ECBM E4040
Neural Networks and Deep Learning

Convolutional Neural Networks (CNN)

Zoran Kostić
Columbia University
Electrical Engineering Department
& Data Sciences Institute

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

References and Acknowledgments

- **Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, http://www.deeplearningbook.org/ , chapter 6.**
- **Lecture material by bionet group / Prof. Aurel Lazar (http://www.bionet.ee.columbia.edu/).**
- **NVIDIA GPU Teaching Kit (NVIDIA and New York University)**

Convolutional Neural Network Introduction

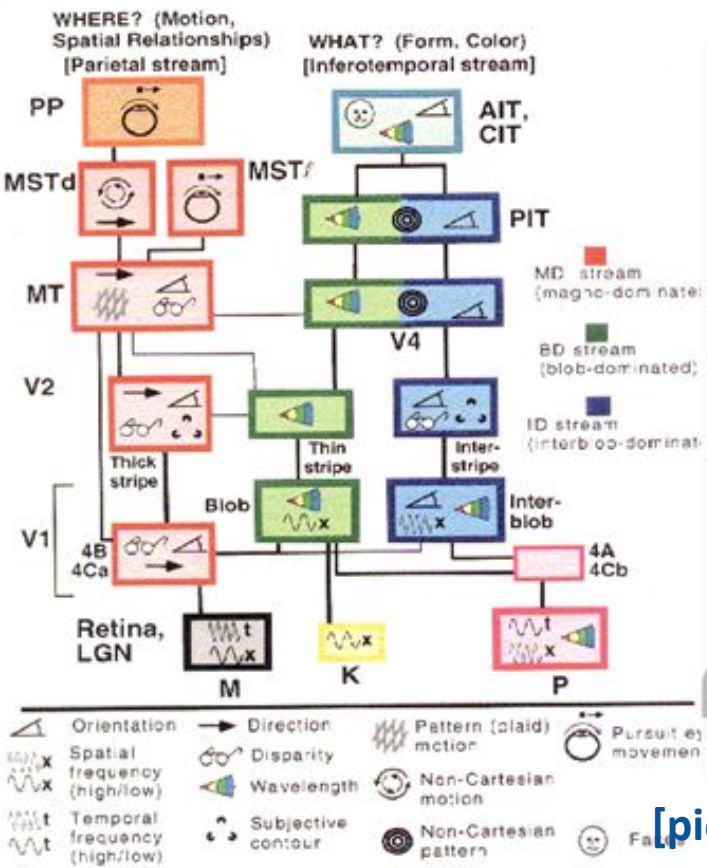
Evolution of Pattern Recognition

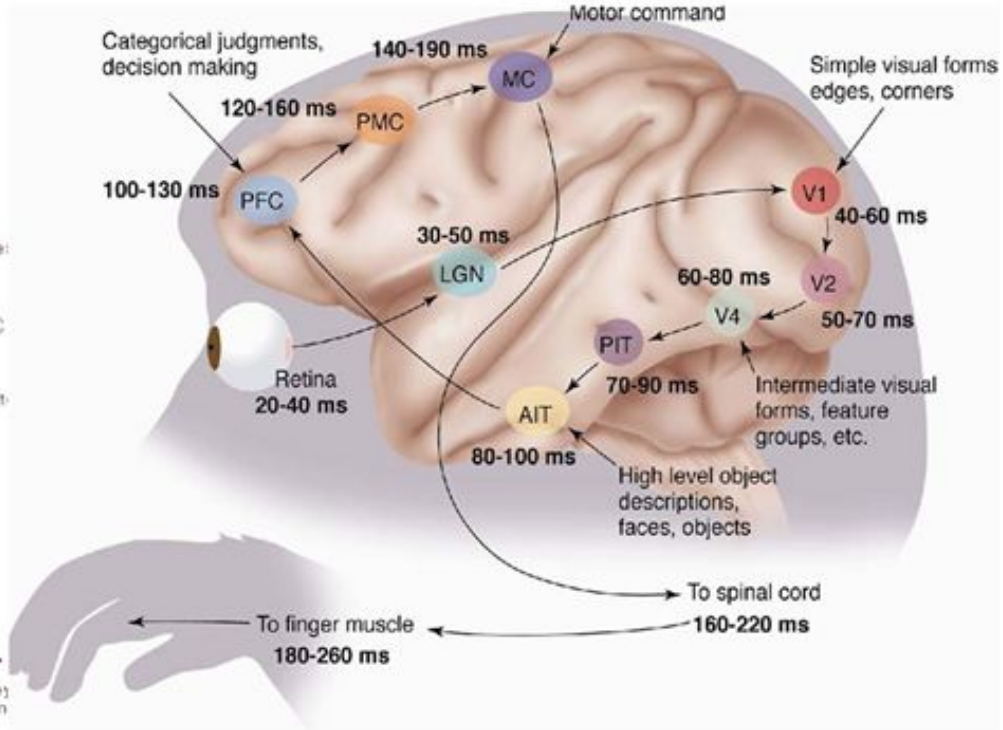
Inspiration from Biology

- Mammalian Visual System

Mathematical Similarity - Gabor Filters

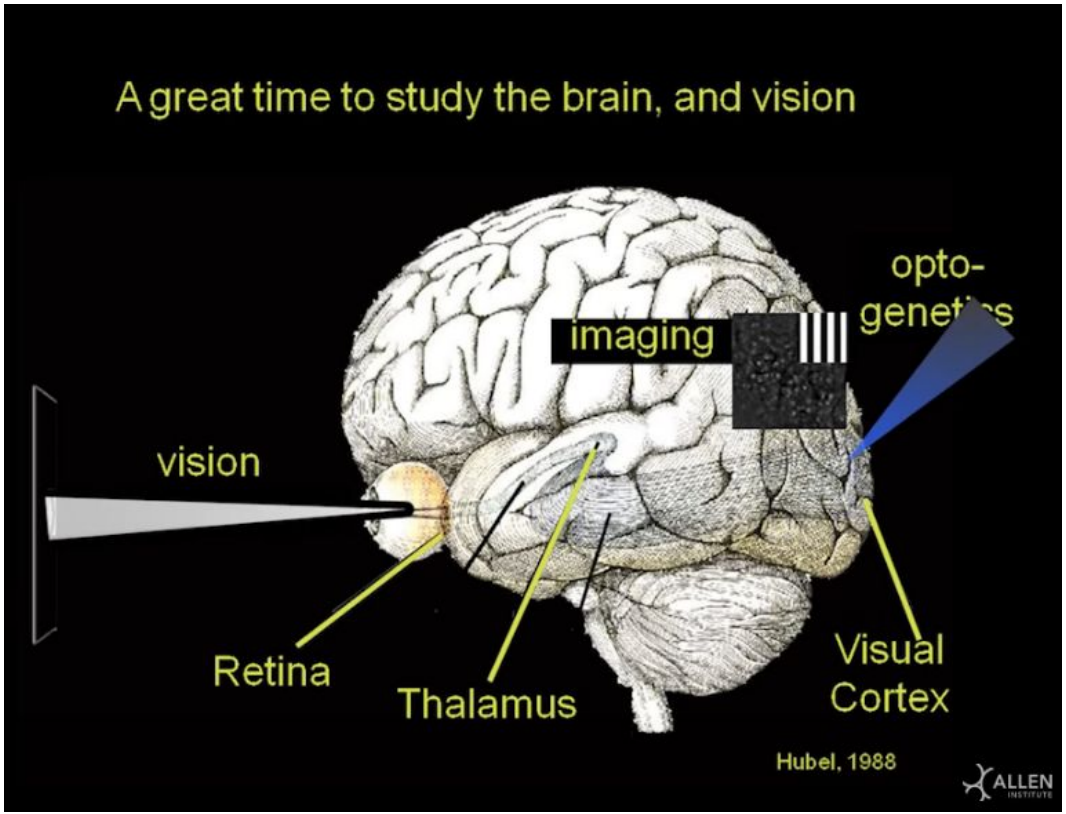
The Mammalian Visual Cortex is Hierarchical





[picture from Simon Thorpe] [Gallant & Van Essen]

A Walk-through of the Mammalian Visual System

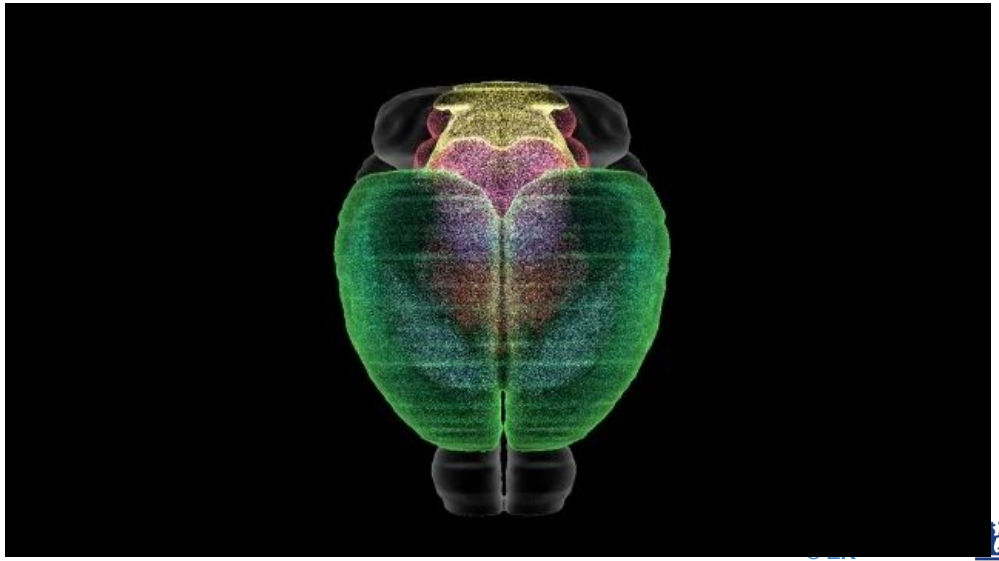


Allen Institute: https://www.youtube.com/watch?v=mtPgW1ebxmE

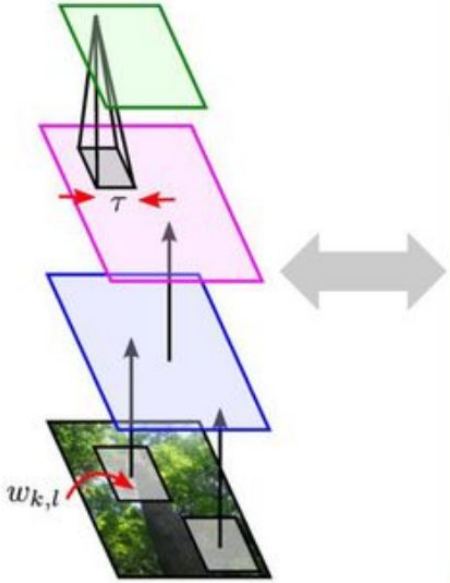
Blue Brain Project

https://www.epfl.ch/research/domains/bluebrain/

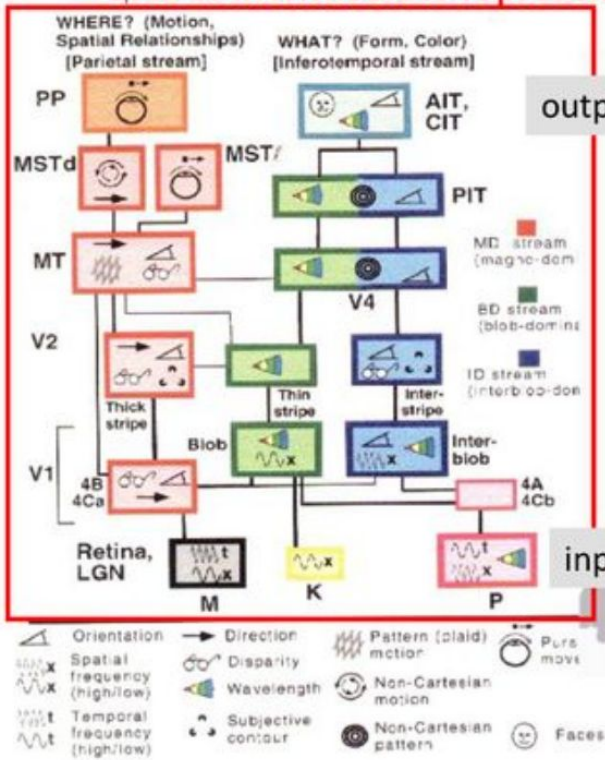
The goal of the Blue Brain Project is to build biologically detailed digital reconstructions and simulations of the rodent, and ultimately the human brain.



Convolutional Neural Net

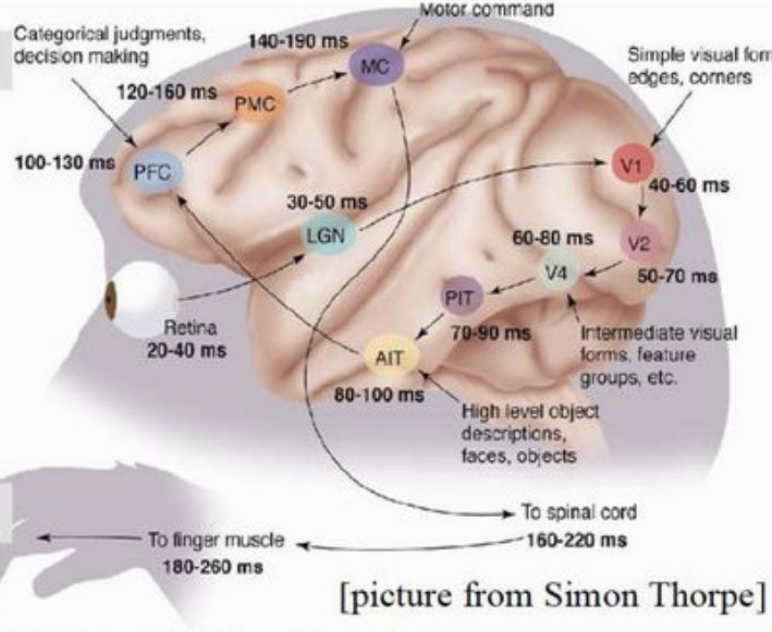


- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



output

input



[picture from Simon Thorpe]

[Gallant & Van Essen]

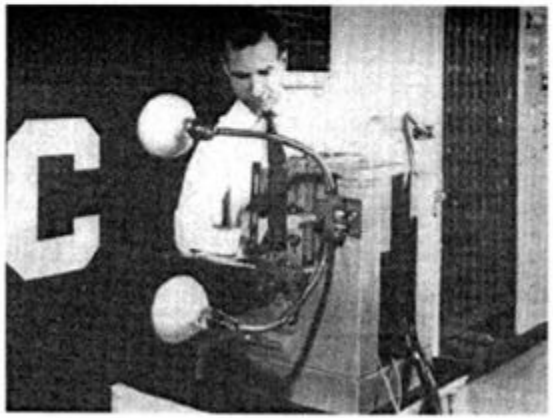
History of Convolutional Neural Networks

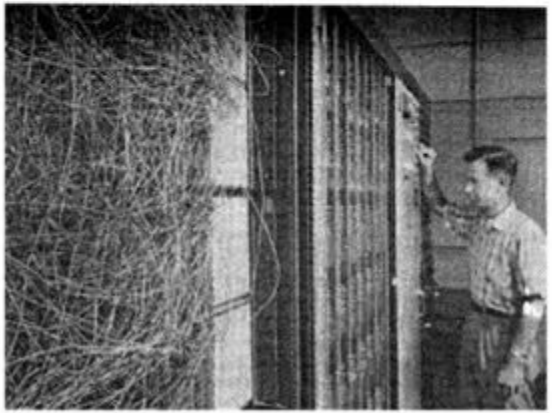
Pattern Recognition

Hand-crafted Feature Extractor → "Simple" Trainable Classifier →

The traditional model of pattern recognition (since the late 50's)

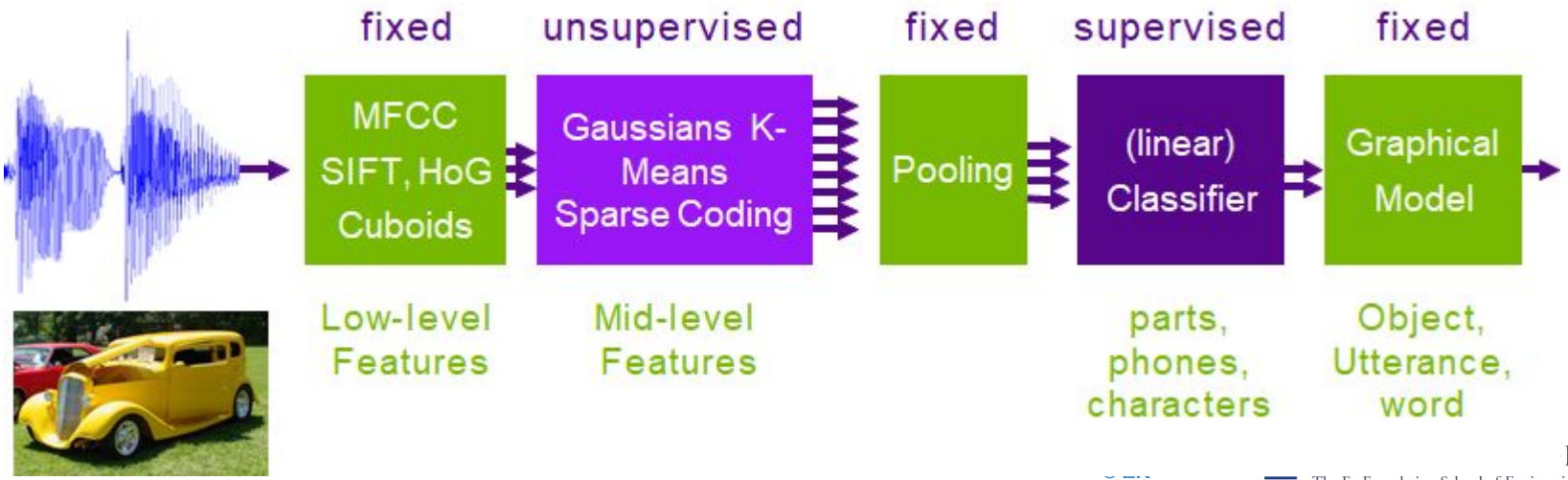
Fixed/engineered features (or fixed kernel) + trainable classifier





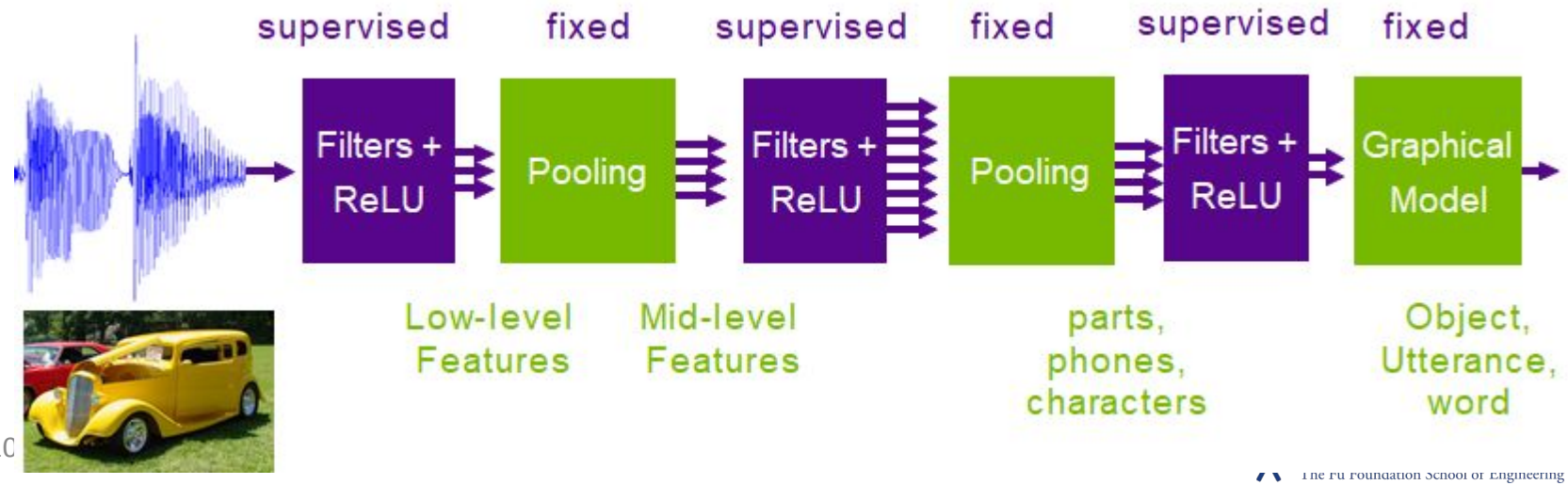
Pattern Recognition: Classical Architecture

- **Speech recognition: 1990-2011**
- **Object Recognition: 2005-2012**
- **Handwriting recognition (long ago)**
- **Graphical model has latent variables (locations of parts)**



