ENGR 4399
Machine Learning

# Lesson 14: Deep Learning - Convolutional Neural Networks

Instructor: Gonzalo D. Parra, PhD

# Lesson 14

- Convolutional Neural Networks (CNNs)
  - Introduction
  - Applications
  - Implementation

# Today in Deep Learning..

# ImageNet Roulette

ImageNet Roulette uses a neural network trained on the "people" categories from the ImageNet dataset to classify pictures of people. It's meant to be a peek into the politics of classifying humans in machine learning systems and the data they're trained on.

ImageNet Roulette isn't designed to handle heavy traffic so if it's not working for you please be a little patient.

Start Webcam or Provide an image URL Classify image from URL

or upload an image:

Choose File no file selected

**NOTES AND WARNING:**
ImageNet Roulette regularly classifies people in dubious and cruel ways. This is because the underlying training data contains those categories (and pictures of people that have been labelled with those categories). We did not make the underlying training data responsible for these classifications. We imported the categories and training images from a popular data set called ImageNet, which was created at Princeton and Stanford University and which is a standard benchmark used in image classification and object detection.

ImageNet Roulette is meant in part to demonstrate how various kinds of politics propagate through technical systems, often without the creators of those systems even being aware of them.

**TECHINCAL:**
ImageNet Roulette uses a Caffe model trained on the "people" categories from the popular ImageNet dataset (in ImageNet these nodes are all found under the "person, individual, someone, somebody, mortal, soul" class in the WordNet hierarchy; proper nouns have been removed). When you upload a photo, we first run a face detector to find any faces. If we find faces, we send up to 10 of them to the Caffe model. The image you get back shows the face bounding box with the Caffe label. If no faces are detected, we send the entire scene to the Caffe model.

***ImageNet Roulette does not store the photos people upload or any other data.***

**CREDITS:**
Software developed by Leif Ryge for Trevor Paglen Studio

https://imagenet-roulette.paglen.com

# IMAGENET ILSVRC

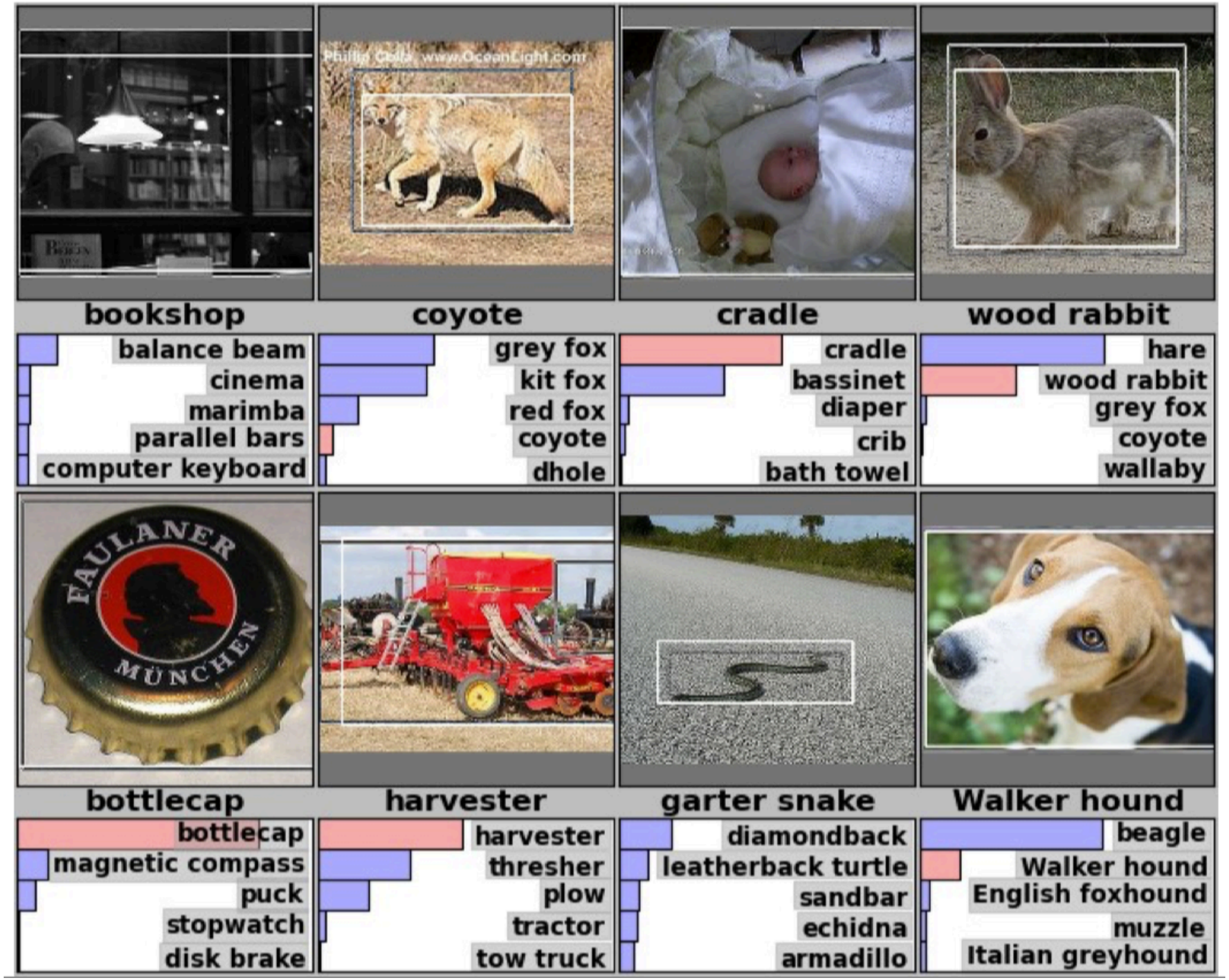
- Annual competition: large scale image classification
- Challenges:
  - **Object localization**
    - *1.2 million images in 1000 categories provided, guess 5 best labels*

    ILSVRC 2012
  - **Object detection (image)**
    - *200 basic-level categories, fully annotated on test data via bounding boxes*
    - *Task: produce a set of annotations: class labels, confidence scores, & bounding boxes*

    ILSVRC 2014/16
  - **Object detection (video)** - 30 fully labeled categories
    - *30 basic-level categories, fully annotated on test data*
    - *Task: produce a set of annotations: frame number, class labels, confidence scores, & bounding boxes*

    ILSVRC 2017

http://www.image-net.org/challenges/LSVRC/


bookshop
balance beam
cinema
marimba
parallel bars
computer keyboard
coyote
grey fox
kit fox
red fox
coyote
dhole
cradle
cradle
bassinet
diaper
crib
bath towel
wood rabbit
hare
wood rabbit
grey fox
coyote
wallaby
bottlecap
bottlecap
magnetic compass
puck
stopwatch
disk brake
harvester
harvester
thresher
plow
tractor
tow truck
garter snake
diamondback
leatherback turtle
sandbar
echidna
armadillo
Walker hound
beagle
Walker hound
English foxhound
muzzle
Italian greyhound

# CNNs

# Motivation

- Problem:
  - **How to use neural networks to extract features from <u>images</u>?**
- Solution:
  - **Models with large learning capacity and prior knowledge to compensate..**
- Goal:
  - **A new category of model, capable of more dimensions**

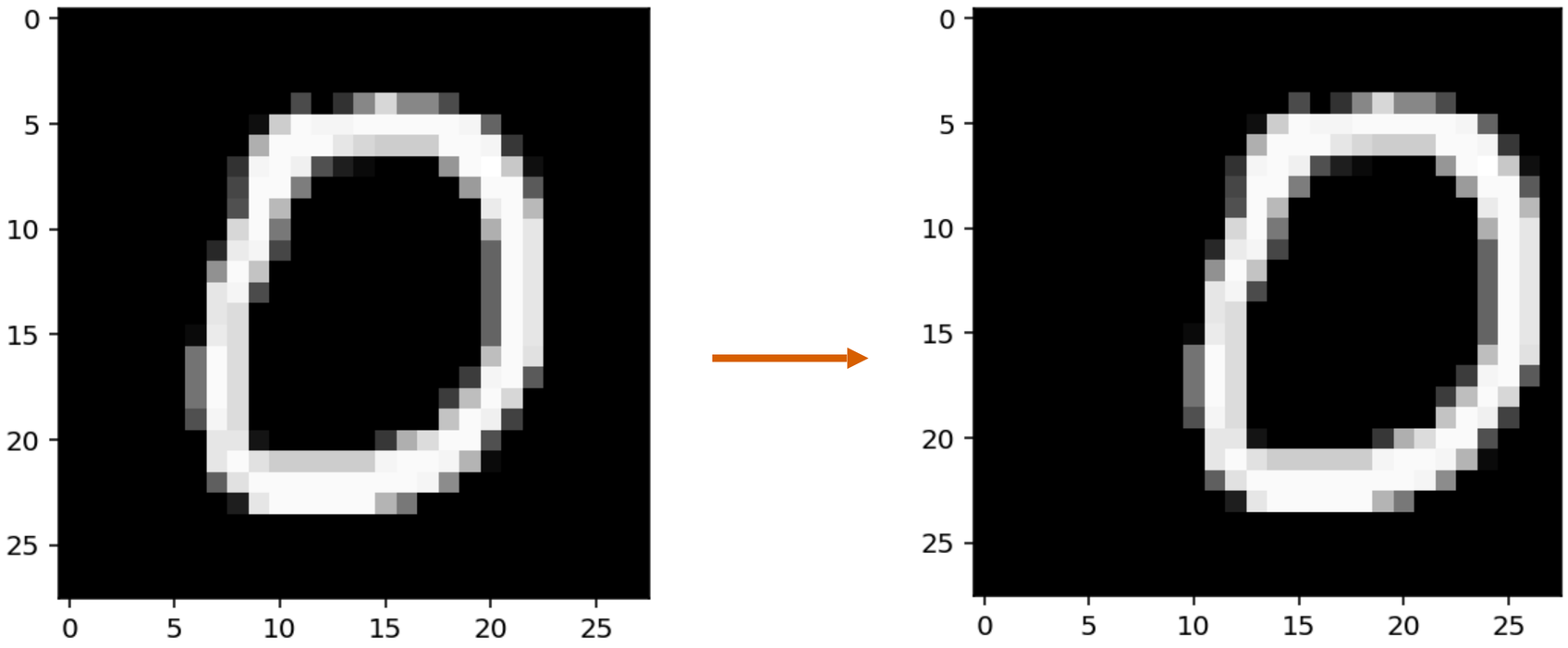
# Motivation

- **Parameter sharing**
    - A feature detector which is useful in 1 part of an image is probably useful in another part of the image.

