Deep Learning

Jay Urbain, PhD

Many of the pictures, results, and other materials are taken from:
Andrew Ng,
Ruslan Salakhutdinov
Joshua Bengio
Geoffrey Hinton
Yann LeCun

Major Directions

- Self-taught learning
- Learning feature hierarchies (Deep learning)
- Scaling up

Supervised learning






Cars





Motorcycles

Testing:
What is this?

Semi-supervised learning








Unlabeled images (all cars/motorcycles)

Car

Motorcycle

Testing:
What is this?

Self-taught learning

Unlabeled images (random internet images)

Car

Motorcycle

Testing:
What is this?

Self-taught learning

Unlabeled images (random internet images)

→ Sparse coding, LCC, etc. → $f_1, f_2, \ldots, f_k$

If labeled training set is small, can give huge performance boost.

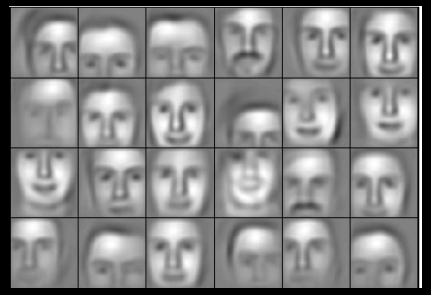
Use learned $f_1, f_2, \ldots, f_k$ to represent training/test sets.

Car

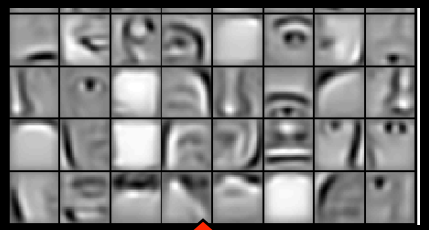
Motorcycle

Using $f_1, f_2, \ldots, f_k$ → $a_1, a_2, \ldots, a_k$

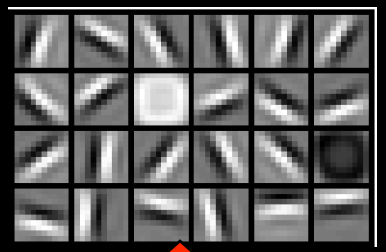
Feature hierarchies



object models

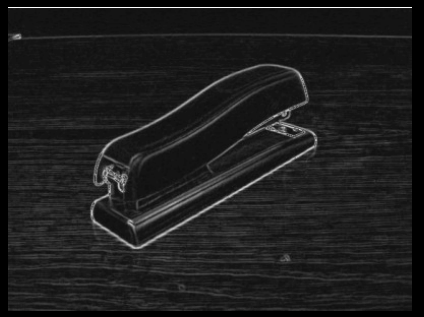


object parts
(combination
of edges)



edges

pixels



Deep learning algorithms

- Stack sparse coding algorithm
- Deep Belief Network (DBN) (Hinton)
- Deep sparse autoencoders (Bengio)

[Other related work: LeCun, Lee, Yuille, Ng …]

Deep learning with autoencoders

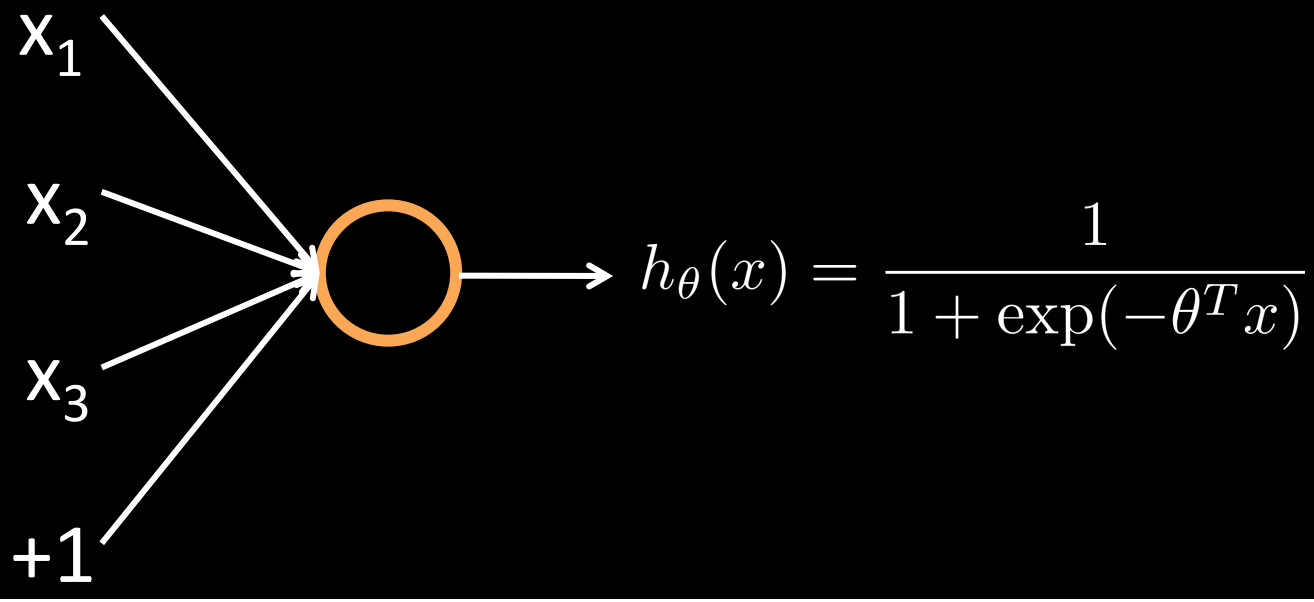
- Logistic regression
- Neural network
- Sparse autoencoder
- Deep autoencoder

Logistic regression has a learned parameter vector q.
On input x, it outputs:

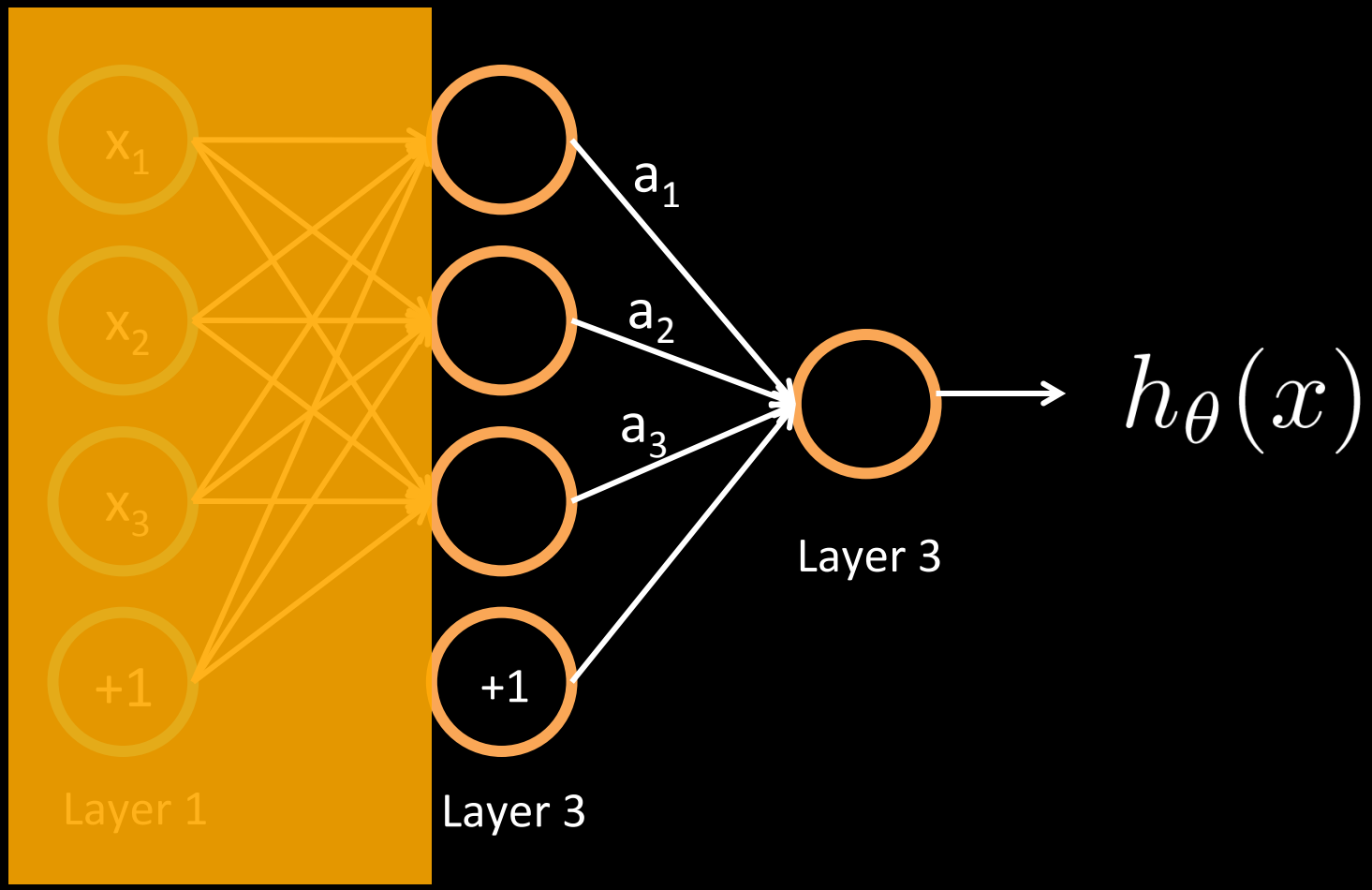
$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1+\exp(-\theta^T x)}$$

