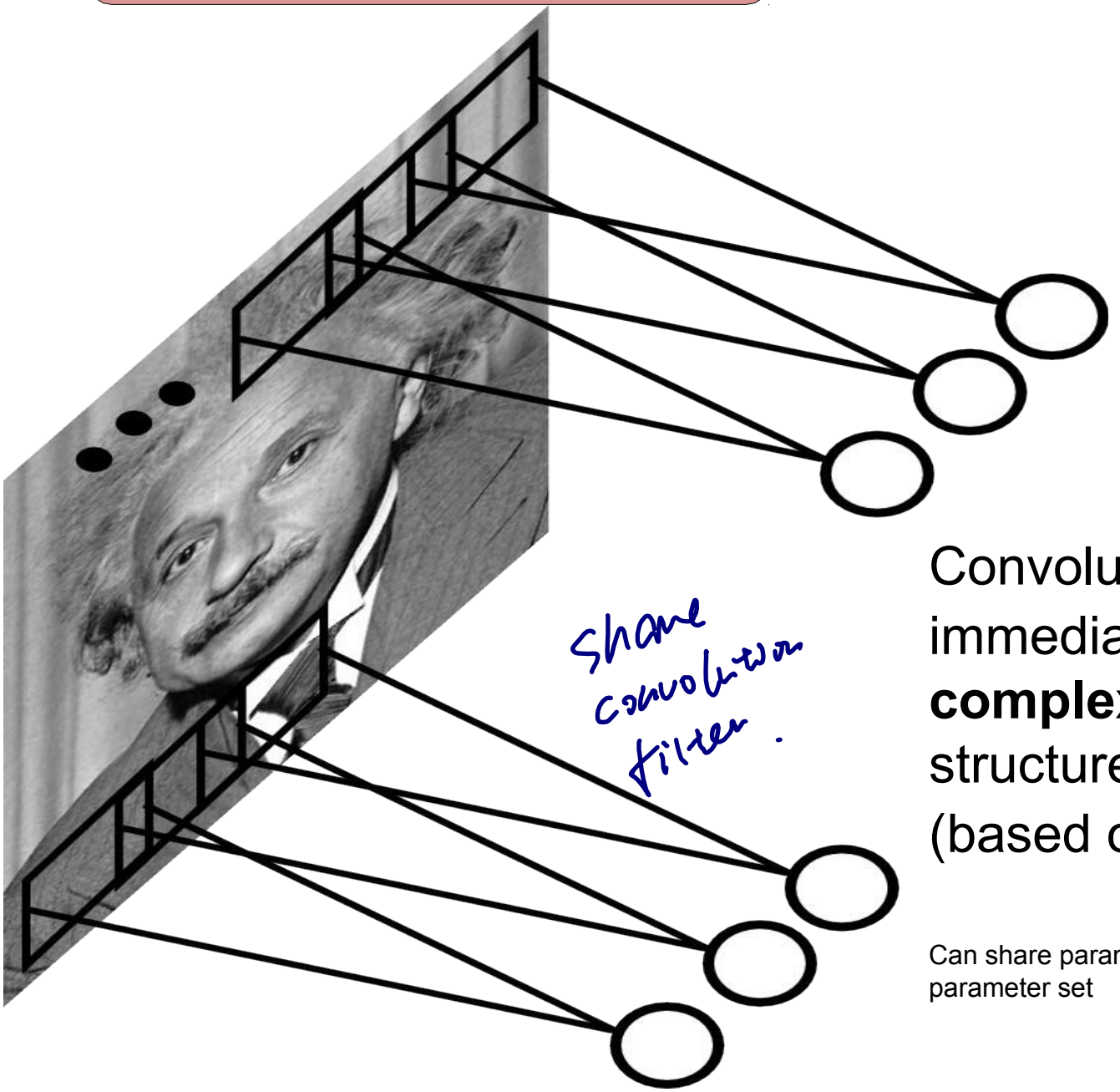
Train convolution filters by backpropogation



Example: LeNet—5 (http://book.paddlepaddle.org/03.image_classification/)



Convolution Layer(s)



Convolution in the first layer immediately **reduces the complexity of the input** in a structured and meaningful way (based on learnt weights)

Can share parameters across filters to reduce to size of parameter set

Convolution Layer(s)

- Convolutional
  - dependencies are local
  - translation invariance
  - few parameters (filter weights)
  - filter stride can be > 1 (faster, less memory)

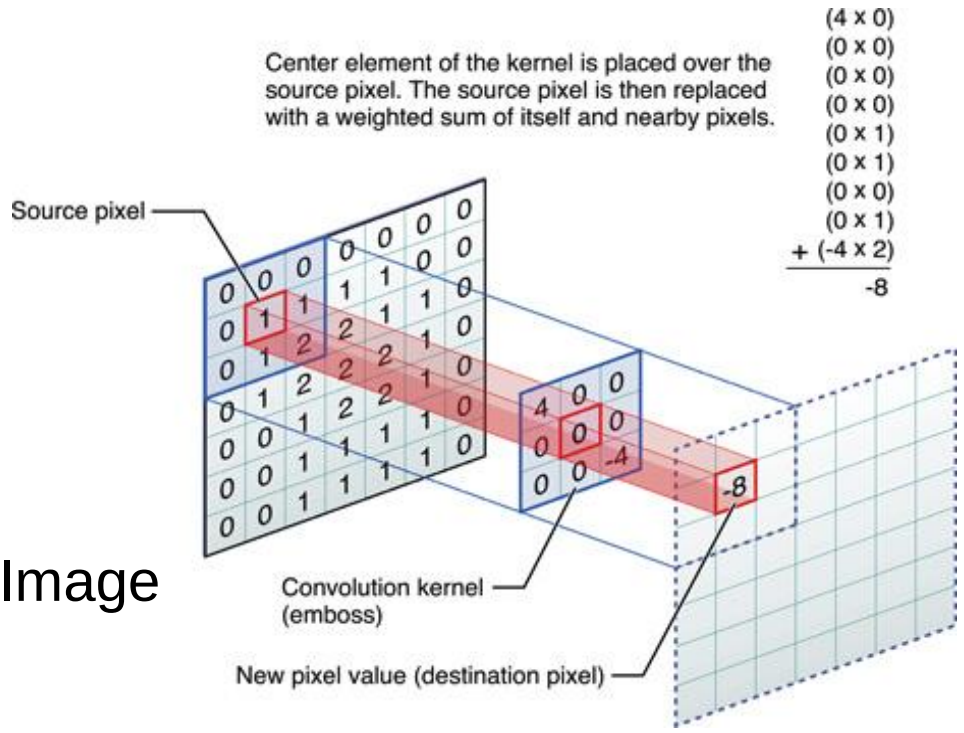
Input (image)

Intermediate Feature Map

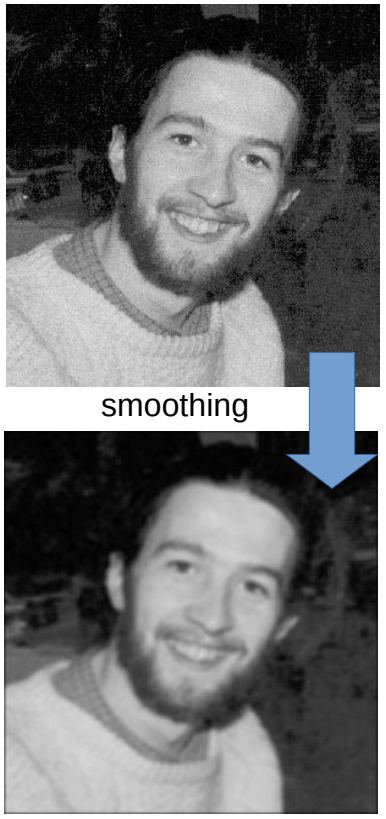
Aside : image convolution

(in general image processing)

- … is essentially the **localised weighted sum of the image and a convolution kernel (mask weights)** over a N x M pixel neighbourhood, at a given location within the image (x,y).



Input Image



Output Image

[used in image filtering operations (e.g. smoothing)]

Image source: developer.apple.com

Convolution is *very powerful* ...

original

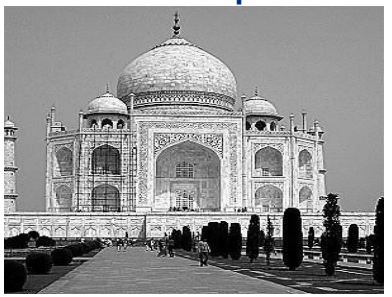
filter (3 x 3)

1	1	1
1	1	1
1	1	1

blur

0	-1	0
1	5	1
0	-1	0

sharpen

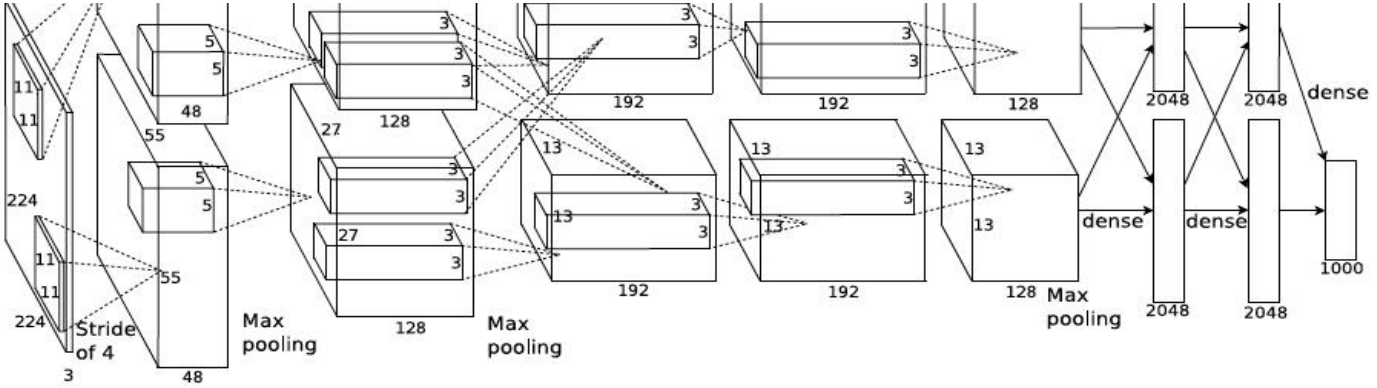


1	2	1
0	0	0
-1	-2	-1

edges

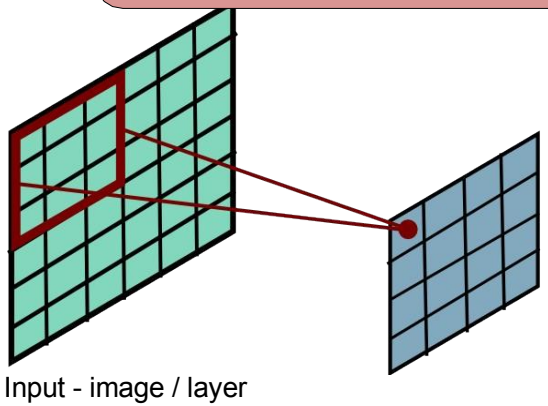
Different weights have do a wide range of effects on input data ….

hence → multiple layers of convolution ("convolutions upon convolutions")



can approximate and provide most **feature extraction** and **image pre-filtering** *(de-noising)* approaches ...

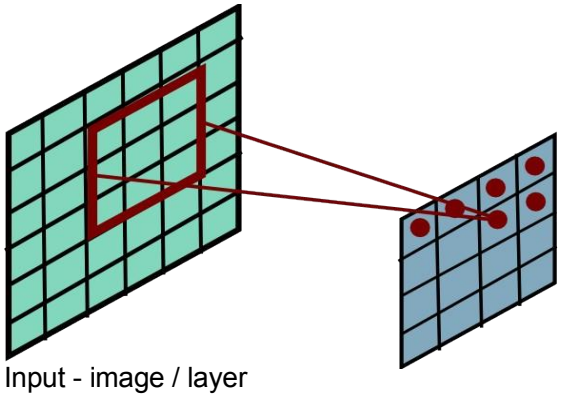
Convolution Layer(s)



Input - image / layer

Output - Feature Map

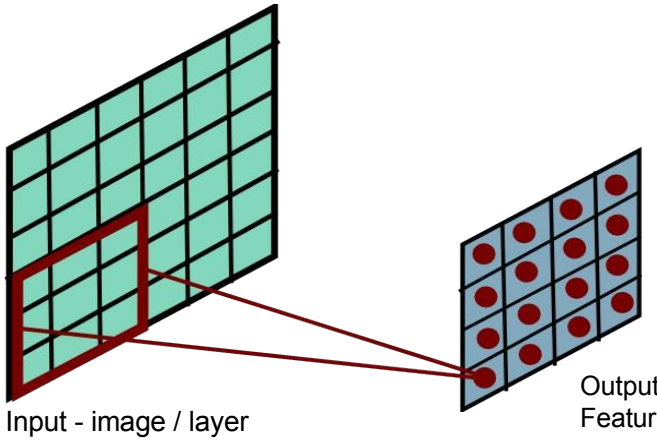
....



