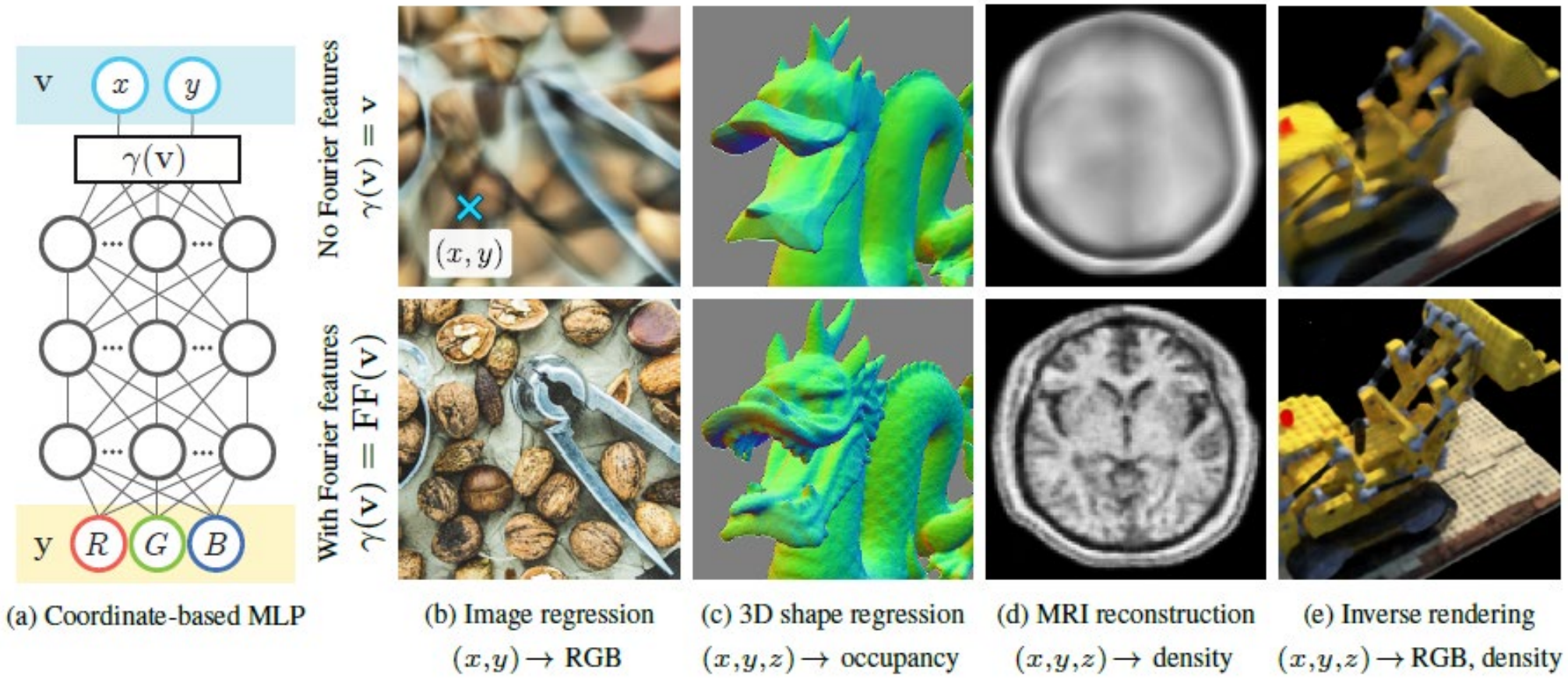


# CNNs and Key Ingredients of Deep Learning

Applied Machine Learning
Derek Hoiem

Dall-E

# MLPs are powerful, but input representation matters. Example: mapping position/ray to color



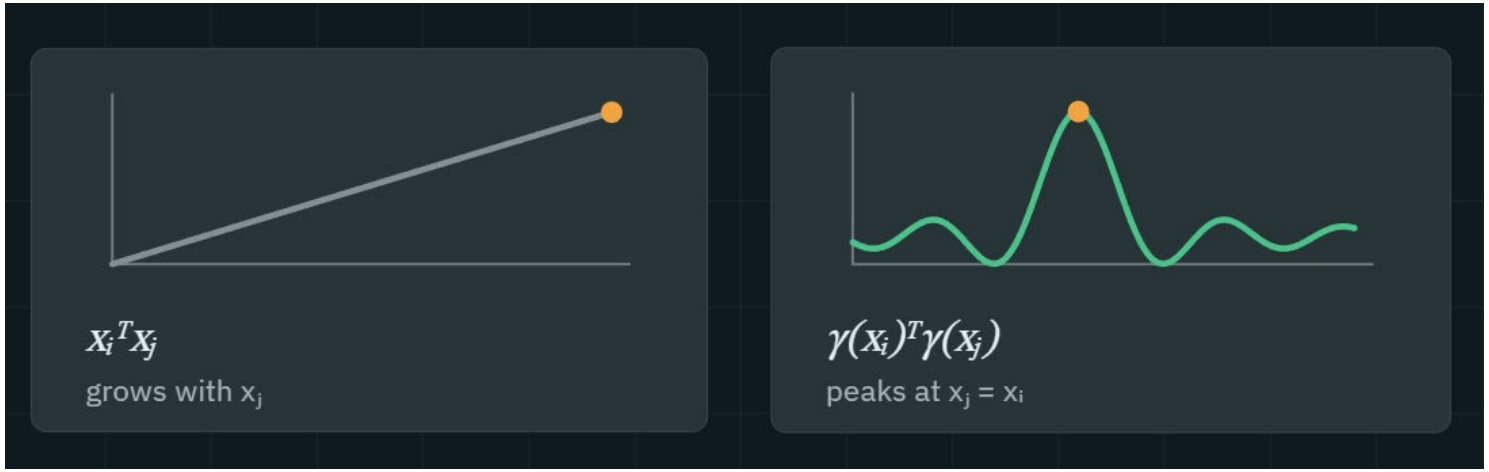

(a) Coordinate-based MLP | (b) Image regression $(x,y) \rightarrow$ RGB | (c) 3D shape regression $(x,y,z) \rightarrow$ occupancy | (d) MRI reconstruction $(x,y,z) \rightarrow$ density | (e) Inverse rendering $(x,y,z) \rightarrow$ RGB, density

- L2 loss
- ReLU MLP with 4 layers and 256 channels (nodes per layer)
- Sigmoid activation on output
- 256 frequency positional encoding

Fourier Features (Tancik et al. 2020) NeRF (Mildenhall et al. 2020)

# Generalized insight from “Fourier Features”

- Input matters – it’s best to represent data in a way that makes it linearly predictive, even if you have a non-linear model
- $f(x, y) \rightarrow R, G, B$ requires a complex network to model because $x_i^T x_j$ is a bad similarity function (maximized when $x_j$ is large, instead of similar to $x_i$)
- Representing $x$ with a Fourier encoding, e.g. $\gamma(x) = [\sin(x), \cos(x), \sin(2x), \cos(2x), \ldots]$ enables a simpler network because $\gamma(x_i)^T \gamma(x_j)$ falls off smoothly as $x_j$ moves away from $x_i$
  - This means the initial network layer can model similarity to different positions with each hidden unit

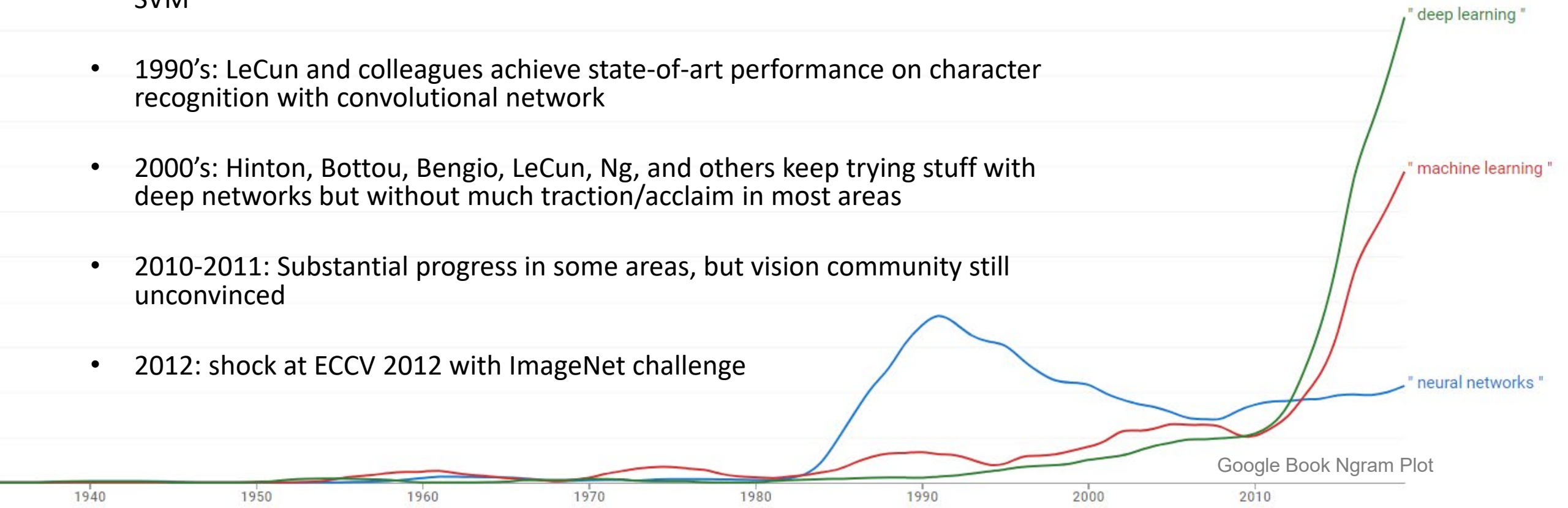


# Today's Lecture

- A bit of deep learning history
- Early advances in deep networks
  - Convolutional networks
  - Residual connections
  - BatchNorm
- Enhancements to SGD