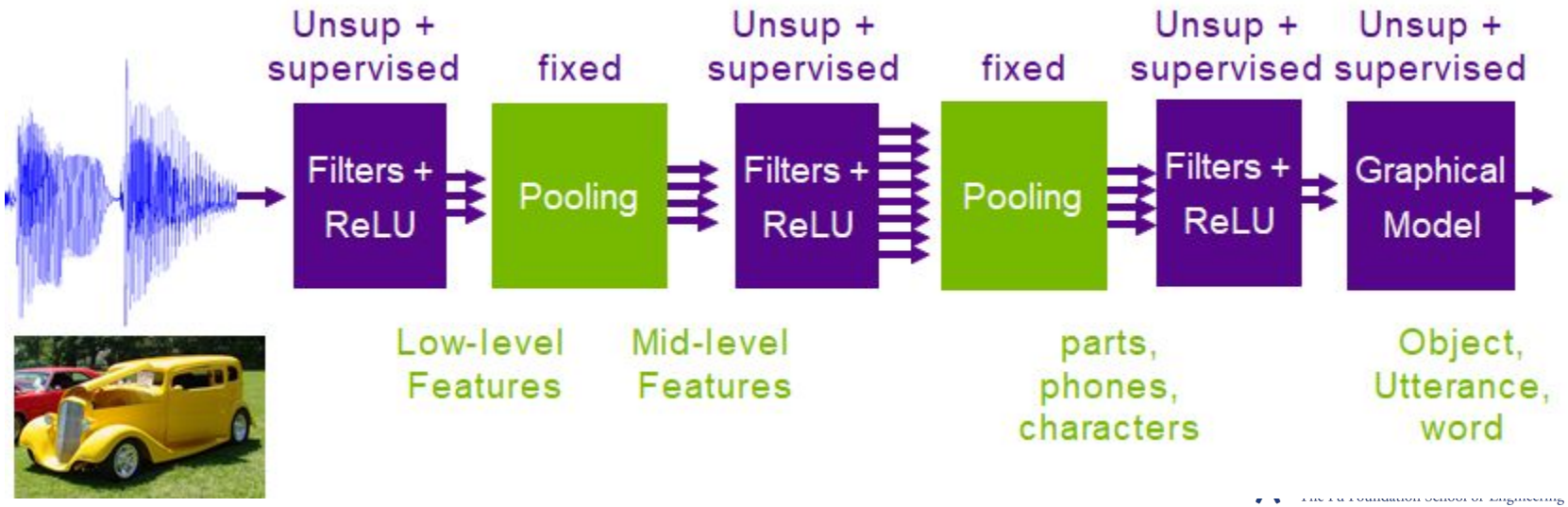
Pattern Recognition: Deep Architecture

- **Speech, and Object recognition: since 2011/2012**
- **Handwriting recognition: since the early 1990s**
- **Convolutional Net with optional Graphical Model on top**
- **Trained purely supervised**
- **Graphical model has latent variables (locations of parts)**



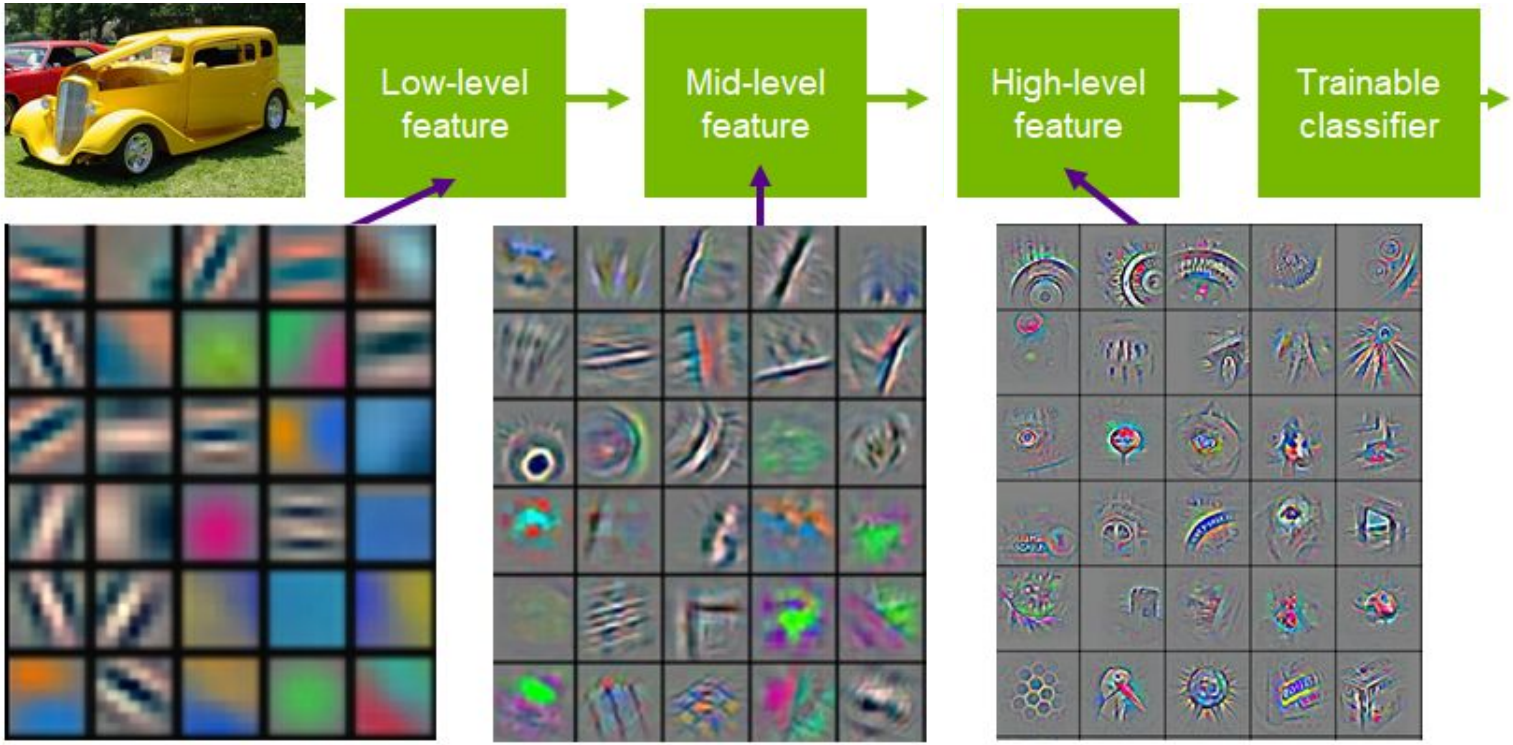
Future Systems: Deep Learning + Structured Prediction with Global Training

- **Handwriting recognition: since the mid 1990s**
- **Speech Recognition: since 2011**
- **All modules are trained with combination of unsupervised/supervised learning**
- **End-to-end training == deep structured prediction**



Deep Learning (of) Hierarchical Representations

- **Deep: if it has more than one stage of non-linear feature transformation**



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

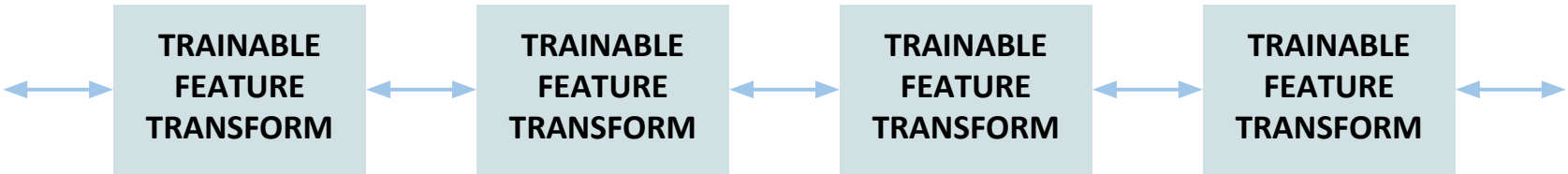
Deep Learning (Needs) Trainable Feature Hierarchy

Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable feature transform.

Image recognition: Pixel → edge → texton → motif → part → object

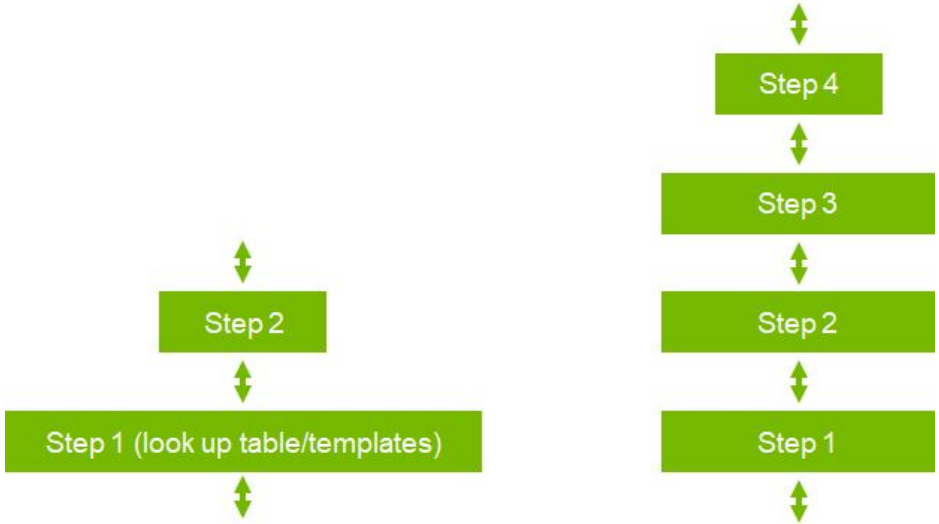
Text: Character → word → word group → clause → sentence → story

Speech: Sample → spectral band → sound → … → phone → phoneme → word



Shallow vs. Deep == Lookup Table vs. Multi-Step Algorithm

- **"Shallow & wide" vs "deep and narrow" == "more memory" vs "more time".**
- **Look-up table vs. algorithm.**
- **Few functions can be computed in two steps without an exponentially large lookup table.**
- **Using more than 2 steps can reduce the "memory" by an exponential factor.**



Discovering the Hidden Structure in High-Dimensional Data -> the Manifold Hypothesis

Learning representations of data:

Discovering & disentangling the independent explanatory factors.

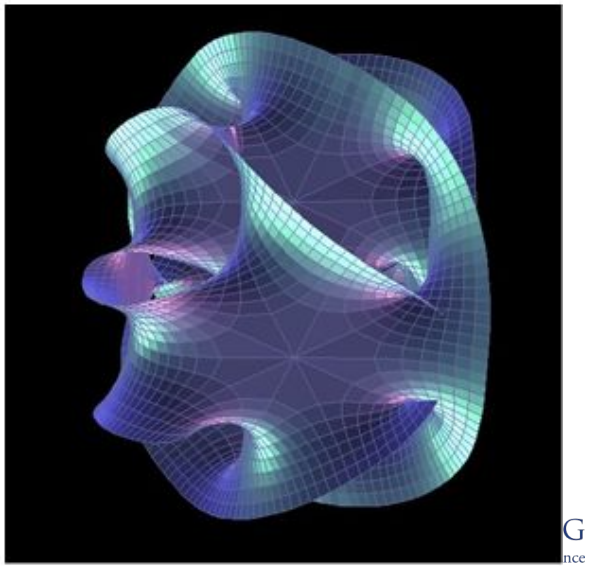
The manifold hypothesis:

Natural data lives in a low-dimensional

(non-linear) manifold.

Because variables in

natural data are mutually dependent.



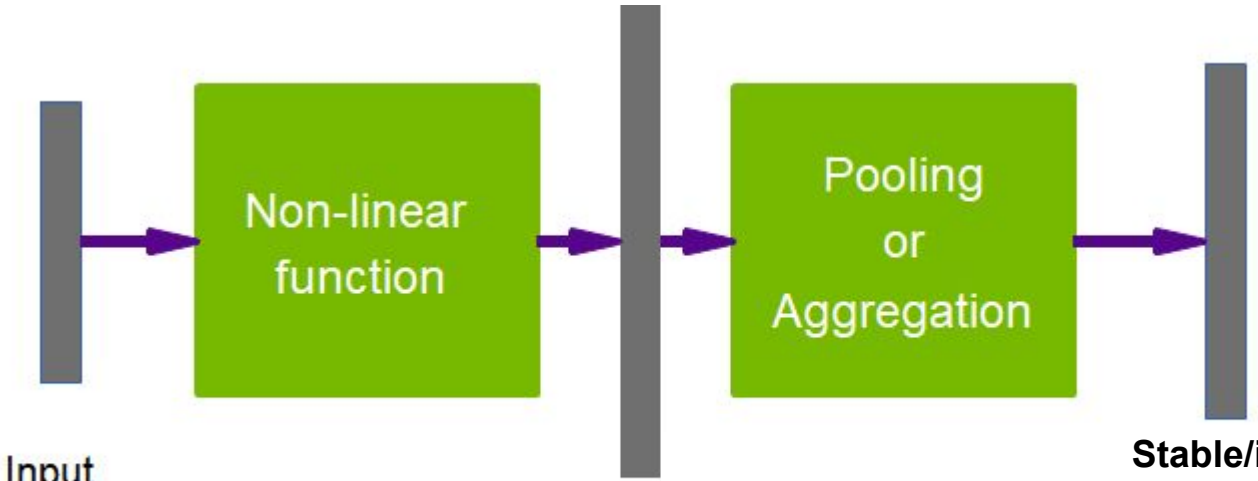
Basic Idea for Invariant Feature Learning

Embed the input non-linearly into a high(er) dimensional space.

- **In the new space, things that were non separable may become separable**

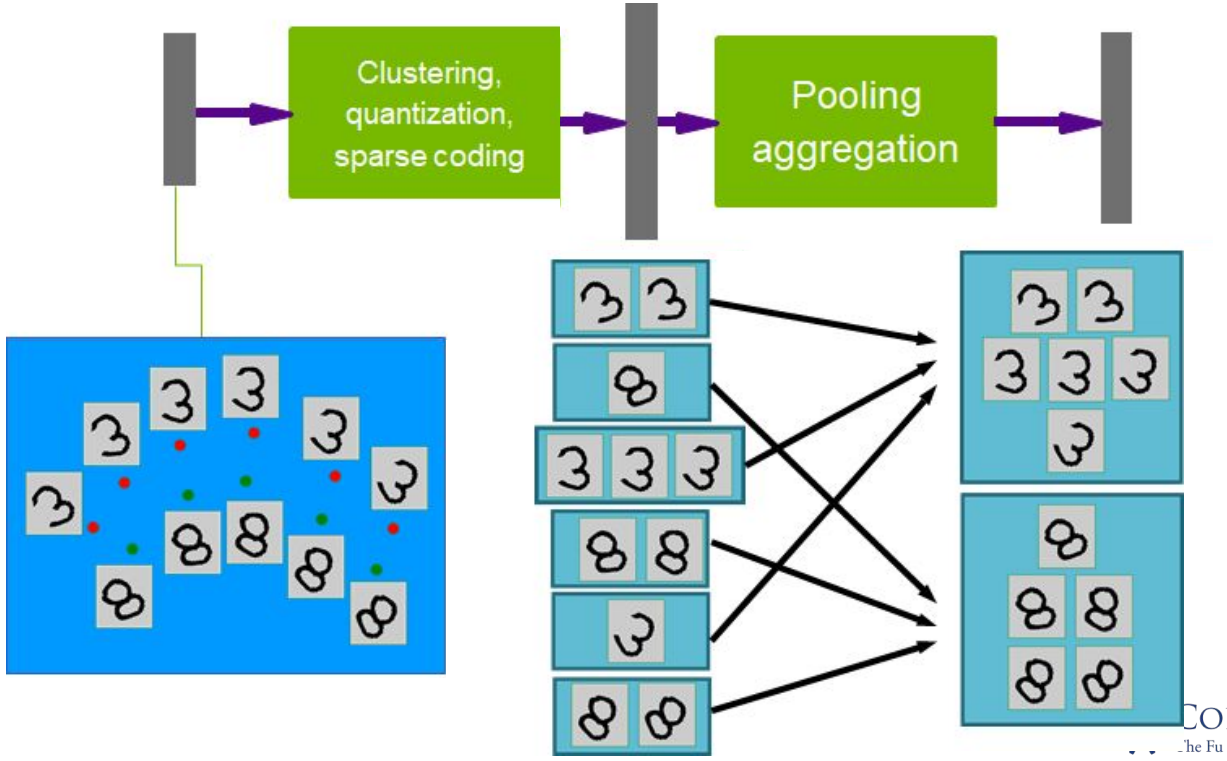
Pool regions of the new space together

- **Bringing together things that are semantically similar. Like pooling.**

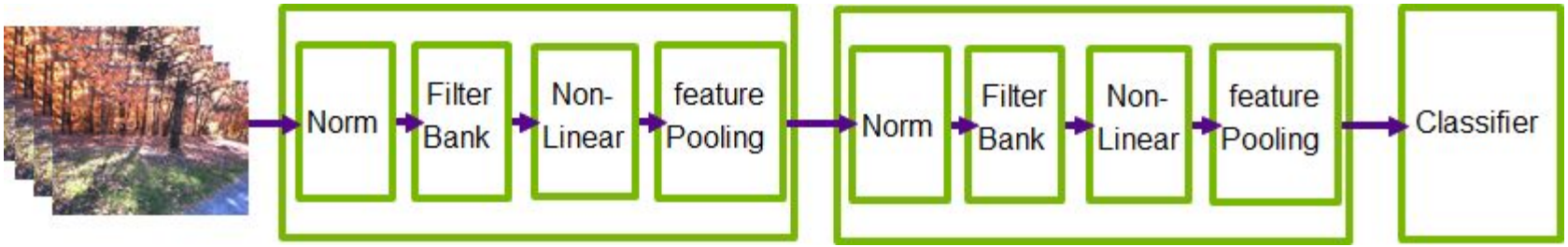


Sparse Non-Linear Expansion → Pooling

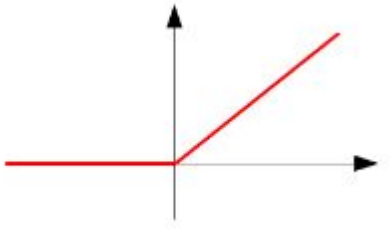
Use clustering to break things apart, pool together similar things



Overall ANN Architecture: Multiple Stages of Normalization→filter bank→non-linearity→ pooling



- **Normalization: variations on whitening**
  - **Subtractive: average removal, high pass filtering**
  - **Divisive: local contrast normalization, variance normalization**
- **Filter bank: dimension expansion, projection on overcomplete basis**
- **Non-linearity: sparsification, saturation, lateral inhibition....**
  - **Rectification (relu), component-wise shrinkage, tanh,..**
  - **ReLU (x)=max (x , 0)**
- **Pooling: aggregation over space or feature type**
  - **Max, Lp norm, log prob.**



$$X_i;\quad L_p:\sqrt[p]{X_i^p};\quad PROB:\frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

Learning Representations: A Challenge for ML, Computer Vision, AI, Neuroscience, Cognitive Sci.

How do we learn representations of the perceptual world?

- **How can a perceptual system build itself by looking at the world?**
- **How much prior structure is necessary**

ML/AI: how do we learn features or feature hierarchies?

- **What is the fundamental principle, the learning algorithm, the architecture?**

Neuroscience: how does the cortex learn perception?

- **Does the cortex "run" a single, general learning algorithm? (or a small number)**

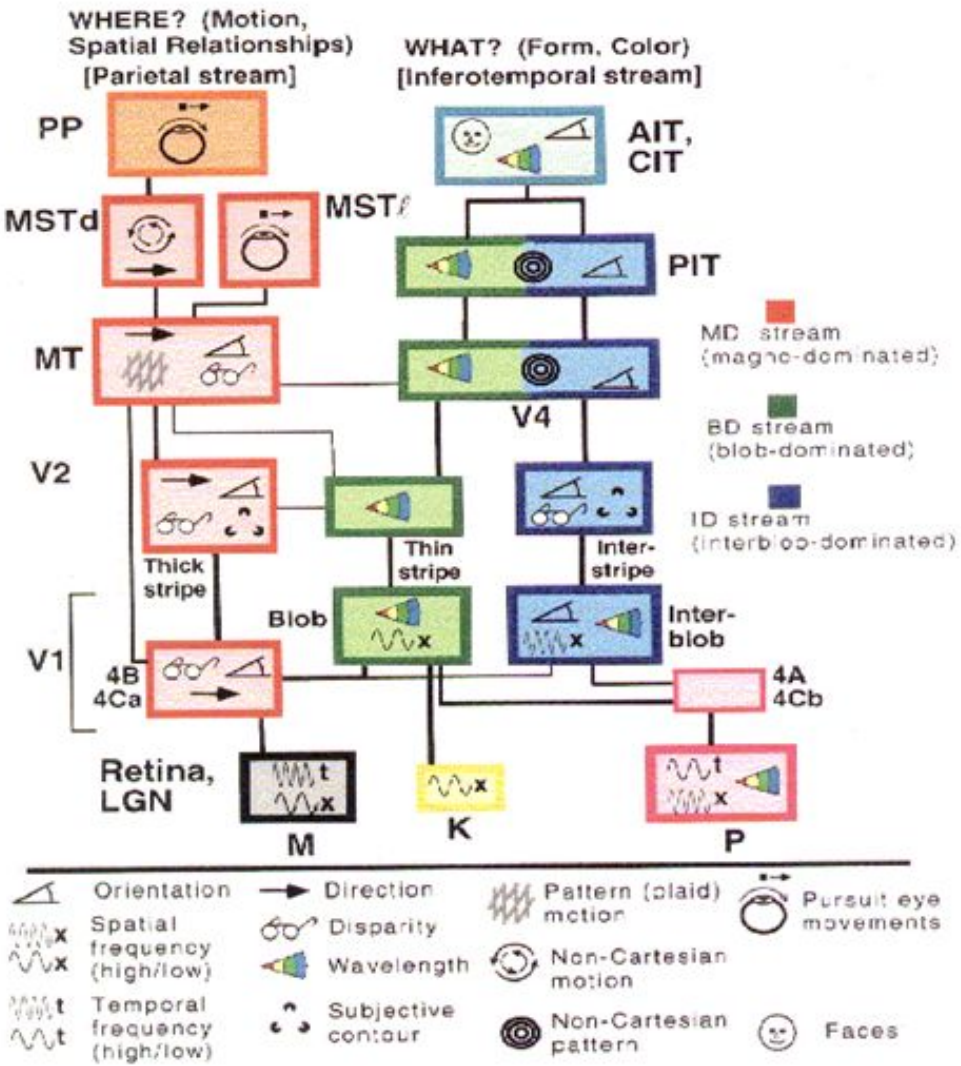
CogSci: how does the mind learn abstract concepts on top of less abstract ones?

