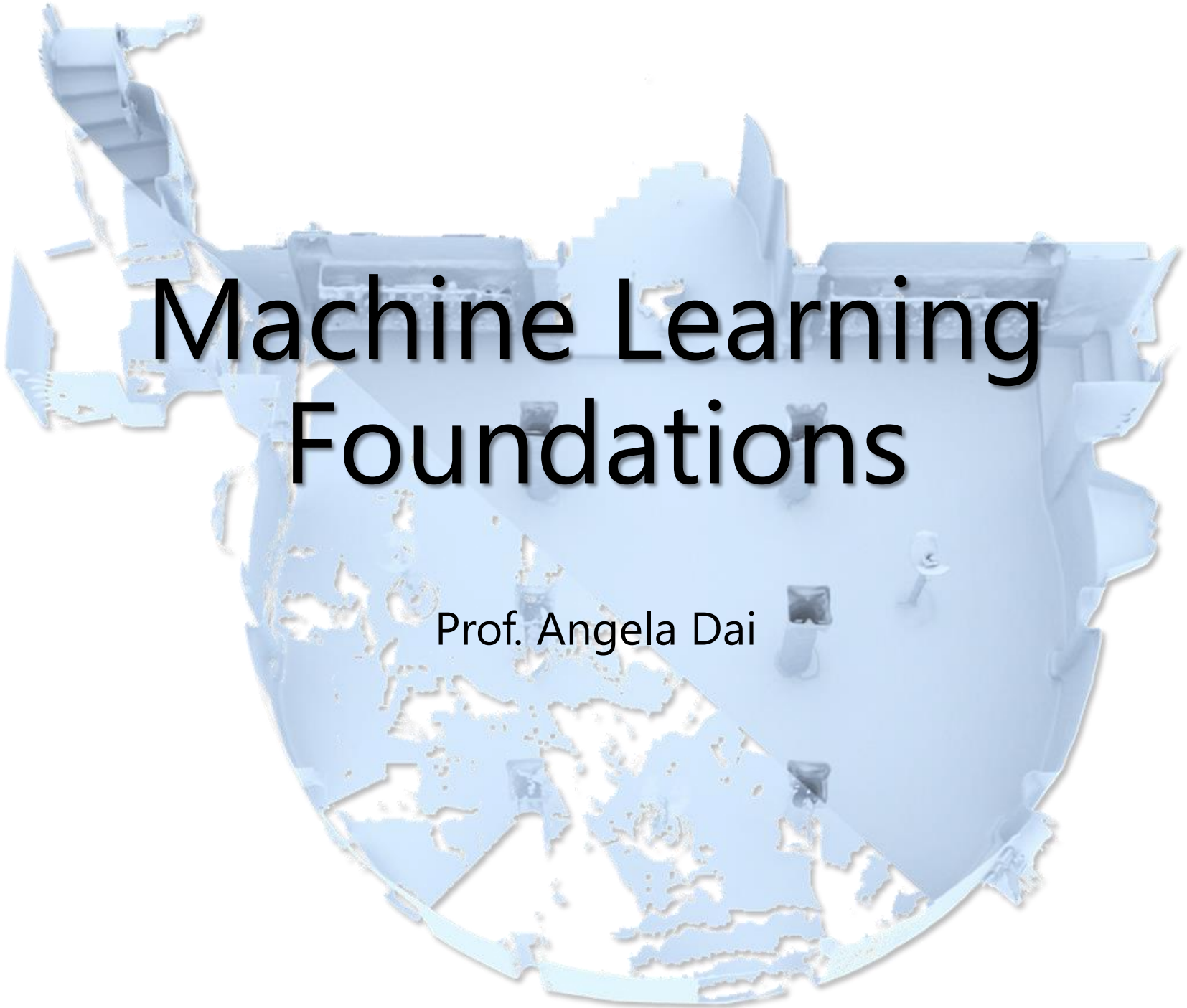
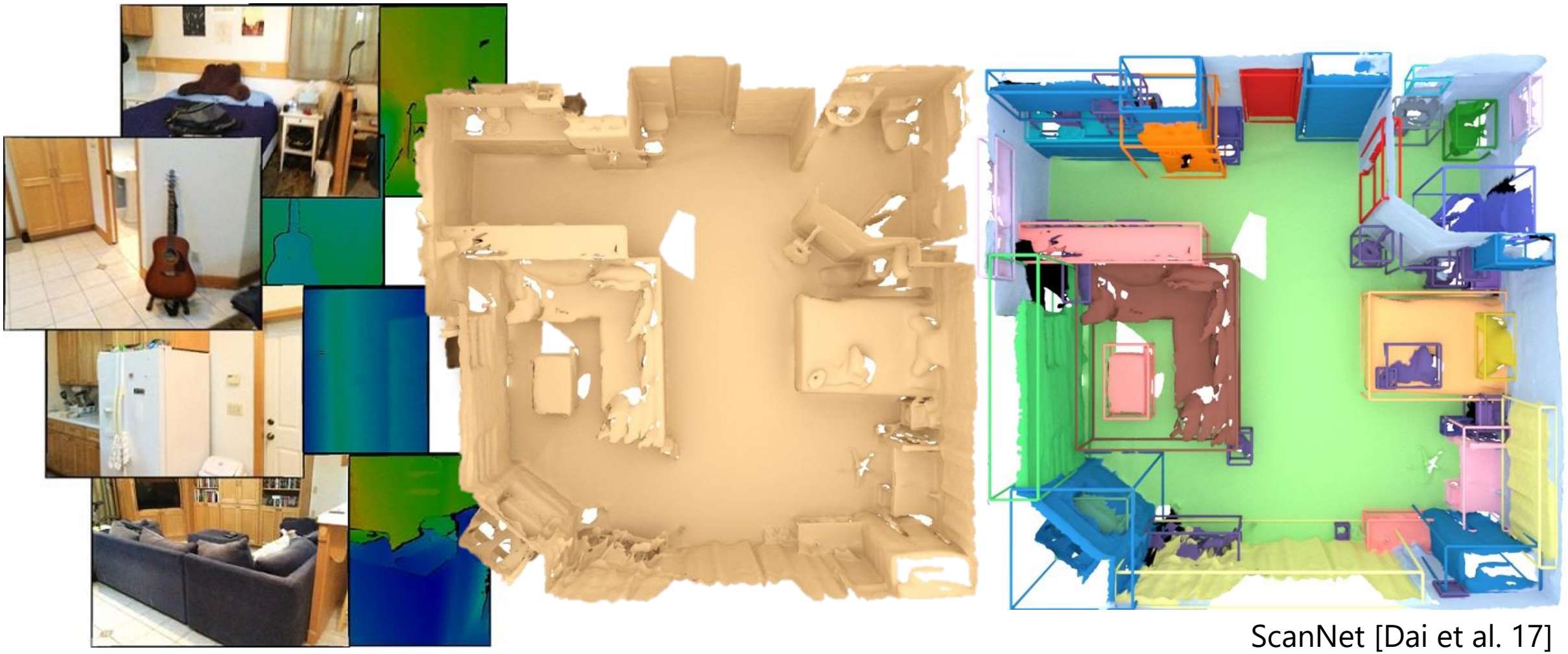


Machine Learning Foundations

Prof. Angela Dai

Brief Recap

Machine Perception of Real-World Environments



ScanNet [Dai et al. 17]

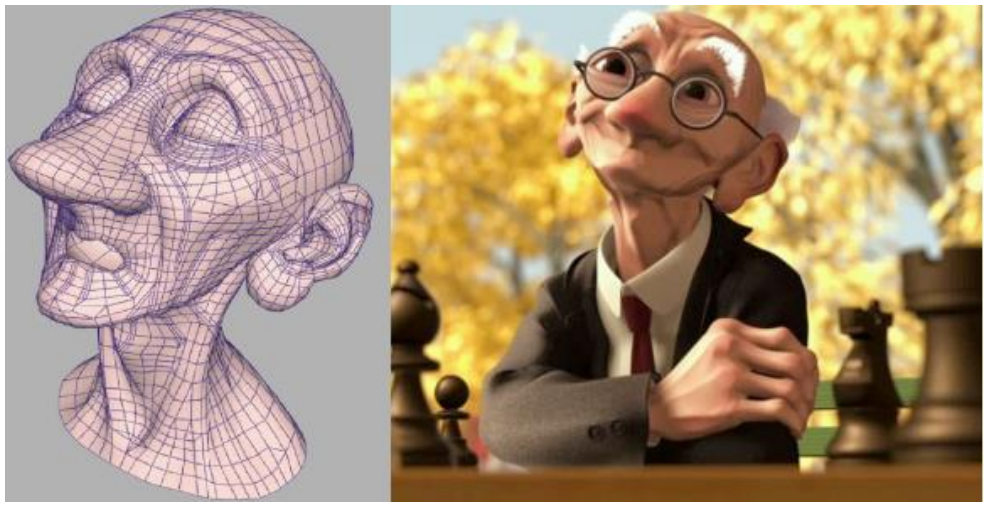
We *perceive* and *interact* with a 3D world

ASIMO, Honda

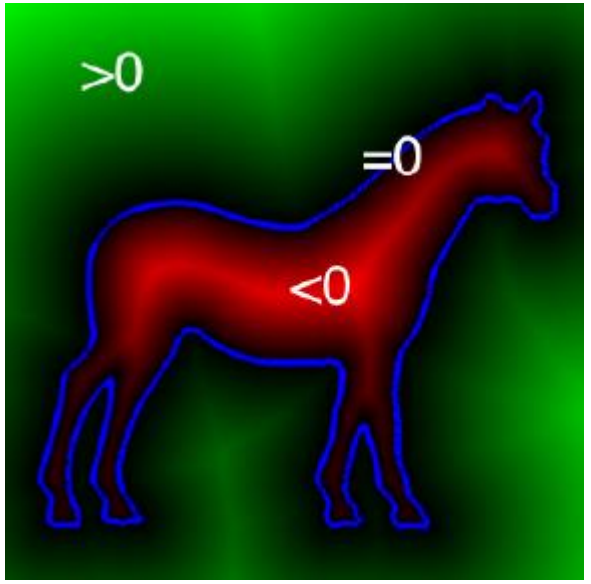
Star Trek TNG (Phantasms)

How to represent 3D?

Discrete:
Meshes,
Point Samples



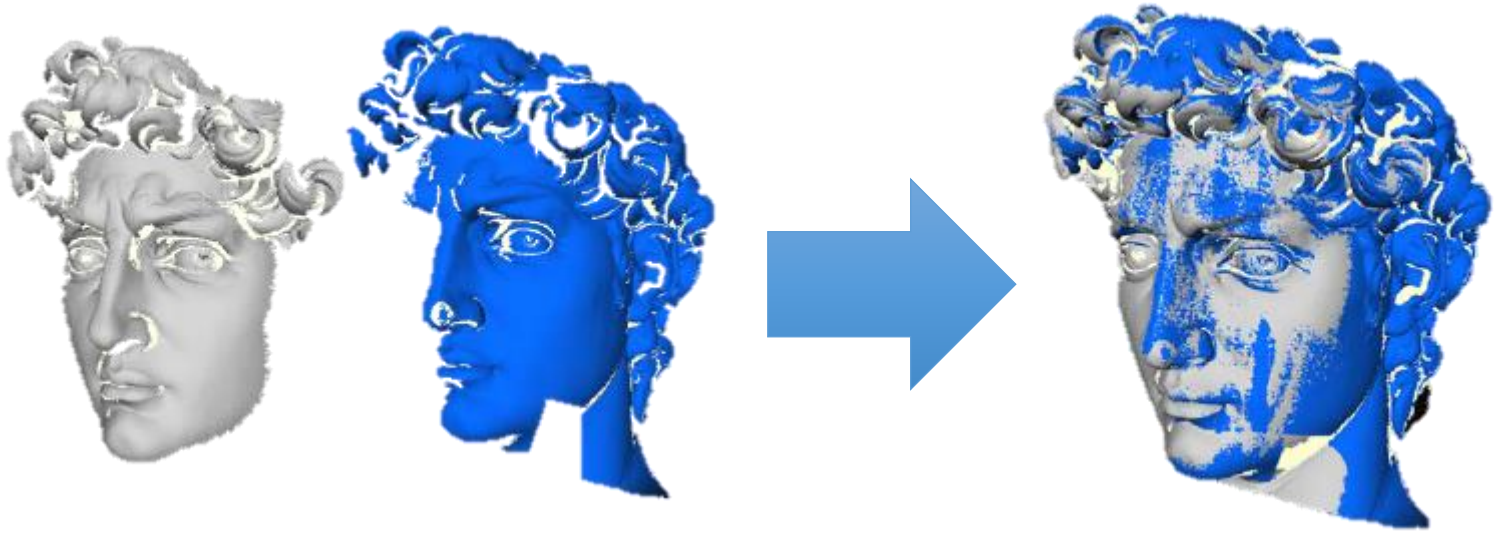
Parametric

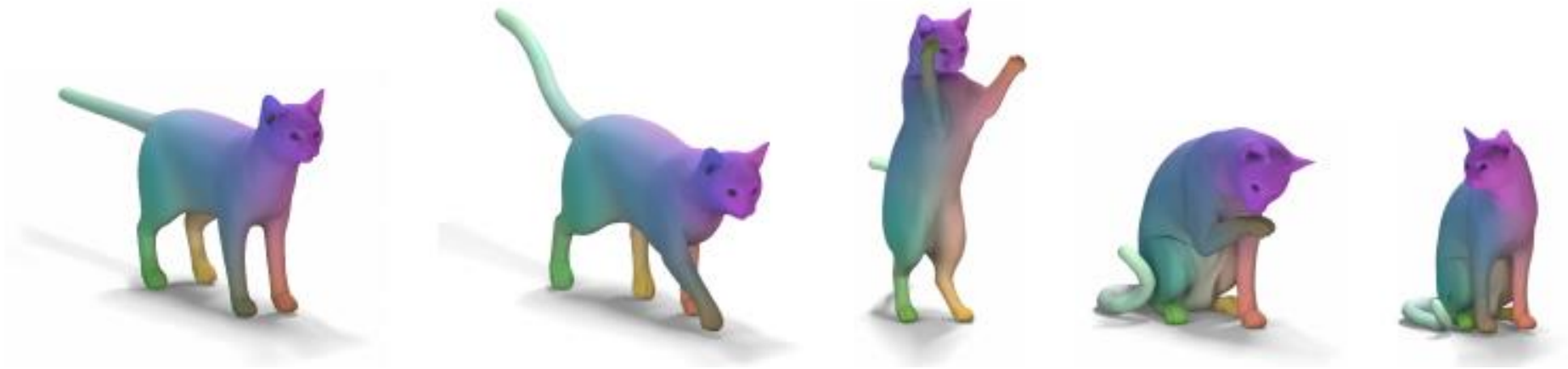


Implicit:
Distance Fields

Classical Geometry Processing

- Meshing
- Rigid Alignment
- Non-Rigid Alignment





Supervised Learning

Cat

Cat

Cat

Cat

Cat

Puppy

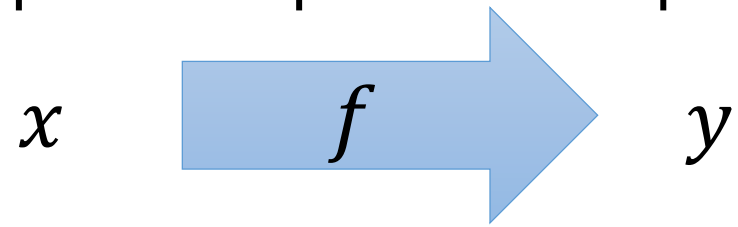
Puppy

Bagel

Bagel

Supervised Learning

- Given labeled training data (x_i, y_i)
- Want to learn to map new inputs to outputs



- Classification task: output y is discrete
- Regression task: output y is continuous

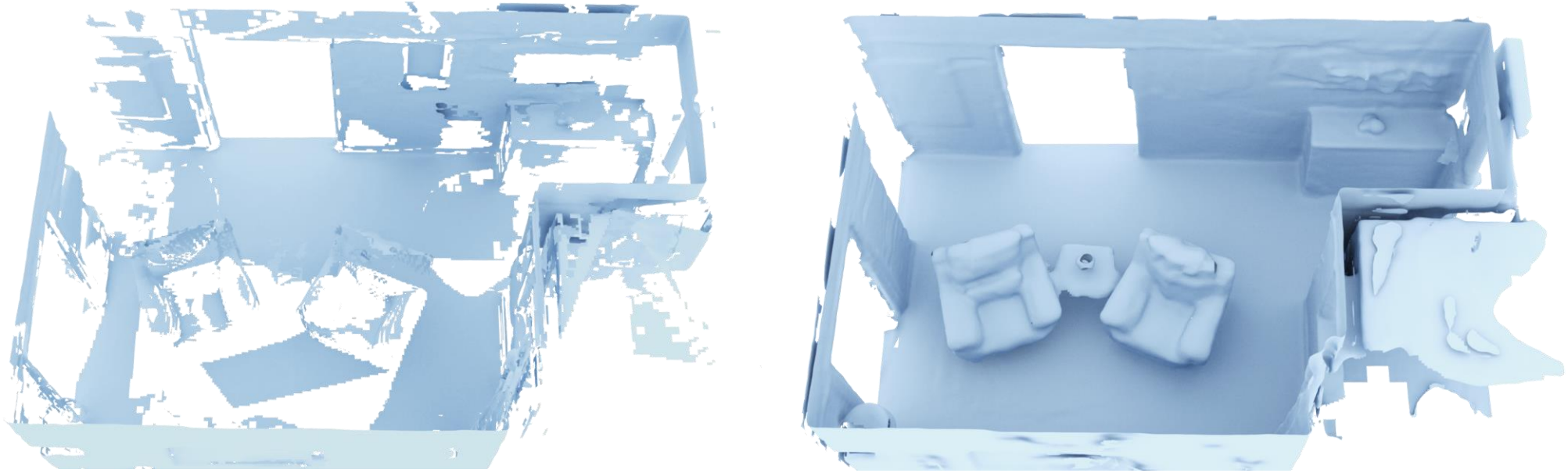
3D Supervised Learning Tasks

- Labeling points on meshes
- Classification for each point

ScanNet

3D Supervised Learning Tasks

- Geometric completion



[Dai et al ‘20] SGNN

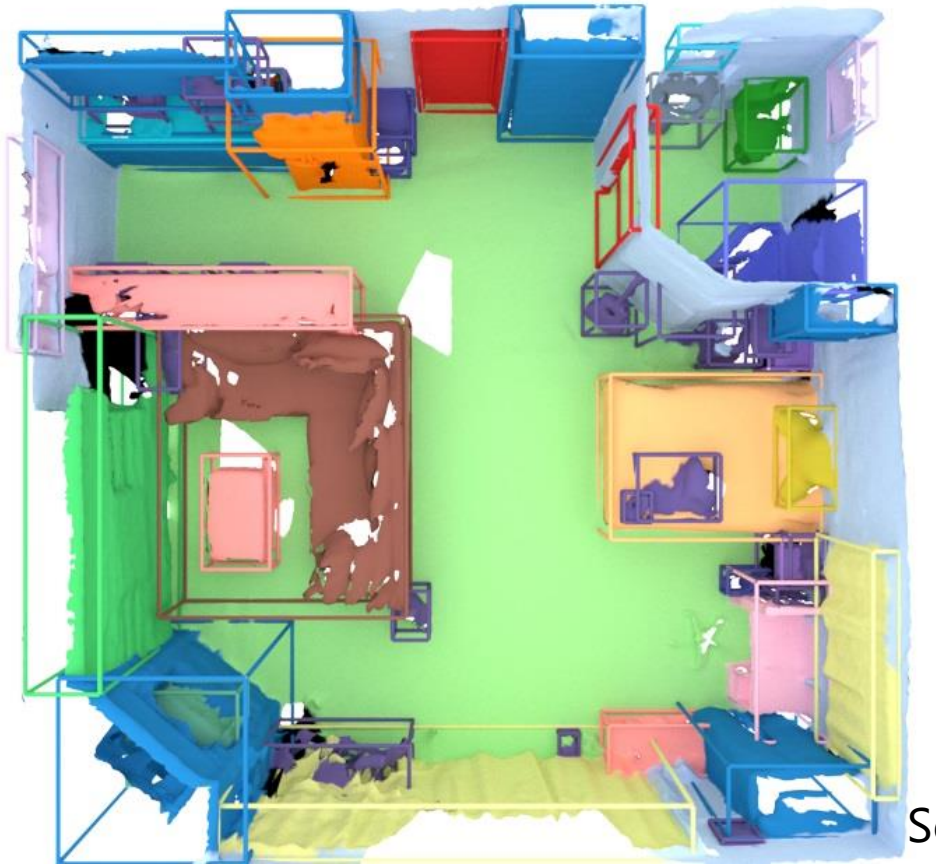
This approach: regressing SDF (distance per voxel)

