



Robot Vision

Introduction to Deep Learning

Dr. Chen Feng

cfeng@nyu.edu

ROB-UY 3203, Spring 2024

Overview

- Overview of Deep Learning
- Computational Graph
- Back Propagation
- DNN Architectures
 - CNN
 - ResNet
 - AutoEncoder
- Common Practices

References

- Go2016
 - Section 6-6.1, 6.5, 9-9.4, 10-10.2, 14-14.2
- Sz2022
 - Section 5.3.1-5.3.2, 5.3.4-5.3.5, 5.4.1-5.4.4, 5.5.2, 5.5.4
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition.” In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.
- <https://github.com/ajaymache/machine-learning-yearning>

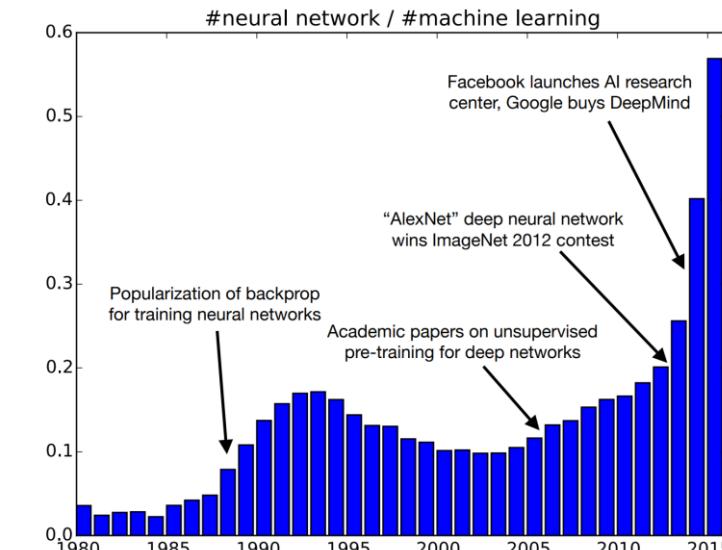
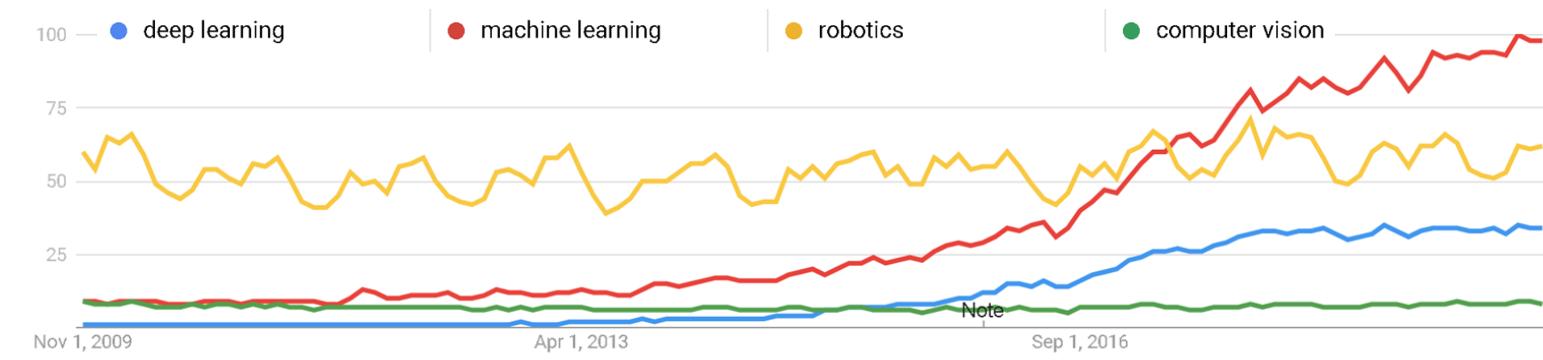
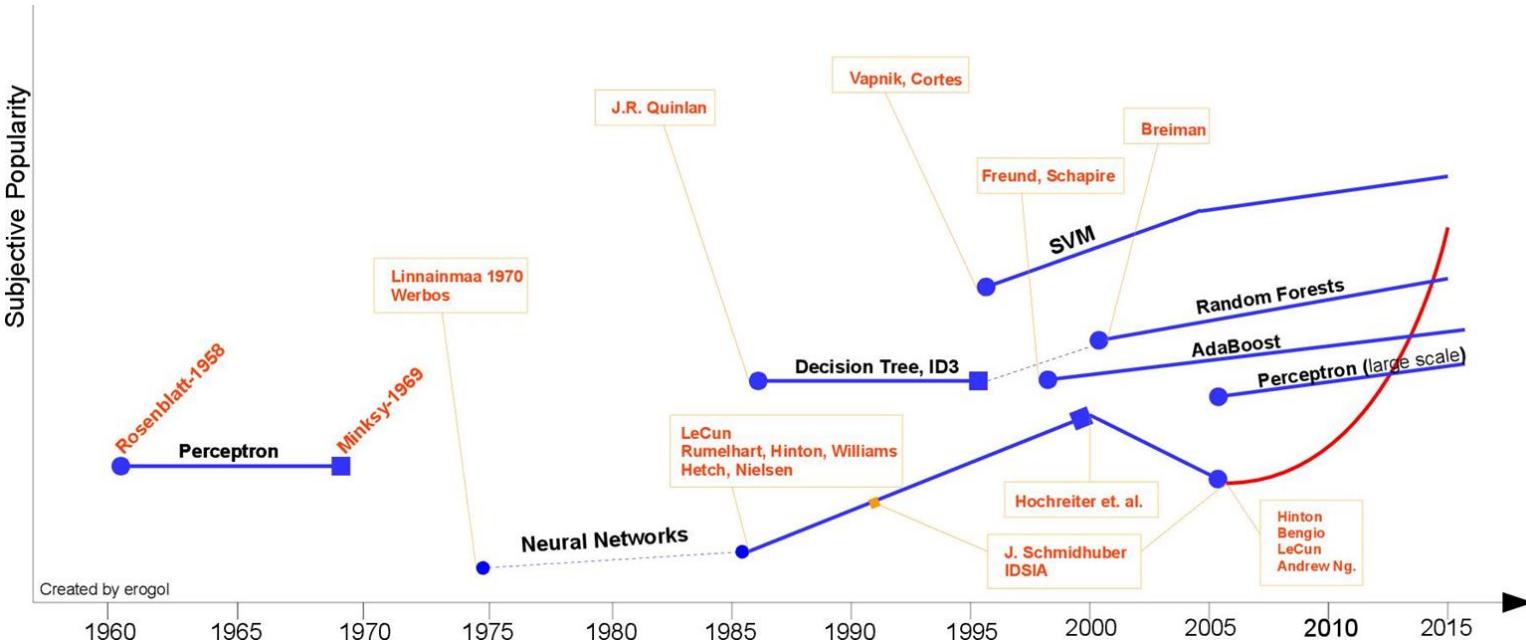


What to expect in this lecture?

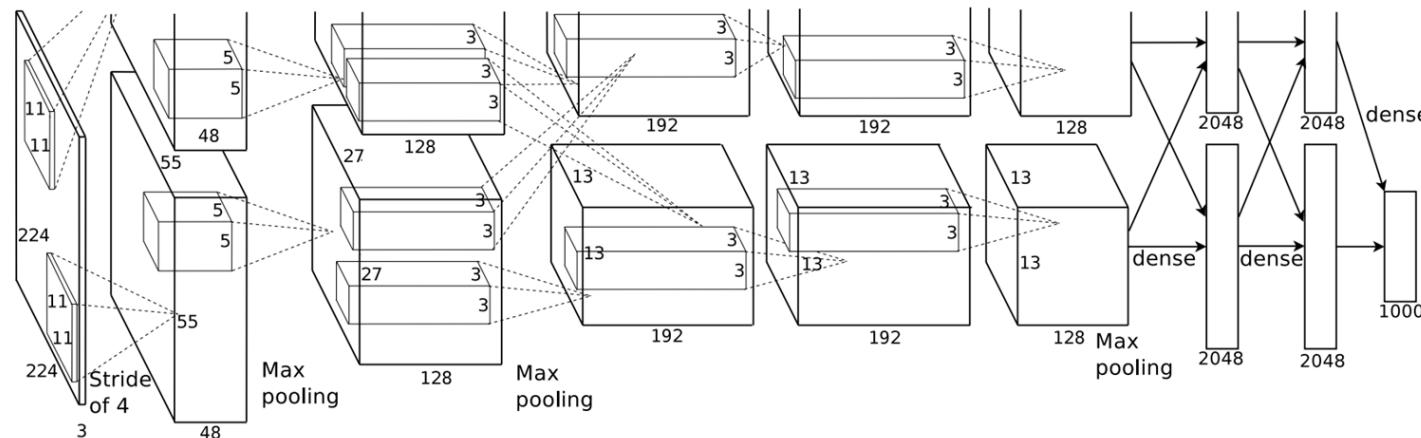
- This lecture is about:
 - Quick overview/reminder of some of the basics of deep learning
 - Establishing a common ground of understanding for future weeks
 - Especially for students who have not taken ML courses before
- This lecture is NOT about:
 - Replacing a full machine learning course
 - But we hope this lecture is enough for you to get started on reading and using these tools
 - A comprehensive review nor an in-depth analysis of deep learning
 - But we hope this lecture opens the doors for those who are interested



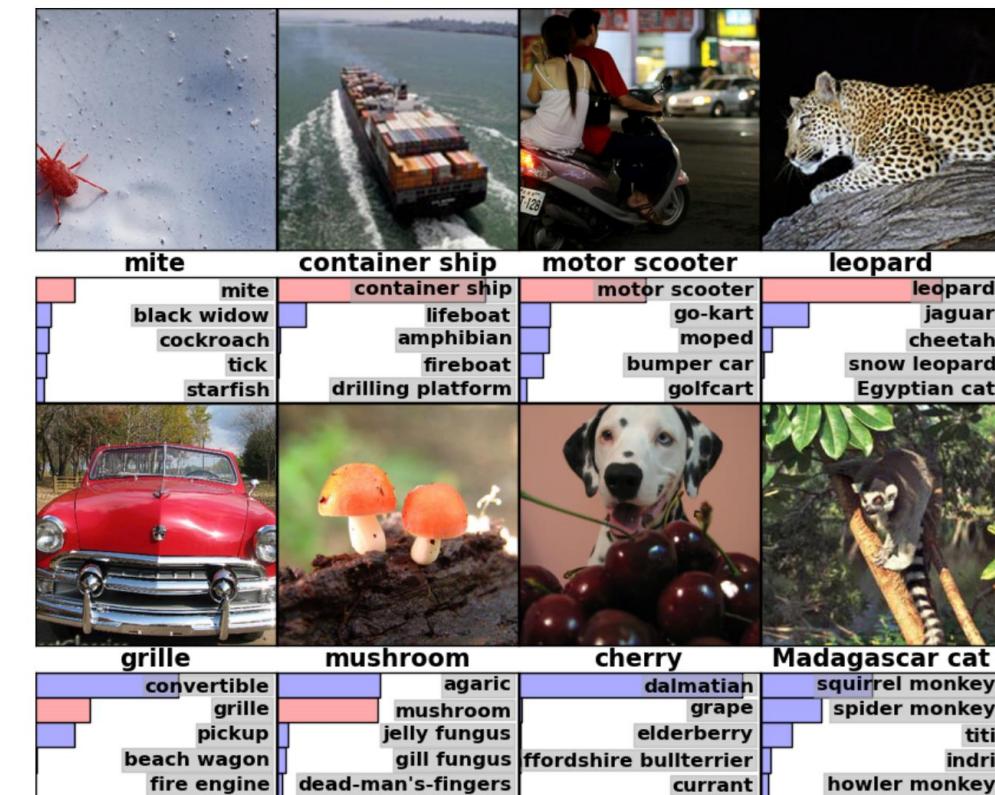
Recent History in Machine Learning



AlexNet and ImageNet



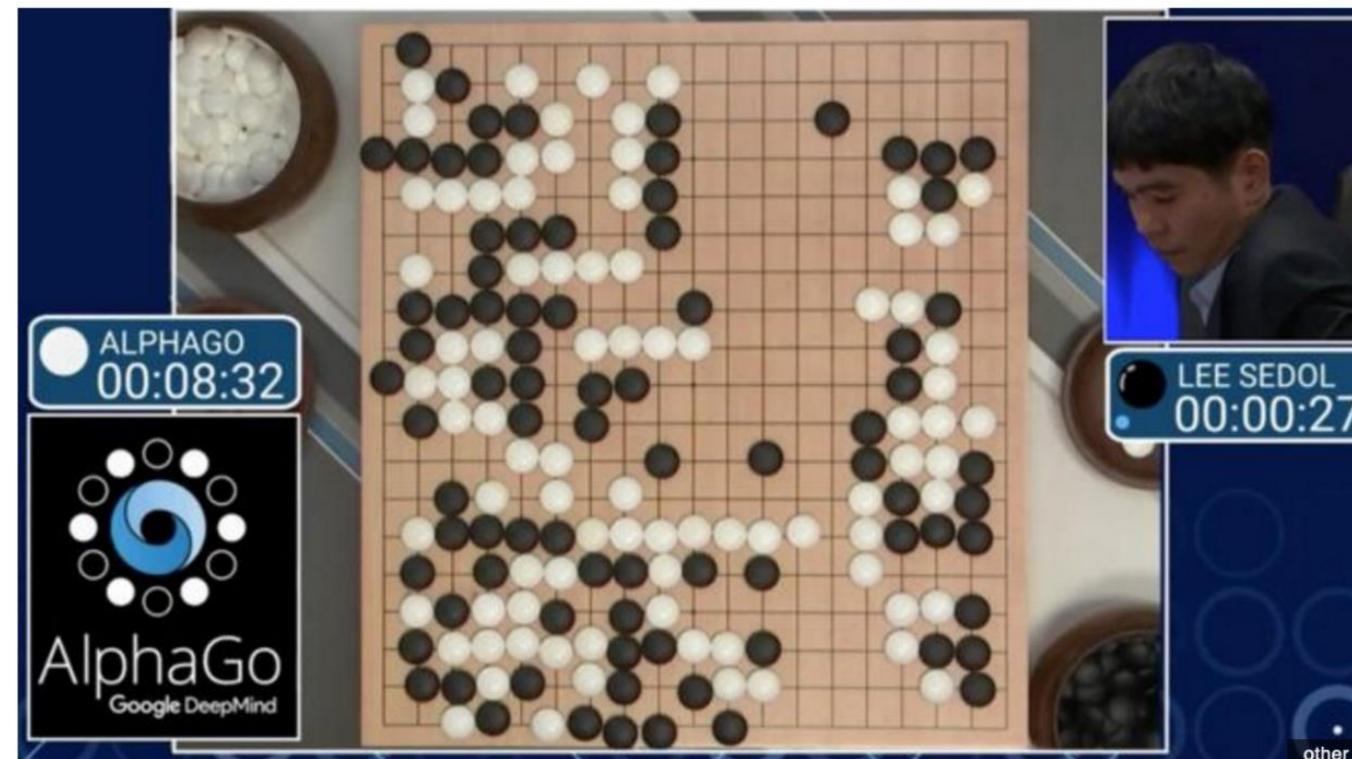
“AlexNet” (Krizhevsky et al., 2012), winning entry of ImageNet 2012 competition with a Top-5 error rate of 15.3% (next best system with highly engineered features based got 26.1% error)



AlphaGo

Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol

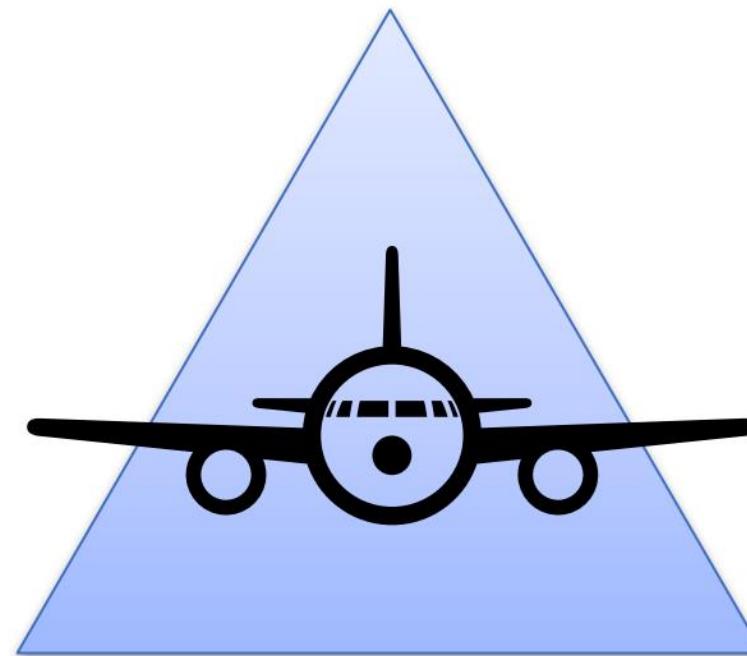
⌚ 12 March 2016 | Technology





Why Deep Learning Now?

