香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

COMP7015 Artificial Intelligence

Lecture 8: Deep Learning II

Instructor: Dr. Kejing Yin

November 3, 2022

Logistics

- Programming Assignment 1. Due: 23:59 pm on Nov. 6
- Lab 3: Getting started with PyTorch on Nov. 5
- Project Progress Report. Due: 23:59 pm on Nov. 4

About Programming Assignment 1

- Computing the effective branching factor:
  see http://ozark.hendrix.edu/~ferrer/courses/335/f11/lectures/effective-branching.html for a reference.
  see https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html for a numerical solver.

- If you know binary search, compute it using binary search.
- If you can understand the scipy.optimize.fsolve documentation, use fsolve.
- Otherwise, use a simple approximation:

$$b^* = N^{(1/d)}$$

Agenda for Today

- Recap of Deep Learning Basics
  - Activation functions
  - Feedforward Neural Networks / Multilayer Perceptron (MLP)
  - Computational graphs and backpropagation algorithm
- Regularization Techniques for Deep Learning
- Deep Learning in Practice: Hardware and Software
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Pretrained Models

Recap: Feature (Representation) Learning

Interpret $h(x)$ as a learned feature representation

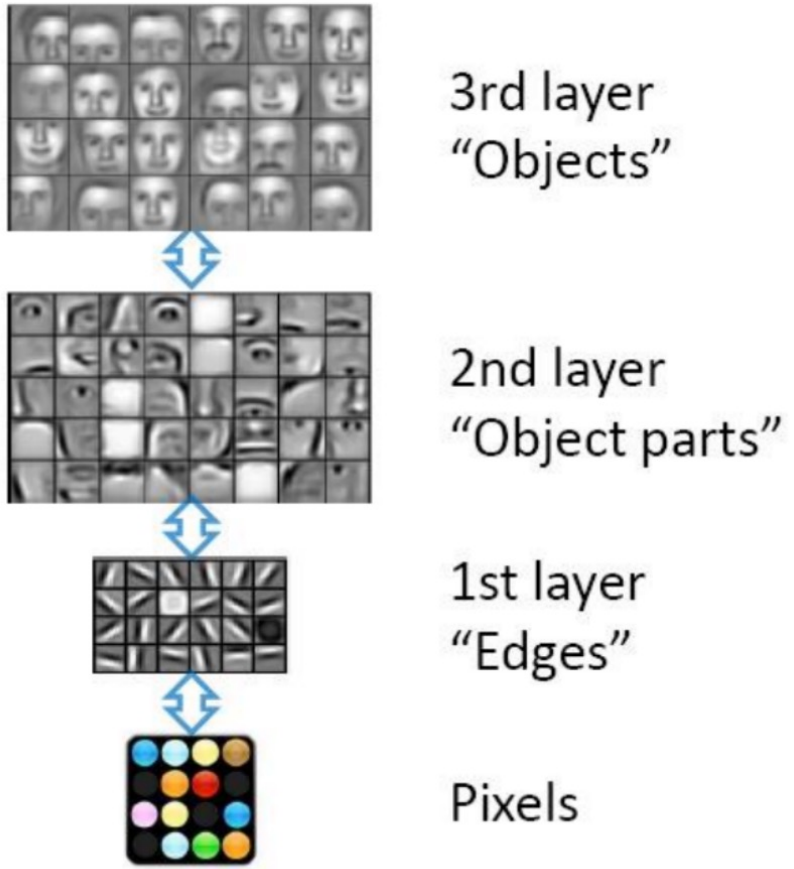
Key idea: feature learning

Before: apply linear predictor on manually specify features

$$\phi(x)$$

Now: apply linear predictor on automatically learned features

$$h(x) = [h_1(x), \ldots, h_k(x)]$$



multiple levels of abstractions

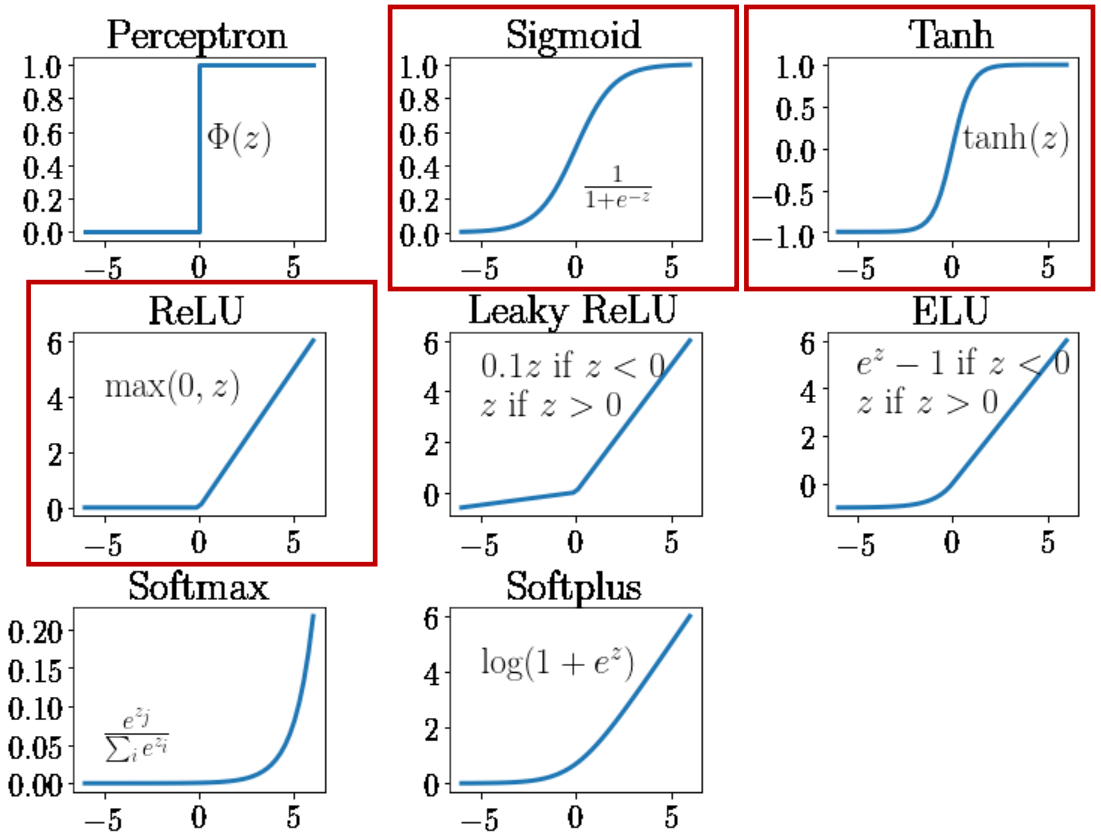
Recap: Activation Functions

- Commonly used activation functions
  - Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Rectified Linear Unit (ReLU):

$$\sigma(z) = \max\{0, z\}$$

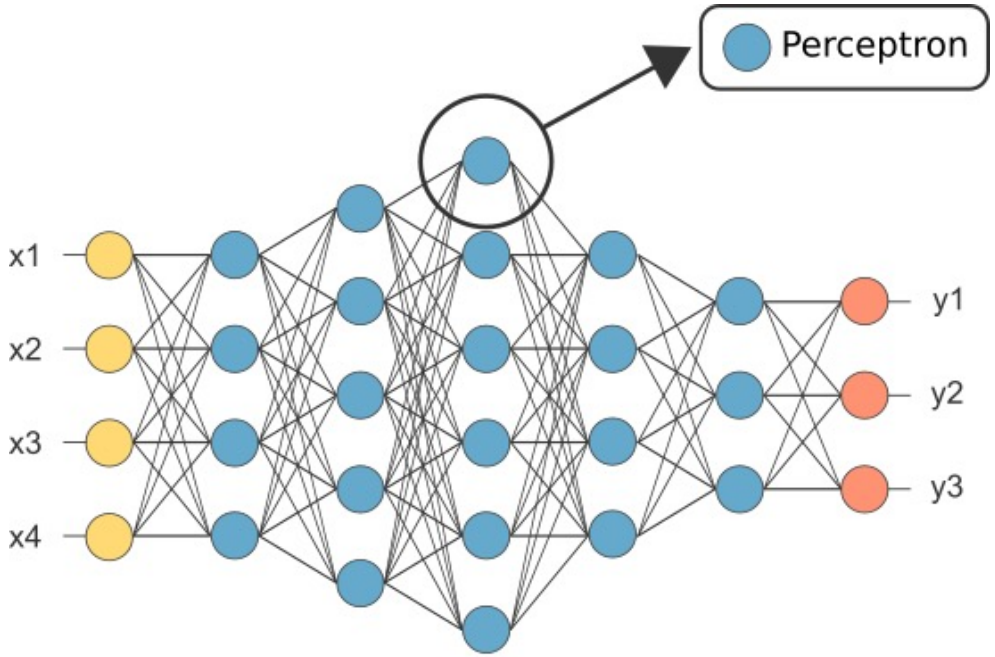
ReLU is recommended for most FNNs

  - Tanh: $\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



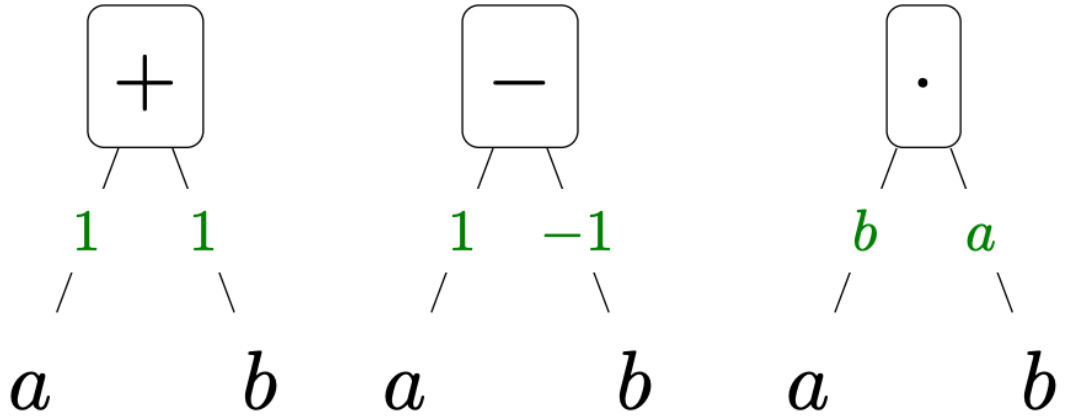
Recap: Feedforward Neural Network

- Feedforward: Information only travels forward in the network
- Use fully connected layers
- Also known as Multilayer Perceptron (MLP)

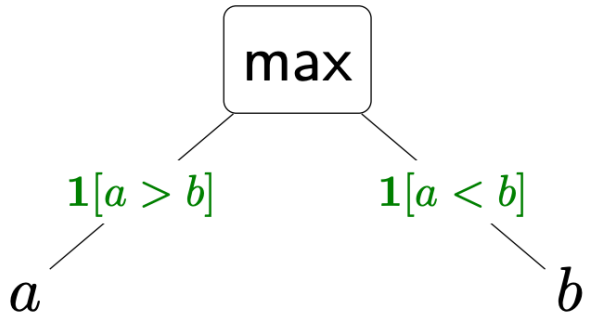


Recap: Computational Graphs and Backpropagation

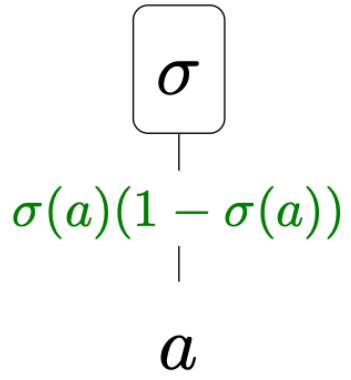
- Building blocks (Five examples)



$a + b$



$\max\{a, b\}$



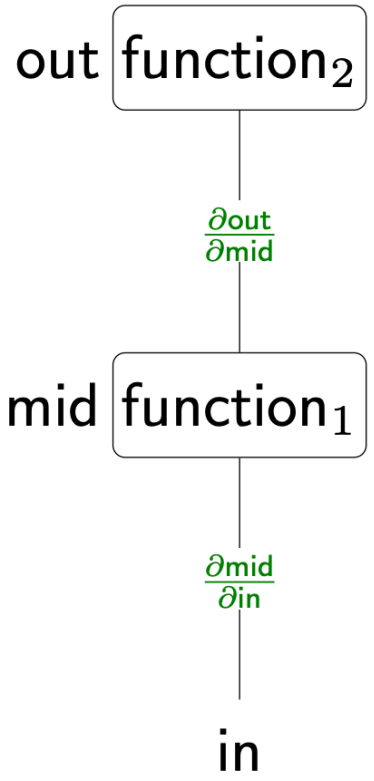
$a - b$

$a \cdot b$

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Recap: Computational Graphs and Backpropagation

- Composing functions



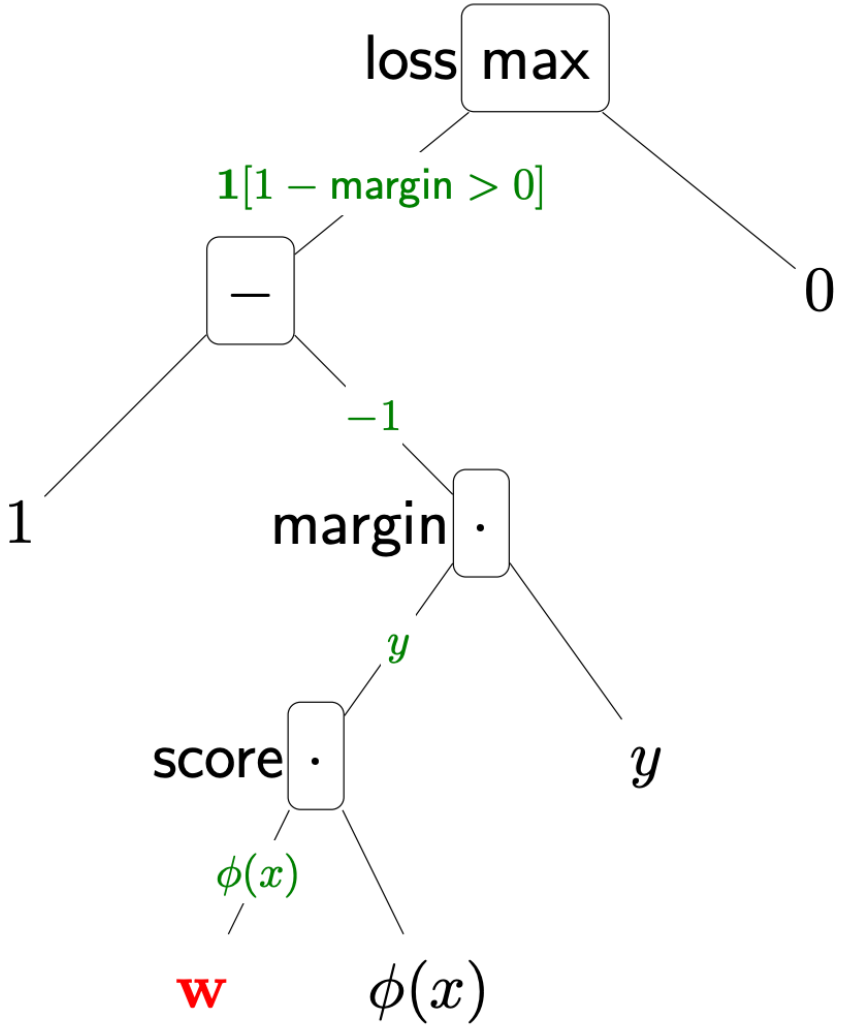
Chain rule: $\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$

Recap: Computational Graphs and Backpropagation

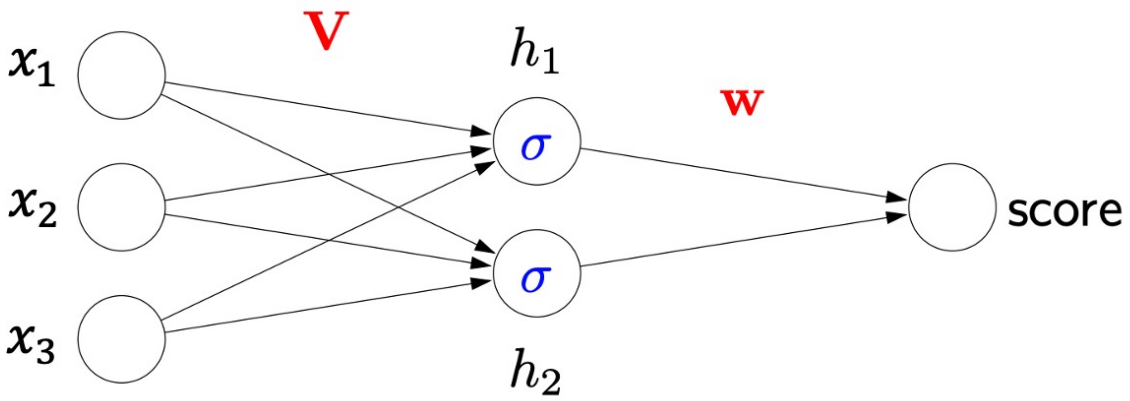
- Example of constructing a computational graph
  - Hinge loss: $\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x)y, 0\}$
  - Compute:

$$\frac{\partial \text{Loss}(x, y, \mathbf{w})}{\partial \mathbf{w}}$$

  - Gradient:
    multiplying the edges: $-\mathbf{1}[\text{margin} < 1]\phi(x)y$



Recap: Computational Graphs and Backpropagation

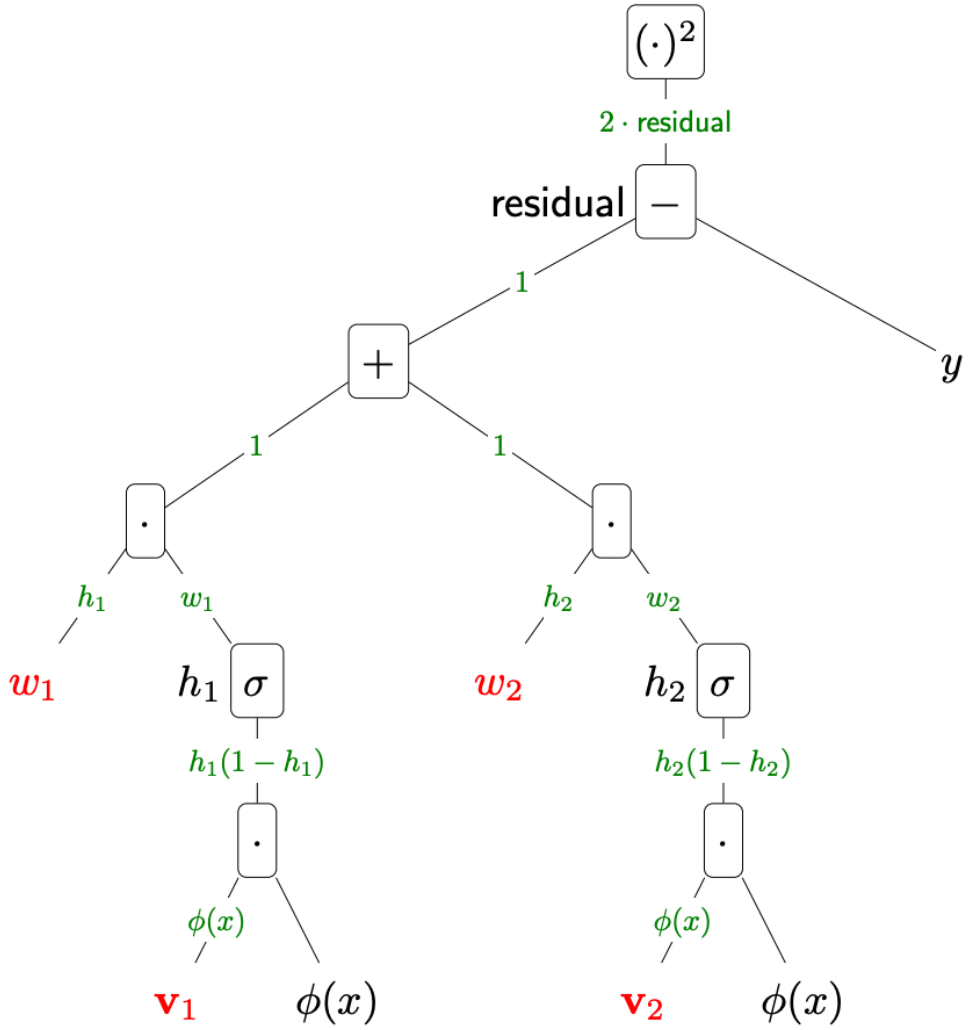


- Exercise:

$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

Construct a computational graph for this neural network

Recap: Computational Graphs and Backpropagation

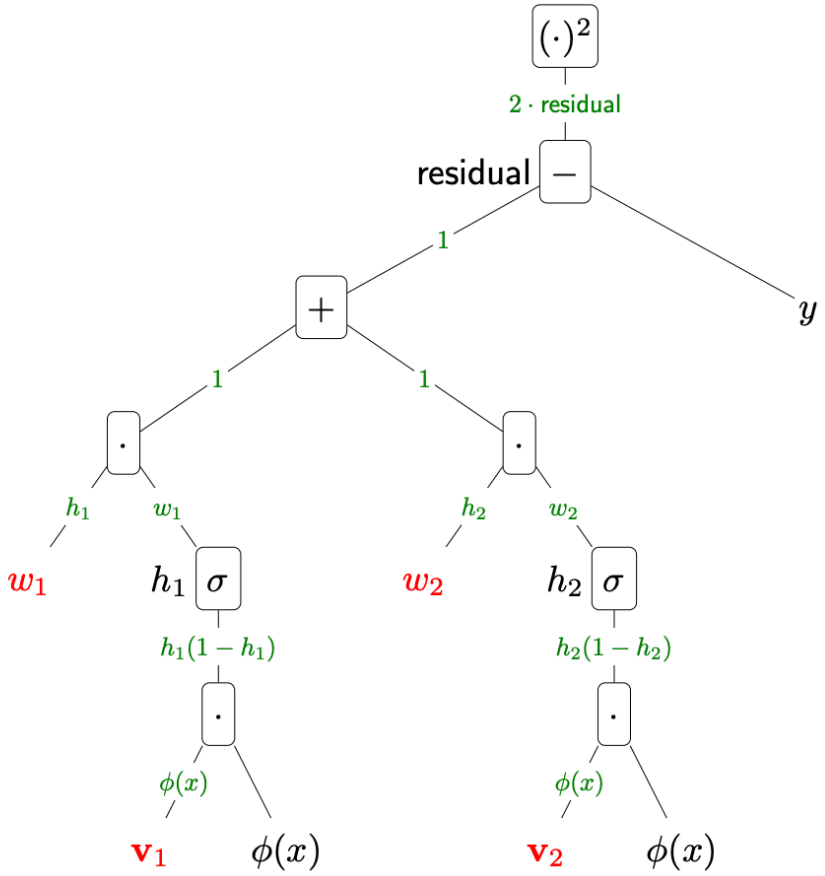


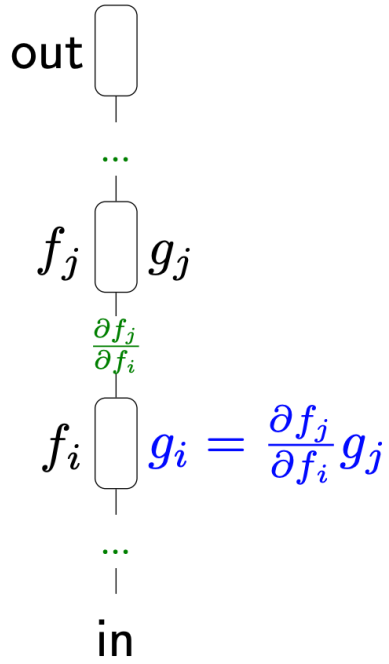
$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

$$\frac{\partial \text{Loss}(x, y, \mathbf{w})}{\partial \mathbf{v}_1} = ?$$

$$2 \cdot \text{residual} \cdot 1 \cdot 1 \cdot w_1 \cdot h_1(1 - h_1) \cdot \phi(x)$$

Recap: Computational Graphs and Backpropagation





Backpropagation

Definition: Forward/backward values

Forward: f_i is value for subexpression rooted at i

Backward: $g_i = \frac{\partial \text{out}}{\partial f_i}$ is how f_i influences output

Algorithm: backpropagation

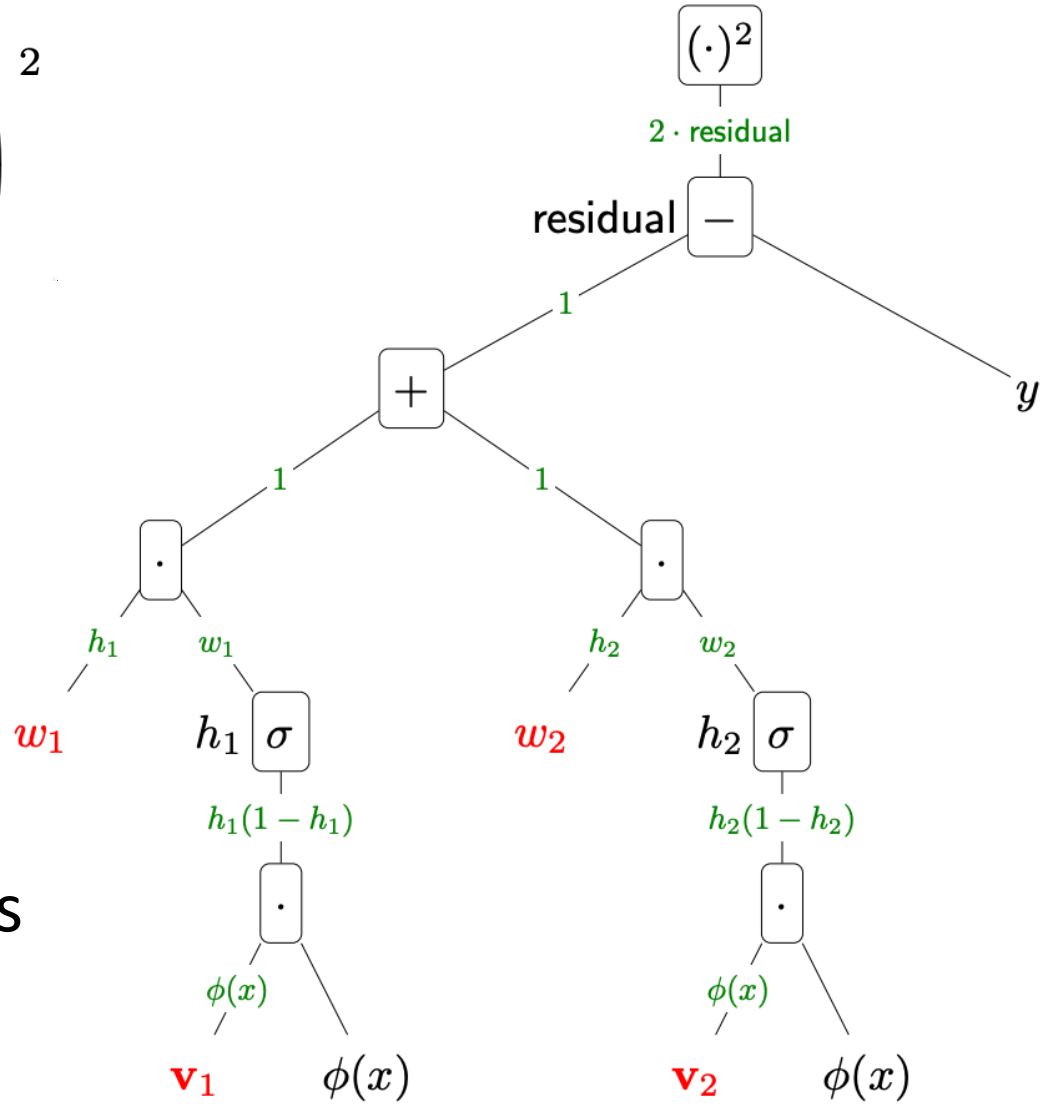
Forward pass: compute each f_i (from leaves to root)

Backward pass: compute each g_i (from root to leaves)

Recap: Computational Graphs and Backpropagation

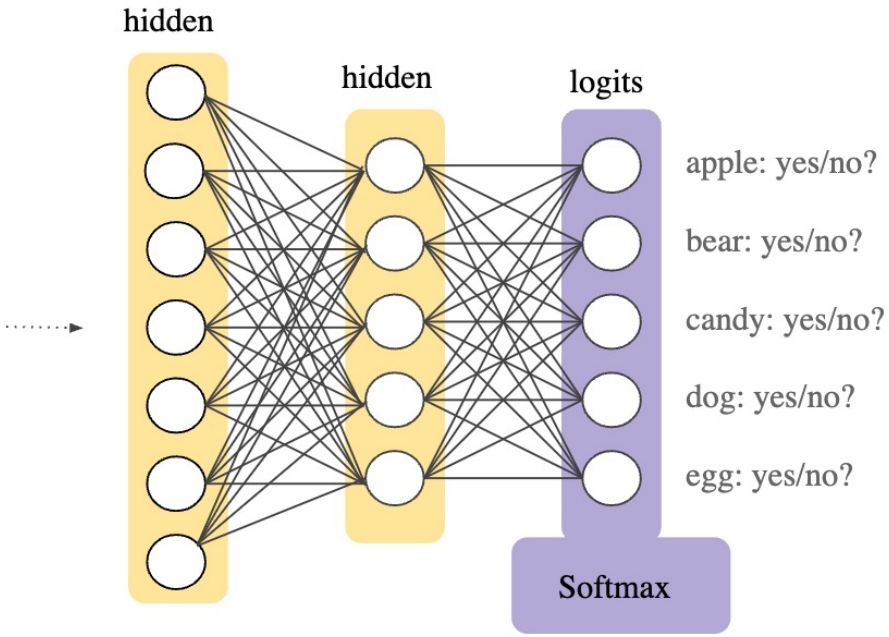
- Exercise: $\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$

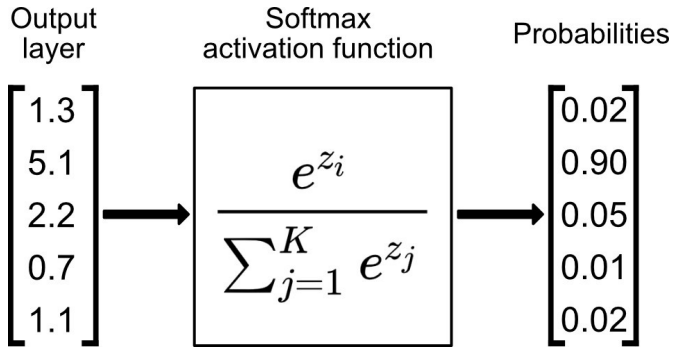
  - Given a data point:
    $\phi(x) = [1.5, -0.7, 0.6], \quad y = 1.8$
  - Model parameters:
    $\mathbf{v}_1 = [1.1, -1, -0.5], \ \mathbf{v}_2 = [2, 1, 1.2]$
    $\mathbf{w} = [2, 0.6]$
  - Compute the loss function in forward pass and gradients in the backward pass



Recap: Feedforward Neural Network for Multicalss Classification

Recall: interpret h as learned features for x





Use softmax activation function for the output layer

Maps real-valued scores to a distribution over the classes

$$\text{Loss}(\mathbf{x}, \mathbf{y}, \Theta) = -\sum_{k=1}^{K} \mathbf{1}[y_k = +1] \log p(y_k = +1 \mid \mathbf{x}) + \mathbf{1}[y_k = -1] \log p(y_k = -1 \mid \mathbf{x})$$

Cross-Entropy Loss

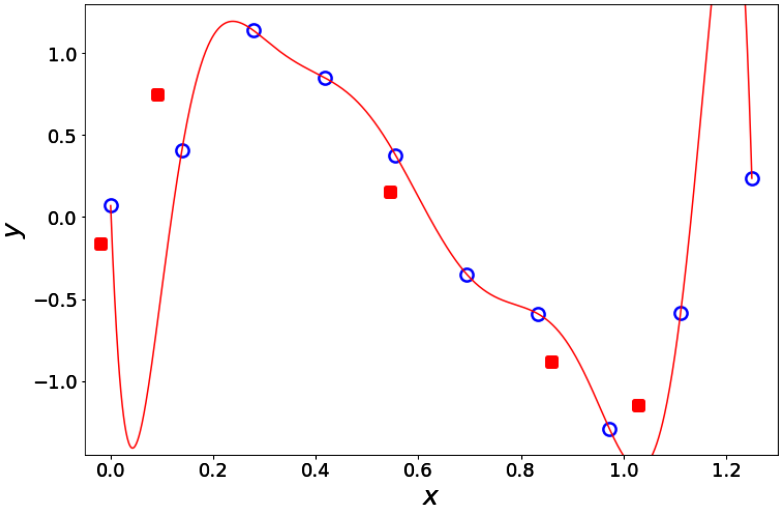
https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Regularization Techniques for Deep Learning

Regularization Prevents Overfitting

- Recall: Overfitting happens if the model is too complex or training data is not enough.
- Solution: Use regularization techniques
- Occam's Razar: Among multiple competing hypotheses, the simplest is the best.