where $\sigma(z) = 1/(1+\exp(-z))$

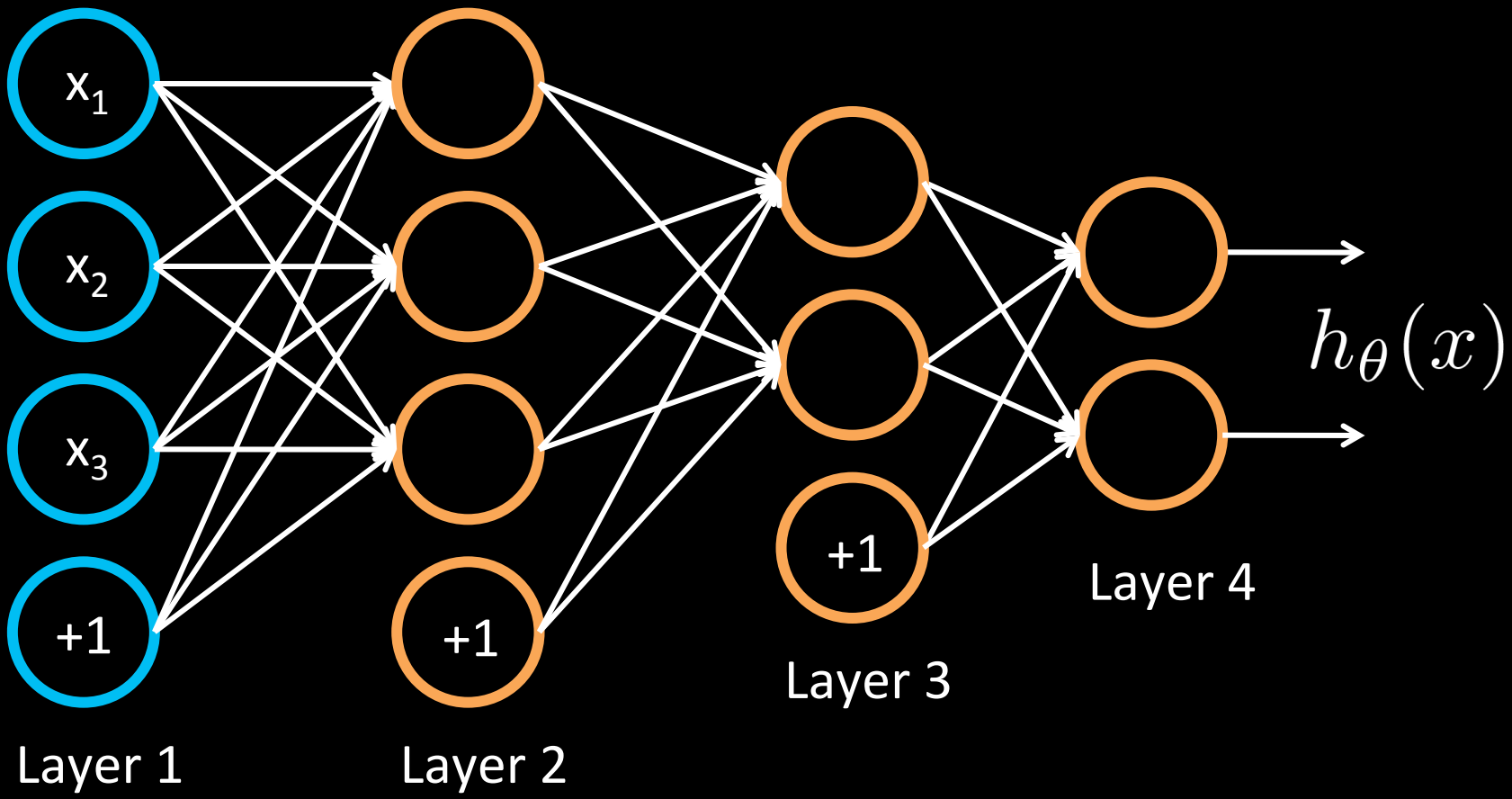
Draw a logistic regression unit as:



String a lot of logistic units together. Example 3 layer network:



Example 4 layer network with 2 output units:

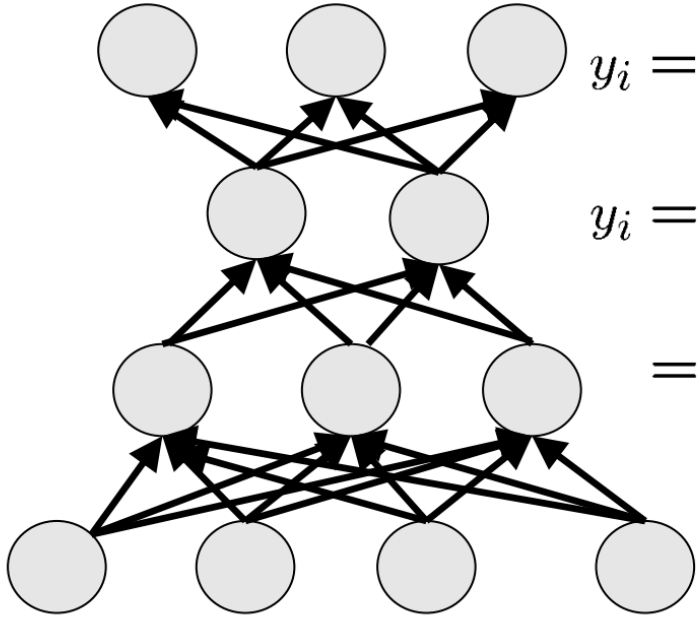


Defintion: Deep architectures are composed of *multiple levels* of non-linear operations, such as neural nets with many hidden layers.

Output layer

Hidden layers

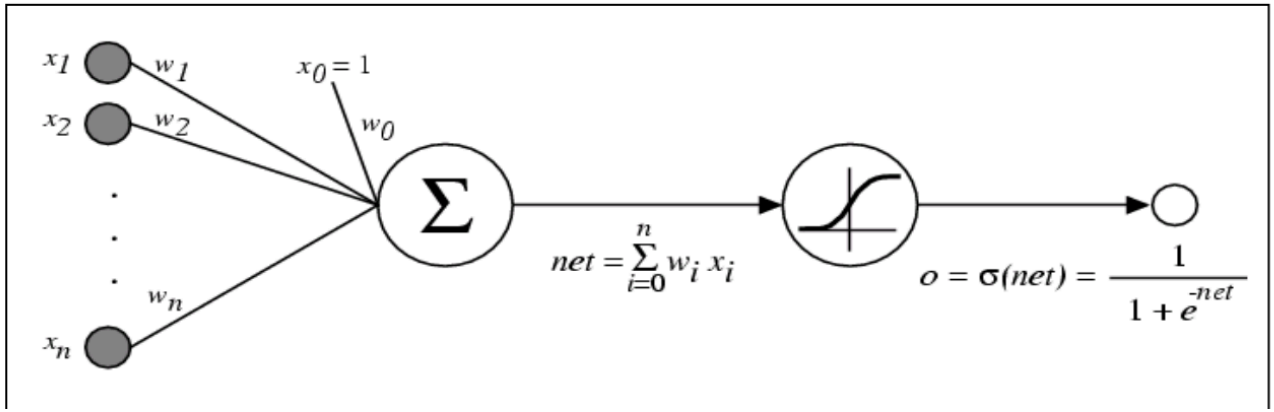
Input layer



$$y_i = \tanh(\sum_j W_{ij}x_j + W_{i0})$$

$$y_i = \sigma(\sum_j W_{ij}x_j + W_{i0})$$

$$= \frac{1}{1+\exp[-(\sum_j W_{ij}x_j + W_{i0})]}$$



Goal of Deep Architectures

Goal: Deep learning methods aim at

- learning *feature hierarchies*
- where features from higher levels of the hierarchy are formed by lower level features.

edges, local shapes, object parts

Low level representation

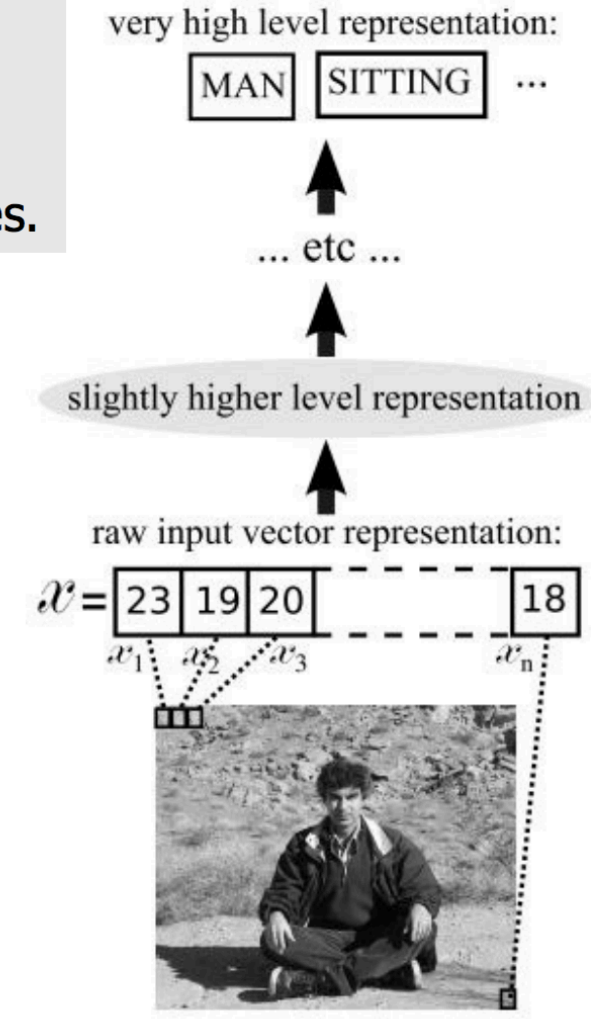
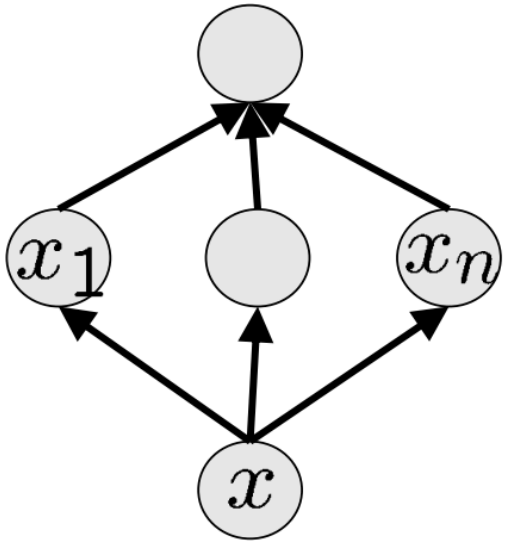


Figure is from Yoshua Bengio

Neurobiological Motivation

Most current learning algorithms are shallow architectures (1-3 levels)
(SVM, kNN, MoG, KDE, Parzen Kernel regression, PCA, Perceptron,…)

$$\text{SVM: } \hat{f}(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$



The mammal brain is organized in a deep architecture (Serre, Kreiman, Kouh, Cadieu, Knoblich, & Poggio, 2007)
(E.g. visual system has 5 to 10 levels)

Deep Learning History

- **Inspired** by the architectural depth of the brain, researchers wanted for decades to train deep multi-layer neural networks.
- **No success**ful attempts were reported before 2006 …

  Researchers reported positive experimental results with typically two or three levels (i.e. one or two hidden layers), but training deeper networks consistently yielded poorer results.

- **Exception**: convolutional neural networks, LeCun 1998
- **SVM**: Vapnik and his co-workers developed the Support Vector Machine (1993). It is a shallow architecture.
- **Digression**: In the 1990's, many researchers abandoned neural networks with multiple adaptive hidden layers because SVMs worked better, and there was no successful attempts to train deep networks.
- **Breakthrough in 2006**

Breakthrough Training Deep Networks

Deep Belief Networks (DBN)

Hinton, G. E, Osindero, S., and Teh, Y. W. (2006).
A fast learning algorithm for deep belief nets.
Neural Computation, 18:1527-1554.