High capacity models
(i.e., large VC
dimension)



Lots of
computing power

Lots of data



Neural Networks for Machine Learning

- Three Components of Machine Learning
 - **Hypothesis Class**: what does the search space look like?
 - **Loss Function**: how good is a hypothesis?
 - **Optimization**: how to find a good hypothesis?
- Neural Network is just a class of hypothesis functions in machine learning
 - Composition of non-linear functions
 - Typically, these functions should be continuous and differentiable at *almost everywhere* between input and output

Before Deep Learning

$$h_{\theta}(x) = \theta^T x$$

Compose Multiple Hypothesis



Deep Learning

$$h_{\theta}(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

How to Compose Multiple Linear Hypothesis Meaningfully?



Given $x \in \mathbb{R}^n$, suppose I run two machine learning algorithms with the hypothesis functions $h, \tilde{h} : \mathbb{R}^n \rightarrow \mathbb{R}$:

$$\begin{aligned} h(x) &= W_2(W_1x + b_1) + b_2 & \tilde{h}(x) &= W_3x + b_3 \\ W_1 \in \mathbb{R}^{k \times n}, b_1 \in \mathbb{R}^k, W_2 \in \mathbb{R}^{1 \times k}, b_2 \in \mathbb{R} & & W_3 \in \mathbb{R}^{1 \times n}, b_3 \in \mathbb{R} & \end{aligned}$$

Suppose we use the same data, minimize the same loss function, and are (somehow) able to achieve the *global* optima of both problems (assumed to be unique), which of the following will be true:

1. h achieves lower training loss but higher validation loss than \tilde{h}
2. h achieves lower training loss *and* lower validation loss than \tilde{h}
3. \tilde{h} achieves lower training loss *and* lower validation loss than h
4. They will both perform identically
5. The performance depends on the data or choice of loss function



Neural Networks Are Universal Function Approximators

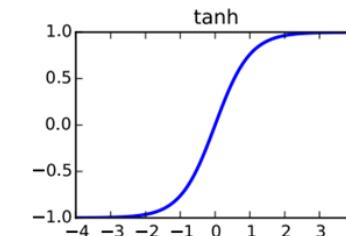
Neural networks are a simple extension of this idea, where we additionally apply a non-linear function after each linear transformation

$$h_{\theta}(x) = f_2(W_2 f_1(W_1 x + b_1) + b_2)$$

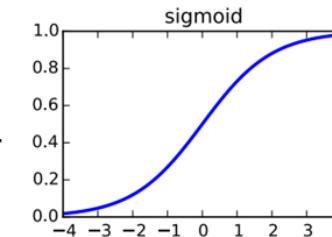
where $f_1, f_2: \mathbb{R} \rightarrow \mathbb{R}$ are a non-linear functions (applied elementwise)

Common choices of f_i :

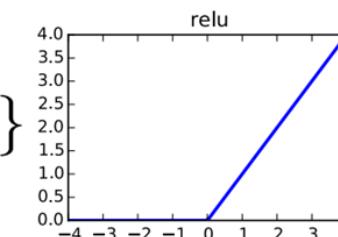
Hyperbolic tangent: $f(x) = \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$



Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$

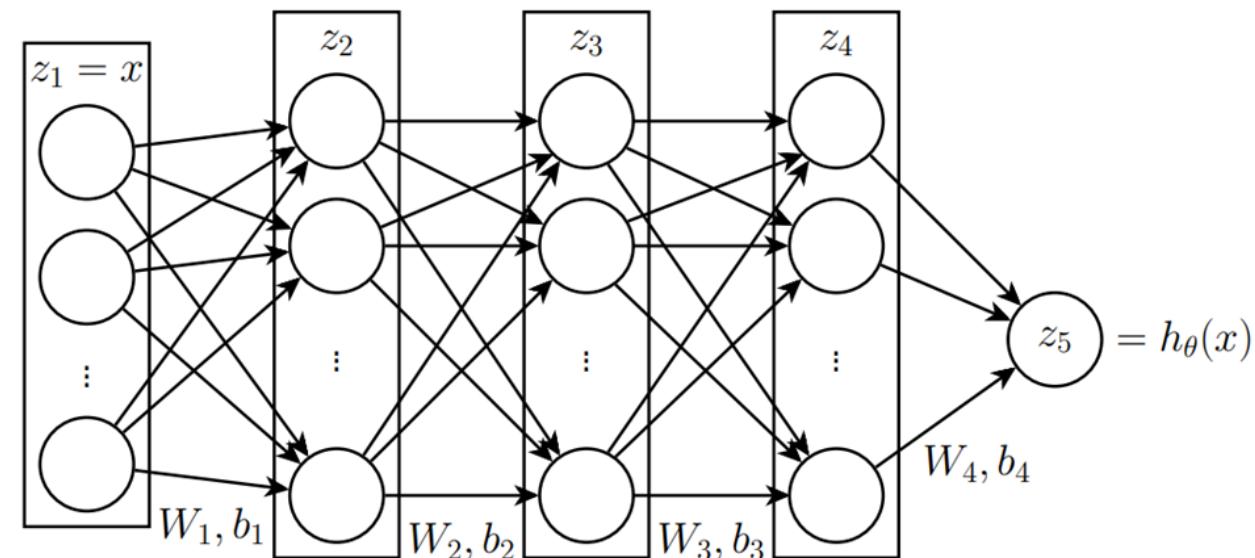


Rectified linear unit (ReLU): $f(x) = \max\{x, 0\}$



Deep Learning

“Deep learning” refers (almost always) to machine learning using neural network models with multiple hidden layers



Hypothesis function for k -layer network

$$z_{i+1} = f_i(W_i z_i + b_i), \quad z_1 = x, \quad h_\theta(x) = z_k$$

(note the z_i here refers to a vector, not an entry into vector)

Training Deep Networks

How do we solve the optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^m \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

Not a convex problem, so we don't expect to find global optimum, but we will instead be content with *local* solutions

Just use gradient descent as normal (or rather, a version called stochastic gradient descent)

Stochastic Gradient Descent

Key challenge for neural networks: often have very large number of samples, computing gradients can be computationally intensive.

Traditional gradient descent computes the gradient with respect to the sum over *all examples*, then adjusts the parameters in this direction

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \sum_{i=1}^m \ell(h_{\theta}(x^{(i)}, y^{(i)})) = \theta - \alpha \sum_{i=1}^m \nabla_{\theta} \ell(h_{\theta}(x^{(i)}, y^{(i)}))$$

Alternative approach, *stochastic gradient descent* (SGD): adjust parameters based upon just one sample

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

and then repeat these updates for all samples



Gradient Descent vs. SGD

Gradient descent, repeat:

- For $i = 1, \dots, m$:

$$g^{(i)} \leftarrow \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

- Update parameters:

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^m g^{(i)}$$



Stochastic gradient descent, repeat:

- For $i = 1, \dots, m$:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

In practice, stochastic gradient descent uses a small collection of samples, not just one, called a *minibatch*



A Unified Way to Understand All Existing Deep Neural Networks

- Let's start from the simplest neural network example

- Univariate logistic least squares (logistic regression)

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

- x: **input**
- w,b: learnable parameters of the model, i.e., **weights** of the model
- z: learned **feature**
- σ : sigmoid function, a **non-linear activation**
- y: **output** of the model
- t: **ground truth**, aka, target
- L: MSE/L2 **loss**
- Let's compute the loss' derivative w.r.t. the weights

Chain Rule

How you would have done it in calculus class

$$\mathcal{L} = \frac{1}{2}(\sigma(wx + b) - t)^2$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w} \left[\frac{1}{2}(\sigma(wx + b) - t)^2 \right]$$

$$= \frac{1}{2} \frac{\partial}{\partial w} (\sigma(wx + b) - t)^2$$

$$= (\sigma(wx + b) - t) \frac{\partial}{\partial w} (\sigma(wx + b) - t)$$

$$= (\sigma(wx + b) - t) \sigma'(wx + b) \frac{\partial}{\partial w} (wx + b)$$

$$= (\sigma(wx + b) - t) \sigma'(wx + b) x$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial}{\partial b} \left[\frac{1}{2}(\sigma(wx + b) - t)^2 \right]$$

$$= \frac{1}{2} \frac{\partial}{\partial b} (\sigma(wx + b) - t)^2$$

$$= (\sigma(wx + b) - t) \frac{\partial}{\partial b} (\sigma(wx + b) - t)$$

$$= (\sigma(wx + b) - t) \sigma'(wx + b) \frac{\partial}{\partial b} (wx + b)$$

$$= (\sigma(wx + b) - t) \sigma'(wx + b)$$

What are the disadvantages of this approach?

A More Structured Way to Apply Chain Rule

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\frac{d\mathcal{L}}{dy} = y - t$$

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{dy} \cdot \sigma'(z)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{d\mathcal{L}}{dz} \cdot x$$

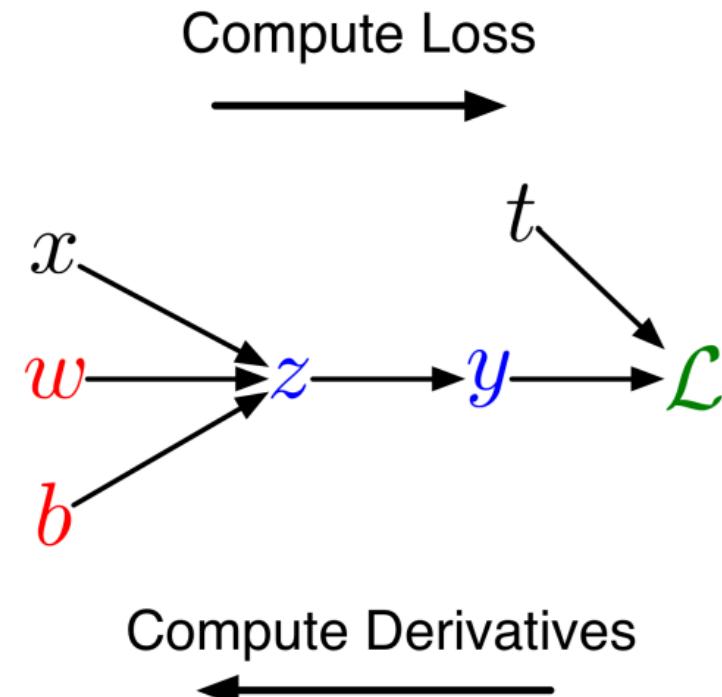
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dz}$$

Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.



A Symbolic Computation Graph

- We can diagram out the computations using a **computation graph**.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



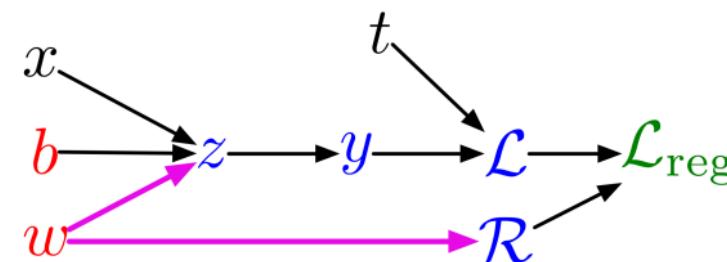


A Symbolic Computation Graph

Problem: what if the computation graph has **fan-out > 1**?

This requires the **multivariate Chain Rule**!

