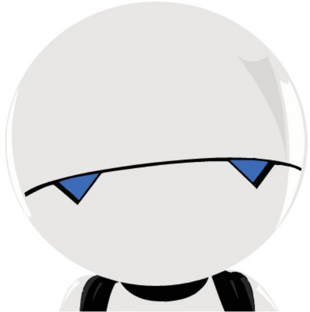
Deep Learning: CNN

Prof. Marios Savvides

Some slides borrowed from Princeton Vision Group

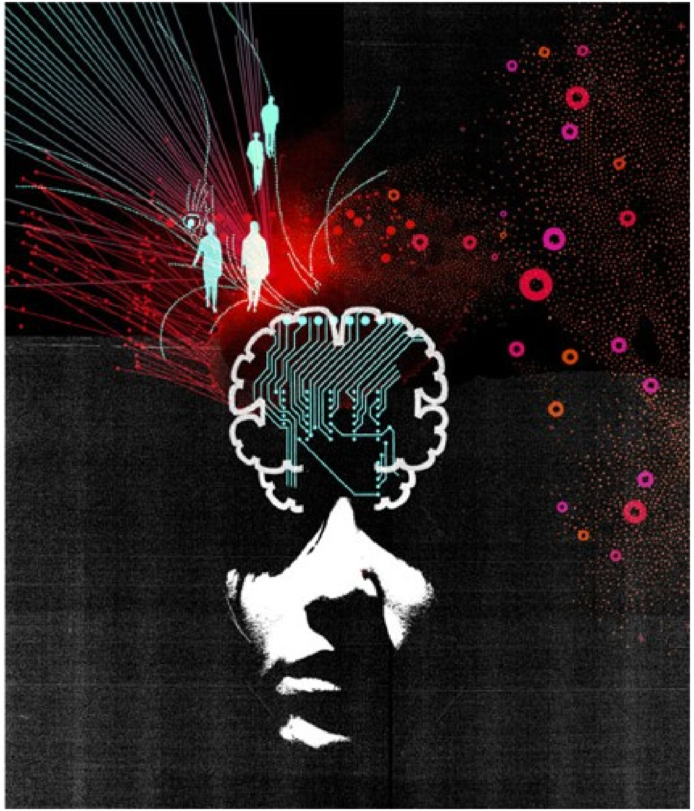


MIT Technology Review

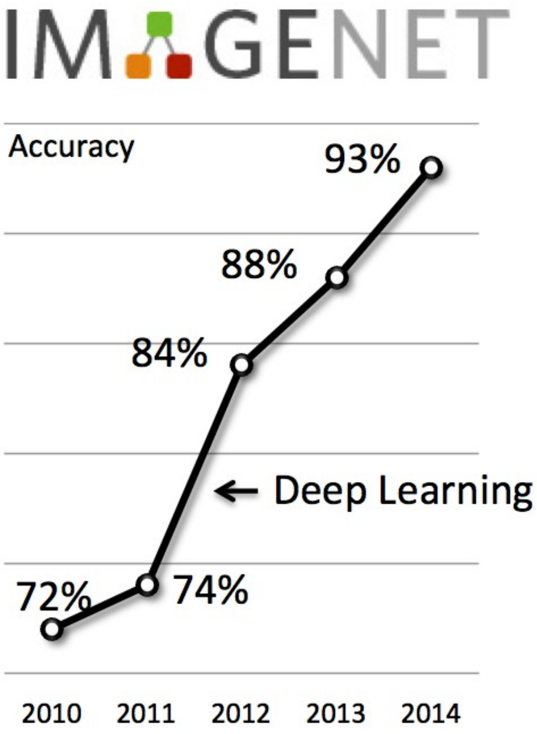
10 BREAKTHROUGH TECHNOLOGIES 2013

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.



Deep Learning



Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

Microsoft 95.06%, Feb 06, 2015

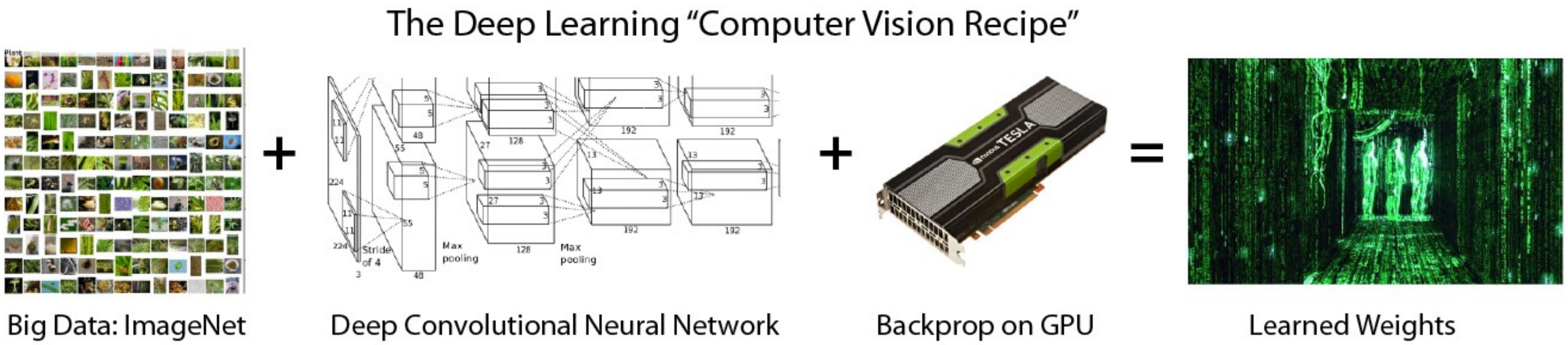
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariant Shift

Google 95.18%, Feb 11, 2015

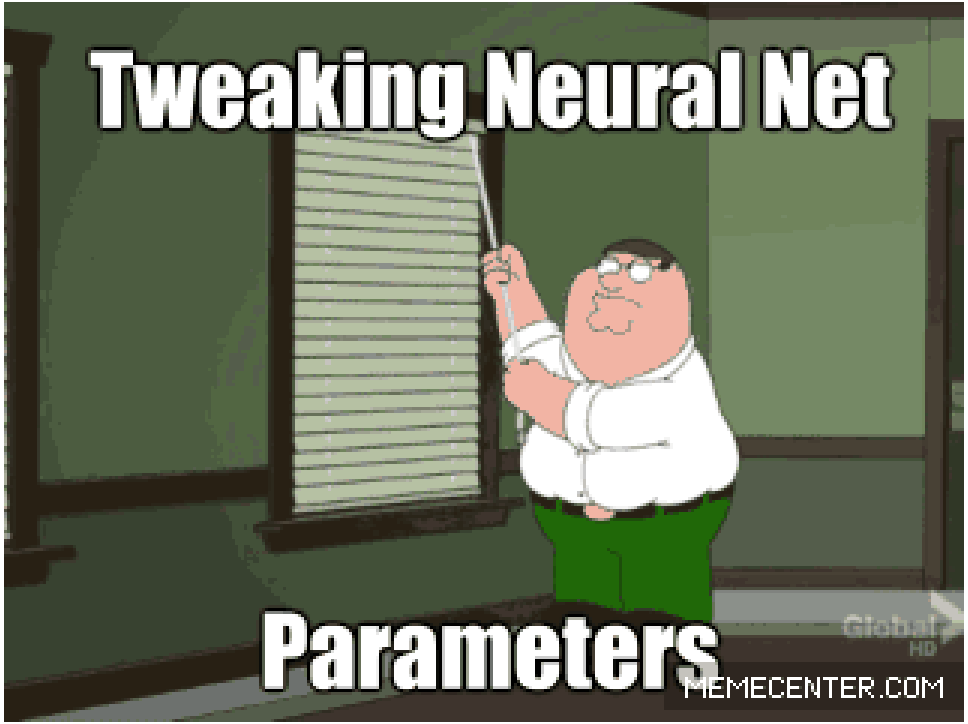
Deep Image: Scaling up Image Recognition

Baidu 百度 95.42%, May 11, 2015

Deep Learning Recipe



Meanwhile...

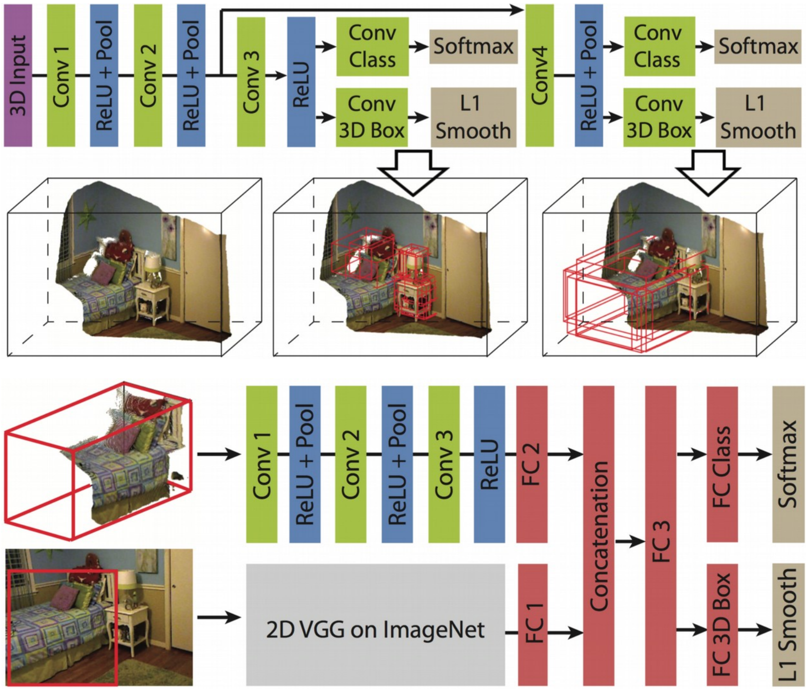


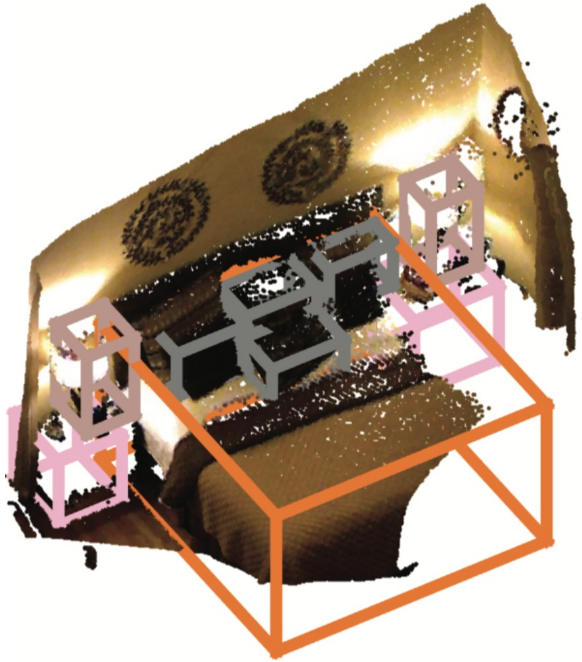
Big Data + Deep Learning

Big Data is the headache;
deep learning is the solution.

