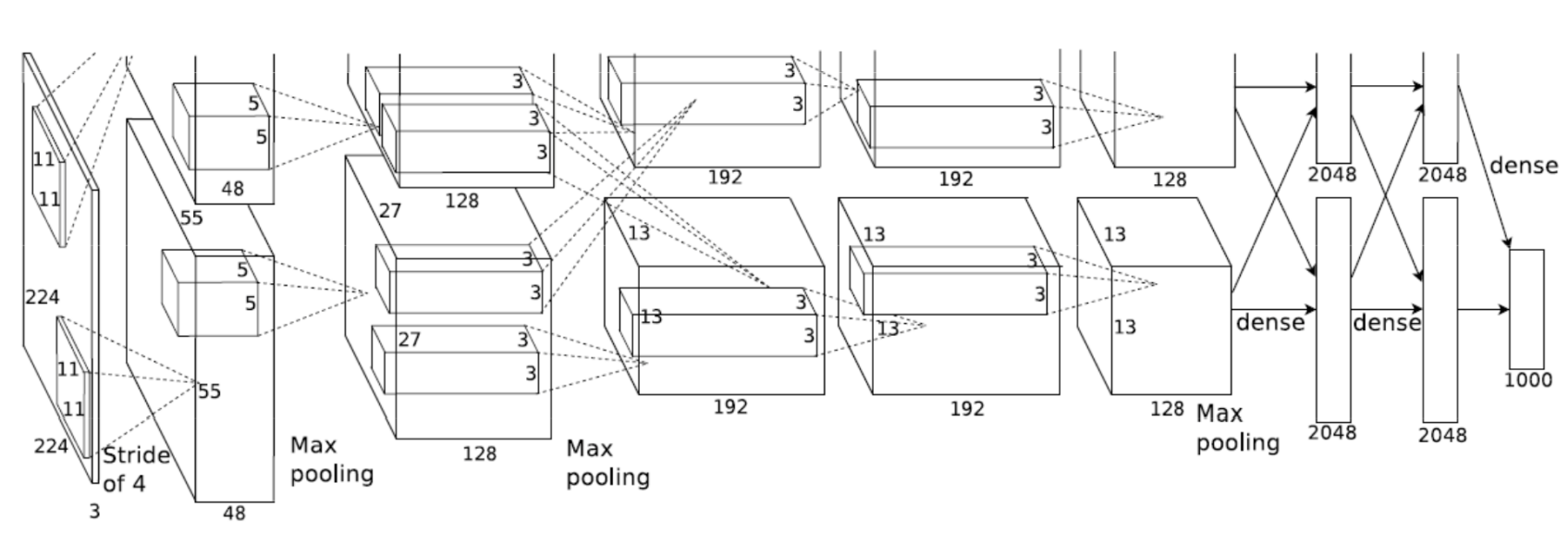
# Deep Neural Networks

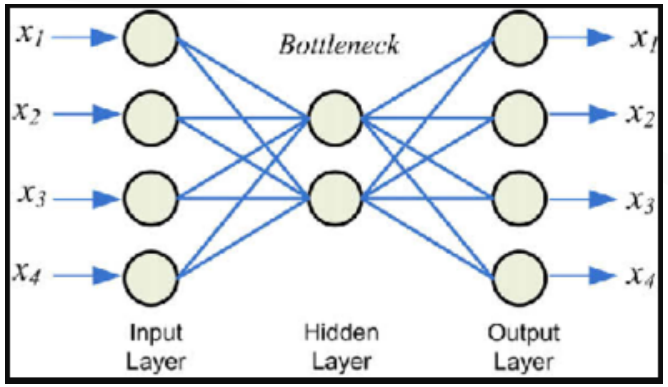


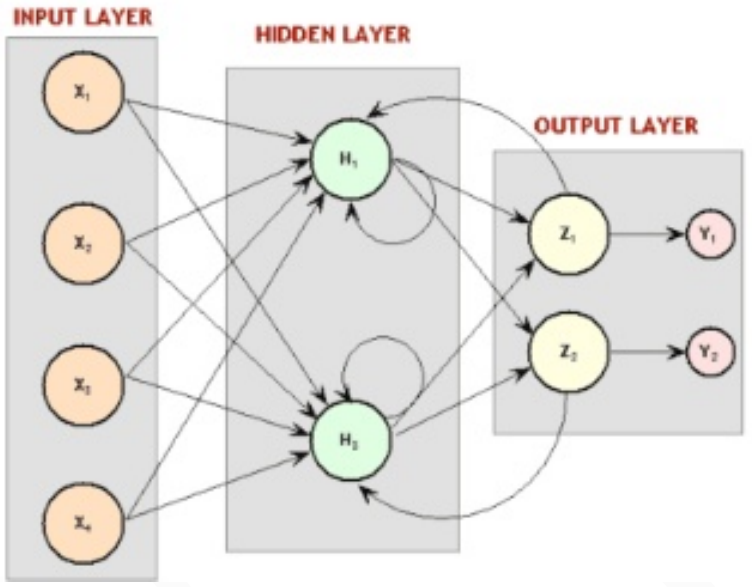

Berrin Yanikoglu
Sabancı University

Images/some text from http://cs231n.github.io/convolutional-networks/

# Deep Learning Approaches

- Supervised learning
  - Convolutional Networks
  - Recurrent Networks (LSTM,...)
- Unsupervised learning
  - AutoEncoders
  - Restricted Boltzmann Machines (RBMs)