L_2 -Regularized regression



$$z = wx + b$$

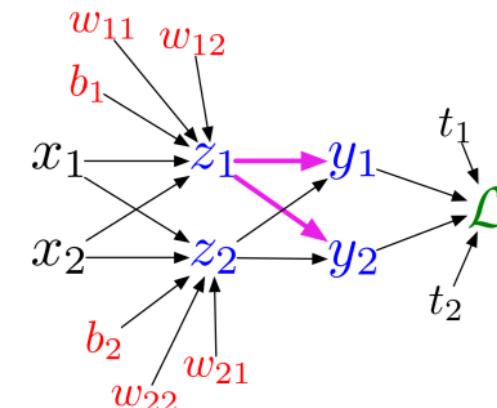
$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \mathcal{R}$$

Multiclass logistic regression



$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$

$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$

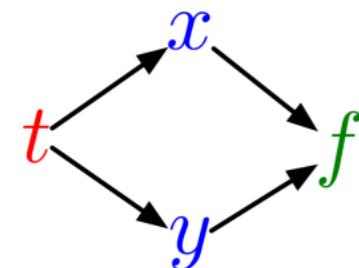
$$\mathcal{L} = - \sum_k t_k \log y_k$$



A Symbolic Computation Graph

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. (All the variables here are scalar-valued.) Then

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$



- Example:

$$f(x, y) = y + e^{xy}$$

$$x(t) = \cos t$$

$$y(t) = t^2$$

- Plug in to Chain Rule:

$$\begin{aligned}\frac{df}{dt} &= \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t\end{aligned}$$



A Symbolic Computation Graph

- In the context of backpropagation:

Mathematical expressions
to be evaluated

$$\frac{df}{dt} = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$

Values already computed
by our program

$$\bar{t} = \bar{x} \cdot \frac{dx}{dt} + \bar{y} \cdot \frac{dy}{dt}$$

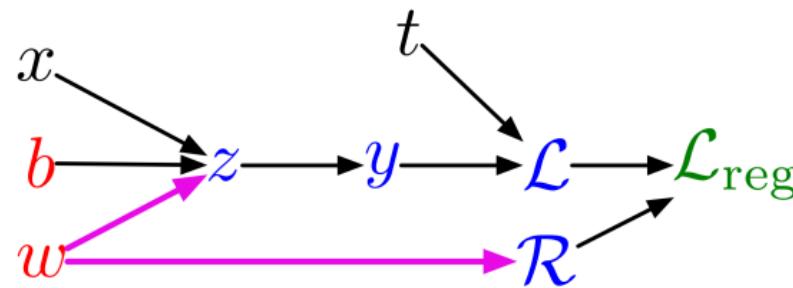
- In our notation:

Simplify notation
(because there is
always only one final
loss function in DL):

$$\frac{\partial \text{Loss}}{\partial \text{Variable}} \Leftrightarrow \overline{\text{Variable}}$$

A Symbolic Computation Graph

Example: univariate logistic least squares regression



Forward pass:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$

$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{R}}$$

$$= \overline{\mathcal{L}_{\text{reg}}} \lambda$$

$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{L}}$$

$$= \overline{\mathcal{L}_{\text{reg}}}$$

$$\overline{y} = \overline{\mathcal{L}} \frac{d\mathcal{L}}{dy}$$

$$= \overline{\mathcal{L}}(y - t)$$

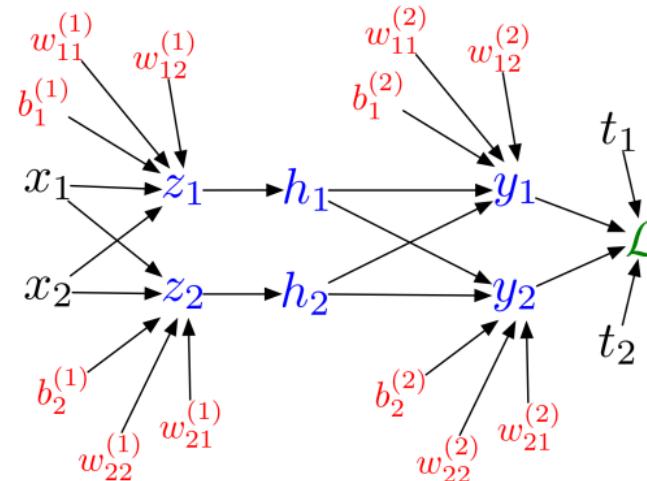
$$\begin{aligned}\overline{z} &= \overline{y} \frac{dy}{dz} \\ &= \overline{y} \sigma'(z)\end{aligned}$$

$$\begin{aligned}\overline{w} &= \overline{z} \frac{\partial z}{\partial w} + \overline{\mathcal{R}} \frac{d\mathcal{R}}{dw} \\ &= \overline{z}x + \overline{\mathcal{R}}w\end{aligned}$$

$$\begin{aligned}\overline{b} &= \overline{z} \frac{\partial z}{\partial b} \\ &= \overline{z}\end{aligned}$$

A Symbolic Computation Graph

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\bar{\mathcal{L}} = 1$$

$$\bar{y}_k = \bar{\mathcal{L}} (y_k - t_k)$$

$$\bar{w}_{ki}^{(2)} = \bar{y}_k h_i$$

$$\bar{b}_k^{(2)} = \bar{y}_k$$

$$\bar{h}_i = \sum_k \bar{y}_k w_{ki}^{(2)}$$

$$\bar{z}_i = \bar{h}_i \sigma'(z_i)$$

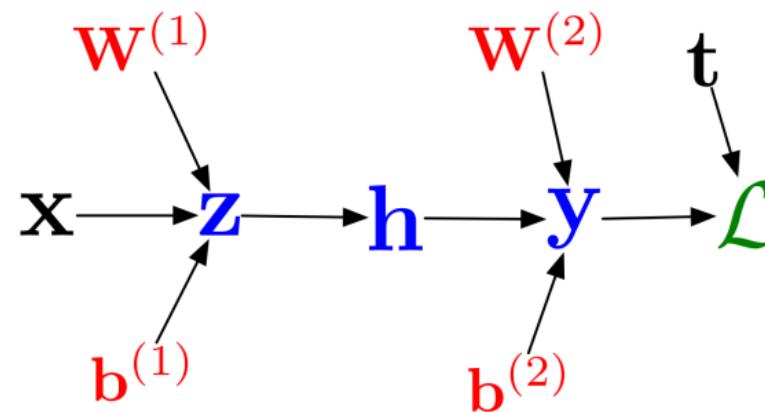
$$\bar{w}_{ij}^{(1)} = \bar{z}_i x_j$$

$$\bar{b}_i^{(1)} = \bar{z}_i$$



A Symbolic Computation Graph

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\bar{\mathcal{L}} = 1$$

$$\bar{\mathbf{y}} = \bar{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$

$$\bar{\mathbf{W}}^{(2)} = \bar{\mathbf{y}}\mathbf{h}^\top$$

$$\bar{\mathbf{b}}^{(2)} = \bar{\mathbf{y}}$$

$$\bar{\mathbf{h}} = \mathbf{W}^{(2)\top}\bar{\mathbf{y}}$$

$$\bar{\mathbf{z}} = \bar{\mathbf{h}} \cdot \sigma'(\mathbf{z})$$

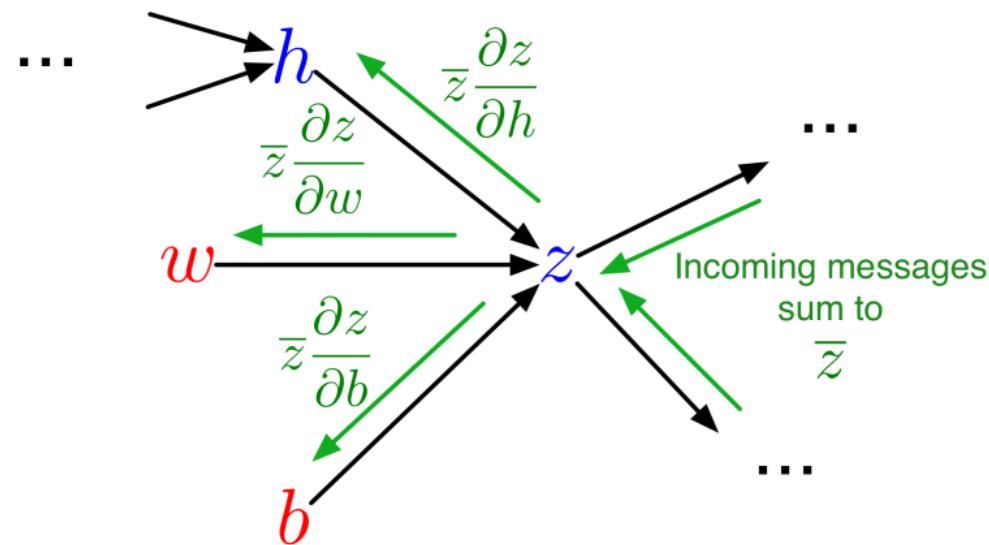
$$\bar{\mathbf{W}}^{(1)} = \bar{\mathbf{z}}\mathbf{x}^\top$$

$$\bar{\mathbf{b}}^{(1)} = \bar{\mathbf{z}}$$



A Symbolic Computation Graph

Backprop as message passing:



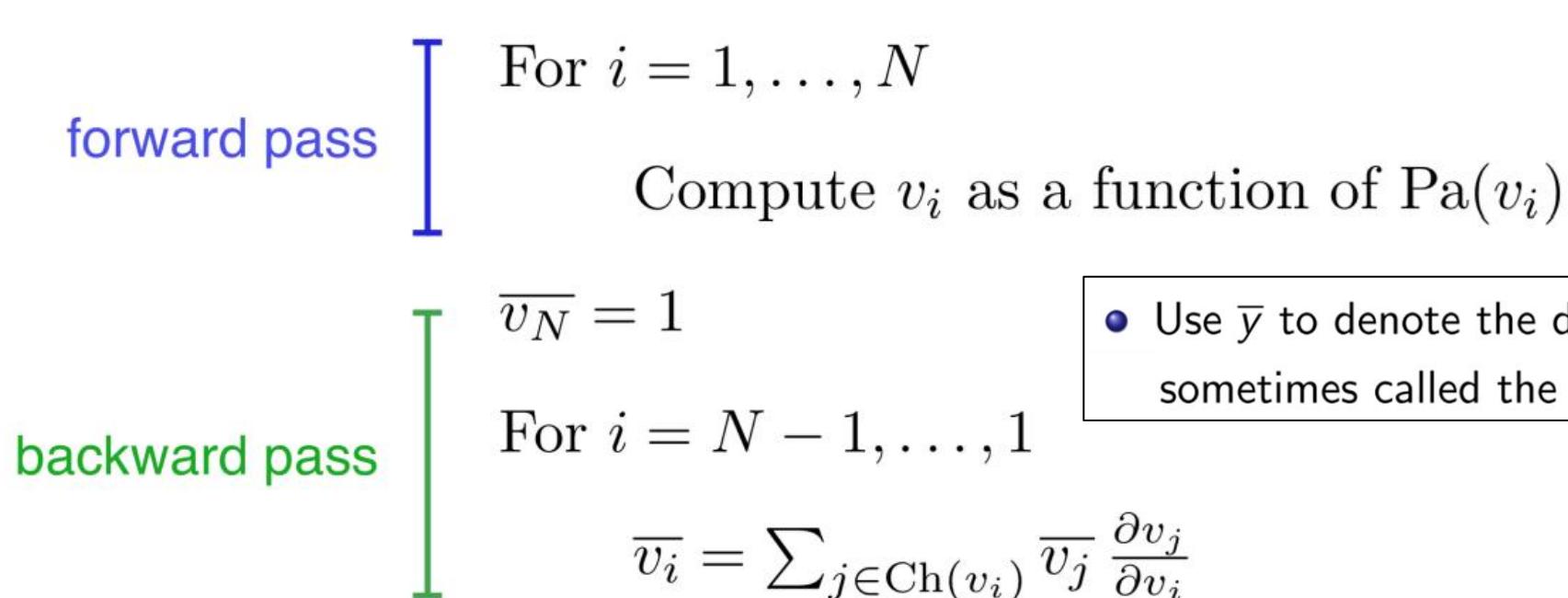
- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.
- This provides modularity, since each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph.

A Symbolic Computation Graph

Full backpropagation algorithm:

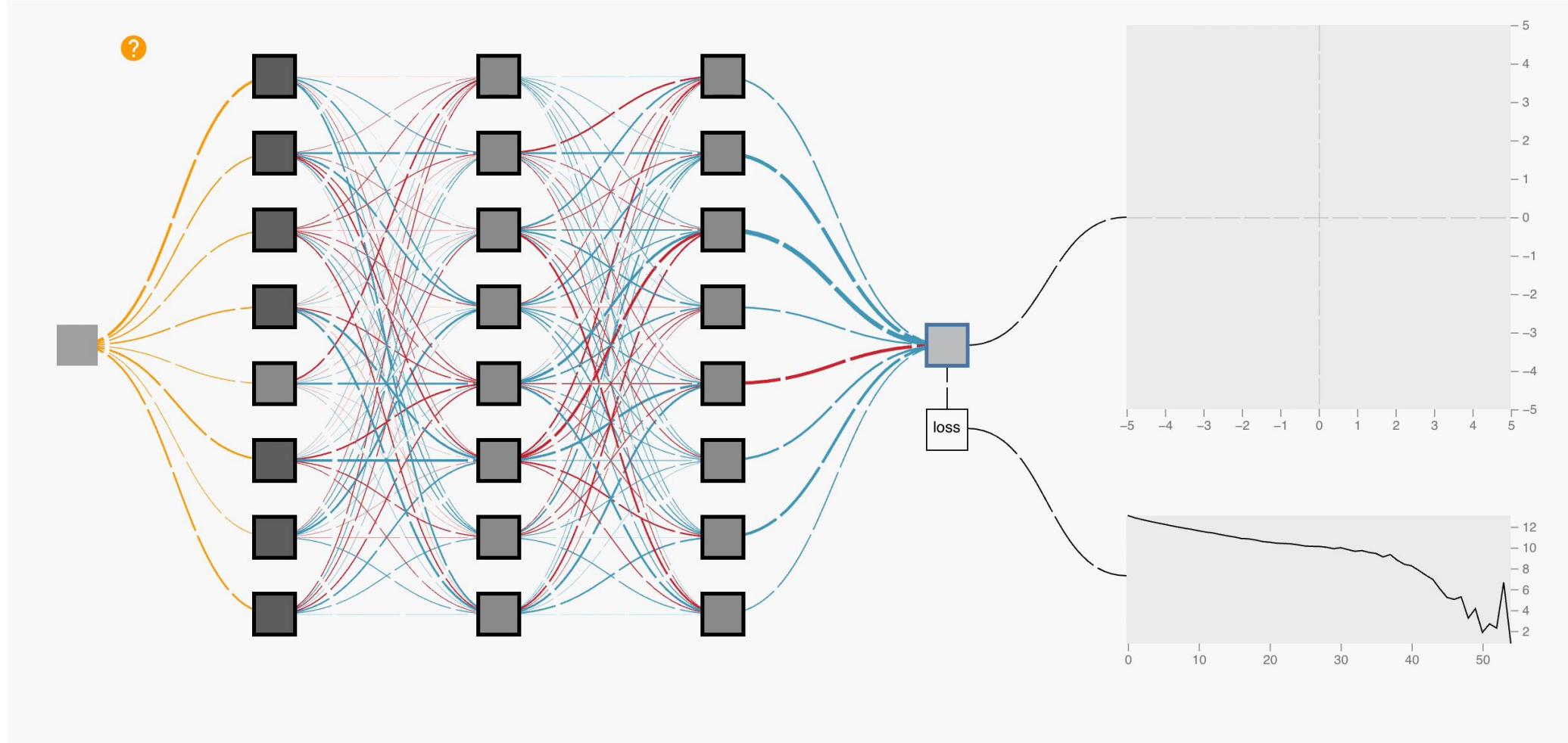
Let v_1, \dots, v_N be a **topological ordering** of the computation graph
(i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss).