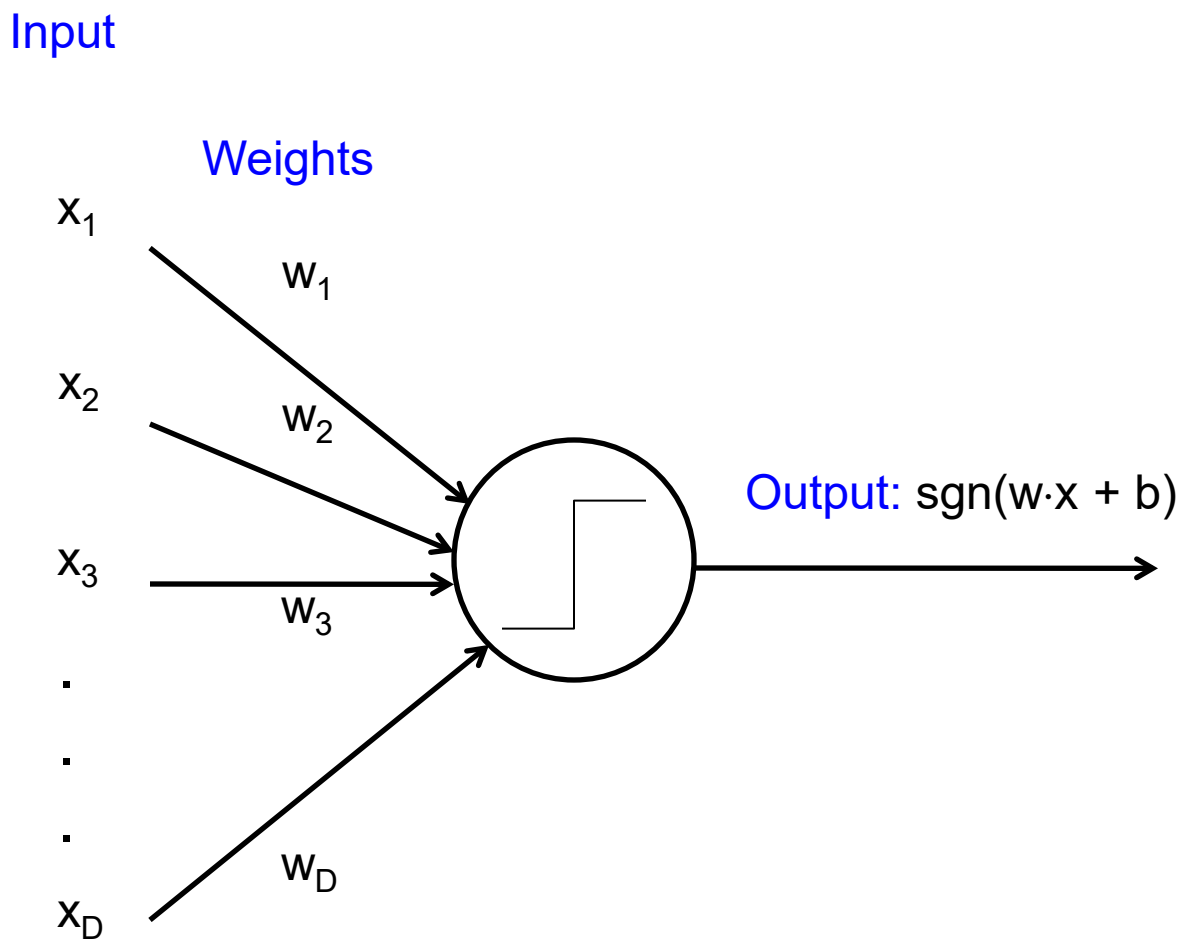
# Brief history of deep learning

- 1958: neural nets (perceptron and MLP) invented by Rosenblatt
- 1967: First use of SGD in deep-learning network (Amari)
- 1980's/1990's: Neural nets are popularized and then abandoned as being interesting idea but too difficult to optimize or "unprincipled", supplanted by SVM
- 1990's: LeCun and colleagues achieve state-of-art performance on character recognition with convolutional network
- 2000's: Hinton, Bottou, Bengio, LeCun, Ng, and others keep trying stuff with deep networks but without much traction/acclaim in most areas
- 2010-2011: Substantial progress in some areas, but vision community still unconvinced
- 2012: shock at ECCV 2012 with ImageNet challenge




Google Book Ngram Plot

# The Perceptron, the first neural network




Rosenblatt, Frank (1958), The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386–408.

# 1958 New York Times…

# NEW NAVY DEVICE LEARNS BY DOING

## Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

### Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

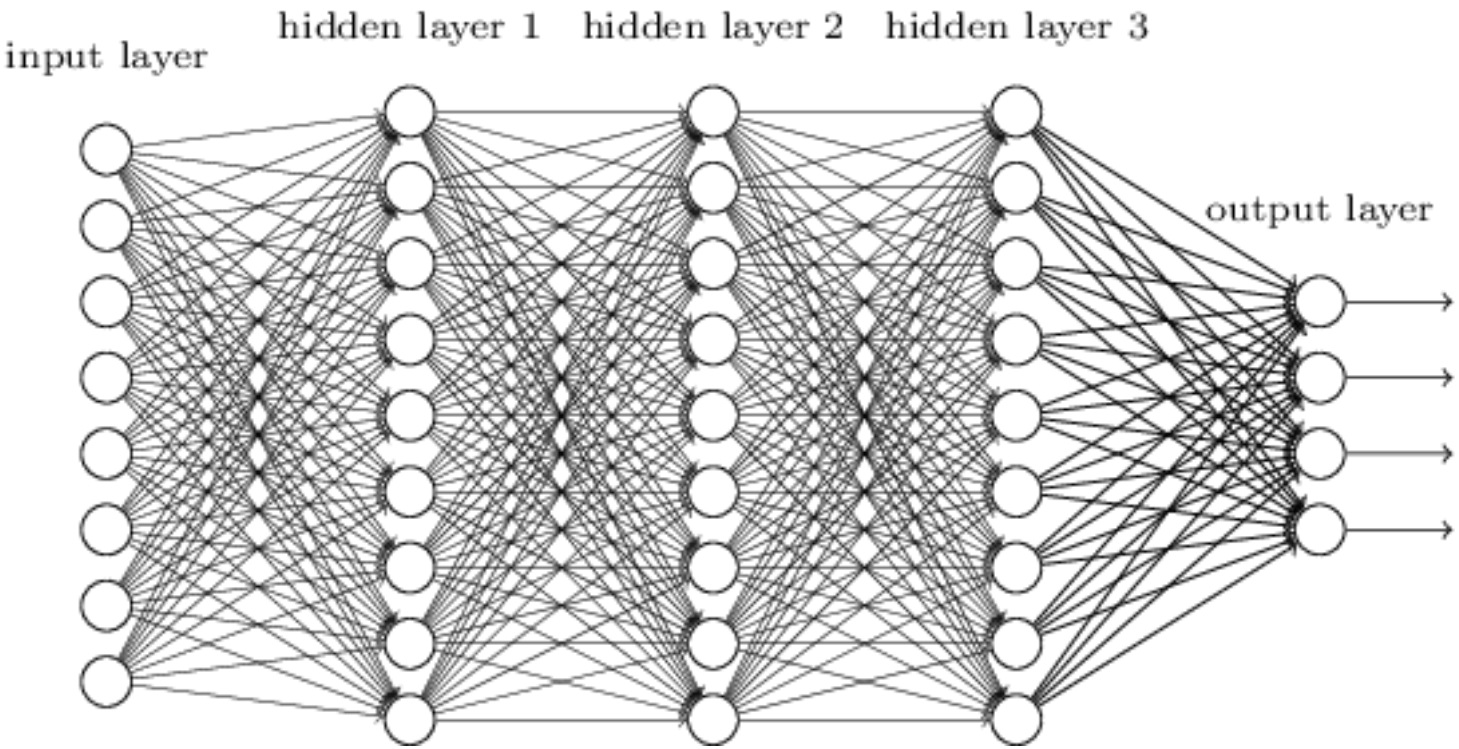
### Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

# **A key problem**: how to model the compositional nature of text and images?

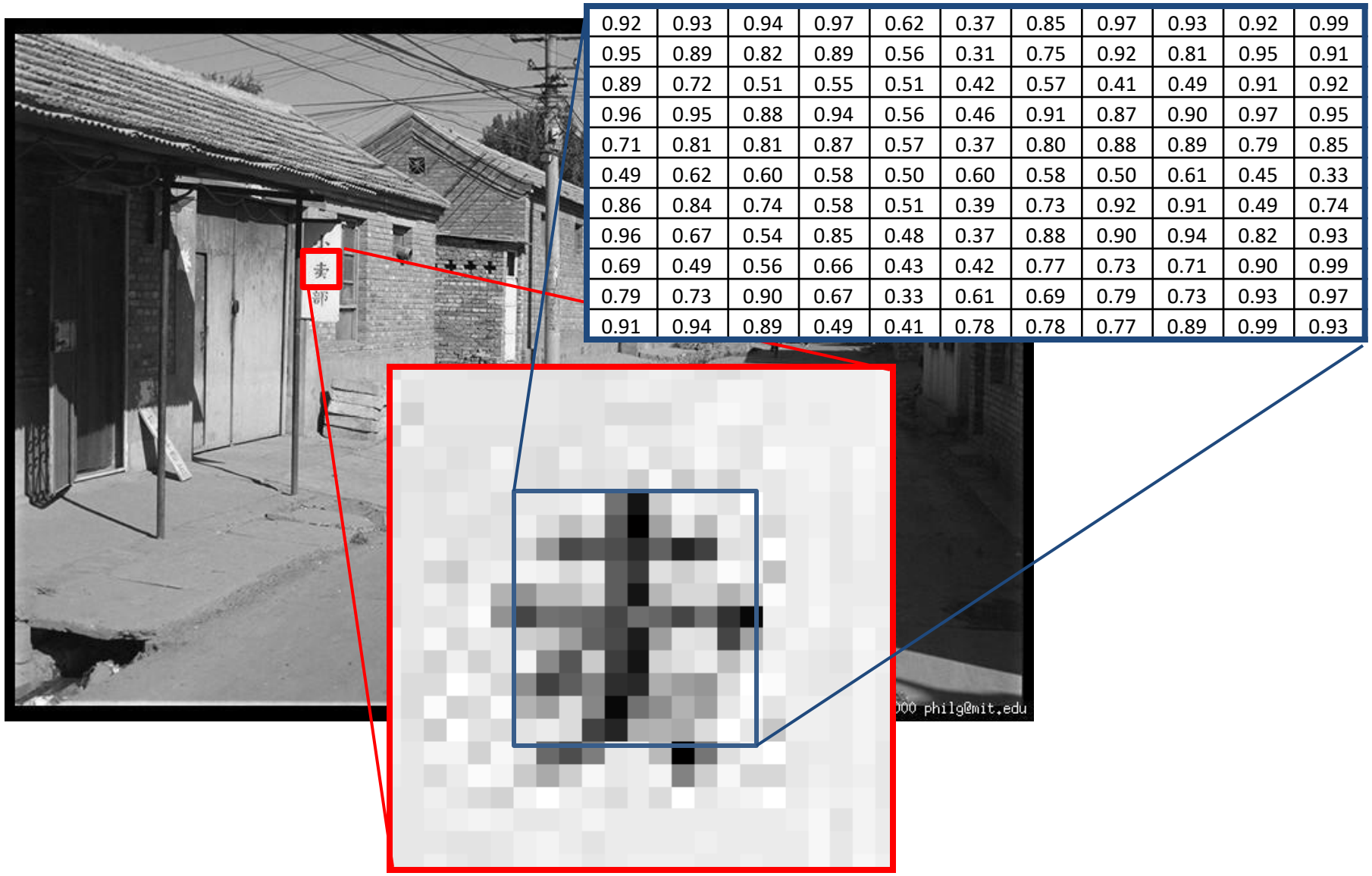



You could treat the image like a vector of values and add fully connected layers (which we do in HW4)

But this doesn't take advantage of the 2D structure of images

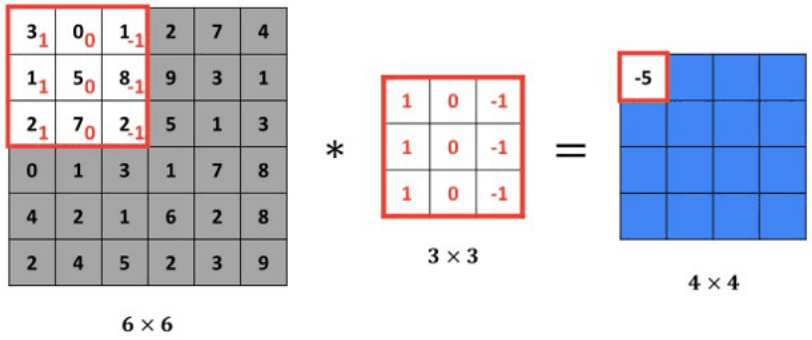
Slide: Lazebnik

# Images have local patterns that can appear at different positions



# Linear filtering is a foundation of image processing

- Linear image filtering: at each pixel, output a weighted sum of pixels in surrounding patch
  - E.g. Gaussian-weighted smoothing filter (right), edge detection (below), local pattern detection