Input - image / layer

Output - Feature Map

....



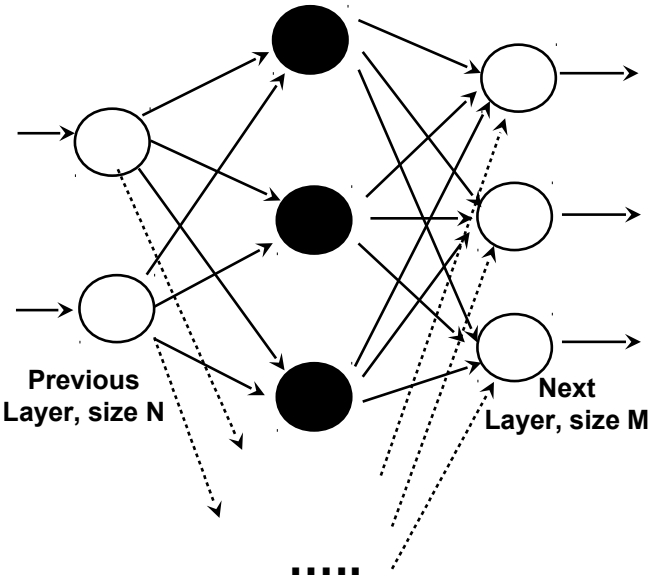
Input - image / layer

Output - Feature Map

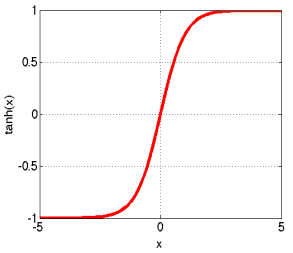
Produces a structured intermediate feature map from the input image (or previous layer in the network)

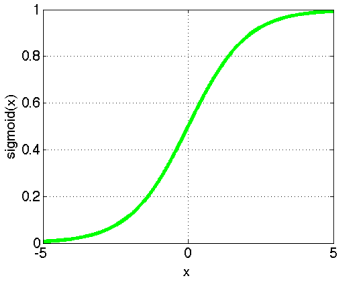
Non-linearity Layer(s)

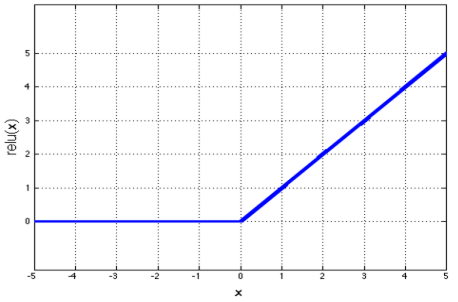
(a.k.a. fully connected)



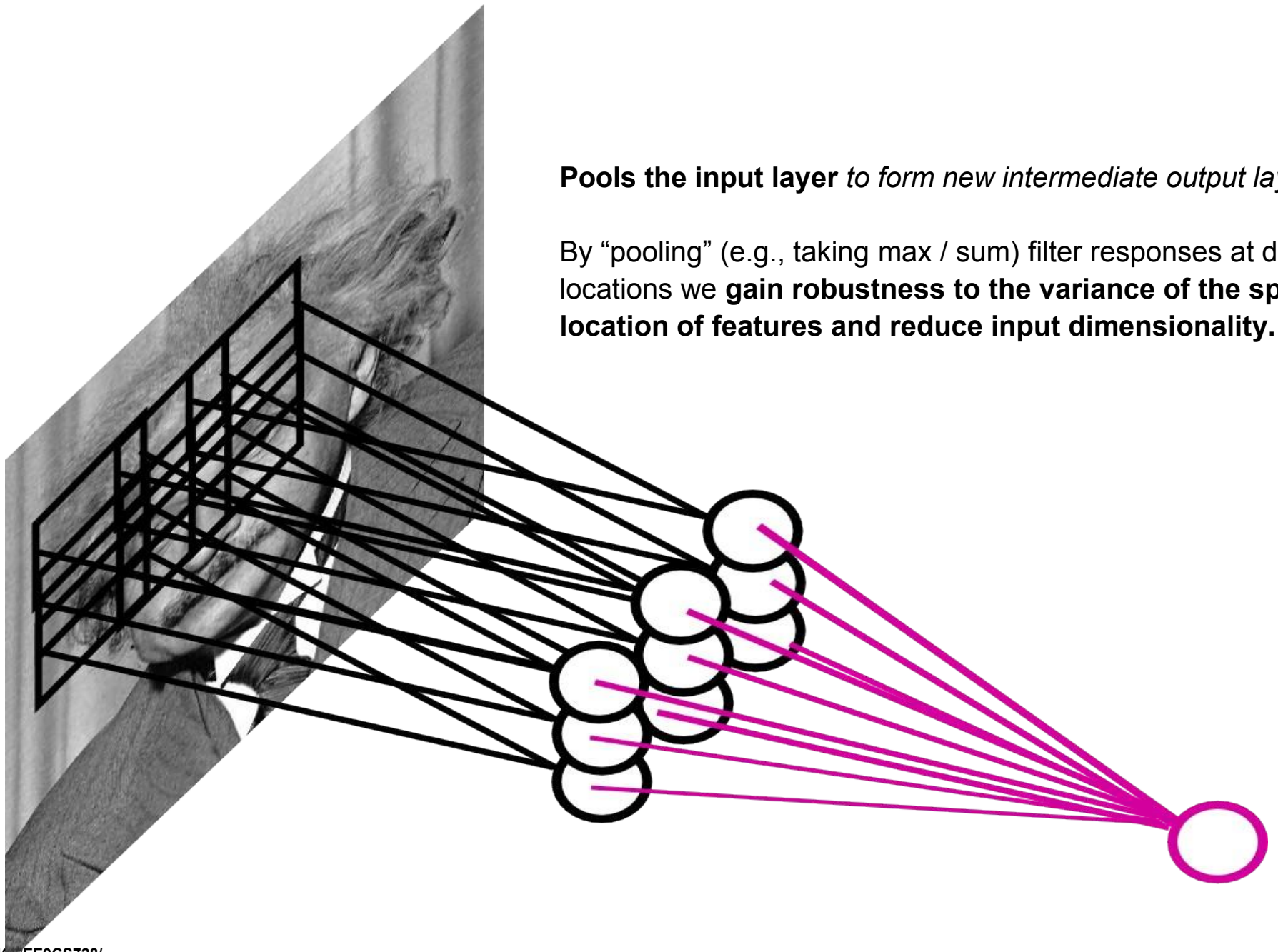
- Provides a non-linear input to output mapping **via a (traditional) activation function approach** layer of neurons
  - maybe either **sub-sampling (N→ M) or fully connected (N → M, N=M)** via per element activation function, e.g.
    - Tanh
    - Sigmoid: 1/(1+exp(-x))
    - Rectified linear (most common)
      - » Simplifies backprop
      - » Makes learning faster
      - » Avoids output saturation issues







Pooling layer(s)

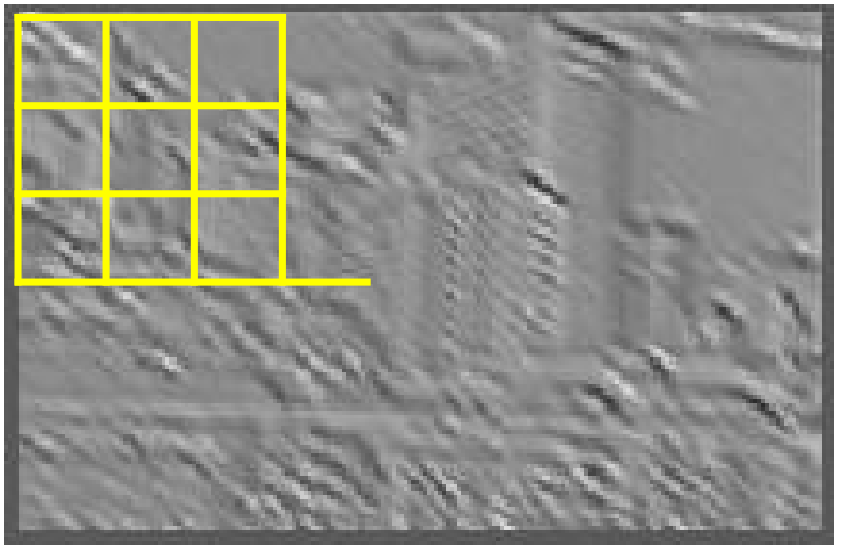


Pools the input layer *to form new intermediate output layer.*

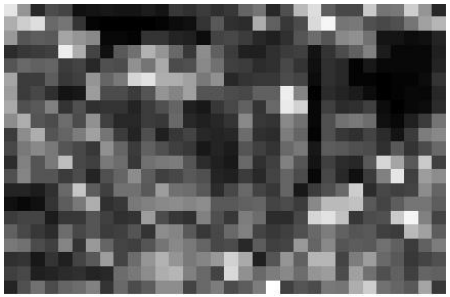
By “pooling” (e.g., taking max / sum) filter responses at different locations we **gain robustness to the variance of the spatial location of features and reduce input dimensionality.**

Pooling layer(s)

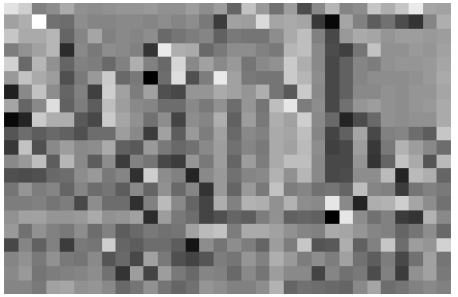
- Performs **localized sum() or max()** over sub windows/regions
  - non-overlapping Vs. overlapping regions
  - Role of pooling:
    - **Invariance** to small transformations
    - **Larger receptive fields** (see more of input)



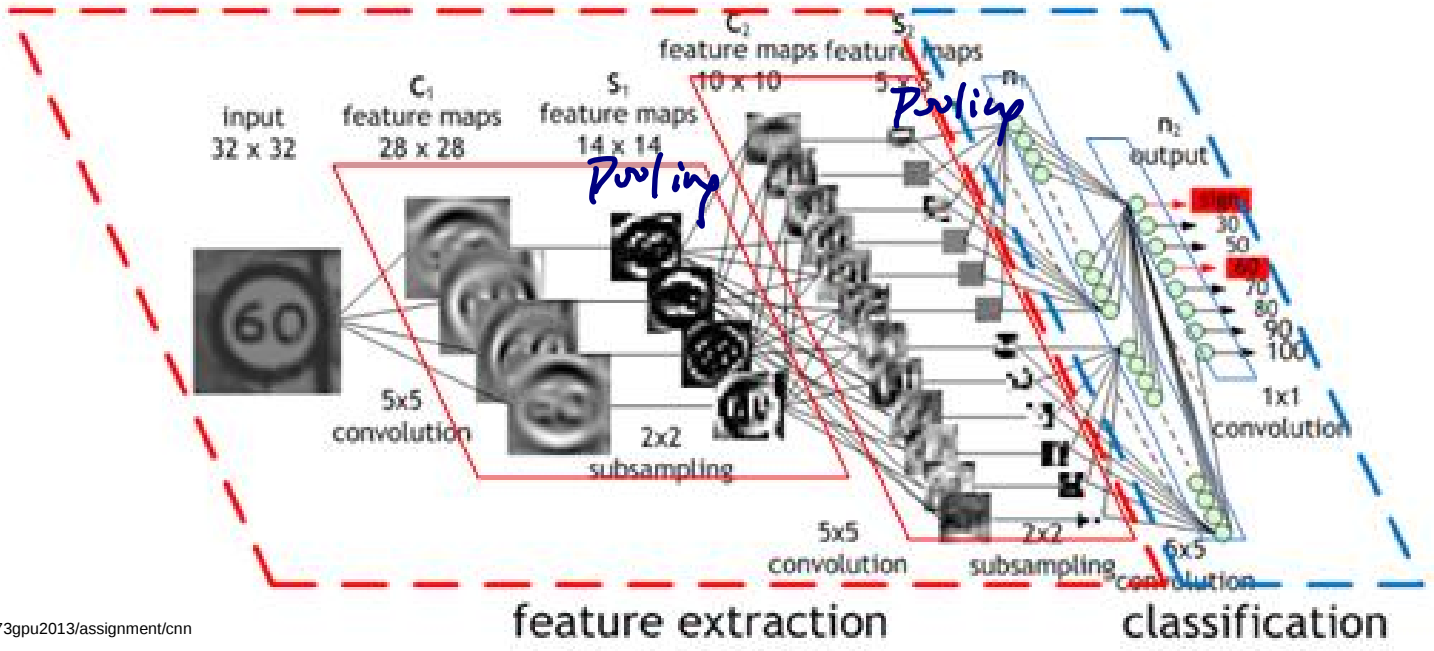
max()



sum()