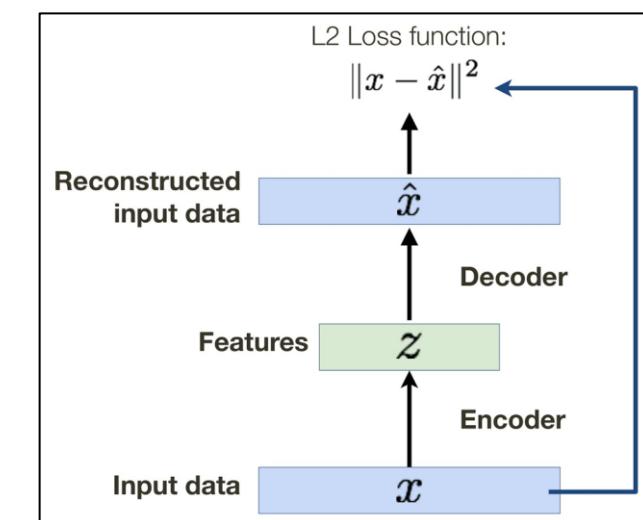
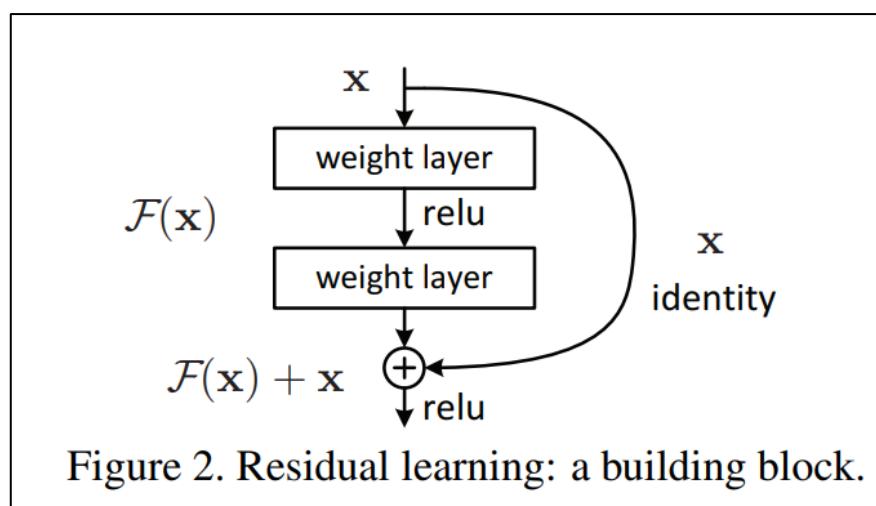
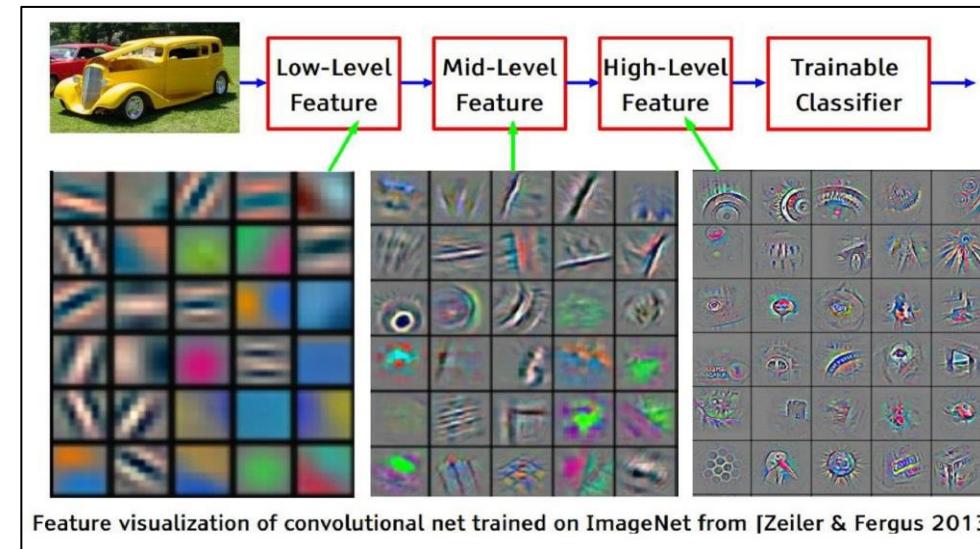


The Whole Picture





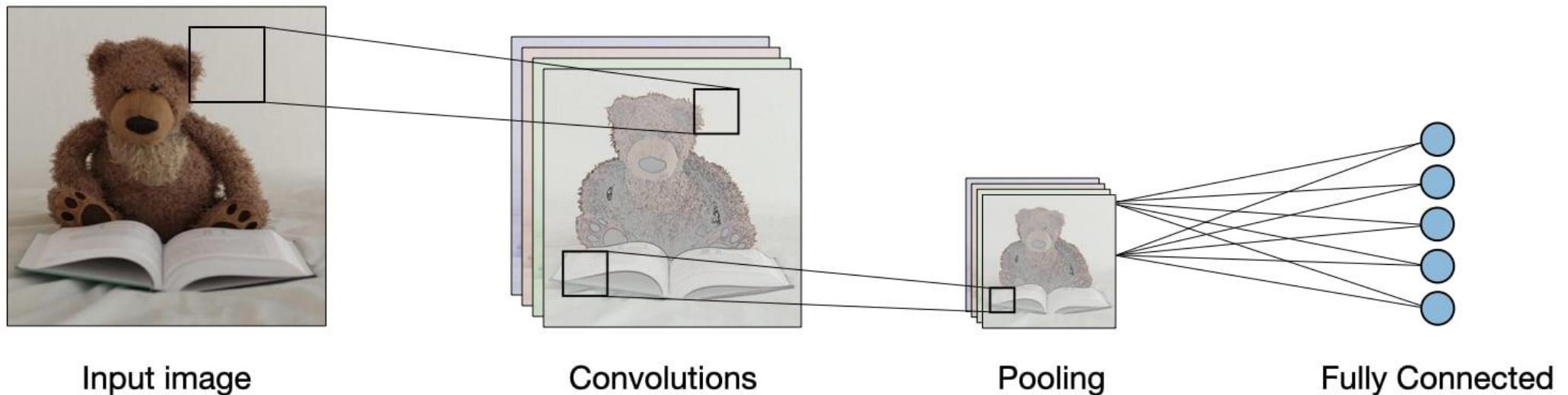
A Crash Course on Various Deep Nets



Convolutional Neural Networks

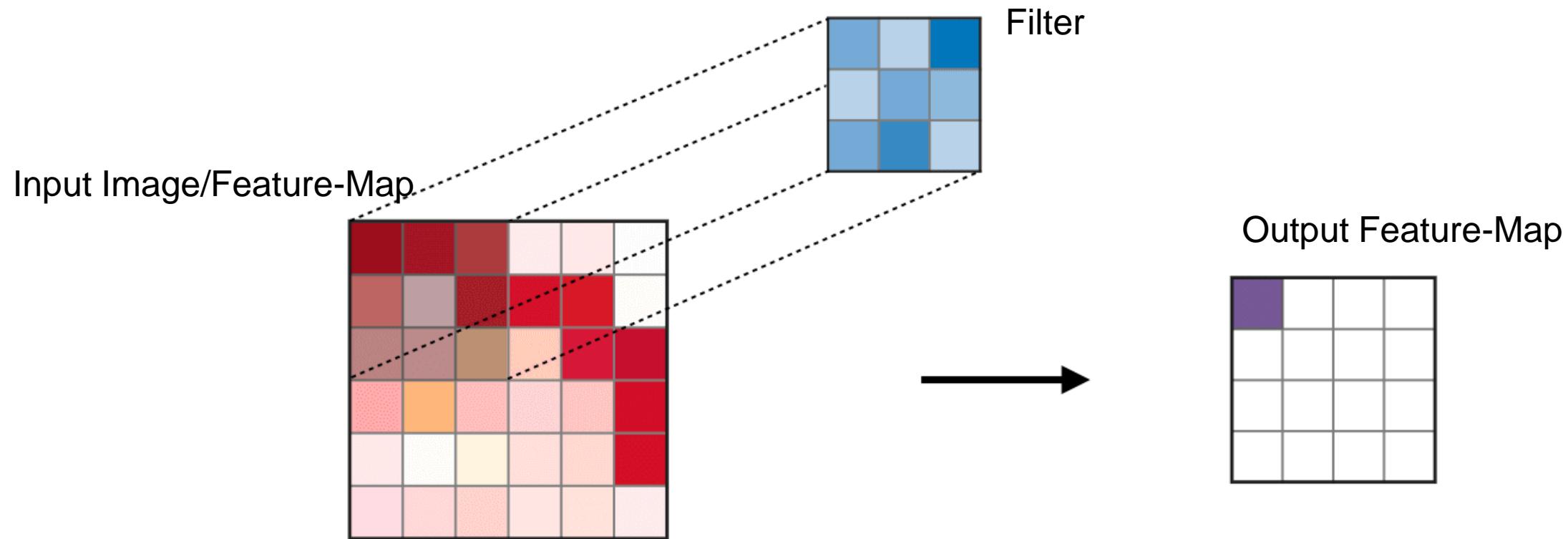
Basics of CNN

Convolutional Neural networks (CNN) are a class of neural networks that uses **convolution** to analyse data with **spatial structures/arrangement/neighborhood**



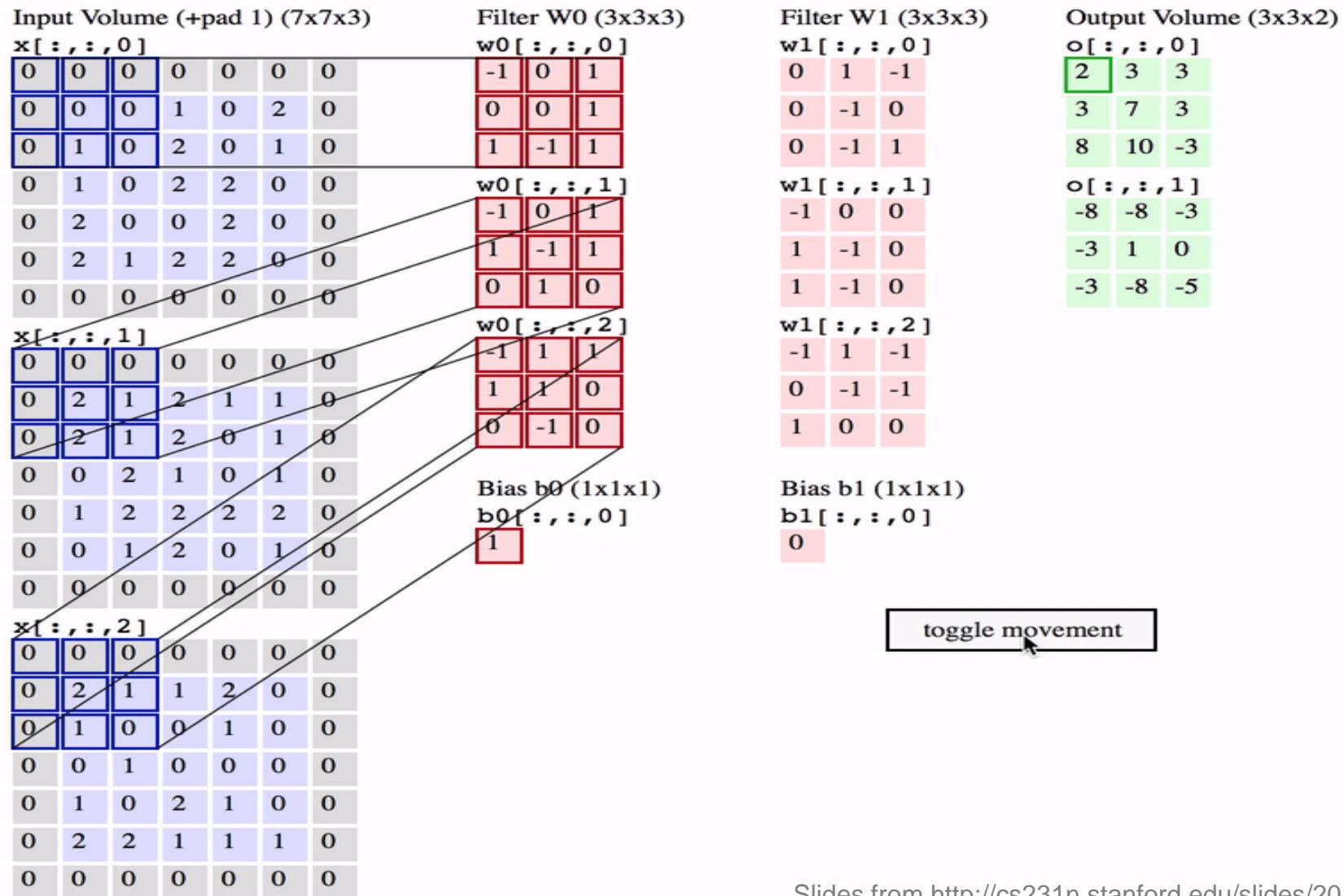
Basics of CNN: Convolution Layer

The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input with respect to its dimensions.





Convolution Operation Animation





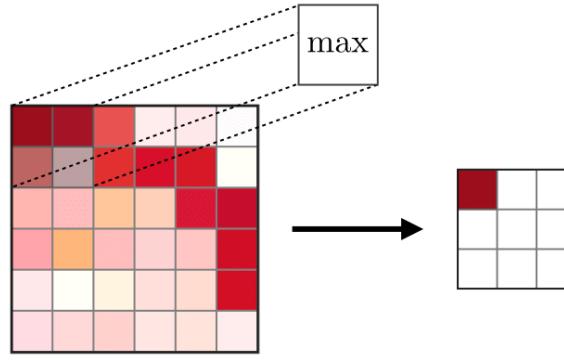
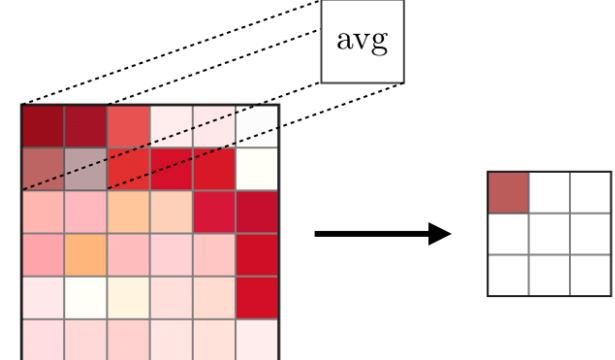
Basics of CNN: Pooling Layer

The pooling layer (POOL) is a **downsampling** operation, typically applied after a convolution layer, which does some spatial invariance.