(William of Ockham 1285-1347)



MSE (original data) = 0

MSE (new data) = 136.76

Parameter Norm Penalties

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

- W: model parameters, λ: regularization strength (hyperparameter)
- Popular Choices:

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- Benefits:
  - Express preferences over weights
  - Make the model simple to avoid overfitting

Parameter Norm Penalties

- Example:
  - $x = [1,\ 1,\ 1,\ 1]$
  - $w_1 = [1, 0, 0, 0]$
  - $w_2 = [0.25, 0.25, 0.25, 0.25]$
  - $w_1^T x = w_2^T x = 1$

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

L2 Regularization

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Which of w_1 or w_2 will the L2 regularization prefer?

L2 regularization tends to "spread out" the weights

Parameter Norm Penalties

- Example:
  - $x = [1,\ 1,\ 1,\ 1]$
  - $w_1 = [1, 0, 0, 0]$
  - $w_2 = [0.25, 0.25, 0.25, 0.25]$
  - $w_1^T x = w_2^T x = 1$

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

L1 regularization

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Which of w_1 or w_2 will the L1 regularization prefer?

L1 regularization tends to encourage sparsity.
→ Useful for feature selection.

Dataset Augmentation

- Idea: Train the model on more datasets.
- Dataset Augmentation: Create fake data by some transformations.
- Commonly used in training CNN models.
- Some commonly used transformation:

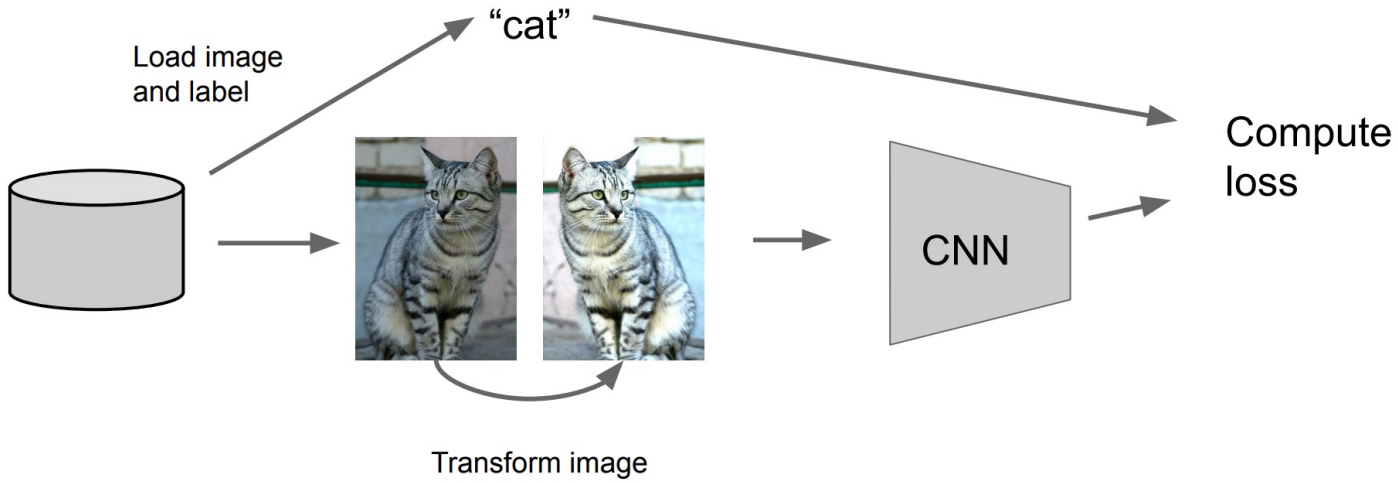


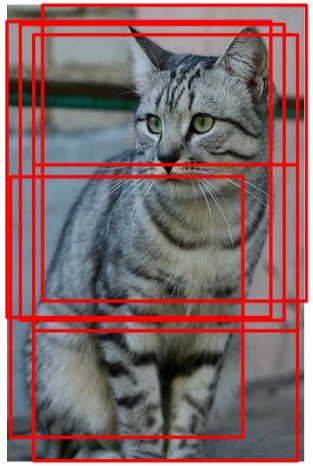
Image from Lecture 7 of Stanford cs231n

Dataset Augmentation

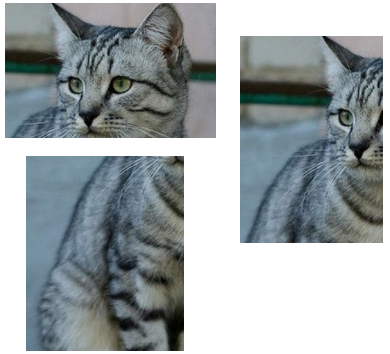
- Some commonly used transformations:

original data

Horizontal flips



Random crops and scales



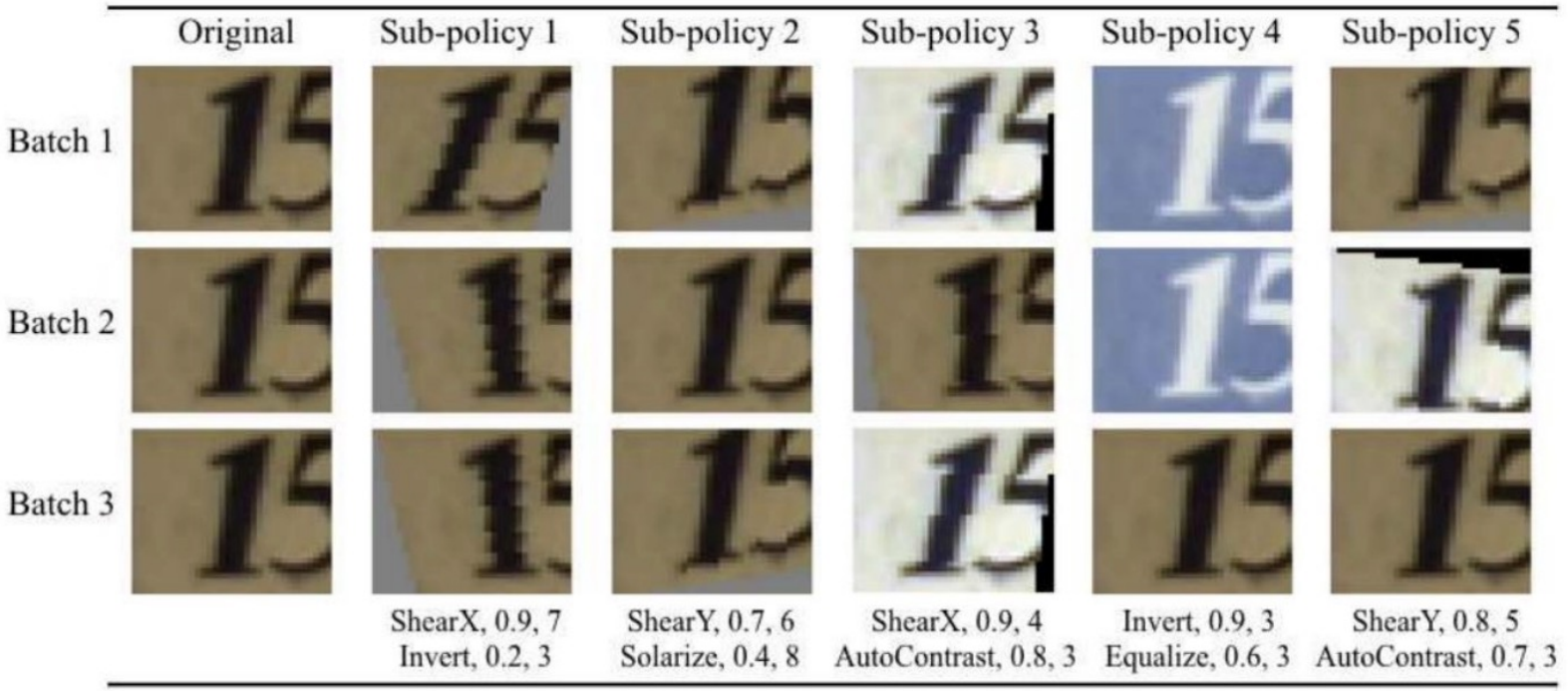
Color jitter (contrast & brightness)

- Translation
- Rotation
- Stretching
- …

Examples from Lecture 7 of Stanford cs231n

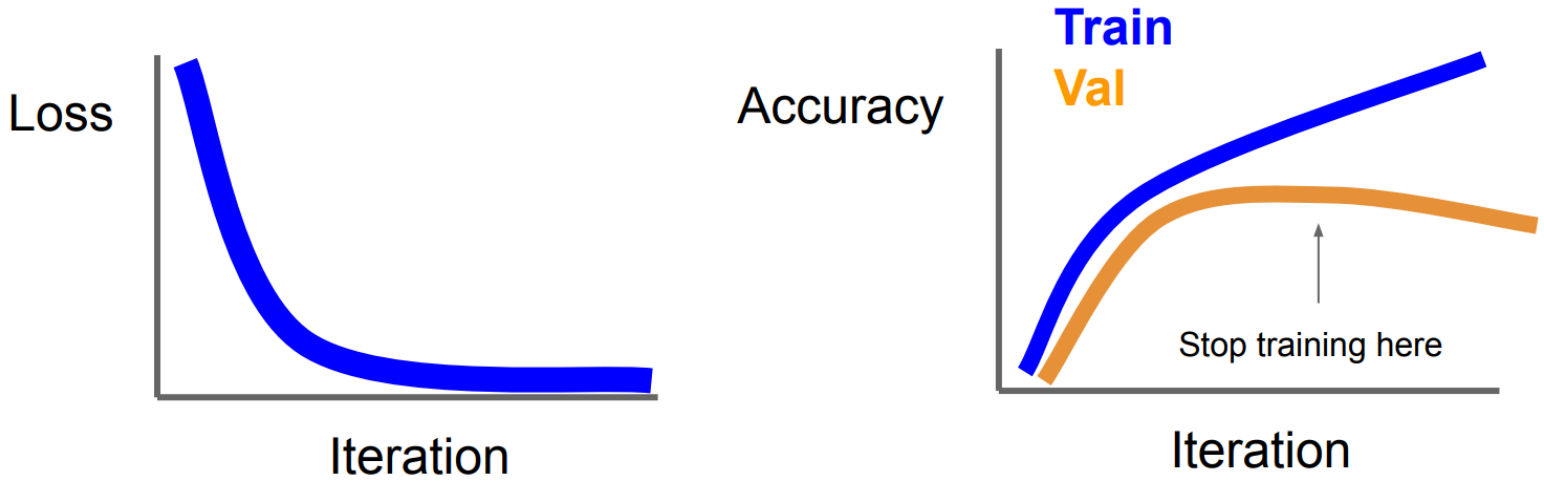
Dataset Augmentation

- Another example: Automatic Data Augmentation



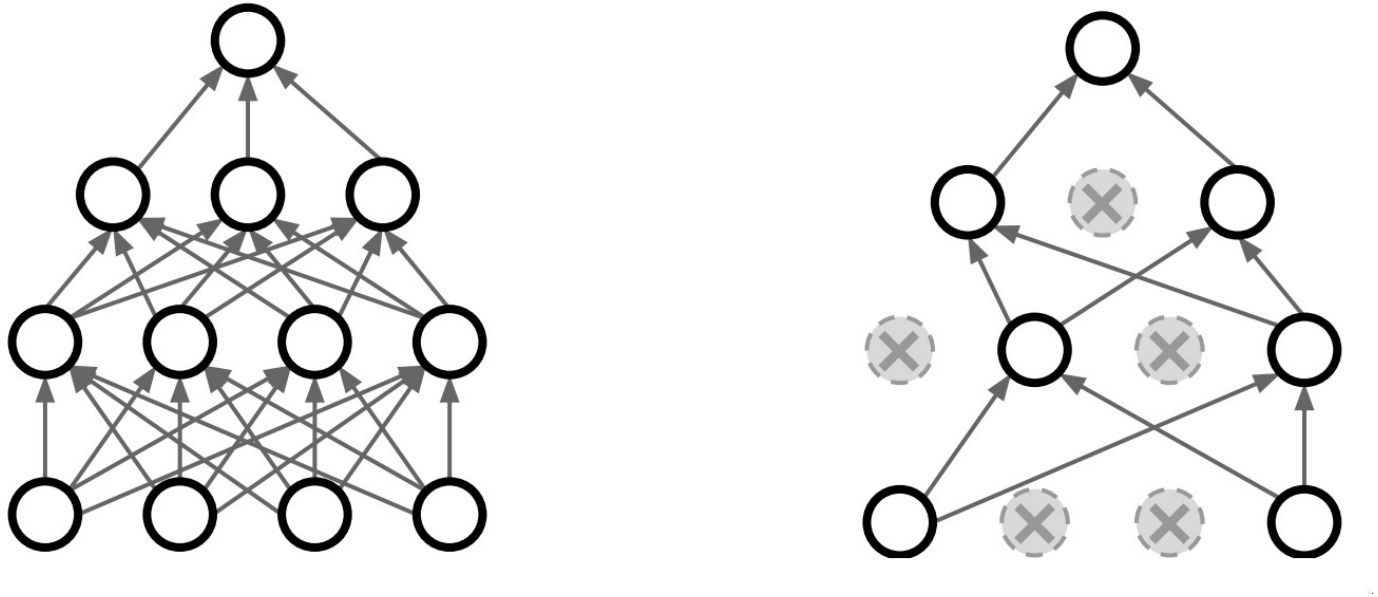
Examples from Lecture 7 of Stanford cs231n
Original source: Cubuk et al., "AutoAugment: Learning Augmentation Strategies from Data", CVPR 2019

Early Stopping



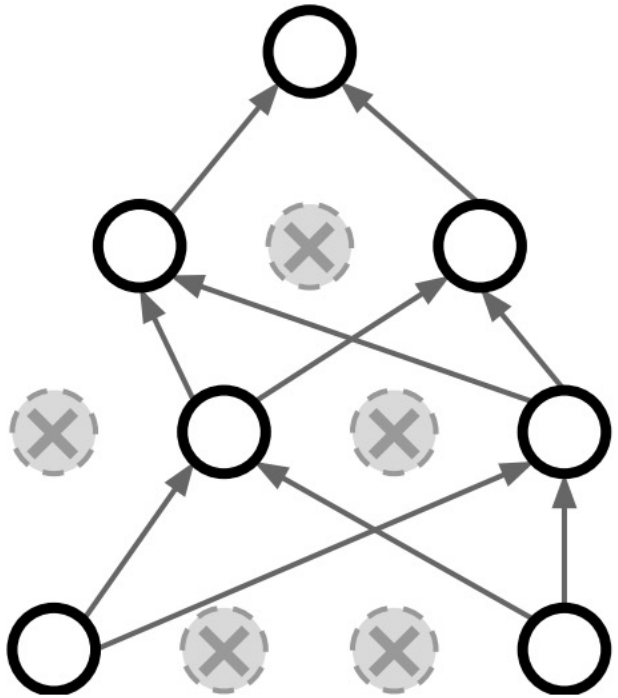
- When training the model, we usually hold out a part of the training data as validation set. *(Different from the test set, which is used for performance evaluation)*
- During model training, we monitor the accuracy (or other metrics) in training and validation sets.
- When accuracy on the validation set decreases, stop the training.
- Rule of thumb: always do this.

Dropout



- In each forward pass during model training, randomly set some neurons to zero according to some probability.
- The probability of dropping: dropout rate (a hyperparameter; common value: 0.2-0.5)

Dropout



- Why it works?

Forces the network to have a redundant representation;
Prevents co-adaptation of features

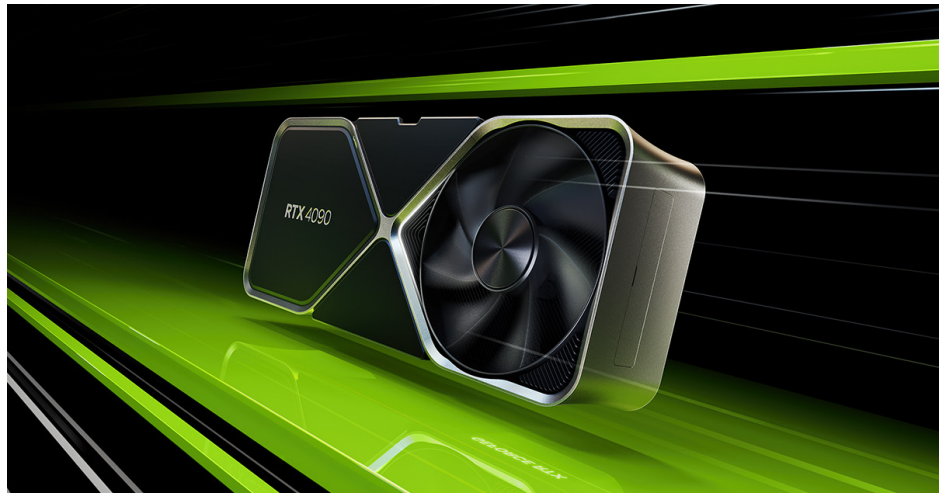
- Note: the testing phase of models with dropout differs from that without using dropout: need weight scaling.
  See page 253-257 of Deep Learning for derivations.
  DL frameworks will automatically handle this in practice.

Deep Learning in Practice: Hardware and Software

Hardware: CPUs and GPUs

- Central processing unit (CPU): Fewer cores but each core is very fast; good at sequential tasks.
- Graphics processing unit (GPU): More cores but each core is much slower and "dumber"; good at parallel tasks: e.g., matrix operations.

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	$1199	~13.4 TFLOPs FP32



Example from https://jasonyanglu.github.io/files/lecture_notes/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0_2021/Lecture%204%20Hardware%20and%20Software.pdf

Hardware: CPUs vs GPUs

A GPU card released 7 years ago

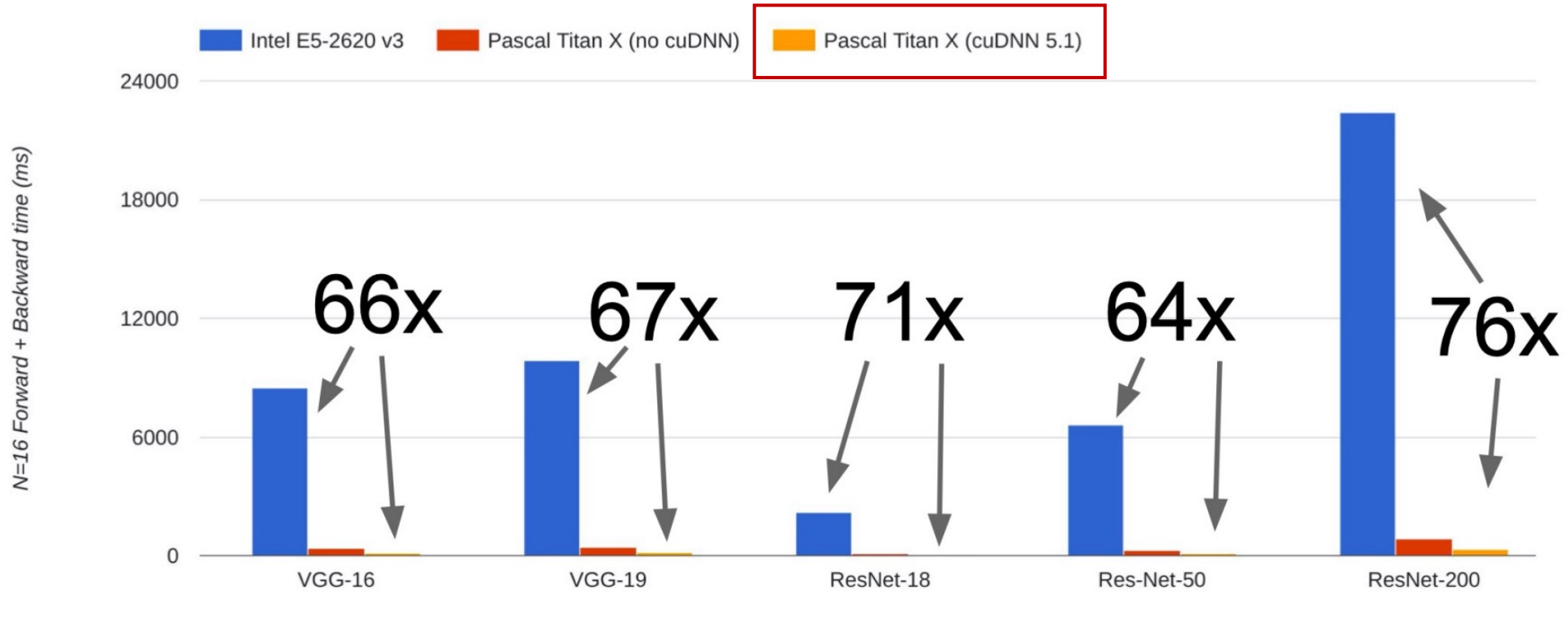


Image from Lecture 6 of Stanford cs231n

Unit Price of Computation Power

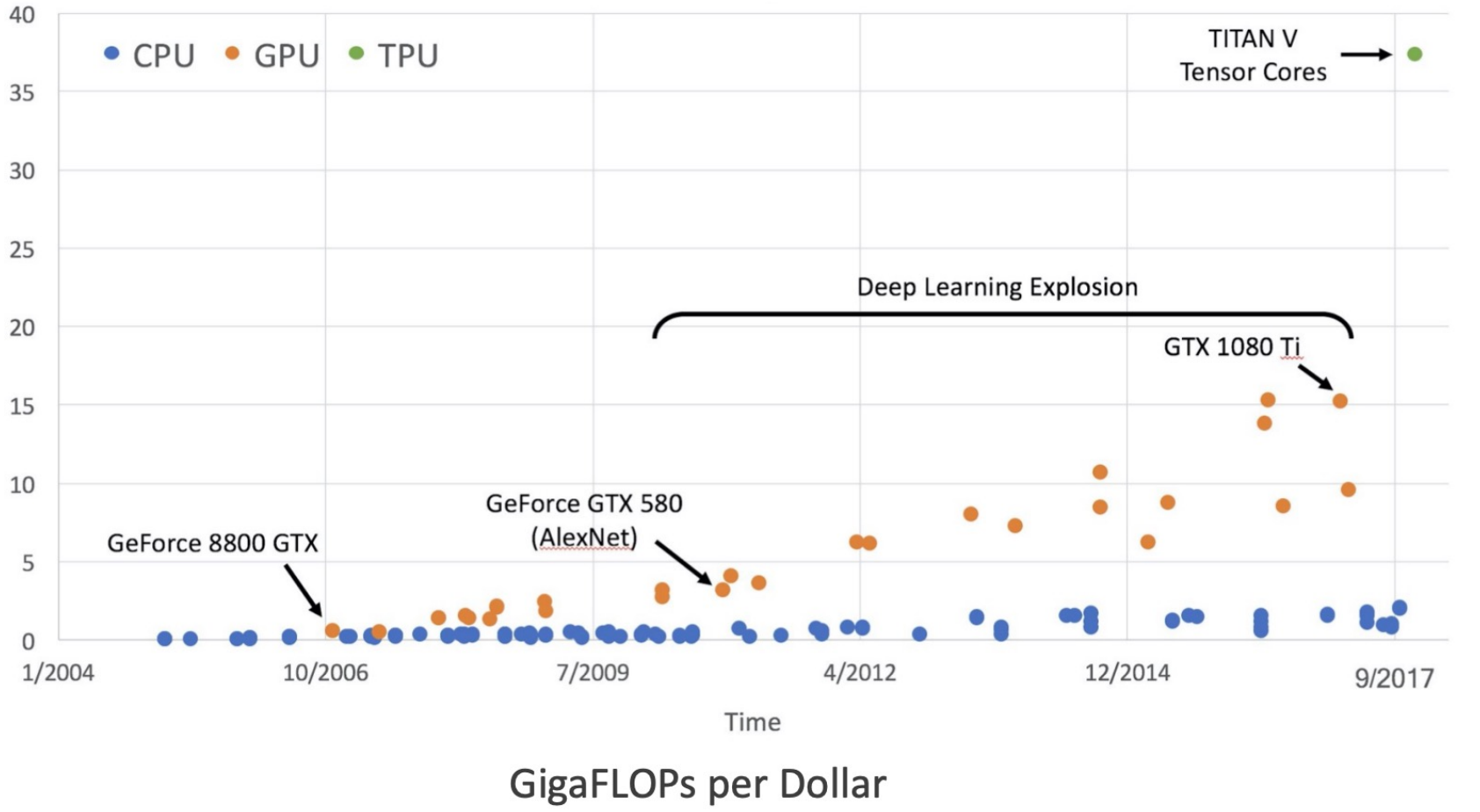
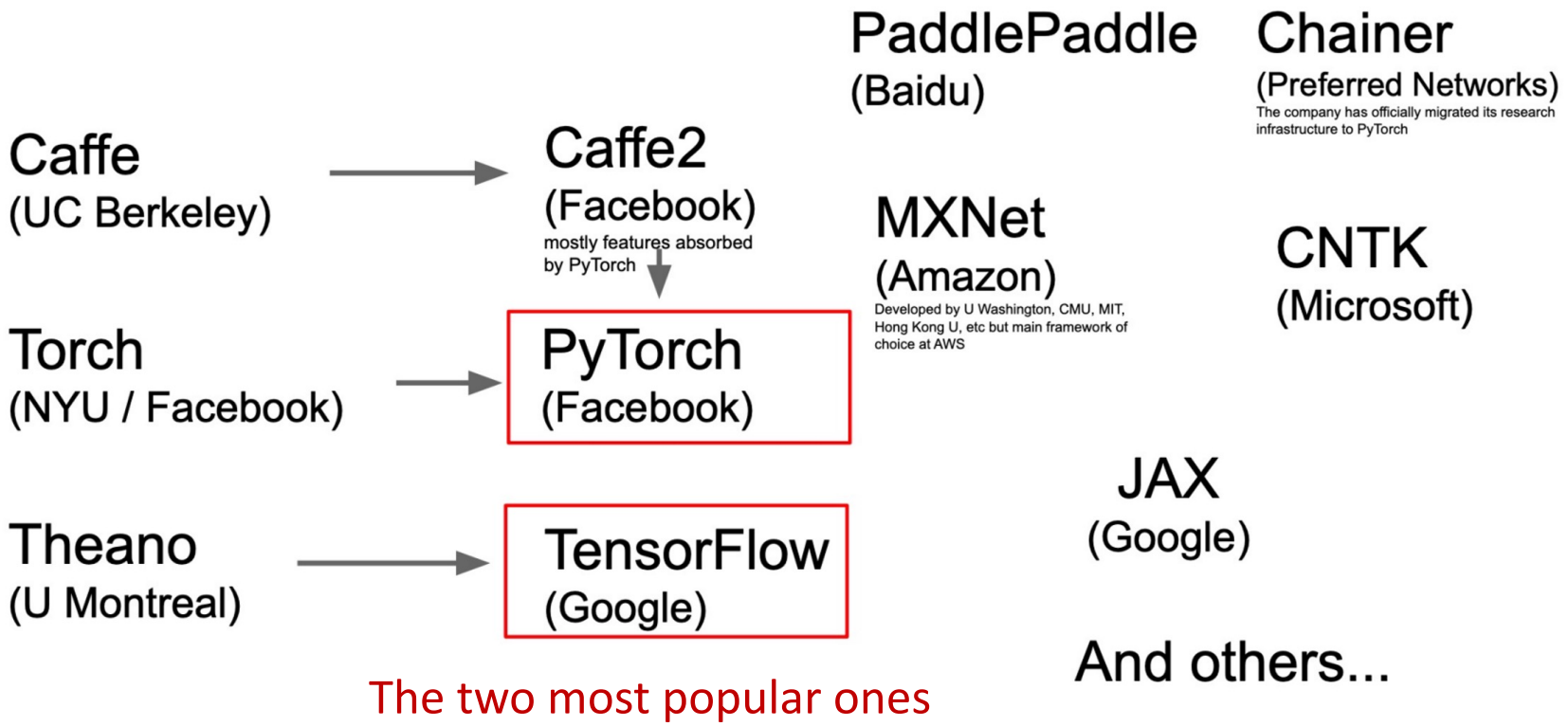