3D Supervised Learning Tasks

- Instance segmentation

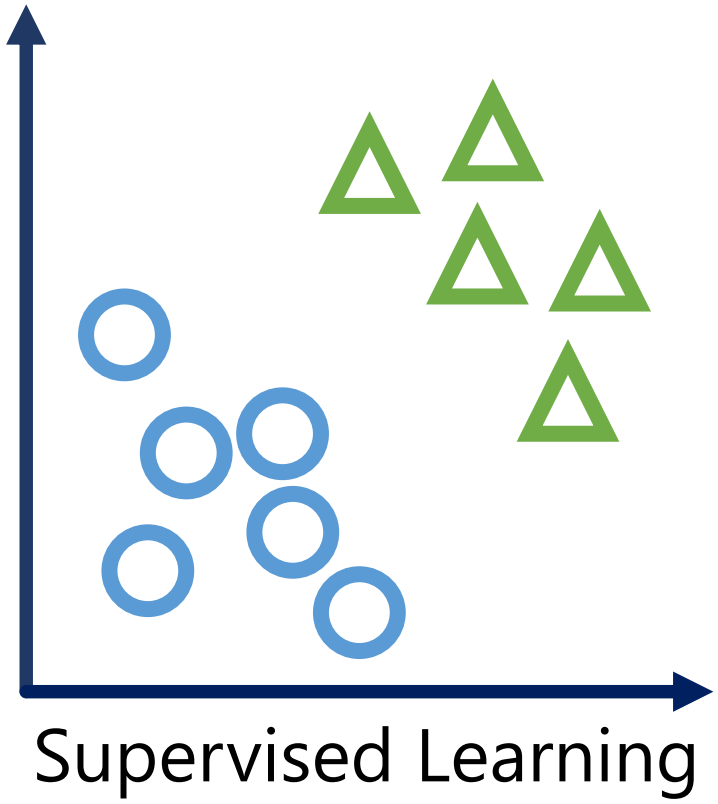
- Classification: object categories
- Regression: object bounding boxes

- Classification/Regression for object instance masks

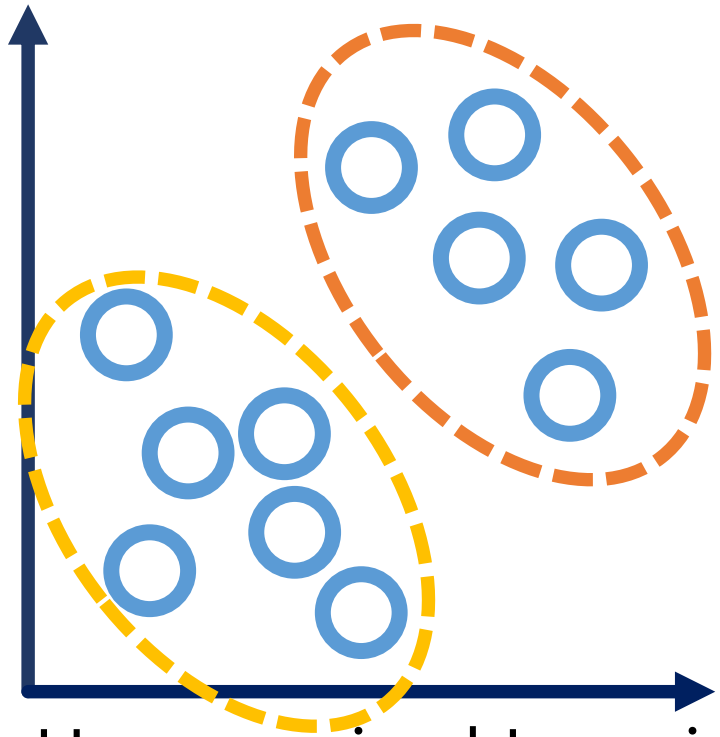


ScanNet

Unsupervised Learning



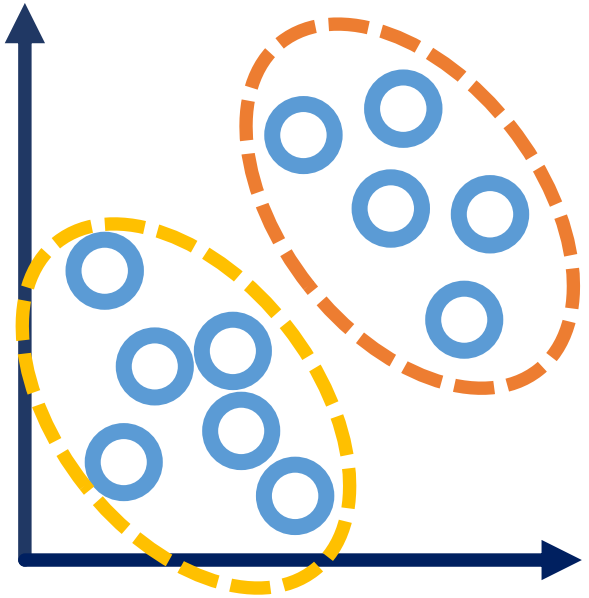
Supervised Learning



Unsupervised Learning

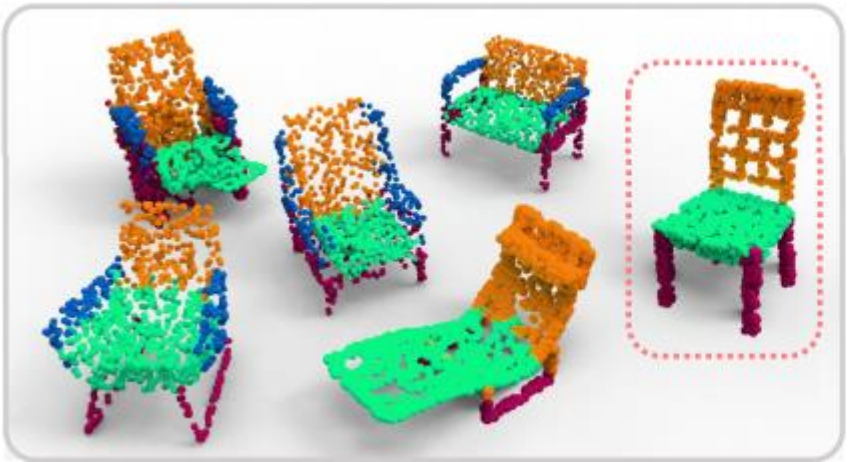
Unsupervised Learning

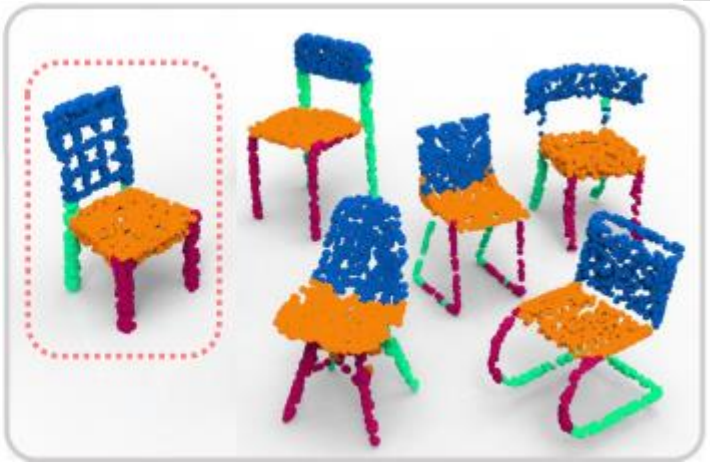
- Given unlabeled dataset
- Want to determine the structure of the data
- Common basic approaches:
  - Clustering
  - Dimensionality reduction (e.g., PCA)



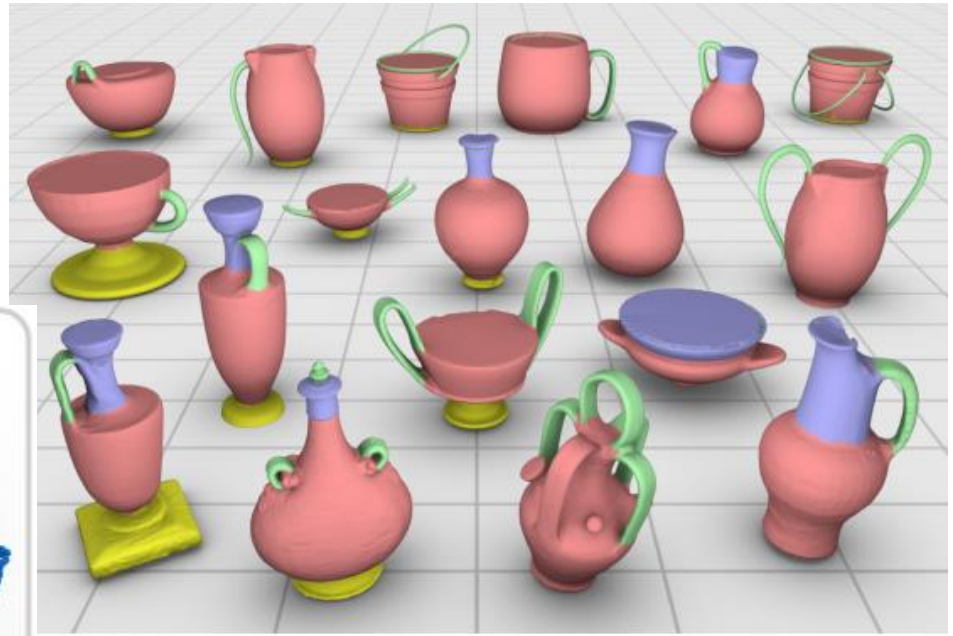
3D Unsupervised Learning Tasks

- Unsupervised co-segmentation of a set of shapes





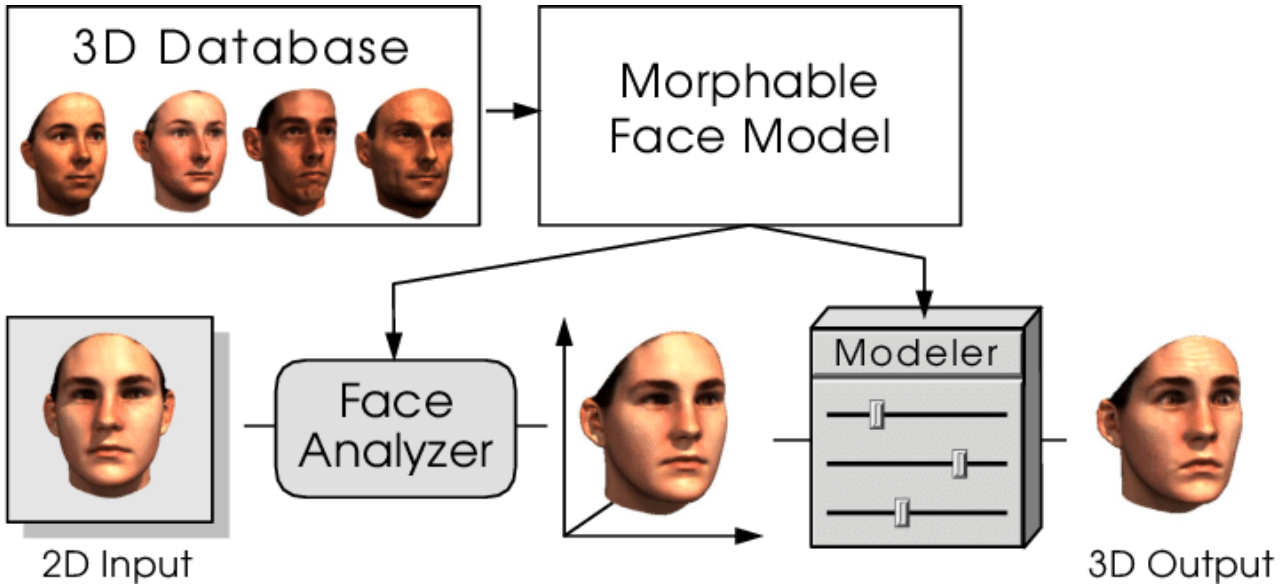
[Zhu et al. ’20]



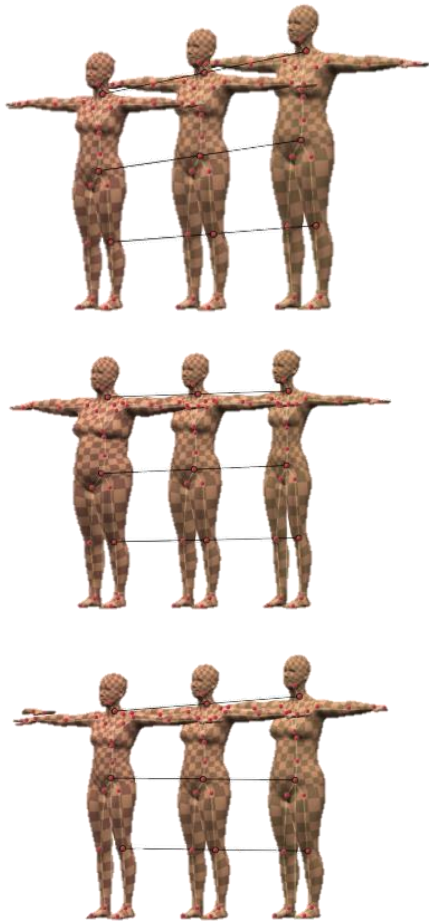
[Sidi et al. ’11]

Dimensionality Reduction for 3D

- Morphable models



[Blanz and Vetter '99]



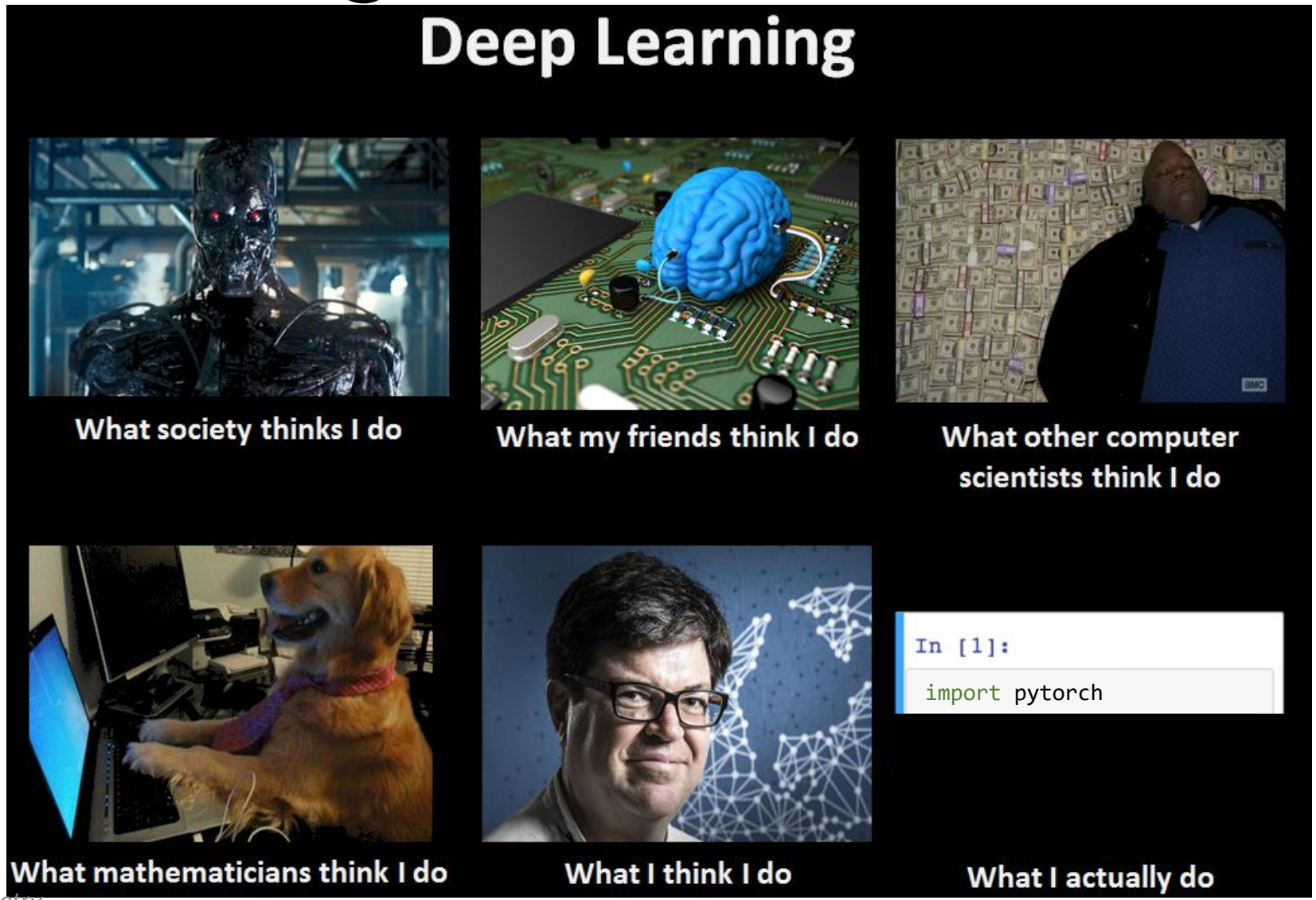
1st principal component

2nd principal component

3rd principal component

[Loper et al. '15]