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters
and stride 2

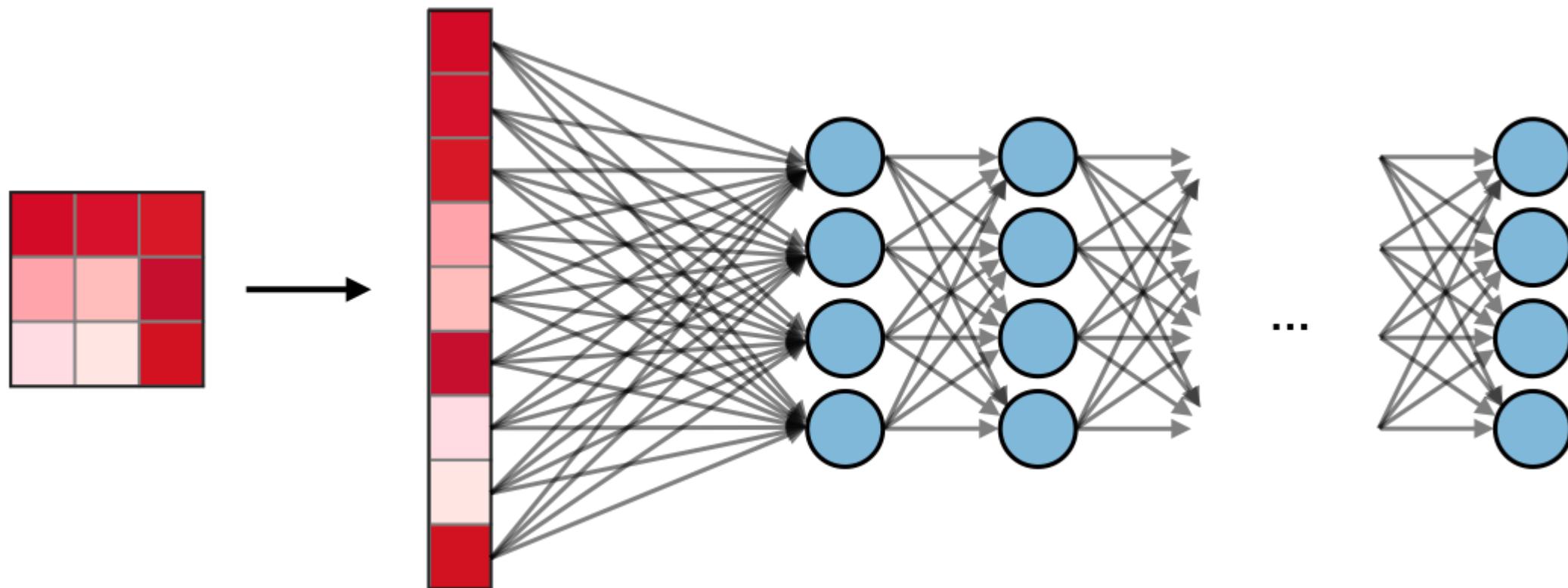
6	8
3	4

Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	<ul style="list-style-type: none">Preserves detected featuresMost commonly used	<ul style="list-style-type: none">Downsamples feature mapUsed in LeNet



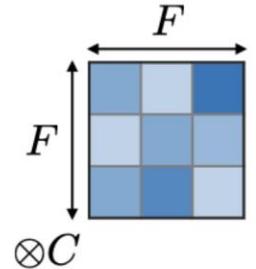
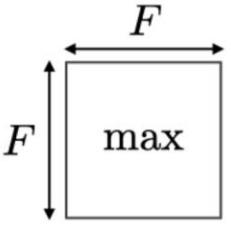
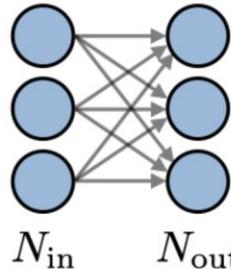
Basics of CNN: Fully Connected Layer

The fully connected layer (FC) operates on a **flattened input** where each input is connected to all neurons.





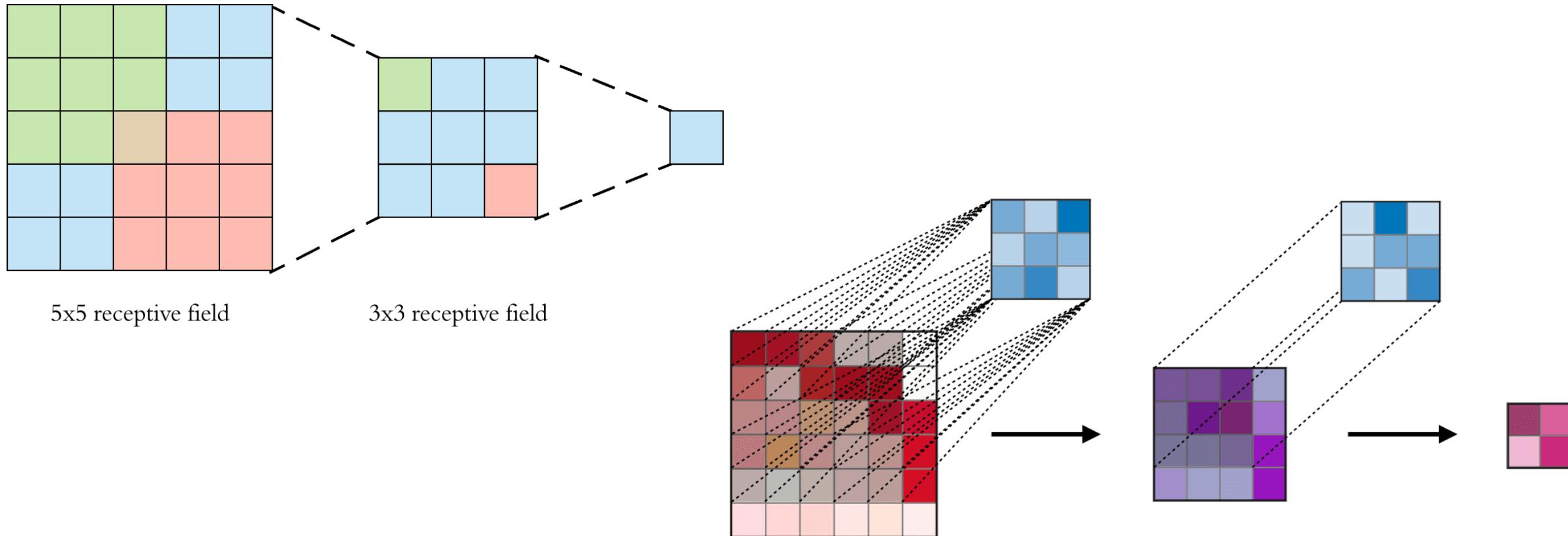
Basics of CNN: Relationship between Input, Output, & Parameters

	CONV	POOL	FC
Illustration	 $\times K$	 \max	
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	<ul style="list-style-type: none">One bias parameter per filterIn most cases, $S < F$A common choice for K is $2C$	<ul style="list-style-type: none">Pooling operation done channel-wiseIn most cases, $S = F$	<ul style="list-style-type: none">Input is flattenedOne bias parameter per neuronThe number of FC neurons is free of structural constraints



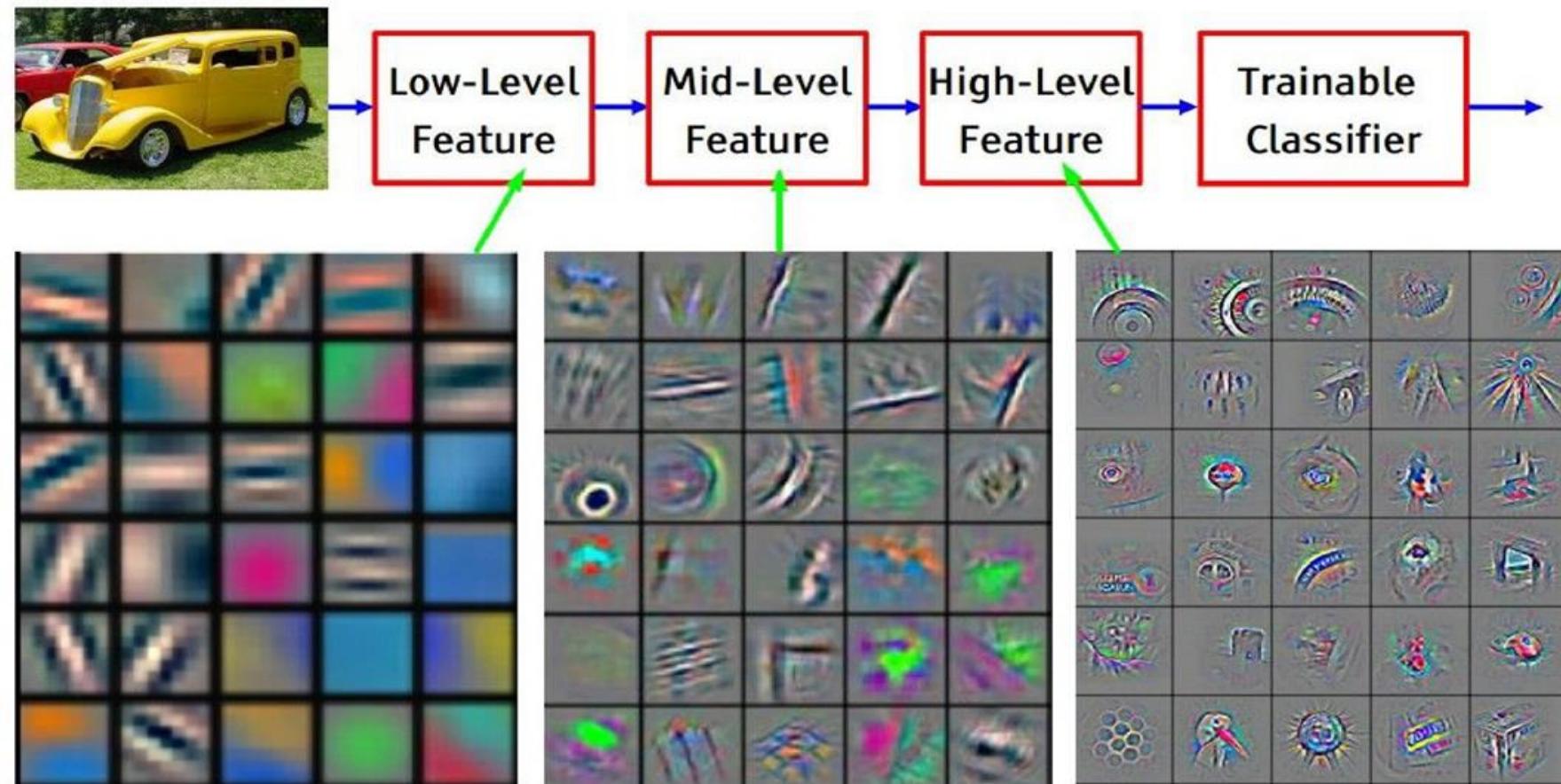
Basics of CNN: Receptive Field

- The **receptive field** in Convolutional Neural Networks (**CNN**) is the region of the input space that affects a particular unit of the network





[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

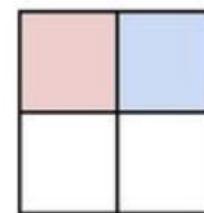


Transposed Convolution

Learnable Upsampling: Transpose Convolution

Other names:

- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

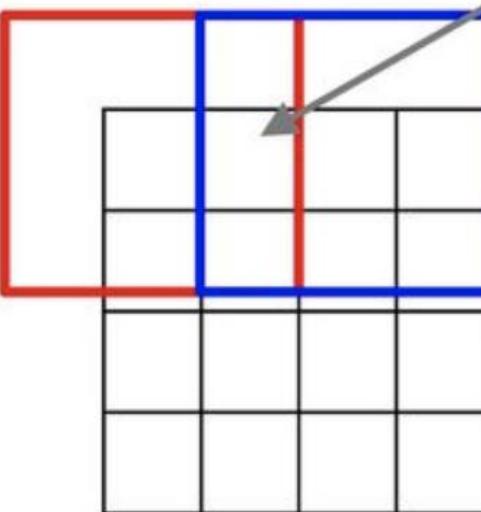


Input: 2 x 2

3 x 3 transpose convolution, stride 2 pad 1



Input gives weight for filter



Output: 4 x 4

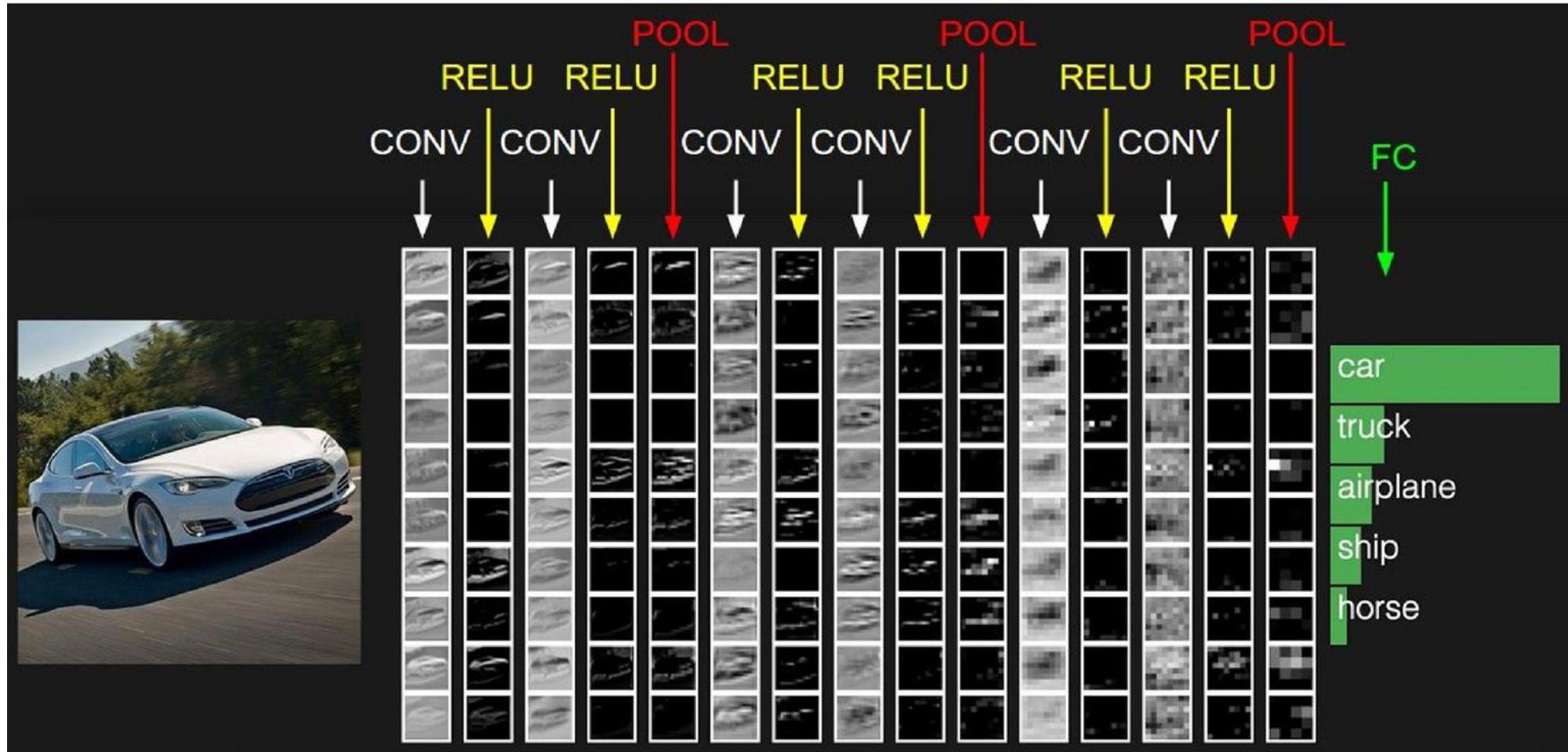
Sum where output overlaps

Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input



CNN – All layers



Residual Networks

One of the most popular CNN



Is Learning Better Networks as Easy as Stacking More Layers?

1. There is a common trend in the research community that network architecture needs to go **deeper**.
2. Simply **stacking more layers** leads to the problem of **vanishing gradients**, which hamper convergence from the beginning.
3. Meanwhile, **degradation** problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.