CNN – example architecture (various layers)

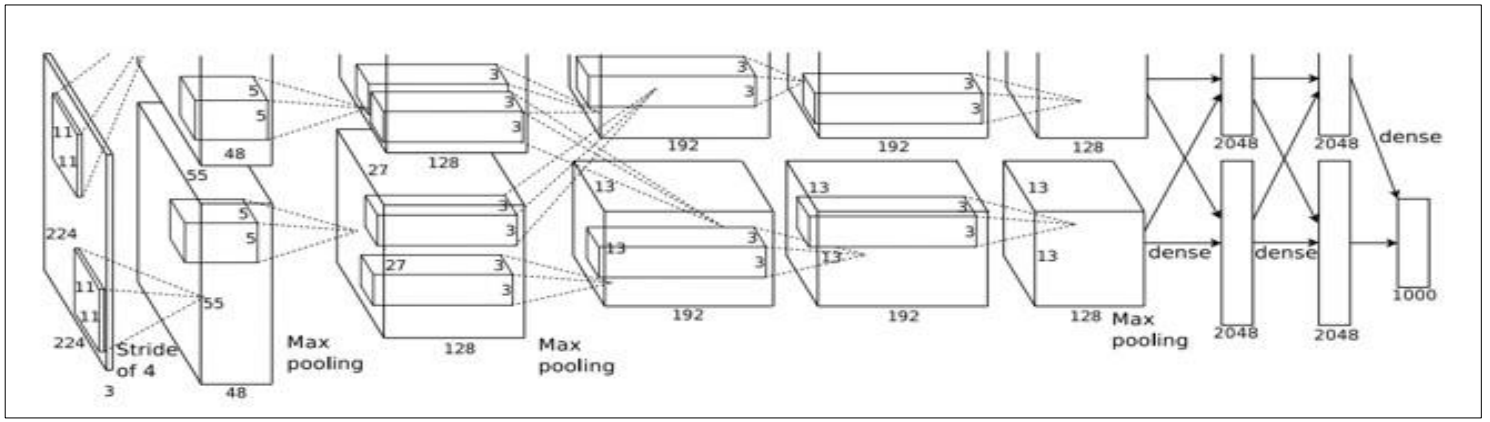


Example CNN design:
https://sites.google.com/site/5kk73gpu2013/assignment/cnn

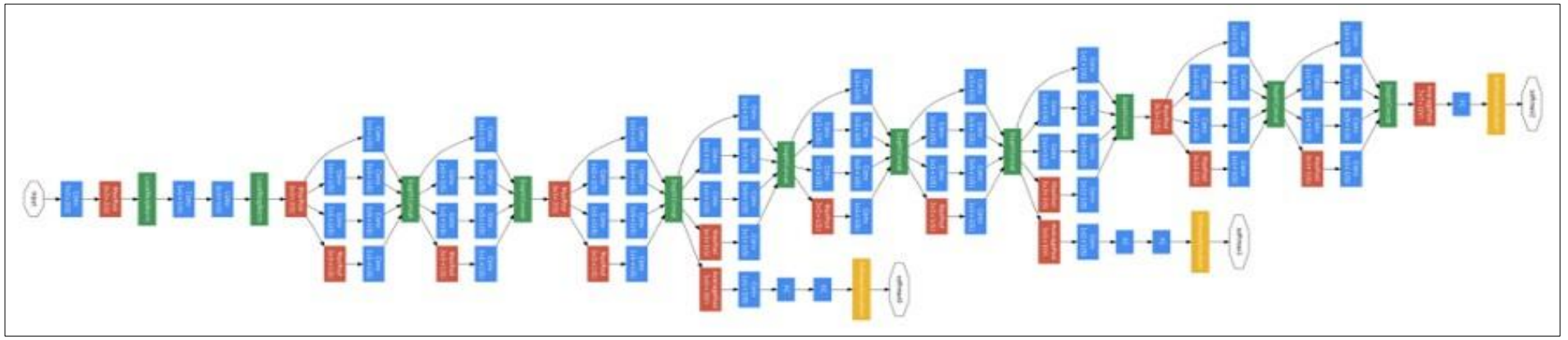
- Number of layers, type of layer, number of nodes/maps per layer – all down to the (human) designer
  - many variants have emerged
- CNN training – efficient **backpropogation**

  as per traditional neural network approaches (with **Dropout**)

Seminal
Deep Learning Architectures



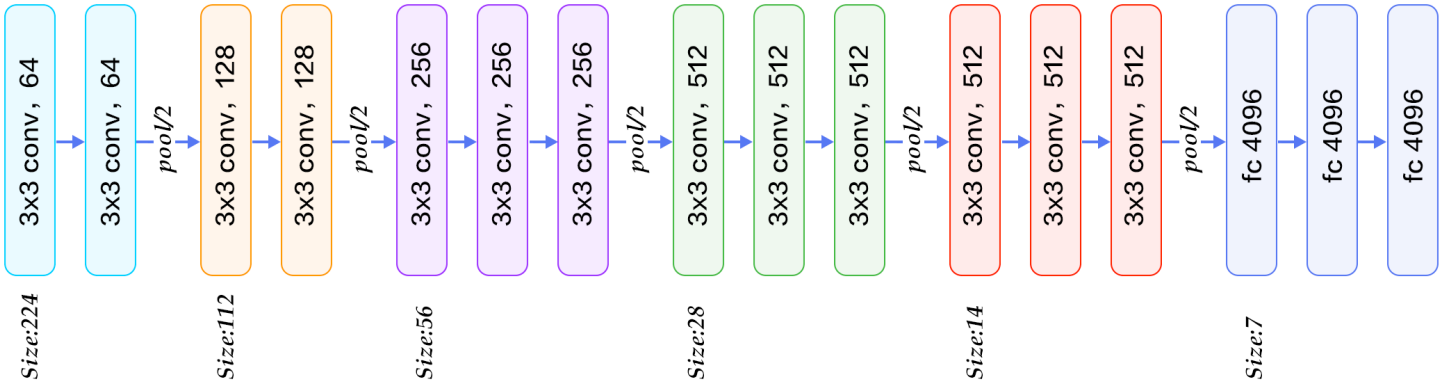
ImageNet Classification with Deep Convolutional Neural Networks
A Krizhevsky I Sutskever, G Hinton (2012) **- "AlexNet"**



Going Deeper with Convolutions, C Szegedy et al (2014) - "**GoogLeNet**"

→ Contemporary architectures ….

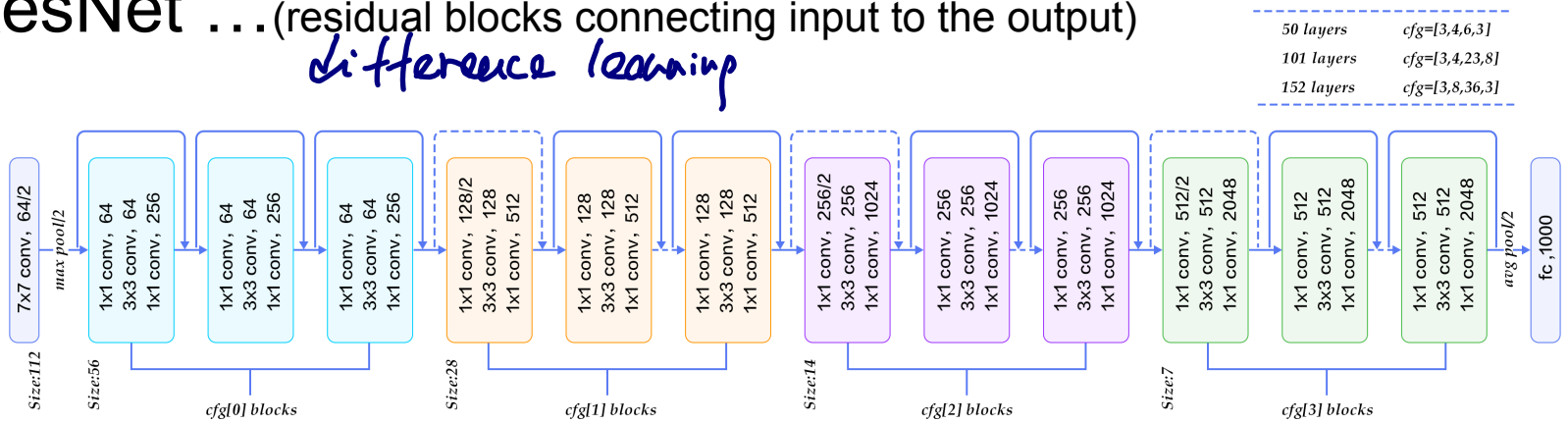
- VGG -16 … (deeper, smaller 3x3 convolutions throughout)



Simonyan Zisserman, 2014: http://www.robots.ox.ac.uk/~vgg/research/very_deep/

- ResNet …(residual blocks connecting input to the output)

difference learning



K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. CVPR 2016.

Figures: http://book.paddlepaddle.org/03.image_classification/

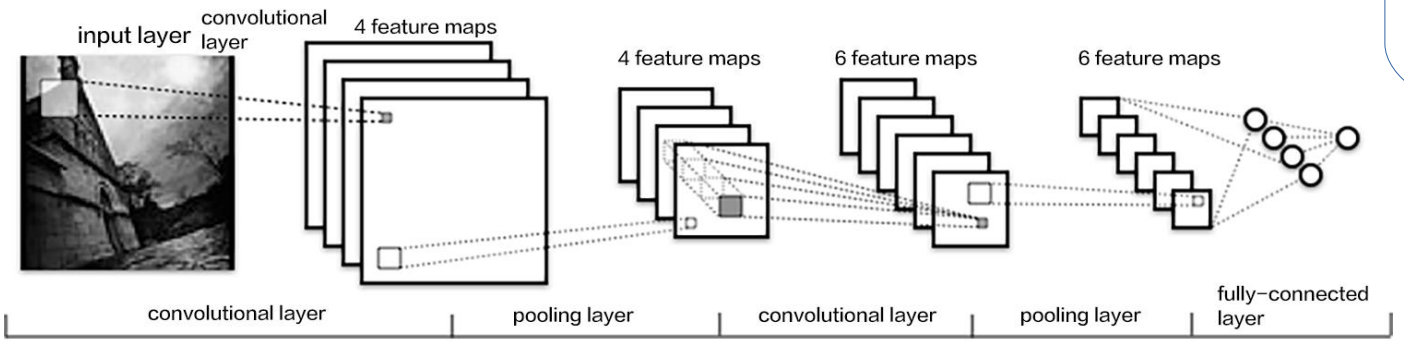
…. and more and more ...

Varying Deep CNN Architectures (and applications) all based on ...

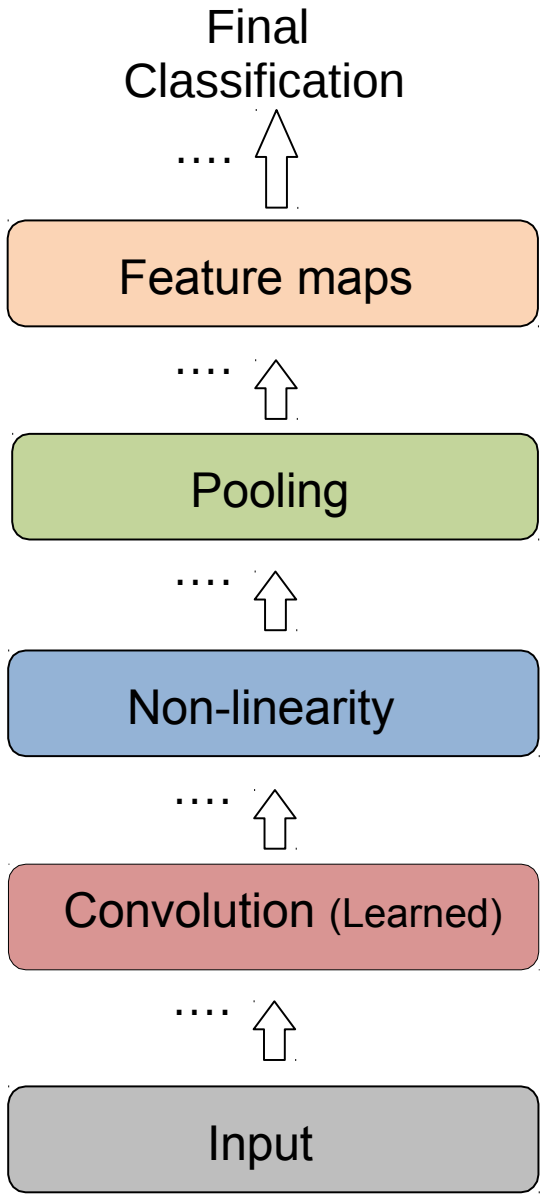
- **Feed-forward network:**
  - Convolve input (feature extraction)
  - Non-linearity (rectified linear)
  - Pooling (local max)

  ... all trained by **backpropogation**

Repeated multiple times over network architecture



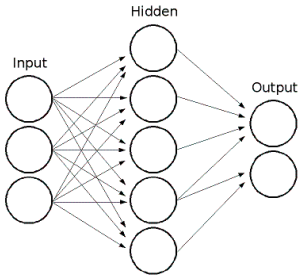
Example: LeNet—5 (http://book.paddlepaddle.org/03.image_classification/)



Train via **Backpropagation**

*Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October **1986**). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.*

- **Neural Network Training: weight modifications** are made in the **"backwards" direction**: *from the output layer, through each hidden layer down to the first hidden layer*, hence **"Backpropagation"**



- **Key Algorithmic Steps**
  - **Initialize** weights (to small random values*) in the network
  - **Propagate the inputs forward**
    - (by applying activation function) at each node
  - **Backpropagate the error backwards**
    - (by updating weights and biases)
  - **Terminating** condition
    - when validation error is very small or enough iterations

Backpropogation details beyond scope/time (see accompanying reading)

So overall deep network layers provide …

feature extraction
data de-noising
dimensionality reduction
feature pooling
spatial invariance
non-linear input → output mapping

… specifically optimized towards a given problem*

(* that is represented by a given set of defined examples)

… which really covers *most* desirable aspects for *most* computer vision problems we encounter.

