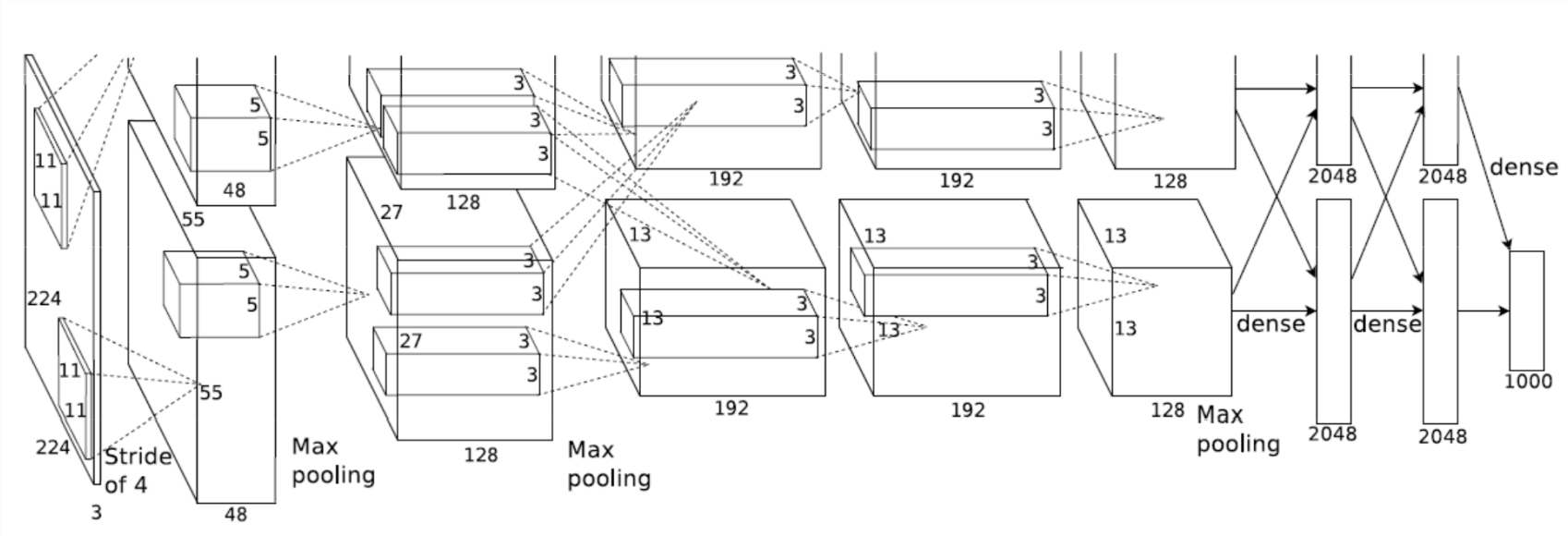
Deep Neural Networks



Berrin Yanikoglu
Sabancı University

.

Deep Learning Approaches

- Supervised learning
  - Convolutional Networks
  - Recursive Networks (LSTM,...)

- Unsupervised learning
  - AutoEncoders
  - Restricted Boltzmann Machines (RBMs)

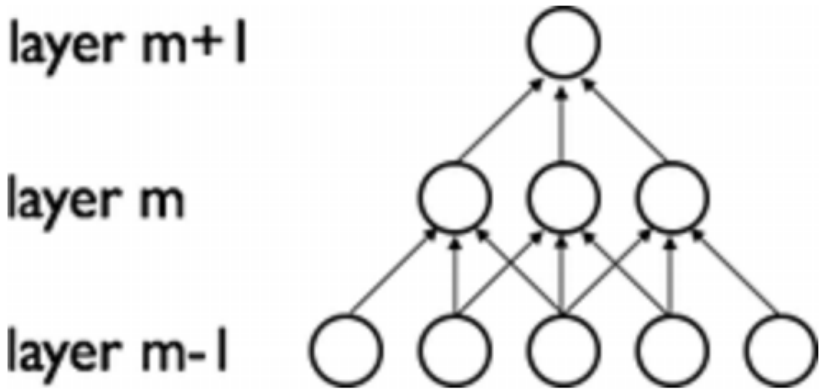
Convolutional Neural Networks
CNNs

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

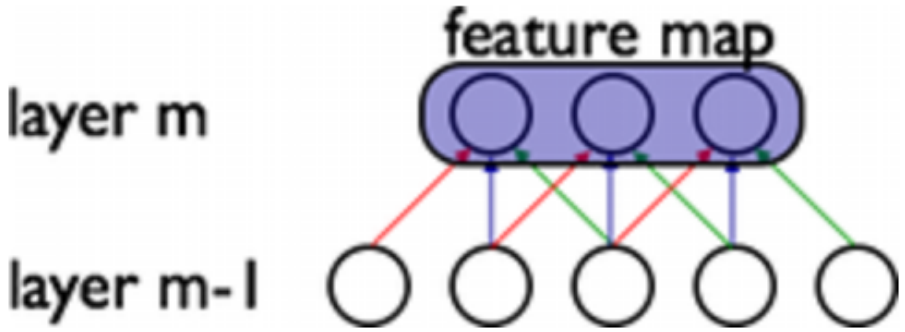
- Local connectivity from a small area
  - small receptive field size
  - spatially contiguous