Deep Learning addresses the problem of learning hierarchical representations with a single algorithm, or perhaps with a few algorithms.

Visual Cortex is Hierarchical

- Ventral pathway = “what”
- Dorsal pathway = “where”
- It's hierarchical
- There is feedback
- There is motion processing
- Learning is mostly unsupervised
- It does recognition, localization, navigation, grasping.....

[Gallant & Van Essen]



Deep Learning Inspired by Nature

It's nice to imitate Nature, but

- How do we know which details are important?
- Which details are merely the result of evolution, and the constraints of biochemistry?

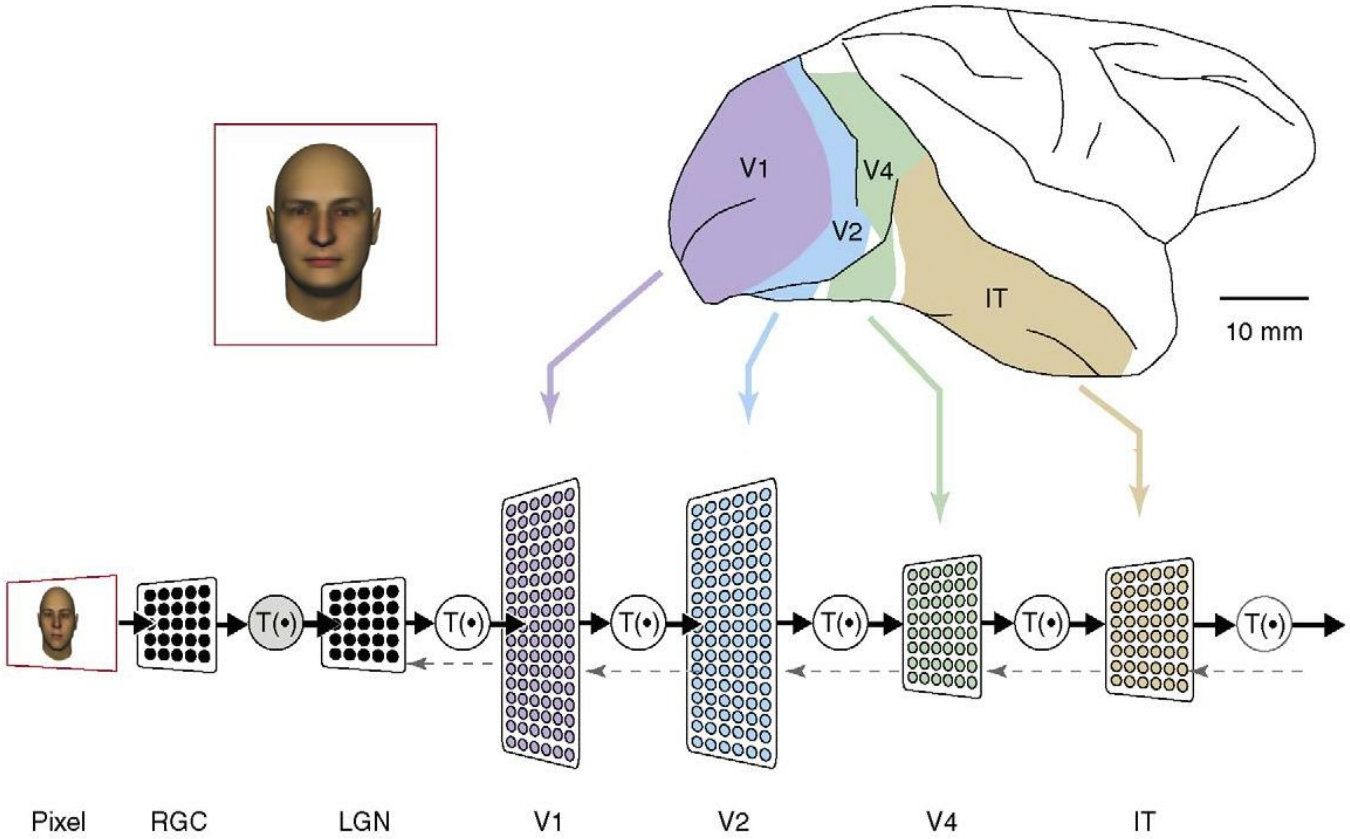
For airplanes, we developed aerodynamics and compressible fluid dynamics.

We figured that feathers and wing flapping weren't crucial.

L'Avion III de Clément Ader, 1897

(Musée du CNAM, Paris)

His Eole took off from the ground in 1890, 13 years before the Wright Brothers, but you probably never heard of it.



RGC: Retinal Ganglion Cells; LGN: Lateral Geniculate Nucleus;
V1: Primary Visual Cortex; V2 and V4: Visual Areas; IT: Inferior Temporal.

The Brain
Basics of the Early Visual System

Neurophysiologists David Hubel and Torsten Wiesel collaborated in the late 50s and early 60s to determine many of the most basic facts about how the mammalian early visual system works.

Images are formed by light arriving in the eye and stimulating the retina, the light-sensitive tissue in the back of the eye.

The neurons in the retina perform preprocessing of the image and the RGCs represent the pre-processed information in the spike domain.

The image then passes through the optic nerve and a brain region called the lateral geniculate nucleus (LGN). LGN neurons project their axons into the primary visual cortex V1 located at the back of the head.

The Brain -> CNN Inspiration

A convolutional network layer is designed to capture three properties of the primary visual cortex V1:

- **V1 is arranged in a spatial map (retinotopy); its 2-D structure mirrors the structure of the image on the retina. Convolutional networks capture this property by having their features defined in terms of 2-D maps.**
- **V1 contains many simple cells; a simple cell's activity can be characterized by a linear function of the image in a small, spatially localized receptive field. The detector units of a convolutional network are designed to emulate the local properties of simple cells.**

The Brain -> CNN Inspiration

A convolutional network layer is designed to capture three properties of the primary visual cortex V1:

- **V1 also contains many complex cells; complex cells are invariant to small shifts in the position of the feature. CNN pooling units attempt to address small shift invariance. Complex cells are also invariant to some changes in lighting that have inspired some of the cross-channel pooling strategies in CNNs, such as maxout units.**

As we repeatedly apply basic strategy of detection followed by pooling and move deeper into the brain, we eventually find cells that respond to some specific concept and are invariant to many transformations of the input. These cells have been nicknamed grandmother cells.

The Brain
Grandmother Cells

The hypothesis is that a person could have a neuron that activates when seeing an image of their grandmother,

regardless of whether she appears on the left or right side of the image, whether the image is a close-up of her face or zoomed out shot of her entire body, whether she is brightly lit, or in shadow, etc.

These grandmother cells have been shown to actually exist in the human brain, in a region called the medial temporal lobe.

These medial temporal lobe neurons/circuits are somewhat more general than modern convolutional networks, which would not automatically generalize to identifying a person or object when reading its name.

The Brain: Differences between CNNs and Mammalian Visual Systems (MVS)

Many differences between CNNs and MVS:

- **The retina is mostly very low resolution except for a tiny patch called the fovea.**
  - **Though we feel as if we can see an entire scene in high resolution, this is an illusion created by our visual cortex, as it stitches together several glimpses of small areas.**
  - **Most CNNs receive large full resolution photographs as input.**
- **The human visual system is integrated with many other senses, such as hearing, and factors like our moods and thoughts.**
  - **Convolutional networks are purely visual.**

The Brain: Differences between CNNs and Mammalian Visual Systems

Many differences:

- **The human visual system does much more than just recognize objects.**
  - **It is able to understand entire scenes including many objects and relationships between objects,**
  - **and processes rich 3-D geometric information needed for our bodies to interface with the world.**
- **CNNs have been applied to some of these problems but these applications are in their infancy.**

The Brain: Differences between CNNs and Mammalian Visual Systems

Many differences:

- **V1 is heavily impacted by feedback from higher brain centers.**
  - **Feedback has been explored extensively in neural network models but has not yet been shown to offer a compelling improvement.**
- **While feed-forward IT firing rates capture much of the same information as convolutional network features, it's not clear how similar the intermediate computations are.**
- **The brain probably uses very different activation and pooling functions.**

The Brain: Differences between CNNs and Mammalian Visual Systems

Many differences:

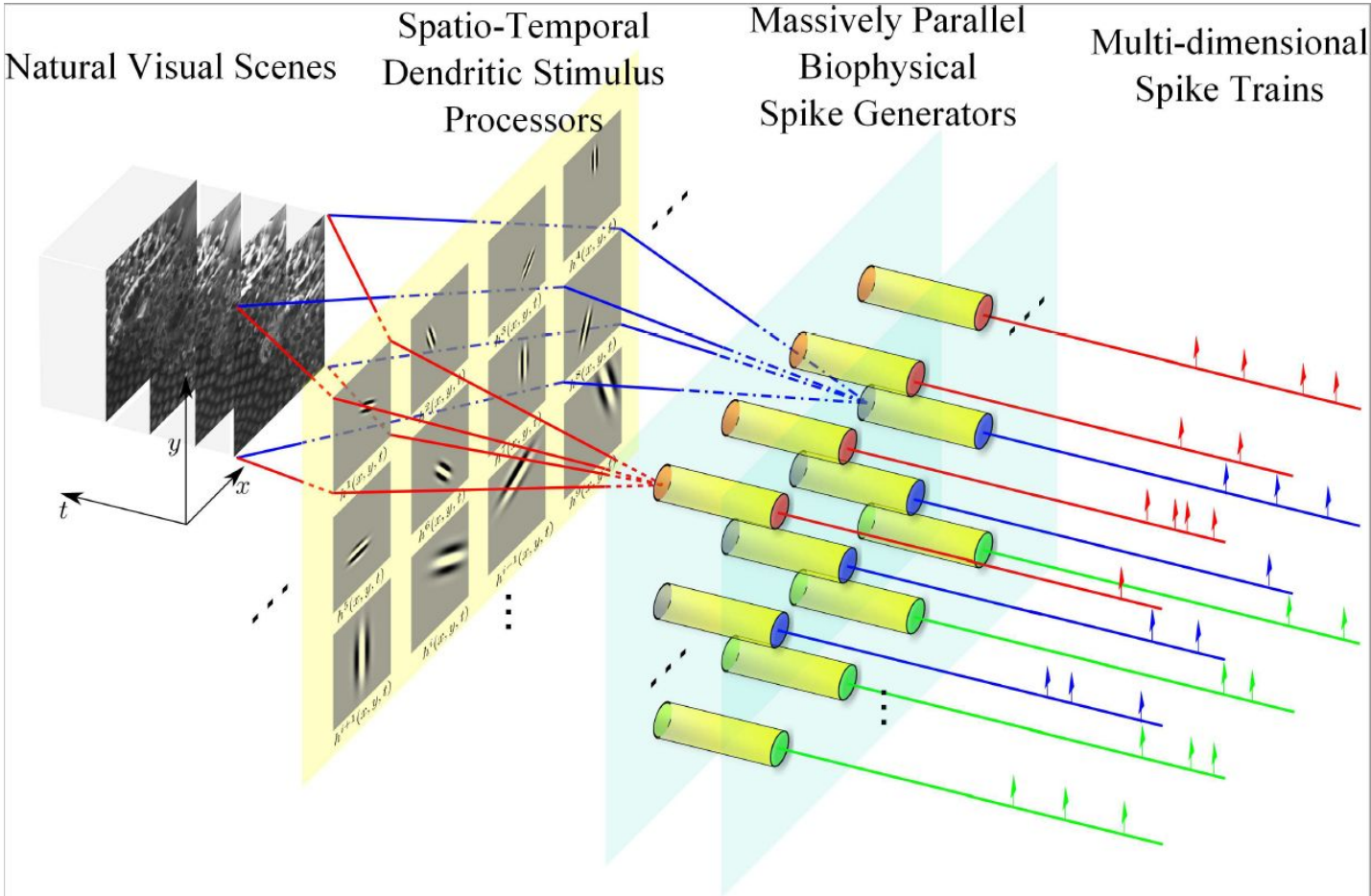
- **An individual neuron's activation probably is not well characterized by a single linear filter response.**
- **Our cartoon picture of simple cells and complex cells might create a nonexistent distinction;**
  - **Simple cells and complex cells might both be the same kind of cell**
    - **but with their "parameters" enabling a continuum of behaviors ranging from what we call "simple" to what we call "complex".**

CNNs and Neuroscience

Neuroscience has told us relatively little about how to train convolutional networks.

- **Back-propagation to train 1-D convolutional networks applied to time series was first introduced in 1988 (Hinton).**
- **Back-propagation applied to these models was not inspired by any neuroscientific observation and is considered to be biologically implausible.**
- **In 1989 (LeCun) developed the modern convolutional network by applying the same training algorithm to 2-D convolution applied to images.**

A Model of Neural Encoding in the Primary Visual Cortex



CNNs with Gabor Filters

Math Capturing the Functionality

We can think of an image as being a function of 2-D coordinates, $I(x, y)$. Likewise, we can think of a simple cell as sampling the image at a set of locations, defined by a set of x coordinates $\mathbb{X}$ and a set of y coordinates, $\mathbb{Y}$, and applying weights that are also a function of the location, $w(x, y)$. From this point of view, the response of a simple cell to an image is given by

$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y).$$

CNNs with Gabor Filters

Here $w(x, y)$ takes the form of a Gabor function:

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp(-\beta_x x'^2 - \beta_y y'^2) \cos(f x' + \phi),$$

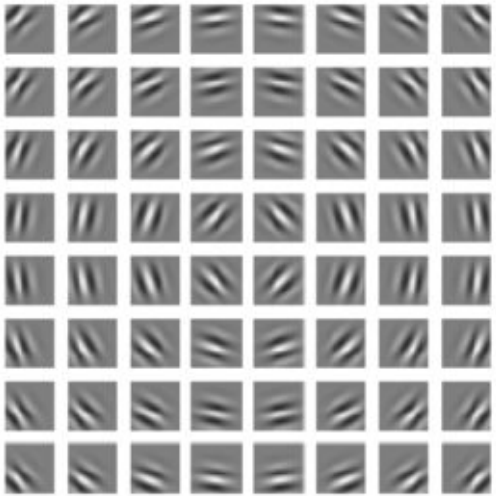
where

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

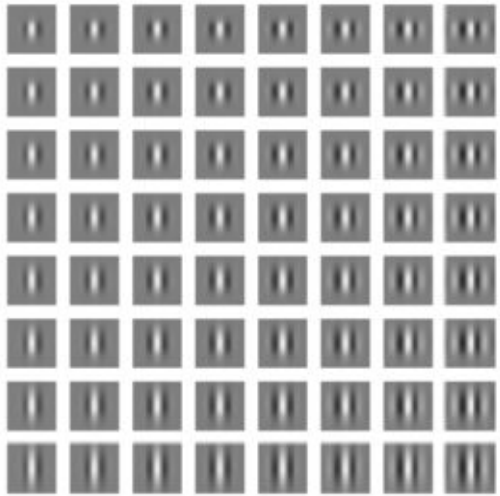
and

$$y' = -(x - y_0)\sin(\tau) + (y - y_0)\cos(\tau).$$

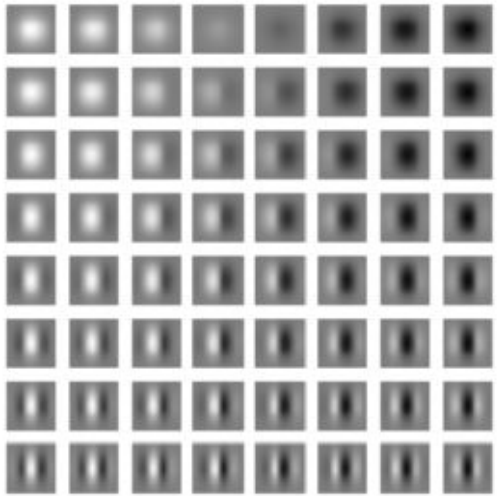
CNNs with Gabor Filters



Gabor functions with different parameter values (x_0, y_0) and τ.



Gabor functions with different Gaussian scale parameters β_x and β_y.



Gabor functions with different sinusoid parameters f and ϕ.

CNNs with Gabor Filters
Look Similar to the Response of Brain Cells

Cartoon view of a simple cell responding to a specific spatial frequency of brightness in a specific direction at a specific location:

- **Cells are most excited when the wave of brightness in an image has the same phase as the weights. This occurs when the image is bright where the weights are positive and dark where the weights are negative.**
- **Simple cells are most inhibited when the wave of brightness is fully out of phase with the weights—when the image is dark where the weights are positive and bright where the weights are negative.**

The cartoon view of a complex cell: it computes the L2 norm of the 2-D vector containing two simple cells' responses.

Convolutional Neural Networks (CNN) Basics

The Convolution Operation

Motivation

Pooling

The Convolution Operation

The convolution in a simplified form is defined by

$$s(t) = \int_{\mathbb{D}} x(a)w(t-a)da$$

and it is usually denoted by

$$s(t) = (x * w)(t).$$

Here x denotes the input, w denotes the kernel and s is the output, also referred to as the feature map. In discrete time notation

$$s[t] = (x * w)[t] = \sum_{a \in \mathbb{N}} x[a]w[t-a].$$

The Convolution Operation

The input is usually a multidimensional array of data and the kernel is usually a multidimensional array of learn-able parameters referred to as tensors. If the input is is a two-dimensional image I and K is a kernel is a two-dimensional kernel

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m, j-n].$$

The convolution operation is commutative, that is,

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n].$$

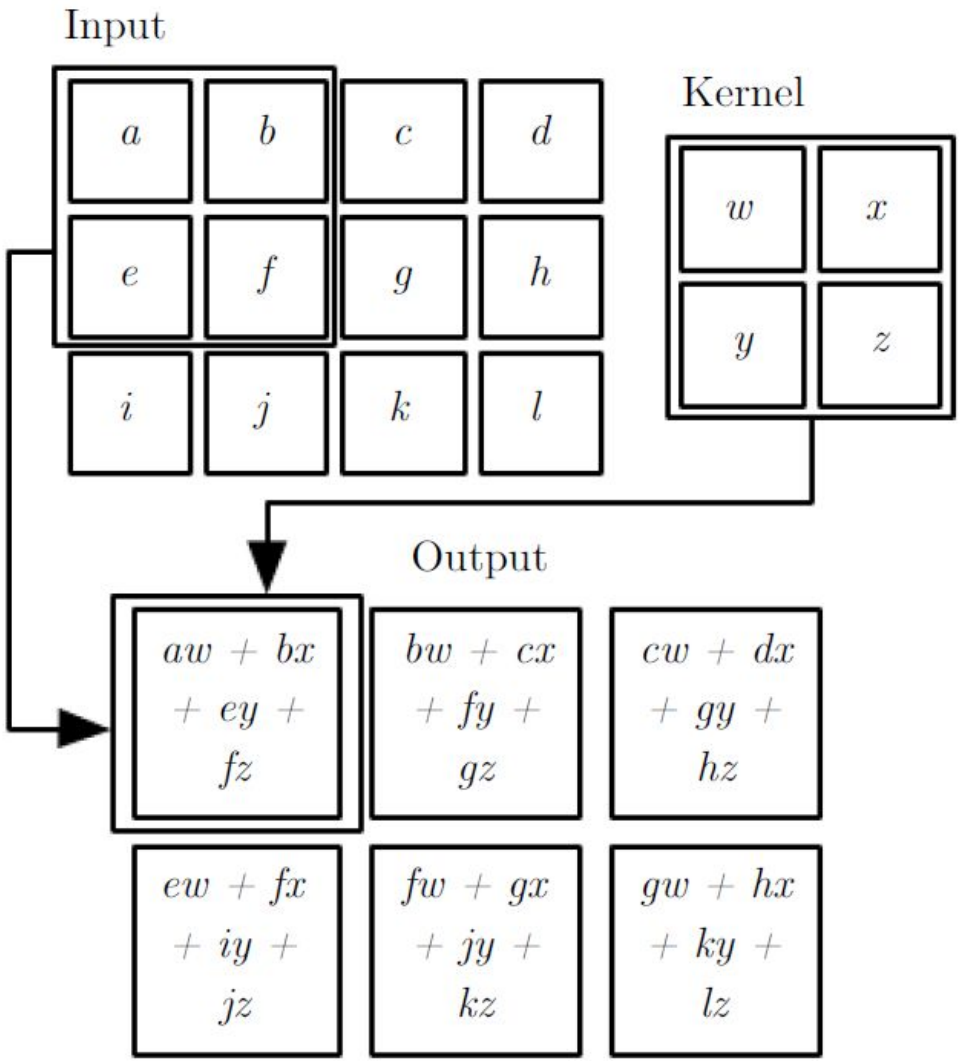
The Convolution Operation

Many neural network libraries implement the cross-correlation defined by (same as the convolution but without flipping the kernel)