[if you think about it]

Key question – **is it *really* that easy ?**

(images in → results out ?!?)

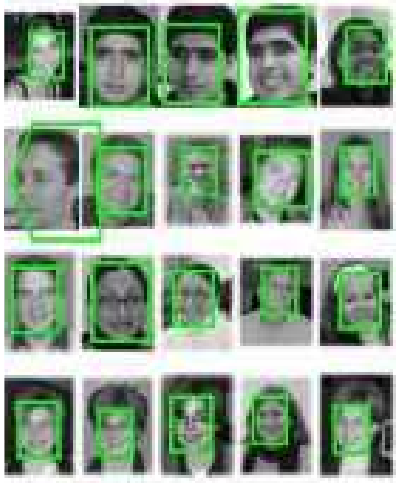
Remember, it's **all about ..**

Learning *from* the Data

- **Training data:** used to train the system
  - i.e. build the rules / learnt target function
  - split into training (back-propogated) and validation (when to stop back-propogating)
  - specific examples (used to learn)

....

- **Test data:** used to test performance of the system
  - *unseen* by the system during training
  - specific examples (used to evaluate)



....

e.g. face gender classification

Simple ? - Well almost .....

provided we avoid
the pitfalls on the way

We must avoid over-fitting …..

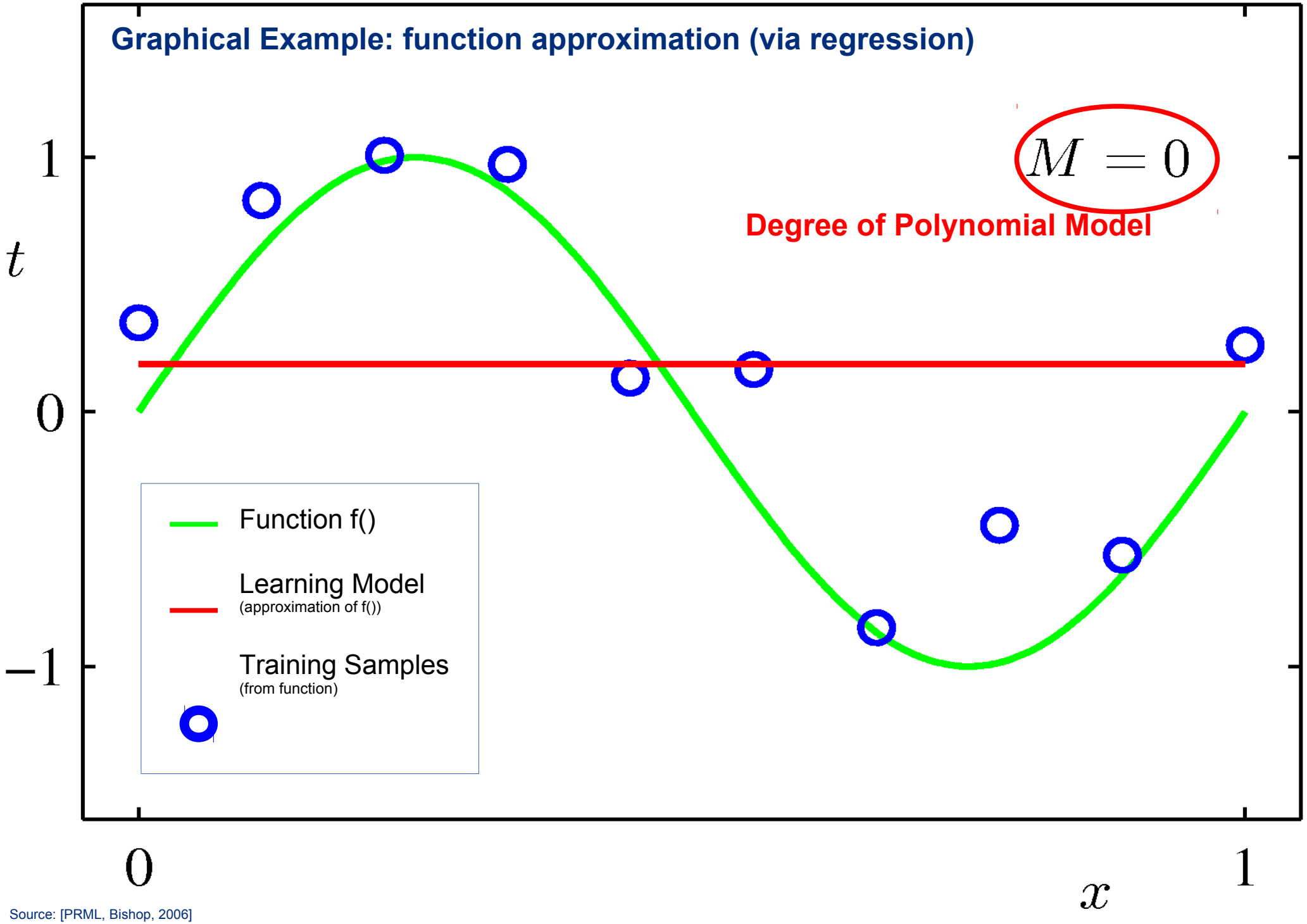
(i.e. over-learning)

Principle of Occam's Razor

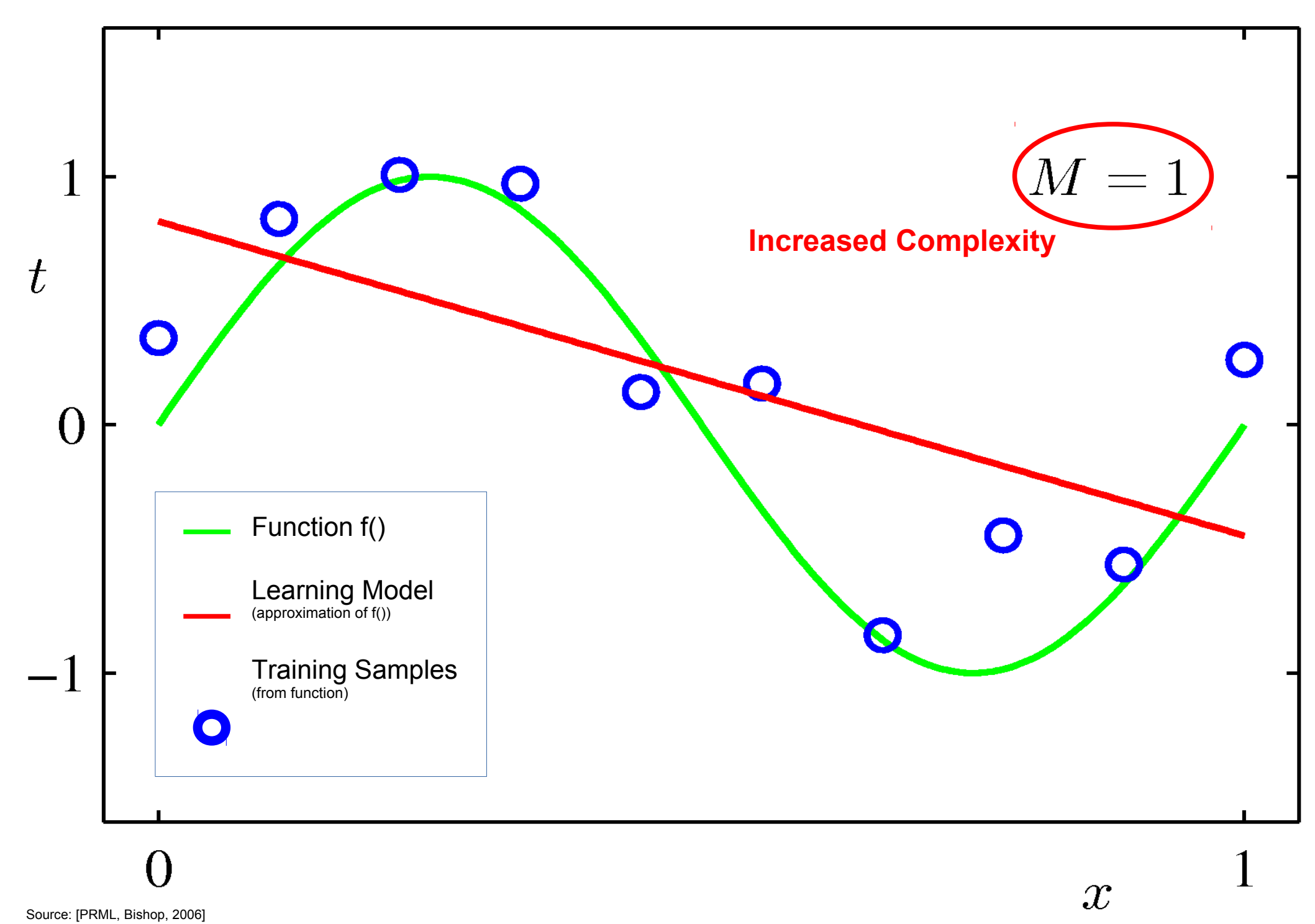
- Occam's Razor
  - "entia non sunt multiplicanda praeter necessitatem" **(latin!)**
  - "entities should not be multiplied beyond necessity" **(english)**
  - **"All things being equal, the simplest solution tends to be the best one"**
- **For Machine Learning :** prefer the simplest {model | hypothesis | …. tree | projection | network } that fits the data

14th-century English logician William of Ockham

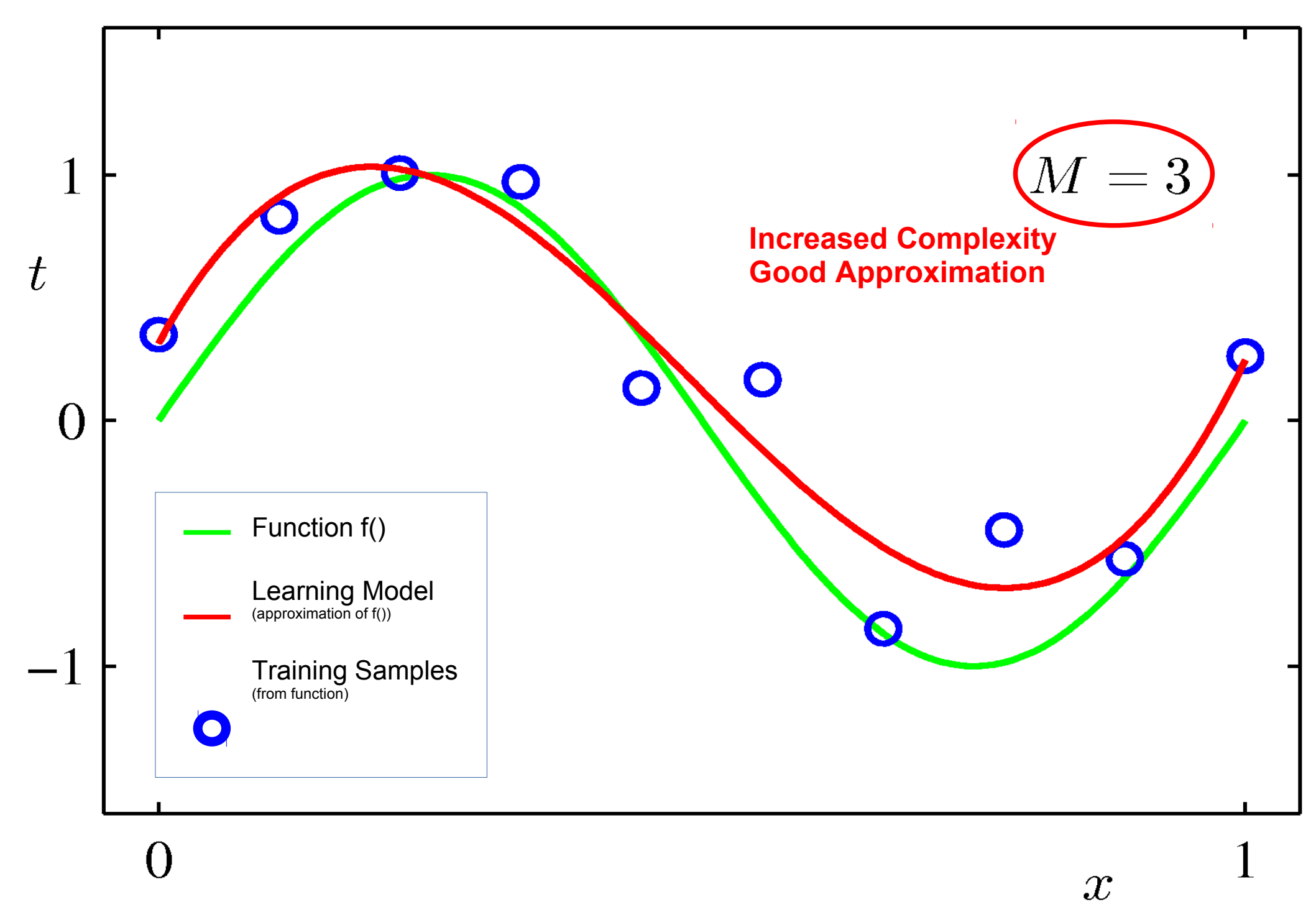
Graphical Example: function approximation (via regression)