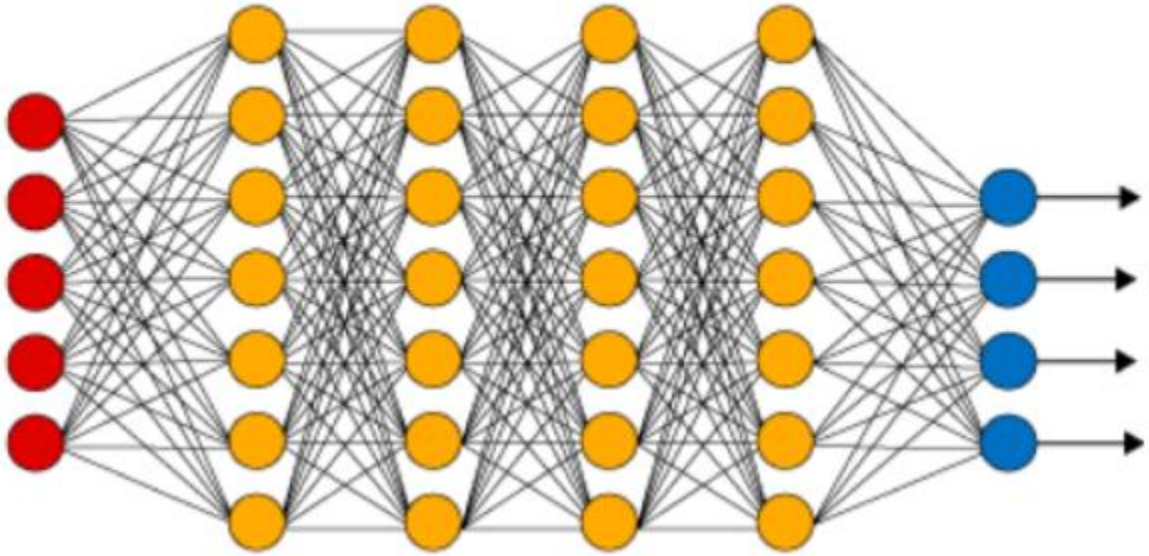
Deep Learning



kdnuggets.com

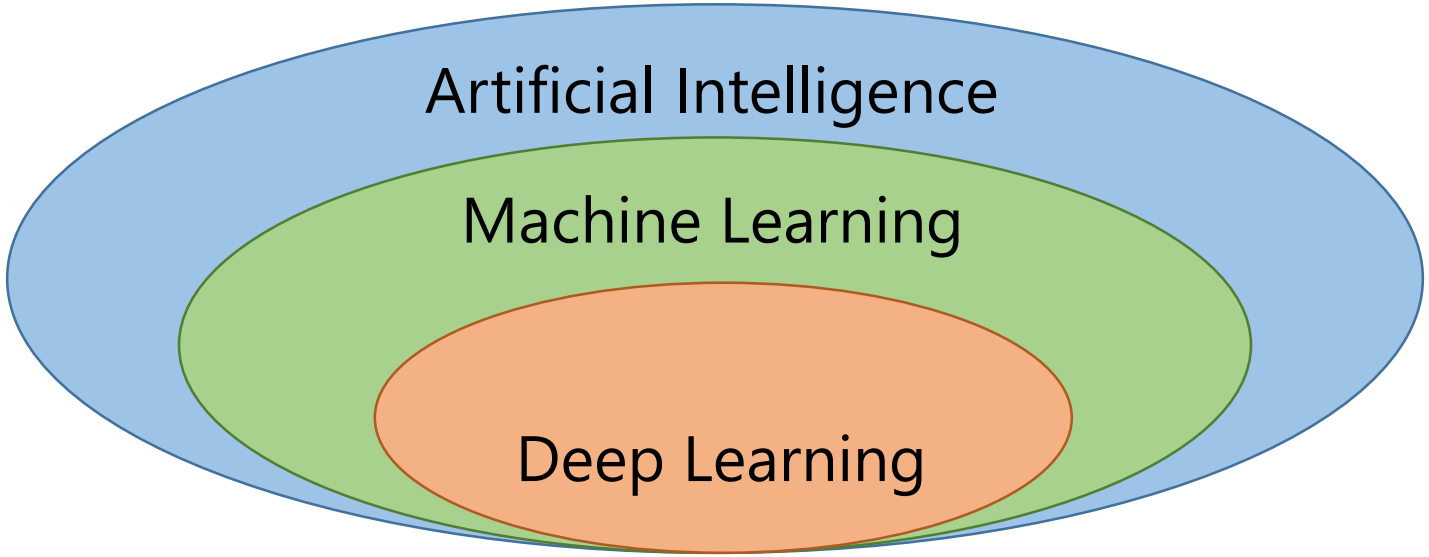
Deep Learning

Introduction to Deep Learning (I2DL) (IN2346)

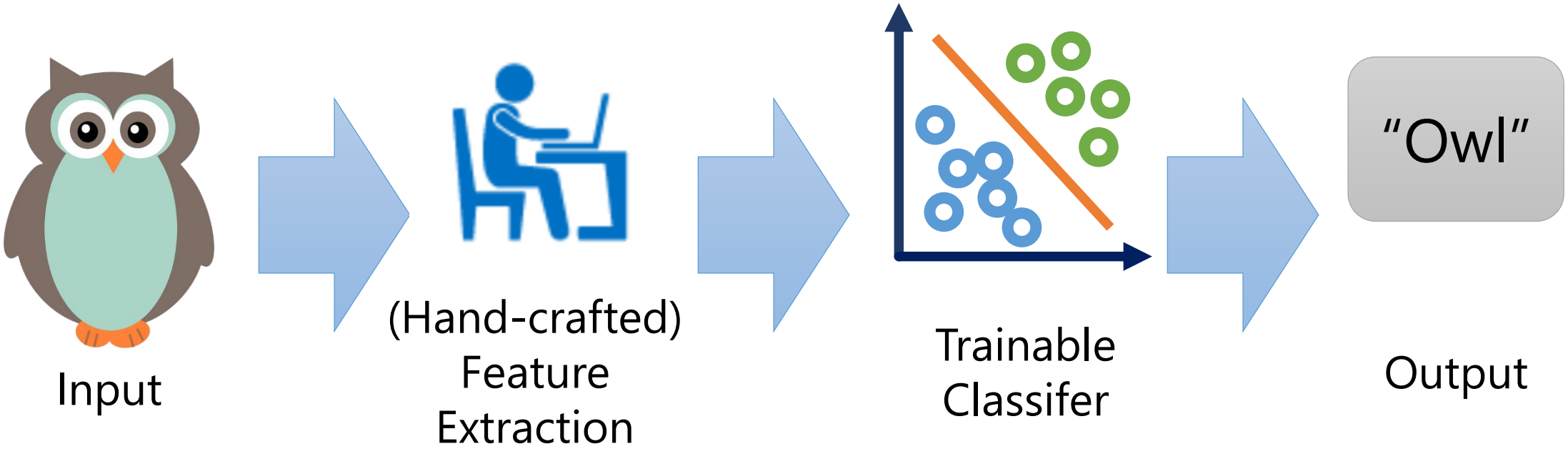


Deep Learning

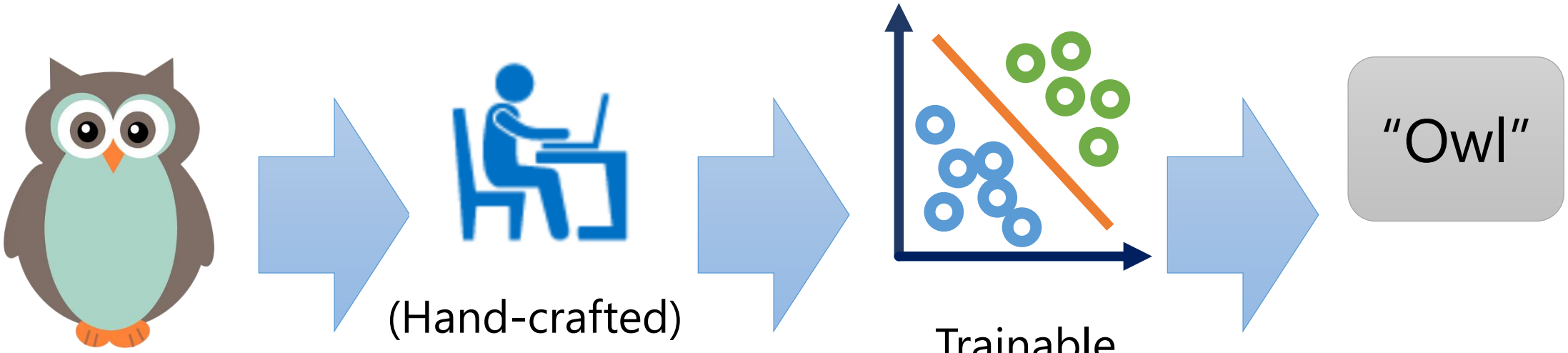
Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.



Traditional pattern recognition



Traditional pattern recognition

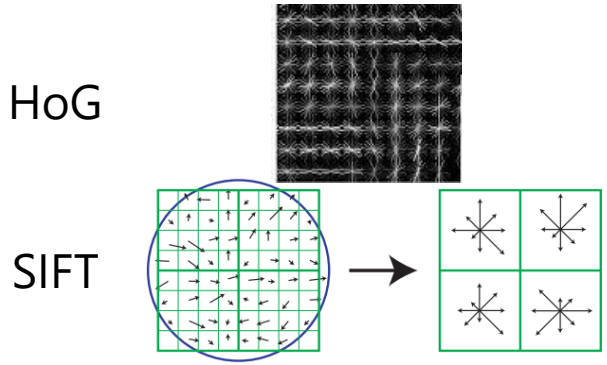


Input

(Hand-crafted) Feature Extraction

Trainable Classifer

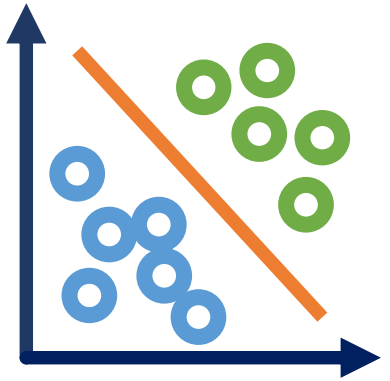
Output



Traditional pattern recognition

Input

(Hand-crafted)
Feature
Extraction

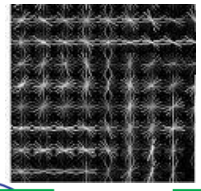


Trainable
Classifer

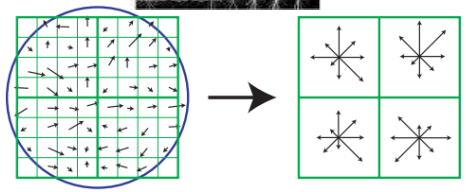
"Owl"

Output

HoG

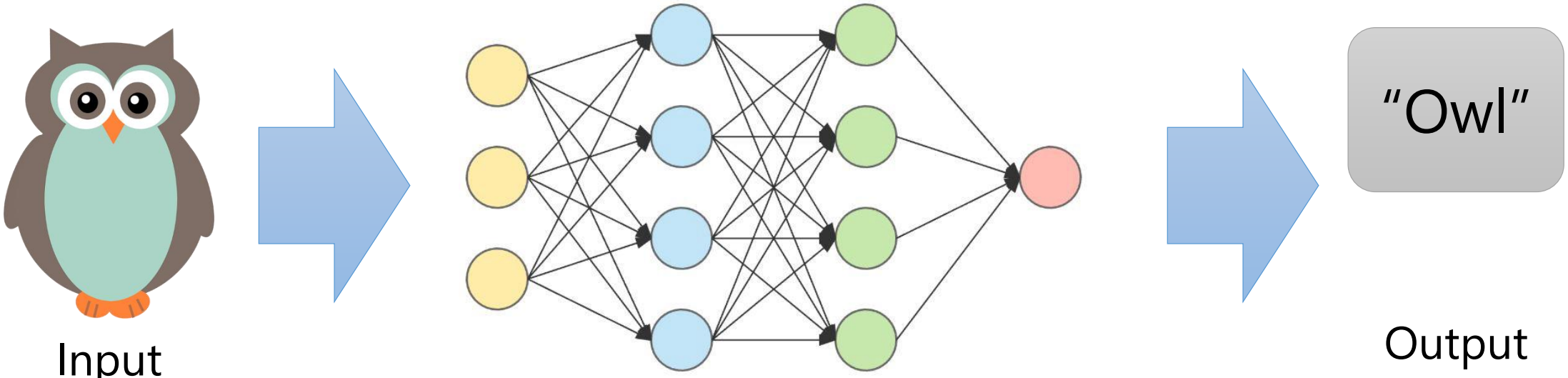


SIFT



Linear Regression
SVM
Decision Trees
Random Forest

Deep Learning



Input

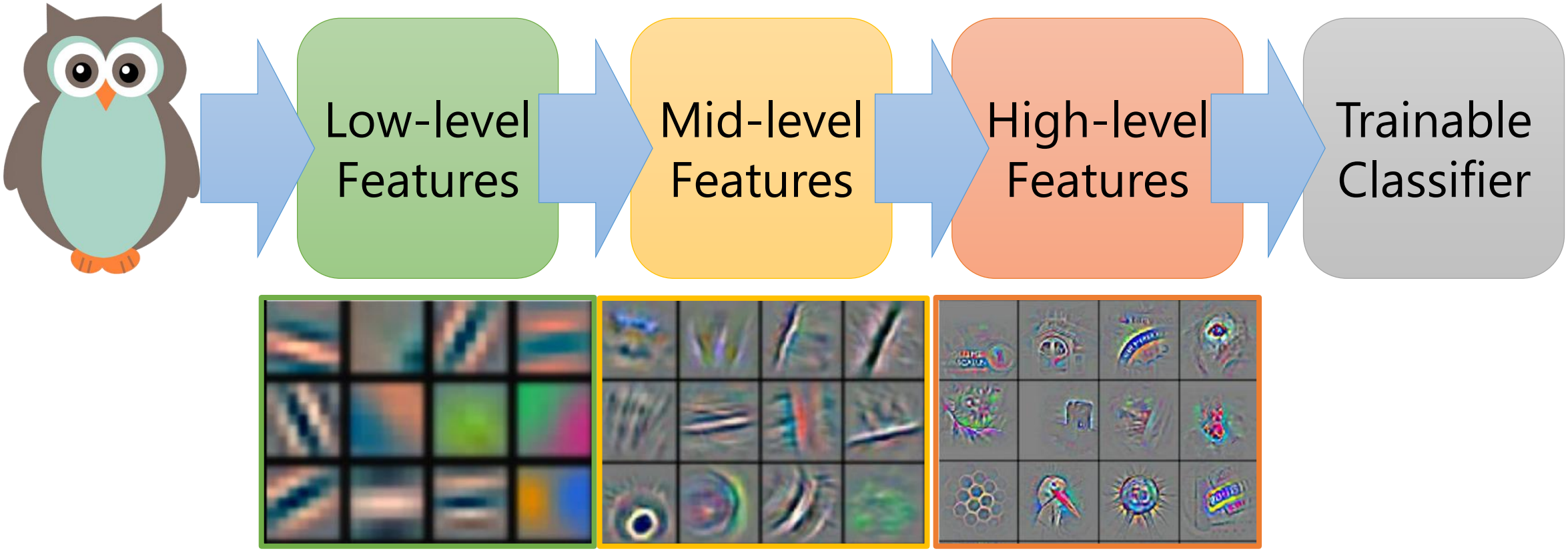
Learned Feature Extraction
and Classification

Output

Want to automatically learn good feature representations for the task

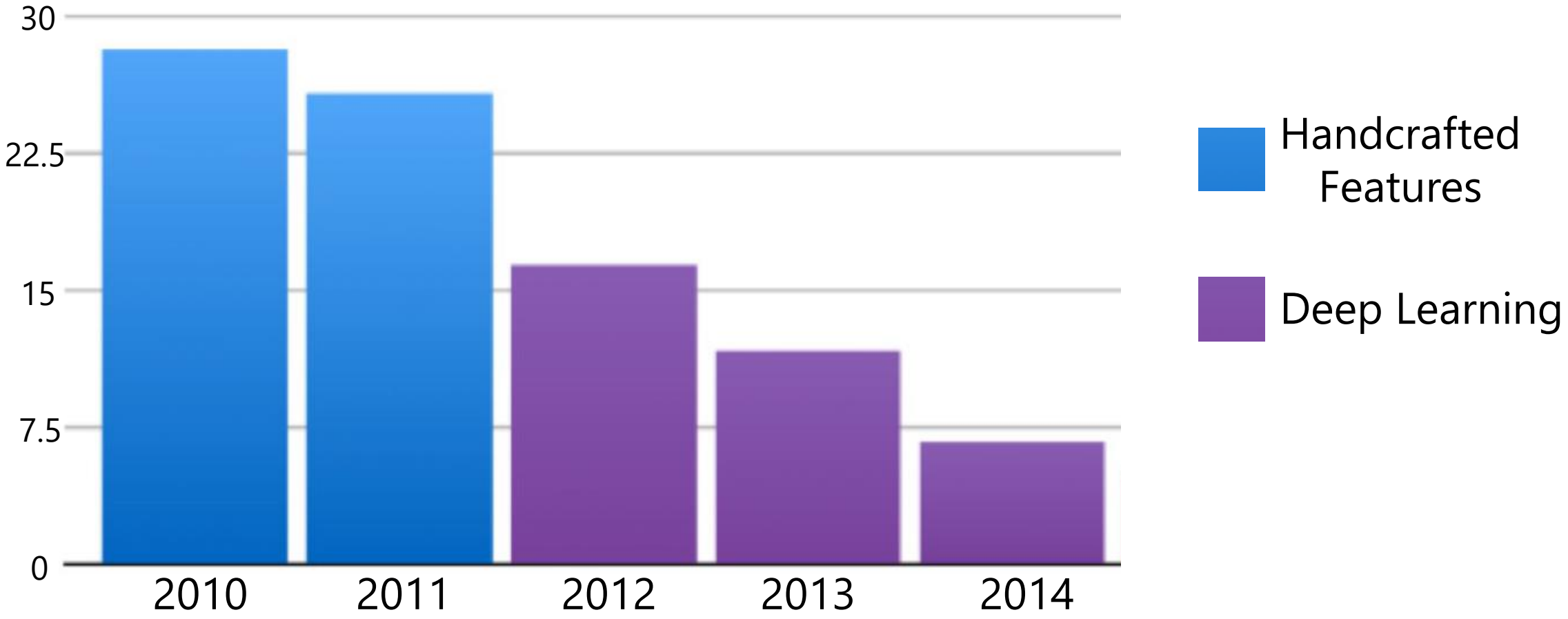
“Deep” Learning

- Multiple stages of non-linear feature transformation



Features learned from ImageNet training [Zeiler and Fergus ’13]

