



# 10-423/10-623 Generative AI

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University

## **Pretraining vs. finetuning + Modern Transformers (RoPE, GQA, Longformer) + CNNs**

Matt Gormley  
Lecture 4  
Jan. 29, 2024

# Reminders

- **Homework 0: PyTorch + Weights & Biases**
  - Out: Wed, Jan 17
  - Due: Wed, Jan 24 at 11:59pm
  - unique policy for this assignment: we will grant (essentially) any and all extension requests
- **Homework 1: Generative Models of Text**
  - Out: Thu, Jan 25
  - Due: Wed, Feb 7 at 11:59pm
- **Quiz 1: Wed, Jan 31**

# Recap So Far

## Deep Learning

- AutoDiff
  - is a tool for **computing gradients** of a differentiable function,  $b = f(a)$
  - the key building block is a **module** with a `forward()` and `backward()`
  - sometimes define `f` as **code** in `forward()` by chaining existing modules together
- Computation Graphs
  - are another way to define `f` (more conducive to slides)
  - so far, we saw two (deep) computation graphs
    - 1) RNN-LM
    - 2) Transformer-LM
    - (Transformer-LM was kind of complicated)

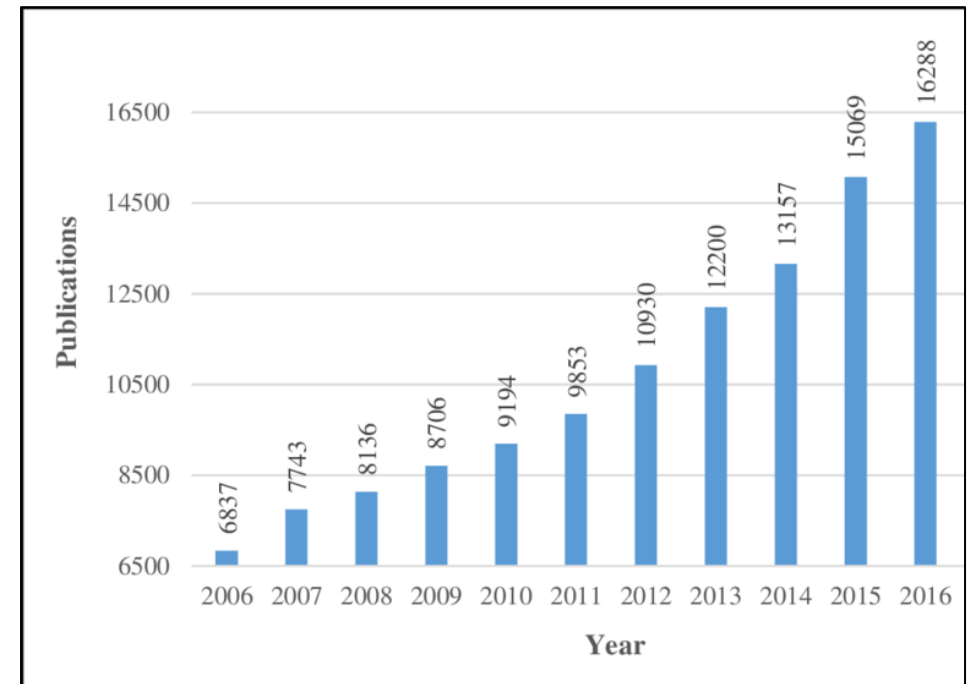
## Language Modeling

- key idea: condition on previous words to **sample the next word**
- to define the **probability** of the next word...
  - ... n-gram LM uses collection of massive 50k-sided **dice**
  - ... RNN-LM or Transformer-LM use a **neural network**
- Learning an LM
  - n-gram LMs are easy to learn: just **count** co-occurrences!
  - a RNN-LM / Transformer-LM is trained by **optimizing an objective function with SGD; compute gradients with AutoDiff**

# **PRE-TRAINING VS. FINE-TUNING**

# The Start of Deep Learning

- The architectures of modern deep learning have a long history:
  - 1960s: Rosenblatt's 3-layer multi-layer perceptron, ReLU )
  - 1970-80s: RNNs and CNNs
  - 1990s: linearized self-attention
- The spark for deep learning came in 2006 thanks to **pre-training** (e.g., Hinton & Salakhutdinov, 2006)



# Deep Network Training

- **Idea #1:**

1. Supervised fine-tuning only

- **Idea #2:**

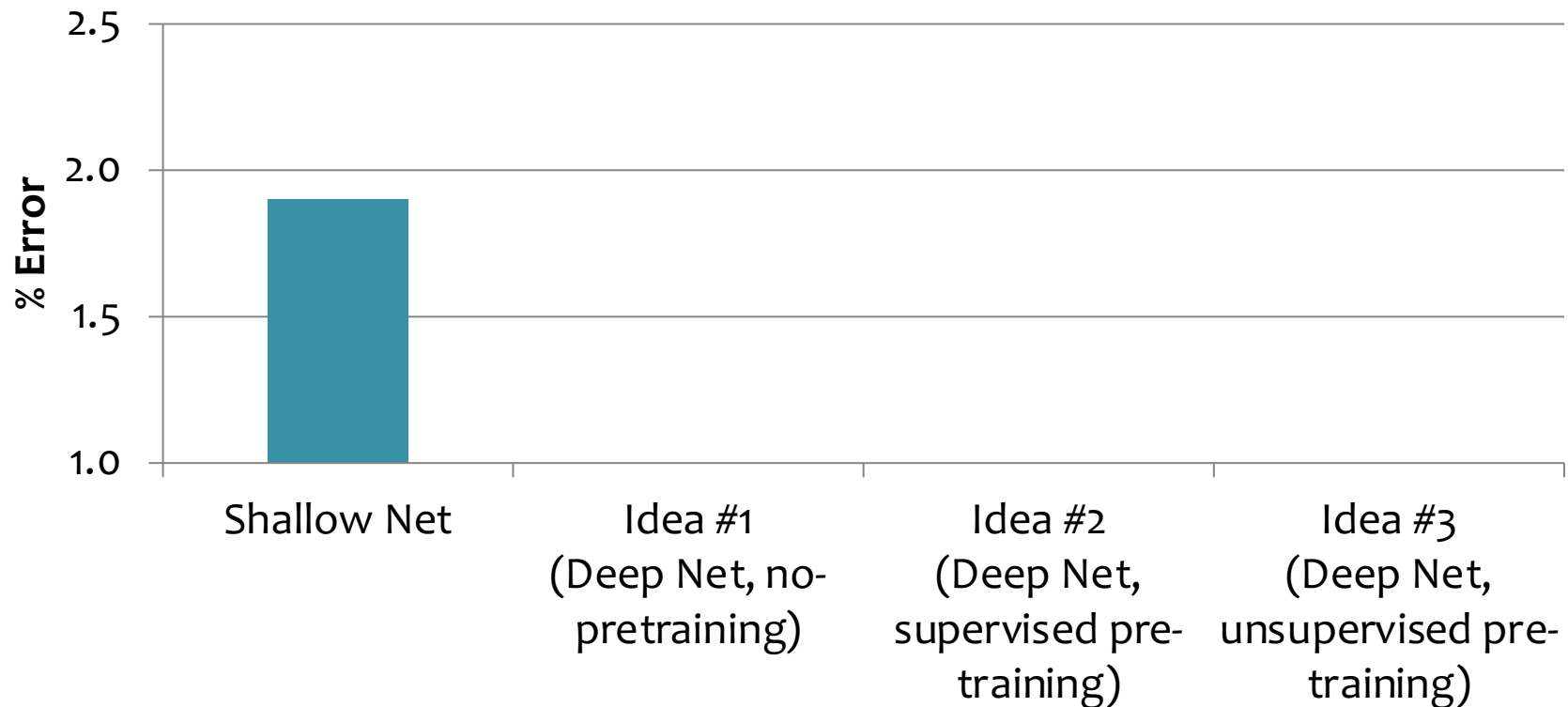
1. Supervised layer-wise pre-training
2. Supervised fine-tuning

- **Idea #3:**

1. Unsupervised layer-wise pre-training
2. Supervised fine-tuning

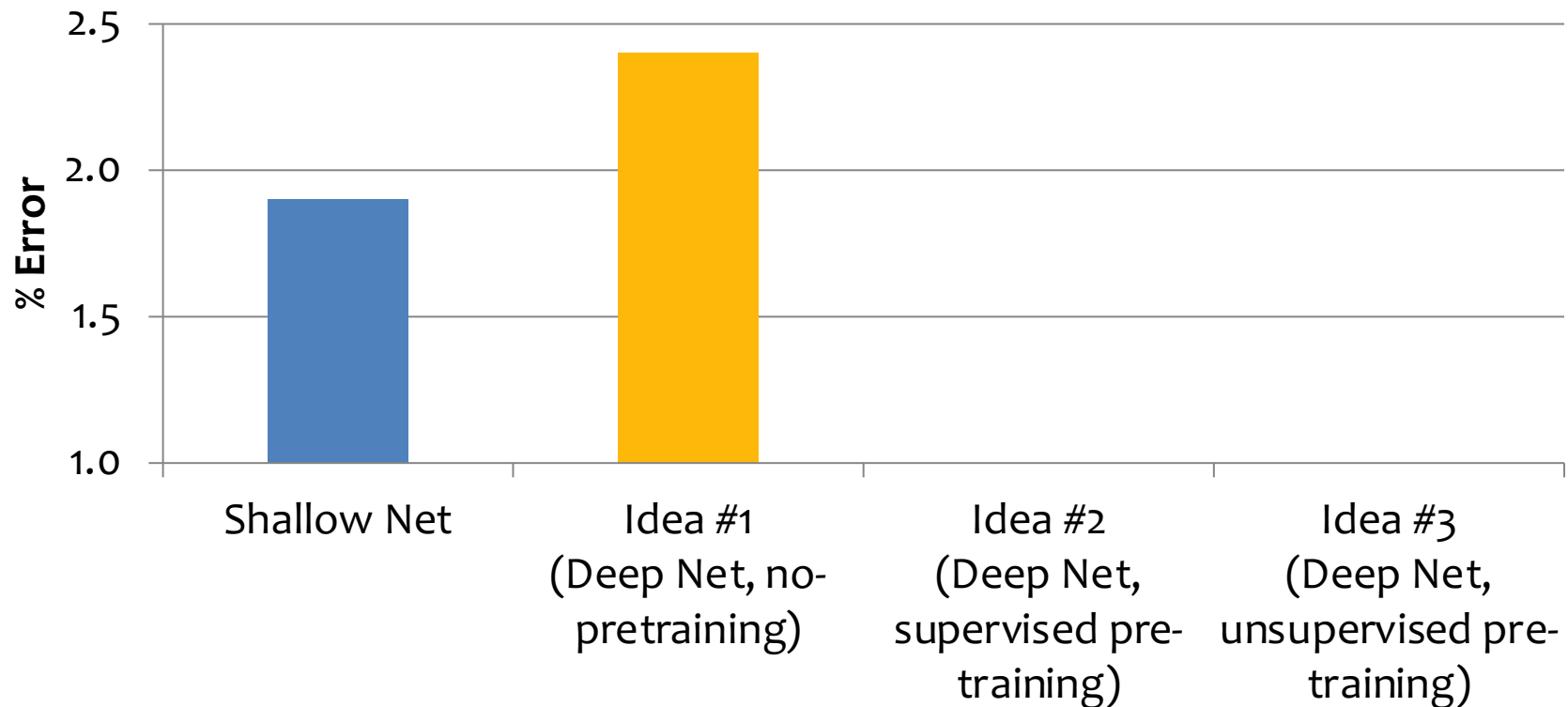
# Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



# Comparison on MNIST

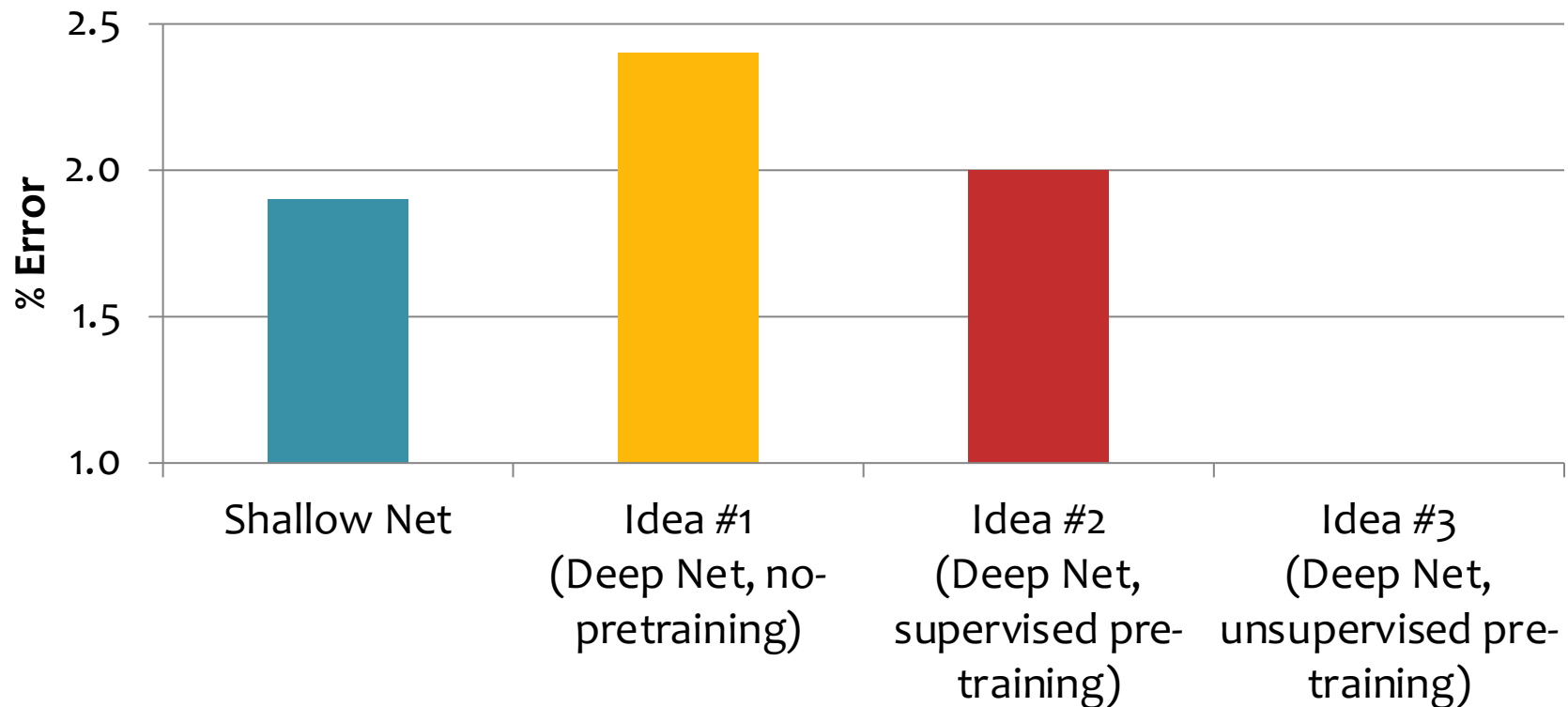
- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)





# Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



# Idea #3: Unsupervised Pre-training

- **Idea #3: (Two Steps)**

- Use our original idea, but **pick a better starting point**
- **Train each level** of the model in a **greedy way**

1. Unsupervised Pre-training

- Use **unlabeled** data
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.

2. Supervised Fine-tuning

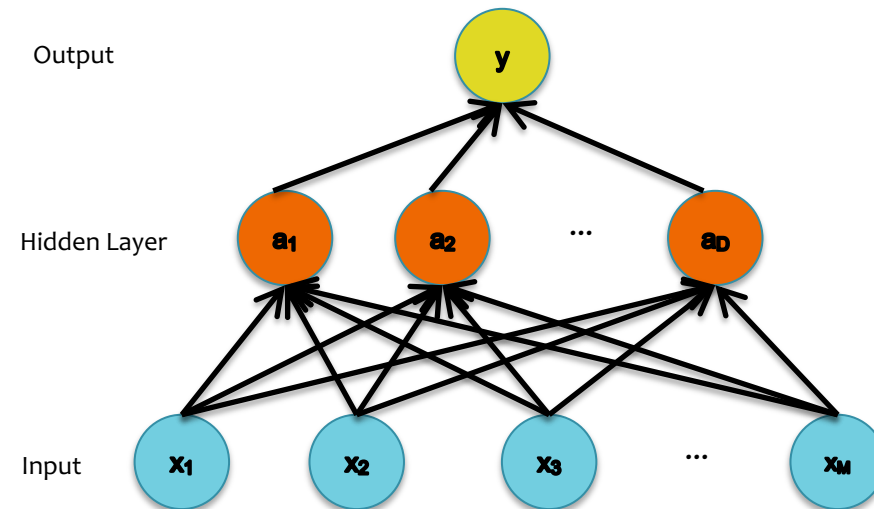
- Use **labeled** data to train following “Idea #1”
- Refine the features by backpropagation so that they become tuned to the end-task

# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**



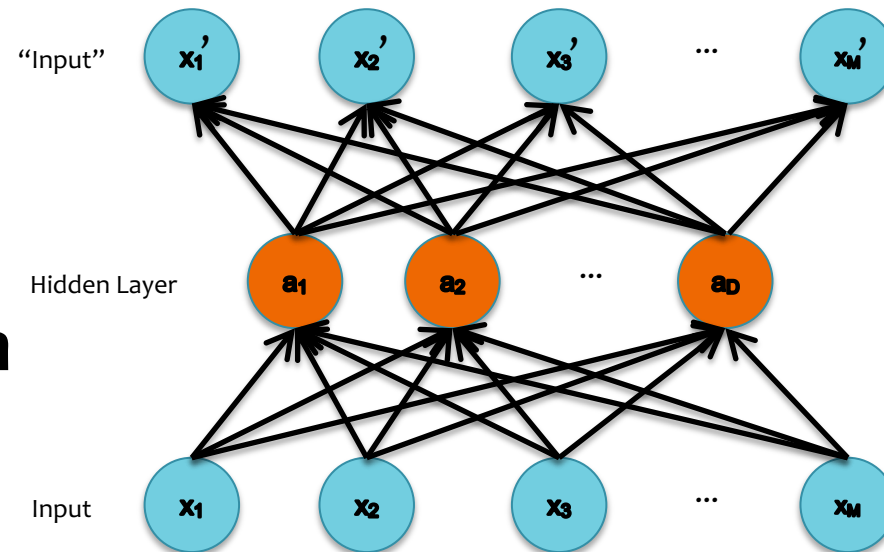
# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

**This topology defines an Auto-encoder.**



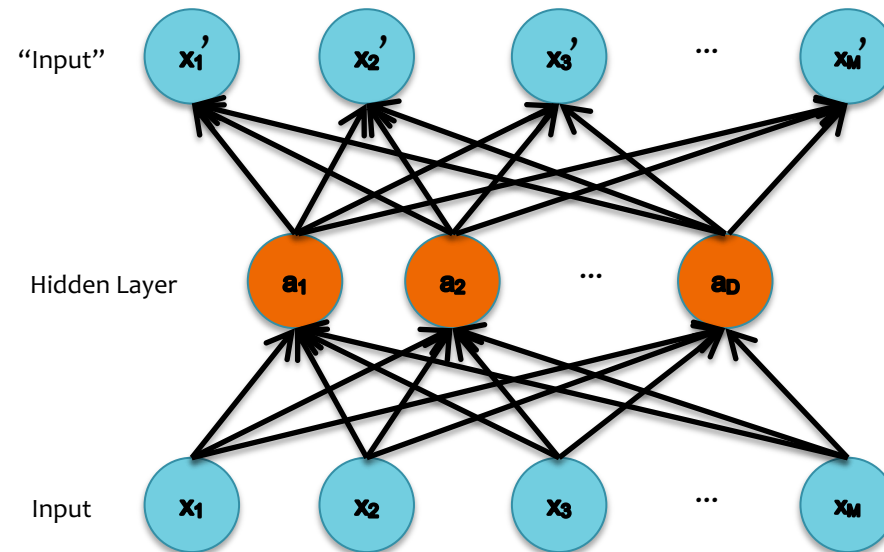
# Auto-Encoders

Key idea: Encourage  $z$  to give small reconstruction error:

- $x'$  is the *reconstruction* of  $x$
- Loss =  $\|x - \text{DECODER}(\text{ENCODER}(x))\|^2$
- Train with the same backpropagation algorithm for 2-layer Neural Networks with  $x_m$  as both input and output.