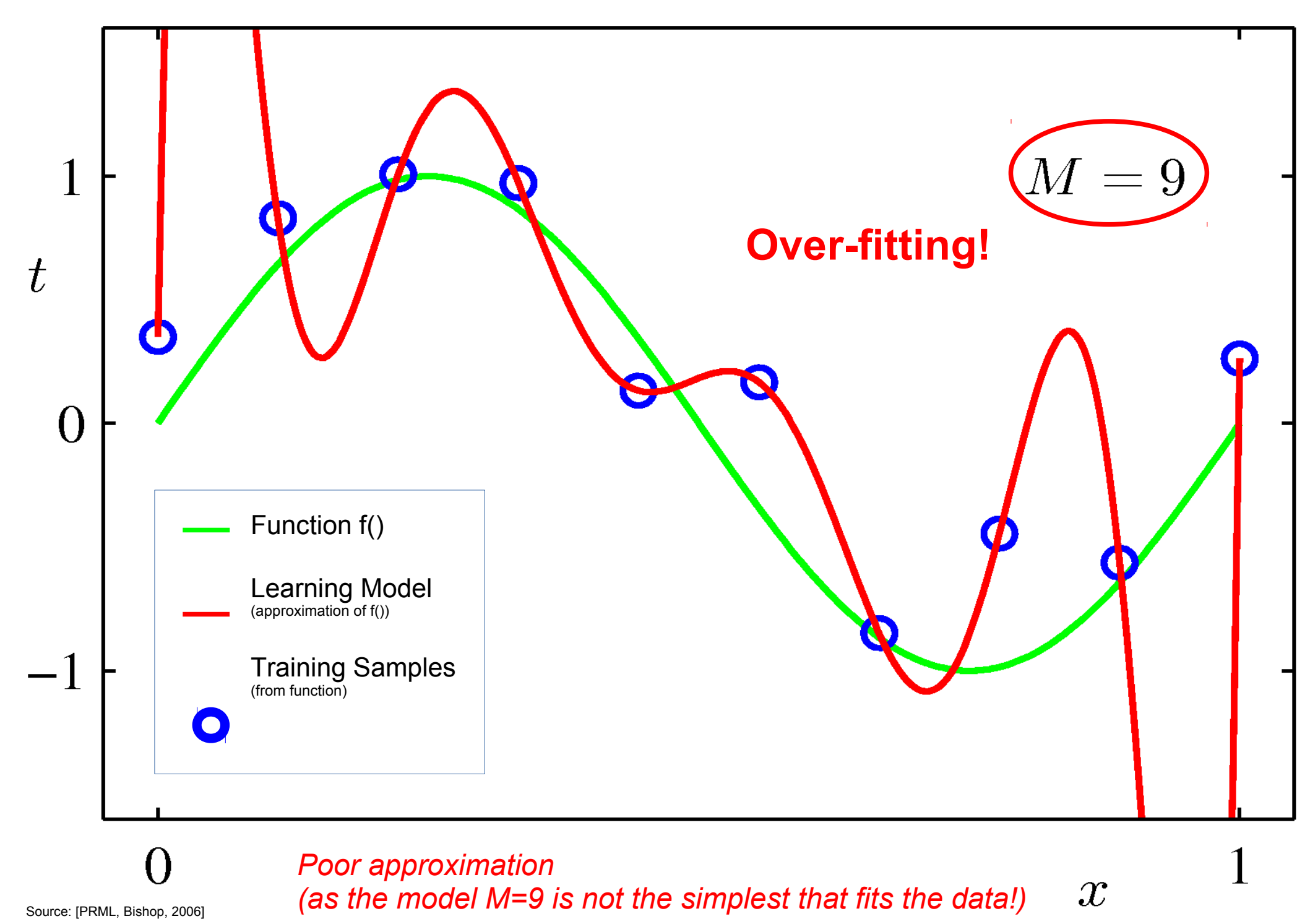
Source: [PRML, Bishop, 2006]



Source: [PRML, Bishop, 2006]



Source: [PRML, Bishop, 2006]

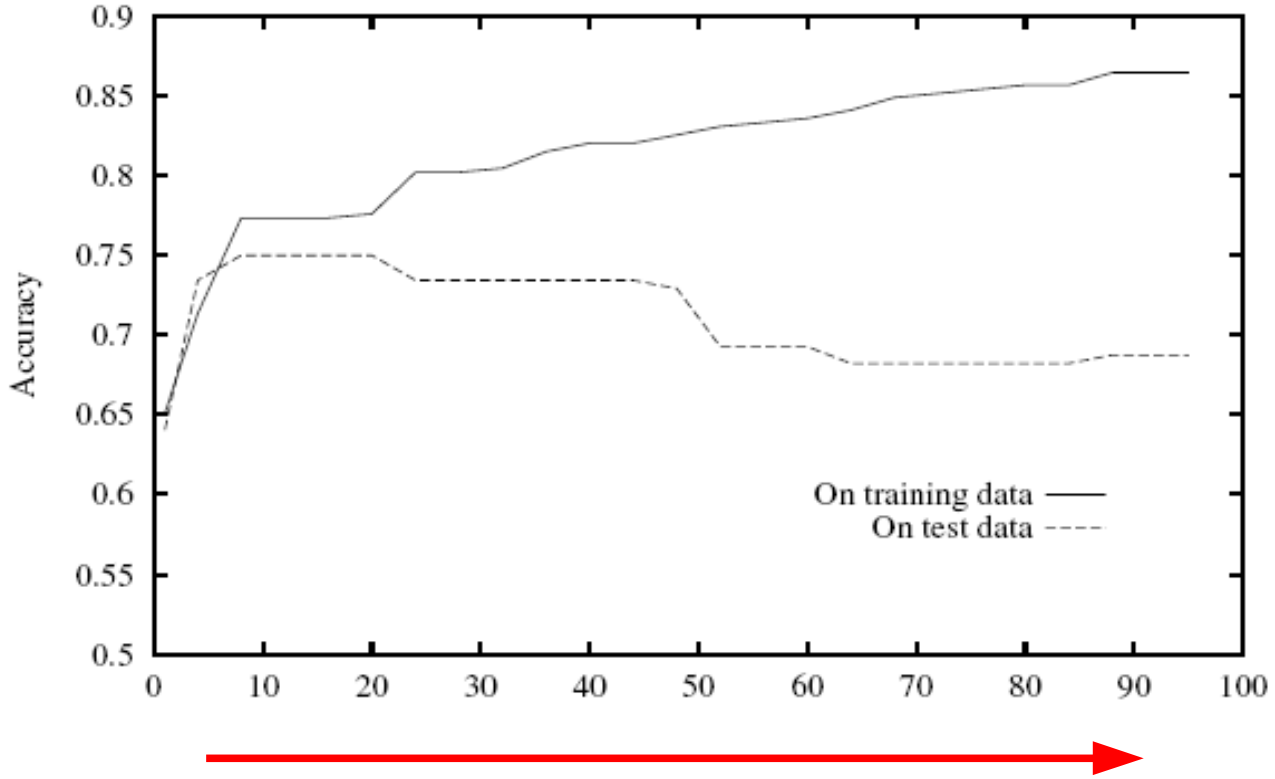


Poor approximation
(as the model M=9 is not the simplest that fits the data!)

Source: [PRML, Bishop, 2006]

How to spot over-fitting ...

- **Performance** on the **training** data **improves**
- **Performance** on the unseen **test** data **decreases**



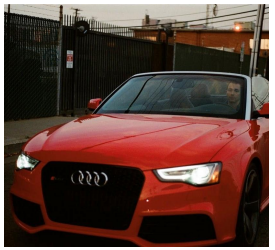
Increasing model complexity or training iterations

Loss *what* ?

Loss Functions: how good is our net ?

Suppose: 3 training examples, 3 classes.
With some parameters, W the scores

$f(x, W) = Wx$ are:


cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Softmax(): mapping output activation scores to probabilities

Want to interpret raw CNN output scores as **probabilities:**

scores → **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

use softmax() function

Probabilities must be >= 0

Probabilities must sum to 1

		exp		normalize	
cat	**3.2**	→	**24.5**	→	**0.13**
car	5.1	→	164.0	→	0.87
frog	-1.7	→	0.18	→	0.00

probabilities

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Softmax(): mapping output activation scores to probabilities

Want to interpret raw CNN output scores as **probabilities:**

scores → **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

use softmax() function

Probabilities must be >= 0

Probabilities must sum to 1

cat	**3.2**	exp →	**24.5**	normalize →	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

compare ← **1.00** / 0.00 / 0.00

probabilities

Correct (probabilities)

Cross-entropy loss (diff. in correct probabilities)

$$L_i = -\log P(Y = y_i|X = x_i)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Regularization: prevent the net over-fitting

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data (i.e. overfitting)

→ Occam's Razor

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Regularization: prevent the net over-fitting

λ = regularization strength (hyperparameter)

avoid too large/small weights

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data (i.e. overfitting)

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

e.g. **L2 Regularization**: how and why ?

Expresses a preference for weight equality:

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

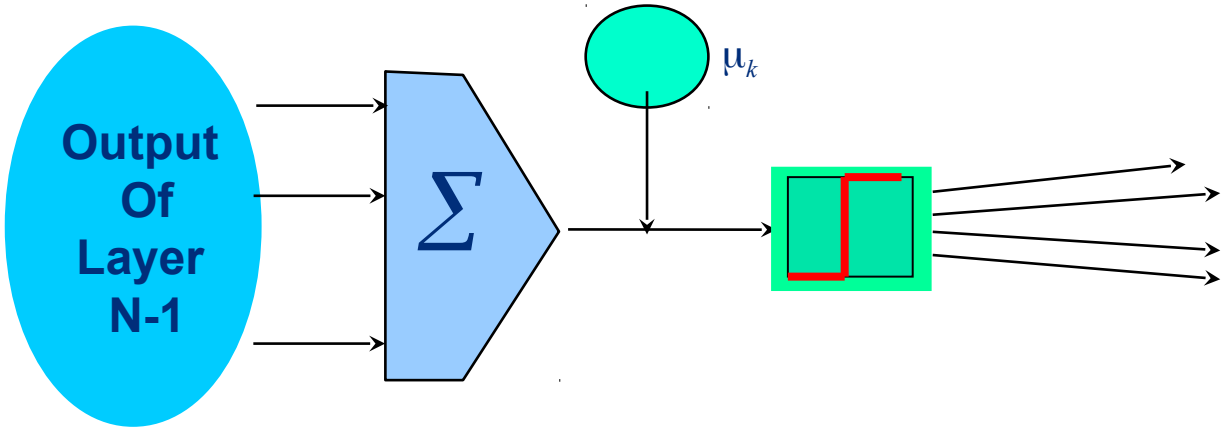
L2 regularization likes to "spread out" the weights

Where several W may have same classification result:

$$w_1^T x = w_2^T x = 1$$

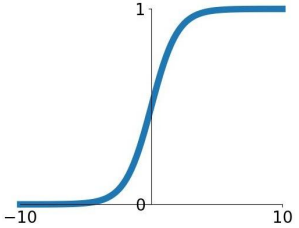
[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

… **in every node:** activation functions



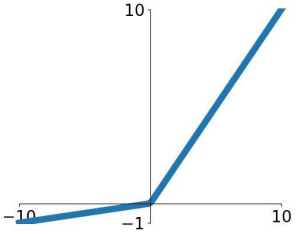
Sigmoid

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



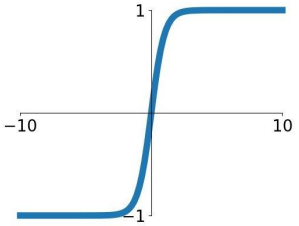
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