Autoencoders

Bengio, Y., Lamblin, P., Popovici, P., Larochelle, H. (2007).
Greedy Layer-Wise Training of Deep Networks,
Advances in Neural Information Processing Systems 19

Theoretical Advantages of Deep Architectures

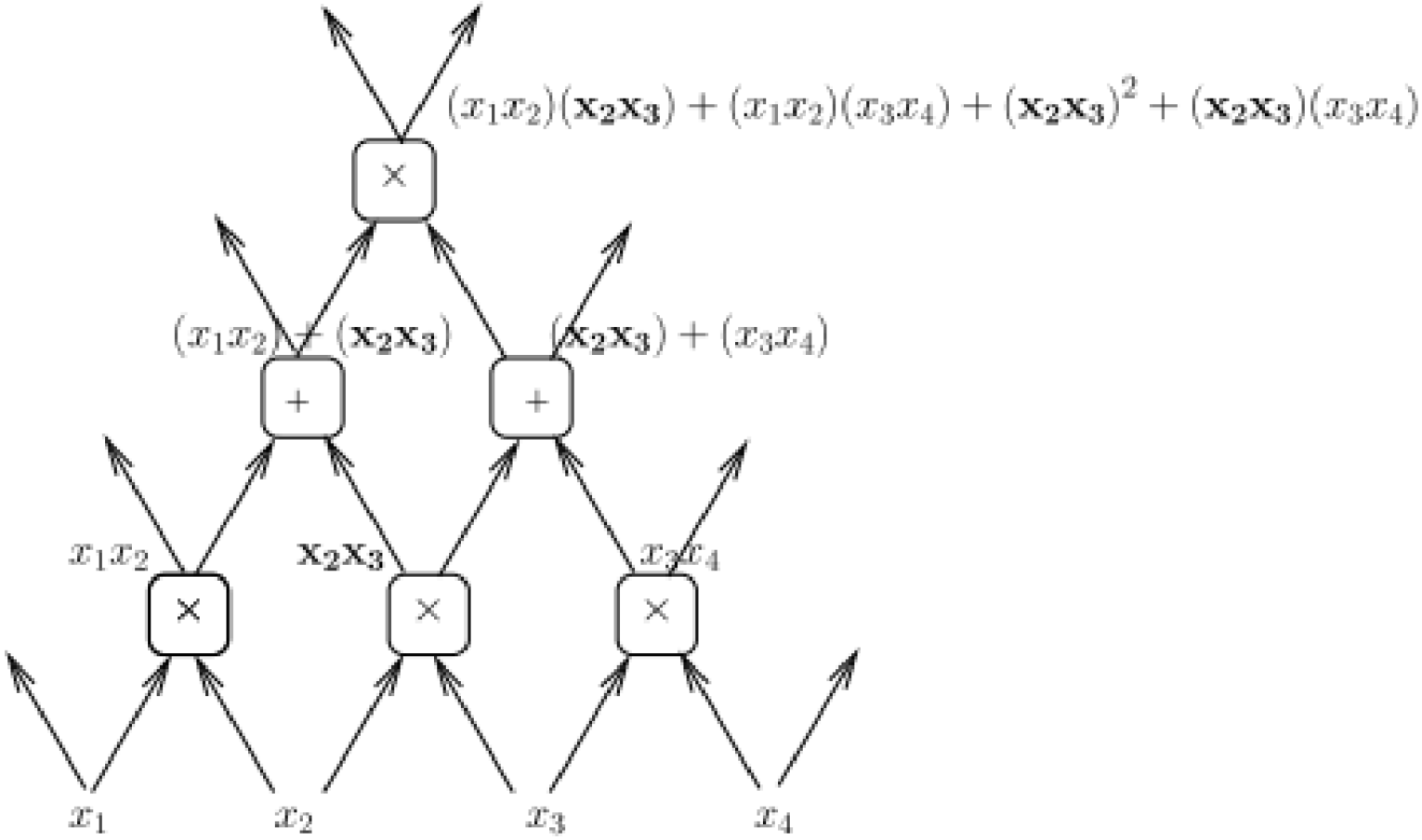
- Some functions cannot be efficiently represented (in terms of number of tunable elements) by architectures that are too shallow.
- Deep architectures might be able to represent some functions otherwise not efficiently representable.
- **More formally**:

  Functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth k – 1 architecture

- The consequences are
  - **Computational**: We don't need exponentially many elements in the layers
  - **Statistical**: poor generalization may be expected when using an insufficiently deep architecture for representing some functions.

Theoretical Advantages of Deep Architectures

The Polynoimal circuit:



Deep Convolution Neural Networks

- Biologically-inspired variants of MLPs (multi-layer perceptrons).
- From Hubel and Wiesel's early work on the cat's visual cortex [Hubel68].
- The visual cortex contains a complex arrangement of cells.
  - 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.
  - These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.
- Two basic cell types have been identified:
  - Simple cells respond maximally to specific edge-like patterns within their receptive field.
  - Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

Building Convolution Neural Networks

There are three types of layers in a Convolutional Neural Network:

- Convolutional Layers.
- Pooling Layers.
- Fully-Connected Layers.

Convolution Layers

Convolutional layers are comprised of filters and feature maps.

Filters

- Filters are the neurons of the layer.
- They have input weights and output a value.
- The input size is a fixed square called a patch or a receptive field.
- If the convolutional layer is an input layer, then the input patch will be pixel values.
- If the input is deeper in the network architecture, then the convolutional layer will take input from a feature map from the previous layer.

Convolution Layers

Feature Maps

- Output of one filter applied to the previous layer.
- A given filter is drawn across the entire previous layer (convolve), moved one pixel at a time.
- Each position results in an activation of the neuron and the output is collected in the feature map.
- If the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by (field width - 1) input values.
- The distance that filter is moved across the input from the previous layer each activation is referred to as the stride.

Pooling Layers

- Pooling layers down-sample the previous layers feature map (dimensionality reduction) .
- Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layers feature map.
- As such, pooling may be consider a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.
- Have a receptive field, often much smaller than the convolutional layer.
- The stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap.
- Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

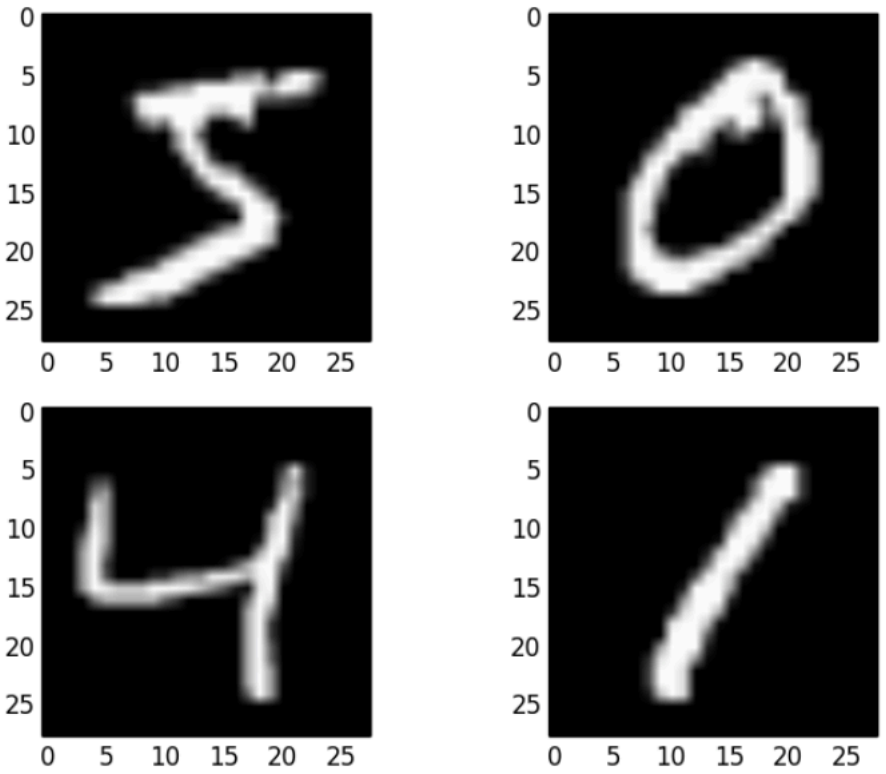
Fully Connected Layers

- Fully connected layers are the normal flat feed-forward neural network layer.
- These layers may have a nonlinear activation function or a softmax activation in order to output probabilities of class predictions.
- Fully connected layers are used at the end of the network after feature extraction and consolidation has been performed by the convolutional and pooling layers.
- They are used to create final nonlinear combinations of features and for making predictions by the network.

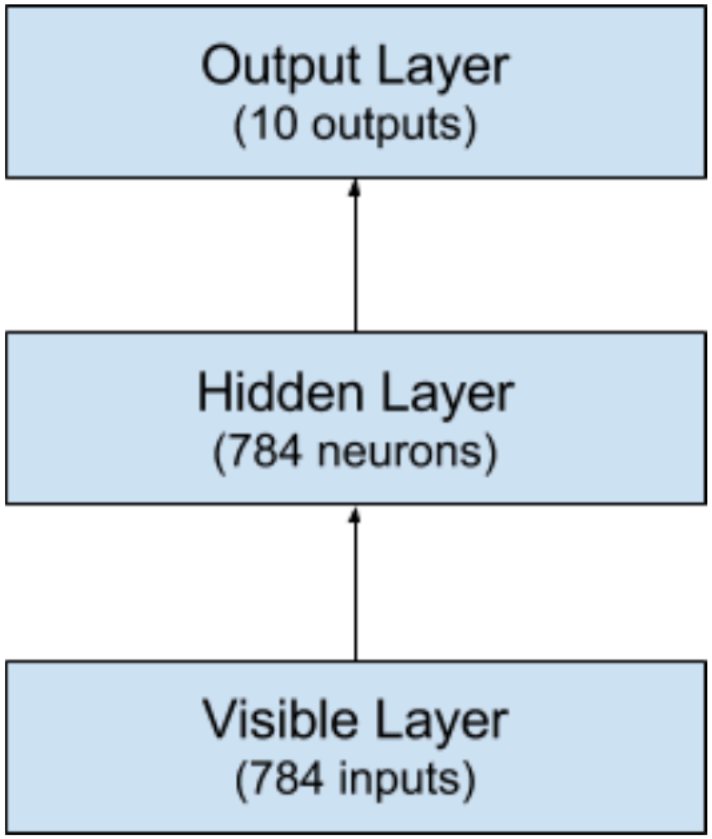
Handwritten Digit Recognition Dataset

- MNIST is a dataset developed by Yann LeCun, Corinna Cortes and Christopher Burges for evaluating machine learning models on the handwritten digit classification.
- The dataset was constructed from a number of scanned document datasets available from the National Institute of Standards and Technology (NIST).
- Images of digits were taken from a variety of scanned documents, normalized in size and centered.
- Each image is a 28 × 28 pixel square (784 pixels total).
- A standard split of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it.
- It is a digit recognition task. 10 digits (0 to 9) or 10 classes to predict.
- Results are reported using prediction error, which is the inverted classification accuracy.
- Excellent results achieve a prediction error of less than 1%.
- State-of-the-art prediction error of approximately 0.2% can be achieved with large Convolutional Neural Networks.