- **Sparsity of connections**
    - In each layer, each output value depends only on a small number of inputs.

# Motivation

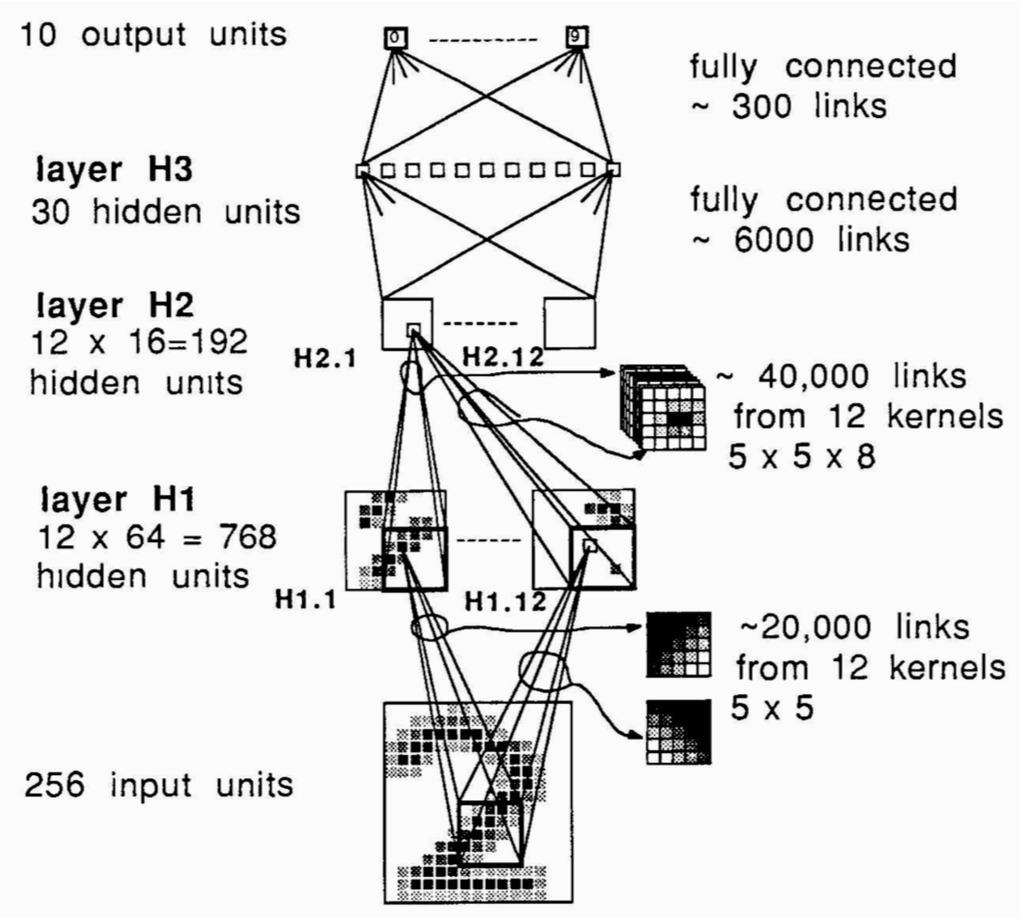
- **Dimensionality reduction**
- **Feature extraction** *with rotation & position invariance*




Source: MNIST

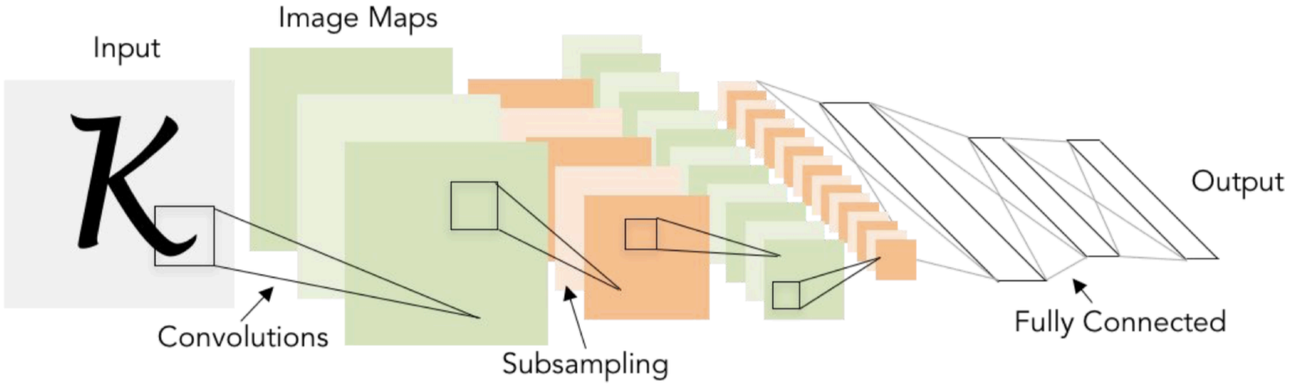
# LeNet

**Zip codes**




*Yann LeCun, et. al., 1989*

**Digits on checks**

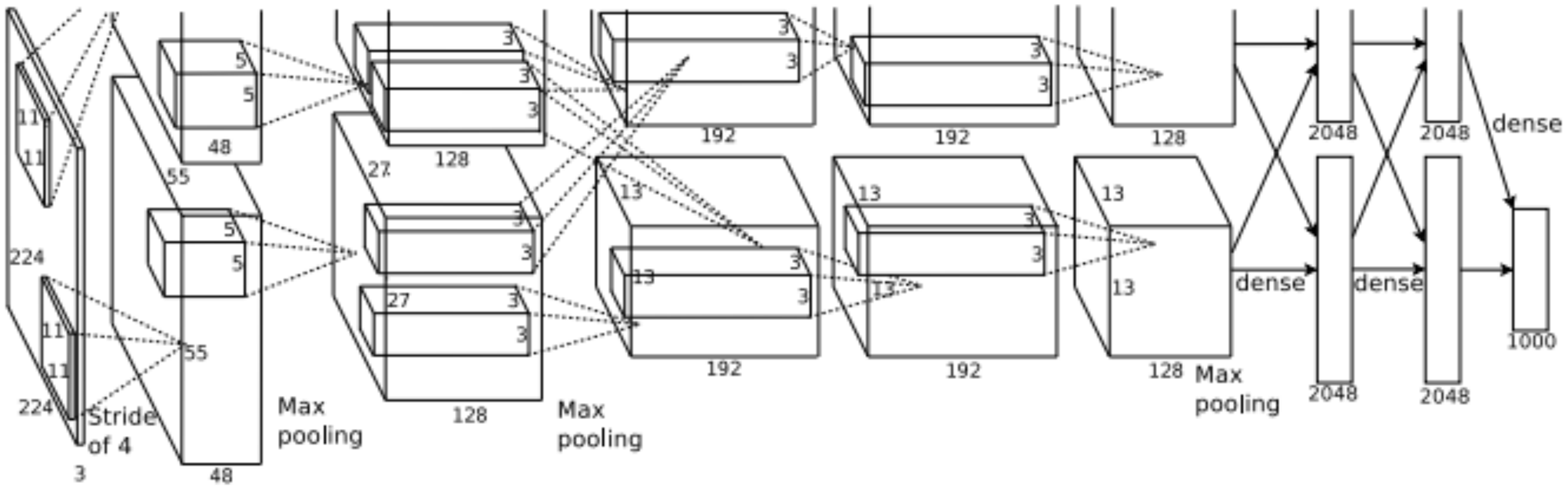



**ILSVRC winner 1998**

7 layer CNN.
# parameters: 60k

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition; AT&T Bell Laboratories, Neural Computation, 1989.

# AlexNet




**ILSVRC winner 2012**
**Top-5 error rate: 15.3%**
(runner up: ~26%)

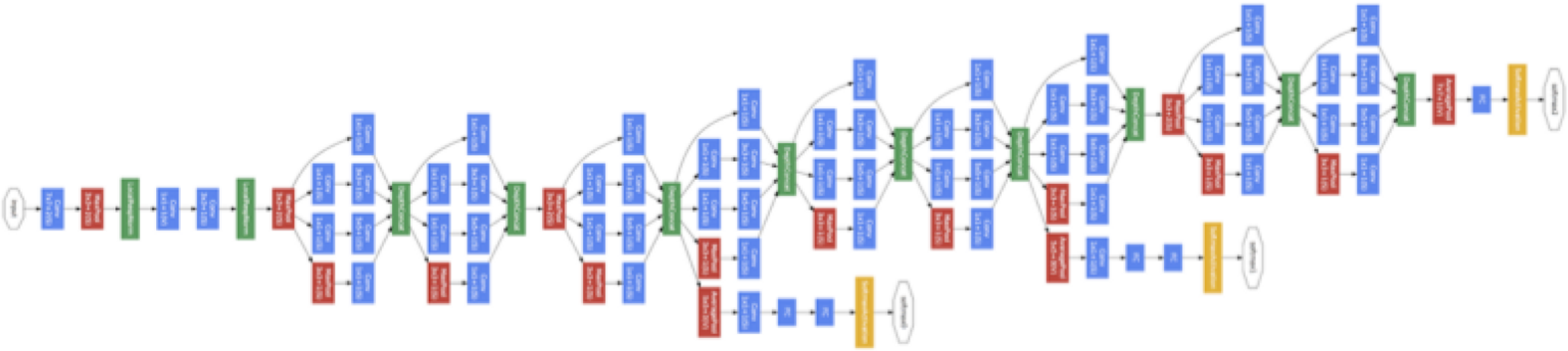
*Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever*
*2012*

7 hidden weight layer deep CNN.
# parameters: 60 million.

