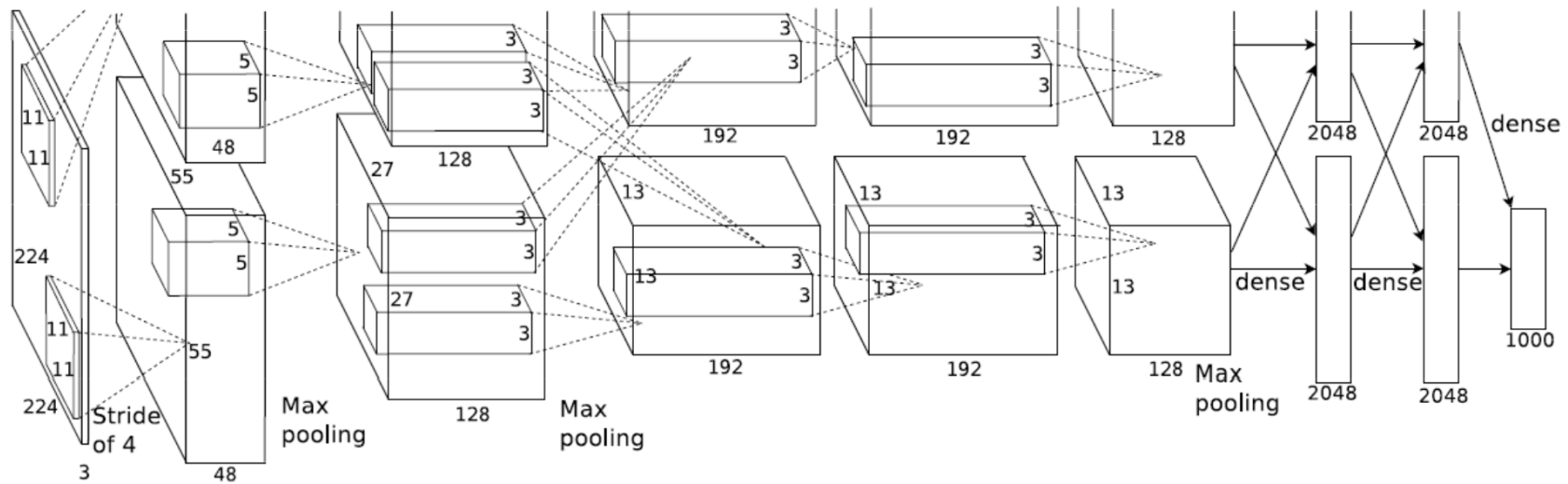


Deep Neural Networks



Berrin Yanikoglu
Sabancı University

Deep Learning Approaches

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

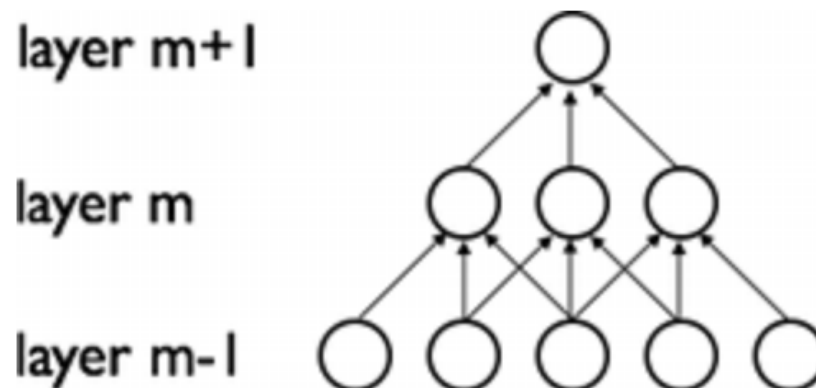
Convolutional Neural Networks CNNs

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

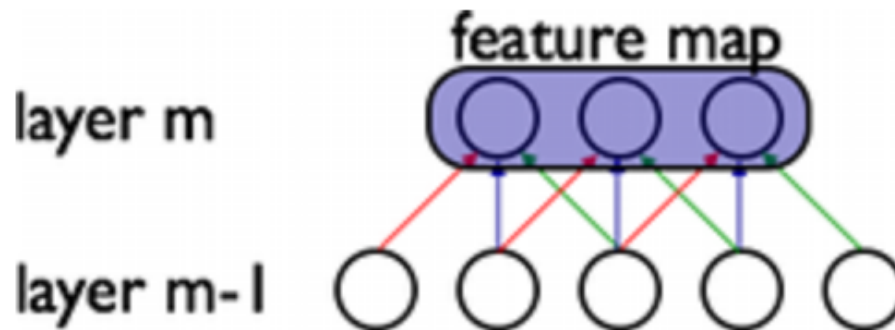
- Small and local connectivity
 - small receptive field size
 - spatially contiguous