Image from Lecture 6 of Stanford cs231n

Software: Deep Learning Frameworks

Software: Deep Learning Frameworks



We will see details about the basic usage of PyTorch on the lab session

Image from Lecture 6 of Stanford cs231n

Convolutional Neural Network (CNN)

Credits: this section is partially adapted from the following sources

- https://jasonyanglu.github.io/files/lecture_notes/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0_2021/Lecture%205%20Basics%20of%20Convolutional%20Neural%20Networks.pdf
- http://cs231n.stanford.edu/slides/2022/lecture_5_ruohan.pdf
- https://home.ttic.edu/~shubhendu/Pages/Files/Lecture7_pauses.pdf

CNN Applications

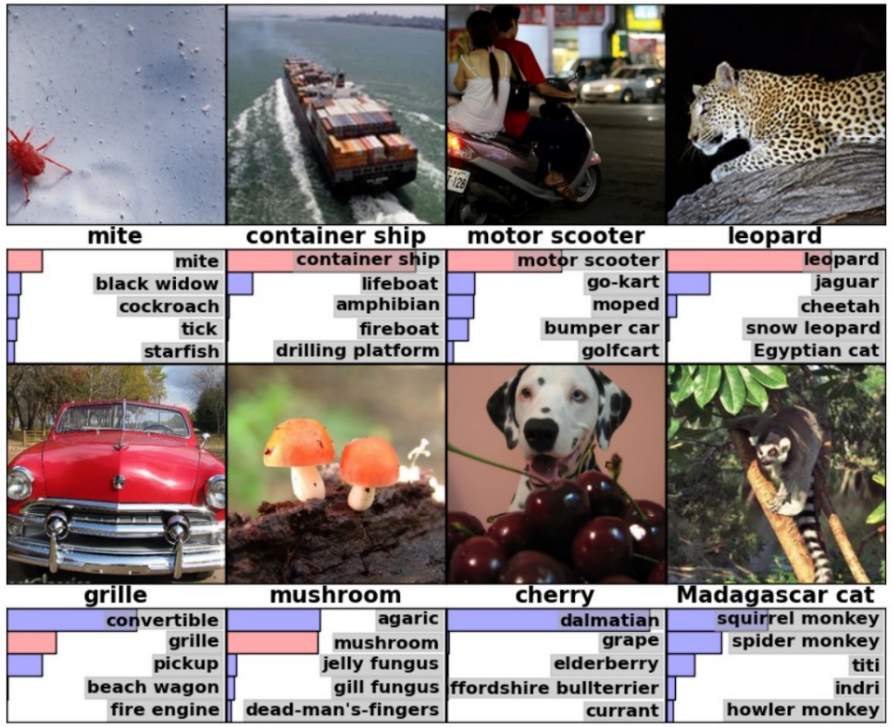


Image classification

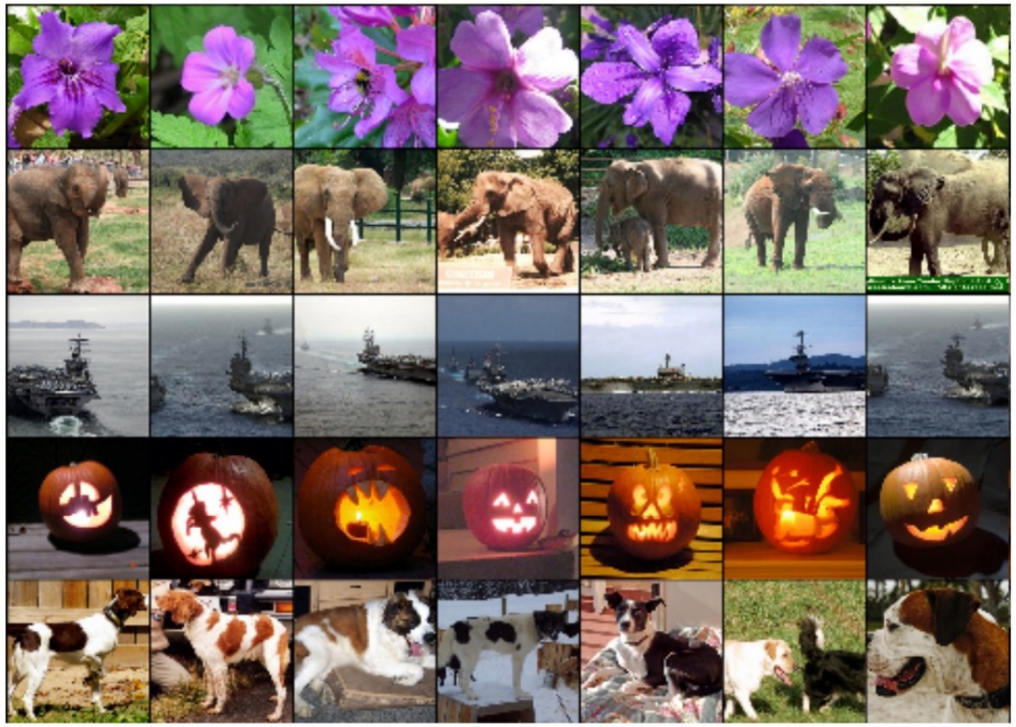
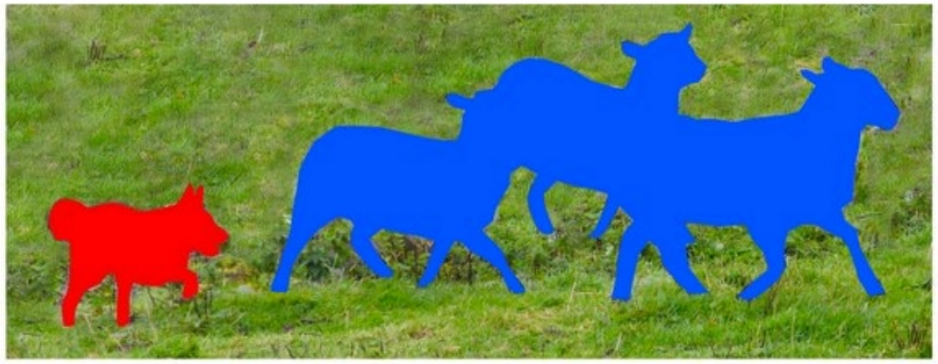


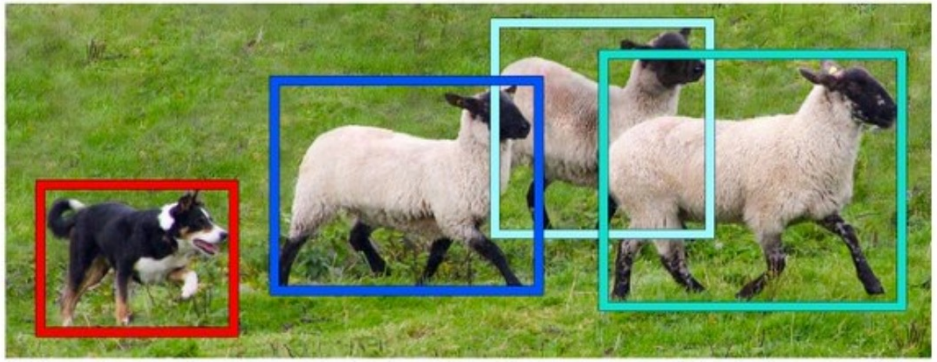
Image retrieval

CNN Applications

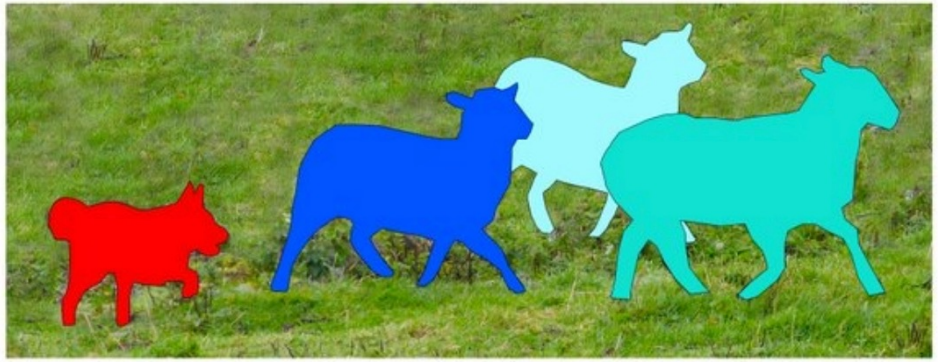
Image Recognition



Semantic Segmentation



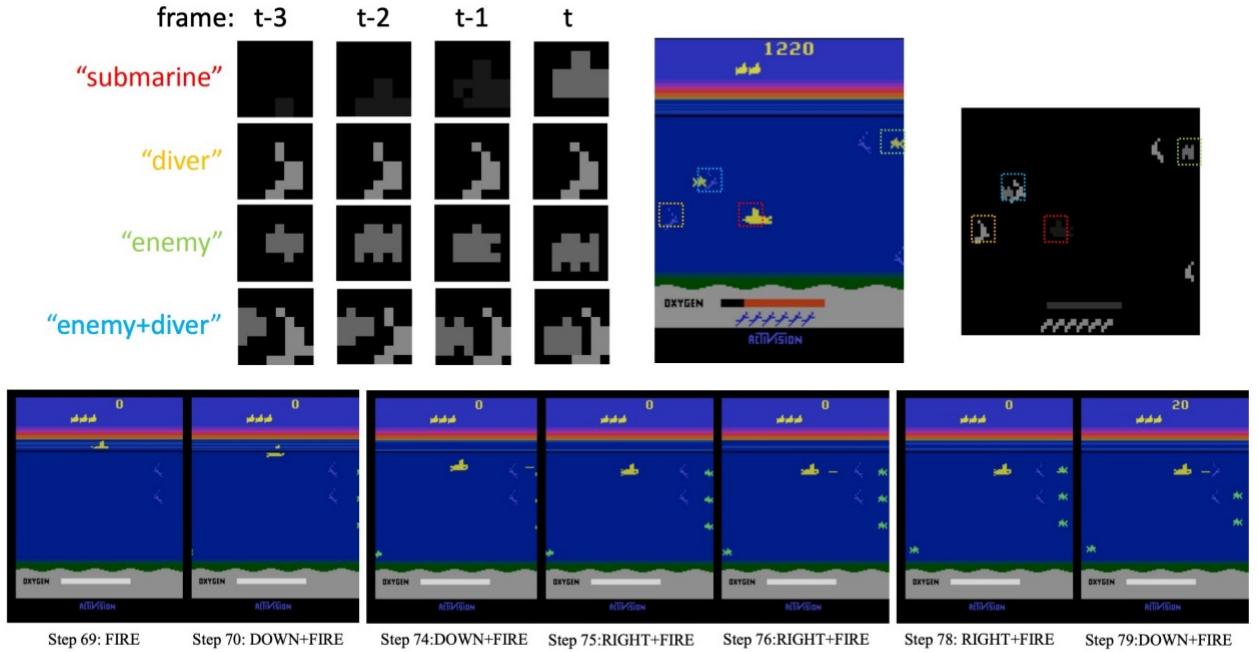
Object Detection



Instance Segmentation

CNN Applications

Pose estimation

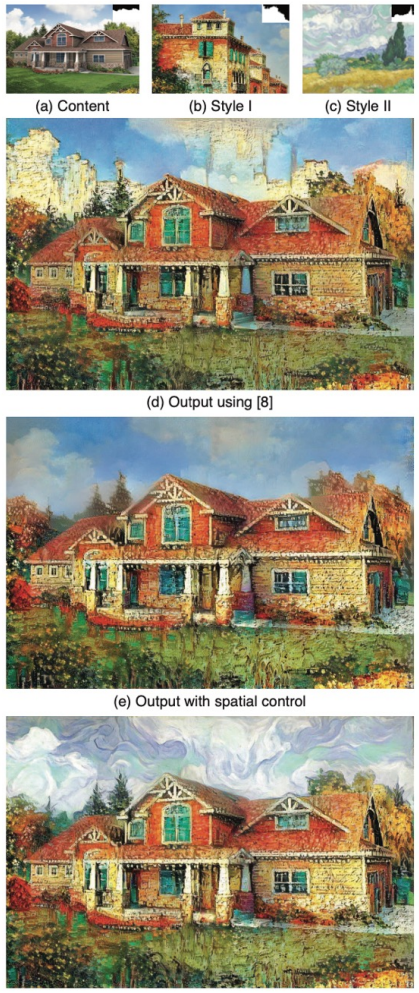


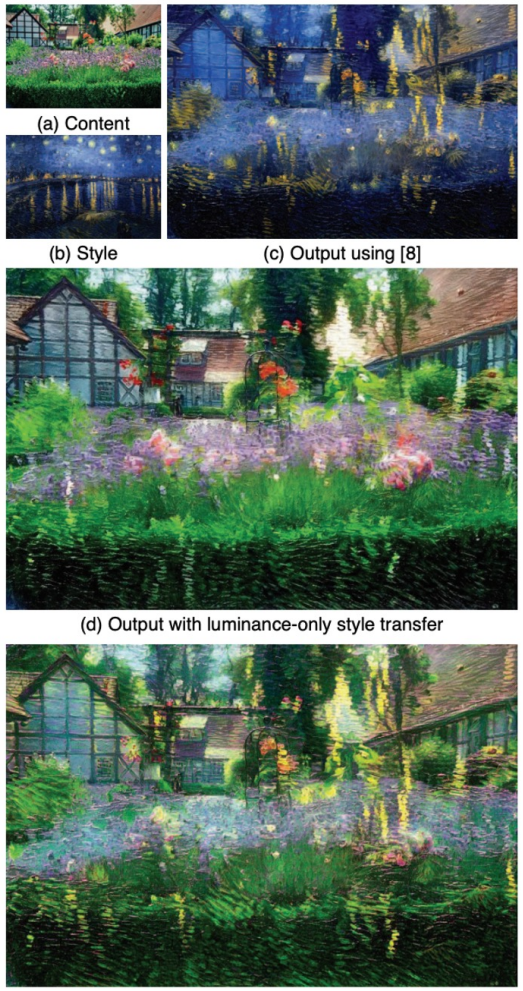
Real-time Atari game play

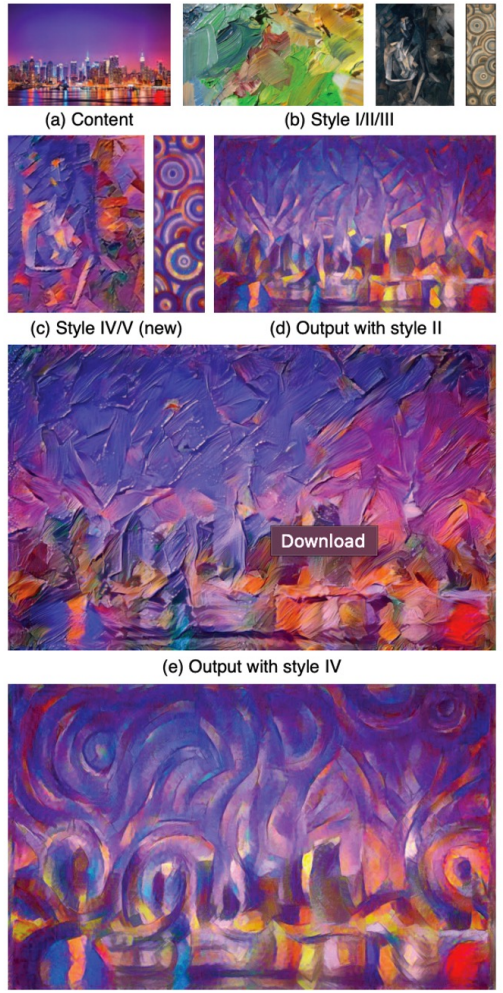
CNN Applications

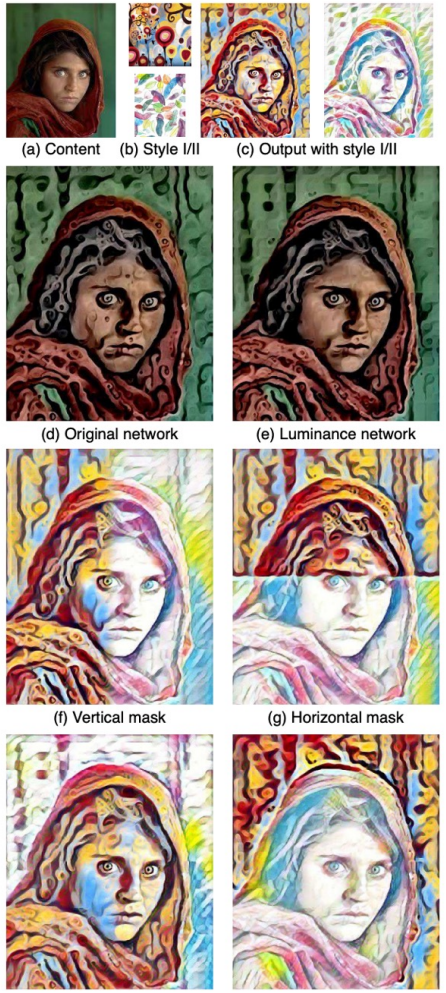
Image captioning

CNN Applications





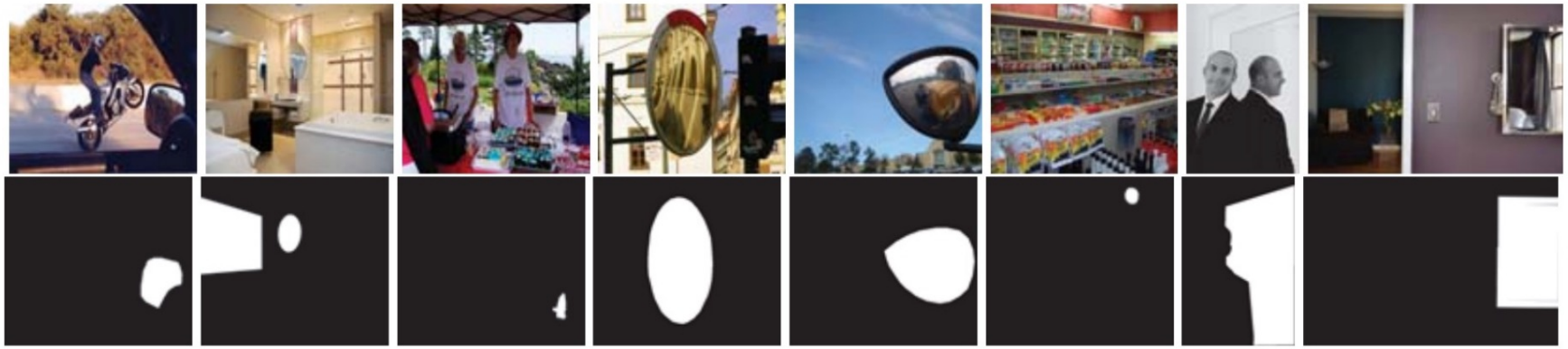




Style transfer

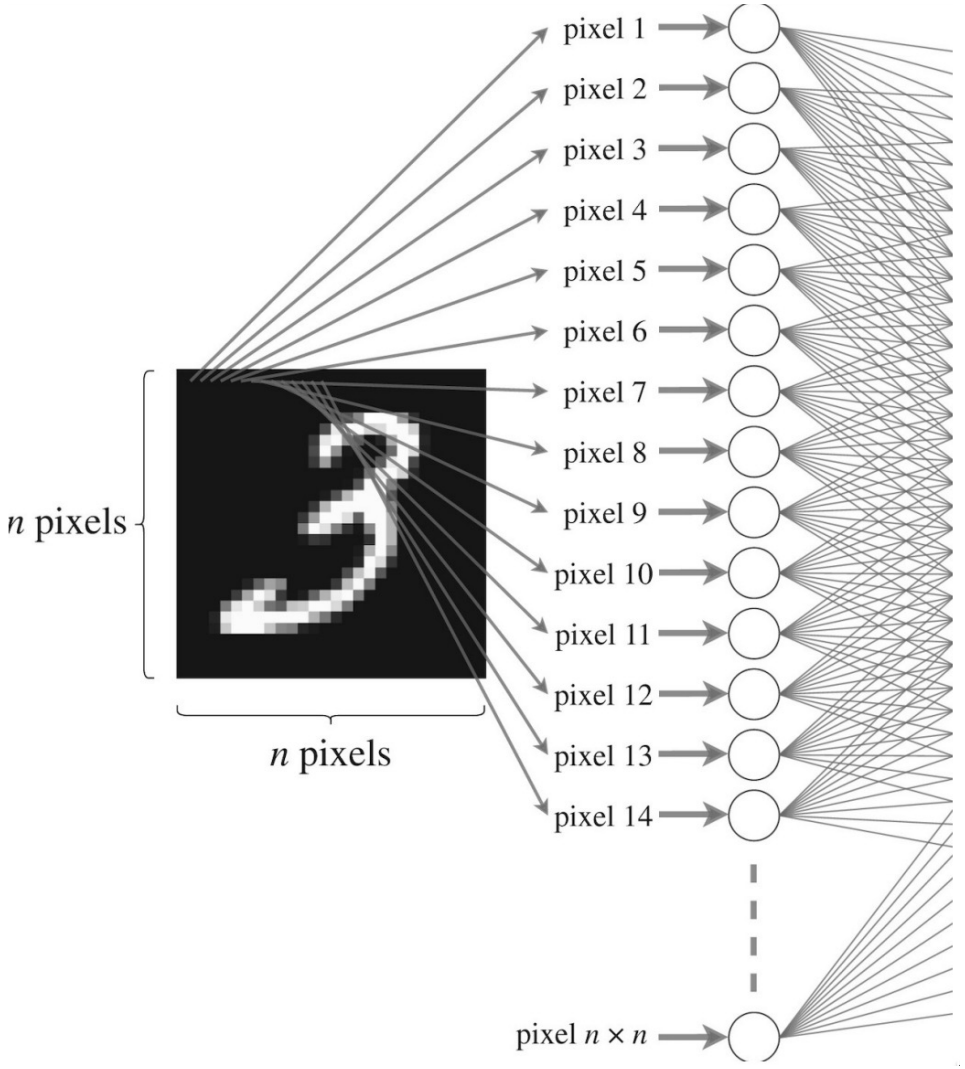
CNN Applications

Rain and fog removal



Mirror detection

Motivating Example: Digit Classification



- Suppose we have a n-by-n image.
- We can vectorize it and use MLP for classification.
- Is it a good idea?
- What could be the potential issue?

2D spatial structure is lost

Convolutional Neural Networks

- Convolutional neural networks (CNNs) are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

- Taking 2-D image as an example. Convolution produces a new image S given an input I with a kernel/filter K (size: $(\mathrm{m}+1)\times(n+1)$), by:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

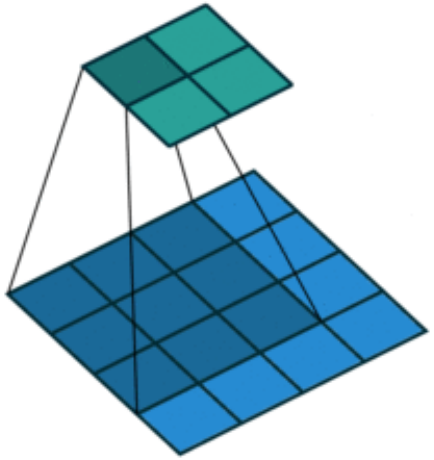
pixel position

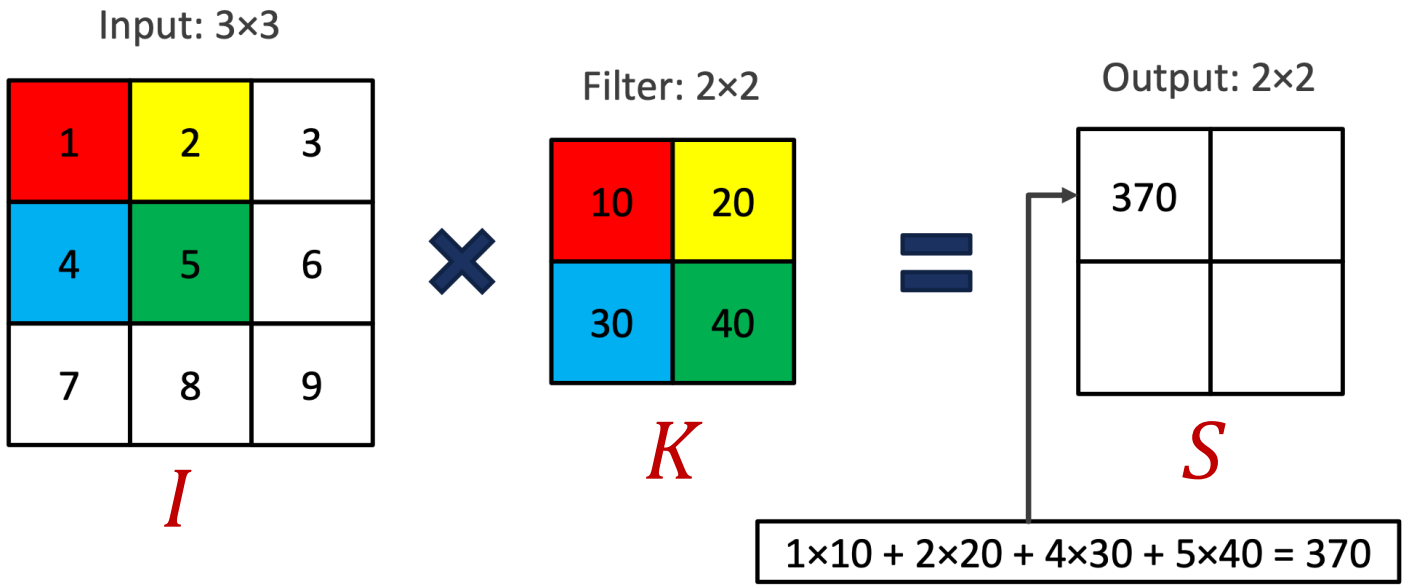
offset

  - MLP: input is x, weight is W, and output is $\mathbf{h}$
  - Convolutional layer: input is I, weight is K, output is S

- The hidden outputs in CNNs (S) are called feature map.

