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 & Henry Chai

Lecture 4

Sep. 9, 2024

Reminders

- Homework 0: PyTorch + Weights & Biases
  - Out: Wed, Aug 28
  - Due: Mon, Sep 9 at 11:59pm
  - unique policy for this assignment: we will grant (essentially) any and all extension requests
- Quiz 1: Wed, Sep 11
- Homework 1: Generative Models of Text
  - Out: Mon, Sep 9
  - Due: Mon, Sep 23 at 11:59pm

Two parts: Deep Learning and Language Modeling

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)

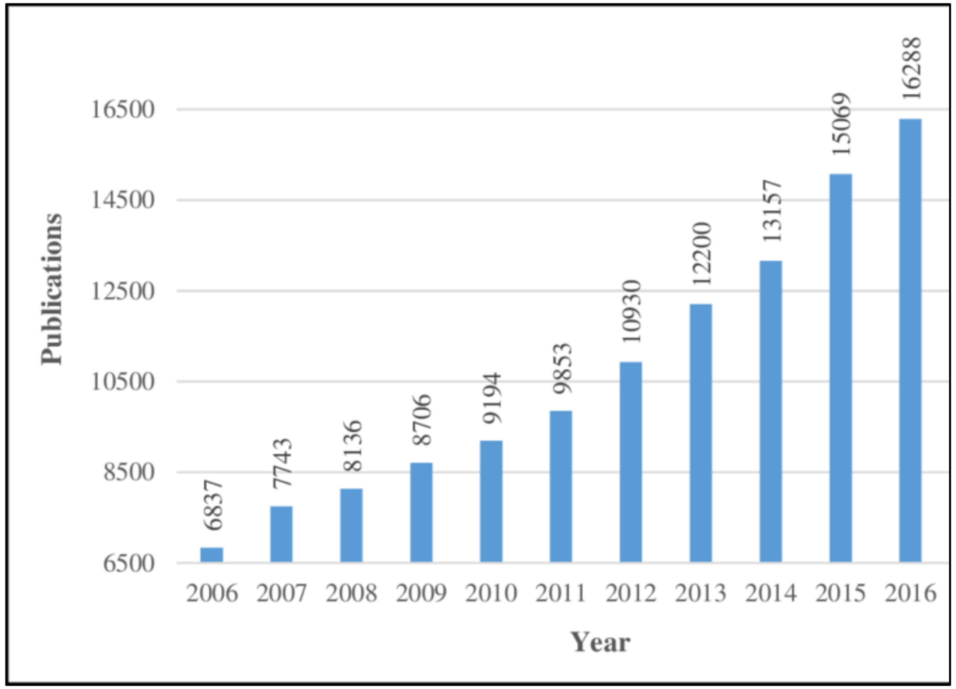


Figure from Vargas et al. (2017)

Deep Network Training

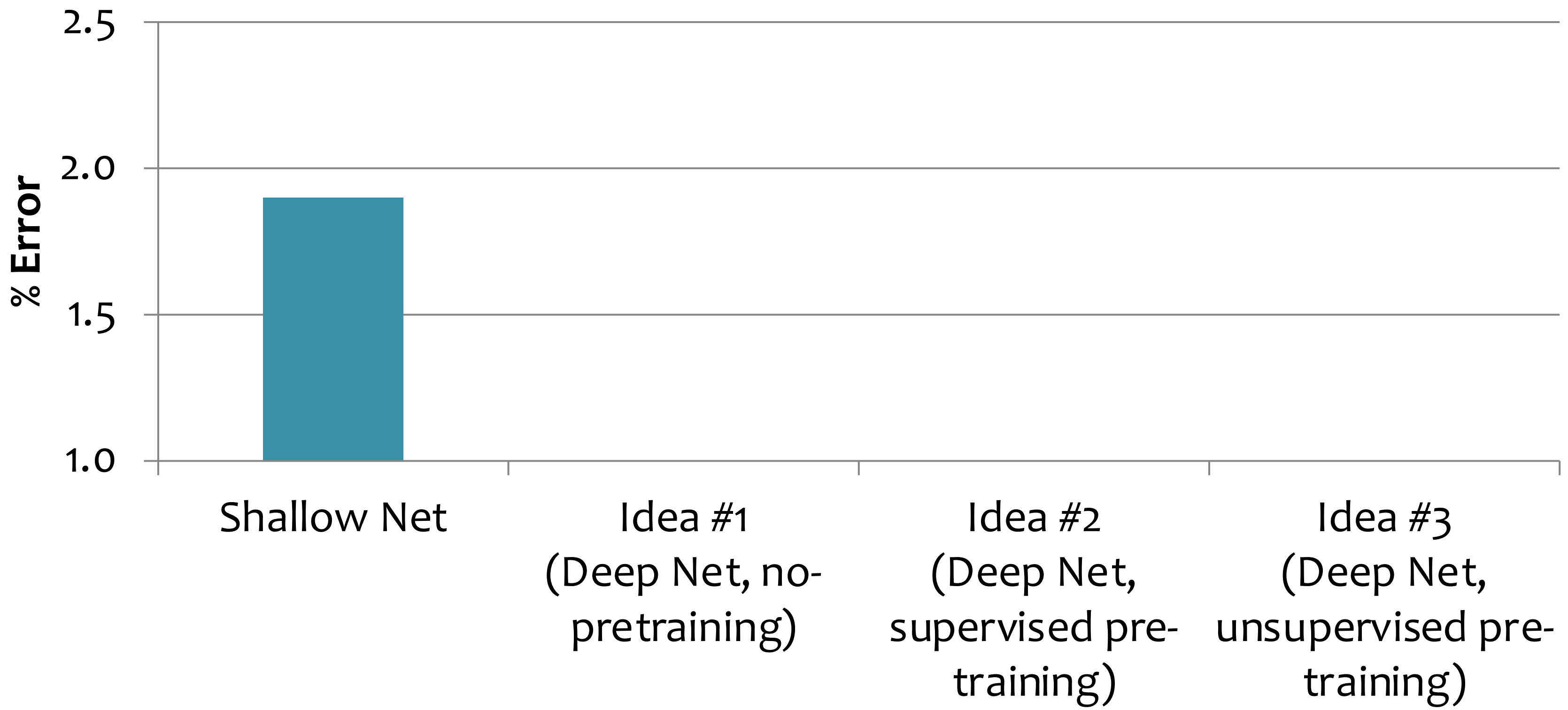
- **Idea #1:**
  1. Supervised fine-tuning only

- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning

- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

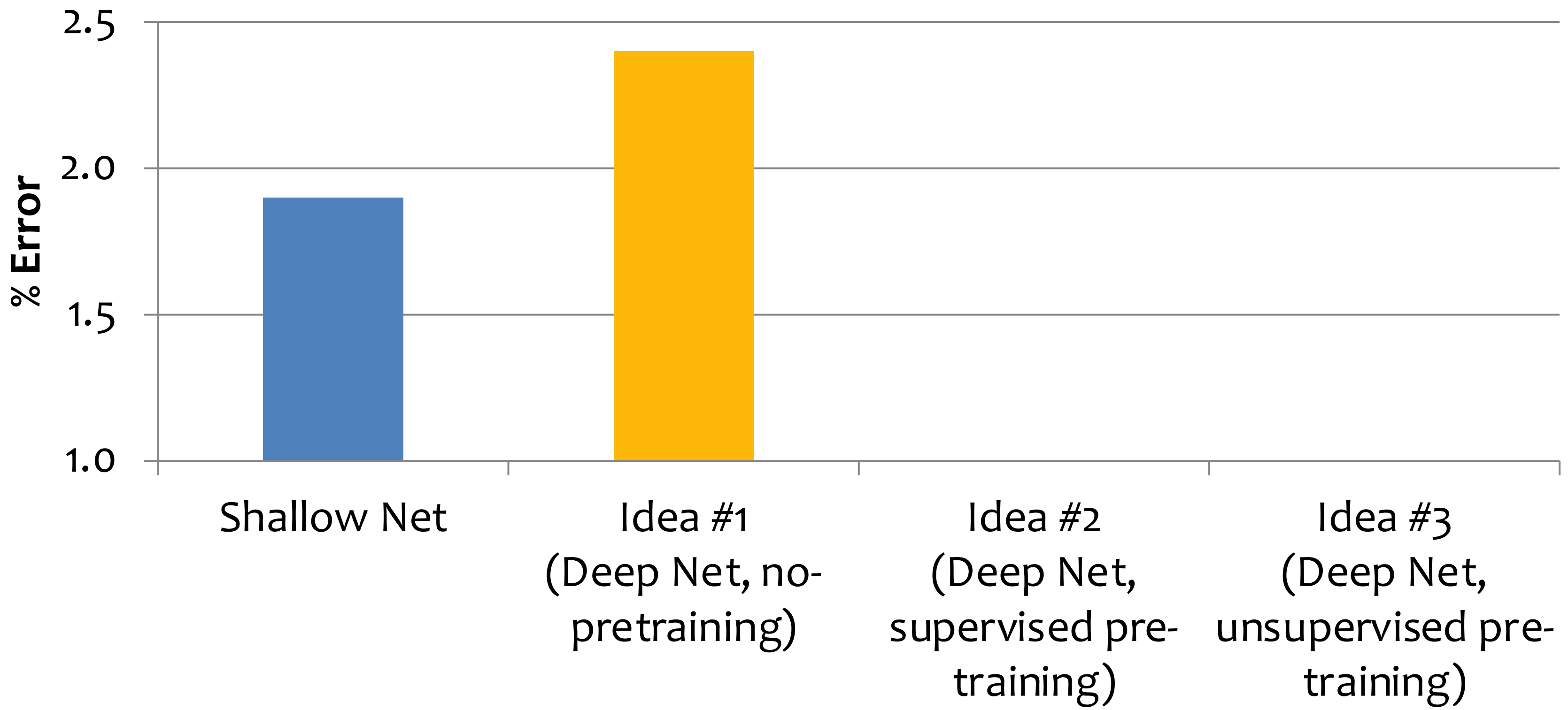
Comparison on MNIST

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



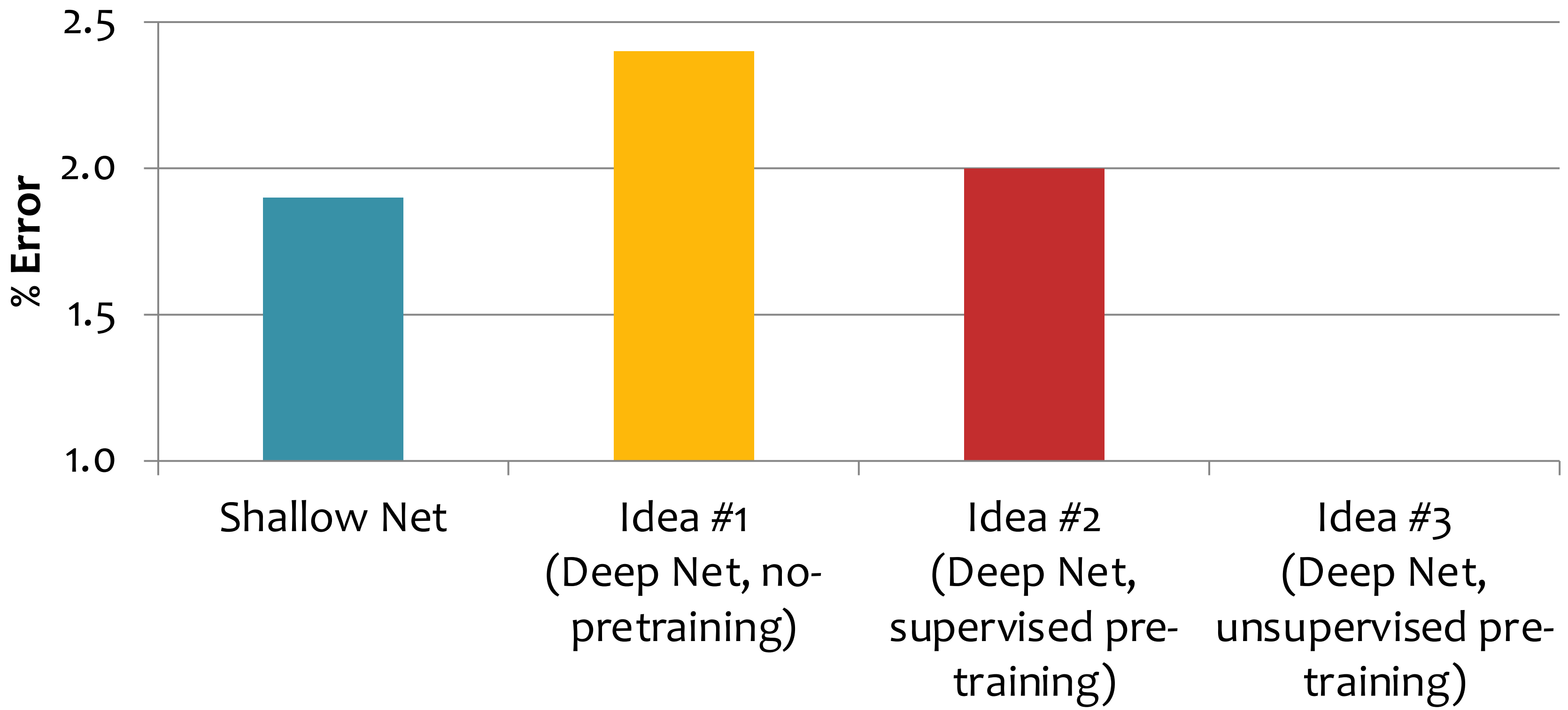
Comparison on MNIST

- 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

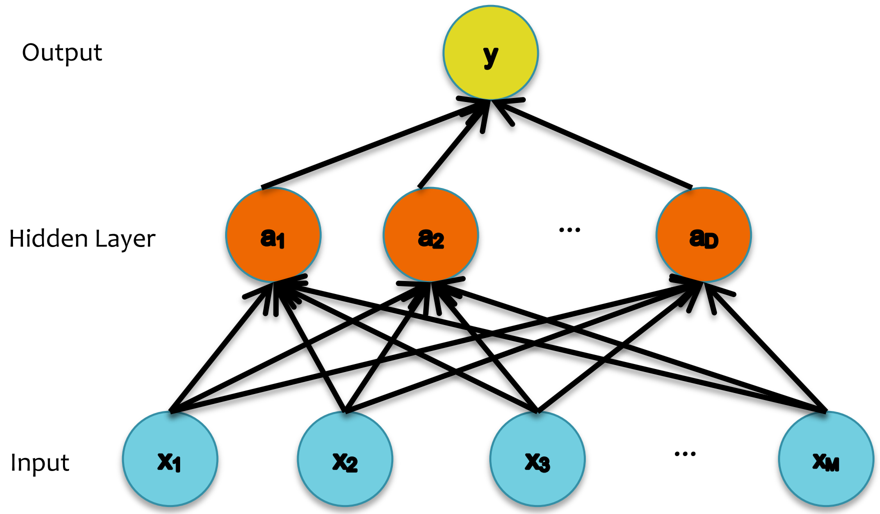
- **Idea #3: (Two Steps)**
  - Use our original idea, but **pick a better starting point**
  - **Train each level** of the model in a **greedy** way

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:

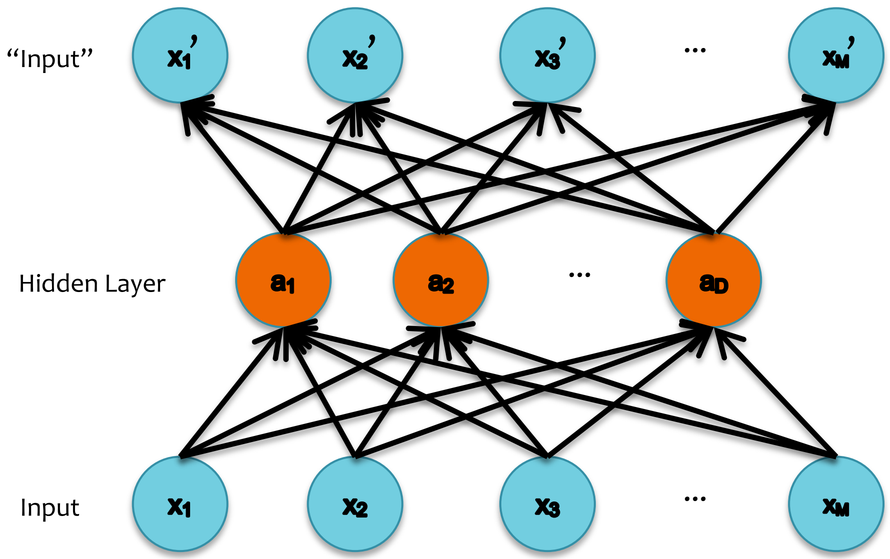
- What should it predict?
- What else do we observe?
- **The input!**



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.

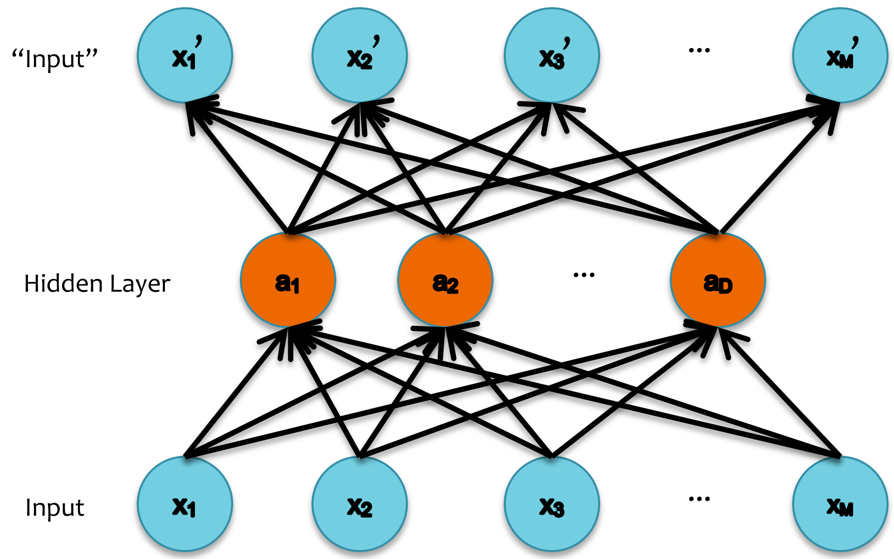
Auto-Encoders

Key idea: Encourage z to give small reconstruction error:

- x' is the *reconstruction* of x
- Loss = || x – DECODER(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)

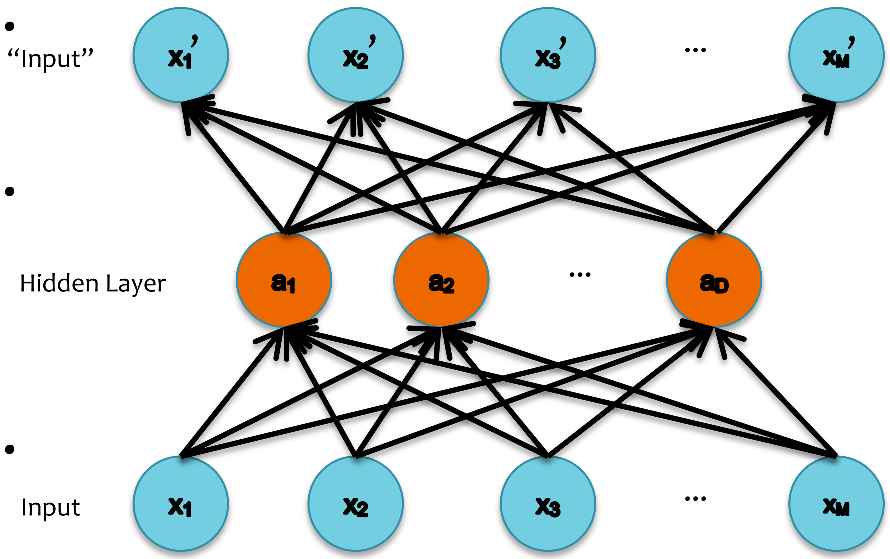


Slide adapted from Raman Arora

The solution:
Unsupervised pre-training

Unsupervised pre-training

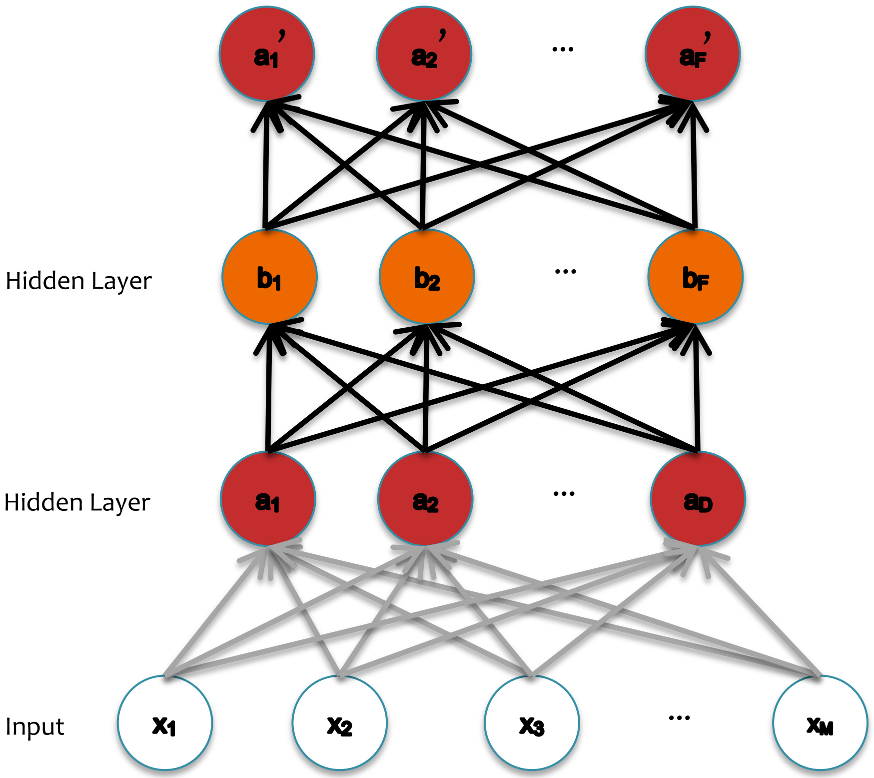
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



The solution:
Unsupervised pre-training

Unsupervised pre-training

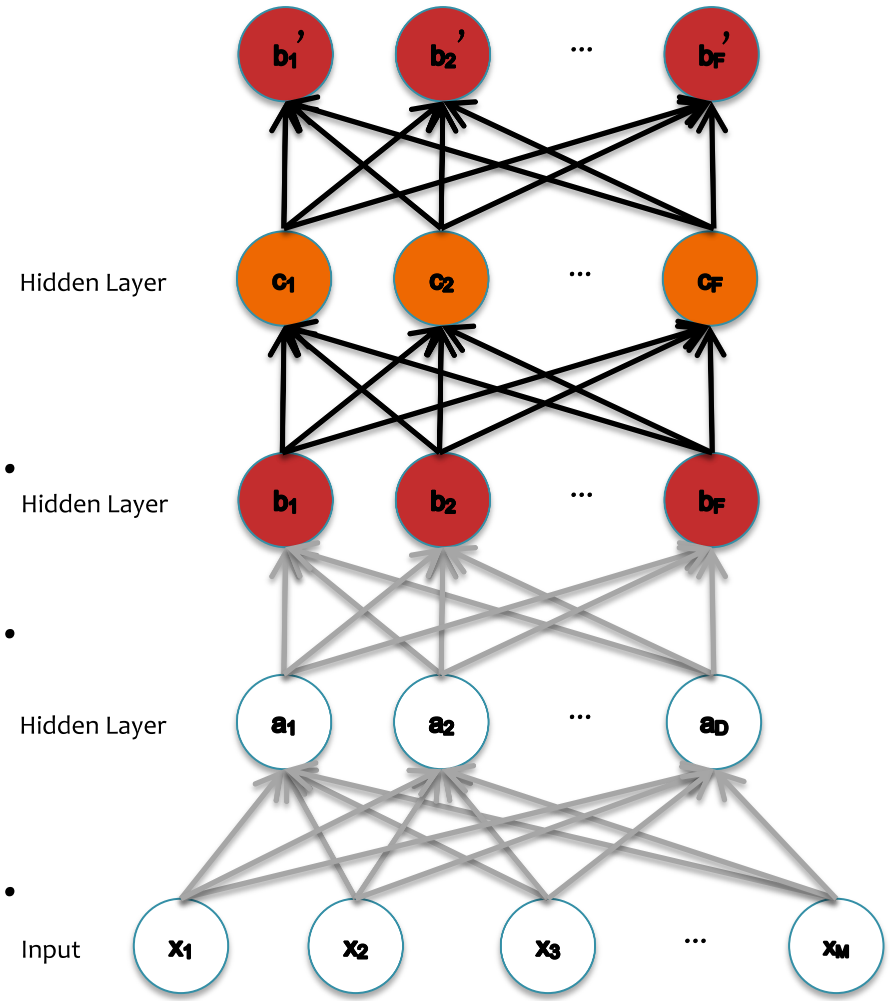
- Work bottom-up
  - Train hidden layer 1. Then fix its parameters.
  - Train hidden layer 2. Then fix its parameters.
  - …
  - Train hidden layer n. Then fix its parameters.



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.