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

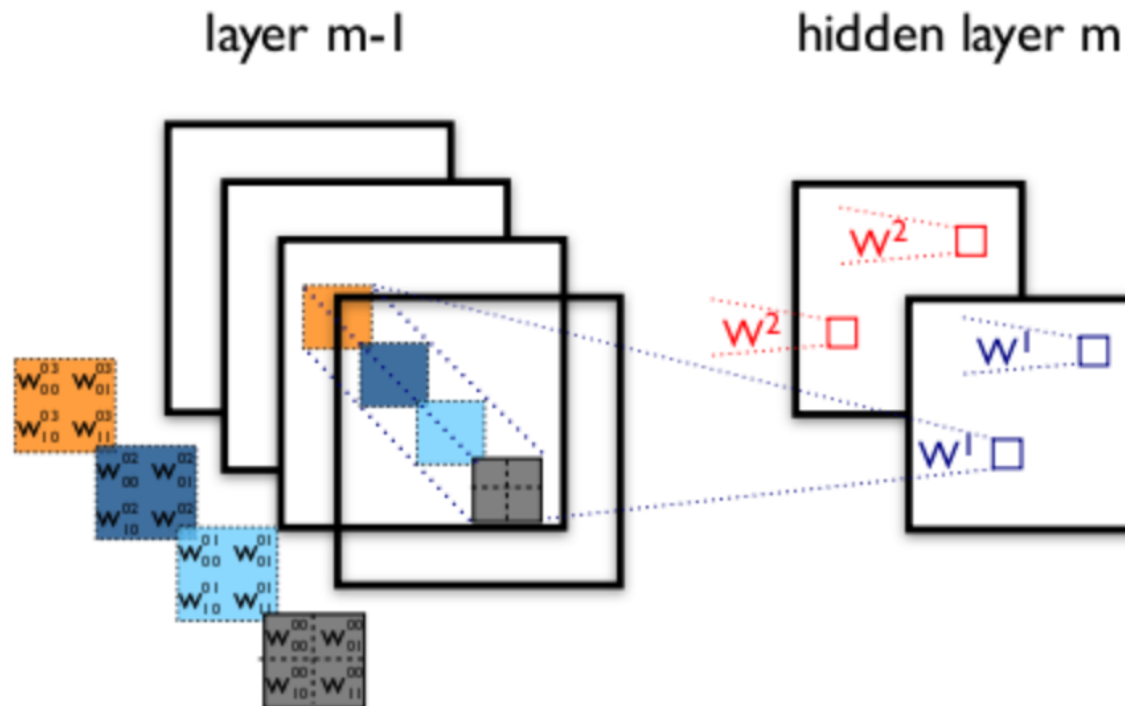
- Each filter is **replicated** across the entire visual field.
- These replicated units share the same weights and form a **feature map**.
- Replicating units in this way allows for features to be detected **regardless of their position in the visual field**.
- Additionally, weight sharing greatly **reduces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