Handwritten Digit Recognition Dataset



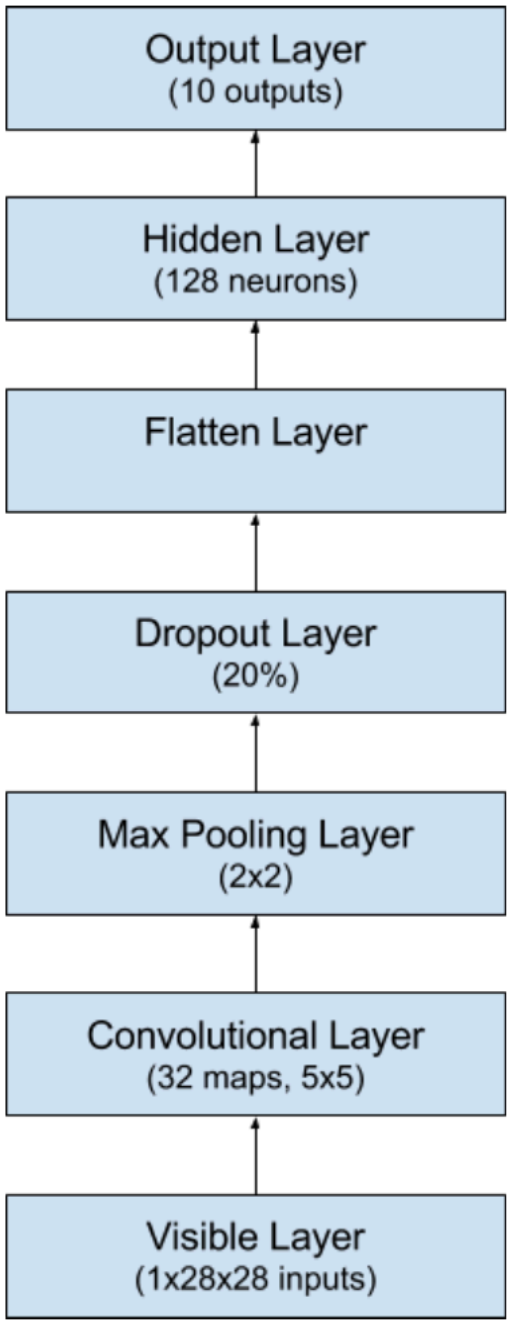
Baseline Model



Baseline Model Results

- runfile('/Users/jayurbain/Dropbox/deep_learning_with_python/workspace/deep_learning_with_python/project-mnist/cnn_mnist_simple.py', wdir='/Users/jayurbain/Dropbox/deep_learning_with_python/workspace/deep_learning_with_python/project-mnist')
- Train on 60000 samples, validate on 10000 samplesEpoch 1/1044s - loss: 0.2065 - acc:
- 0.9370 - val_loss: 0.0756 - val_acc: 0.9756Epoch 2/1044s - loss: 0.0642 - acc: 0.9804 - val_loss: 0.0471 - val_acc: 0.9836Epoch 3/1046s - loss: 0.0448 - acc: 0.9862 - val_loss: 0.0403 - val_acc: 0.9879Epoch 4/1046s - loss: 0.0348 - acc: 0.9888 - val_loss: 0.0360 - val_acc: 0.9879Epoch 5/1046s - loss: 0.0271 - acc: 0.9913 - val_loss: 0.0346 - val_acc: 0.9884Epoch 6/1046s - loss: 0.0211 - acc: 0.9935 - val_loss: 0.0401 - val_acc: 0.9878Epoch 7/1046s - loss: 0.0182 - acc: 0.9945 - val_loss: 0.0342 - val_acc: 0.9888Epoch 8/1047s - loss: 0.0143 - acc: 0.9954 - val_loss: 0.0334 - val_acc: 0.9894Epoch 9/1046s - loss: 0.0119 - acc: 0.9963 - val_loss: 0.0362 - val_acc: 0.9900Epoch 10/1046s - loss: 0.0107 - acc: 0.9965 - val_loss: 0.0361 - val_acc: 0.9891
- Classification Error: 1.09%

Convolution Model



Convolution Model Results

- runfile('/Users/jayurbain/Dropbox/deep_learning_with_python/workspace/deep_learning_with_python/project-mnist/cnn_mnist_deep.py', wdir='/Users/jayurbain/Dropbox/deep_learning_with_python/workspace/deep_learning_with_python/project-mnist')
- Train on 60000 samples, validate on 10000 samples
- Epoch 1/1052s - loss: 0.3258 - acc: 0.8965 - val_loss: 0.0703 - val_acc: 0.9777Epoch 2/1052s - loss: 0.0850 - acc: 0.9736 - val_loss: 0.0431 - val_acc: 0.9863Epoch 3/1052s - loss: 0.0623 - acc: 0.9803 - val_loss: 0.0377 - val_acc: 0.9884Epoch 4/1051s - loss: 0.0501 - acc: 0.9839 - val_loss: 0.0335 - val_acc: 0.9896Epoch 5/1052s - loss: 0.0441 - acc: 0.9858 - val_loss: 0.0341 - val_acc: 0.9893Epoch 6/1052s - loss: 0.0388 - acc: 0.9875 - val_loss: 0.0285 - val_acc: 0.9903Epoch 7/1051s - loss: 0.0335 - acc: 0.9893 - val_loss: 0.0248 - val_acc: 0.9918Epoch 8/1052s - loss: 0.0304 - acc: 0.9898 - val_loss: 0.0241 - val_acc: 0.9916Epoch 9/1052s - loss: 0.0296 - acc: 0.9905 - val_loss: 0.0211 - val_acc: 0.9934Epoch 10/1051s - loss: 0.0264 - acc: 0.9914 - val_loss: 0.0289 - val_acc: 0.9905
- Classification Error: 0.95%

Deep Convolution Networks

- Deep supervised neural networks are generally too difficult to train.
- **One notable exception**: convolutional neural networks (CNN)
- Convolutional nets were inspired by the visual system's structure
- They typically have five, six or seven layers, a number of layers which makes fully-connected neural networks almost impossible to train properly when initialized randomly.

Deep Convolution Networks

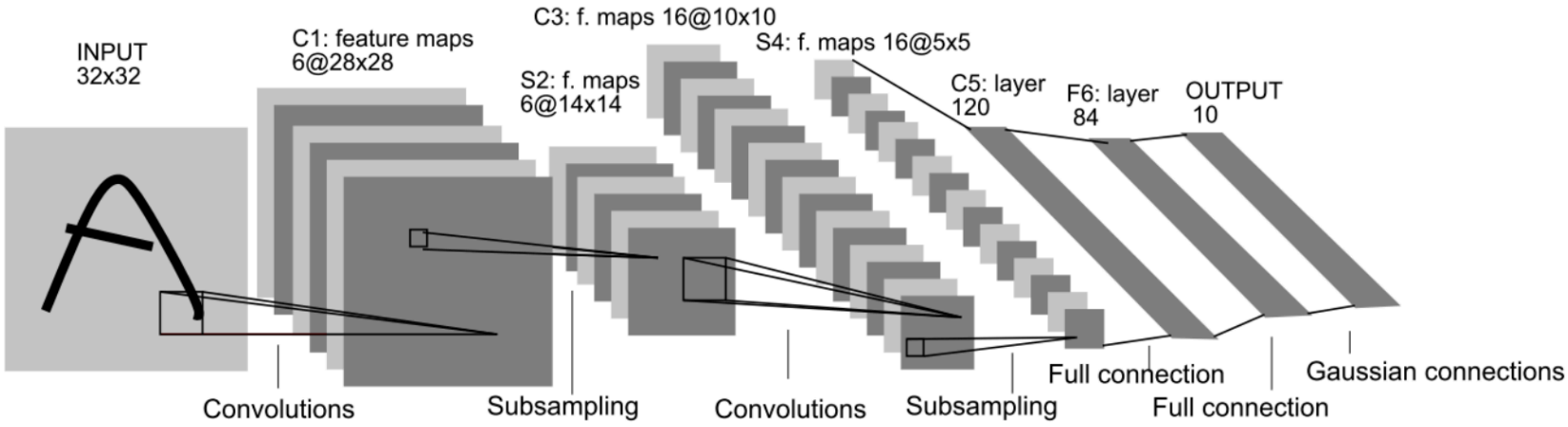
Compared to standard feedforward neural networks with similarly-sized layers,

- CNNs have much fewer connections and parameters
- and so they are easier to train,
- while their theoretically-best performance is likely to be only slightly worse.

LeNet 5

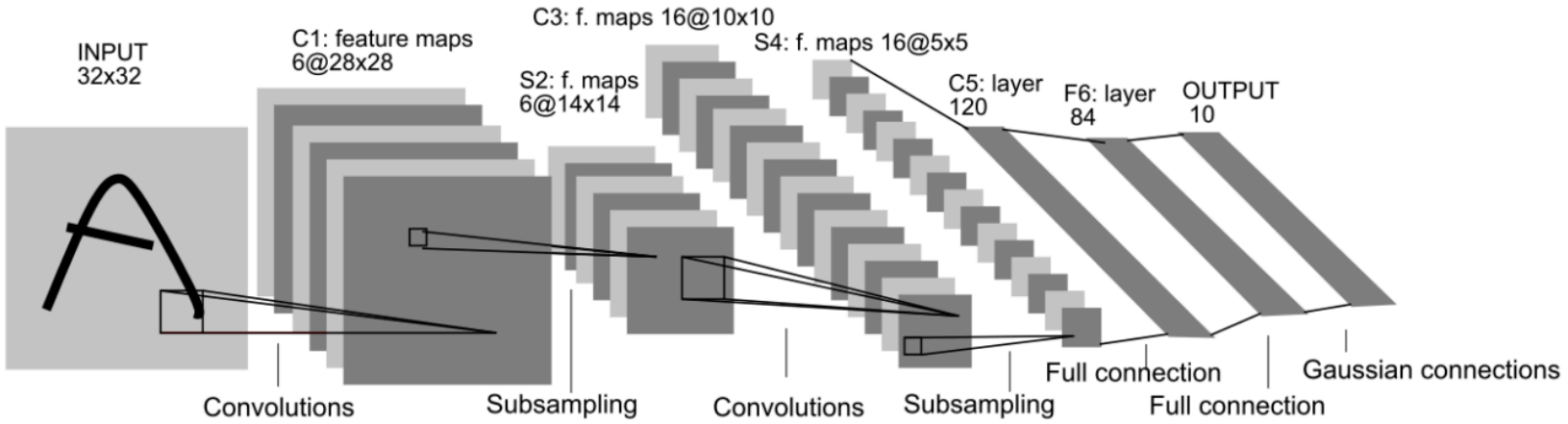
Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: **Gradient-Based Learning Applied to Document Recognition**, *Proceedings of the IEEE, 86(11):2278-2324, November **1998***

LeNet 5, LeCun 1998



- Input: 32x32 pixel image. Largest character is 20x20
  (All important info should be in the center of the receptive field of the highest level feature detectors)
- Cx: Convolutional layer
- Sx: Subsample layer
- Fx: Fully connected layer
- Black and White pixel values are normalized:
  E.g. White = -0.1, Black =1.175 (Mean of pixels = 0, Std of pixels =1)

LeNet 5, Layer C1



C1: Convolutional layer with 6 feature maps of size 28x28. $C1_k$ (k=1…6)

Each unit of C1 has a 5x5 receptive field in the input layer.