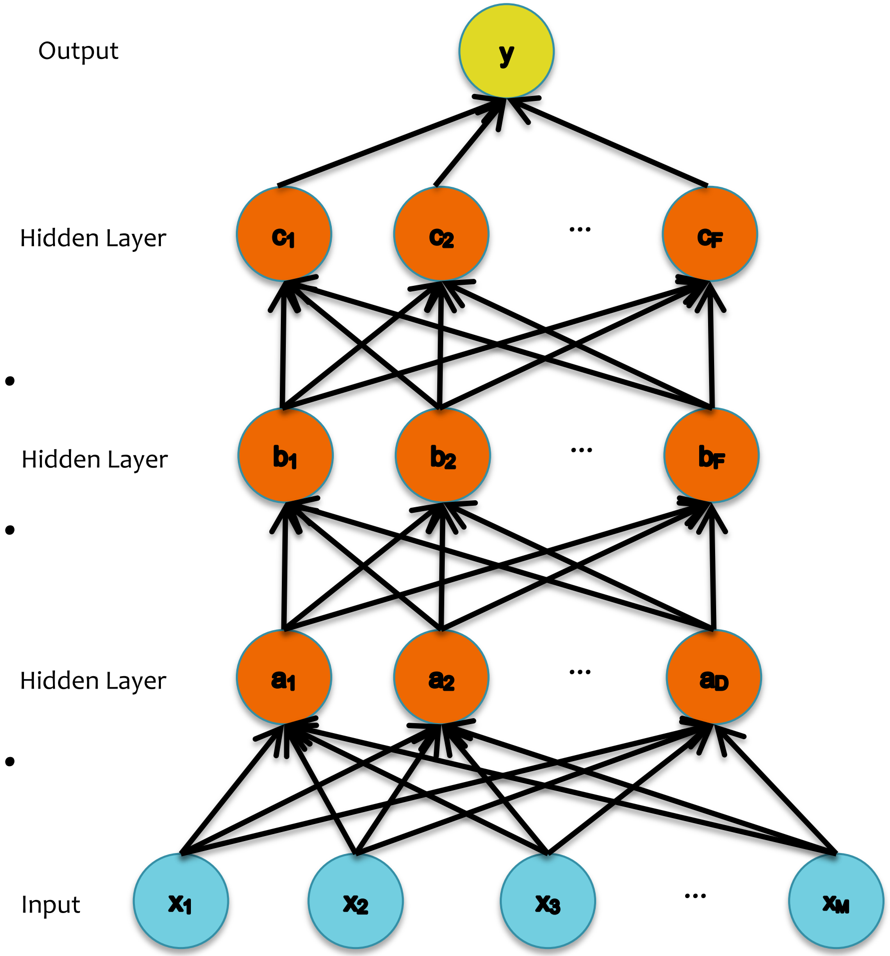


The solution:
Unsupervised pre-training

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

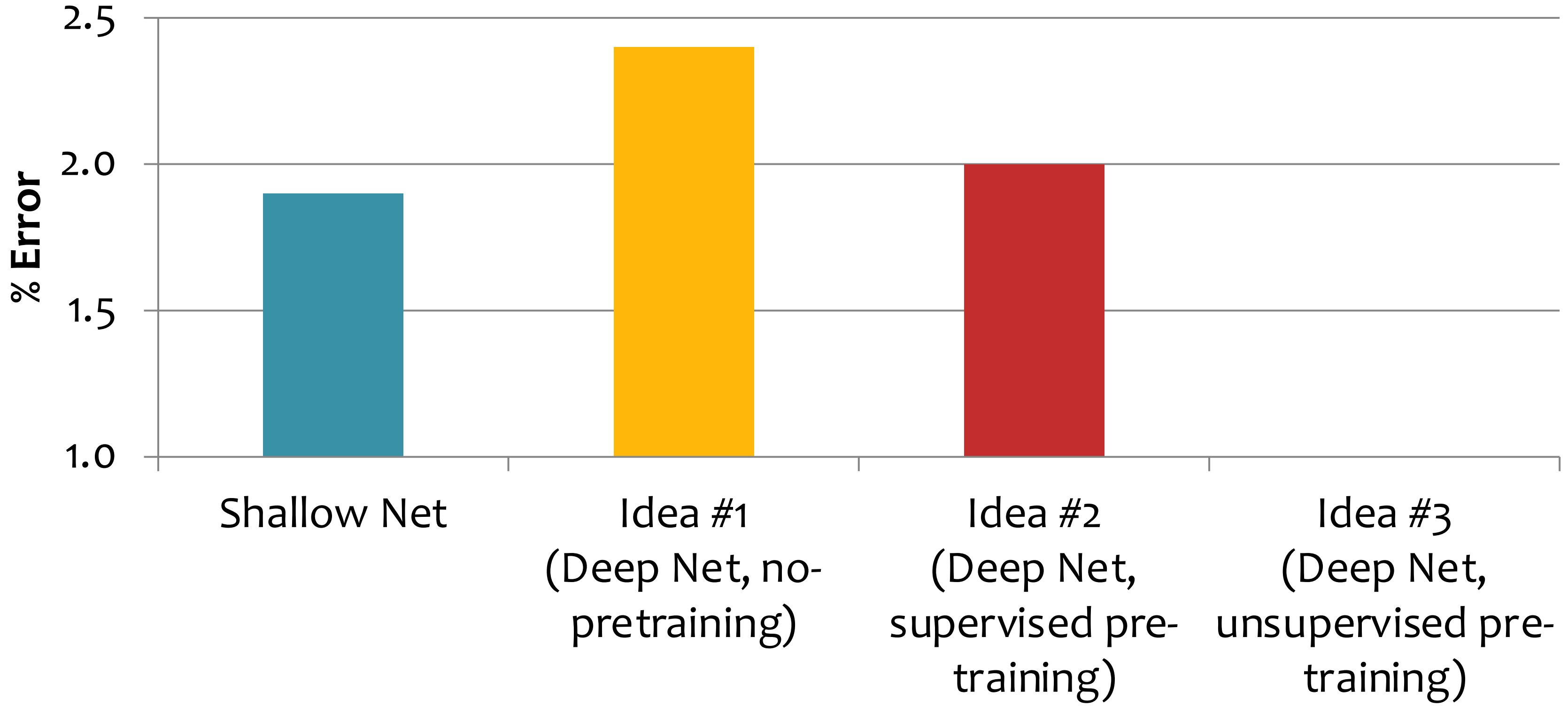
- **Idea #1:**
  1. Supervised fine-tuning only

- **Idea #2:**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning

- **Idea #3:**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

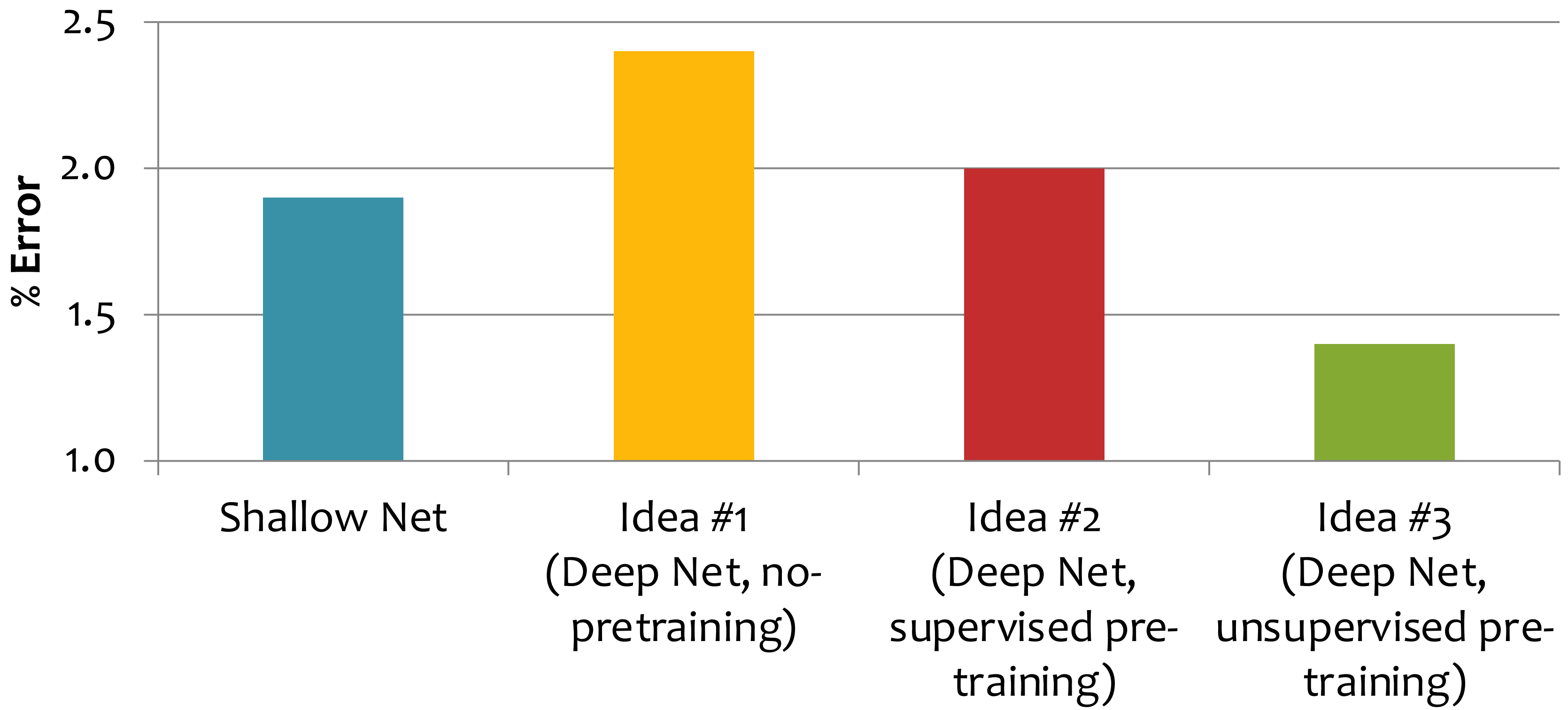
Comparison on MNIST

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

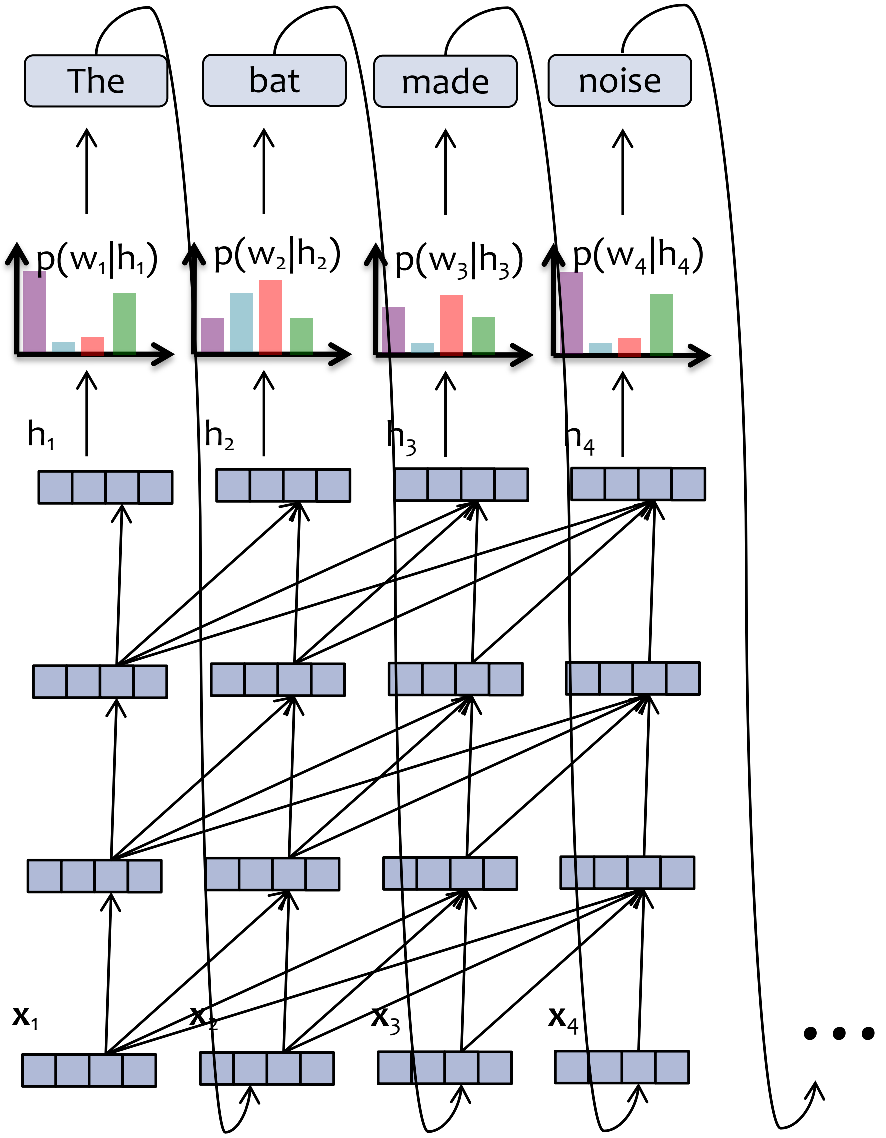


Comparison on MNIST

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



Transformer Language Model



Generative pre-training for a deep language model:

- each training example is an (unlabeled) sentence
- the objective function is the likelihood of the observed sentence

Practically, we can **batch** together many such training examples to make training more efficient

Training Data for LLMs

GPT-3 Training Data:

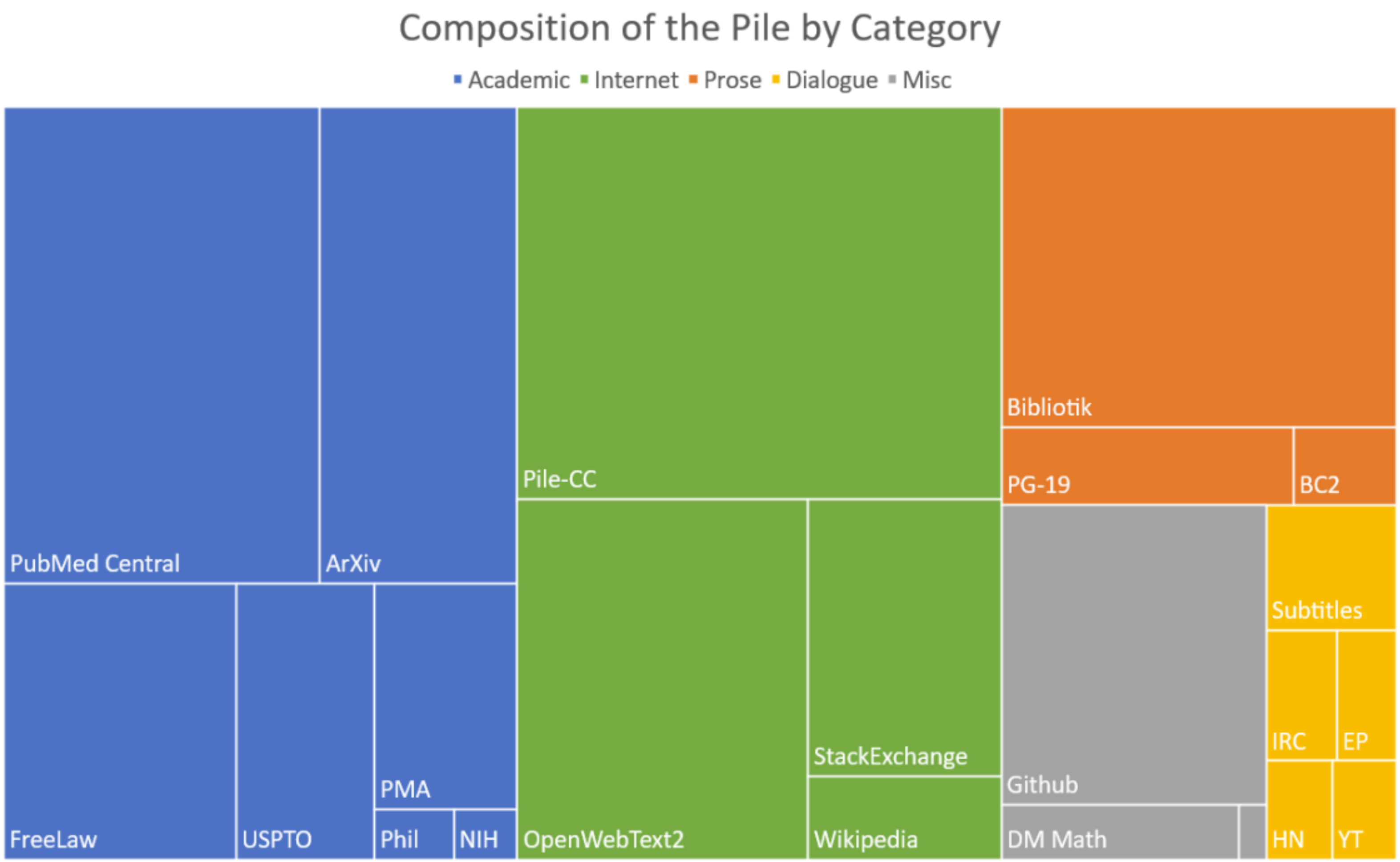
Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table from http://arxiv.org/abs/2005.14165

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 ≈ 1.2 trillion tokens



MODERN TRANSFORMER MODELS

Modern Tranformer Models

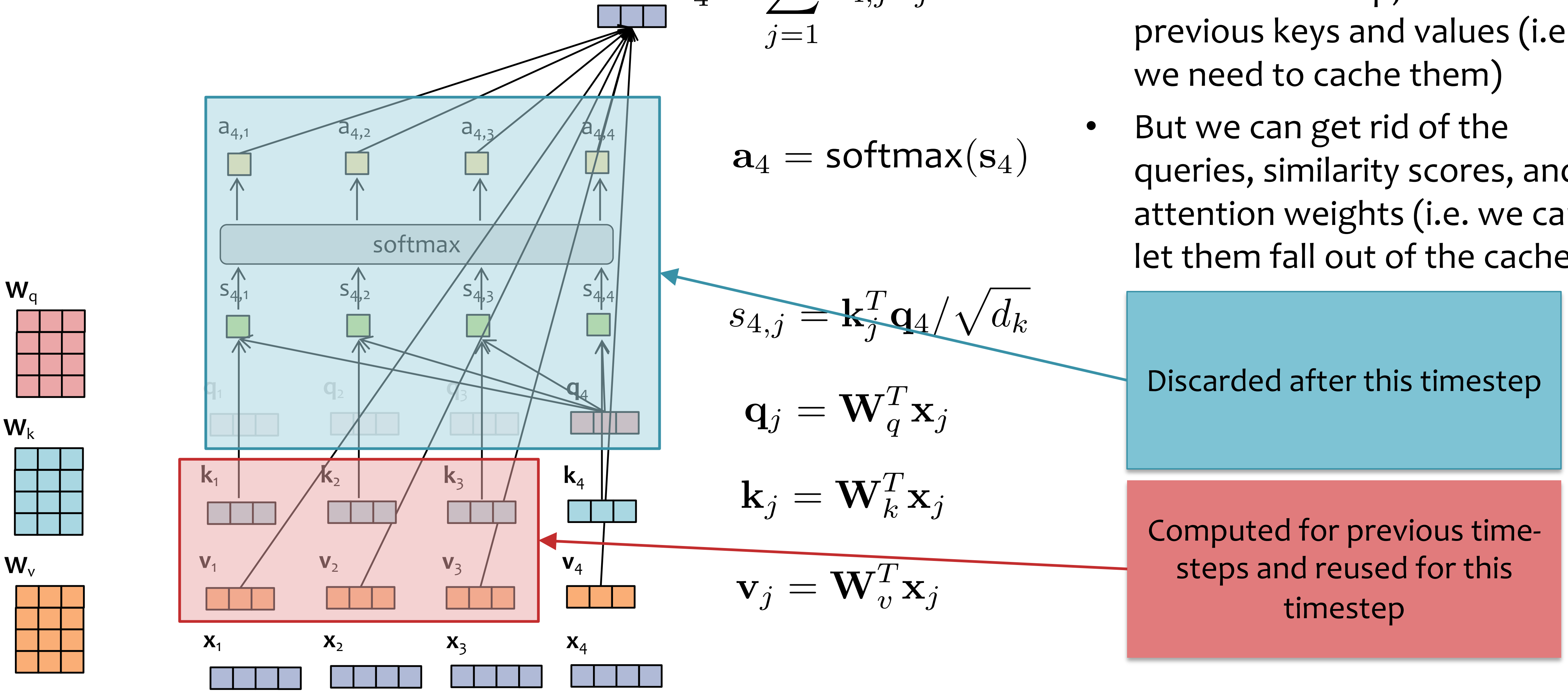
- PaLM (Oct 2022)
  - 540B parameters
  - closed source
  - Model:
    - SwiGLU instead of ReLU, GELU, or Swish
    - **multi-query attention** (MQA) instead of multi-headed attention
    - **rotary position embeddings**
    - **shared input-output embeddings** instead of separate parameter matrices
  - 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 mod W)
  - variant Mixtral offers a **mixture of experts** (roughly 8 Mistral models)

In this section we'll look at four techniques:

1. key-value cache (KV cache)
2. rotary position embeddings (RoPE)
3. grouped query attention (GQA)
4. sliding window attention

Key-Value Cache

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$



$$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$$

$$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$$

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

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

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$

- At each timestep, we reuse all previous keys and values (i.e. we need to cache them)
- But we can get rid of the queries, similarity scores, and attention weights (i.e. we can let them fall out of the cache)

ROTARY POSITION EMBEDDINGS (ROPE)

Rotary Position Embeddings (RoPE)

Q: Why does this slide have so many typos?

A: I'm really not sure. I very meticulously type up the latex for my slides myself and think carefully about all the things I put in them.

RoPE attention:

$$f_q(\mathbf{x}_t, m) \triangleq \mathbf{R}_{\Theta,m} \mathbf{W}_q^T \mathbf{x}_t$$

$$f_k(\mathbf{x}_j, m) \triangleq \mathbf{R}_{\Theta,m} \mathbf{W}_k^T \mathbf{x}_j$$

$$s_{t,j} = f_k(\mathbf{x}_j, m)^T f_q(\mathbf{x}_t, m) / \sqrt{|\mathbf{k}|}, \quad \forall j, t \text{ where } m = t - j$$

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

wrong

wrong

wrong

wrong

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$, and the rotary matrix $\mathbf{R}_{\Theta,m} \in \mathbb{R}^{d_k \times d_k}$ is given by:

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

The θ_i parameters are fixed ahead of time and defined as below:

wrong

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

Rotary Position Embeddings (RoPE)

Q: Why does this slide have so many typos?

A: I'm really not sure. I very meticulously type up the latex for my slides myself and think carefully about all the things I put in them.

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

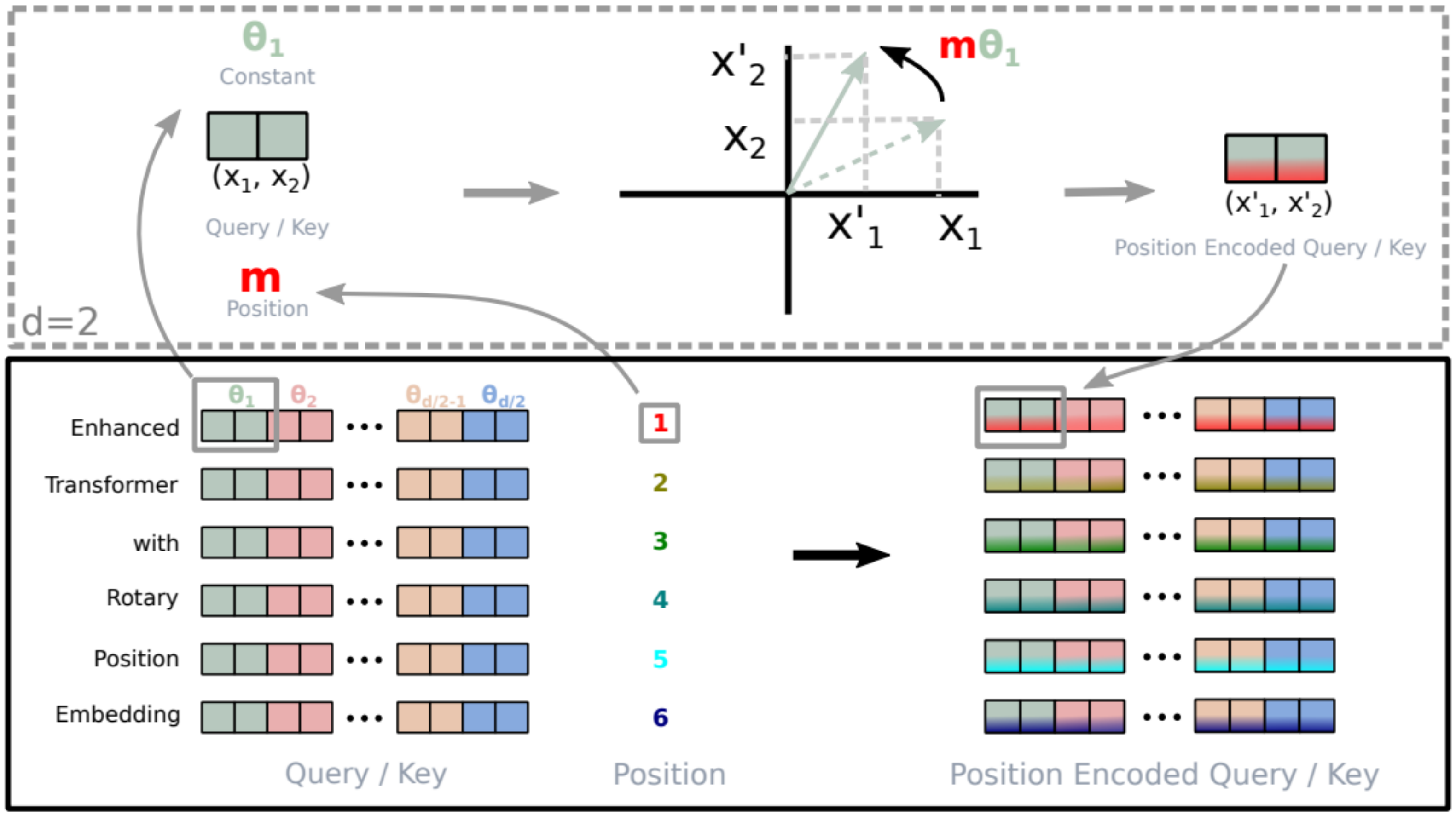


Figure from http://arxiv.org/abs/2104.09864

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:

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

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

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

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

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$. Herein we use $d = d_k$ for brevity.

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 θ_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)

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:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j \qquad \mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$
$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j \qquad \tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$
$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

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:

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

Matrix Version of RoPE

RoPE attention:

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

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

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

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

Matrix Version:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k$$

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

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

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

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

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

$$\begin{aligned} \tilde{\mathbf{Y}} &= g(\mathbf{Y}; \Theta) \\ &= \left[\ \mathbf{Y}_{\cdot,1:d/2} \mid \mathbf{Y}_{\cdot,d/2+1:d}\ \right] \odot \cos(\mathbf{C}) \\ &\quad + \left[\ -\mathbf{Y}_{\cdot,d/2+1:d} \mid \mathbf{Y}_{\cdot,1:d/2}\ \right] \odot \sin(\mathbf{C}) \end{aligned}$$

Matrix Version of RoPE

Q: Is this slide correct?

A: I'm really not sure.

But I did write it myself!