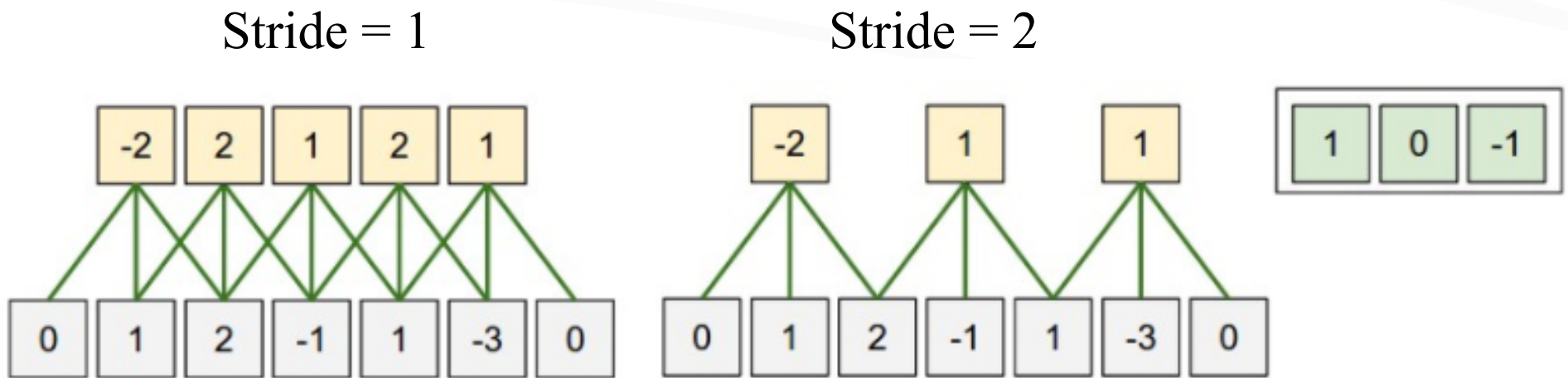
- There are 4 and 2 feature maps in layers $m-1$ and m .
- The receptive field of nodes in layer m spans **all four input feature maps**.
- The weights W^0 and W^1 are 3D weight tensors.
 - The leading dimension indexes the input feature maps, while the other two refer to the pixel coordinates.



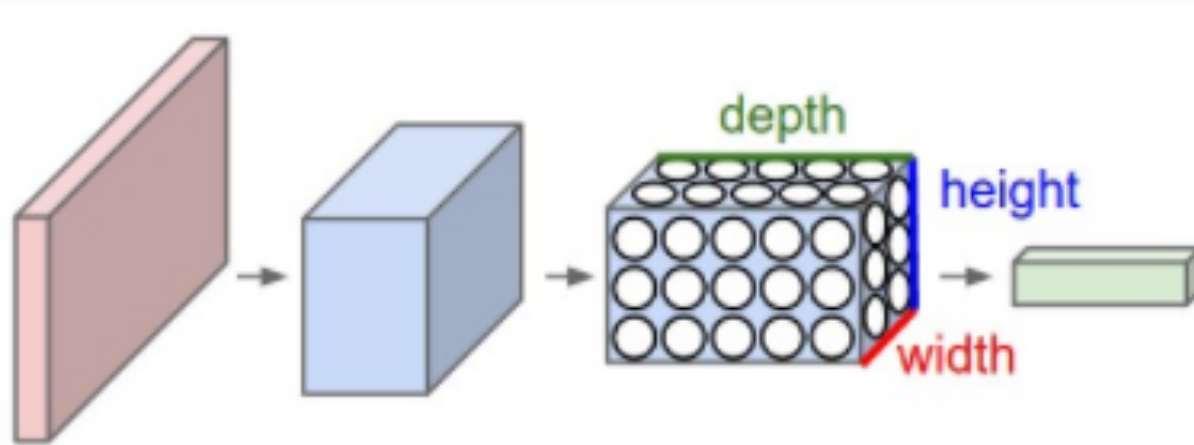
Stride, Zero-Padding

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.



Real-world example



- The Krizhevsky et al. architecture that won the ImageNet challenge in 2012 accepted images of size $[227 \times 227 \times 3]$.
- 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 $[55 \times 55 \times 96]$.
 - Each of the $55 \times 55 \times 96$ neurons in this volume was connected to a region of size $[11 \times 11 \times 3]$ in the input volume.
 - Moreover, all 96 neurons in each depth column are connected to the same $[11 \times 11 \times 3]$ region of the input, but of course with different weights.