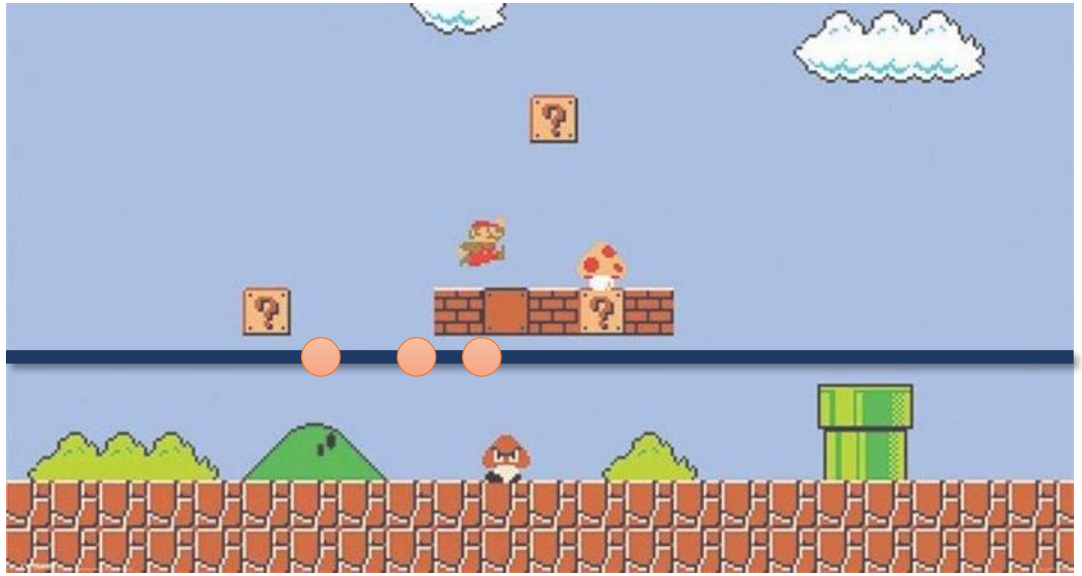
Dramatic improvements in many tasks

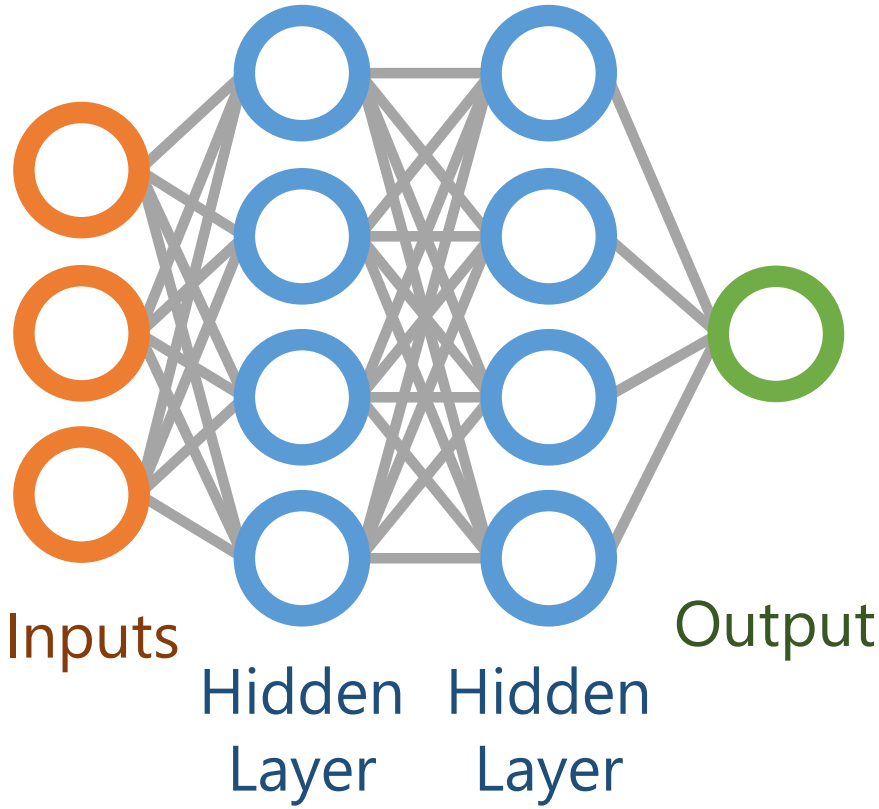


ImageNet 1000-class classification error

A simple neural network

- Given last three horizontal locations of Mario, predict the next horizontal location

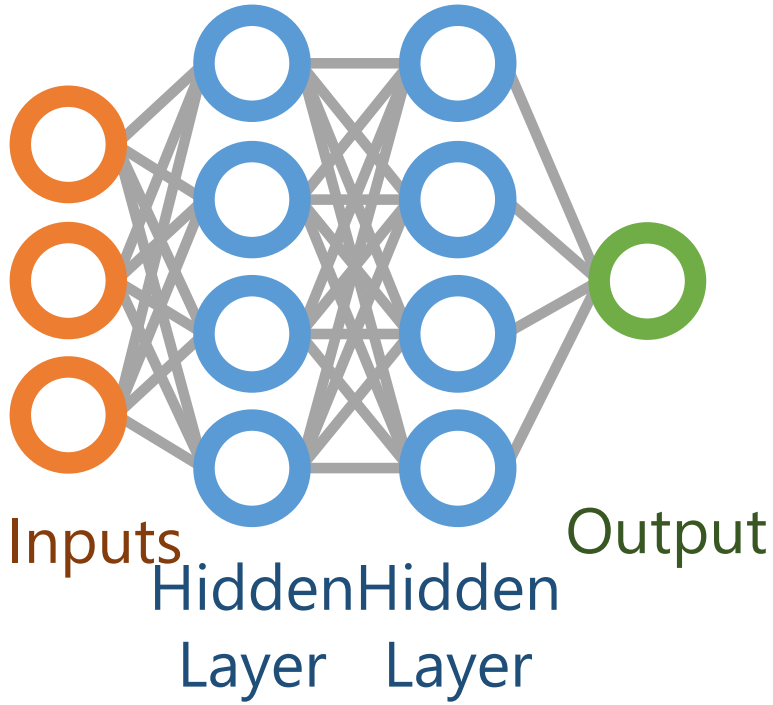




A simple neural network

- Forward pass

```
# sigmoid activation function
f = lambda x: 1.0/(1.0 + np.exp(-x))
# random input vector of 3 numbers
x = np.random.randn(3, 1)
# calculate first hidden layer activations
h1 = f(np.dot(W1, x) + b1)
# calculate second hidden layer activations
h2 = f(np.dot(W2, h1) + b2)
# output neuron
out = np.dot(W3, h2) + b3
```



- Parameters to be learned: `W1,b1,W2,b2,W3,b3`

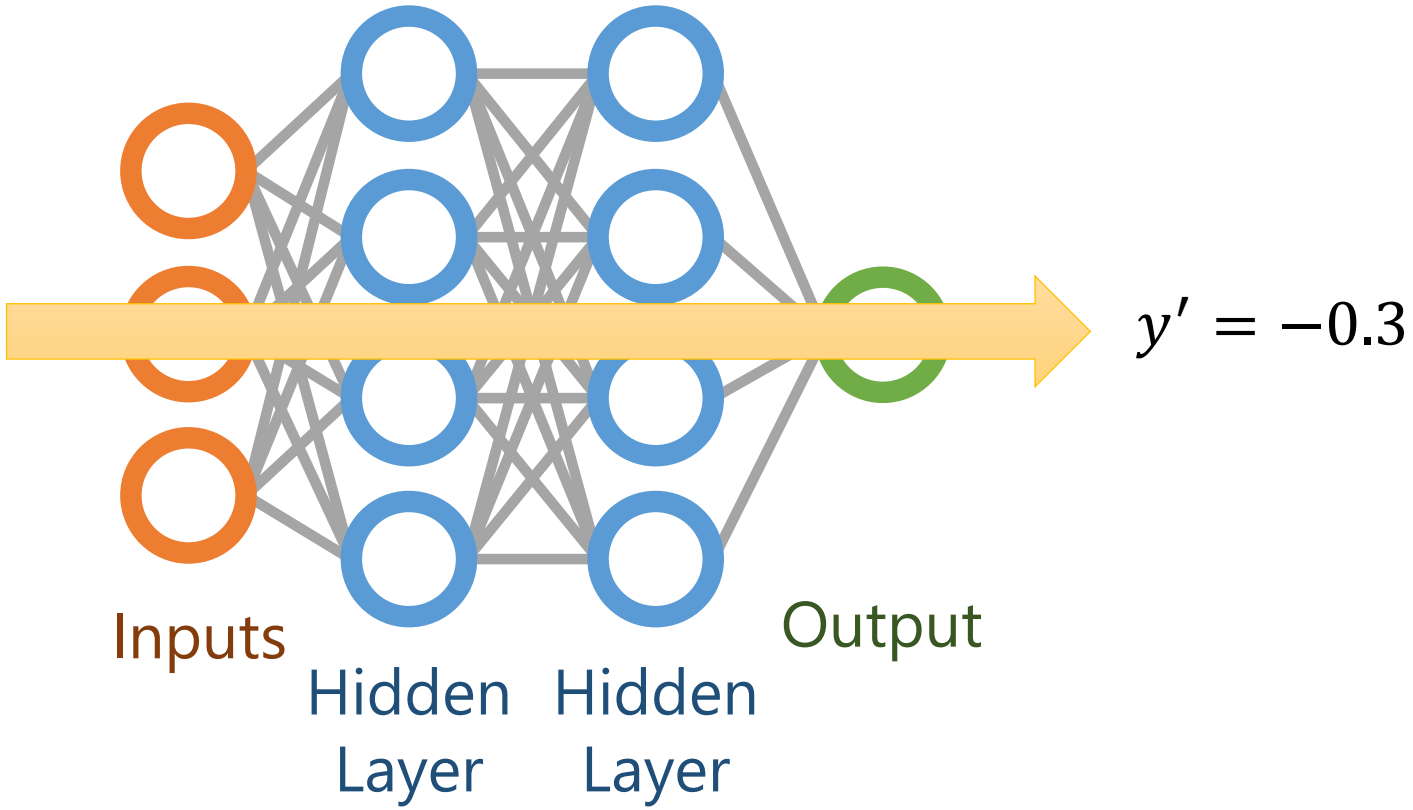
A simple neural network

- Forward pass

$$x_0 = 22.5$$

$$x_1 = 24.7$$

$$x_2 = 25.3$$



$$y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3$$

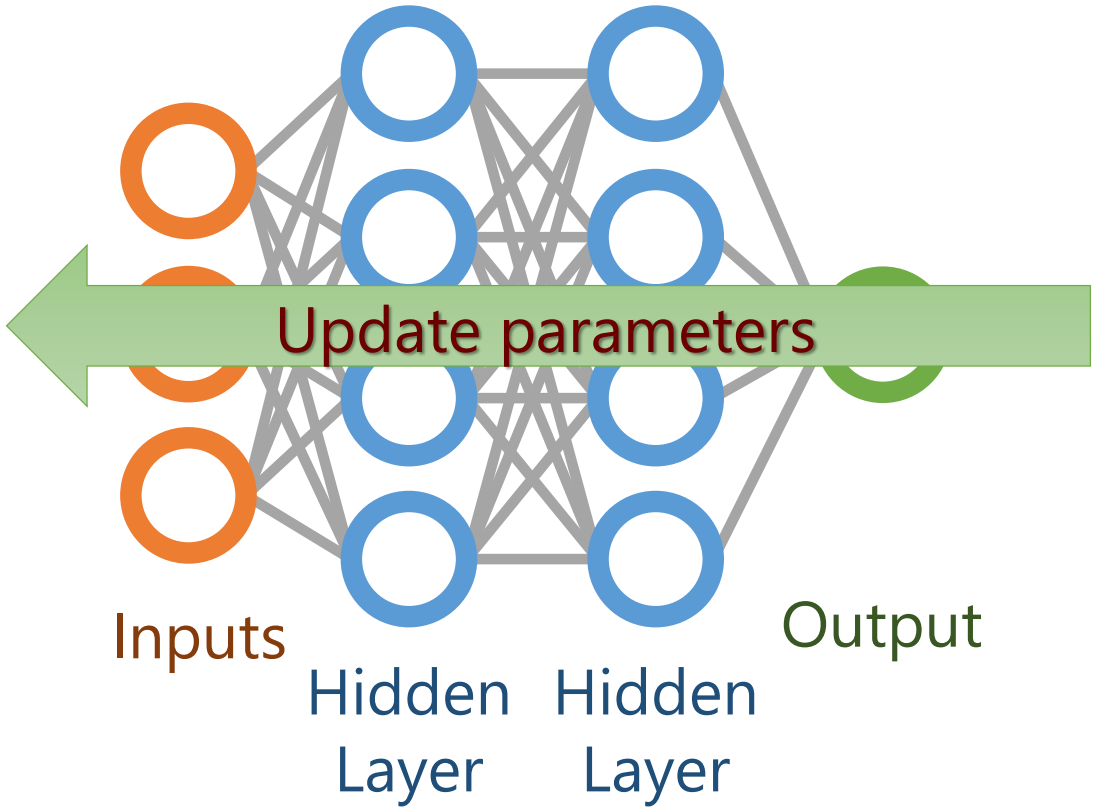
A simple neural network

- Backward pass

$x_0 = 22.5$

$x_1 = 24.7$

$x_2 = 25.3$



$y' = -0.3$ Prediction

$y = 26.9$ Ground Truth

ℓ_2 error:
$(-0.3 - 26.9)^2$

Minimize: $\mathrm{L}(\mathrm{x}, \mathrm{y}; \mathrm{W}, \mathrm{b}) = \sum_{i=1}^{N} \left((W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3) - y_i \right)^2$

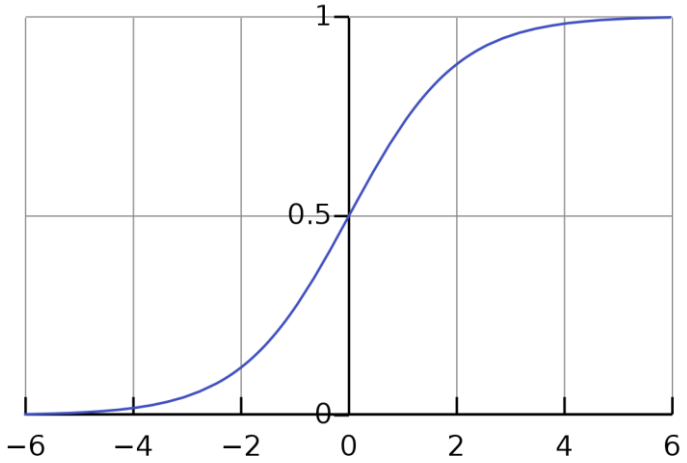
Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

A simple neural network

- Backward pass

Minimize: $\mathrm{L}(\mathrm{x}, \mathrm{y}; \mathrm{W}, \mathrm{b}) = \sum_{i=1}^{N} \left((W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3) - y_i \right)^2$
Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

Non-convex Optimization



Non-linear activation function

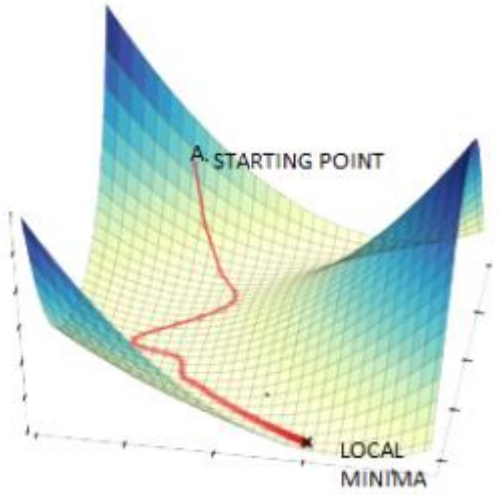
A simple neural network

- Backward pass

Minimize: $\mathrm{L}(\mathrm{x}, \mathrm{y}; \mathrm{W}, \mathrm{b}) = \sum_{i=1}^{N} \left((W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3) - y_i \right)^2$
Given N training pairs: $\{x_i, y_i\}_{i=1}^{N}$

Non-convex Optimization

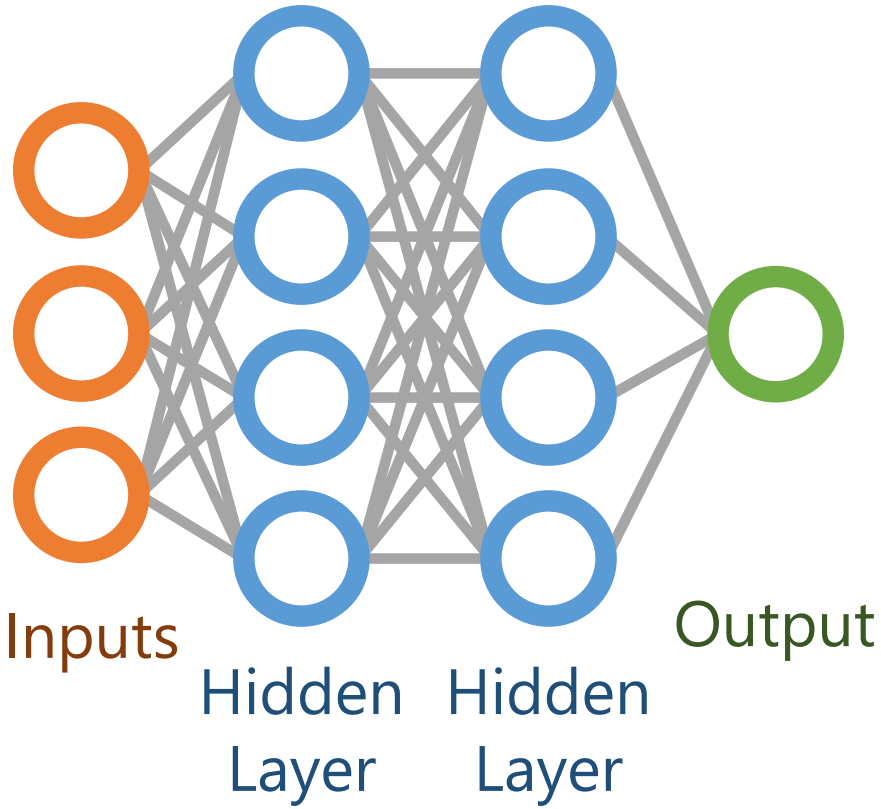
Gradient Descent



$$W = w - \eta \frac{\partial L}{\partial W}$$

A simple neural network

- Model:
  - Multi-Layer Perceptron (MLP)
  - $y' = W_3 f(W_2 f(W_1 x + b_1) + b_2) + b_3$
- Loss function:
  - ℓ_2 loss: $L(y, y') = (y - y')^2$
- Optimization:
  - Gradient Descent
  - $W = w - \eta \frac{\partial L}{\partial W}$