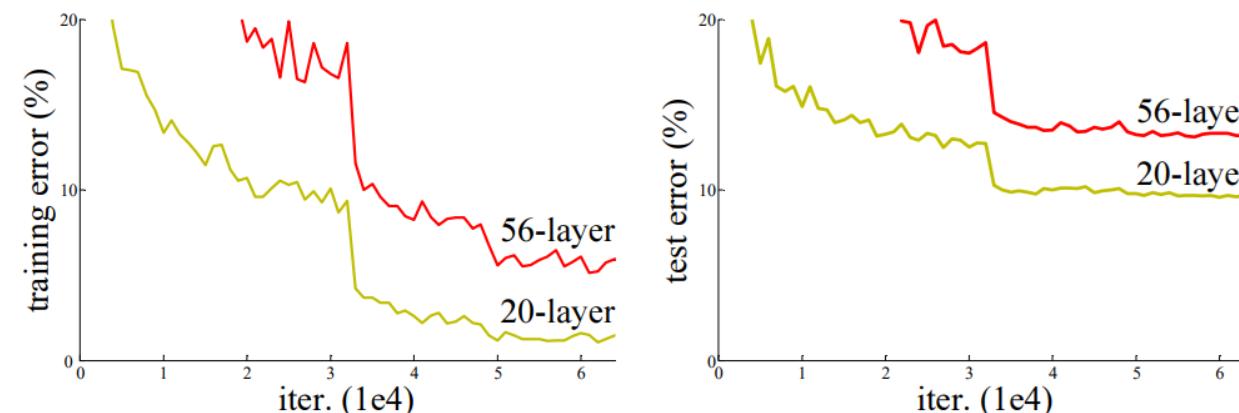


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Residual Learning

Residual learning block is formulated as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \quad (1)$$

Here \mathbf{x} and \mathbf{y} are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ in which σ denotes ReLU

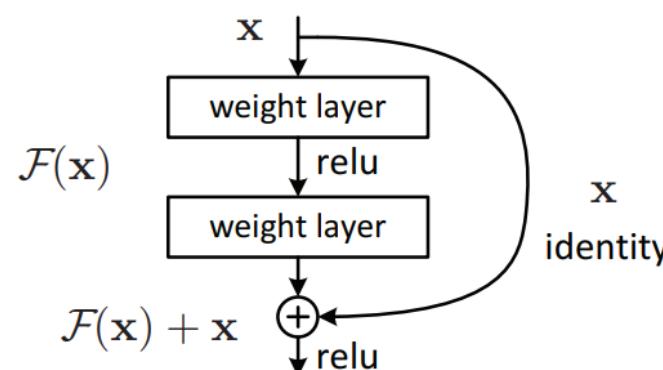


Figure 2. Residual learning: a building block.



Network architecture

Two types of architecture are tested: plain network similar as VGG-19 and residual network.

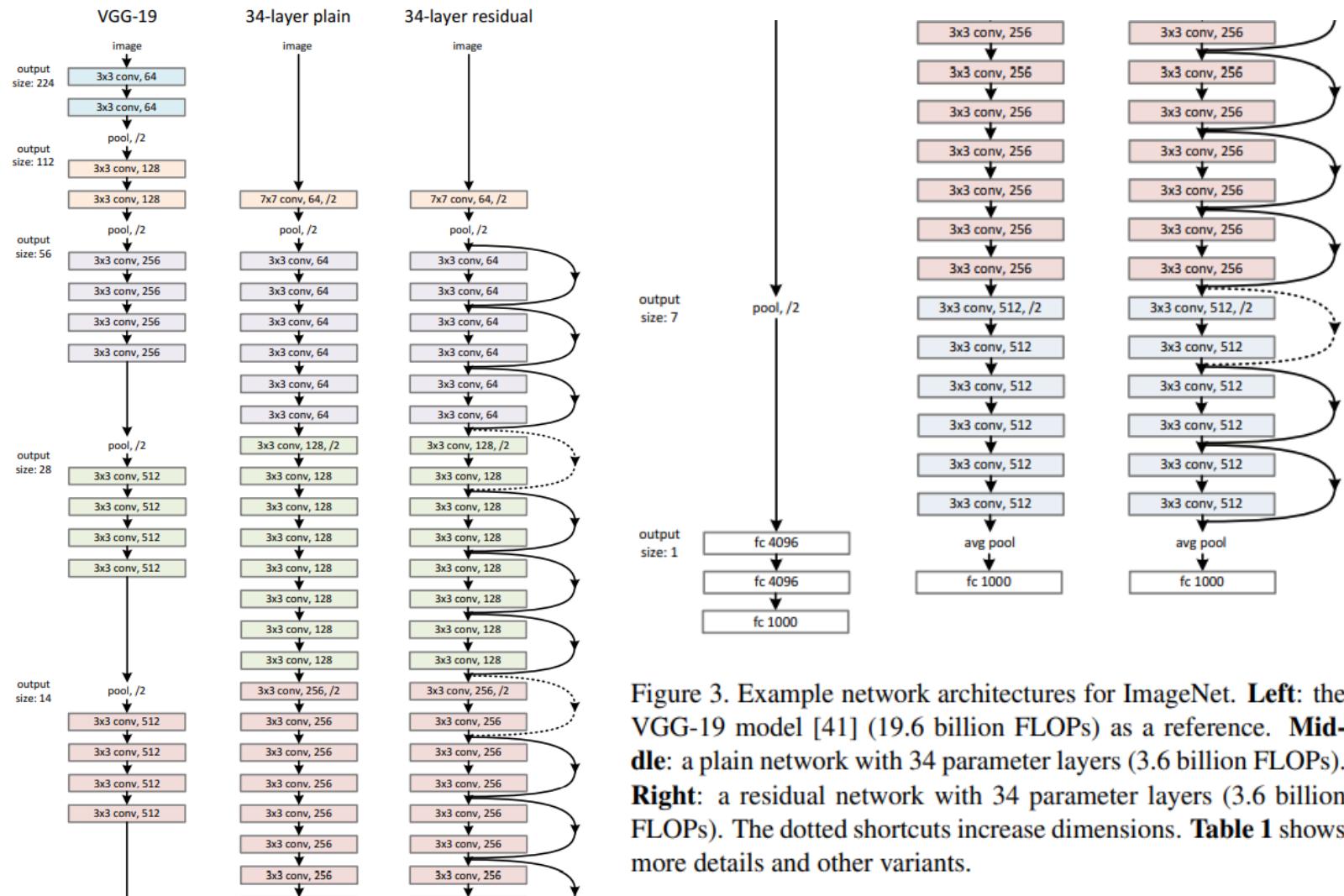


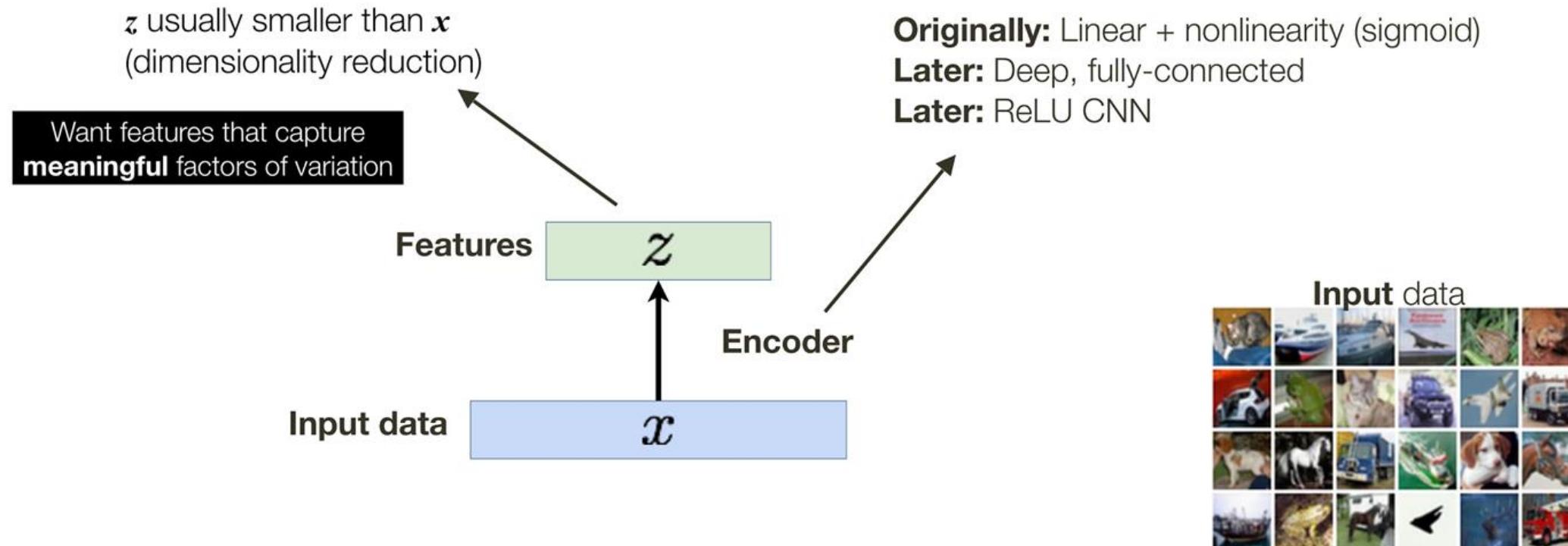
Figure 3. Example network architectures for ImageNet. **Left:** the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle:** a plain network with 34 parameter layers (3.6 billion FLOPs). **Right:** a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Self-Supervised Learning

Autoencoders (AE)

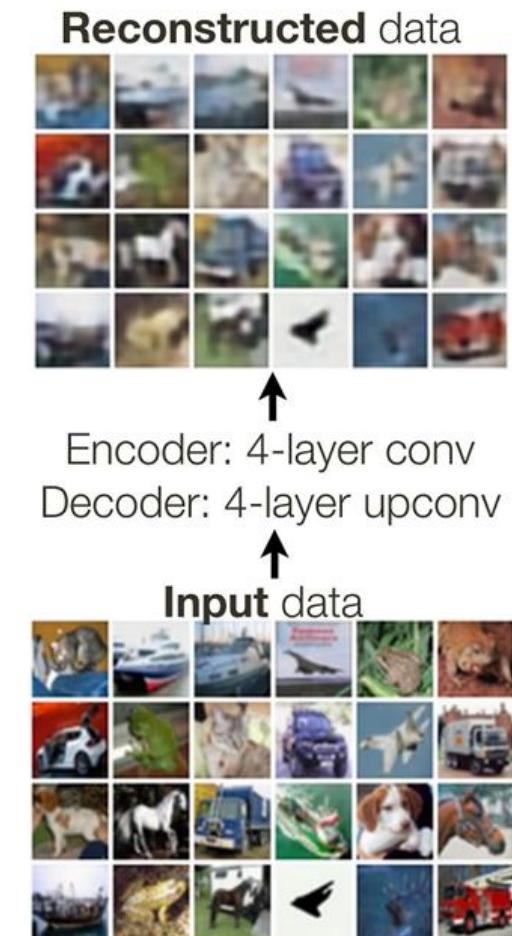
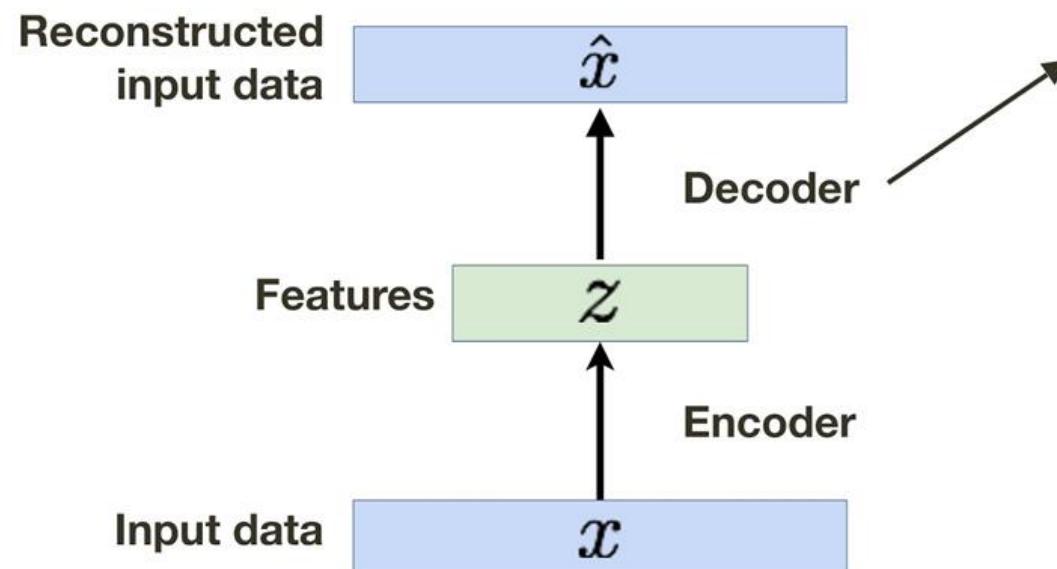
Autoencoders (AE)

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



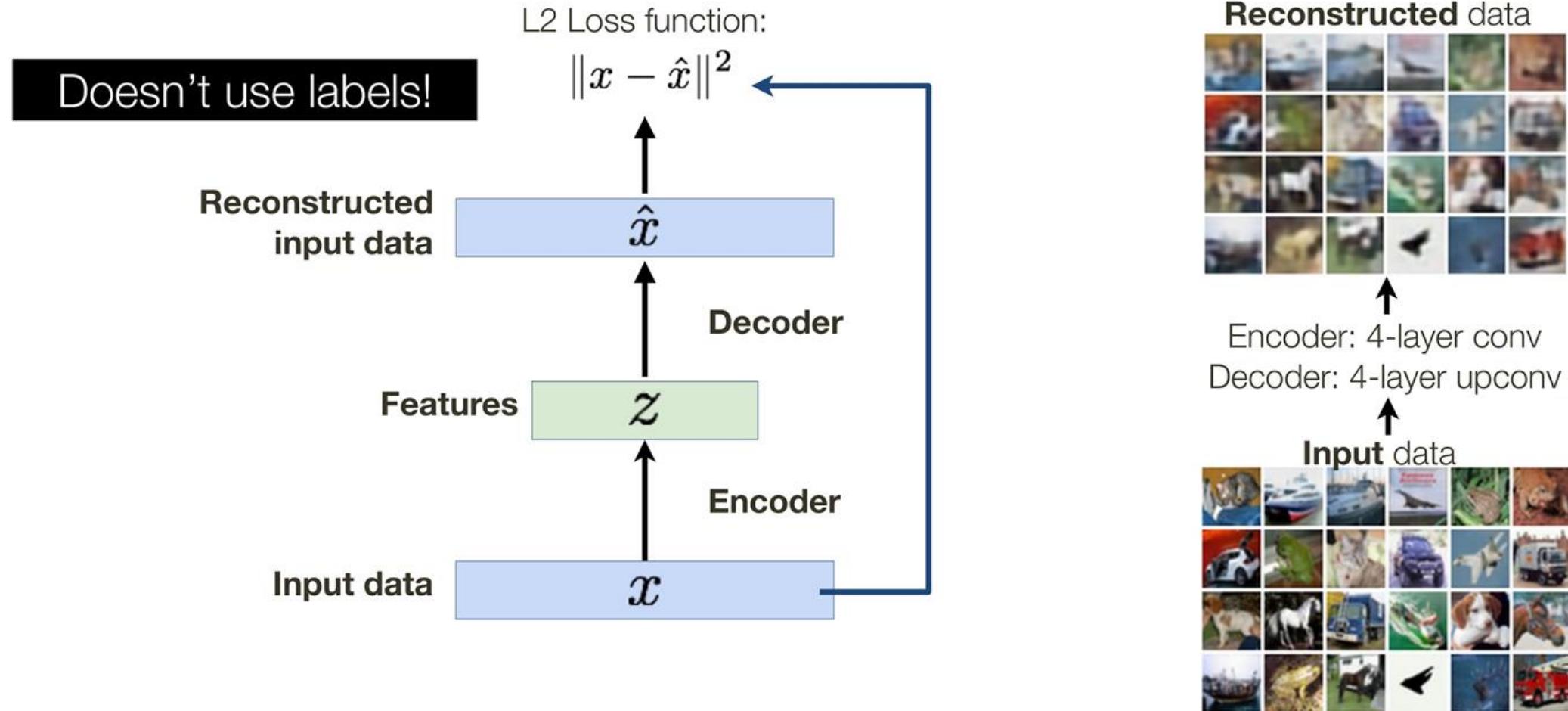
Autoencoders (AE)