Source paper: Krizhevsky et. al., NIPS, 2012

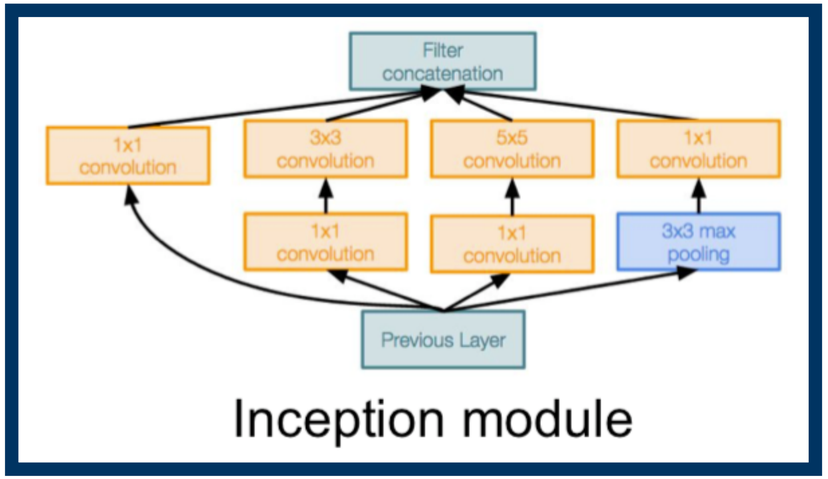
# Inception

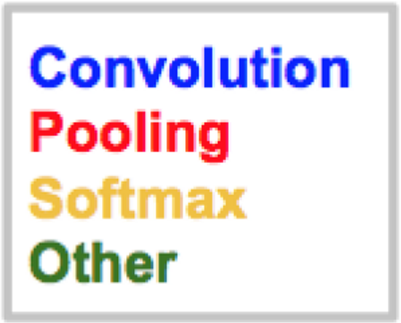
GoogleNet
GoogLeNet
Inception V1



**ILSVRC winner 2014**
**Top-5 error rate: 6.67%**

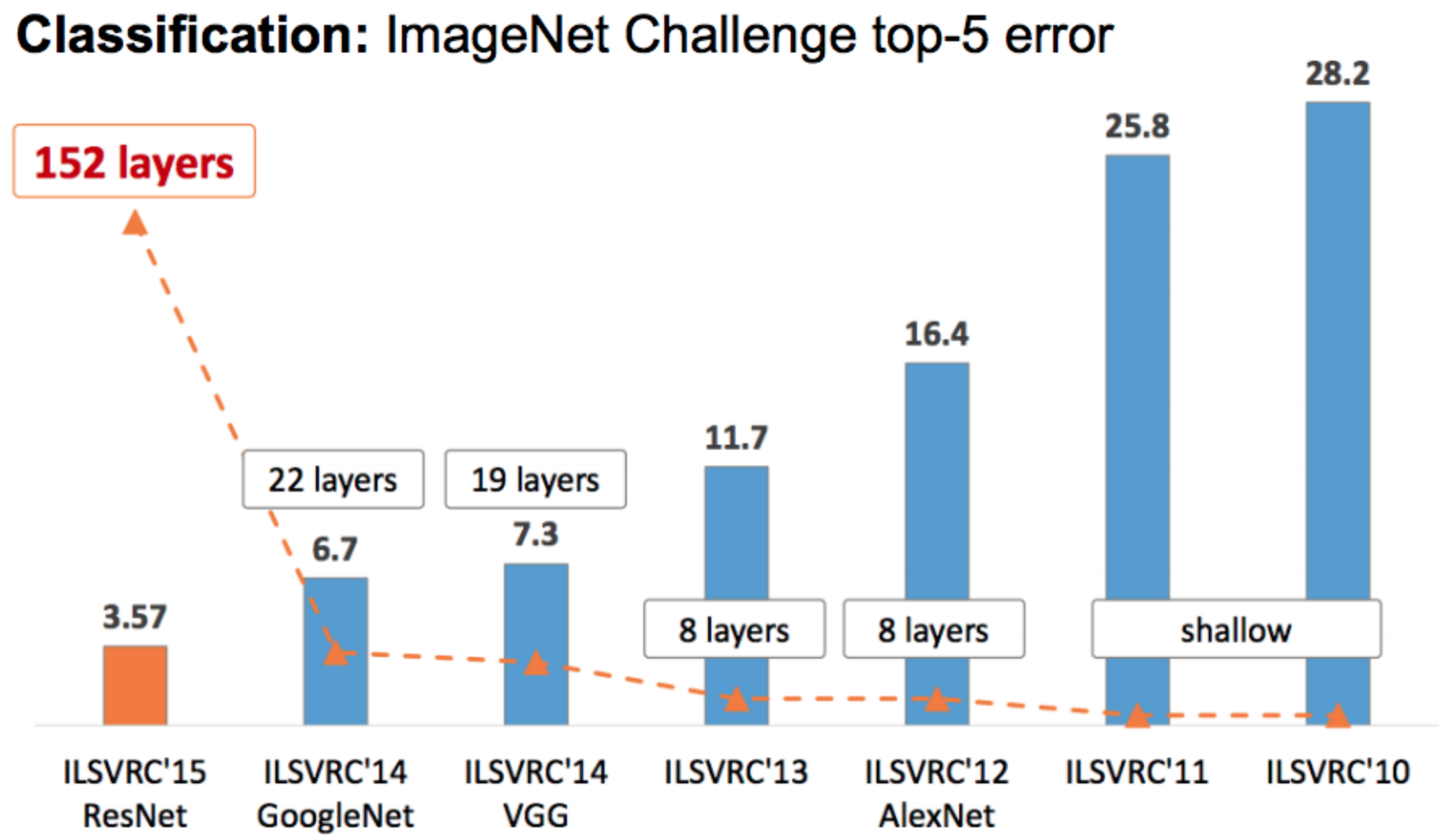
22 layer deep CNN.
# parameters: 4 million.




Source paper: *Going Deeper with Convolutions, Szegedy et al,* https://arxiv.org/abs/1409.4842

# ResNet

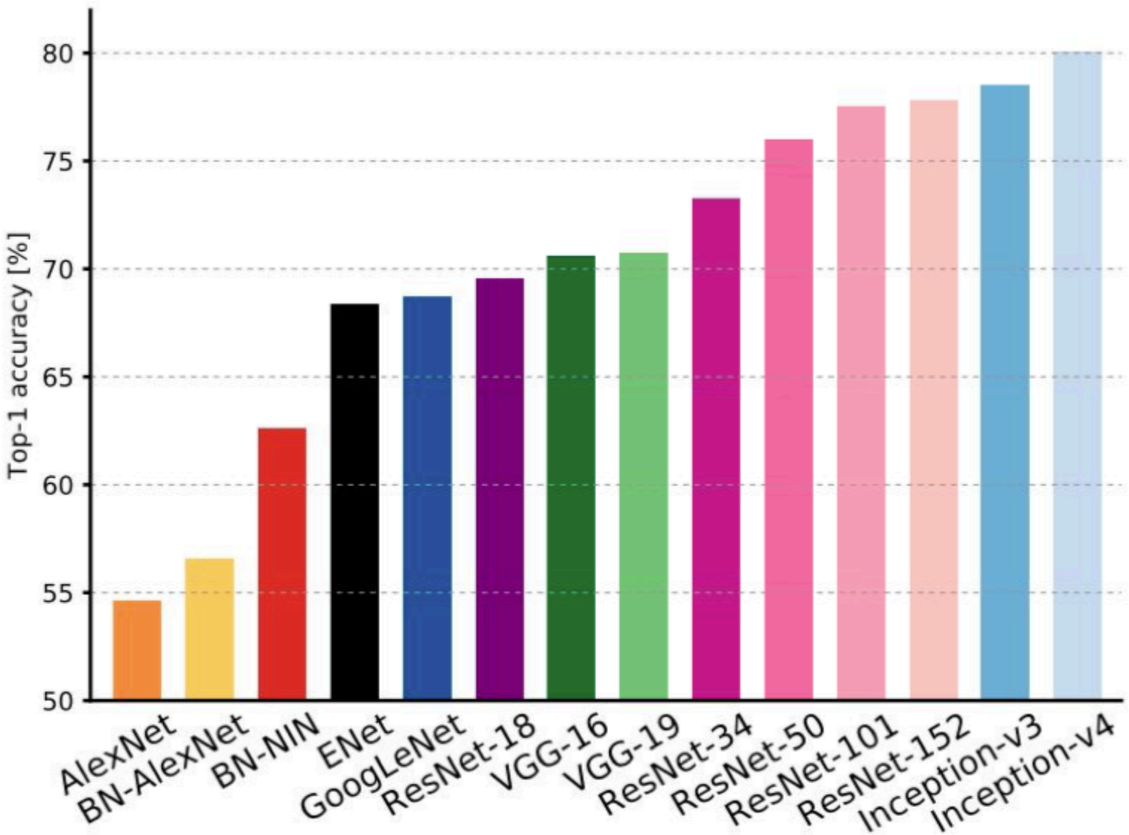


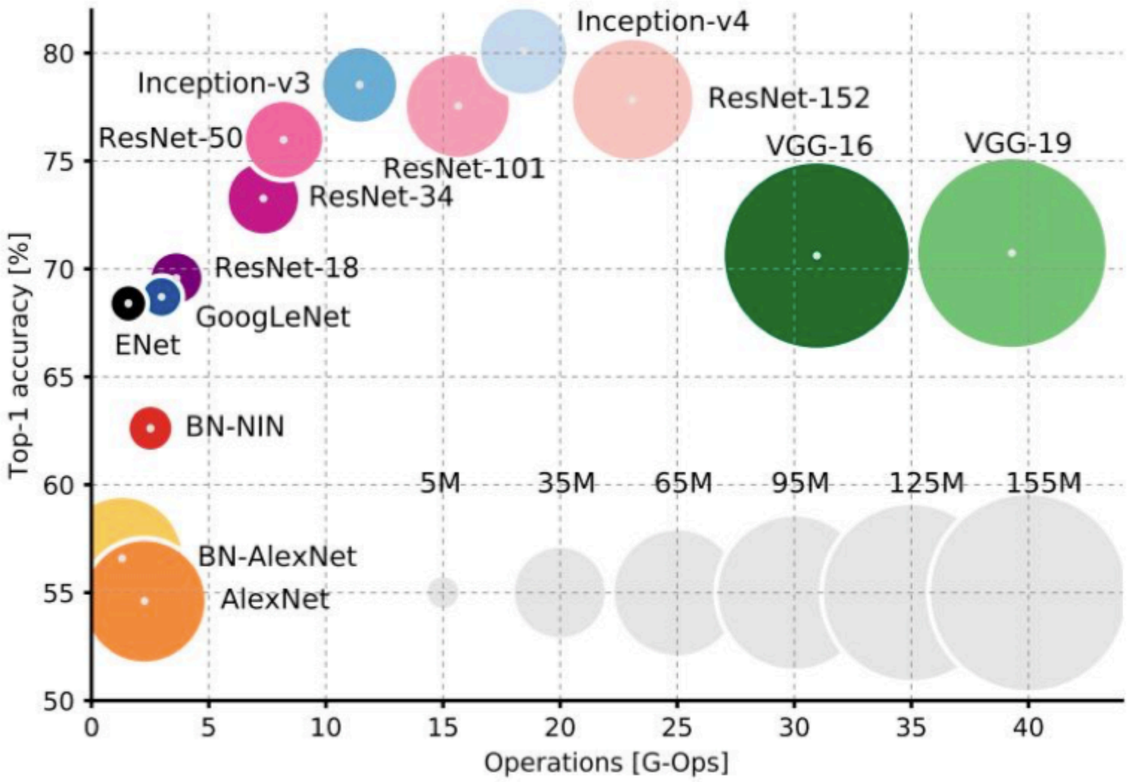

**ILSVRC winner 2015**
**Top-5 error rate: 3.57%**

152 layer deep RNN.
# parameters: 60 million.