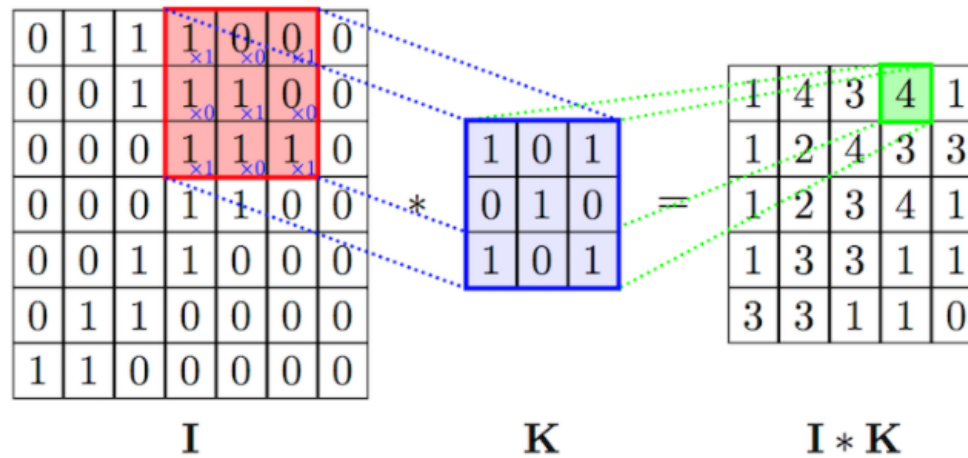
Weight Sharing

- Using the real world example, we see that there are $55 \times 55 \times 96 = 290,400$ neurons in the first Conv Layer.
- Each has $11 \times 11 \times 3 = 363$ weights and 1 bias.
- Together, this adds up to $290400 \times 364 = 105,705,600$ 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 $96 \times 11 \times 11 \times 3 = 34,944$ (+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



Input Volume (+pad 1) (7x7x3)

$x[:, :, 0]$						
0	0	0	0	0	0	0
0	2	1	1	2	1	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
0	1	2	2	2	2	0
0	0	2	1	1	1	0
0	0	0	0	0	0	0
$x[:, :, 1]$						
0	0	0	0	0	0	0
0	1	0	0	2	2	0
0	2	1	1	1	2	0
0	2	1	0	1	1	0
0	0	0	0	2	2	0
0	2	1	0	0	0	0
0	0	0	0	0	0	0
$x[:, :, 2]$						
0	0	0	0	0	0	0
0	1	1	1	0	2	0
0	2	1	0	0	1	0
0	1	2	1	1	2	0
0	0	0	2	2	1	0
0	2	1	2	0	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

$w0[:, :, 0]$		
0	-1	0
1	0	1
0	0	1

$w0[:, :, 1]$		
1	0	0
0	-1	-1
0	-1	0

$w0[:, :, 2]$		
1	1	1
0	1	-1
-1	1	0

Bias b0 (1x1x1)

$b0[:, :, 0]$		
1		

Filter W1 (3x3x3)

$w1[:, :, 0]$		
0	1	-1
-1	1	1
-1	1	1

$w1[:, :, 1]$		
1	-1	1
-1	0	-1
1	0	0

$w1[:, :, 2]$		
-1	-1	1
0	0	1
-1	1	1

Bias b1 (1x1x1)

$b1[:, :, 0]$		
0		

Output Volume (3x3x2)