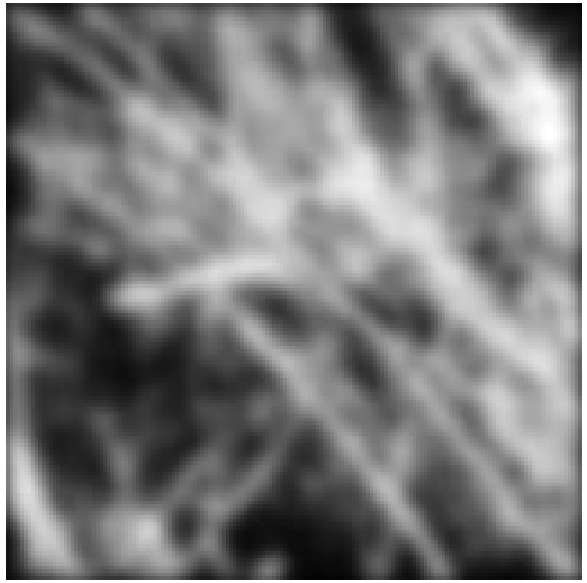



$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$



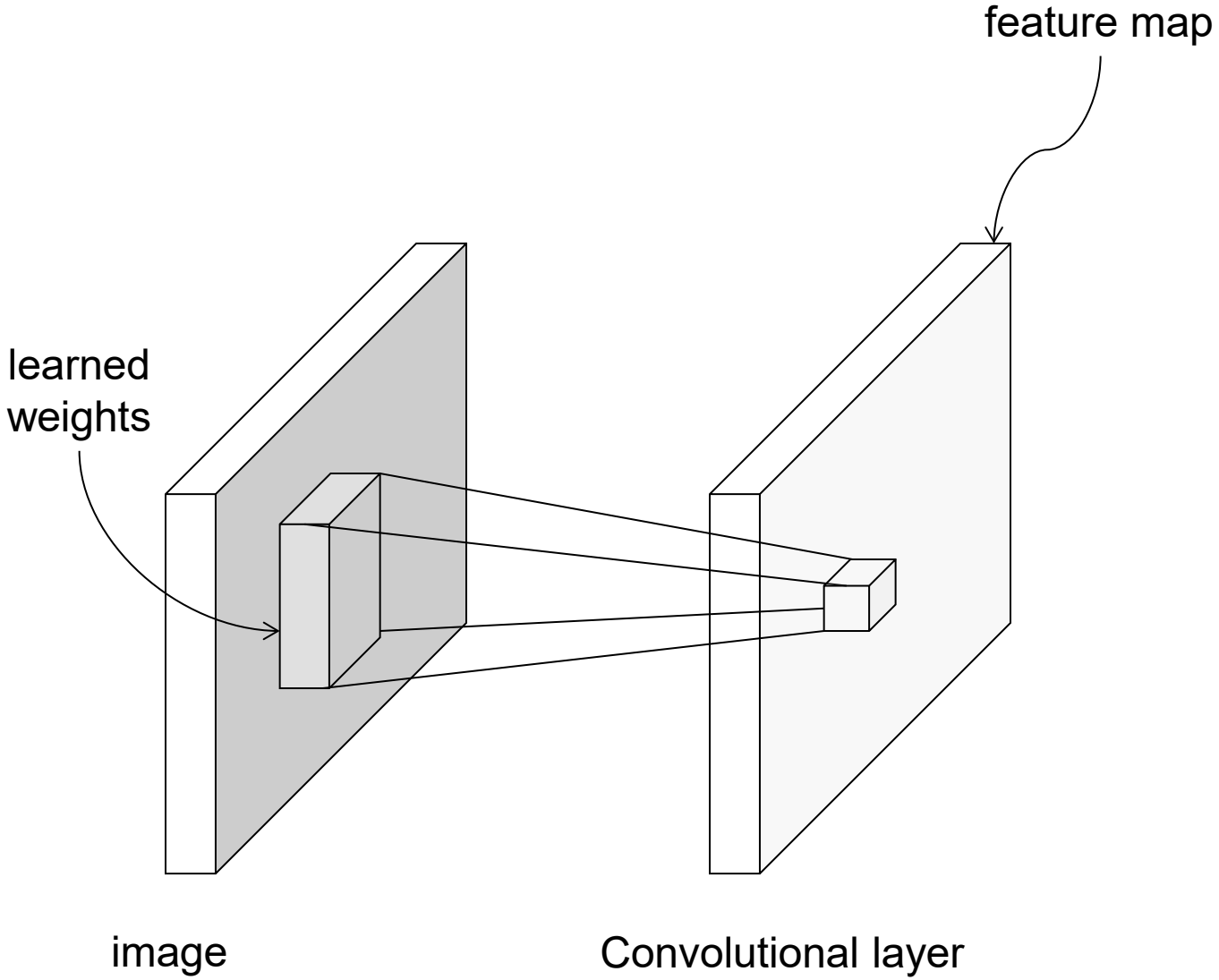
Smoothing Filter





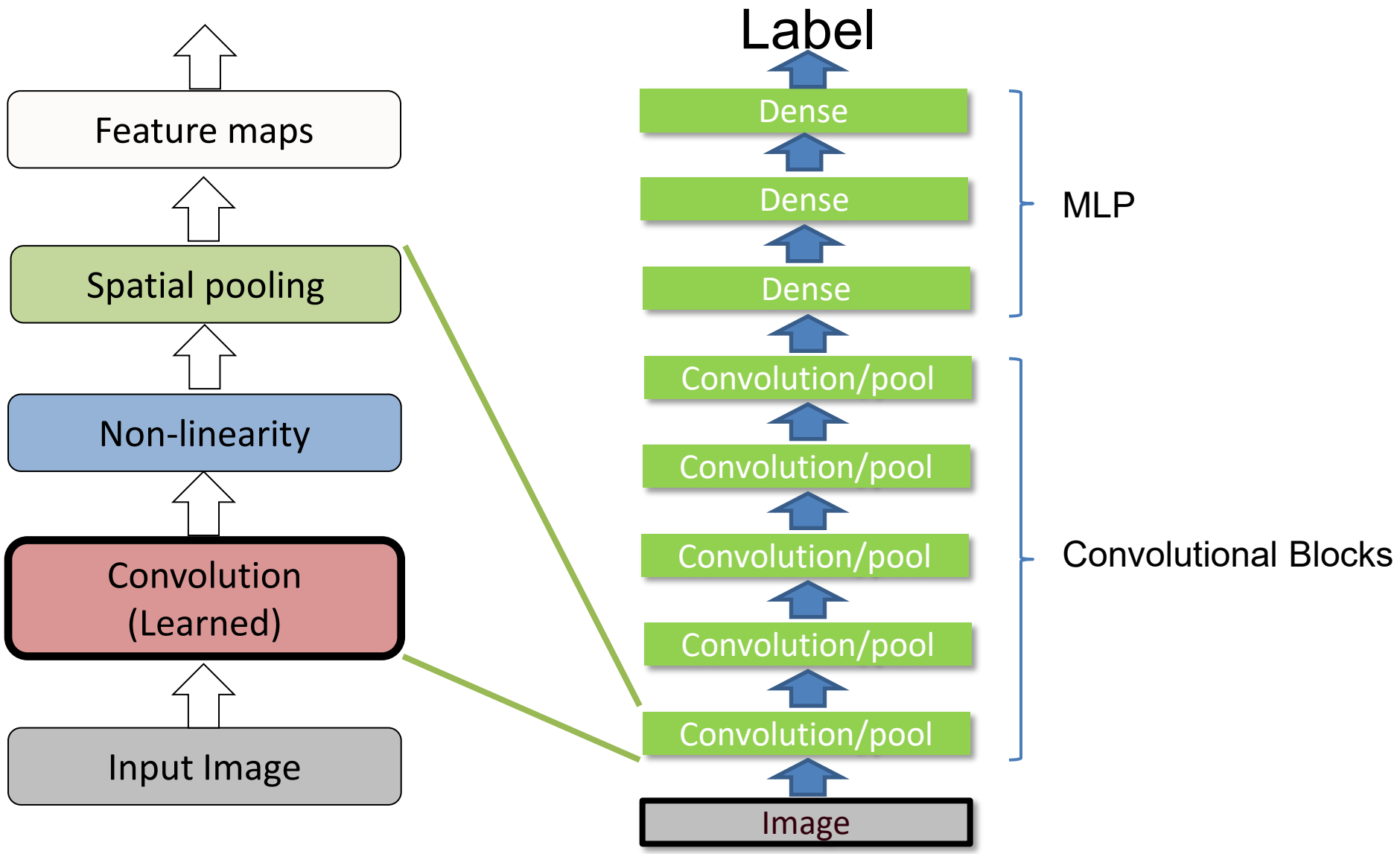
Animation: https://datahacker.rs/004-how-to-smooth-and-sharpen-an-image-in-opencv/

# A CNN (convolutional network) learns filter weights to create grids of features (“feature map”)

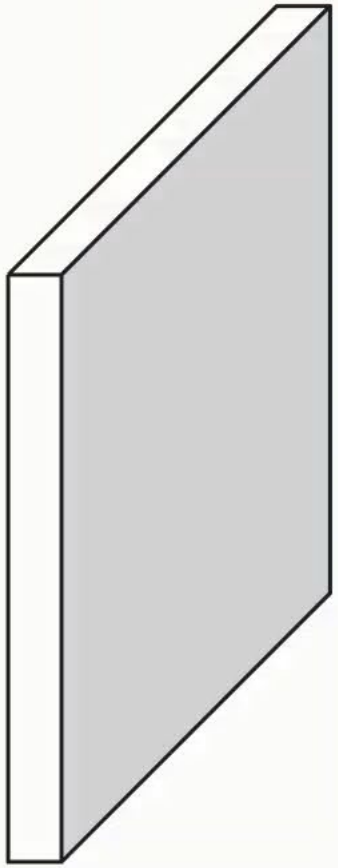



Slide: Lazebnik

# Repeatedly filter and pool to detect patterns of patterns of patterns of …

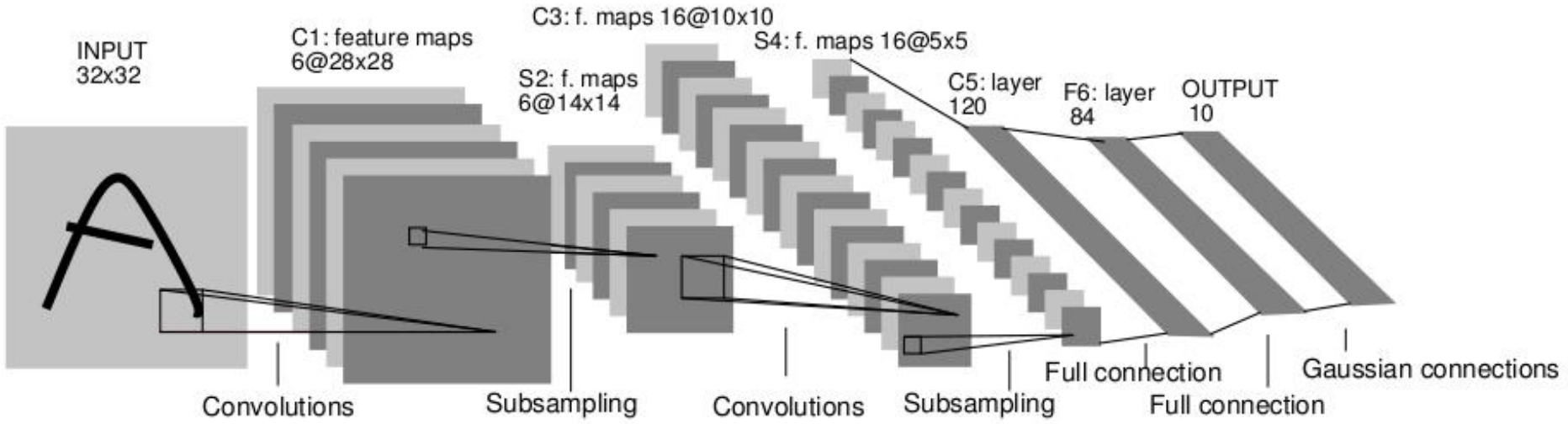


# A CNN learns filter weights to create a feature map

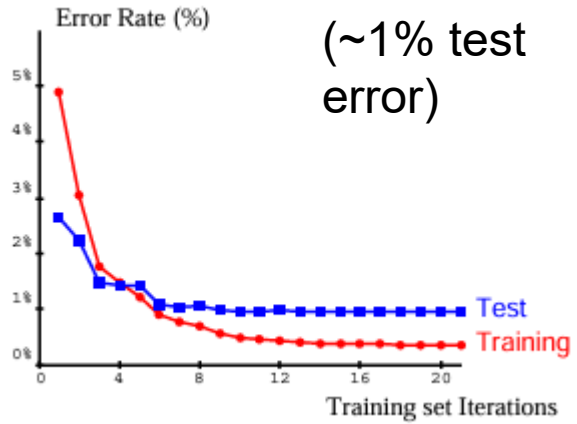


image

# LeNet-5 for character/digit recognition




- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples




MNIST results (~1% test error)