Source paper: *Deep Residual Learning for Image Recognition,* He et. al., 2015, https://arxiv.org/abs/1512.03385

# Comparing complexity...



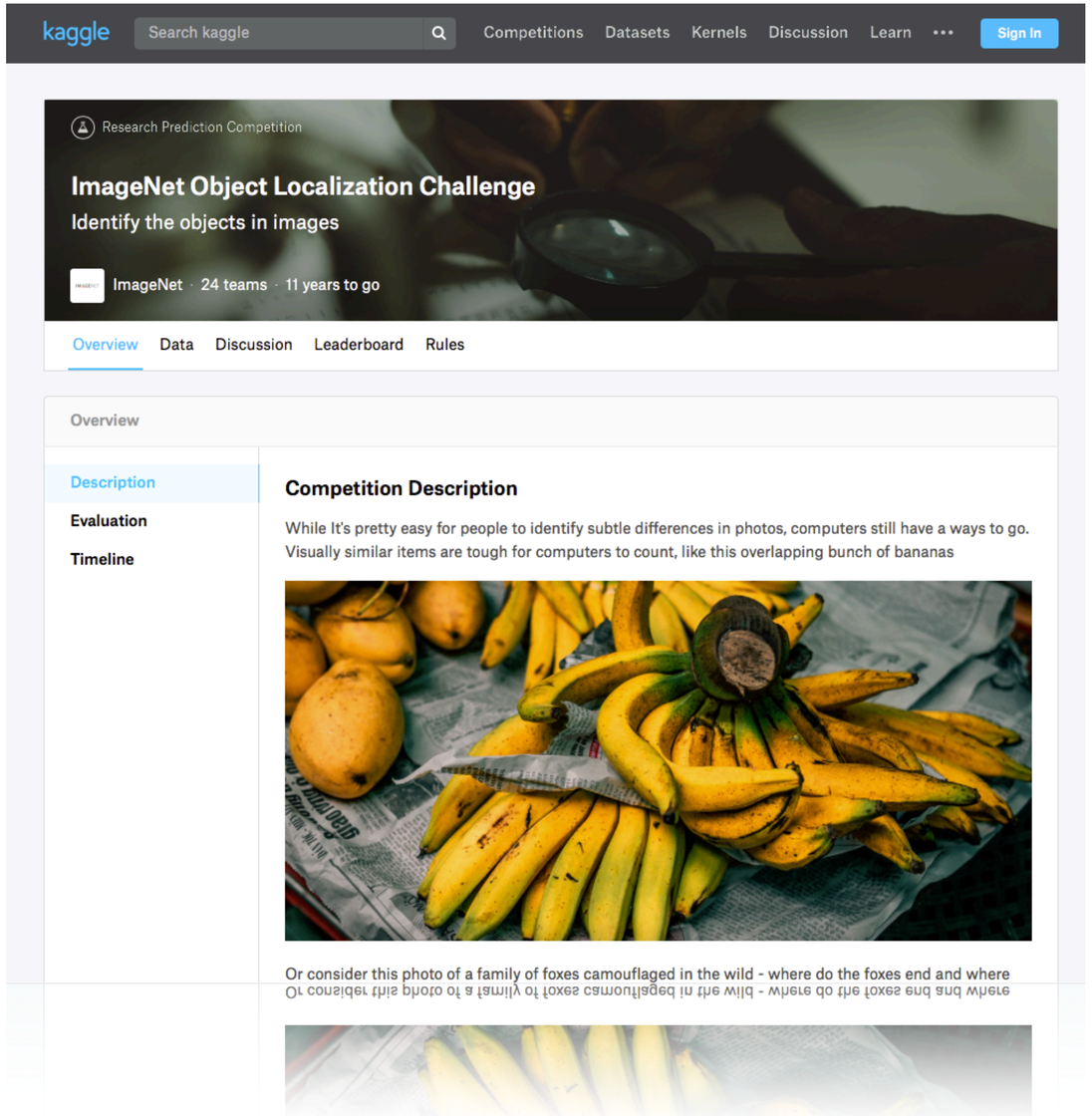

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

https://arxiv.org/abs/1605.07678

# Transfer Learning: CNNs

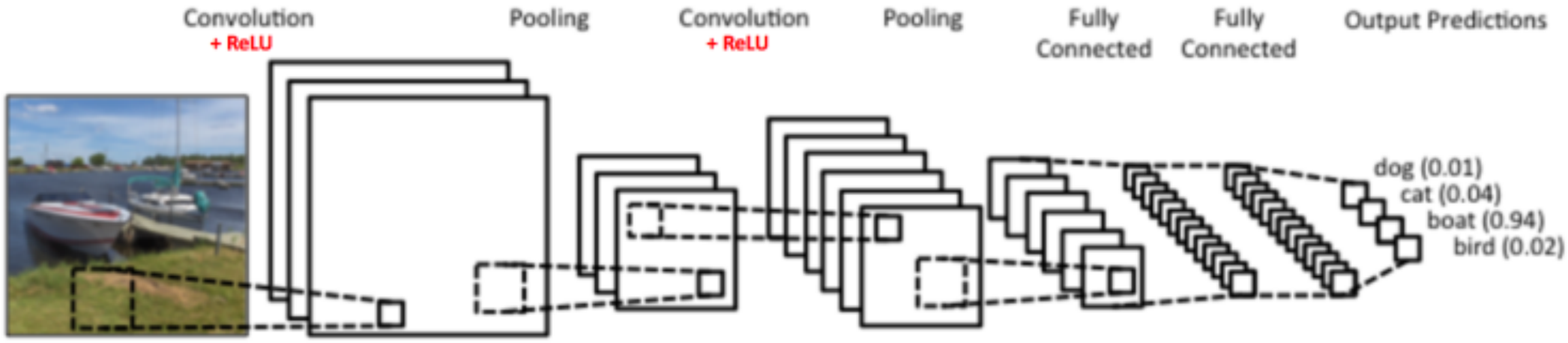
- Pre-trained finely-tuned CNN models
  - AlexNet
  - Inception V3
  - VGG16
  - Resnet50



# CNN Architecture & Components

# Architecture

**Keywords:**
1. **Convolution**
2. **Pooling**




Source: https://www.clarifai.com/technology

# 1. Convolution

**<u>Kernels</u>:** square grid of weights overlaid on image, centered on one pixel, and moved around the image

Each weight multiplied with pixel underneath it

Output for the centered pixel is

Used for traditional image processing techniques:

- Blur
- Sharpen
- Edge detection
- Emboss

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | | |
| | | |

# Imagine kernel is stacked on top of input

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | | |
| | | |

$$= (3 \cdot -1)$$

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | | |
| | | |

$$= (3 \cdot -1) + (2 \cdot 0)$$

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | | |
| | | |

$$= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1)$$

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | | |
| | | |

$$= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1) + (1 \cdot -2)$$

# Example: 3x3

Input

| | | |
|---|---|---|
| 3 | 2 | 1 |
| 1 | 2 | 3 |
| 1 | 1 | 1 |

Kernel

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Output

| | | |
|---|---|---|
| | | |
| | 2 | |
| | | |

# Here’s what the process looks like over a larger input:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 0 | 3 | 1 |
| 1 | 0 | 0 | 2 | 2 |
| 2 | 1 | 2 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 2 | 1 | 1 | 1 |

input

| | | |
|---|---|---|
| -1 | 1 | 2 |
| 1 | 1 | 0 |
| -1 | -2 | 0 |

kernel

| | | |
|---|---|---|
| -2 | | |
| | | |
| | | |

output

# 1. Convolution (settings)

- **Height & width** - *# pixels the kernel operates on, both must be odd*

- **Stride** - *step size from center to center (also has height/width, generally the same)*
  - *If stride > 1, scales down the output dimensions*

- **Depth** - *# of output channels.*
  - *RGB 3 colors = 3 channels. CMYK 4 colors = 4 channels.*
  - *# of separate kernels needed in a layer*

- **Padding** - *adding pseudo-pixels off the edge of input (all zeros)*
  - *VALID = no padding*
  - *SAME = padding is enough to offset kernel size, output same dim as input*

# Padding: 1 (‘SAME’)

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 2 | 0 | 3 | 1 | 0 |
| 0 | 1 | 0 | 0 | 2 | 2 | 0 |
| 0 | 2 | 1 | 2 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 2 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

input

| | | |
|---|---|---|
| -1 | 1 | 2 |
| 1 | 1 | 0 |
| -1 | -2 | 0 |

kernel

| | | | | |
|---|---|---|---|---|
| -1 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

output

# 2. Pooling

- Goal: reduce neighboring pixels, reducing dim of input

- Approaches:
    - Max
    - Average
    - Global

# Max pooling

| | | | |
|---|---|---|---|
| 2 | 1 | 0 | -1 |
| -3 | 8 | 2 | 5 |
| 1 | -1 | 3 | 4 |
| 0 | 1 | 1 | -2 |

$\xrightarrow{\text{maxpool}}$

| | |
|---|---|
| 8 | 5 |
| 1 | 4 |

# Average pooling

| | | | |
|---|---|---|---|
| 2 | 1 | 0 | -1 |
| -3 | 8 | 2 | 5 |
| 1 | -1 | 3 | 4 |
| 0 | 1 | 1 | -2 |

$\xrightarrow{\text{avgpool}}$

| | |
|---|---|
| 2 | 1.5 |
| 0.25 | 1.5 |

# Global pooling

(Average pool over whole layer)