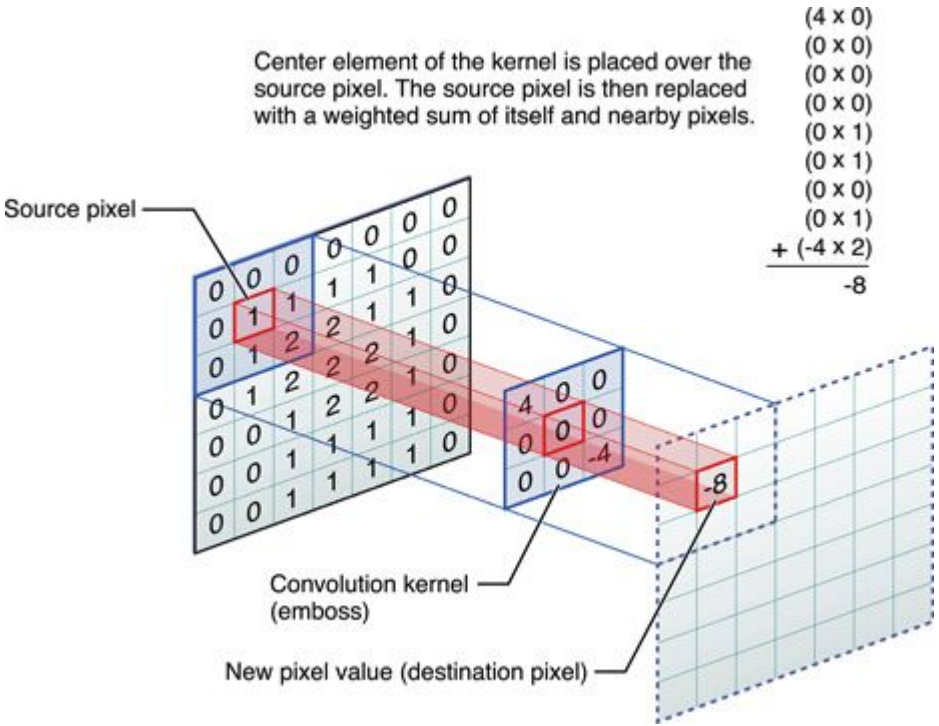
$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n] K[m,n].$$

Convolution Example

2-D Convolution without kernel flipping



Convolution Example



https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html

The Convolution Operation

Illustrative examples for basic convolution:

- https://en.wikipedia.org/wiki/Convolution
- http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/kernelmaskoperation/
- http://mathworld.wolfram.com/Convolution.html
- http://www.ti.com/lit/an/snaa080/snaa080.pdf
- https://lpsa.swarthmore.edu/Convolution/ConvolveGUI.html

Convolution in Neural Networks - Motivation

Convolution leverages three concepts that substantially improve machine learning:

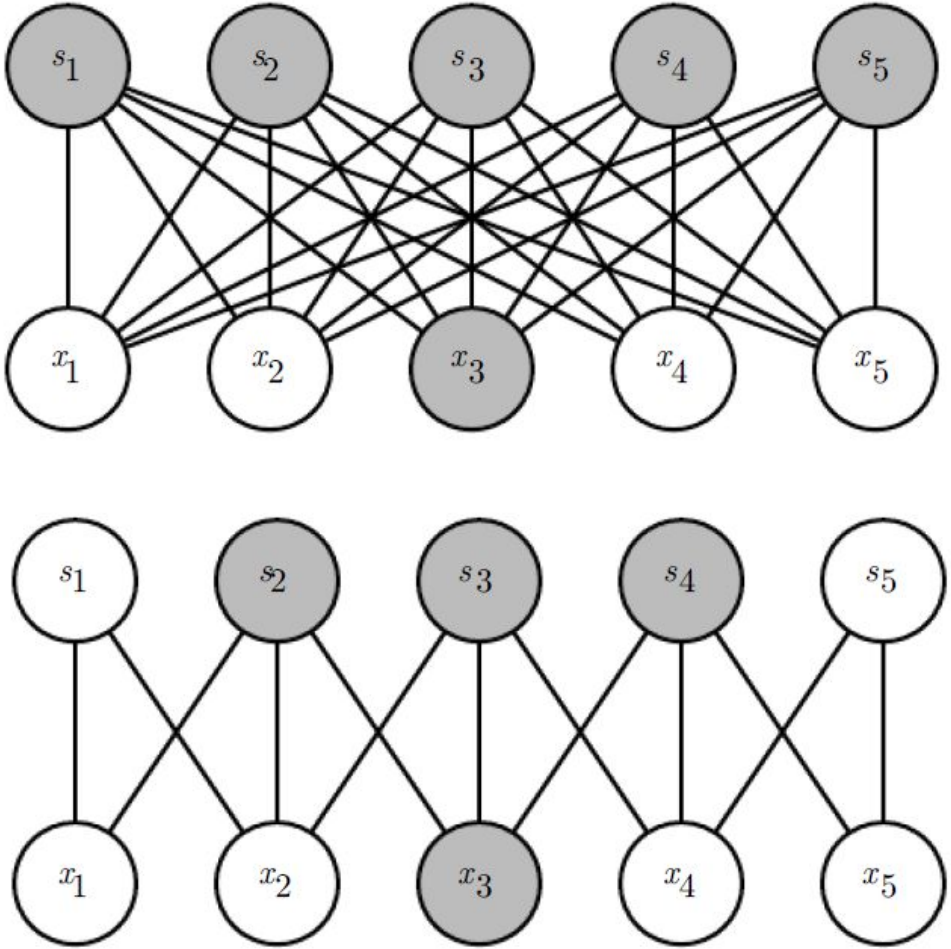
- sparse connectivity,
- parameter sharing,
- equivariant representation.

Convolution provides a means for working with inputs of variable size.

Convolutional Networks Have Sparse Connectivity

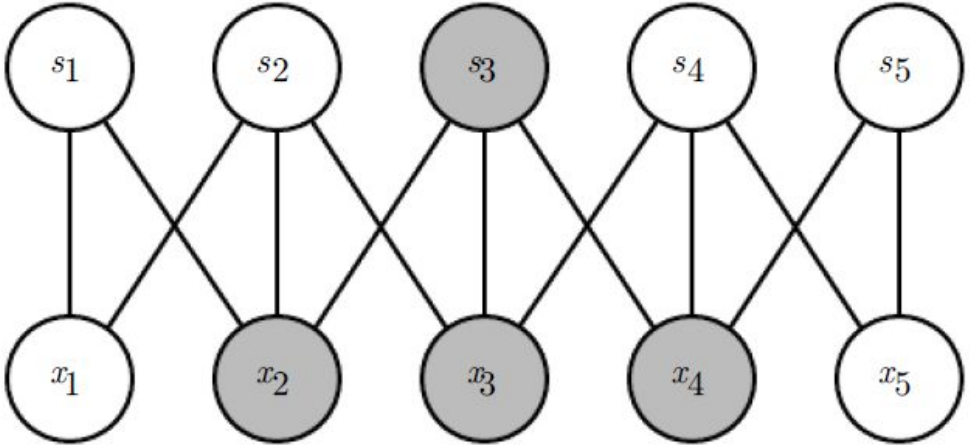
Matrix multiplication (top) with full connectivity.

Convolution with a kernel highlights sparse connectivity.

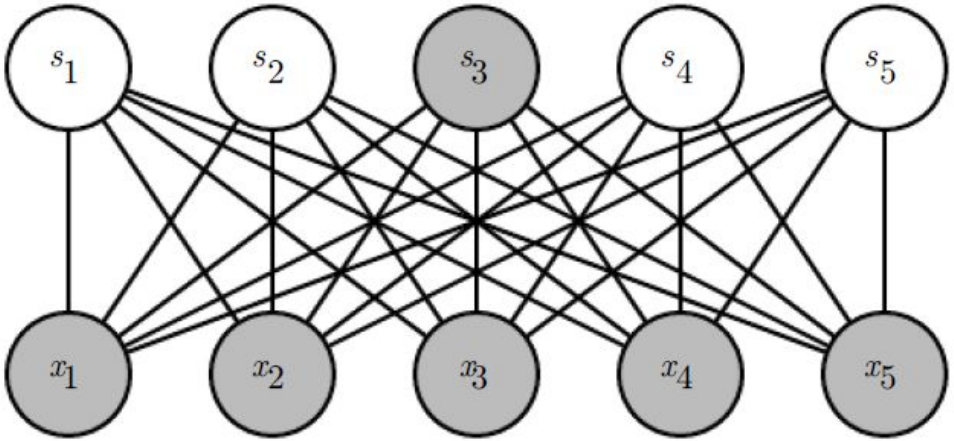


Convolutional Networks
Receptive Fields

Top: s_3 receives 3 inputs.

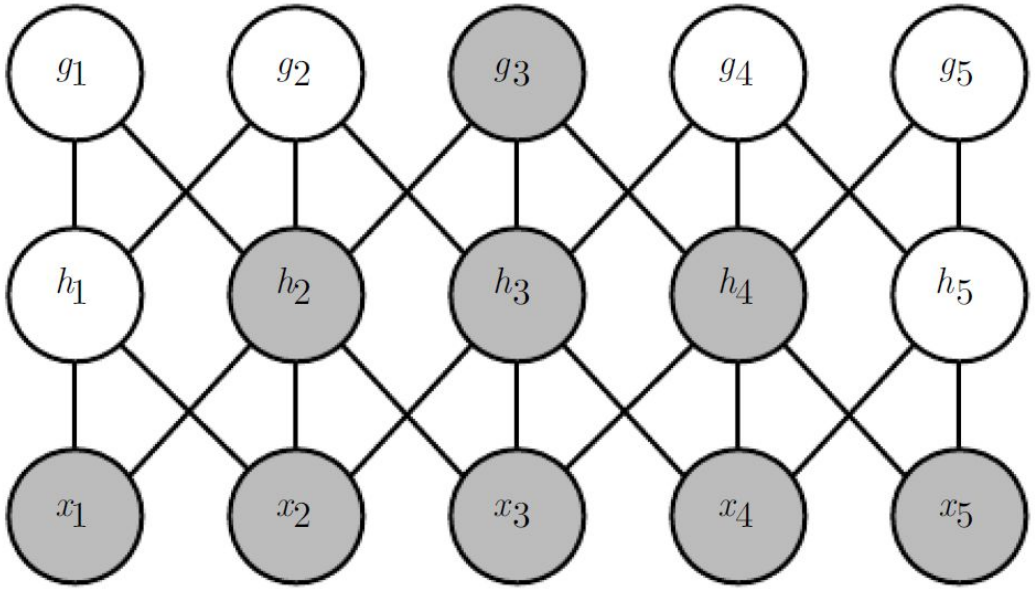


Bottom: s_3 receives full input



Convolutional Network
Complex Interactions - Receptive Field

The receptive field of the units in the deeper layers of a convnet is larger than the receptive field of the units in the shallow layers.



Convolutional Networks
Parameter Sharing

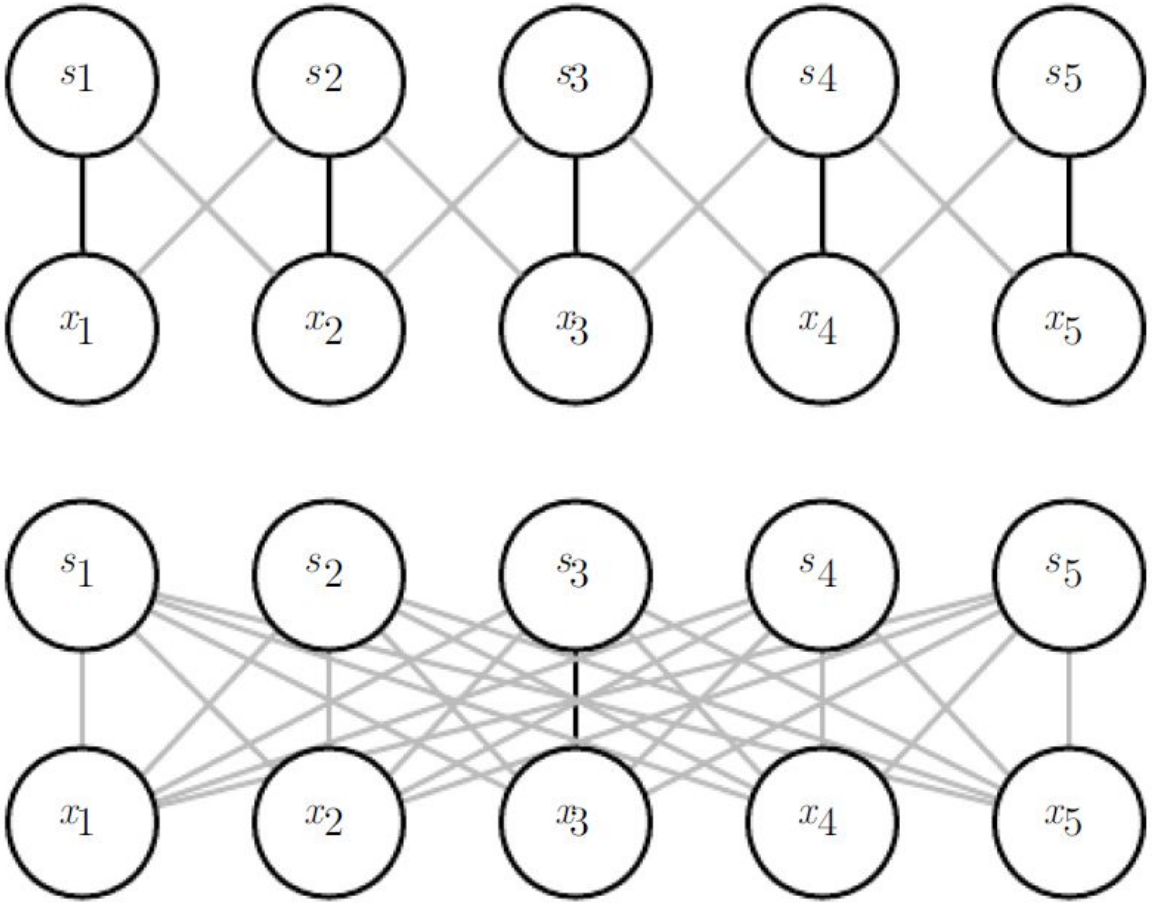
In a convnet,

each member of the

kernel is used at

every position

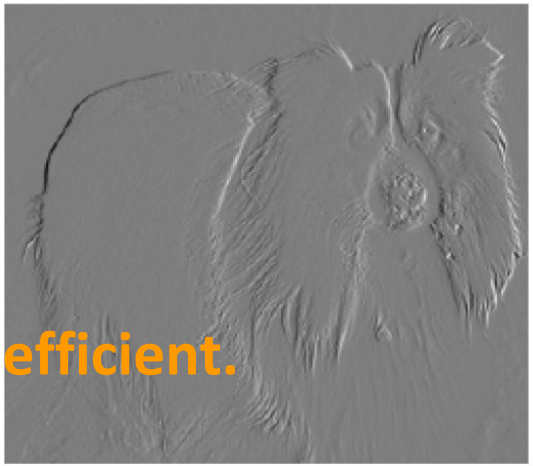
of the input.



Convolutional Networks- Parameter Sharing
Efficient Edge Detection

Input image has

280 × 320 pixels.



Convolution is 4 billion times more efficient.

Matrix multiplication operations for edge detection require 320 × 280 × 319 × 280, or over 8 billion matrix entries.

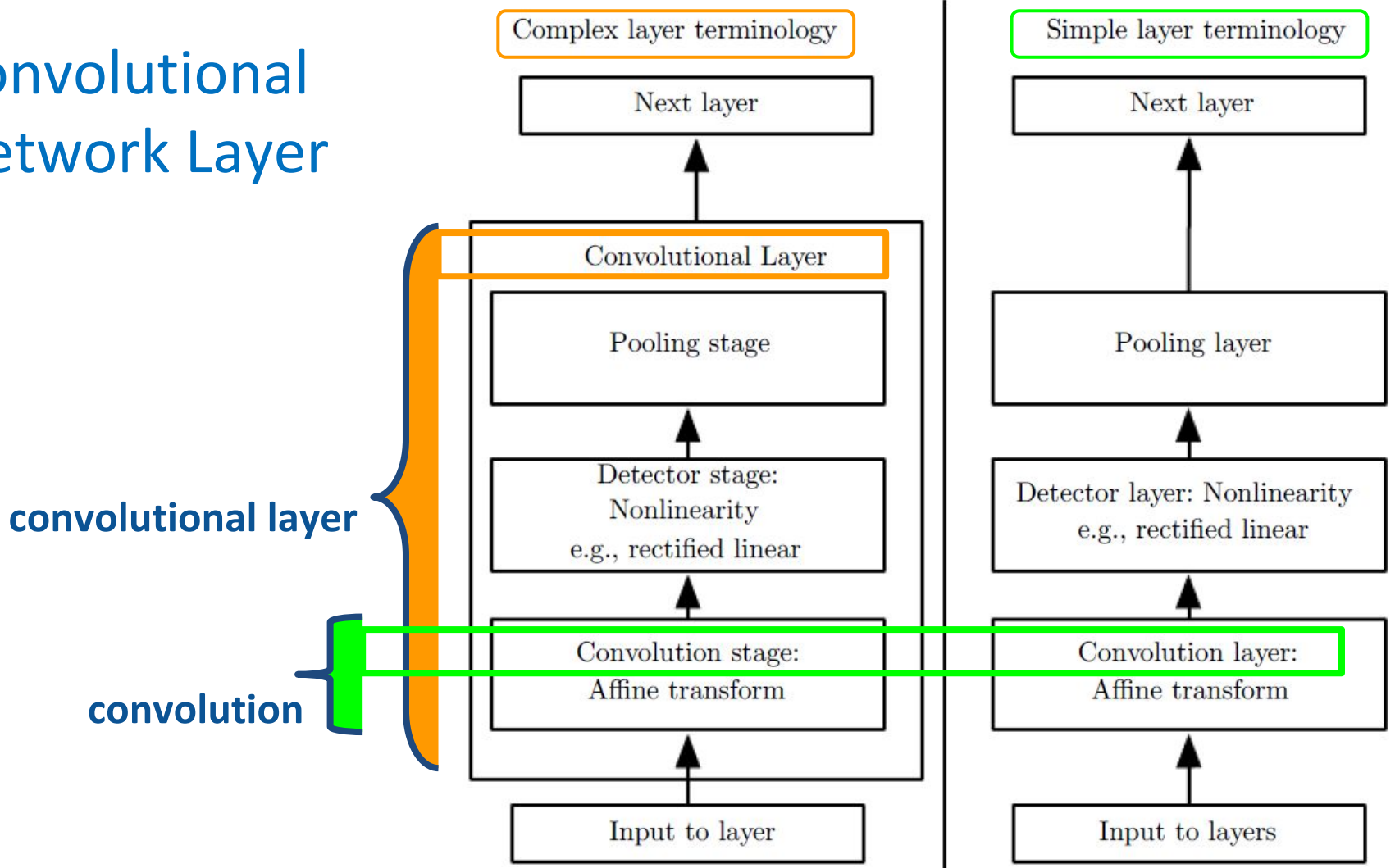
Convolution kernel has 2 elements and requires 319 × 280 × 3 = 267, 960 floating point operations (2 multiplications and 1 addition per output pixel) to compute.

Convolutional Network Layer

Convolution (as an affine transformation)

Pooling

Convolutional Network Layer



Convolutional Network Layer
Pooling

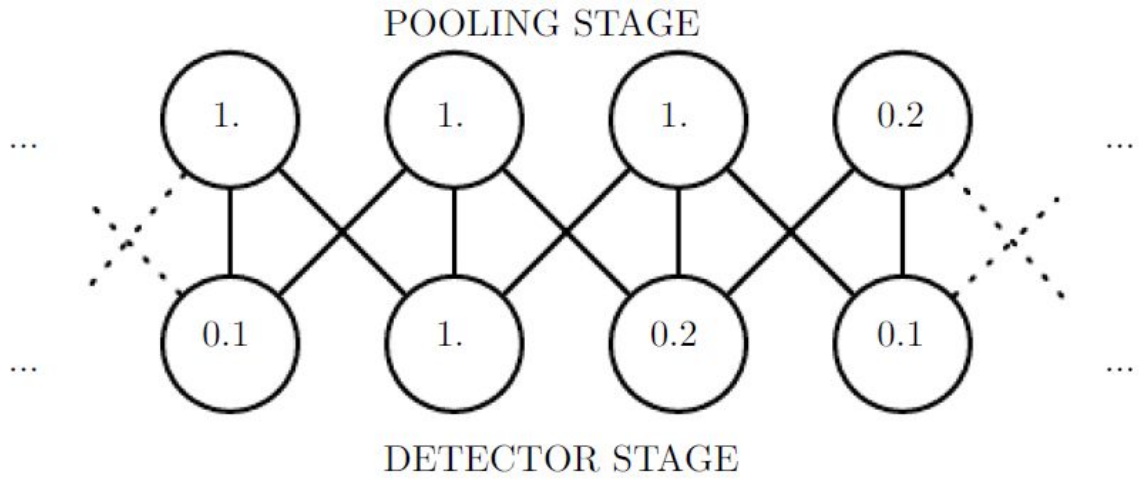
A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs:

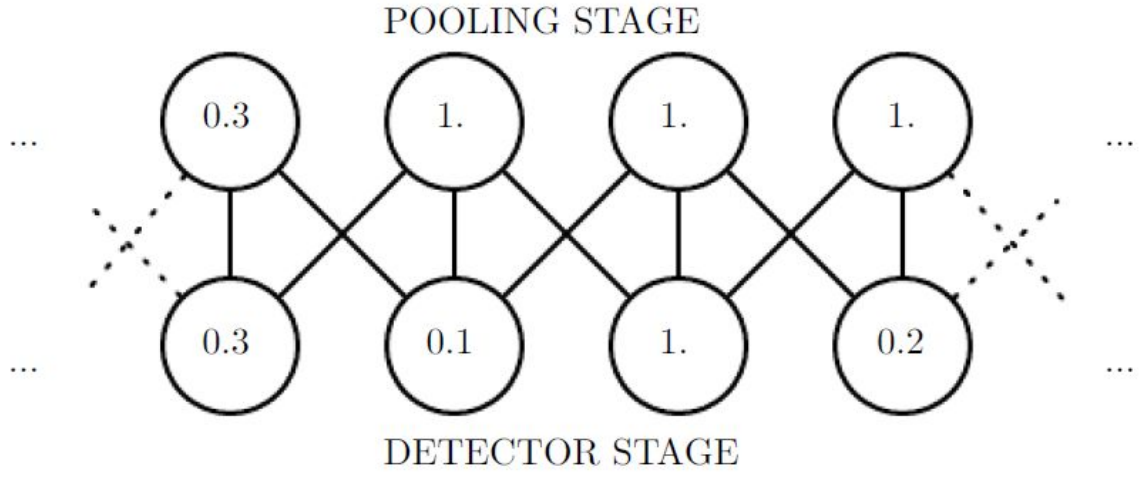
- **max pooling reports the maximum output within a rectangular neighborhood;**
- **the average of a rectangular neighborhood;**
- **the L2 norm of a rectangular neighborhood;**
- **a weighted average based on the distance from the central pixel.**

Pooling can make the representation become invariant to small local translations of the input. Invariance is useful if we care more about whether some feature is present than exactly where it is.