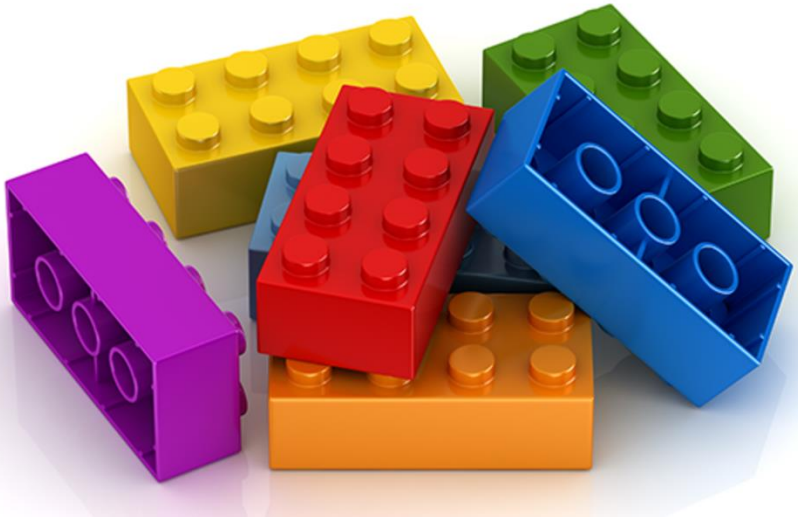
Deep Network Architectures

What people think I do

What I actually do

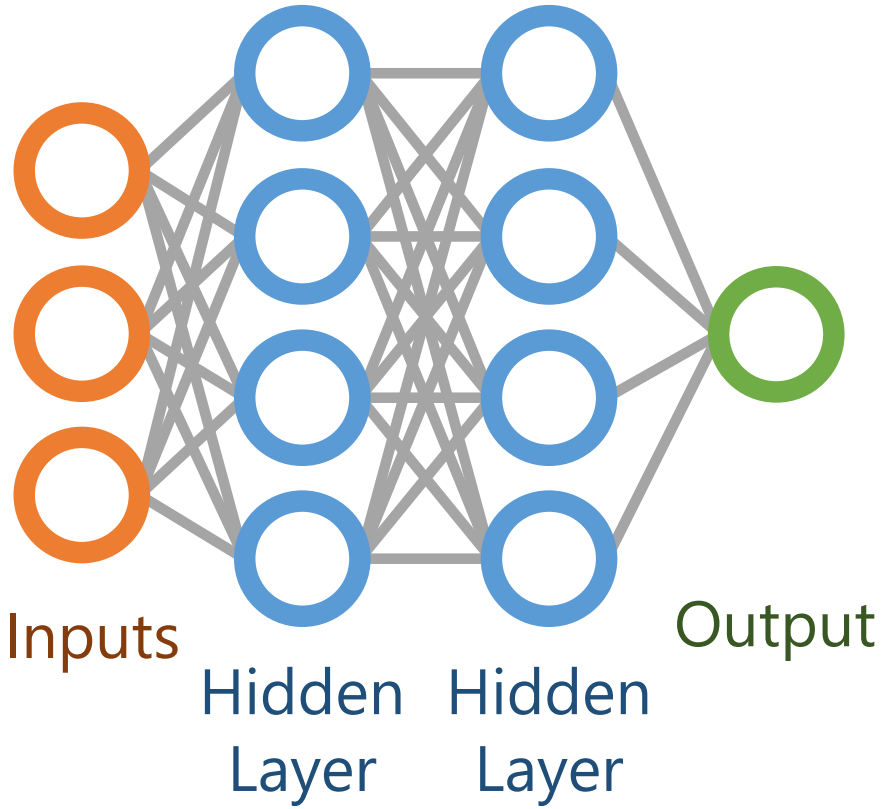
Deep Network Architectures

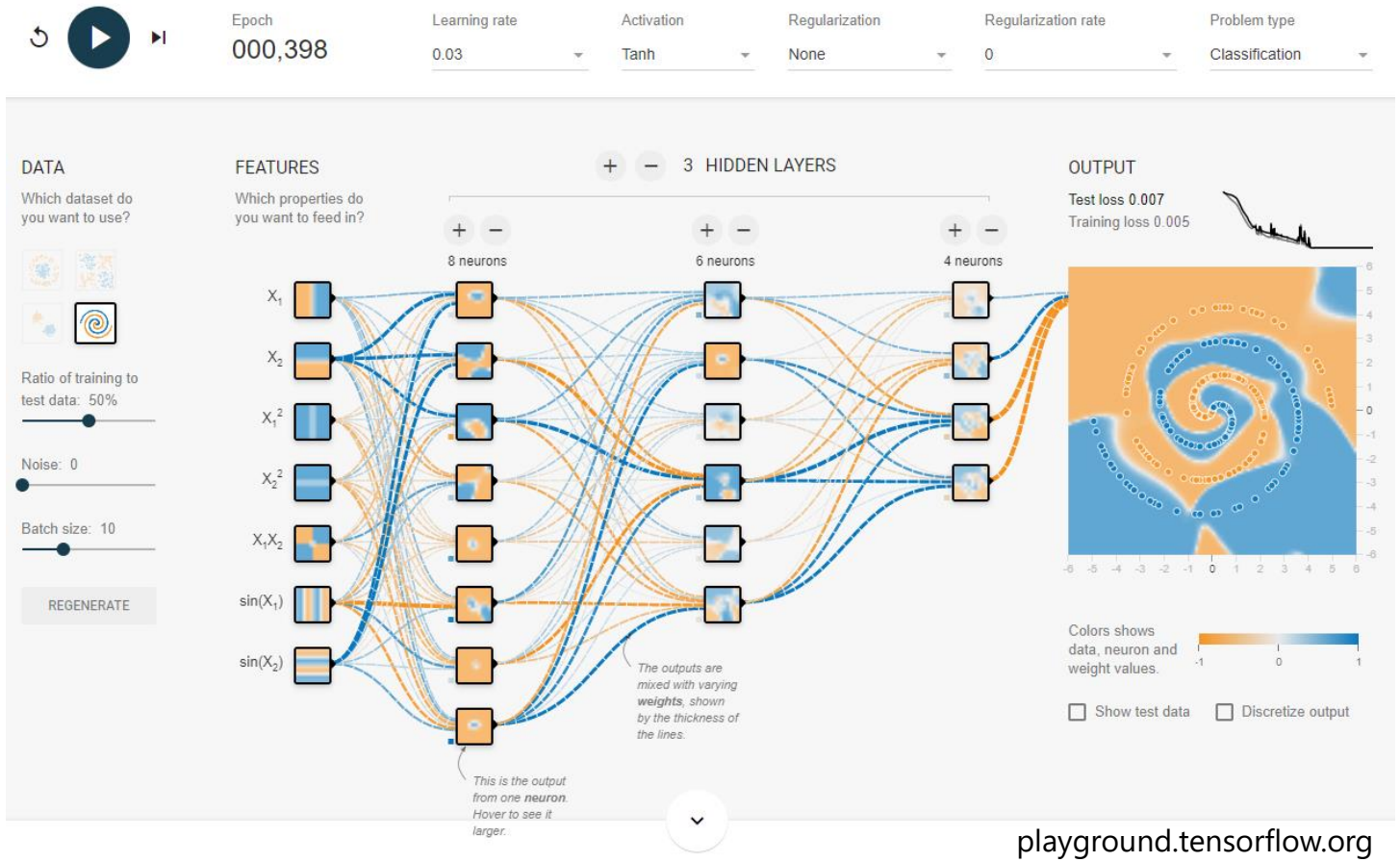
- Building Blocks
  - Fully connected layers
  - ReLUs
  - Convolutions, Tranpsosed Convolutions
  - Pooling
  - Dilated Convolutions
- Classic Architectures
  - MLP
  - AlexNet
  - VGG
  - ResNet
  - …



Multi-Layer Perceptron

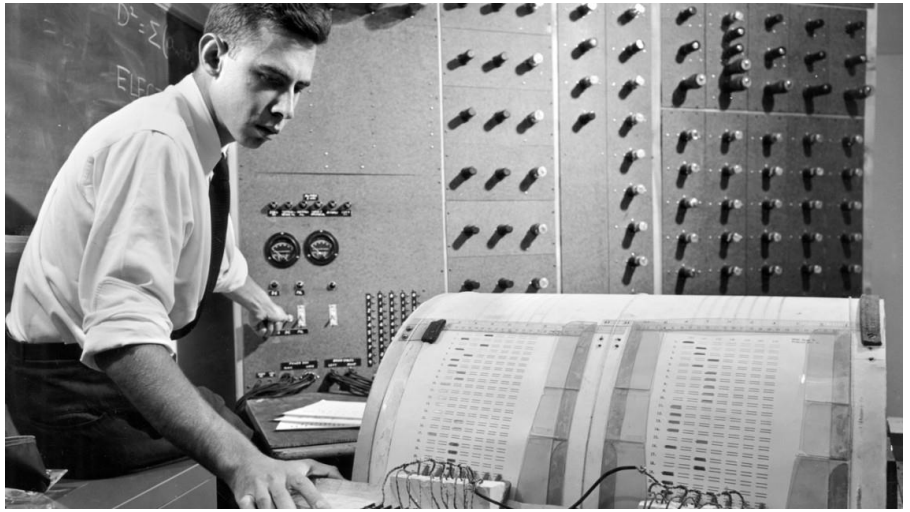
- Fully connected layers + non-linear activation





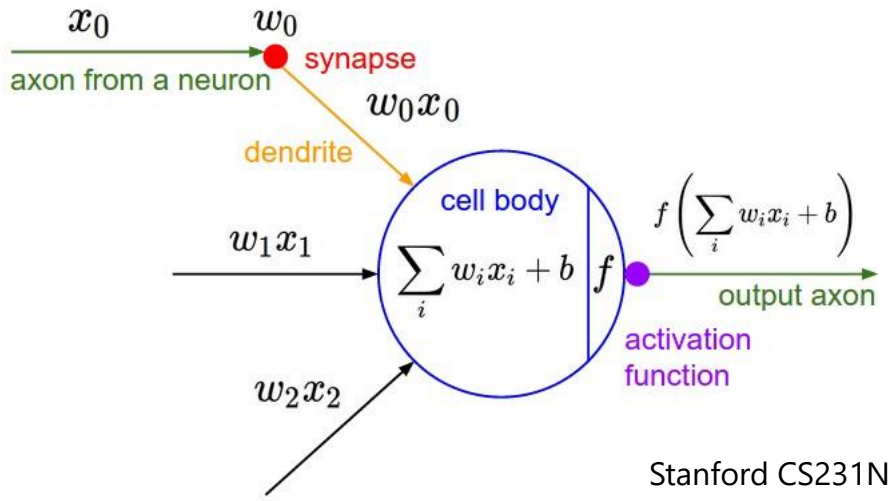
playground.tensorflow.org

Fully-Connected Layer



- 1960: Perceptron, Cornell University
- Binary linear classifier on top of a simple feature extractor

$$y = sign\left(\sum_{i=1}^{N} W_i F_i(x) + b\right)$$



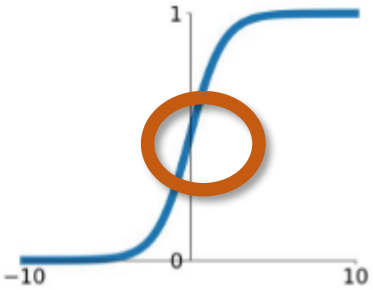
Stanford CS231N

Non-linear Activation

- Early common activations
- Zero-centered?

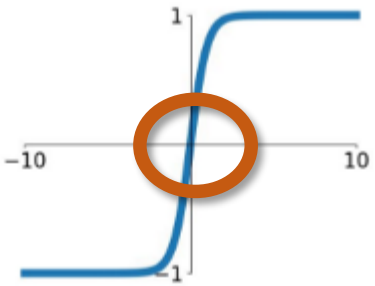
Sigmoid

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



tanh

$$\tanh(x)$$

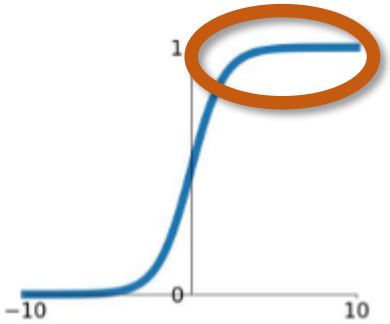


Non-linear Activation

- Early common activations
- Zero-centered?
- Vanishing gradients?

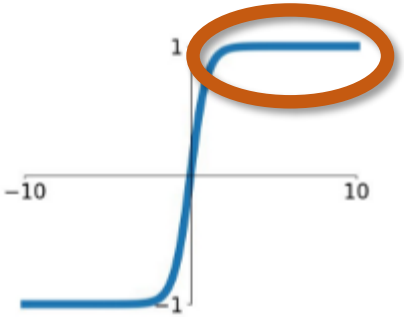
Sigmoid

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



tanh

$$\tanh(x)$$

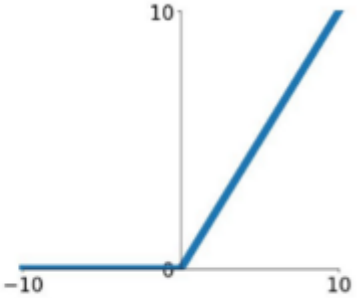


Non-linear Activation

- Most common activation

ReLU

$$\max(0, x)$$



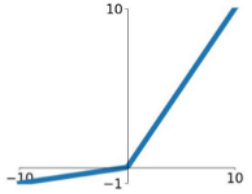
- Cheap operation
- No gradient saturation, faster convergence

- Potential for “dead” neurons

- Related: Leaky ReLU, MaxOut

Leaky ReLU

$$\max(0.1x, x)$$

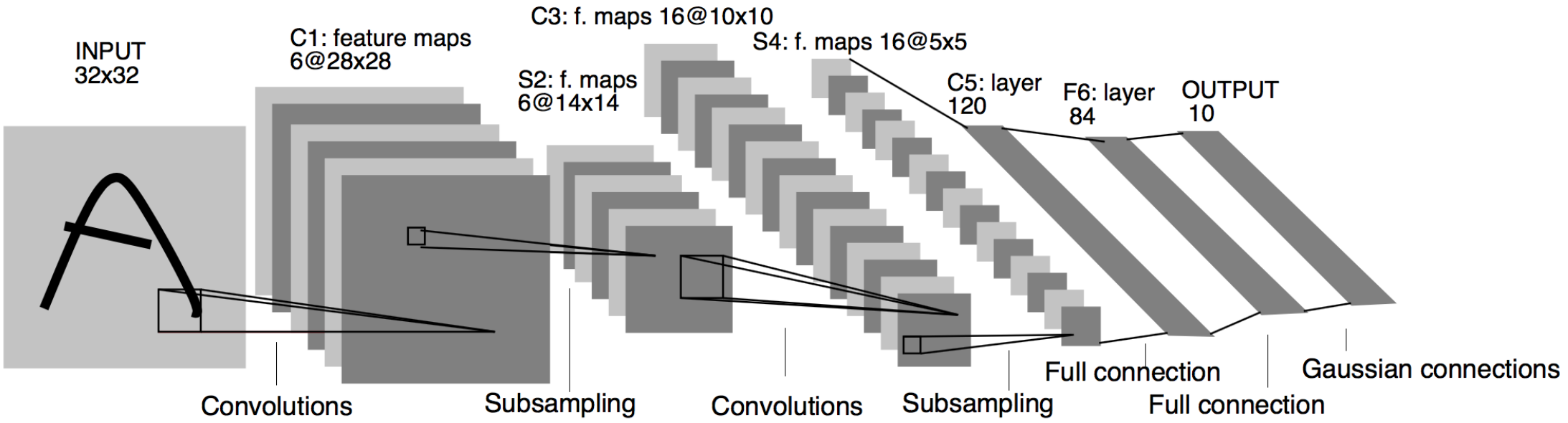


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

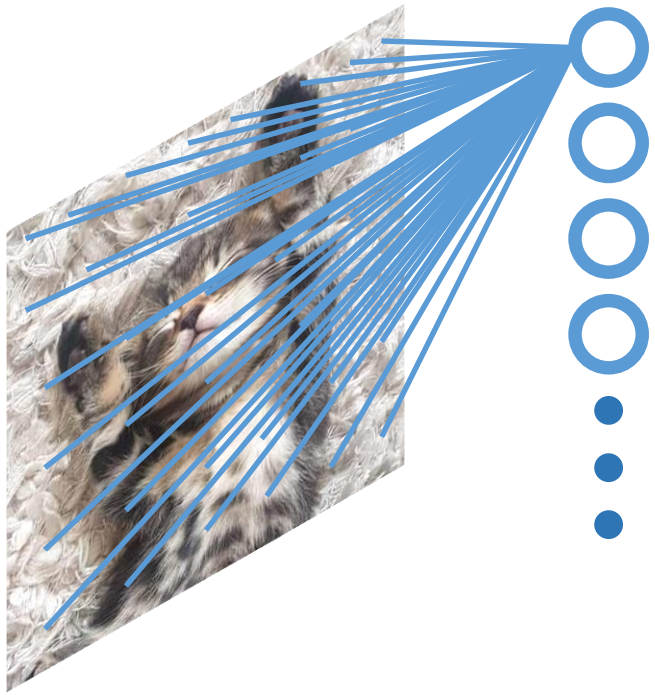
LeNet (Convolutional Neural Network)

- LeCun et al. 1998
- Classify hand-written digits on bank checks
- One of first successful applications of CNN



Convolutions

- Fully-connected layers applied to images?
- For a $200 \times 200 \times 3$ image:



Stretch $200 \times 200 \times 3$ to $1 \times 120{,}000$

Apply Wx with $120{,}000 \times 1000$ weights

Convolutions

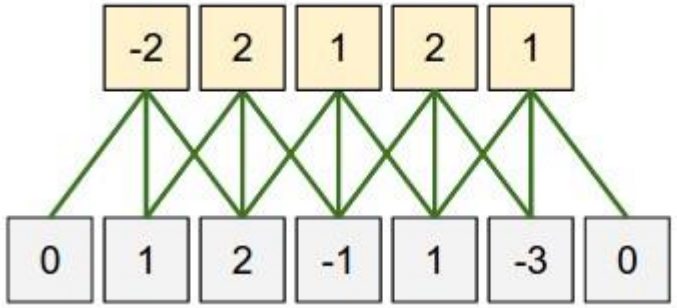
- Locally connect and preserve spatial structure
- For a $200 \times 200 \times 3$ image:

Local $5 \times 5 \times 3$ filter

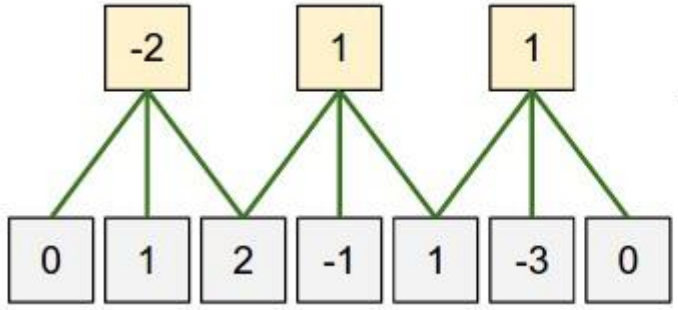
Convolutions

- (pytorch) `torch.nn.Conv3d(in_channels, out_channels, kernel_size, stride, padding, bias)`
  - `in_channels`: determined by input
  - `out_channels`: hyperparameter determining number of filters
  - `kernel_size`: hyperparameter determining spatial size of filters
  - `stride`: hyperparameter determining how to slide the filters
  - `padding`: hyperparameter determining amount of zero-padding on boundaries
  - `bias`: whether to apply a bias

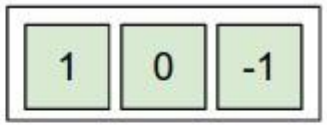
Convolutions



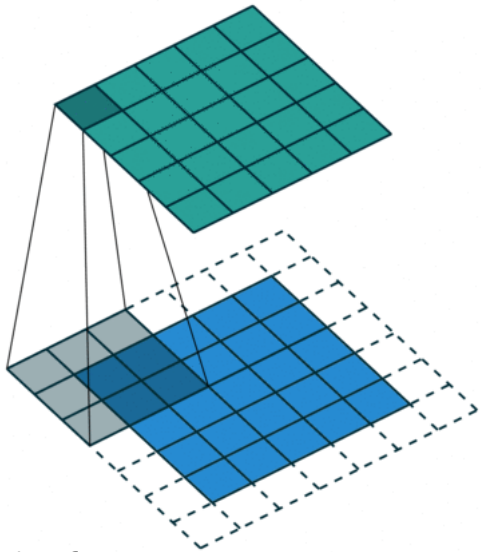
Stride = 1



Stride = 2

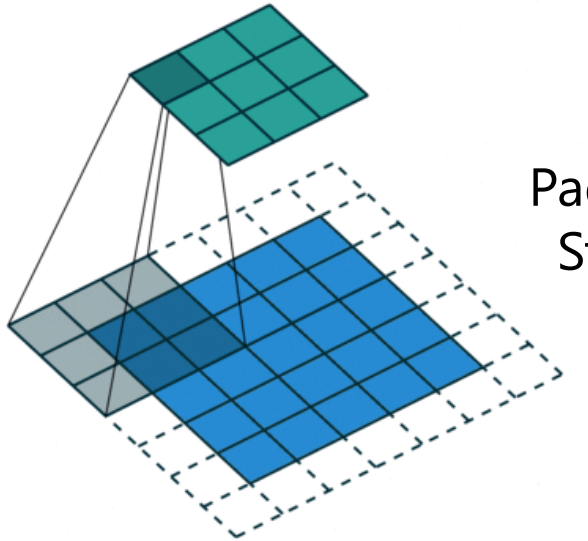


Stanford CS231n



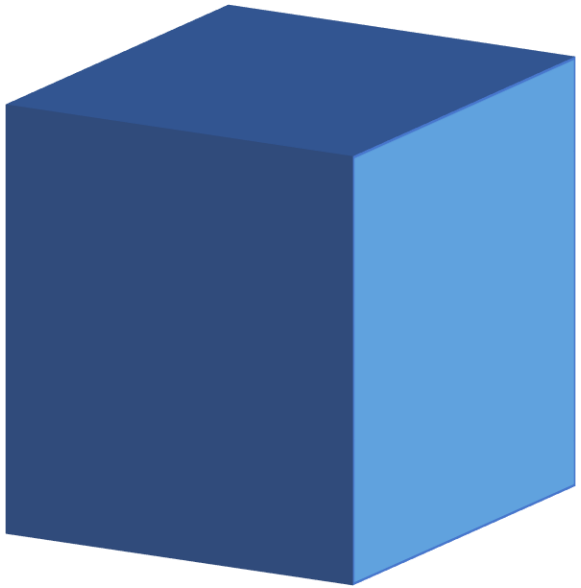
Padding = 1
Stride = 1

github.com/vdumoulin/conv_arithmetic

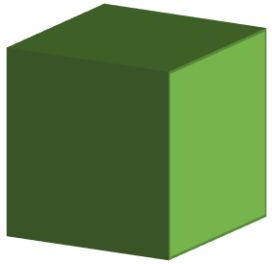


Padding = 1
Stride = 2

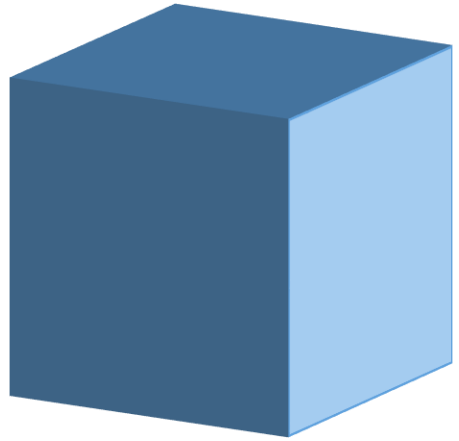
Example 3D Convolution



Input
32^3 volume
$1 \times 32 \times 32 \times 32$ array



4^3 kernel, 2 output channels,
padding 1, stride 2
weight: $2 \times 1 \times 4 \times 4 \times 4$
bias: 2×1

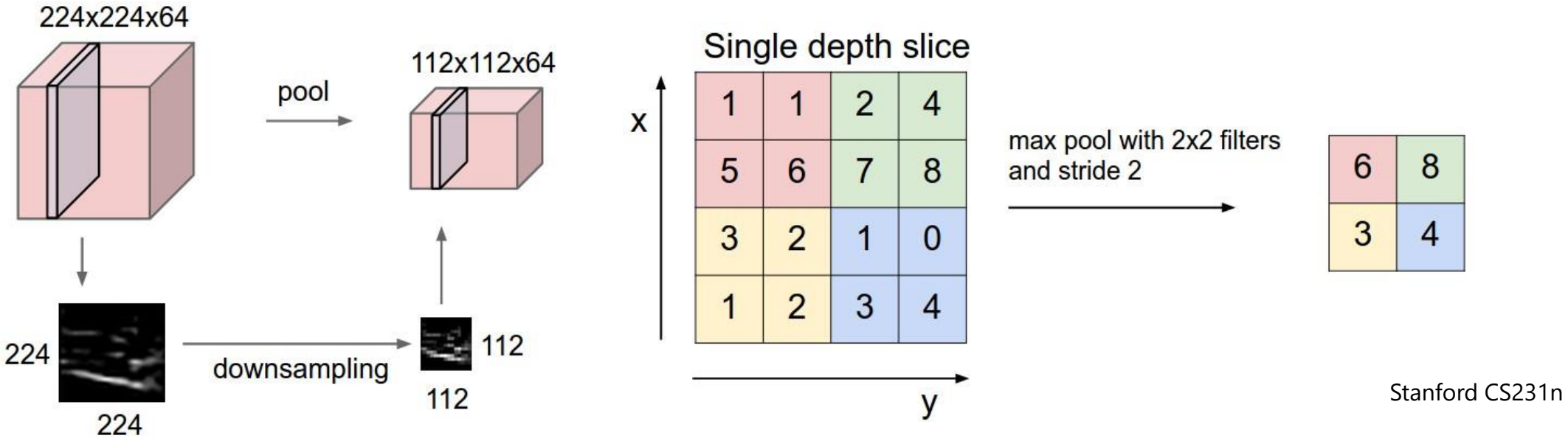


Output
$2 \times 16 \times 16 \times 16$ array

$$d' = \left\lfloor \frac{d + 2p - k}{s} \right\rfloor + 1$$

Pooling

- Reduces spatial size without introducing parameters
- In-between convolution layers
- Fixed function (typically $\max$, sometimes avg)

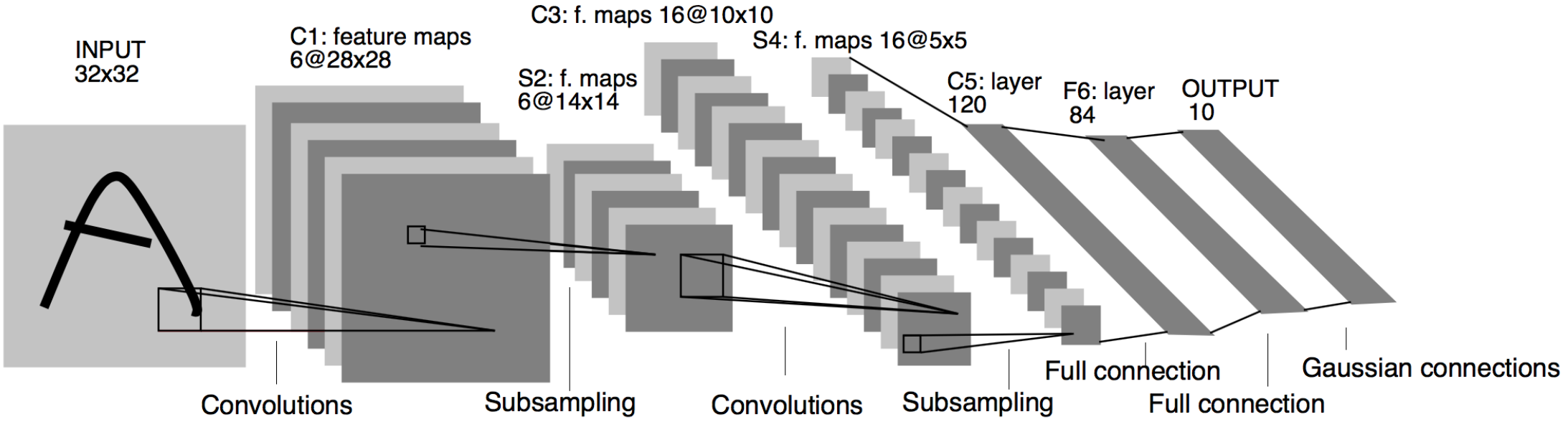


Stanford CS231n

Recently common: removing pooling layers, especially for generative models

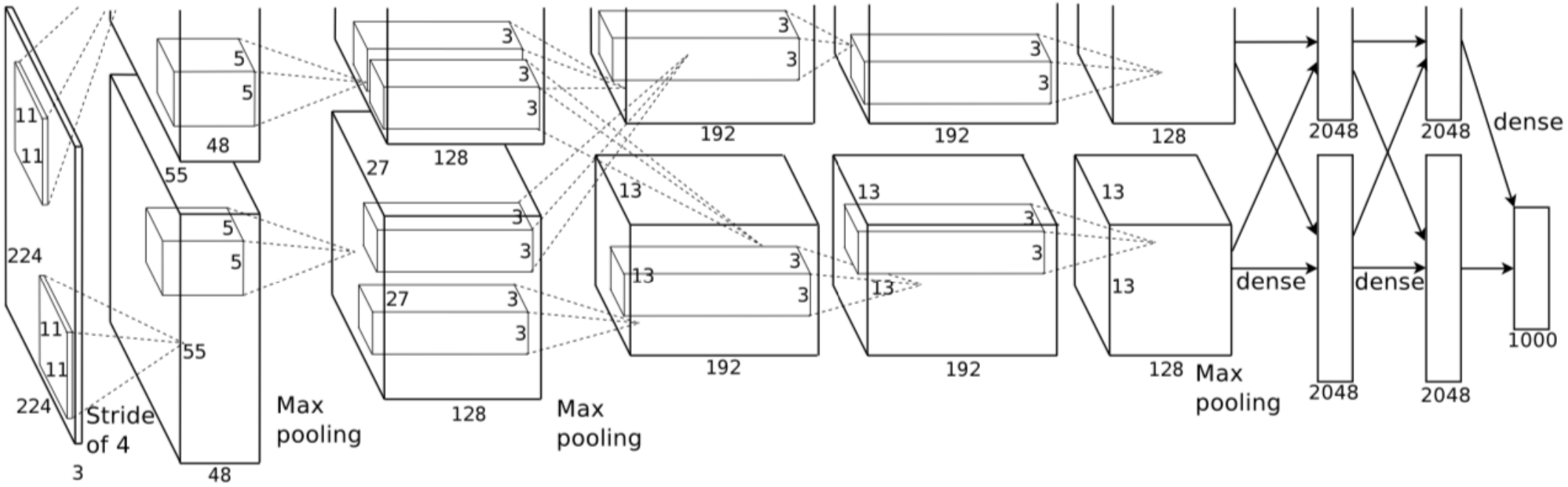
LeNet (Convolutional Neural Network)

- Convolutions + Activations + Pooling + Fully-Connected



AlexNet

- Krizhevsky et al. 2012
- Popularized convolutional networks in computer vision



AlexNet

- Krizhevsky et al. 2012
- Popularized convolutional networks in computer vision

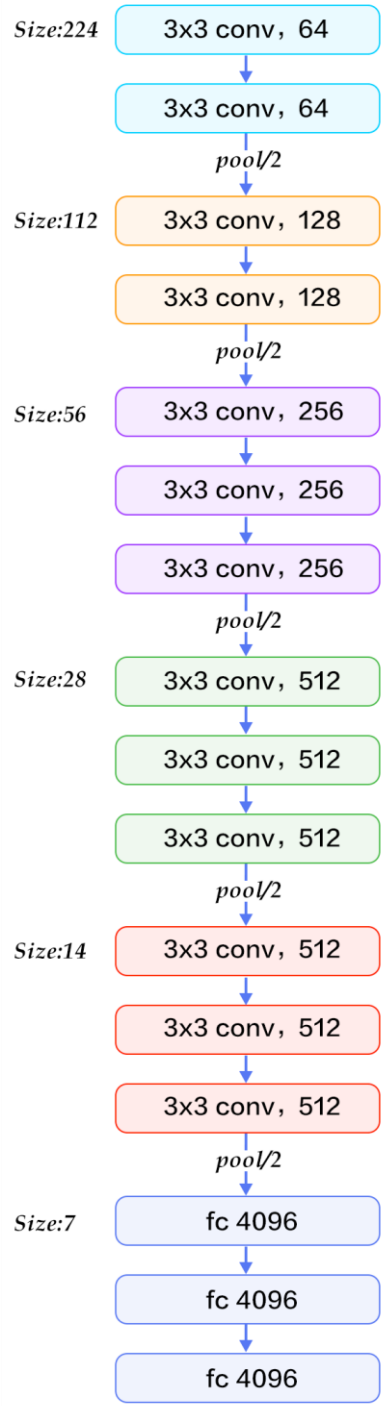
➢Big data: ImageNet

➢GPU implementation (more than 10x speedup)
  ➢Train a deeper network

➢Algorithmic improvements: data augmentation, ReLU, dropout, normalization layers, etc.

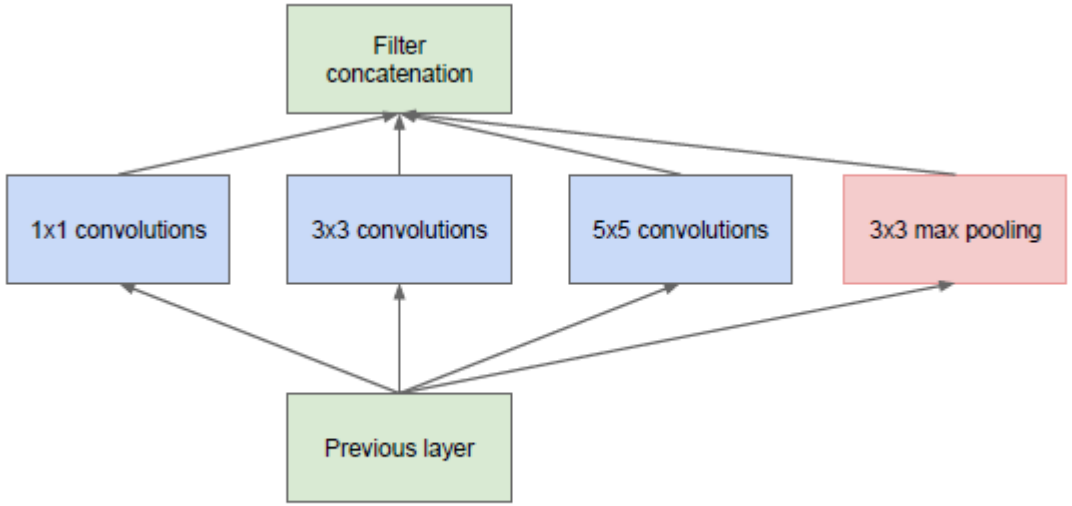
VGG

- Simonyan and Zisserman 2014
- Showed that depth of the network is crucial for performance
- VGG-16: final best network with 16 conv/fc layers, very homogeneous architecture with only 3×3 convs and 2×2 pooling



GoogleNet (Inception v1)