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

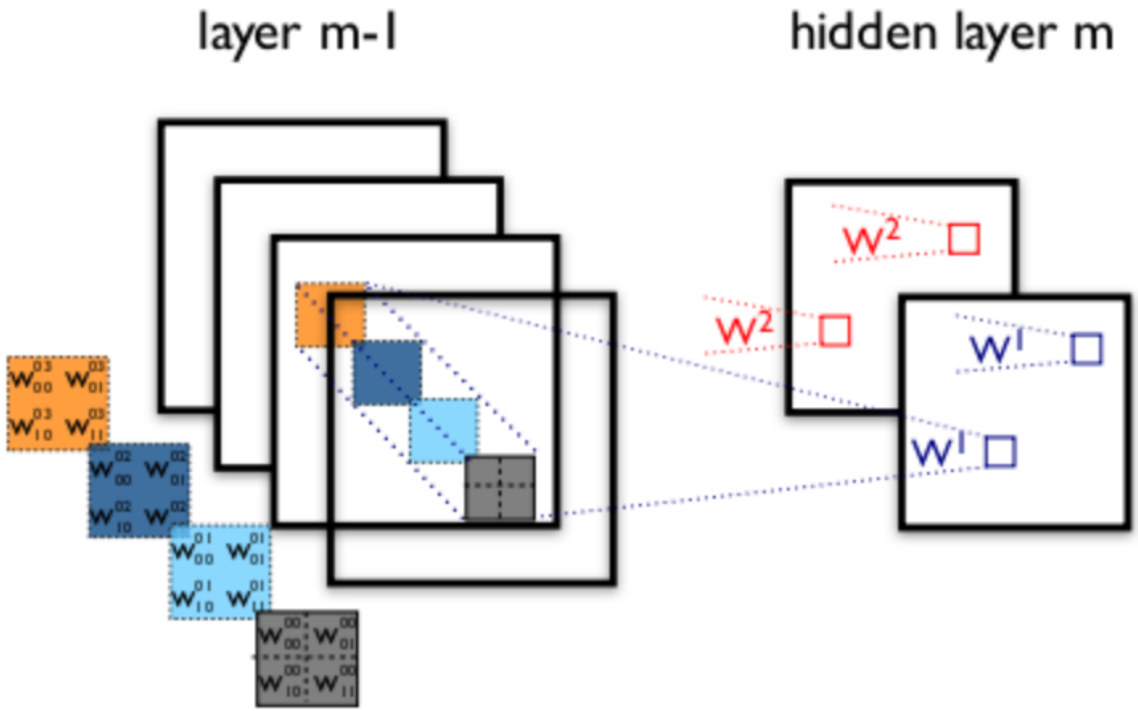
- Each filter is replicated across the entire visual field.
- These replicated units share the same weights and form a feature map.
- Replicating units in this way allows for features to be detected regardless of their position in the visual field.
- Additionally, weight sharing greatly reduces the number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



Stride, Zero-Padding

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

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

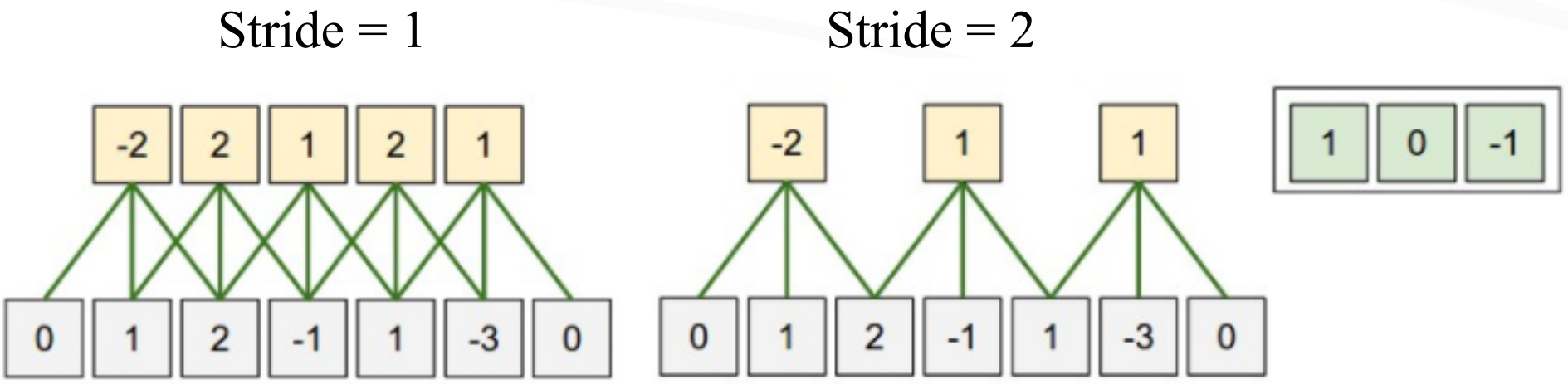
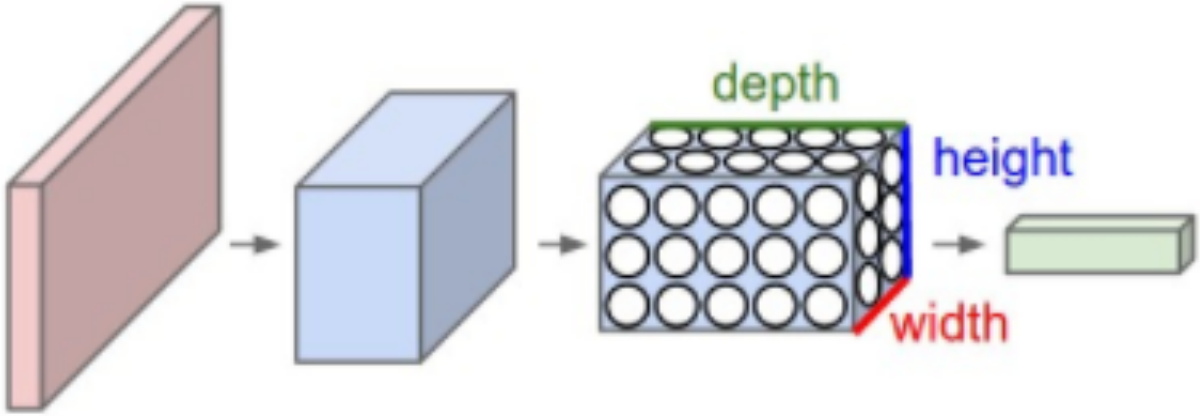


