COMP 551 – Applied Machine Learning
Lecture 16: Deep Learning

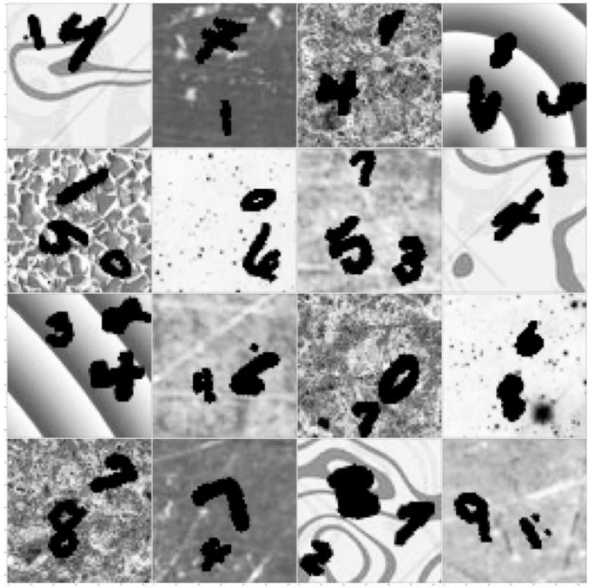
Instructor: Ryan Lowe *(ryan.lowe@cs.mcgill.ca)*

Slides mostly by: Joelle Pineau

Class web page: *www.cs.mcgill.ca/~hvanho2/comp551*

Announcements

- Project 4 released! Due on **March 21st**

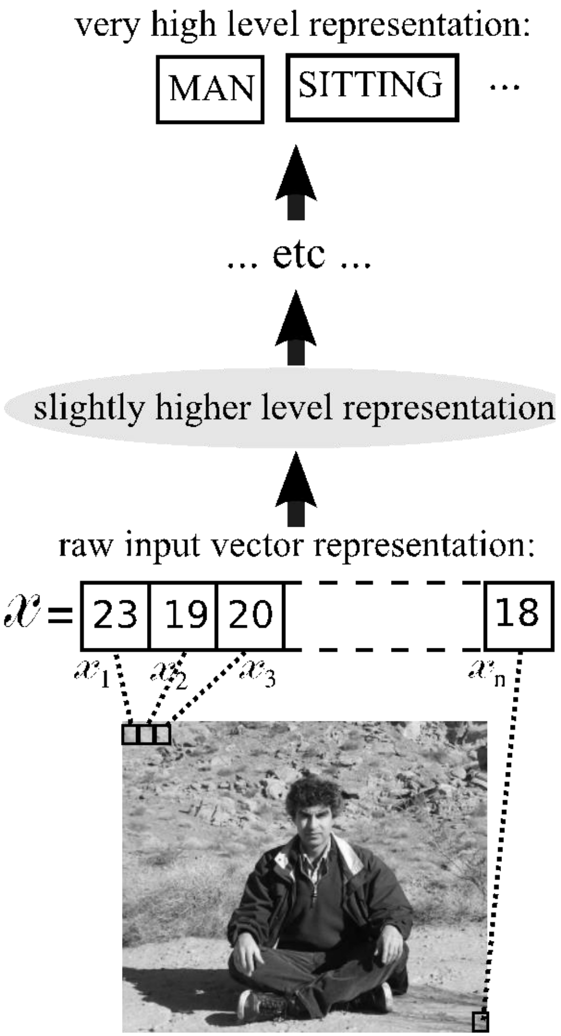


- 3rd tutorial on Friday, 5-6pm, on PyTorch

What is deep learning?

- Processing of data through **multiple layers** of **non-linear functions** to produce an output
- Not just neural networks!
  - Includes neural networks, but also Boltzmann machines, deep belief networks, CNNs, RNNs, etc.
- Main goal is to learn a **representation** of the data that is useful for many different tasks
  - **Representation** of data: function or transformation of the data into a (usually smaller) form that is easier to use (to solve various tasks)

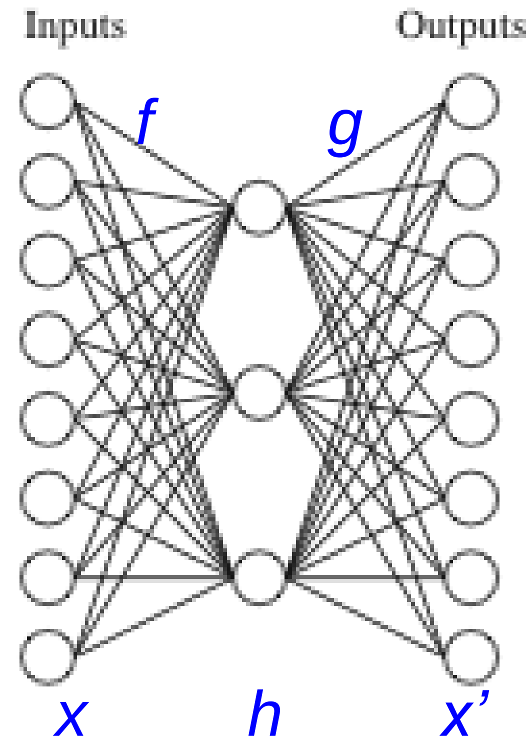
The deep learning objective



Learning an autoencoder function

- **Goal**: Learn a compressed representation of the input data.
- We have two functions:
  - **Encoder**: $h = f_W(x) = s_f(Wx)$
  - **Decoder**: $x' = g_{W'}(h) = s_g(W'h)$

  where $s()$ is the activation function and W, W' are weight matrices.



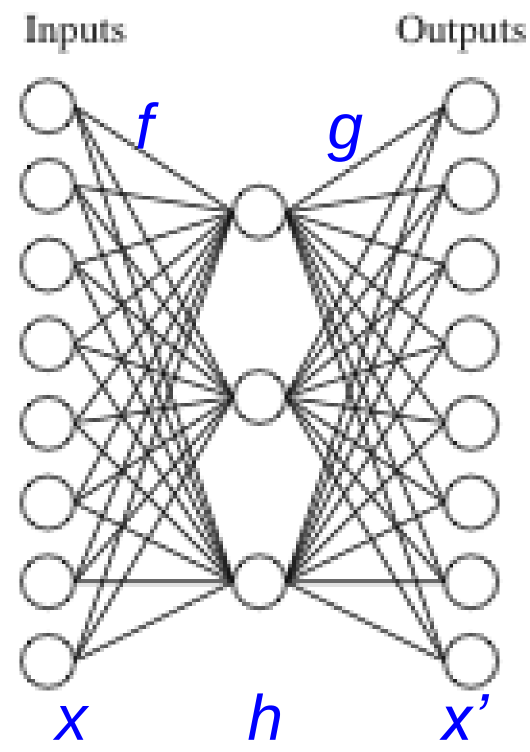
Learning an autoencoder function

- **Goal**: Learn a compressed representation of the input data.
- We have two functions:
  - **Encoder**: $h = f_W(x) = s_f(Wx)$
  - **Decoder**: $x' = g_{W'}(h) = s_g(W'h)$

  where $s()$ is the activation function and W, W' are weight matrices.
- To train, minimize reconstruction error:

  $Err(W,W') = \sum_{i=1:n} L [\, x_i , g_{W'}(f_W(x_i)) \,]$

  using squared-error loss (continuous inputs) or cross-entropy (binary inputs).



PCA vs autoencoders

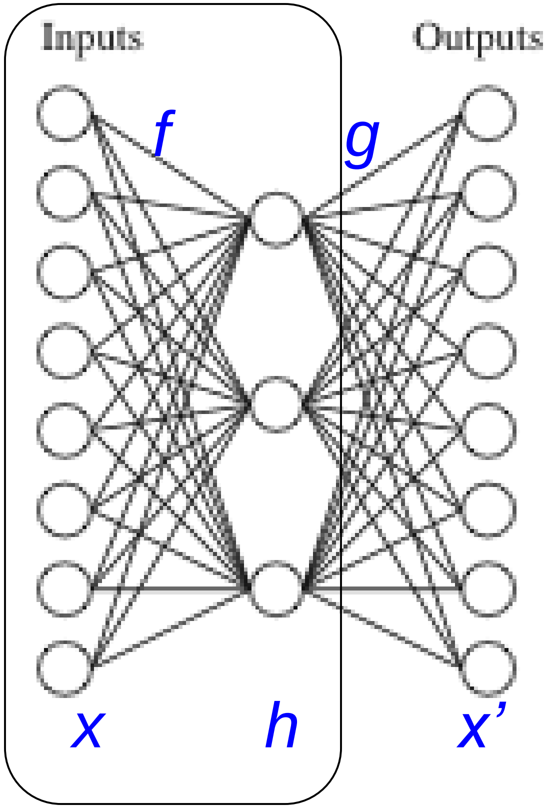
In the case of a linear function:

$f_W(x) = Wx \qquad g_{\hat{W}}(h) = W'h$,

with squared-error loss:

$Err(W,W') = \sum_{i=1:n} || x_i - g_{W'}(f_W(x_i)) ||^2$

we can show that the minimum error solution W yields the same subspace as PCA.



Regularization of autoencoders

- Weight tying of the encoder and decoder weights (*W=W'*) to explicitly constrain (regularize) the learned function.

- How can we generate **sparse** autoencoders? (And also, why?)
  - Directly penalize the output of the hidden units (e.g. with L1 penalty) to introduce sparsity in the weights.
  - Helps 'disentangle' some of the factors of variation in the data

Denoising autoencoders

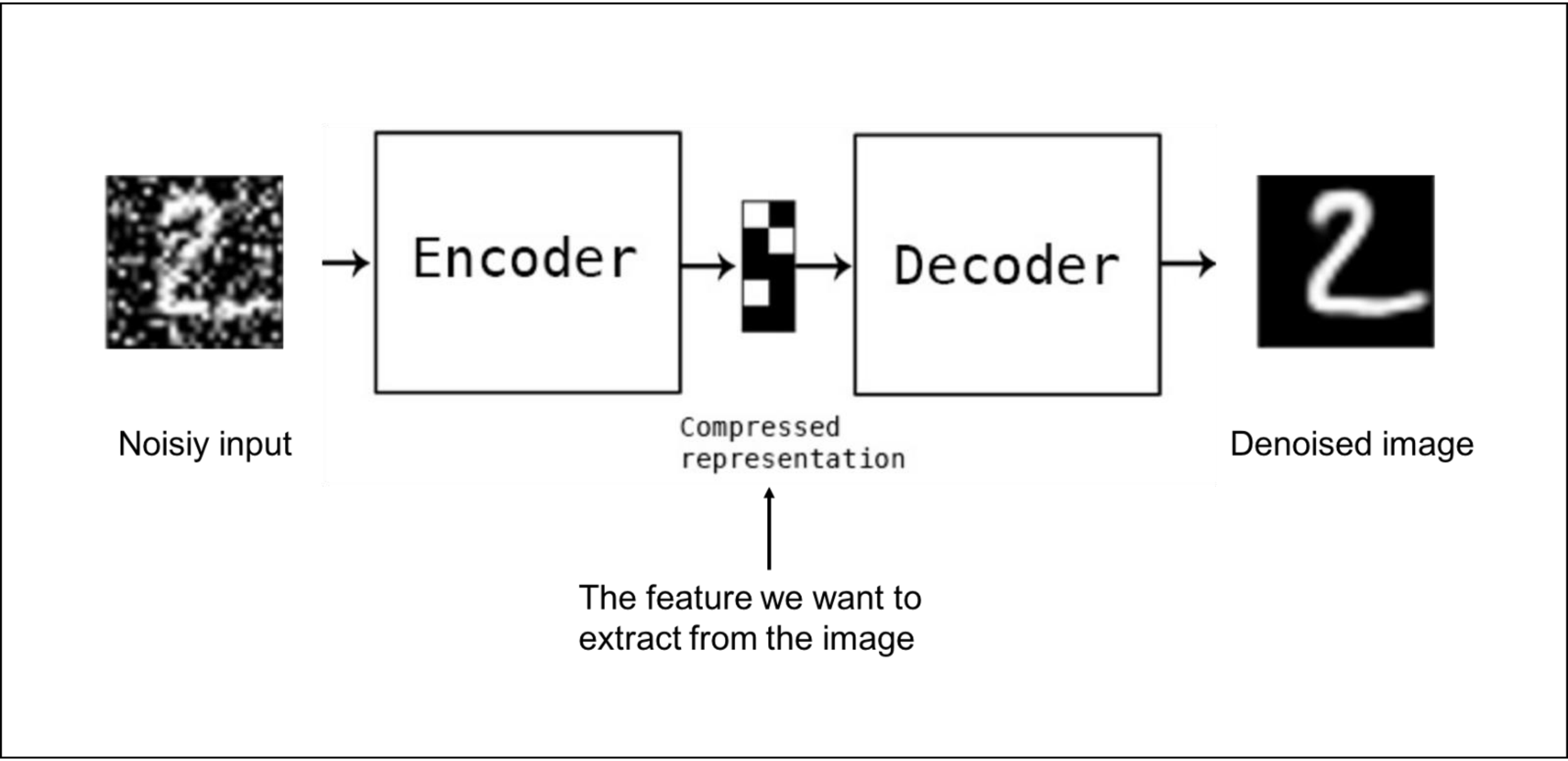
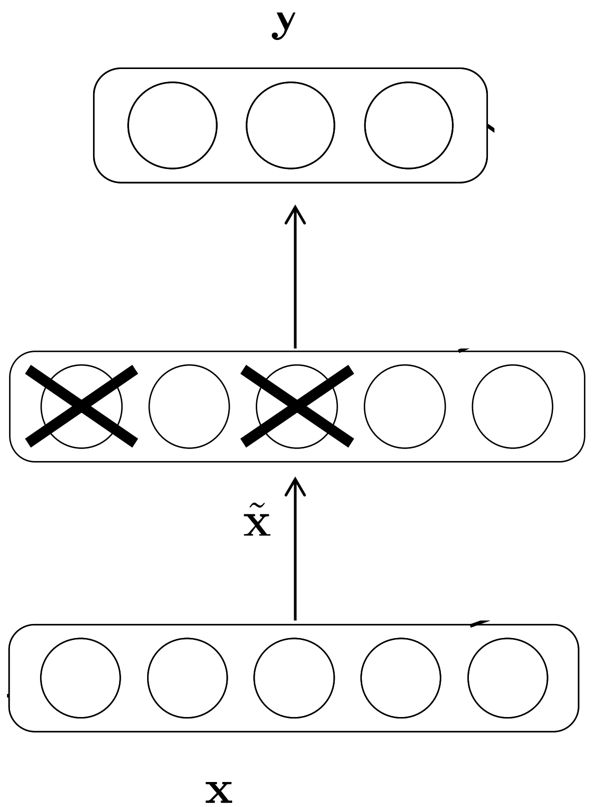