- Steve Jurvetson

Successful Case: 3D Object Detector





Improve over 2D deep learning by 13.8 in mAP

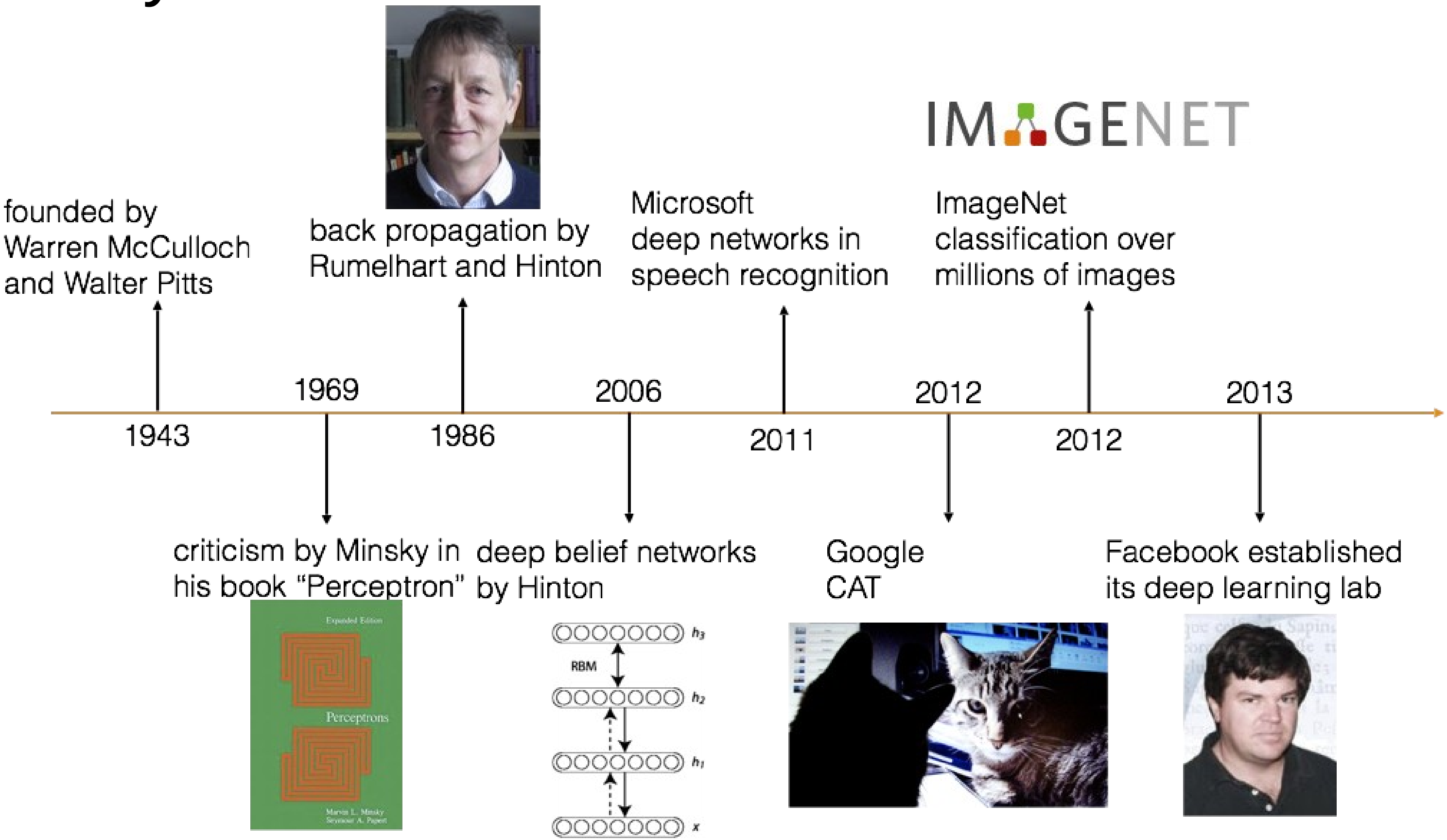
Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images. Song and Xiao, 2015.
http://dss.cs.princeton.edu

Deep Learning

Deep Neural Networks
Convolutional Neural Networks
Deep Belief Networks
Convolutional Deep Belief Networks
Deep Boltzmann Machines
Stacked (Denoising) Auto-Encoders
Deep Stacking Networks
Tensor Deep Stacking Networks (T-DSN)
Spike-and-Slab RBMs (ssRBMs)
Compound Hierarchical-Deep Models

Deep Coding Networks
Multilayer Kernel Machine
Deep Q-Networks
Memory Networks
Long short-term memory
Semantic Hashing
Neural Turing Machines
Memory Networks
Encoder–Decoder networks

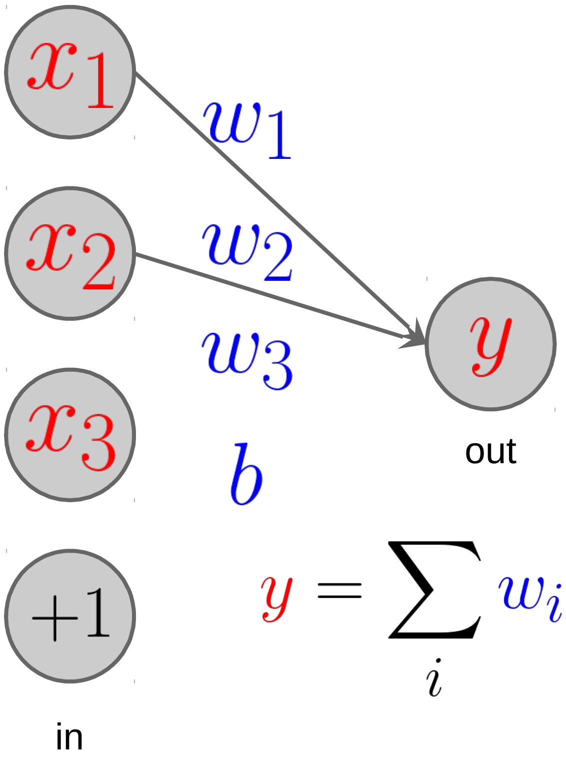
History



founded by Warren McCulloch and Walter Pitts
1943
1969
criticism by Minsky in his book "Perceptron"
Expanded Edition
Perceptrons
Marvin L. Minsky
Seymour A. Papert
back propagation by Rumelhart and Hinton
1986
2006
deep belief networks by Hinton
RBM
h_3
h_2
h_1
x
Microsoft deep networks in speech recognition
2011
2012
Google CAT
IMAGENET
ImageNet classification over millions of images
2012
2013
Facebook established its deep learning lab

Neural Network in 60 Seconds

- Operations are called layers
- Results (in & out) are called responses (aka activation)