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

# CNN: Convolutional Neural Network

- Architecture explicitly designed to model compositional patterns with translational invariance
- Combines convolutional layers with MLPs
- Can have 1D CNNs for text or signals, or 3D CNNs for volumetric data

Q1

https://tinyurl.com/441AML-L16



For the next 10+ years, neural networks did not gain traction, and they were dismissed as an interesting idea that just didn’t work

- 2009 – Raina et al. train CNN with GPU
- 2011 – Glorot et al. found that using ReLUs improved training
- 2012…

# Fast forward to the arrival of big visual data…

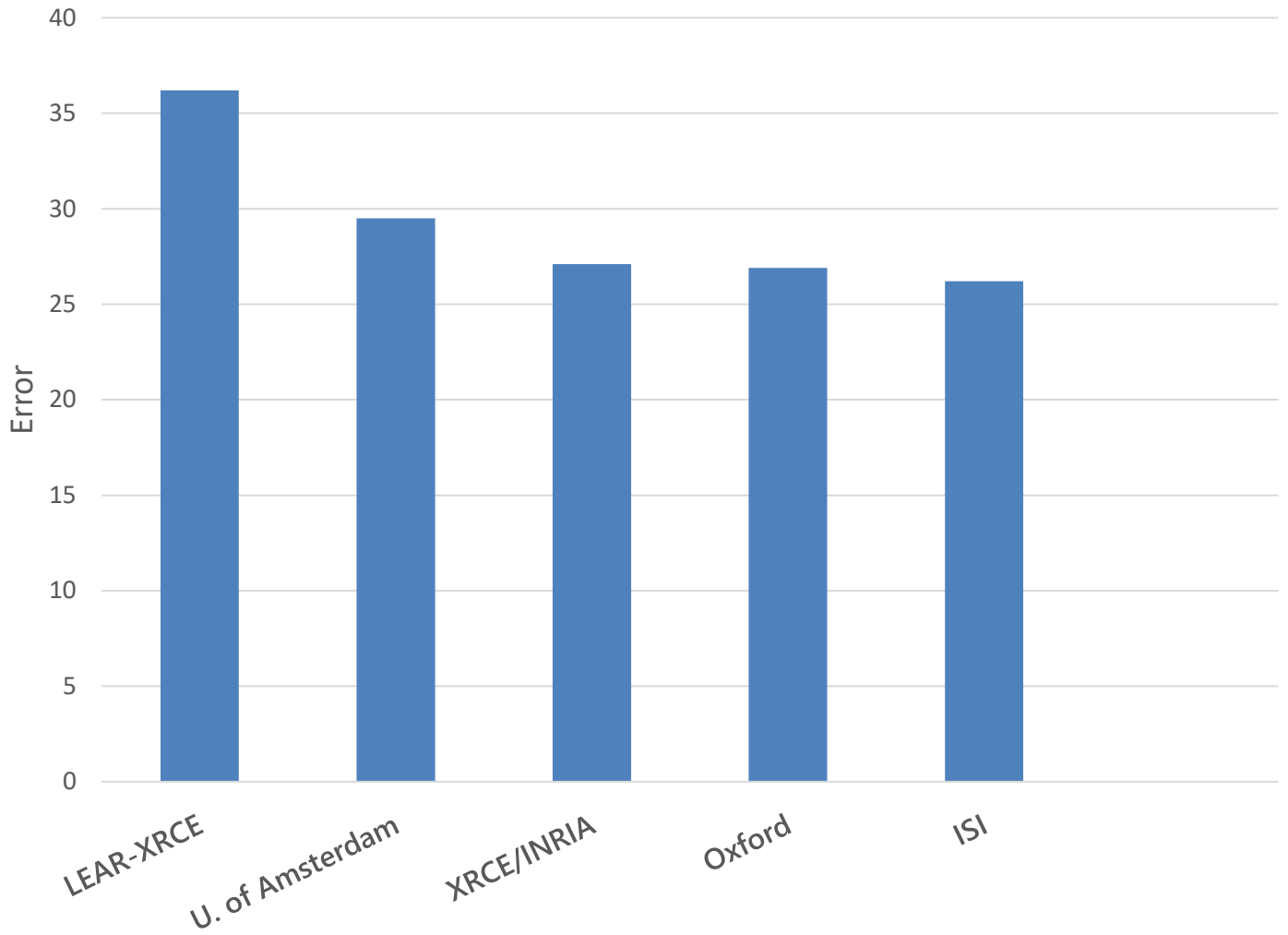



- Created in 2006
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes

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

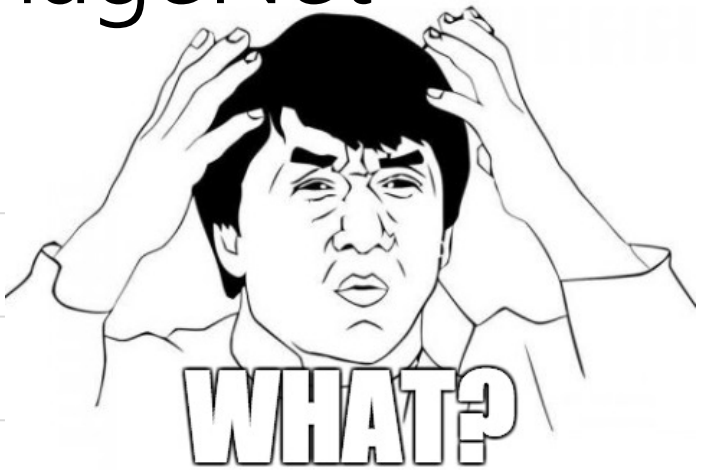
Slide: Lazebnik

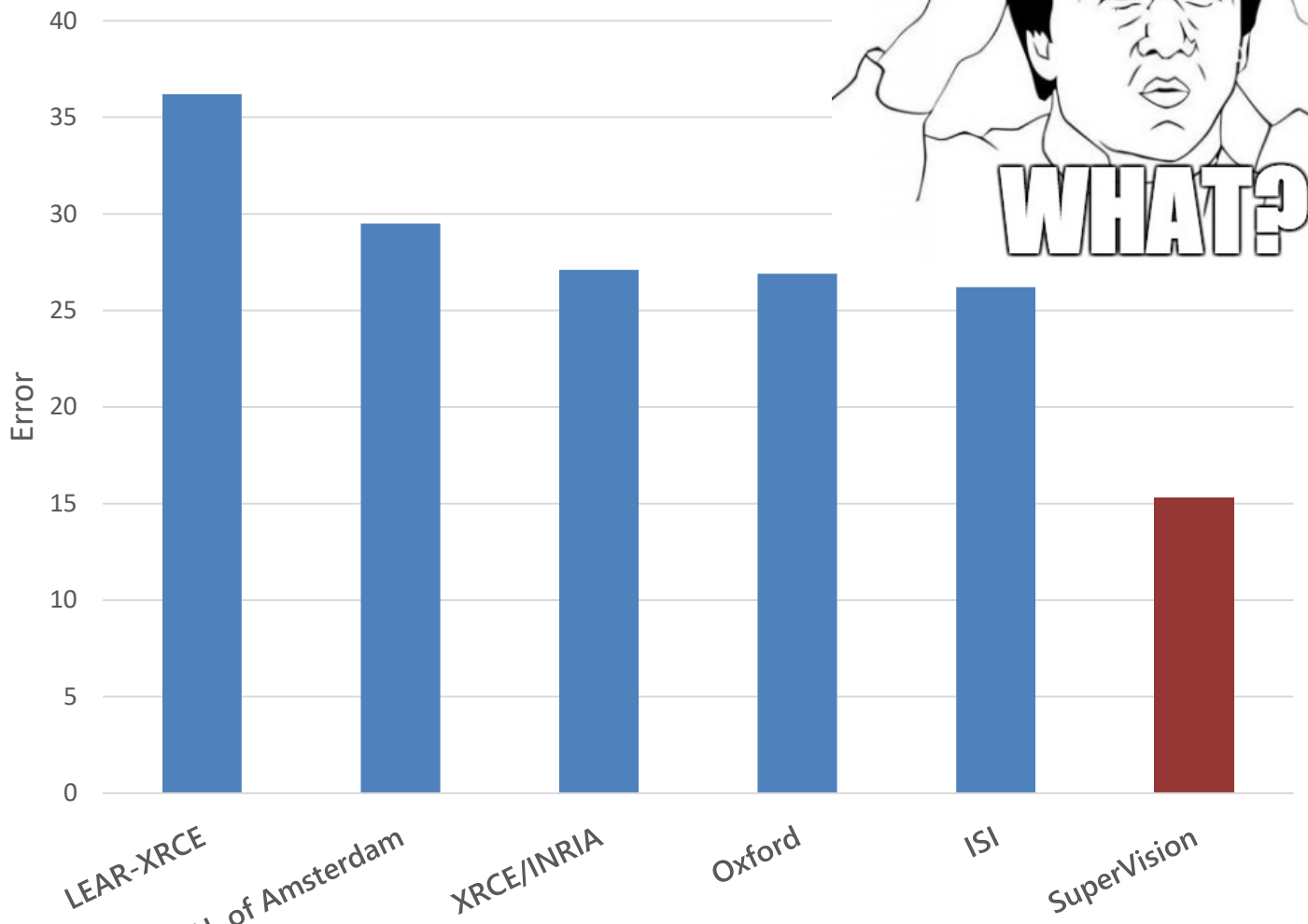
# Surprise in the 2012 ImageNet competition at ECCV




Slide: Jia-bin Huang

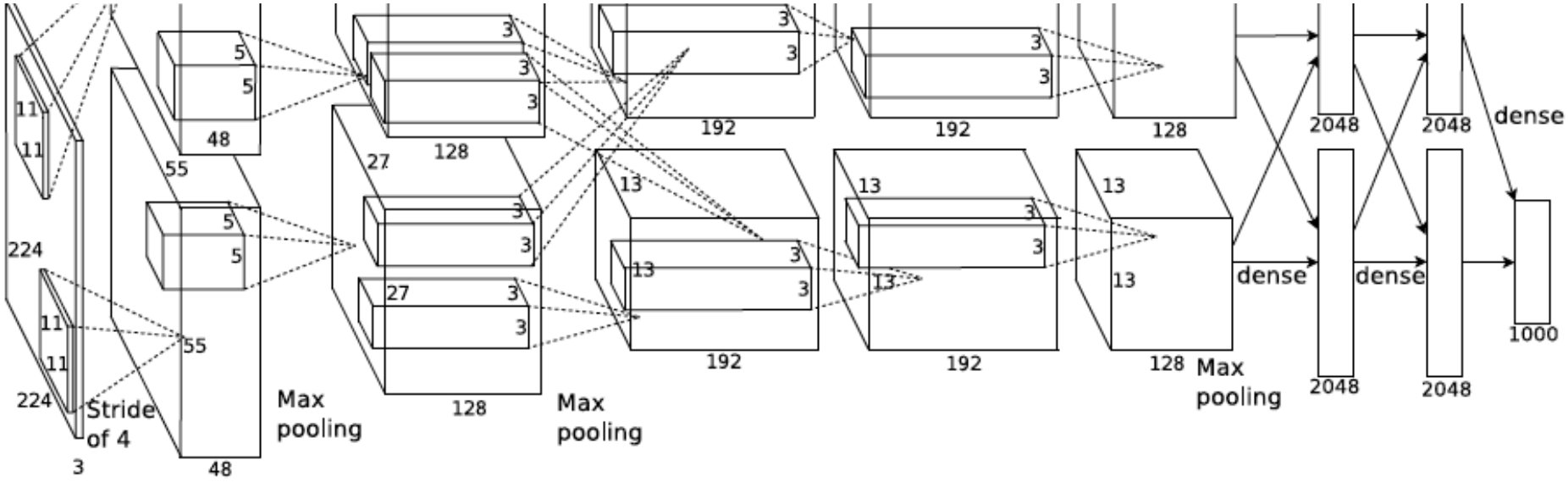
# Surprise in the 2012 ImageNet competition at ECCV