- Inception Module

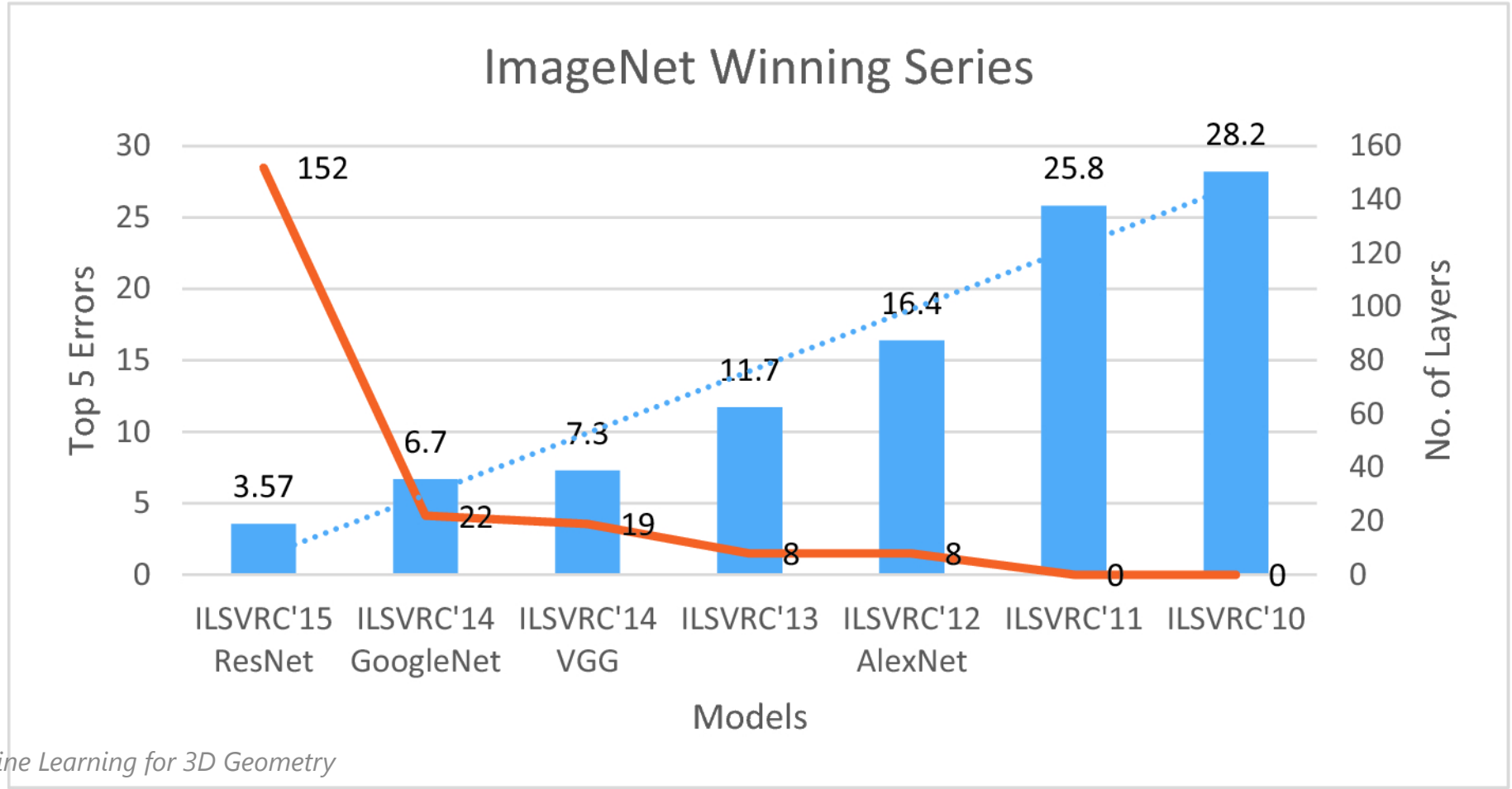


(a) Inception module, naïve version

- Using 1×1 convolutions as dimension reduction
- Different conv sizes extract different features
- Average pooling instead of fully-connected

ResNet

- He et al. 2016

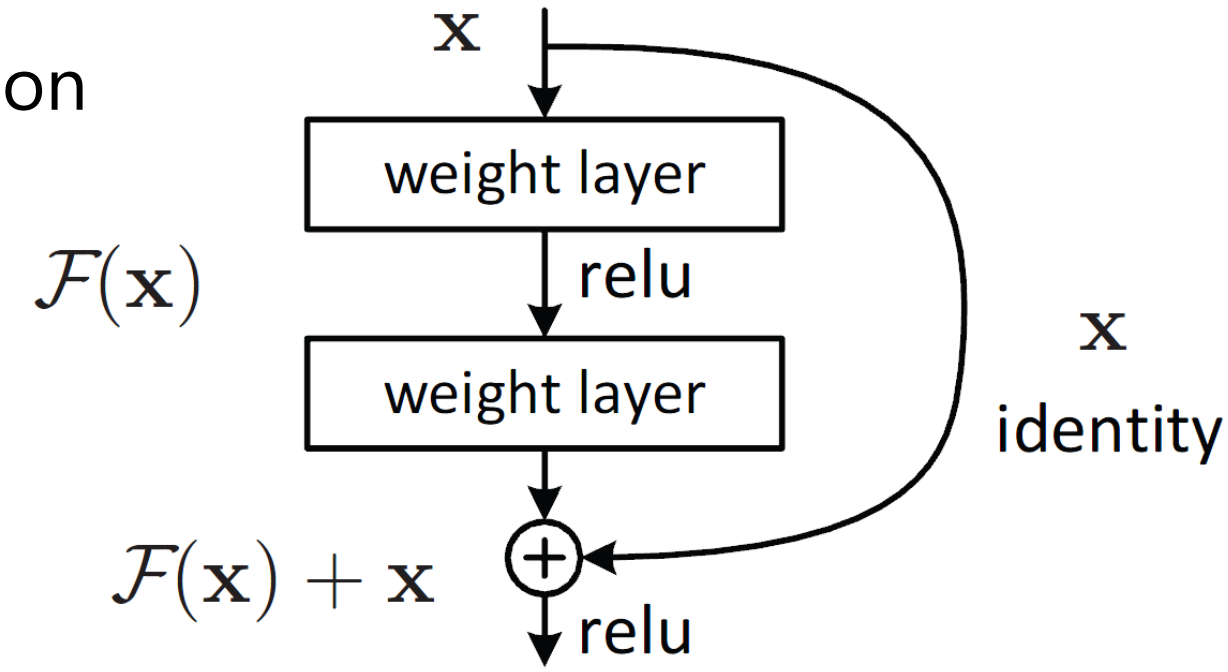


ResNet

- How to train deeper networks?
- Use skip connections for residual learning

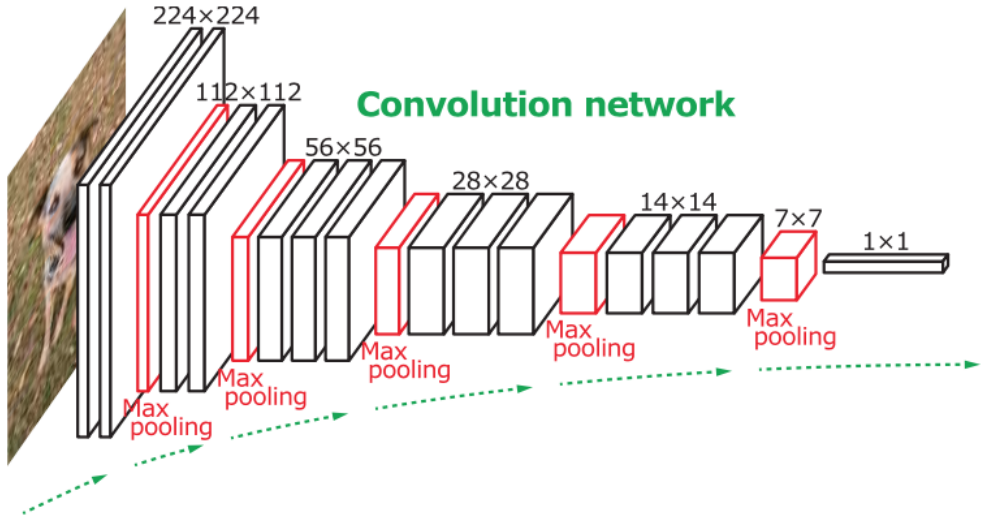
- Heavy use of batch normalization

- No fully connected layers



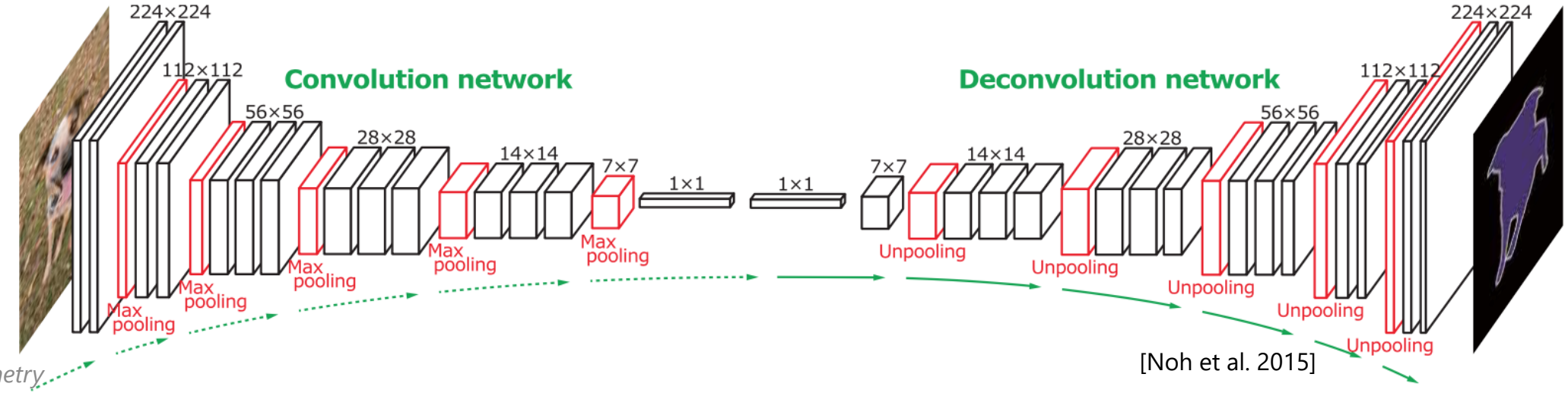
Transposed Convolutions

Classification



Dog

Segmentation

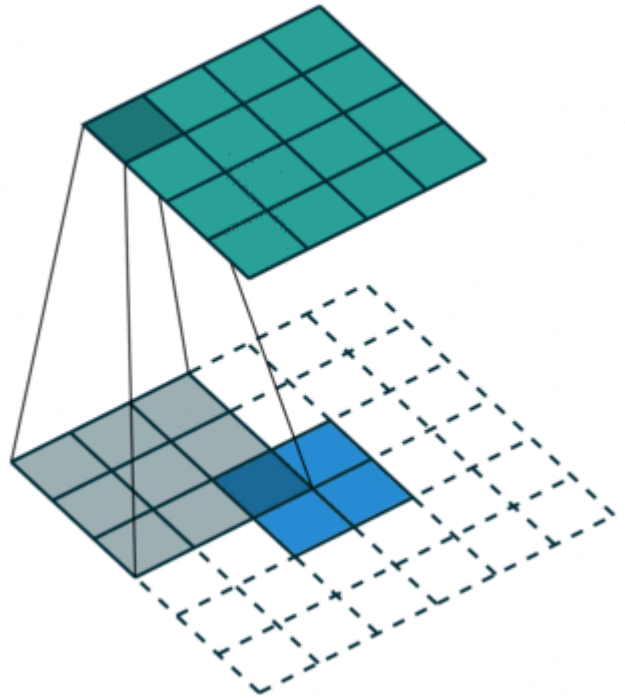


[Noh et al. 2015]

Transposed Convolutions

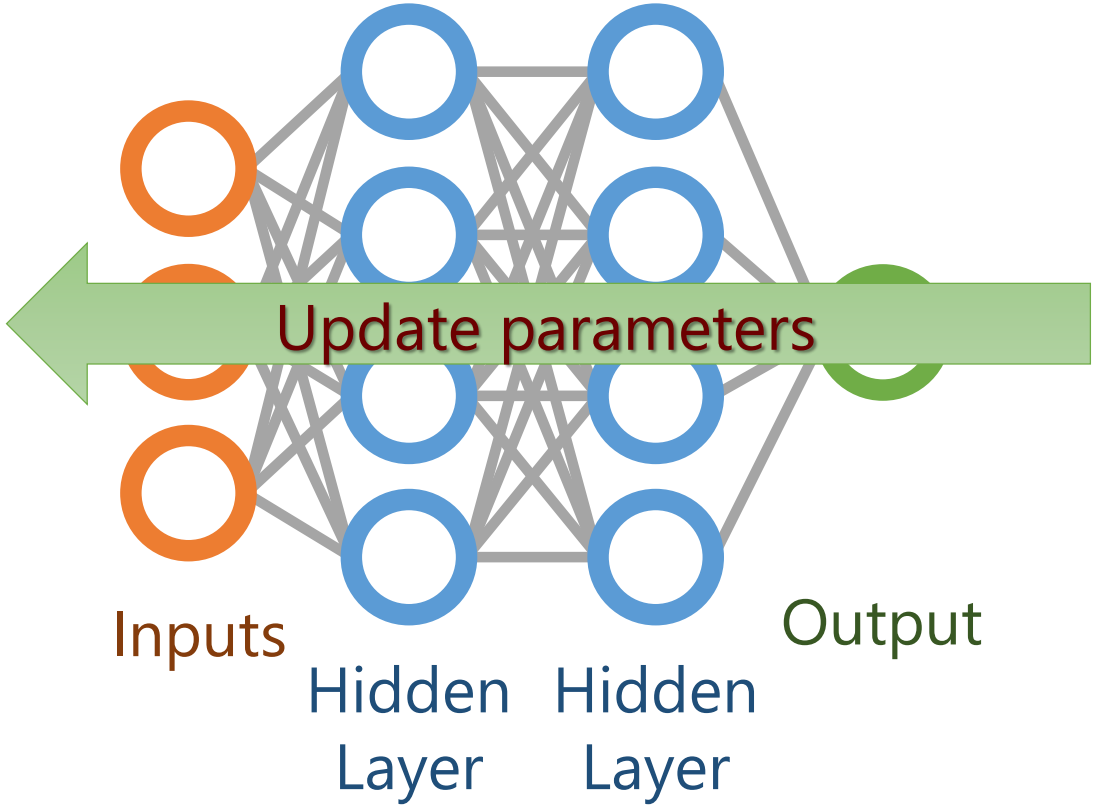
- “upsample” in spatial resolution

- Learned upsampling vs traditional
  - Traditional upsampling:
    - nearest neighbor,
    - Bilinear/bicubic interpolation



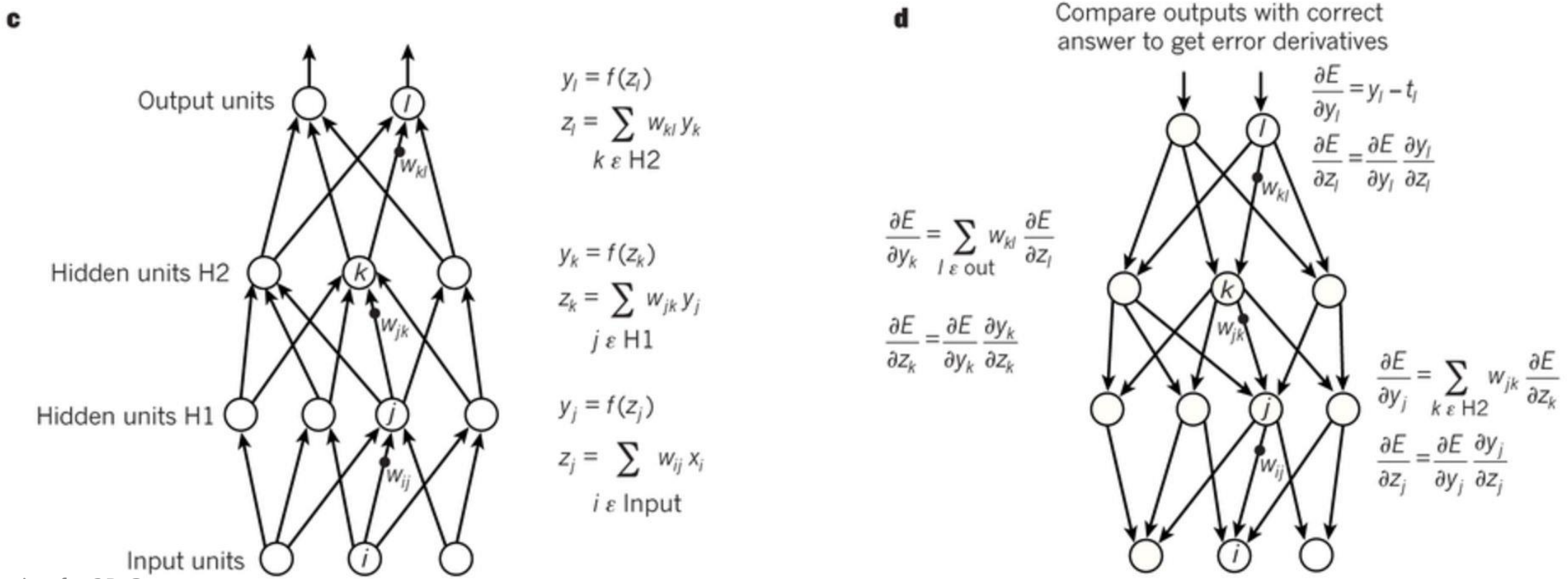
github.com/vdumoulin/conv_arithmetic

Optimizing Deep Networks



Backpropagation

- Compute gradients for weights with respect to loss
- Gradient descent with chain rule



[LeCun, Bengio and Hinton 2015]

Stochastic Gradient Descent

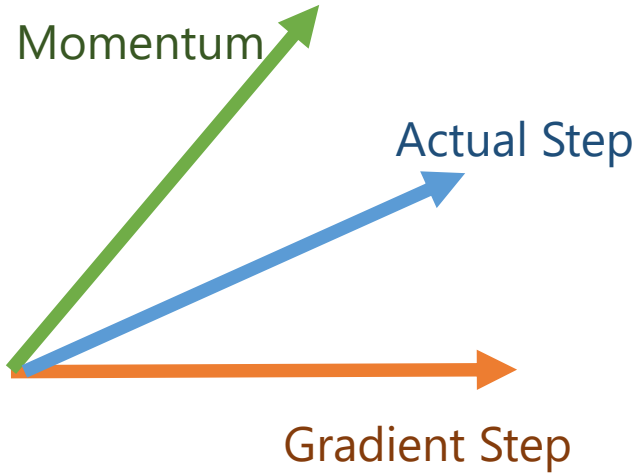
- Gradient Descent
  - Update weights once after seeing all training data
- Stochastic Gradient Descent
  - Update weights after looking at a subset sampled from the training data
  - Subset -> mini-batch
- Vanilla update: `x += -lr * dx`

SGD Variants

- SGD Vanilla update: `x += -lr * dx`

- Momentum

```
v = mu * v - lr * dx # integrate velocity
x += v # integrate position
```