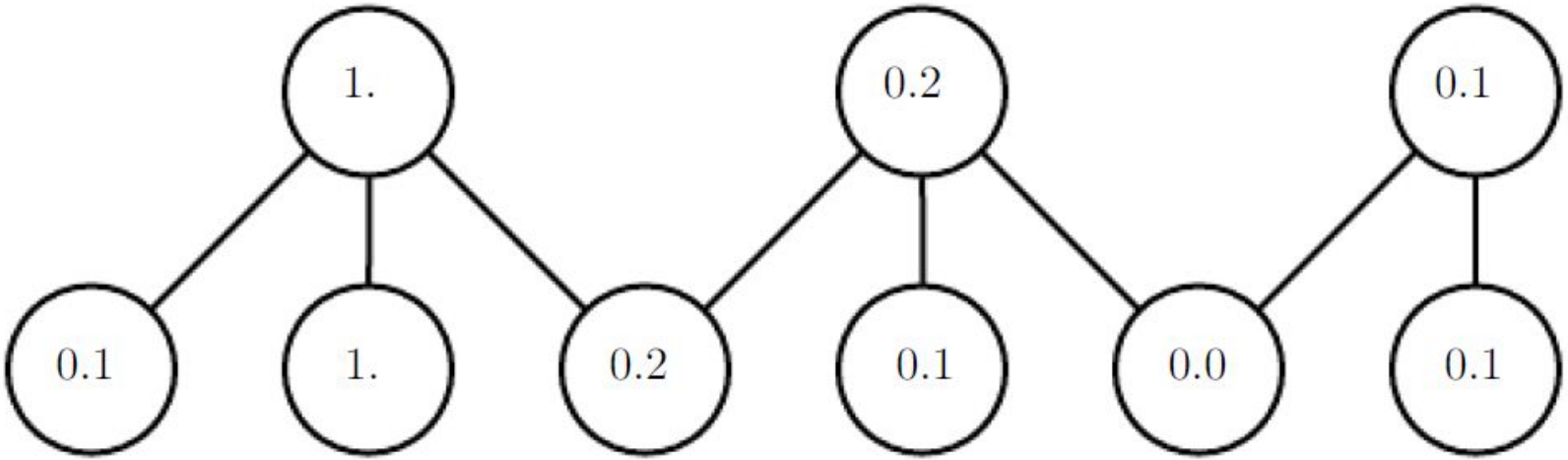
Convolutional Network Layer Pooling Invariance

Max pooling Introduces invariance with respect to small translations.





Convolutional Network Layer
Pooling with Downsampling



Max-pooling with a pool width of 3 and a stride between pools of 2 reduces the representation size by a factor of 2. The computational and statistical burden on the next layer is considerably reduced.

Convolutional Neural Networks (CNN) Variants

Multi-Dimensional

With Stride

With Zero-Padding

Unshared “Convolution”

Connectivity Restriction, Tiled Convolution

Convolution - Basic

Convolutional neural networks consist of many applications of convolution in parallel.

Convolution with a single kernel extracts a single feature at many locations. The input is typically a grid of vector-valued observations.

In a multilayer convolutional network, the input to the second layer is the output of the first layer, which usually has the output of many different convolutions at each position.

Convolution: Multi-Dimensional

When working with images, we usually think of the input and output of the convolution as being 3-D tensors,

- **with one index for the different channels and**
- **two indices for the spatial coordinates of each channel.**

Convolution: Multi-Dimensional

Assume we have a 4-D kernel tensor K with element K_{ijkl} giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit.

Assume our input consists of observed data V with element V_{ijk} giving the value of the input unit within channel i at row j and column k. Assume our output consists of Z with the same format as V. If Z is produced by convolving K across V without flippping K, then

$$Z_{ijk} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{ilmn},$$

where the summation over l, m and n is over all values for which the tensor indexing operations inside the summation is valid.

Convolution: Multi-Dimensional

We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as downsampling the output of the full convolution function. If we want to sample only every s pixels in each direction in the output, then we can define a downsampled convolution function c such that

$$Z_{ijk} = c(K, V, s)_{ijk} = \sum_{l,m,n} [V_{l,(j-1)s+m,(k-1)s+n} K_{ilmn}].$$

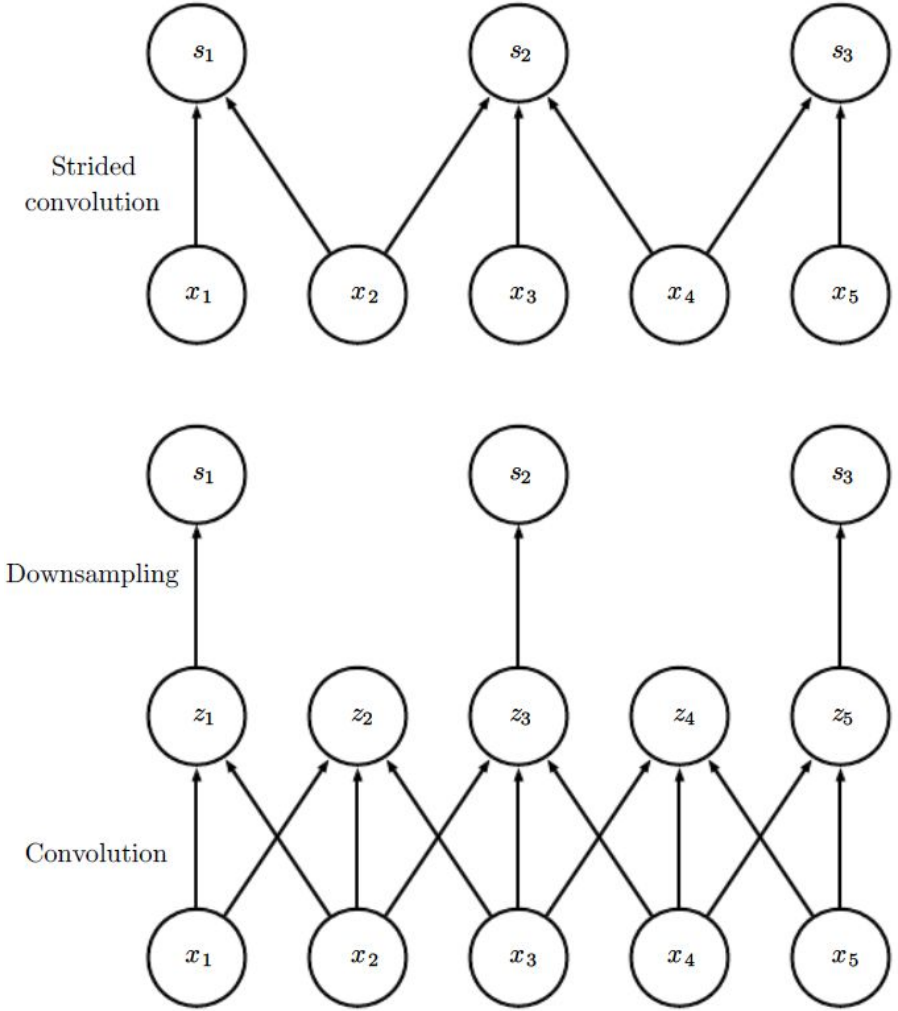
We refer to s as the stride of this downsampled convolution. It is also possible to define a separate stride for each direction of motion.

Convolution with a Stride

(Top) Convolution with a stride length of two implemented in a single operation.

(Bottom) Convolution with a stride greater than 1 pixel is equivalent to convolution with unit stride followed by downsampling.

2-step approach with downsampling is wasteful, because it computes values that are later discarded.



Convolution Variant - Zero Padding

An essential feature of any convolutional network implementation is the ability to implicitly zero-pad the input V in order to make it wider. Zero padding the input allows us to control the kernel width and the size of the output independently.

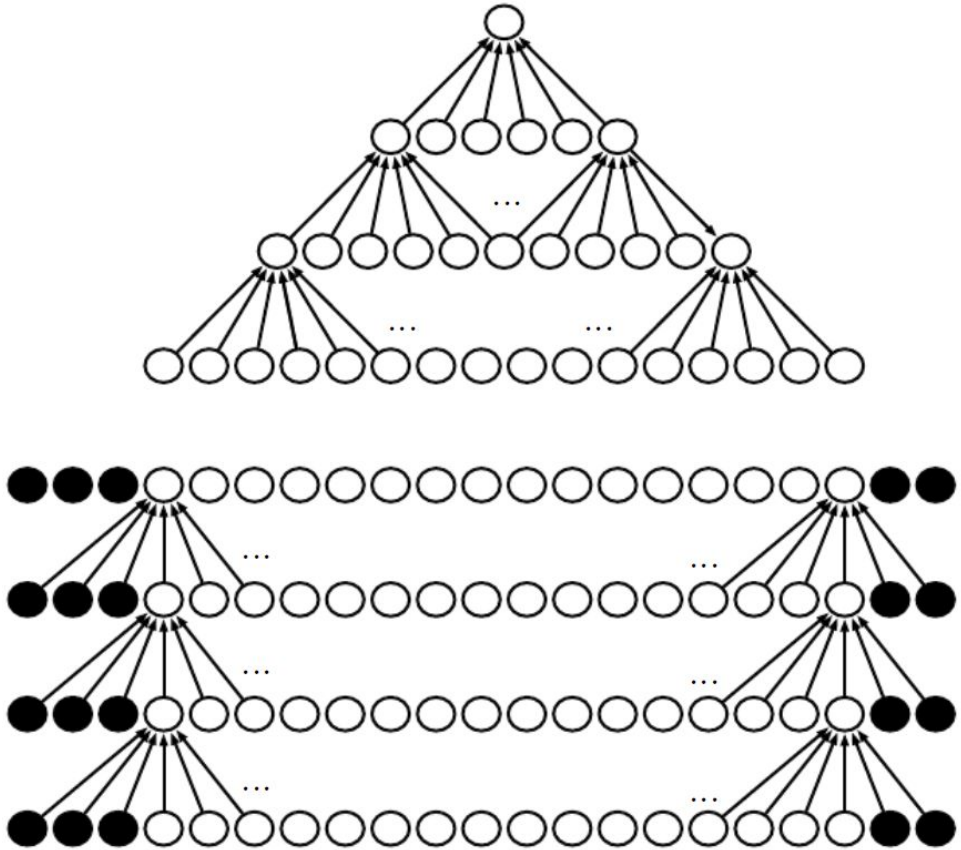
Three special cases of the zero-padding setting are worth mentioning:

- the extreme case in which no zero-padding is used whatsoever, and the convolution kernel is only allowed to visit positions where the kernel is contained entirely within the image.
- the case when just enough zero-padding is added to keep the size of the output equal to the size of the input.
- the case in which enough zeroes are added for every pixel to be visited k times in each direction, resulting in an output image of width $m + k - 1$.

Convolution Variants: Zero Padding & Network Size

(Top) No implicit zero padding. This causes the representation to shrink by 5 pixels at each layer. Starting from an input of 16 pixels, there are only three convolutional layers available.

(Bottom) By adding 5 implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.



Convolution Variants:
Unshared "Convolution"

In some cases, we do not actually want to use convolution, but rather locally connected layers -> quasi-convolution.

In this case, the adjacency matrix in the graph of MLP is the same, but every connection has its own weight, specified by a 6-D tensor **W**.

Convolution Variants: Unshared "Convolution"

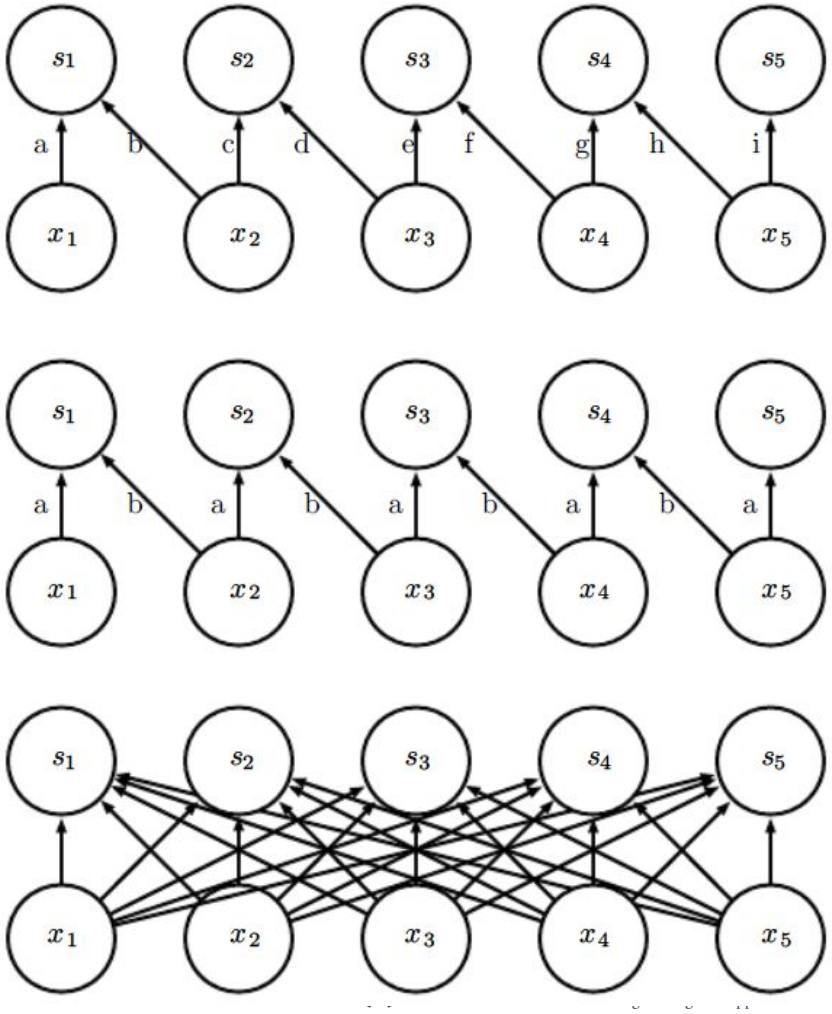
The indices into W are respectively: i, the output channel, j, the output row, k, the output column, l, the input channel, m, the row offset within the input, and n, the column offset within the input. The linear part of a locally connected layer is then given by

$$Z_{ijk} = \sum_{l,m,n} [V_{l,j+m-1,k+n-1} w_{ijklmn}].$$

This is sometimes also called unshared convolution, because it is a similar operation to discrete convolution with a small kernel, but without sharing parameters across locations.

Convolution Variants: Unshared "Convolution"

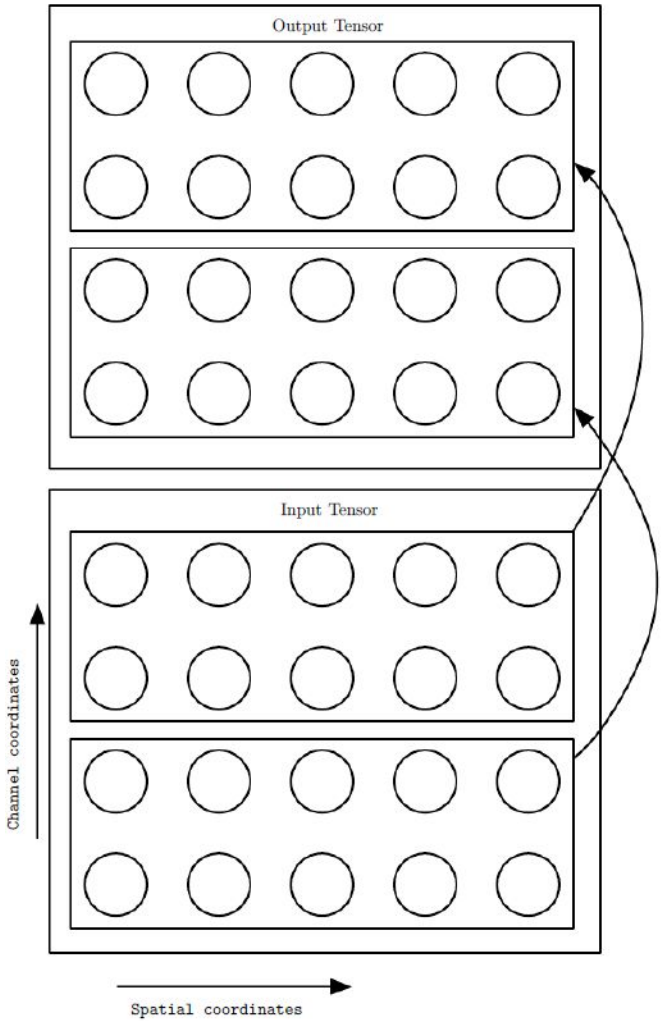
- **(Top) Locally connected layer with a patch size of two pixels.**
- **(Center) Convolutional layer with a kernel width of two pixels. Locally connected layer has no parameter sharing. The conv layer uses the same 2 weights repeatedly across entire input, indicated by the repetition of letters labeling each edge.**
- **(Bottom) A fully connected layer resembles a locally connected layer in that each edge has its own parameter. However, it does not have the restricted connectivity of the locally connected layer**



Convolution Variants: Restricting Connectivity

A convolutional network

- **with the first two output channels connected to only the first two input channels, and**
- **the second two output channels connected to only the second two input channels.**

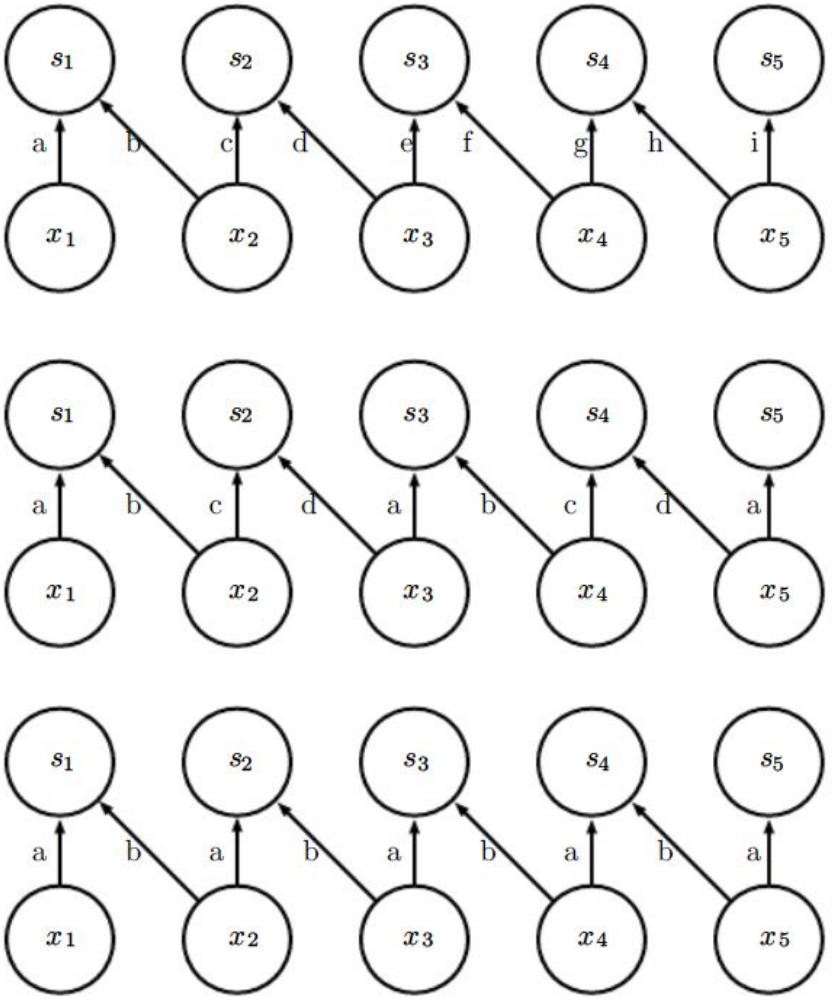


Convolution Variants: Tile Convolution

(Top) Locally connected layer has no sharing at all. Each connection has its own weights / label with a unique letter.