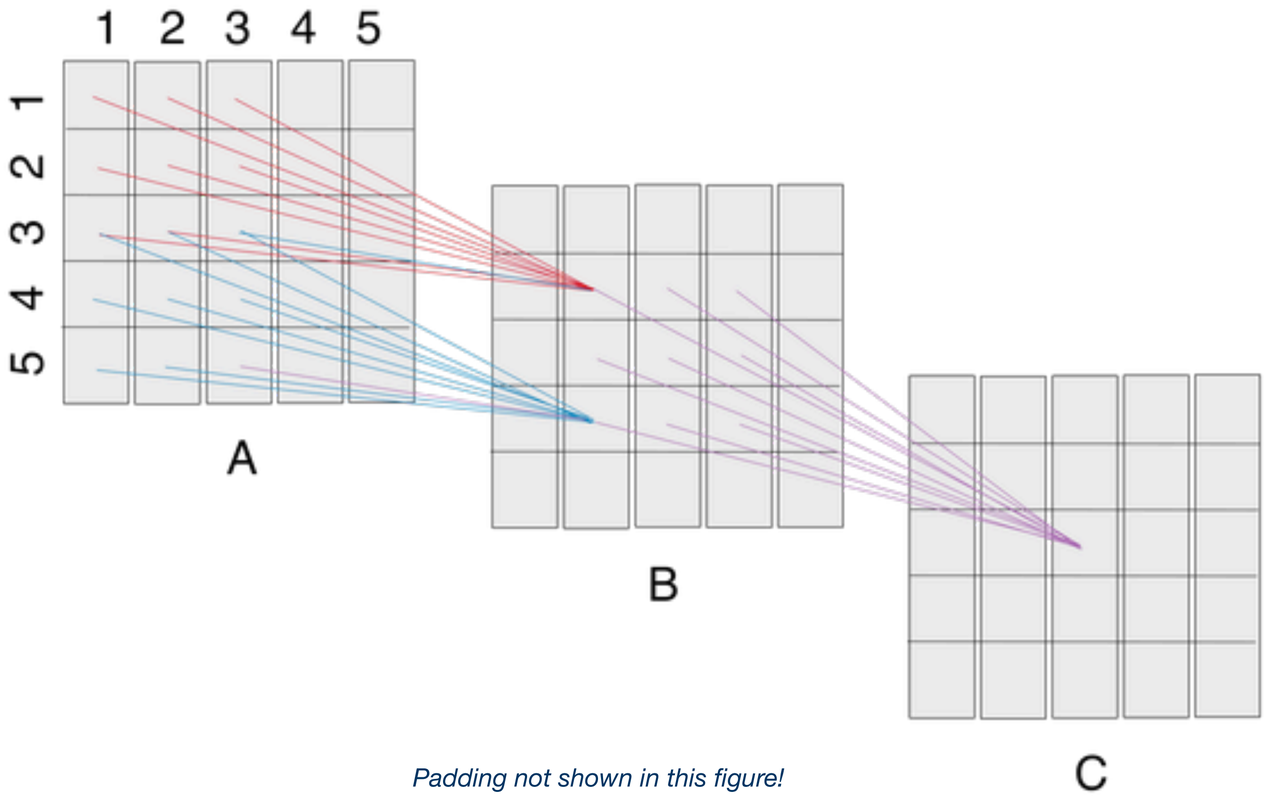
| | | | |
|---|---|---|---|
| 2 | 1 | 0 | -1 |
| -3 | 8 | 2 | 5 |
| 1 | -1 | 3 | 4 |
| 0 | 1 | 1 | -2 |

→ global pool

| |
|---|
| 1.3125 |

# ERF

- The **effective receptive field** of a neuron is the **total spatial area of input**.
- **Consider:** If you stacked conv layers (stride 1), size 3x3, what would be the effective receptive field?




*Padding not shown in this figure!*

# Parameters & Output Volume

- **Parameters**
  - *Total number of parameters to a layer*
  - *These are <u>what is learned</u>*

**f * f * channels * depth**

$$v = \frac{i - f}{stride} + 1$$

**[v x v x depth]**

- **Output volume**
  - *Total size of the output of a layer*

f = filter size
channels = depth of input
depth = # filters (kernel)
i = input size

# Let’s Try AlexNet

ImageNet 2012

- **Input:** 227x227 color images
- **1st Layer:** Conv1
    - 96 filters, size 11x11, stride 4
- **2nd Layer:** Pool2
    - 3x3 filters, stride 2

# LeNet-5

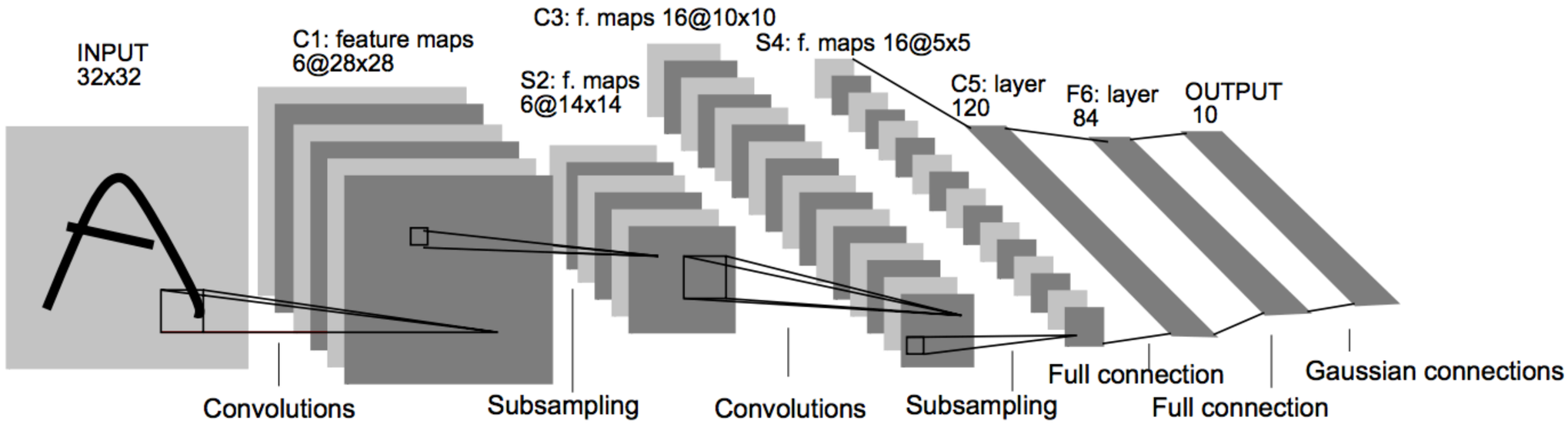
# LeNet

Created by Yann LeCun in the 1990s

Used on the MNIST dataset

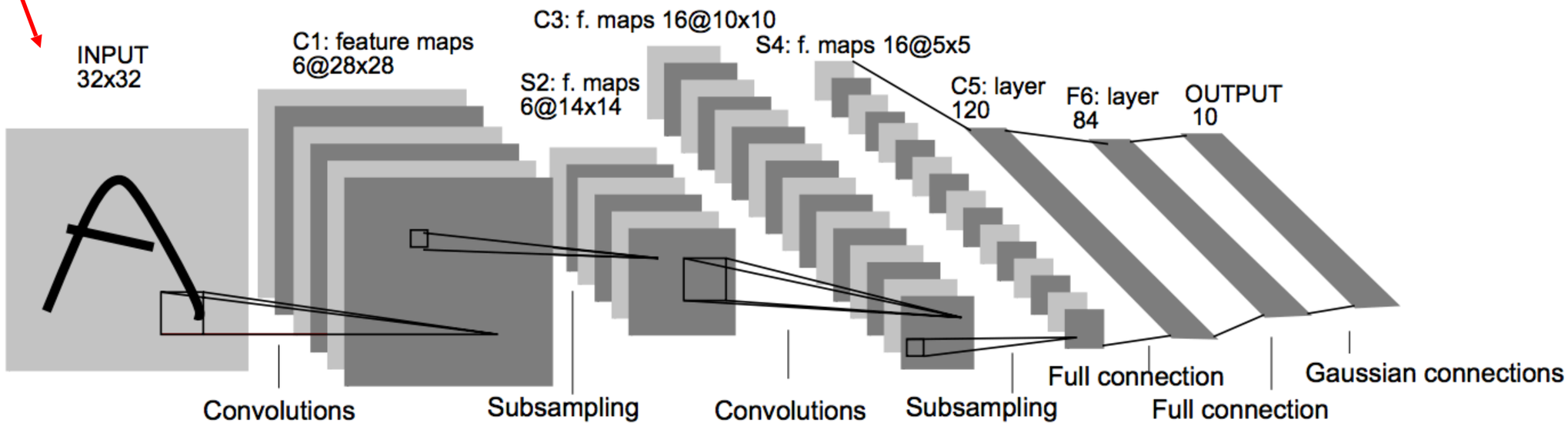
Idea: Use convolutions to efficiently learn features on image data

# Network diagram

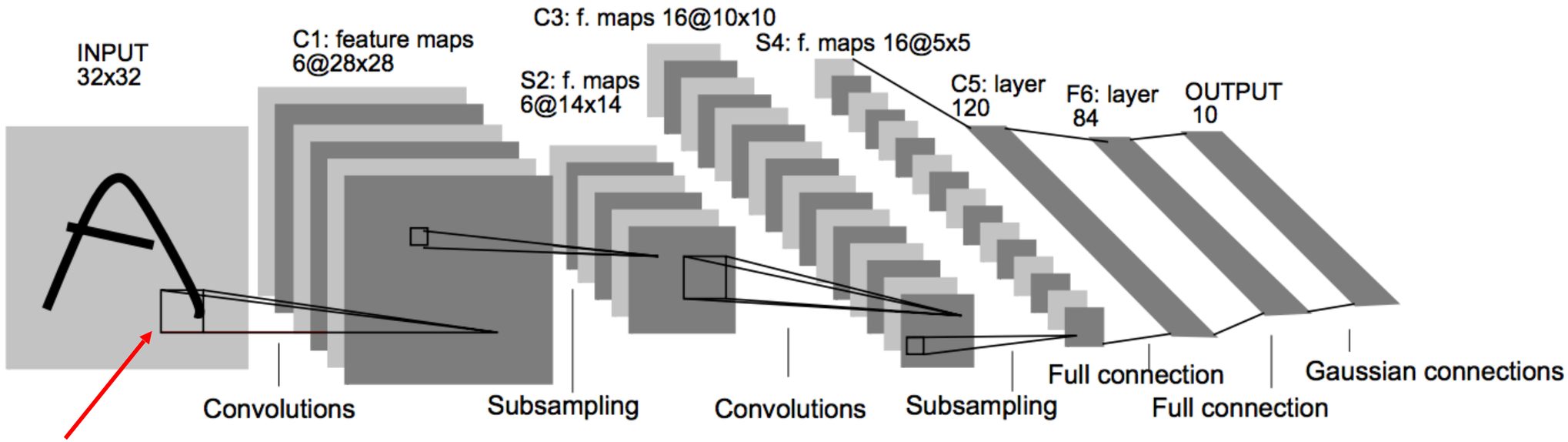


# Network diagram

Input: 28x28, with 2 pixels of padding (on all sides)