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

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

Matrix Version:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k$$

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

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

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

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

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

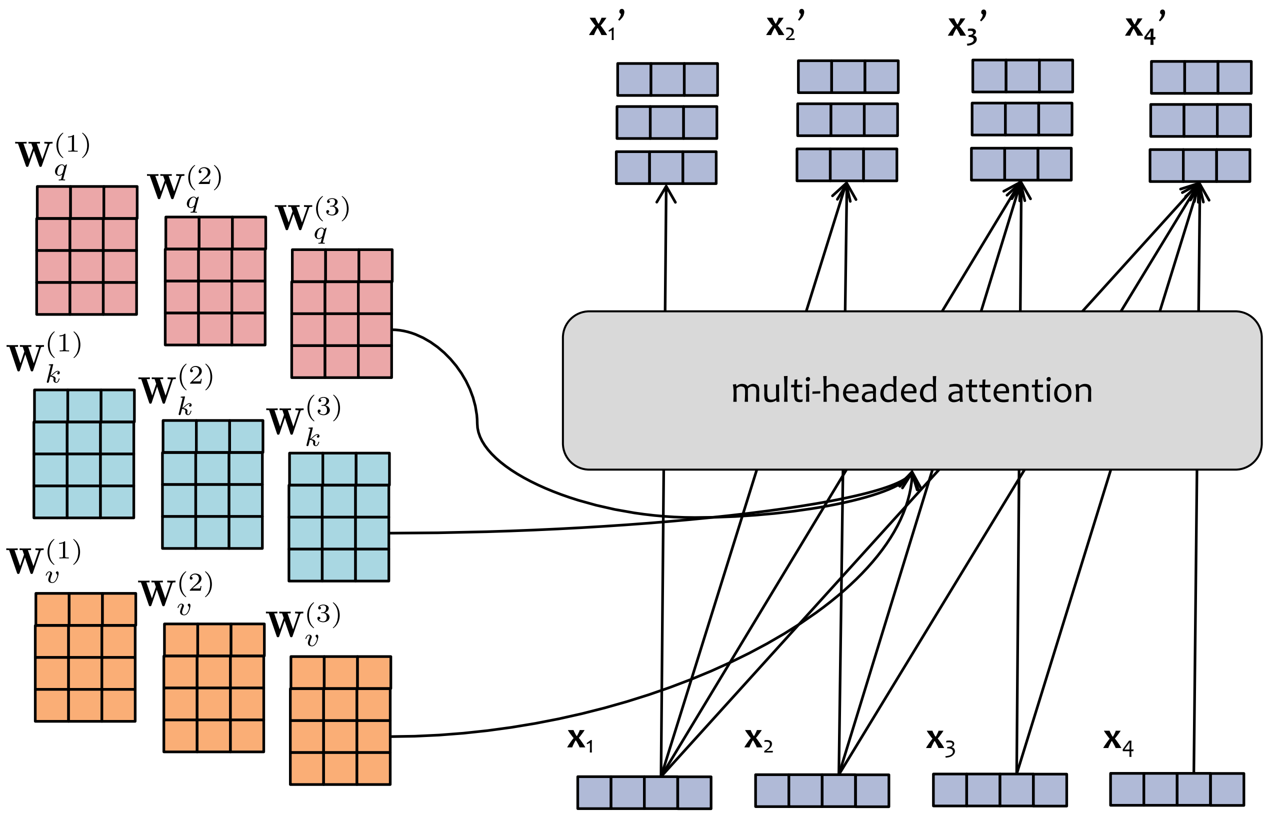
$$\begin{aligned} \tilde{\mathbf{Y}} &= g(\mathbf{Y}; \Theta) \\ &= \left[\ \mathbf{Y}_{\cdot,1:d/2} \ \middle|\ \mathbf{Y}_{\cdot,d/2+1:d}\ \right] \odot \cos(\mathbf{C}) \\ &\quad + \left[\ -\mathbf{Y}_{\cdot,d/2+1:d} \ \middle|\ \mathbf{Y}_{\cdot,1:d/2}\ \right] \odot \sin(\mathbf{C}) \end{aligned}$$

GROUPED QUERY ATTENTION (GQA)

Matrix Version of Multi-Headed (Causal) Attention

Recall…

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



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

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

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

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

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

Grouped Query Attention (GQA)

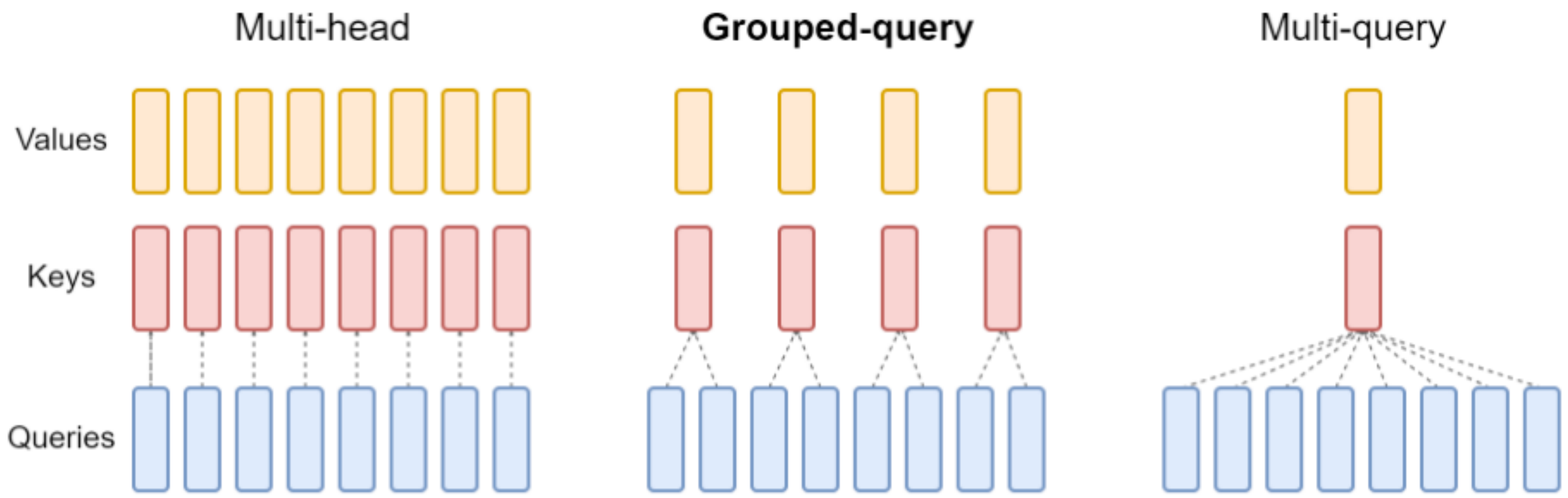
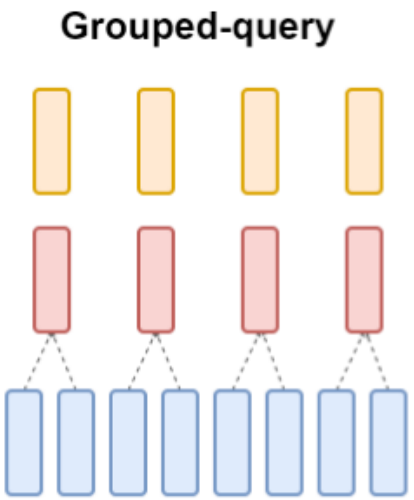


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between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

Figure from http://arxiv.org/abs/2305.13245

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).

$$\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_T]^T$$
$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \dots, h_{kv}\}$$
$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \dots, h_{kv}\}$$
$$\mathbf{Q}^{(i,j)} = \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \dots, h_{kv}\}, \forall j \in \{1, \dots, g\}$$

Figure from http://arxiv.org/abs/2305.13245

SLIDING WINDOW ATTENTION

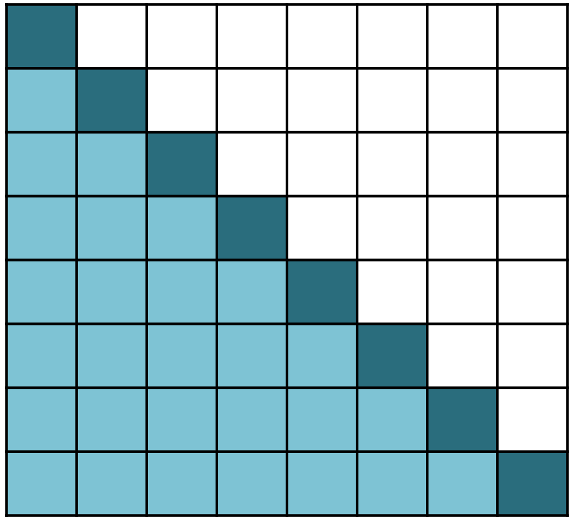
Sliding Window Attention

Sliding Window Attention

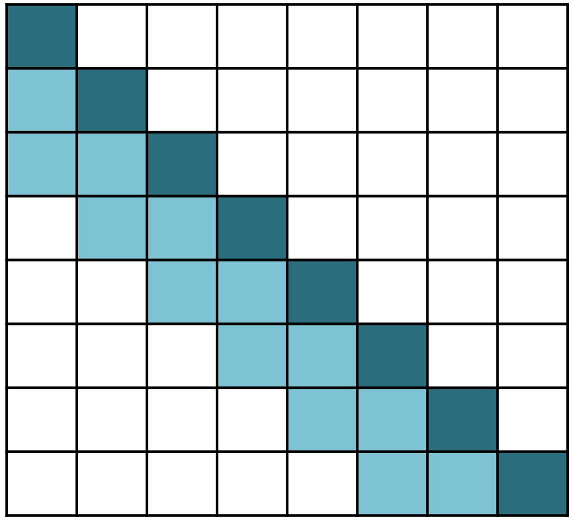
- also called "local attention" and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

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

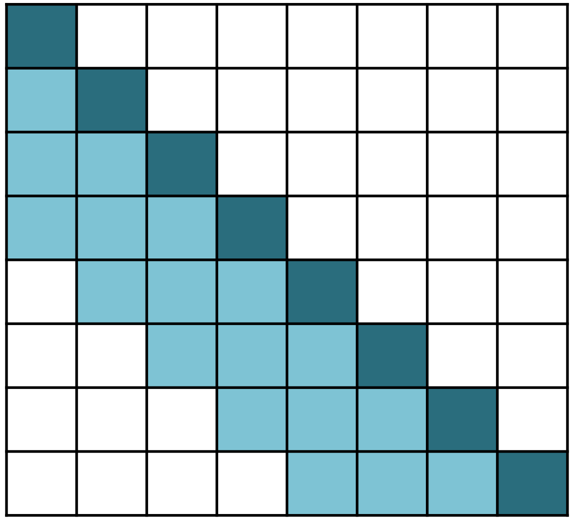
regular causal attention



sliding window attention (w=4)



sliding window attention (w=6)



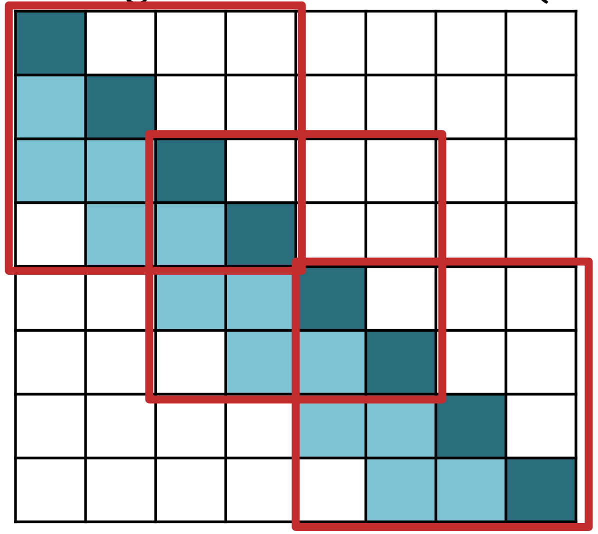
Sliding Window Attention

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 (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

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

sliding window attention (w=4)



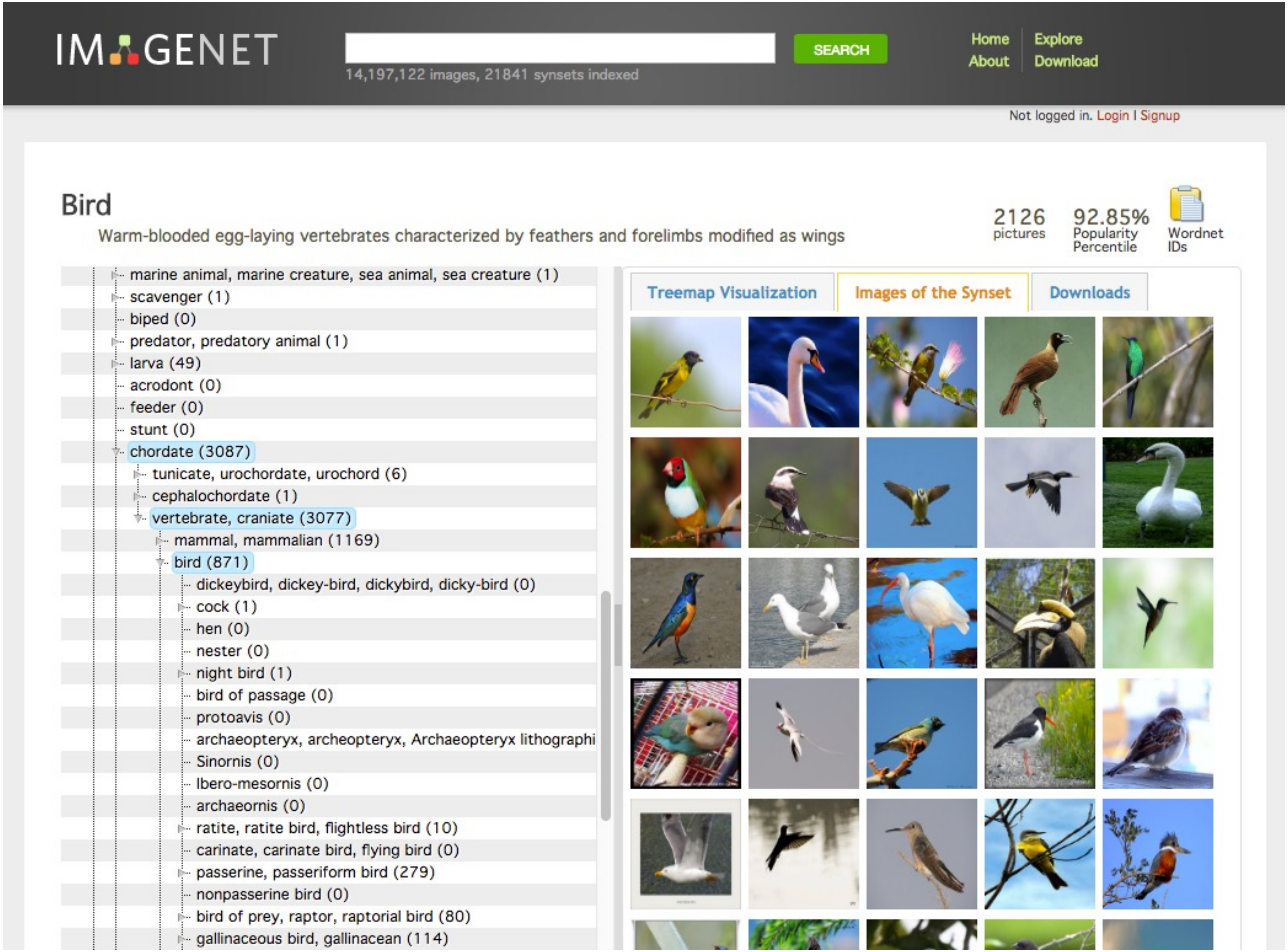
3 ways you could implement

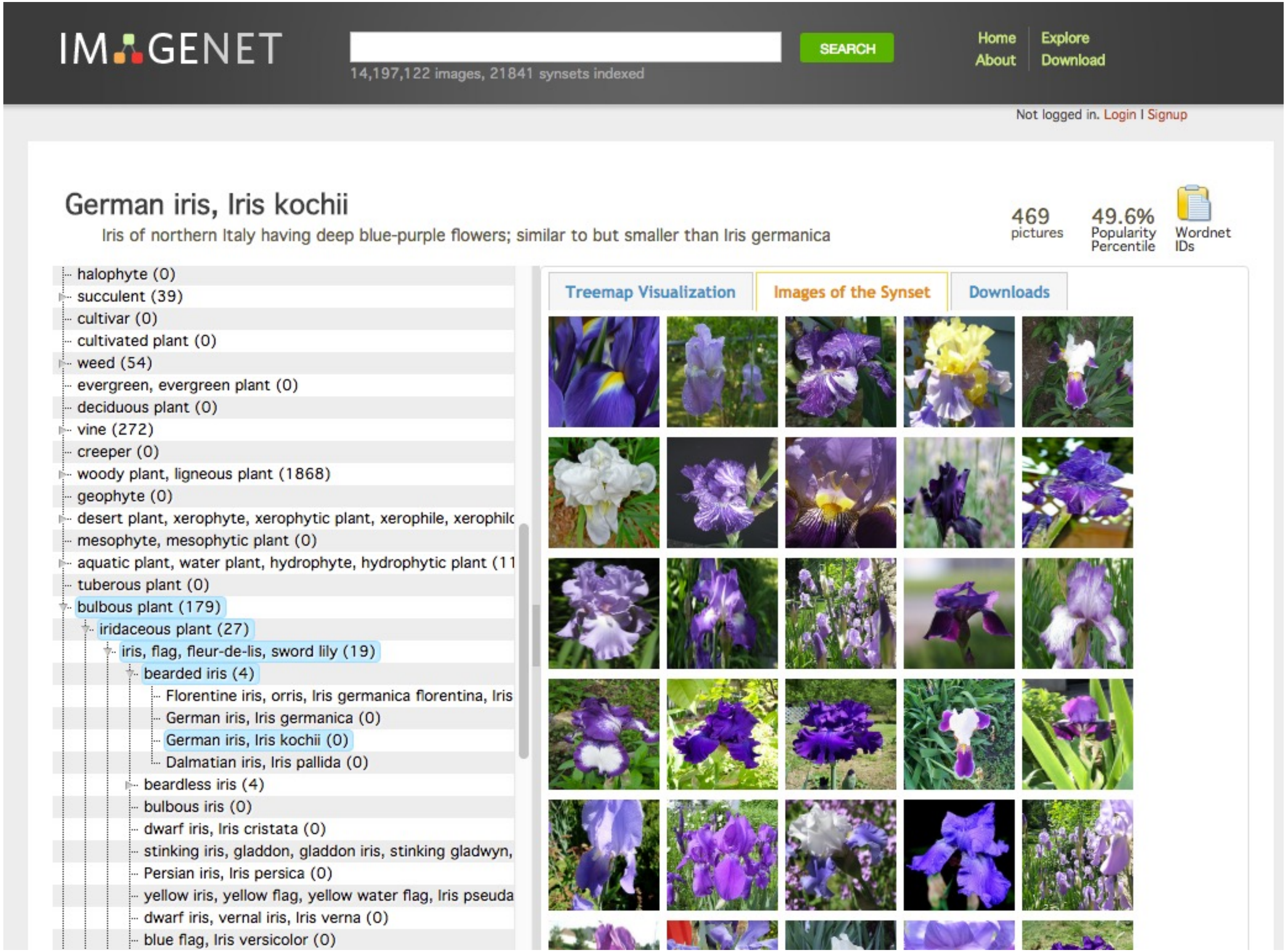
1. *naïve implementation:* just do the matrix multiplication, but this is still slow
2. *for-loop implementation:* asymptotically faster / less memory, but unusable in practice b/c for-loops in PyTorch are too slow
3. *sliding chunks implementation:* break into Q and K into chunks of size w x w, with overlap of ½w; then compute full attention within each chunk and mask out chunk (very fast/low memory in practice)

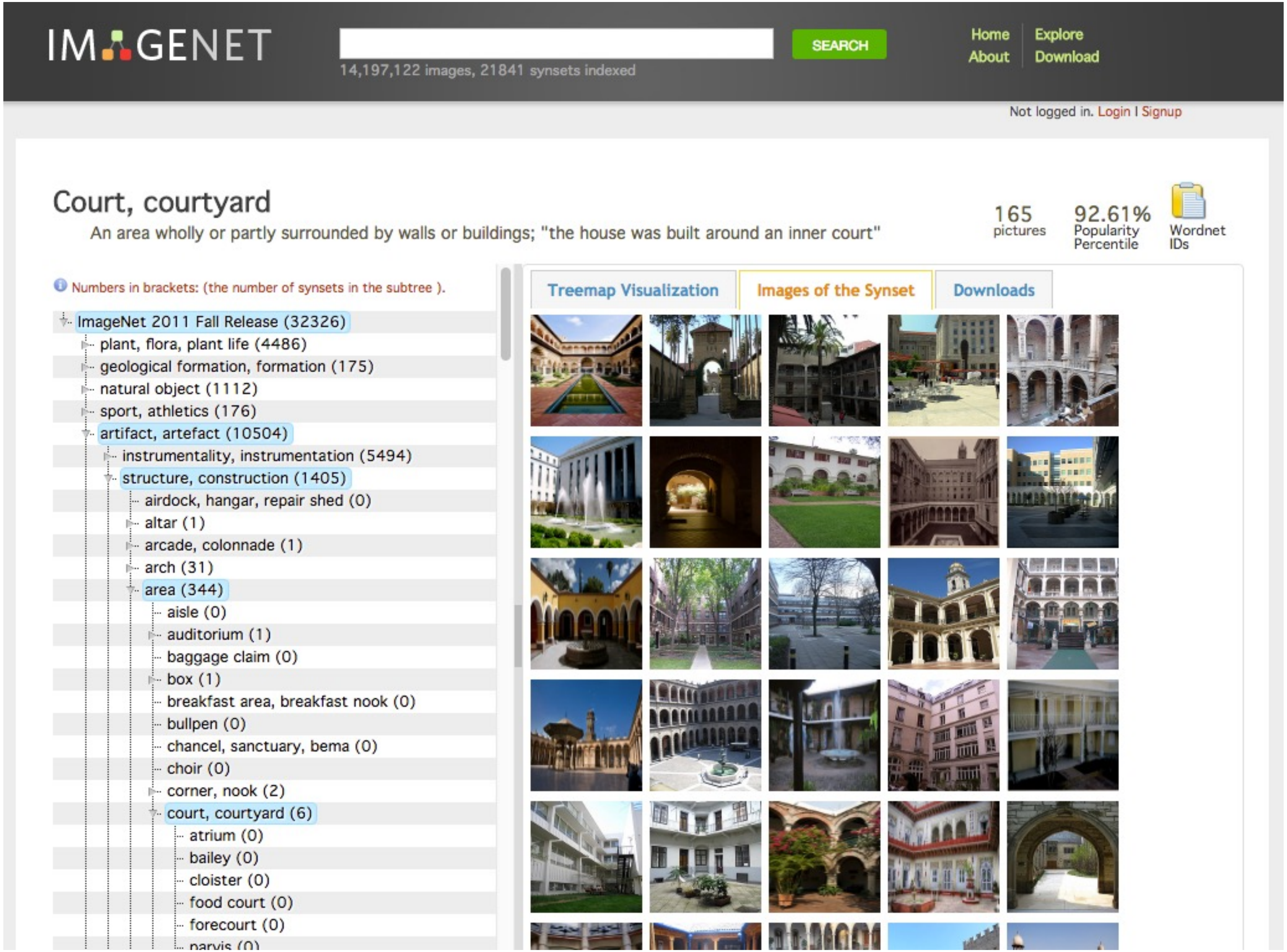
BACKGROUND: COMPUTER VISION

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/


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings
2126 pictures
92.85% Popularity Percentile
Wordnet IDs
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

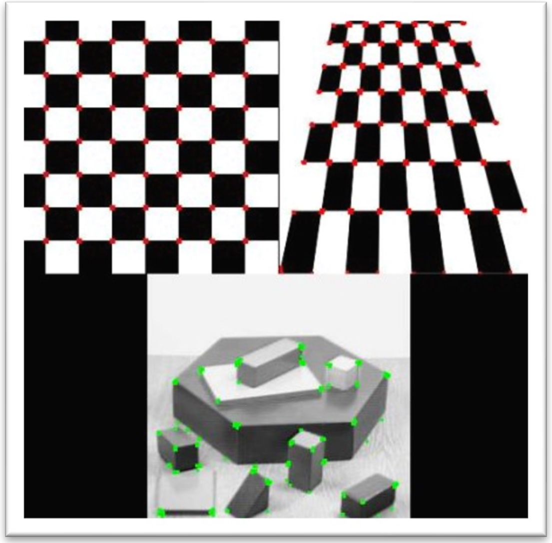

IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
German iris, Iris kochii
Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica
469
pictures
49.6%
Popularity Percentile
Wordnet IDs
halophyte (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, xerophilc
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 pseuda
dwarf iris, vernal iris, Iris verna (0)
blue flag, Iris versicolor (0)
Treemap Visualization
Images of the Synset
Downloads


IMAGENET
SEARCH
14,197,122 images, 21841 synsets indexed
Home
About
Explore
Download
Not logged in. Login I Signup
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
Wordnet IDs
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)
Treemap Visualization
Images of the Synset
Downloads

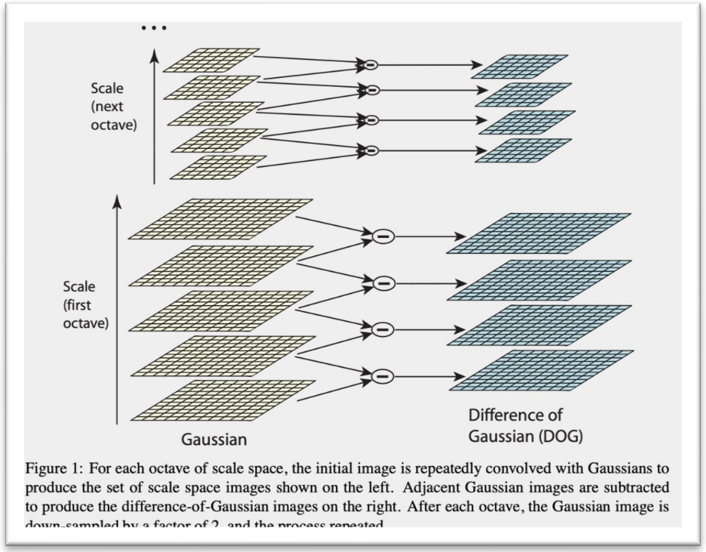
Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)



Scale Invariant Feature Transform (SIFT)



Figures from http://opencv.org

Figure from Lowe (1999) and Lowe (2004)