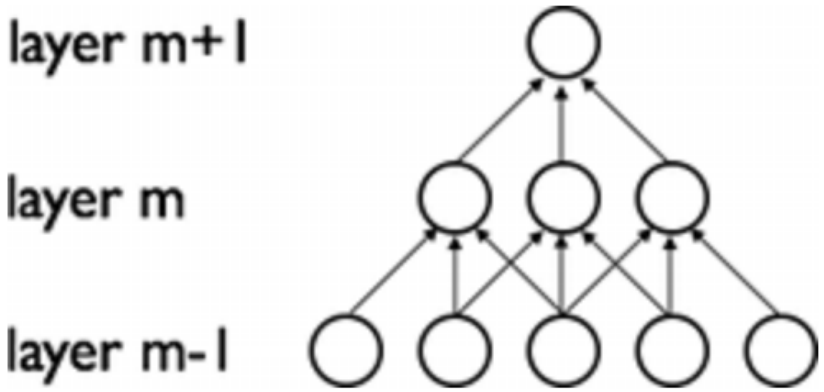
# Convolutional Neural Networks
# CNNs

**Motivation**

The visual cortex contains a complex arrangement of cells (Hubel&Wiesel1968).

These cells are sensitive to small sub-regions of the visual field, called a receptive field.

The sub-regions are tiled to cover the entire visual field.



# Receptive Fields

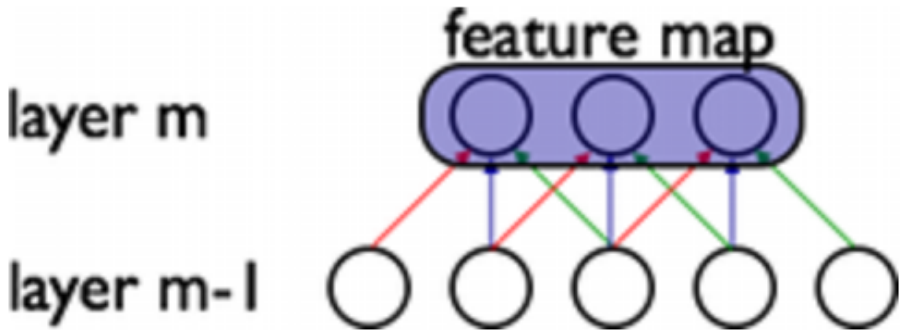
- Local connectivity from a small area
  - small receptive field size
  - spatially contiguous

responds to spatially local input patterns.

- Stacking many such layers leads to (non-linear) “filters” that become increasingly “global” (i.e. responsive to a larger region of pixel space).

# Shared Weights

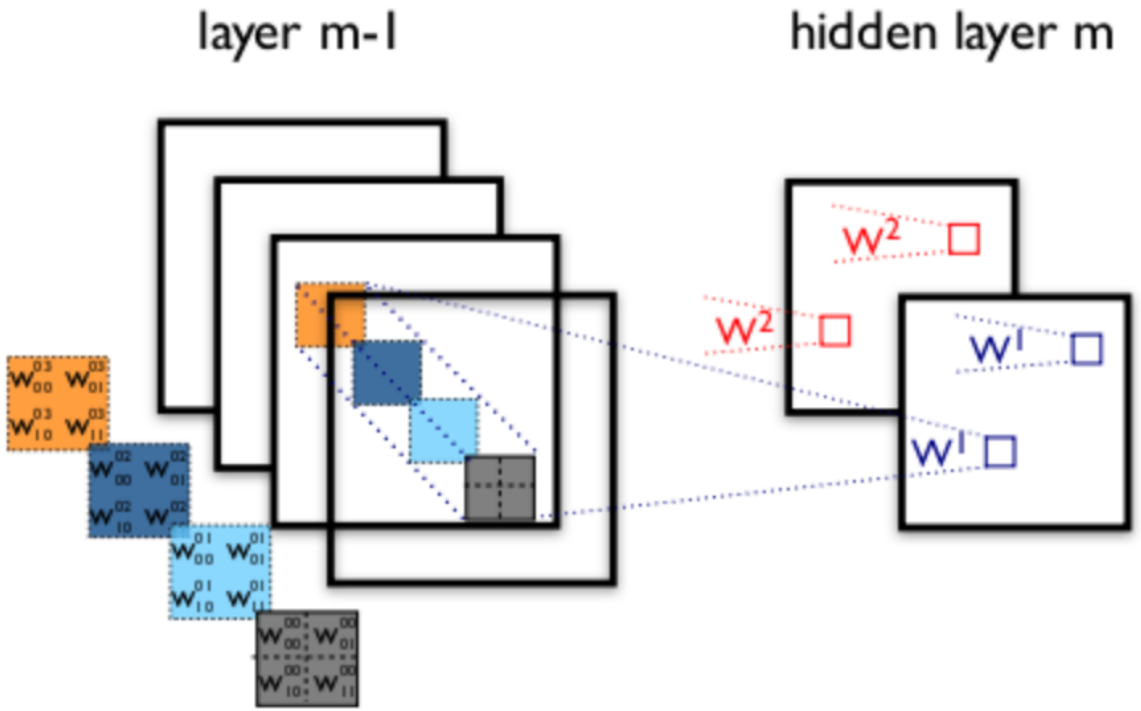
- Each filter is replicated across the entire visual field.
- These replicated units share the same weights and form a feature map.
  - Replicating units in this way allows for features to be detected regardless of their position in the visual field.
  - Additionally, weight sharing greatly reduces the number of free parameters being learnt.