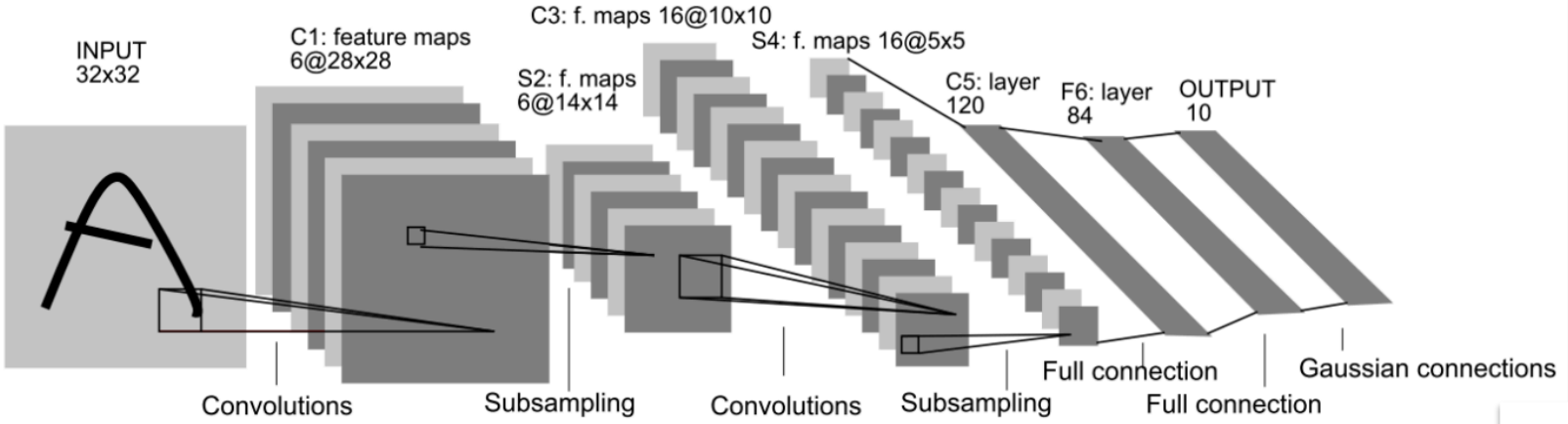
- Topological structure
- Sparse connections
- Shared weights

(5*5+1)*6=156 parameters to learn

Connections: 28*28*(5*5+1)*6=122304

If it was fully connected we had (32*32+1)*(28*28)*6 parameters

LeNet 5, Layer S2



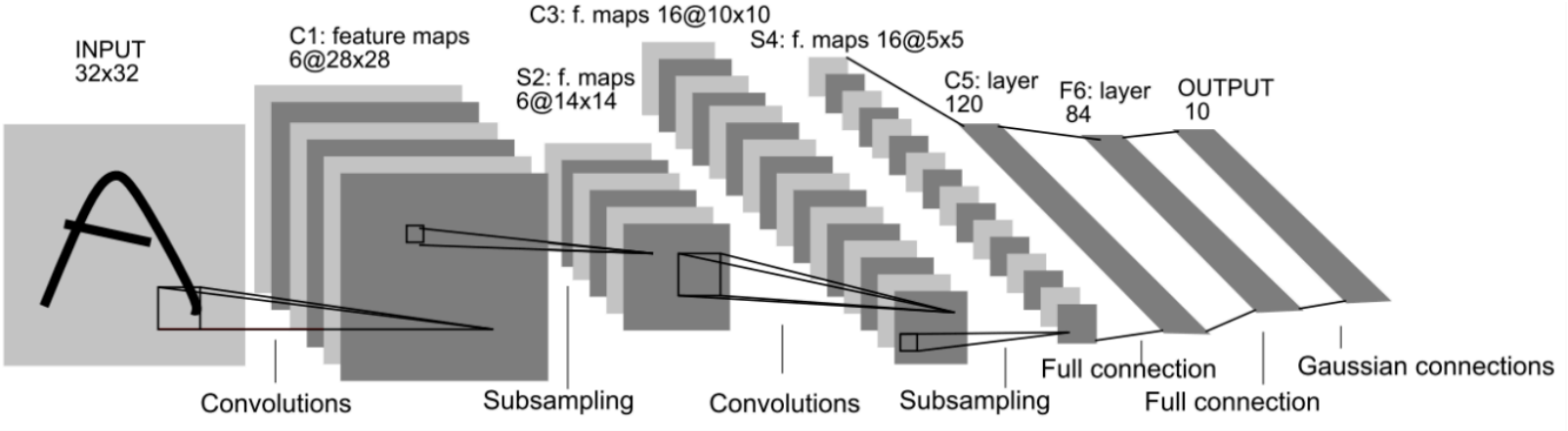
S2: Subsampling layer with 6 feature maps of size 14x14

2x2 nonoverlapping receptive fields in C1

Layer S2: 6*2=12 trainable parameters.

Connections: 14*14*(2*2+1)*6=5880

LeNet 5, Layer C3



- C3: Convolutional layer with 16 feature maps of size 10x10
- Each unit in C3 is connected to several! 5x5 receptive fields at identical locations in S2

Layer C3:

1516 trainable parameters.

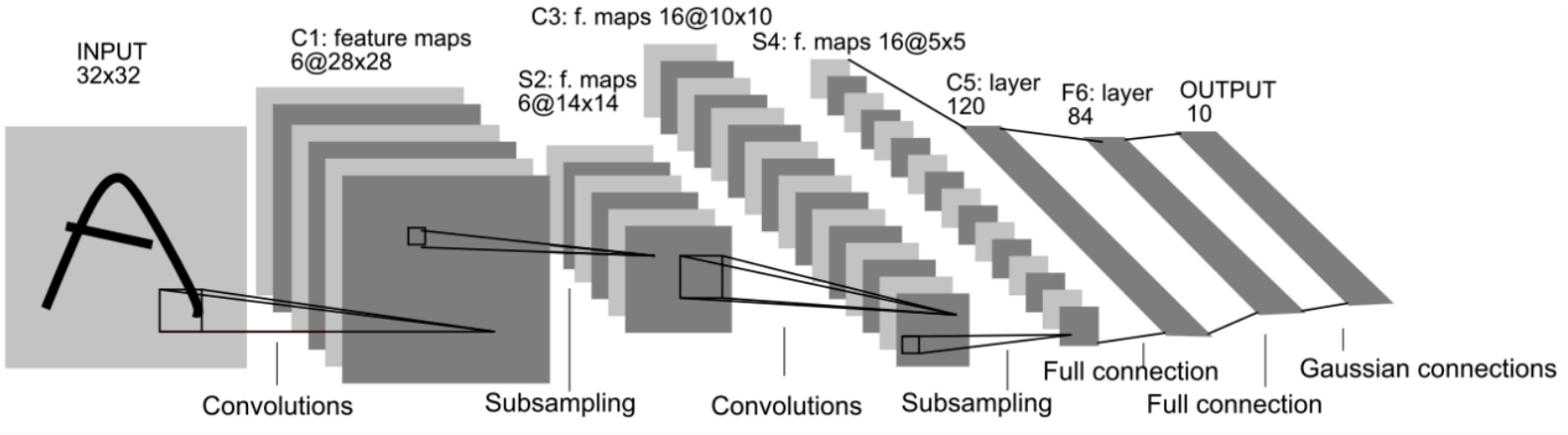
Connections: 151600

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

LeNet 5, Layer S4

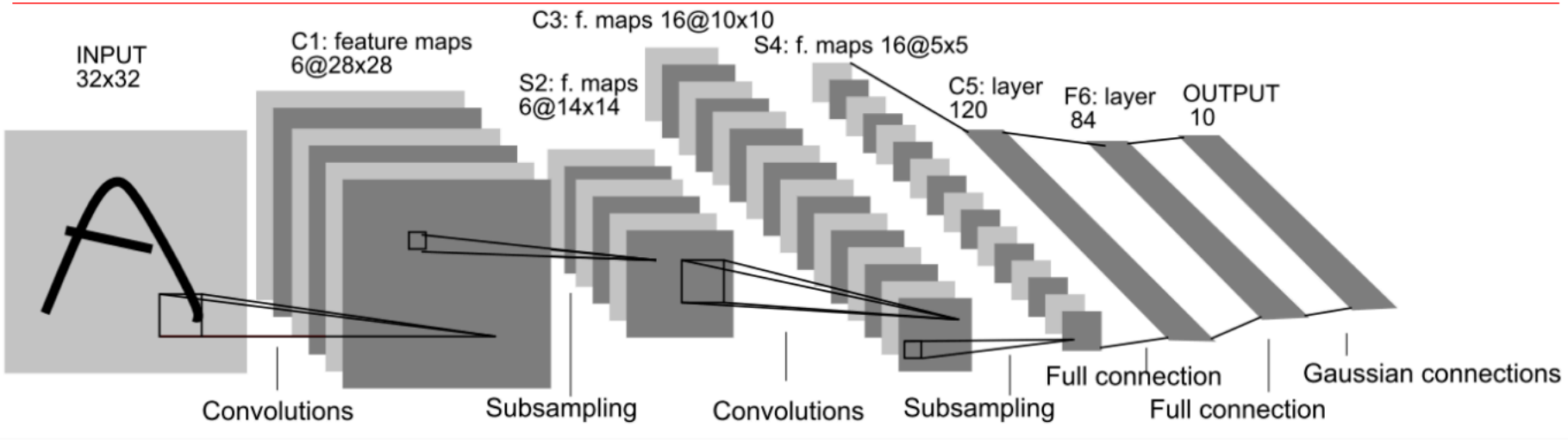


- S4: Subsampling layer with 16 feature maps of size 5x5
- Each unit in S4 is connected to the corresponding 2x2 receptive field at C3

Layer S4: 16*2=32 trainable parameters.

Connections: 5*5*(2*2+1)*16=2000

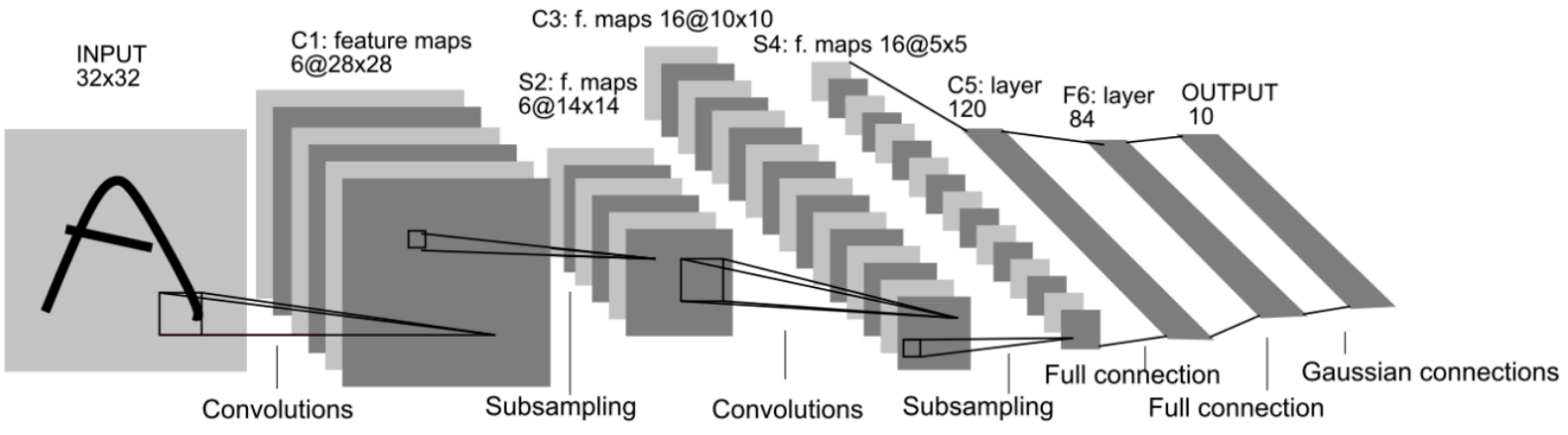
LeNet 5, Layer C5



- C5: Convolutional layer with 120 feature maps of size 1x1
- Each unit in C5 is connected to all 16 5x5 receptive fields in S4

Layer C5: 120*(16*25+1) = 48120 trainable parameters and connections
(Fully connected)

LeNet 5, Layer C5



Layer F6: 84 fully connected units. 84*(120+1)=10164 trainable parameters and connections.