$o[:, :, 0]$		
2	1	2
2	7	1
0	8	7
$o[:, :, 1]$		
9	-1	-1
4	5	-2
1	1	-1

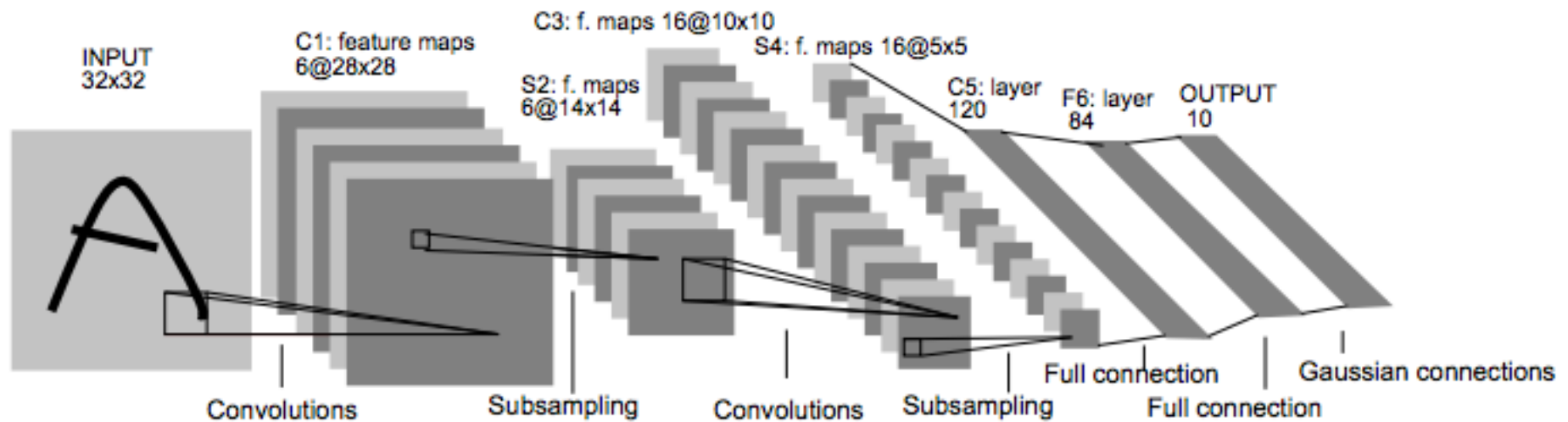
toggle movement

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

LeNet

PROC. OF THE IEEE, NOVEMBER 1998

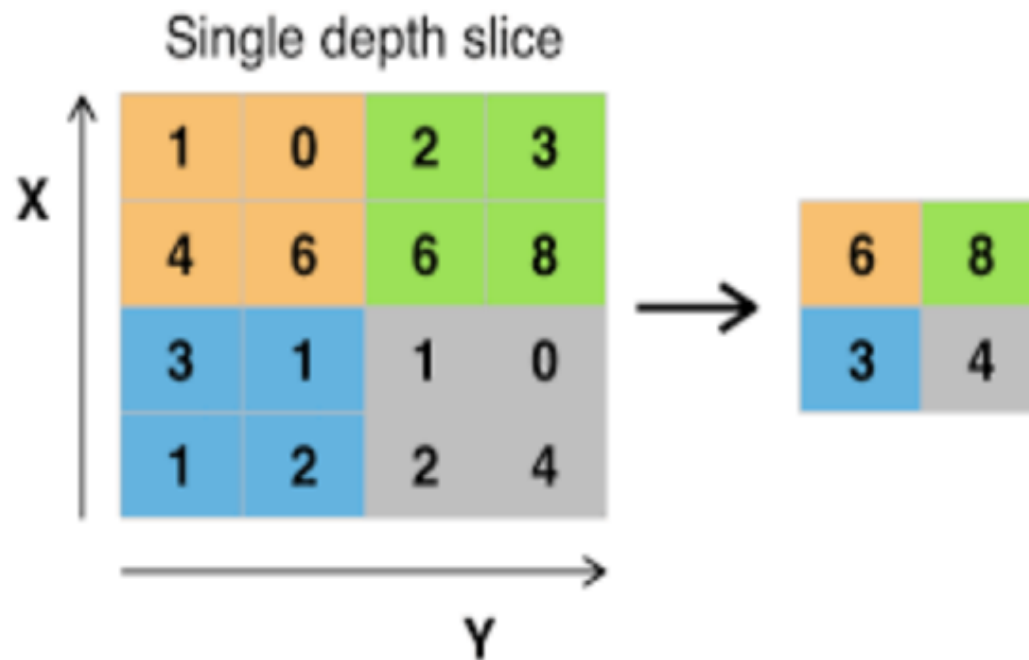
7



- 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 ($6 \times 5 \times 5 + 6$).
- 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