(Center) Tiled convolution with a set of t = 2 different kernels. One of these kernels has edges labeled a and b, while the other has edges labeled c and d. If two output units are separated by a multiple of t steps, then they share parameters.

(Bottom) Traditional convolution is equivalent to tiled convolution with t = 1. There is only 1 kernel applied everywhere with weight labels a and b.

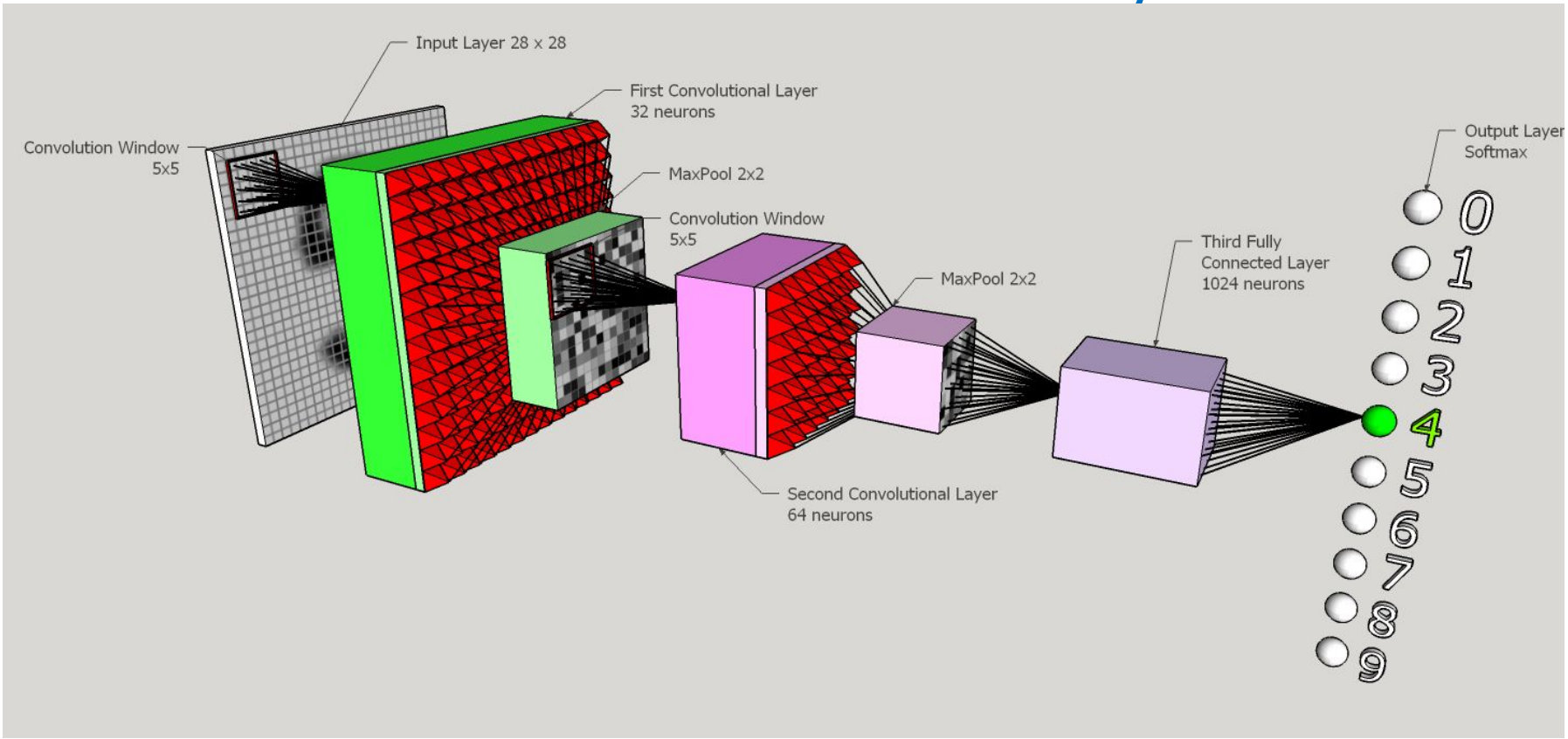


Convolutional Neural Networks (CNN)

Assembled Layers

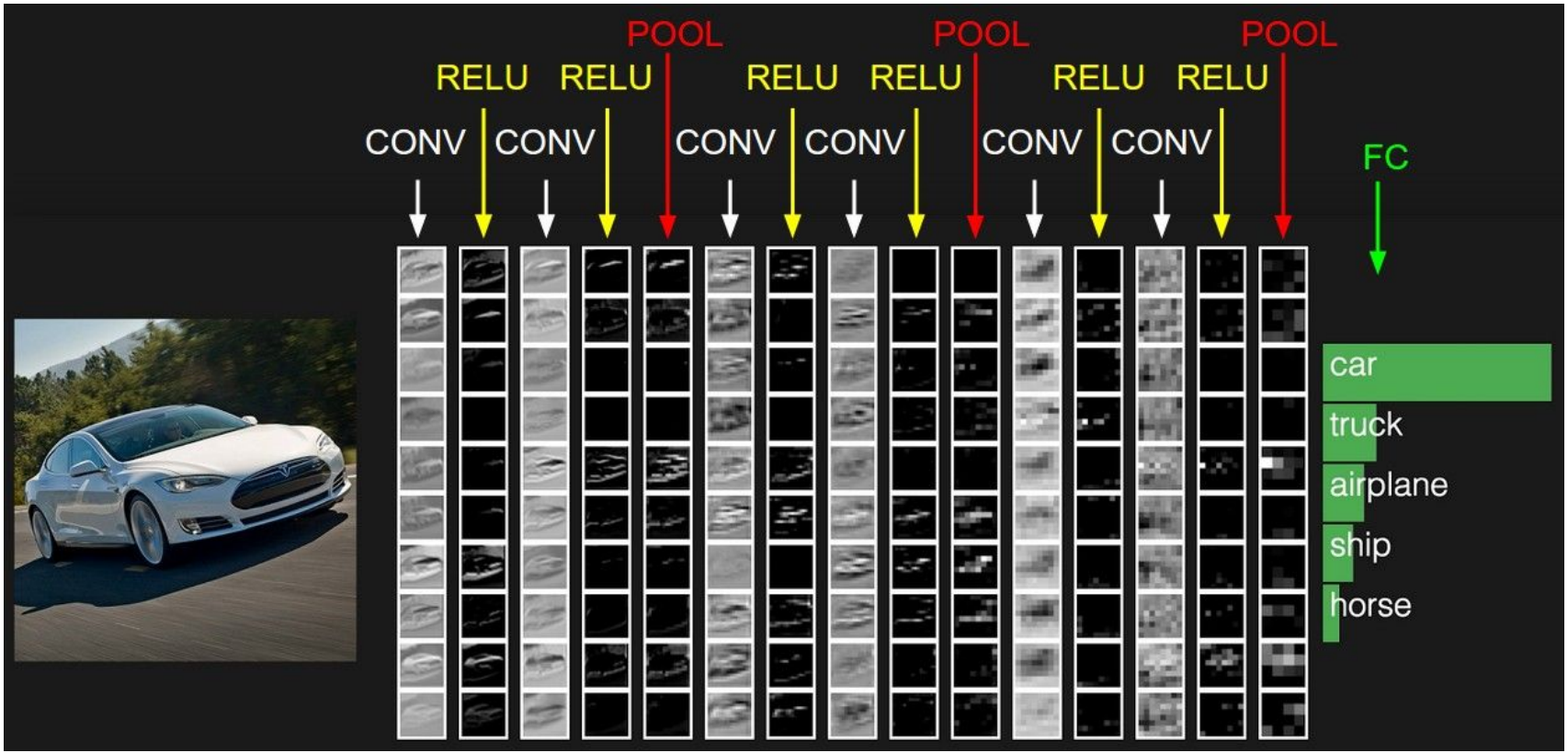
Assembling the Layers

Convolutional Network- Assembled Layers



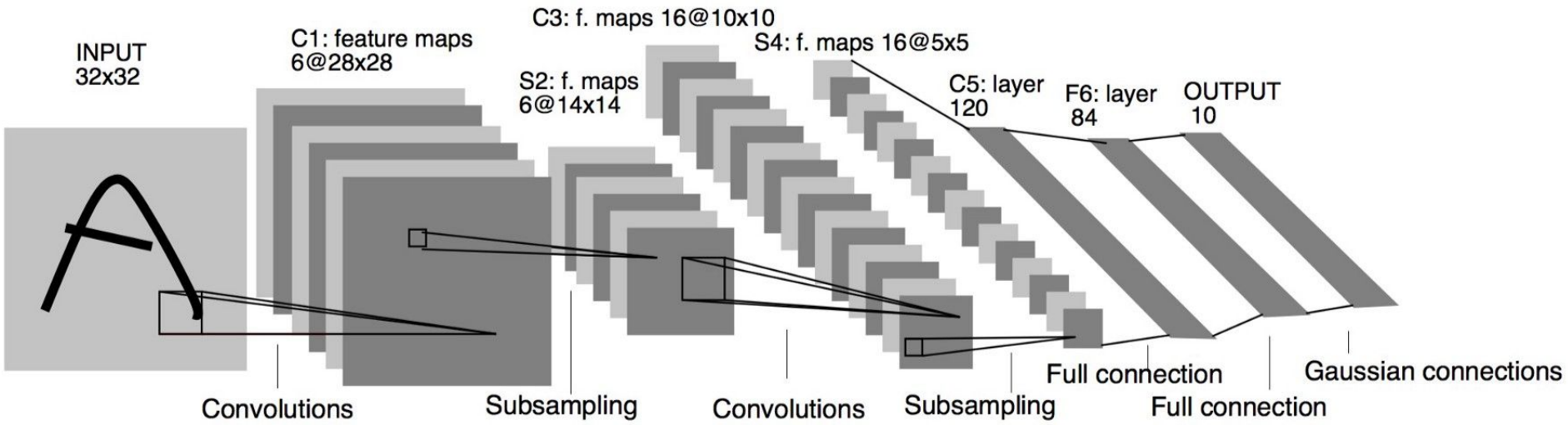
http://zderadicka.eu/revival-of-neural-networks/

Convolutional Network- Assembled Layers



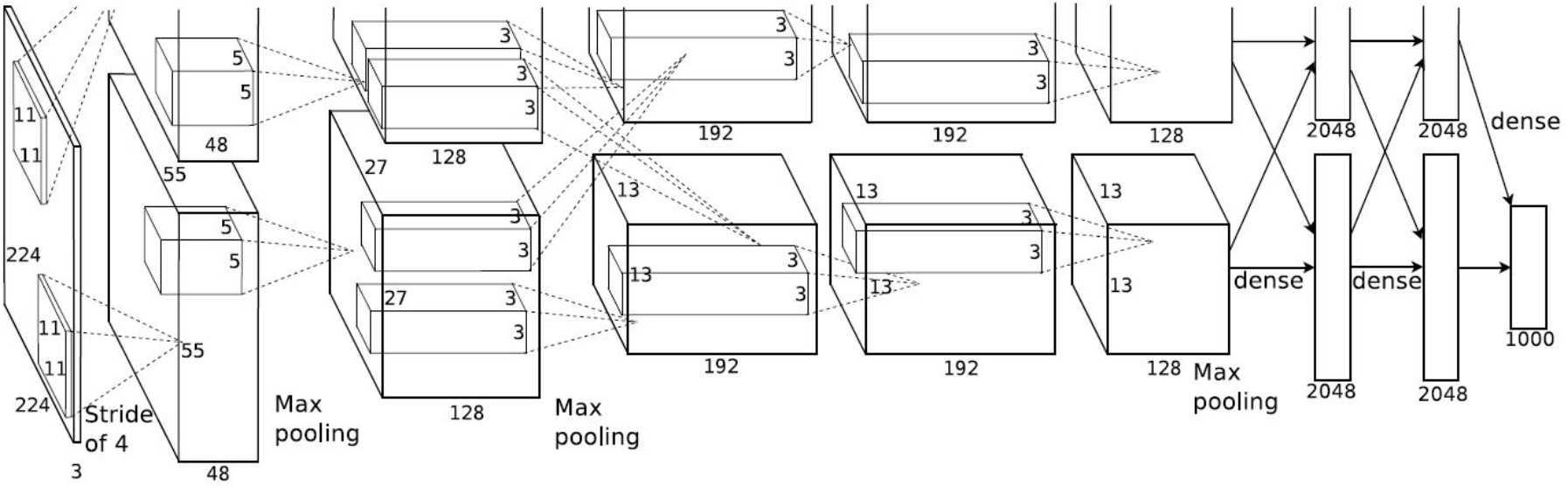
http://cs231n.github.io/convolutional-networks/ Stanford Course CS231

Convolutional Network- Assembled Layers



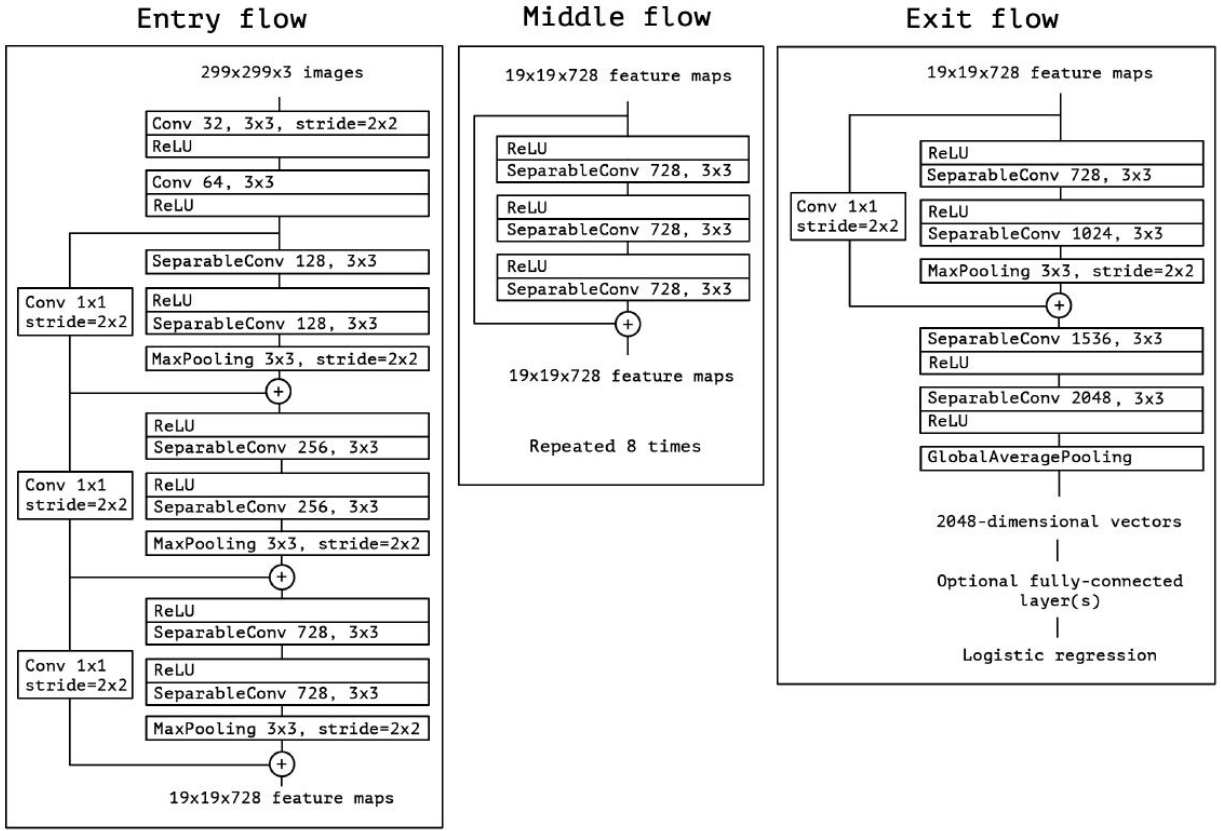
http://yann.lecun.com/exdb/lenet/ lenet 5 - see lenet in action: demonstrations of robustness

Convolutional Network- Assembled Layers



alexnet https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
https://www.learnopencv.com/understanding-alexnet/

Convolutional Network- Assembled Layers



Xception: Deep Learning with Depthwise Separable Convolutions https://arxiv.org/abs/1610.02357

One Convolutional Kernel Extracts One Feature

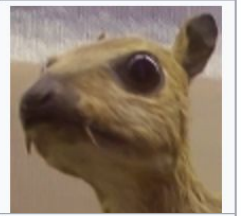
How does one capture a feature using a (part of a) layer?

Do we have to plan for a particular feature?

- **Or just a placeholder?**
  - **which is going to be filled by the (parameter) learning process**

One convolutional kernel (type) can extract one feature.

One Convolutional Kernel Extracts One Feature (from all over an image)

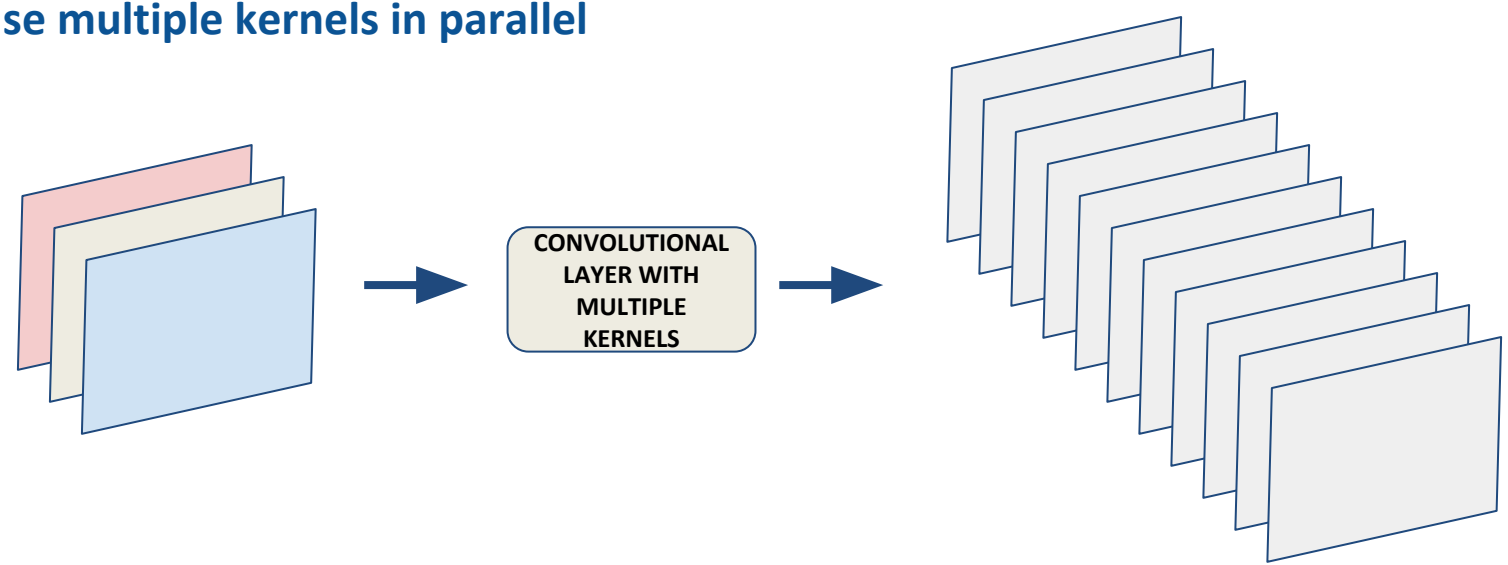
Operation	Kernel	Image result
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	

If a kernel is shared across image it can only extract one feature.

https://en.wikipedia.org/wiki/Kernel_(image_processing)

In a Convolutional Network, one Wants to Extract Many Features

Use multiple kernels in parallel



Input: 3-color image (3-feature representation of image)

Output: N-feature representation of input

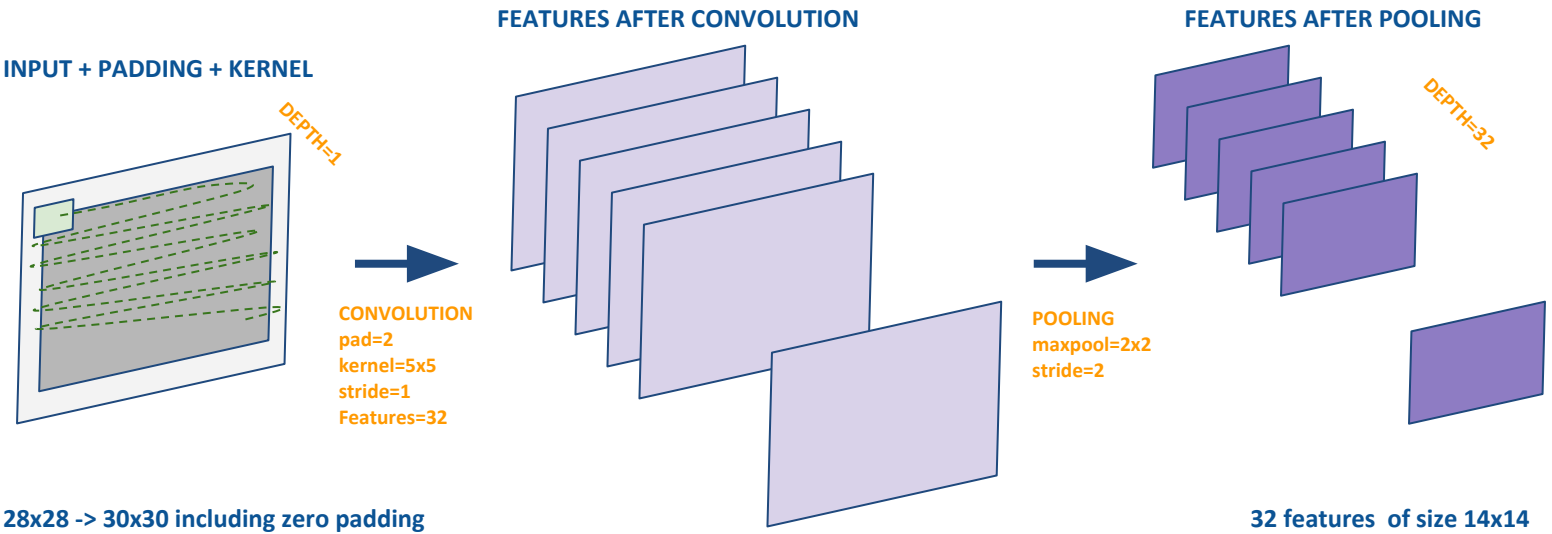
In a Convolutional Network, one Wants to Extract Many Features - Layer 1