Convolution

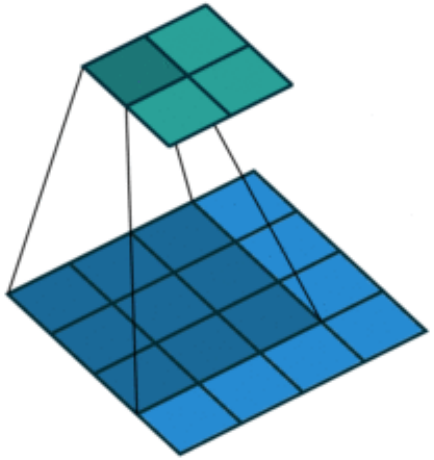


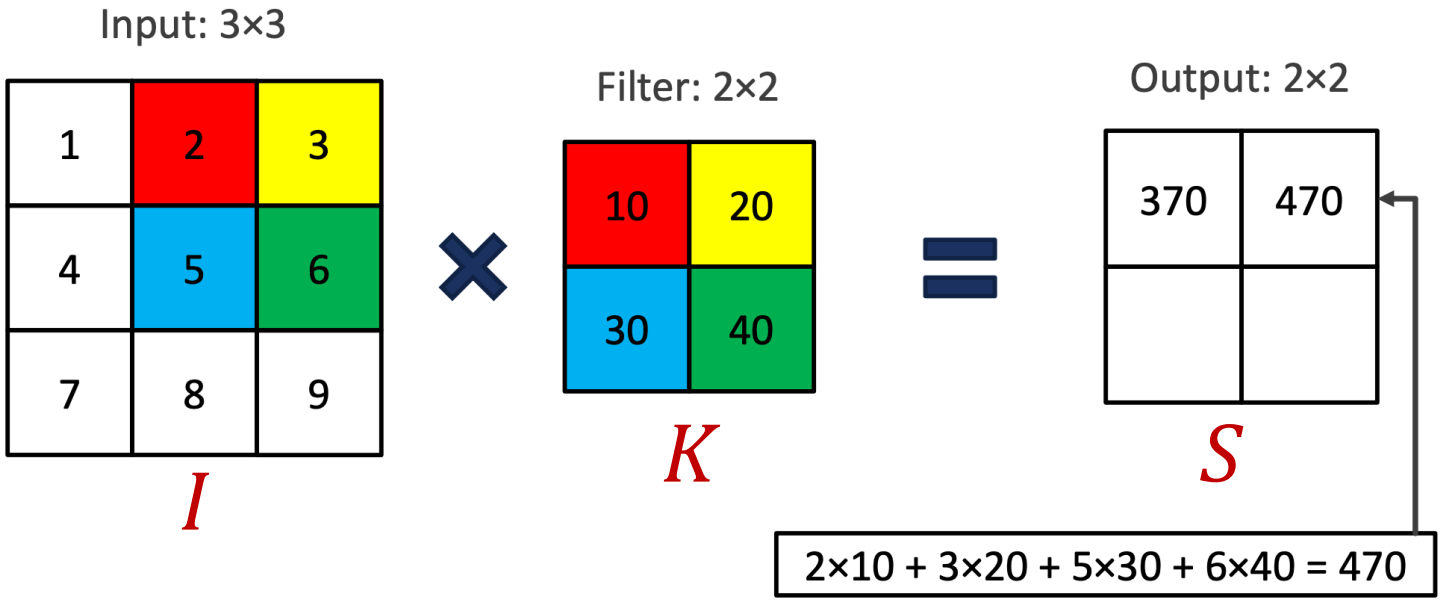


$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

$$S(0,0) = (I * K)(0,0) = I(0+0,0+0)K(0,0) + I(0+0,0+1)K(0,1) + I(0+1,0+0)K(1,0) + I(0+1,0+1)K(1,1)$$

Convolution

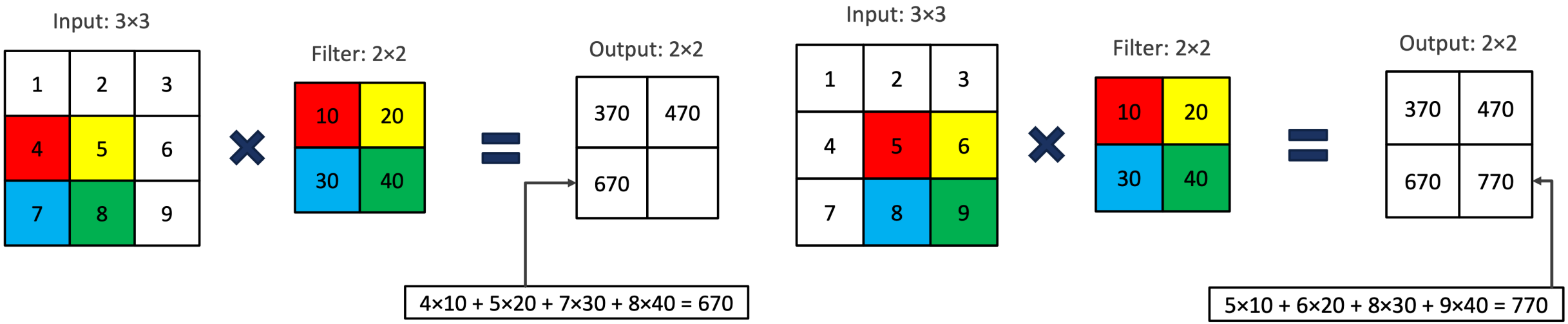




$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

$$S(0,1) = (I * K)(0,1) = I(0+0, 1+0)K(0,0) + I(0+0, 1+1)K(0,1) + I(0+1, 1+0)K(1,0) + I(0+1, 1+1)K(1,1)$$

Convolution

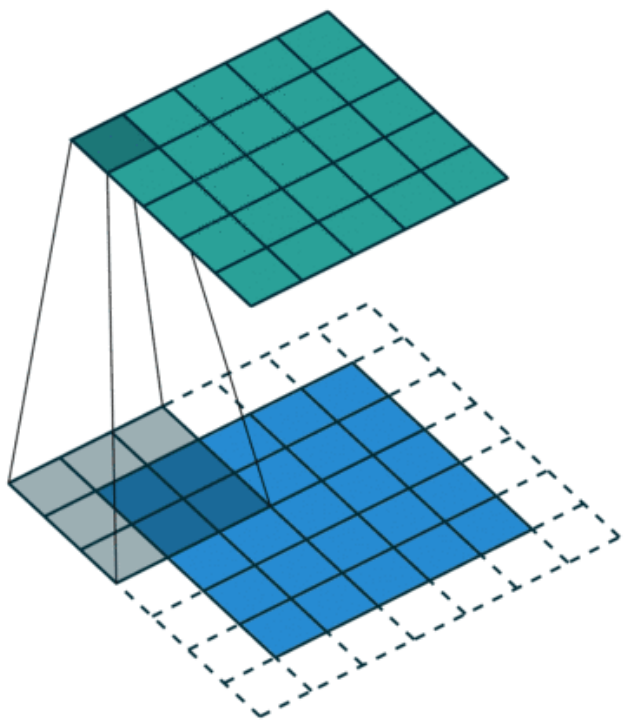


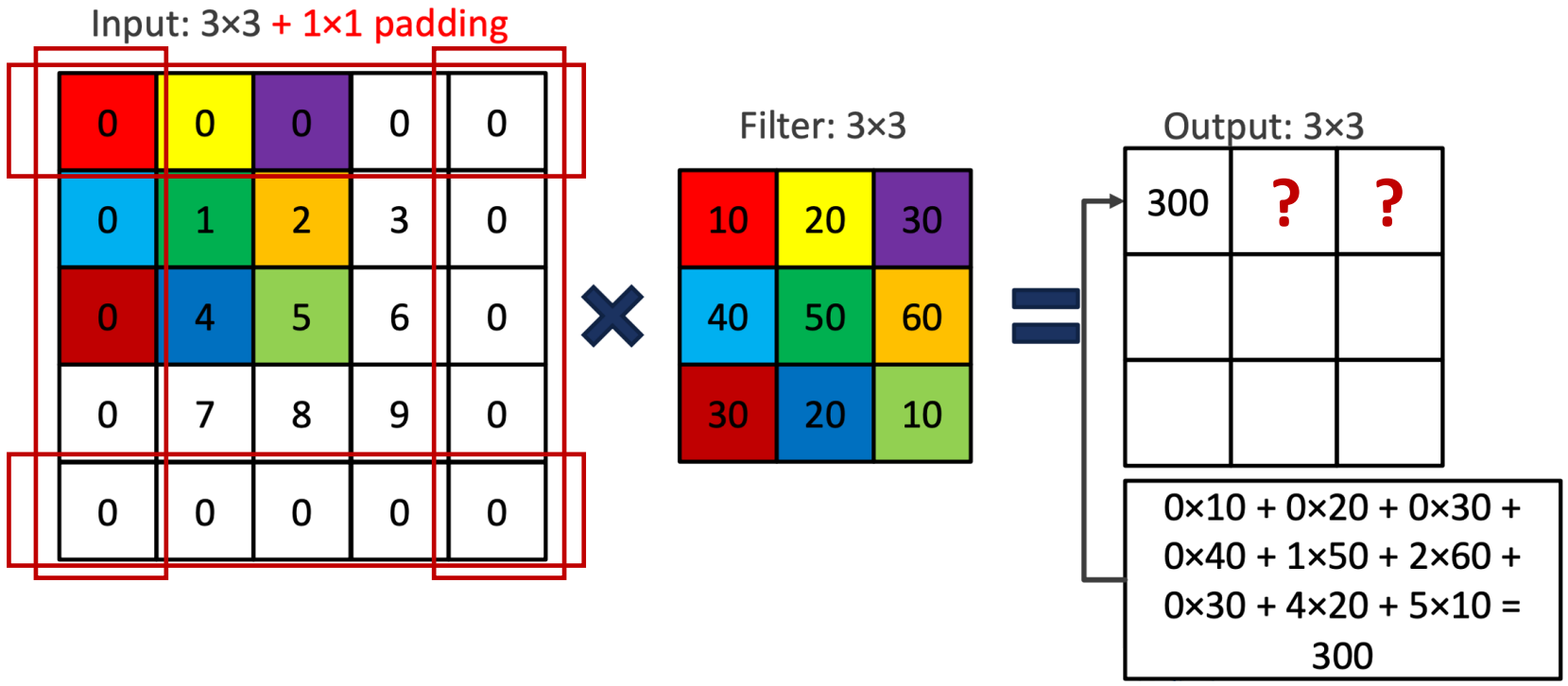
Information at conners are “lost”.

Size of the new image S shrinks after convolution.

Padding

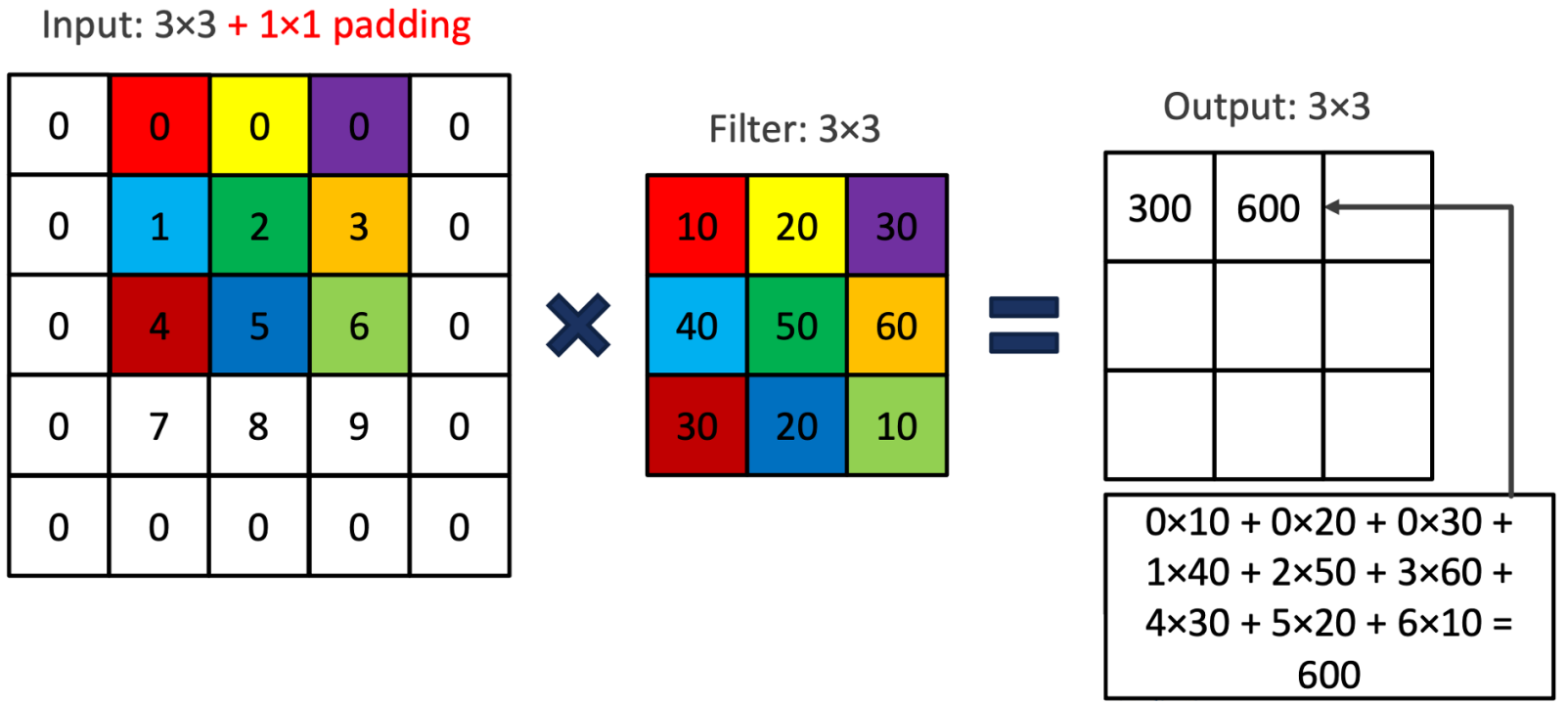
- Zero pad the borders
- Padding keeps the dimension of input and output matrix the same





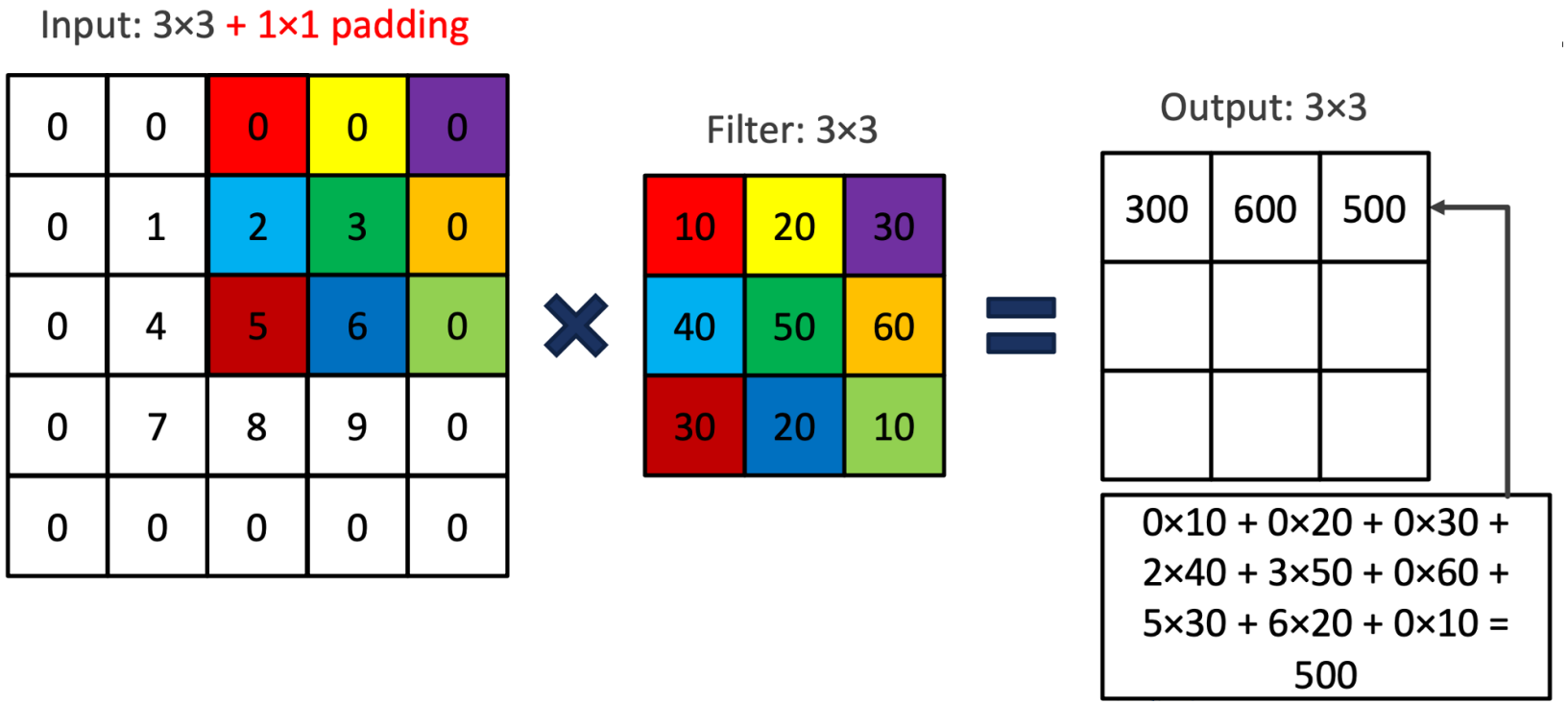
Padding

- Zero pad the borders
- Padding keeps the dimension of input and output matrix the same



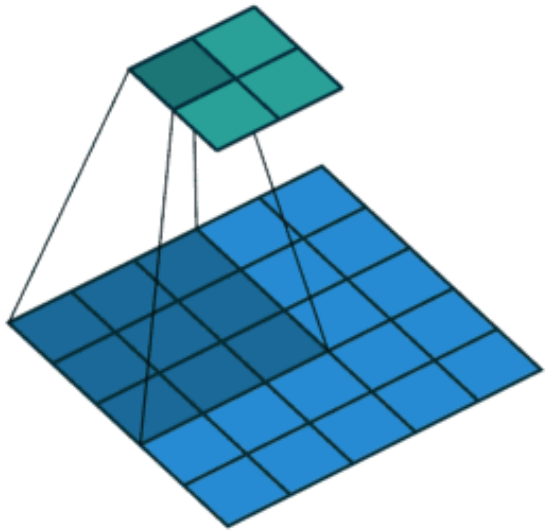
Padding

- Zero pad the borders
- Padding keeps the dimension of input and output matrix the same

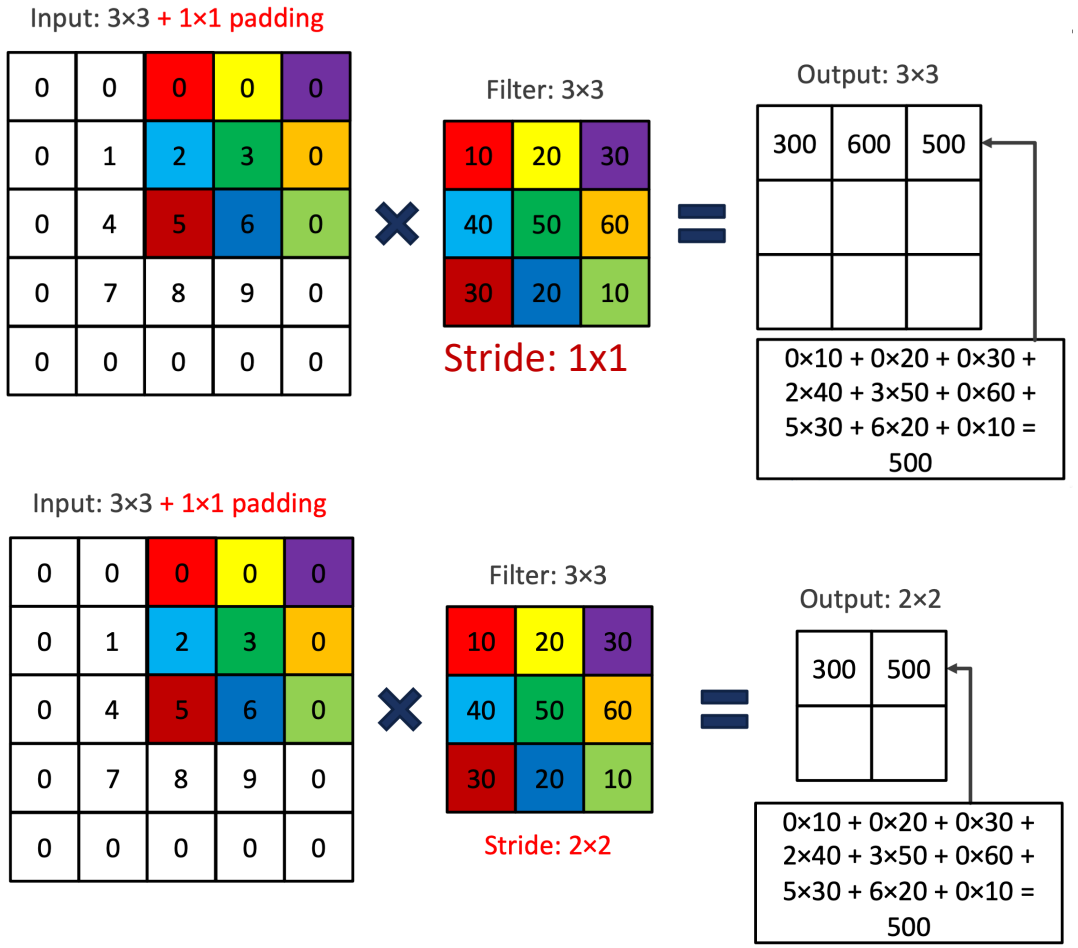


Stride

- **Stride**: the number of pixels shifts over the input matrix.



Stride: 2x2



Padding and Stride

- Padding: to avoid loss of information and the output has the same size as input.
- Stride: to compress information and the output has smaller size than input.

Output Size

- Input size: $n_h \times n_w$; kernel size $k_h \times k_w$; padding size $p_h \times p_w$; stride: $s_h \times s_w$

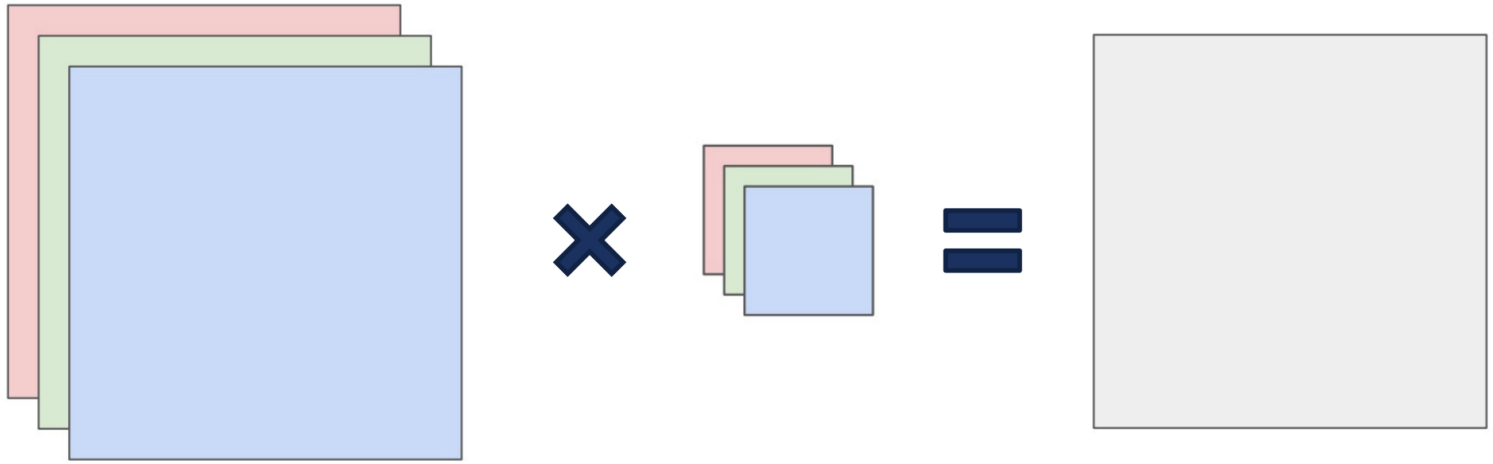
- The output size is:

$$\left\lfloor \frac{n_h + 2p_h - k_h}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{n_w + 2p_w - k_w}{s_w} + 1 \right\rfloor$$

- Example:
  Input size: 16×16; kernel size 3×3; padding size 1×1; stride: 2×2
  What is the output size? $\left\lfloor \frac{16 + 2 - 3}{2} + 1 \right\rfloor \times \left\lfloor \frac{16 + 2 - 3}{2} + 1 \right\rfloor = 8 \times 8$

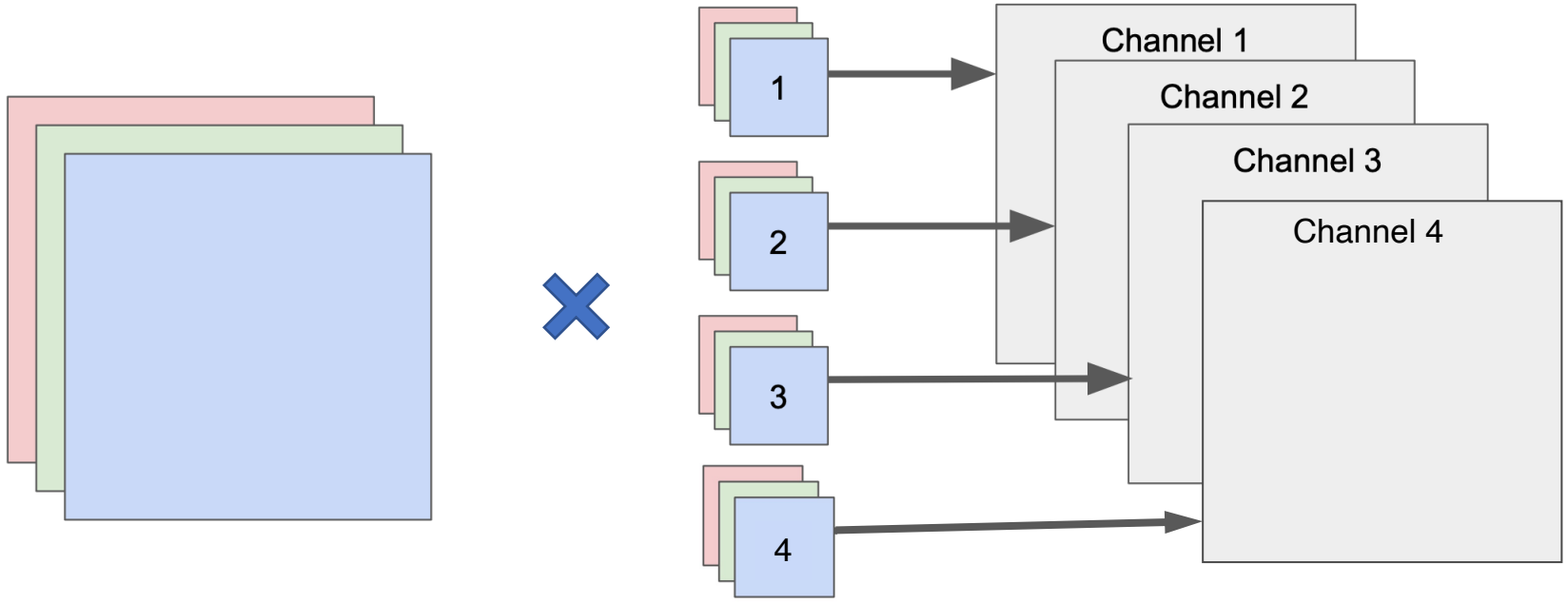
Channel and Depth

- The depth of the kernel is always the same as the channel of the input image.
- For an RGB image, there are three channels: Red, Green, and Blue



Channel and Depth

- The depth of the feature map (number of kernels) is a hyperparameter
- It corresponds to the number of filters we would like to use, each learning to look for something different in the input.