Figure from http://cs231n.github.io/convolutional-networks/

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

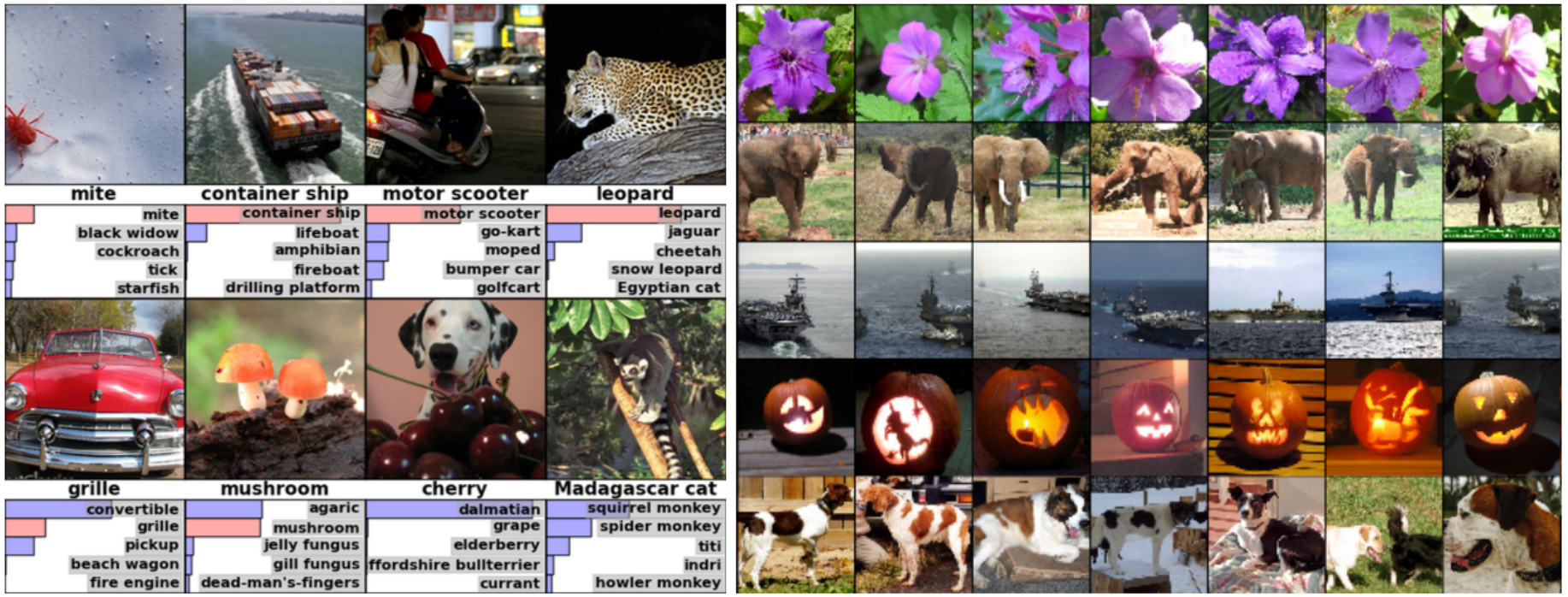


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 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

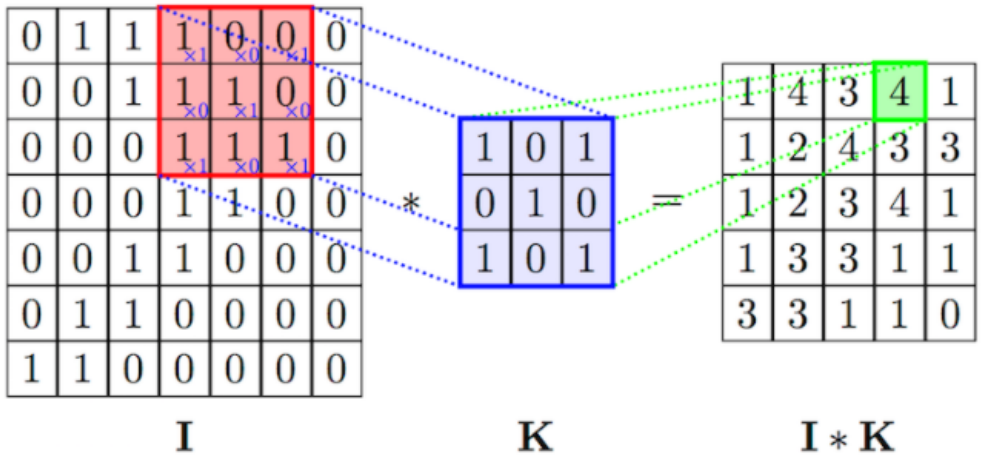
- Using the real world example, we see that there are 55*55*96 = 290,400 neurons in the first Conv Layer.
- Each has 11*11*3 = 363 weights and 1 bias.
- Together, this adds up to 290400 * 364 = 105,705,600 parameters on the first layer of the ConvNet alone.