Note: The gradient of a shared weight is simply the sum of the gradients of the weights being shared.

# Example

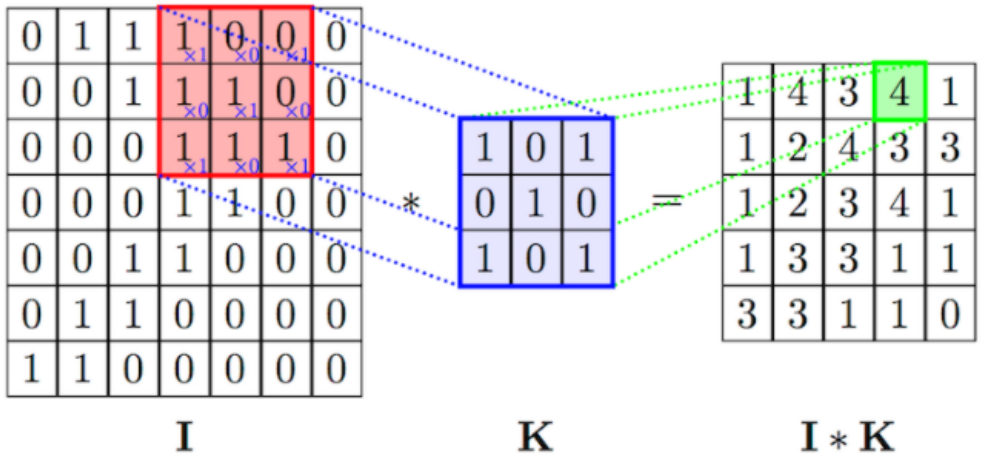
- There are 4 and 2 feature maps in layers m-1 and m, respectively.
- The receptive field of nodes in layer m spans all four input feature maps.
- The weights are 3D weight tensors.
    - The leading dimension indexes the input feature maps, while the other two refer to the pixel coordinates.



# Convolution

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

Dot product between pixels in the receptive field and the weights



# Stride, Zero-Padding

Stride: How many pixels the filters are shifted spatially. Typically 1 or 2

Zero-padding: How many zero pixels are padded to the image to have a good/matching size of output filters.

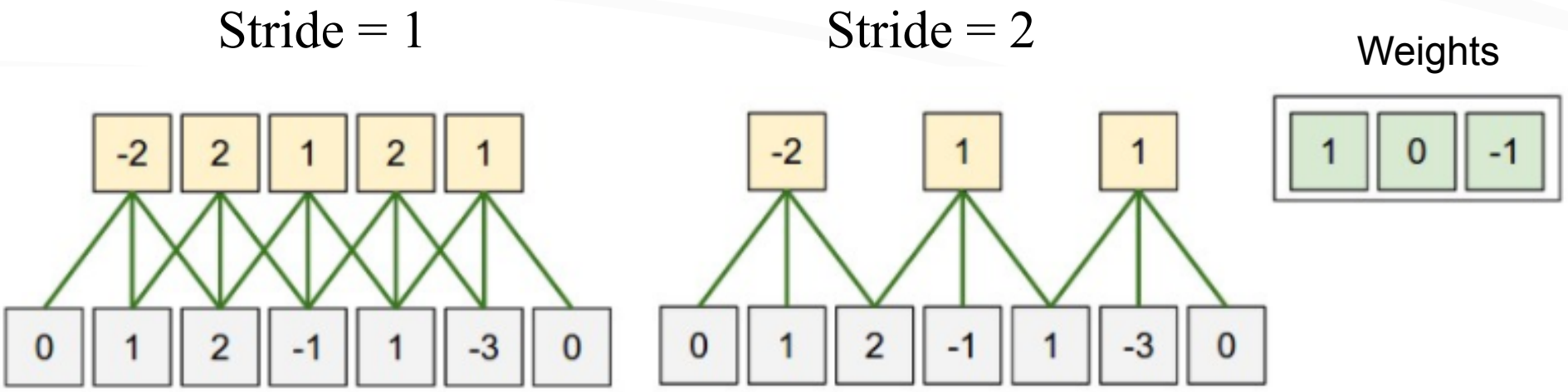
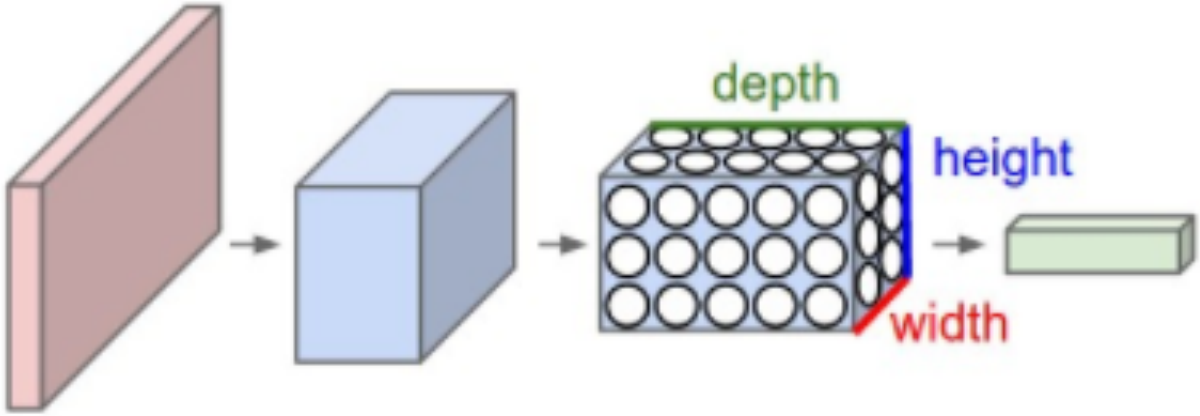