- Build velocity in any direction that has consistent gradients

- Often better convergence than vanilla SGD

SGD Variants

- Adagrad

```
cache += dx**2
x += -lr * dx / (np.sqrt(cache) + eps)
```

  - Weights with high gradients -> effective lr reduced
- RMSProp

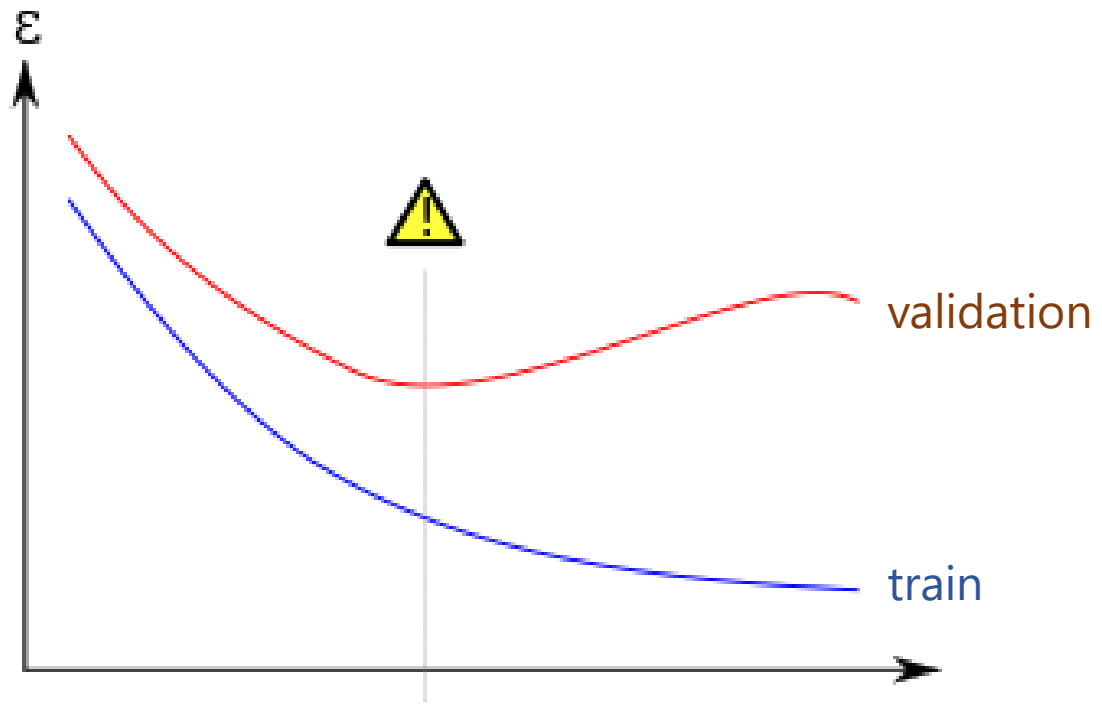
```
cache = decay_rate * cache + (1 – decay_rate) * dx**2
x += -lr * dx / (np.sqrt(cache) + eps)
```

  - Moving average to reduce Adagrad's aggressive, monotonically decreasing lr
- Adam

```
m = beta*m + (1-beta1) * dx
v = beta2*v + (1-beta2) * (dx**2)
x += -lr * m / (np.sqrt(v) + eps)
```

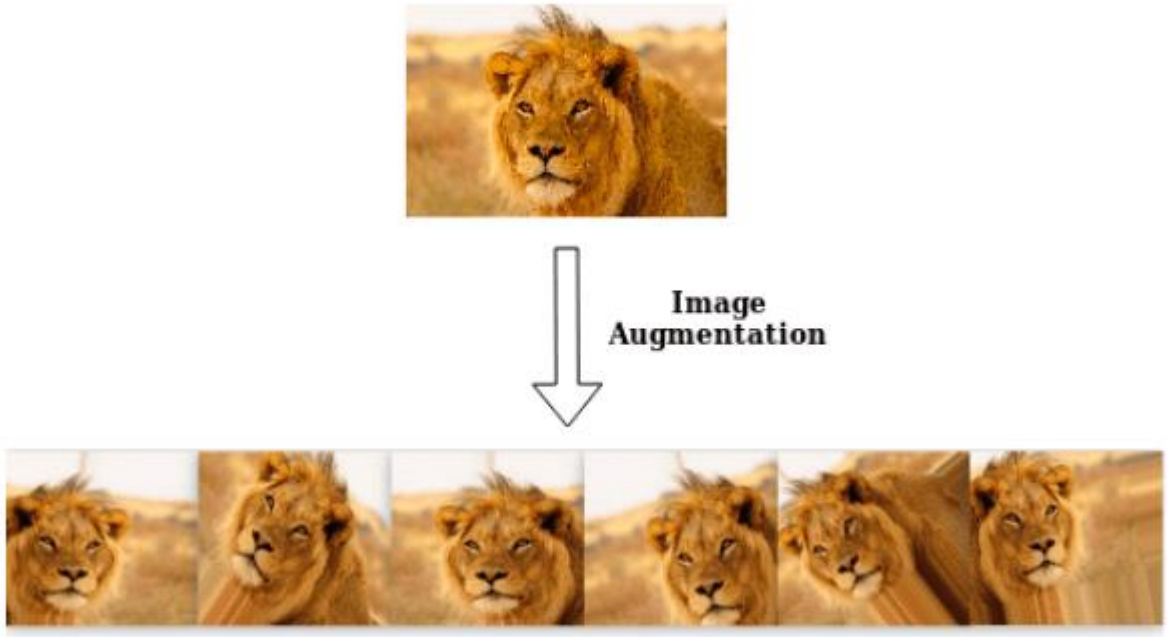
  - Use smoothed version of gradients

Overfitting



Combat Overfitting: Data Augmentation

- 2D: Flipping, random crops, color/brightness change, adding noise, …
- 3D: Rotations, random translational jitter, …

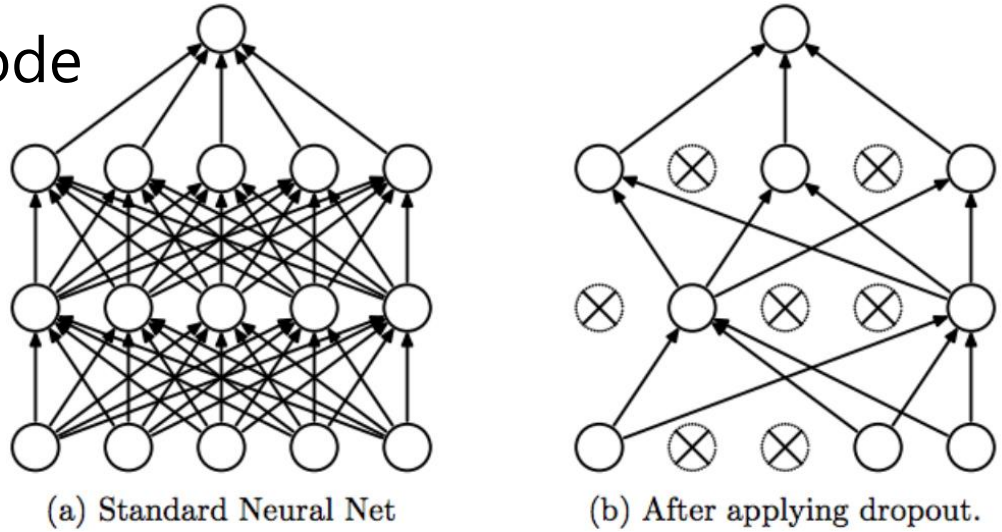


towardsdatascience.com/machinex-image-data-augmentation-using-keras-b459ef87cd22

Combat Overfitting

- Regularization on weights
  - ℓ_2 penalty on weights
  - Limit network capacity by encouraging distributed weights

- Dropout
  - Network cannot rely on any sole input node
  - Weights spread across features
  - Not applied at test time



(a) Standard Neural Net

(b) After applying dropout.

Weight initialization

- Random Gaussian initialization
  - Distribution of outputs has a variance that grows with the number of inputs
  - ➢Exploding/diminishing output in very deep network

- Calibrate variances with $\frac{1}{\sqrt{n}}$

```
w = np.random.randn(n) / sqrt(n)
```

- For ReLU

```
w = np.random.randn(n) / sqrt(2.0/n)
```

Batch Normalization

- Compute average and variance on mini-batch
- Center and re-scale logits
- Force Gaussian distribution
- More robust to bad initialization
- Add learnable scale and offset: $BN(x) = \alpha\hat{x} + \beta$

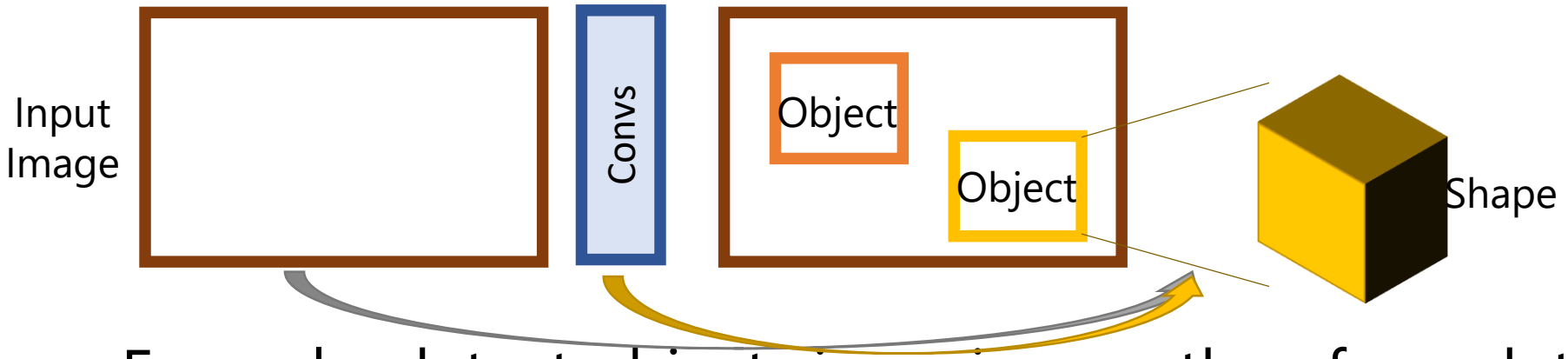
$$\hat{x} = \frac{x - avg_{batch}(x)}{stddev_{batch}(x) + \epsilon}$$

Useful architectures/paradigms

End-to-end Training

- Optimization and gradient propagation from final output result

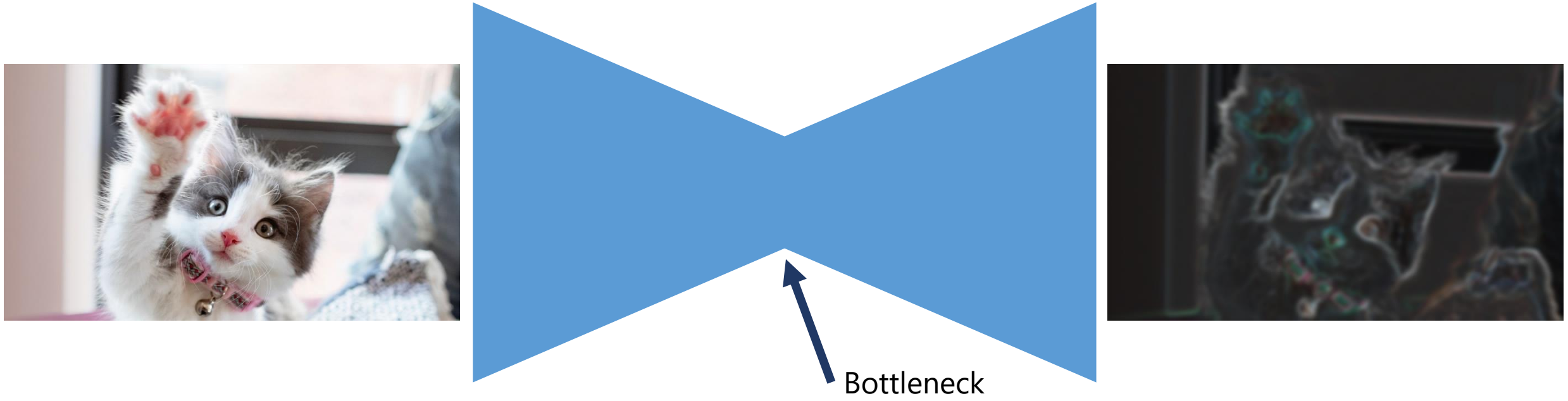
End-to-end Training



- Example: detect objects in an image, then from detected object boxes, predict their shape
- No end-to-end training: train object detector, train shape predictor based on ground truth or predicted object boxes, but no gradient information from shape to detection
- End-to-end: train simultaneously, gradient flow from shape predictor to detector (e.g., use shared feature encoder)

UNet

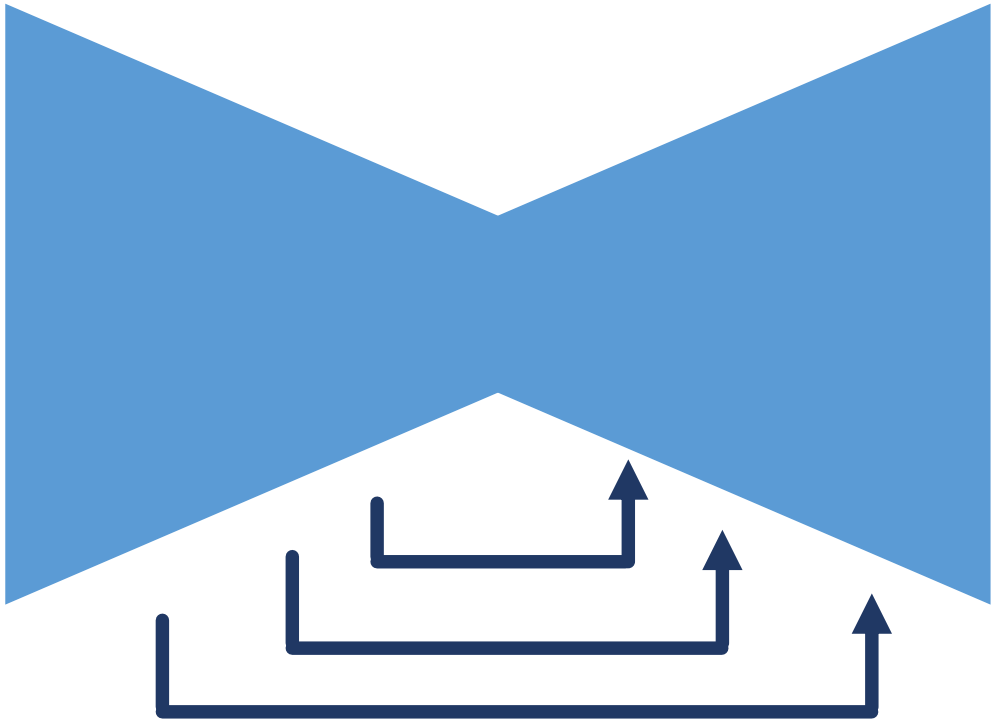
- Traditional encoder-decoder:

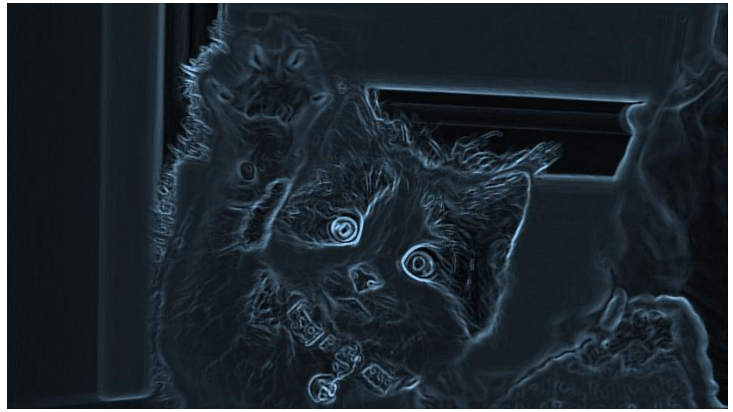


[Ronneberger et al. ‘15]

UNet

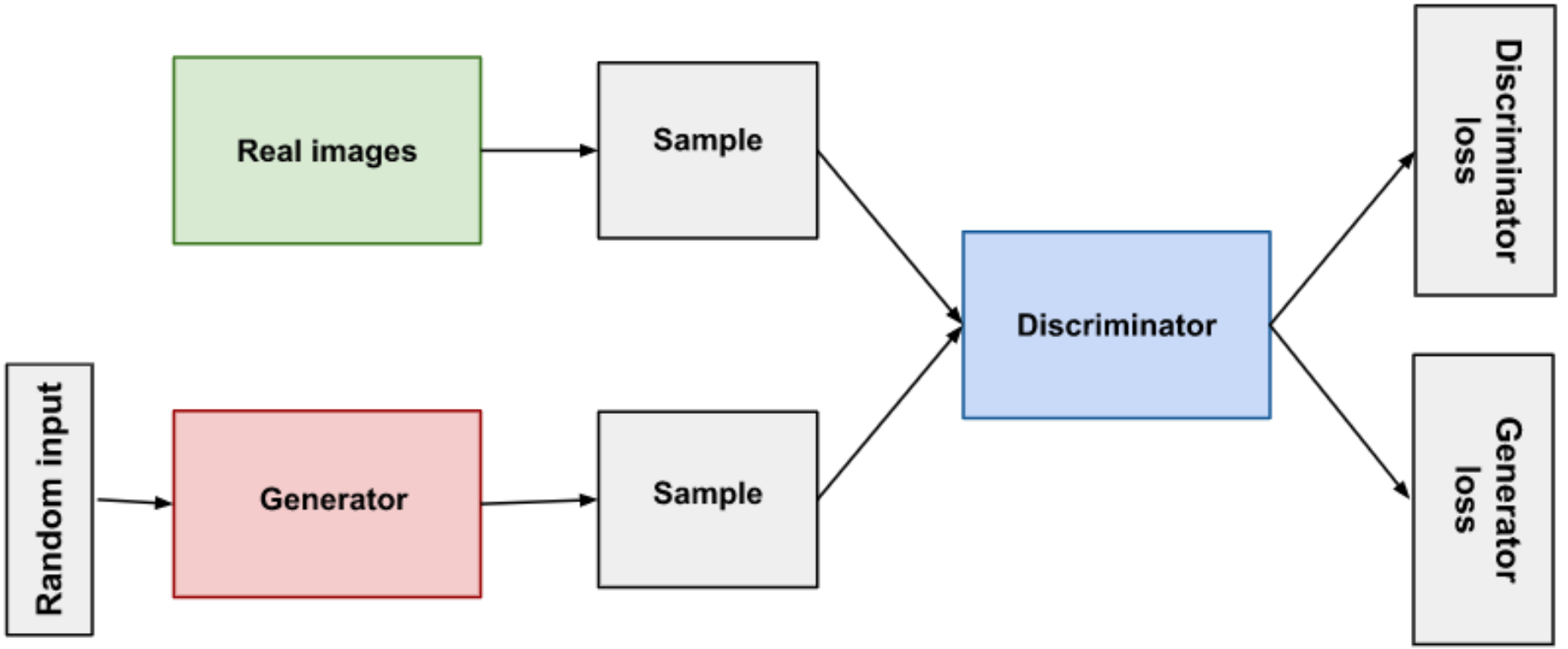
- Symmetric skip connections from encoder to decoder





[Ronneberger et al. ‘15]

Generative Adversarial Networks



[Goodfellow et al. '14]

In Practice

- Data preparation is important!
  - Collection, cleaning, pre-processing
- Deep learning frameworks
  - PyTorch, TensorFlow (Caffe, Theano, Matlab, …)
- Debugging: quantitative + qualitative!
  - Overfit first
  - Visualize

Resources

- TUM I2DL: https://niessner.github.io/I2DL/
- Stanford cs231n: https://cs231n.github.io
- Oxford Machine Learning: https://www.cs.ox.ac.uk/people/nando.defreitas/machinelearning/