Slide: Jia-bin Huang

# AlexNet: ILSVRC 2012 winner




- Similar framework to LeNet but:
  - Max pooling, **ReLU nonlinearity**
  - **More data** and **bigger model** (7 hidden layers, 650K units, 60M params)
  - GPU implementation (**50x speedup** over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

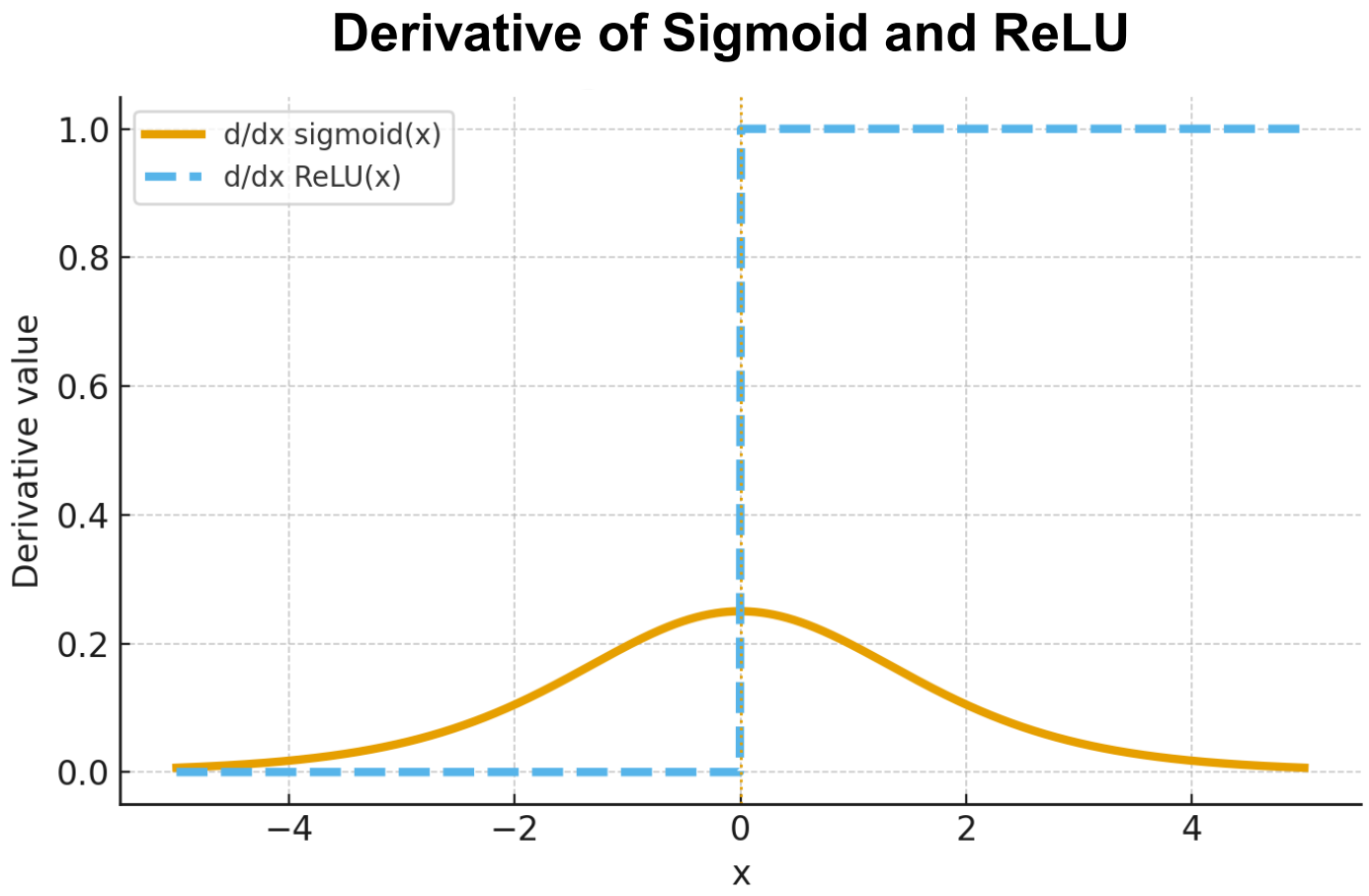
# What enabled the breakthrough?

1. ReLU activation enabled large models to be optimized
2. ImageNet provided diverse and massive annotation to take advantage of the models
3. GPU processing made the optimization practicable



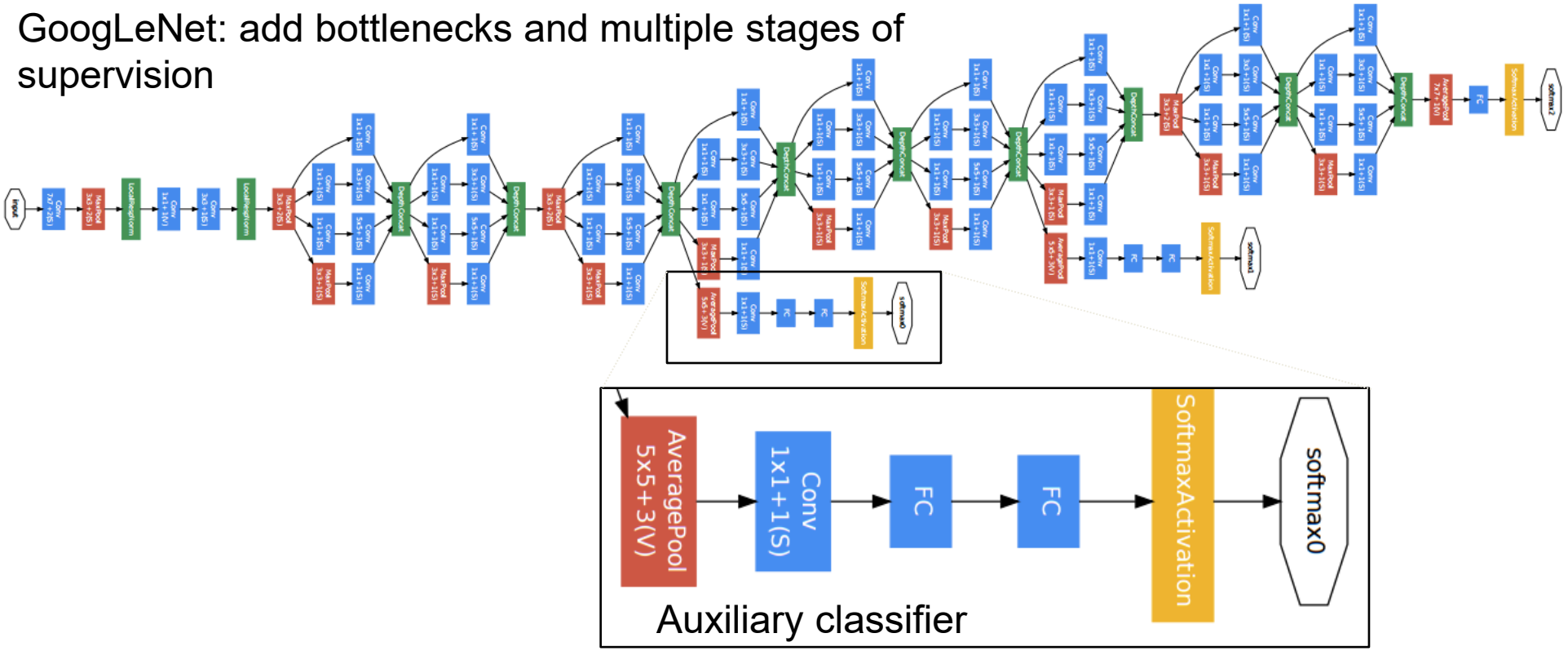
# Sigmoid vs ReLU

- Sigmoid has derivative less than 1 everywhere
- Long products of gradients will have vanishingly small values
- ReLU have gradient of 1 for positive values, so a long product can potentially have a gradient of 1



# Even with ReLU, it was hard to get very deep networks to work well

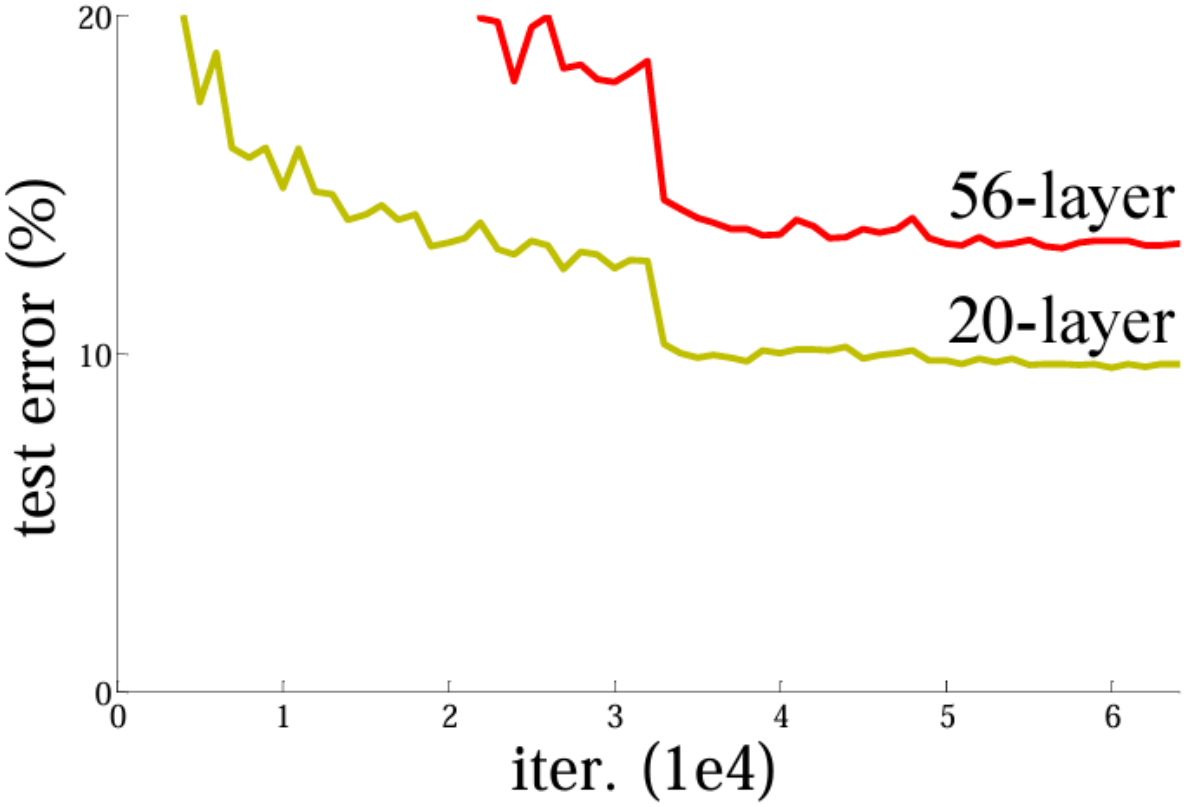
GoogLeNet: add bottlenecks and multiple stages of supervision