DECODER:  $x' = h(W'z)$

ENCODER:  $z = h(Wx)$

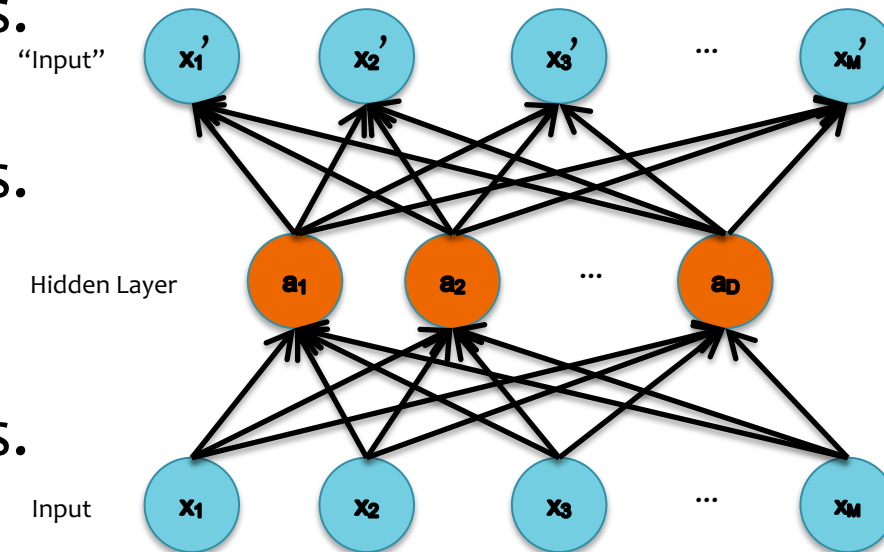


# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.

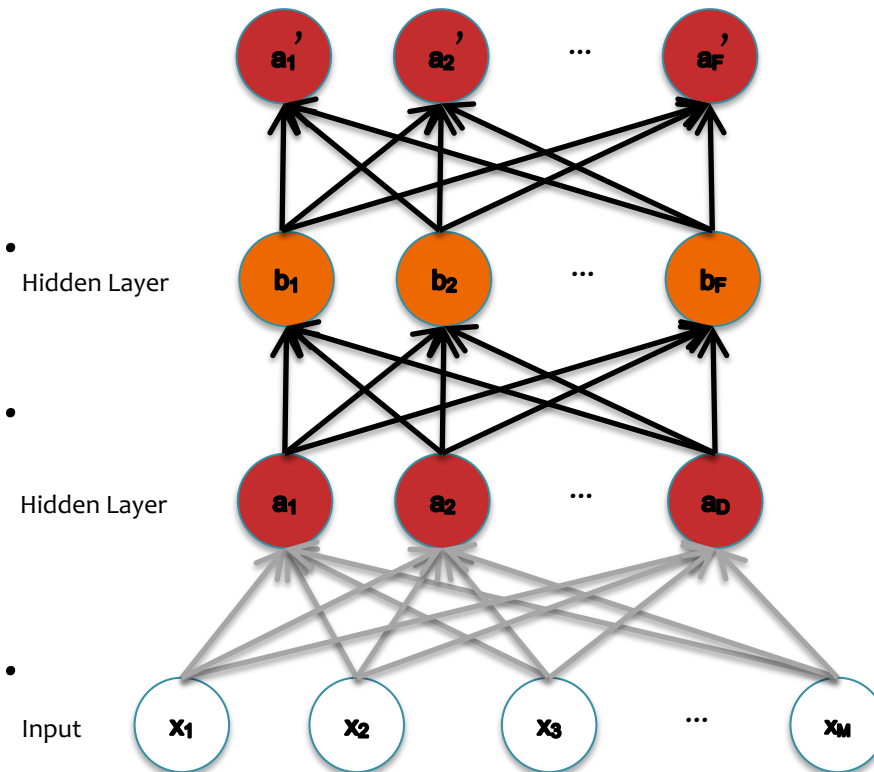


# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer n. Then fix its parameters.

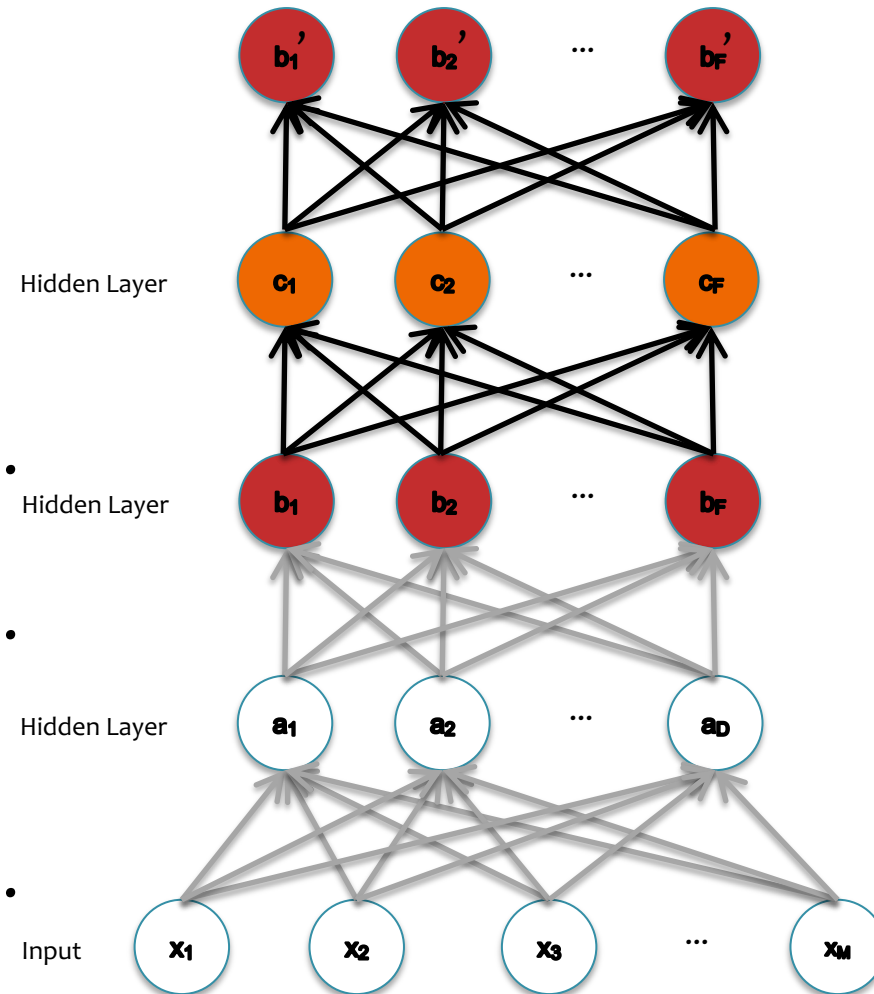


# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer  $n$ . Then fix its parameters.





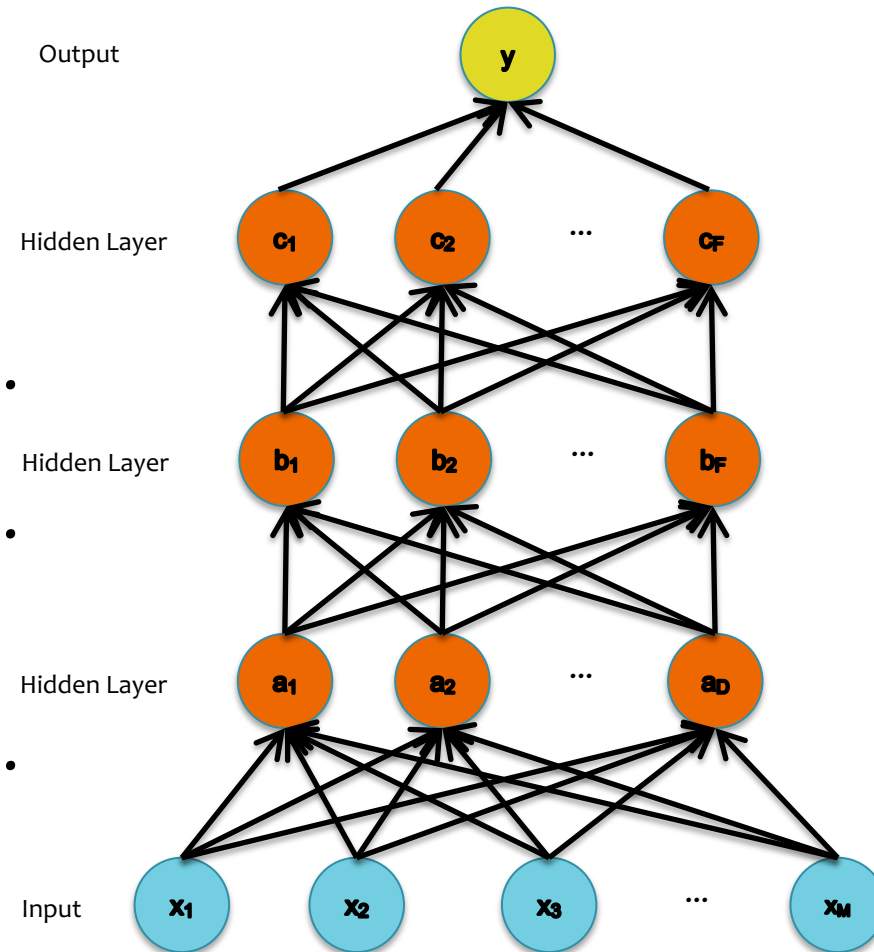
# The solution:

## *Unsupervised pre-training*

### Unsupervised pre-training

- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - ...
  - Train hidden layer  $n$ . Then fix its parameters.

**Supervised fine-tuning**  
Backprop and update all parameters



# Deep Network Training

- **Idea #1:**

1. Supervised fine-tuning only

- **Idea #2:**

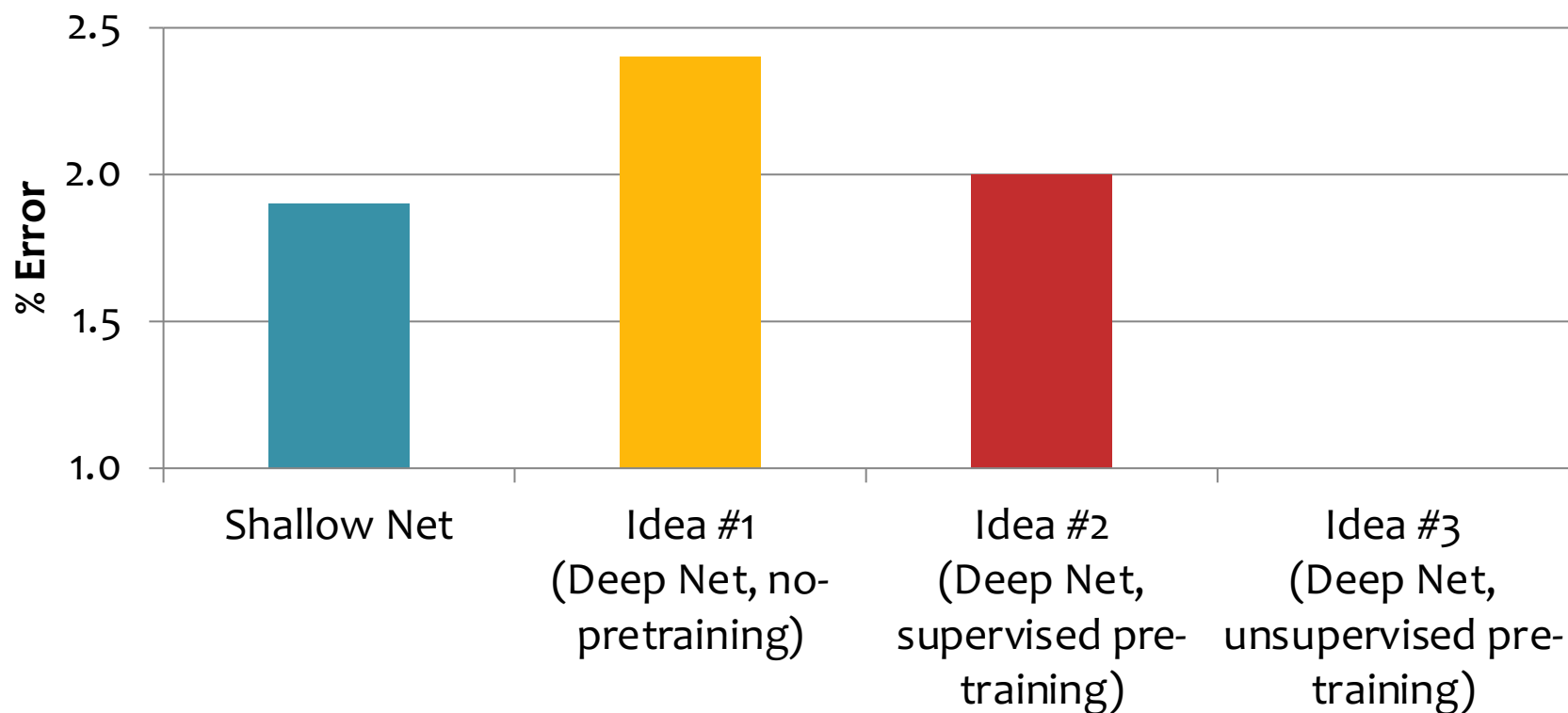
1. Supervised layer-wise pre-training
2. Supervised fine-tuning

- **Idea #3:**

1. Unsupervised layer-wise pre-training
2. Supervised fine-tuning

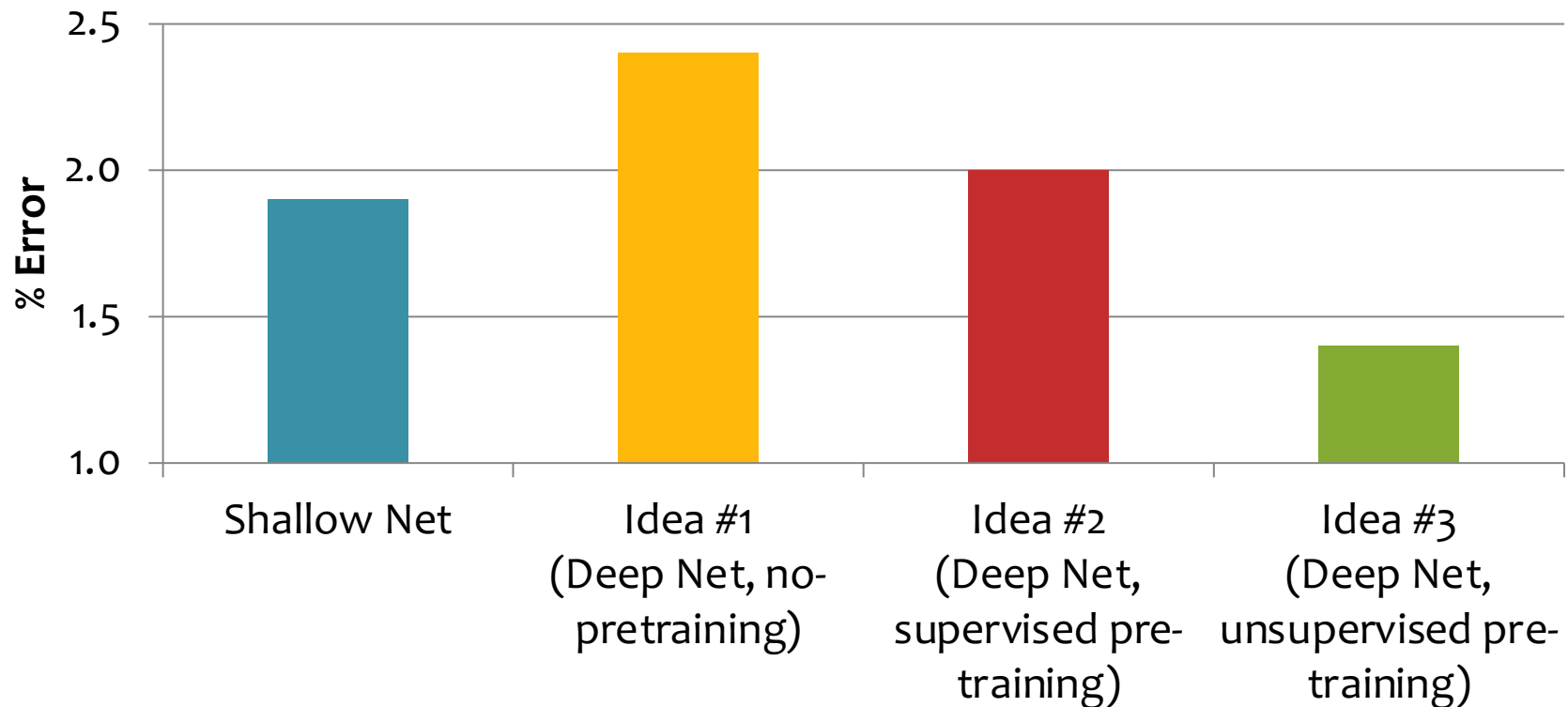
## Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)