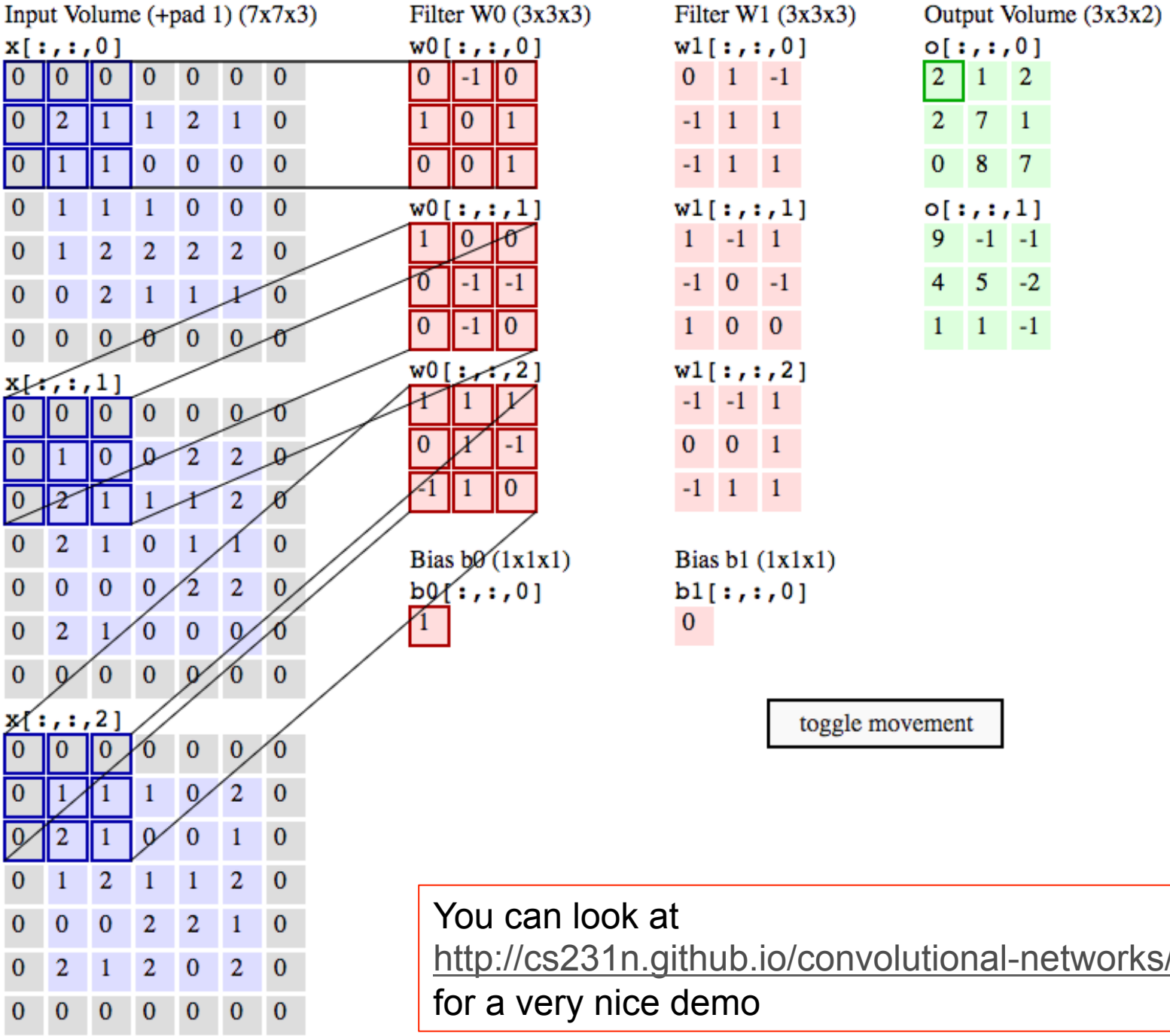
- With weight sharing within a feature map, we have 96 unique **set** of weights, for a total of 96 x11x11x3 = 34,944 (+96 biases).

Convolution

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

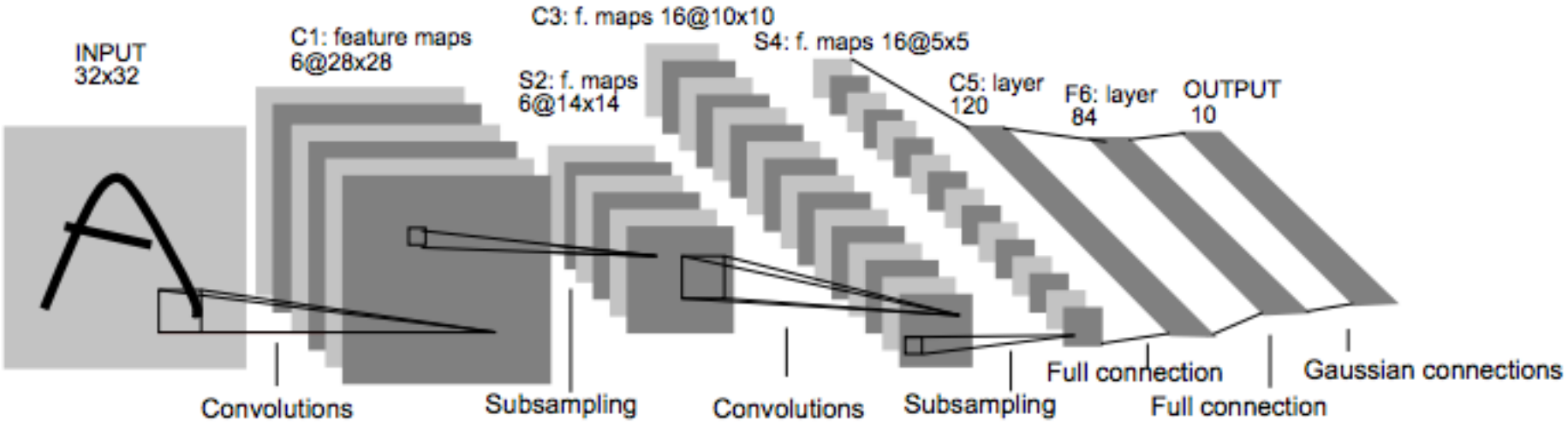
Dot product between pixels in the receptive field and the weights