Output layer: 10RBF (One for each digit)

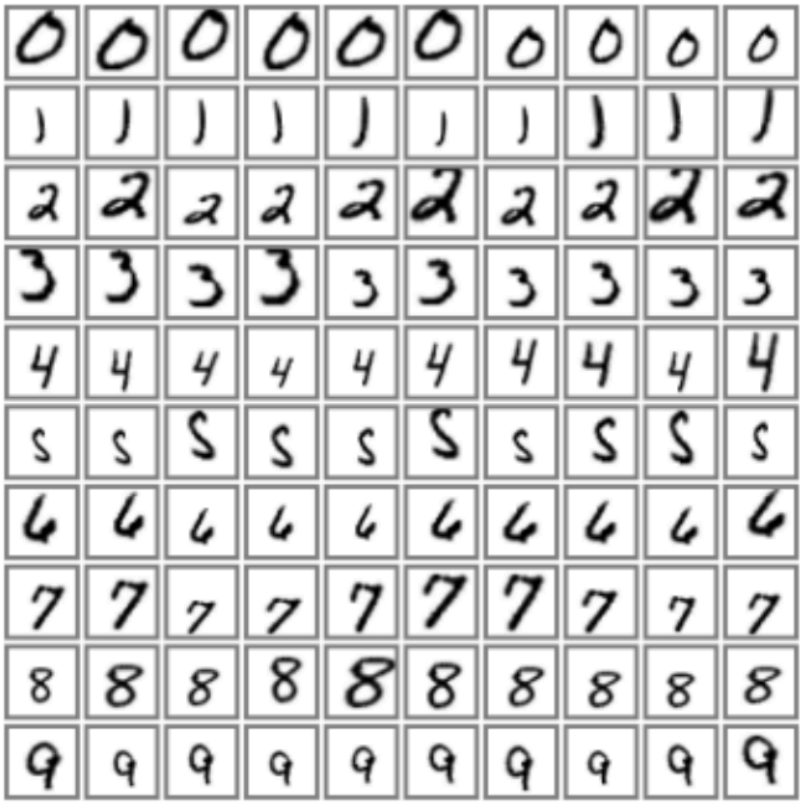
84=7x12, stylized image

Weight update: Backpropagation

MINIST Dataset

60,000 original datasets

Test error: 0.95%



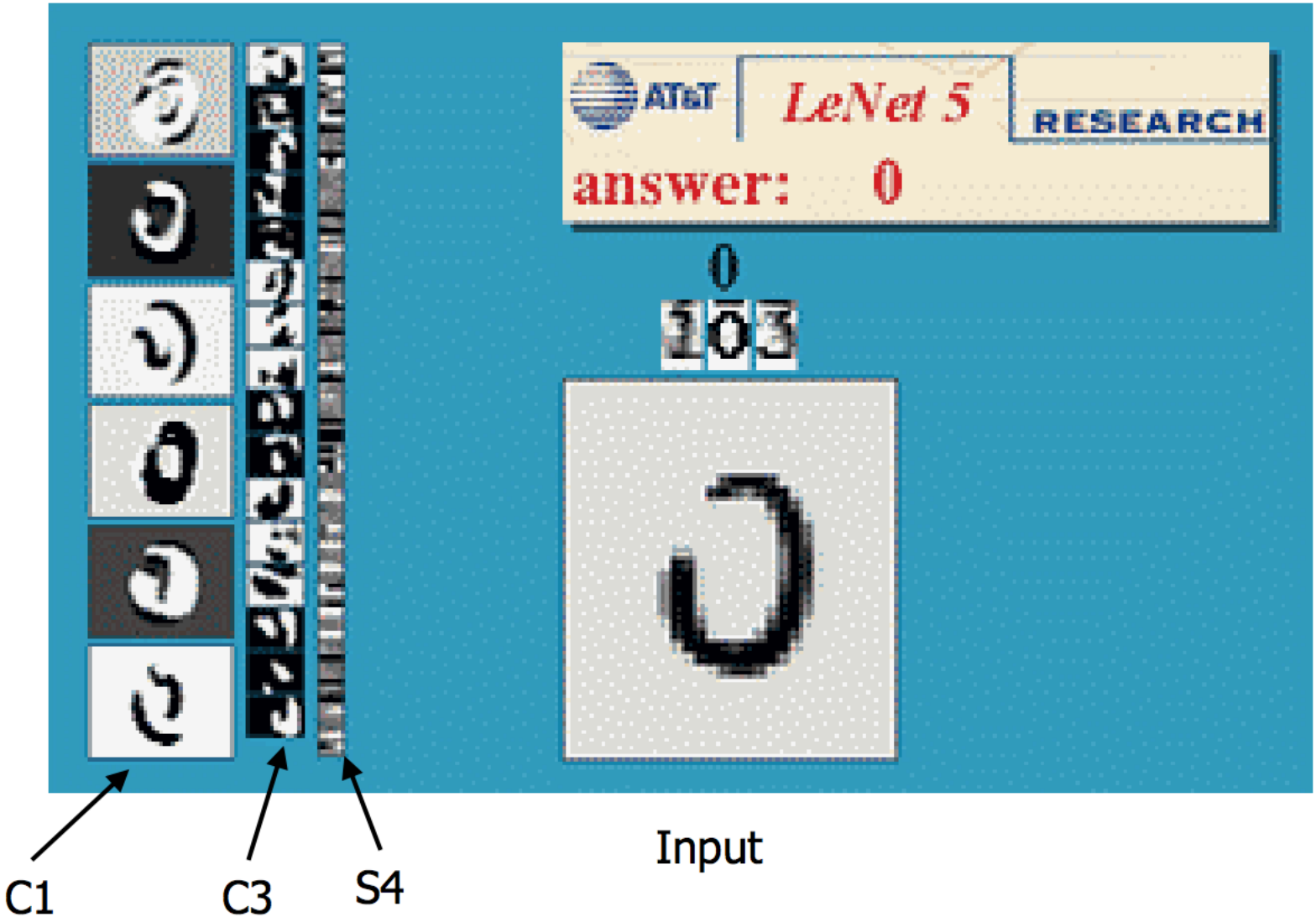
540,000 artificial distortions

+ 60,000 original

Test error: 0.8%

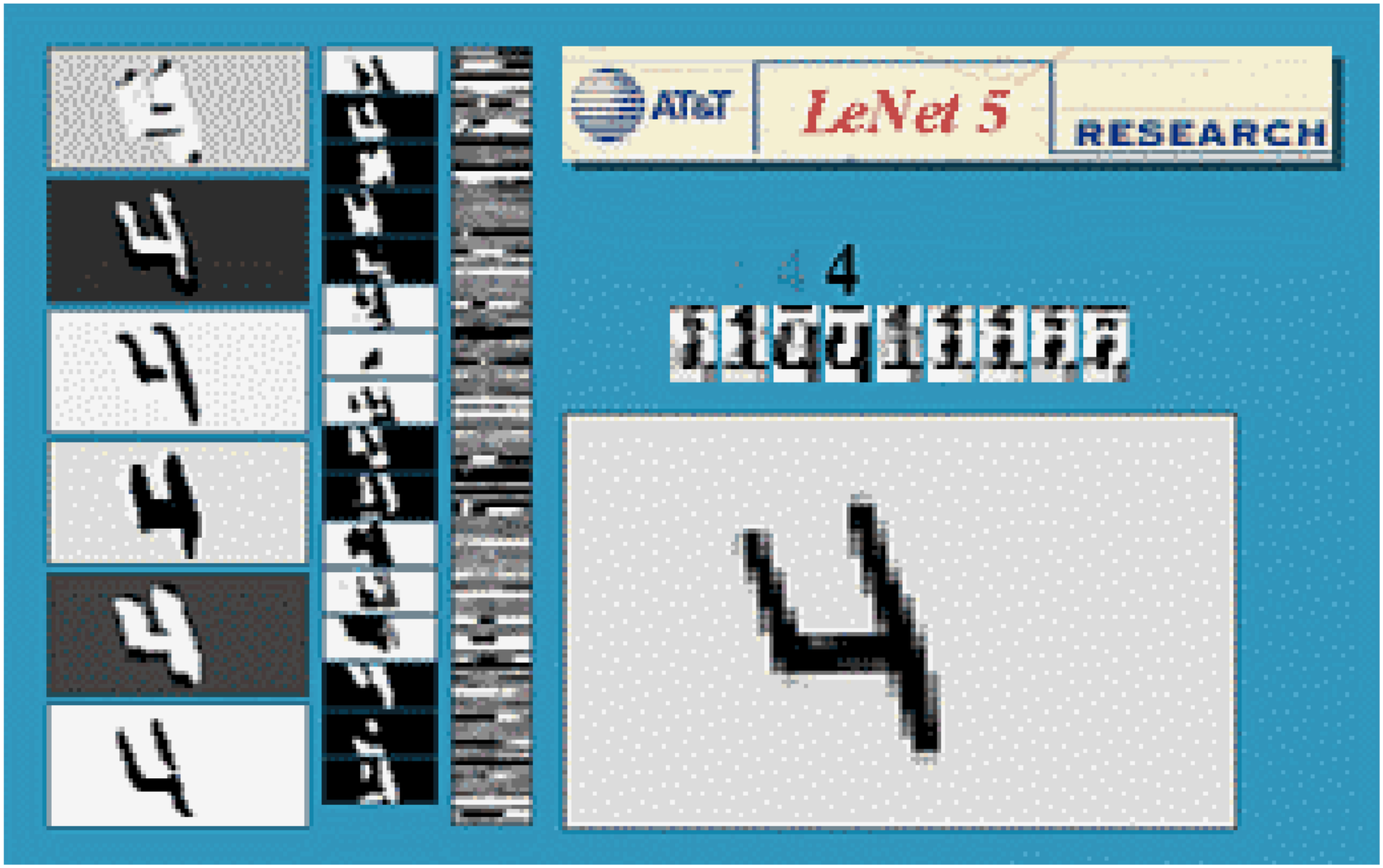
Misclassified Examples

LeNet 5 Processing

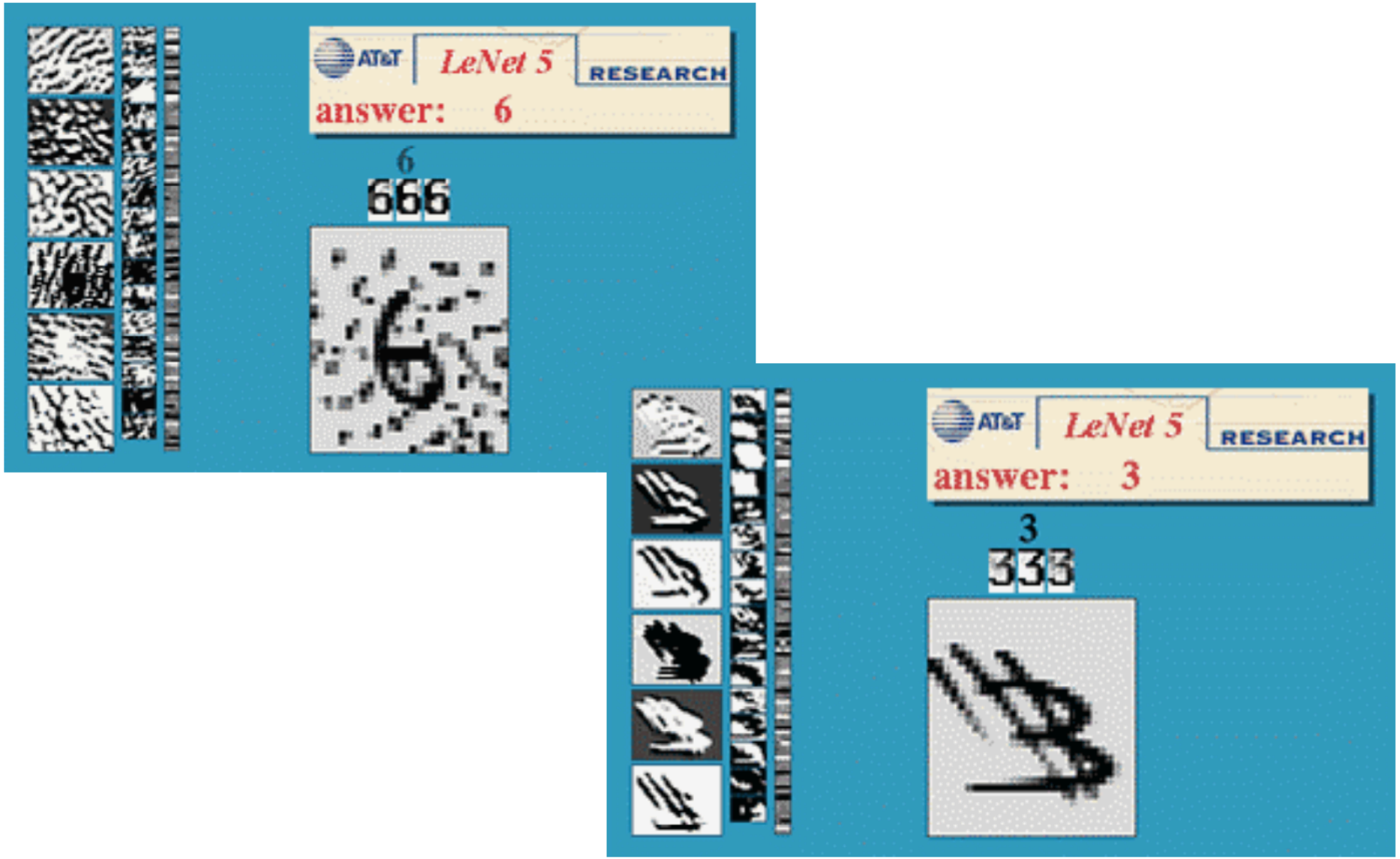


LeNet 5 Shift Invariance

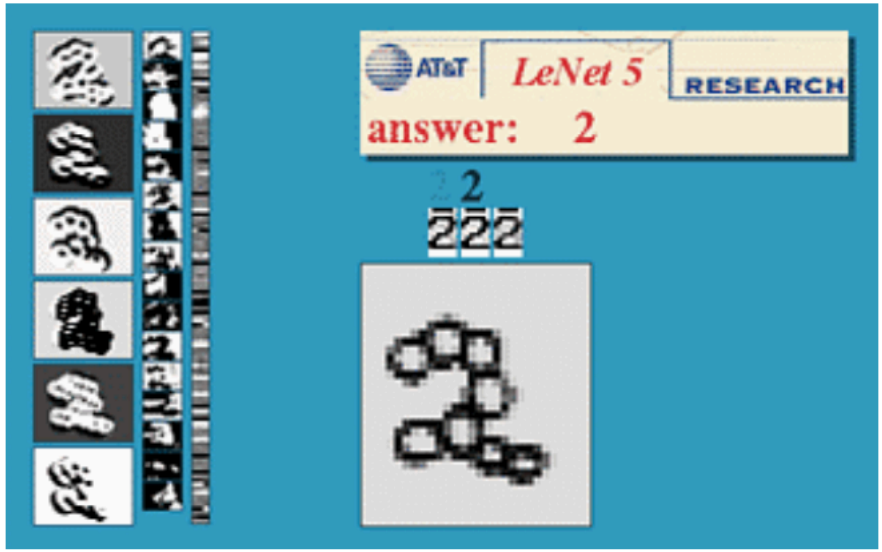
LeNet 5 Rotation Invariance

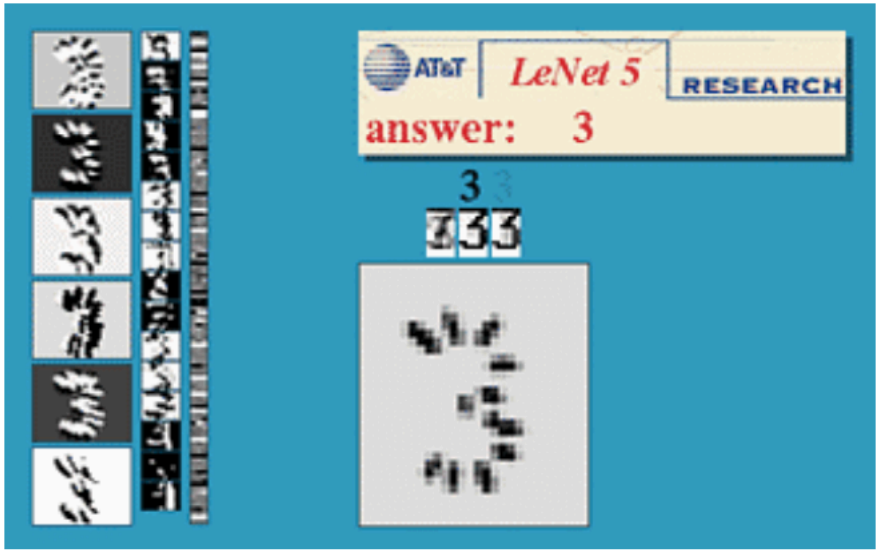


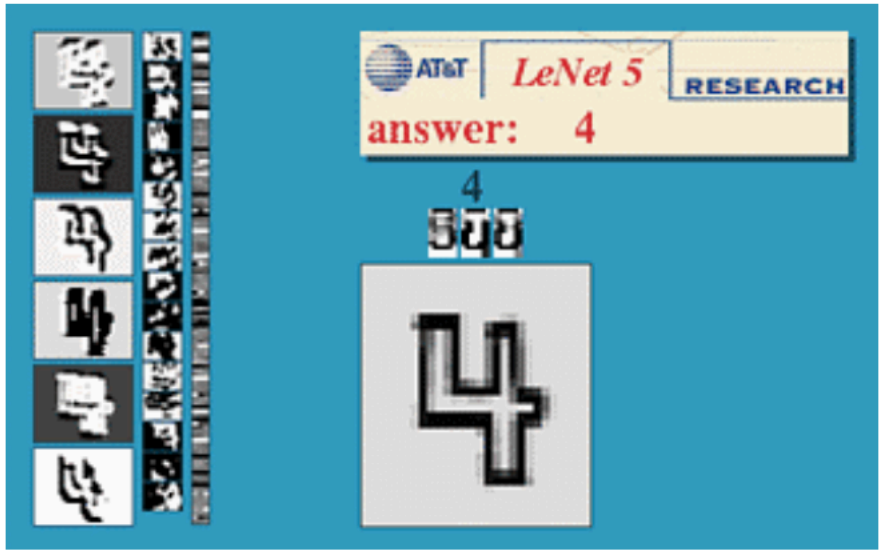
LeNet 5 Noise Invariance