Train such that features can reconstruct original data best they can



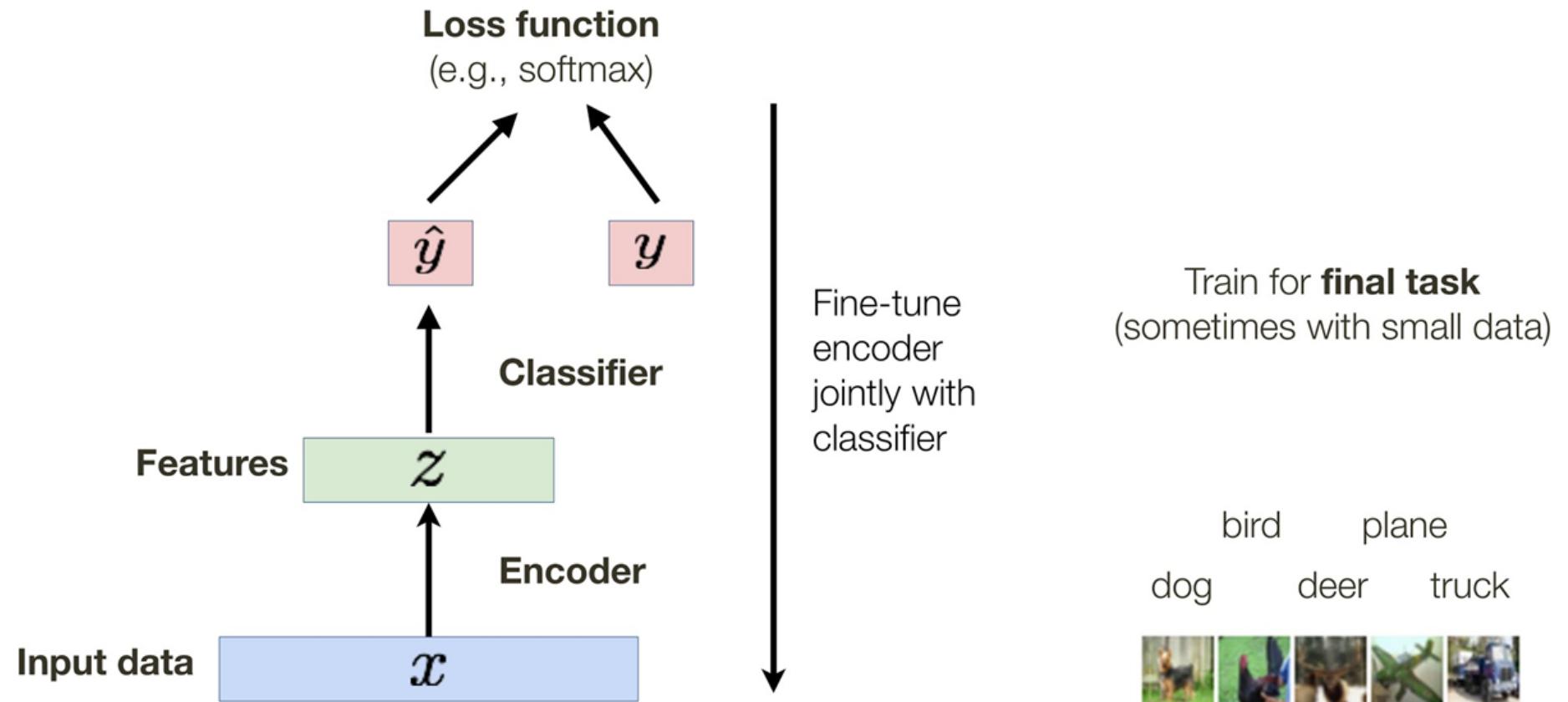


Autoencoders (AE)



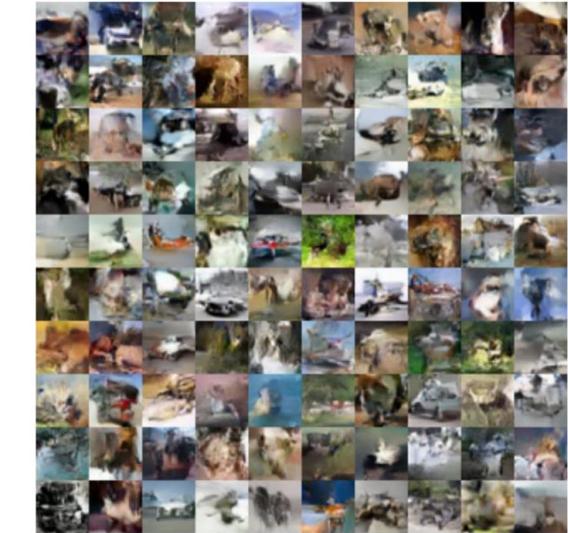
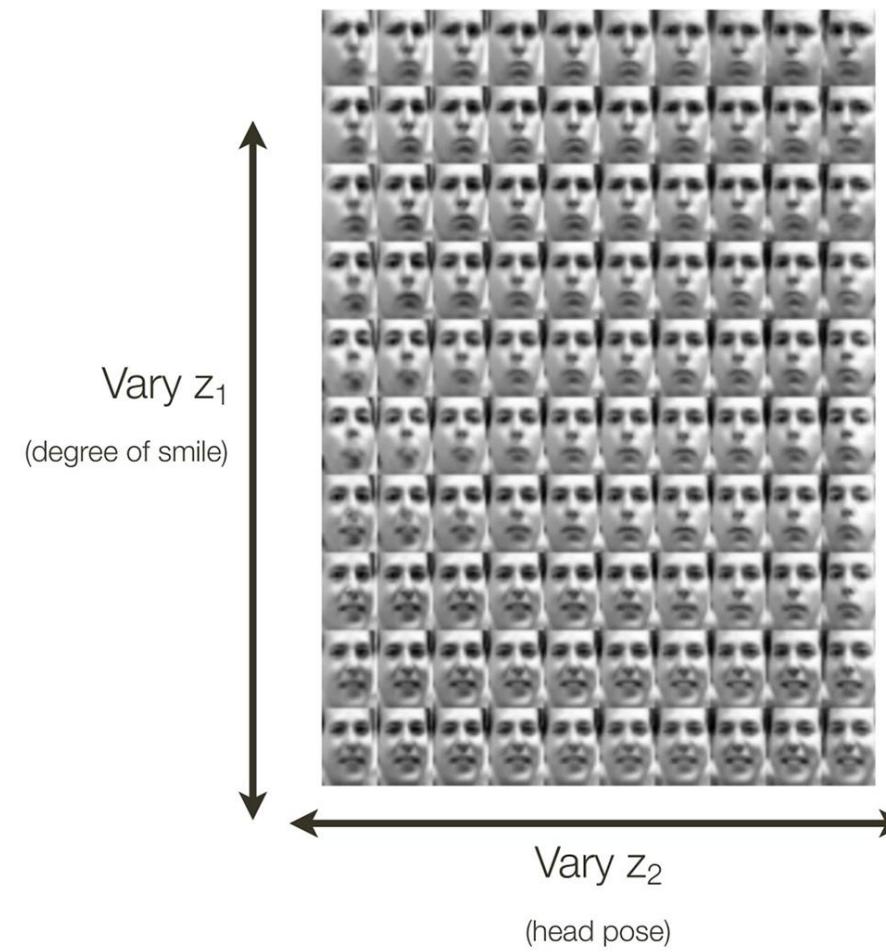
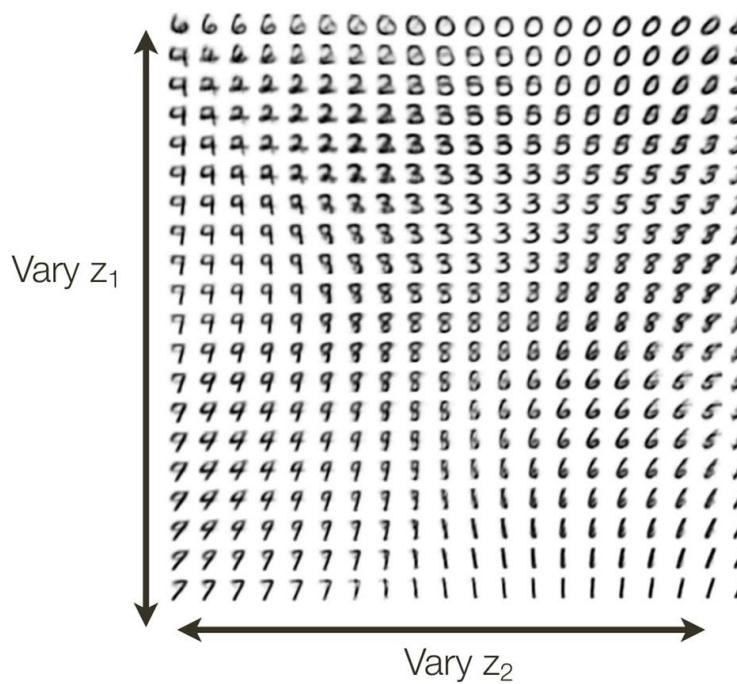


Autoencoders (AE)





VAE – Data Generation Examples



32x32 CIFAR-10



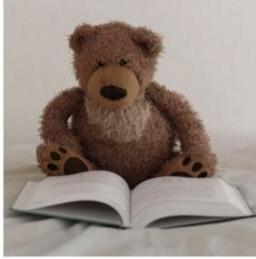
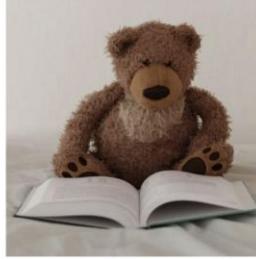
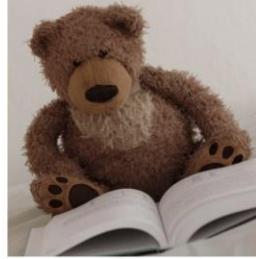
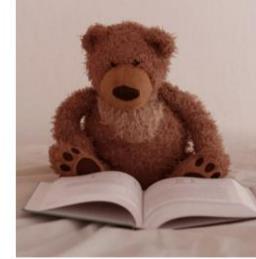
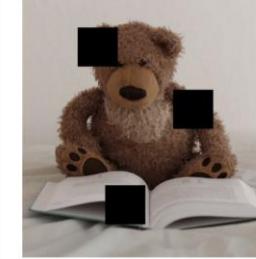
Labeled Faces in the Wild

How To Train Deep Networks Successfully in Practice?

- Data Augmentation
- Weight Initialization
- Learning Rate Tuning
- Other Common Practices

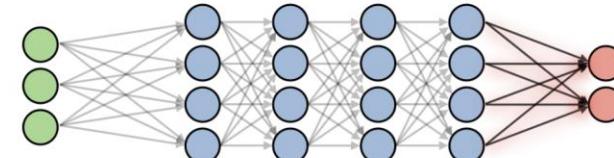
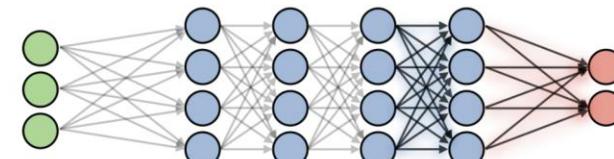
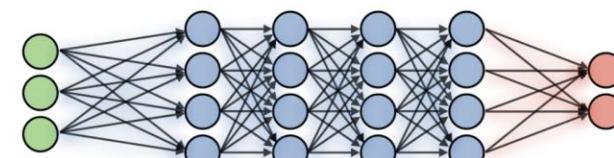
Data Augmentation

- Deep learning models usually need a lot of data to be properly trained.
- It is often useful to **get more data from the existing ones**.
- Usually augmented on the fly during training.

Original	Flip	Rotation	Random crop	Color shift	Noise addition	Information loss	Contrast change
							
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row	• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

Weight Initialization

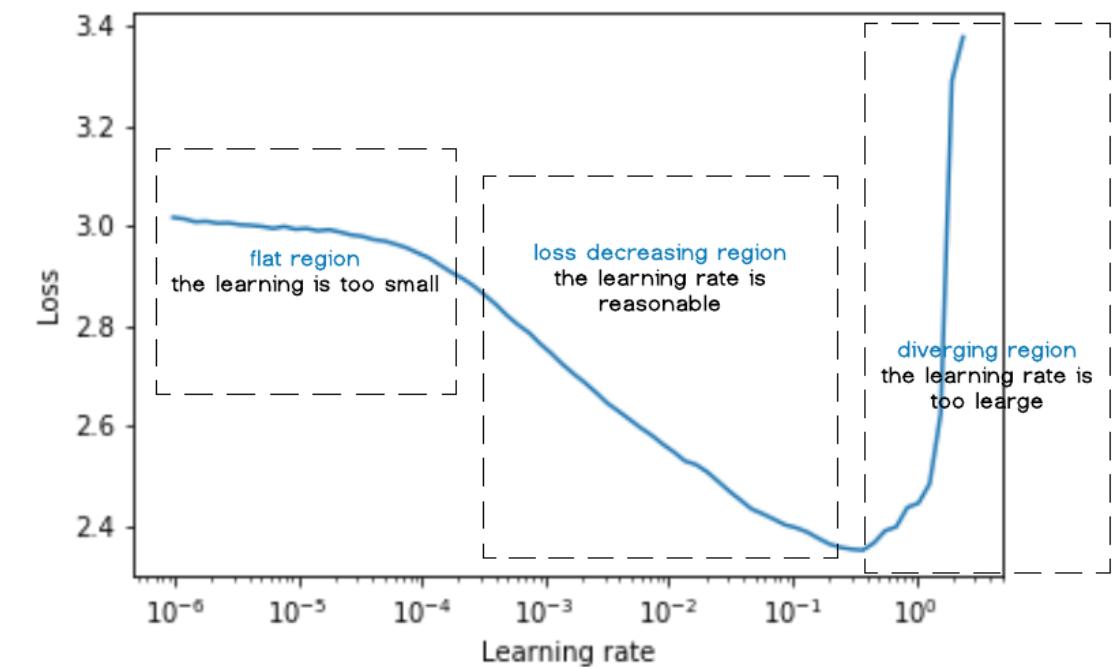
- Train from scratch
 - Randomly initialize weights
 - Different random initialization
 - Xavier
 - Kaiming/MSRA
- Transfer Learning
 - Training need lots of data and time.
 - Leverage pre-trained weights on huge datasets towards your own use case.