Gray scale image example:

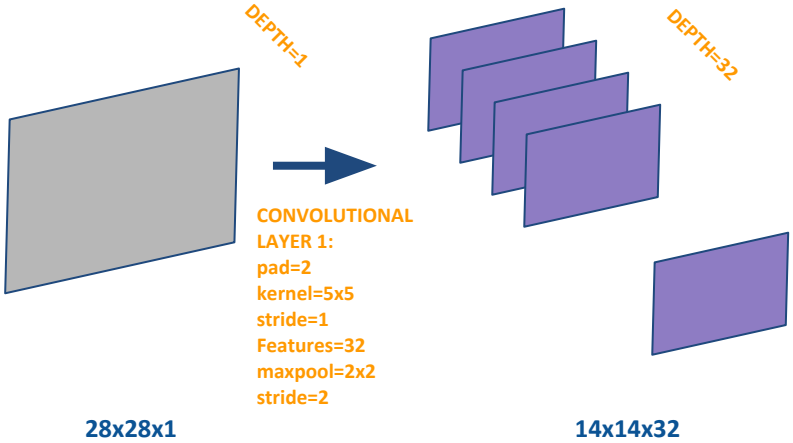
- **Images of size 28x28, at the input of the convolutional network**
- **Convolutional kernel of size 5x5**
- **Padded convolution with stride 1**
- **Max pooling with a 2x2 pooling region and a stride of 2**
- **Extract 32 features after the first layer**
- **Resulting output from this layer will be 14x14x32**
  - **Padded convolution propagates the size 28x28**
  - **14 comes from pooling (which reduces the size from 28x28 to 14x14)**
  - **and we have 32 different kernels applied in parallel, to get 32 different features**

Convolutional Layer Construction
Layer 1

Start image of size 28x28, padding to accommodate 5x5 convolutional kernel:



Convolutional Network Construction
Layer 1 (Short Notation)



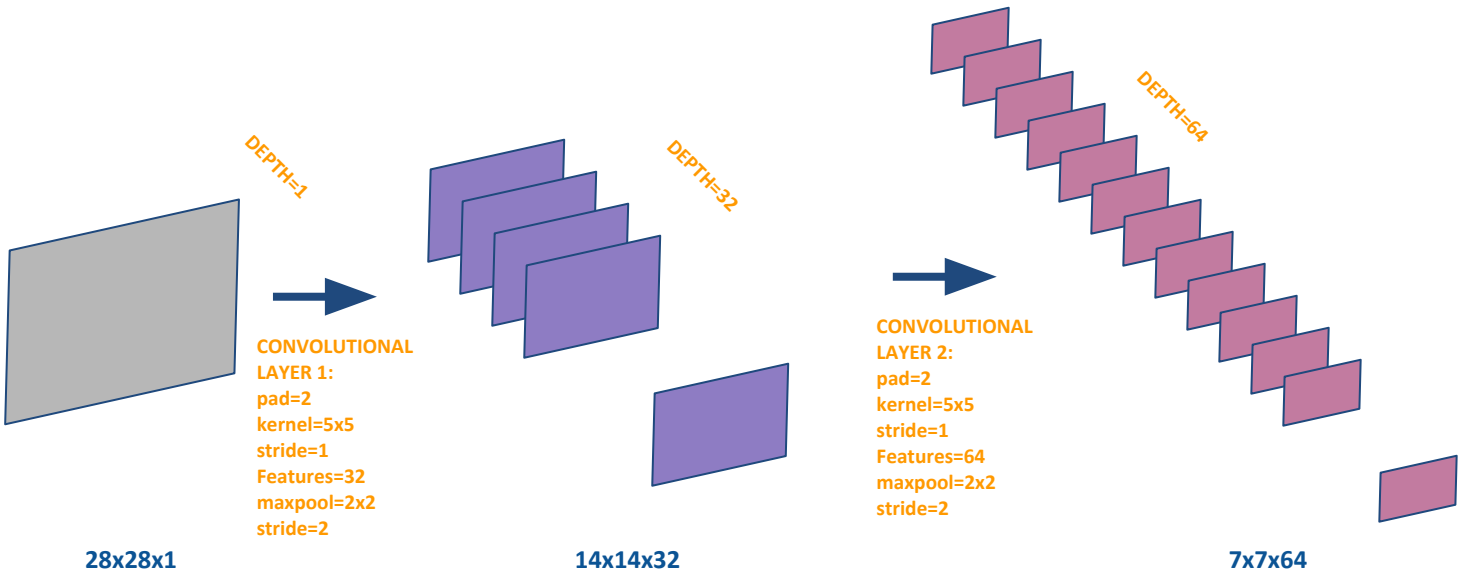
Convolutional Layer Construction

Layer 2

Layer 2 - convolutional (again):

- **Input is 14x14x32**
- **Convolutional kernel of size at 5x5 (again)**
- **Zero-padded inputs with stride = 1 (again)**
- **Max pooling with a 2x2 pooling region and a stride of 2 (again)**
- **Extract 64 features after layer 2**
- **Resulting output from this layer will be 7x7x64.**

Convolutional Network Construction
Layer 1 and Layer 2

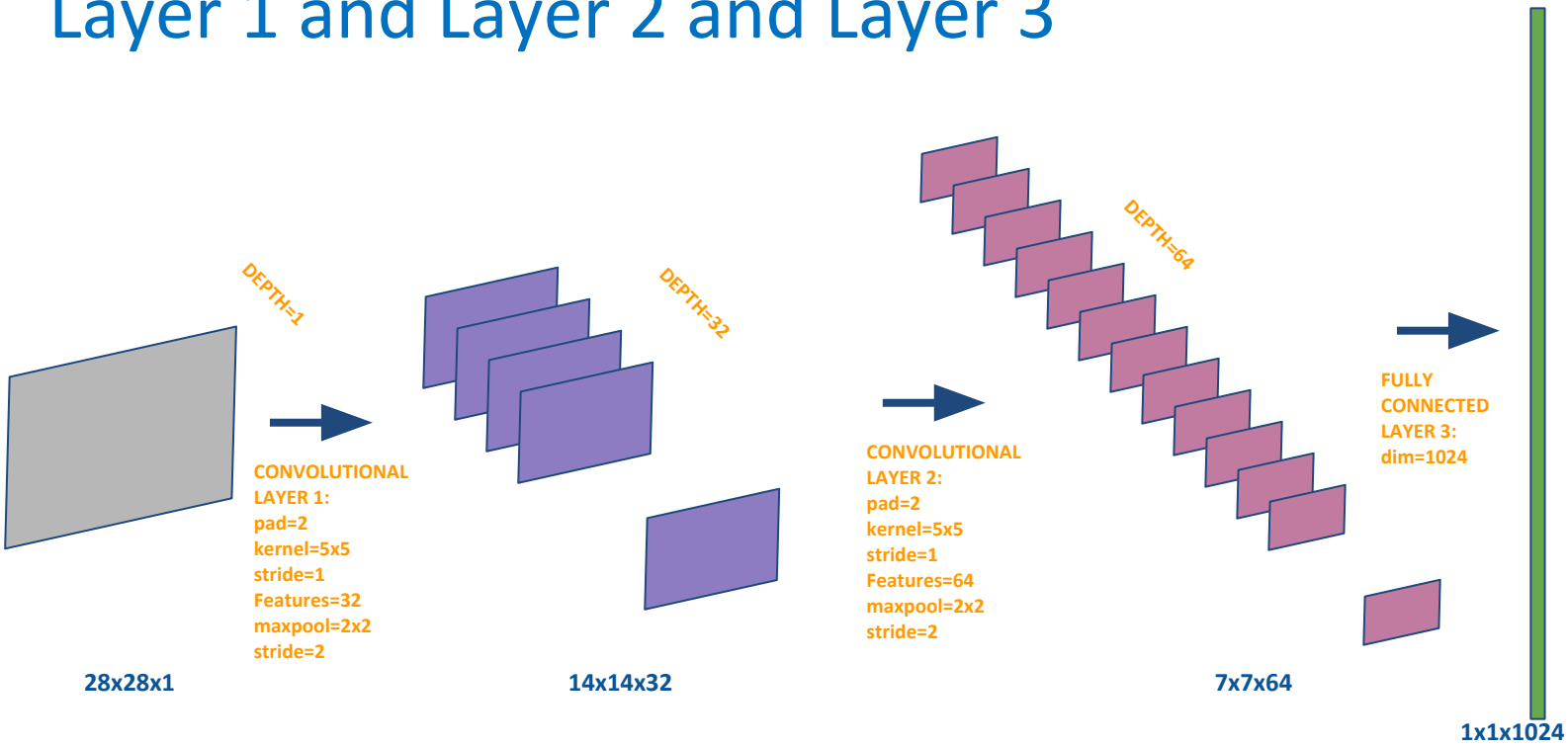


Convolutional Layer Construction
Layer 3

Layer 3 - fully connected layer:

- **Input is 7x7x64**
- **Linear transformation followed by ReLU**
- **Mapping convolutional features to a 1024 dimensional feature space**
- **Resulting output from this layer will be 1x1x1024**

Convolutional Network Construction
Layer 1 and Layer 2 and Layer 3

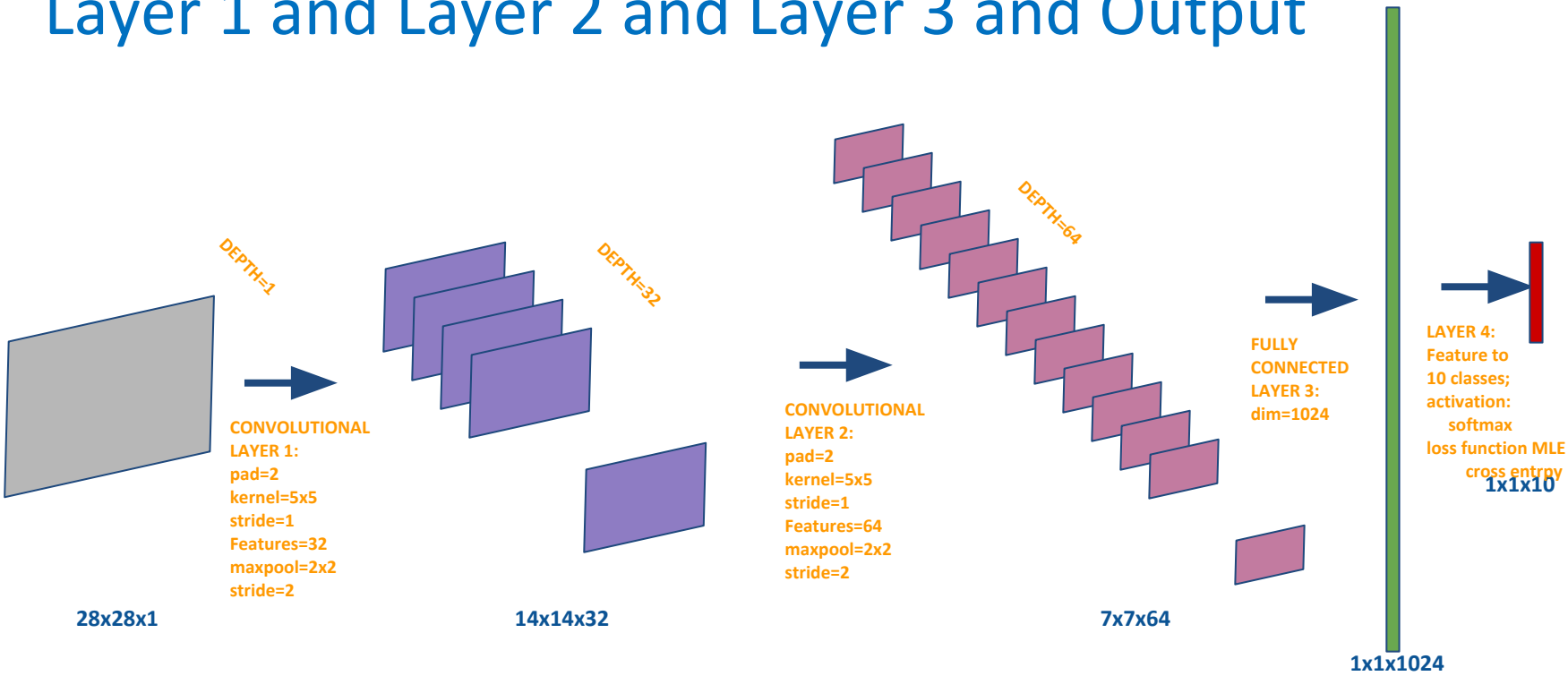


Convolutional Layer Construction
Output

Output layer:

- **Input is 1x1x1024**
- **Map 1024 feature space to 10 classes**
- **Softmax activation**
- **MLE cross-entropy loss function**

Convolutional Network Construction
Layer 1 and Layer 2 and Layer 3 and Output



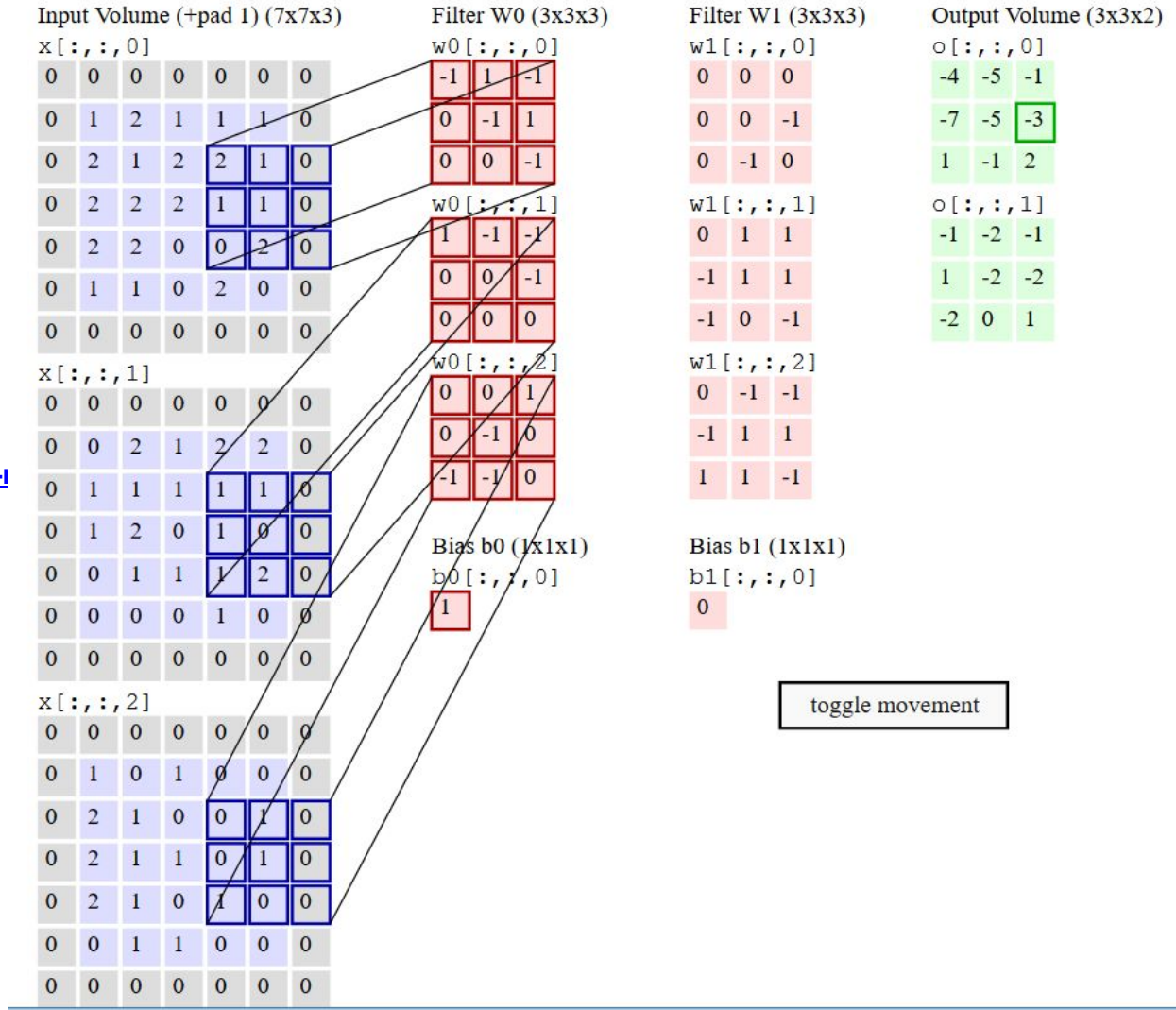
Number of parameters = (5x5x1x32+32) + (5x5x32x64+64) + (7x7x64x1024+1024) + (1024x10+10)

Convolutional Demo - Multiple Kernels

Andrej Karpathy,

Convolution Demo in

http://cs231n.github.io/convolutional-



Convolutional Network Construction and Execution

Visualization: http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

- **Instantiate a Network and Trainer**
  - layer_defs = [];
  - layer_defs.push({type:'input', out_sx:32, out_sy:32, out_depth:3});
  - layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
  - layer_defs.push({type:'pool', sx:2, stride:2});
  - layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
  - layer_defs.push({type:'pool', sx:2, stride:2});
  - layer_defs.push({type:'conv', sx:5, filters:20, stride:1, pad:2, activation:'relu'});
  - layer_defs.push({type:'pool', sx:2, stride:2});
  - layer_defs.push({type:'softmax', num_classes:10});

  - net = new convnetjs.Net();
  - net.makeLayers(layer_defs);
  - trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:4, l2_decay:0.0001});

CNNs - Historical Breakthroughs

Problems in Object Recognition

LeNet

Object Recognition in Images is HARD

- **Segmentation - objects clutter the scene**
  - **which pieces go together as a part of an object, some parts are hidden behind others**
- **Lighting - pixel intensity changes with lighting exposure**
- **Deformation - affine or non-affine ways**
  - **hand-written number 2 can have a loop or a cusp**
- **Affordances - which class the object belongs to depends on ...**
- **How it is used**
  - **chairs are for sitting, and a barrel could serve as a chair**

Object Recognition - Viewpoints Matter

Example: images, databases

- **Different viewpoints make it hard for traditional learning methods to deal with**
  - **Information moves around: a face of a child can be in the middle of an image, or in a corner**
- **Example of dimension-hopping in databases**
  - **the first field is sometimes a name, and sometimes a title in a company hierarchy**

Object Recognition - Viewpoint Invariance

Humans are good at viewpoint invariance

- **But it is very hard for computers**
- **No definite computing solution to the most general problem**
- **Approaches to achieve invariance**
  - **use redundant invariant features**
  - **draw a box around an object and normalize pixels**
  - **convolutional neural nets which use replicated features with pooling**
  - **hierarchy of parts**

Object Recognition - Viewpoint Invariance with CNNs

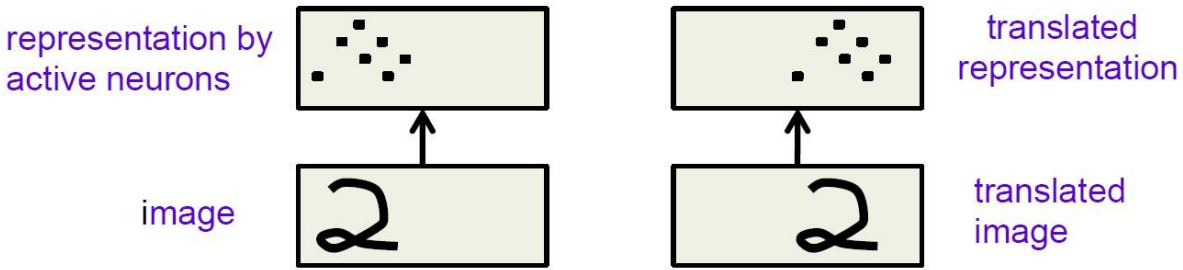
CNNs implement replicated features

- **Use many copies of the same feature detector**
  - **in different positions**
  - **dramatically reduces the number of free parameters to be learned**
  - **not easy to replicate across scale and orientation**
- **Use several different feature types**
  - **each with its own map of replicated detectors**
  - **allows each patch of an image to be represented in several ways**

Object Recognition - Viewpoint Invariance with CNNs

Replicated Feature Detectors - What do they achieve?