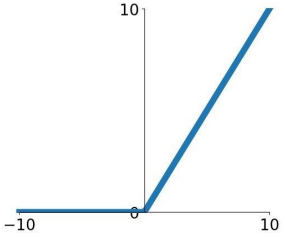


Maxout

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

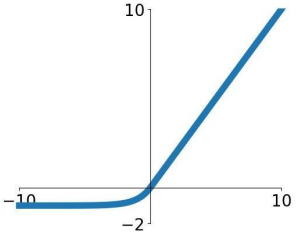
ReLU

$$\max(0, x)$$



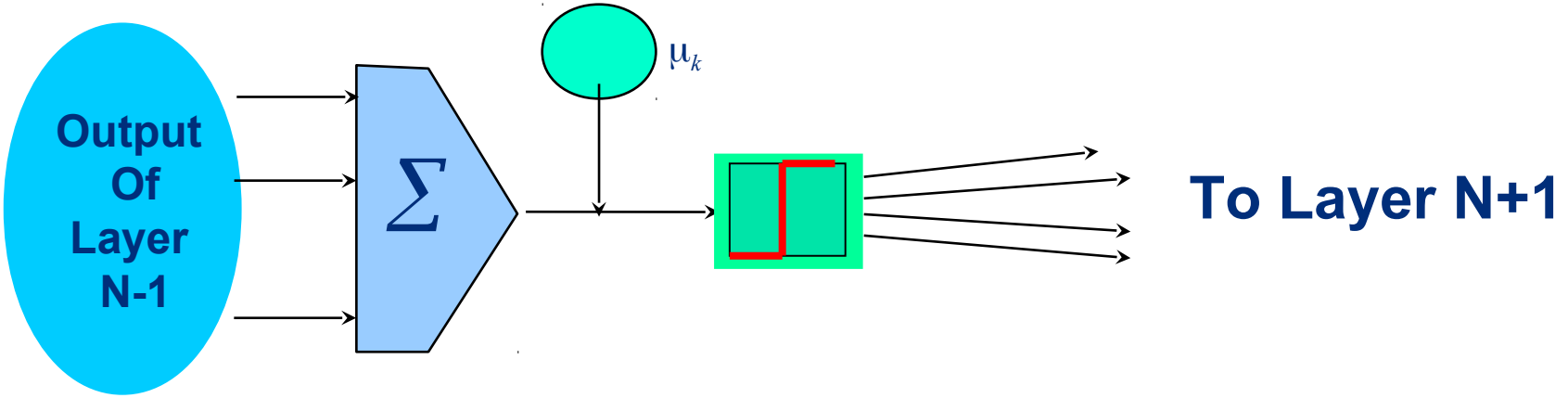
ELU

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



[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

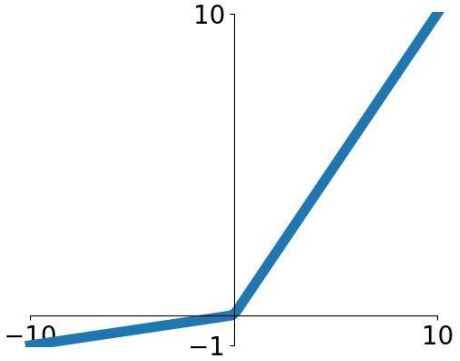
… in every node: activation functions



- 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

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Activation Functions – **ReLU variations**



Leaky ReLU

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

[Mass et al., 2013]
[He et al., 2015]

- Does **not saturate**
- Computationally **efficient**
- **Converges** much **faster** than sigmoid/tanh in practice! (e.g. 6x)
- **will not "die"** (like ReLU → 0 output)

also **Parametric Rectifier (PReLU)**

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

backprop parameter

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

… but how **many network architectures** are we training?

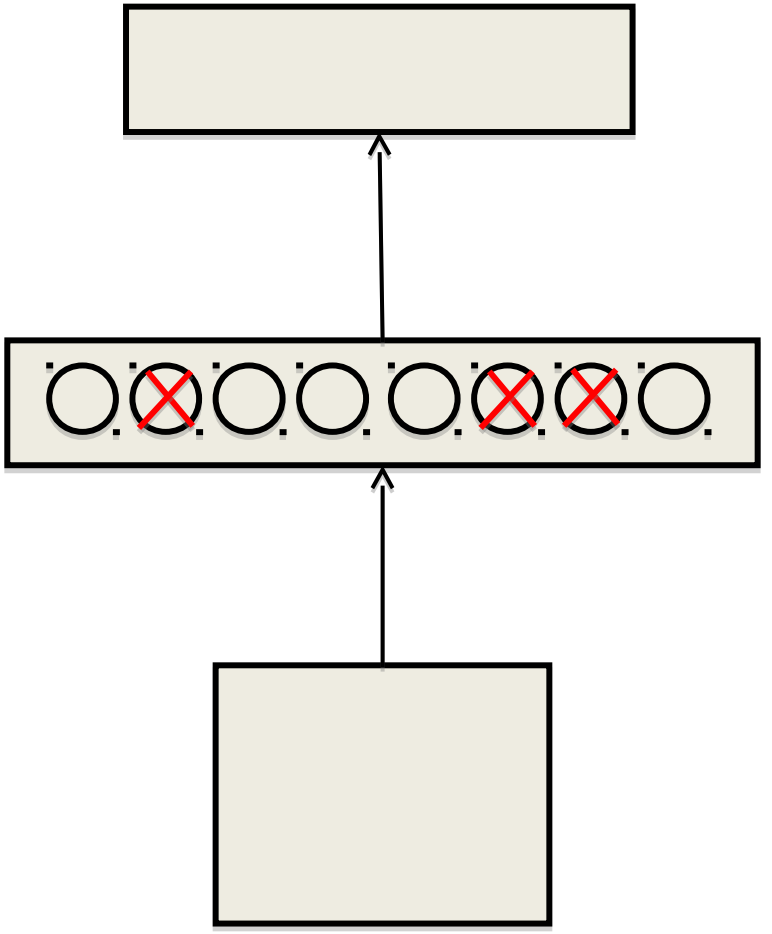
(are all the node weights updated in each backpropogation cycle?)

Dropout: efficient and robust training for large neural nets (Hinton et al., 2012 http://arxiv.org/abs/1207.0580)

- Consider a neural net with H hidden units
- Each time we present a **training** example within backpropogation, we **randomly omit each hidden unit** in *{all | some}* layers with probability 0.5.

  → we are randomly sampling from 2^H different architectures

- At **test time** – use all hidden units but **halve all the outgoing weights**
  - computes an approximate mean of the predictions of all 2^H models.



[This slide – adapted from: G. Hinton]

Get the **setup correct ...**

Input **pre-processing** – image data

e.g. consider CIFAR-10

dataset with [32,32,3] images

Zero-centre all the image pixel data inputs so backpropogation gradients are both +ve and -ve

Otherwise – major issue!

How -

- Subtract the mean image (e.g. AlexNet)
  (CIFAR - mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)

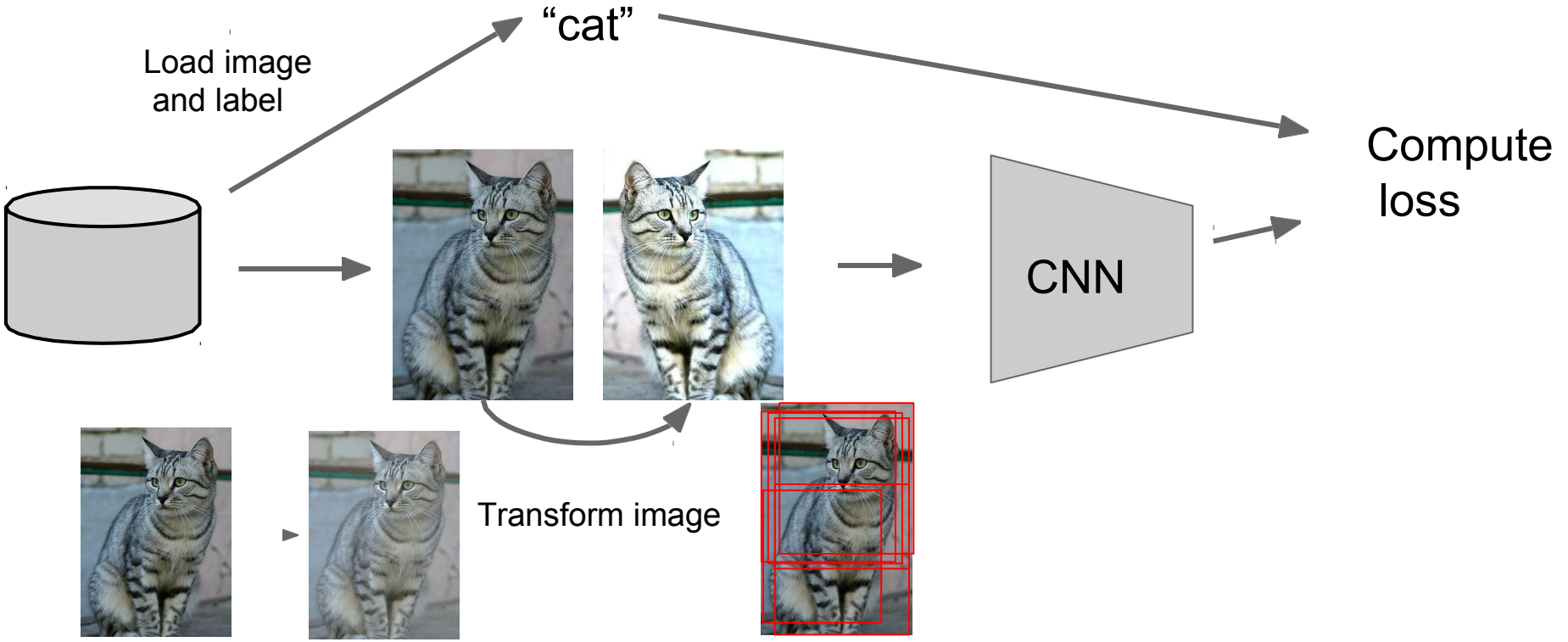
  (CIFAR - mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening

Remember t**o zero-centre inputs at run-time (test) also!**

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Data **Augmentation** – *generate more data!*



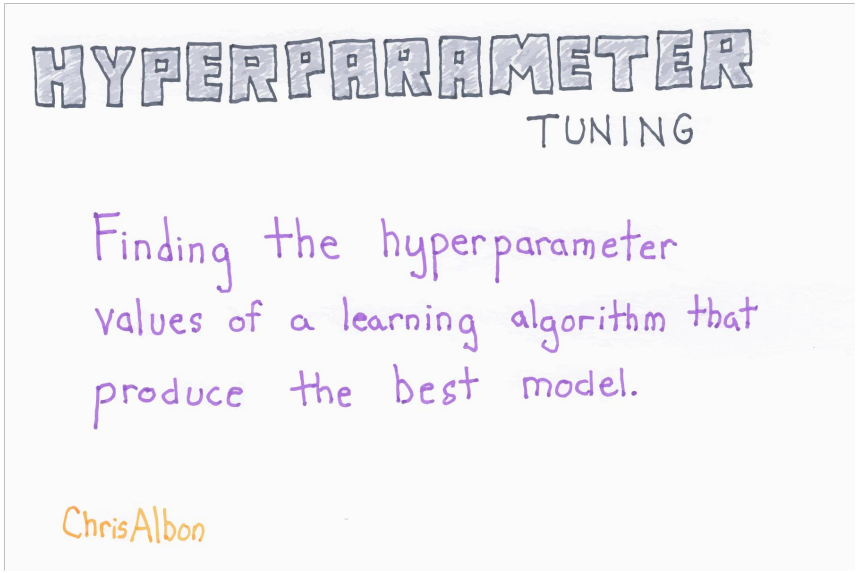
- Use random mix/combinations of : flipping, translation, rotation, stretching, shearing, illumination changes (log/exp/gamma transform), lens distortions, …(go crazy!)