Figure from http://cs231n.github.io/convolutional-networks/

# Real-world example




- AlexNet: The Krizhevsky et al. architecture that won the ImageNet challenge in 2012 (ILSVRC) accepted images of size [227x227x3].
- On the first Convolutional Layer, it used neurons with receptive field size F=11, stride S=4 and no zero padding. Since (W+2P-F)/S+1 = (227 - 11)/4 + 1 = 55, and since the Conv layer had a depth of K=96, the Conv layer output volume had size [55x55x96].
  - Each of the 55*55*96 neurons in this volume was connected to a region of size [11x11x3] in the input volume.
  - Moreover, all 96 neurons in each depth column are connected to the same [11x11x3] region of the input, but of course with different weights.

# ILSVRC-2010 images

## from AlexNet paper

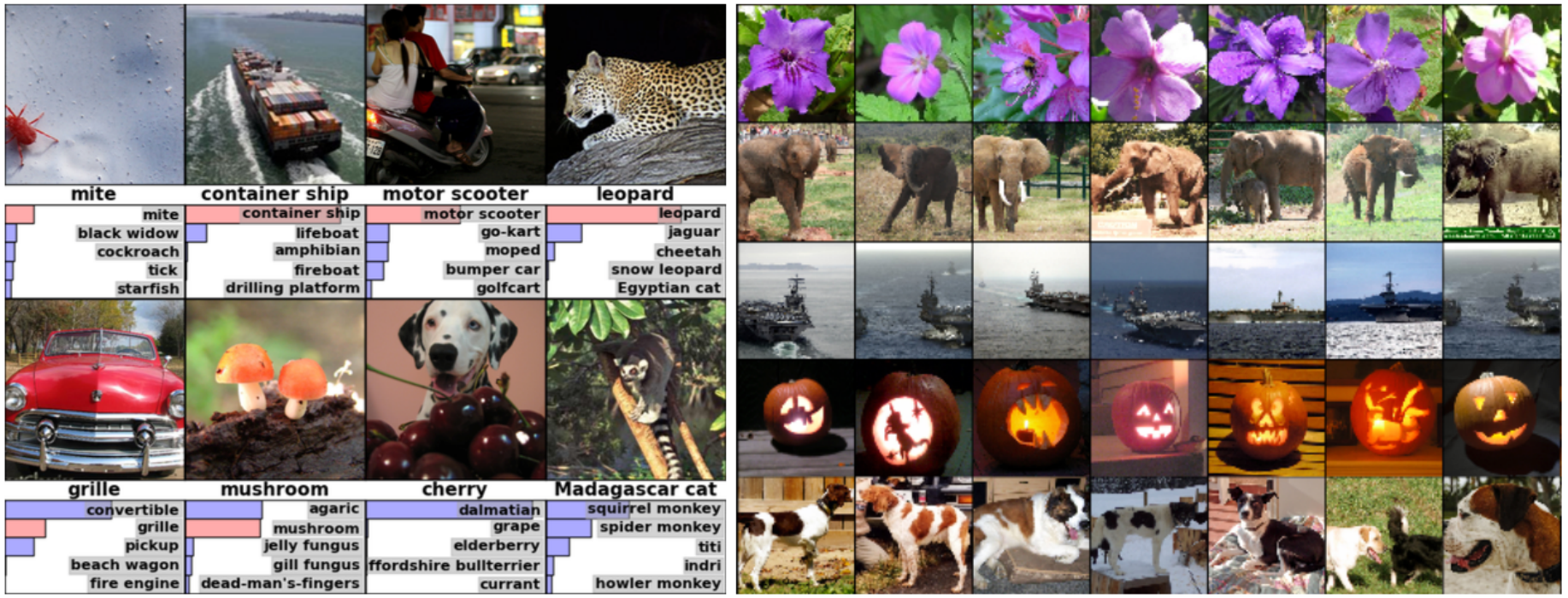



Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

# Weight Sharing

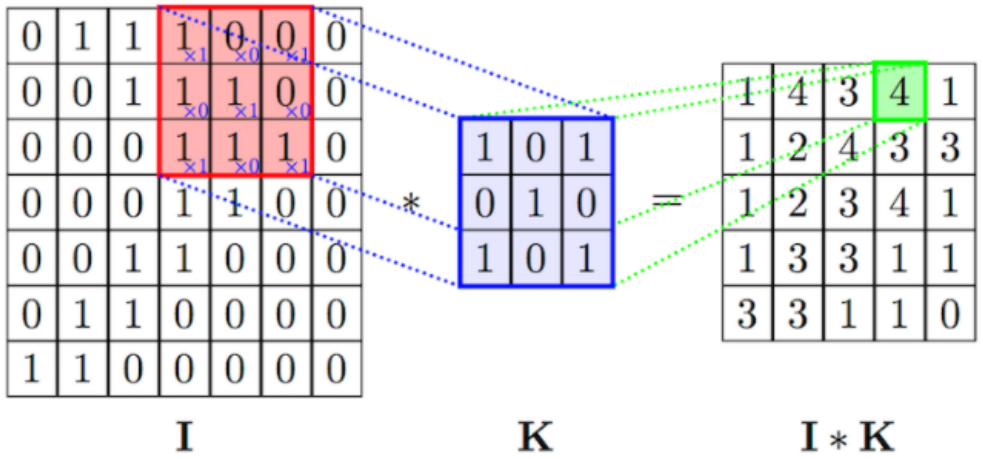
- Using the real world example, we see that there are 55*55*96 = 290,400 neurons in the first Conv Layer.
- Each has 11*11*3 = 363 weights and 1 bias.
- Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone.