## Comparison on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)





# Training Data for LLMs

## GPT-3 Training Data:

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

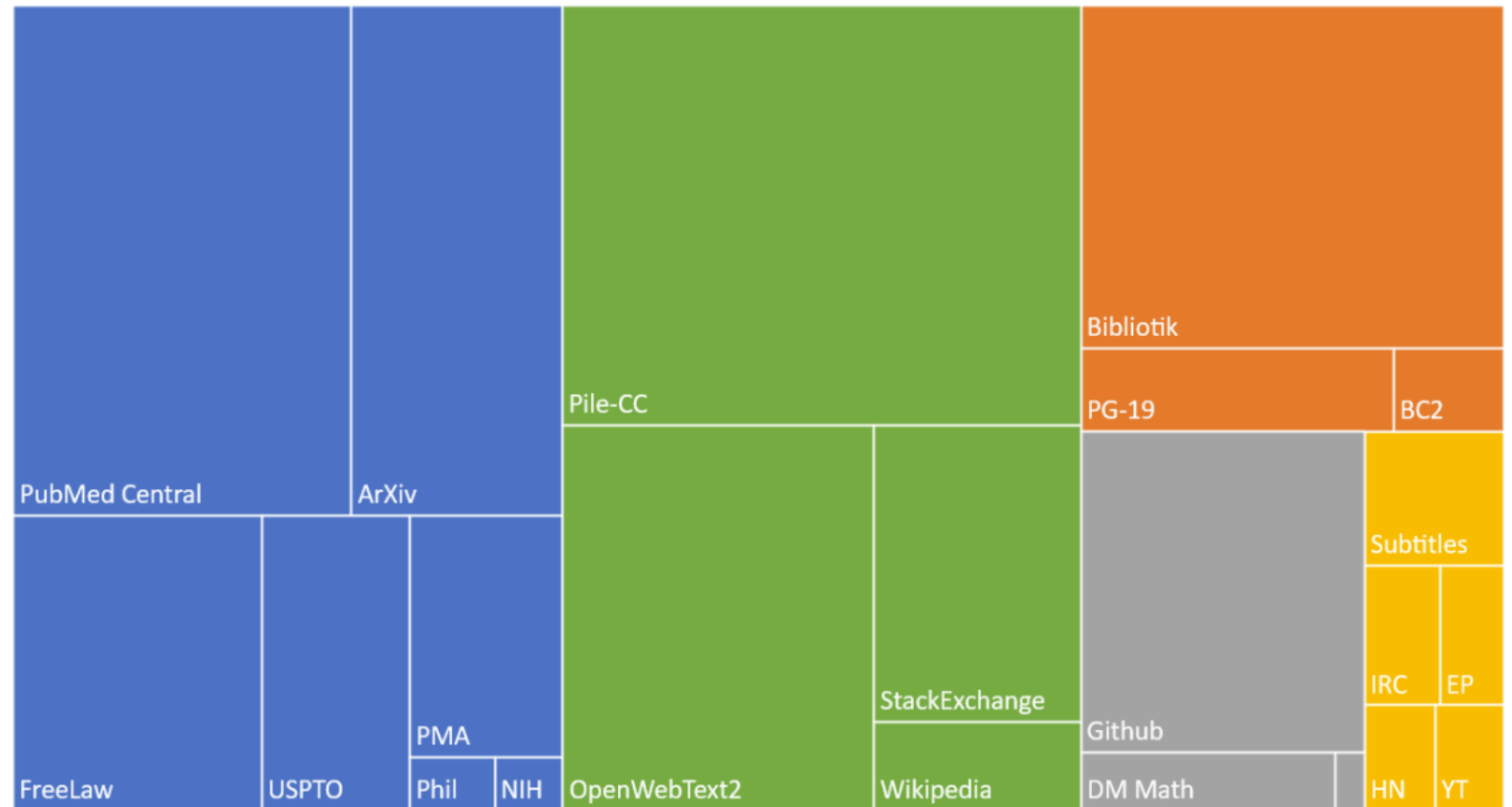
# Training Data for LLMs

## The Pile:

- An open source dataset for training language models
- Comprised of 22 smaller datasets
- Favors high quality text
- 825 Gb  $\approx$  1.2 trillion tokens

Composition of the Pile by Category

■ Academic ■ Internet ■ Prose ■ Dialogue ■ Misc



# **MODERN TRANSFORMER MODELS**



# Modern Transformer Models

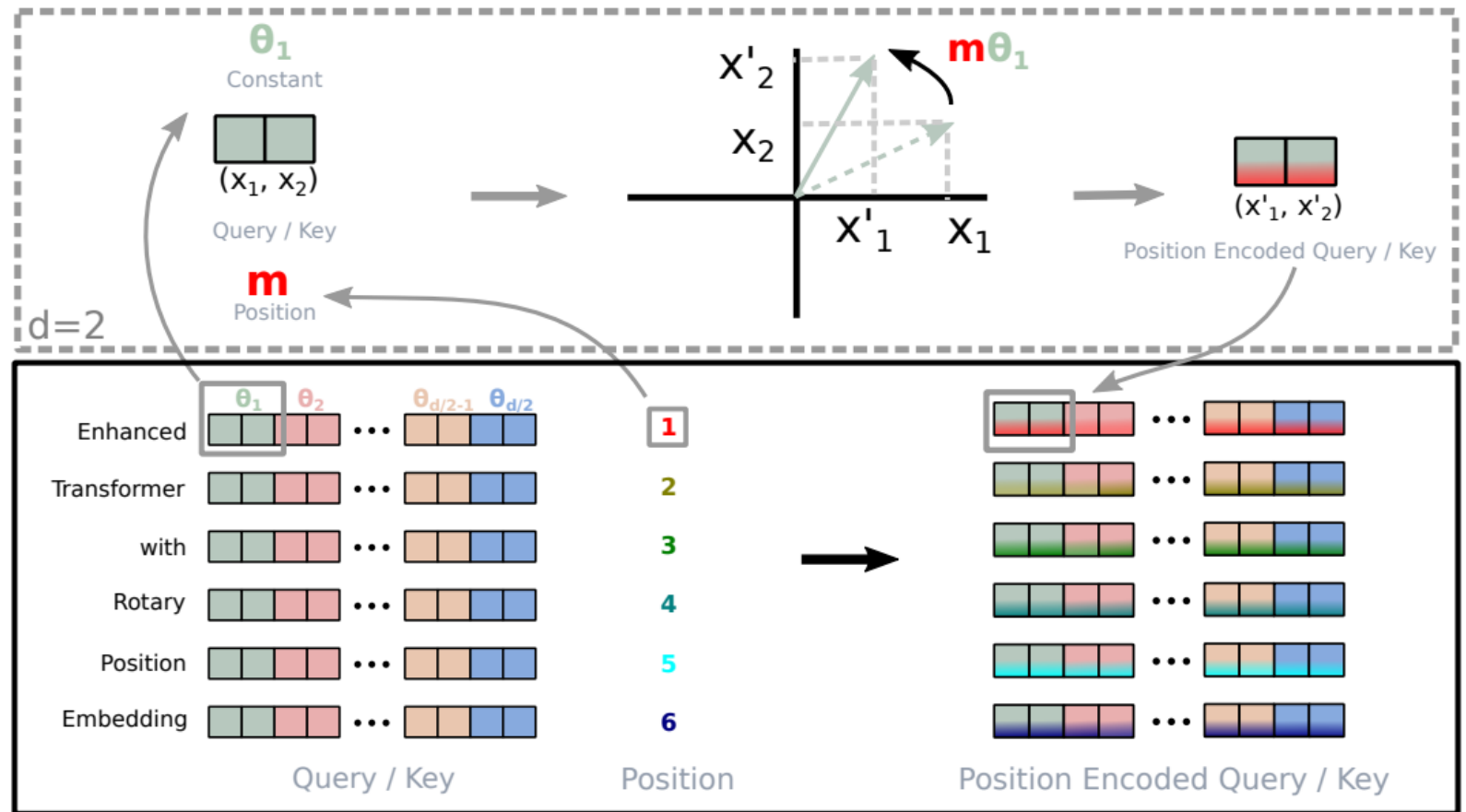
- PaLM (Oct 2022)
  - 540B parameters
  - closed source
  - Model:
    - SwiGLU instead of ReLU, GELU, or Swish
    - **multi-query attention (MQA)** instead of multi-headed attention
    - **rotary position embeddings**
    - **shared input-output embeddings** instead of separate parameter matrices
  - Training: **Adafactor** on 780 billion tokens
- Llama-1 (Feb 2023)
  - collection of models of varying parameter sizes: 7B, 13B, 32B, 65B
  - semi-open source
  - Llama-13B outperforms GPT-3 on average
  - Model compared to GPT-3:
    - **RMSNorm** on inputs instead of LayerNorm on outputs
    - **SwiGLU** activation function instead of ReLU
    - **rotary position embeddings (RoPE)** instead of absolute
  - Training: **AdamW** on 1.0 – 1.4 trillion tokens
- Falcon (June - Nov 2023)
  - models of size 7B, 40B, 180B
  - first fully open source model, Apache 2.0
  - Model compared to Llama-1:
    - (GQA) instead of multi-headed attention (MHA) or **grouped query attention multi-query attention (MQA)**
    - **rotary position embeddings** (worked better than Alibi)
    - **GeLU** instead of SwiGLU
  - Training: AdamW on up to 3.5 trillion tokens for 180B model, using **z-loss** for stability and **weight decay**
- Llama-2 (Aug 2023)
  - collection of models of varying parameter sizes: 7B, 13B, 70B.
  - introduced Llama 2-Chat, fine-tuned as a dialogue agent
  - Model compared to Llama-1:
    - **grouped query attention (GQA)** instead of multi-headed attention (MHA)
    - context length of 4096 instead of 2048
  - Training: AdamW on 2.0 trillion tokens
- Mistral 7B (Oct 2023)
  - outperforms Llama-2 13B on average
  - introduced Mistral 7B – Instruct, fine-tuned as a dialogue agent
  - truly open source: Apache 2.0 license
  - Model compared to Llama-2
    - **sliding window attention** (with  $W=4096$ ) and grouped-query attention (GQA) instead of just GQA
    - context length of 8192 instead of 4096 (can generate sequences up to length 32K)
    - **rolling buffer cache** (grow the KV cache and the overwrite position  $i$  into position  $i \bmod W$ )
  - variant Mixtral offers a **mixture of experts** (roughly 8 Mistral models)

In this section we'll look at three techniques:

1. rotary position embeddings (RoPE)
2. grouped query attention (GQA)
3. sliding window attention

# Rotary Position Embeddings (RoPE)

- Rotary position embeddings are a kind of **relative** position embeddings
- Key idea:
  - break each  $d$ -dimensional input vector into  $d/2$  vectors of length 2
  - rotate each of the  $d/2$  vectors by an amount scaled by  $m$
  - $m$  is the absolute position of the query or the key



# Rotary Position Embeddings (RoPE)

Standard attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j$$

$$\tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$

$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

where  $d = d_k/2$ ,  $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$ .

For some fixed absolute position  $m$ , the rotary matrix  $\mathbf{R}_{\Theta,m} \in \mathbb{R}^{d_k \times d_k}$  is given by:

$$R_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos m\theta_{d_k/2} & -\sin m\theta_{d_k/2} \\ 0 & 0 & 0 & 0 & \dots & \sin m\theta_{d_k/2} & \cos m\theta_{d_k/2} \end{pmatrix}$$

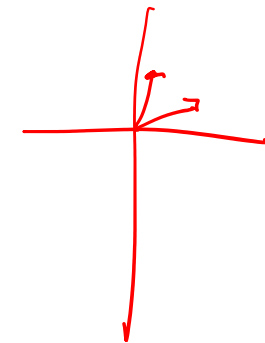
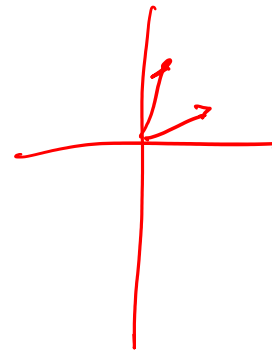
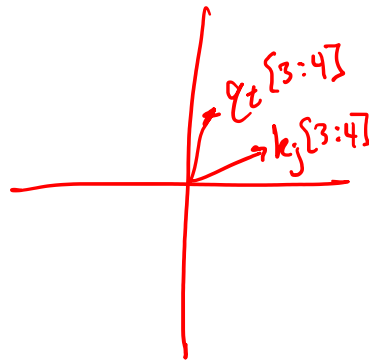
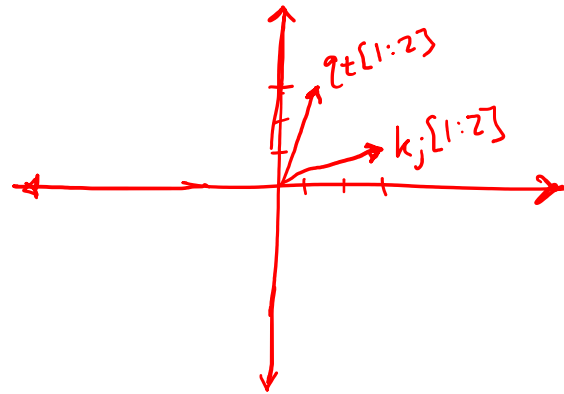
The  $\theta_i$  parameters are fixed ahead of time and defined as below.

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$$

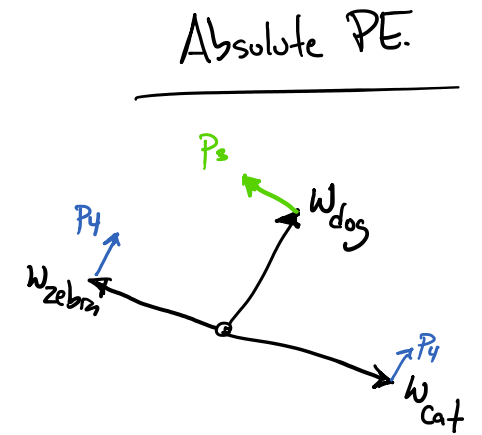
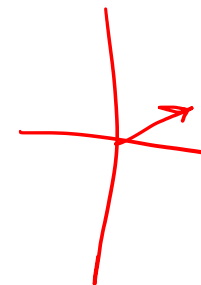
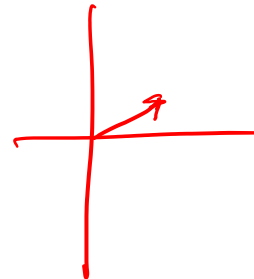
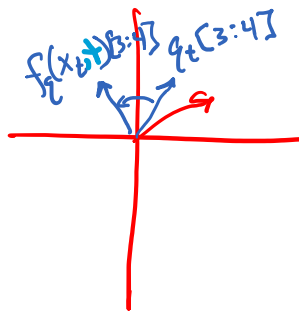
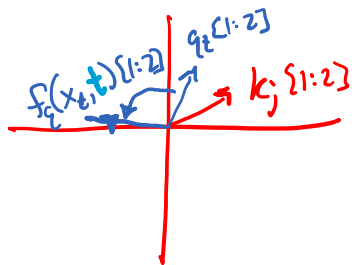
# Rotary Position Embeddings (RoPE)

$$q_t = W_q^T x_t = [1, 3, 1, 3, 1, 3]$$

$$k_j = W_k^T x_j = [3, 1, 3, 1, 3, 1]$$



$$f_q(x_t, t) = R_{\theta, t} q_t$$



# Rotary Position Embeddings (RoPE)

Standard attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j$$

$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$

Because of the block sparse pattern in  $\mathbf{R}_{\theta,m}$ , we can efficiently compute the matrix-vector product of  $\mathbf{R}_{\theta,m}$  with some arbitrary vector  $\mathbf{y}$  in a more efficient manner:

$$\mathbf{R}_{\Theta,m} \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{d-1} \\ y_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -y_2 \\ y_1 \\ -y_4 \\ y_3 \\ \vdots \\ -y_d \\ y_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

# Matrix Version of RoPE

## RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta, j} \mathbf{q}_j$$

$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta, j} \mathbf{k}_j$$

## Matrix Version:

$$\mathbf{Q} = \mathbf{XW}_q$$

$$\tilde{\mathbf{Q}} = g(\mathbf{Q}; \Theta)$$

$$\mathbf{S} = \tilde{\mathbf{Q}}\tilde{\mathbf{K}}^T / \sqrt{d_k}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$

$$\mathbf{K} = \mathbf{XW}_k$$

$$\tilde{\mathbf{K}} = g(\mathbf{K}; \Theta)$$

**Goal:** to construct a new matrix  $\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$  such that  $\tilde{\mathbf{Y}}_{m,\cdot} = \mathbf{R}_{\Theta, m} \mathbf{y}_m$

$$\mathbf{C} = \left[ \begin{array}{ccc|ccc} 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} & 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ N\theta_1 & \cdots & N\theta_{\frac{d}{2}} & N\theta_1 & \cdots & N\theta_{\frac{d}{2}} \end{array} \right]$$

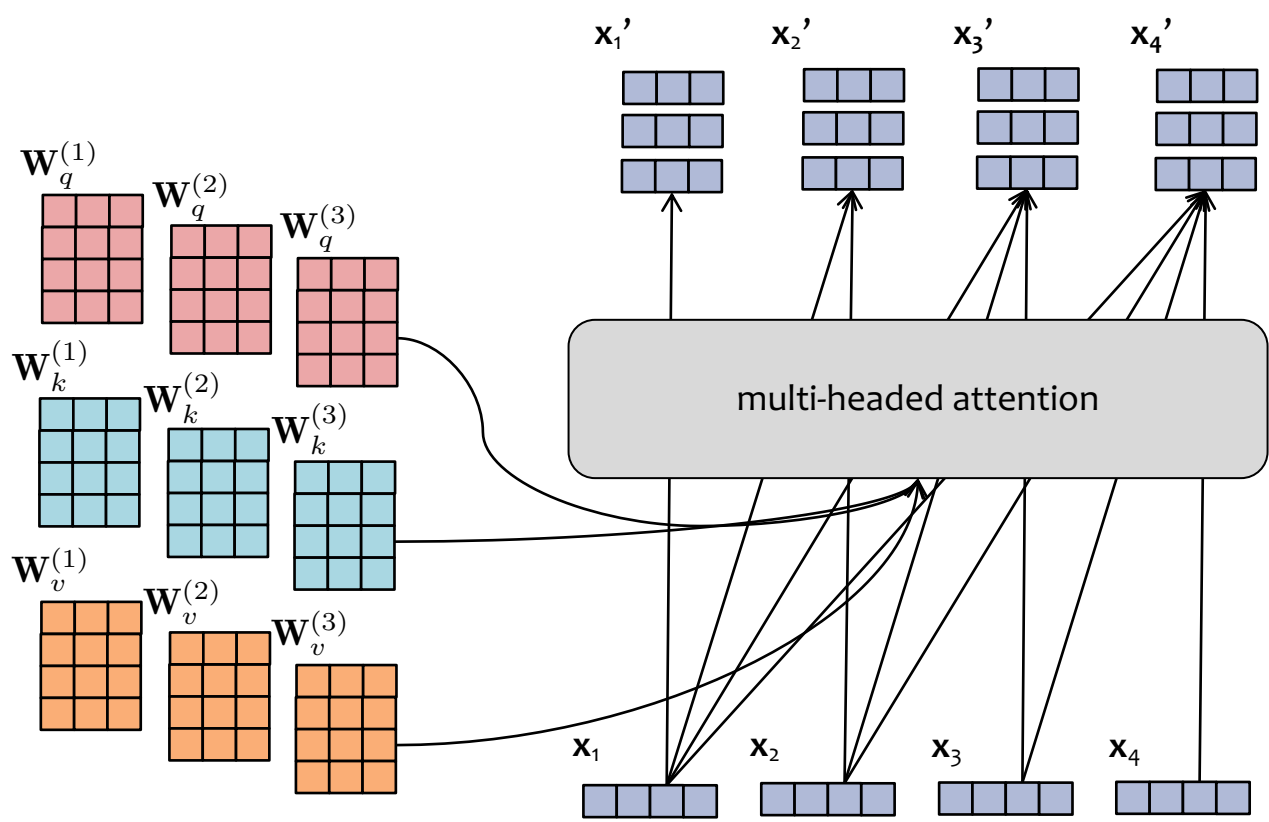
$$\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$$

$$= \left[ \mathbf{Y}_{\cdot, 1:d/2} \mid \mathbf{Y}_{\cdot, d/2+1:d} \right] \otimes \cos(\mathbf{C})$$

$$+ \left[ -\mathbf{Y}_{\cdot, d/2+1:d} \mid \mathbf{Y}_{\cdot, 1:d/2} \right] \otimes \sin(\mathbf{C})$$

# Matrix Version of Multi-Headed (Causal) Attention

$$\mathbf{X} = \text{concat}(\mathbf{X}'^{(1)}, \dots, \mathbf{X}'^{(h)})$$



$$\mathbf{X}'^{(i)} = \text{softmax} \left( \frac{\mathbf{Q}^{(i)} (\mathbf{K}^{(i)})^T}{\sqrt{d_k}} + \mathbf{M} \right) \mathbf{V}^{(i)}$$

$$\mathbf{Q}^{(i)} = \mathbf{X} \mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X} \mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X} \mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_4]^T$$