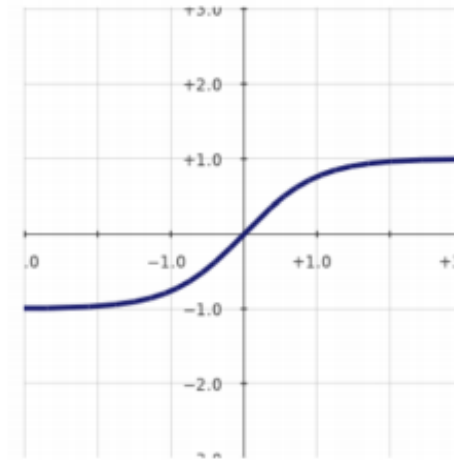
RELU Nonlinearity

- Standard way to model a neuron

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

Very slow to train

$$f(x) = \tanh(x)$$

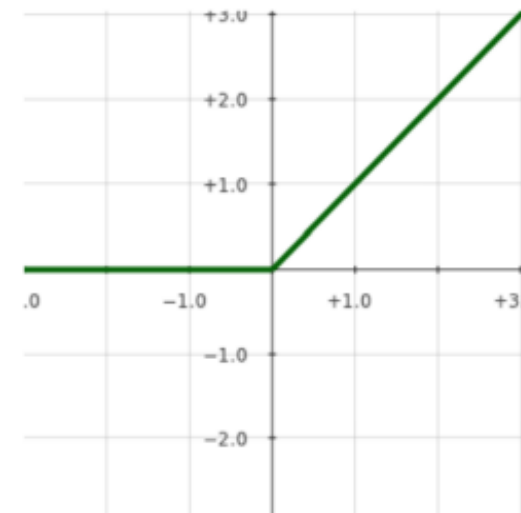


- Non-saturating nonlinearity (RELU)

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

Quick to train

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



ReLu

ReLU layer will apply an element-wise activation function:

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

- Easy gradient computation
- 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 $M \times N$ pixel positions and there are K^l filters at layer l of shape $m \times n$.
 - 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

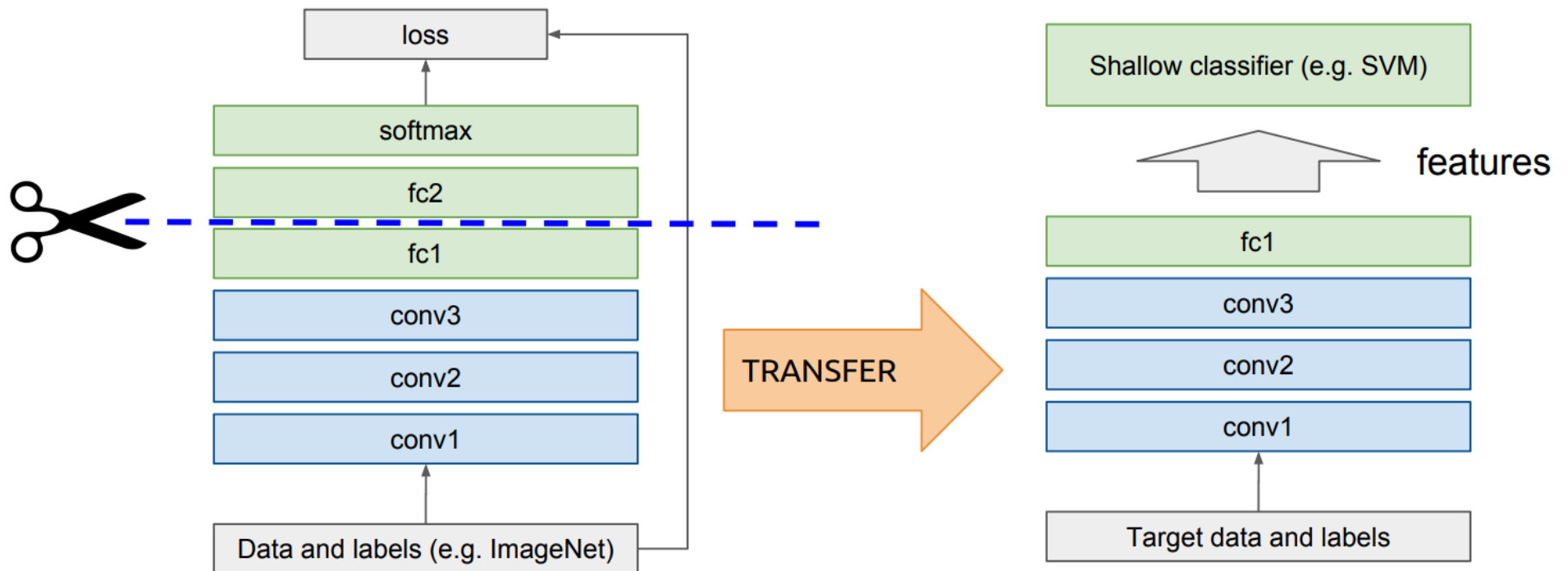


Image from <http://imatge-upc.github.io/telecombcn-2016-dlcv/slides/D2L5-transfer.pdf>

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 10x fold or more and is very effective in reducing **overfitting**.

Dropout

Deep convolutional neural networks have a large number of parameters, especially in the fully connected layers. 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 constraints 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, requiring the network to cope with failures.

Weight Visualization



From: <http://cs231n.github.io/convolutional-networks/>

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.