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones



Learning Rate Tuning

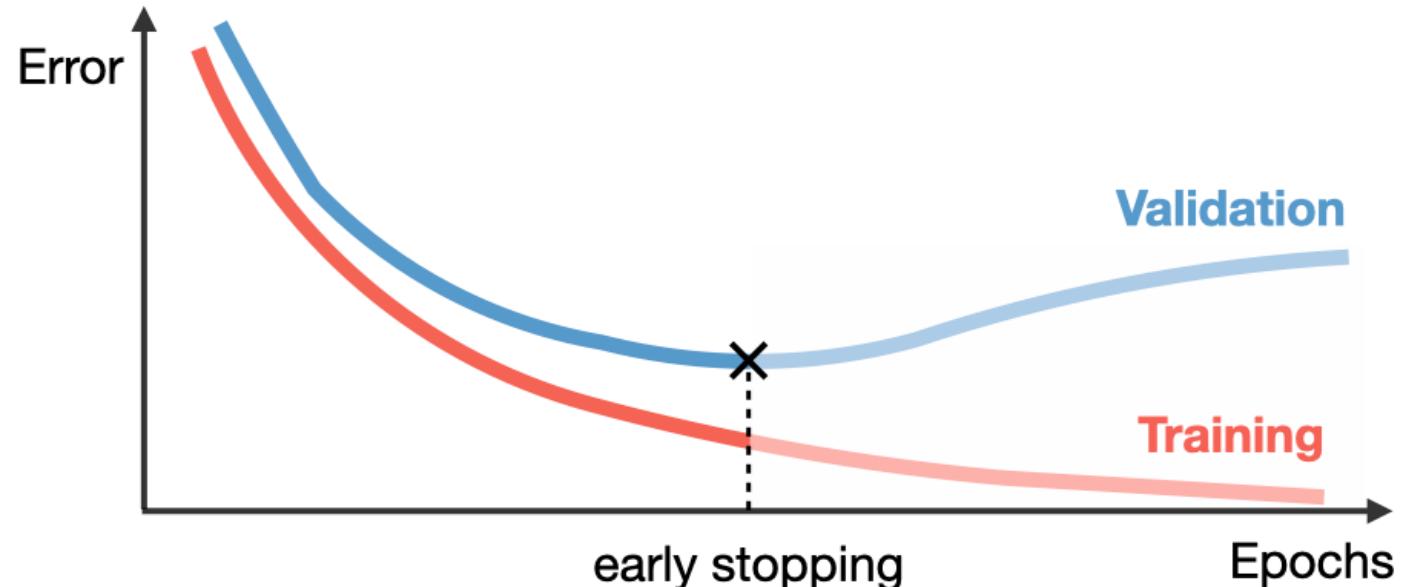
- The learning rate is the most important hyperparameter in deep learning.
 - If bound by time, focus on tuning it.
- Tuning by monitoring the loss curve that plot the objective function over time.
 - Start with a small learning rate
 - Increase it until loss starts to diverge/increase/NaN
- Tune it on a validation set!!!
 - Commonly split all your data into train-validation-test by ratio: 8:1:1
- Grid search if you have lots of GPU
- Schedule your learning rate, or
- ...use Adam instead of SGD solver





Other Common Practices

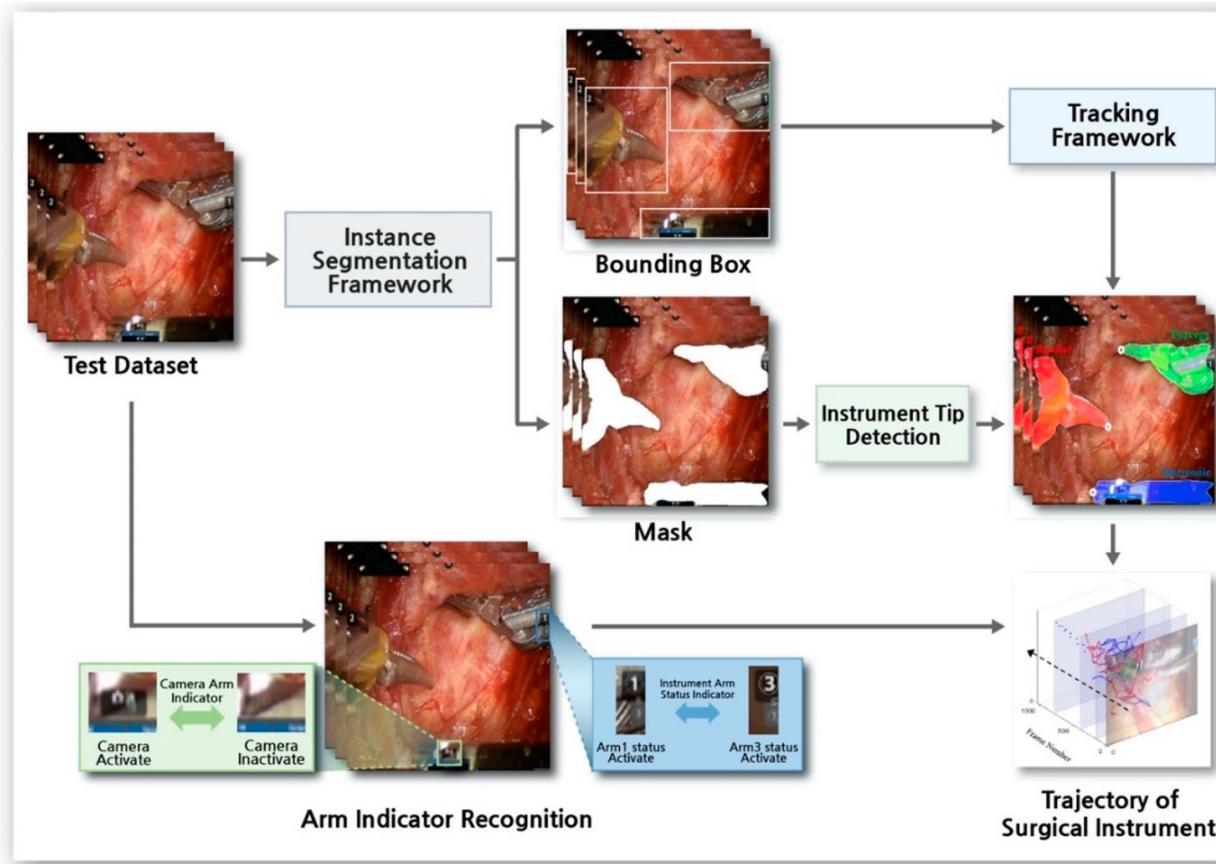
- Early Stopping
- Check gradient
 - Magnitude too small/large?
- Overfitting a small batch first
- Add regularization if
 - Overfitting: training loss is small, but validation loss is large
 - Dropout, L2 regularization of weights
- Normalization!
 - Normalize your data
 - Use normalization layers like BatchNorm/LayerNorm/etc.
- Always start with a small training set for sanity check





When Do Deep Learning Methods Face Challenges?

1. When it's **hard to collect data** for the task - simulated or real.
2. When it's hard to obtain **reliable annotation** and self-supervision is not an option.



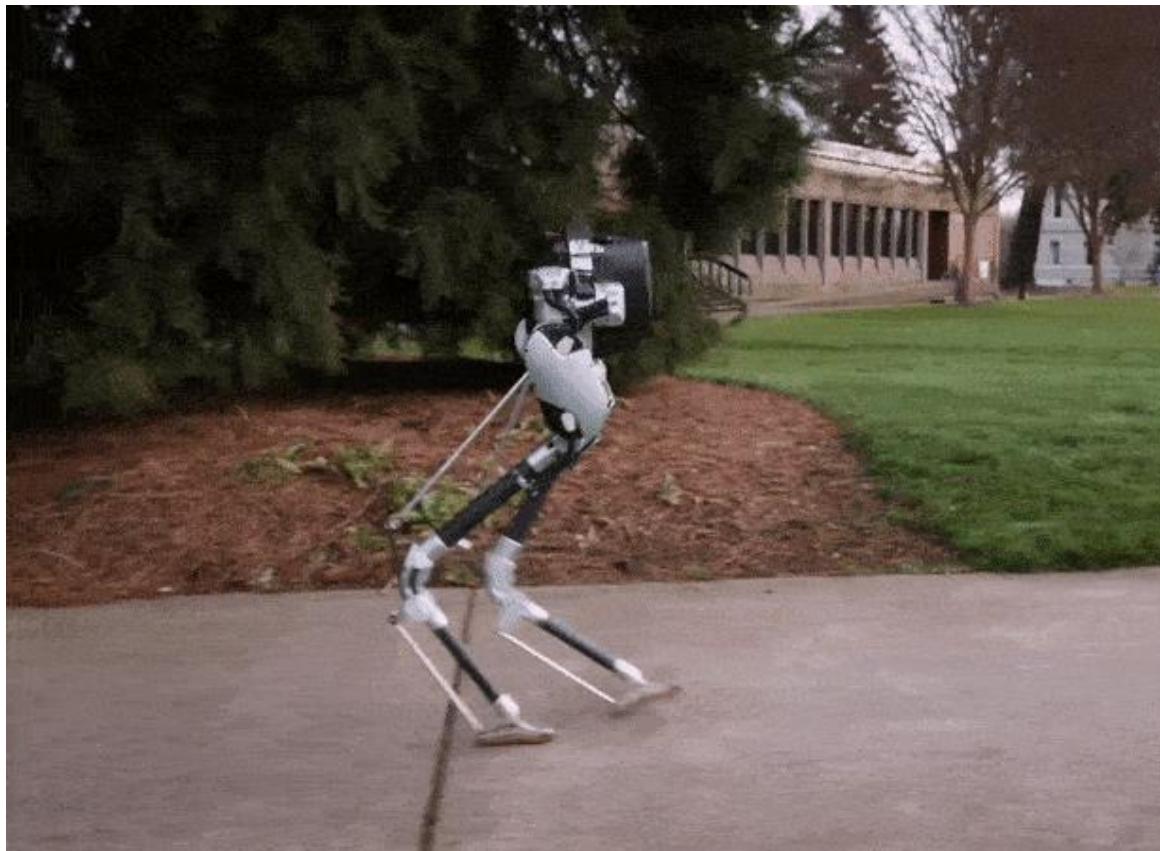
Example

- Robots for surgical assistance
- Pharmaceutical studies - e.g. Simulation of biochemical reactions of new drugs with unknown activation mechanisms

Above example use cases are NOT impossible to tackle via Deep Learning - just that there may be simpler or better ways to approach them in the absence of a proper large-scale dataset.

When Do Deep Learning Methods Face Challenges?

3. When the function to be optimized is **NOT** differentiable



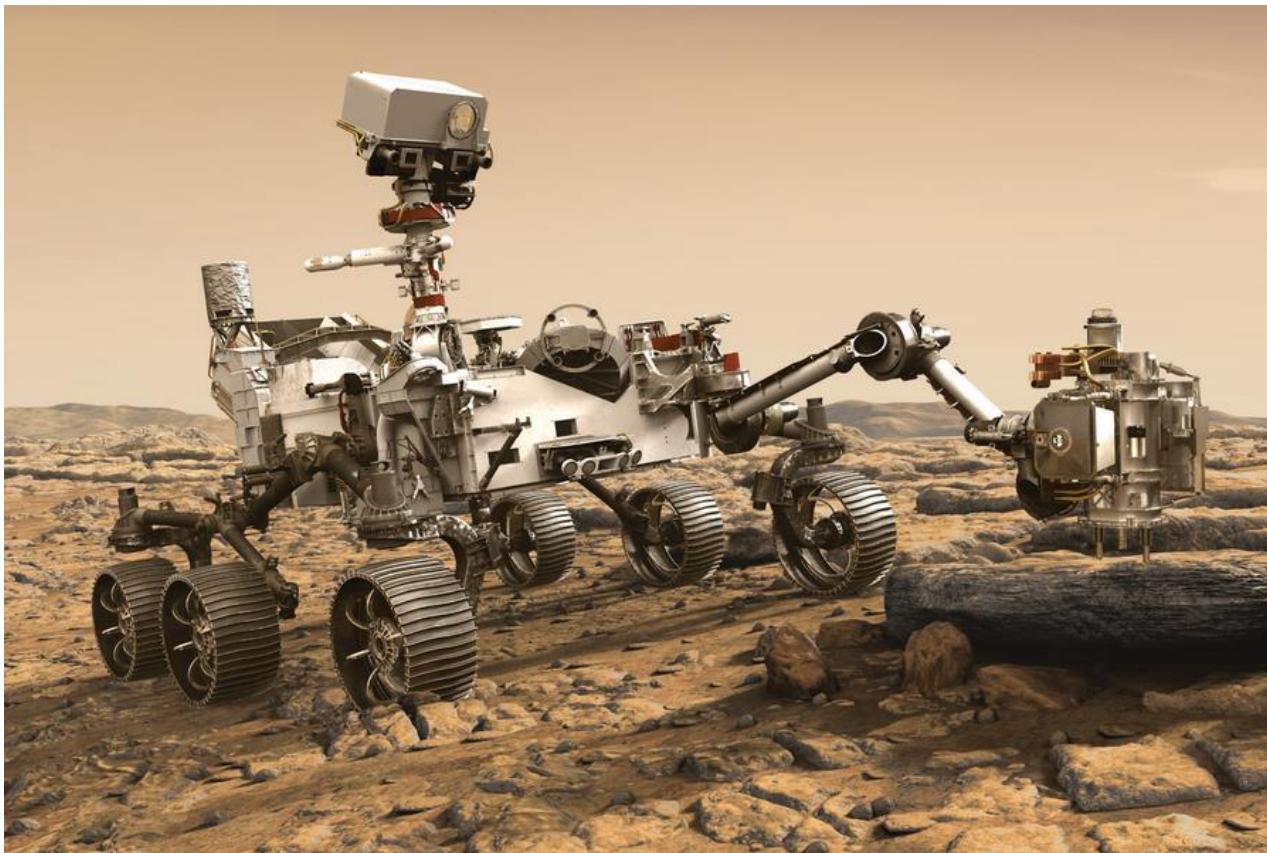
Example

- Legged robots locomotion
- Game playing - e.g. Atari, Go, etc

“Typically” problems that are tackled via RL usually have optimization functions that are hard to define in a differentiable manner.

When Do Deep Learning Methods Face Challenges?

4. When the data is of **lower dimension and complexity** - e.g. in tasks where simpler algorithms suffice.
5. When **computational constraint** is the main bottleneck - e.g. on a Raspberry Pi.



Example

- Edge detection for non-critical processes (e.g. Canny detector may suffice)
- Certain vision task on the Mars rover (where onboard compute and power is limited)



Next Week

- Digital camera & image
 - Camera selection
- Color transformations (RGB, Grey, HSV)
- Pinhole camera model
 - 3D rotation representations (Euler angle, axis-angle, quaternion)
- Lens distortion
- Homogeneous coordinates
- Geometric transformations

References for Next Week

- FP2011:
 - Chapter 1, Section 3.1, 3.4, 3.5
- Sz2011:
 - Section 2.1, 2.3, 2.3.2, 6.3.5
- Co2011:
 - Chapter 2, Section 11.1.4
- HZ2003:
 - Section 2.2, 2.3, 2.4, 3.1, 3.4, 6.1
- Linear algebra:
 - Sz2011: section A.1.1, A.1.2, A.1.3, A.2, A.2.1
 - HZ2003: A5.1, A5.2, A5.3