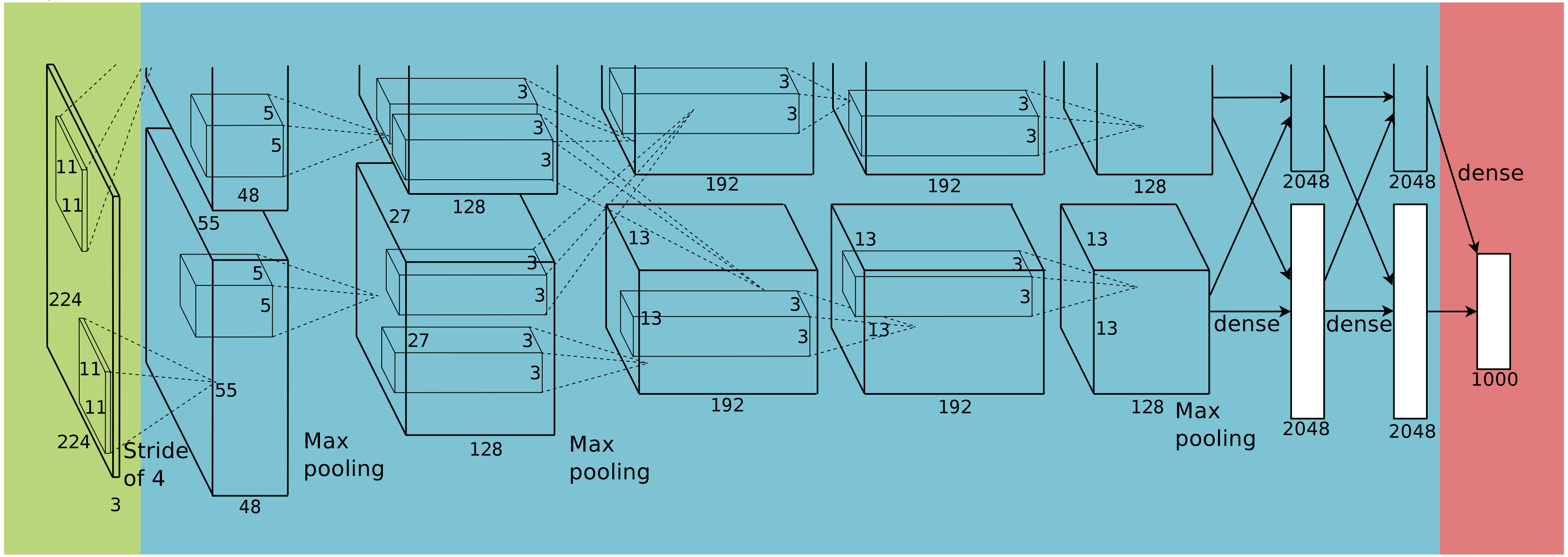
Example: Image Classification

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

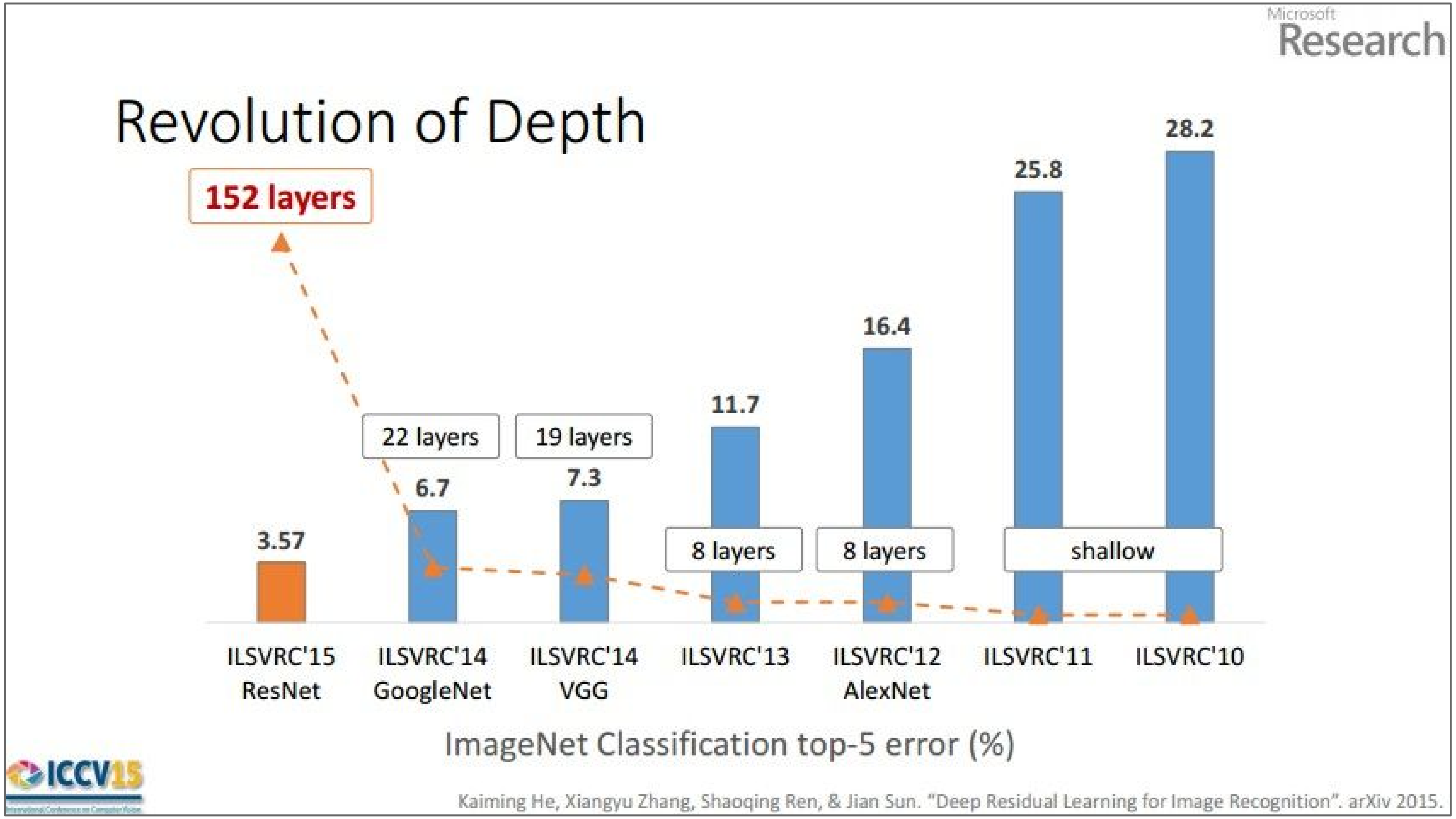
Input image (pixels)

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

1000-way softmax



CNNs for Image Recognition



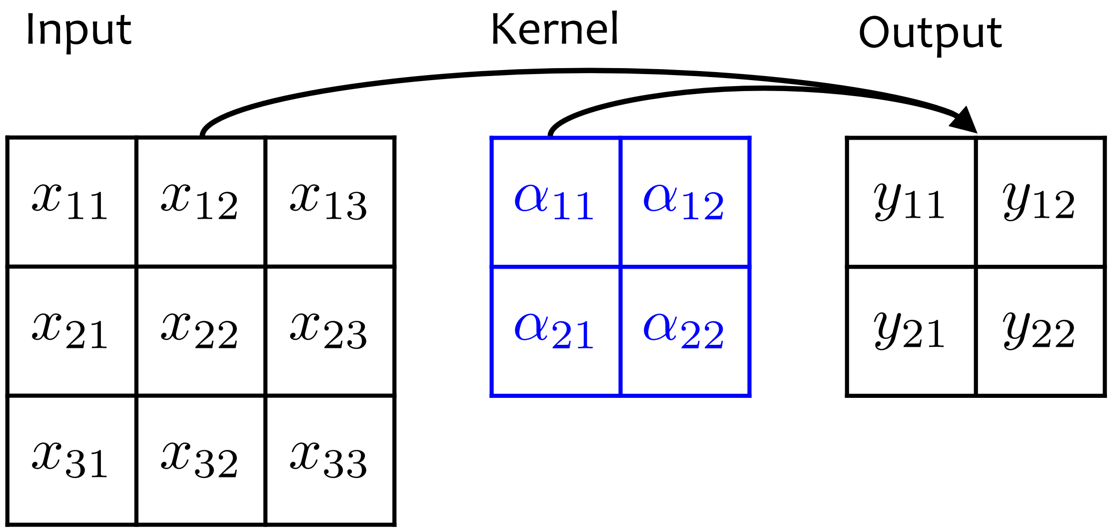
Slide from Kaiming He

CONVOLUTION

2D Convolution

- Basic idea:
  - Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
  - 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

Example: 1 input channel, 1 output channel



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

Slide adapted from William Cohen

2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

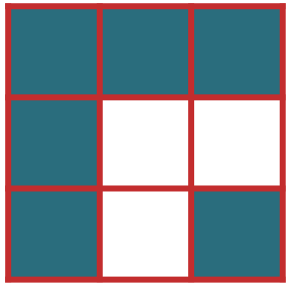
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

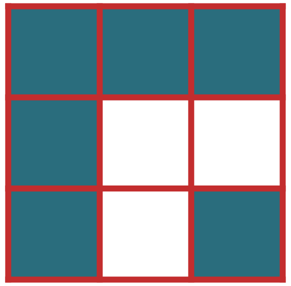
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

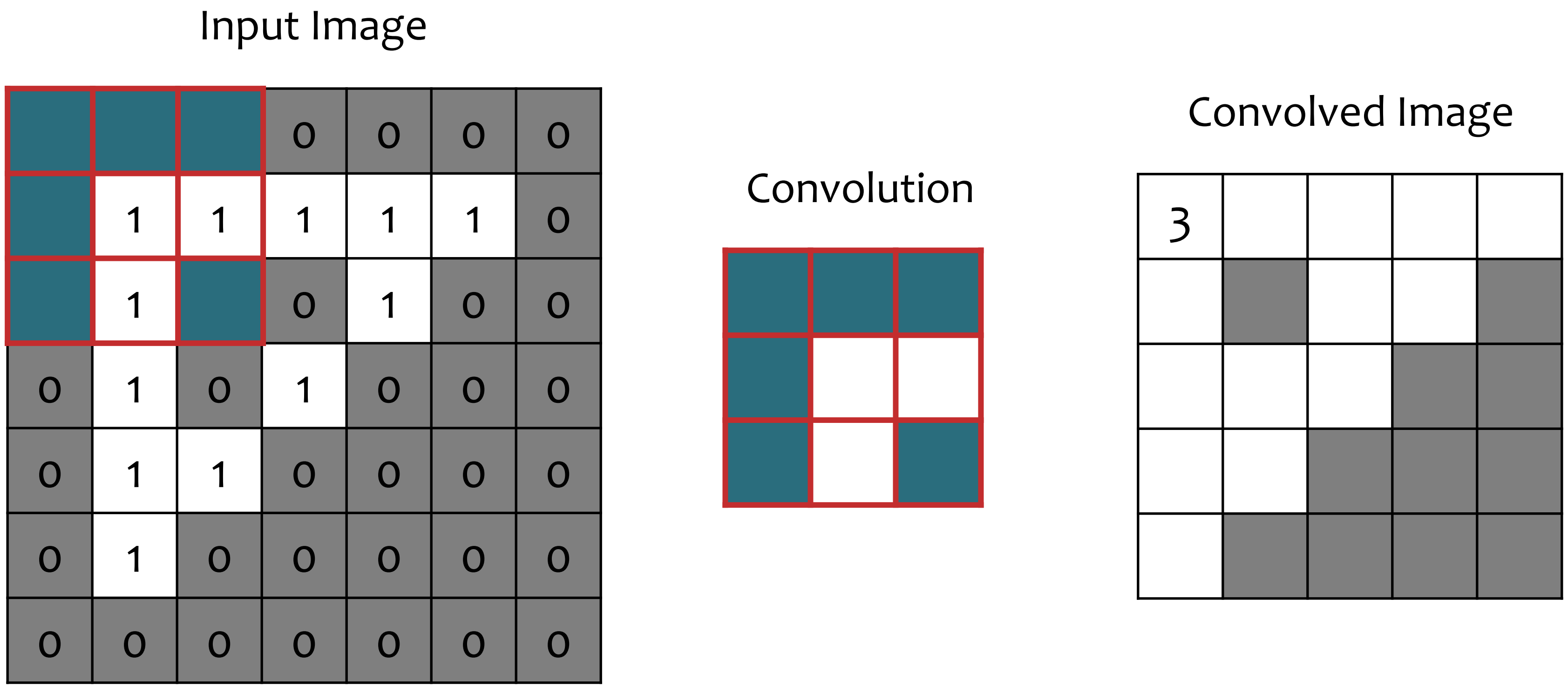


Convolved Image

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

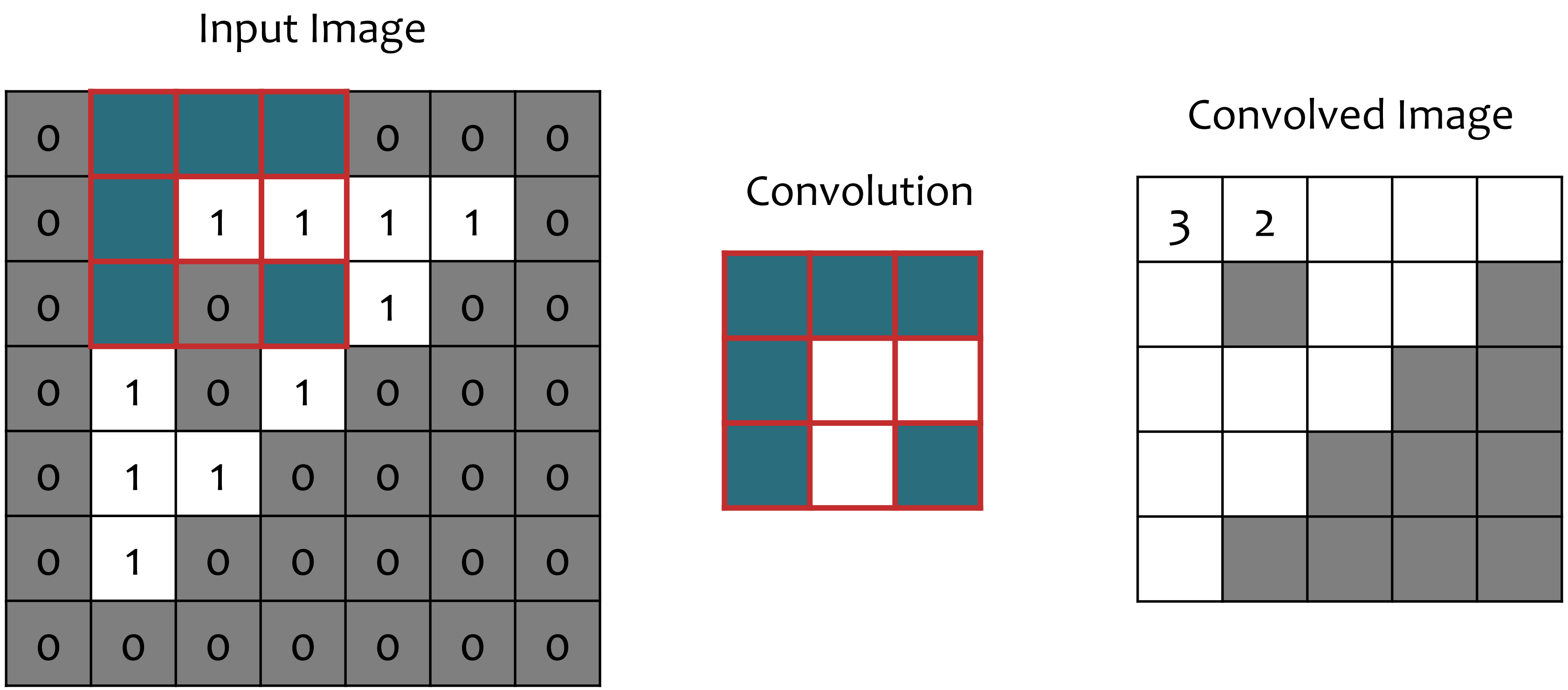
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



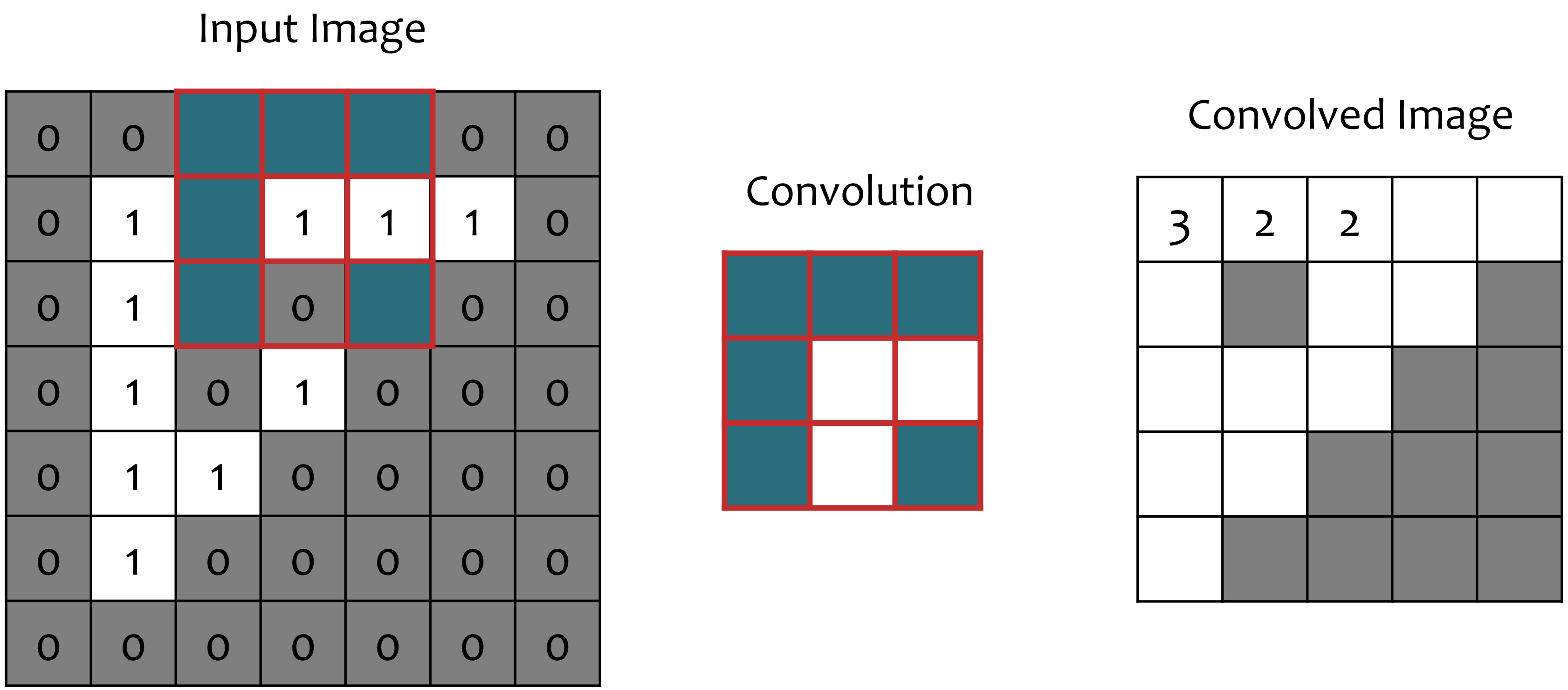
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



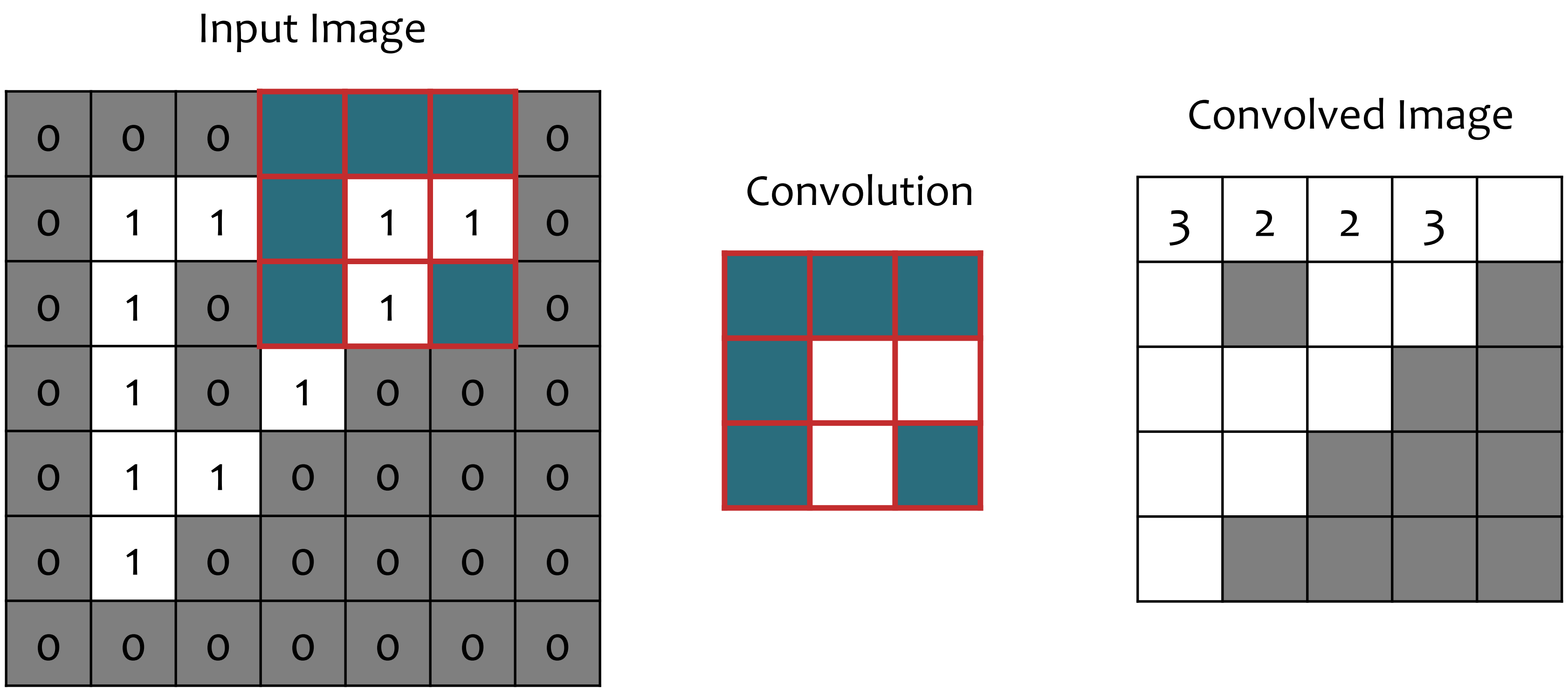
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



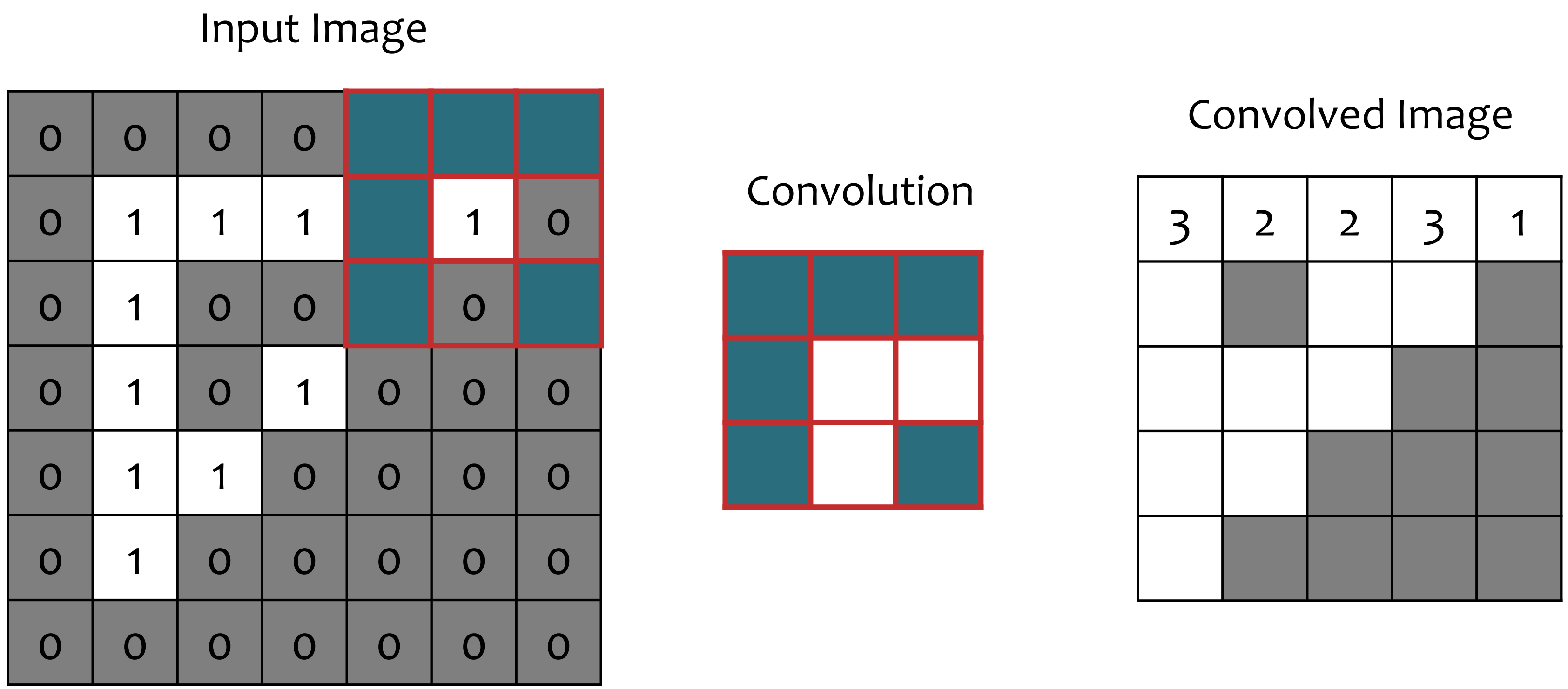
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



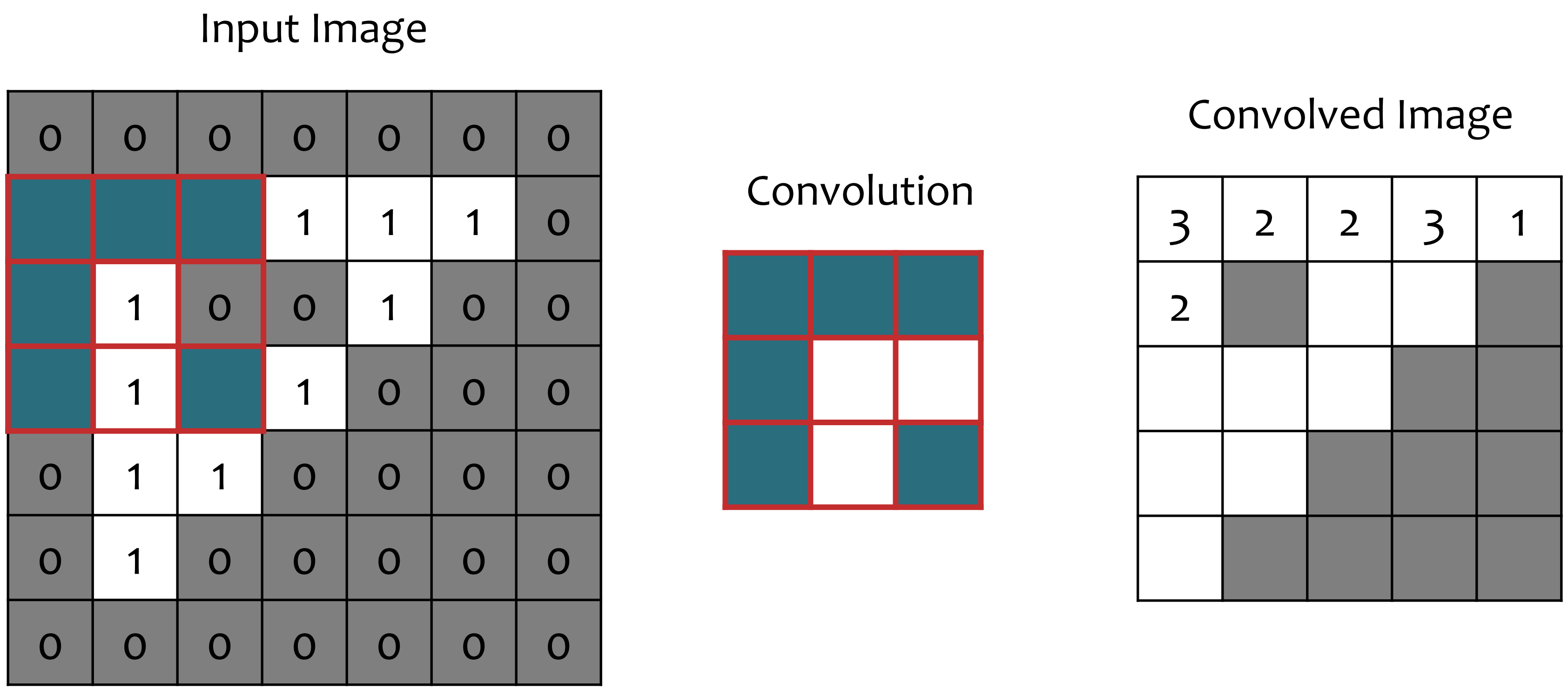
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



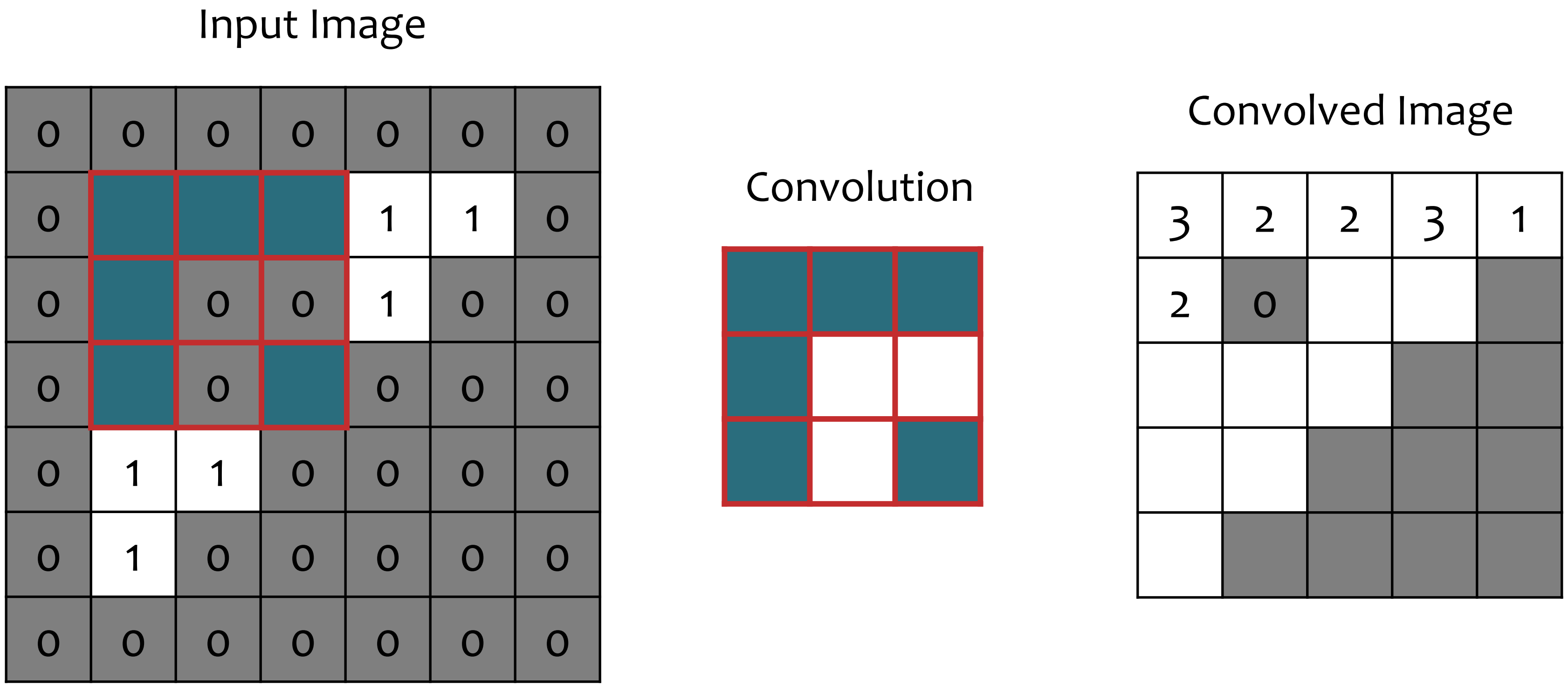
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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



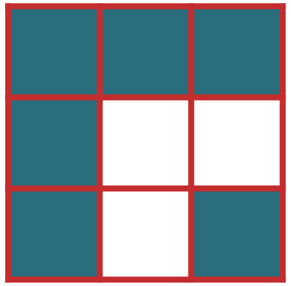
2D Convolution

- Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
- 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

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

Padding

Suppose you want to preserve the size of the original input image in your convolved image.