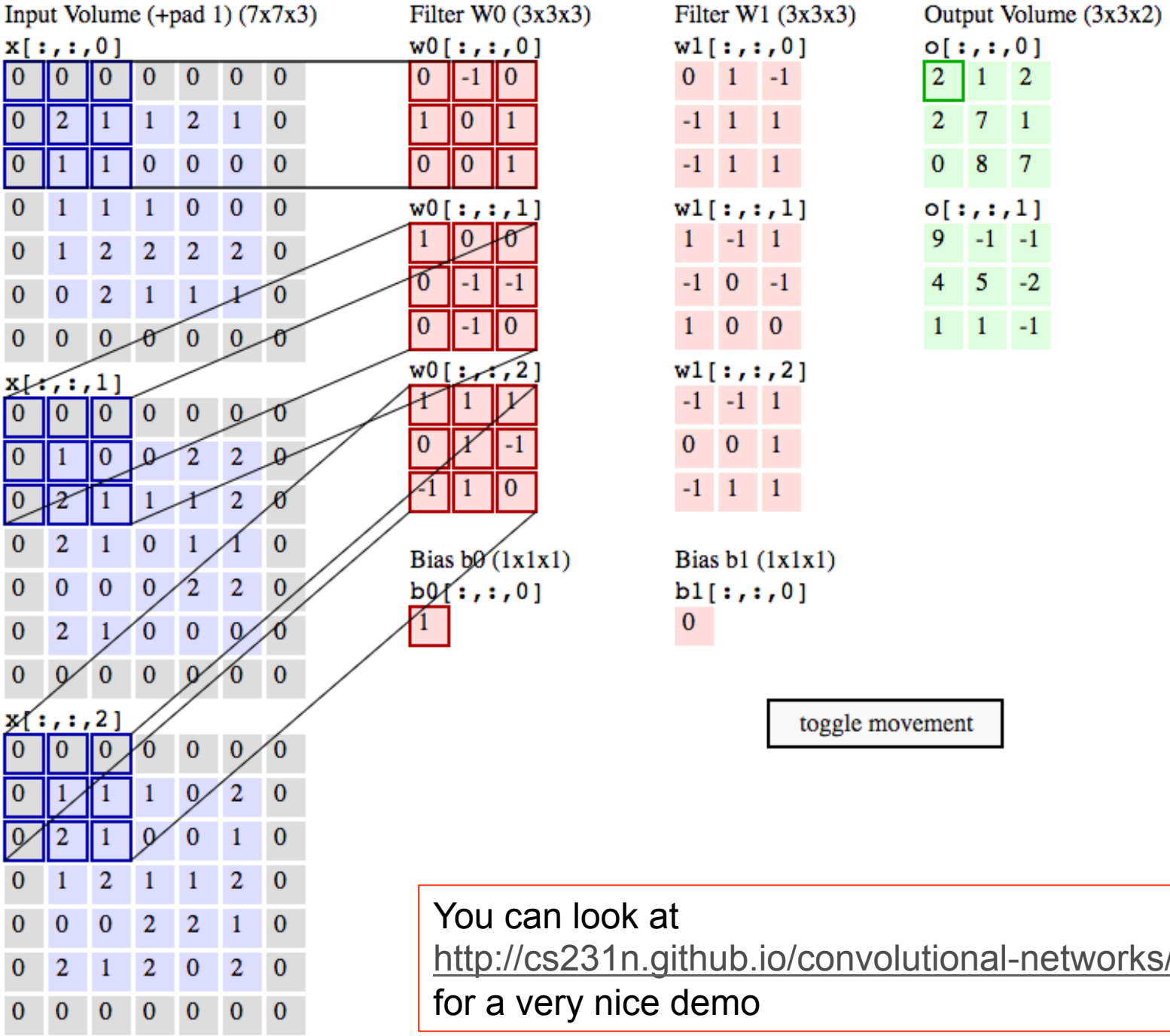
- With weight sharing within a feature map, we have 96 unique **set** of weights, for a total of 96 x11x11x3 = 34,944 (+96 biases).