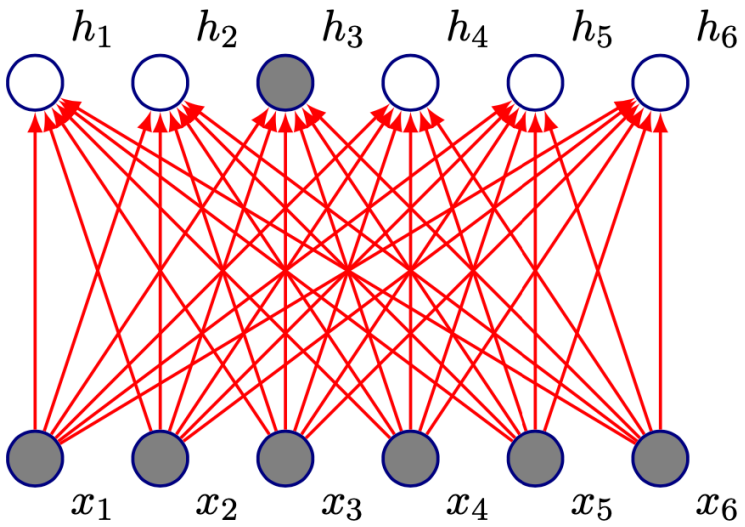


Why Convolution? Benefits/Motivations

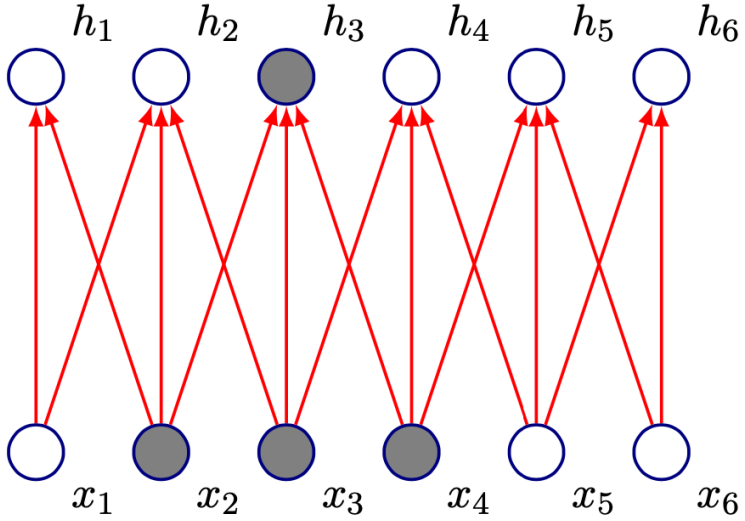
- **Sparse Connectivity**
  - Input: $55\times55\times3$, output: $55\times55\times96$.
  - If we adopt a fully connected layer, the number of parameters for one single layer is:
$$(55\times55\times3+1)\times55\times55\times96=2{,}635{,}670{,}400$$
  - Now, if we use 96 11×11 kernels with 5×5 padding and 1×1 stride.
  - We can reduce the number of parameters to
$$96\times(3\times11\times11+1)=34{,}944$$
  - Benefits: less parameters to learn, more efficient, and preserves spatial structure.

Why Convolution? Benefits/Motivations

- **Sparse Connectivity**



Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity.



Kernel of size 3, moved with stride of 1. h_3 only depends on x_2, x_3, x_4.

Why Convolution? Benefits/Motivations

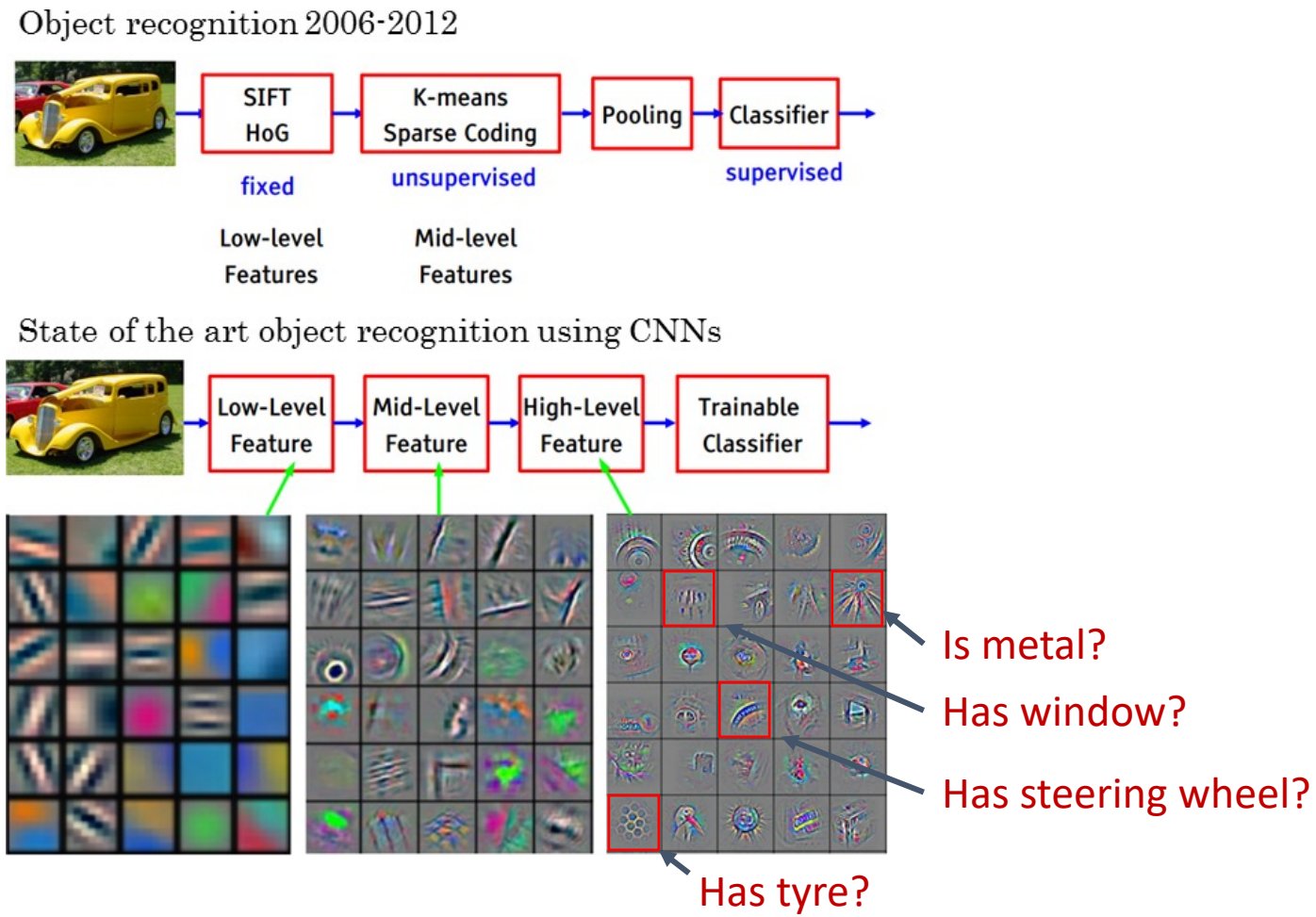
- **Parameter Sharing**

  We have been using this assumption
  (recall that we used the same kernels for all locations:

  If one kernel is useful at some spatial position (x_1, y_1),
  it should also be useful at a different position (x_2, y_2).

  - Benefits: invariance to translations.

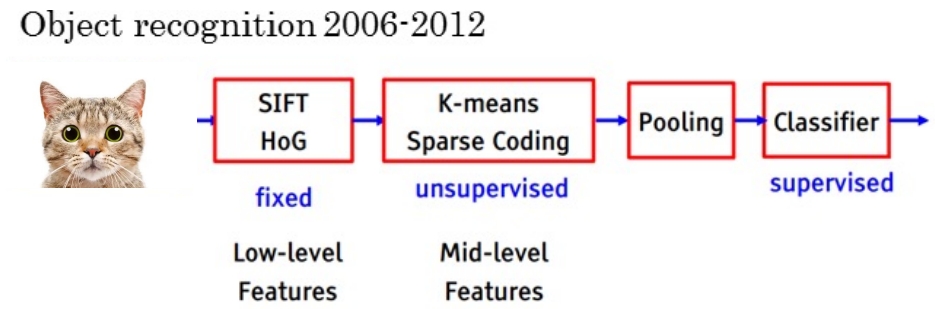
CNNs vs. Traditional Methods

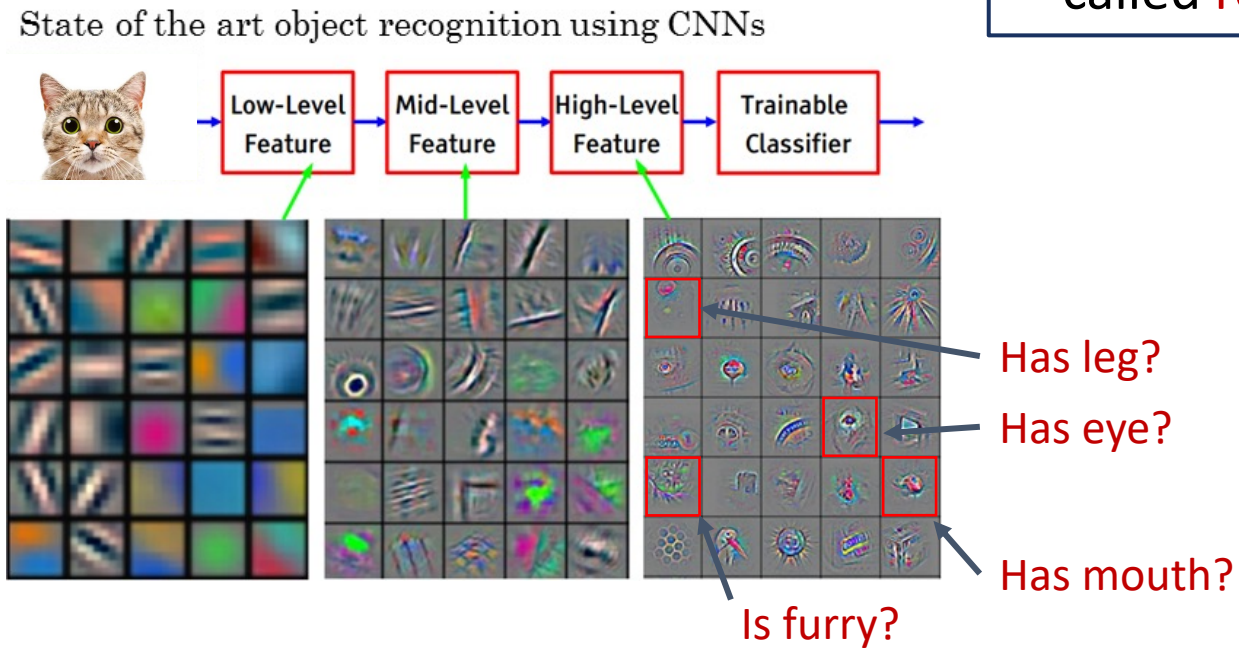


High-level feature contains semantic information, indicating if this image has certain components or not.

Image source: https://rsipvision.com/exploring-deep-learning/

CNNs vs. Traditional Methods





High-level features are also called representations.

High-level feature contains semantic information, indicating if this image has certain components or not.

Image source: https://rsipvision.com/exploring-deep-learning/

Receptive Field

- Receptive field: the size of the region in the input that produces the feature.
- It is a measure of association of an output feature to the input region.
- Increasing number of layers is a straightforward way to increase receptive field.

- Is there any other way to increase the receptive field?

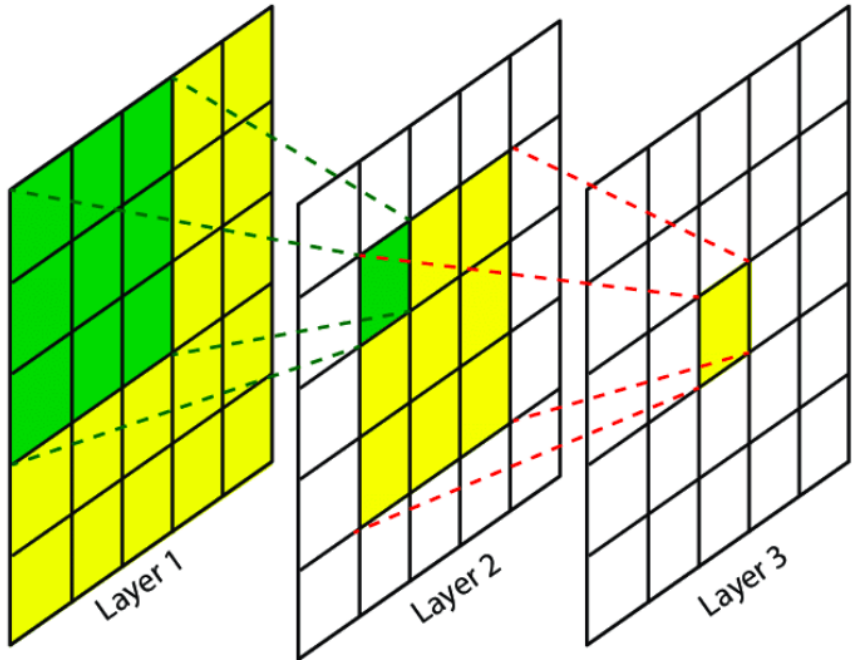
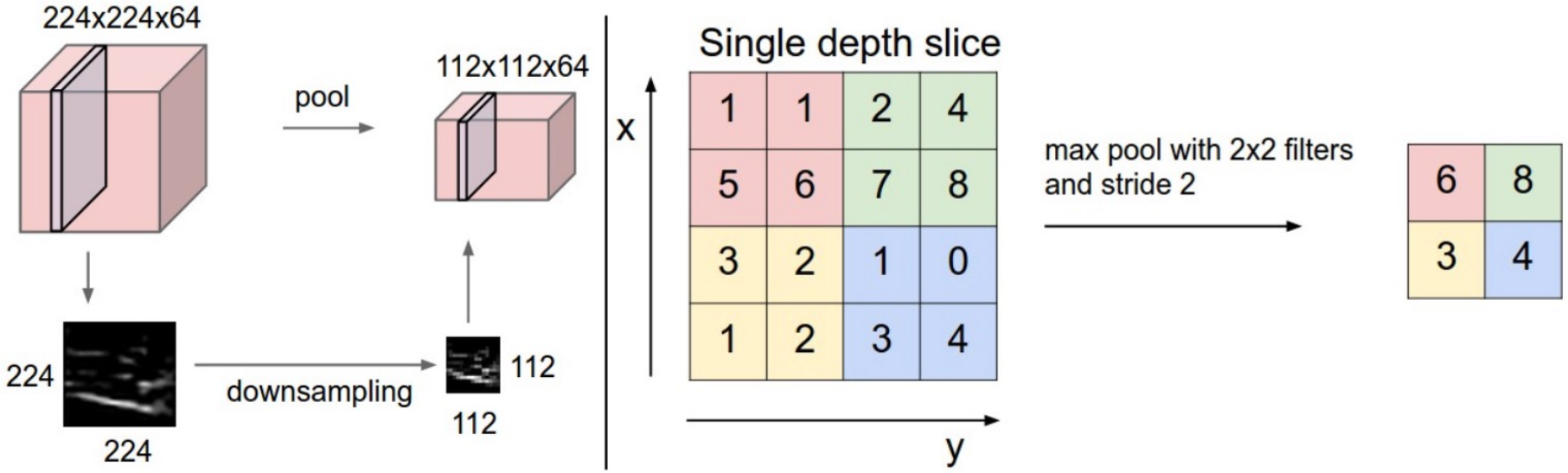


Image source: cs231n.github.io/convolutional-networks

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

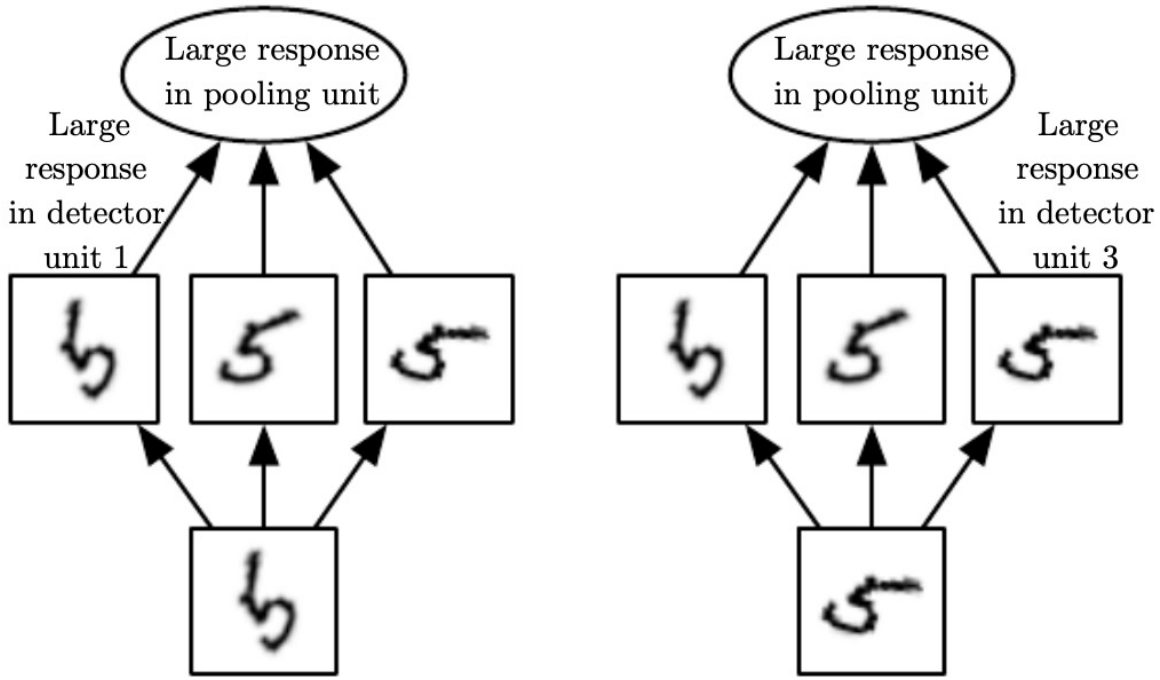


- Commonly used: max pooling, average pooling

Image source: cs231n.github.io/convolutional-networks

Pooling

- Benefits:
  - pooling increases the receptive field.
  - pooling introduces invariance.



Example of invariance

- Each filter attempts to match a slightly different orientation of the 5.
- When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit.
- The max pooling unit then has a large activation regardless of which pooling unit was activated.

Image source: cs231n.github.io/convolutional-networks

Convolutional Layer

- A typical convolutional layer contains three stages:
  - Convolution stage: apply convolution with a set of kernels (still linear).
  - Detector stage: apply nonlinear activation functions.
  - Pooling stage: further modify the layer output.

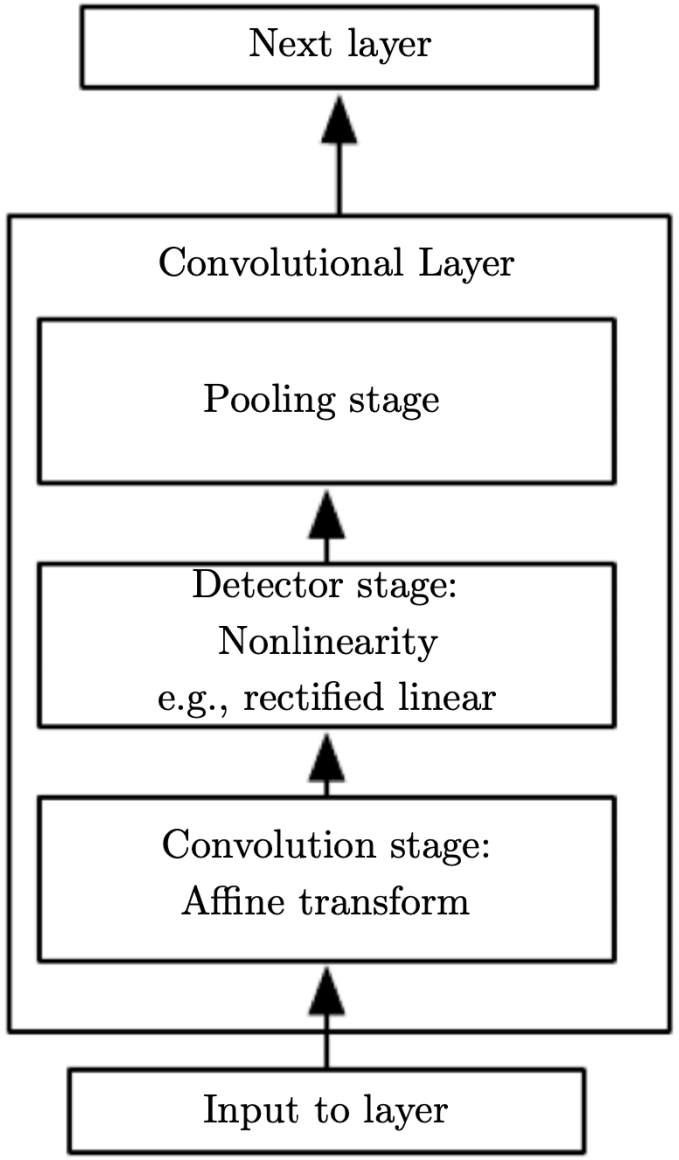
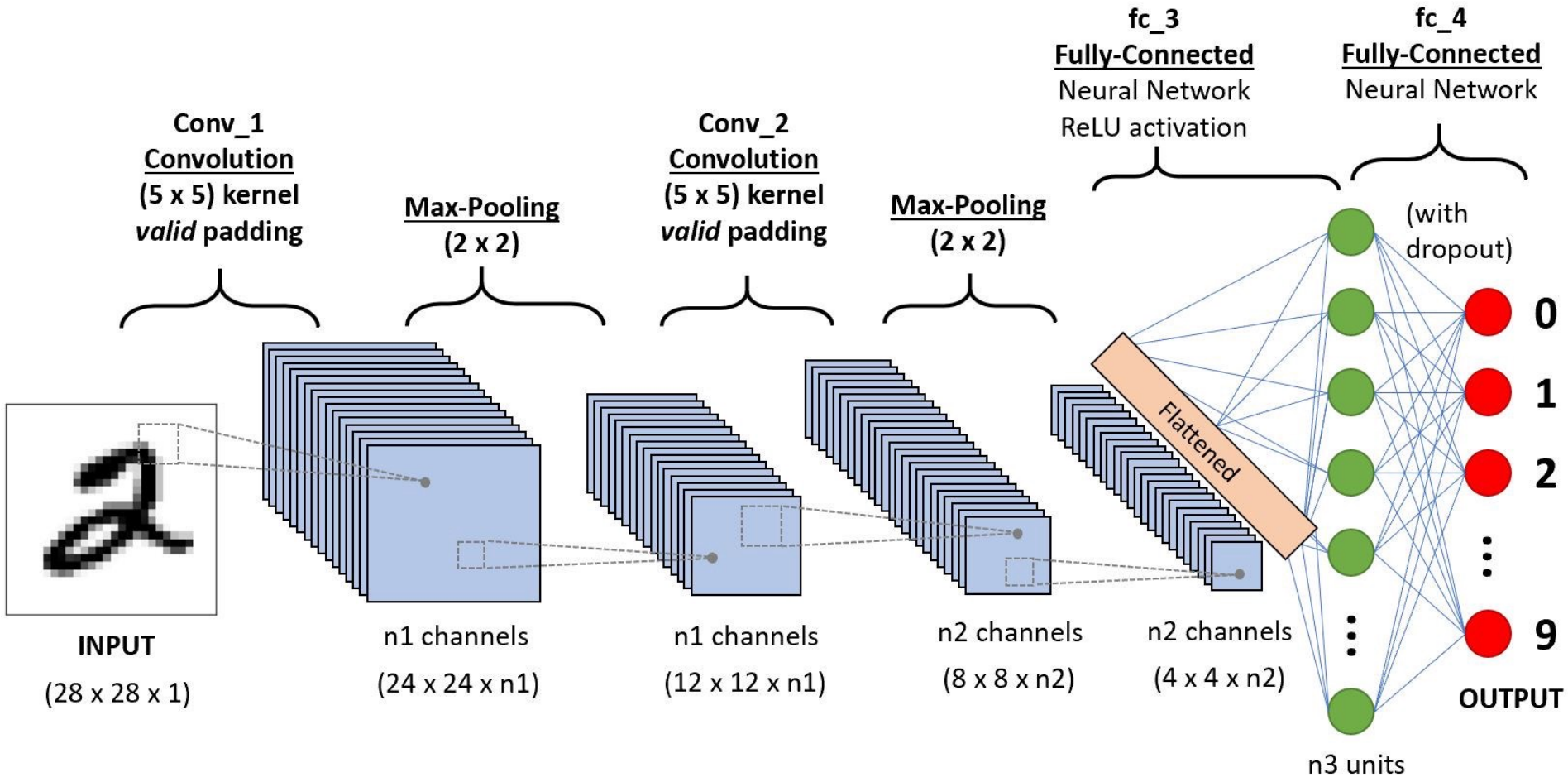


Image source: cs231n.github.io/convolutional-networks

CNN Architecture

Stacking convolutional layers and FC layers to build a convolutional network:



CNN Architecture

- We can also use global average pooling (GAP) to replace flattening.

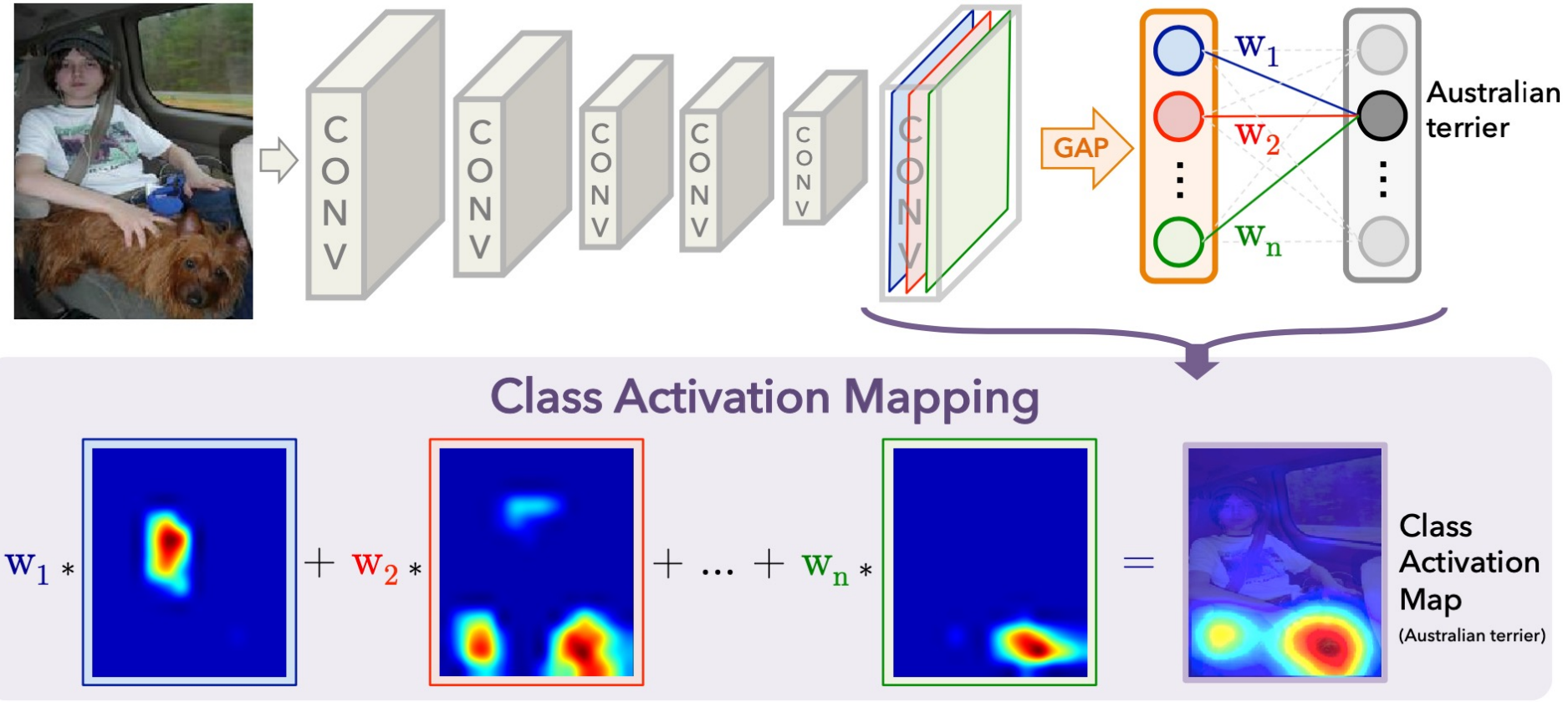


Image source: B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," in CVPR, 2016, vol. 2016-Decem, pp. 2921–2929.

CNN Architecture

- The most common form of a CNN architecture:
  - stacking a few *Conv-ReLU* layers;
  - follows them with *Pooling* layers;
  - repeats this pattern until the image has been merged spatially to a small size;
  - transits to fully-connected layers to produce output (e.g., class scores).

- The most common CNN architecture follows the pattern:

  INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

  - The * indicates repetition, and the indicates an optional pooling layer.
  - N >= 0 (and usually N <= 3), , K >= 0 (and usually K < 3).