You can accomplish this by padding your input image with zeros.

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

Padding

Suppose you want to preserve the size of the original input image in your convolved image.

You can accomplish this by padding your input image with zeros.

Input Image

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

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

Kernels for Image Processing

A **convolution matrix** (aka. kernel) 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	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	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

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

Kernels for Image Processing

A **convolution matrix** (aka. kernel) 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	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	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

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

Kernels for Image Processing

A **convolution matrix** (aka. kernel) 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	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Vertical Edge Detector

-1	0	1
-1	0	1
-1	0	1

Convolved Image

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

Kernels for Image Processing

A **convolution matrix** (aka. kernel) 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	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Horizontal Edge Detector

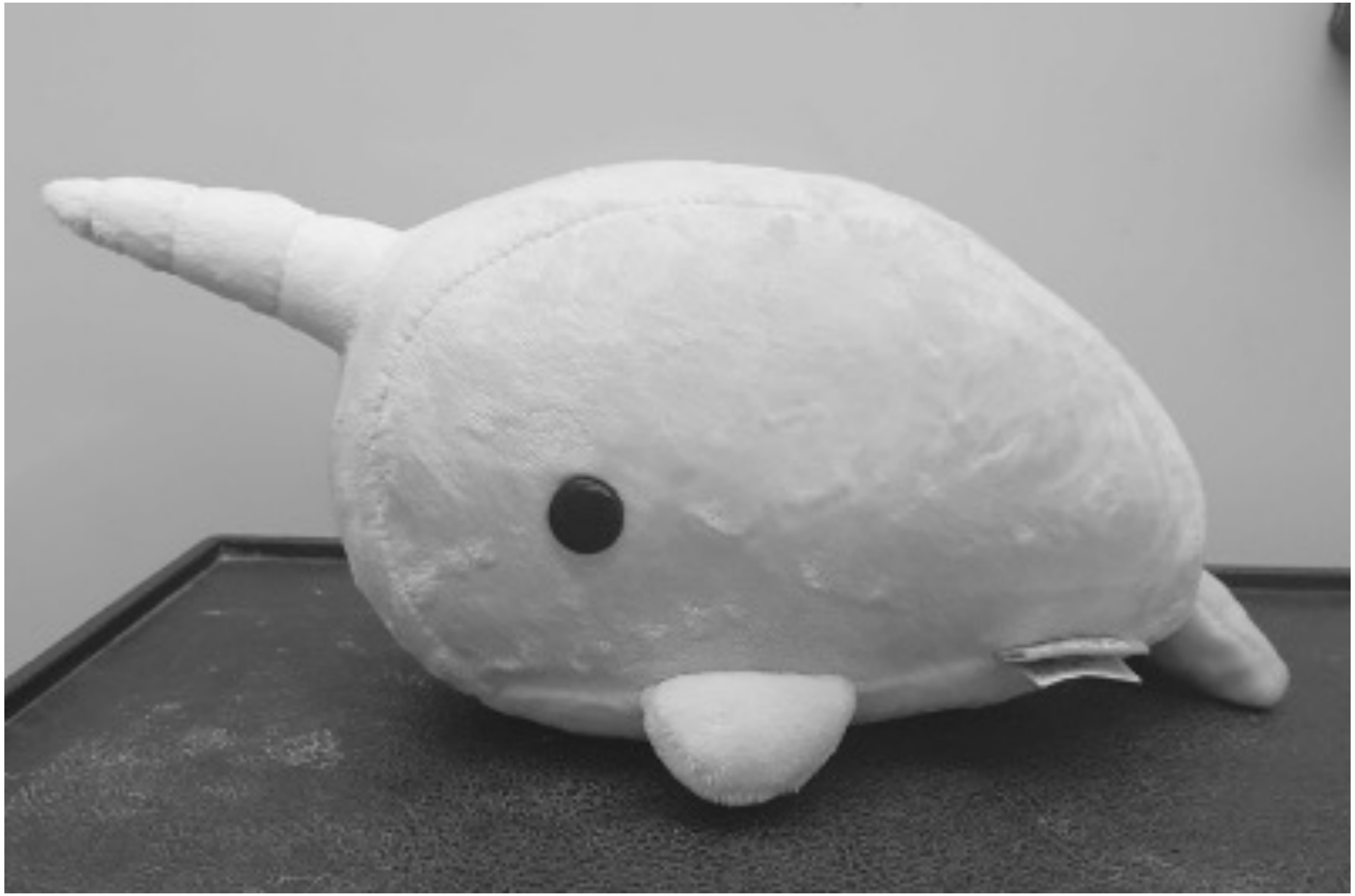
-1	-1	-1
0	0	0
1	1	1

Convolved Image

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

Convolution Examples

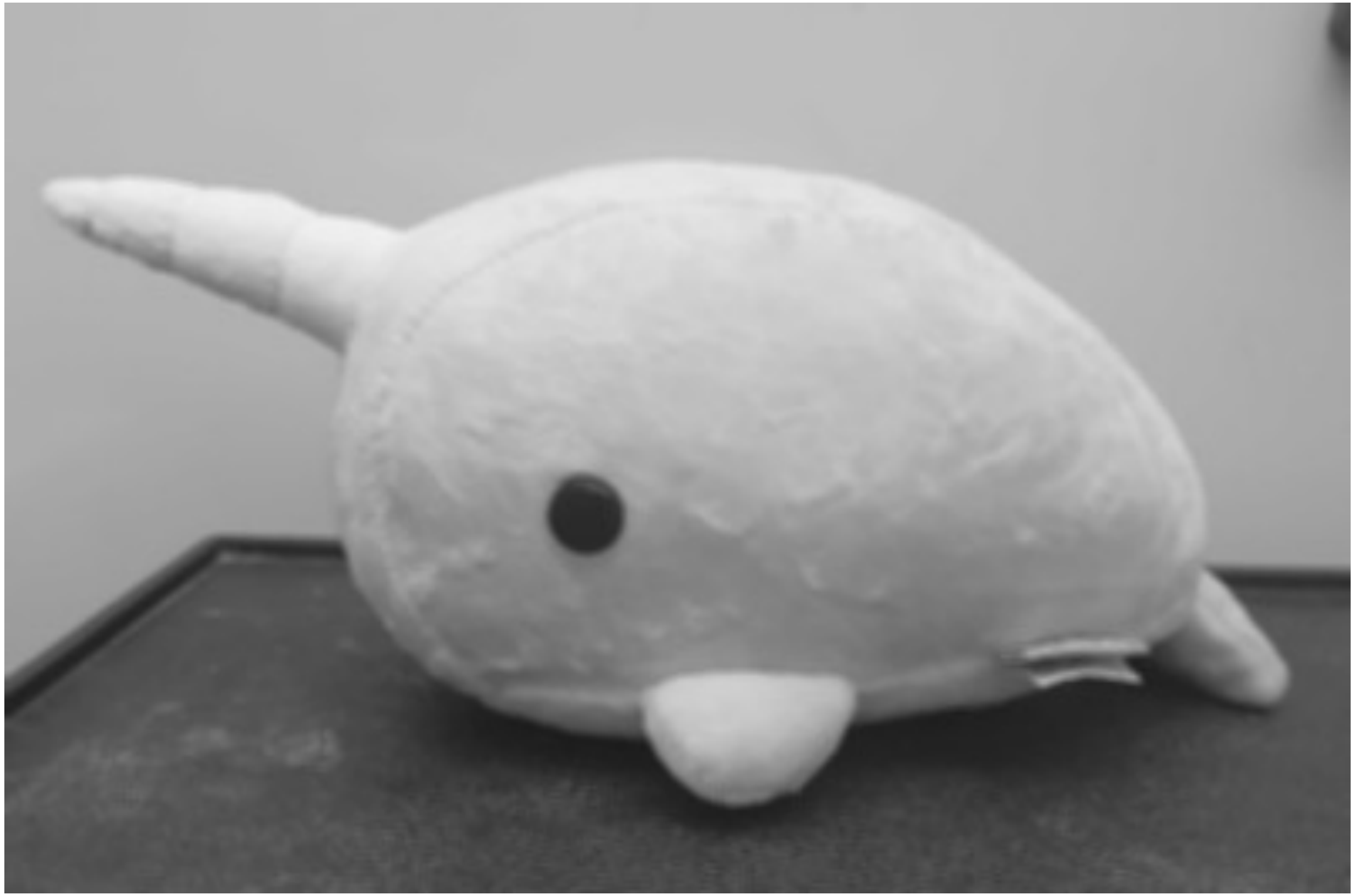
Original Image



Convolution Examples

Smoothing Convolution

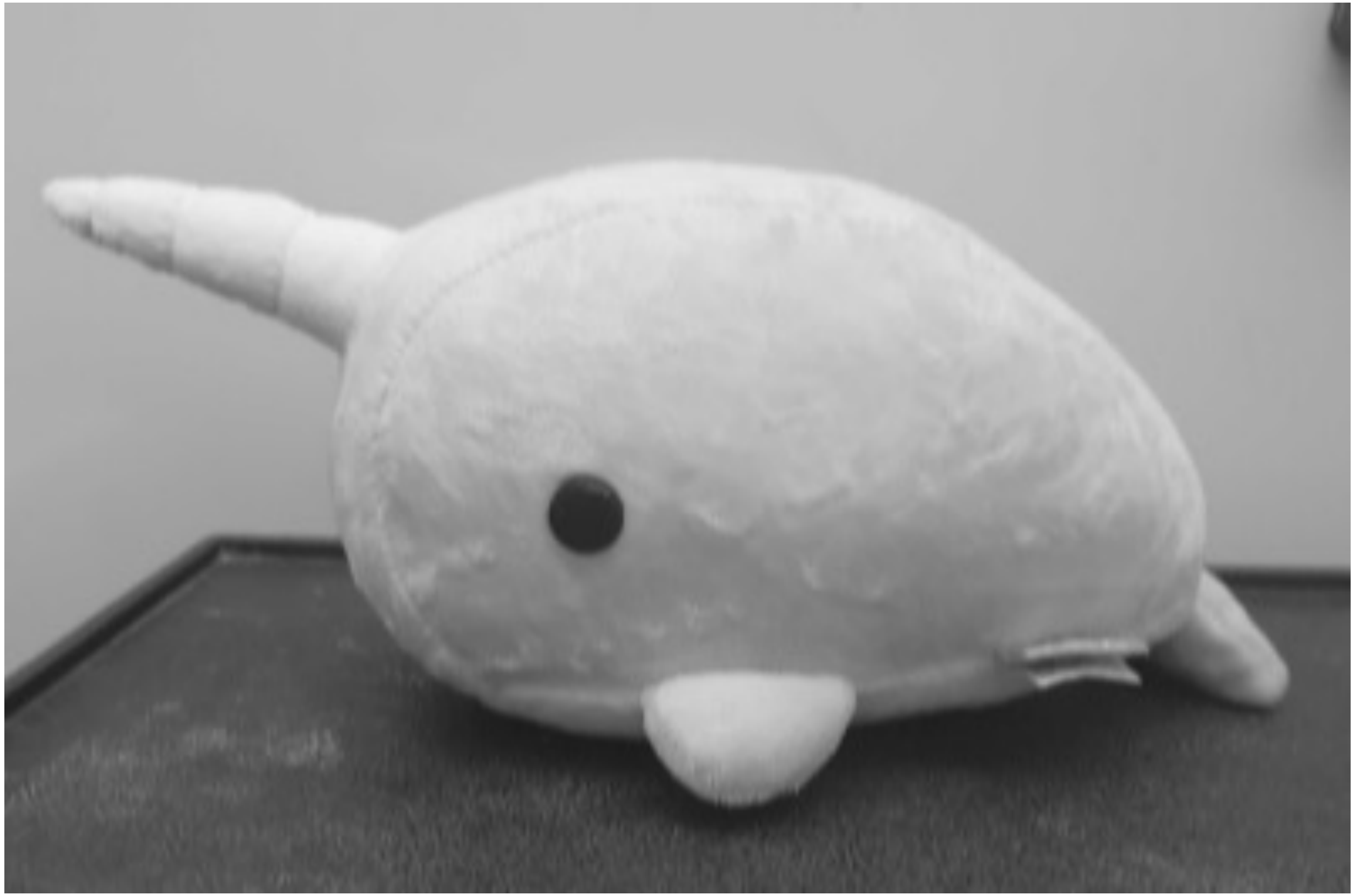
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9



Convolution Examples

Gaussian Blur

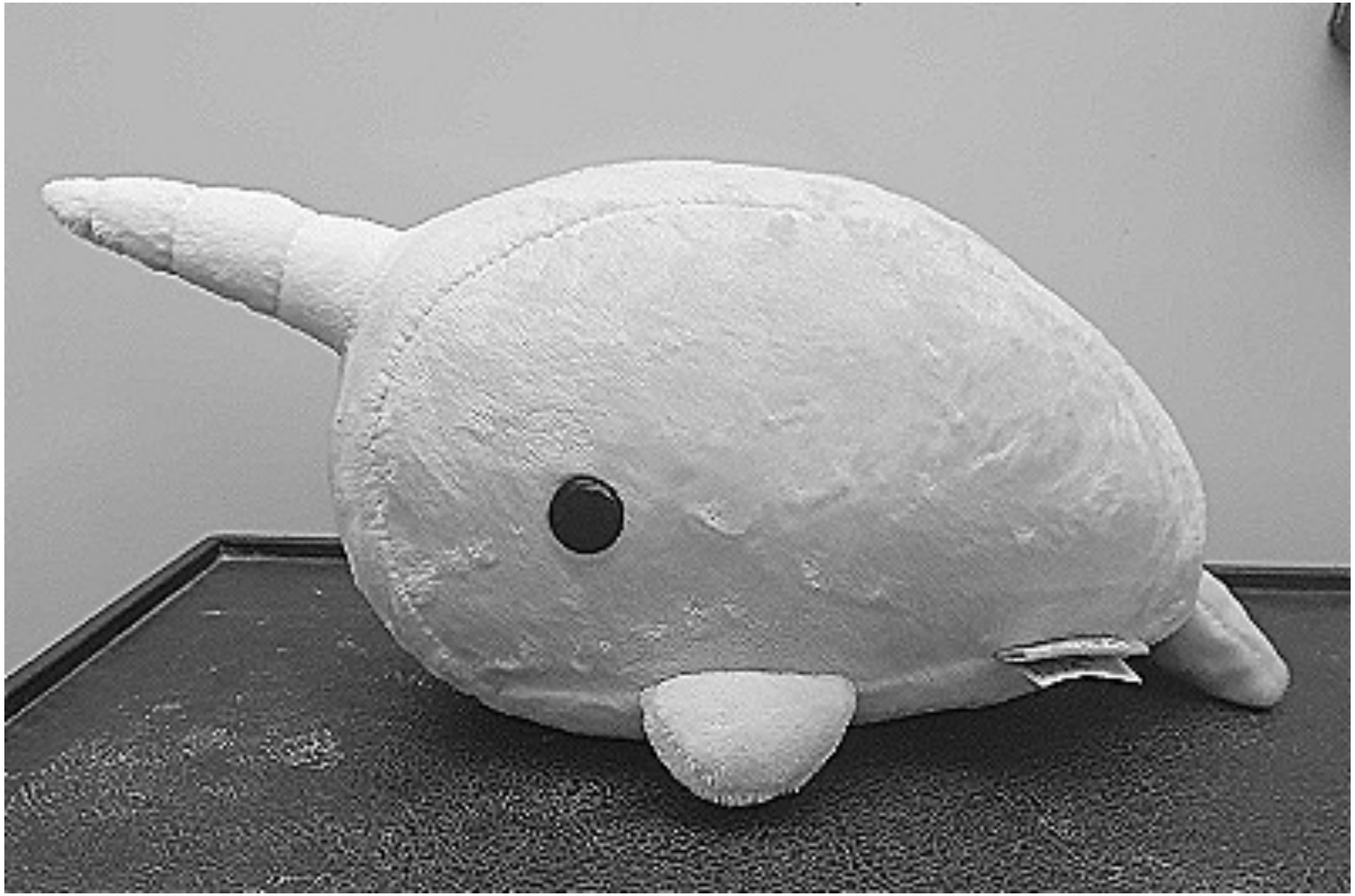
.01	.04	.06	.04	.01
.04	.19	.25	.19	.04
.06	.25	.37	.25	.06
.04	.19	.25	.19	.04
.01	.04	.06	.04	.01



Convolution Examples

Sharpening
Kernel

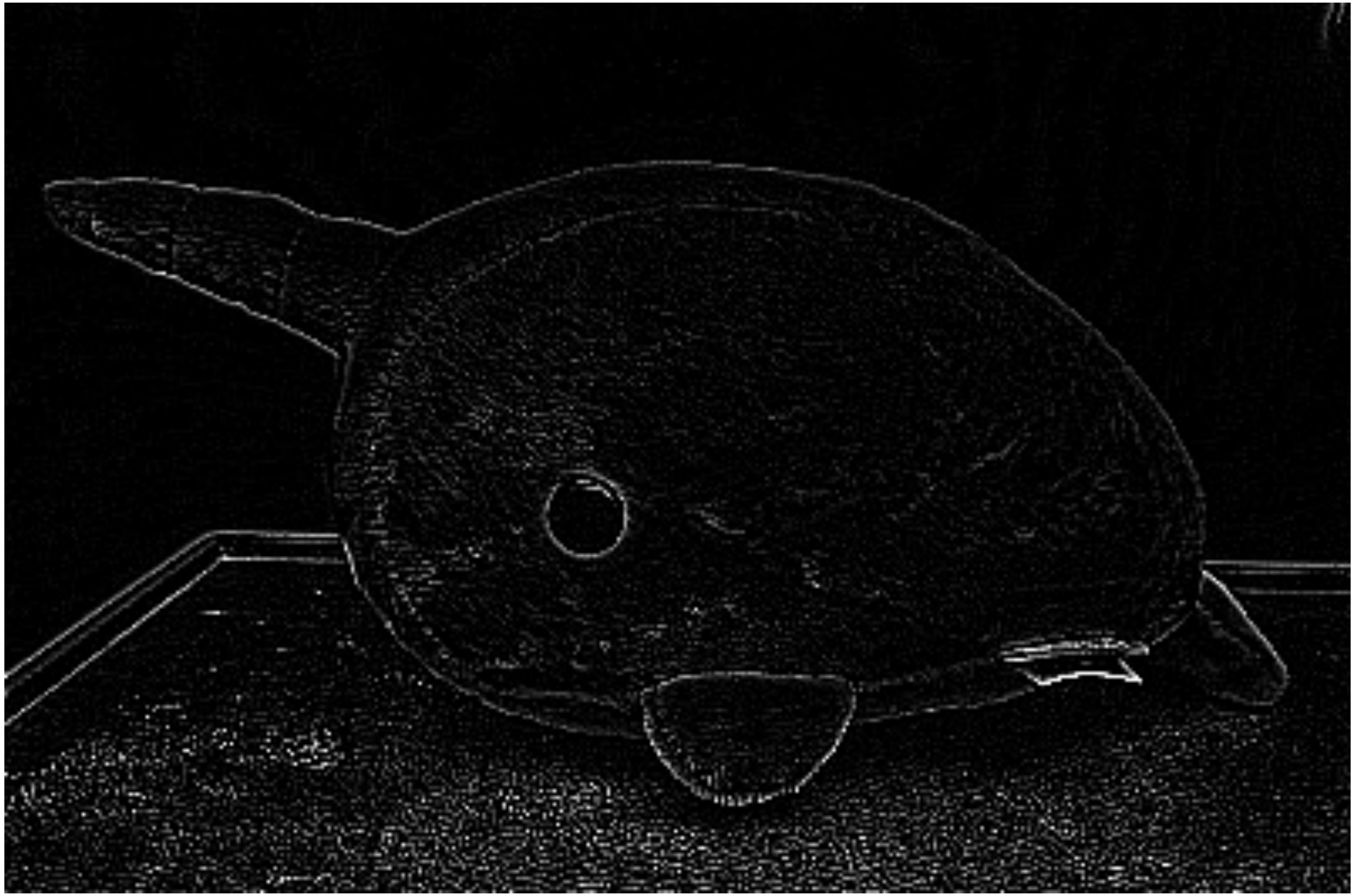
0	-1	0
-1	5	-1
0	-1	0



Convolution Examples

Edge Detector

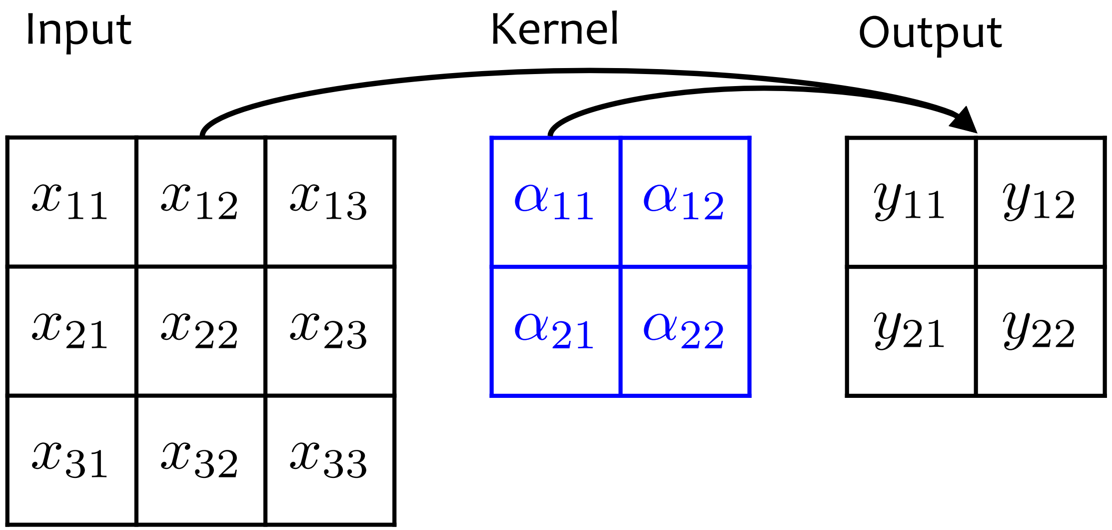
-1	-1	-1
-1	8	-1
-1	-1	-1



2D Convolution

- Basic idea:
  - Pick a 2x2 matrix F of weights (called a kernel or convolution matrix)
  - 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