# Convolution

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1}$$

Dot product between pixels in the receptive field and the weights

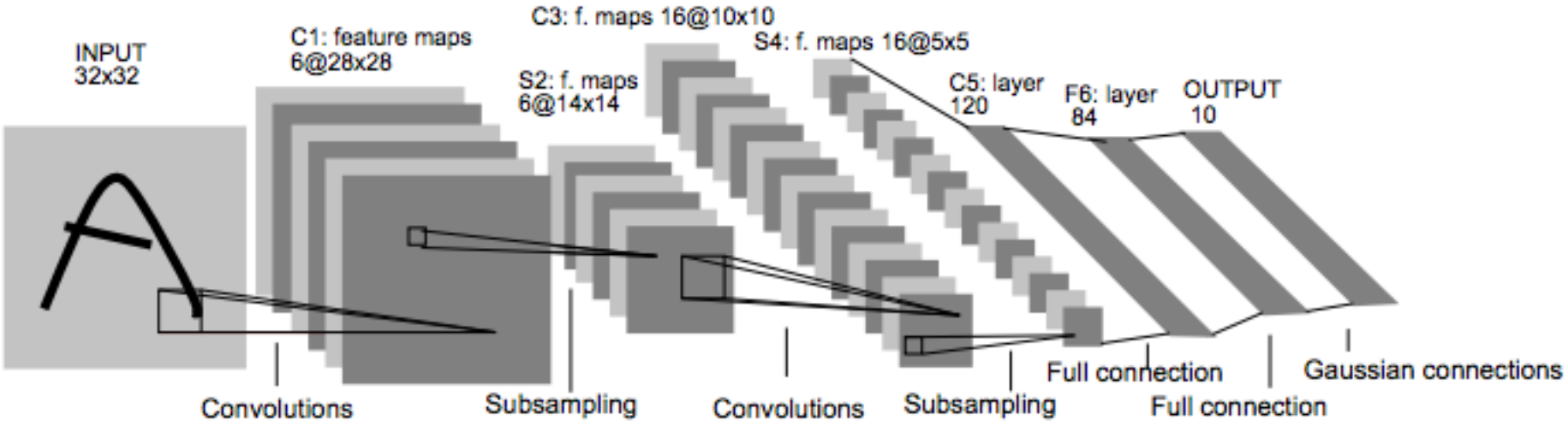





You can look at
http://cs231n.github.io/convolutional-networks/
for a very nice demo

# LeNet

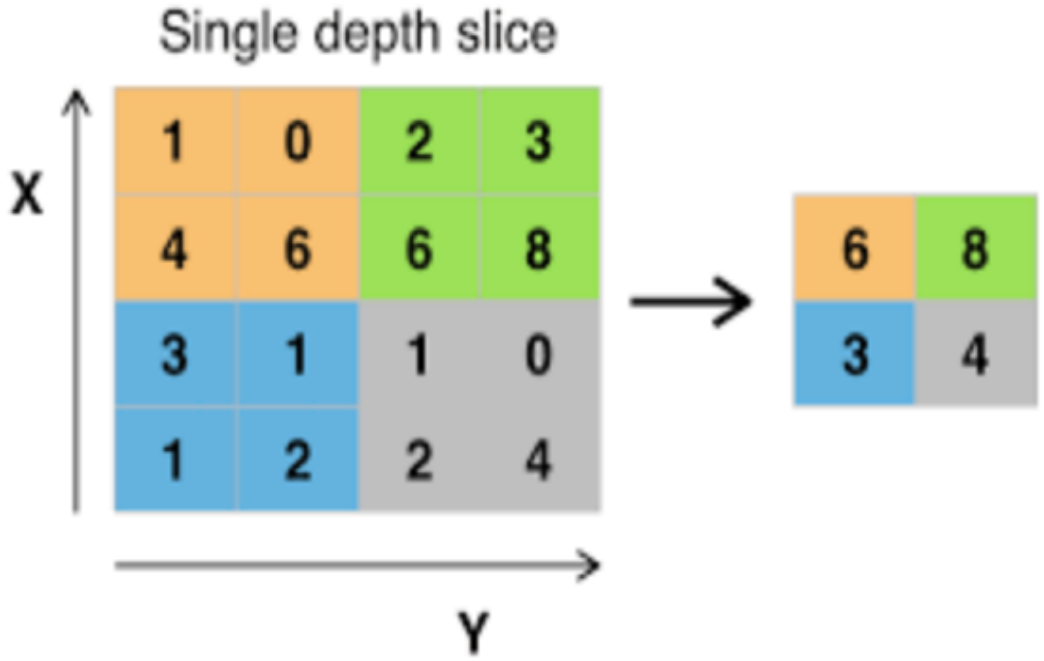





- The lower-layers are composed to alternating convolution and max-pooling layers.
- First CONV layer has 6 feature maps where each node has with 5x5 receptive fields. Total of 156 free parameters (6x5x5 + 6).
- The upper-layers are fully-connected and correspond to a traditional MLP (hidden layer + logistic regression).
- The input to the first fully-connected layer is the set of all features maps at the layer below.

# Max Pooling

- Another important concept of CNNs is max-pooling, which is a form of non-linear down-sampling.
- A node in a MaxPooling layer gets as input the maximum of the activation of the nodes in its receptive field.



# Max Pooling

Max-pooling is useful in vision for two reasons:

- By eliminating non-maximal values, it reduces computation for upper layers
- Provides translation invariance

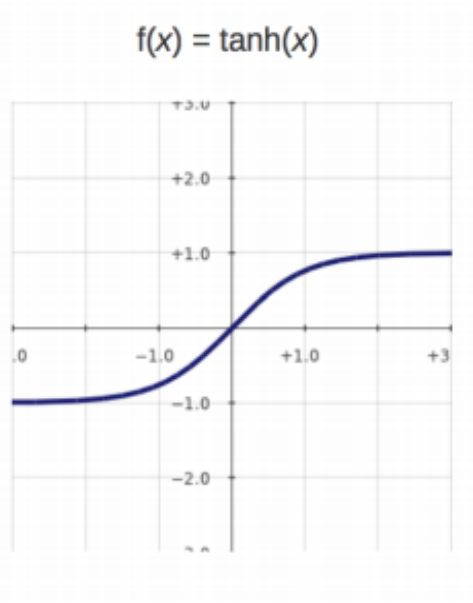
- Common application of MaxPool is done on a 2x2 region with a stride of 2