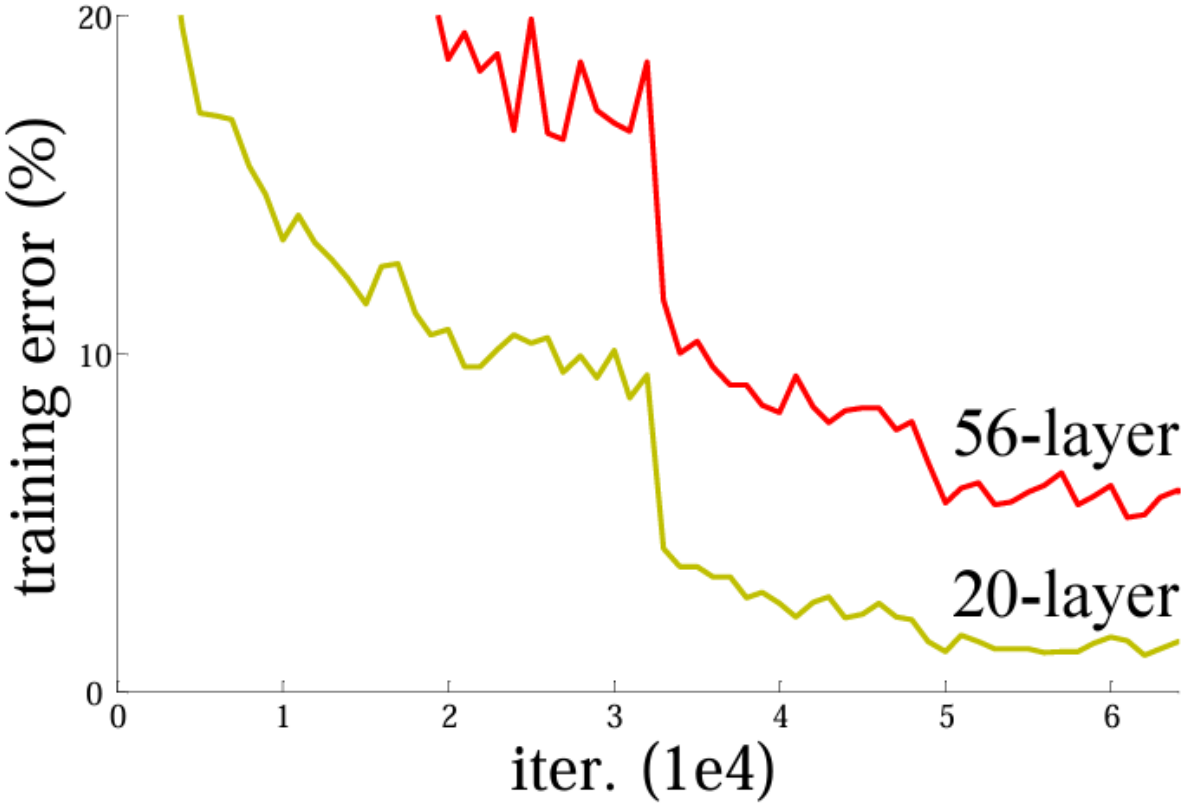

C. Szegedy et al., Going deeper with convolutions, CVPR 2015

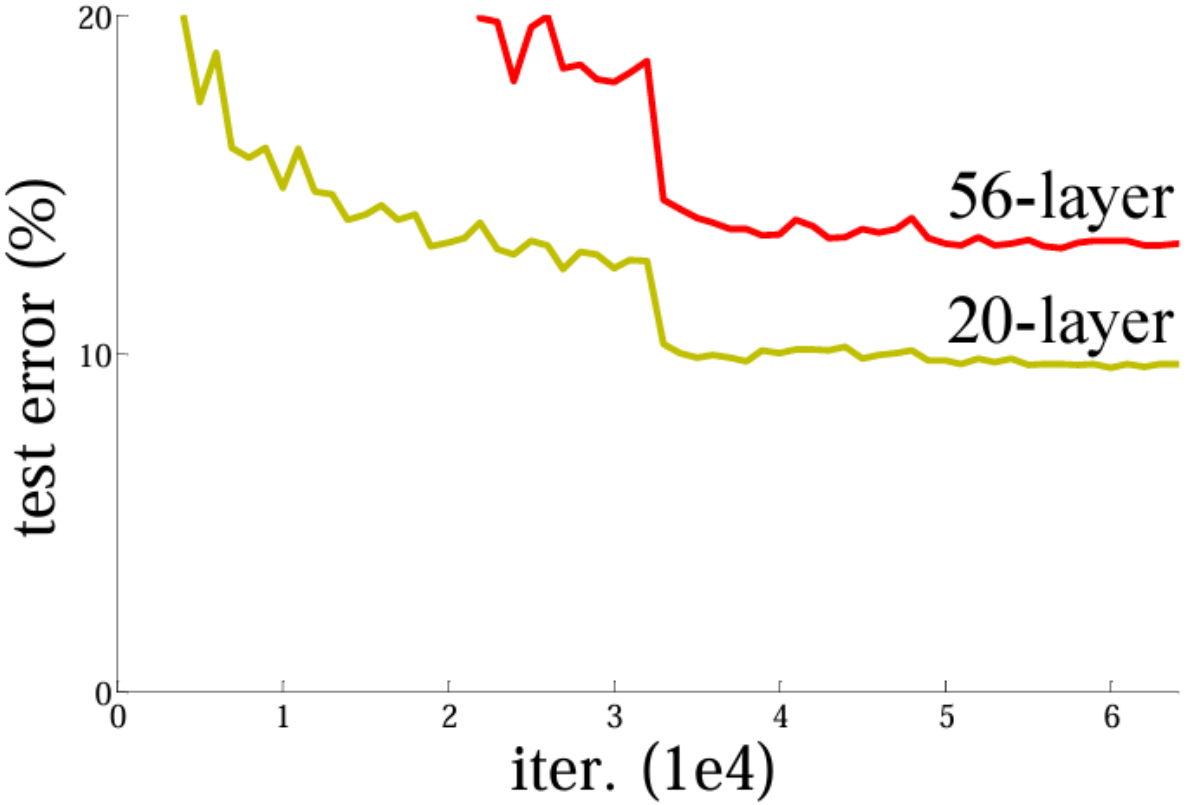
# What was the problem?

- Were deeper networks overfitting the training data?
- Or was the problem just that we couldn't optimize them?
- How could we answer this question?



# Look at the training error!




With deeper networks, the training error goes up!?!

Fig: He et al. 2016

# Very deep networks, vanishing gradients, and information propagation

Vanishing gradients

- Early weights have a long path to reach output
- Any zeros along that path kill the gradient
- Early layers cannot be optimized
- Multiple stages of supervision can help, but it's complicated and time-consuming

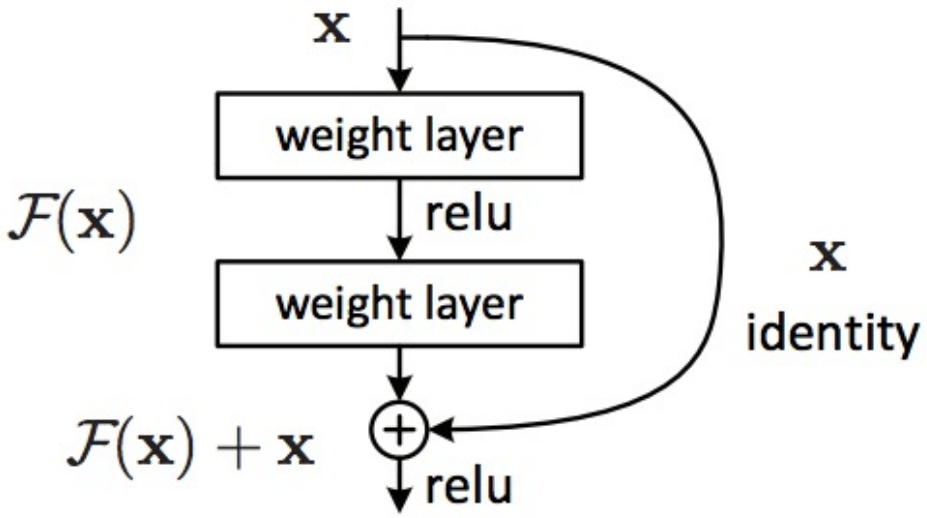
Information propagation

- Networks need to continually maintain and add to information represented in previous layers



# ResNet: the residual module

- Use *skip* or *shortcut* connections around 2-3 layer MLPs or CNNs
- Gradients can flow quickly back through skip connections
- Each module needs only add information to the previous layers

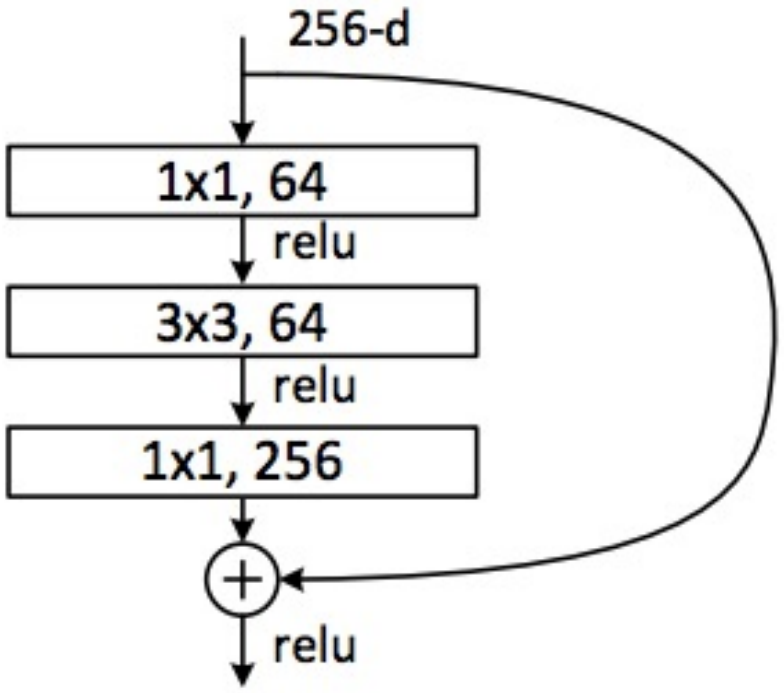



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper), 346K citations

Most cited paper of the 21st century!