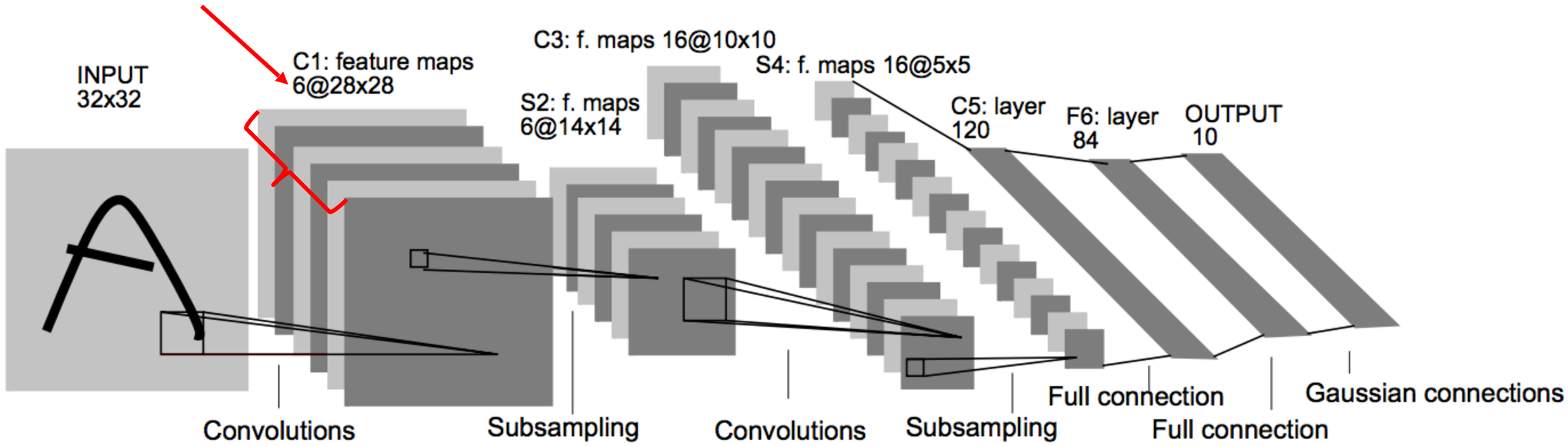
# Network diagram



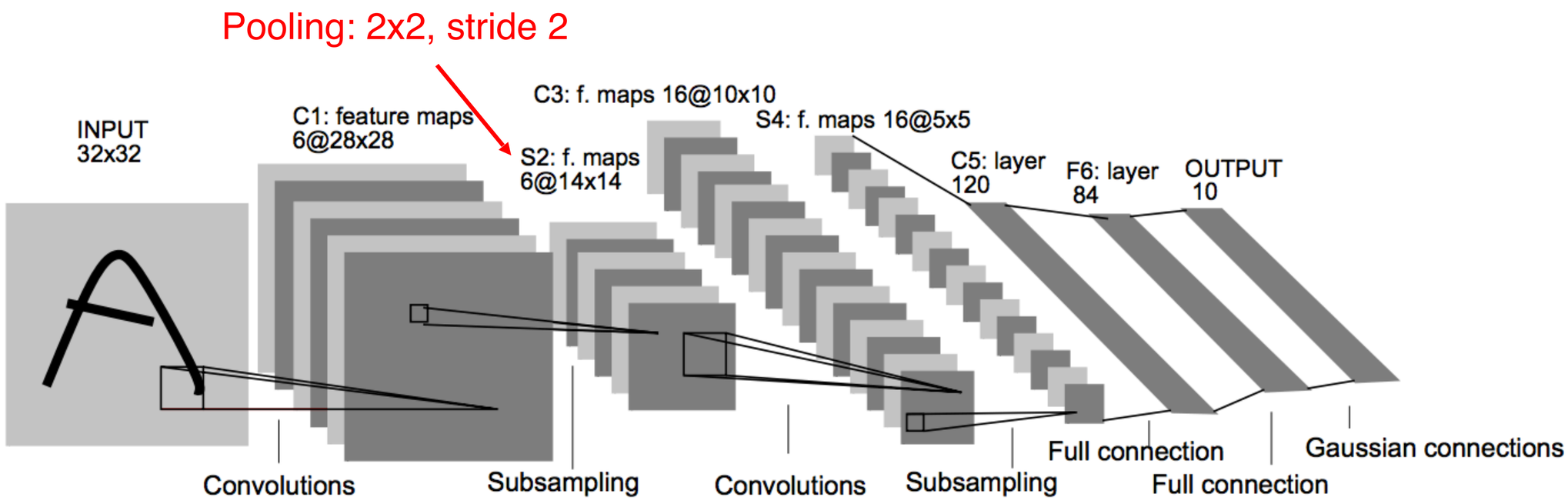

Convolution size: 5x5 (not labeled, but in paper)

# Network diagram

First convolutional layer depth: 6

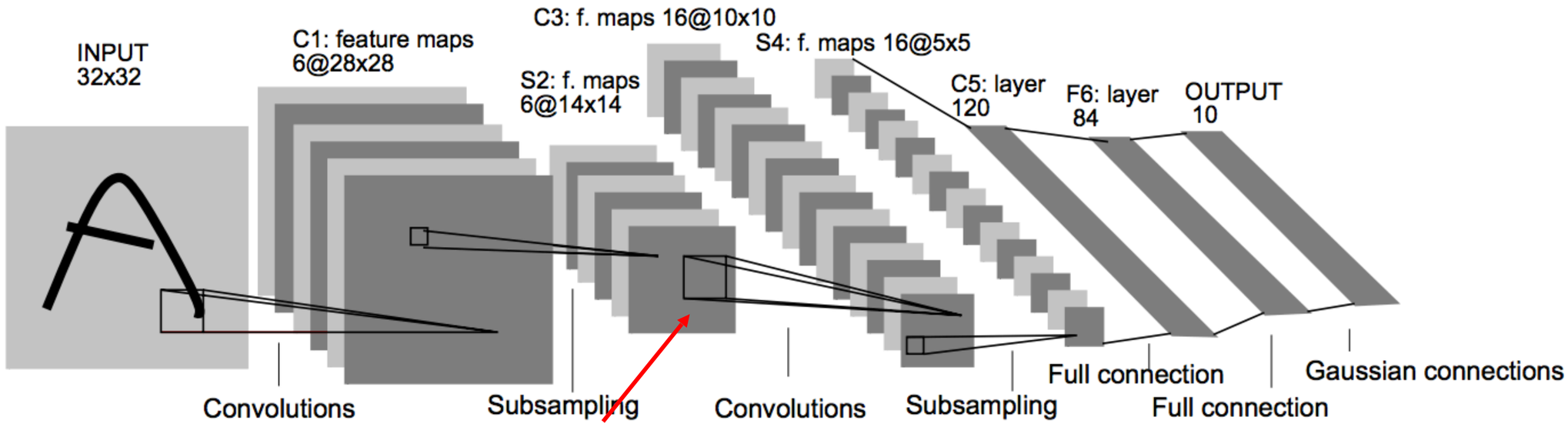


# Network diagram



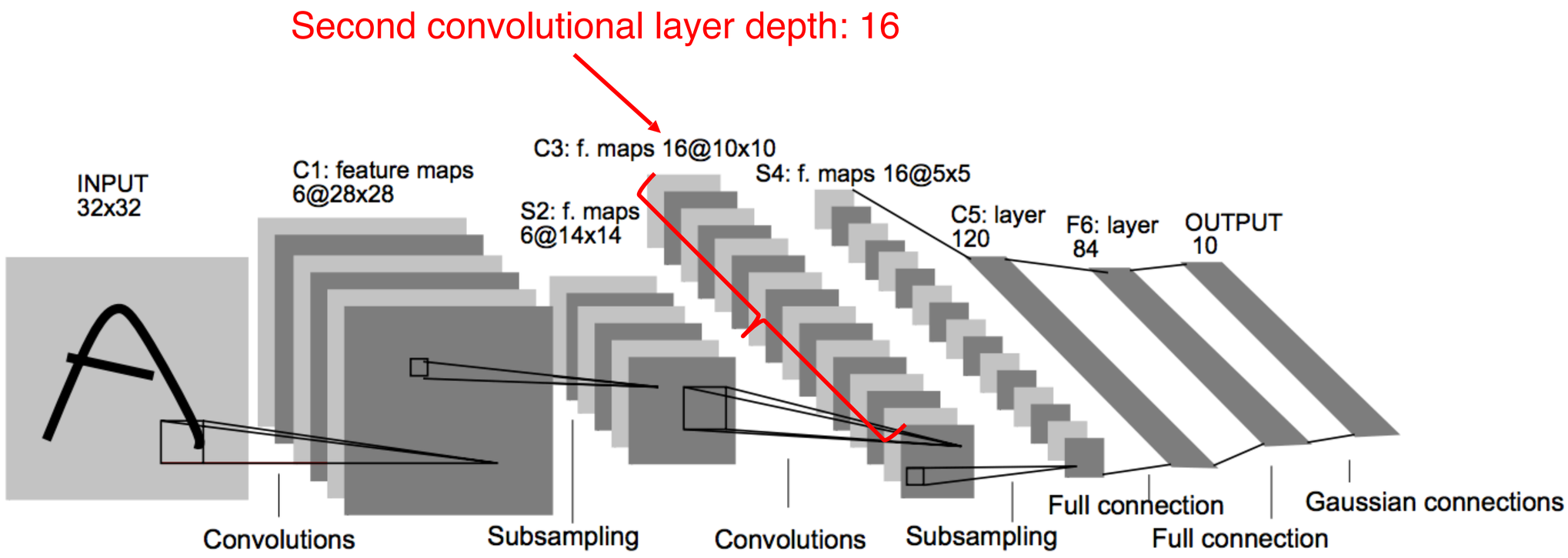

Note: in the paper, the model uses a more complex parameter based pooling operation.
Max/average pooling turns out to work better in practice