Example: 1 input channel, 1 output channel



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

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

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

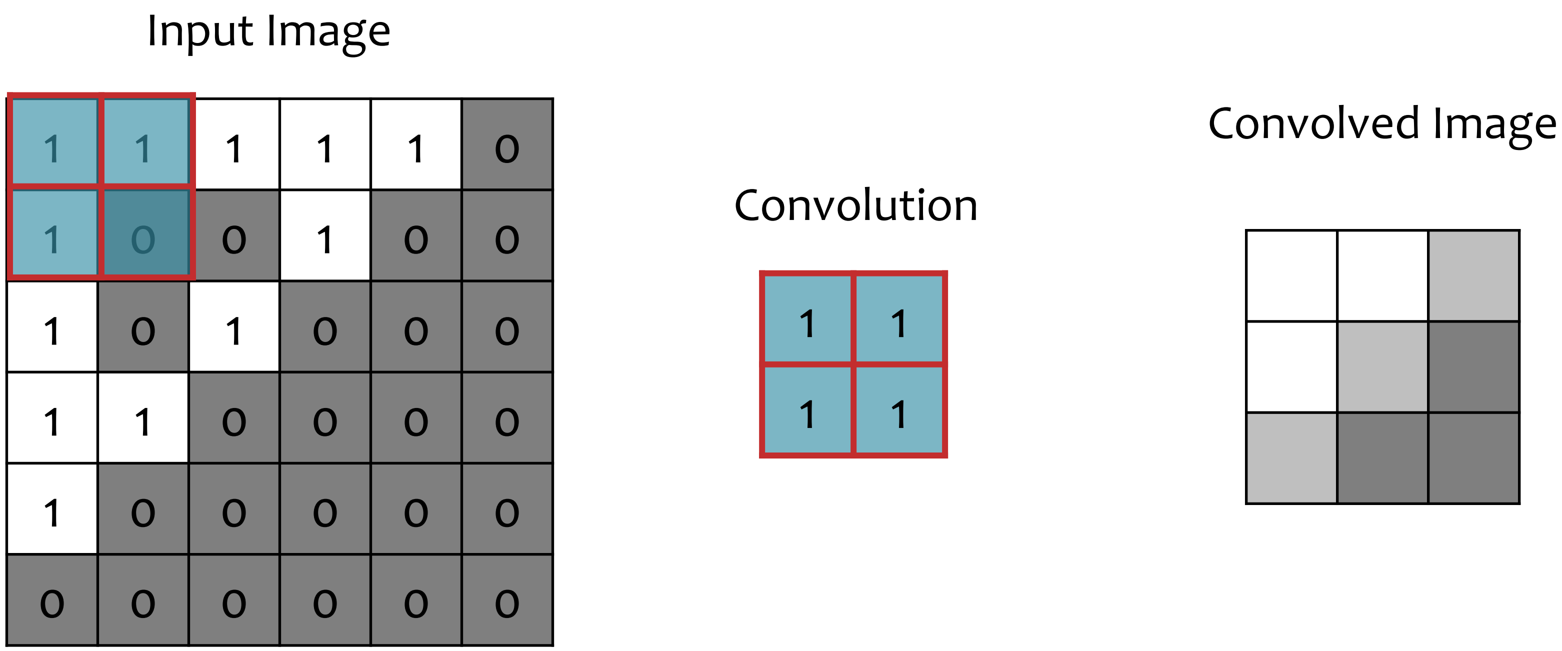
$$y_{22} = \alpha_{11}x_{22} + \alpha_{12}x_{23} + \alpha_{21}x_{32} + \alpha_{22}x_{33} + \alpha_0$$

Slide adapted from William Cohen

DOWNSAMPLING

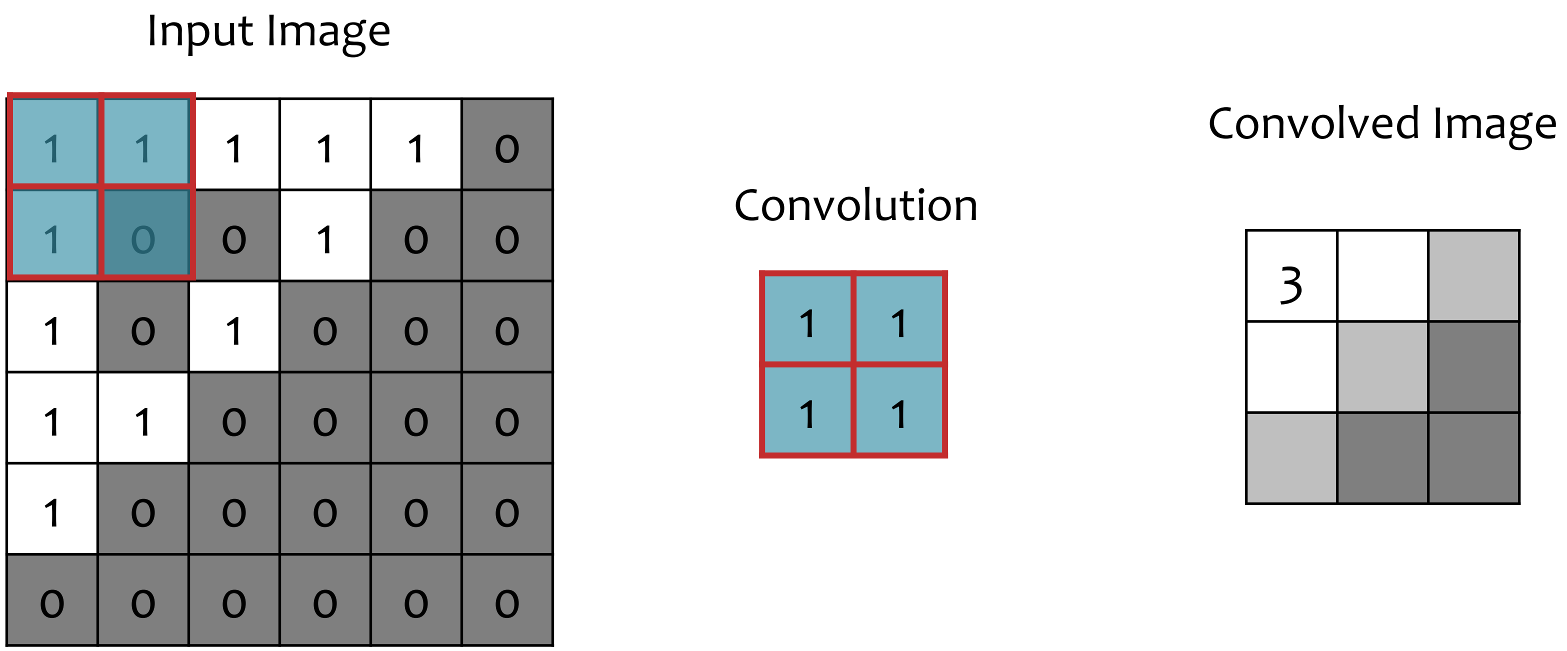
Downsampling

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



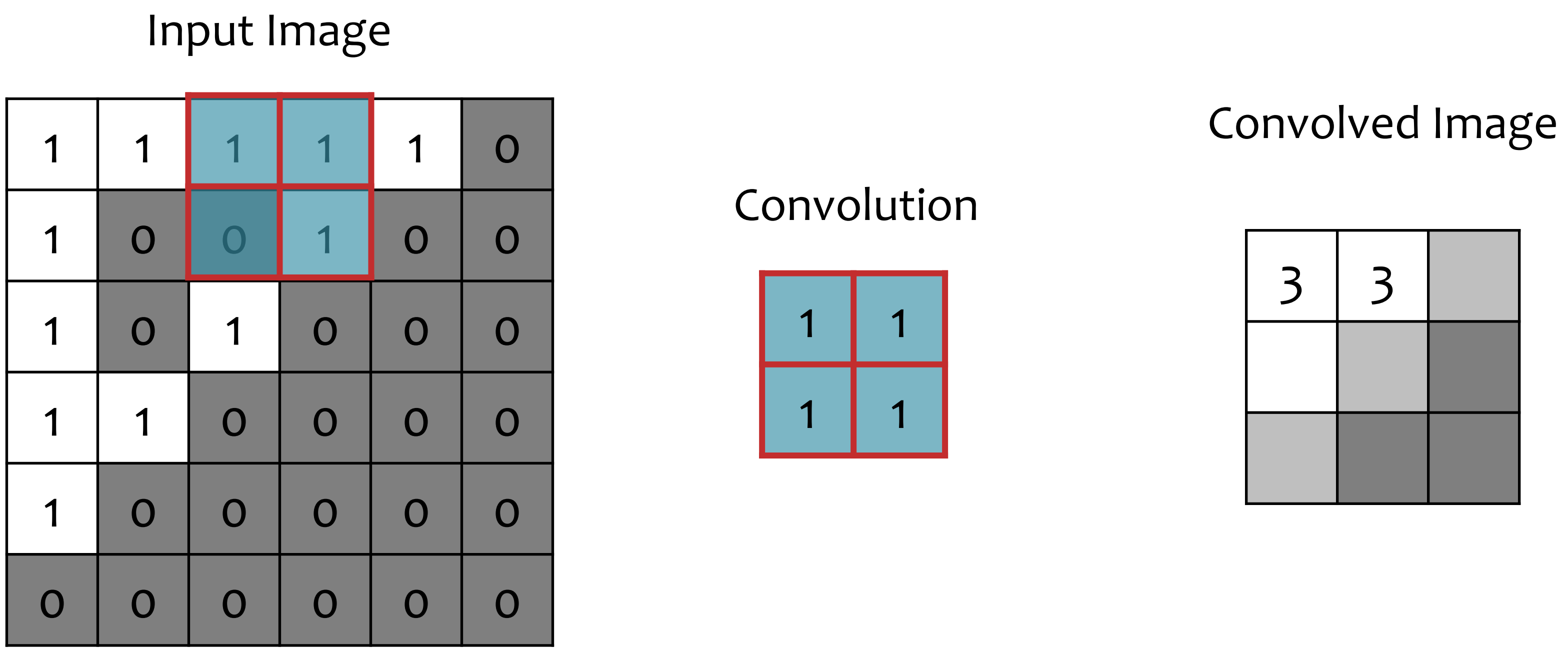
Downsampling

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



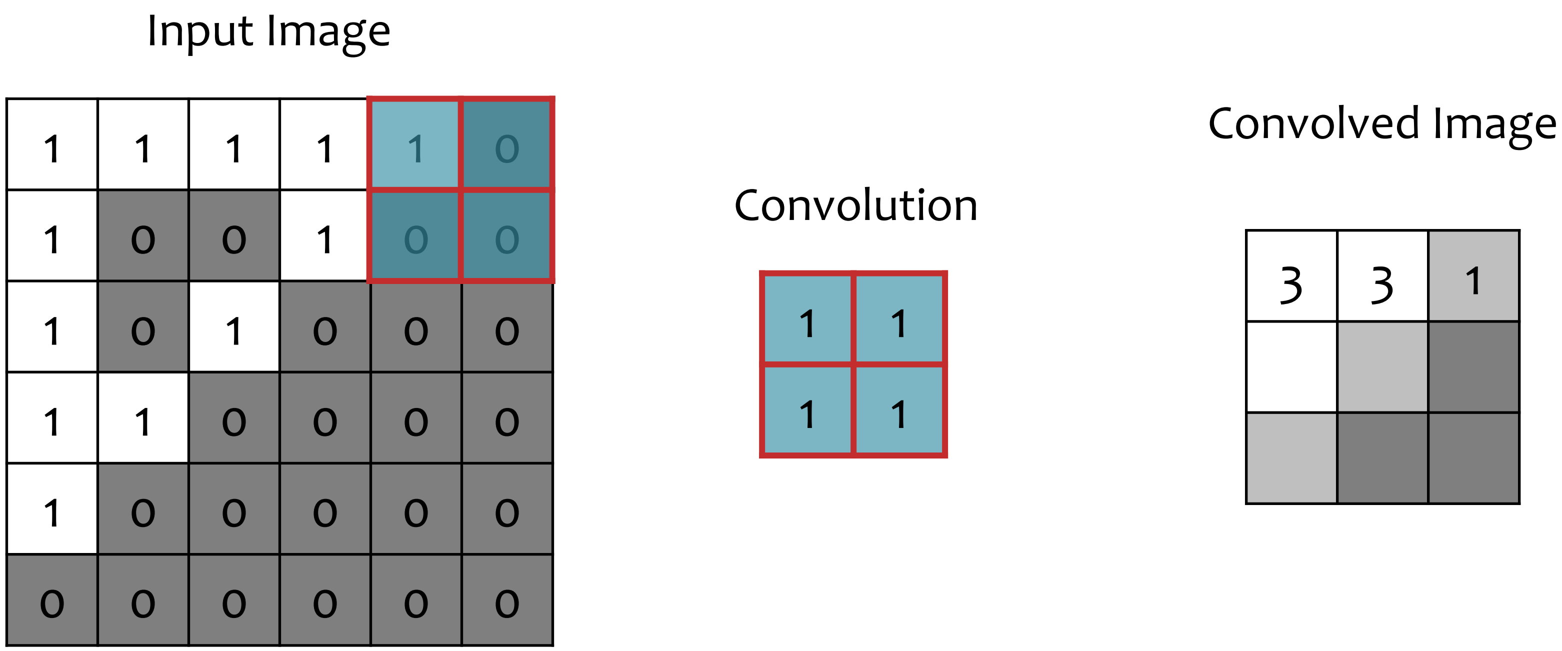
Downsampling

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



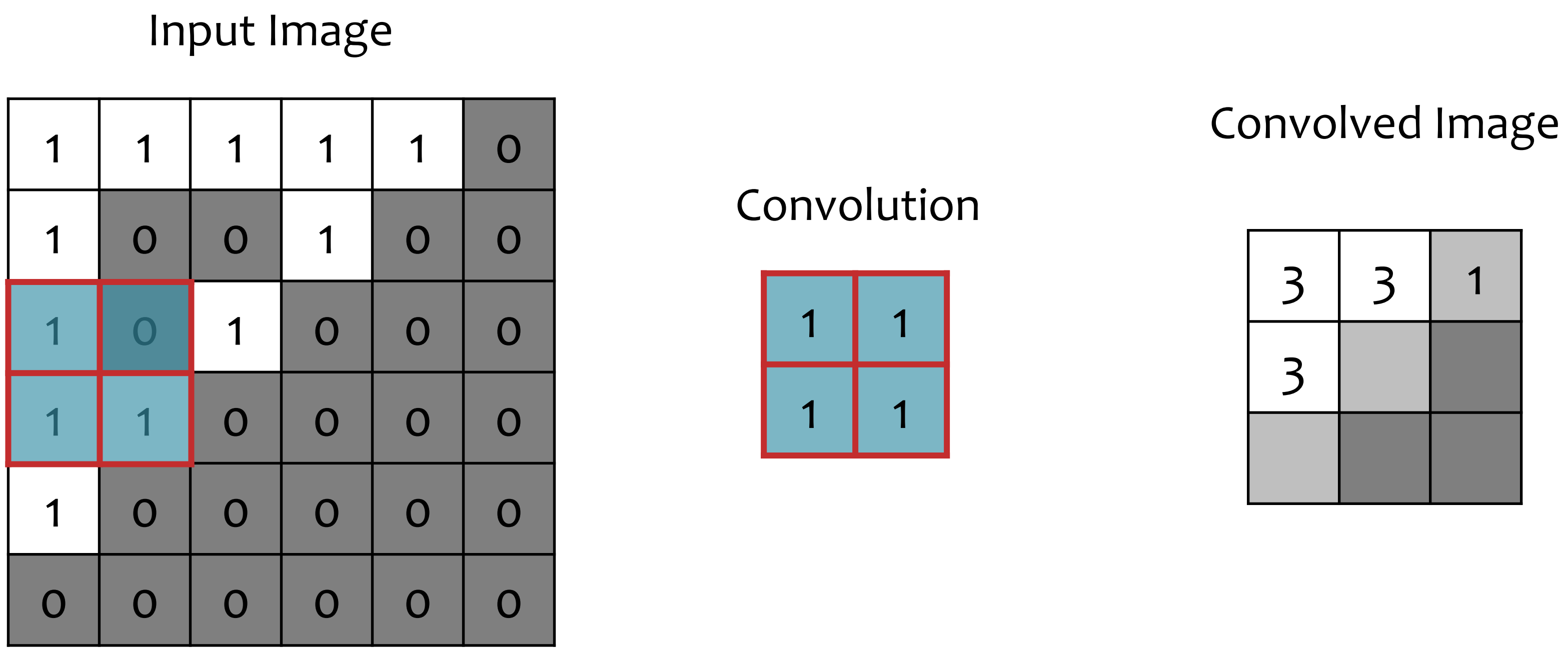
Downsampling

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



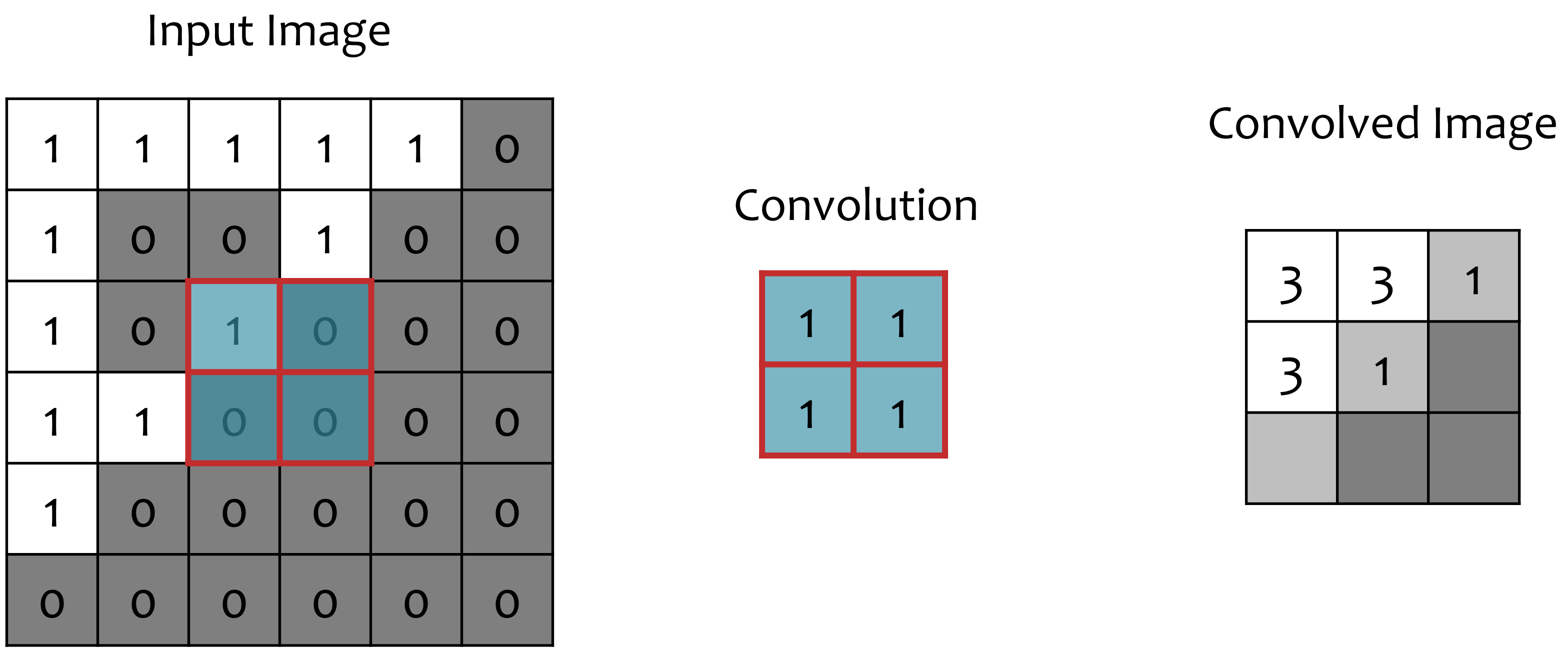
Downsampling

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



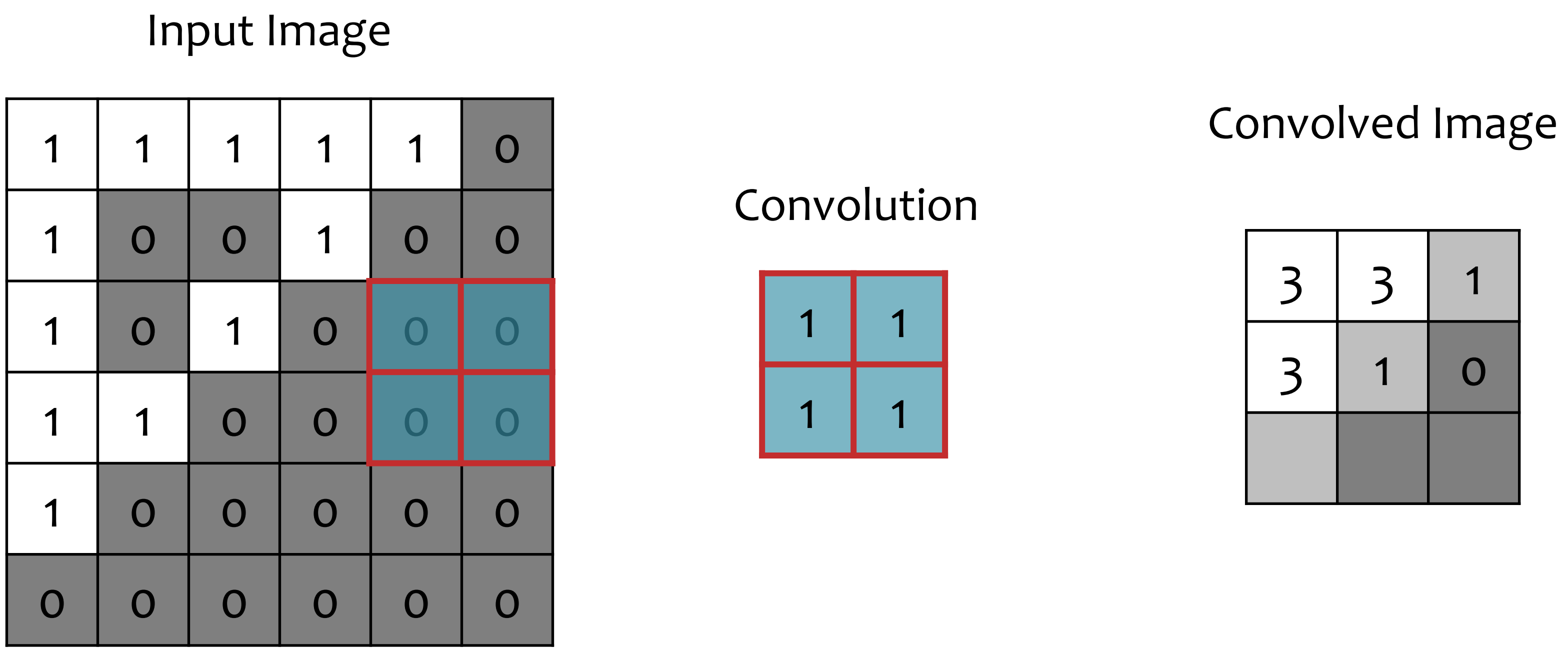
Downsampling

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



Downsampling

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



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

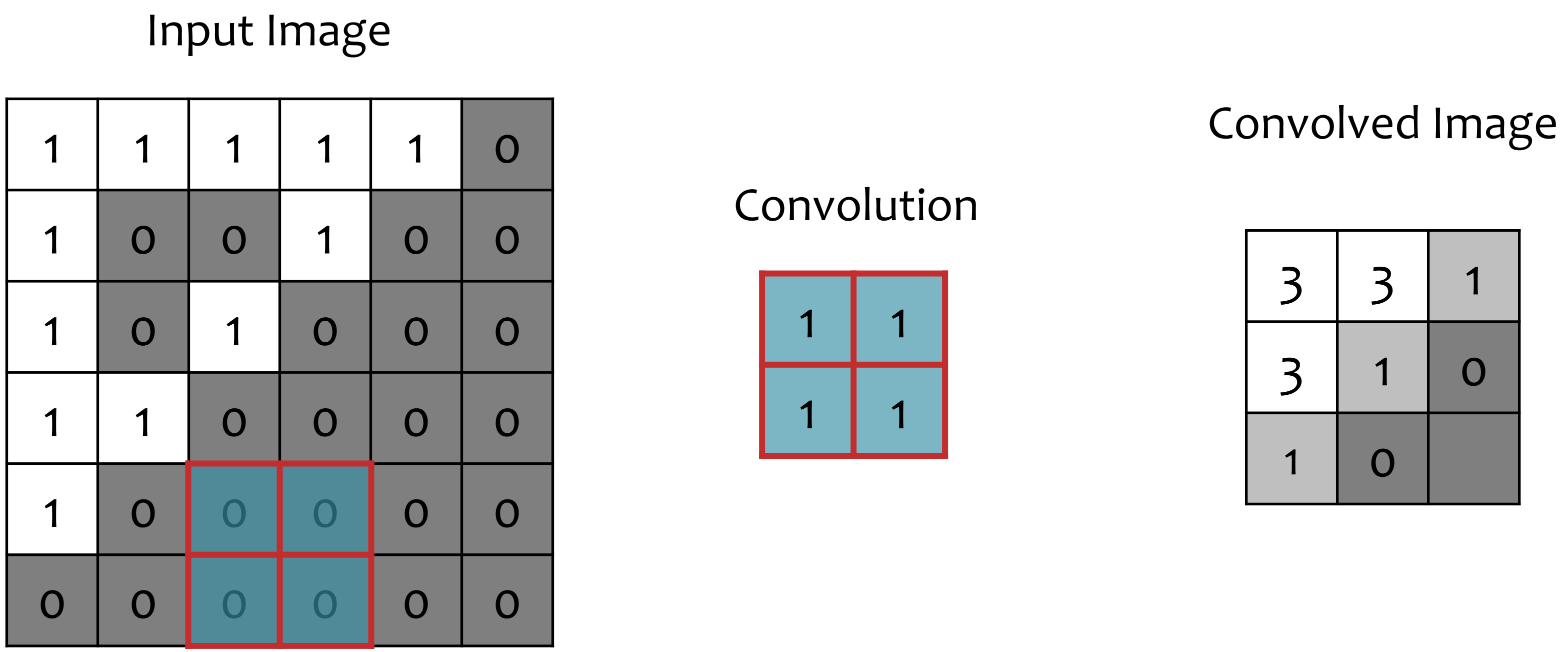
1	1
1	1

Convolved Image

3	3	1
3	1	0
1		

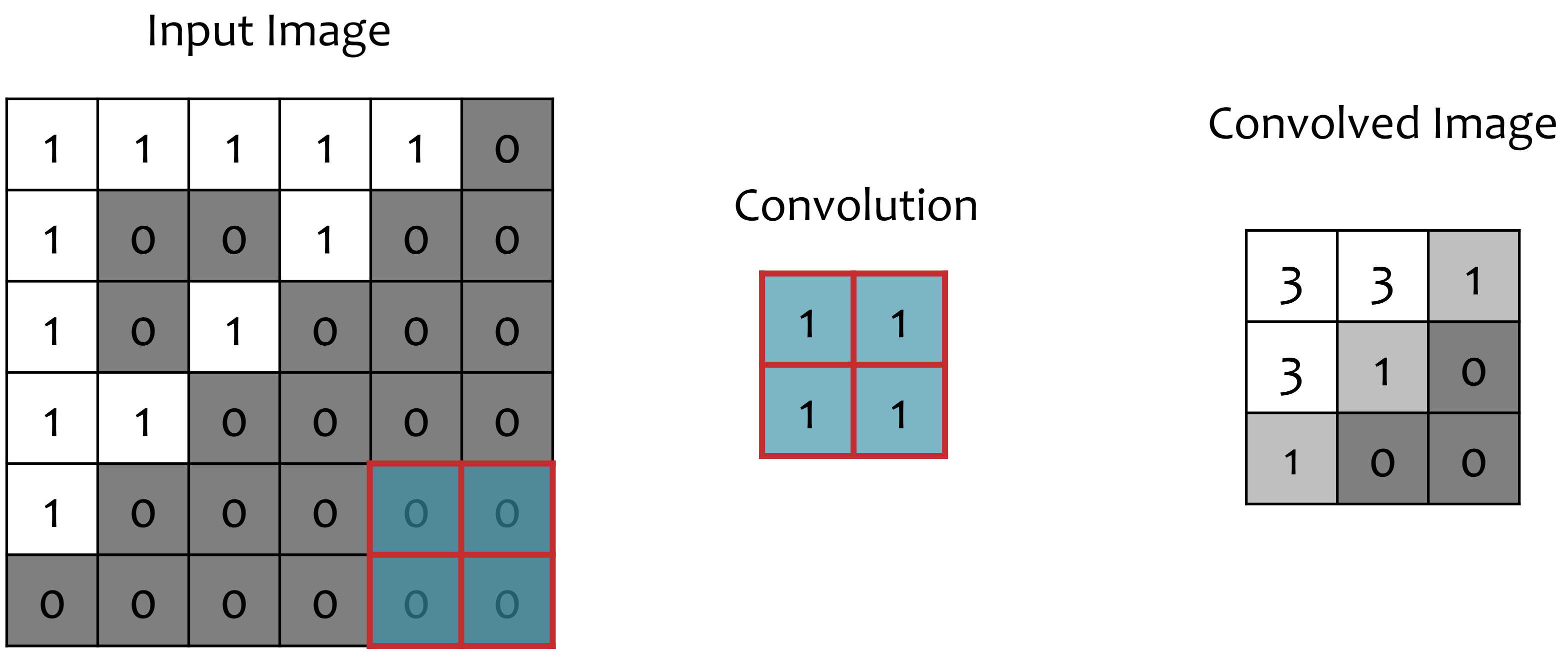
Downsampling

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



Downsampling

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



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:

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

2. Choose each of these:
   - Decision function

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

   - Loss function

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

3. Define goal:

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

4. Train with SGD:
(take small steps opposite the gradient)

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

Background

A Recipe for Machine Learning

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

2. Choose each of these:
   - Decision function

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

   - Loss function

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

Train with SGD:

...ke small steps opposite the gradient)

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

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

θ_{11}	θ_{12}	θ_{13}
θ_{21}	θ_{22}	θ_{23}
θ_{31}	θ_{32}	θ_{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

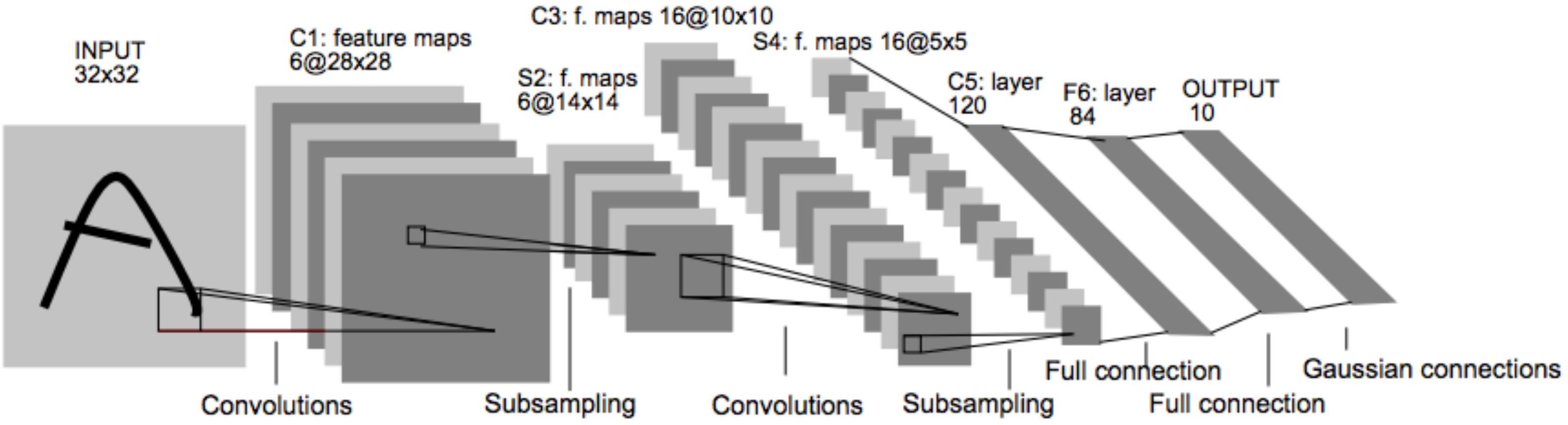


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

TRAINING CNNS

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:
   - Decision function

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

   - Loss function

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

3. Define goal:

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

4. Train with SGD:

(take small steps opposite the gradient)

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

Background

A Recipe for Machine Learning

1. Given training data:

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

3. Define goal:

2. Choose each of th

– Decision function

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

– Loss function

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

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

opp site the gradient)

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

SGD for CNNs

Example: Simple CNN Architecture

Given $\mathbf{x}, \mathbf{y}^*$ and parameters $\boldsymbol{\theta} = [\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{W}]$

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

Algorithm 1 Stochastic Gradient Descent (SGD)

1: Initialize $\boldsymbol{\theta}$
2: **while** not converged **do**
3: Sample $i \in \{1, \ldots, N\}$
4: Forward: $\mathbf{y} = h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$,
5: $J(\boldsymbol{\theta}) = \ell(\mathbf{y}, \mathbf{y}^{(i)})$
6: Backward: Compute $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
7: Update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

LAYERS OF A CNN

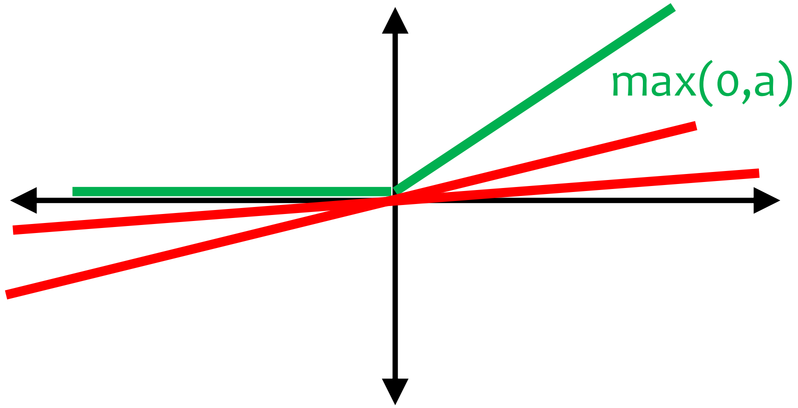
ReLU Layer

Output: $\mathbf{y} \in \mathbb{R}^K$

Forward:

$$\mathbf{y} = \sigma(\mathbf{x}), \text{ element-wise}$$
$$\sigma(a) = \max(0, a)$$

Input: $\mathbf{x} \in \mathbb{R}^K$



Input: $\frac{\partial J}{\partial \mathbf{y}} \in \mathbb{R}^K$

Backward: for each j,

$$\frac{\partial J}{\partial x_j} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial x_j}$$

where

$$\frac{\partial y_j}{\partial x_j} = \begin{cases} 1 & \text{if } x_j > 0 \\ 0 & \text{otherwise} \end{cases}$$

subderivative

Output: $\frac{\partial J}{\partial \mathbf{x}} \in \mathbb{R}^K$

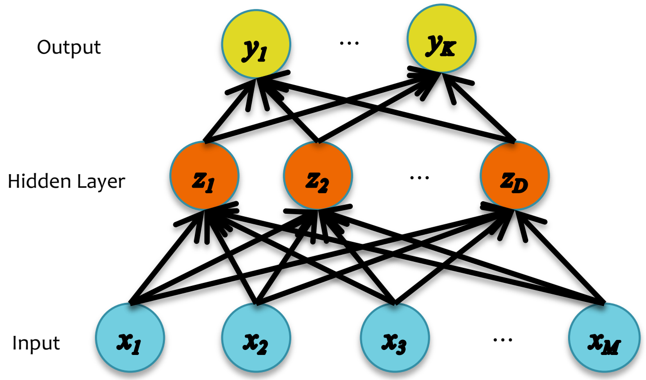
Softmax Layer

Output: $\mathbf{y} \in \mathbb{R}^K$

Forward: for each i,

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

Input: $\mathbf{x} \in \mathbb{R}^K$



Input: $\frac{\partial J}{\partial \mathbf{y}} \in \mathbb{R}^K$

Backward: for each j,

$$\frac{\partial J}{\partial x_j} = \sum_{i=1}^{K} \frac{\partial J}{\partial y_i} \frac{\partial y_i}{\partial x_j}$$

where

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

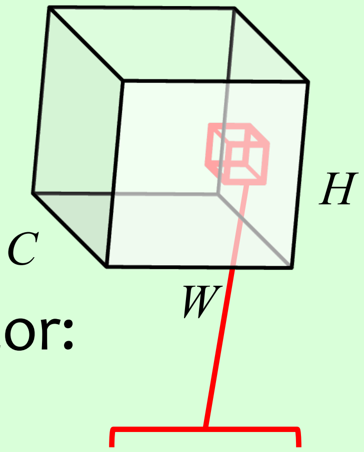
Output: $\frac{\partial J}{\partial \mathbf{x}} \in \mathbb{R}^K$

Fully-Connected Layer (3D input)

Forward:

1. suppose input is a 3D tensor:

$$\mathbf{x} =$$



2. flatten out tensor into a vector:

$$\hat{\mathbf{x}} = [x_1, \ldots, x_{(i \times j \times k)}, \ldots, x_{(C \times H \times W)}]$$

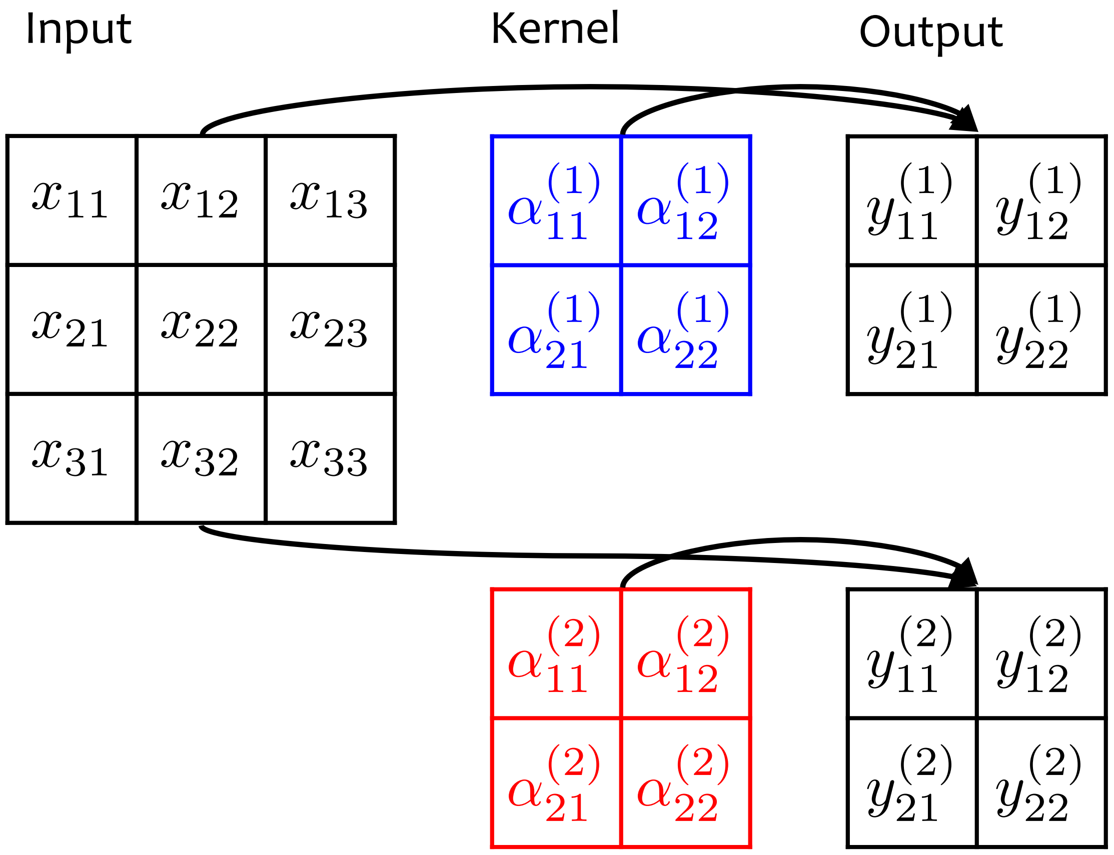
3. then push that vector through a standard linear layer:

$$\mathbf{y} = \boldsymbol{\alpha}^T \hat{\mathbf{x}} + \boldsymbol{\alpha}_0 \quad \text{where } \boldsymbol{\alpha} \in \mathbb{R}^{A \times B}, \quad \boldsymbol{\alpha}_0 \in \mathbb{R}^B$$

$$|\hat{\mathbf{x}}| \in \mathbb{R}^A, \quad |\mathbf{y}| \in \mathbb{R}^B$$

2D Convolution

Example: 1 input channel, 2 output channels



$$y_{11}^{(1)} = \alpha_{11}^{(1)} x_{11} + \alpha_{12}^{(1)} x_{12} + \alpha_{21}^{(1)} x_{21} + \alpha_{22}^{(1)} x_{22} + \alpha_0^{(1)}$$
$$y_{12}^{(1)} = \alpha_{11}^{(1)} x_{12} + \alpha_{12}^{(1)} x_{13} + \alpha_{21}^{(1)} x_{22} + \alpha_{22}^{(1)} x_{23} + \alpha_0^{(1)}$$
$$y_{21}^{(1)} = \alpha_{11}^{(1)} x_{21} + \alpha_{12}^{(1)} x_{22} + \alpha_{21}^{(1)} x_{31} + \alpha_{22}^{(1)} x_{32} + \alpha_0^{(1)}$$
$$y_{22}^{(1)} = \alpha_{11}^{(1)} x_{22} + \alpha_{12}^{(1)} x_{23} + \alpha_{21}^{(1)} x_{32} + \alpha_{22}^{(1)} x_{33} + \alpha_0^{(1)}$$

$$y_{11}^{(2)} = \alpha_{11}^{(2)} x_{11} + \alpha_{12}^{(2)} x_{12} + \alpha_{21}^{(2)} x_{21} + \alpha_{22}^{(2)} x_{22} + \alpha_0^{(2)}$$
$$y_{12}^{(2)} = \alpha_{11}^{(2)} x_{12} + \alpha_{12}^{(2)} x_{13} + \alpha_{21}^{(2)} x_{22} + \alpha_{22}^{(2)} x_{23} + \alpha_0^{(2)}$$
$$y_{21}^{(2)} = \alpha_{11}^{(2)} x_{21} + \alpha_{12}^{(2)} x_{22} + \alpha_{21}^{(2)} x_{31} + \alpha_{22}^{(2)} x_{32} + \alpha_0^{(2)}$$
$$y_{22}^{(2)} = \alpha_{11}^{(2)} x_{22} + \alpha_{12}^{(2)} x_{23} + \alpha_{21}^{(2)} x_{32} + \alpha_{22}^{(2)} x_{33} + \alpha_0^{(2)}$$

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

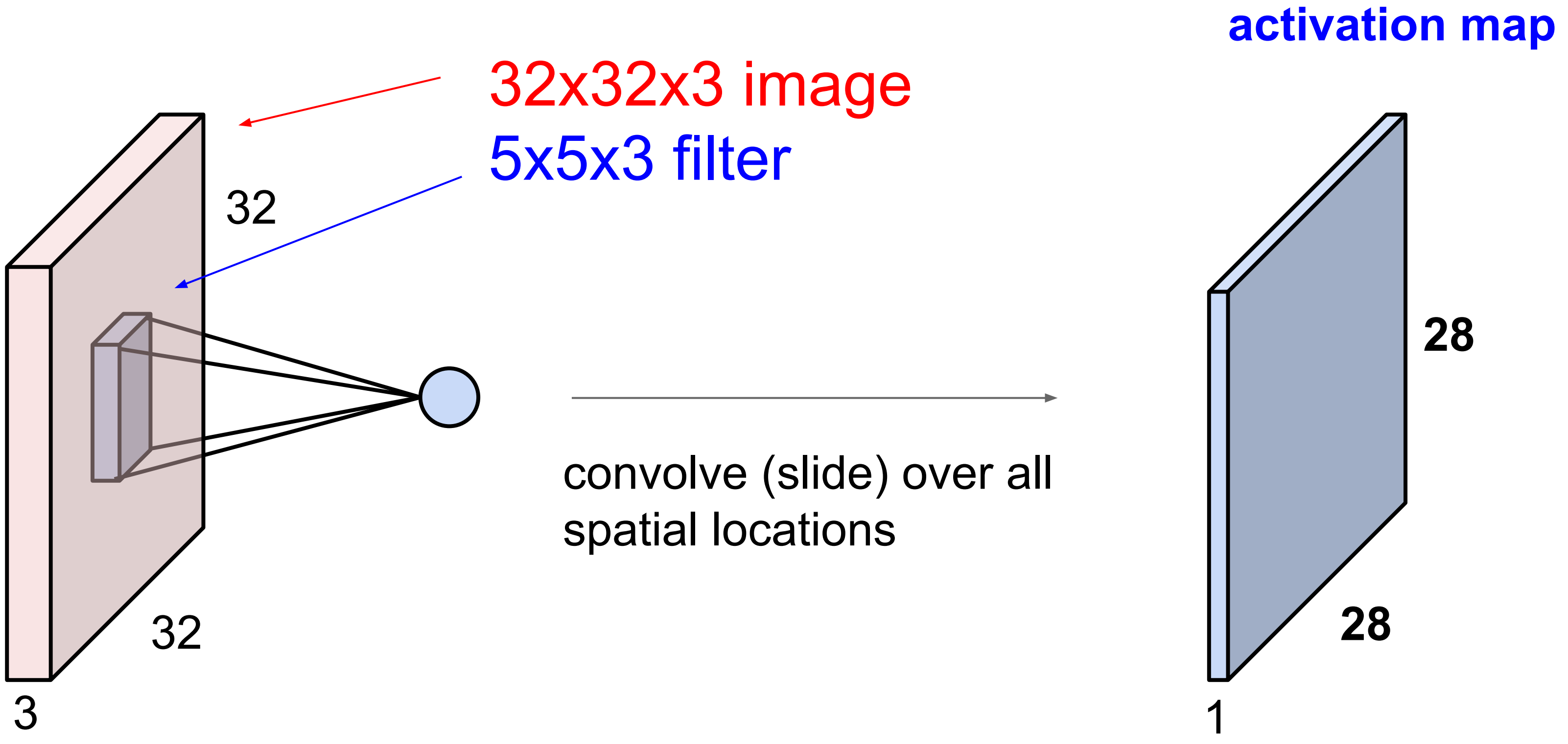


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

Animation of 3D Convolution

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

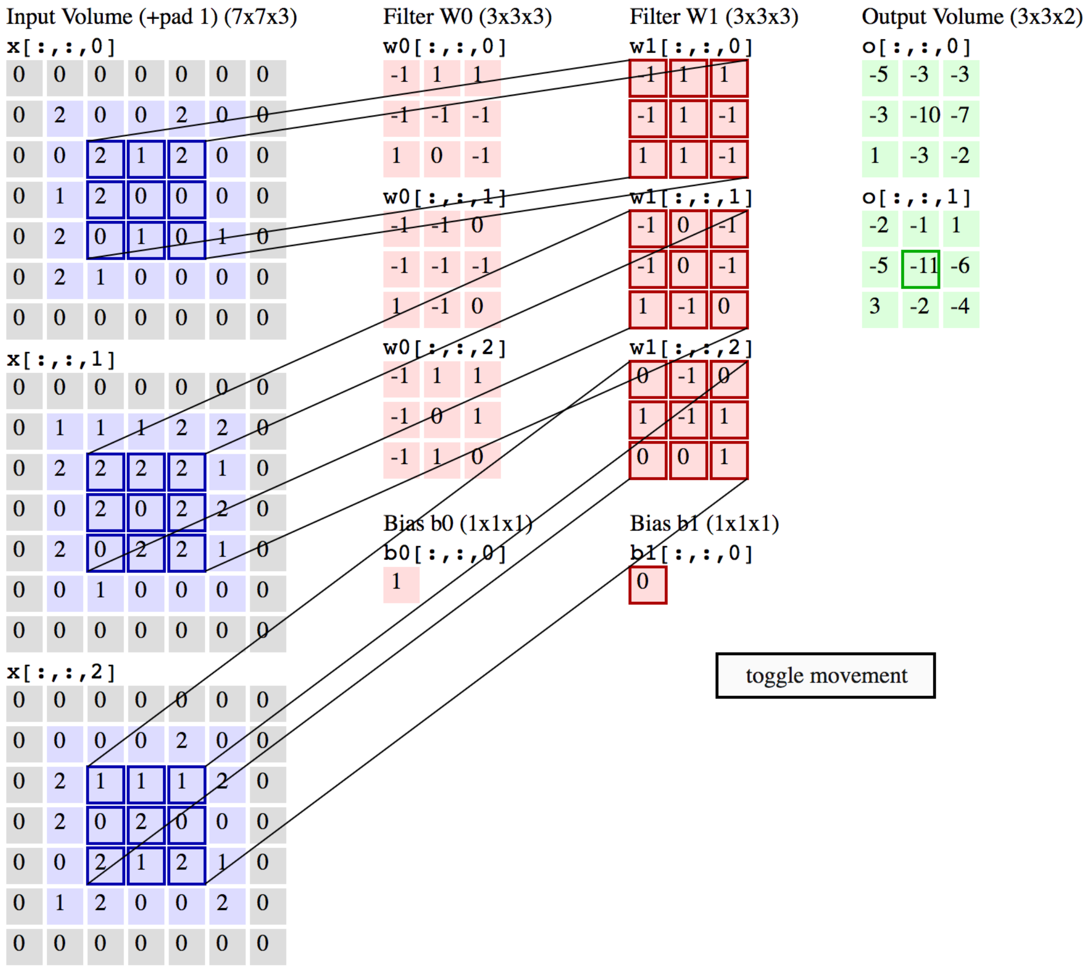
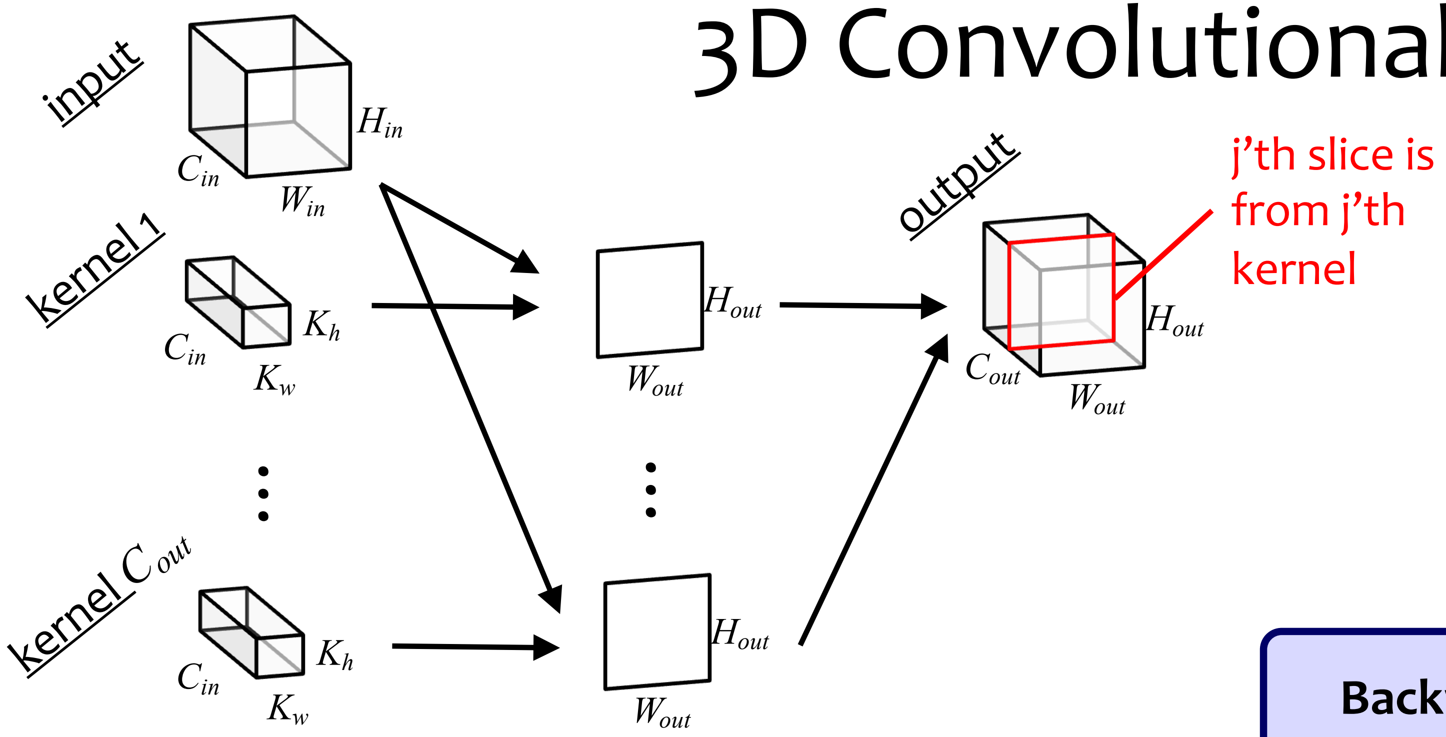
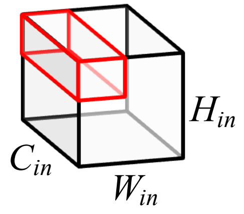


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

3D Convolutional Layer



Convolution in 3D



Forward:

$$y^{(c')}_{h',w'} = \beta^{(c')} + \sum_{c=1}^{C_{\text{in}}} \sum_{m=1}^{K_{\text{h}}} \sum_{n=1}^{K_{\text{w}}} x^{(c)}_{h'+ms,w'+ns} \cdot \alpha^{(c',c)}_{m,n}$$

Backward:

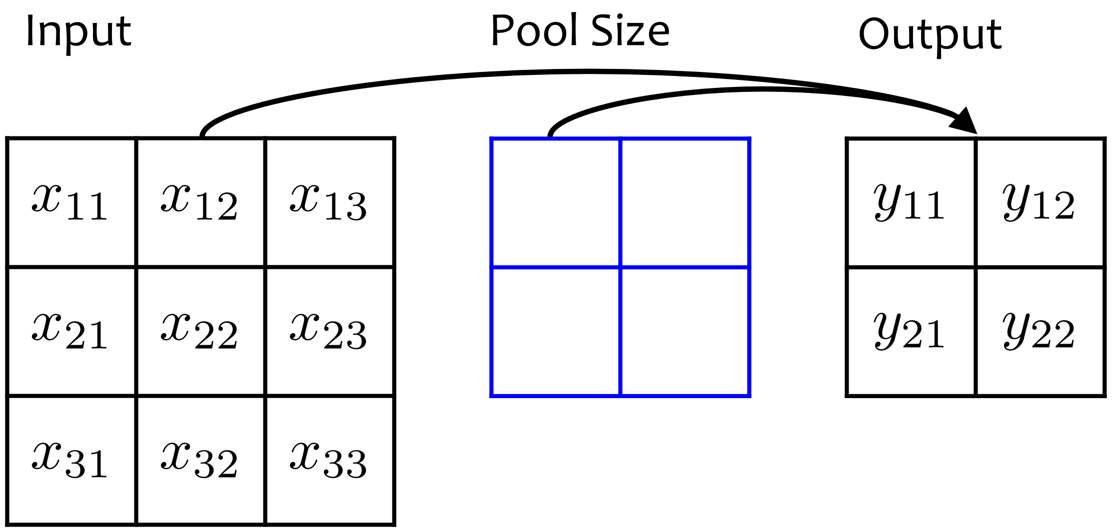
$$\frac{\partial J}{\partial \alpha^{(c',c)}_{m,n}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}} \cdot x^{(c)}_{h'+ms,w'+ns}$$

$$\frac{\partial J}{\partial \beta^{(c')}} = \sum_{h'=1}^{H_{\text{out}}} \sum_{w'=1}^{W_{\text{out}}} \frac{\partial J}{\partial y^{(c')}_{h',w'}}$$

$s \in \mathbb{Z}$ (stride)

Max-Pooling Layer

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

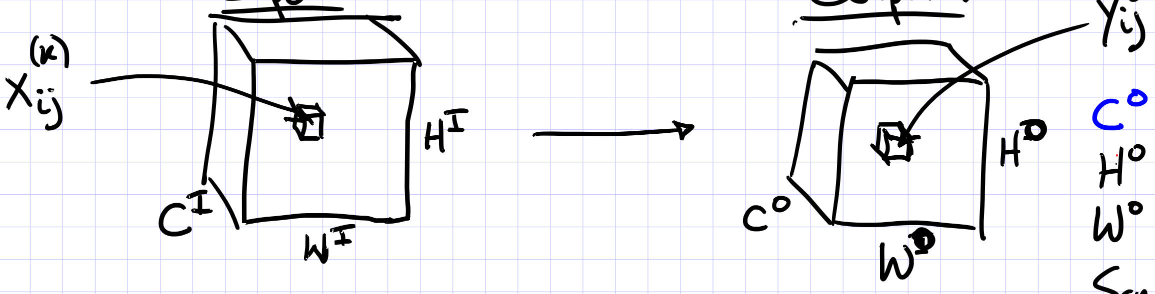


$$y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$
$$y_{12} = \max(x_{12}, x_{13}, x_{22}, x_{23})$$
$$y_{21} = \max(x_{21}, x_{22}, x_{31}, x_{32})$$
$$y_{22} = \max(x_{22}, x_{23}, x_{32}, x_{33})$$

Max-Pooling Layer

Max-Pooling Layer

Input: Output:



$C^O = C^I$

$H^O = \ldots$ } same as conv. layer
$W^O = \ldots$

Same hyperparams too.