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

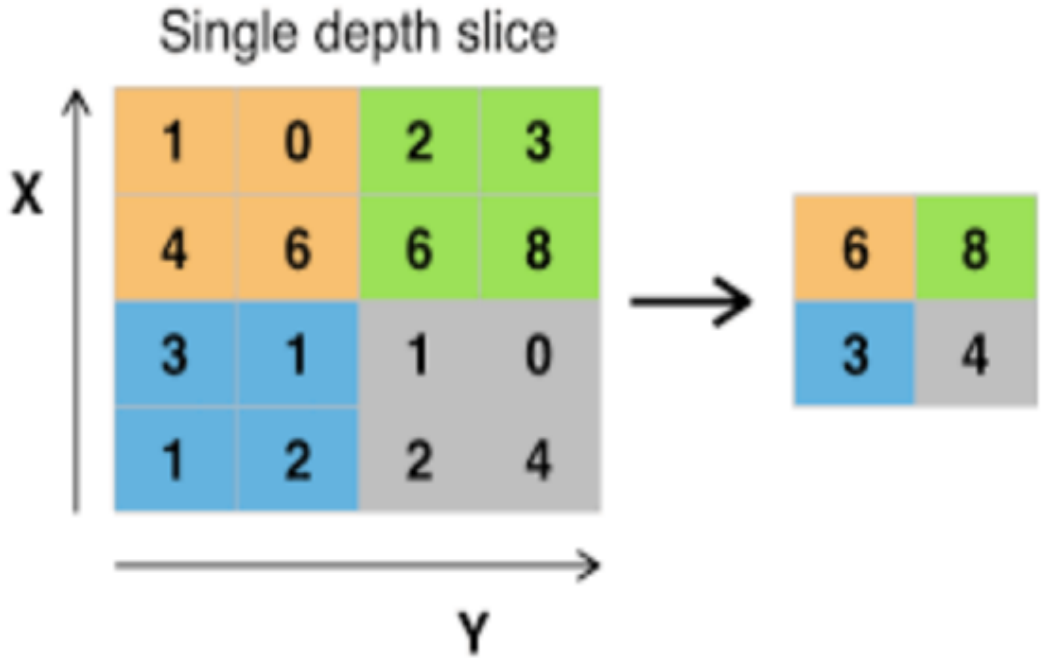
LeNet



- 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

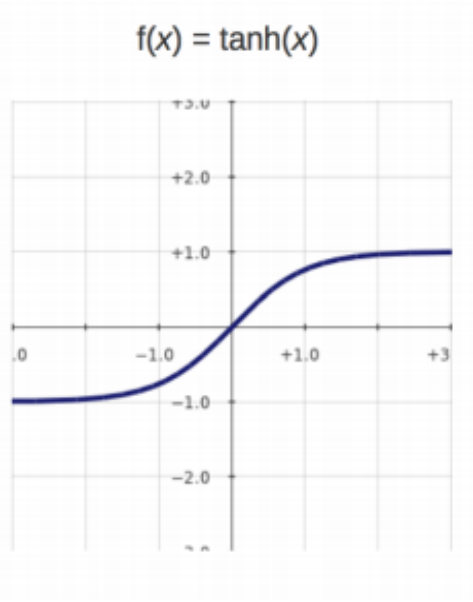
- Common application of MaxPool is done on a 2x2 region with a stride of 2

RELU Nonlinearity

- Standard way to model a neuron

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

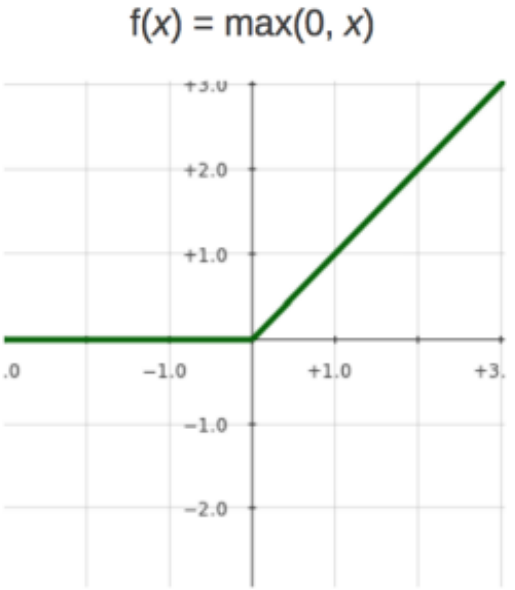
  Very slow to train



- 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

Choosing Hyper Parameters

- **Number of filters**
  - computing the activations of a single convolutional filter is much more expensive than with traditional MLPs
    - Assume layer (l-1) contains $K^{\{l-1\}}$ feature maps and MxN pixel positions and there are K^l filters at layer l of shape mxn.
    - Then computing a feature map costs $(M-m) \times (N-n) \times m \times n \times K^{\{l-1\}}$.
    - The total cost is $(M-m) \times (N-n) \times m \times n \times K^{\{l-1\}} \times K^l$.
    - For a standard MLP, the cost would only be $K^l \times K^{\{l-1\}}$ where there are K^l different neurons at level l.

Notes: Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while layers higher up can have much more. In fact, to equalize computation at each layer, the product of the number of features and the number of pixel positions is typically picked to be roughly constant across layers.

To preserve the information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) to be non-decreasing from one layer to the next (of course we could hope to get away with less when we are doing supervised learning). The number of feature maps directly controls capacity and so that depends on the number of available examples and the complexity of the task.