# Network diagram

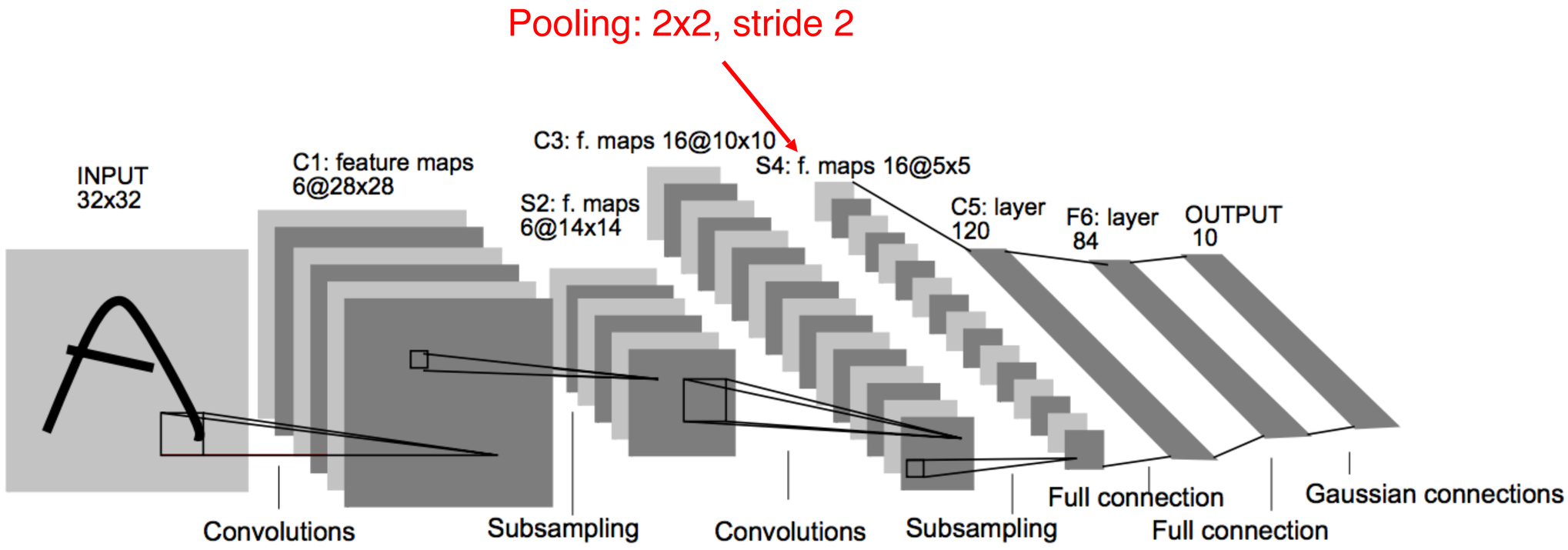



Convolution size: 5x5

# Network diagram

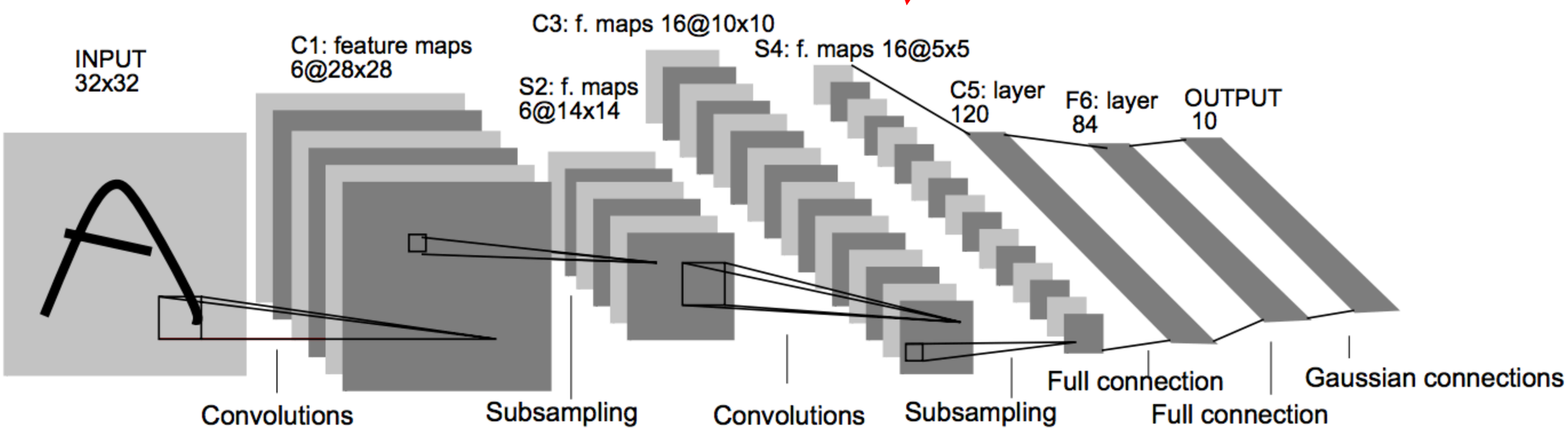


# Network diagram



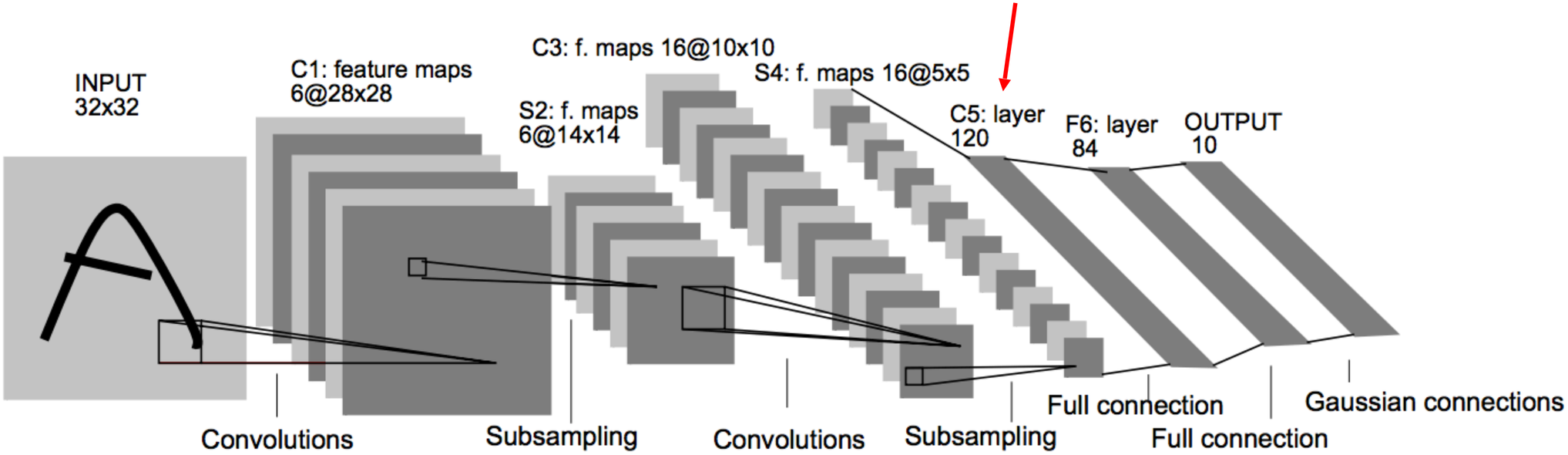
# Network diagram

Flatten from 5x5x16 to 400x1



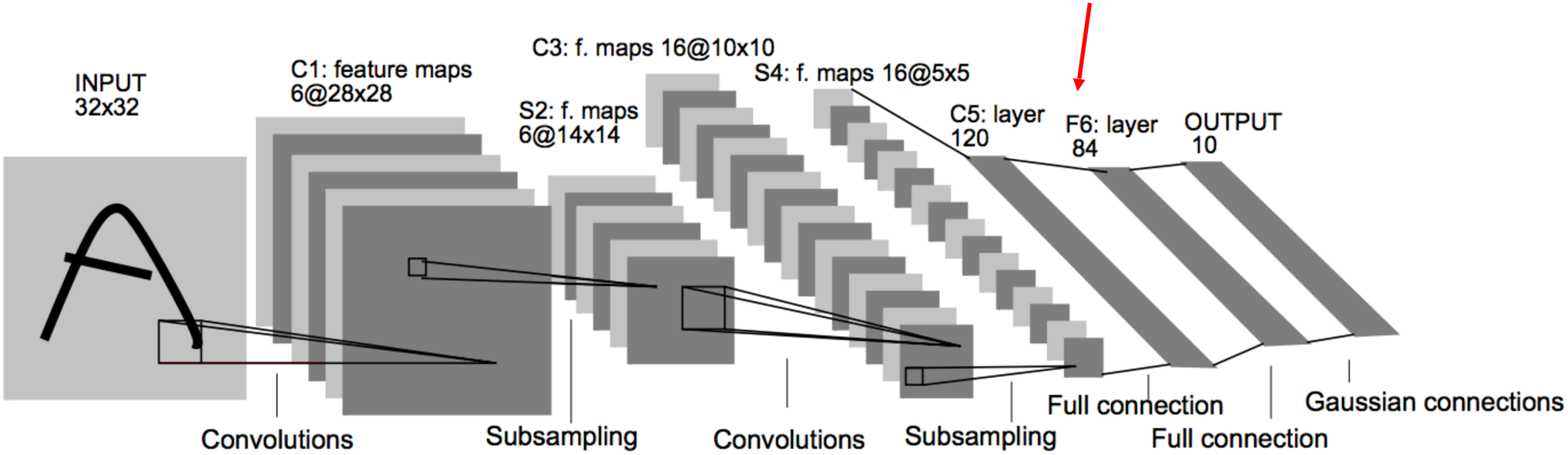
# Network diagram

Fully connected layer: from 400 to 120



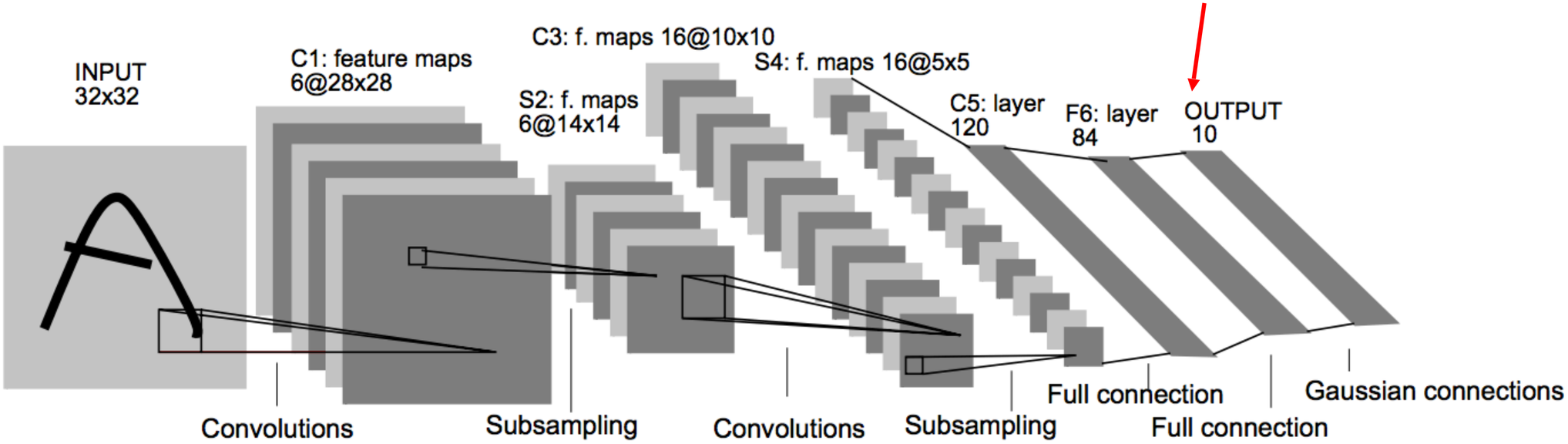
# Network diagram

Fully connected layer: from 120 to 84

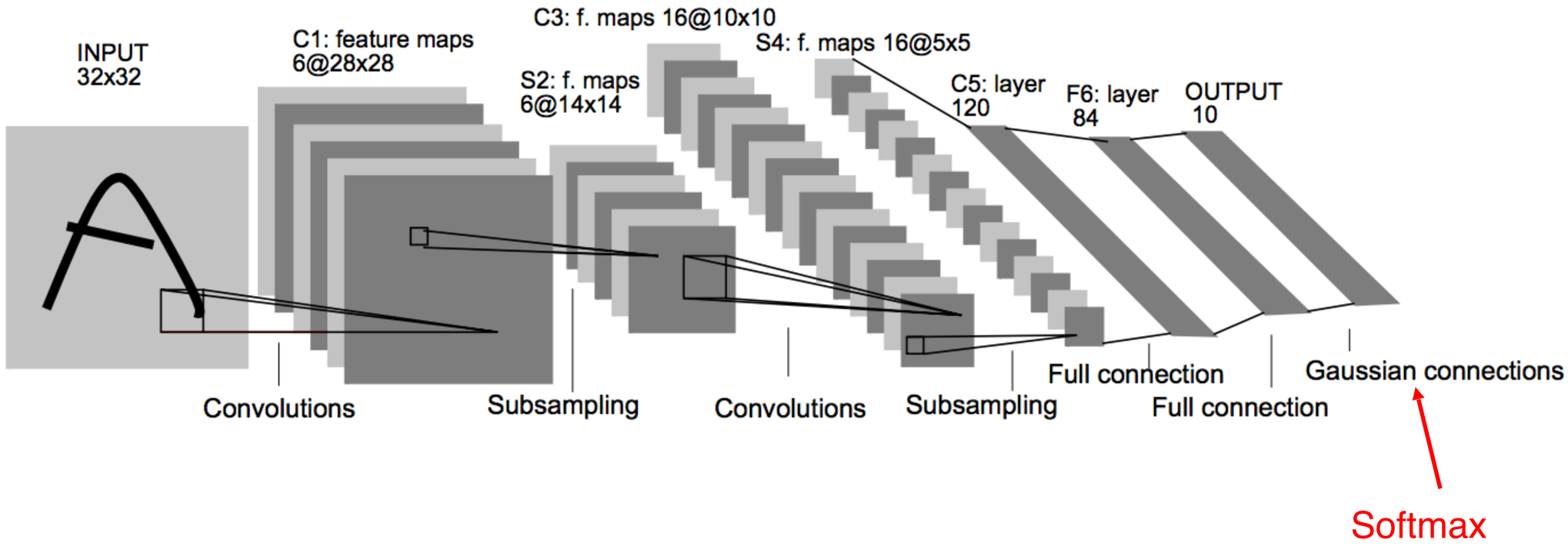


# Network diagram

Fully connected layer: from 84 to 10



# Network diagram



# Table description of LeNet-5

| Layer Name | Parameters |
|---|---|
| 1. Convolution | 5x5, stride 1, padding 2 (‘SAME’) |
| 2. Max Pool | 2x2, stride 2 |
| 3. Convolution | 5x5, stride 1, padding 2 (‘SAME’) |
| 4. Max Pool | 2x2, stride 2 |
| 5. Fully connected (ReLU) | Depth: 120 |
| 6. Fully connected (ReLU) | Depth: 84 |
| 7. Output (fully connected ReLU) | Depth: 10 |

# LeNet

| Layer Name | Parameters |
|---|---|
| 1. Convolution | 5x5, stride 1, padding 2 (‘SAME’), 6 filters |
| 2. Max Pool | 2x2, stride 2 |