CNN Architecture Designs

- INPUT -> FC. *A simple linear classifier. Here N = M = K = 0.*
- INPUT -> CONV -> RELU -> FC.
  - *Only CONV layer and RELU layer are used: learns simple features.*
- INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC.
  - *There is a single CONV layer between every POOL layer.*
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC.
  - *Two CONV layers stacked before every POOL layer.*
  - *This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.*

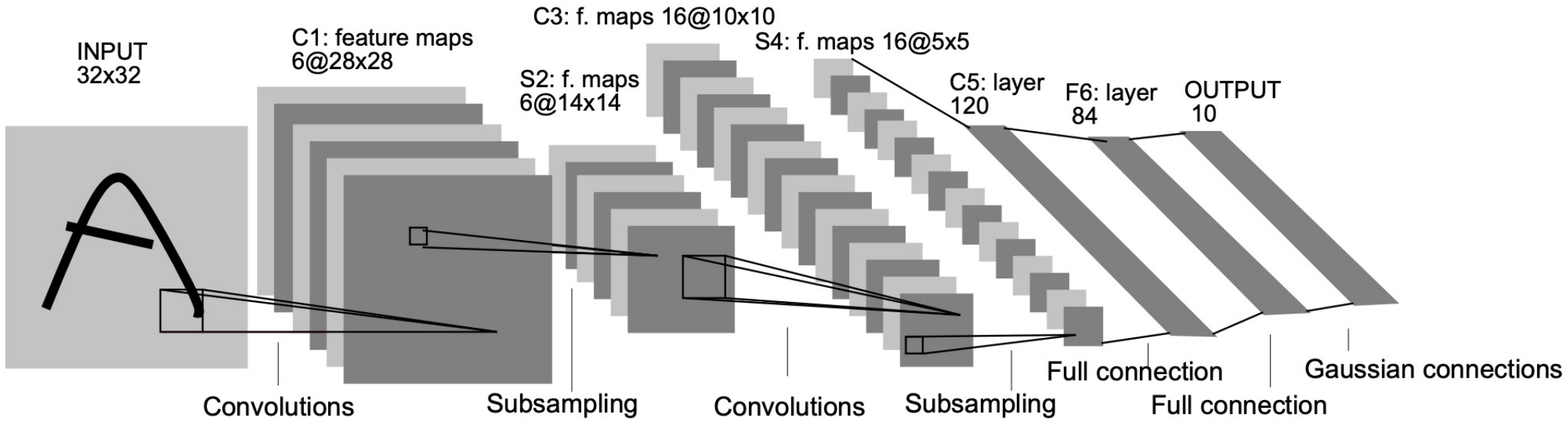
Filter Size

Is a stack of three 3×3 CONV layers equivalent to a single 7×7 CONV layer?

- No. There are several disadvantages for using filters with large size:
  - Less powerful: the neurons would be computing a linear function over the input, while the three stacks of CONV layers contain non-linearities that make their features more expressive.
  - More parameters: if both the input and output of a layer have depth C, 7×7 CONV layer would contain $C\times(7\times7\times C) = 49C^2$, while the three 3×3 CONV layers only contains $3\times C\times(3\times3\times C) = 27C^2$.
- Rule of thumb: stacking CONV layers with tiny filters, instead of having one CONV layer with big filters. This allows us to express more powerful features of the input, and with fewer parameters.

Classical CNN Architectures

Gradient-based learning applied to document recognition
Y LeCun, L Bottou, Y Bengio… - Proceedings of the …, 1998 - ieeexplore.ieee.org
Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns, such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit …
☆ 99 Cited by 30256 Related articles All 38 versions



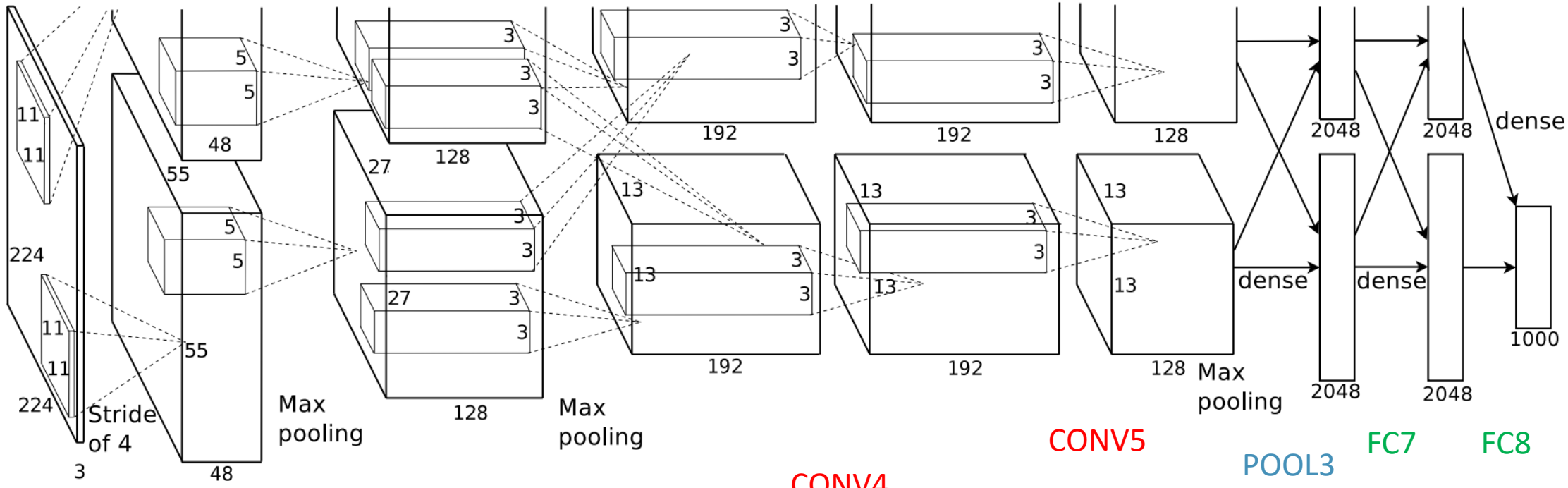
Architecture of LeNet-5: 1998

Image source: LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.

Classical CNN Architectures

We will revisit them in Lab 4

Imagenet classification with deep convolutional neural networks
A Krizhevsky, I Sutskever, GE Hinton - Advances in **neural** …, 2012 - papers.nips.cc
We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is …
☆ 99 Cited by 70691 Related articles All 133 versions »



Architecture of AlexNet: 2012

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In NIPS-12.

1D and 3D CNNs

- For image data, we use 2D CNN.
- The input data and the filter are both 2-dimentional.

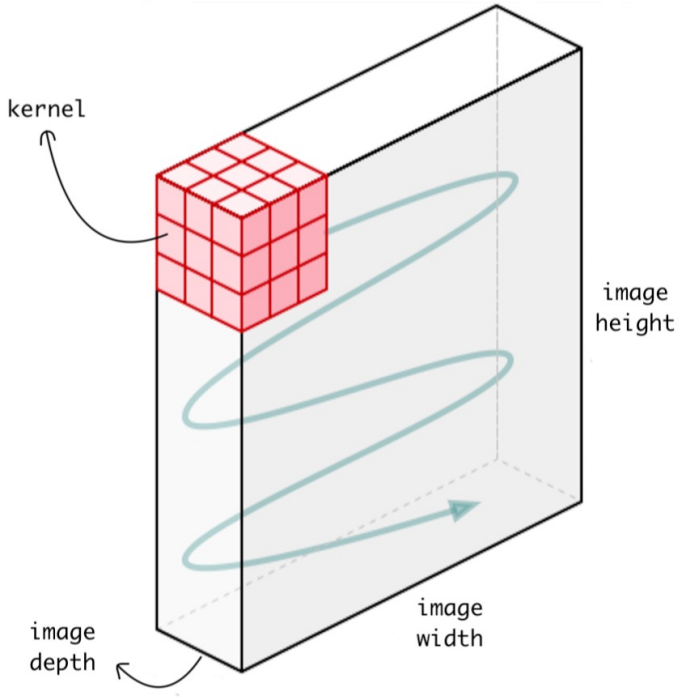
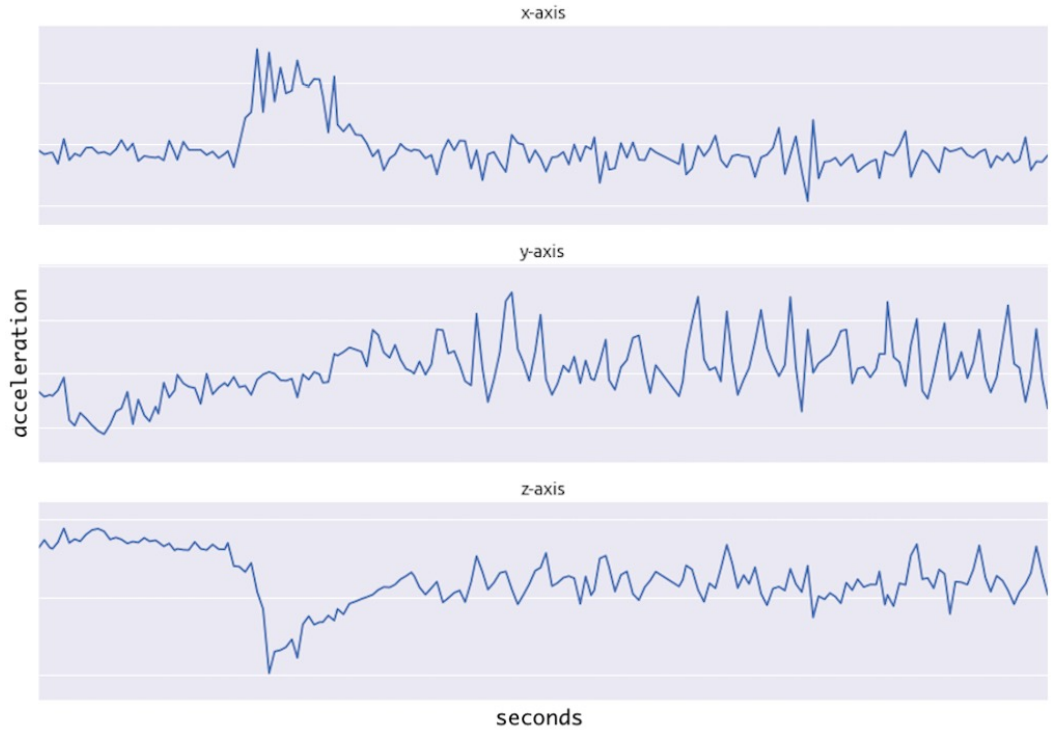


Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

1D and 3D CNNs

- For 1D data like signal or time-series data, we can adopt 1D CNN with 1D filter.
- It is just a sparse and parameter-shared version of MLP.



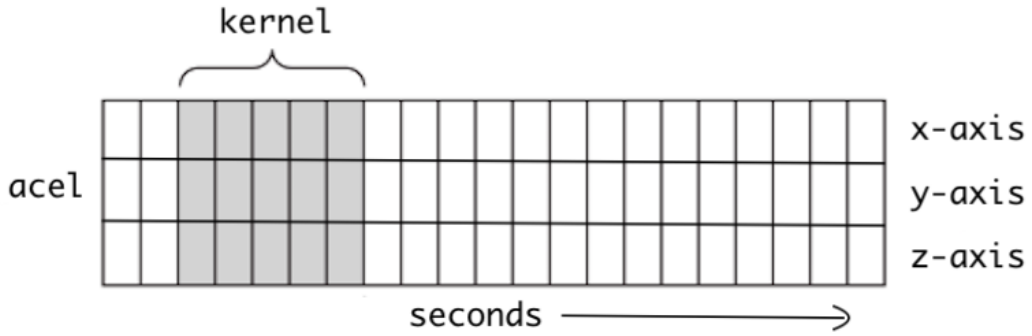
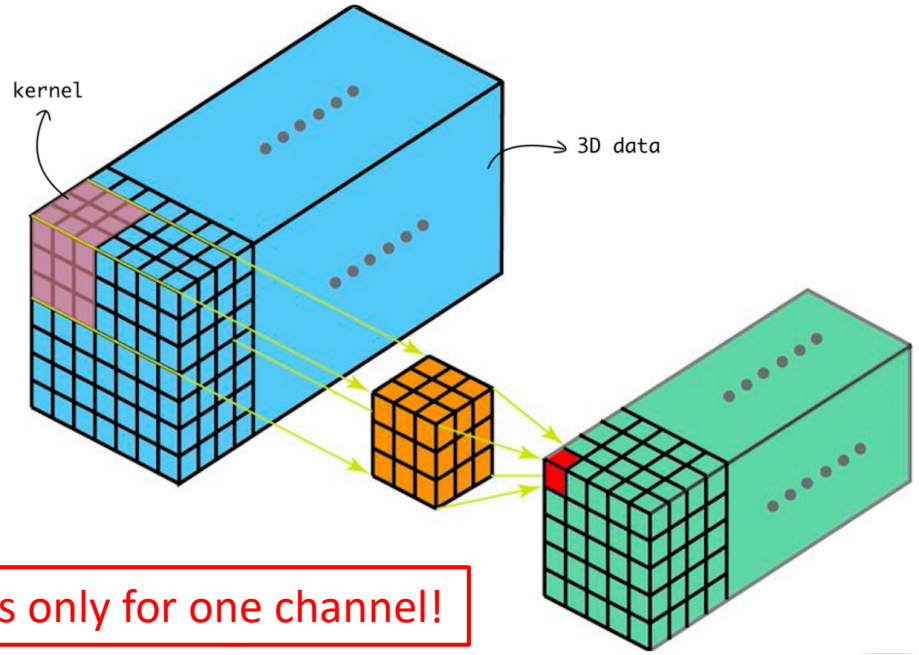


Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

1D and 3D CNNs

- For 3D data like hyperspectral images, medical images, or videos, we can generalize 2D CNNs to 3D.



This is only for one channel!

Image source: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

Recurrent Neural Networks (RNN)

Credits: this section is partially adapted from the following sources
- https://jasonyanglu.github.io/files/lecture_notes/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0_2021/Lecture%207%20Basics%20of%20Recurrent%20Neural%20Networks.pdf

Sequential Data

- The data is ordered as a sequence.
- There are correlations between data in the sequence.
- Example:
  - Speech.
  - Text.
  - Weather.
  - User behavior.
  - …

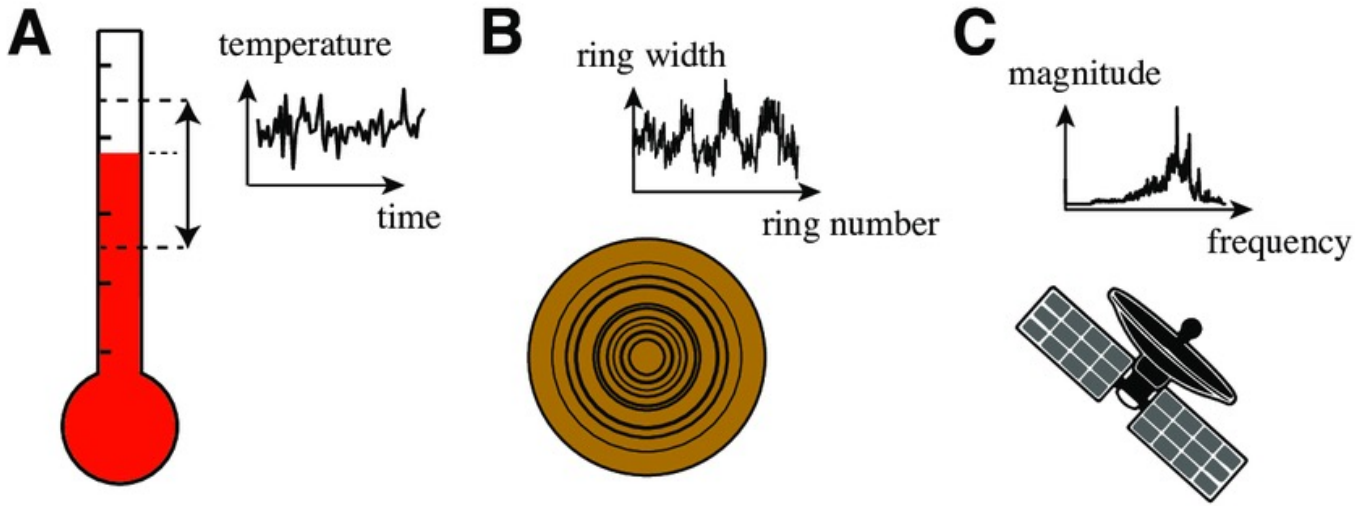
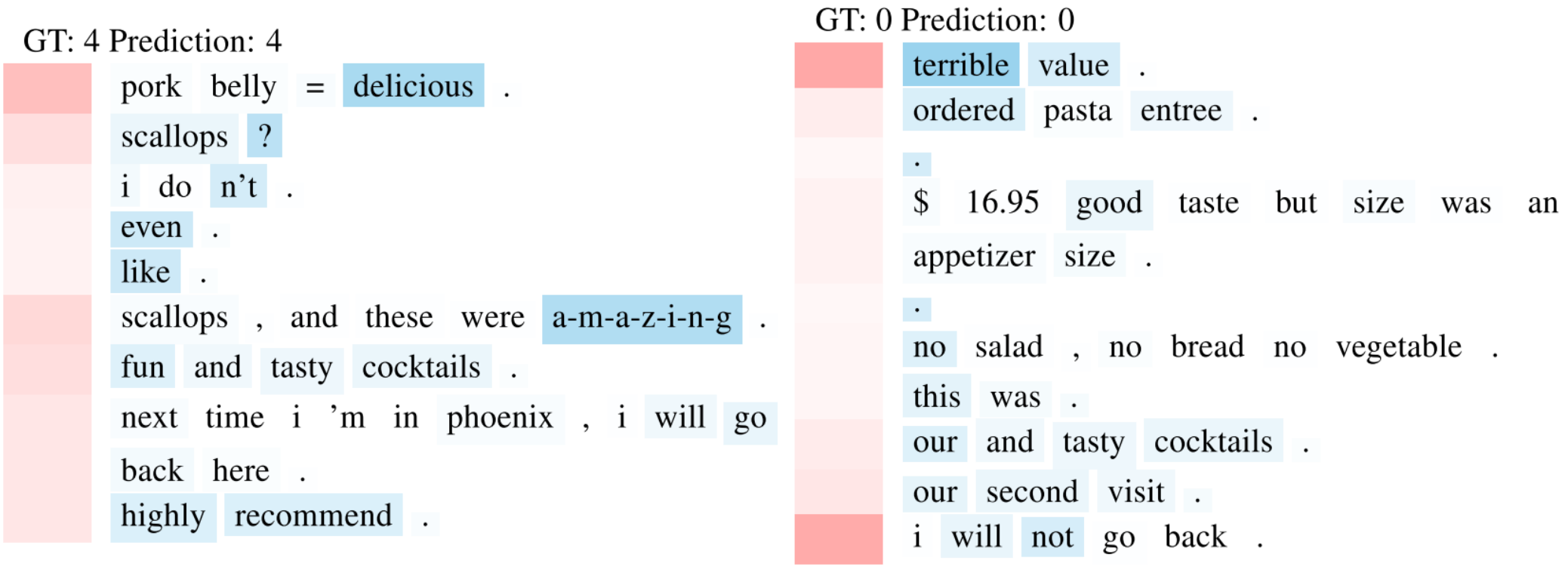


Image source: https://www.researchgate.net/figure/Sequential-data-can-be-ordered-in-many-ways-including-A-temperature-measured-over-time_fig2_320032800

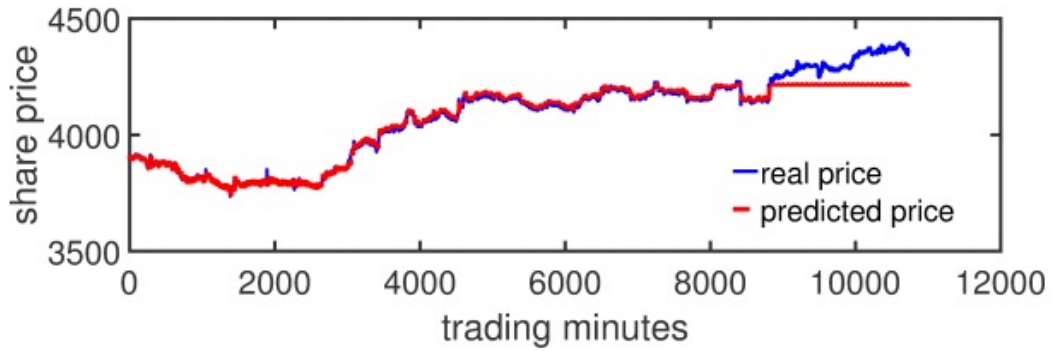
RNN Applications

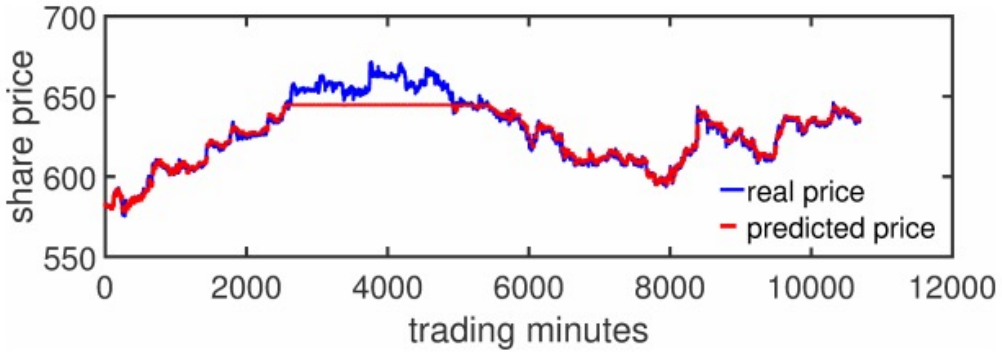


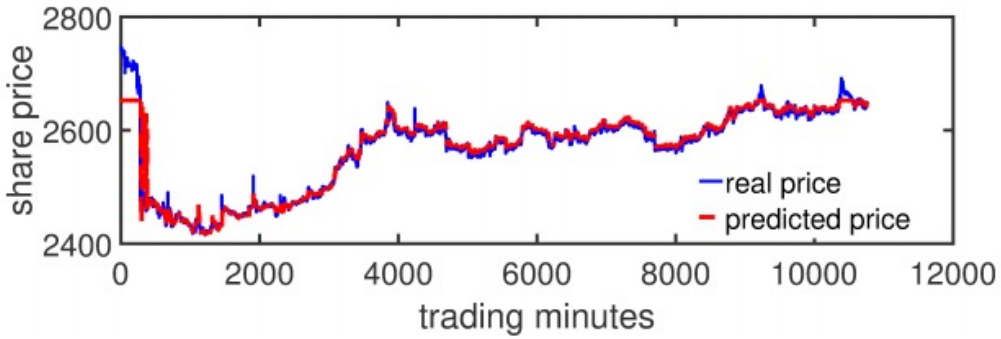
Sentiment analysis

Image source: Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. "Hierarchical attention networks for document classification." In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, pp. 1480-1489. 2016.

RNN Applications



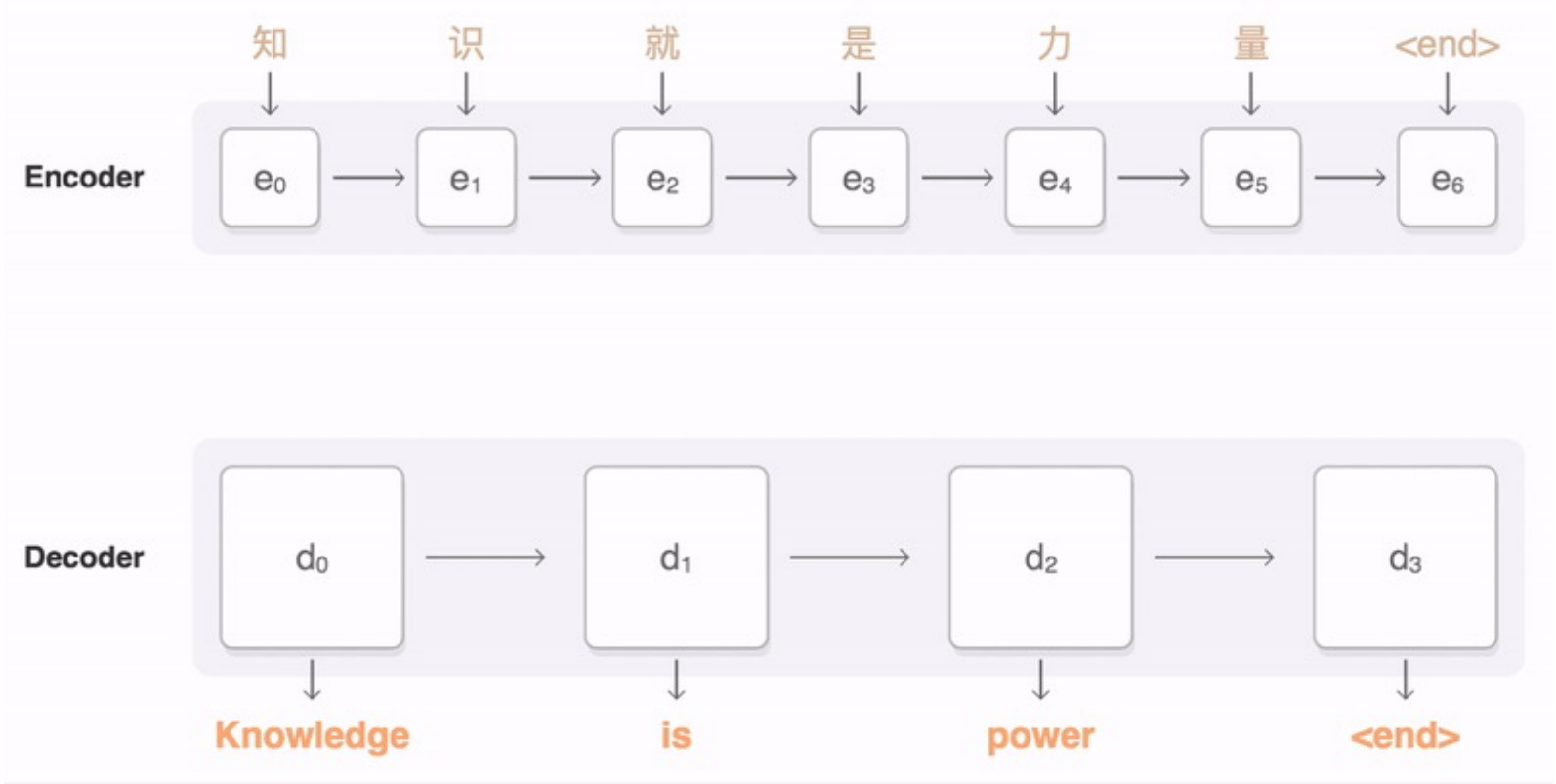




Stock price prediction

Image source: Selvin, Sreelekshmy, R. Vinayakumar, E. A. Gopalakrishnan, Vijay Krishna Menon, and K. P. Soman. "Stock price prediction using LSTM, RNN and CNN-sliding window model." In 2017 international conference on advances in computing, communications and informatics (icacci), pp. 1643-1647. IEEE, 2017.