Image source: blog.sicara.com

Denoising autoencoders

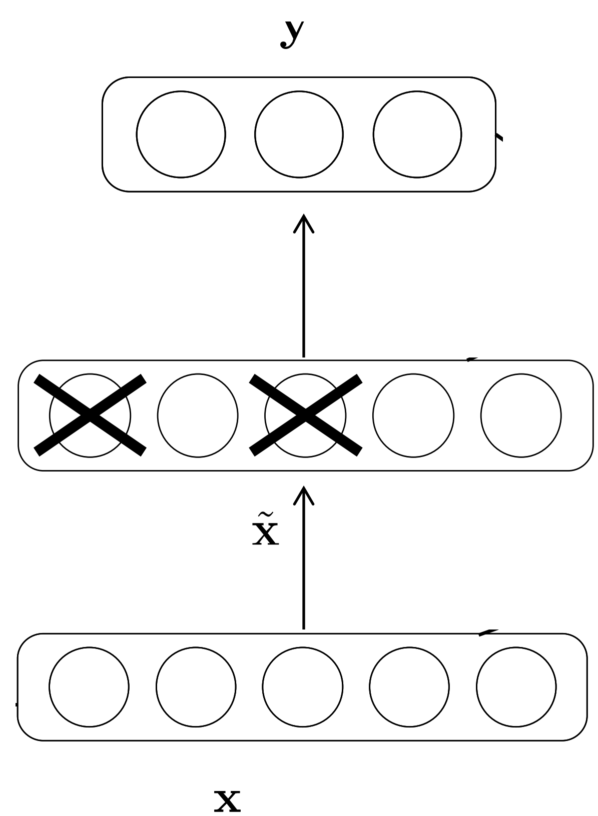
- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.



Denoising autoencoders

- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.

- **Corruption processes**:
  - Additive Gaussian noise
  - Randomly set some input features to zero.
  - *More noise models in the literature*.



Denoising autoencoders

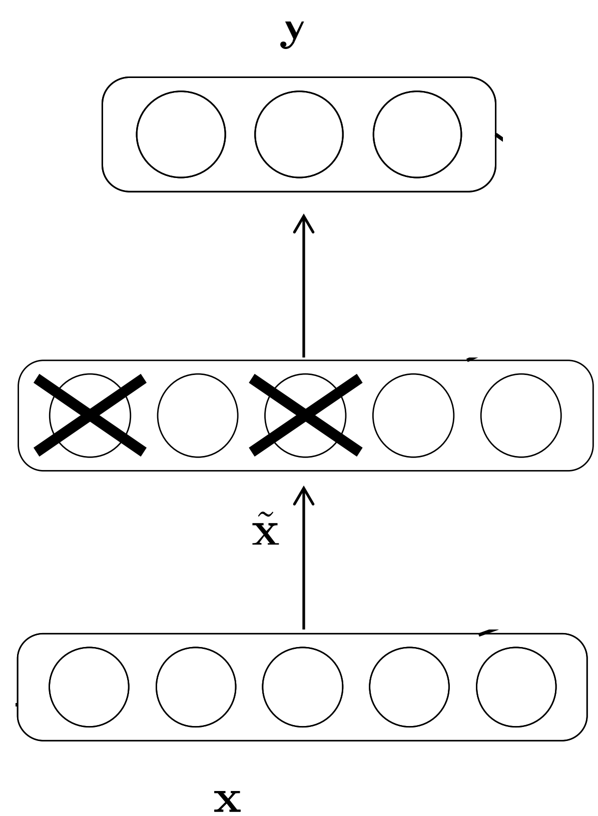
- **Idea**: To force the hidden layer to discover more robust features, train the autoencoder with a corrupted version of the input.

- **Corruption processes**:
  - Additive Gaussian noise
  - Randomly set some input features to zero.
  - *More noise models in the literature*.

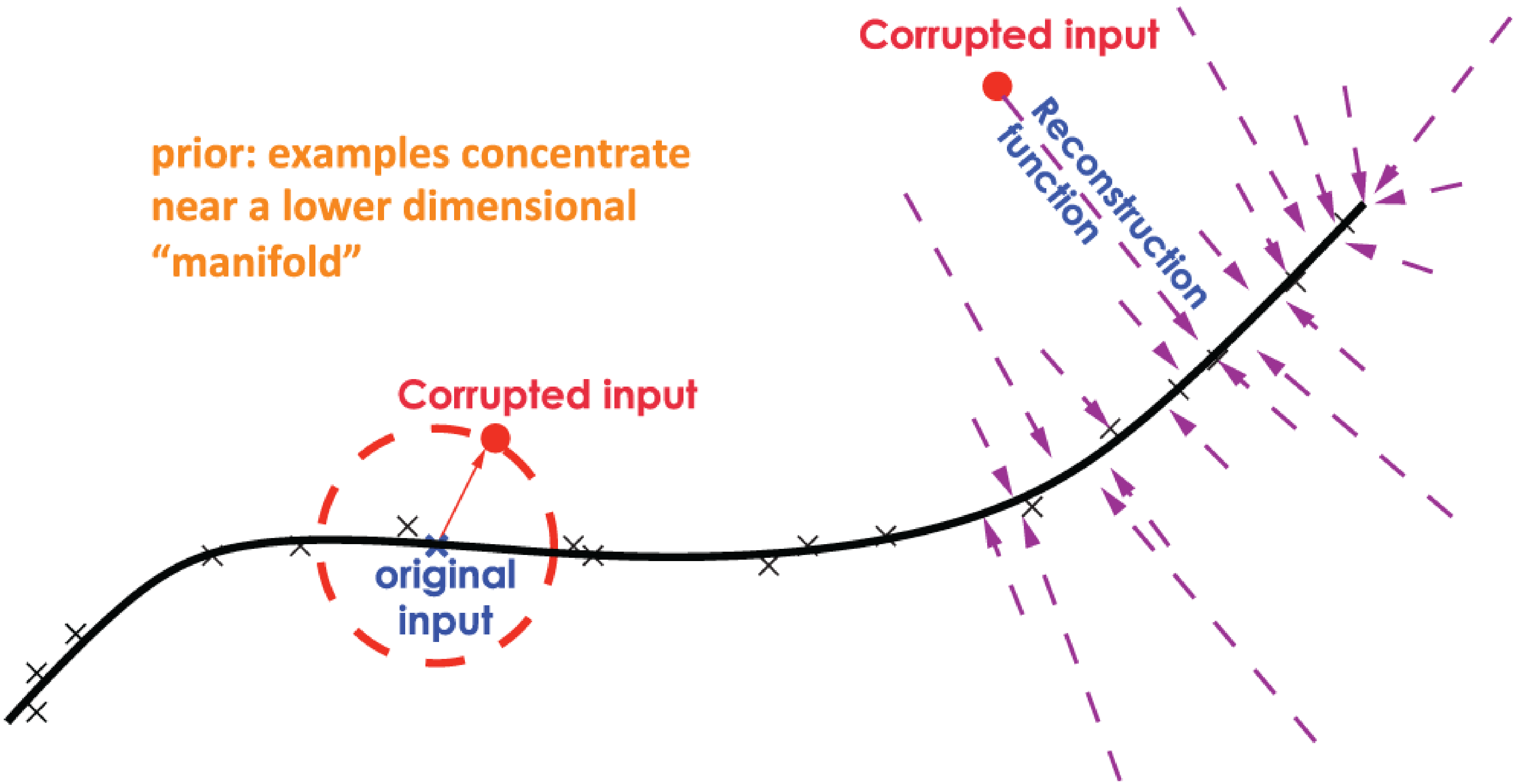
- **Training criterion**:

  $Err(W,W') = \sum_{i=1:n} E_{q(xi'|xi)} L [x_i , g_{W'}(f_W(x_i'))]$

  where x is the original input, x' is the corrupted input, and $q()$ is the corruption process.



Denoising autoencoders



Contractive autoencoders

- **Goal**: Learn a representation that is robust to noise and perturbations of the input data, by regularizing the latent space
- **Contractive autoencoder training criterion**:

$$Err(W,W') = \sum_{i=1:n} L\, [\, x_i\, ,\, g_{W'}(f_W(x_i')) \,] + \lambda ||J(x_i)||_F^2$$

where $J(x_i)=\partial f_W(x_i)/\partial x_i$ is a Jacobian matrix of the encoder evaluated at x_i, F is the Frobenius norm, and λ controls the strength of regularization.

- **Idea:** penalize the model if a small change in input will result in a big change in representation (output of encoder)

Many more similar ideas in the literature…

Unsupervised pretraining

- Autoencoders are a kind of ‘unsupervised learning’
- When do we want to use autoencoders?

1. Want to learn representations (features) of the data, but not sure for what task
2. Useful as a kind of ‘extra data’ for supervised tasks (e.g. ***pretraining***)
3. Can be used for clustering or visualization

Variety of training protocols

- Purely supervised:
  - Initialize parameters randomly.
  - Train in supervised mode (gradient descent w/backprop.)
  - Used in most practical systems for speech and language.
- Unsupervised pretraining + supervised classifier on top:
  - Train an autoencoder to learn features of the data.
  - Train a supervised classifier on top, keeping other layers fixed.
  - Good when very few labeled examples are available.
- Unsupervised pretraining + global supervised fine-tuning.
  - Train an autoencoder to learn features of the data.
  - Add a classifier layer, and retrain the whole thing supervised.
  - Good when label set is poor.
- Unsupervised pretraining often uses regularized autoencoders.