# Grouped Query Attention (GQA)

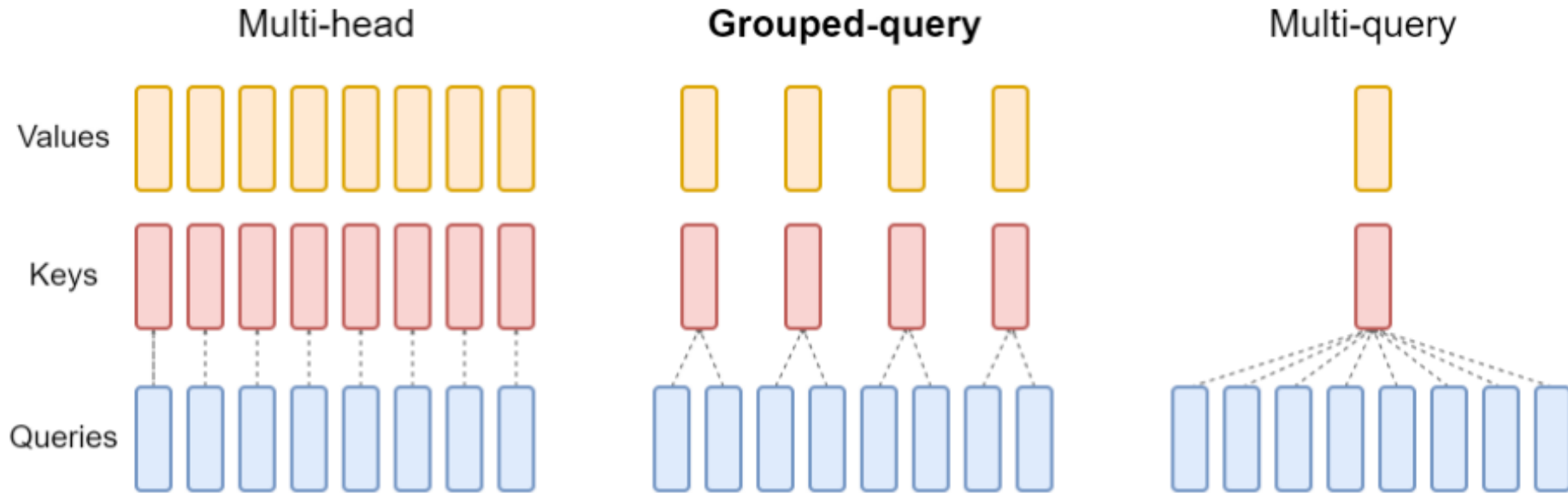


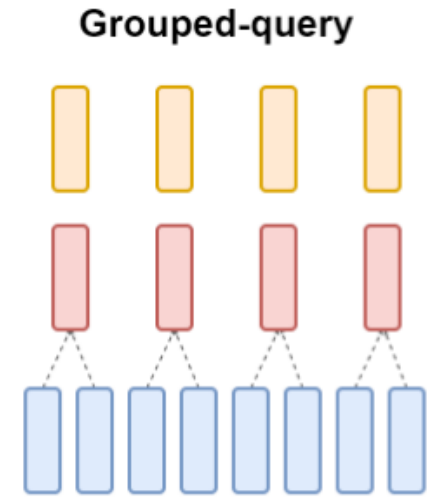
Figure 2: Overview of grouped-query method. Multi-head attention has  $H$  query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.



# Grouped Query Attention (GQA)

- **Key idea:** reuse the same key-value heads for multiple different query heads
- **Parameters:** The parameter matrices are all the same size, but we now have fewer key/value parameter matrices (heads) than query parameter matrices (heads)

- $h_q$  = the number of query heads
- $h_{kv}$  = the number of key/value heads
- Assume  $h_q$  is divisible by  $h_{kv}$
- $g = h_q/h_{kv}$  is the size of each group (i.e. the number of query vectors per key/value vector).



$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_T]^T$$

$$\mathbf{V}^{(i)} = \mathbf{XW}_v^{(i)}, \forall i \in \{1, \dots, d_{kv}\}$$

$$\mathbf{K}^{(i)} = \mathbf{XW}_k^{(i)}, \forall i \in \{1, \dots, d_{kv}\}$$

$$\mathbf{Q}^{(i,j)} = \mathbf{XW}_q^{(i,j)}, \forall i \in \{1, \dots, d_{kv}\}, \forall j \in \{1, \dots, g\}$$

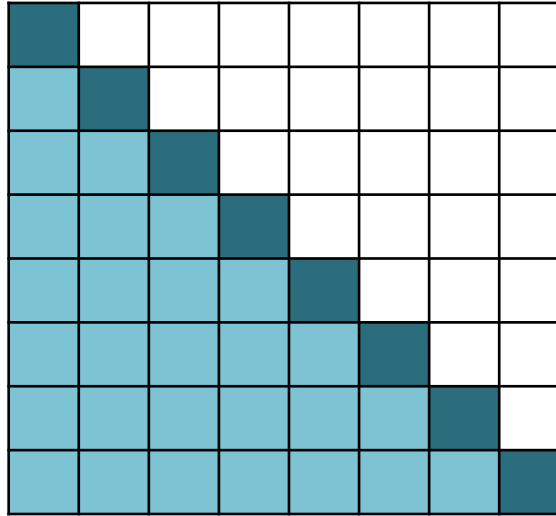
# Sliding Window Attention

## Sliding Window Attention

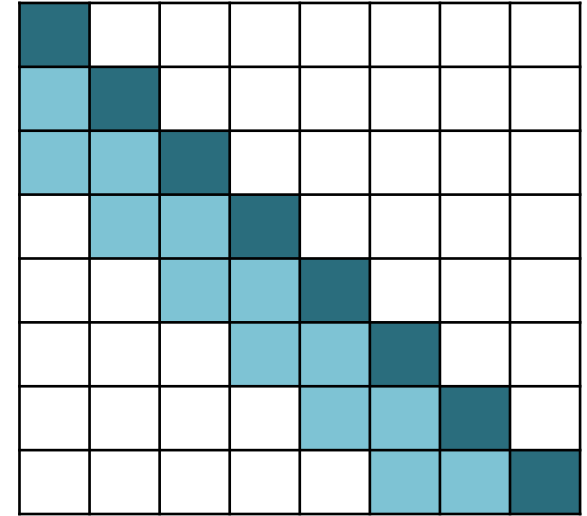
- also called “local attention” and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of  $(\frac{1}{2}w+1)$  tokens, with the rightmost window element being the current token (i.e. on the diagonal)

$$\mathbf{X}' = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M} \right) \mathbf{V}$$

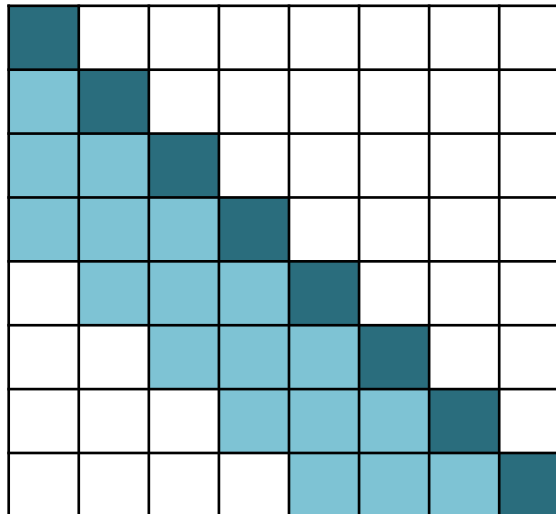
regular causal attention



sliding window attention (w=4)



sliding window attention (w=6)

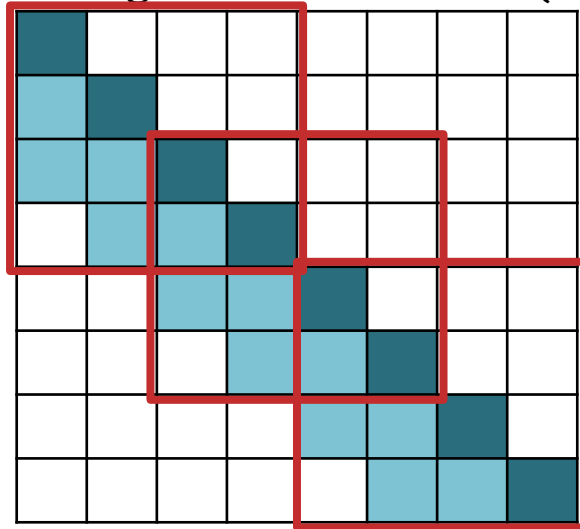


# Sliding Window Attention

## Sliding Window Attention

- also called “local attention” and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of  $(\frac{1}{2}w+1)$  tokens, with the rightmost window element being the current token (i.e. on the diagonal)

sliding window attention (w=4)



## 3 ways you could implement

1. *naïve implementation:* just do the matrix multiplication, but this is still slow
2. *for-loop implementation:* asymptotically faster / less memory, but unusable in practice b/c for-loops in PyTorch are too slow
3. *sliding chunks implementation:* break into Q and K into chunks of size  $w \times w$ , with overlap of  $\frac{1}{2}w$ ; then compute full attention within each chunk and mask out chunk (very fast/low memory in practice)

$$\mathbf{X}' = \text{softmax} \left( \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M} \right) \mathbf{V}$$

# **BACKGROUND: COMPUTER VISION**

# Example: Image Classification

- ImageNet LSVRC-2011 contest:
  - **Dataset:** 1.2 million labeled images, 1000 classes
  - **Task:** Given a new image, label it with the correct class
  - **Multiclass** classification problem
- Examples from <http://image-net.org/>

## Bird

Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings

2126 pictures

92.85% Popularity Percentile

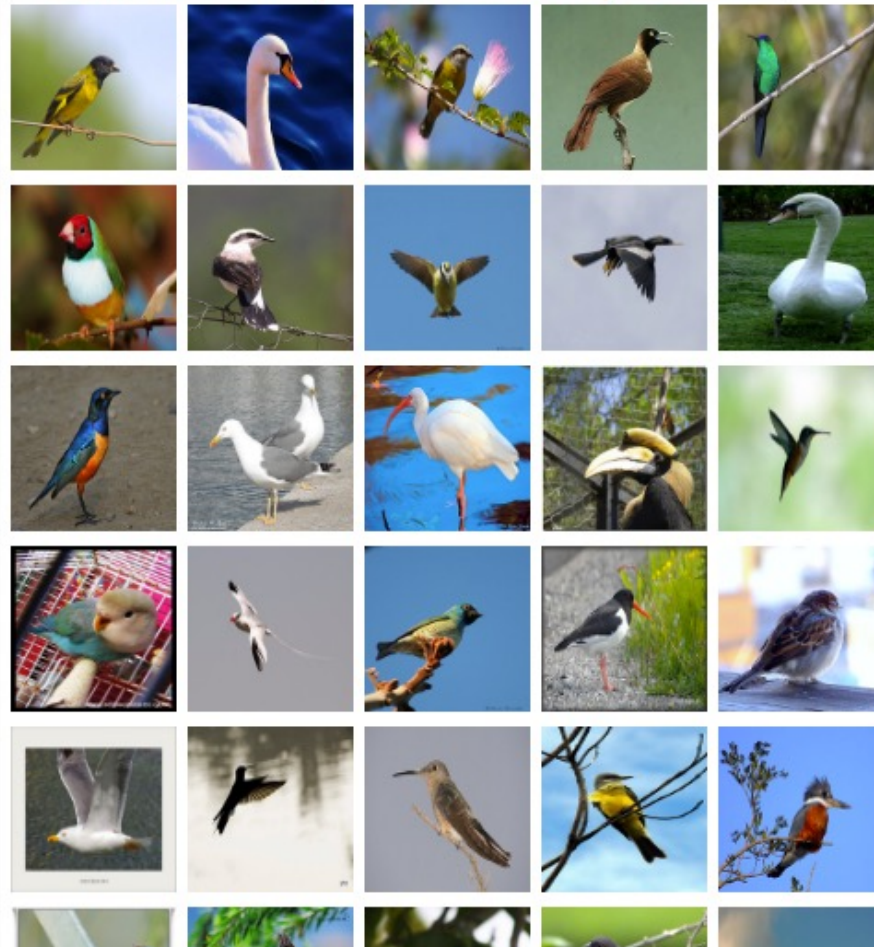


- marine animal, marine creature, sea animal, sea creature (1)
- scavenger (1)
- biped (0)
- predator, predatory animal (1)
- larva (49)
- acrodont (0)
- feeder (0)
- stunt (0)
- chordate (3087)**
  - tunicate, urochordate, urochord (6)
  - cephalochordate (1)
  - vertebrate, craniate (3077)**
    - mammal, mammalian (1169)
    - bird (871)**
      - dickeybird, dickey-bird, dickybird, dicky-bird (0)
      - cock (1)
      - hen (0)
      - nester (0)
      - night bird (1)
      - bird of passage (0)
      - protoavis (0)
      - archaeopteryx, archeopteryx, Archaeopteryx lithographi
      - Sinornis (0)
      - Ibero-mesornis (0)
      - archaeornis (0)
      - ratite, ratite bird, flightless bird (10)
      - carinate, carinate bird, flying bird (0)
      - passerine, passeriform bird (279)
      - nonpasserine bird (0)
      - bird of prey, raptor, raptorial bird (80)
      - gallinaceous bird, gallinacean (114)

Treemap Visualization

**Images of the Synset**

Downloads



# German iris, *Iris kochii*

Iris of northern Italy having deep blue-purple flowers; similar to but smaller than *Iris germanica*

469 pictures

49.6% Popularity Percentile



- ... halophyte (0)
- ... succulent (39)
- ... cultivar (0)
- ... cultivated plant (0)
- ... weed (54)
- ... evergreen, evergreen plant (0)
- ... deciduous plant (0)
- ... vine (272)
- ... creeper (0)
- ... woody plant, ligneous plant (1868)
- ... geophyte (0)
- ... desert plant, xerophyte, xerophytic plant, xerophile, xerophilic mesophyte, mesophytic plant (0)
- ... aquatic plant, water plant, hydrophyte, hydrophytic plant (11)
- ... tuberous plant (0)
- ... bulbous plant (179)
  - ... iridaceous plant (27)
    - ... iris, flag, fleur-de-lis, sword lily (19)
      - ... bearded iris (4)
        - ... Florentine iris, orris, *Iris germanica florentina*, *Iris German iris, Iris germanica* (0)
        - ... **German iris, *Iris kochii*** (0)
        - ... Dalmatian iris, *Iris pallida* (0)
      - ... beardless iris (4)
      - ... bulbous iris (0)
      - ... dwarf iris, *Iris cristata* (0)
      - ... stinking iris, gladdon, gladdon iris, stinking gladwyn, Persian iris, *Iris persica* (0)
      - ... yellow iris, yellow flag, yellow water flag, *Iris pseudo*
      - ... dwarf iris, vernal iris, *Iris verna* (0)
      - ... blue flag, *Iris versicolor* (0)

Treemap Visualization

Images of the Synset

Downloads



# Court, courtyard

An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"

165 pictures

92.61% Popularity Percentile



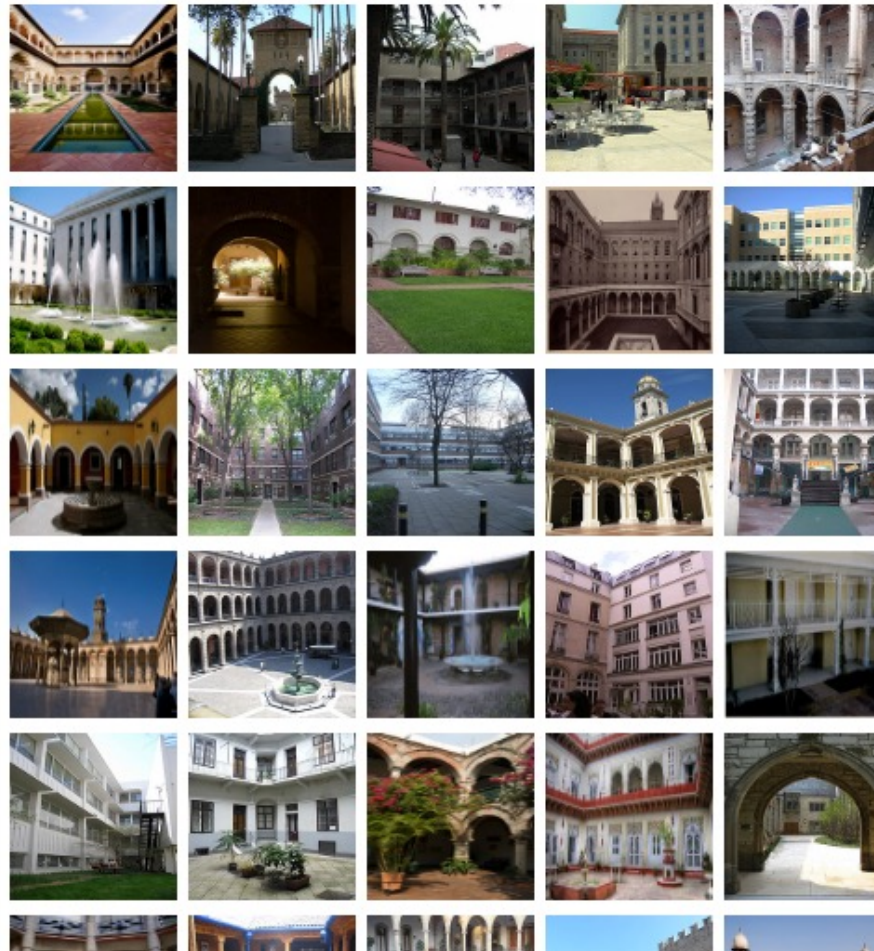
Numbers in brackets: (the number of synsets in the subtree).