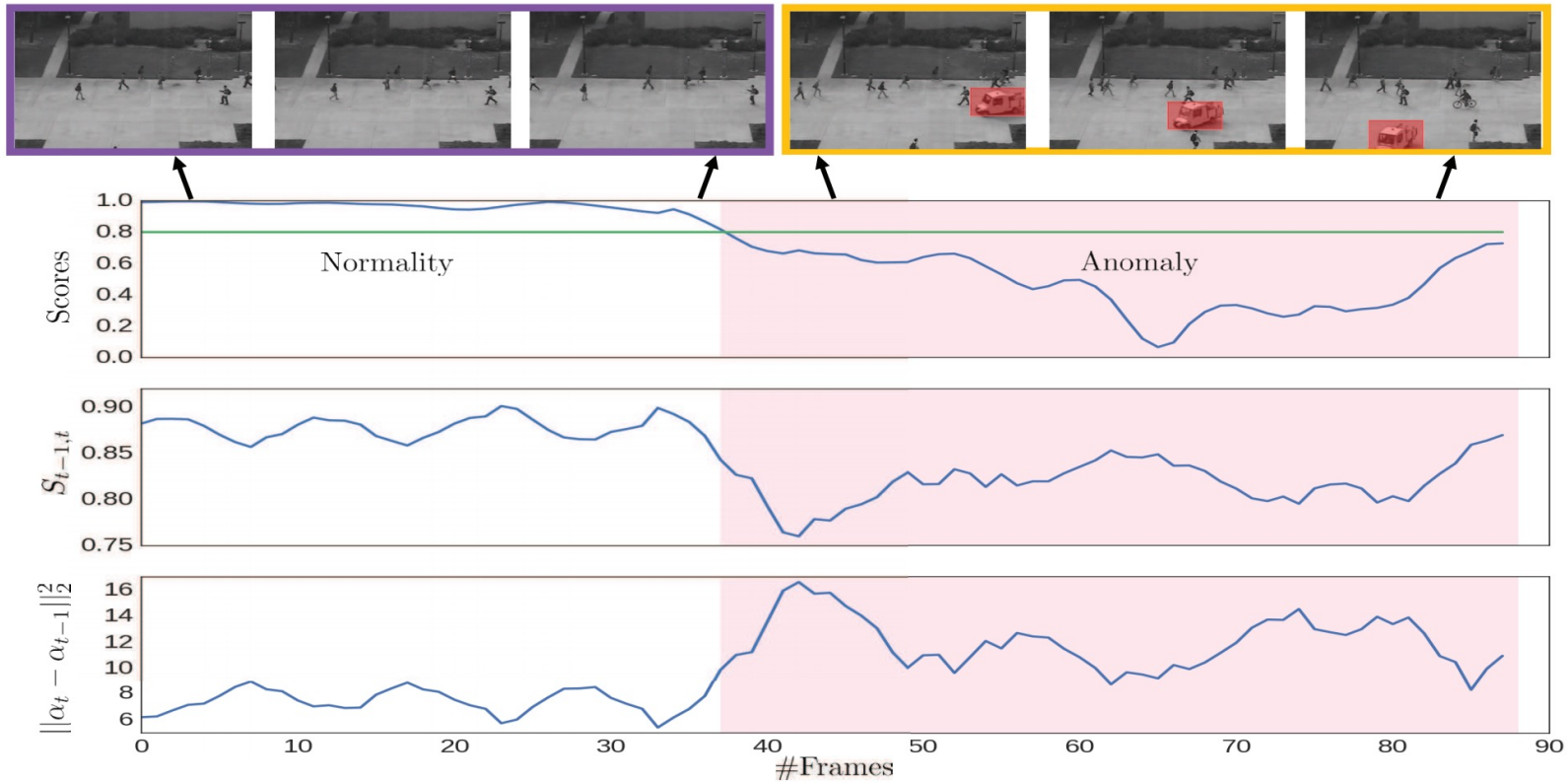
RNN Applications



Machine translation by seq2seq model

Image source: https://google.github.io/seq2seq/

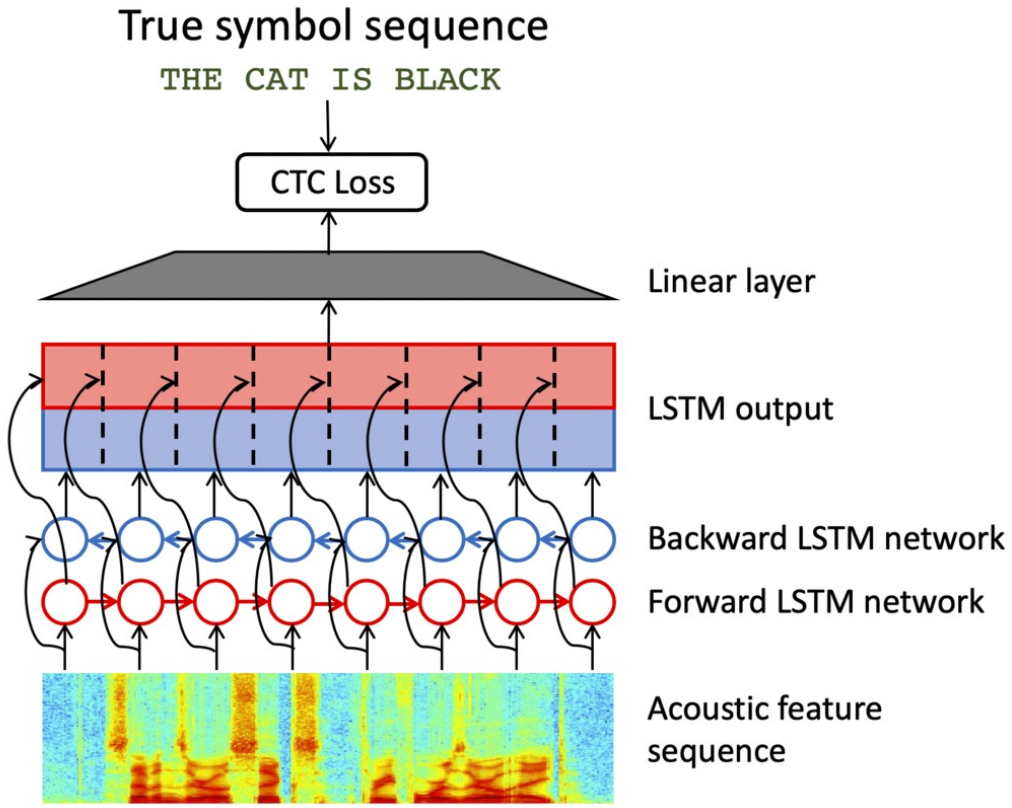
RNN Applications



Anomaly detection

Image source: Luo, Weixin, Wen Liu, and Shenghua Gao. "A revisit of sparse coding based anomaly detection in stacked rnn framework." In Proceedings of the IEEE International Conference on Computer Vision, pp. 341-349. 2017.

RNN Applications



Speech recognition

Image source: https://www.ibm.com/blogs/research/2019/10/end-to-end-speech-recognition/

RNN Applications

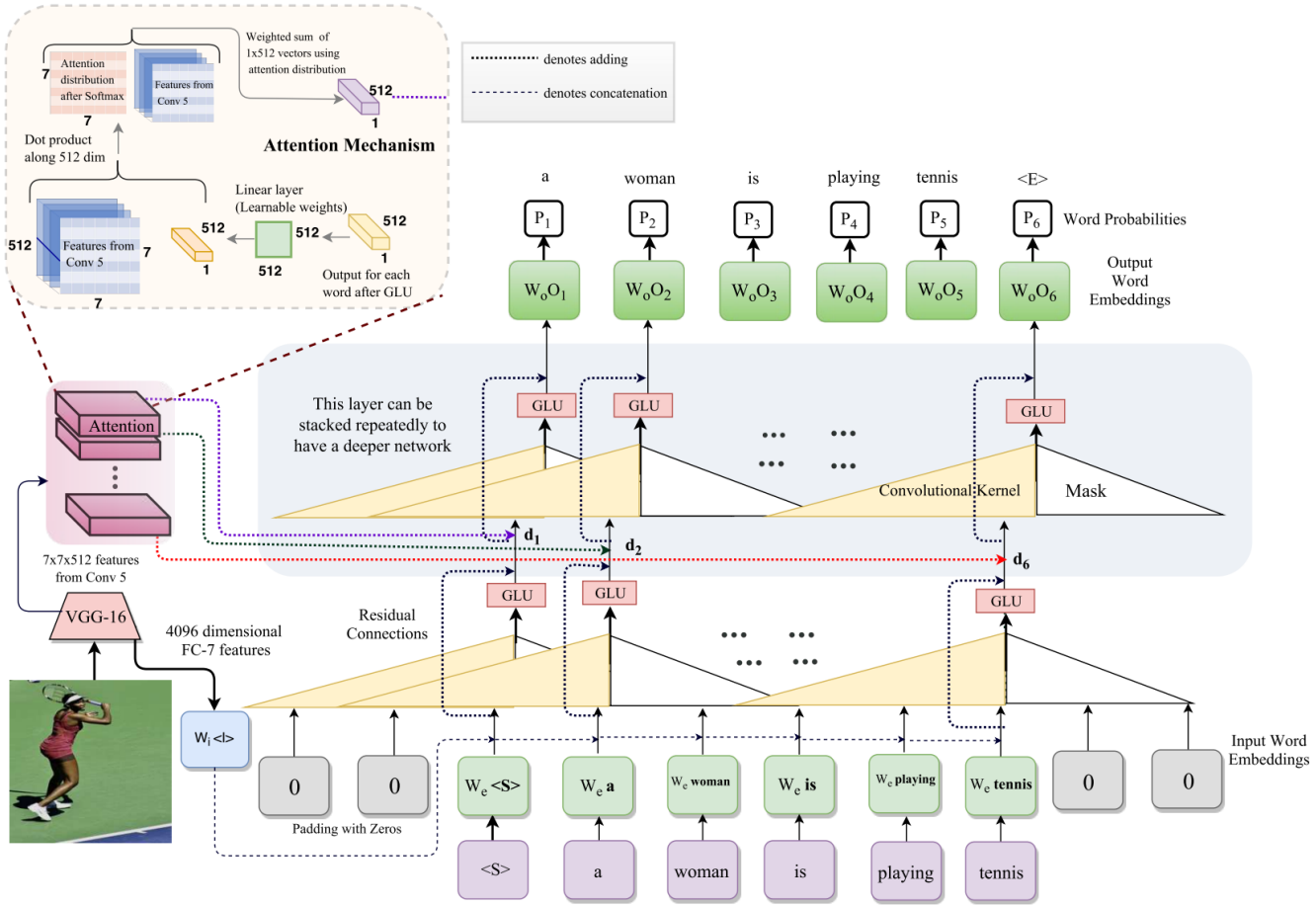


Image Captioning

Image source: Aneja, Jyoti, Aditya Deshpande, and Alexander G. Schwing. "Convolutional image captioning." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5561-5570. 2018.

Tasks with Sequential Data

Task	Example of Input	Example of Output
Speech recognition	Audio recording	Text in English
Language translation	Text in English	Text in Chinese
Image captioning	Image pixels	Short description in English
Sentiment classification	Product review	Positivity score
Stock price prediction	Historical stock prices	Future stock price
	Sequential data	Non-sequential data

Sequential Data

- Usually, we represent each sequential data sample as

$$X = [\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(t)}, \dots, \boldsymbol{x}^{(\tau)}]$$

  where the index t is the time step, τ is number of time steps in total, and each $\boldsymbol{x}^{(t)}$ is a d-dimensional vector.

- $\boldsymbol{x}^{(t)}$ usually depends on $\boldsymbol{x}^{(t')}$ $(t' < t)$.

- We can vectorize X (becomes a vector of size $d \times \tau$) and feed it into an MLP.
- Is it a good idea? What is the potential issue?

Sequential Data

Problems of training sequential data with MLP:

- In any sequence, nearby items are related, and the order of items matters. Fully connected layer treats every item independently, unless it learns otherwise.
- Sequence length can vary across examples within the same task. For MLP, shape of input and output is determined at design time.

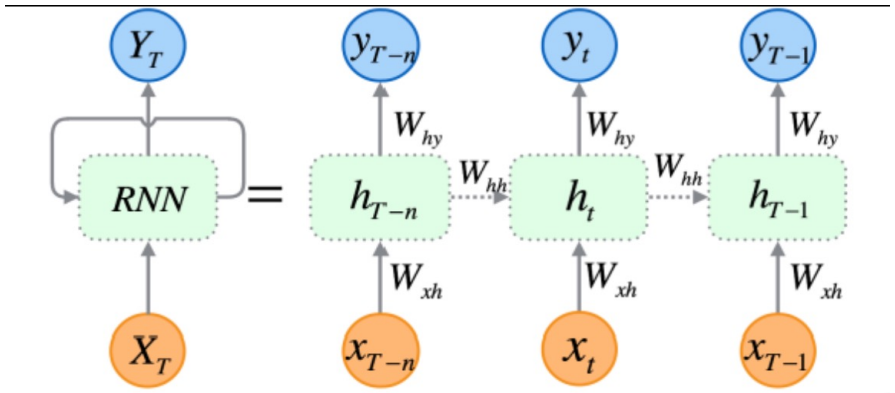
Recurrent Neural Network (RNN)

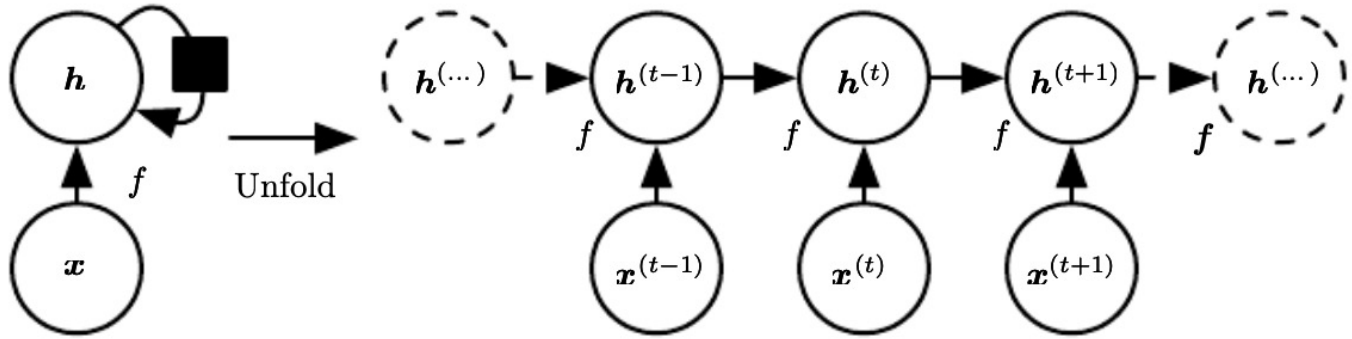
- A typical Recurrent Neural Network (RNN) uses the following equation:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, \theta)$$

  where $\boldsymbol{h}^{(t)}$ is the hidden state at time step t.

- The network typically learns to use $\boldsymbol{h}^{(t)}$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t.





Recurrent Neural Network (RNN)

Two major advantages:

- Regardless of the sequence length, the learned model always has the same input size.
- It is possible to use the same transition function f with the same parameters at every time step.

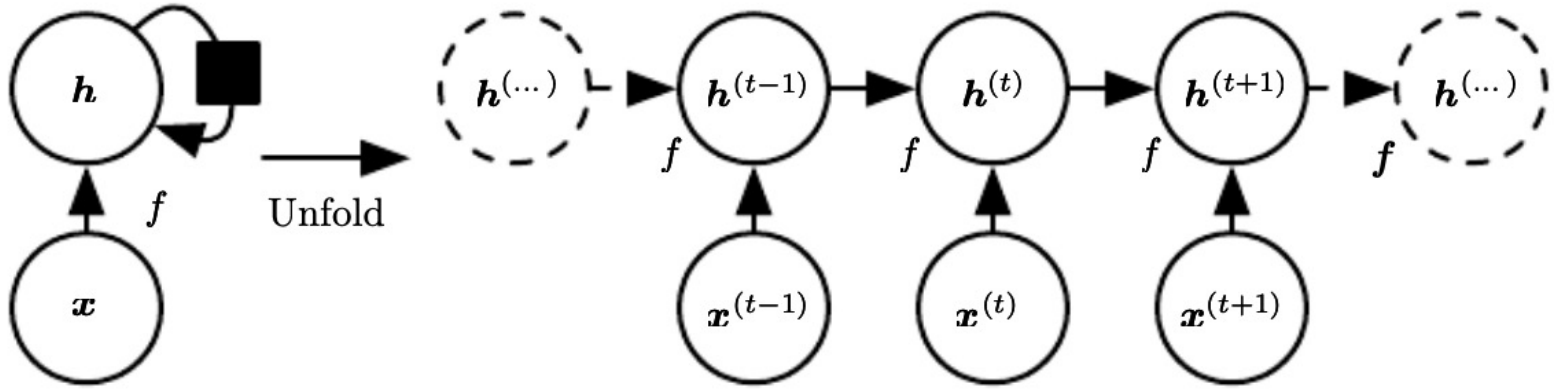


Image source: Figure 10.2, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Recurrent Neural Network (RNN)

- It follows the same idea of parameter sharing as CNN.
- It is possible to learn a single model f that operates on all time steps and all sequence lengths, rather than needing to learn a separate model for all possible time steps.
- Learning a single, shared model allows generalization to sequence lengths that did not appear in the training set.

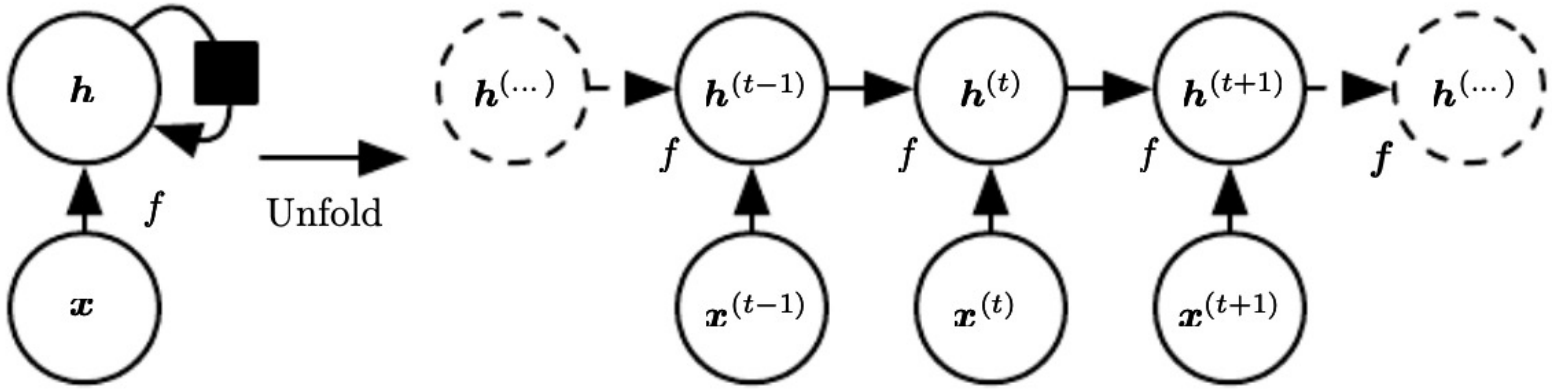
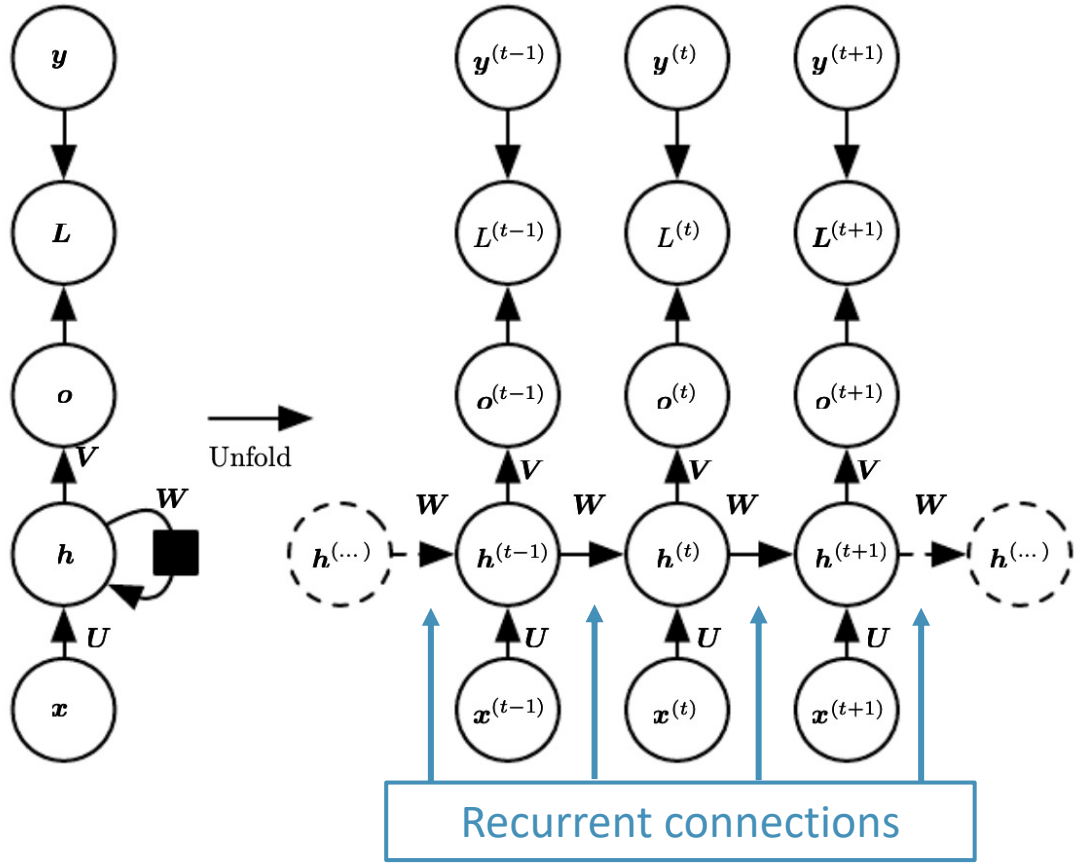


Image source: Figure 10.2, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Recurrent Neural Network (RNN)

- $\boldsymbol{U}$: a weight matrix for input-to-hidden connections.
- $\boldsymbol{W}$: a weight matrix for hidden-to-hidden recurrent connections.
- $\boldsymbol{V}$: a weight matrix for hidden-to-output connections.
- $\boldsymbol{o}^{(t)}$: the corresponding output of input $\boldsymbol{x}^{(t)}$.
- $L^{(t)}$: the loss measures how far each $\boldsymbol{o}^{(t)}$ is from the corresponding training target $\boldsymbol{y}^{(t)}$.



Vanilla RNN

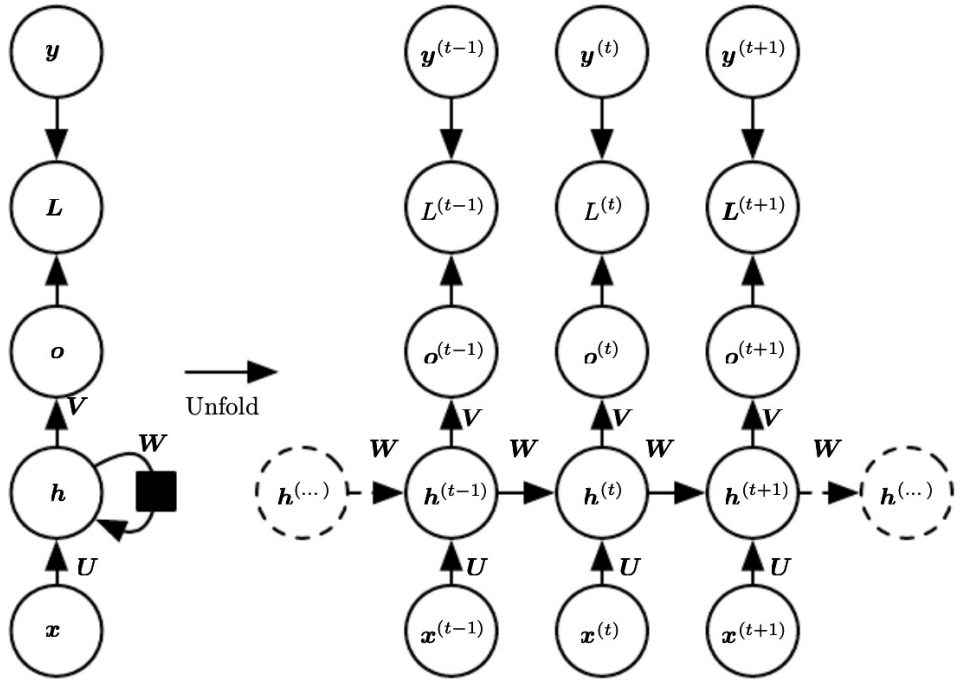
Image source: Figure 10.2, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Vanilla RNN: Forward Propagation

- Forward propagation begins with a specification of the initial state $\boldsymbol{h}^{(0)}$.
- For each time step from $t = 1$ to $t = \tau$, we apply the following update equations:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right),$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{o}^{(t)}\right),$$
$$L^{(t)} = J\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right).$$

Loss function for classification



Vanilla RNN

Image source: Figure 10.2, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

Vanilla RNN: Backpropagation Through Time

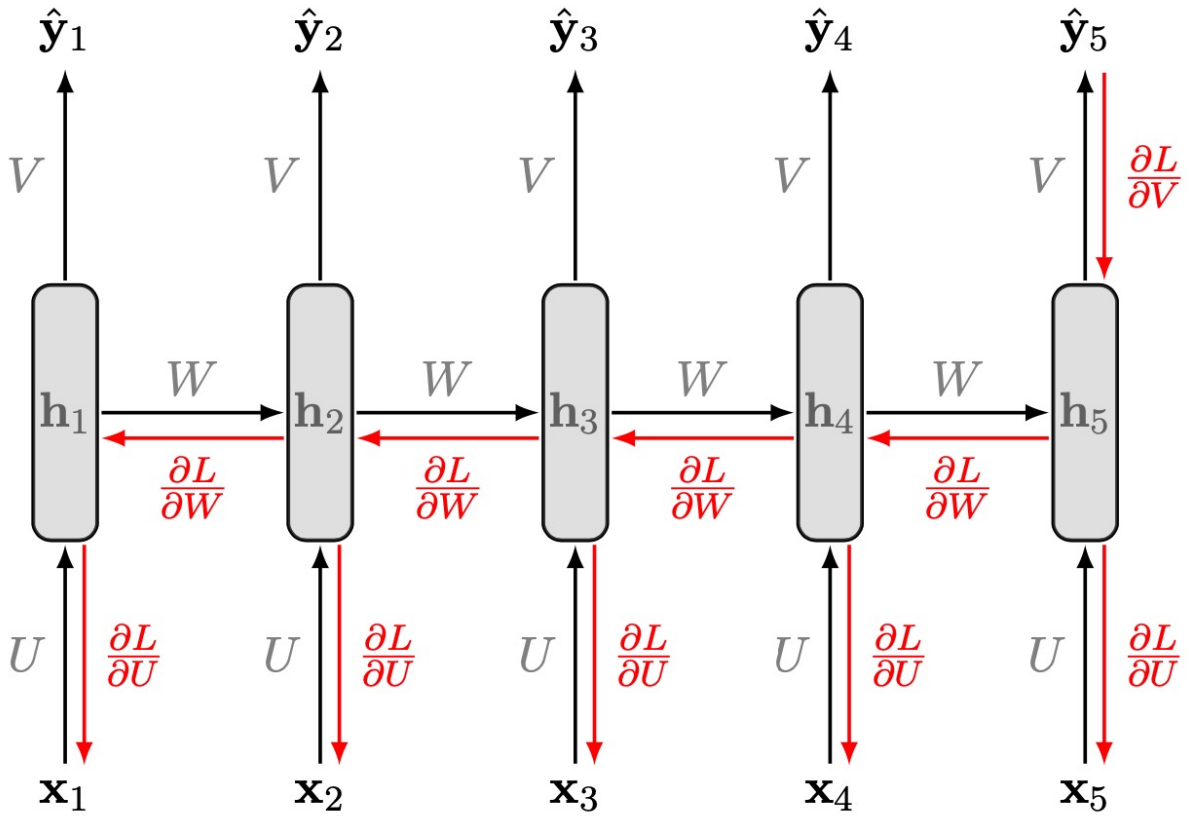
$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right),$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$
$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}\left(\boldsymbol{o}^{(t)}\right),$$
$$L^{(t)} = J\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right).$$

What are parameters to be learned?

$\boldsymbol{b}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{c}$, *and* $\boldsymbol{V}$

- Need to calculate the gradients $\nabla_{\boldsymbol{V}} L$, $\nabla_{\boldsymbol{W}} L$, $\nabla_{\boldsymbol{U}} L$ $\nabla_{\boldsymbol{c}} L$, and $\nabla_{\boldsymbol{b}} L$
- Treat the recurrent network as a usual multilayer network and apply backpropagation on the unfold network.
- We move from the right to left: called Backpropagation Through Time (BPTT).