# RELU Nonlinearity

- Standard way to model a neuron

  $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$

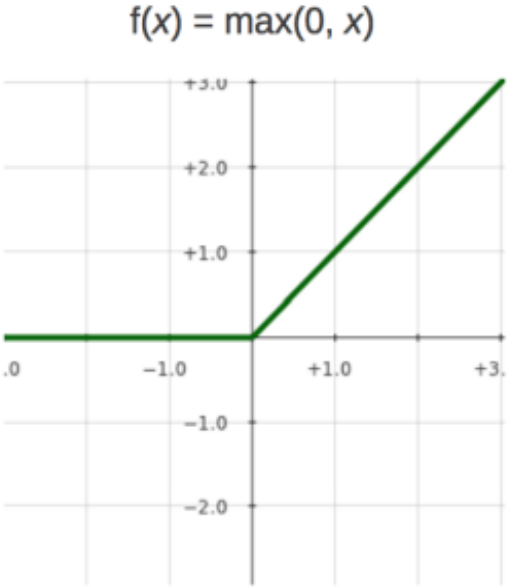
  Very slow to train




- Non-saturating nonlinearity (RELU)

  $f(x) = max(0, x)$

  Quick to train



# ReLu

ReLU layer will apply an element-wise activation function:

$$f(x) = \max(0,x)$$

- **Easy gradient computation**: 1 where x > 0 and 0 elsewhere (undet. at 0)
- Efficient gradient propagation: **No vanishing or exploding gradient problems.**
- Sparse activation: In a randomly initialized network, only about 50% of hidden units are activated (having a non-zero output).
- Biological plausibility
- Scale-invariant: max(0,ax)=a * max(0,x)
- Non-differentiable at zero: however it is differentiable anywhere else, including points arbitrarily close to (but not equal to) zero.
- Unbounded
- **Dying ReLU problem**

# Learning Rate

Large learning rates may cause divergence

Reduce in time

Batch normalization helps decide the learning rate for different layers

...

# Choosing Hyper Parameters

**Filter Shape**

Common filter shapes found in the literature vary greatly, usually based on the dataset.

Best results on MNIST-sized images (28x28) are usually in the 5x5 range on the first layer, while natural image datasets (often with hundreds of pixels in each dimension) tend to use larger first-layer filters of shape 12x12 or 15x15.

The trick is thus to find the right level of “granularity” (i.e. filter shapes) in order to create abstractions at the proper scale, given a particular dataset.

# Choosing Hyper Parameters

**Max Pooling Shape**

Typical values are 2x2 or no max-pooling. Very large input images may warrant 4x4 pooling in the lower-layers.

But this will reduce the dimension of the signal by a factor of 16, and may result in throwing away too much information!

# Transfer learning

Instead of training a deep network from scratch, you can:

- take a network trained on a different domain for a different source task
  - E.g. Network is trained with ImageNet data (millions of images, 1000 classes)
- adapt it for your domain and your target task

Transfer learning is the general term for transferring knowledge acquired in one domain to another one.

# Feature Detection

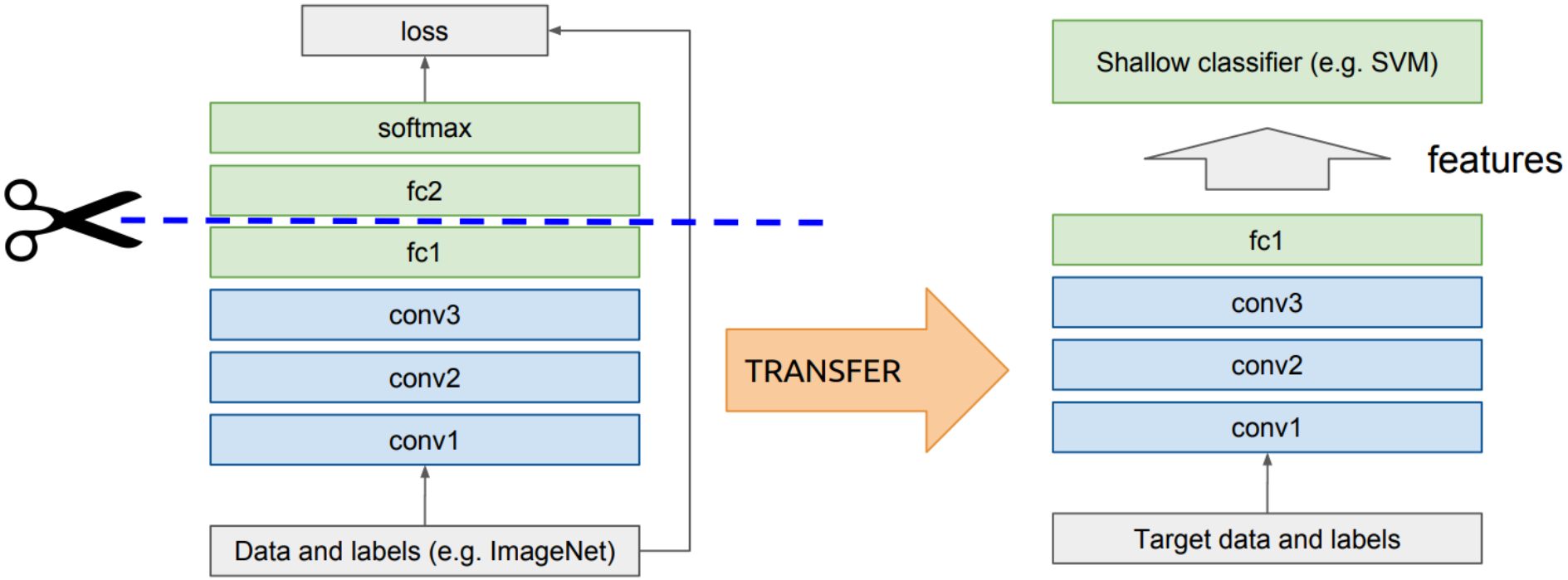



Image from http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L5-transfer.pdf

# Fine-Tuning

- Take a pretrained network, cut-off and replace the last layer as before, fine-tune the network using backpropagation.
- Bottom n layers can be frozen: not updated during backpropagation.