- ImageNet 2011 Fall Release (32326)
  - plant, flora, plant life (4486)
  - geological formation, formation (175)
  - natural object (1112)
  - sport, athletics (176)
  - artifact, artefact (10504)
    - instrumentality, instrumentation (5494)
    - structure, construction (1405)
      - airdock, hangar, repair shed (0)
      - altar (1)
      - arcade, colonnade (1)
      - arch (31)
      - area (344)
        - aisle (0)
        - auditorium (1)
        - baggage claim (0)
        - box (1)
        - breakfast area, breakfast nook (0)
        - bullpen (0)
        - chancel, sanctuary, bema (0)
        - choir (0)
        - corner, nook (2)
        - court, courtyard (6)
          - atrium (0)
          - bailey (0)
          - cloister (0)
          - food court (0)
          - forecourt (0)
          - narvis (0)

Treemap Visualization

Images of the Synset

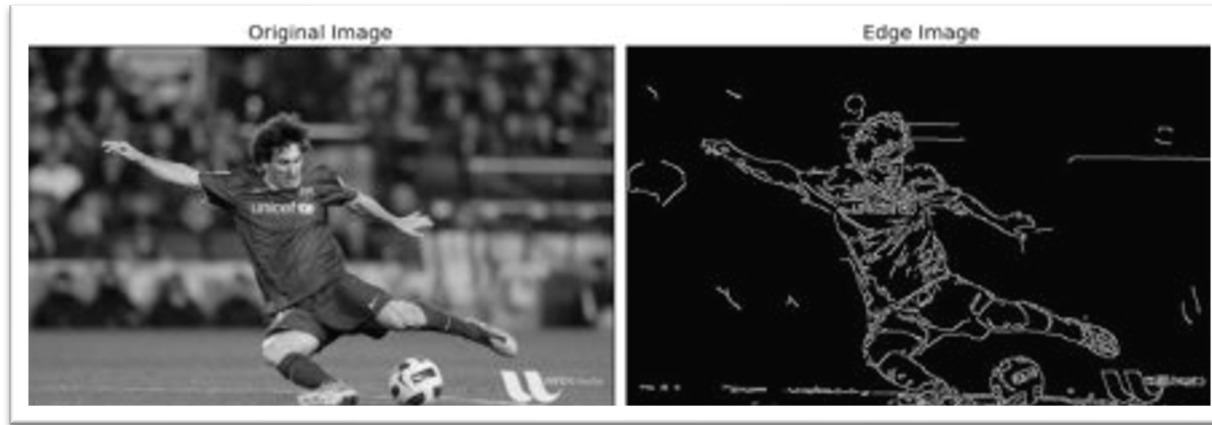
Downloads



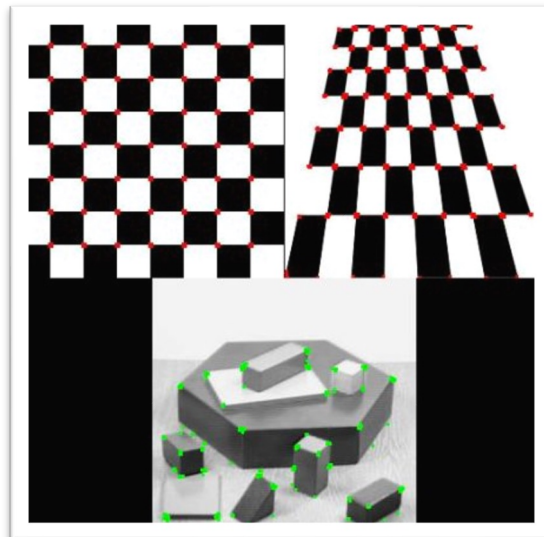


# Feature Engineering for CV

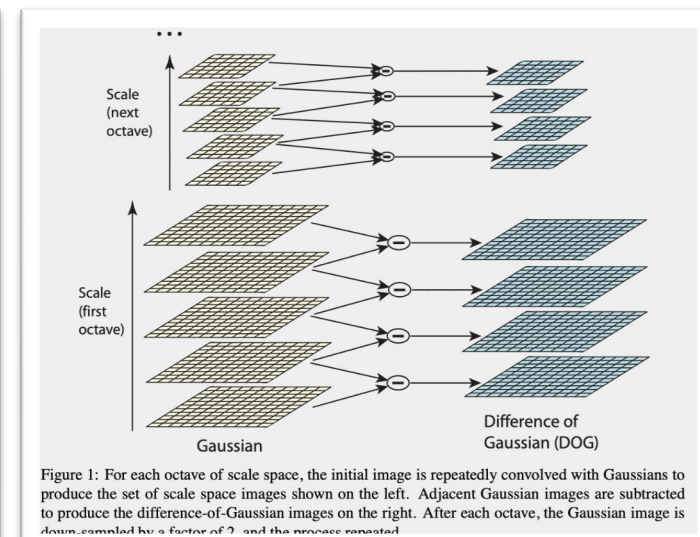
## Edge detection (Canny)



## Corner Detection (Harris)



## Scale Invariant Feature Transform (SIFT)



# Example: Image Classification

## CNN for Image Classification

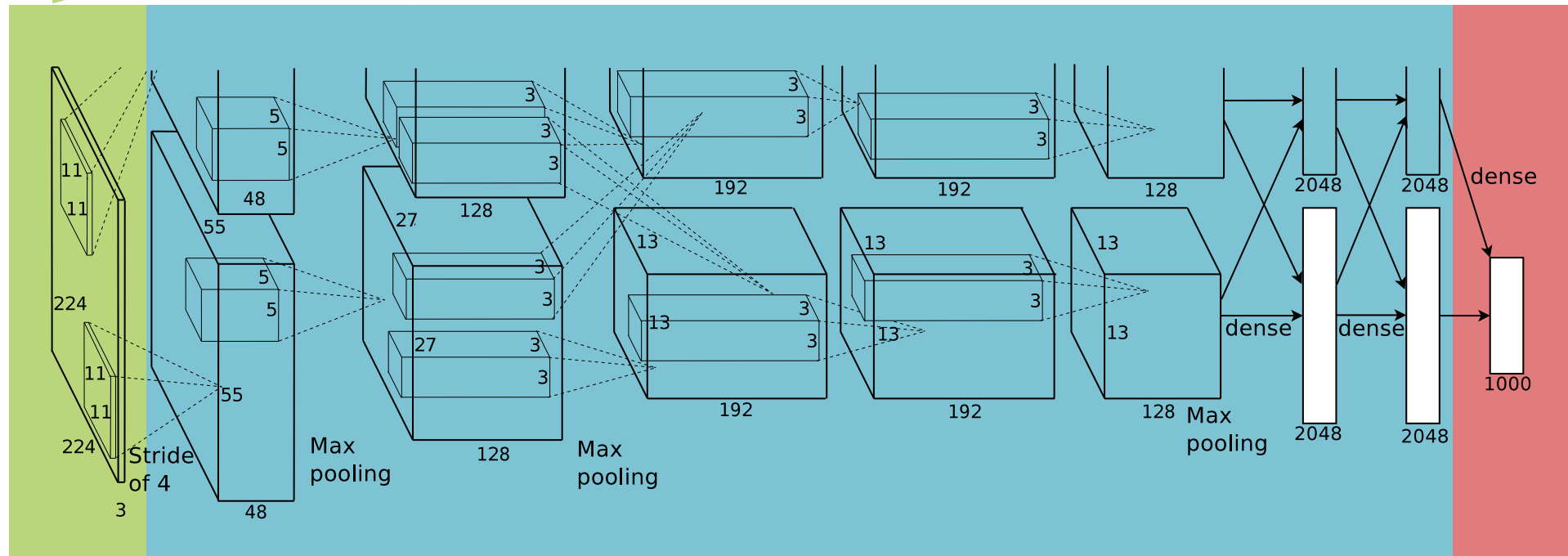
(Krizhevsky, Sutskever & Hinton, 2012)

15.3% error on ImageNet LSVRC-2012 contest

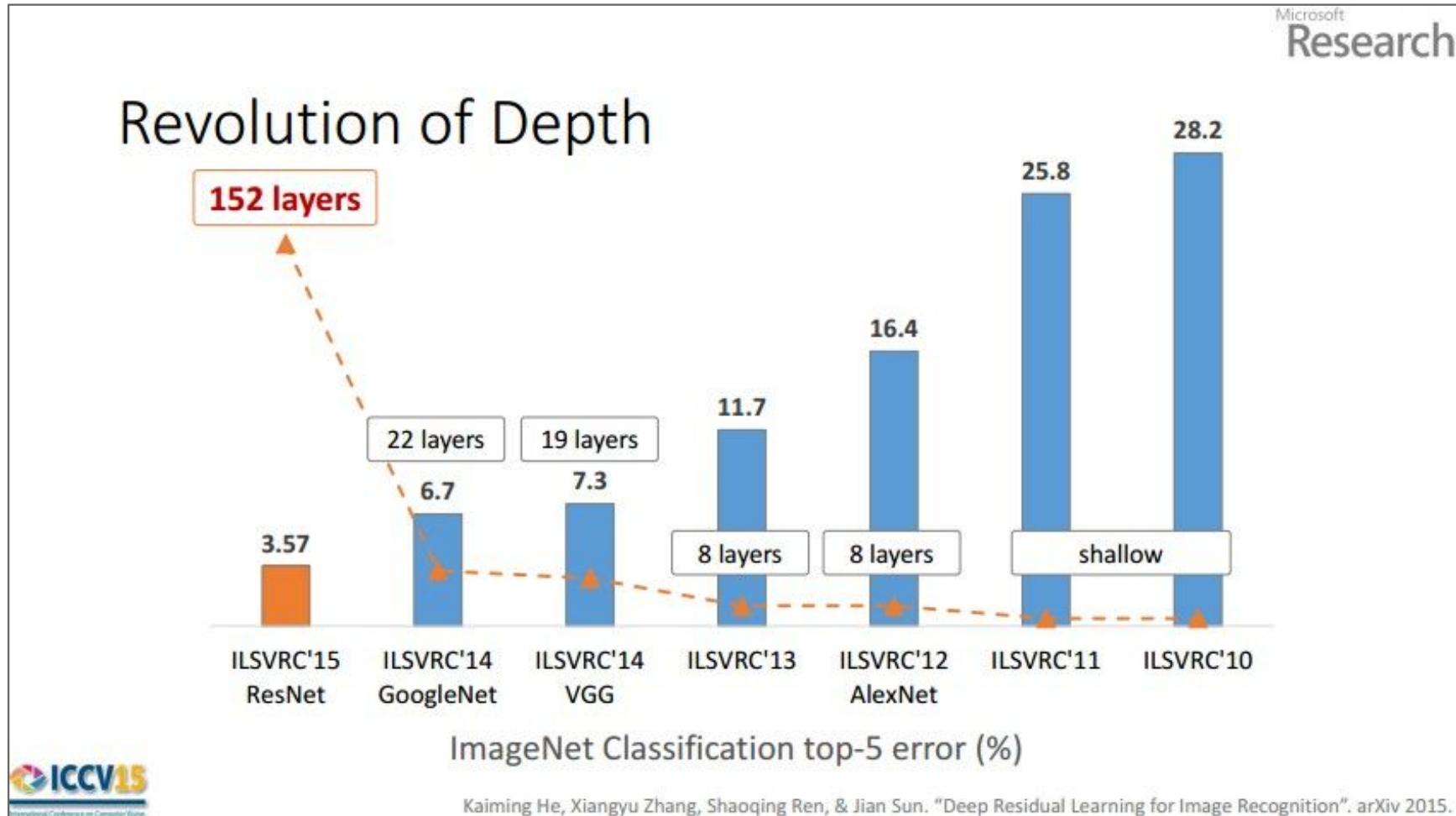
Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax



# CNNs for Image Recognition



# Backpropagation and Deep Learning

**Convolutional neural networks (CNNs)** and **recurrent neural networks (RNNs)** are simply fancy computation graphs (aka. hypotheses or decision functions).

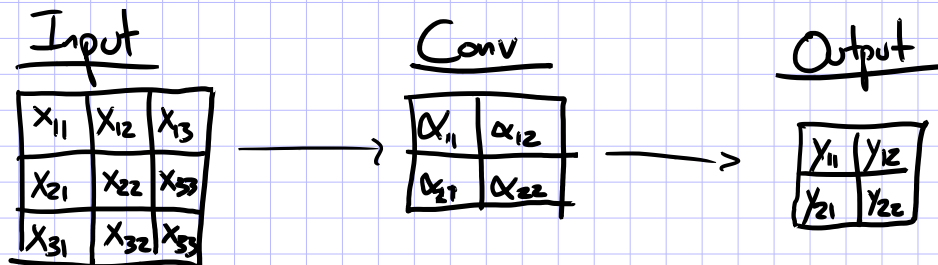
Our recipe also applies to these models and (again) relies on the **backpropagation algorithm** to compute the necessary gradients.

# CONVOLUTION

# What's a convolution?

- Basic idea:
  - Pick a 3x3 matrix F of weights
  - Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation
- Key point:
  - Different convolutions extract different types of low-level “features” from an image
  - All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel



$$\begin{aligned}y_{11} &= \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_0 \\y_{12} &= \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_0 \\y_{21} &= \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_0 \\y_{22} &= \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0\end{aligned}$$

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0



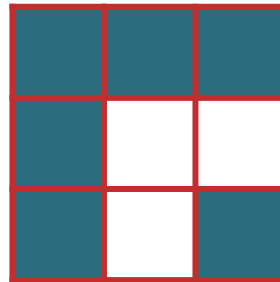
# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

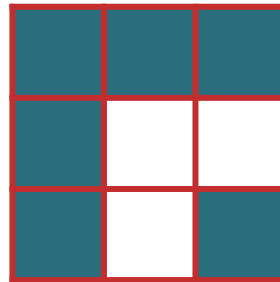
# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

			0	0	0	0
	1	1	1	1	1	0
	1		0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution


Convolved Image

3				

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	0	1	1	1	1	0
0	0	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	1
0	1	0

Convolved Image

3	2	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0				0	0
0	1		1	1	1	0
0	1		0		0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution


Convolved Image

3	2	2		

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	1	1	1	0
0	1	1	1	1	1	0
0	1	0	1	1	1	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

1	1	1
1	0	0
1	0	1

Convolved Image

3	2	2	3	
	1	1		1
	1		1	1
		1	1	1
	1	1	1	1

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0			
0	1	1	1		1	0
0	1	0	0		0	
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution


Convolved Image

3	2	2	3	1

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	1	0
0	1	0

Convolved Image

3	2	2	3	1
2	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0



# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	0	1	0	0
0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution

0	0	0
0	0	0
0	0	0

Convolved Image

3	2	2	3	1
2	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

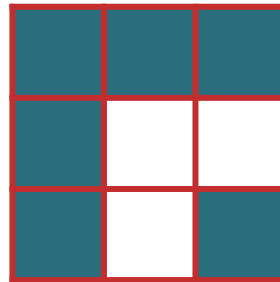
# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Convolution



Convolved Image

3	2	2	3	1
2	0	2	1	0
2	2	1	0	0
3	1	0	0	0
1	0	0	0	0

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Identity  
Convolution

0	0	0
0	1	0
0	0	0

Convolved Image

1	1	1	1	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	0	0	0	0

# Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

Blurring  
Convolution

.1	.1	.1
.1	.2	.1
.1	.1	.1

Convolved Image

.4	.5	.5	.5	.4
.4	.2	.3	.6	.3
.5	.4	.4	.2	.1
.5	.6	.2	.1	0
.4	.3	.1	0	0

# Convolution Examples

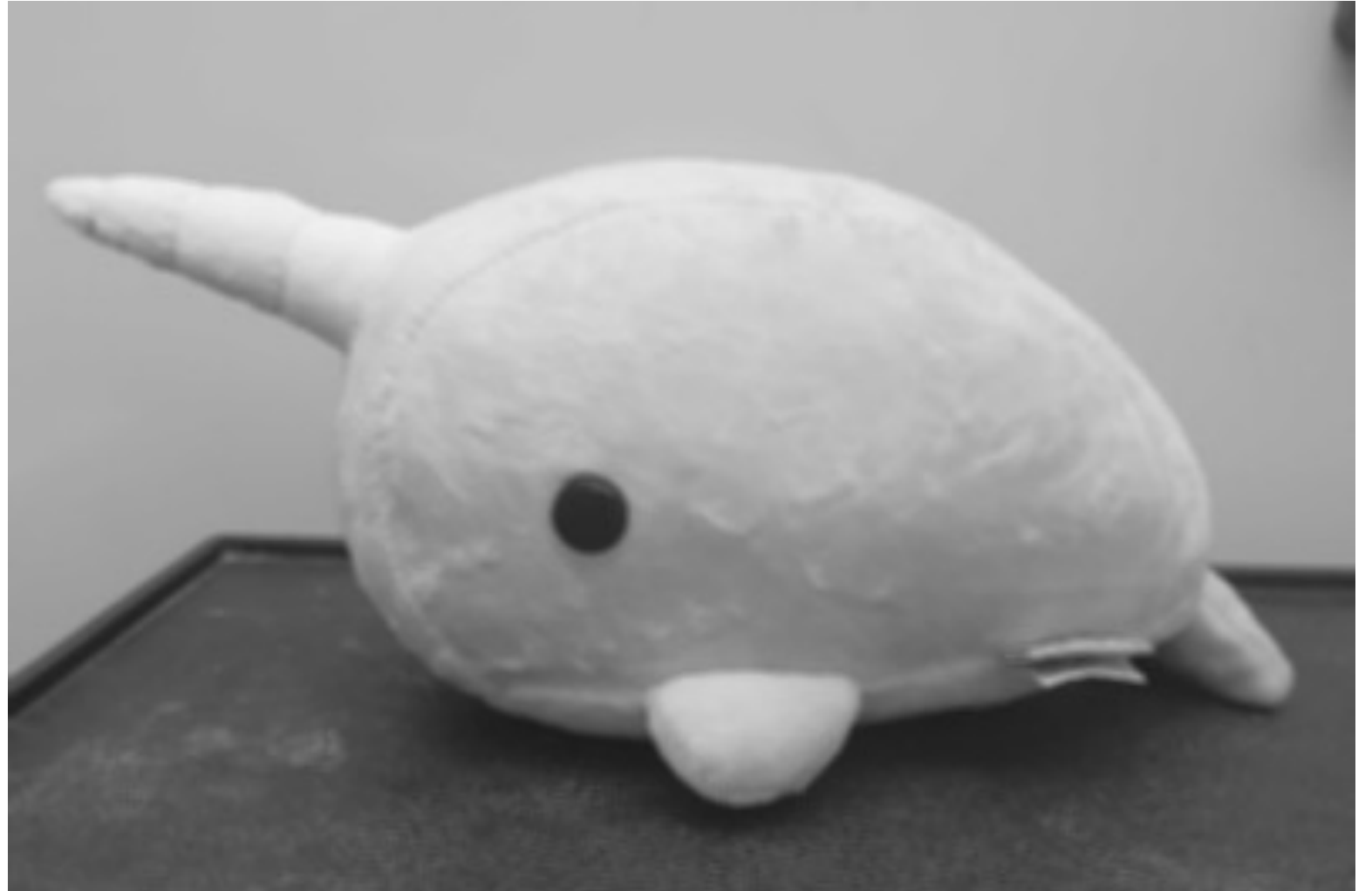
Original  
Image



# Convolution Examples

Smoothing  
Convolution

$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$



# Convolution Examples

Gaussian  
Blur

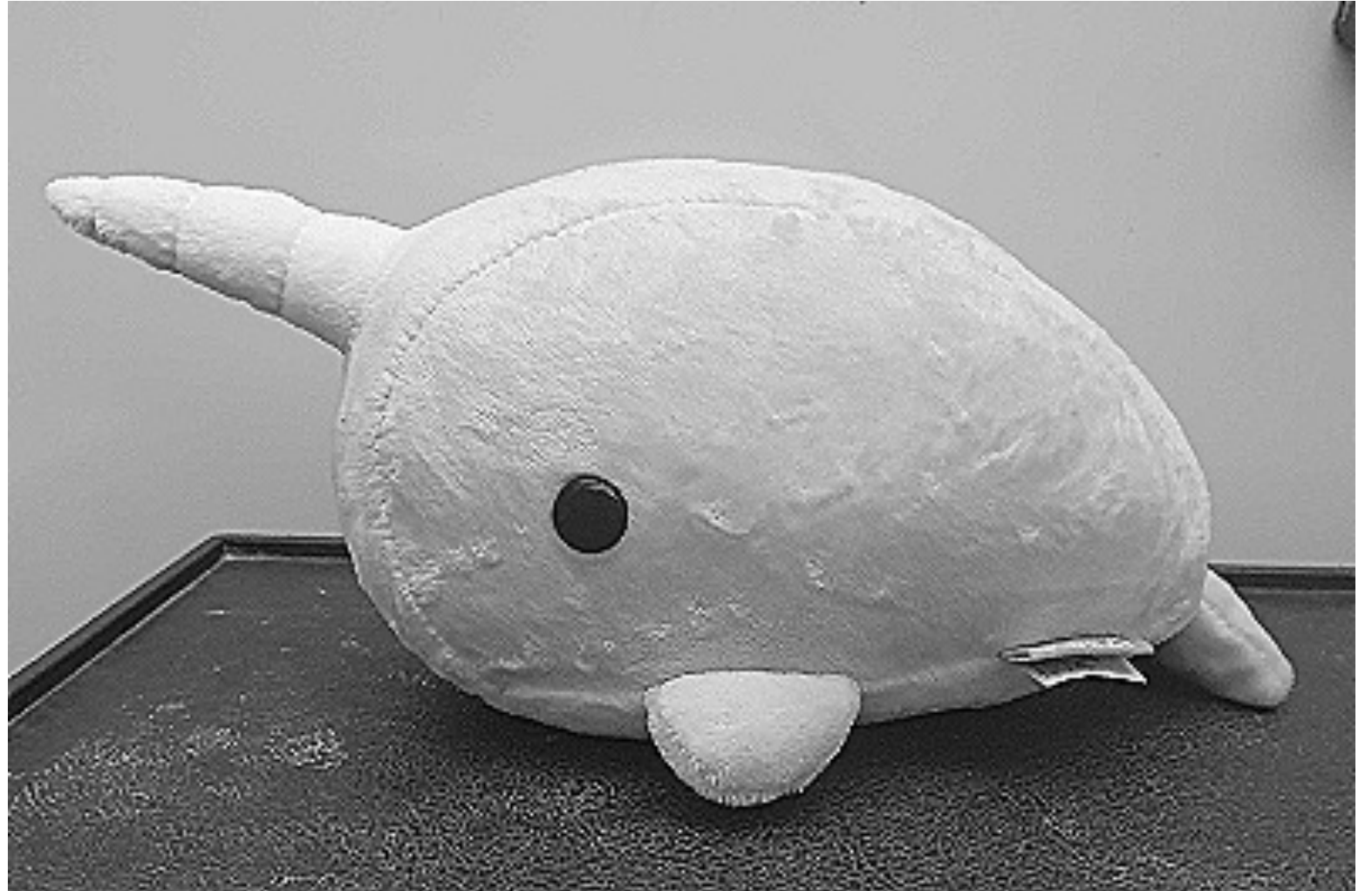
.01	.04	.06	.04	.01
.04	.19	.25	.19	.04
.06	.25	.37	.25	.06
.04	.19	.25	.19	.04
.01	.04	.06	.04	.01