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

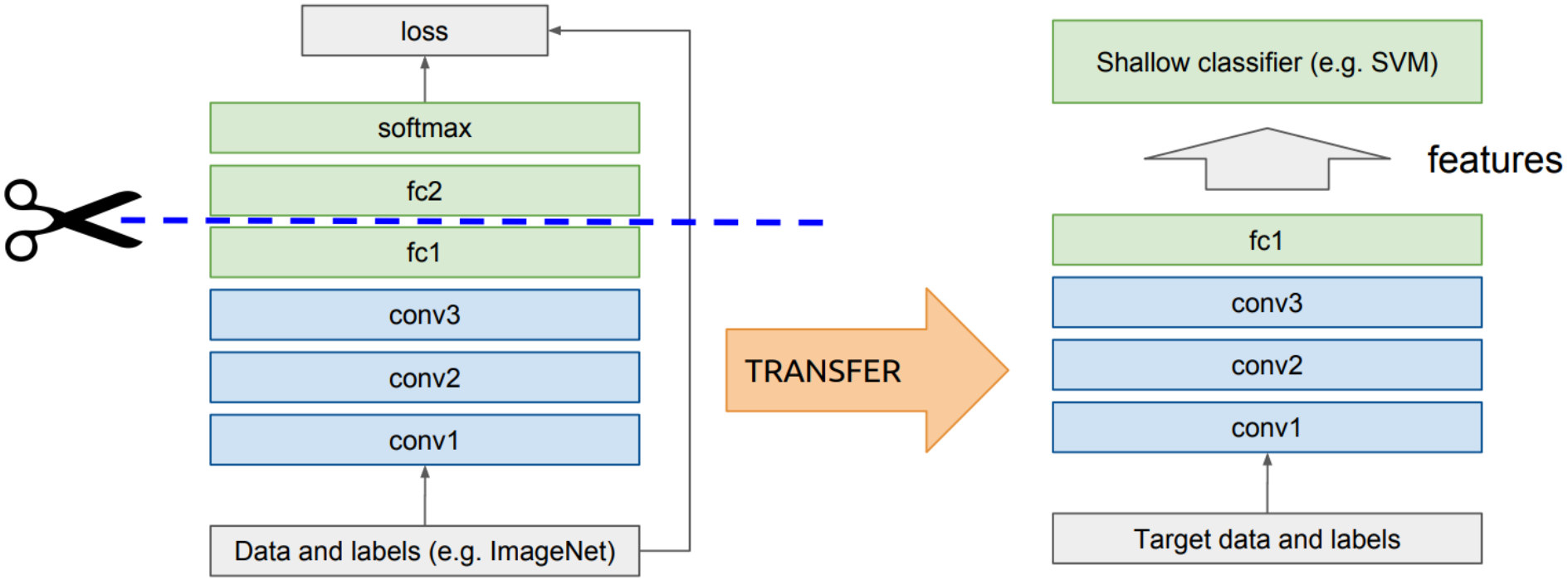


Image from http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L5-transfer.pdf

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.

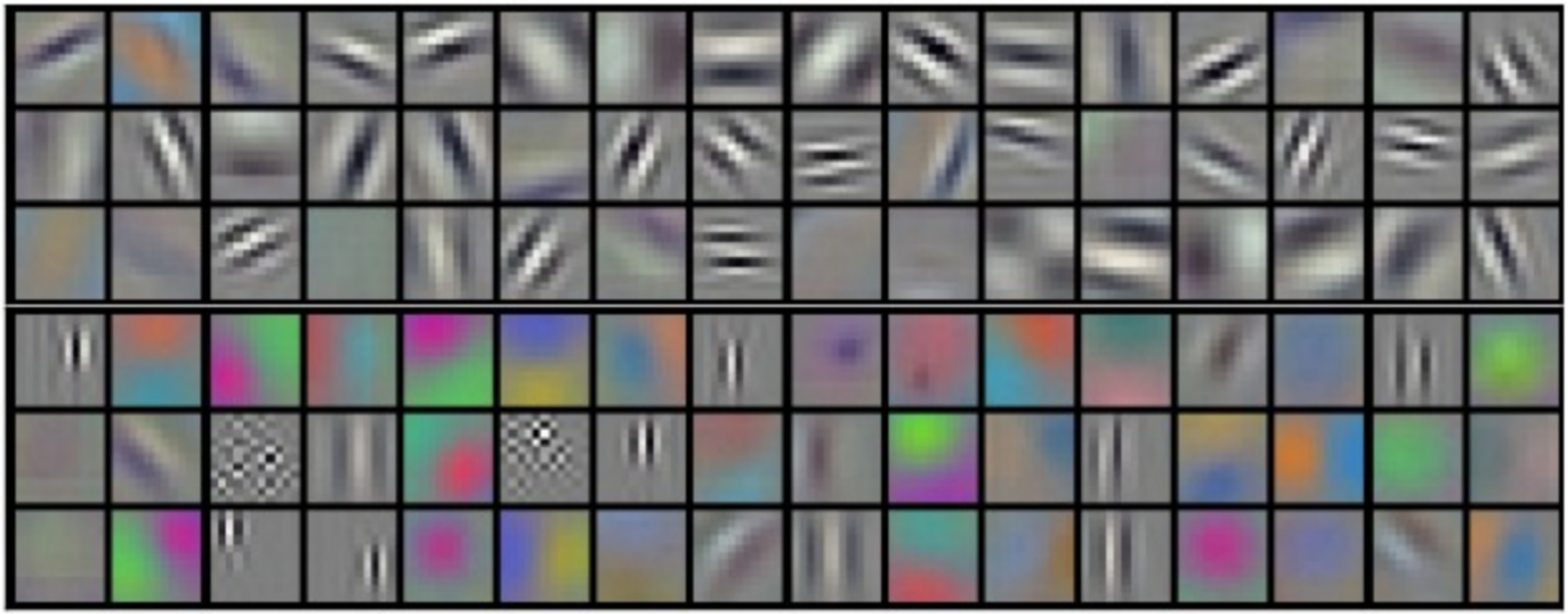
Batch Normalization

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



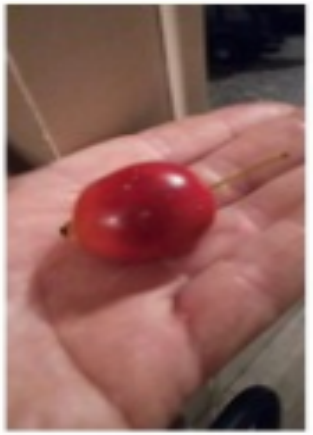
The 96 convolutional kernels of size 11x11x3 learned by the first convolutional layer on the 224x224x3 input images.