# ResNet: Residual Bottleneck Module

Used in 50+ layer networks

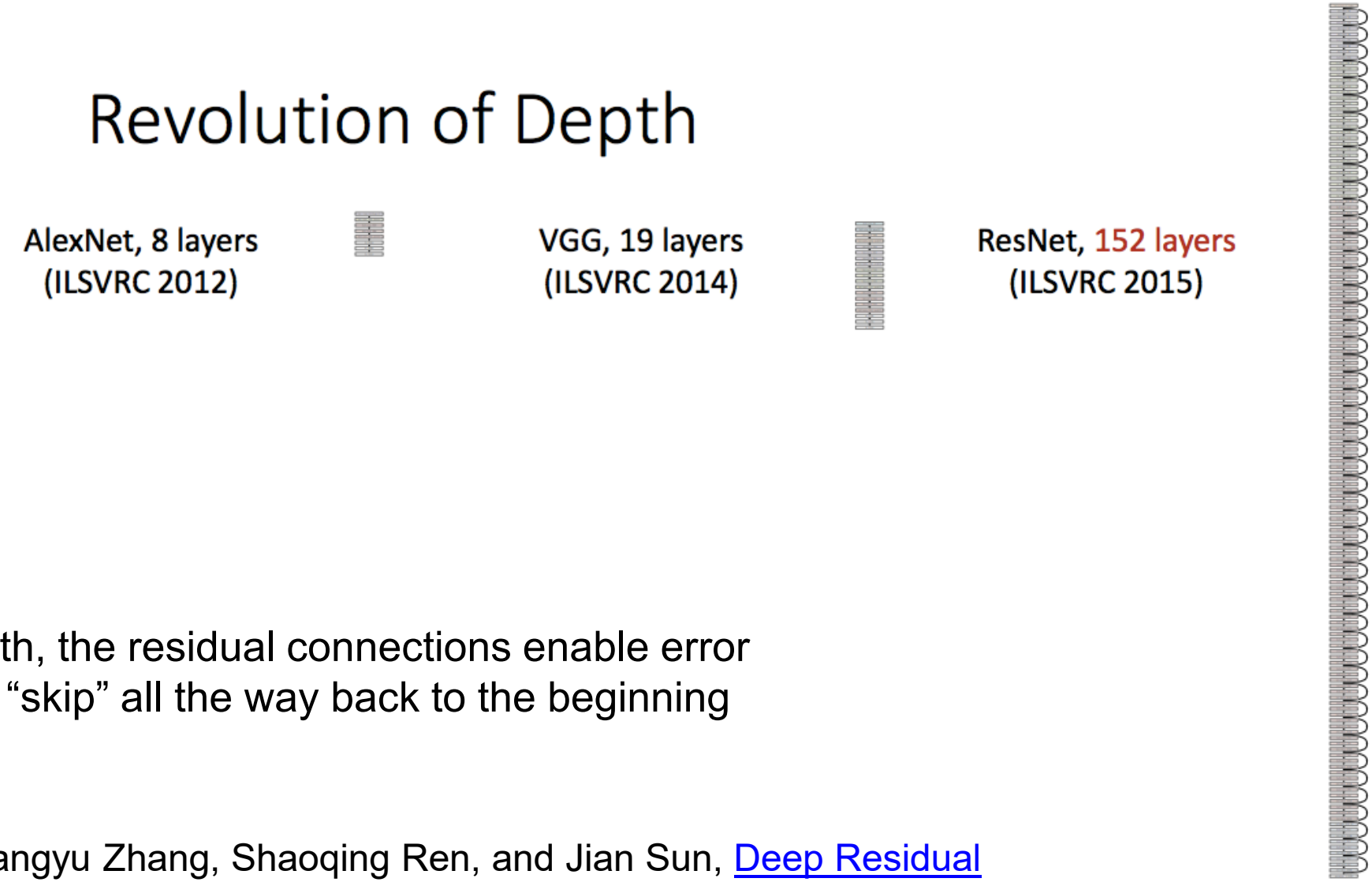



- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

Slide: Lazebnik

# ResNet: going real deep




Despite depth, the residual connections enable error gradients to “skip” all the way back to the beginning

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016

# Example code: ResBlock

"channels" = # feature maps
kernel_size = filter size, e.g. 3x3
stride = # pixels to skip when evaluating convolution
padding: to calculate filter values near edge of image/map

```
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample):
        super().__init__()
        if downsample:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1)
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=2),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.shortcut = nn.Sequential()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, input):
        output = nn.ReLU()(self.bn1(self.conv1(input)))
        output = nn.ReLU()(self.bn2(self.conv2(output)))
        shortcut = self.shortcut(input)
        output = output + shortcut
        return nn.ReLU()(output)
```

If downsampling, do it here too so dimensions match

This "+ shortcut" is the skip connection

# Example code: ResNet-18 architecture for ImageNet

```
class Network(nn.Module):
    def __init__(self, num_classes=1000):
        super().__init__()
        resblock = ResBlock
        self.layer0 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.layer1 = nn.Sequential(
            resblock(64, 64, downsample=False),
            resblock(64, 64, downsample=False)
        )
        self.layer2 = nn.Sequential(
            resblock(64, 128, downsample=True),
            resblock(128, 128, downsample=False)
        )
        self.layer3 = nn.Sequential(
            resblock(128, 256, downsample=True),
            resblock(256, 256, downsample=False)
        )
        self.layer4 = nn.Sequential(
            resblock(256, 512, downsample=True),
            resblock(512, 512, downsample=False)
        )
        self.gap = torch.nn.AdaptiveAvgPool2d(1)
        self.fc = torch.nn.Linear(512, num_classes)
```

```
    def forward(self, input):
        input = self.layer0(input)
        input = self.layer1(input)
        input = self.layer2(input)
        input = self.layer3(input)
        input = self.layer4(input)
        input = self.gap(input)
        input = torch.flatten(input, 1)
        input = self.fc(input)

        return input
```

**Forward** applies prediction, going through each layer

**Backward** applies backpropagation to compute the loss gradient with respect to parameters in each layer

Pretrained Torch models

# Batch Normalization

- As early layers adjust, the features fed into later layers keeps changing
  - This destabilizes training
- BatchNorm unit-normalizes features of each mini-batch and rescales/shifts with learned parameters $\gamma$, $\beta$
- Using BatchNorm often improves stability and speed of training

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$

**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

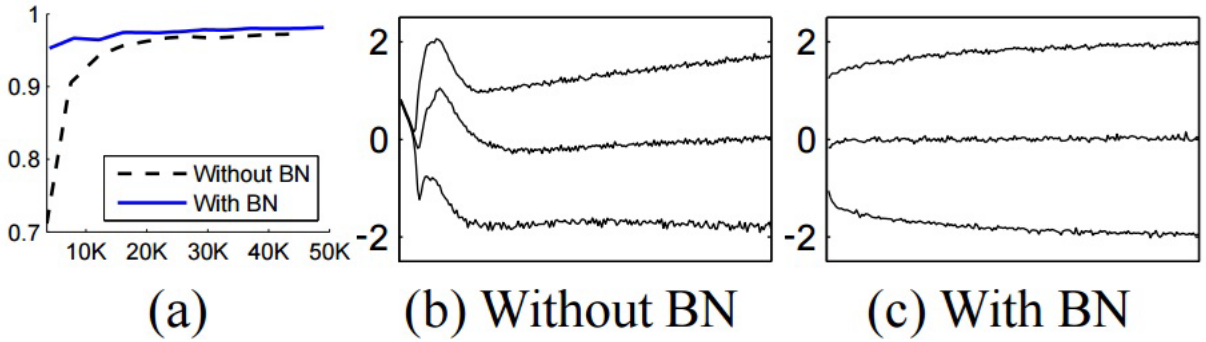
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

With BN reaches higher accuracy faster

With BN, the mean (center line) and variance (15th/85th pctl lines shown) of activations are more stable




Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]

# ResNet Architectures and Results

Network design is starting to become formulaic

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$ |
| conv5_x | 7×7 | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$ | $\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$ |
| | 1×1 | average pool, 1000-d fc, softmax | | | | |
| FLOPs | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

First case where bigger keeps getting better

| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [41] (ILSVRC'14) | - | 8.43† |
| GoogLeNet [44] (ILSVRC'14) | - | 7.89 |
| VGG [41] (v5) | 24.4 | 7.1 |
| PReLU-net [13] | 21.59 | 5.71 |
| BN-inception [16] | 21.99 | 5.81 |
| ResNet-34 B | 21.84 | 5.71 |
| ResNet-34 C | 21.53 | 5.60 |
| ResNet-50 | 20.74 | 5.25 |
| ResNet-101 | 19.87 | 4.60 |
| ResNet-152 | **19.38** | **4.49** |

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Q2-3

https://tinyurl.com/441AML-L16



# Improvements to SGD

## Great site by Lili Jiang

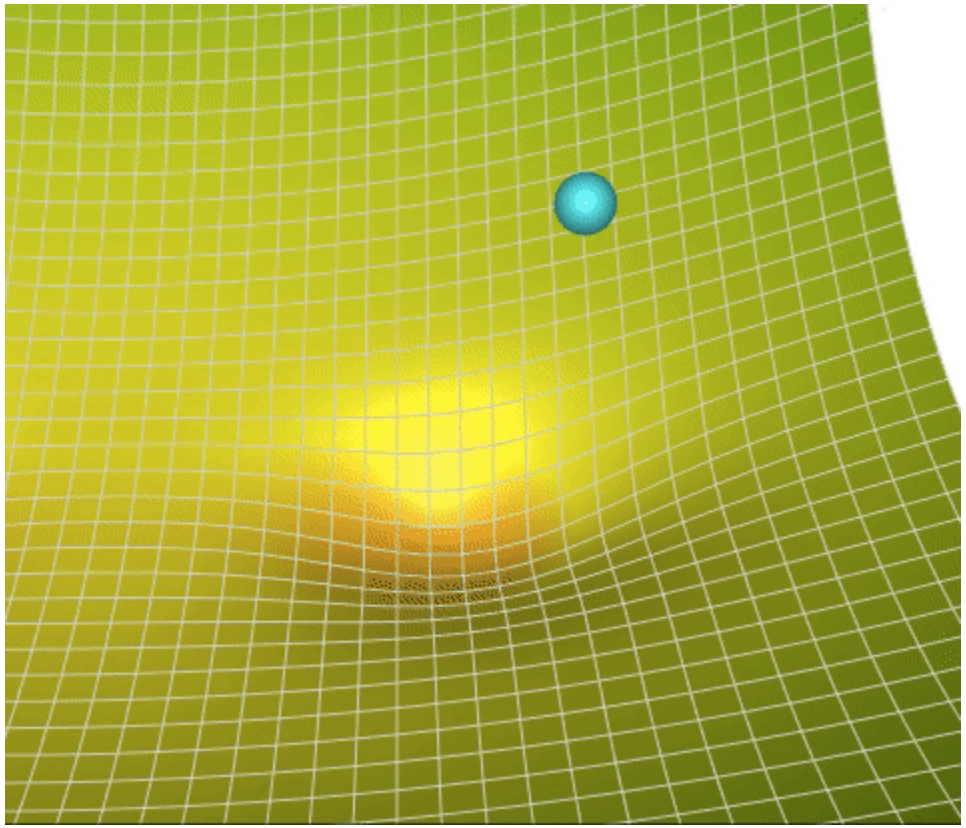
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Gradient of loss wrt weights

Basic SGD:

$\Delta w_t = -\eta g(w_t)$

$w_{t+1} = w_t + \Delta w_t$



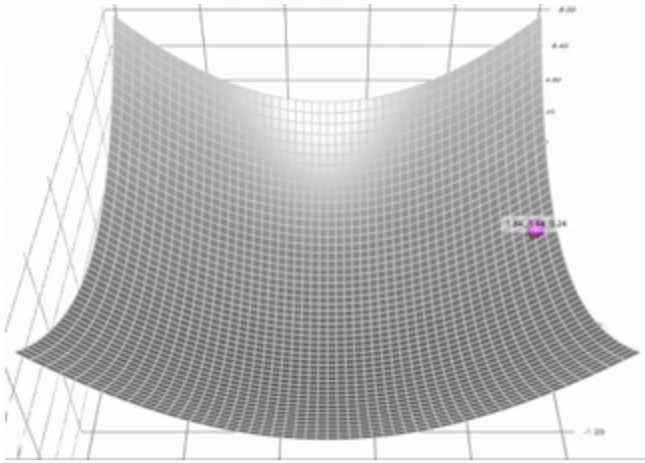
# SGD + Momentum

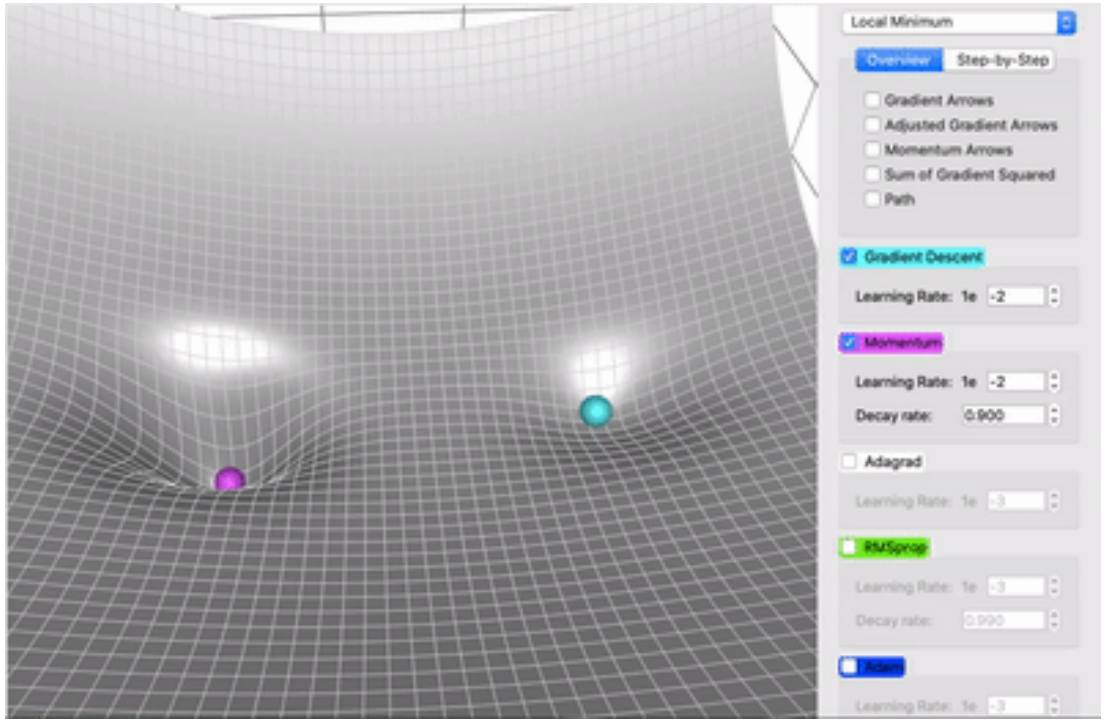
SGD + Momentum:

$m_t = \beta \cdot m_{t-1} + g(w_t)$ e.g. $\beta = .9$

$\Delta w_t = -\eta \cdot m_t$

$w_{t+1} = w_t + \Delta w_t$






Momentum (magenta) converges faster and carries the ball through a local minimum

Figure source

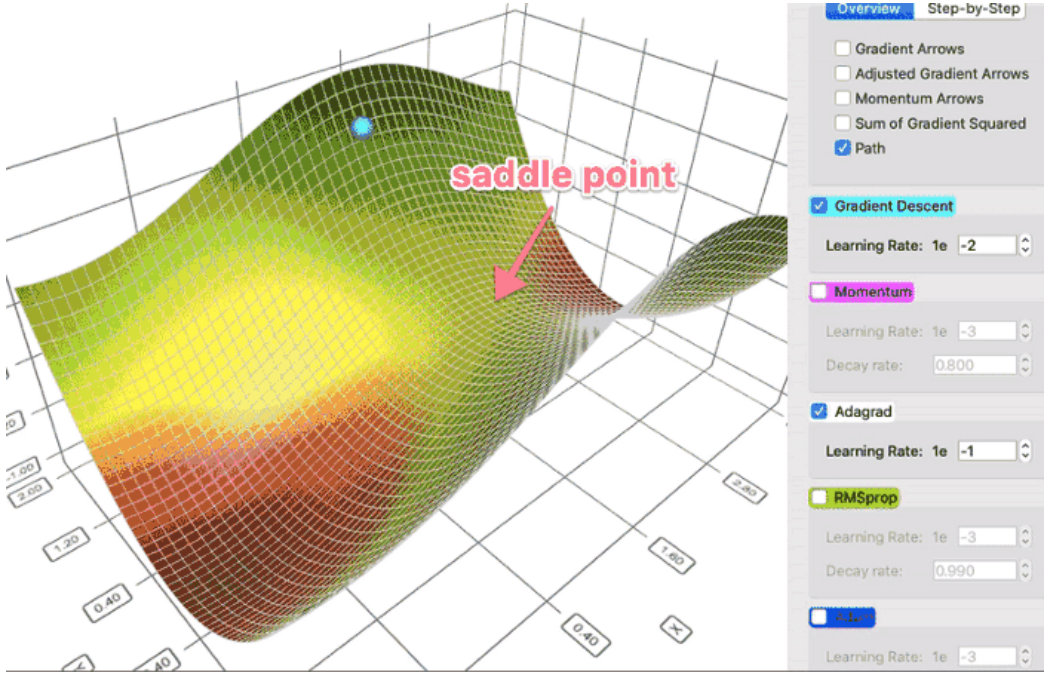
# AdaGrad: Adaptive Gradient

AdaGrad:

$g_{sq}(t) = g_{sq}(t-1) + g(w_t)^2$

$\Delta w_t = -\eta g(w_t)/\sqrt{g_{sq}(t)}$ (normalize each weight's update by path length of all previous updates)

$w_{t+1} = w_t + \Delta w_t$




AdaGrad (white) avoids moving in only one weight direction, and can lead to smoother convergence

Can be seen as setting a per-weight learning rate

Figure source

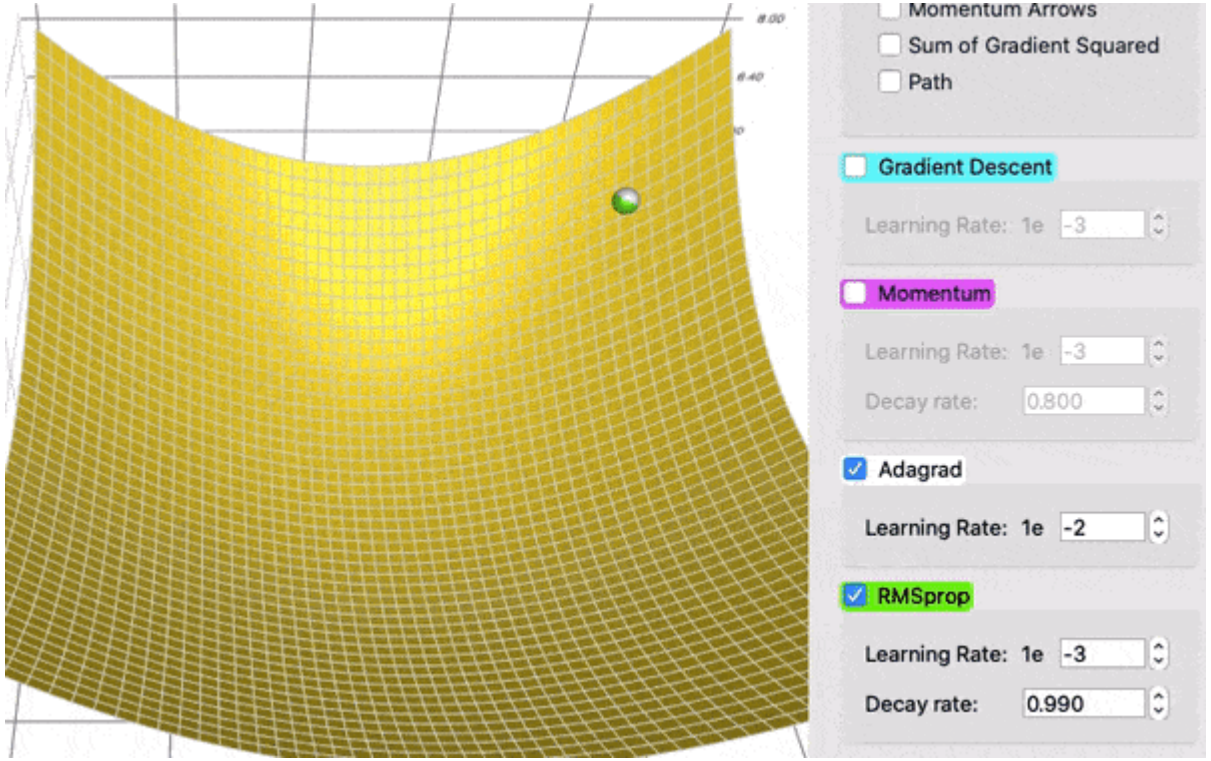
# RMSProp: Root Mean Squared Propagation

RMSProp:

$g_{sq}(t) = \epsilon \cdot g_{sq}(t-1) + (1-\epsilon) \cdot g(w_t)^2$ (introducing decay rate turns this into moving avg)

$\Delta w_t = -\eta g(w_t)/\sqrt{g_{sq}(t)}$ (normalize by moving average length of previous updates)

$w_{t+1} = w_t + \Delta w_t$




RMSProp (green) moves faster than AdaGrad (white)

Figure source

# Adam: Adaptive Moment Estimation

Adam:

$m_t = \beta \cdot m_{t-1} + (1-\beta) \cdot g(w_t)$ [momentum, $\beta = 0.9$]

$g_{sq}(t) = \epsilon \cdot g_{sq}(t-1) + (1-\epsilon) \cdot g(w_t)^2$ [RMSProp, $\epsilon = 0.999$]

$\Delta w_t = -\eta \cdot m_t / \sqrt{g_{sq}(w_t)}$

$w_{t+1} = w_t + \Delta w_t$

"AdamW" is a fix on Adam to correctly update weight decay

- use AdamW in code with weight decay, but I'll refer to both as Adam

Videos

Adam is widely used and easier to tune than SGD + momentum

Figure source

# What to use?

- Adam is less sensitive to hyperparameters (easier to get a decent solution working)
- Some computer vision practitioners say SGD+momentum can achieve the best performance, if you're able to optimize over hyperparameters
- Adam is the most used optimizer in research papers; newer optimizers such as Muon are gaining adoption in large-scale LLM pretraining

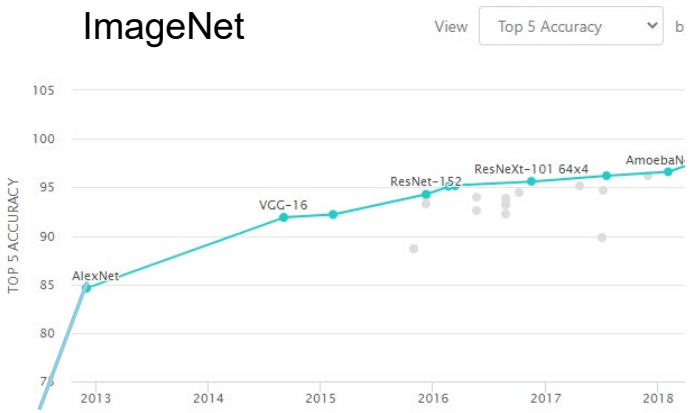
Q4

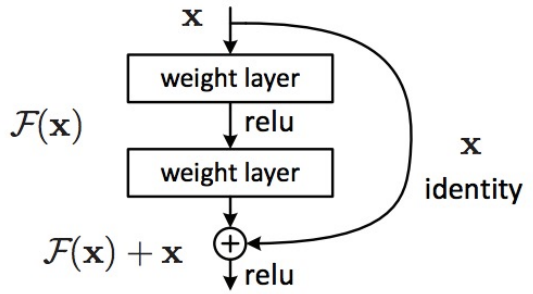
https://tinyurl.com/441AML-L16

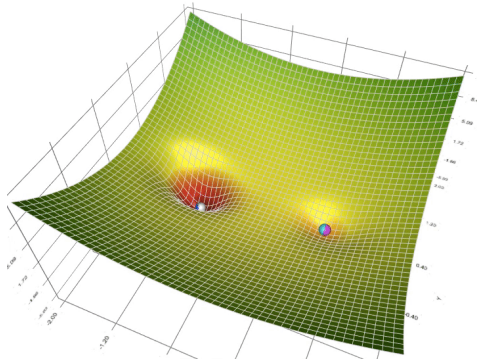


# What to remember

- Deep networks provide huge gains in performance
  - Large capacity, optimizable models
  - Learn from new large datasets
- ReLU and skip connections simplify optimization
- SGD+momentum and AdamW are the most commonly used optimizers



# Next lecture

- Defining and training a deep network w/ PyTorch
- Adopting the network to new tasks
  - Linear probe
  - Fine-tuning
- Mask RCNN recognition system