# Convolution Examples

Sharpening  
Kernel

0	-1	0
-1	5	-1
0	-1	0

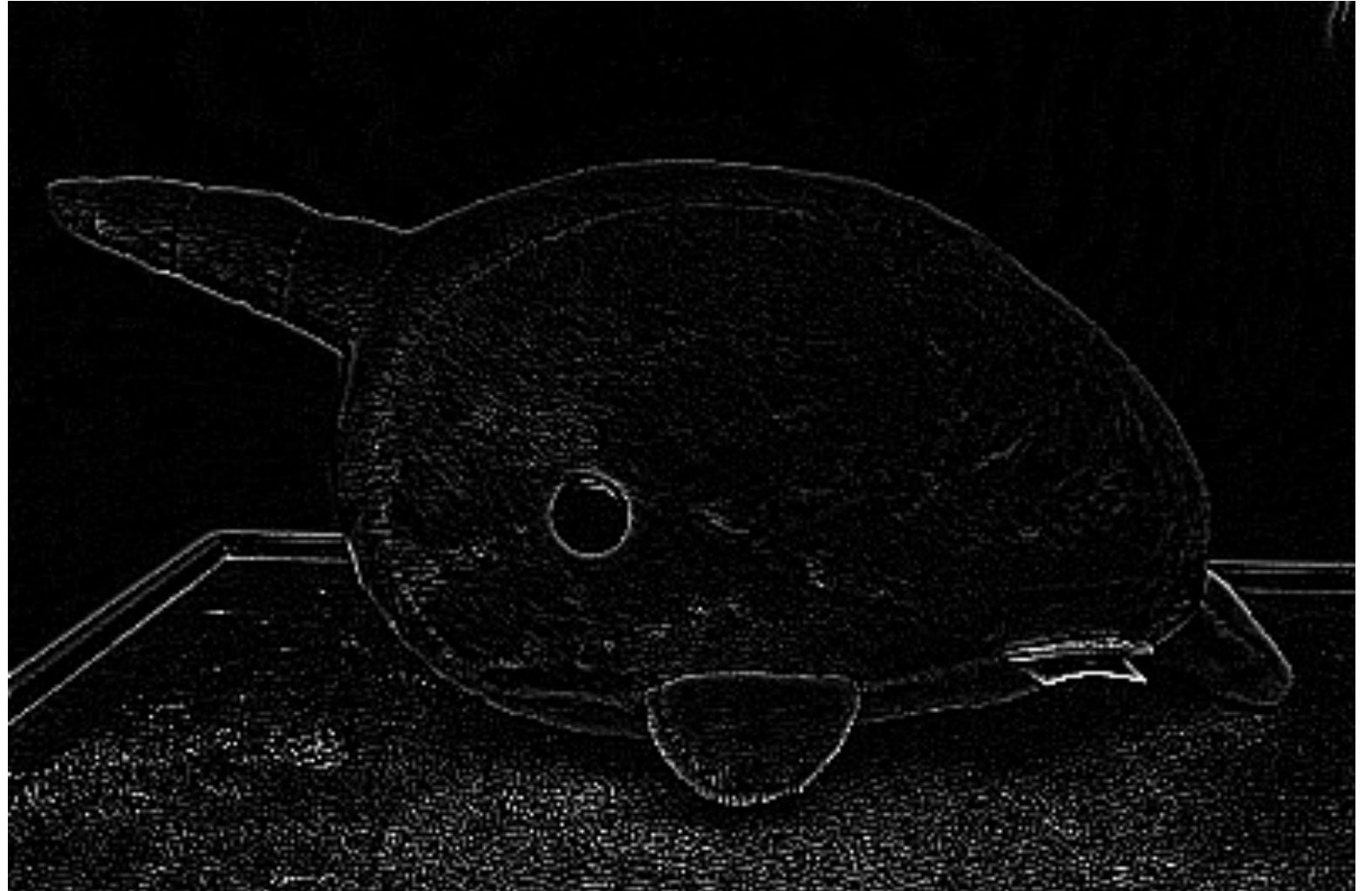




# Convolution Examples

Edge  
Detector

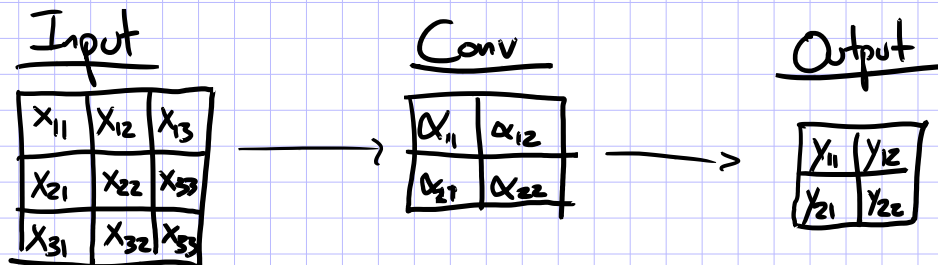
-1	-1	-1
-1	8	-1
-1	-1	-1



# What's a convolution?

- Basic idea:
  - Pick a 3x3 matrix F of weights
  - Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation
- Key point:
  - Different convolutions extract different types of low-level “features” from an image
  - All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel



$$\begin{aligned}y_{11} &= \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_0 \\y_{12} &= \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_0 \\y_{21} &= \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_0 \\y_{22} &= \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0\end{aligned}$$

# **DOWNSAMPLING**

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image


# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3		

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3		



# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	0

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	0
1		

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	0
1	0	0

# Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

1	1
1	1

Convolved Image

3	3	1
3	1	0
1	0	0

# Downsampling by Averaging

- Downsampling by averaging is a special case of convolution where the weights are fixed to a uniform distribution
- The example below uses a stride of 2

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Convolution

$1/4$	$1/4$
$1/4$	$1/4$

Convolved Image

$3/4$	$3/4$	$1/4$
$3/4$	$1/4$	0
$1/4$	0	0

# Max-Pooling

- Max-pooling with a stride  $> 1$  is another form of downsampling
- Instead of averaging, we take the max value within the same range as the equivalently-sized convolution
- The example below uses a stride of 2

Input Image

1	1	1	1	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	0

Max-pooling

$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j}$	$x_{i+1,j+1}$

$$y_{ij} = \max(x_{ij}, x_{i,j+1}, x_{i+1,j}, x_{i+1,j+1})$$

Max-Pooled Image

1	1	1
1	1	0
1	0	0

# CONVOLUTIONAL NEURAL NETS



# A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

# A Recipe for Machine Learning

1. • Convolutional Neural Networks (CNNs) provide another form of **decision function**  
• Let's see what they look like...

2. CHOOSE EACH OF THESE.

– Decision function

$$\hat{y} = f_{\theta}(x_i)$$

– Loss function

$$\ell(\hat{y}, y_i) \in \mathbb{R}$$

4. Train with SGD:

(Take small steps opposite the gradient)

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i)$$



# Convolutional Layer

**CNN key idea:**  
Treat convolution matrix as parameters and learn them!

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0



Learned Convolution

$\theta_{11}$	$\theta_{12}$	$\theta_{13}$
$\theta_{21}$	$\theta_{22}$	$\theta_{23}$
$\theta_{31}$	$\theta_{32}$	$\theta_{33}$

Convolved Image

.4	.5	.5	.5	.4
.4	.2	.3	.6	.3
.5	.4	.4	.2	.1
.5	.6	.2	.1	0
.4	.3	.1	0	0

# Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

## Architecture #1: LeNet-5

PROC. OF THE IEEE, NOVEMBER 1998

7

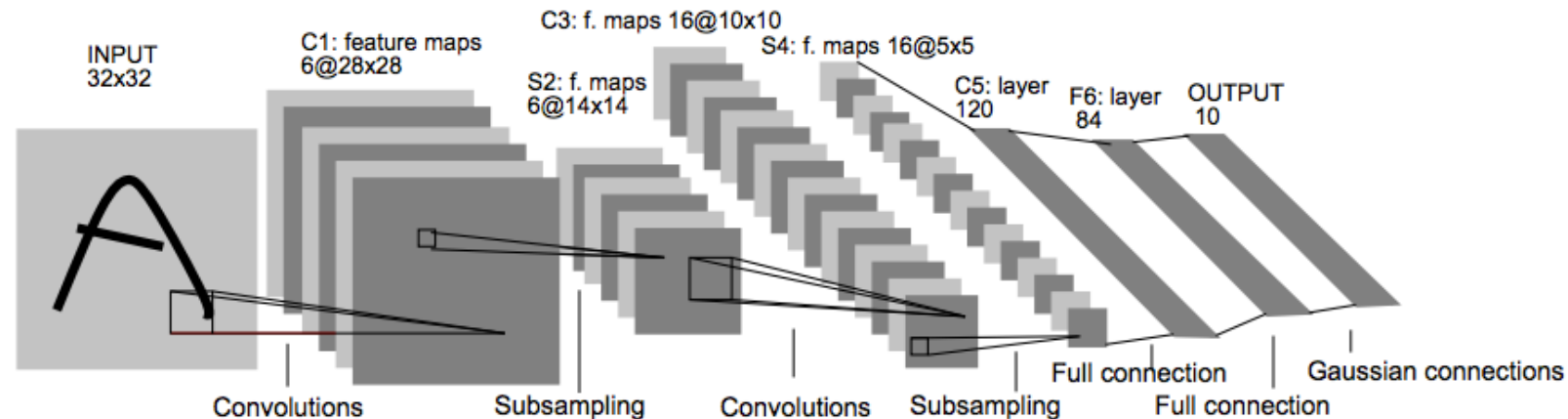


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# TRAINING CNNs

# A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

# Background

# A Recipe for Machine Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of the

– Decision function

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}_i)$$

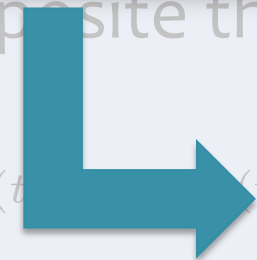
– Loss function

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

- Q: Now that we have the CNN as a decision function, how do we compute the gradient?
- A: Backpropagation of course!

opposite the gradient)


$$\theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i)$$

# SGD for CNNs

## SGD for CNNs

Ex: Architecture: Given  $\vec{x}, y^*$

$$\begin{aligned} J &= \ell(y, y^*) \\ y &= \text{softmax}(z^{(5)}) \\ z^{(5)} &= \text{linear}(z^{(4)}, W) \\ z^{(4)} &= \text{relu}(z^{(3)}) \\ z^{(3)} &= \text{conv}(z^{(2)}, \beta) \\ z^{(2)} &= \text{max-pool}(z^{(1)}) \\ z^{(1)} &= \text{conv}(\vec{x}, \alpha) \end{aligned}$$

Parameters  $\vec{\theta} = [\alpha, \beta, W]$

SGD:

① Init  $\vec{\theta}$

② While not converged:

Sample  $i \in \{1, \dots, N\}$

Forward:  $y = h_{\theta}(\vec{x}^{(i)})$ ,  $J_i(\theta) = \ell(y, y^*)$

Backward:  $\nabla_{\vec{\theta}} J_i(\theta) = \dots$

Update:  $\vec{\theta} \leftarrow \vec{\theta} - \lambda \nabla_{\vec{\theta}} J_i(\theta)$



# **LAYERS OF A CNN**

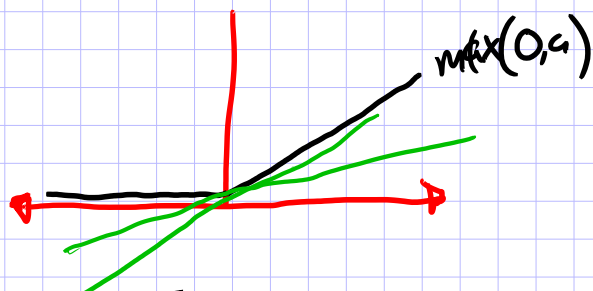
# ReLU Layer

ReLU Layer Input:  $\vec{x} \in \mathbb{R}^k$  Output:  $\vec{y} \in \mathbb{R}^k$

Forward:

$$\vec{y} = \sigma(\vec{x}) \leftarrow \text{element-wise}$$

$$\sigma(a) = \max(0, a)$$



Backward:

$$\frac{dJ}{dx_i} = \frac{dJ}{dy_i} \frac{dy_i}{dx_i}$$

where  $\frac{dy_i}{dx_i} = \begin{cases} 1 & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases}$

subderivative

# Softmax Layer

Softmax Layer

Input:  $\vec{x} \in \mathbb{R}^k$  Output:  $\vec{y} \in \mathbb{R}^k$

Forward:

$$y_i = \frac{\exp(x_i)}{\sum_{k=1}^k \exp(x_k)}$$

Backward:

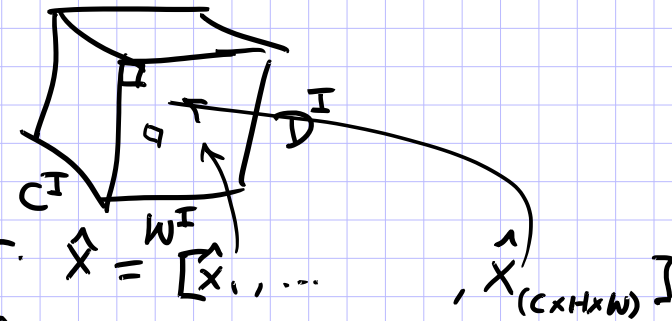
$$\frac{dJ}{dx_j} = \sum_{i=1}^k \frac{dJ}{dy_i} \frac{dy_i}{dx_j}$$

$$\text{where } \frac{dy_i}{dx_j} = \begin{cases} y_i(1-y_i) & \text{if } i=j \\ -y_i y_j & \text{otherwise} \end{cases}$$

# Fully-Connected Layer

## Fully Connected Layer (w/ tensor input)

- Suppose input is a 3D Tensor:  $X =$



- stretch out into a long vector.  $\hat{x} = [x_1, \dots$

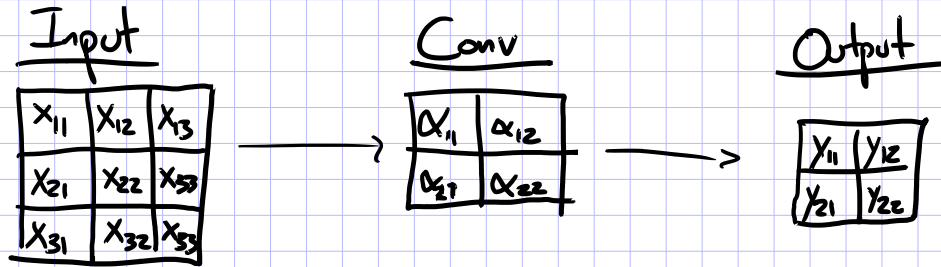
- then standard linear layer:

$$y = \alpha^T \hat{x} + \alpha_0 \quad \text{where } \alpha \in \mathbb{R}^{A \times B}$$

$|\hat{x}| = A, |y| = B$

# Convolutional Layer

Ex: 1 input channel, 1 output channel



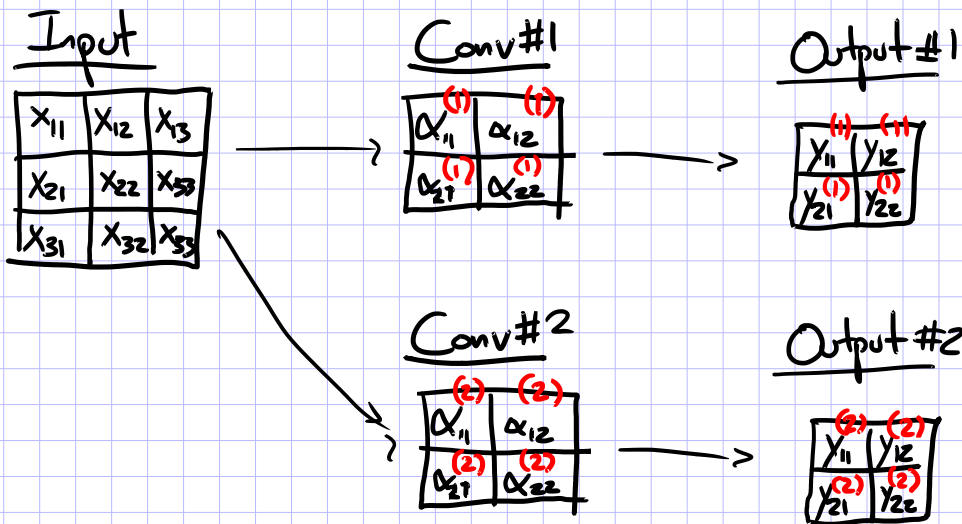
$$y_{11} = \alpha_{11}x_{11} + \alpha_{12}x_{12} + \alpha_{21}x_{21} + \alpha_{22}x_{22} + \alpha_0$$

$$y_{12} = \alpha_{11}x_{12} + \alpha_{12}x_{13} + \alpha_{21}x_{22} + \alpha_{22}x_{23} + \alpha_0$$

$$y_{21} = \alpha_{11}x_{21} + \alpha_{12}x_{22} + \alpha_{21}x_{31} + \alpha_{22}x_{32} + \alpha_0$$

$$y_{22} = \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0$$

Ex: 1 input channel, 2 output channels



$$y_{11}^{(1)} = \alpha_{11}^{(1)}x_{11} + \alpha_{12}^{(1)}x_{12} + \alpha_{21}^{(1)}x_{21} + \alpha_{22}^{(1)}x_{22} + \alpha_0^{(1)}$$

$$y_{12}^{(1)} = \dots$$

$$y_{21}^{(1)} = \dots$$

$$y_{22}^{(1)} = \alpha_{11}^{(1)}x_{22} + \alpha_{12}^{(1)}x_{23} + \alpha_{21}^{(1)}x_{32} + \alpha_{22}^{(1)}x_{33} + \alpha_0^{(1)}$$

$$y_{11}^{(2)} = \alpha_{11}^{(2)}x_{11} + \alpha_{12}^{(2)}x_{12} + \alpha_{21}^{(2)}x_{21} + \alpha_{22}^{(2)}x_{22} + \alpha_0^{(2)}$$