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.

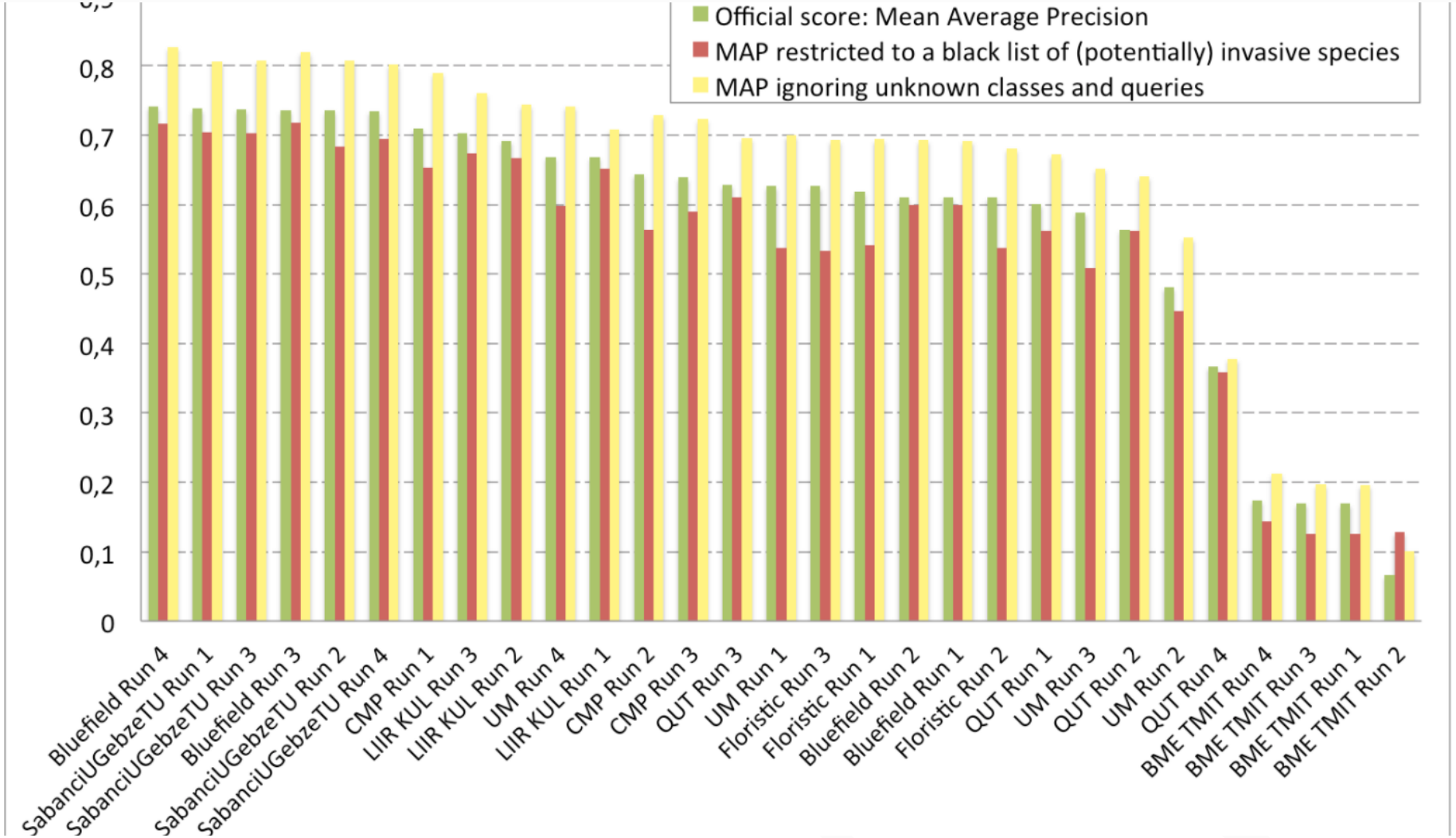
APPLICATION TO PLANT RECOGNITION PROBLEM