- **Equivariant activities of neurons**
  - **Activities of individual neuron(al) activities are not invariant to translation**
  - **The activities are equivariant**
- **Invariant knowledge**
  - **in training, a feature may occur in several locations only**
  - **in testing, the detector for that feature will be available in all location**

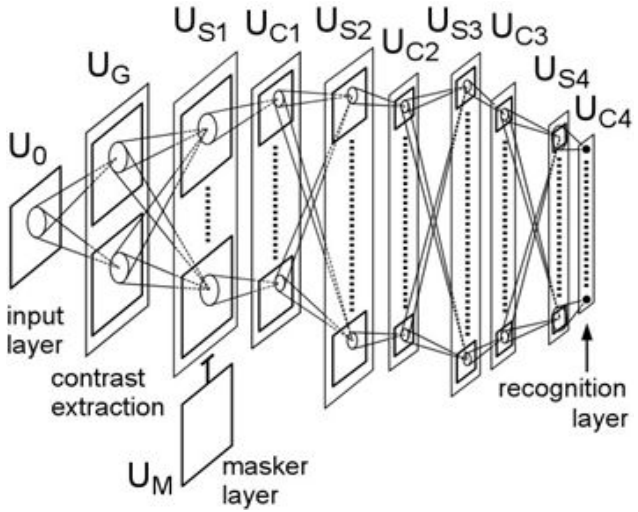


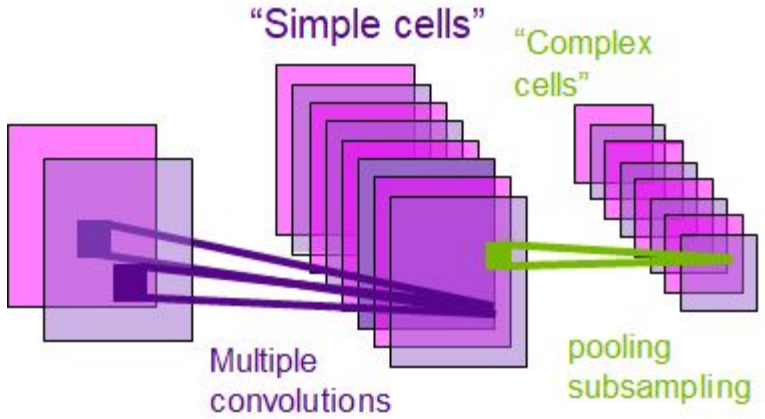
from Hinton, course on deep learning

Early Hierarchical Feature Models for Vision

[Hubel & Wiesel 1962]: Simple cells detect local features

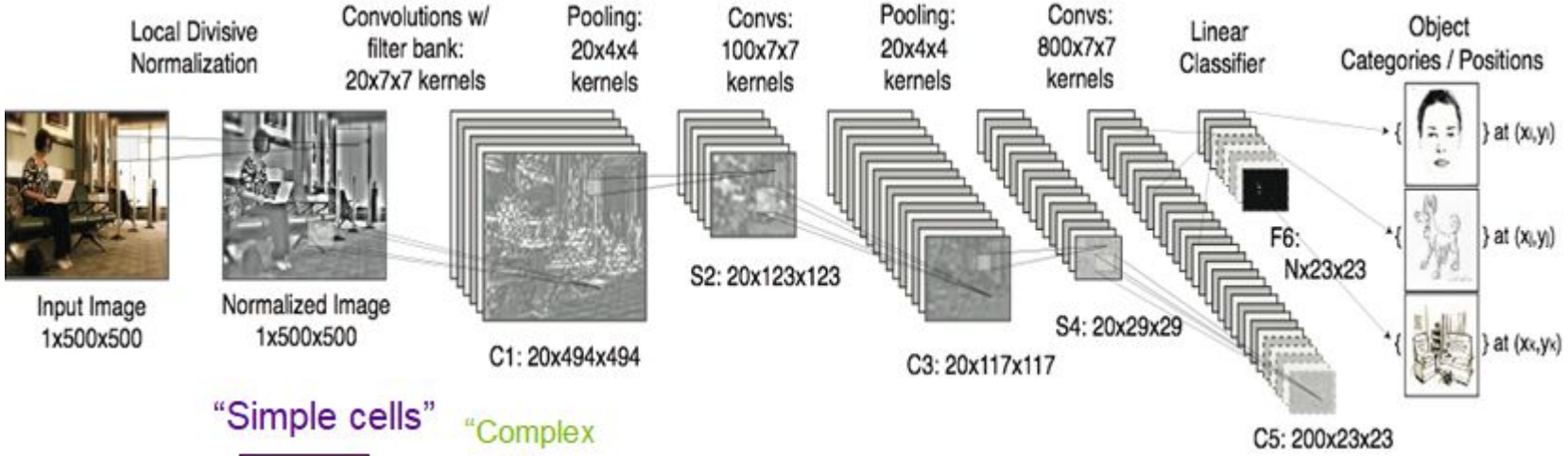
Complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.

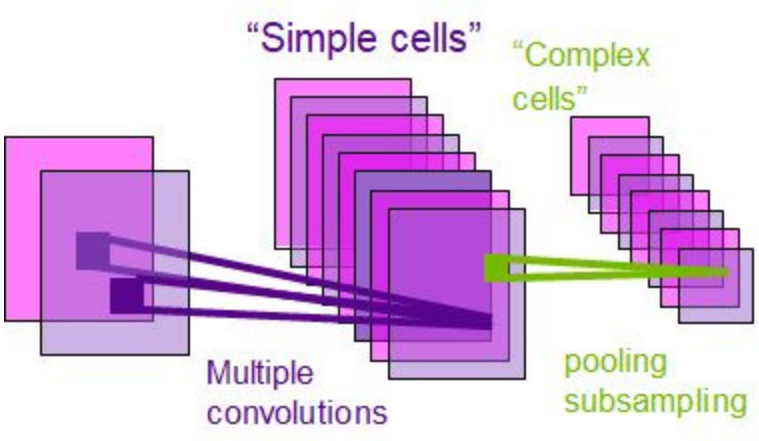




Cognitron & Neocognitron [Fukushima 1974-1982]

The Convolutional Net Model: Multistage Hubel-Wiesel System

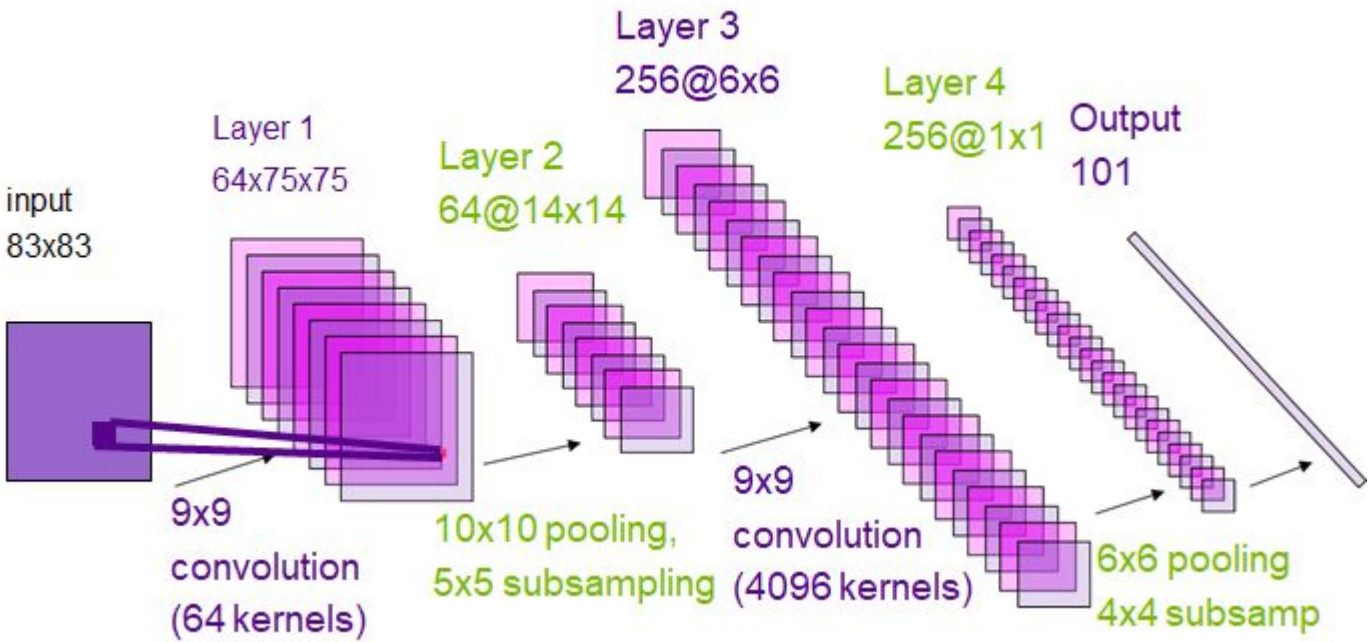




Training is supervised, with stochastic gradient descent.

[LeCun et al. 89] [LeCun et al. 98]

Convolutional Network (ConvNet)



Non-Linearity: half-wave rectification (ReLU), shrinkage function, sigmoid

Pooling: max, average, L1, L2, log-sum-exp

Training: Supervised (1988-2006), Unsupervised+Supervised (2006-now)

CNNs Major Milestone
Digit Recognition using LeNet

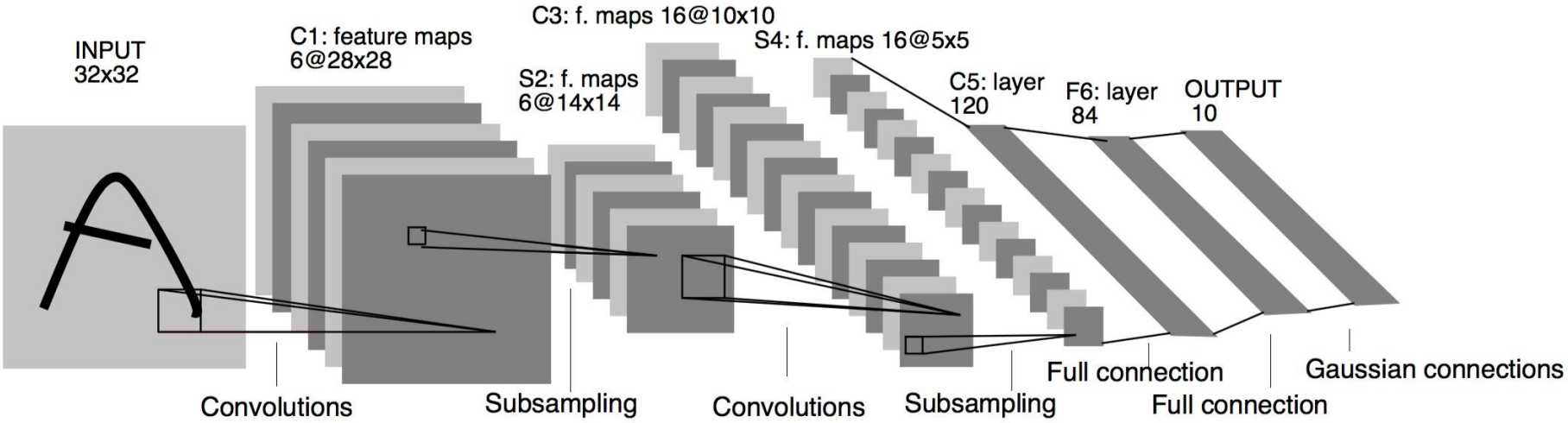
High quality recognizer of handwritten digits with feedforward NN

- **using backpropagation**
- **multiple hidden layers**
- **pooling of outputs of nearby replicated units (neurons)**
- **a wider net can deal with several characters at once even if they overlap**
- **new way of training a complete system**

Was used by 10 percent of bank check-reading applications

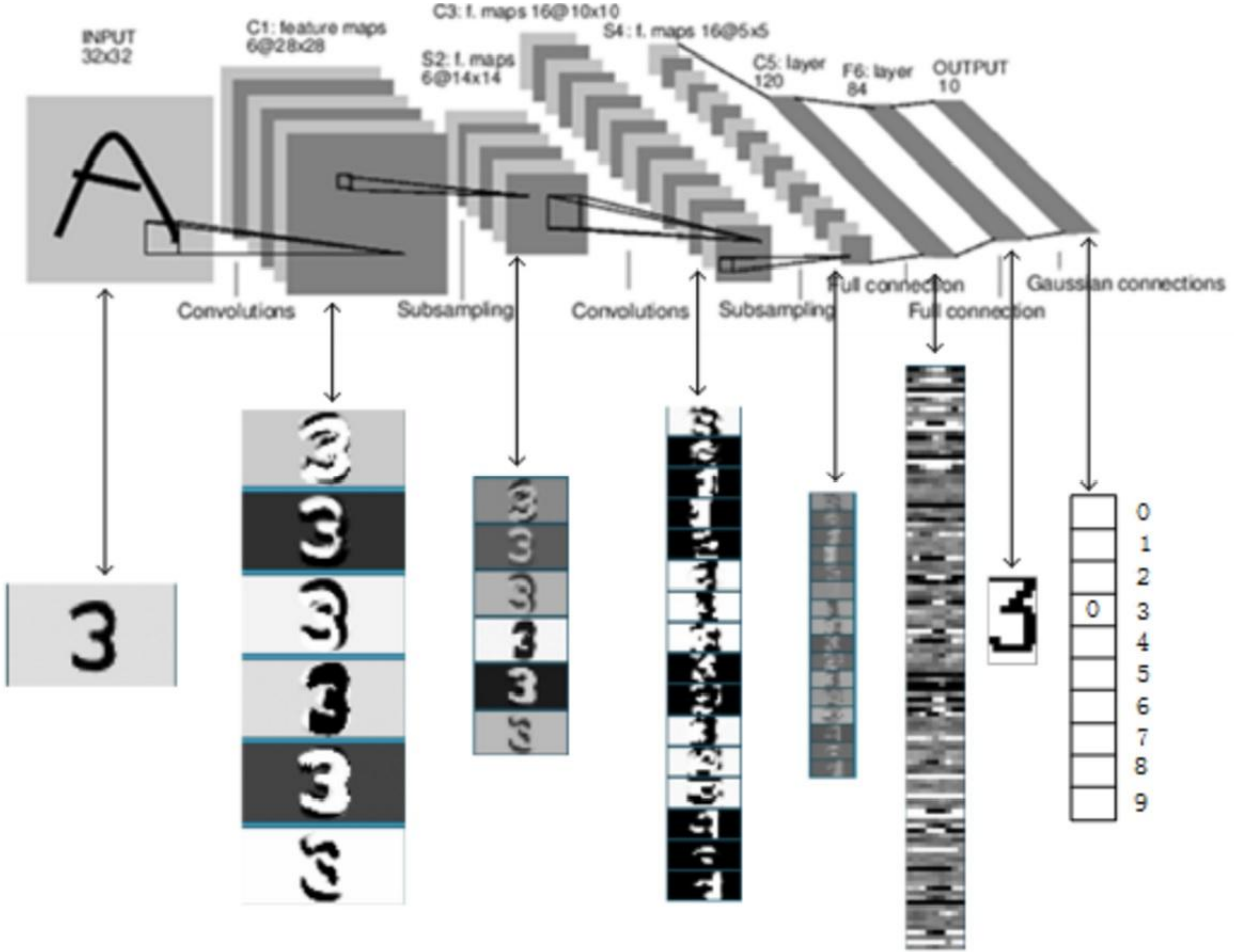
- **See: http://yann.lecun.com**

CNN
LeNet Architecture



–filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

CNN LeNet Intermediate Data Representation



LeNet1 Demo from 1993 - LeCun

Holmdel, New Jersey, 40 miles south of NYC

1990: Bell Labs

Running on a 486 PC with an AT&T DSP32C add-on board (20 Mflops!)

LeNet5 Demo(s)

http://yann.lecun.com/exdb/lenet/index.html

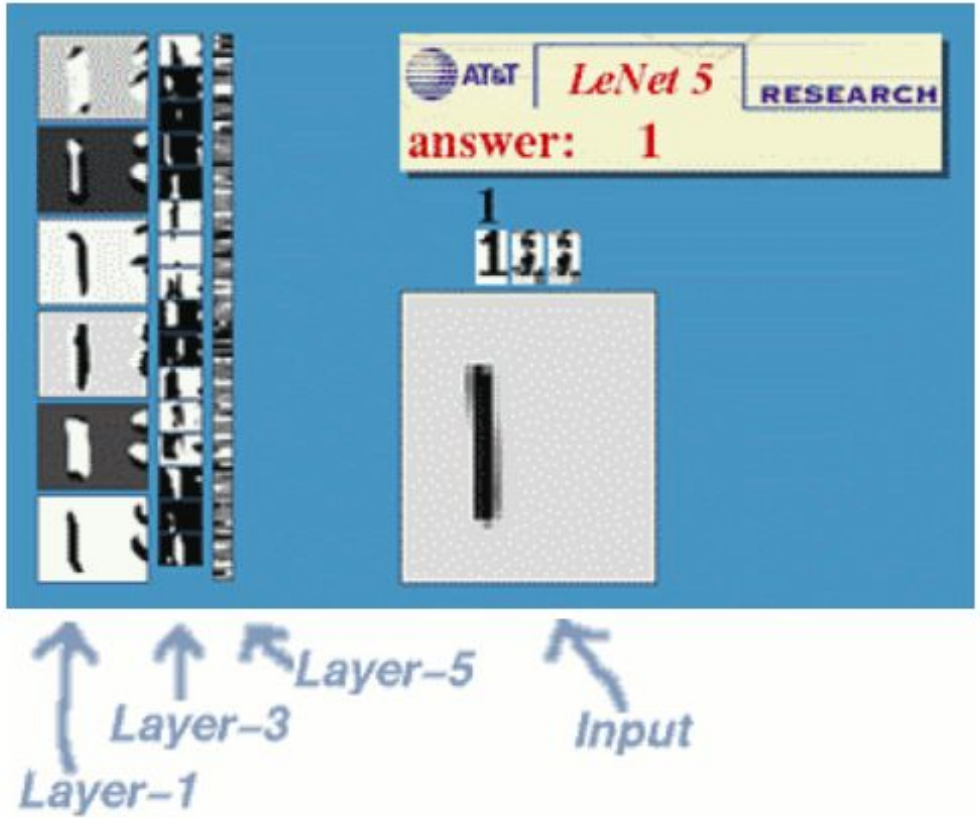
Unusual patterns , unusual styles , weirdos.

Invariance: translation, scale, rotation,
squeezing , stroke width.

Noise Resistance

Multiple Characters

Here is an example of LeNet-5 in action.



CNNs demos

Various

CNN Demos and Tutorials

CNN Visualization http://scs.ryerson.ca/~aharley/vis/conv/flat.html & http://scs.ryerson.ca/~aharley/vis/

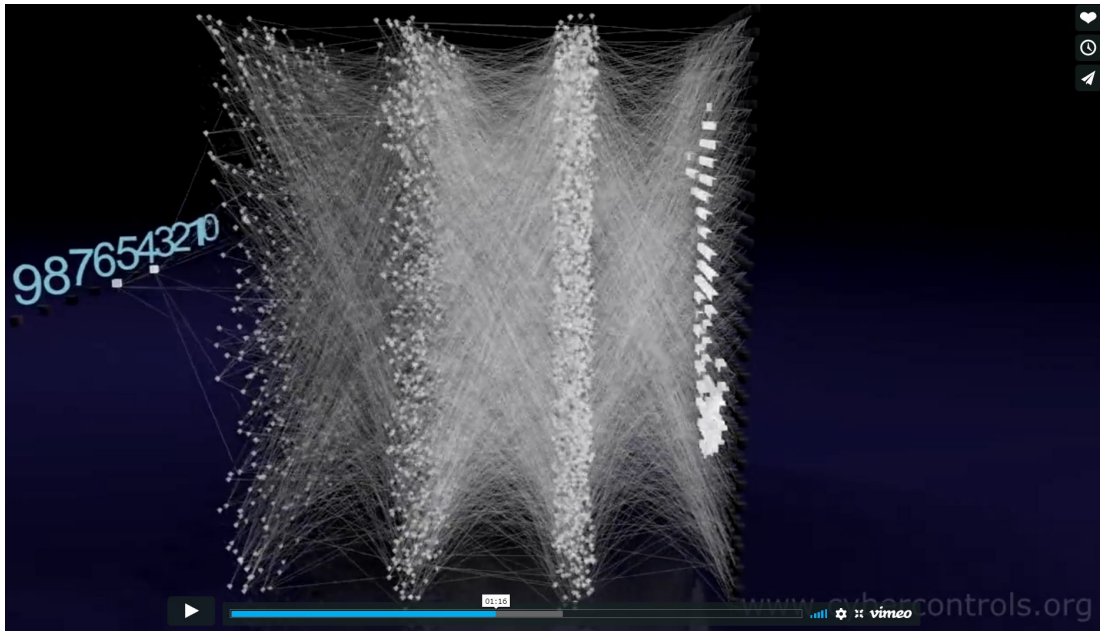
3-D illustrations: http://scs.ryerson.ca/~aharley/vis/conv/ (try to grab the mouse and rotate)

An Intuitive Explanation of Convolutional Neural Networks
http://www.kdnuggets.com/2016/11/intuitive-explanation-convolutional-neural-networks.html/3

Convolutional Neural Networks http://cs231n.github.io/convolutional-networks/

CNN Demos and Tutorials

https://www.cybercontrols.org/neuralnetworks



Backup Slides

Various