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

And **finally ...**

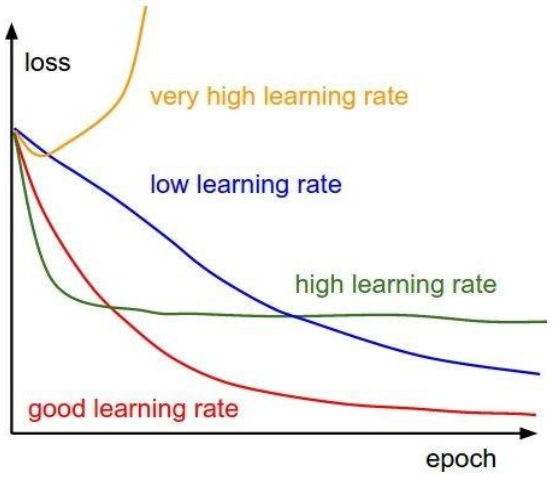
Hyper-parameters: within deep learning

- choices about the algorithm that we set rather than learn
  - e.g.
    - **drop-out** rate
    - **weight initialization**
    - **Backprop parameters ...**
    - …
    - **cross-validation** folds (?)



https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_grid_search/

- Highly problem-dependent.
  - must explore the hyper-parameter space (via glorified **"trial and error"**) to find optimal set



Key **Enablers**

(i.e. what changed to make all this happens)

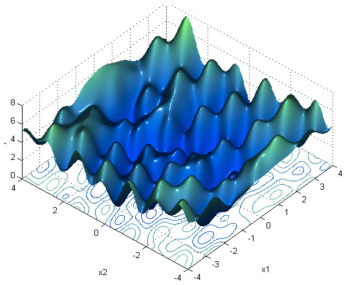
- As of now (~2012 → 2016+) we now have three key things that made deep learning possible:
  - **data** *(lots of it available)*

~14 million labeled images, 20k classes

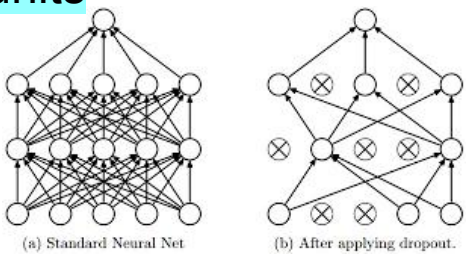
  - low-cost, **high-performance GPU hardware**
    (to train larger networks than before)
  - **key algorithmic insights** to backpropogation **training**
    (to train networks more efficiently + guard against local minima and overfitting)

Neural Networks – *a deep reprize*

- Recent work shows **local minima are less of a problem than thought**
  (Pascanu et al., 2014, Dauphin et. al., 2014, Choromanska et al., 2015)
  - local minima dominate low dimensions but saddle points (ridges) dominate high dimensions
  - most local minima are close to global minima in high dimensions
    - … *and* **deep neural networks use a very high dimensional weight space**



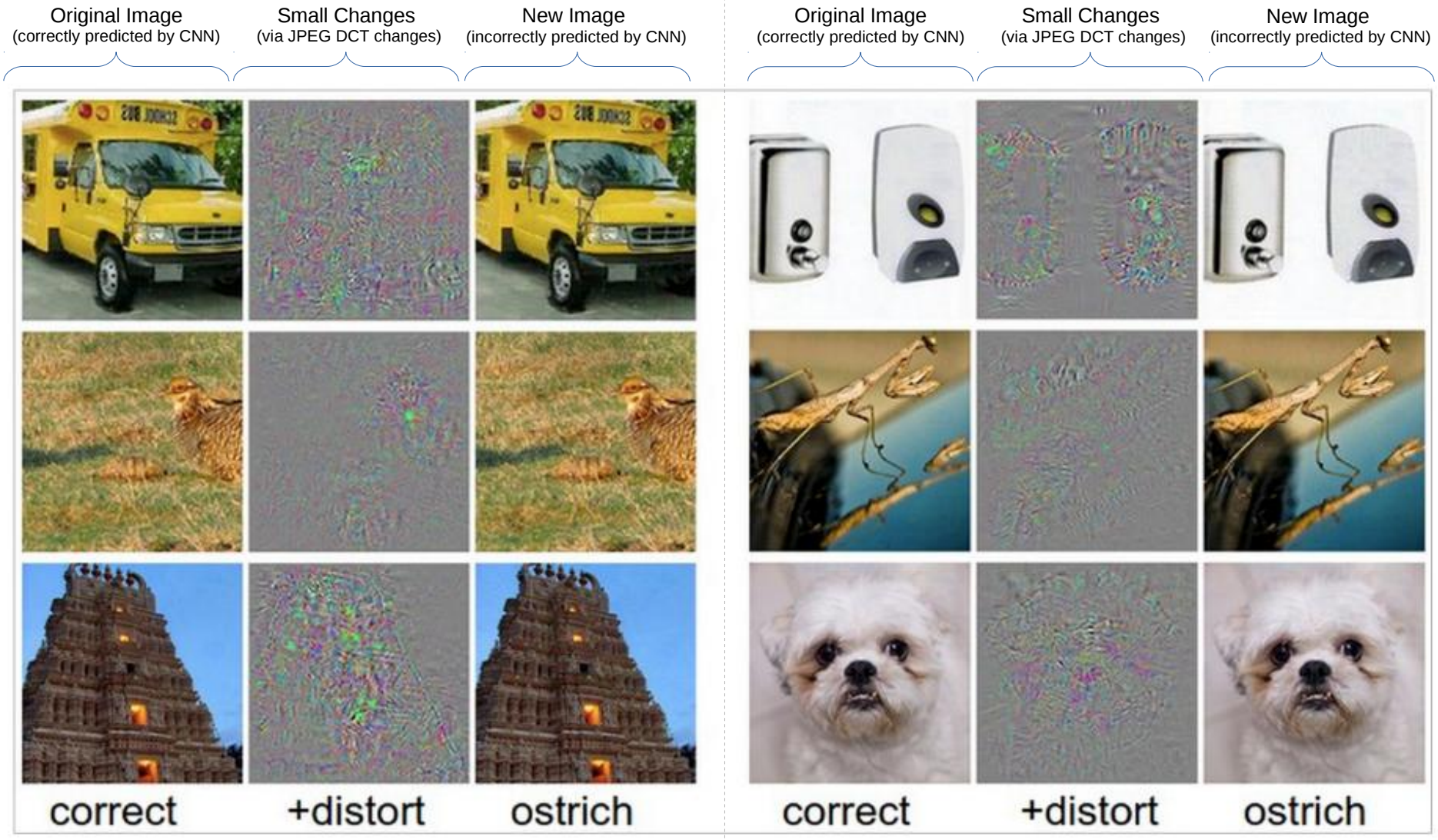
- **Advances in training:** use of **drop-out** to regularize the weights in the globally connected layers (which contain most of the parameters)
  - **dropout:** half of the hidden units in a layer are randomly removed for each training example.
  - **effect:** stops hidden units from relying too much on other hidden units (hence **reduces likelihood of over-fitting**)



Beyond Classification ...

(applications in computer vision beyond classification include : Detection Segmentation, Regression, Pose estimation, Image Synthesis ...)

Are they always right ? - fooling CNNs



Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

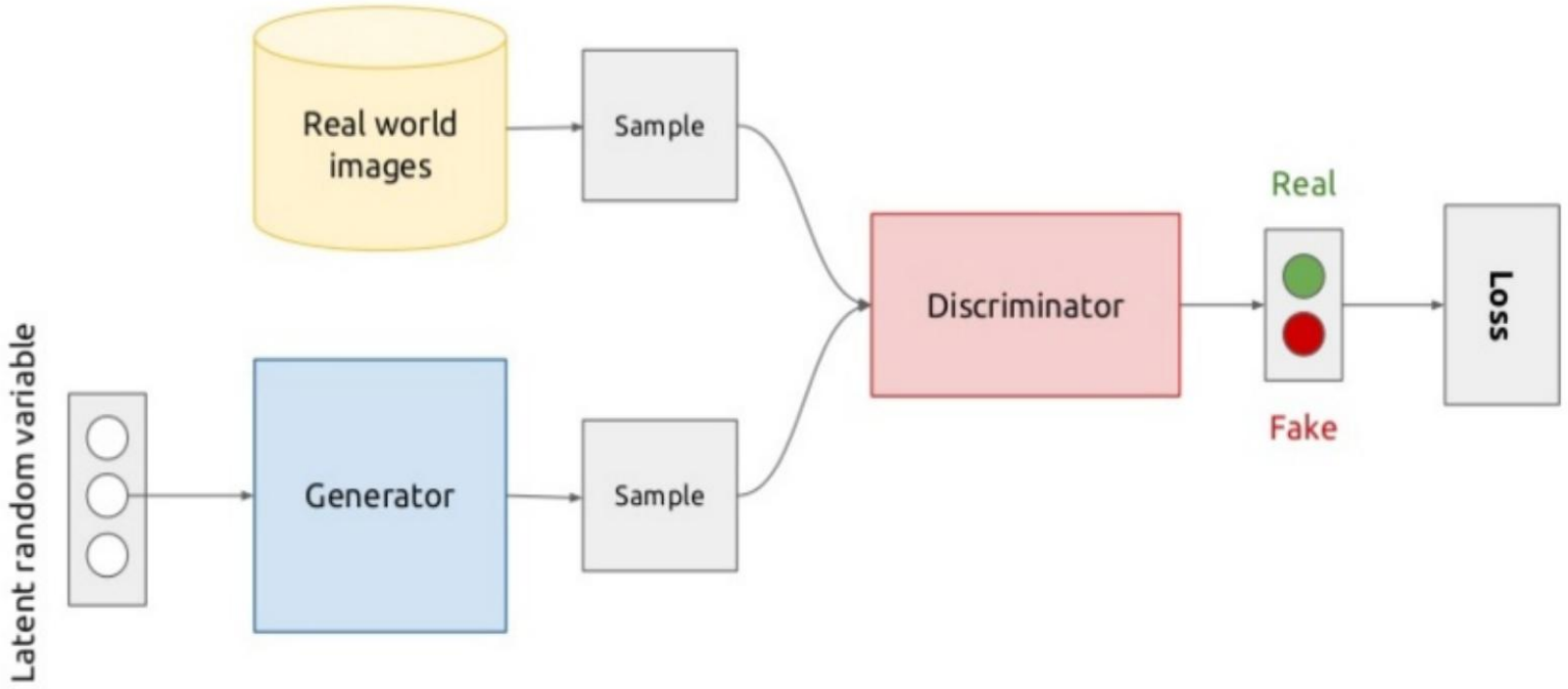
Reference: Intriguing properties of neural networks [Szegedy ICLR 2014]

Press article: http://www.i-programmer.info/news/105-artificial-intelligence/7352-the-flaw-lurking-in-every-deep-neural-net.html

… in today's terminology we call these **adversarial examples**

… which brings us on to a whole new sub-topic of deep learning

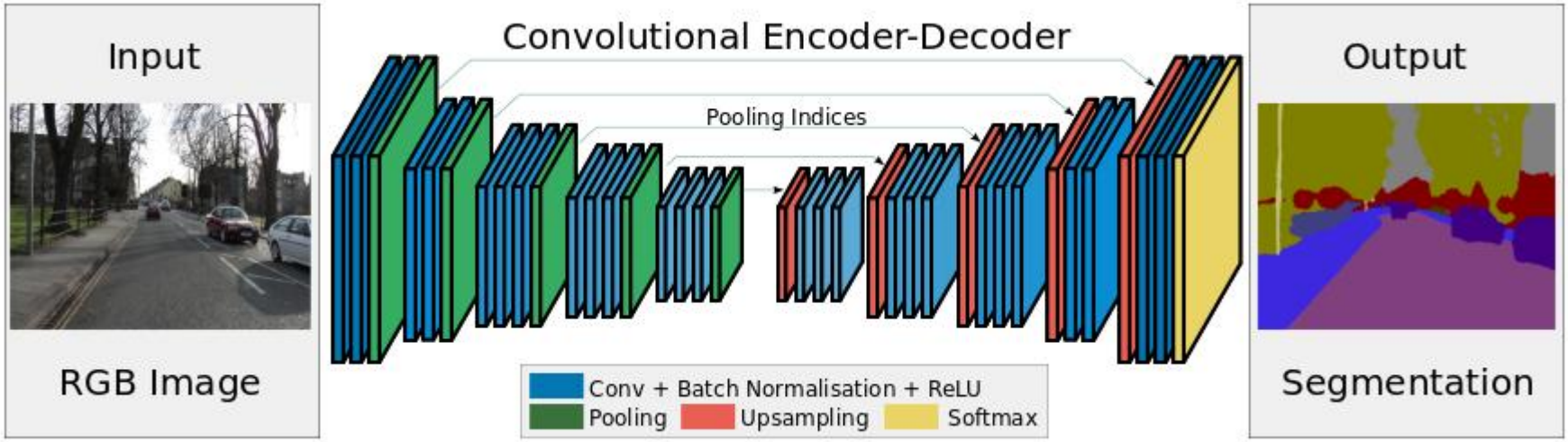
Generative Adversarial Networks (GAN)



[Goodfellow et al., 2014
https://arxiv.org/abs/1406.2661]

- train two models:
  - one to generate some sort of fake examples from random noise (or some conditioned distribution)
  - one to discern fake model examples from real examples
- **Many applications** in improving deep network performance

Example: semantic pixel labelling via SegNet ….