Vanilla RNN: Backpropagation Through Time



$$\nabla_{\boldsymbol{c}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \,,$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \,,$$

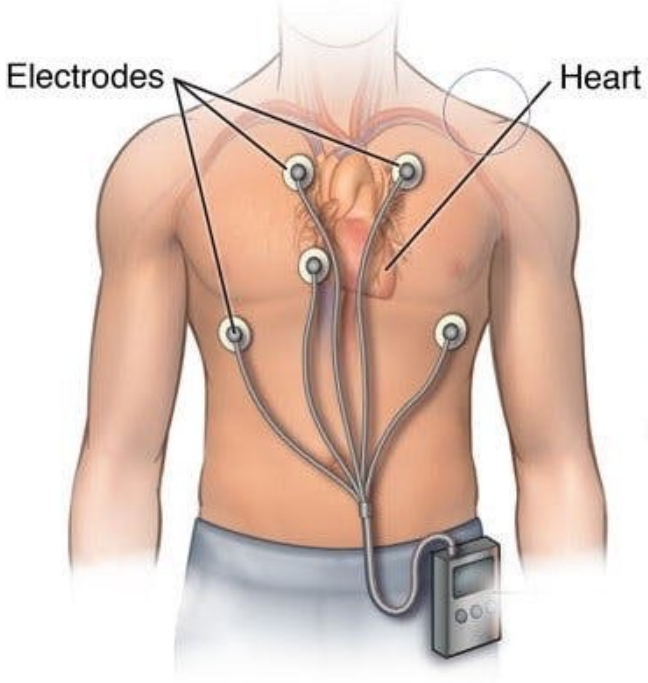
$$\nabla_{\boldsymbol{V}} L = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)^T} \,,$$

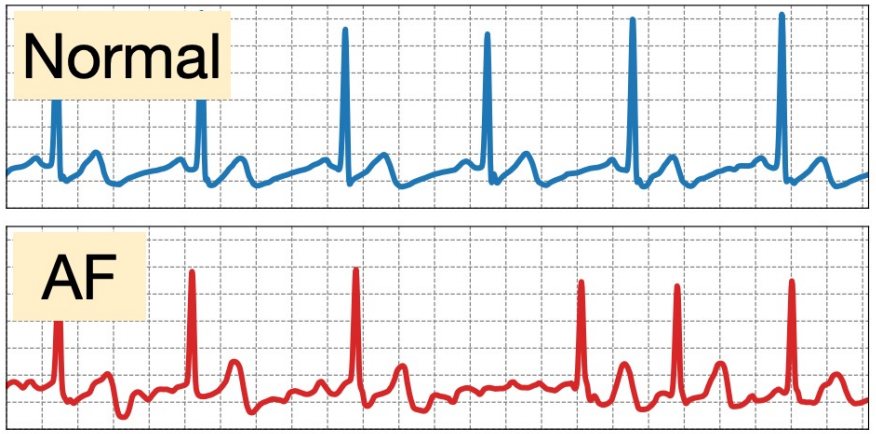
$$\nabla_{\boldsymbol{W}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)^T} \,,$$

$$\nabla_{\boldsymbol{U}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t-1)^T}$$

Reference: Equation 10.22-10.28, Goodfellow, Bengio, and Courville, Deep Learning, Cambridge: MIT press, 2016.

A Real Example: ECG Classification with CNN & RNN

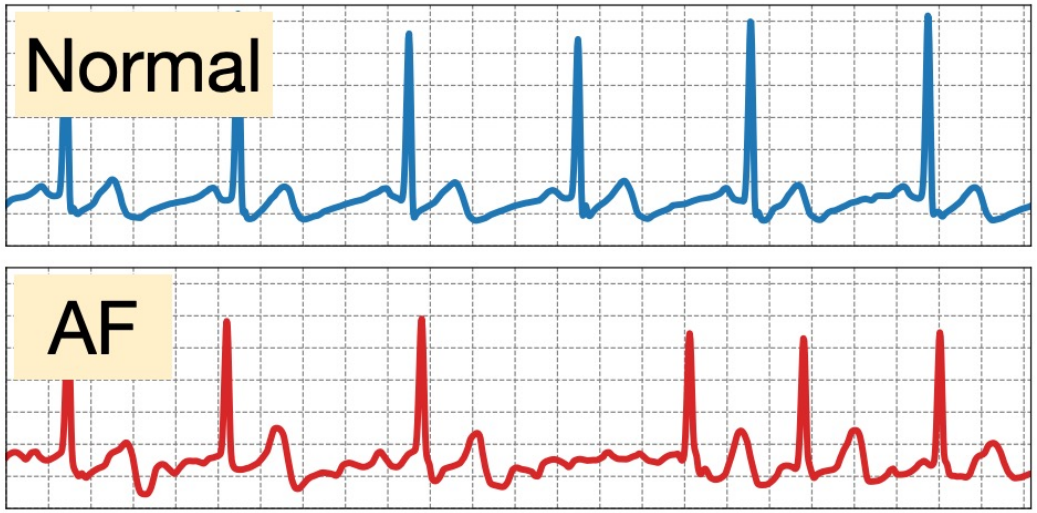




Electrocardiogram (**ECG** or **EKG**) is a recording of the heart's electrical activity.

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

A Real Example: ECG Classification with CNN & RNN



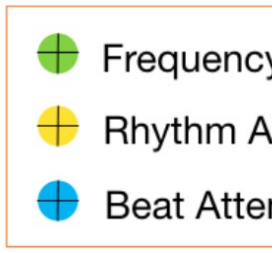
Atrial fibrillation (AF) is an irregular and often very rapid heart rhythm (arrhythmia) that can lead to blood clots in the heart.

Early detection of AF could decrease risk for heart failure and stroke.

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

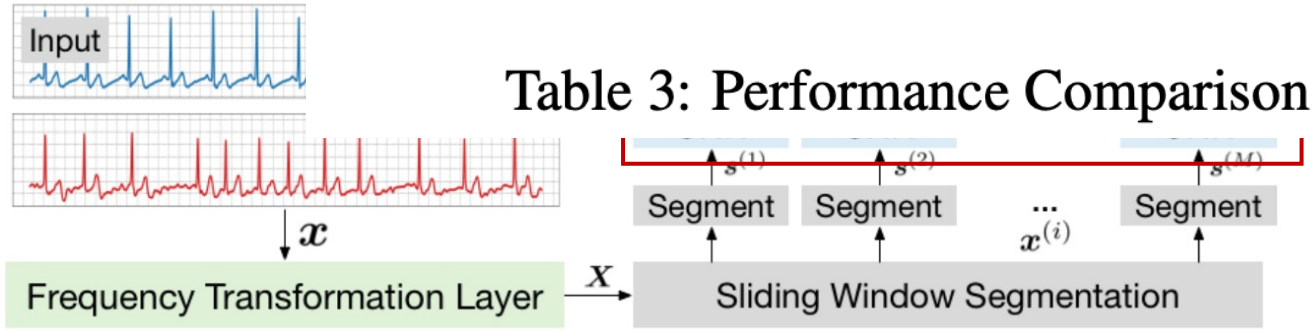
A Real Example: ECG Classification with CNN & RNN

A cutting-edge model called MINA (IJCAI-19):



	ROC-AUC	PR-AUC	F1
ExpertLR	0.9350 ± 0.0000	0.8730 ± 0.0000	0.8023 ± 0.0000
ExpertRF	0.9394 ± 0.0000	0.8816 ± 0.0000	0.8180 ± 0.0000
CNN	0.8711 ± 0.0036	0.8669 ± 0.0068	0.7914 ± 0.0090
CRNN	0.9040 ± 0.0115	0.8943 ± 0.0111	0.8262 ± 0.0215
ACRNN	0.9072 ± 0.0047	0.8935 ± 0.0087	0.8248 ± 0.0229
MINA	**0.9488** ± 0.0081	**0.9436** ± 0.0082	**0.8342** ± 0.0352

Table 3: Performance Comparison on AF Prediction



Extract features using CNN

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

A Real Example: ECG Classification with CNN & RNN

A cutting-edge model called MINA (IJCAI-19):

	ROC-AUC	PR-AUC	F1
ExpertLR	0.9350 ± 0.0000	0.8730 ± 0.0000	0.8023 ± 0.0000
ExpertRF	0.9394 ± 0.0000	0.8816 ± 0.0000	0.8180 ± 0.0000
CNN	0.8711 ± 0.0036	0.8669 ± 0.0068	0.7914 ± 0.0090
CRNN	0.9040 ± 0.0115	0.8943 ± 0.0111	0.8262 ± 0.0215
ACRNN	0.9072 ± 0.0047	0.8935 ± 0.0087	0.8248 ± 0.0229
MINA	**0.9488** ± 0.0081	**0.9436** ± 0.0082	**0.8342** ± 0.0352

Table 3: Performance Comparison on AF Prediction

Hong, Shenda, et al. "MINA: Multilevel Knowledge-Guided Attention for Modeling Electrocardiography Signals." *IJCAI*. 2019.

Another Example: Cardiologist-level arrhythmia detection

Stanford ML Group

Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network

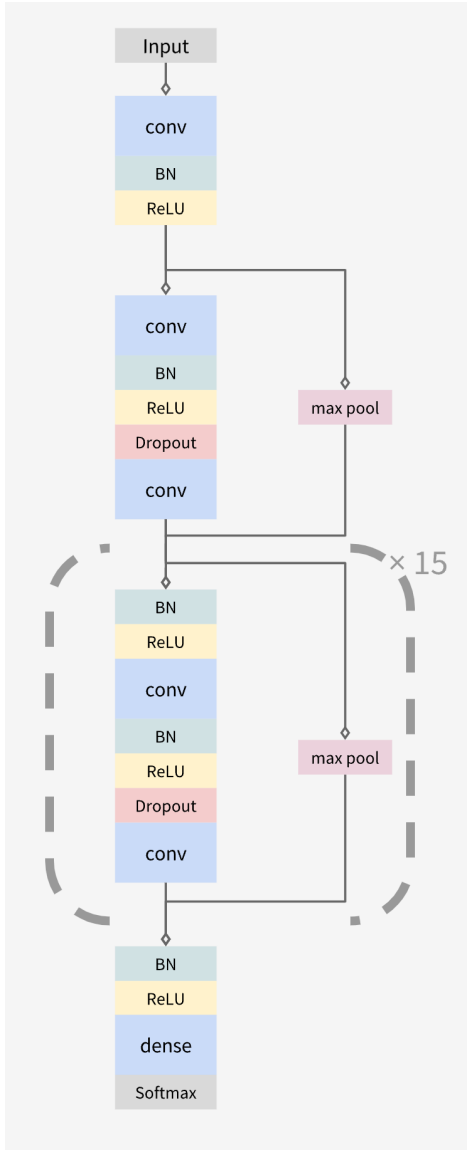
Awni Y. Hannun *, Pranav Rajpurkar *, Masoumeh Haghpanahi *, Geoffrey H. Tison *, Codie Bourn, Mintu P. Turakhia, Andrew Y. Ng

A collaboration between the Stanford Machine Learning Group and iRhythm Technologies

We developed a deep neural network which can diagnose irregular heart rhythms, also known as arrhythmias, from single-lead ECG signals at a high diagnostic performance similar to that of cardiologists.

The electrocardiogram (ECG) is a fundamental tool in the everyday practice of clinical medicine, with more than 300 million ECGs obtained annually worldwide, and is pivotal for diagnosing a wide spectrum of arrhythmias. In a study published in Nature Medicine, we developed a deep neural network to classify 10 arrhythmias as well as sinus rhythm and noise from a single-lead ECG signal, and compared its performance to that of cardiologists.

Can you identify the heart arrhythmia in the above example? The neural network is able to correctly detect AVB_TYPE2. Below, you can see other rhythms which the neural network is successfully able to detect.



1D convolutional deep neural network

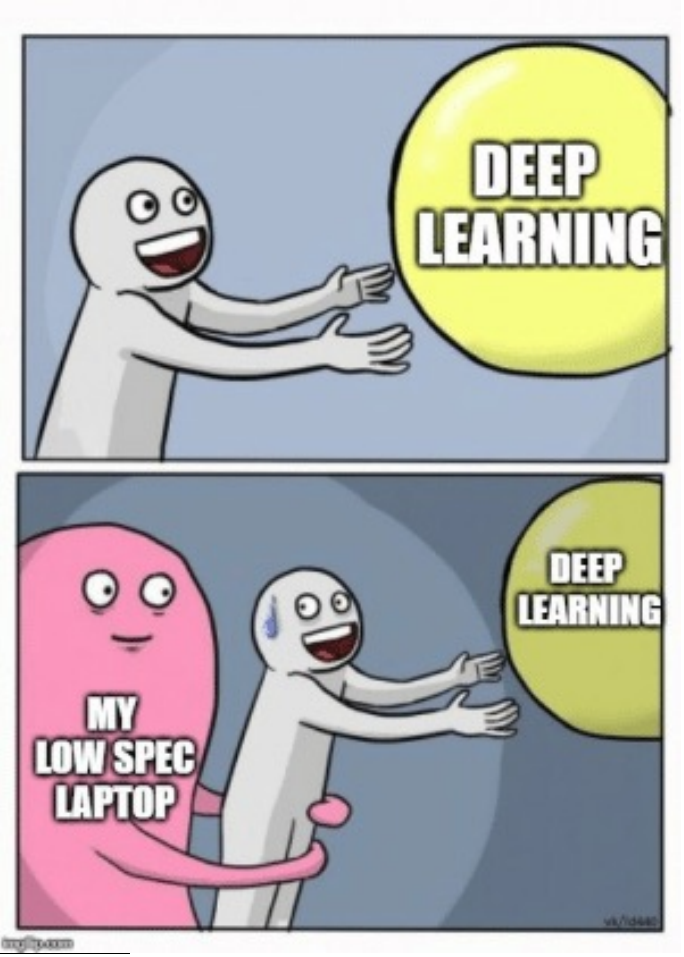
https://stanfordmlgroup.github.io/projects/ecg2/

Pretrained Models

Training a Deep Learning Model Can Be Difficult Sometimes

- Deep learning models is increasingly large.
- Requires time, GPU and huge datasets to train.

- E.g., GPT-3 (a language model in 2020):
  - 175 billion parameters;
  - 45TB of text data for training;
  - Cost of training: >4.6M US dollars.



Deep Models are Growing Fast

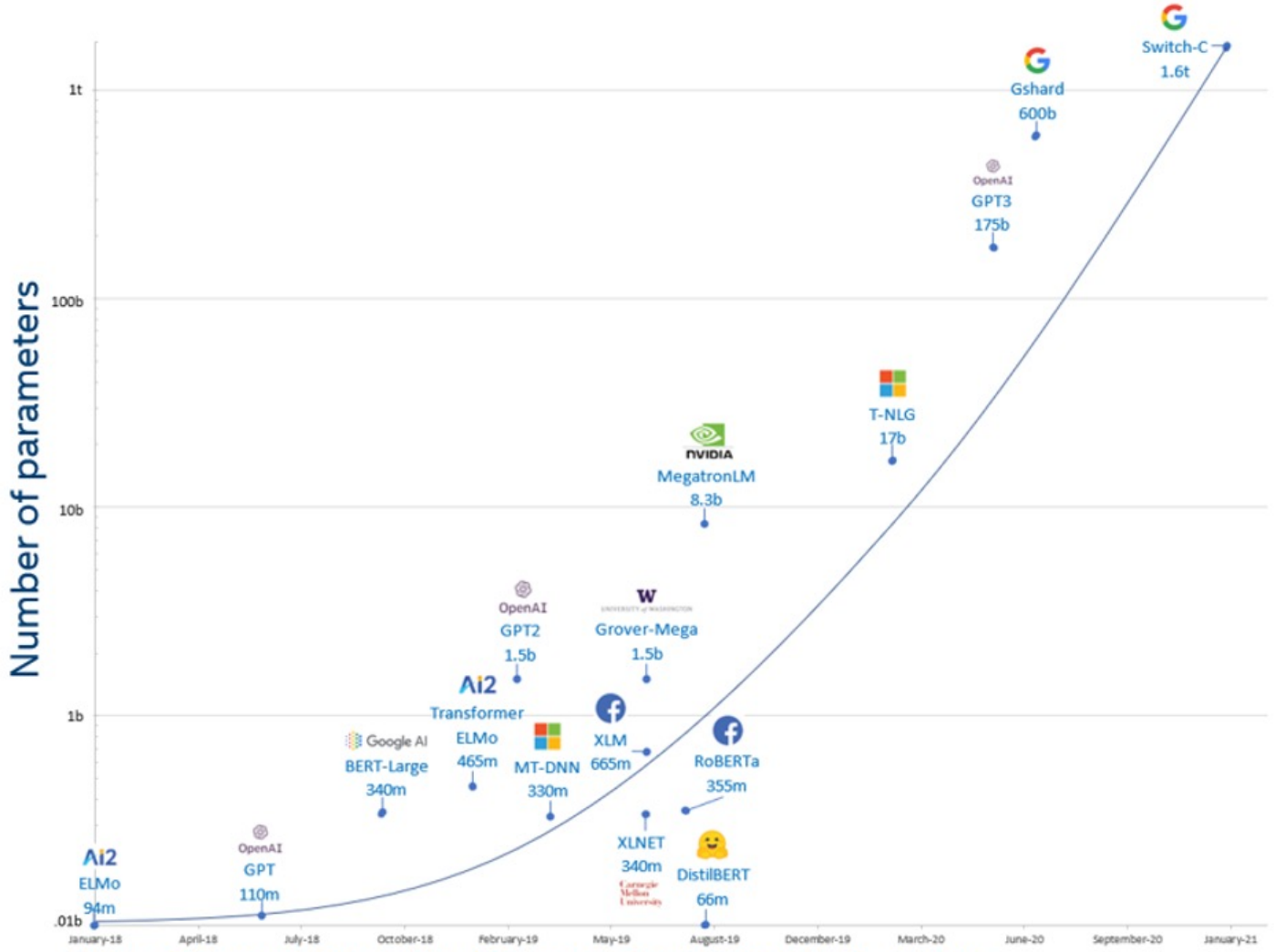


Figure 1: Exponential growth of number of parameters in DL models

Transfer Learning

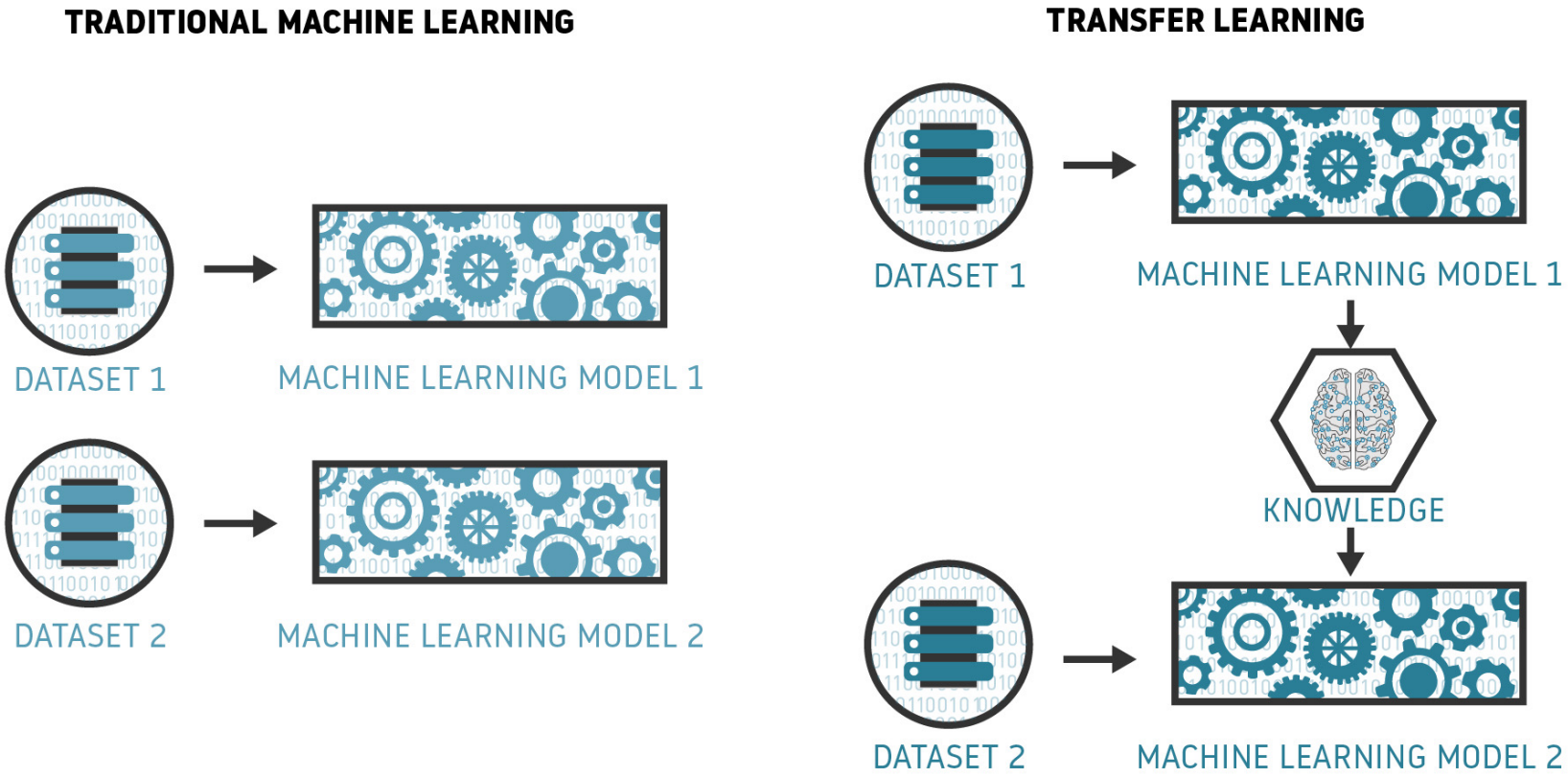
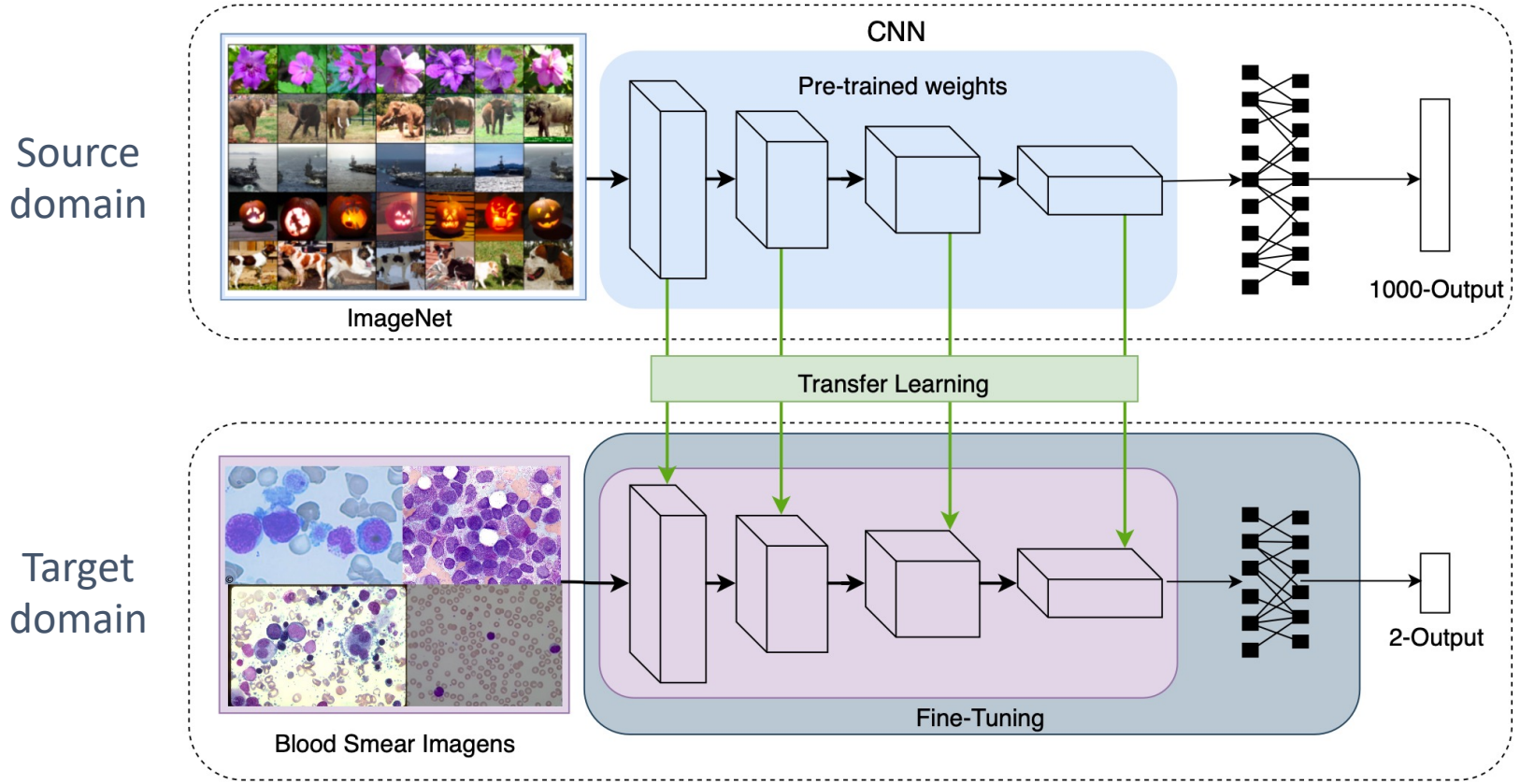


Image source: https://datascience.aero/transfer-learning-aviation/

Pre-Trained Models

- In deep learning, transfer learning is usually expressed through the use of pre-trained models.
- A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve.
- Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature.
  - Examples of pretrained CNN models: VGG, GoogLeNet, ResNet, MobileNet…
  - Examples of pretrained language models: BERT, GPT, GPT-2…

Fine-Tuning


Source domain
CNN
Pre-trained weights
ImageNet
1000-Output
Transfer Learning
Target domain
Blood Smear Imagens
Fine-Tuning
2-Output

Fine-Tuning Strategies

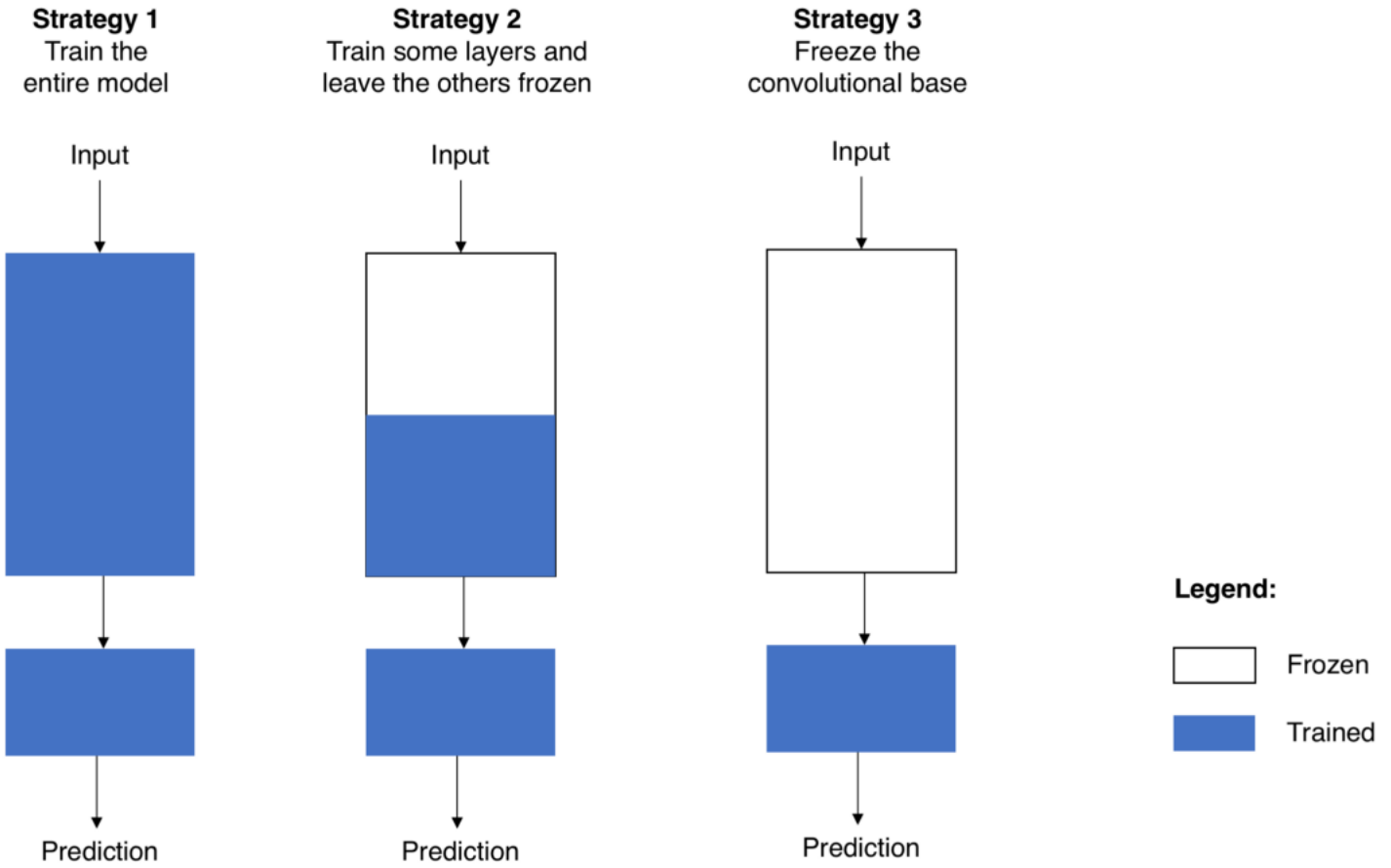


Image source: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Fine-Tuning Strategies

Choose fine-tuning strategy based on your target dataset.

Data amount \ Data similarity	Similar	Different
Little	Finetune linear classifier on top layer	You're in trouble... Try data augmentation / collect more data
Large	Finetune a few layers	Finetune a larger number of layers

Reference: Lecture 9, cs231n

Conclusion

After this lecture, you should know:

- What is convolution and filter/kernel.
- What are the commonly used layers in CNN.
- How to calculate the output size of after a convolutional layer.
- Why do we need pooling.
- What are the typical CNN architectures.
- What is the basic structure of RNNs.
- What is the hidden state.
- How do RNNs handle sequential data.