| 3. Convolution | 5x5, stride 1, padding 2 (‘SAME’), 16 filters |
| 4. Max Pool | 2x2, stride 2 |
| 5. Fully connected (ReLU) | Depth: 120 |
| 6. Fully connected (ReLU) | Depth: 84 |
| 7. Output (fully connected ReLU) | Depth: 10 |

**MNIST digit recognition data**

70,000 images

Hand-written digits between 0-9

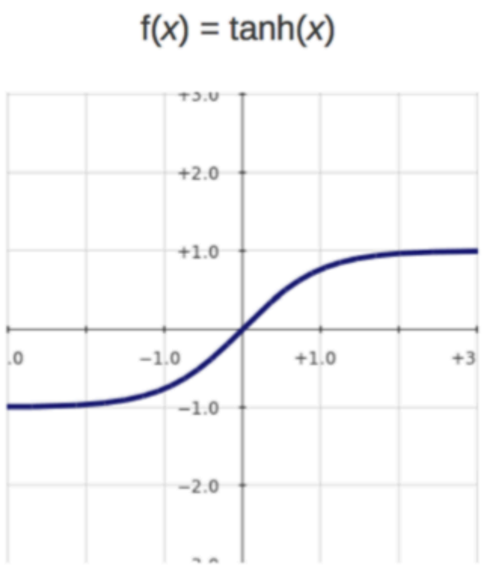
28x28 grayscale, range [0-255]

# RELU Nonlinearity

- Standard way to model a neuron

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

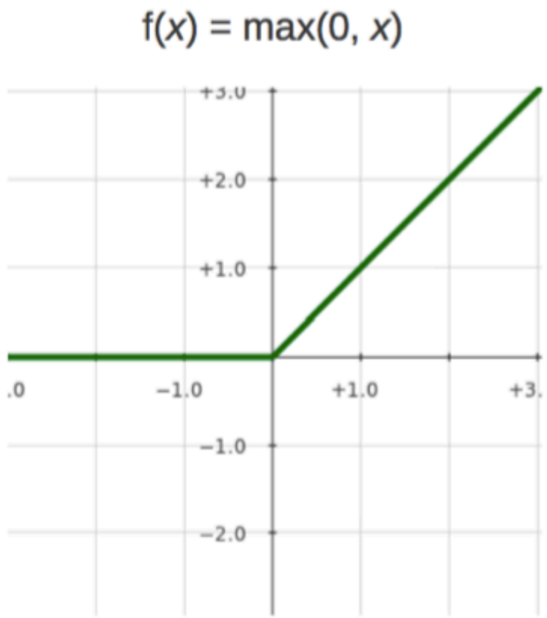
Very slow to train




- Non-saturating nonlinearity (RELU)

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

Quick to train



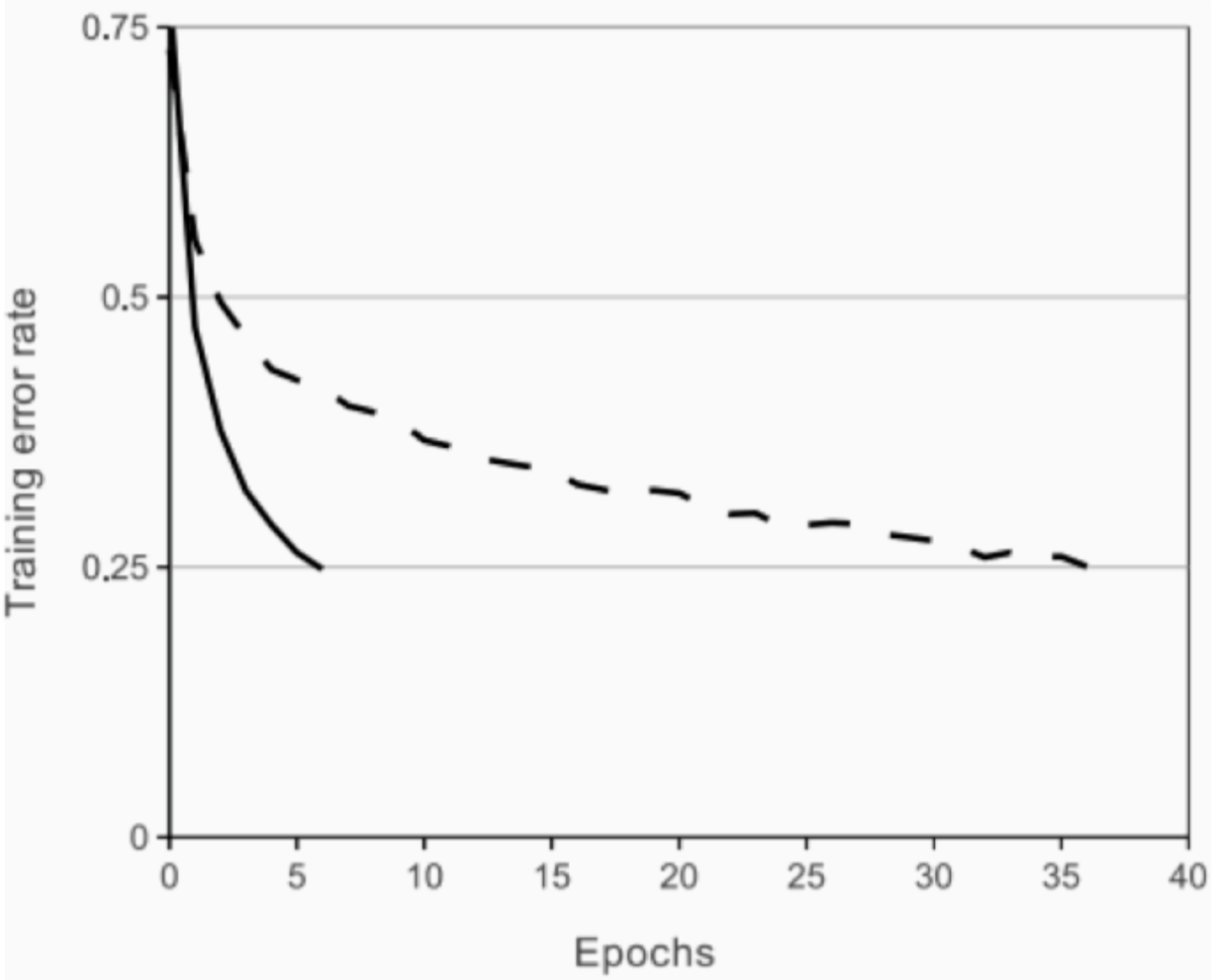

We didn't get to this in class, but remember RELU?

Better than tanh!

But <u>how much better?</u>

*..see next slide!*

# RELU Nonlinearity




A 4 layer CNN with ReLUs (solid line) converges six times faster than an equivalent network with tanh neurons (dashed line) on CIFAR-10 dataset

# Summary

# Lesson 14:
# Convolutional Neural Networks

- Motivation for CNNs
- Applications
- Review of convolution operations

**<u>Next week:</u>**

- Improving CNNs