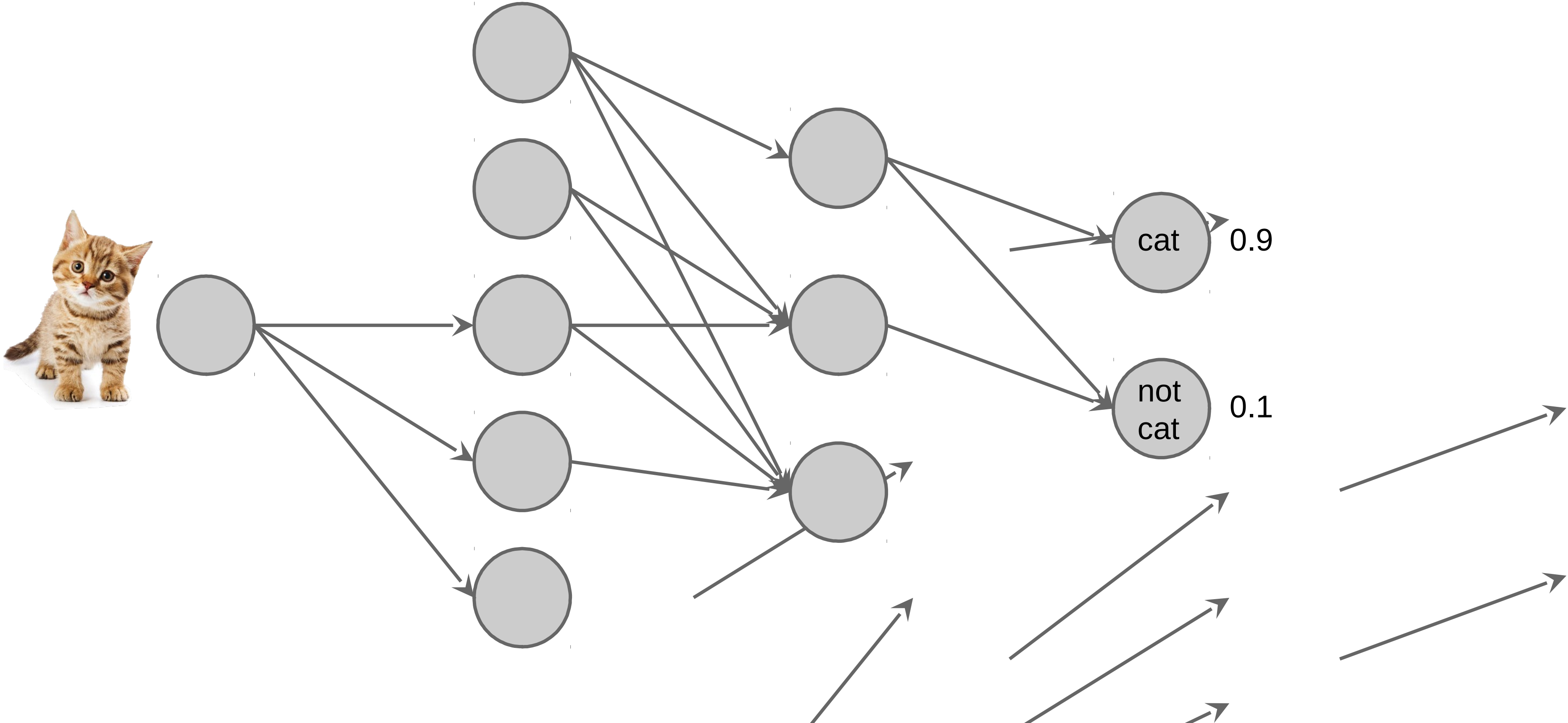


$$y = \sum_i w_i x_i + b$$

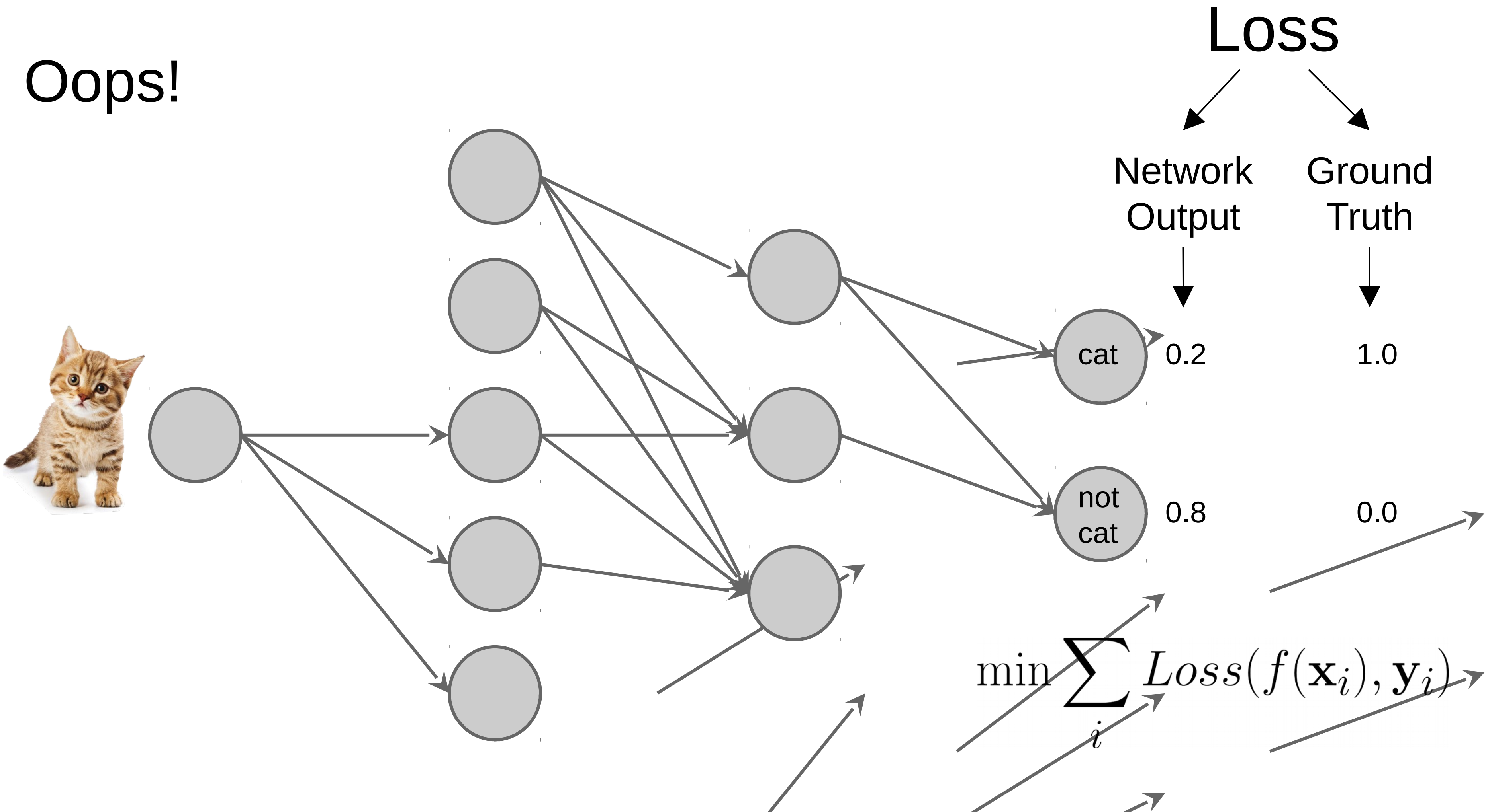
$$y = f(x)$$

any (almost) differentiable function

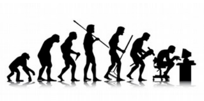
A Good Network



Oops!



$$\min \sum_i Loss(f(\mathbf{x}_i), \mathbf{y}_i)$$

Evolution: Random Search 

Evolutionary adversarial training: the discriminator is the predator. This seaweed-looking fish is the product of the generator. The generator is the biochemical machinery of life, fed with randomly-sampled genome. The problem is that the fish doesn't get to see the gradient back-propagated through the predator. The only way to evaluate the gradient is to modify your genes and see if you get eaten.

- Yann LeCun

Training: Searching for the best weights

$$\min_{\text{weights}} \sum_i Loss(\text{weights}, \text{data}_i)$$

We need to solve

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w})$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_i L(\mathbf{w}, \mathbf{d}_i)$$

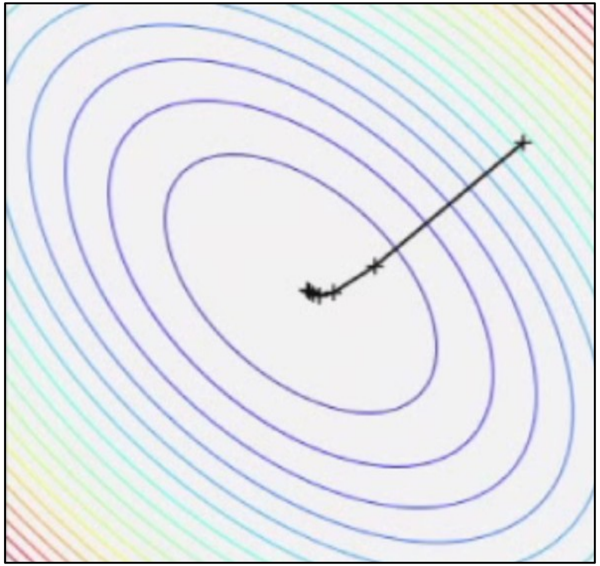
by gradient descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}_t)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \sum_i \frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}_t, \mathbf{d}_i)$$

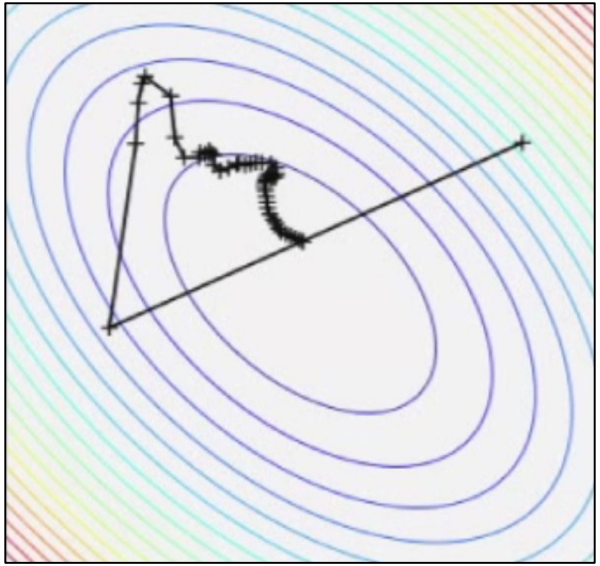
$$\sum_i \frac{\partial L}{\partial \mathbf{w}}(\mathbf{w}_t, \mathbf{d}_i) = \sum_i g(\mathbf{d}_i)$$

Gradient Descent



sum over all training data

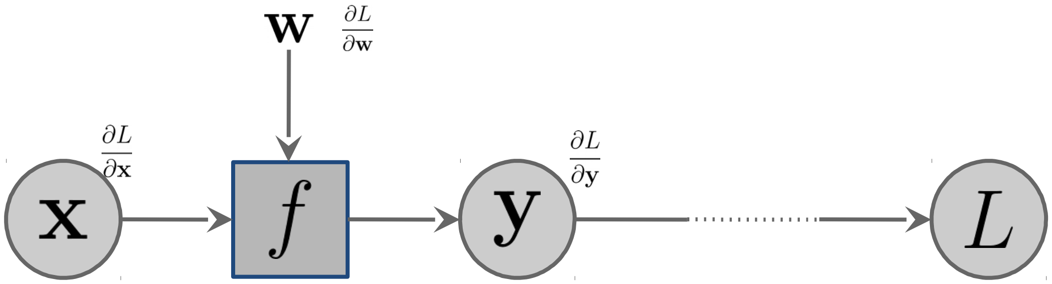
Stochastic Gradient Descent (SGD)



sum over a random subset of training data (called mini-batch). different every iteration.

source of figures: John C. Duchi

Training by Backpropagation (a.k.a. chain rule)



$$\mathbf{y} = f(\mathbf{w}, \mathbf{x})$$

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_i \frac{\partial L}{\partial \mathbf{y}_i} \frac{\partial \mathbf{y}_i}{\partial \mathbf{w}} = \sum_i \frac{\partial L}{\partial \mathbf{y}_i} f_1'(\mathbf{w}_t, \mathbf{x}_i)$$

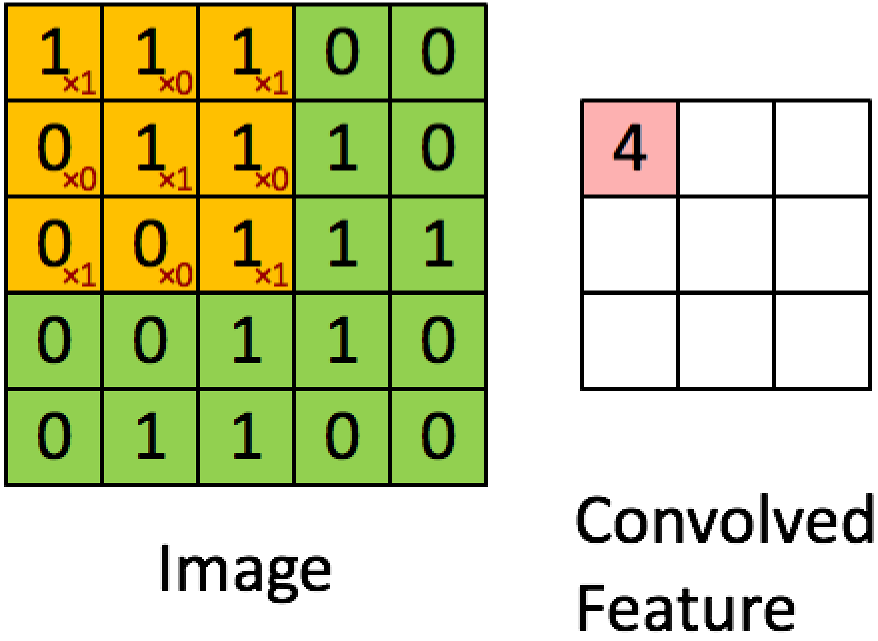
$$\frac{\partial L}{\partial \mathbf{x}_i} = \frac{\partial L}{\partial \mathbf{y}_i} \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}_i} = \frac{\partial L}{\partial \mathbf{y}_i} f_2'(\mathbf{w}_t, \mathbf{x}_i)$$