LeNet 5 Unusual Patterns







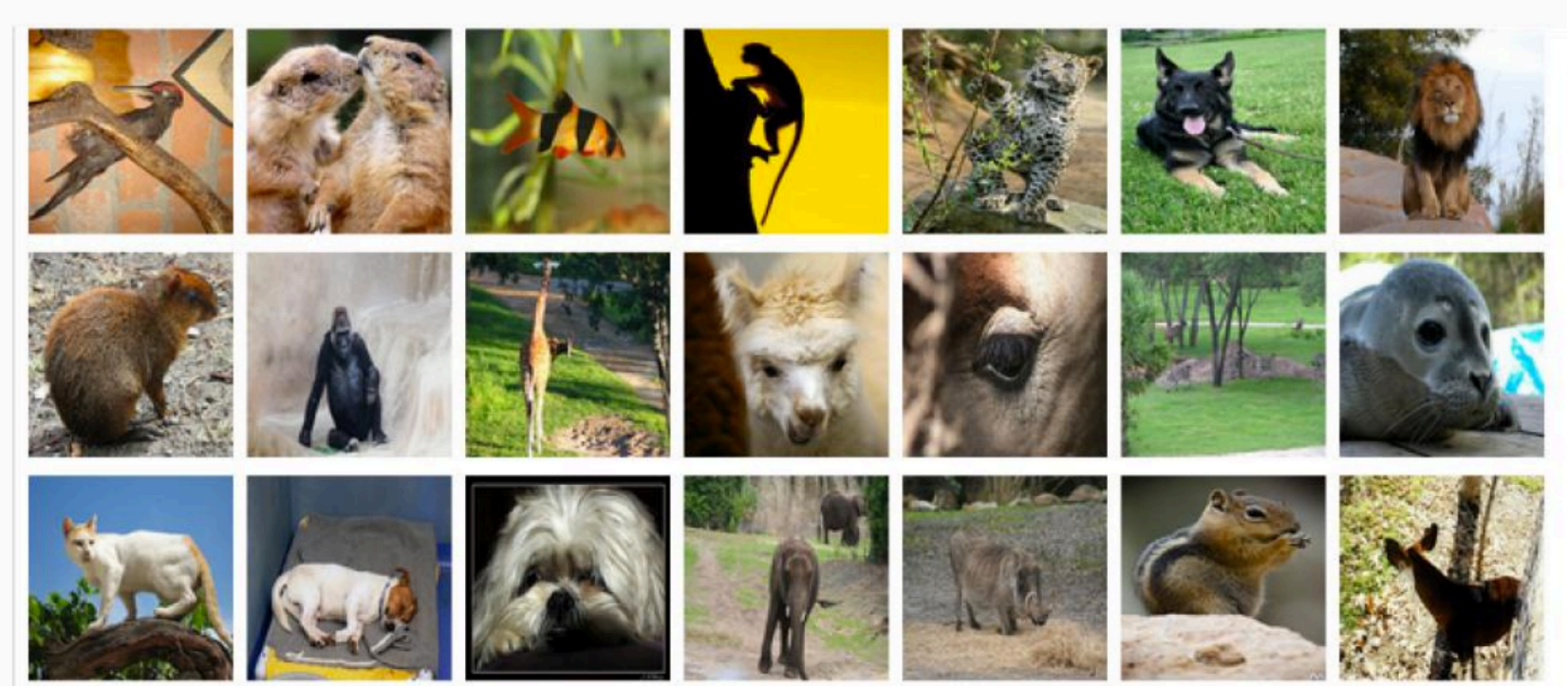
ImageNet Classification with CNN

- **Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton, Advances in Neural Information Processing Systems 2012**
  - 15M images
  - 22K categories
  - Images collected from Web
  - Human labelers (Amazon's Mechanical Turk crowd-sourcing)
  - ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010)
    - 1K categories
    - 1.2M training images (~1000 per category)
    - 50,000 validation images
    - 150,000 testing images
  - RGB images
  - Variable-resolution, but this architecture scales them to 256x256 size