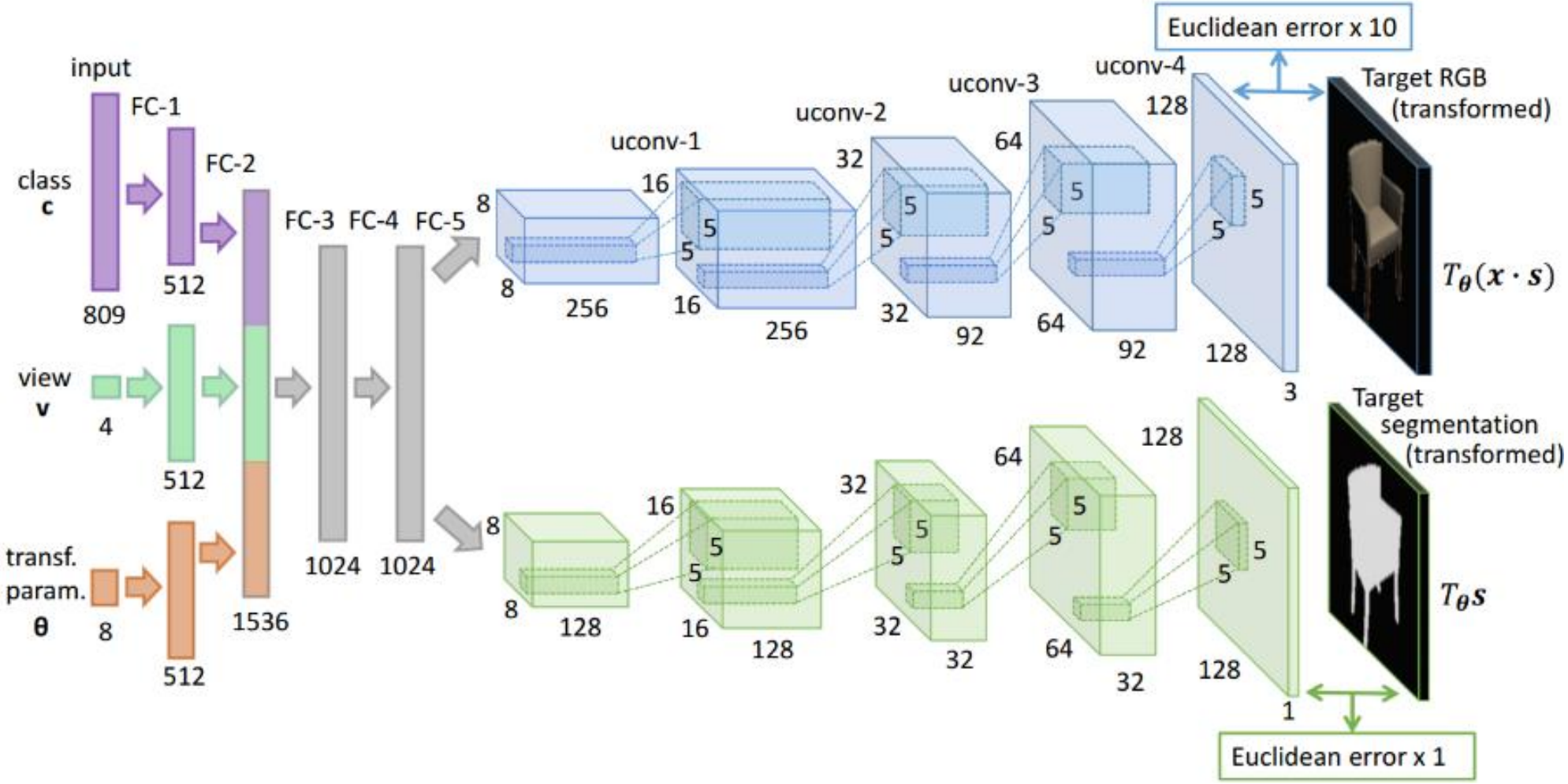
- Application to Semantic Image Segmentation via CNN
  - use of per complex network to perform per-pixel classification by object type (i.e. semantic pixel labelling)
  - Encoder ↔ Decoder architecture (but same concepts of layer types)

http://www.youtube.com/embed/e9bHTlYFwhg?rel=0

Vijay Badrinarayanan, Alex Kendall and Roberto Cipolla "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." arXiv preprint arXiv:1511.00561, 2015. http://arxiv.org/abs/1511.00561

http://mi.eng.cam.ac.uk/projects/segnet/

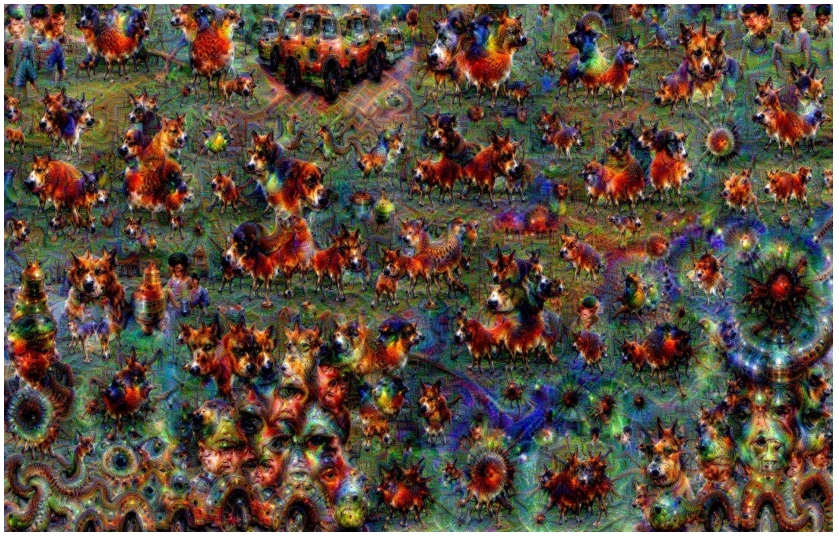
CNN for Image Generation (synthesis)

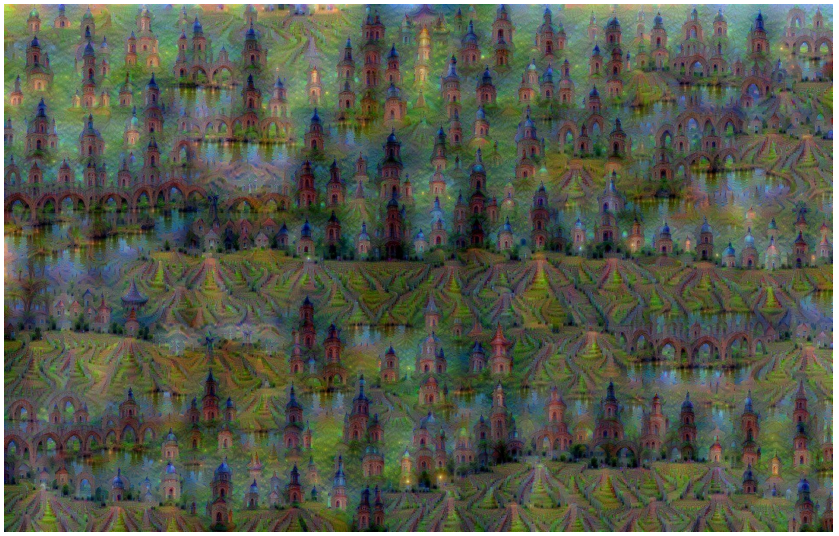


Video: http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15/Generate_Chairs_mov_morphing.avi

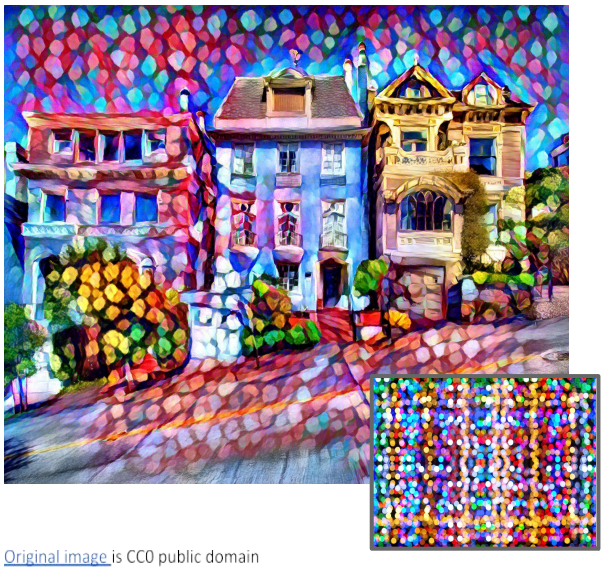
Learning to Generate Chairs with Convolutional Neural Networks [Dosovitskiy et al. CVPR 2015]

Style Transfer ….





Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Images ↔ text ….

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

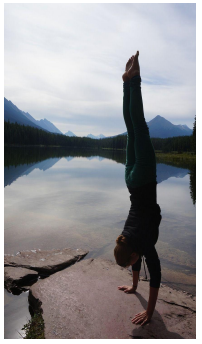
Somewhat related

A woman is holding a cat in her hand

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor



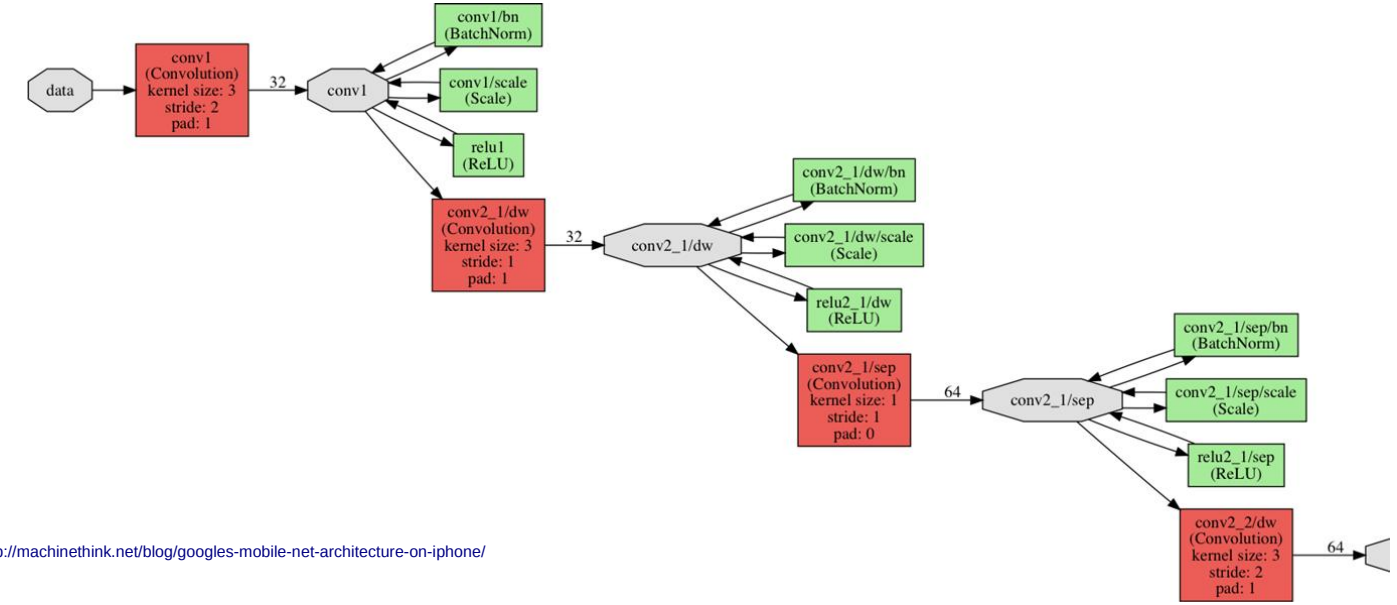
A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

[This slide – adapted from: Fei-Fei Li & Justin Johnson & Serena Yeung]

Efficient Networks – *MobileNets* (et al.)

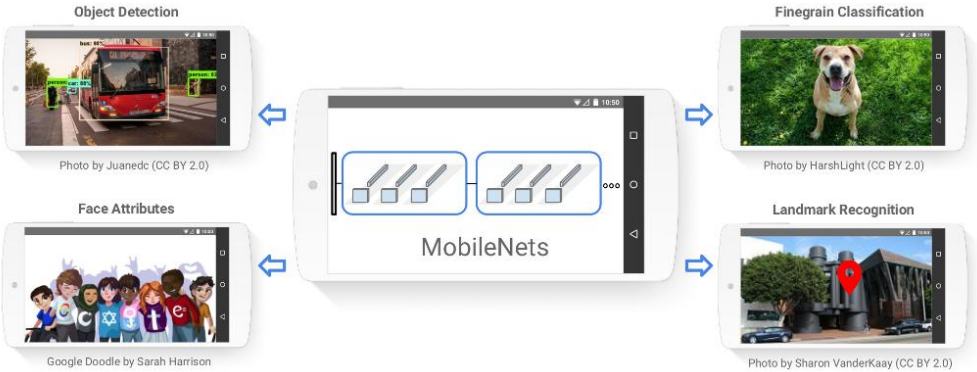


http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

- **MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications**

  Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam (Google)

  https://arxiv.org/abs/1704.04861



- **MobileNetV2: Inverted Residuals and Linear Bottlenecks**

  Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen

  CVPR 2018

  https://openaccess.thecvf.com/content_cvpr_2018/CameraReady/3427.pdf