Training

for many iterations:

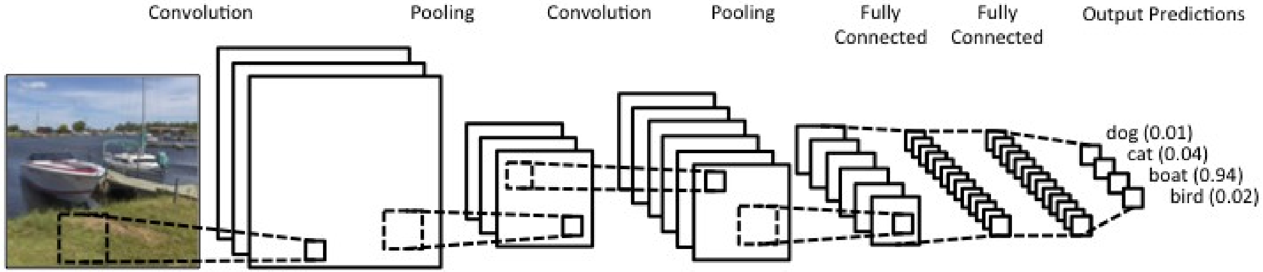
1. randomly sample a mini-batch
2. forward to compute responses
3. backward to compute gradients (for both weights and responses)
4. update weights with the gradients

Convolutional Neural Network

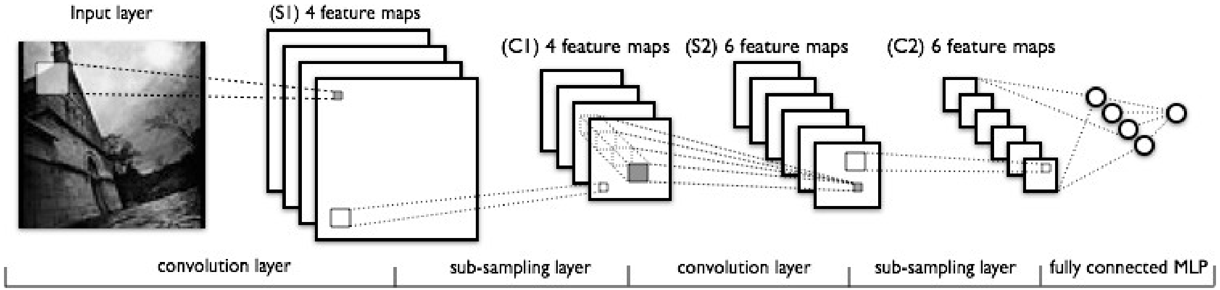


Convolution with 3×3 Filter. Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Convolutional Neural Network



source: http://clarifai.com/technology



source: http://deeplearning.net/tutorial/lenet.html

Parameters for weight update rules

Learning rate: parameter that determines how much an updating step influences the current value of the weights.
Weight decay: an additional term in the weight update rule that causes the weights to exponentially decay to zero, if no other update is scheduled. Similar to the regularization term in SVM <u>to reduce overfitting</u>.
Momentum: <u>To smooth out the gradients over iterations</u>, each update to make is averaged with the weight update we made in the previous iteration. Momentum controls how strong the influence of the previous update.

SGD+L2:

update_history ← momentum * update_history - learning_rate * (gradient + weight_decay * weight)
weight ← weight - update_history

Marvin A minimalist GPU-only N-dimensional ConvNet framework

A Layer is an operation on responses: $\mathbf{y} = f(\mathbf{x})$

in's and out's

$$[\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_m] = f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$$

in-place operation

$$\mathbf{x} = f(\mathbf{x})$$

allow sharing in's with other backprop layer

Parallel in's and out's

$$\mathbf{y}_1 = f_1(\mathbf{x}) \qquad \mathbf{y}_2 = f_2(\mathbf{x})$$

$$\mathbf{y}_1 = f(\mathbf{x}_1, \mathbf{x}_2) \qquad \mathbf{y}_2 = f(\mathbf{x}_3, \mathbf{x}_4) \qquad \mathbf{y}_3 = f(\mathbf{x}_5, \mathbf{x}_6)$$

Layer: Convolution

Common:
name:
phase:
“Training”, “Testing”, “TrainingTesting”
train_me: true / false
Convolution filters:
num_output: number of filters
window: e.g., [5,5,5] for 3D
padding: [0,0,...]
stride: [1,1,...]
upscale: [1,1,...]
group: 1

Weight update and initialization:
weight_filler:
“Xavier”, “Gaussian”, “Constant”
weight_filler_param: 0.0
weight_lr_mult: 1.0
weight_decay_mult: 1.0

bias_filler:
“Xavier”, “Gaussian”, “Constant”
bias_filler_param: 0.0
bias_lr_mult: 2.0
bias_decay_mult: 1.0

Layer: Convolution

Common:

name:

phase:

“Training”, “Testing”, “TrainingTesting”

train_me: true / false

Convolution filters:

num_output: number of filters

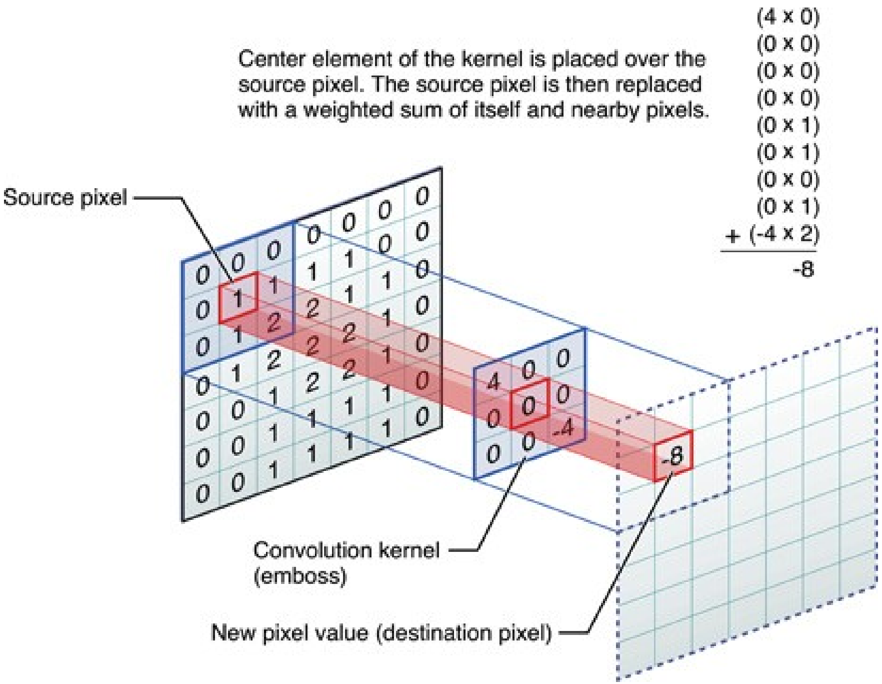
window: e.g., [5,5,5] for 3D

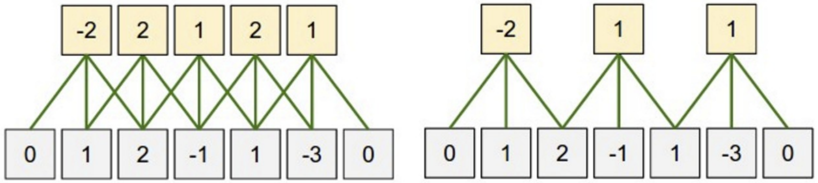
padding: [0,0,...]

stride: [1,1,...]

upscale: [1,1,...]

group: 1





Convolution Stride Size. Left: Stride size 1. Right: Stride size 2. Source: http://cs231n.github.io/convolutional-networks/

Layer: Convolution

Common:

name:

phase:

"Training", "Testing", "TrainingTesting"

train_me: true / false

Convolution filters:

num_output: number of filters

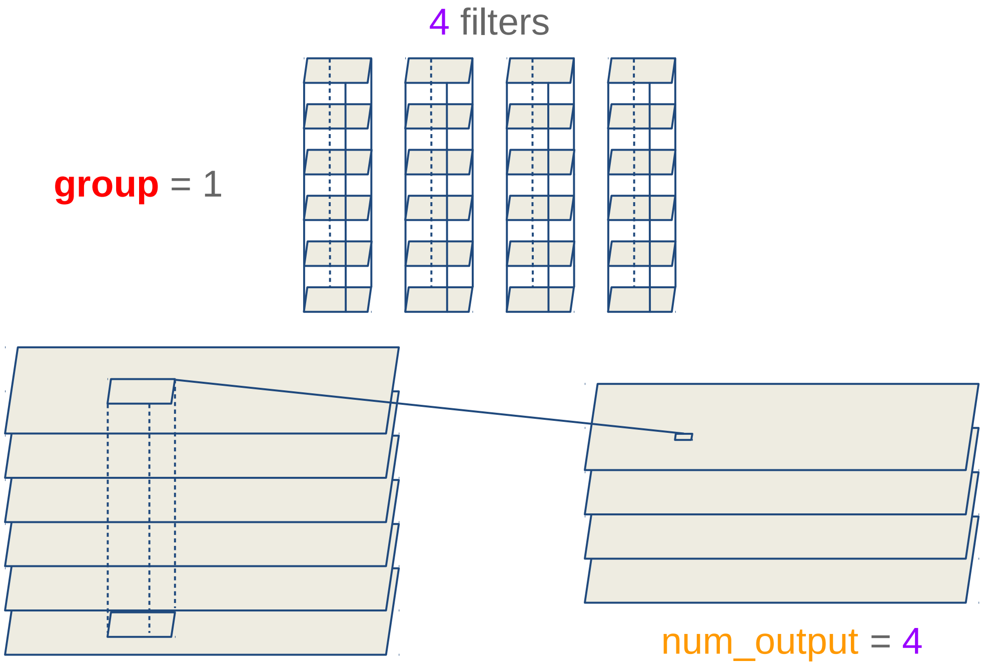
window: e.g., [5,5,5] for 3D

padding: [0,0,...]

stride: [1,1,...]

upscale: [1,1,...]

group: 1



Layer: Convolution

Common:

name:

phase:

"Training", "Testing", "TrainingTesting"

train_me: true / false

Convolution filters:

num_output: number of filters

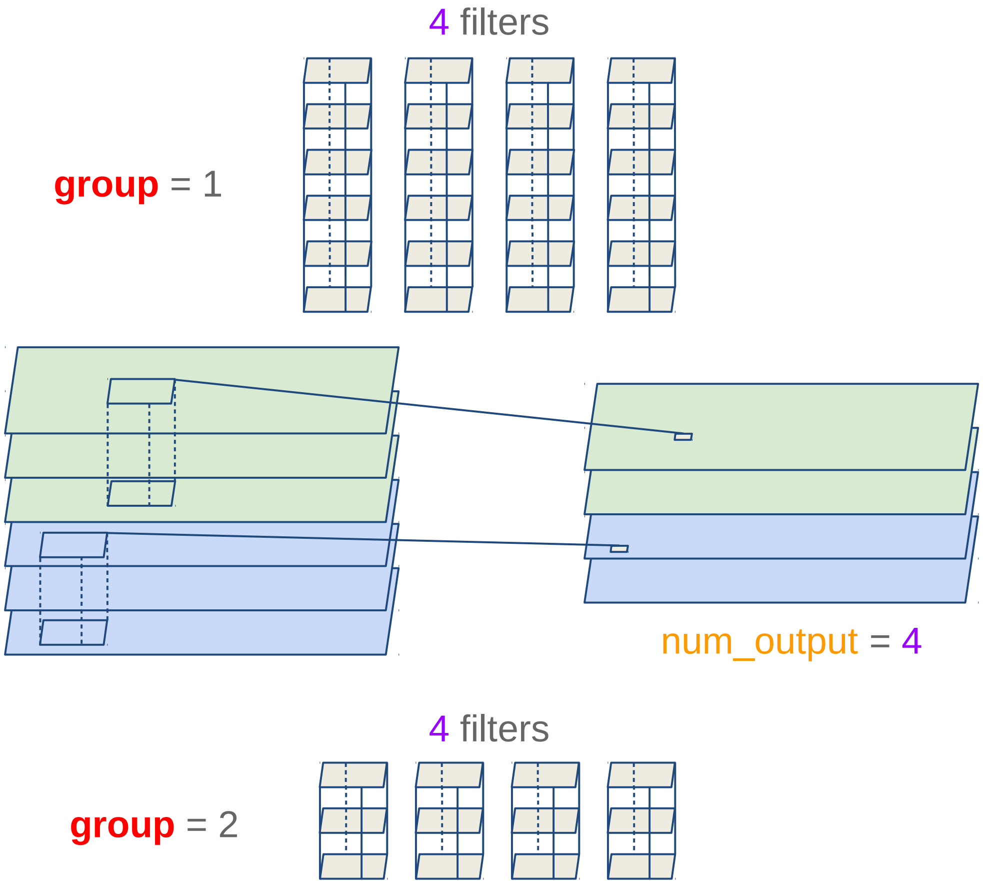
window: e.g., [5,5,5] for 3D

padding: [0,0,...]

stride: [1,1,...]

upscale: [1,1,...]

group: 1



Layer: InnerProduct (a.k.a. **F**ully **C**onnected)

Common:
name:
phase:
"Training", "Testing", "TrainingTesting"
train_me: true / false
Parameters:
num_output: number of output of the layer

Weight update and initialization:
weight_filler:
"Xavier", "Gaussian", "Constant"
weight_filler_param: 0.0
weight_lr_mult: 1.0
weight_decay_mult: 1.0

bias_filler:
"Xavier", "Gaussian", "Constant"
bias_filler_param: 0.0
bias_lr_mult: 2.0
bias_decay_mult: 1.0

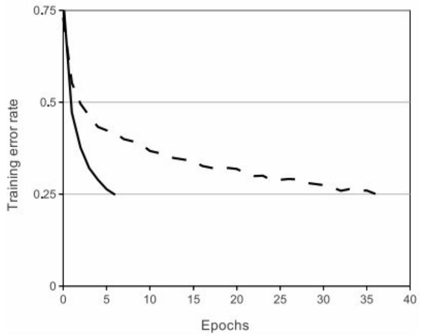
$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

size: num_output × sizeofitem($\mathbf{x}$)

size: num_output

Layer: Activation

A plot from Krizhevsky *et al.* indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.



Common:

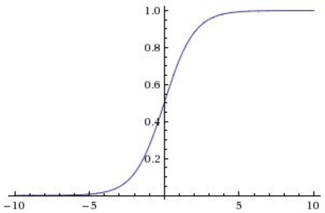
name:

phase:

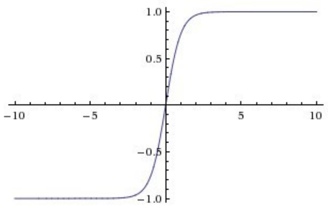
“Training”, “Testing”, “TrainingTesting”

mode:

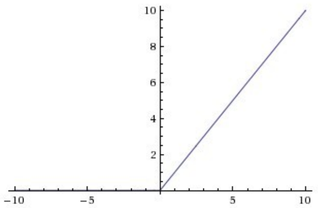
“Sigmoid”, “ReLU”, “TanH”



Sigmoid



TanH



ReLU

Layer: Dropout

Dropout helps to avoid overfitting by randomly drop units (along with their connections) from the network during training to prevent units from co-adapting too much.
During training, dropout layer randomly set some part of an in response to be zeros in the forward based on the dropout_rate. In backward, the gradient is multiply by 1/dropout_rate.
During testing, the dropout layer does nothing.

name:

phase:

“Training”, “Testing”, “TrainingTesting”

dropout_rate: 0.5

Layer: Softmax

name:
phase:
"Training", "Testing", "TrainingTesting"
stable_gradient: true / false
Strong Recommendation:
For numerical stability, it is highly recommend to set stable_gradient = true, and use "MultinomialLogistic_StableSoftmax" for the mode in the loss layer.
For more details about the reasons behind, refer to
http://freemind.pluskid.org/machine-learning/softmax-vs-softmax-loss-numerical-stability/

Layer: Loss

name:
phase:
“Training”, “Testing”, “TrainingTesting”
mode:
“MultinomialLogistic_StableSoftmax”, “MultinomialLogistic”, “SmoothL1”, “Contrastive”, “EuclideanSSE”, “HingeL1”, “HingeL2”, “SigmoidCrossEntropy”, “Infogain”
loss_weight: 1.0
margin: 1.0
loss_weights: an array of real numbers for weight each channels
in:
the first in is the network prediction
the second in is the ground truth label
optionally, the third in is a weight array (for the first three mode only)

Layer: Pooling

Common:

name:

phase:

“Training”, “Testing”, “TrainingTesting”

Parameters:

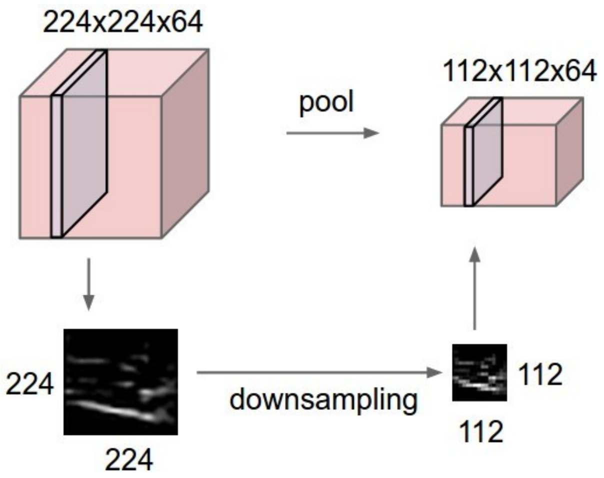
mode:

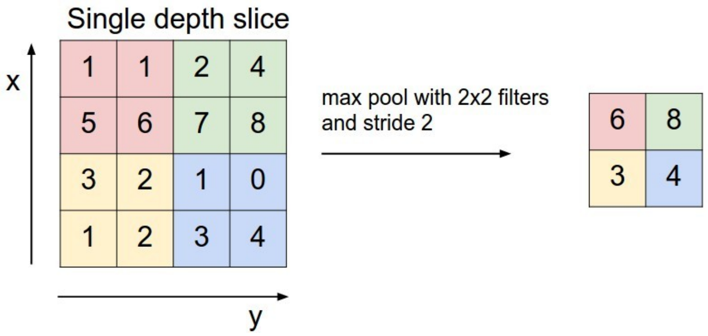
“max”, “average_include”, “average_exclude”

window: e.g., [5,5,5] for 3D

padding: [0,0,...]

stride: [1,1,...]





Source: http://cs231n.github.io/convolutional-networks/#pool

Layer: Concat

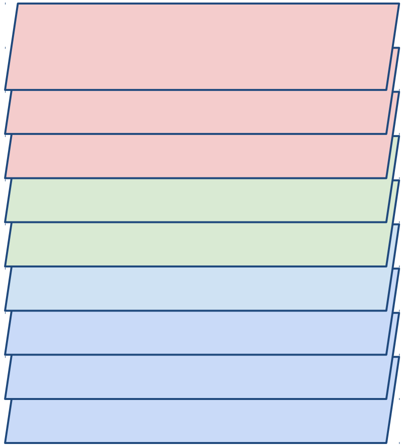
Concatenate inputs in channels:

name:

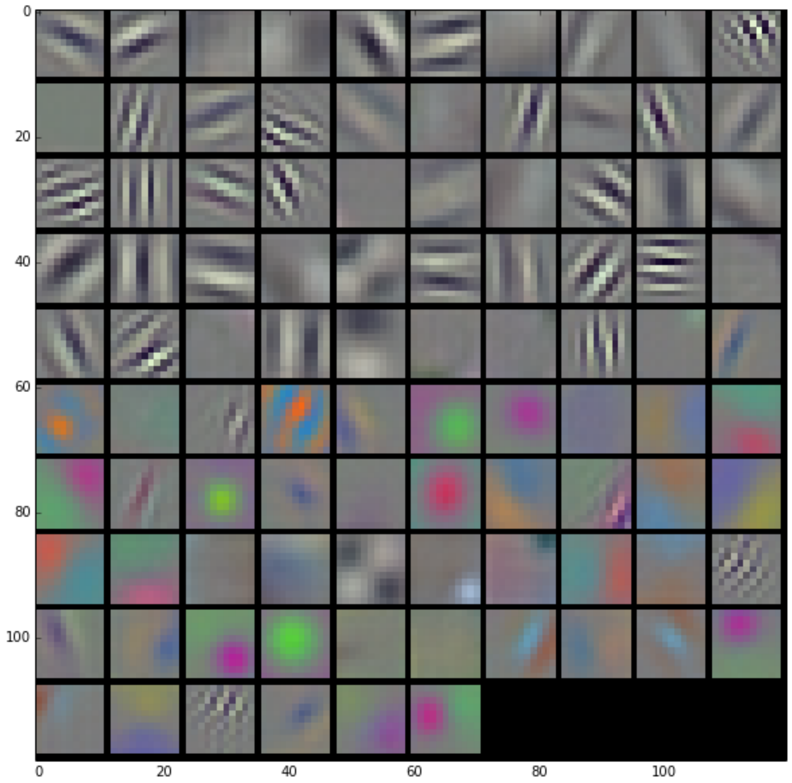
phase:

“Training”, “Testing”, “TrainingTesting”

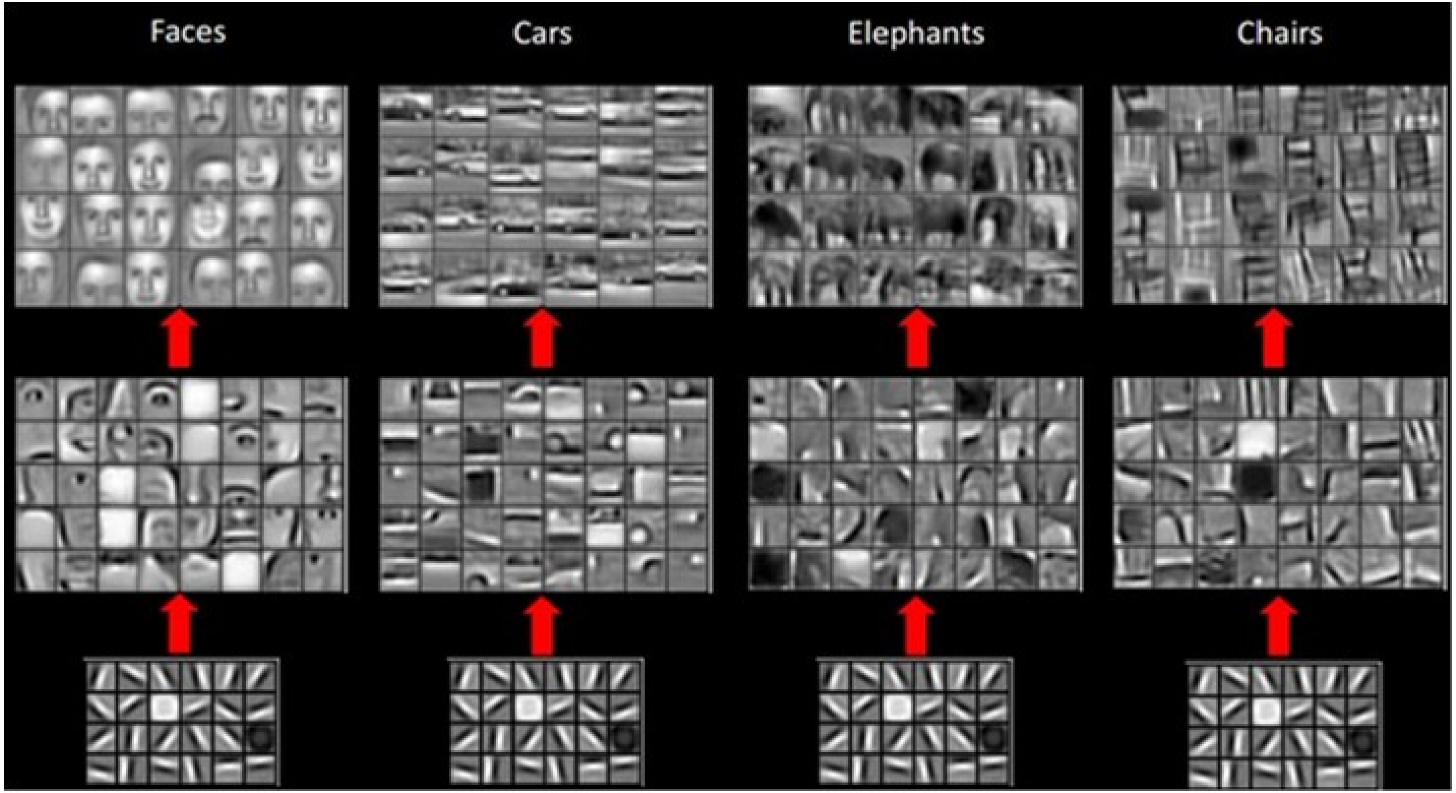
Let’s say that there are N in’s and M out’s. It will divide the N in’s into M groups. Each of the M groups of ins will be concatenated to generate M outs in total.



Deeper into deep learning: Hierarchy is key



Deeper into deep learning: Hierarchy is key



Deeper into deep learning: Why CNNs ?

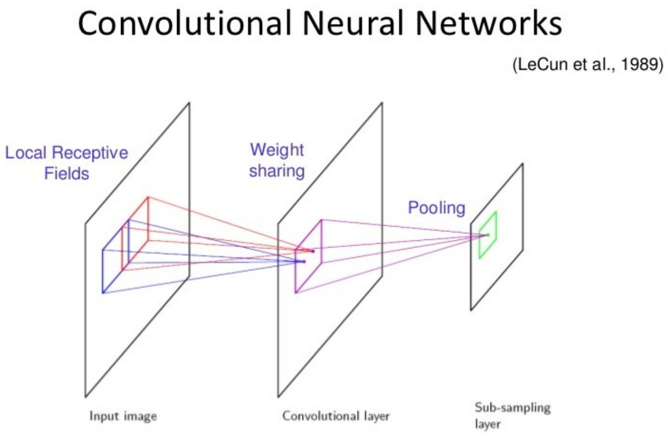
Neural Networks were introduced during the 1960s.
Through the 80s, they contributed to the AI winter
(a lot of expectations and claims, little delivery)
Mainly because data and computing power were not available.

Today, we have both
 (but we still need better learning algorithms)

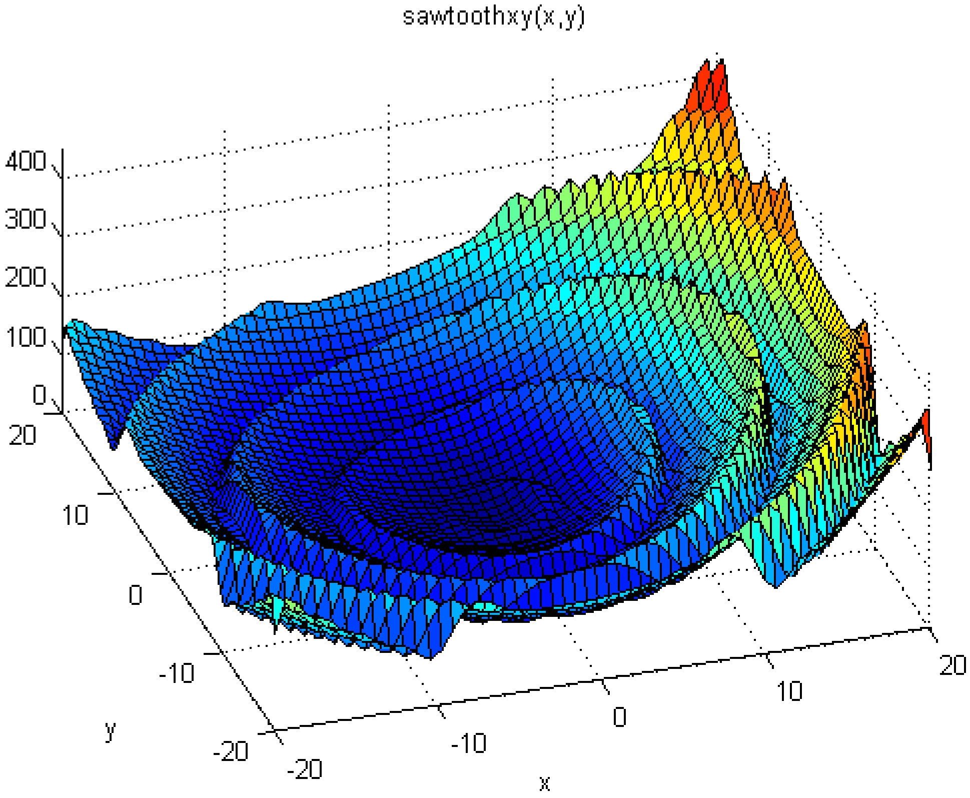
Deeper into deep learning: Why CNNs ?

Neural Networks vastly overfitted in the 90s with little data.
Yann LeCun ('89) proposed weight sharing within each layer.
Reduced the number of learnable parameters by a lot.
Helps reduce overfitting, why ?

Weight sharing effectively is convolution.

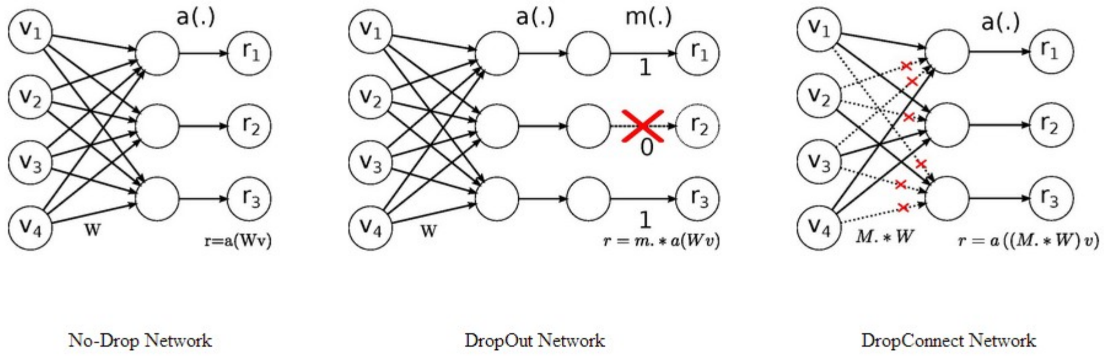


Deeper into deep learning: Local Minima



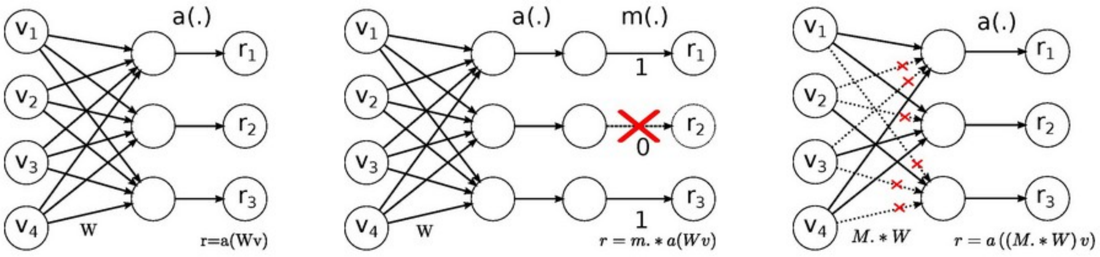
Deeper into deep learning: Dropout

Drop out helps the network generalize better. One of the most successful recent techniques since scaling up.
Recall, during training it randomly puts activations to zero (proportional to drop out rate). At test time, it does nothing.