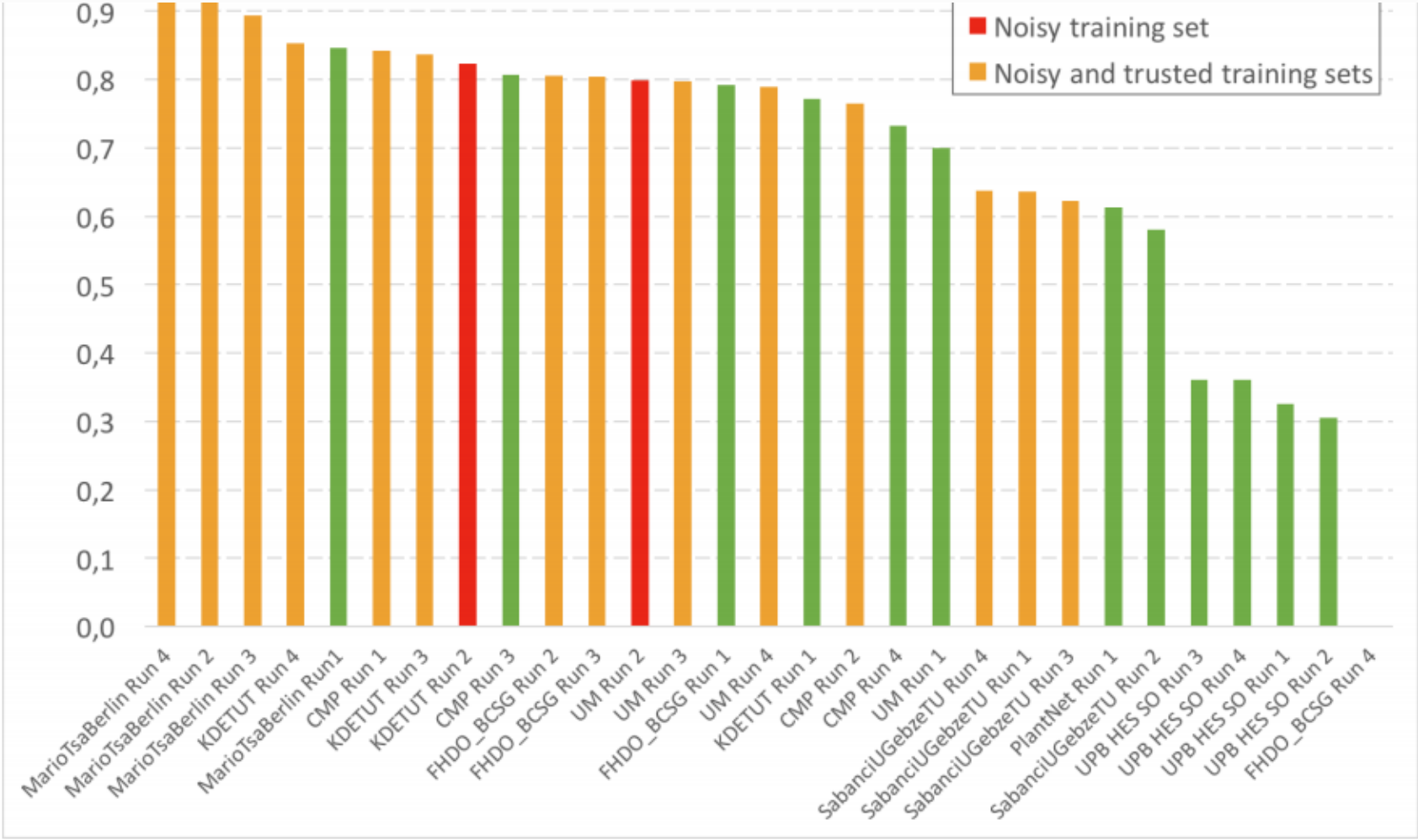







Official score: Mean Average Precision
MAP restricted to a black list of (potentially) invasive species
MAP ignoring unknown classes and queries
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
Bluefield Run 4
SabanciUGebzeTU Run 1
SabanciUGebzeTU Run 3
Bluefield Run 3
SabanciUGebzeTU Run 2
SabanciUGebzeTU Run 4
CMP Run 1
LIIR KUL Run 3
LIIR KUL Run 2
UM Run 4
LIIR KUL Run 1
CMP Run 2
CMP Run 3
QUT Run 3
UM Run 1
Floristic Run 3
Floristic Run 1
Bluefield Run 2
Bluefield Run 1
Floristic Run 2
QUT Run 1
UM Run 3
QUT Run 2
UM Run 2
QUT Run 4
BME TMIT Run 4
BME TMIT Run 3
BME TMIT Run 1
BME TMIT Run 2

PlantCLEF 2017 Results



PlantCLEF

- GoogLeNet
  - Winner of ILSVRC 2014, 6.8 million parameters

  Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition. (2015)

- VGGNet
  - Runner-up in ILSVRC 2014, 144 million parameters

  Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Computing Research Repository (CoRR) (2014) arXiv: 1409.1556.

How to fine-tune?

-> **How many iterations**: as many as you can afford.

-> **Data augmentation**: how many more samples are to be generated per input image?

-> **Batch size**: the number of input samples used during gradient calculation for weight update

+ **learning rate**, **weight decay**, **momentum** (typically initialized to default values)

The first 3 affect training time strongly!

- All tests were run on a linux system with a Tesla K40c and 12GB of video memory

- Average training time per iteration with a batch size of 20: 1.79 seconds for VGGNet and 0.45 seconds for GoogLeNet

	Branch	Entire	Flower	Fruit	Leaf	LeafScan	Stem	Overall
GoogLeNet (100K, 20, 10x)	44.09	38.36	67.93	57.65	60.57	94.16	37.01	61.06
GoogLeNet (300K, 20, 10x)	55.08	46.05	76.23	67.96	68.07	95.77	45.76	68.57
GoogLeNet (500K, 20, 10x)	55.02	47.66	76.43	69.11	69.13	95.58	45.49	69.11
GoogLeNet (100K, 40, 10x)	45.63	41.03	70.05	61.23	60.73	93.20	36.49	62.48
GoogLeNet (100K, 60, 10x)	50.98	41.09	73.07	63.59	65.50	94.19	41.52	65.18
GoogLeNet (100K, 20, 80x)	54.01	48.06	73.38	64.99	68.63	94.85	39.84	67.32
GoogLeNet (300K, 60, 80x) (*)	67.56	60.28	83.30	76.88	76.30	96.93	54.33	76.87

- The best GoogLeNet results are obtained with (300K,60,80x) as 76.87% on the test set.
- Increasing the number of iterations is much more beneficial than increasing the batch size, for the same total running time (68.57% vs. 65.18 %).
- Data augmentation is the second most important training parameter (69.11 % vs. 67.32 %)