From: *http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013*

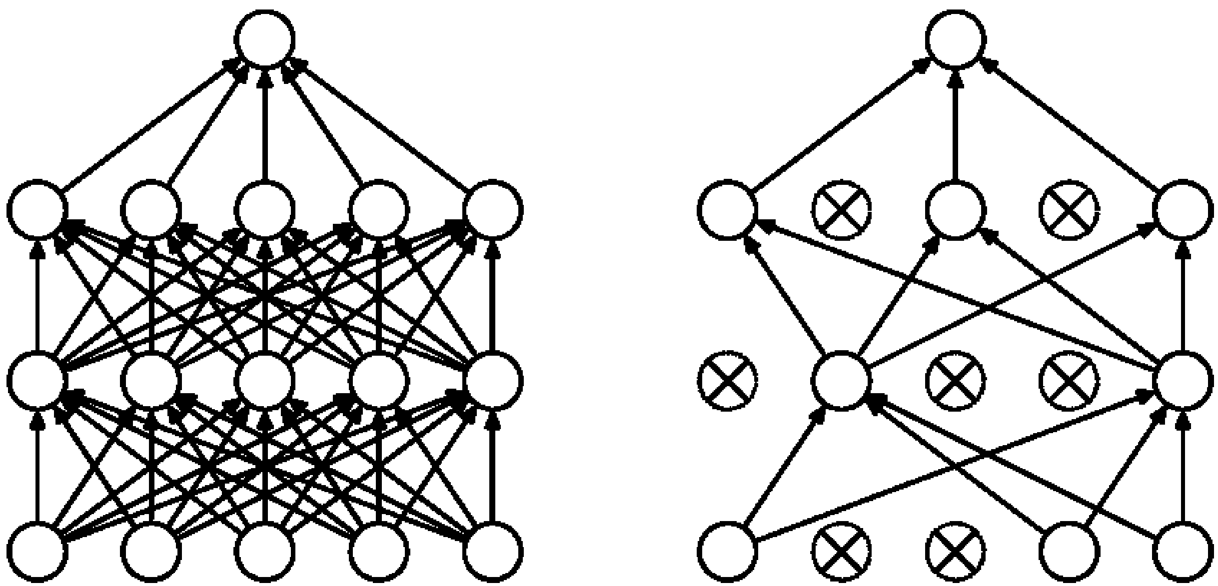
Problem #1: feature co-adaptation

- In neural networks, derivative received by each parameter tells it what to do, *given what the other parameters are doing*
- This could lead to some neurons ‘fixing’ the problems caused by other neurons -> ***co-adaptation***
- While this is okay on the training set, these fixes often don’t generalize to the test set

- *“Dropout: a simple way to prevent neural networks from overfitting,” Srivastava et al., 2014*

Dropout

- **Independently set each hidden unit activity to zero with probability *p*** (usually *p*=0.5 works best).
- Neurons are forced to work with random subset of neighbours



(a) Standard Neural Net (b) After applying dropout.

Problem #2: internal covariate shift

- During training, each layer of a neural network gets ‘used to’ the distribution of its inputs from the lower layer
- But the *distribution of outputs at each layer changes over time* as the network trains!
  - Each layer has to keep re-adapting to the new distribution
  - This problem is called ***internal covariate shift***
- This can slow down and destabilize learning

- *“Batch normalization: Accelerating deep network training by reducing internal covariate shift,” Ioffe & Szegedy, 2015.*

Batch normalization

- Idea: Feature scaling makes gradient descent easier.
  - We already apply this at the input layer; extend to other layers.
  - Use empirical batch statistics to choose re-scaling parameters.

- For each mini-batch of data, at each layer *k* of the network:
  - Compute empirical mean and var independently for each dimension
  - Normalize each input: $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{VAR[x^{(k)}]}}$
  - Output has tunable parameters (γ,β) for each layer: $y^k = \gamma^k . \hat{x}^{(k)} + \beta^k$

- Effect: More stable gradient estimates, especially for deep networks.

Batch normalization

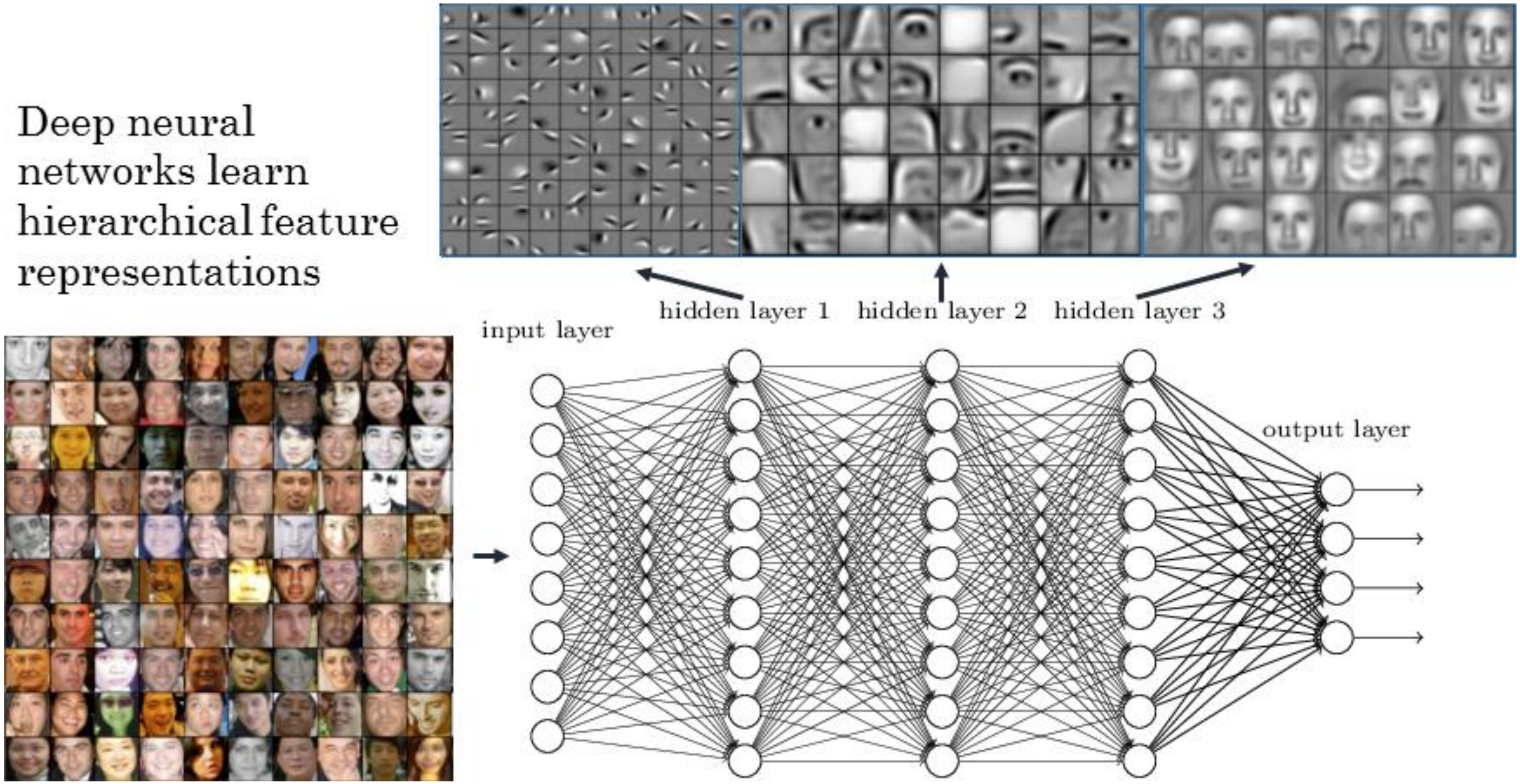
- Many other kinds of normalization: *e.g. weight normalization, layer normalization, batch re-normalization, etc.*
- Dropout and batch normalization empirically act as ***regularizers***
  - Usually don't need to use an L2 penalty on the weights
- Can use both, but *batch normalization alone works extremely well*

Do we really need deep architectures?

- We can approximate any function with a one-hidden-layer neural network. Why go deeper?
- **Deep networks are more efficient for representing certain classes of functions, with certain types of structure.**
  - Natural signals (images, speech) typically have such structure.
- Deep architectures can represent more complex functions with *fewer parameters*.
- So far, very little theoretical analysis of deep learning.

Do we really need deep architectures?

Deep neural networks learn hierarchical feature representations

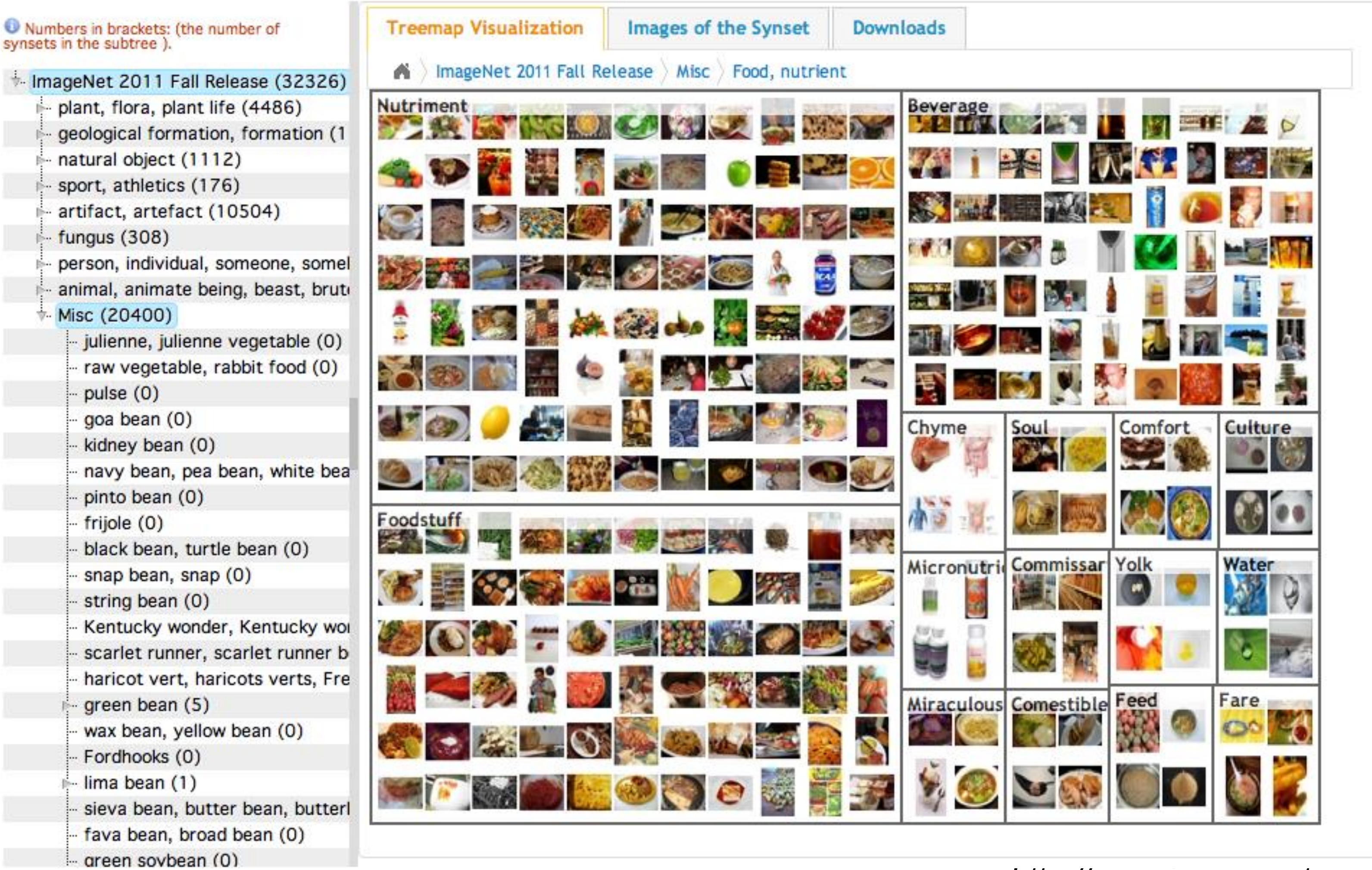


Major paradigms for deep learning

- **Deep neural networks**
  - Supervised training: Feed-forward neural networks.
  - Unsupervised pre-training: Autoencoders.

- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets. (*Next class.)*

ImageNet dataset



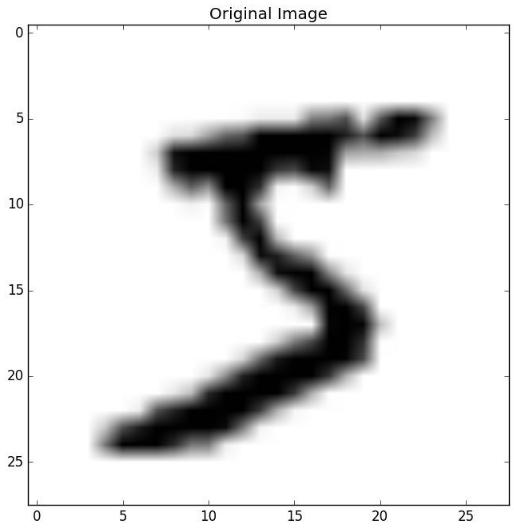
http://www.image-net.org

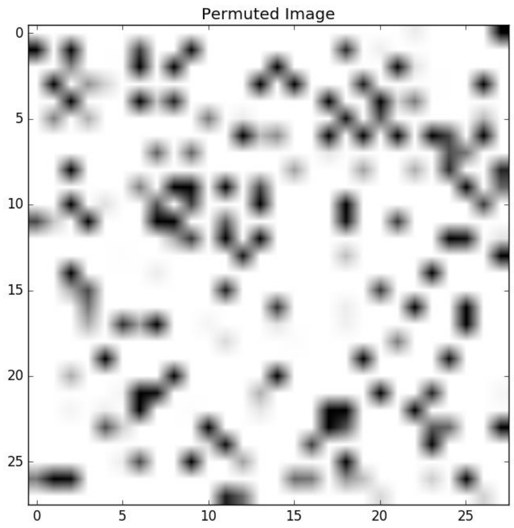
Neural networks for computer vision

- Design neural networks that are specifically adapted to:
  - Deal with very high-dimensional inputs
    - E.g. 150x150 pixels = 22,500 inputs, or 3x22,500 if RGB
  - Exploit 2D topology of pixels (or 3D for video)
  - Built-in invariance to certain variations we can expect
    - Translations, illumination, etc.

Why not feed-forward networks?

- Don't take into account the structure of the data!
- Since input neurons have no ordering, an **image looks the same as a shuffled image**



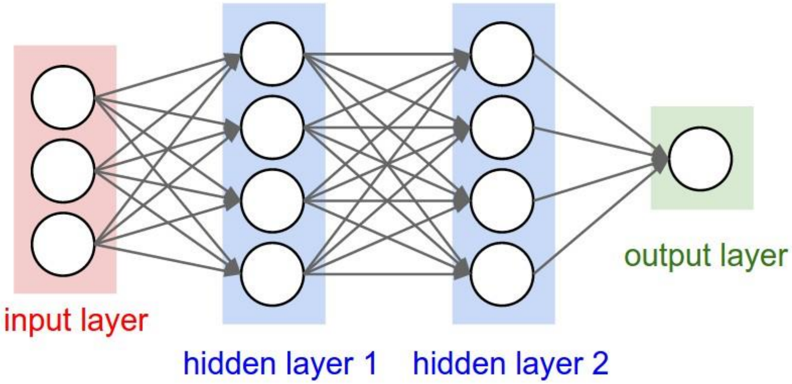


These look the same to a feed-forward network!

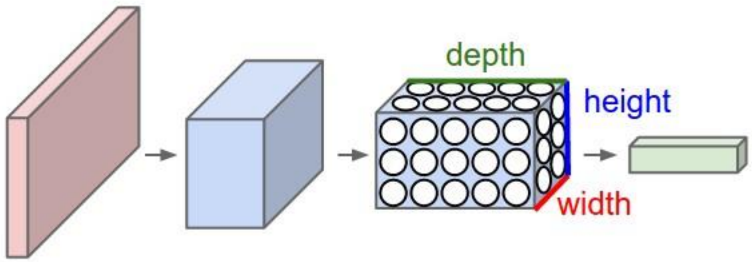
(so long as the same shuffling is applied to all of the data)

Convolutional Neural Networks

<u>Feedforward network</u>



<u>Convolutional neural network (CNN)</u>

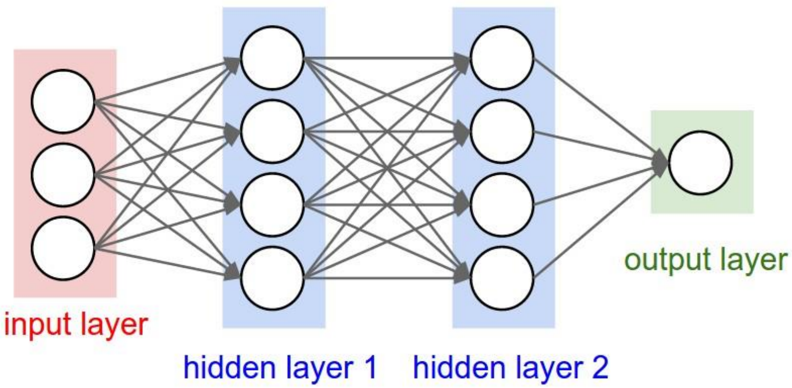


- CNN characteristics:
  - Input is a 3D tensor: 2D image x 3 colours (or 2D if grayscale)
  - Each layer transforms an input 3D tensor to an output 3D tensor using a differentiable function.

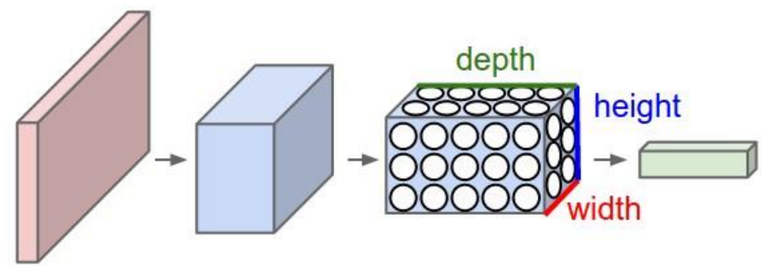
From: *http://cs231n.github.io/convolutional-networks/*

Convolutional Neural Networks

Feedforward network



Convolutional neural network (CNN)

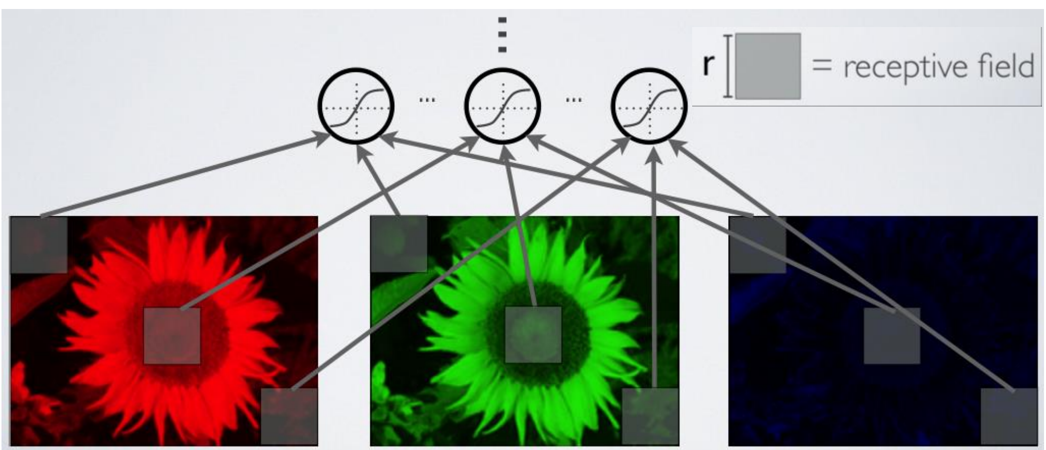


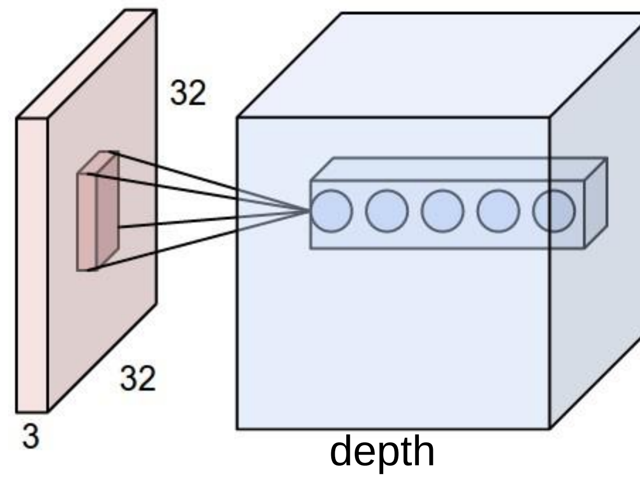
- Convolutional neural networks leverage several ideas.
  1. Local connectivity.
  2. Parameter sharing.
  3. Pooling hidden units.

From: *http://cs231n.github.io/convolutional-networks/*

Convolutional Neural Networks

- A few key ideas:
  1. Features have **local receptive fields.**
     - Each hidden unit is connected to a patch of the input image.
     - Units are connected to all 3 colour channels.

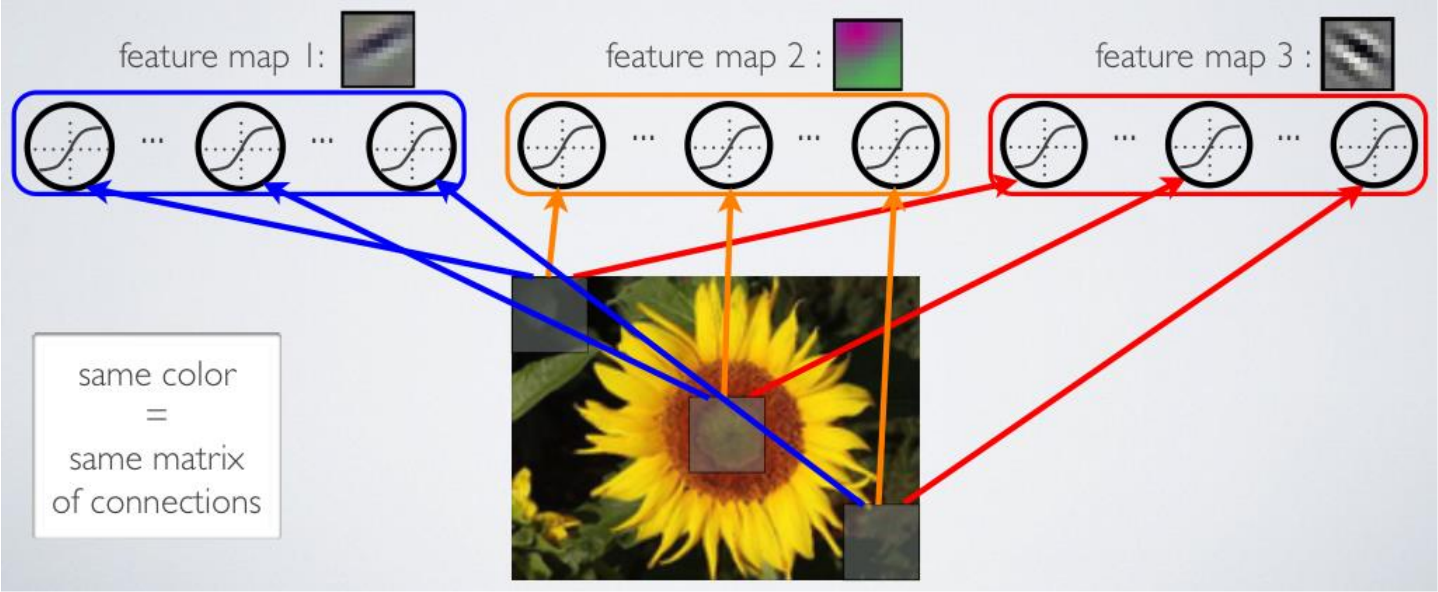




depth = # filters
(a hyperparameter)

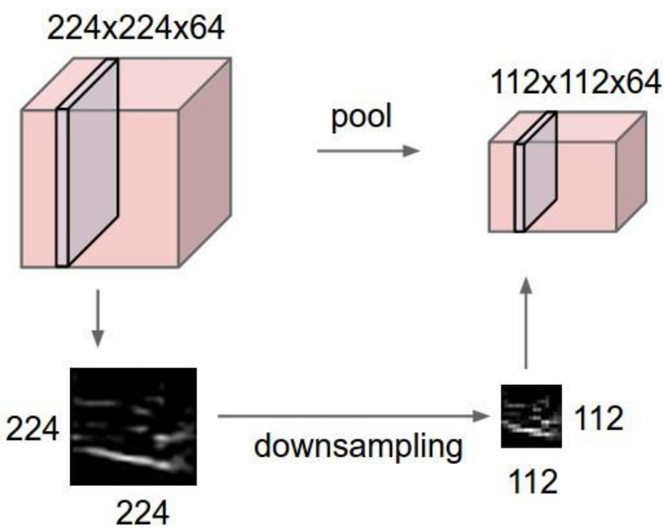
Convolutional Neural Networks

- A few key ideas:
  1. Features have **local receptive fields.**
  2. **Share matrix of parameters** across units.
     - Constrain units within a depth slice (at all positions) to have **same** weights.
     - Feature map can be computed via discrete convolution with a kernel matrix.