Deeper into deep learning: Dropout

Randomly “inhibiting” the network keeps it from falling into a premature local minima.
Randomization might help in other things ! (Dropout/DropConnect)



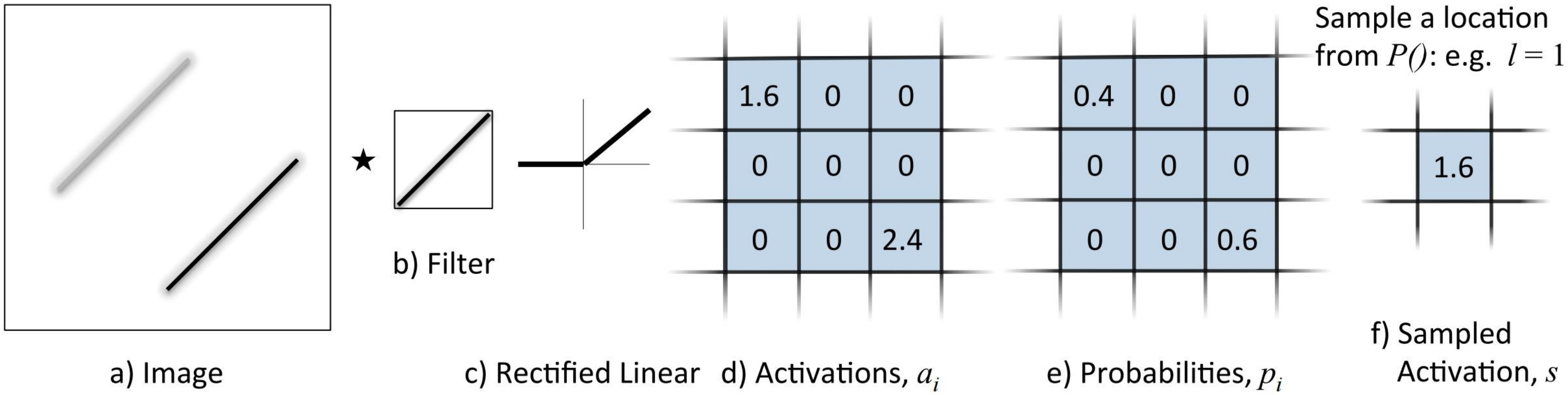
No-Drop Network

DropOut Network

DropConnect Network

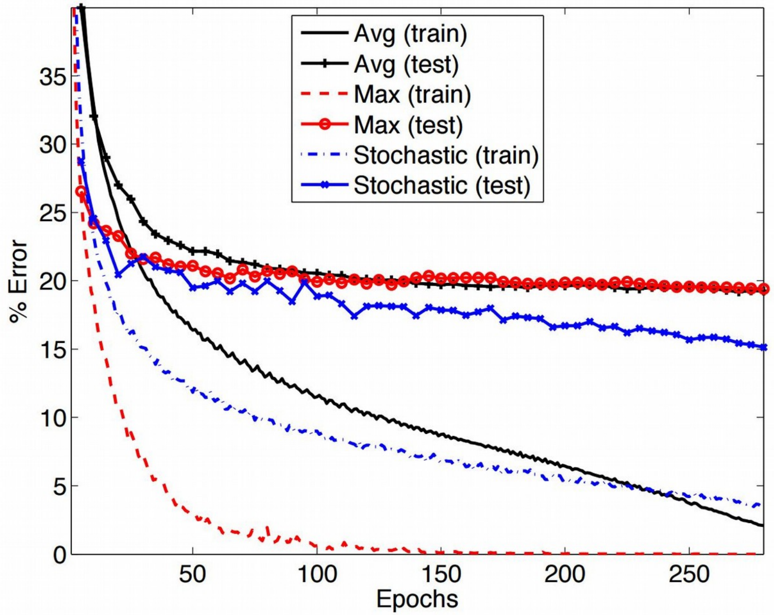
Deeper into deep learning: Stochastic Pooling

Matt Zeiler and Rob Fergus proposed stochastic pooling in 2013.



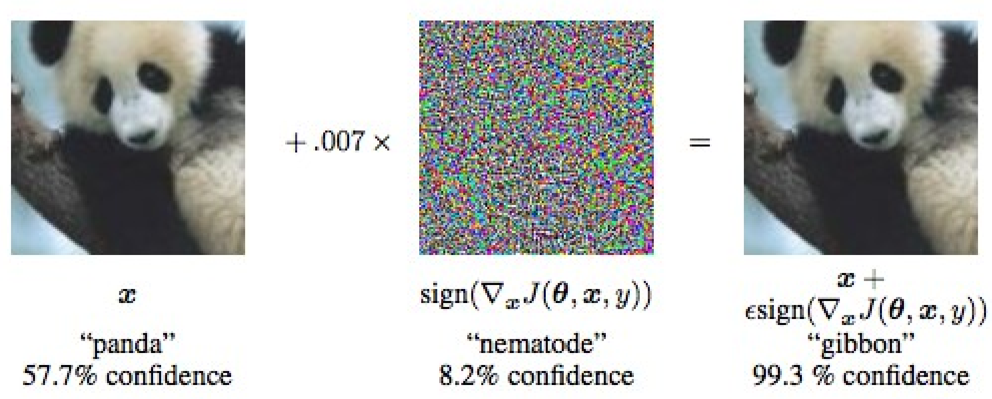
Deeper into deep learning: Stochastic Pooling

CIFAR-10 results
Stochastic generalizes the best
Max and mean pooling work similarly
Max generalizes better earlier
Randomization clearly helps !



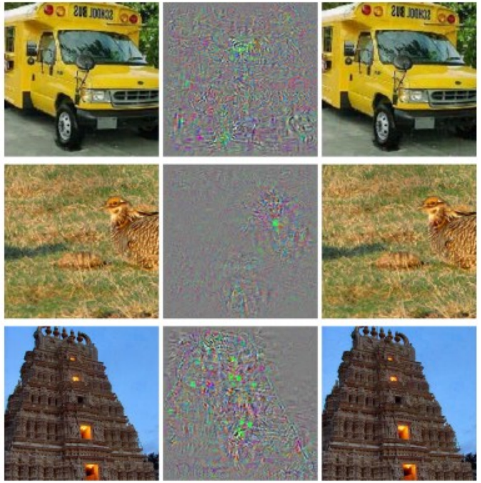
Deeper into deep learning: Perception Solved ?

CNNs have pushed the state-of-the-art for most vision tasks.
However, they have limitations.
The mapping from images to the label space is not “stable”.
Specifically, a small carefully chosen distortion (imperceivable to the human eye) can make the CNN misclassify deterministically.



Deeper into deep learning: Perception Solved ?

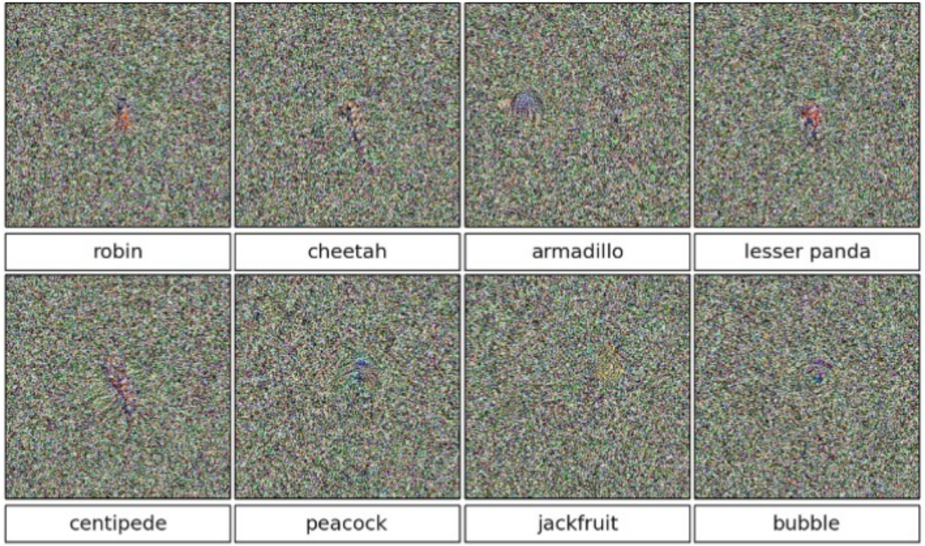
Left column original image, center is the difference image, right column classified as Ostrich (for both image blocks).



“Intriguing properties of neural networks”, Szegedy et. al., 2014

Deeper into deep learning: Perception Solved ?

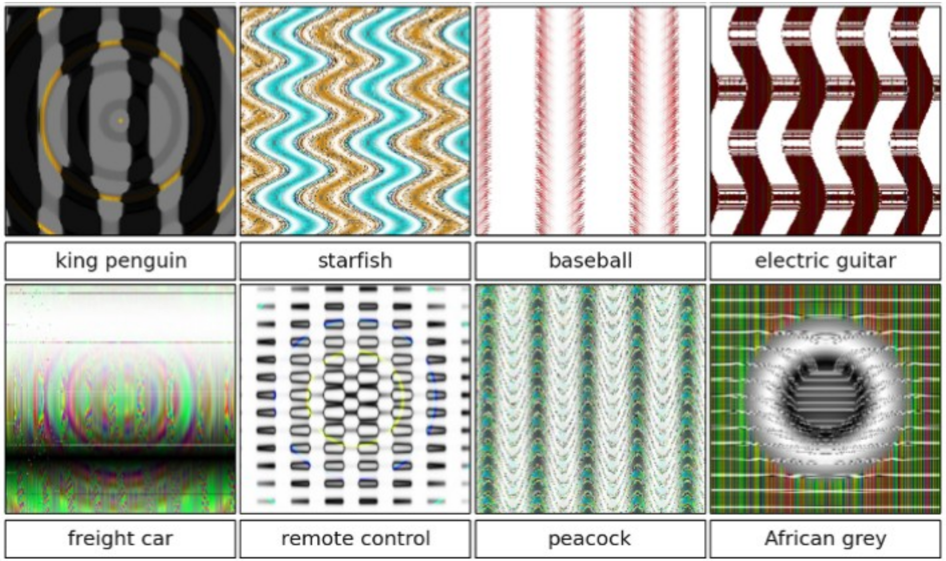
More images can be evolved (genetic algorithms) or found **directly (back propagation)** that fool CNNs.



“Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”, Nguyen et. al., CVPR 2015

Deeper into deep learning: Perception Solved ?