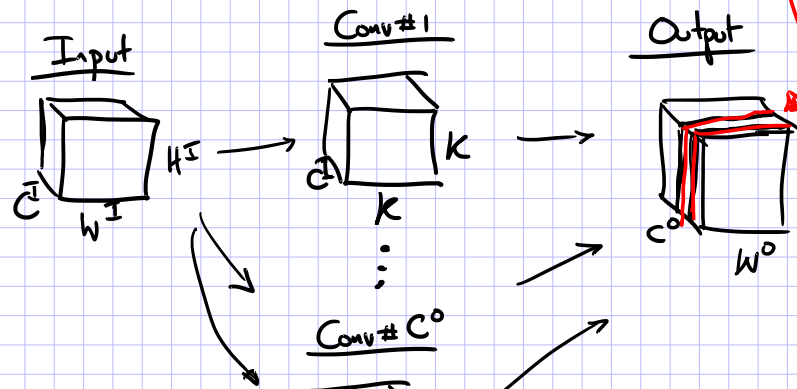
$$y_{12}^{(2)} = \dots$$

$$y_{21}^{(2)} = \dots$$

$$y_{22}^{(2)} = \alpha_{11}^{(2)}x_{22} + \alpha_{12}^{(2)}x_{23} + \alpha_{21}^{(2)}x_{32} + \alpha_{22}^{(2)}x_{33} + \alpha_0^{(2)}$$

# Convolutional Layer

Ex:  $C^I$  input channels,  $C^O$  output channels

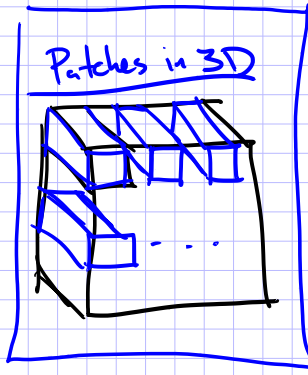


i-th slice is output from j-th convolution matrix

$$H^O = \lfloor (H^I + 2p - K) / s + 1 \rfloor$$

$$W^O = \lfloor (W^I + 2p - K) / s + 1 \rfloor$$

where  $p$  = # pixels of padding on input  
 $k$  = size of conv. matrix  
 $s$  = stride length



Forward:

$$y_{ij}^{(k)} = \alpha_0^{(k)} + \sum_{c=1}^{C^I} \sum_{q=1}^K \sum_{r=1}^K \alpha_{qr}^{(k)} x_{mn}^{(c)} \quad \text{where } m = s(i-1) + q, n = s(j-1) + r$$

Backward:

$$\frac{dJ}{d\alpha_0^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{d\alpha_0^{(k)}}$$

$$\frac{dJ}{d\alpha_{qr}^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{d\alpha_{qr}^{(k)}}$$

$$\frac{dJ}{dx_{mn}^{(c)}} = \sum_i \sum_j \sum_k \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{dx_{mn}^{(c)}}$$

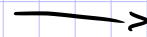
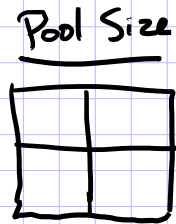
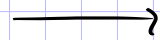
just some calculus

# Max-Pooling Layer

Ex: 1 input channel, 1 output channel, stride of 1

Input

$x_{11}$	$x_{12}$	$x_{13}$
$x_{21}$	$x_{22}$	$x_{23}$
$x_{31}$	$x_{32}$	$x_{33}$

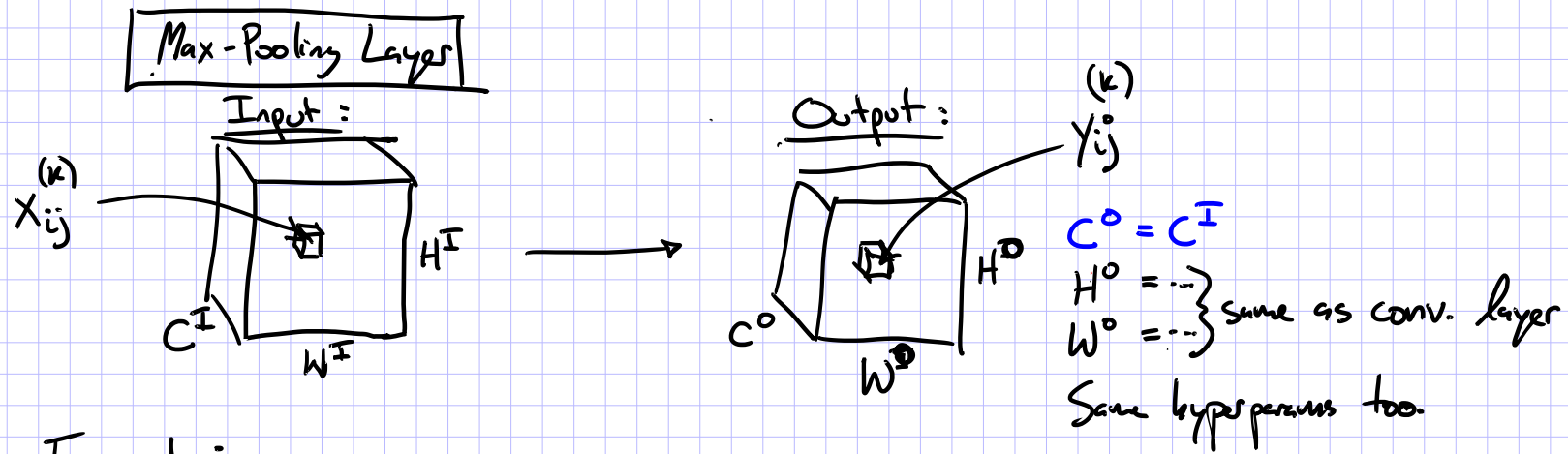


Output

$y_{11}$	$y_{12}$
$y_{21}$	$y_{22}$

$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$
$$y_{12} = \max(x_{12}, x_{13}, x_{22}, x_{23})$$
$$y_{21} = \max(x_{21}, x_{22}, x_{31}, x_{32})$$
$$y_{22} = \max(x_{22}, x_{23}, x_{32}, x_{33})$$

# Max-Pooling Layer



Forward:

$$Y_{ij}^{(k)} = \max_{\substack{q \in \{1, \dots, k\} \\ r \in \{1, \dots, k\}}} X_{mn}^{(k)} \text{ where } \begin{cases} m = s(i-1) + q \\ n = s(j-1) + r \end{cases}$$

Backward:

$$\frac{dJ}{dx_{mn}^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{dx_{mn}^{(k)}}$$

## Subderivatives

- +  $\max()$  is not differentiable, but subdifferentiable.
- + There are a set of derivatives and we can just choose one for SGD.

$$y = \max(a, b)$$

$$\Rightarrow \frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da} \text{ where } \frac{dy}{da} = \begin{cases} 1 & \text{if } a > b \\ 0 & \text{otherwise} \end{cases}$$



# Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

## Architecture #1: LeNet-5

PROC. OF THE IEEE, NOVEMBER 1998

7

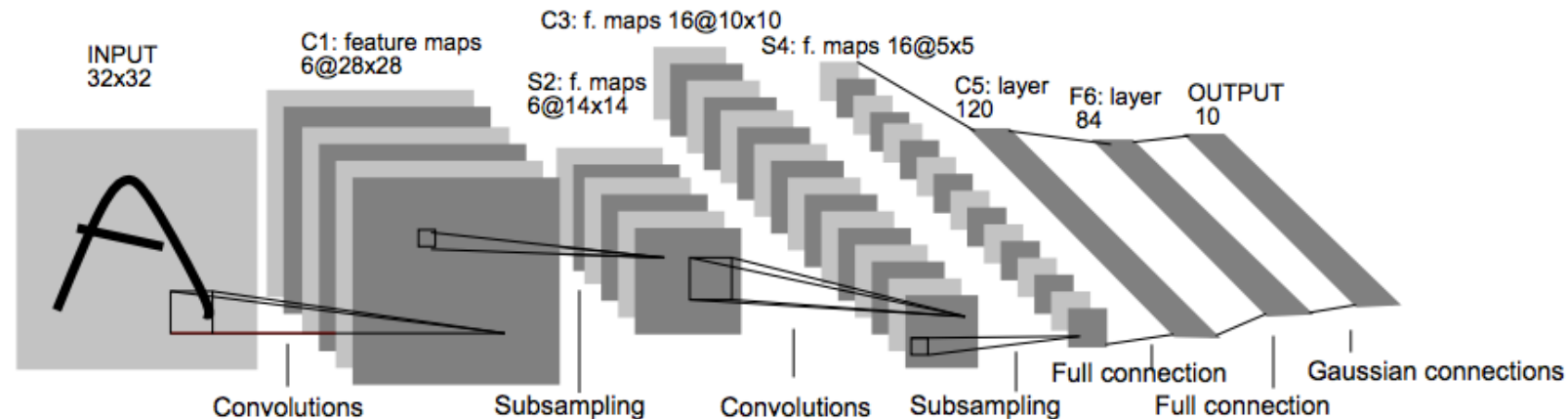


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

# Architecture #2: AlexNet

## CNN for Image Classification

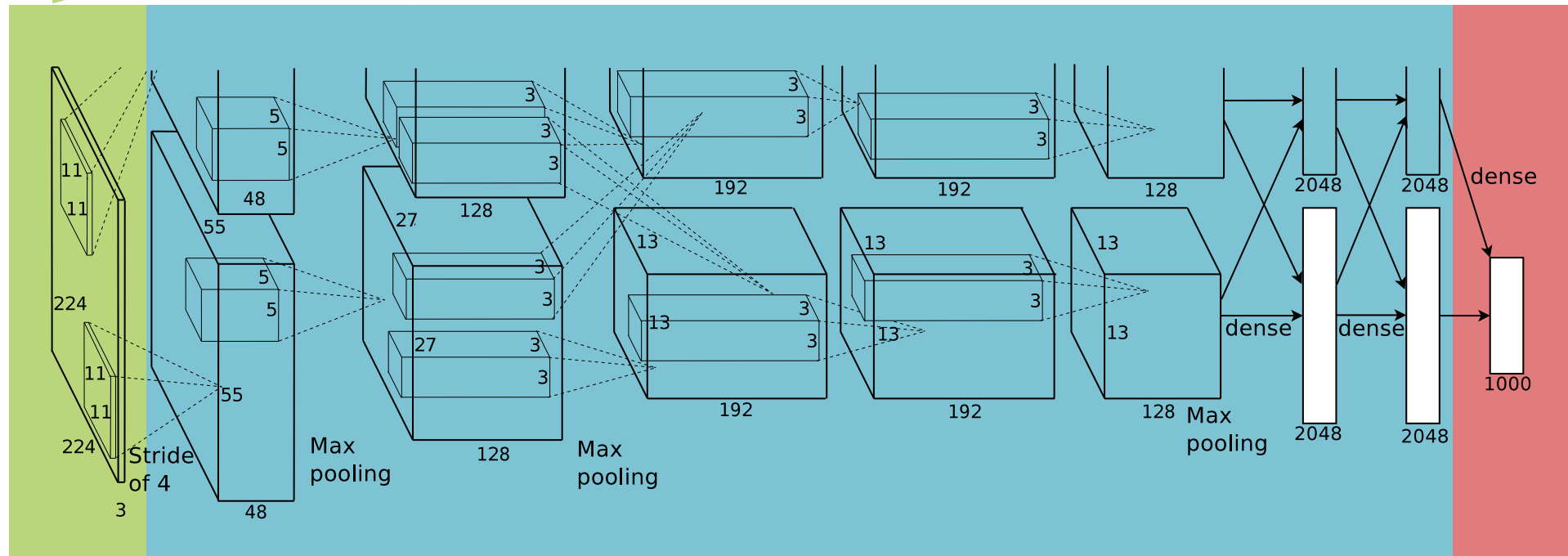
(Krizhevsky, Sutskever & Hinton, 2012)

15.3% error on ImageNet LSVRC-2012 contest

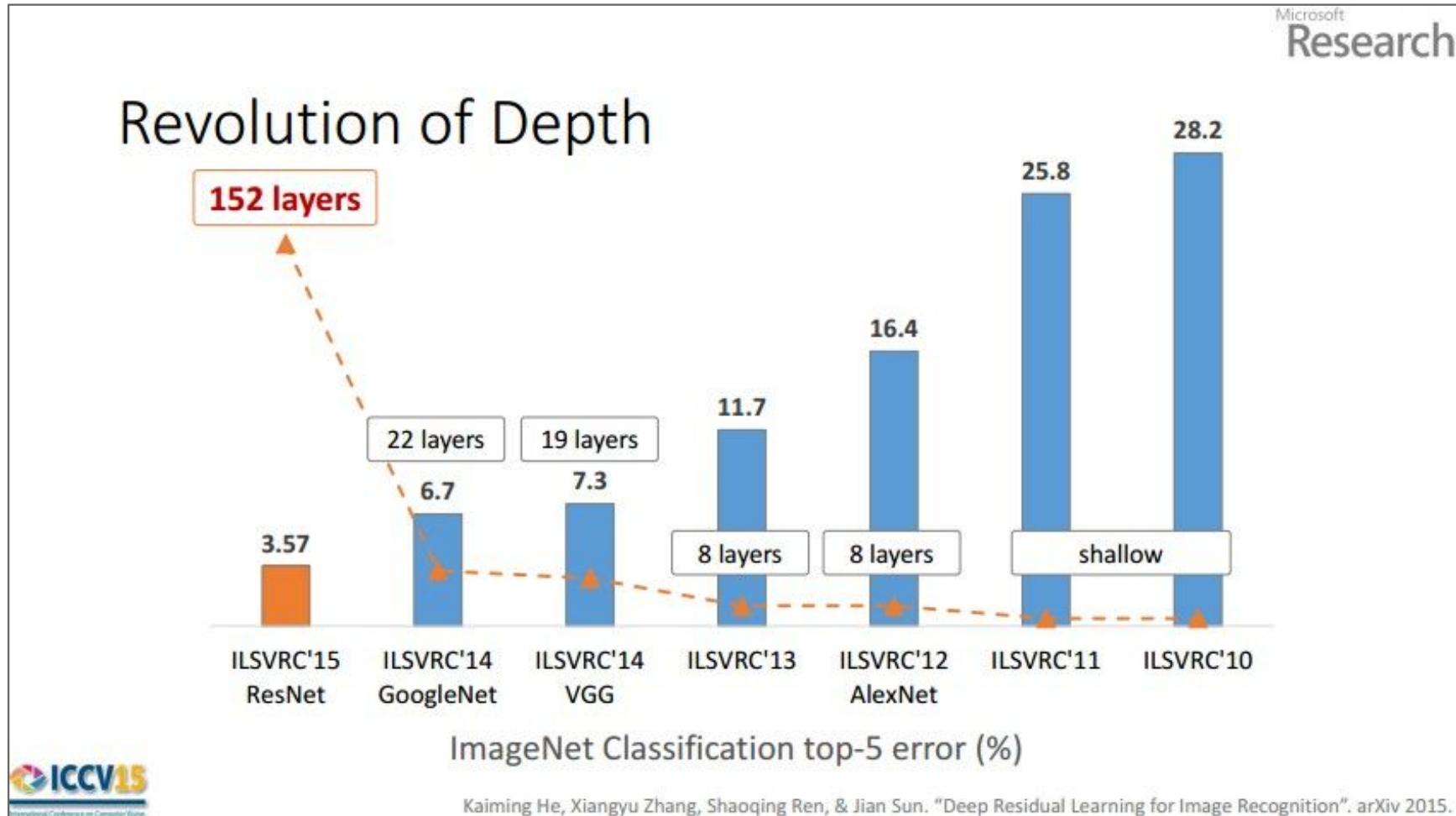
Input image (pixels)