# Mini-Batch

Gradient descent can be done in

- batch mode (more accurate gradients) or
- stochastic fashion (much faster learning with large amounts of training data).

In deep learning, we use mini-batches, which is the set of training images from which the gradient is calculated, before a single update.

- Mini-batch size should be as high as possible, as your computer allows (20-256 are typical)

# Data Augmentation

Training data is augmented with artificial samples, through translation, rotation, scale, reflection, elastic deformations, intensity variations..., in order to obtain more robust systems.

- Data augmentation is done internally within CAFFE and can be supplemented as well.
- Data augmentation is done to increase train data size by 10xfold or more and is very effective in reducing overfitting.

# Dropout

Deep convolutional neural networks have a large number of parameters.

- If we don't have sufficiently many training examples to constrain the network, the neurons can learn the noise (idiosyncrasies) in the data.
- Regularization puts constrains on a learning system to reduce overfitting (e.g. penalizing for large weight magnitudes in NNs, ...).

Another novel idea in deep learning is dropout, whereby some nodes are “dropped out” during the network training with a preset probability.

- Network learns to cope with failures.
- Sampling from an ensemble of deep networks, to harvest benefits of ensembles.
- Must scale weights during testing.

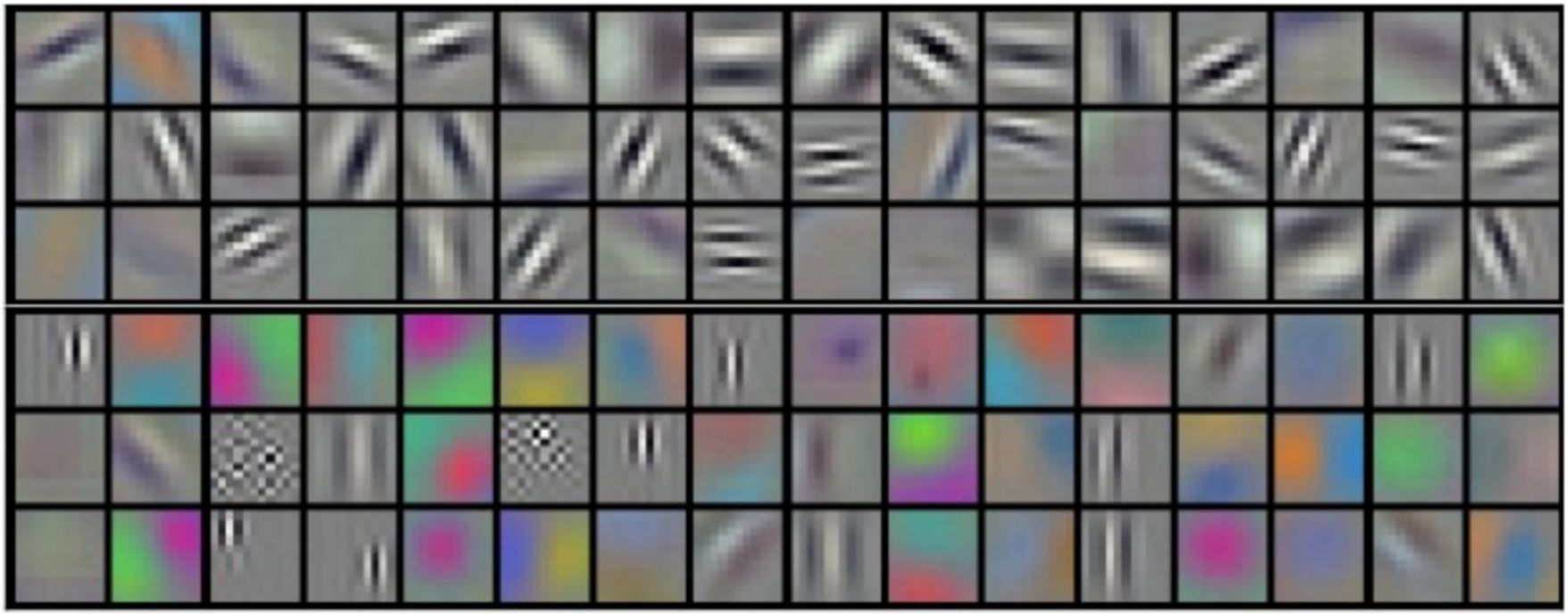
# Batch Normalization

From Ioffe and Szegedy, 2015:

- The distribution of each layer's inputs changes during training, as the parameters of the previous layers change.
- This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it difficult to train models with saturating nonlinearities.
- Make normalization a part of the model architecture and perform normalization for each training mini-batch.
    - Same accuracy with 14x fewer iterations.

# Weight Visualization

from AlexNet



The 96 convolutional kernels of size 11x11x3 learned by the first convolutional layer on the 224x224x3 input images.

**Further reading** (where some of the content and images are taken from):

- deeplearning.net
- http://cs231n.github.io/convolutional-networks/
- Papers by Hinton, Krizhevsky et al.