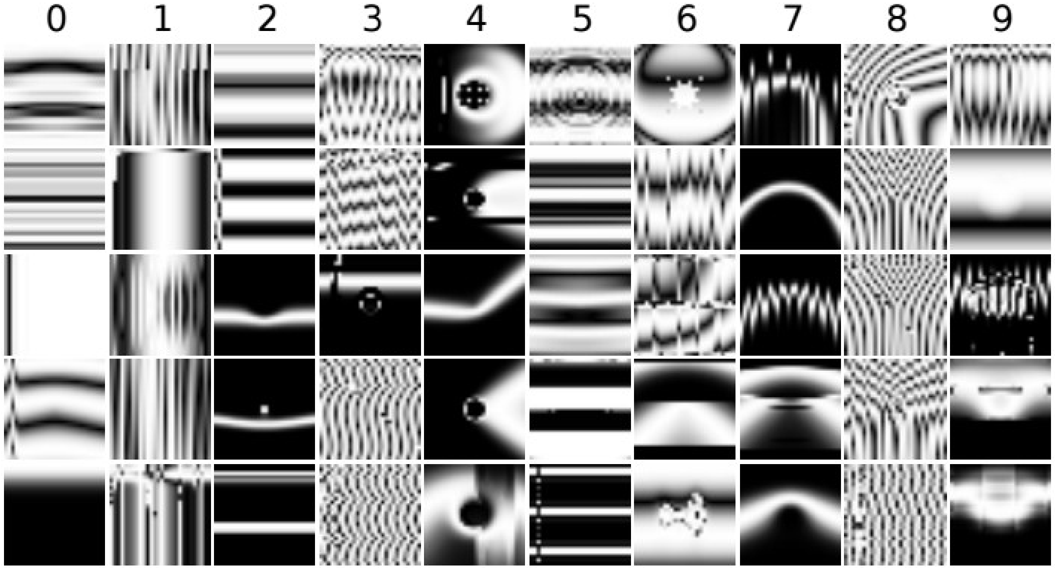
More images can be **evolved (genetic algorithms)** or found directly (back propagation) that fool CNNs.



“Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”, Nguyen et. al., CVPR 2015

Deeper into deep learning: Perception Solved ?

Images that appear to the CNN (LeNet) to be from MNIST.

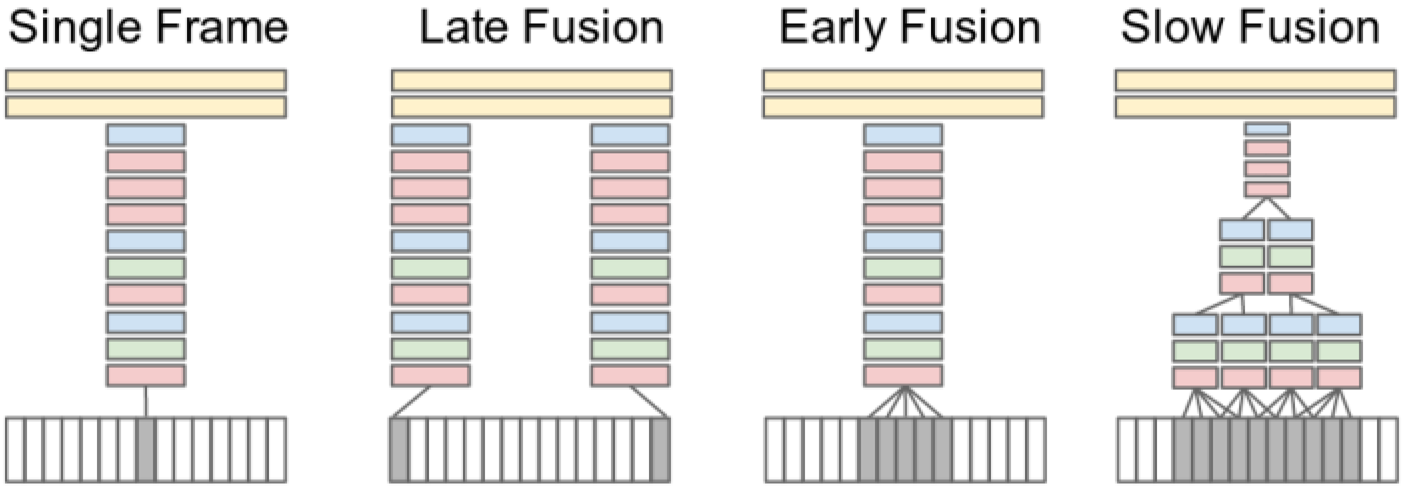


"Deep neural networks are easily fooled: High confidence predictions for unrecognizable images", Nguyen et. al., CVPR 2015

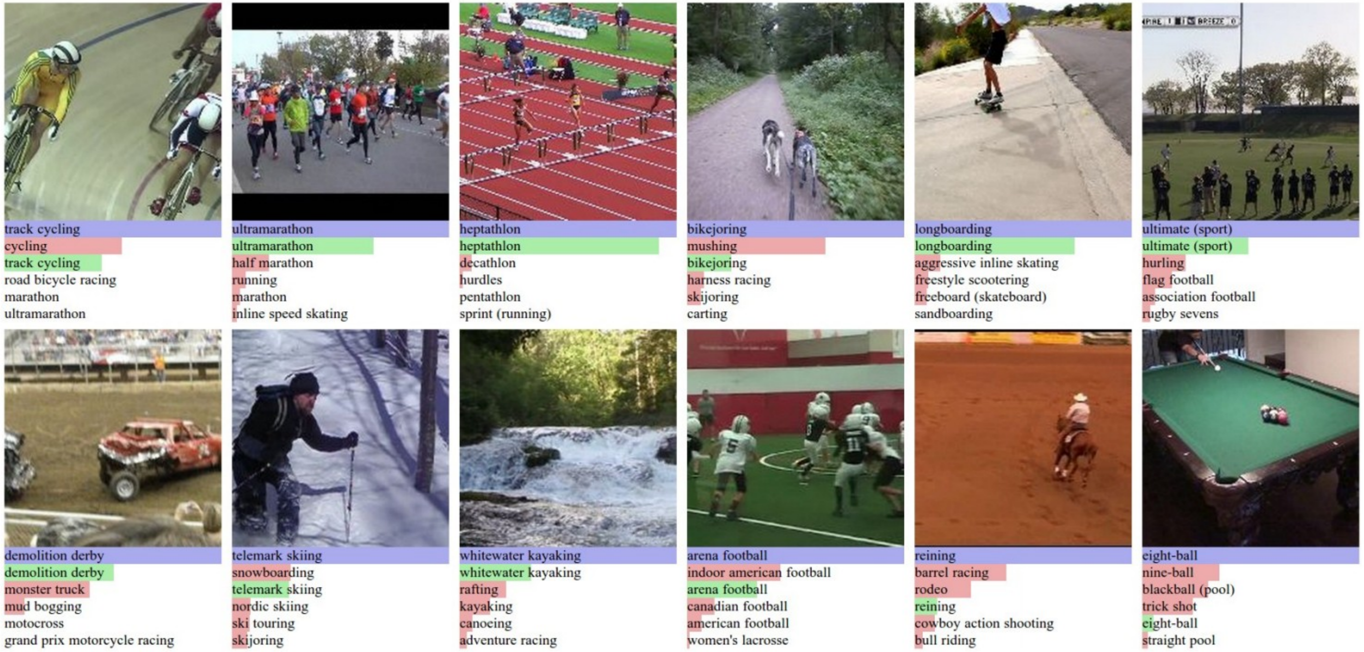
Applications of CNNs

Video Classification
Image Segmentation
Object Detection
Face Landmarking
Face Recognition
Face Detection

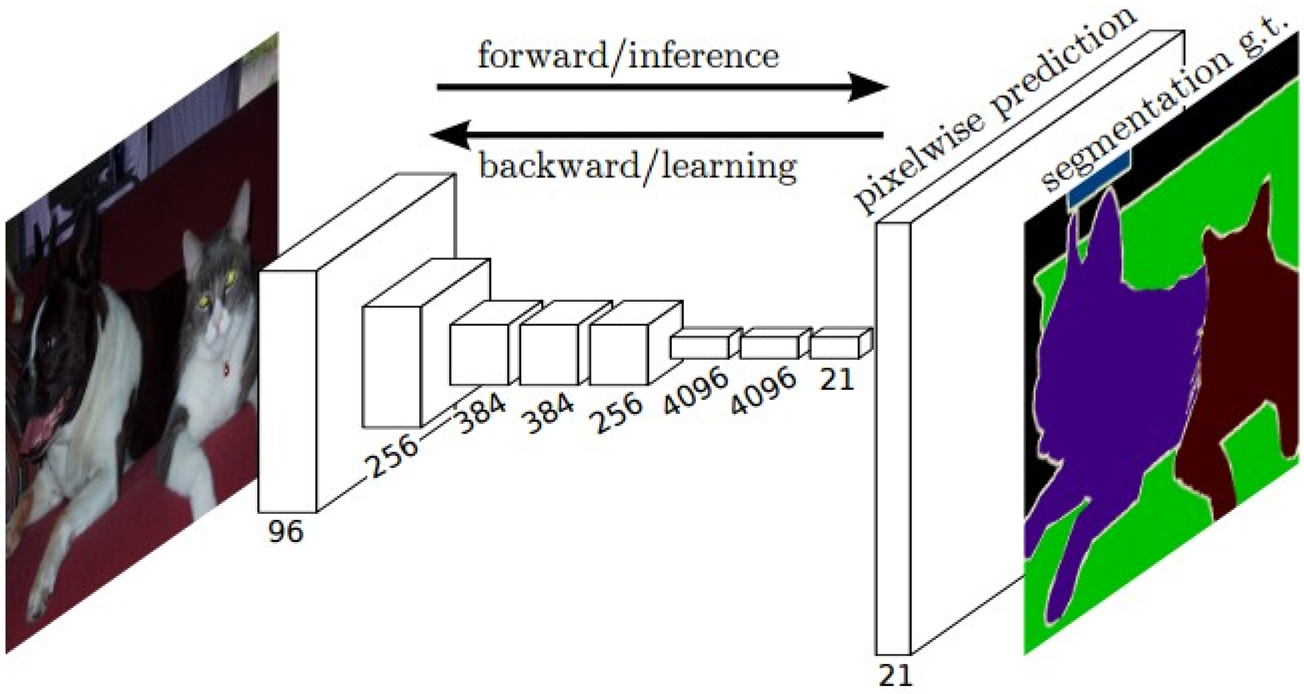
Applications of CNNs: Video Classification



Applications of CNNs: Video Classification



Applications of CNNs: Image Segmentation



Applications of CNNs: Image Segmentation

Applications of CNNs: Face Recognition