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

1000-way softmax

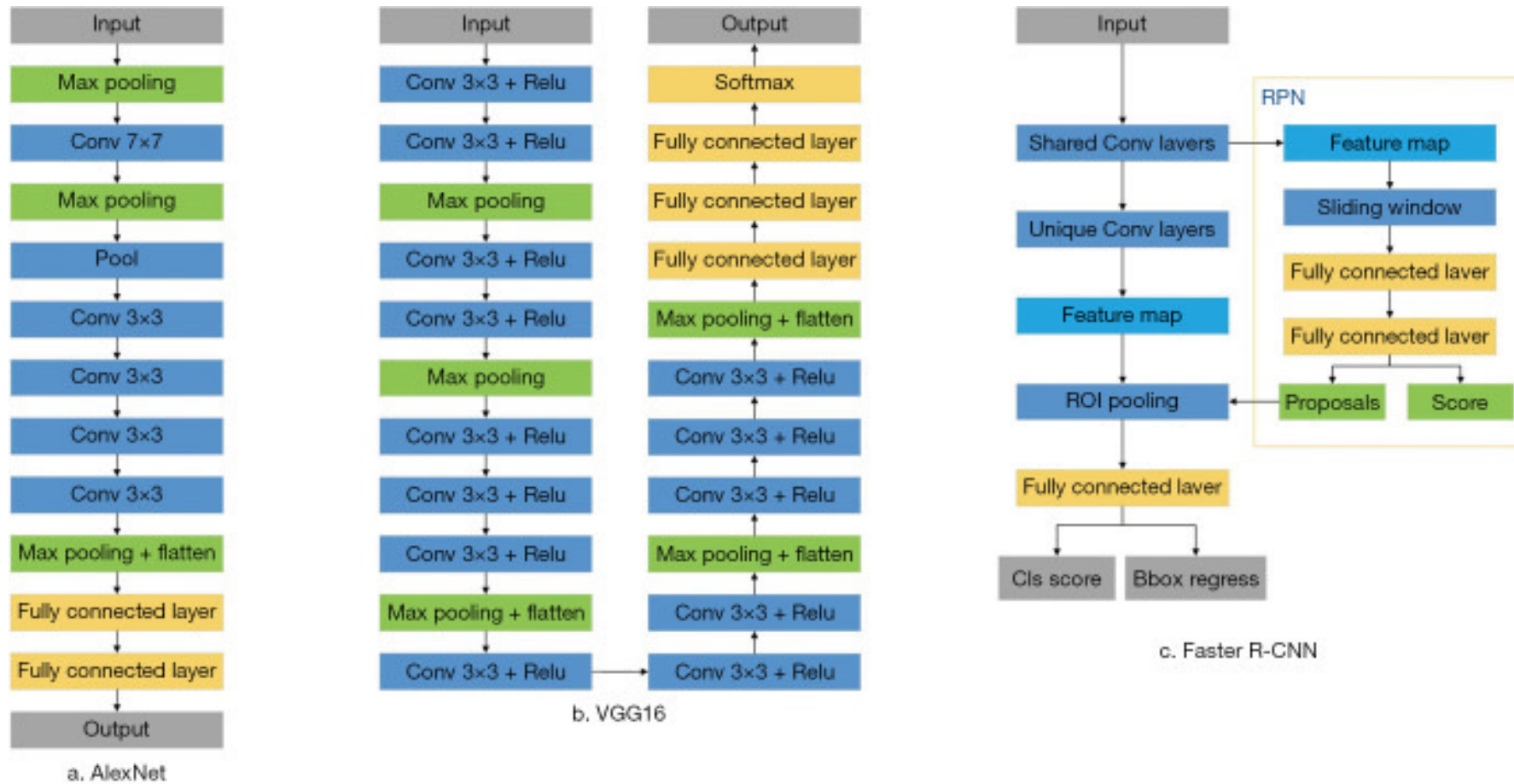


# CNNs for Image Recognition



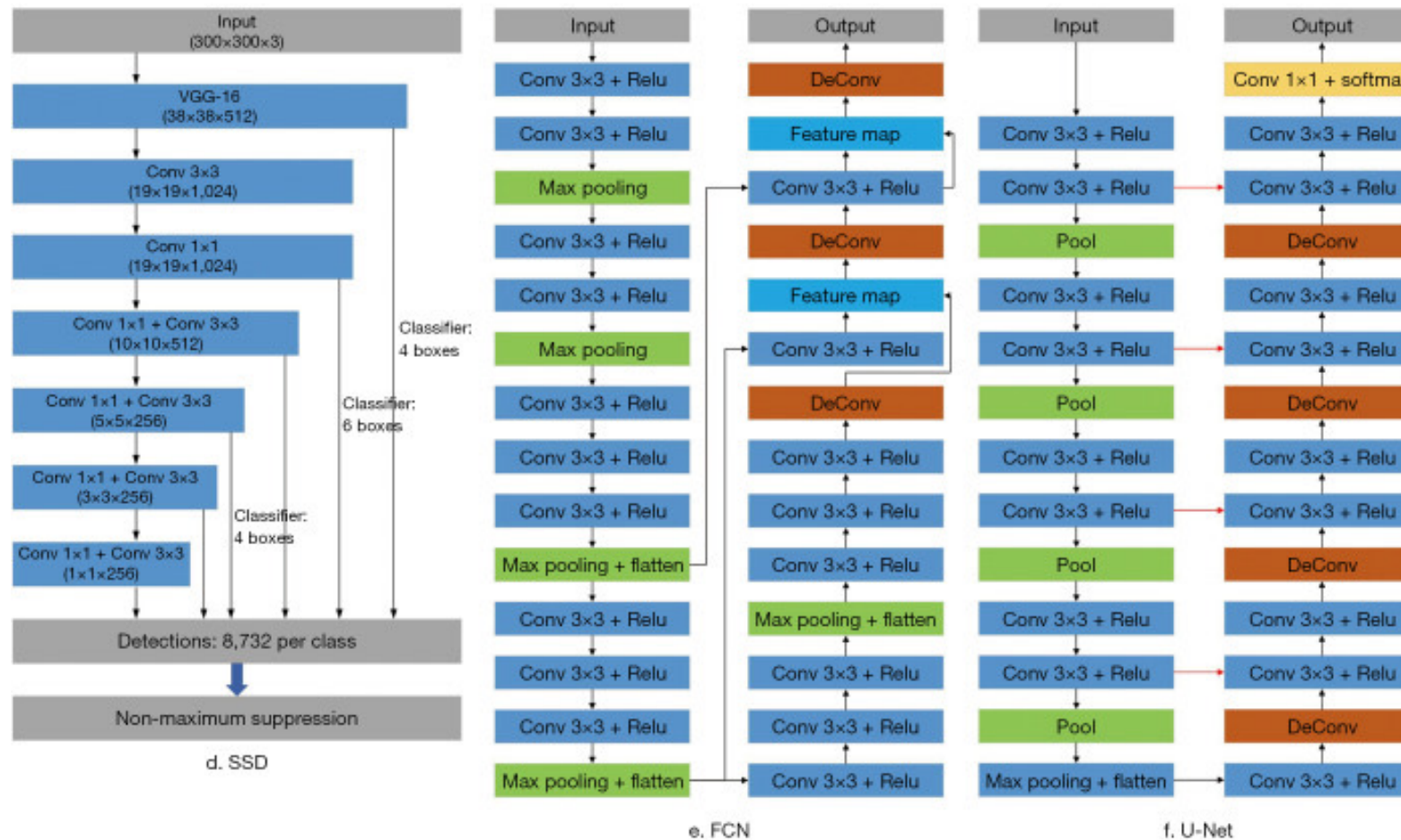
# Convolutional Neural Network (CNN)

## Typical Architectures



# Convolutional Neural Network (CNN)

## Typical Architectures



# Convolutional Neural Network (CNN)

## Typical Architectures

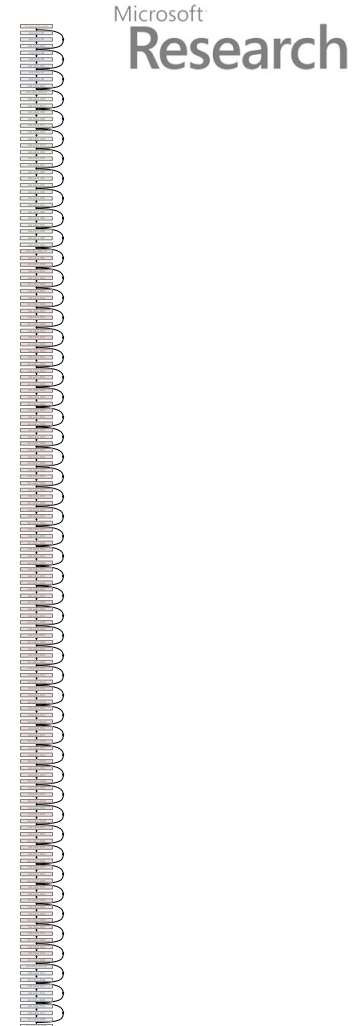
AlexNet, 8 layers  
(ILSVRC 2012)



VGG, 19 layers  
(ILSVRC 2014)



ResNet, 152 layers  
(ILSVRC 2015)



# In-Class Poll

## Question:

Why do many layers used in computer vision *not have* location specific parameters?

## Answer:

# Convolutional Layer

For a convolutional layer, how do we pick the kernel size (aka. the size of the convolution)?

Input Image

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

2x2  
Convolution

$\theta_{11}$	$\theta_{12}$
$\theta_{21}$	$\theta_{22}$

3x3  
Convolution

$\theta_{11}$	$\theta_{12}$	$\theta_{13}$
$\theta_{21}$	$\theta_{22}$	$\theta_{23}$
$\theta_{31}$	$\theta_{32}$	$\theta_{33}$

4x4  
Convolution

$\theta_{11}$	$\theta_{12}$	$\theta_{13}$	$\theta_{14}$
$\theta_{21}$	$\theta_{22}$	$\theta_{23}$	$\theta_{24}$
$\theta_{31}$	$\theta_{32}$	$\theta_{33}$	$\theta_{34}$
$\theta_{41}$	$\theta_{42}$	$\theta_{43}$	$\theta_{44}$

- A small kernel can only see a very small part of the image, but is fast to compute
- A large kernel can see more of the image, but at the expense of speed



# **CNN VISUALIZATIONS**

# Visualization of CNN

[https://adamharley.com/nn\\_vis/cnn/2d.html](https://adamharley.com/nn_vis/cnn/2d.html)

60 FPS (1-61) Draw your number here 0123456789

Layer visibility

Input layer	Hide
Convolution layer 1	Hide
Downsampling layer 1	Hide
Convolution layer 2	Hide
Downsampling layer 2	Hide
Fully-connected layer 1	Hide
Fully-connected layer 2	Hide
Output layer	Hide

Downsampled drawing: 4  
First guess: 7  
Second guess: 8

The visualization shows the digit '4' being processed through various layers of a CNN. The input layer shows the original drawing. Subsequent layers (Convolution 1, Downsampling 1, Convolution 2, Downsampling 2, Fully-connected 1, Fully-connected 2) show the digit being broken down into its constituent features and then reassembled. The output layer shows the final classification result, which is '4'.

# Convolution of a Color Image

- Color images consist of 3 floats per pixel for RGB (red, green blue) color values
- Convolution must also be 3-dimensional

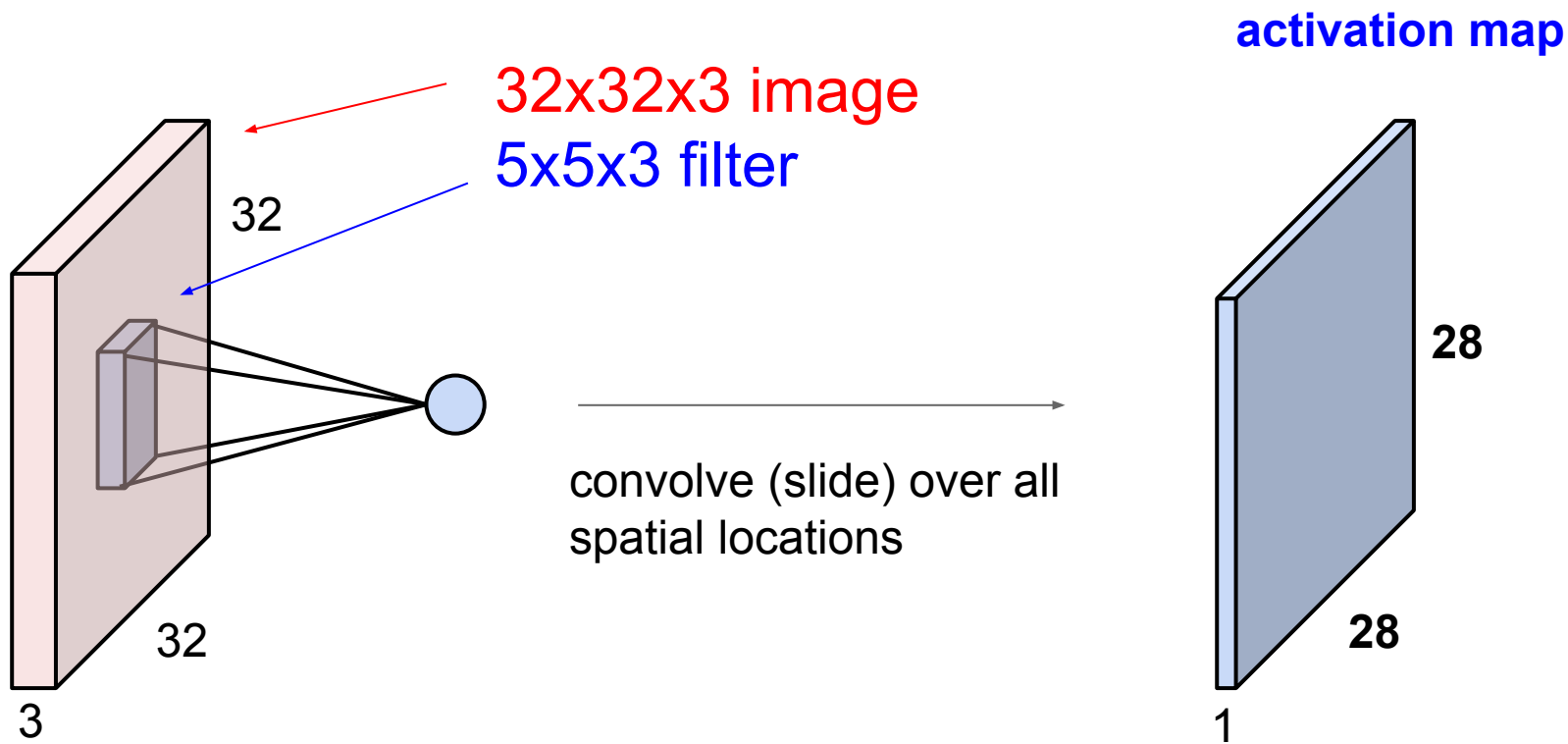


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

# Animation of 3D Convolution

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

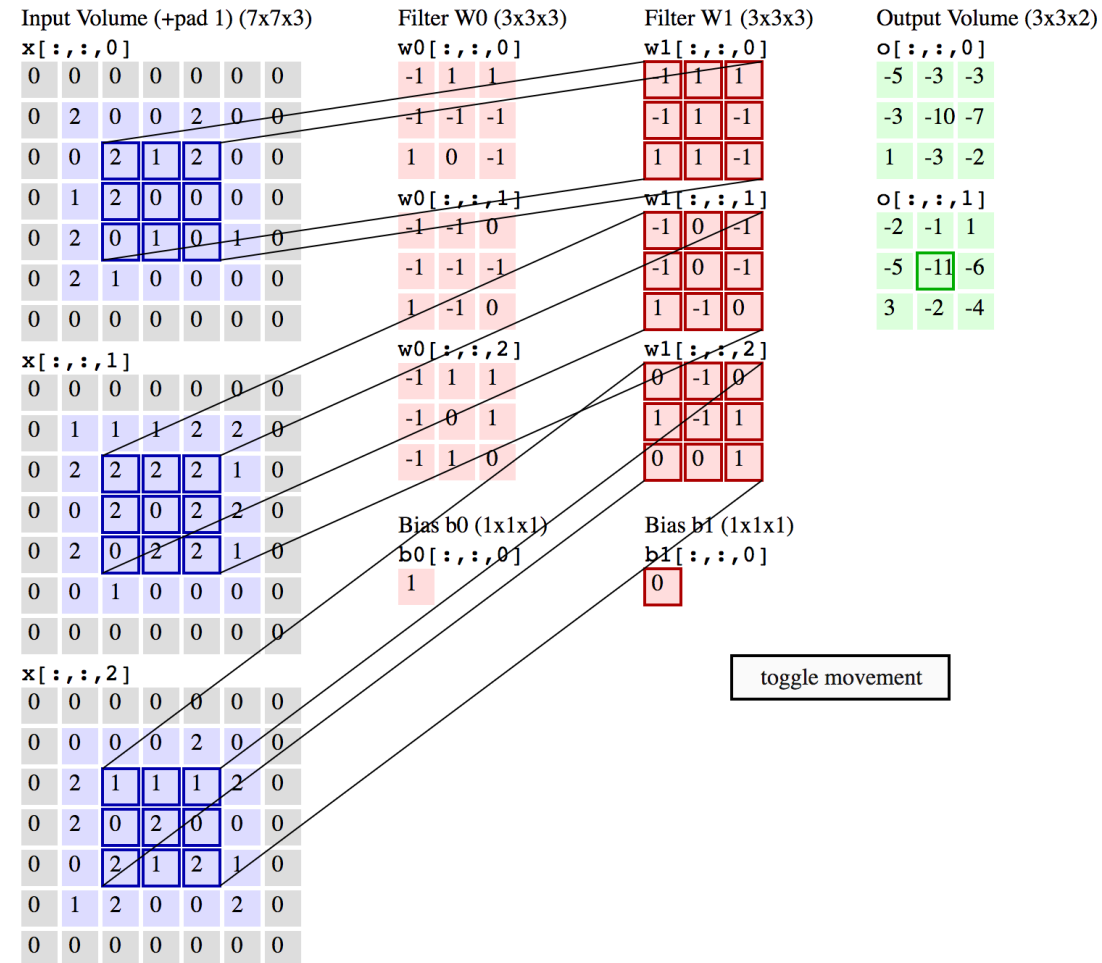


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

# MNIST Digit Recognition with CNNs (in your browser)

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html>

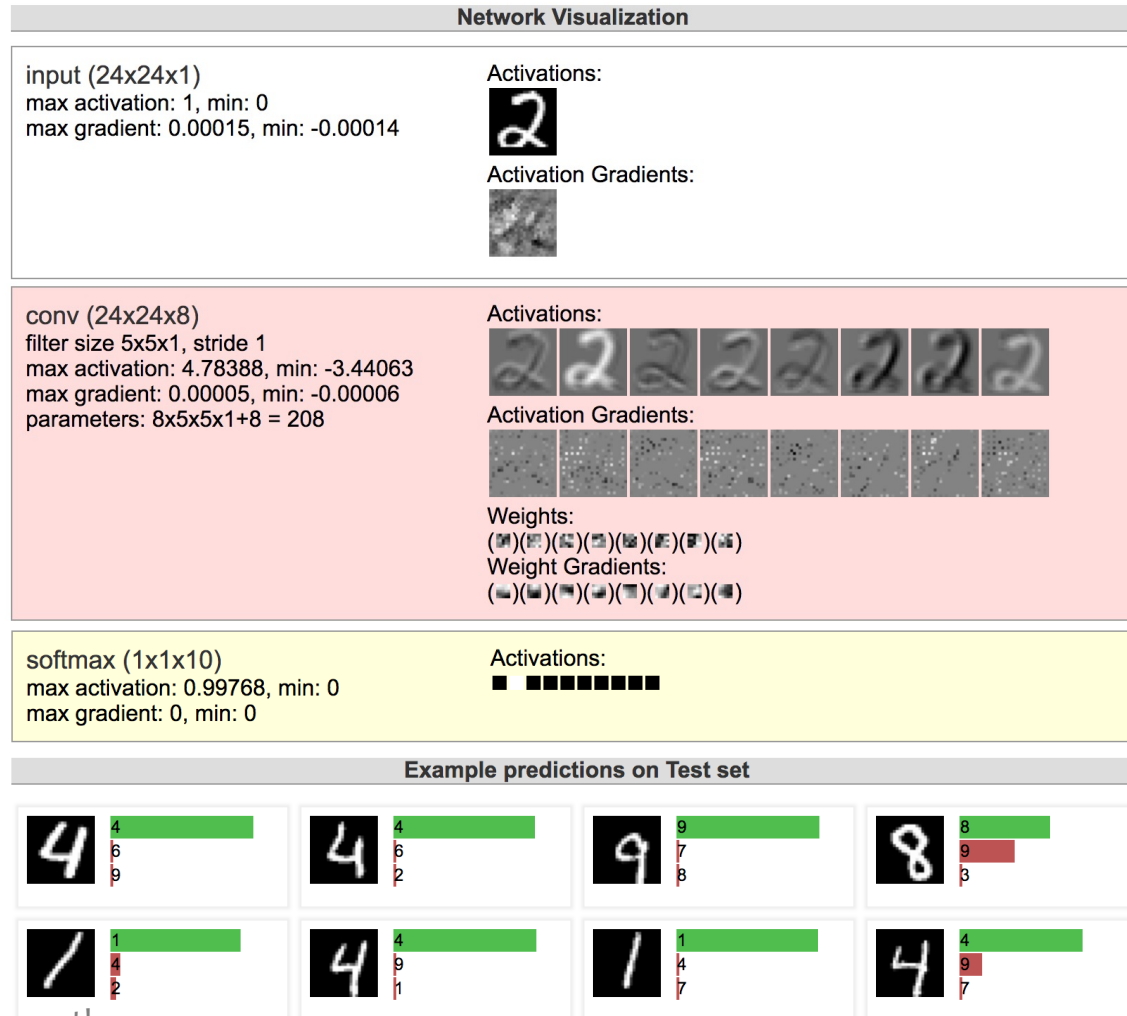


Figure from Andrej Karpathy

# CNN Summary

## CNNs

- Are used for all aspects of **computer vision**, and have won numerous pattern recognition competitions
- Able learn **interpretable features** at different levels of abstraction
- Typically, consist of **convolution** layers, **pooling** layers, **nonlinearities**, and **fully connected** layers