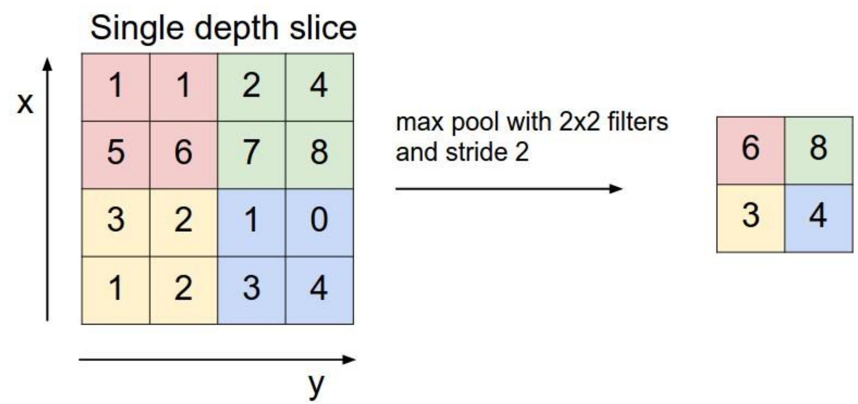


Convolutional Neural Networks

- A few key ideas:
    1. Features have **local receptive fields.**
    2. **Share matrix of parameters** across units.
    3. **Pooling/subsampling** of hidden units in same neighbourhood.



Example:



From: *http://cs231n.github.io/convolutional-networks/*

Convolutional Neural Networks

- Local receptive fields
  - **Intuition:** there are some data features (e.g. edges, corners) that ***only depend on a small region of the image***
- Parameter sharing
  - **Intuition:** processing these local features should be done ***the same way*** regardless of where the feature is in the image
  - ***Much more efficient to train***
- Pooling/ subsampling
  - **Intuition:** usually doesn't matter where *exactly* a feature occurs, only that it occurs somewhere
  - As we go deeper in the network, want to consider features that cover more area (i.e. ***more global features***)

Convolution

- What is a convolution?
- Formula: $(x * w)(t) = \sum_a x(a)w(t - a)$
- x is the ***input data***, w is the ***kernel***
- The kernel is a function of learned parameters repeatedly applied to various parts of the input

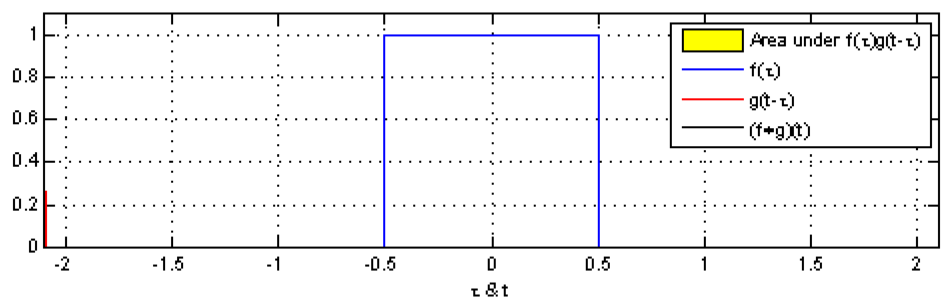


Image: wikipedia.org

Convolution

- *w, x, y, z* are **learned parameters**
- Can have multiple kernels in a layer

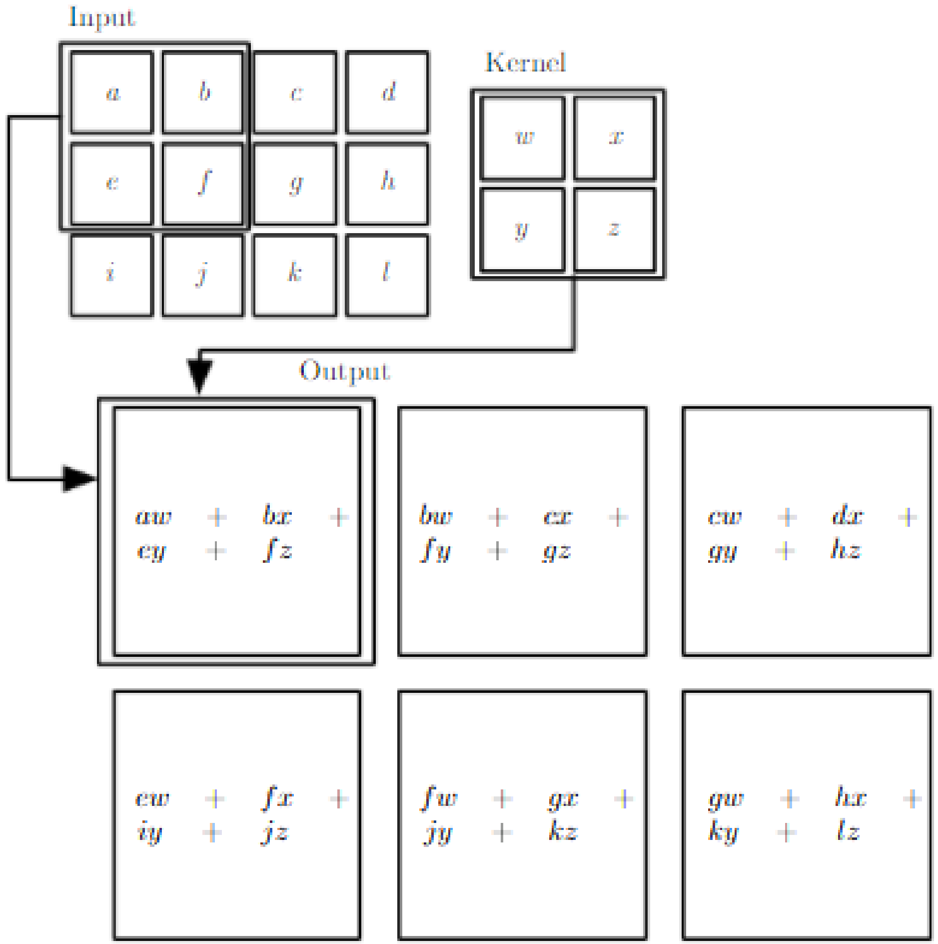
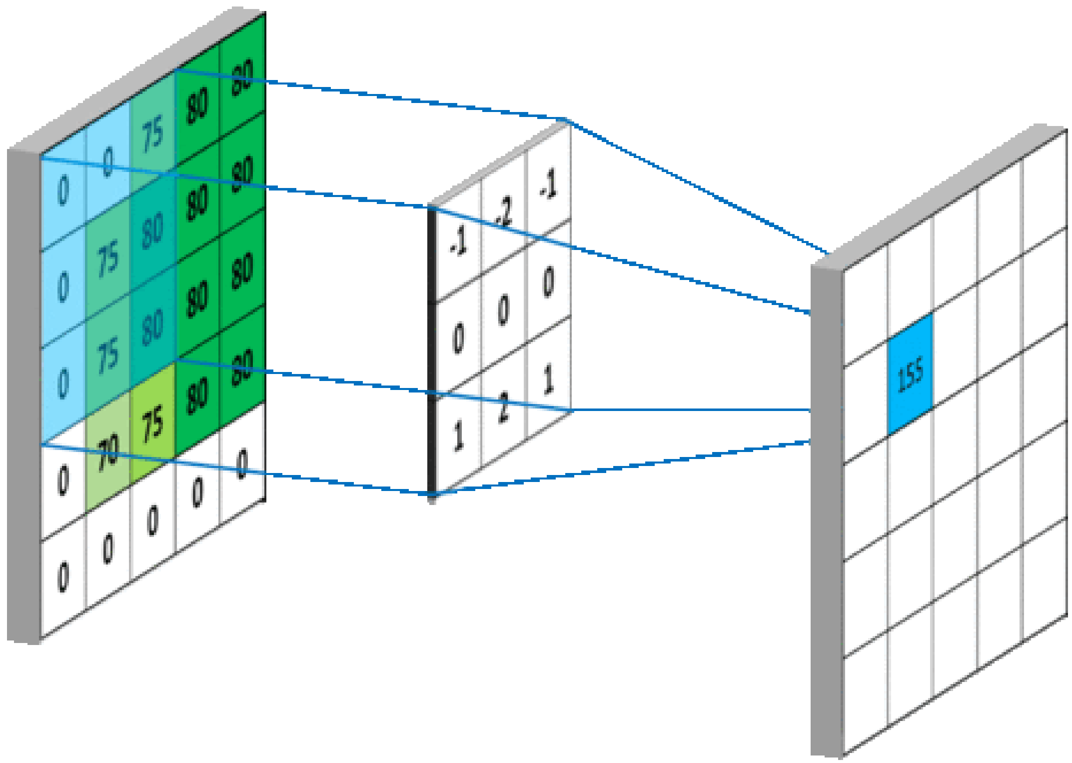


Image: deeplearningbook.org

Convolution



Input Kernel Output (feature map)

Convolution

- Averaging in a 3x3 box blurs the image

0	0	0	0	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	0	0	0	0

- Can be used for edge detection

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

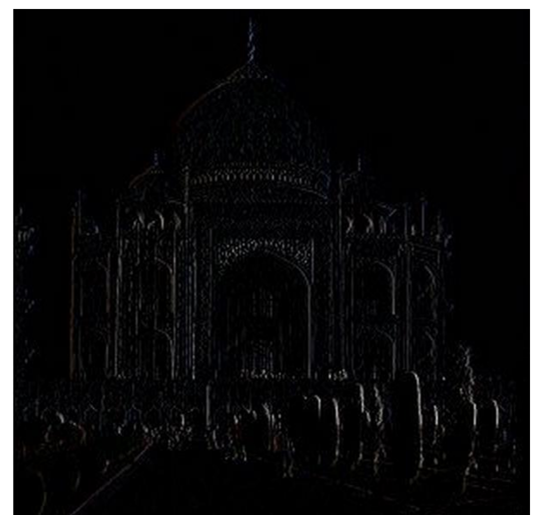
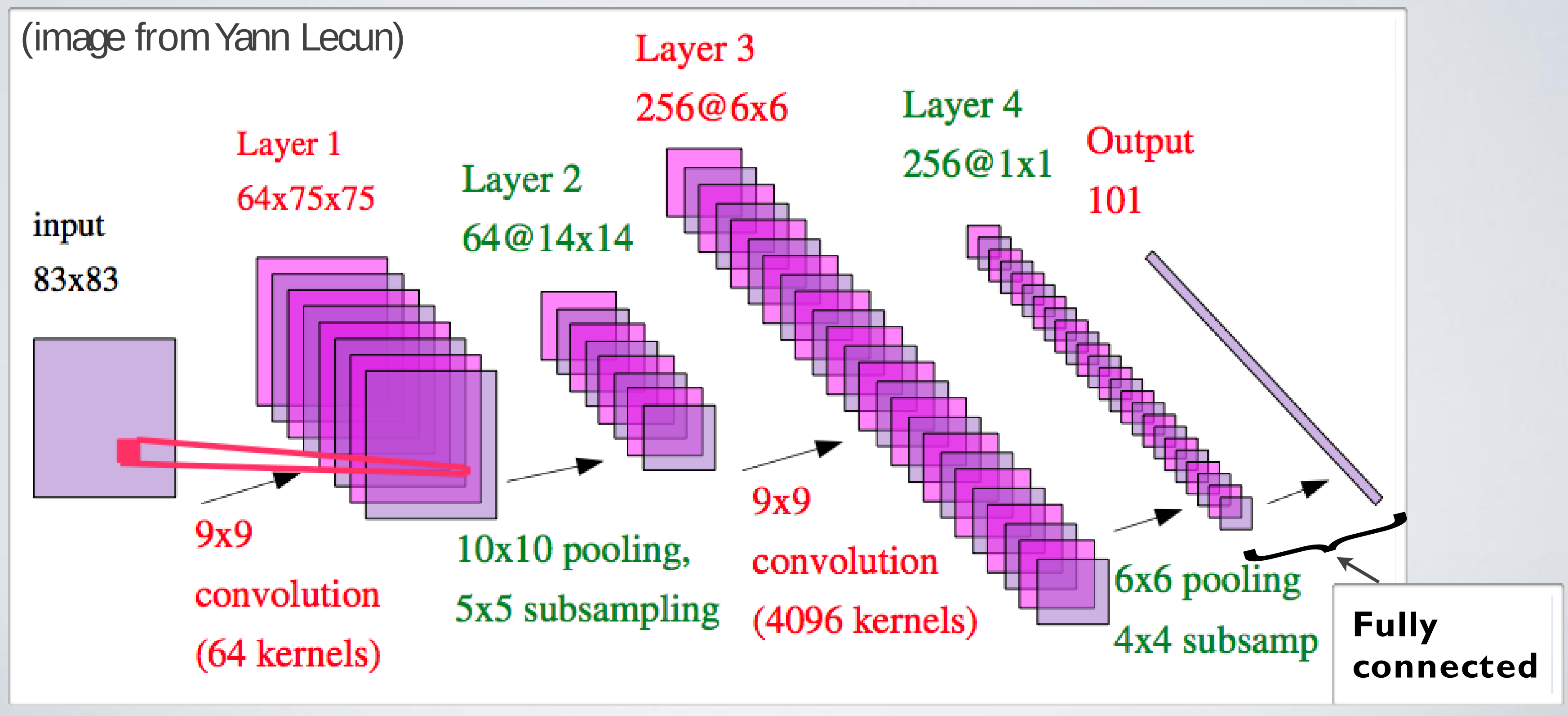