ImageNet

Classification goals:

- ❑ Make 1 guess about the label (Top-1 error)
- ❑ make 5 guesses about the label (Top-5 error)



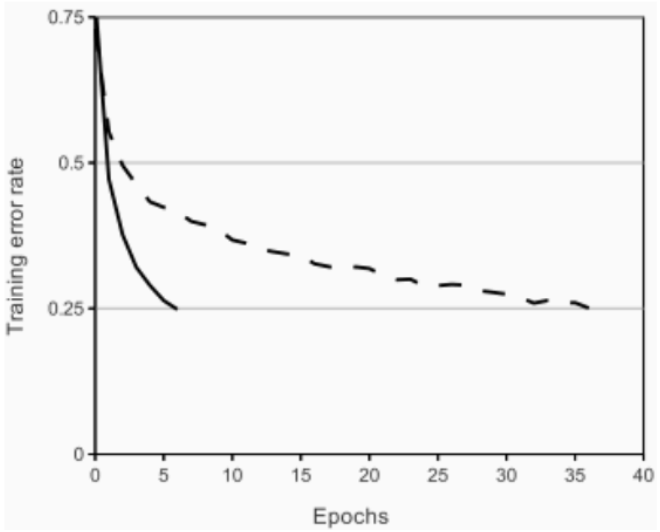
Architecture

Typical nonlinearities: $f(x) = \tanh(x)$

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

Here, however, Rectified Linear Units (ReLU) are used: $f(x) = \max(0, x)$

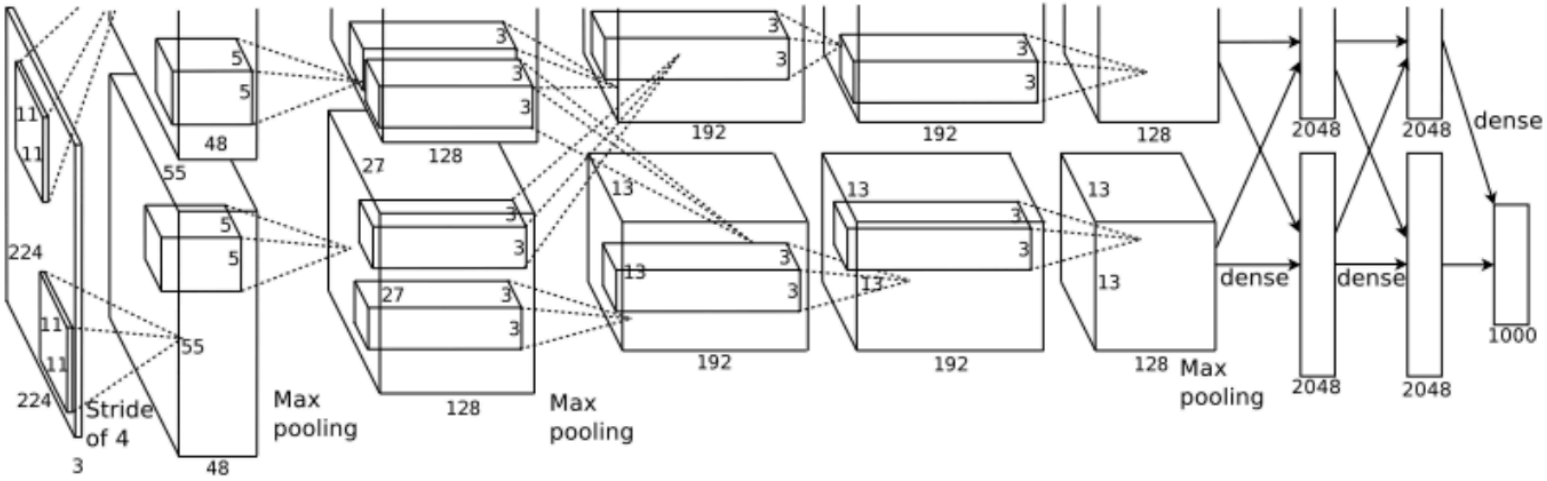
Empirical observation: Deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units



A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

(dashed line)

Architecture



The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in the kernel map. 224/4=56

The pooling layer: form of non-linear down-sampling. Max-pooling partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum value

Architecture

- Trained with stochastic gradient descent
- on two NVIDIA GTX 580 3GB GPUs
- for about a week

- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- 5 convolutional layer, 3 fully connected layer
- Final feature layer: 4096-dimensional

Data Augmentation

The easiest and most common method to **reduce overfitting** on image data is to artificially **enlarge the dataset** using label-preserving transformations.

We employ two distinct forms of data augmentation:

- image translation
- horizontal reflections
- changing RGB intensities

Dropout

- We know that combining different models can be very useful (Mixture of experts, majority voting, boosting, etc)

- Training many different models, however, is very time consuming.

The solution:

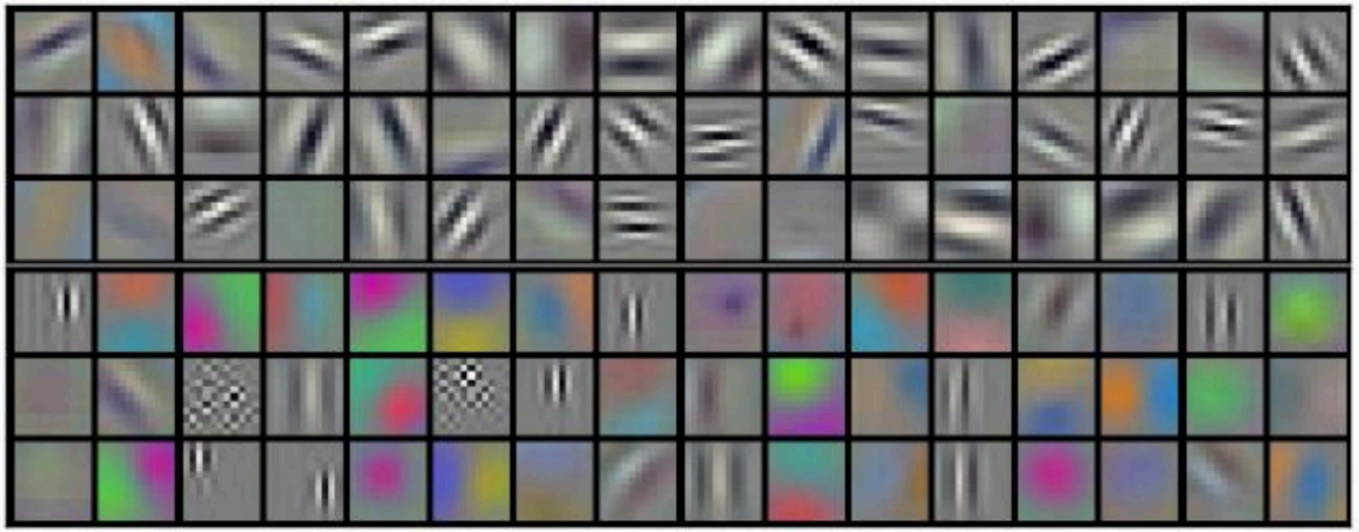
Dropout: set the output of each hidden neuron to zero w.p. 0.5.

Dropout

Dropout: set the output of each hidden neuron to zero w.p. 0.5.

- The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in backpropagation.
- So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.
- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Without dropout, our network exhibits substantial overfitting.
- Dropout roughly doubles the number of iterations required to converge.

First Convolution Layer



96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images.

The top 48 kernels were learned on GPU1 while the bottom 48 kernels were learned on GPU2

Looks like Gabor wavelets, ICA filters…

Results

Results on the test data:

top-1 error rate: 37.5%

top-5 error rate: 17.0%

ILSVRC-2012 competition:

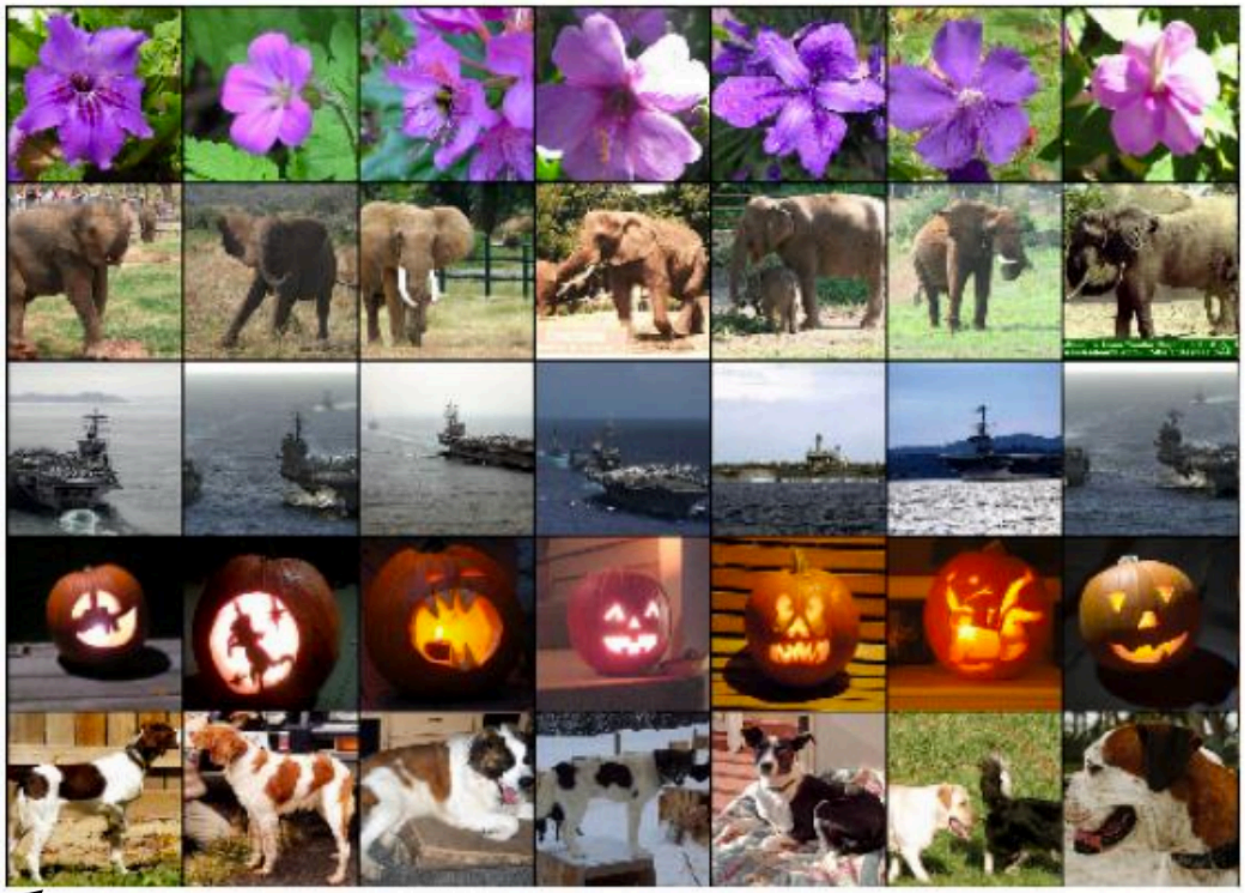
15.3% accuracy

2nd best team: 26.2% accuracy

Results


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

Results



Test column

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.