Forward:

$$y_{ij}^{(k)} = \max_{\substack{q \in \{1,\ldots,K\} \\ r \in \{1,\ldots,K\}}} x_{mn}^{(k)} \quad \text{where} \quad \begin{aligned} m &= s(i-1)+q \\ n &= s(j-1)+r \end{aligned}$$

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.

$y = \max(a, b)$

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

CNN ARCHITECTURES

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

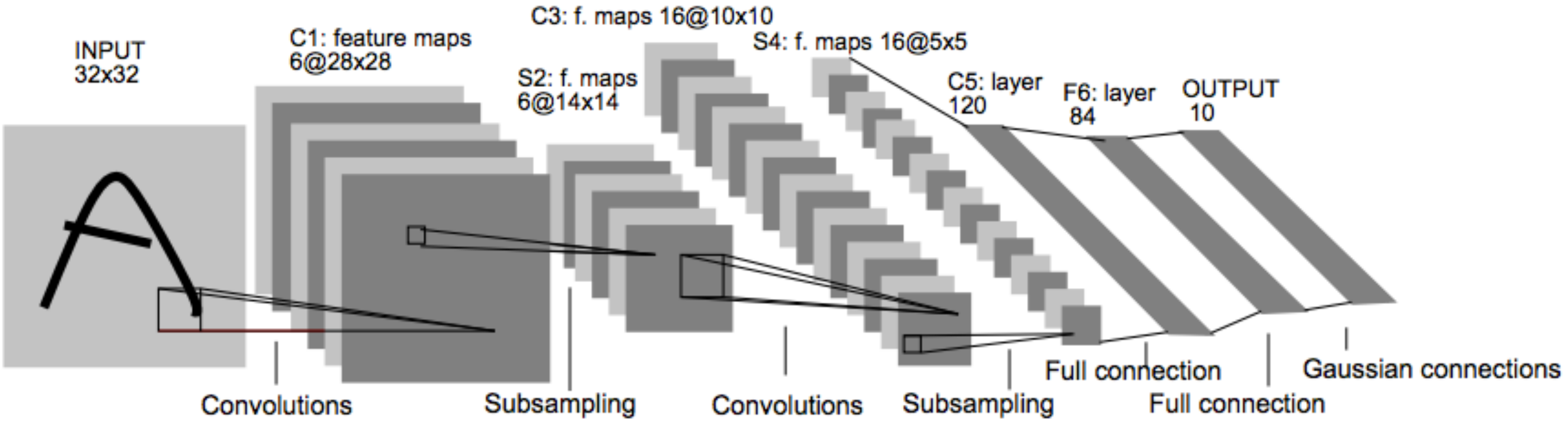


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

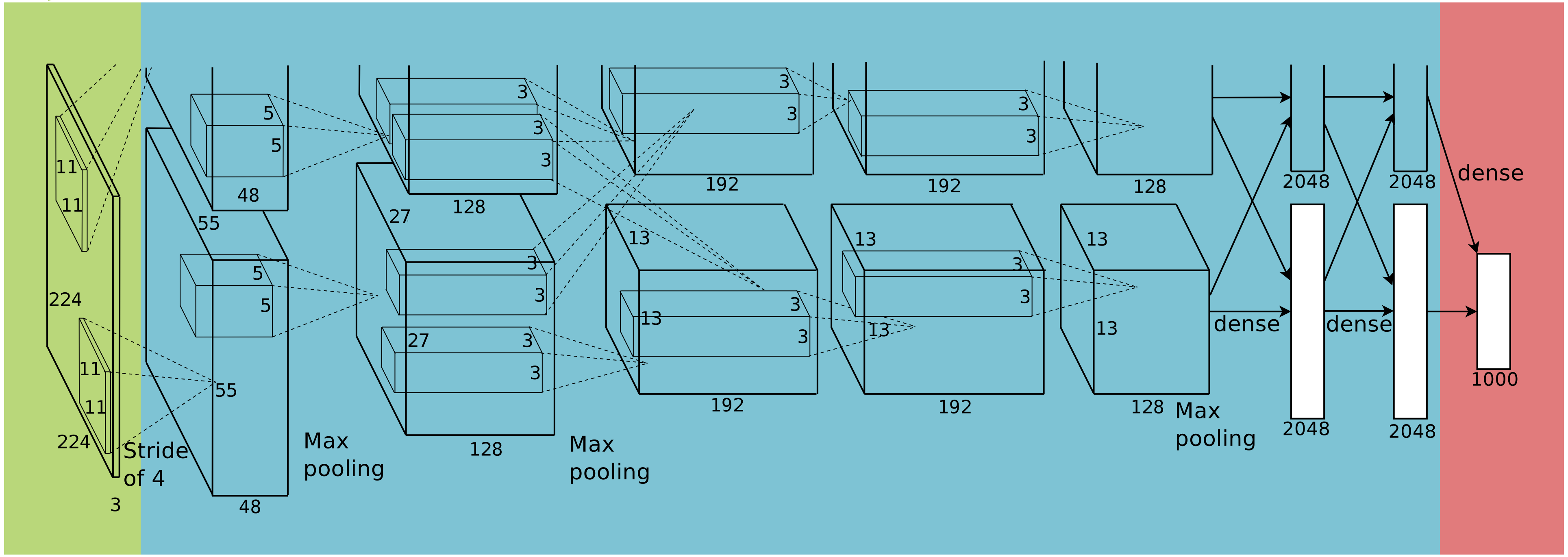
Architecture #2: AlexNet

CNN for Image Classification
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

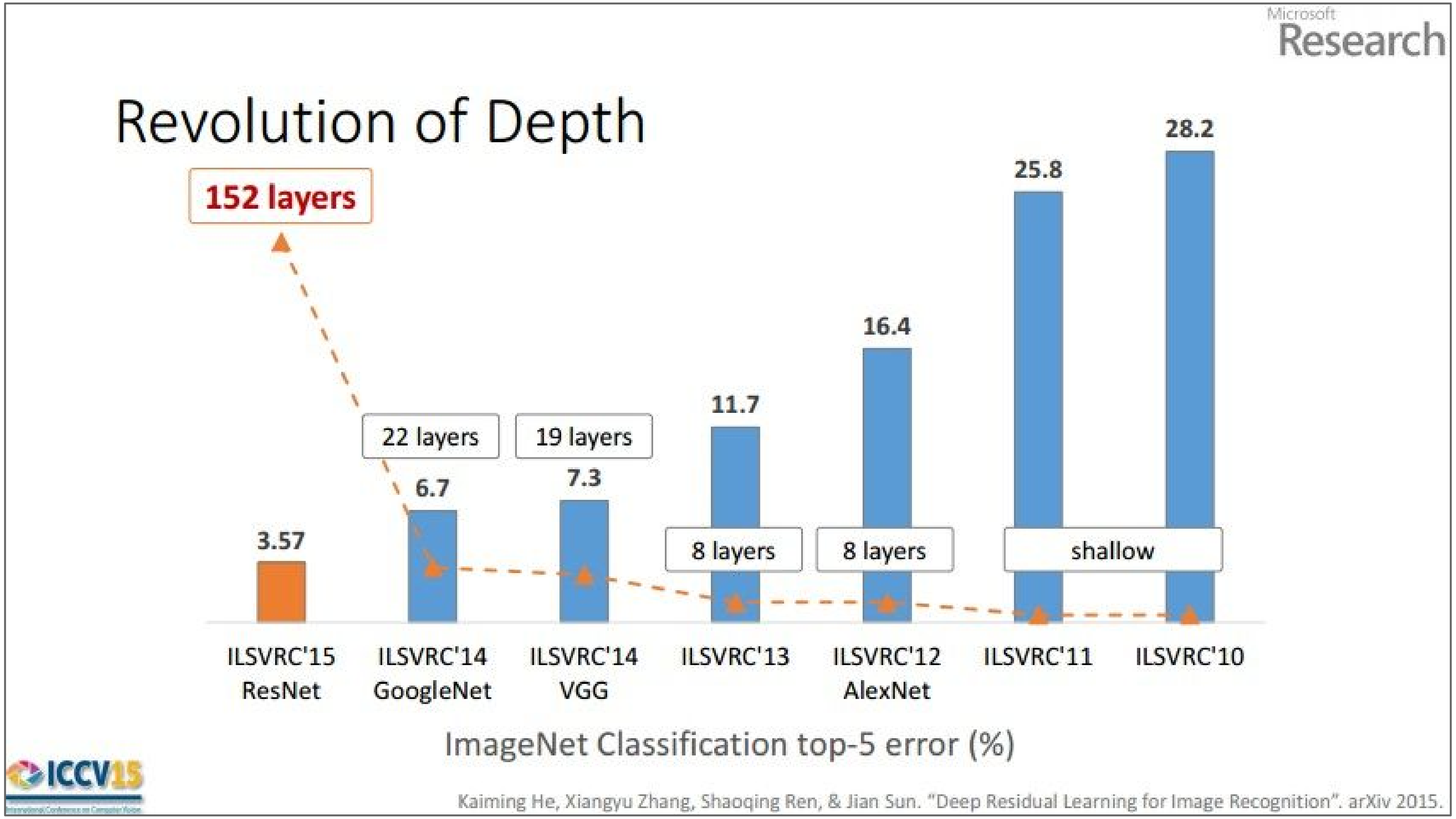
Input image (pixels)

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

1000-way softmax



CNNs for Image Recognition



Slide from Kaiming He

Convolutional Neural Network (CNN)

Typical Architectures

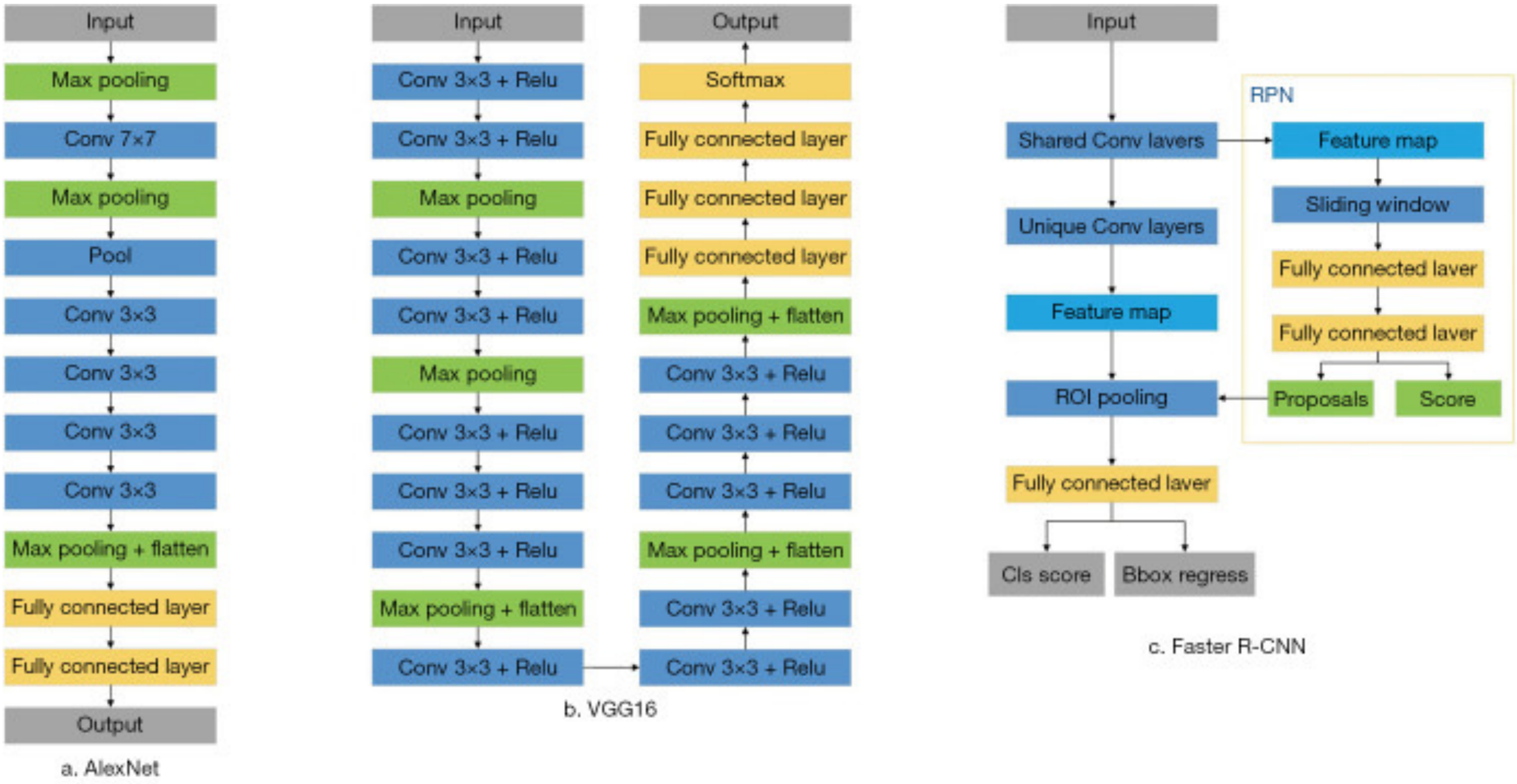


Figure from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7327346/

Convolutional Neural Network (CNN)

Typical Architectures

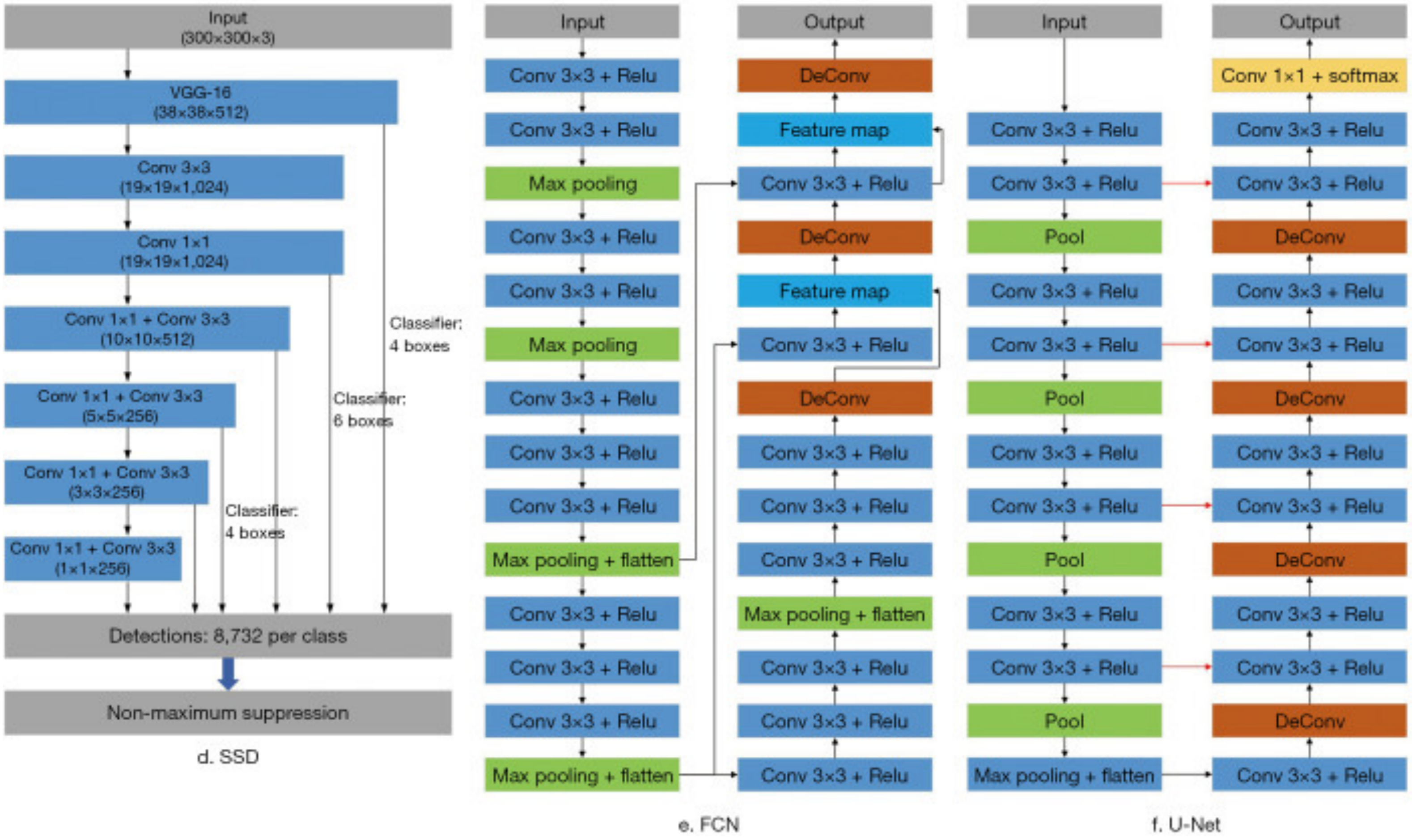
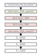


Figure from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7327346/

Convolutional Neural Network (CNN)

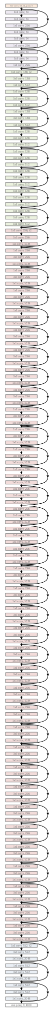
Typical Architectures

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)



Microsoft Research

ICCV15 International Conference on Computer Vision

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 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

θ_{11}	θ_{12}
θ_{21}	θ_{22}

3x3 Convolution

θ_{11}	θ_{12}	θ_{13}
θ_{21}	θ_{22}	θ_{23}
θ_{31}	θ_{32}	θ_{33}

4x4 Convolution

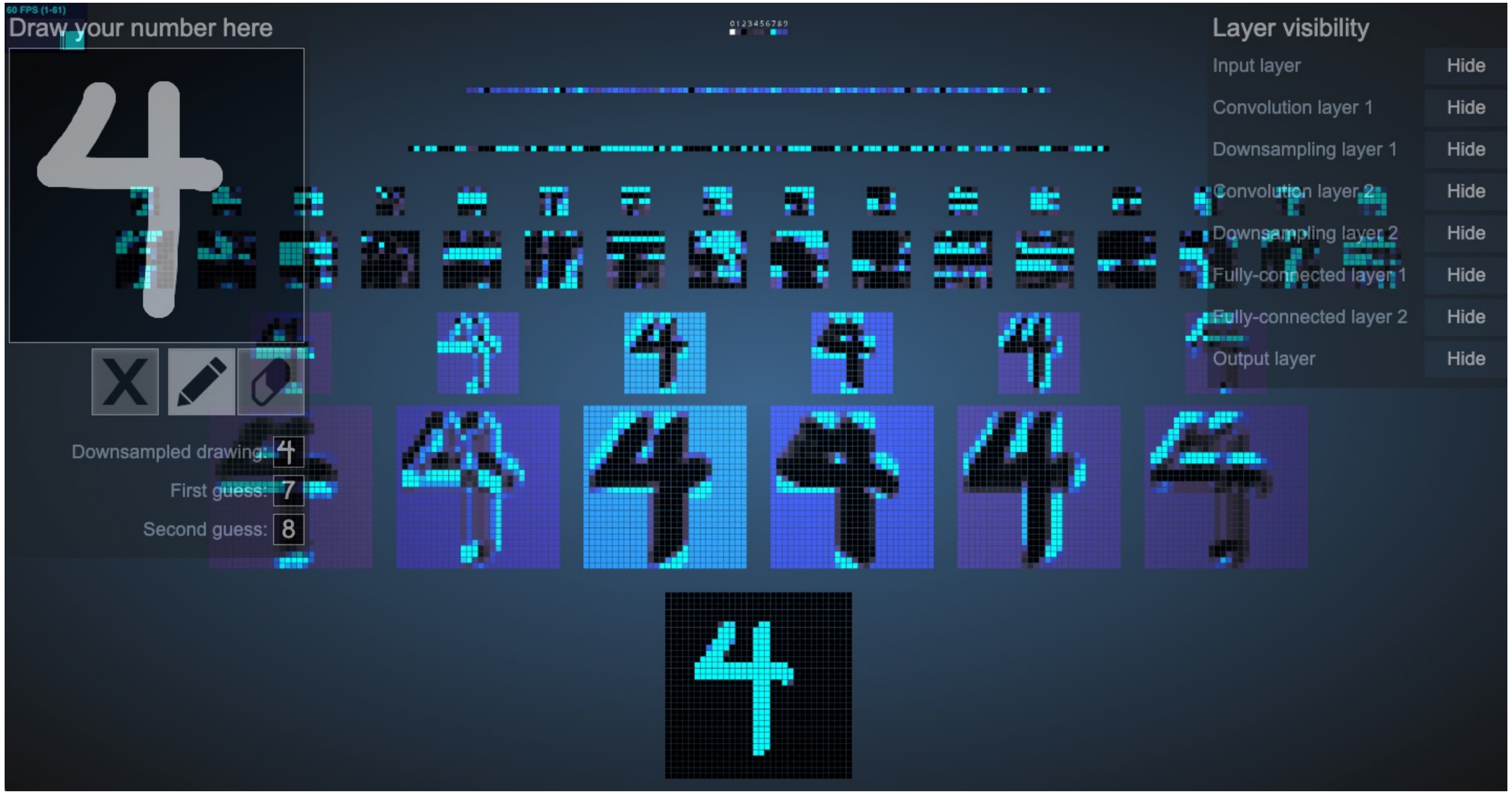
θ_{11}	θ_{12}	θ_{13}	θ_{14}
θ_{21}	θ_{22}	θ_{23}	θ_{24}
θ_{31}	θ_{32}	θ_{33}	θ_{34}
θ_{41}	θ_{42}	θ_{43}	θ_{44}

- 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



MNIST Digit Recognition with CNNs (in your browser)

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

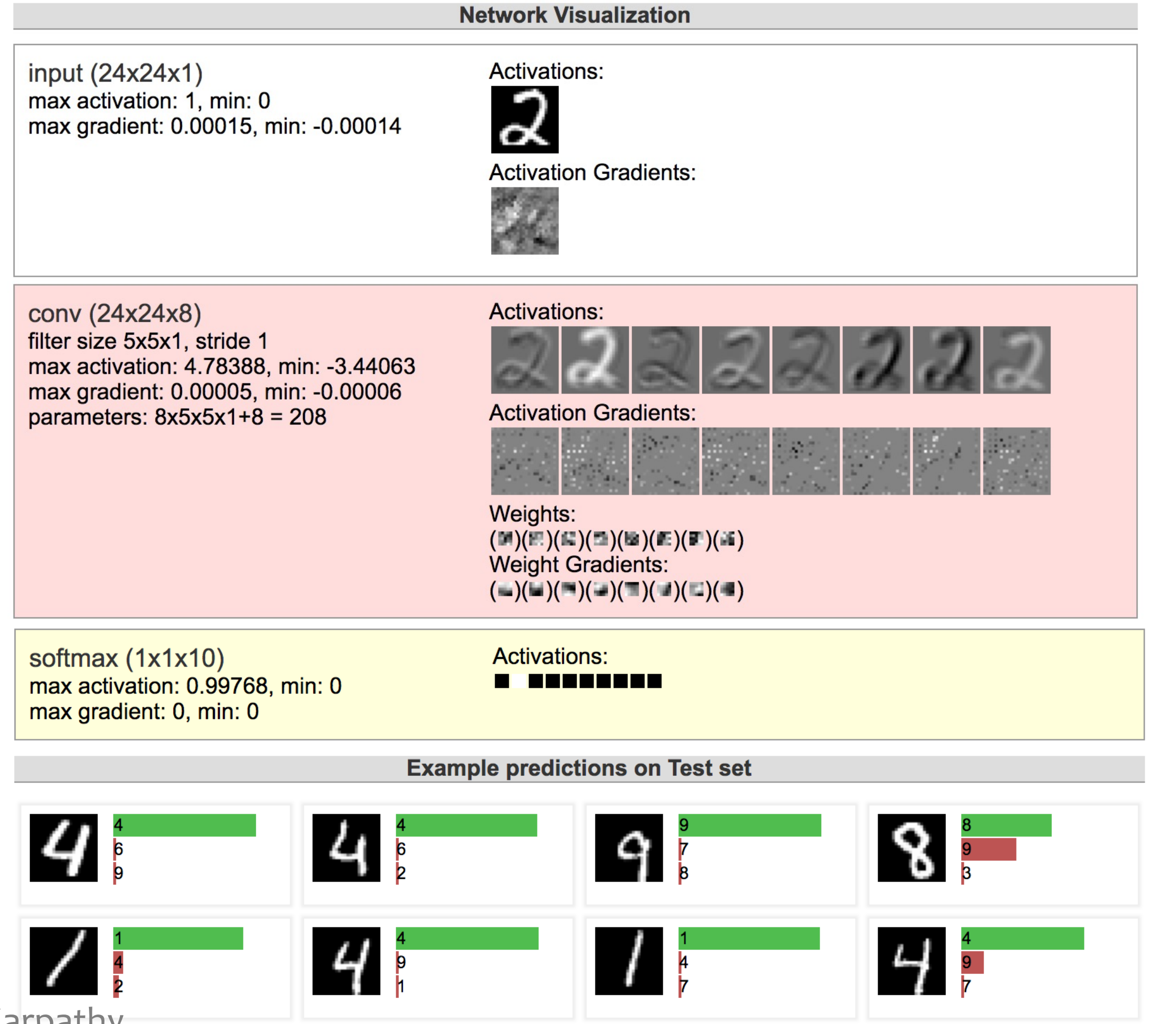


Figure from Andrej Karpathy

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