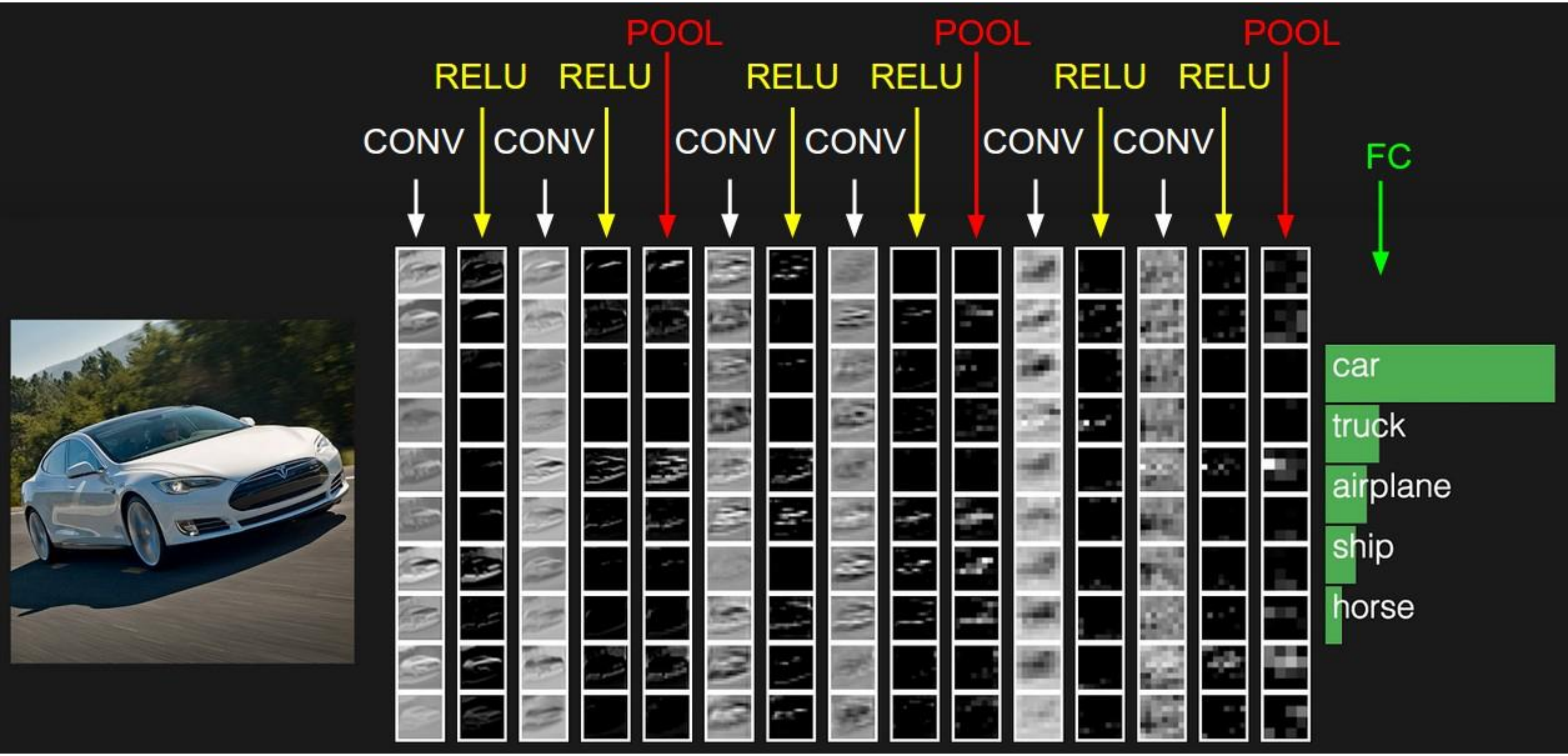
Image: Gimp documentation

Convolutional neural nets (CNNs)

- Alternate between **convolutional**, **pooling,** **and** **fully connected** layers.
  - Fully connected layer typically only at the end.
- Train full network using **backpropagation**.



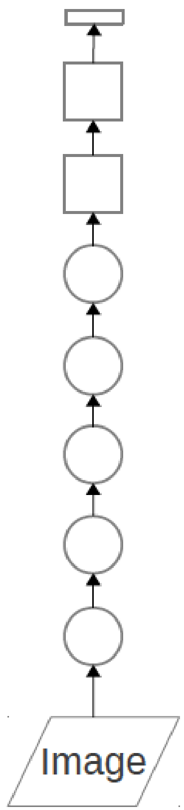
Convolutional neural nets (CNNs)



From: *http://cs231n.github.io/convolutional-networks/*

Example: ImageNet

- SuperVision (a.k.a. AlexNet, 2012):



- **Deep**: 7 hidden "weight" layers
- **Learned**: all feature extractors initialized at white Gaussian noise and learned from the data
- Entirely supervised
- **More data = good**

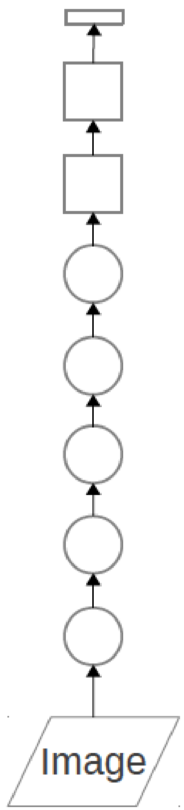
○ **Convolutional layer:** convolves its input with a bank of 3D filters, then applies point-wise non-linearity

□ **Fully-connected layer:** applies linear filters to its input, then applies point-wise non-linearity

From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Example: ImageNet

- SuperVision (a.k.a. AlexNet, 2012):



- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

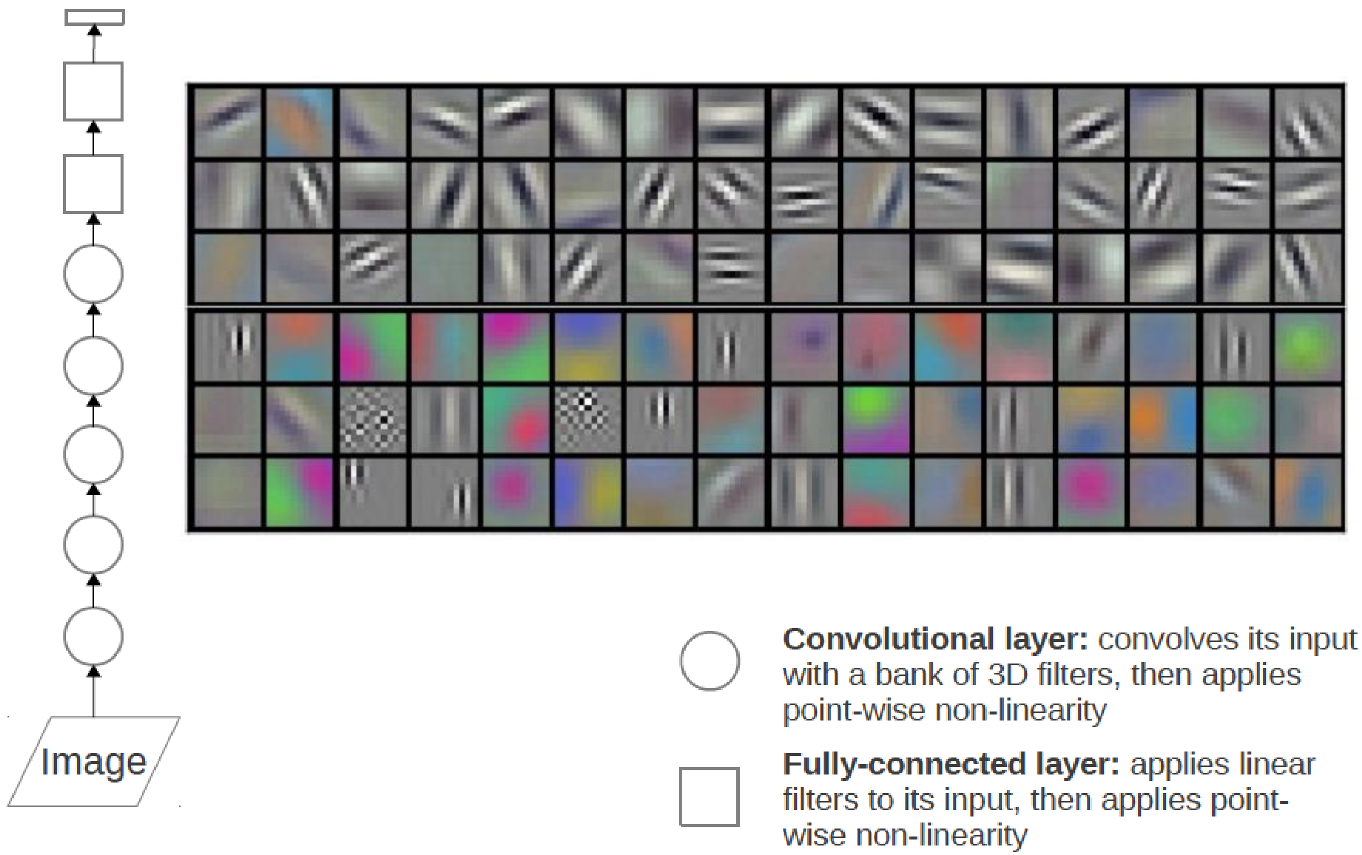
Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity

Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Training results: ImageNet

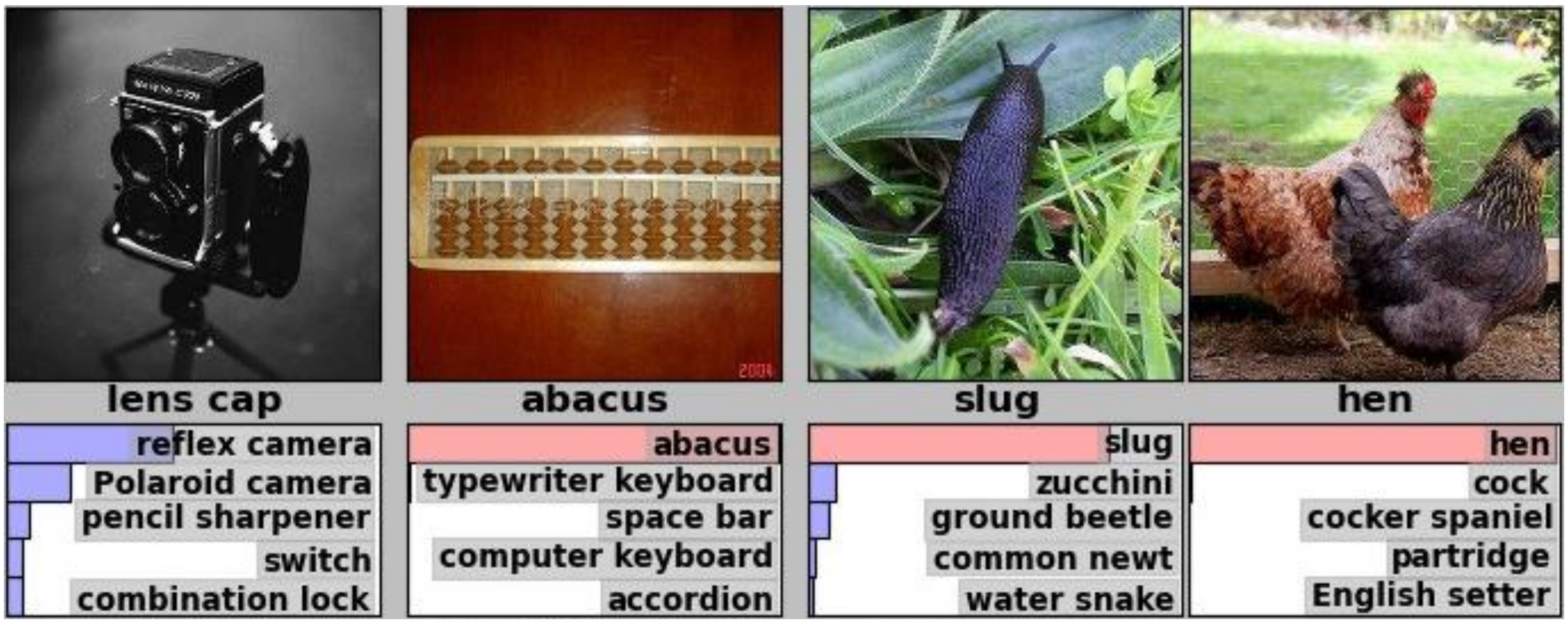
- 96 learned low-level filters



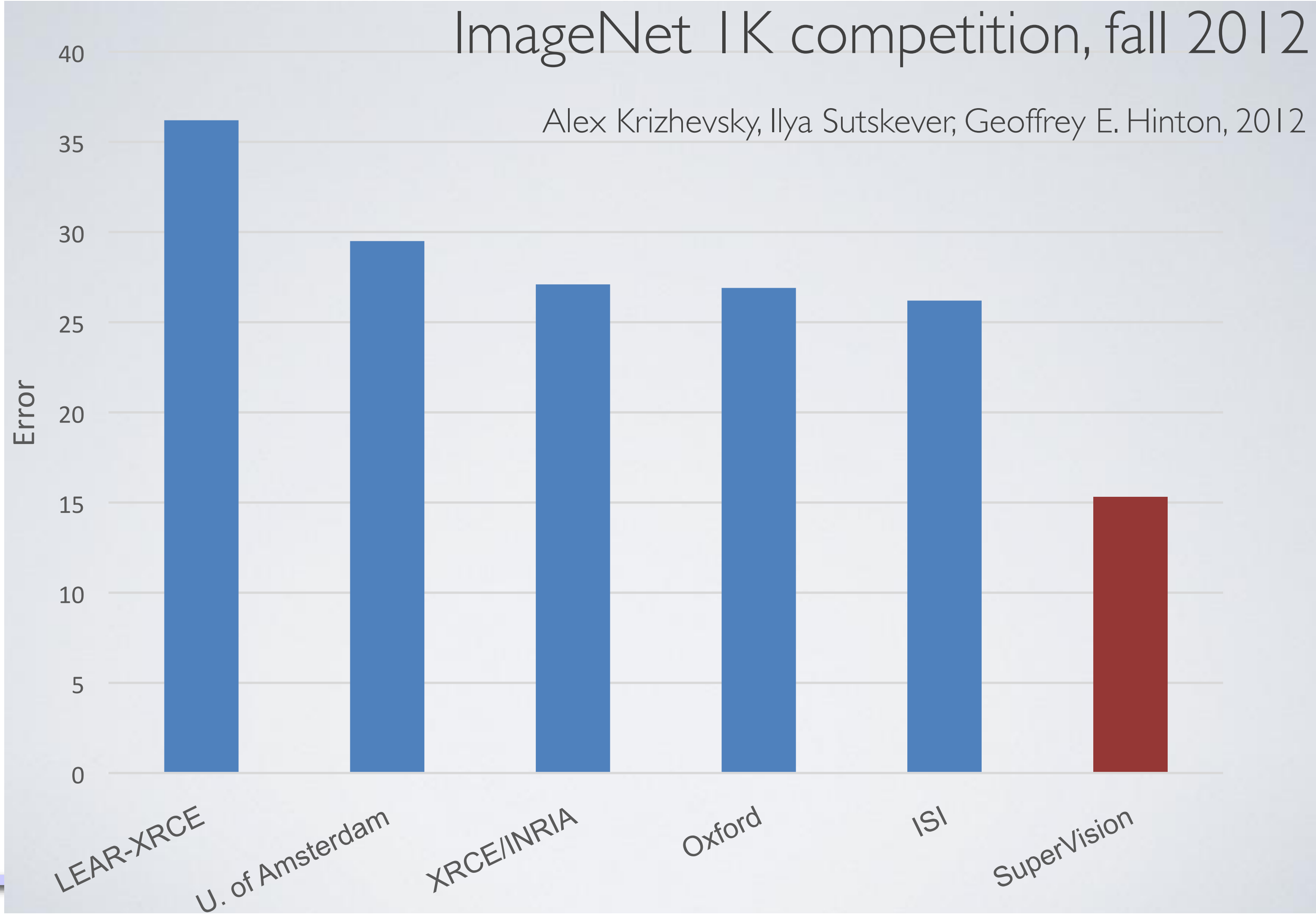
From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

Image classification

- 95% accuracy (on top 5 predictions) among 1,000 categories. Better than average human.

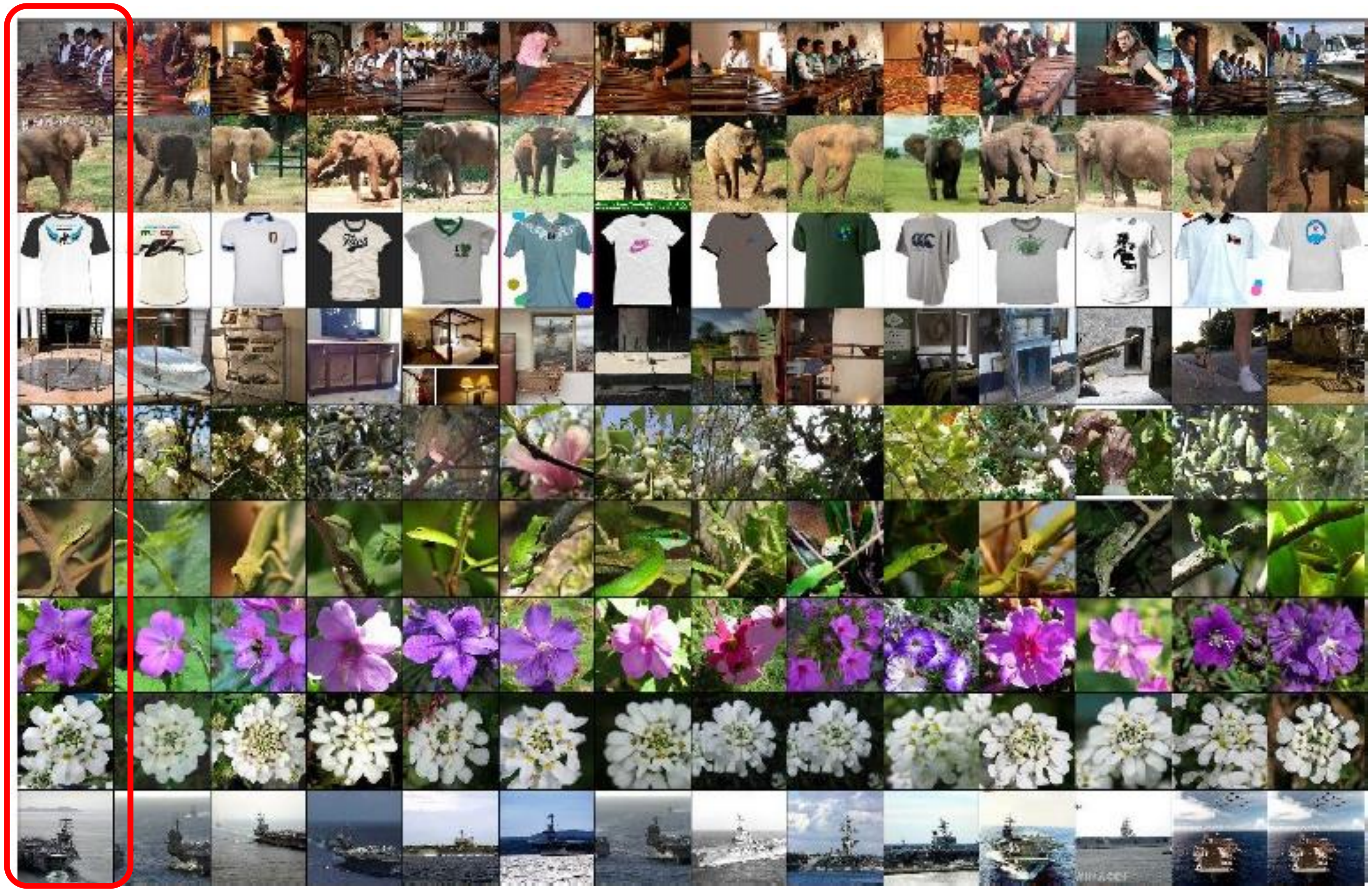


Empirical results (2012)



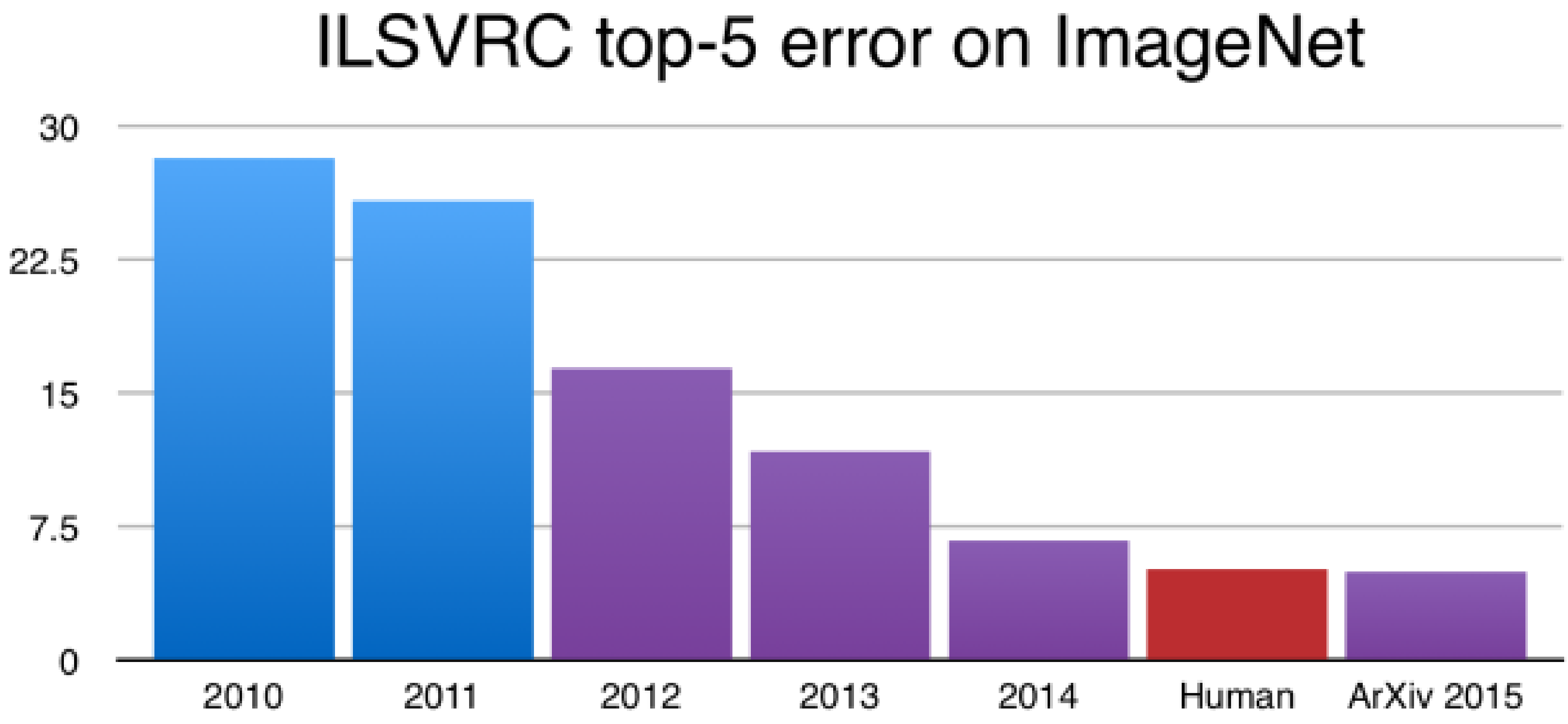
Empirical results for image retrieval

- Query items in leftmost column:



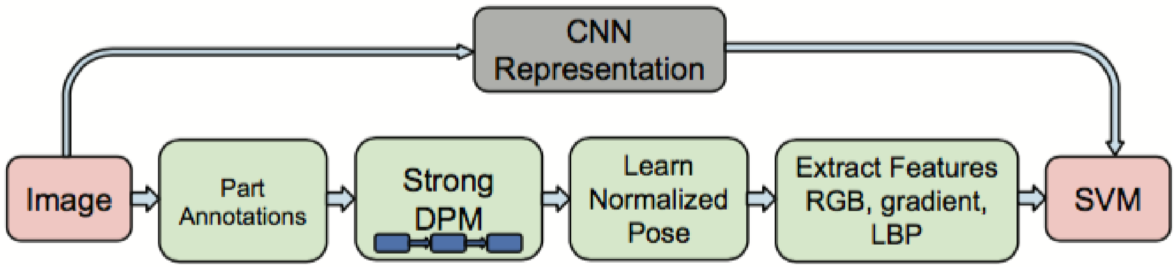
From: *http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf*

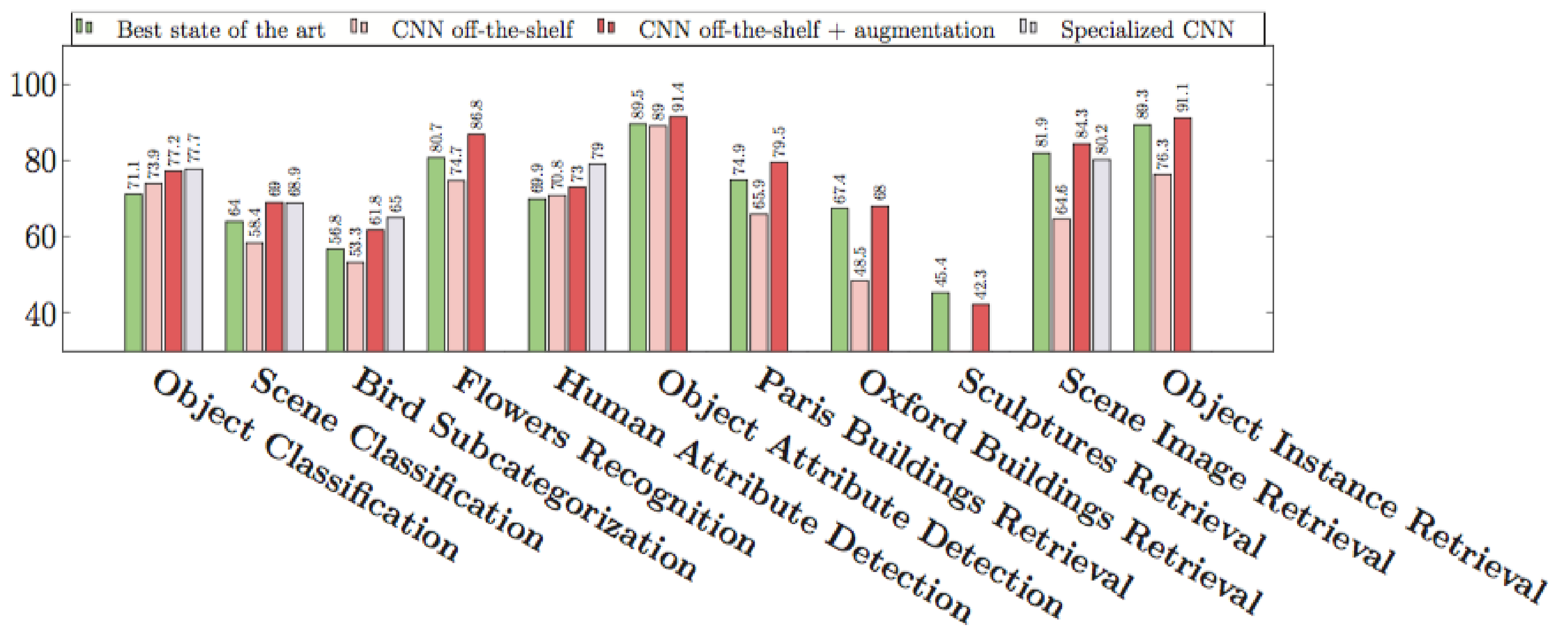
Empirical results (2015)



http://devblogs.nvidia.com/parallelforall/mocha-jl-deep-learning-julia/

CNNs vs traditional computer vision





From: *Razavian et al. CVPR workshop paper. 2014.*

Picture tagging (From *clarifai.com*)

Predicted Tags:

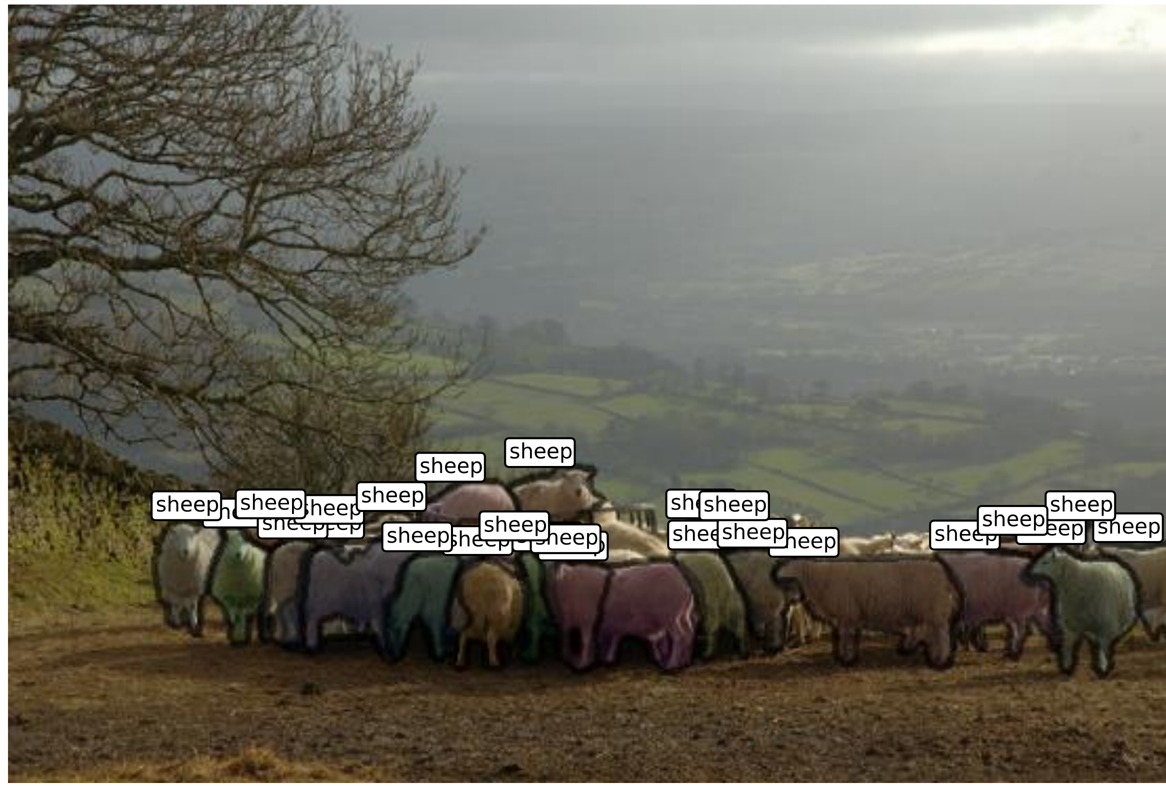
food	(16.00%)
dinner	(3.10%)
bbq	(2.90%)
market	(2.50%)
meal	(1.40%)
turkey	(1.40%)
grill	(1.30%)
pizza	(1.30%)
eat	(1.10%)
holiday	(1.00%)

Stats:

Size: 247.24 KB

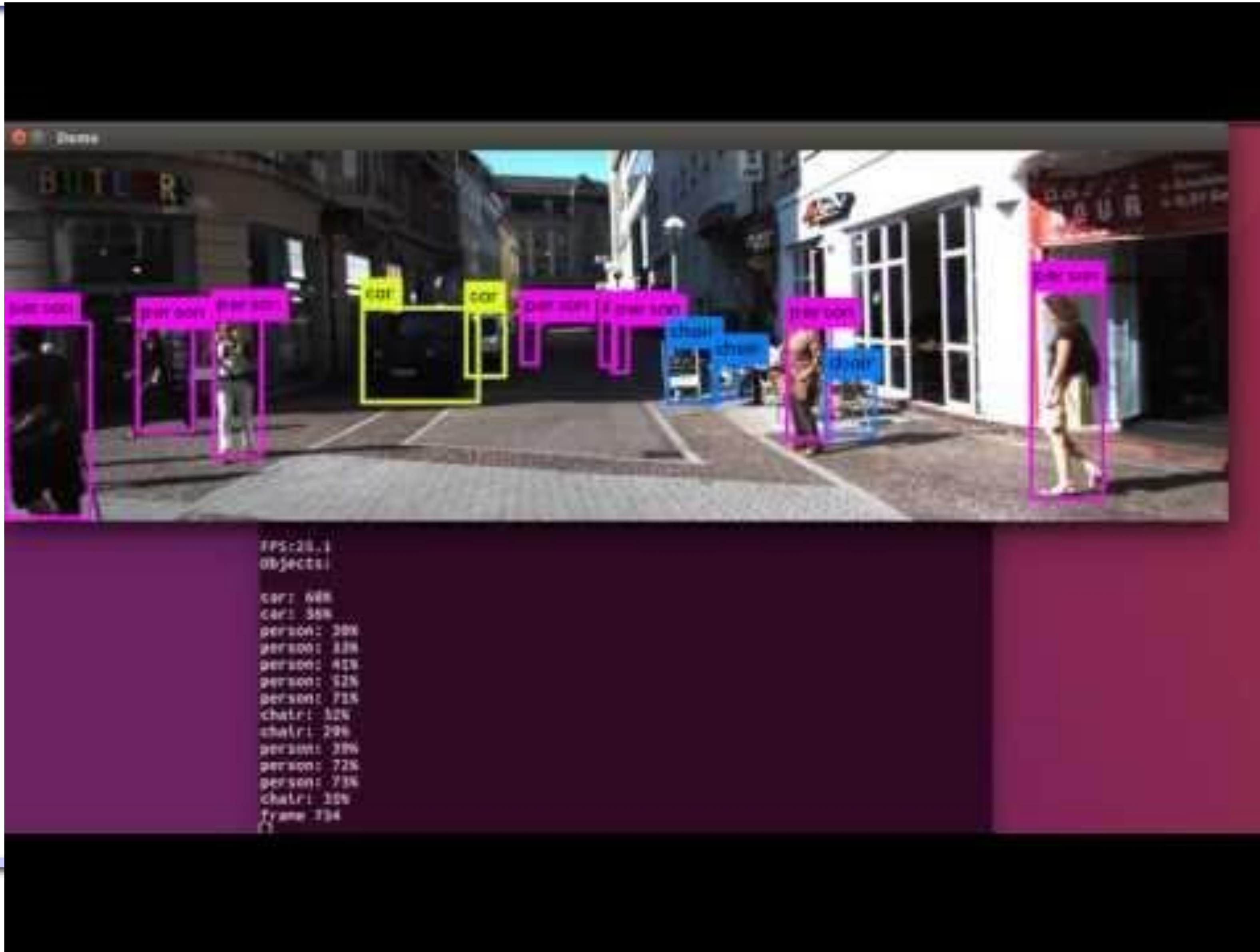
Time: 110 ms

Scene parsing



(Farabet et al., 2013)

YOLO: Real-time object detection



Practical tips for CNNs

- Many hyper-parameters to choose!
- Architecture: filters (start small, e.g. 3x3, 5x5), pooling, number of layers (start small, add more).
- Training: learning rate, regularization, dropout rate (=0.5), initial weight size, batch size, batch norm.
- **Read papers, copy their method, then do local search.**

What you should know

- Types of deep learning architectures:
  - Autoencoders
  - Convolutional neural networks
  - Tricks to get neural networks to work
- Typical training approaches (unsupervised / supervised).
- Examples of successful applications.

More resources

- Deep learning textbook
  - In-depth treatment of all deep learning fundamentals
  - Available online for free: http://www.deeplearningbook.org/
- All articles on colah.github.io (highly recommended)
  - Well-explained articles on various neural network topics
  - Two posts on ConvNets: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/, http://colah.github.io/posts/2014-07-Understanding-Convolutions/
- Convolution arithmetic
  - https://arxiv.org/pdf/1603.07285.pdf

More resources

- Notes and images in today's slides taken from:
  - *http://cs231n.github.io/convolutional-networks/*
  - *http://www.cs.toronto.edu/~hinton/csc2535*
  - *http://deeplearning.net/tutorial/*
  - *http://www.slideshare.net/philipzh/a-tutorial-on-deep-learning-at-icml-2013*
  - *http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf*
  - *http://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf*