Mastering Deep Learning Within a Few Hours

Stanley Zheng, CSE, CUHK

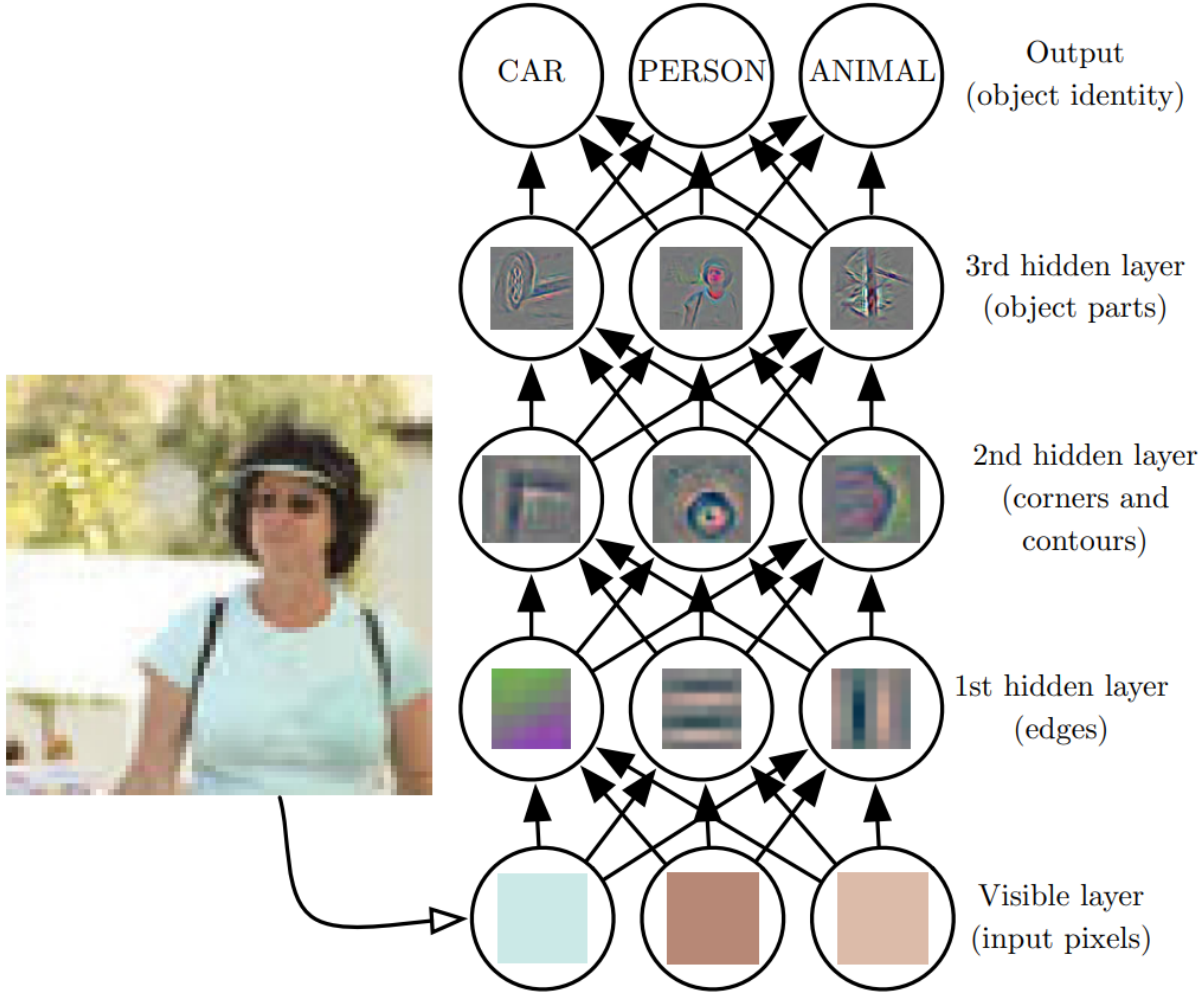
- Introduction
- Problem Formulation
- Implementation
- Experiments

Introduction

Artificial v.s. Intelligence

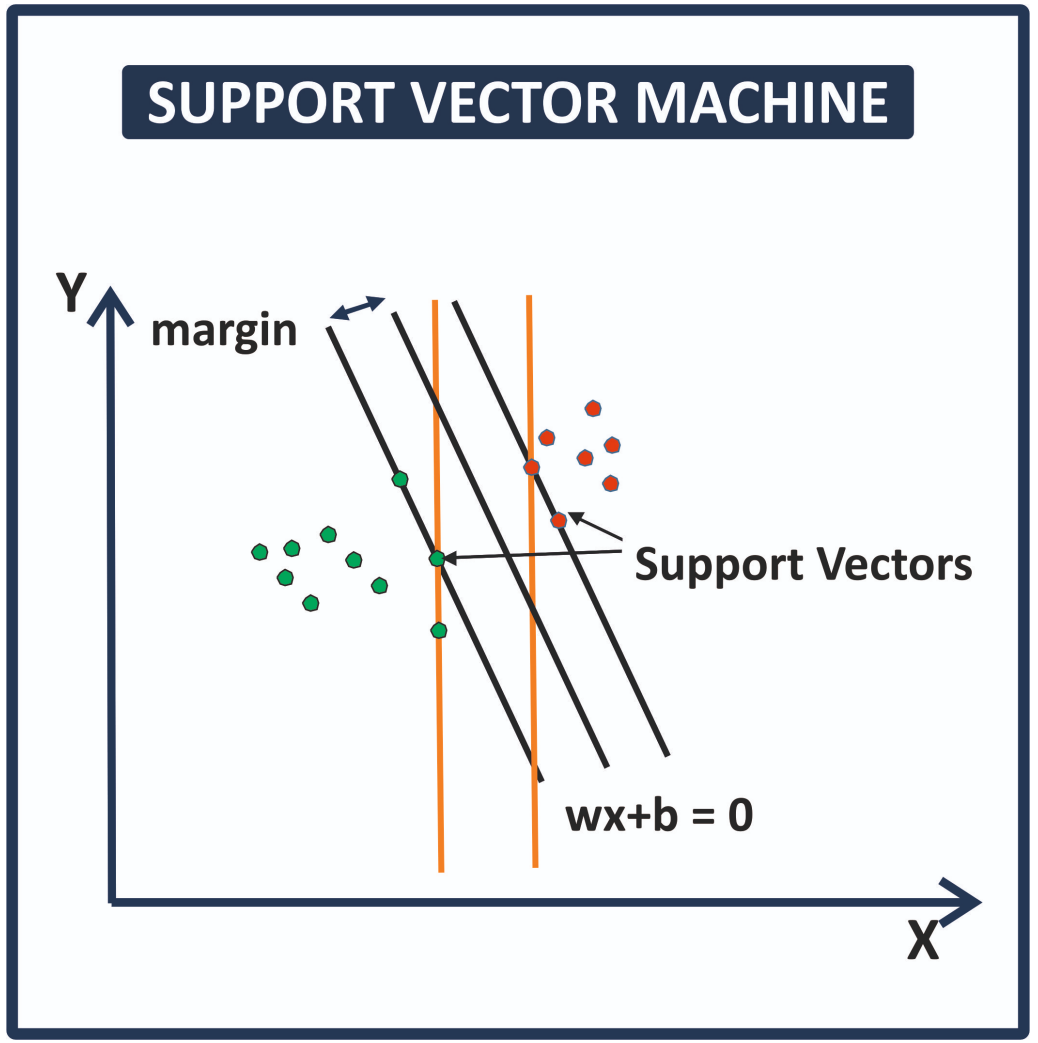
Intelligence

- Human Perception

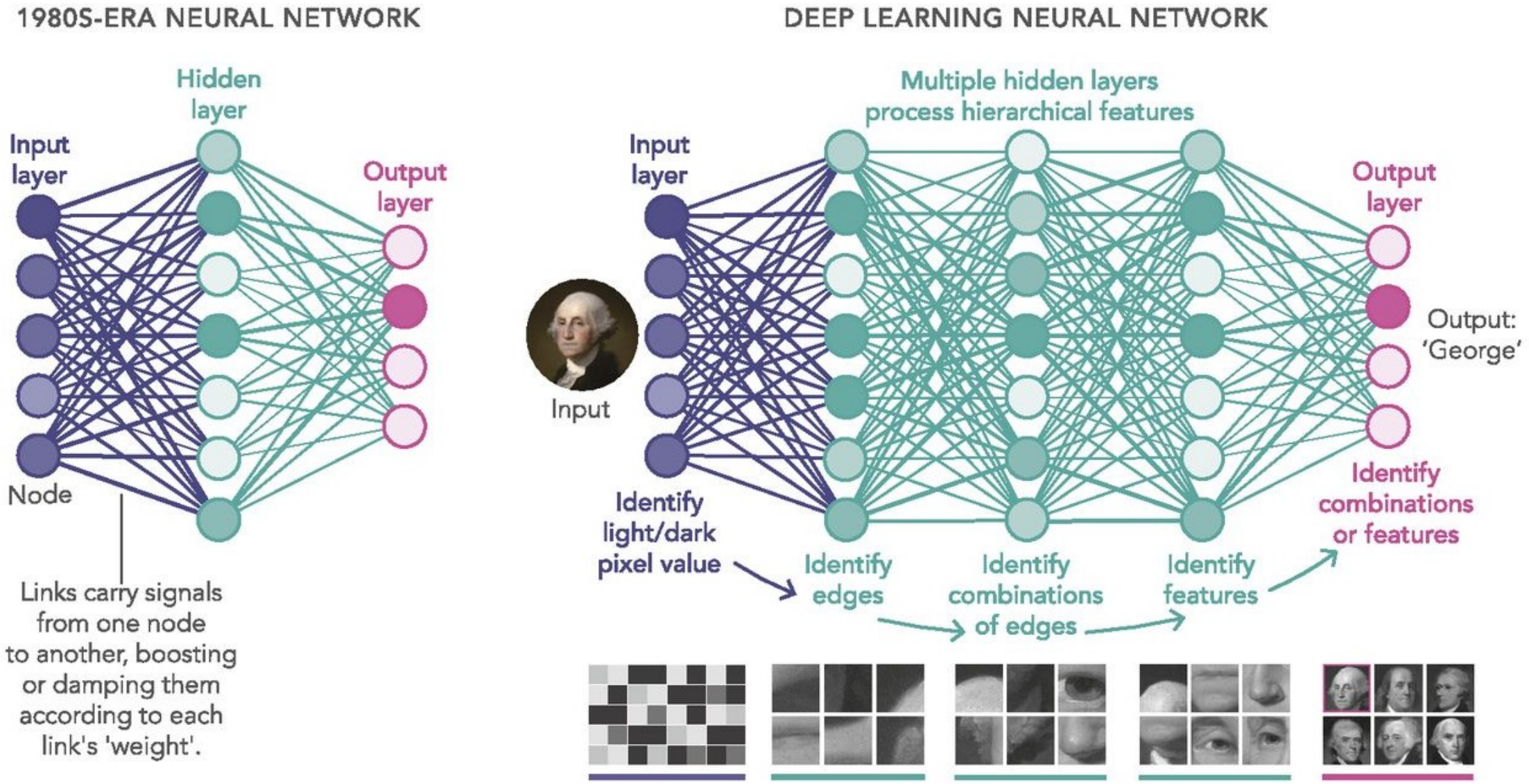


Artificial

- Machine Learning

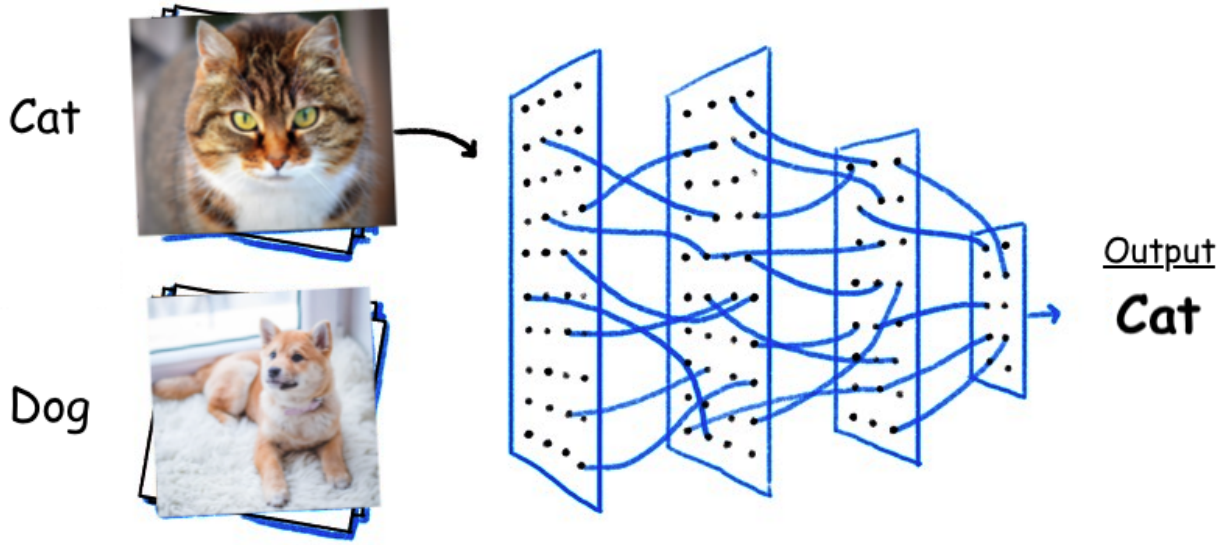


Deep Learning



What Can Deep Learning Do?

- Classification
- What is this?



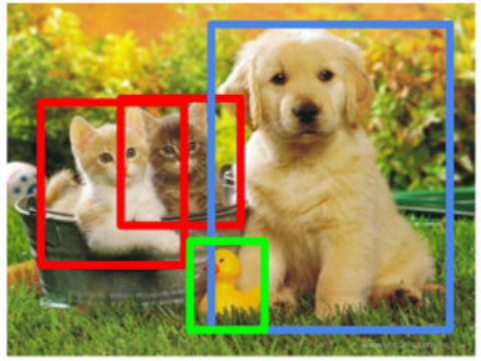
What Can Deep Learning Do?

- Objective Detection
- Where is it?

Classification

CAT

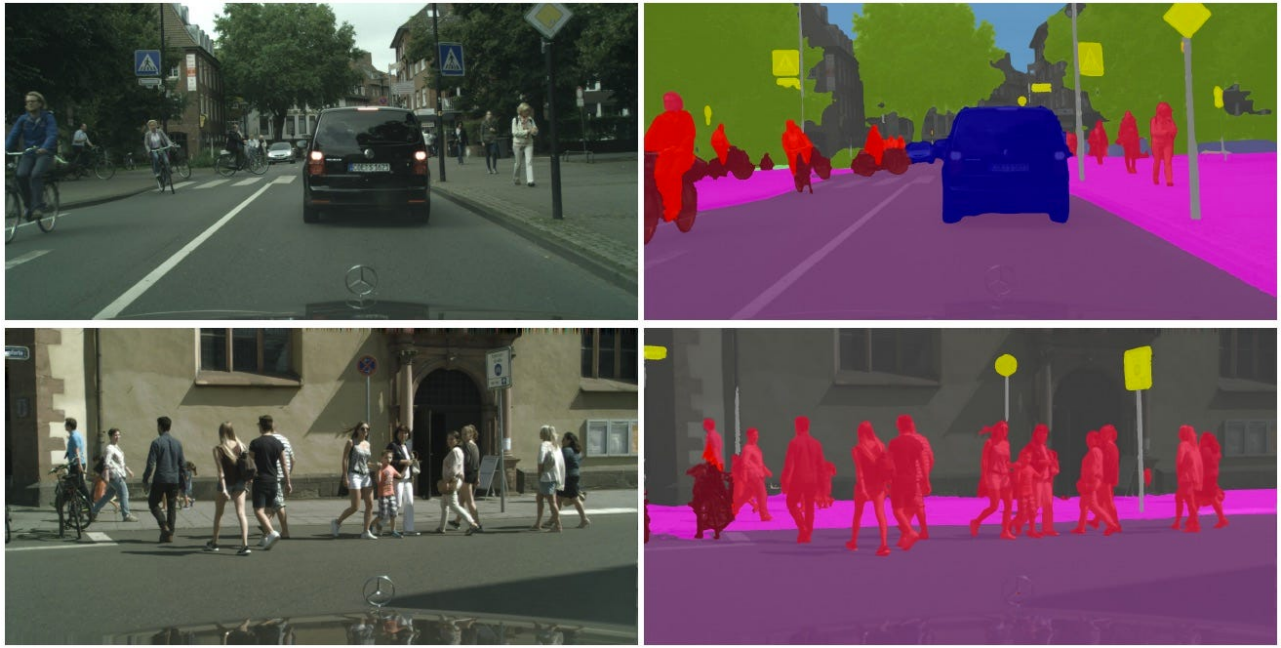
Object Detection



CAT, DOG, DUCK

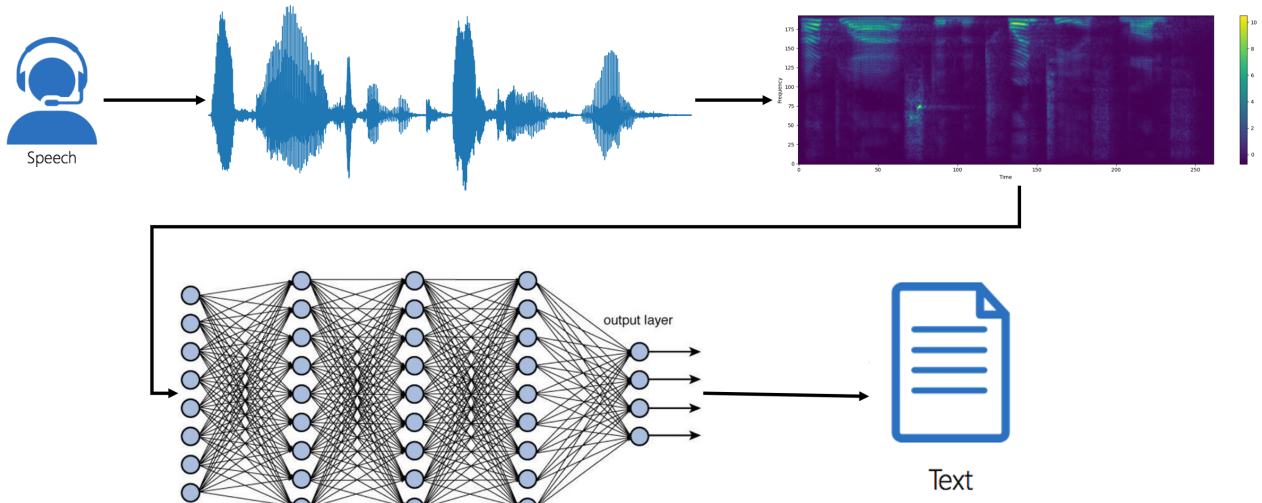
What Can Deep Learning Do?

- Semantic Segmentation
- Classify every pixel



What Can Deep Learning Do?

- Speech Recognition
- Voice to text

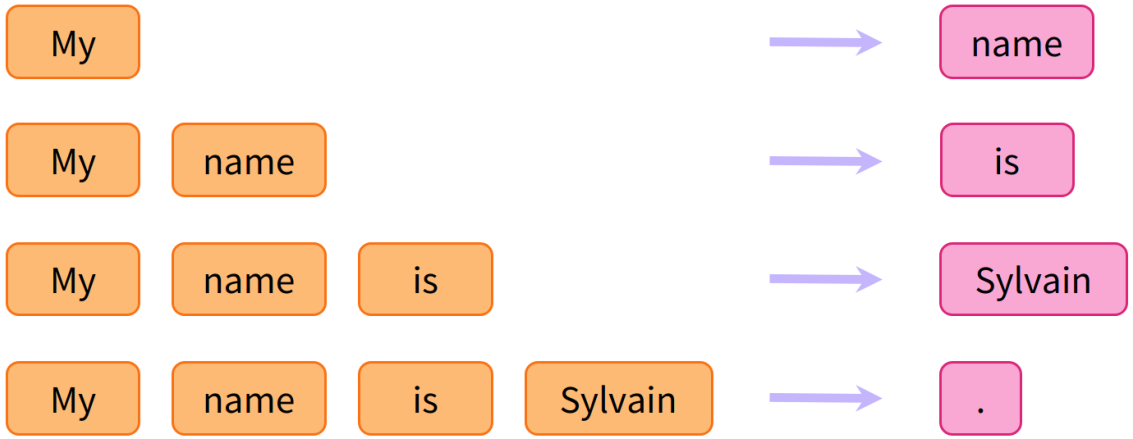


What Can Deep Learning Do?

- AIGC: AI-Generated Content
- Text-to-image/image-to-image

What Can Deep Learning Do?

- GPT: Generative Pre-trained Transformer
- Autoregressive model



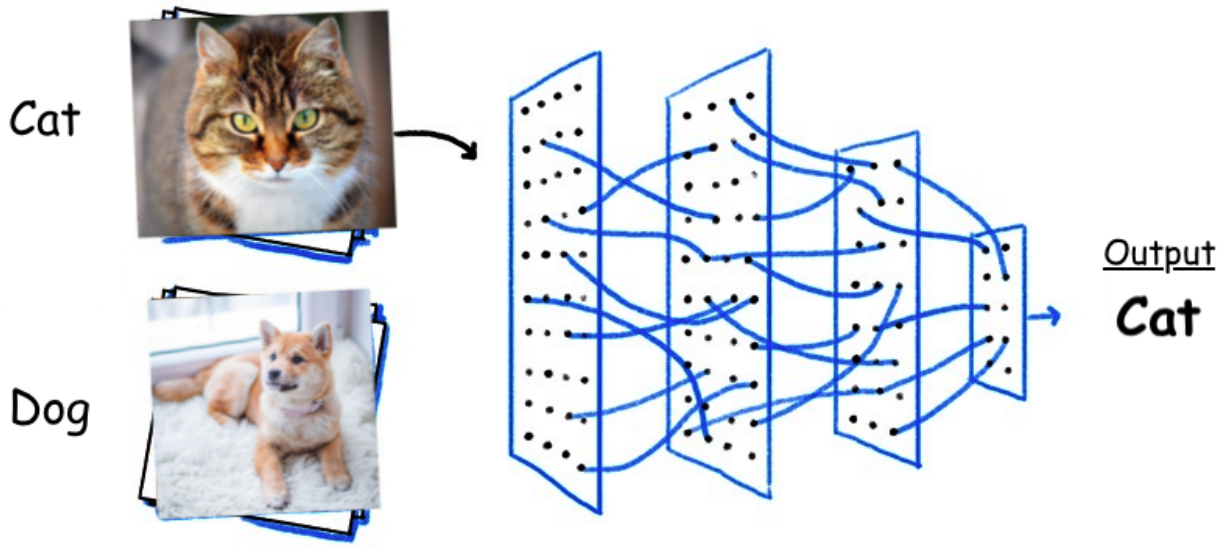
Problem Formulation

$$\overline{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{W}, \boldsymbol{x}) = \boldsymbol{f}_M(\boldsymbol{W}_M, \boldsymbol{f}_{M-1}(\boldsymbol{W}_{M-1}, \ldots\ldots \boldsymbol{f}_1(\boldsymbol{W_1}, \boldsymbol{x})))$$

$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} \sum_{i=1}^{N} L(\overline{\boldsymbol{y}}, \boldsymbol{y})$$

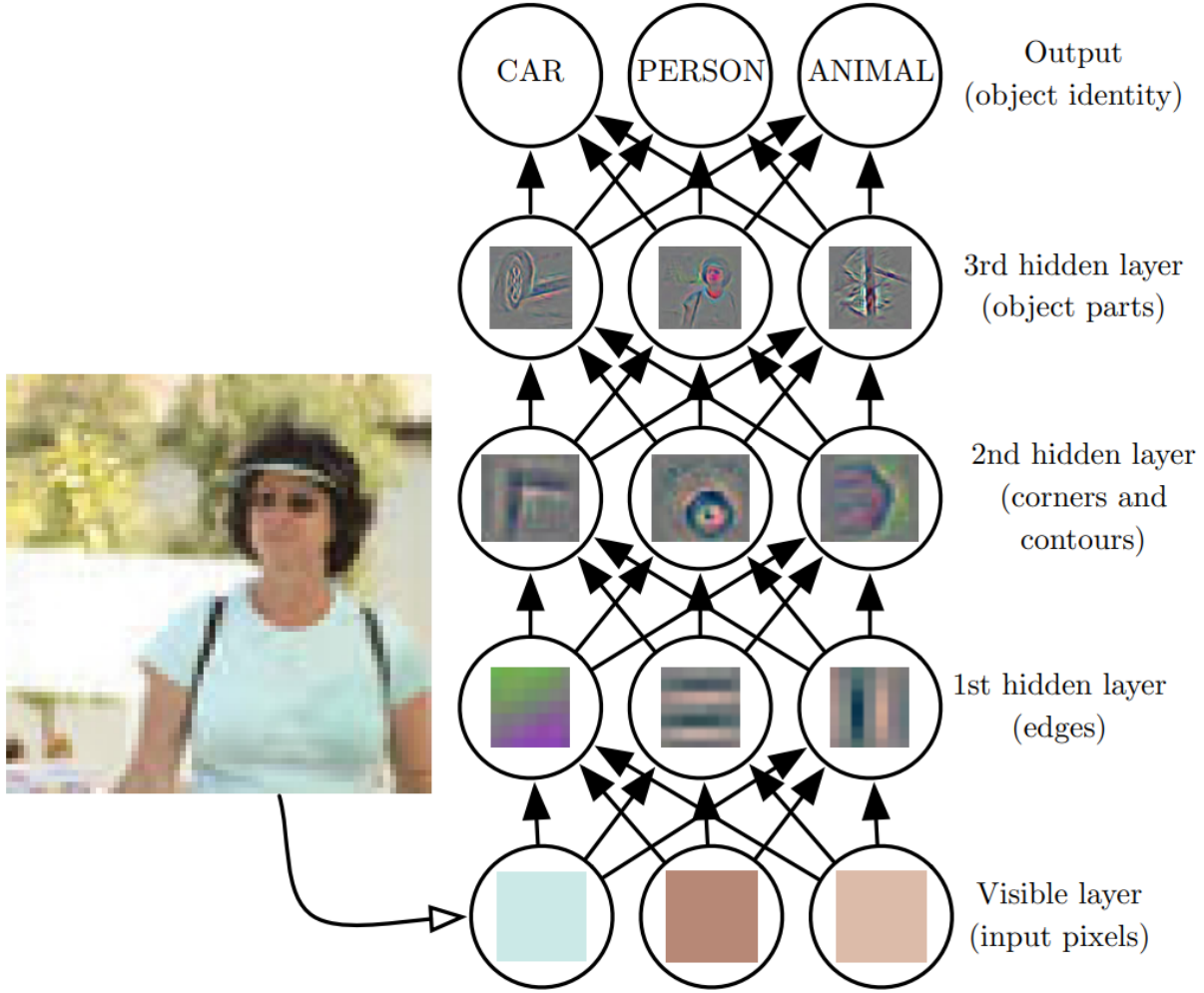
Revisiting Classification

- Input: $\boldsymbol{x}$ is the image (RGB)
- Output: cat: $\boldsymbol{y} = [1, 0]$; dog: $\boldsymbol{y} = [0, 1]$
- $\overline{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{W}, \boldsymbol{x})$



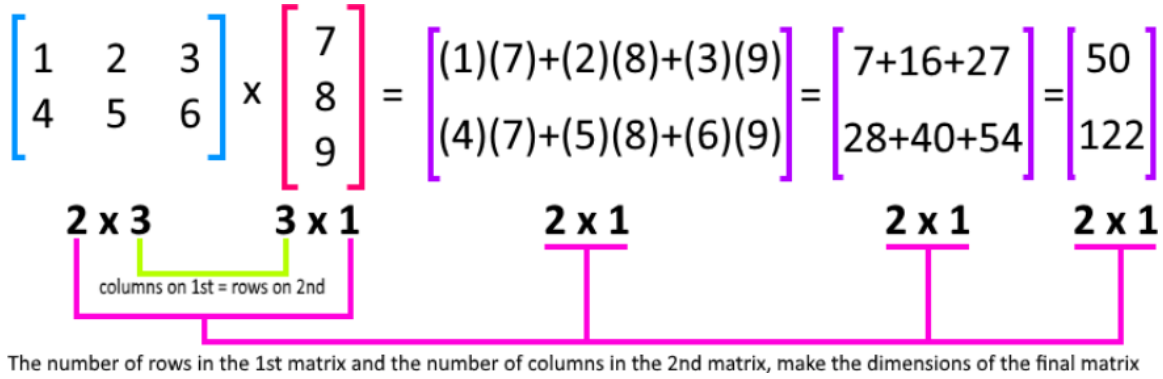
Imitating Human Perception

- Input: $\boldsymbol{x}$ is the image (RGB)
- 1st hidden layer:
  $\boldsymbol{x}_1 = \boldsymbol{f}_1(\boldsymbol{x}) = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$
- 2nd hidden layer:
  $\boldsymbol{x}_2 = \boldsymbol{f}_2(\boldsymbol{x}) = \sigma(\boldsymbol{W}_2\boldsymbol{x}_1 + \boldsymbol{b}_2)$
- ... ...



Matrix Multiplication

- $\boldsymbol{W}_1\boldsymbol{x}$



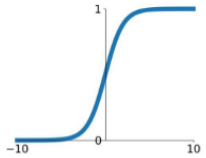
Activation Function

- $\boldsymbol{f}_1(\boldsymbol{x}) = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$

Activation Functions

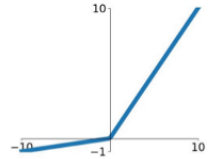
Sigmoid

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



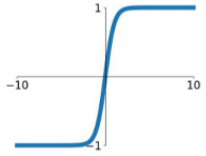
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

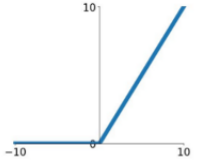


Maxout

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

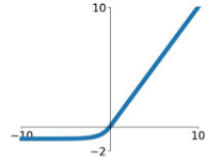
ReLU

$\max(0, x)$



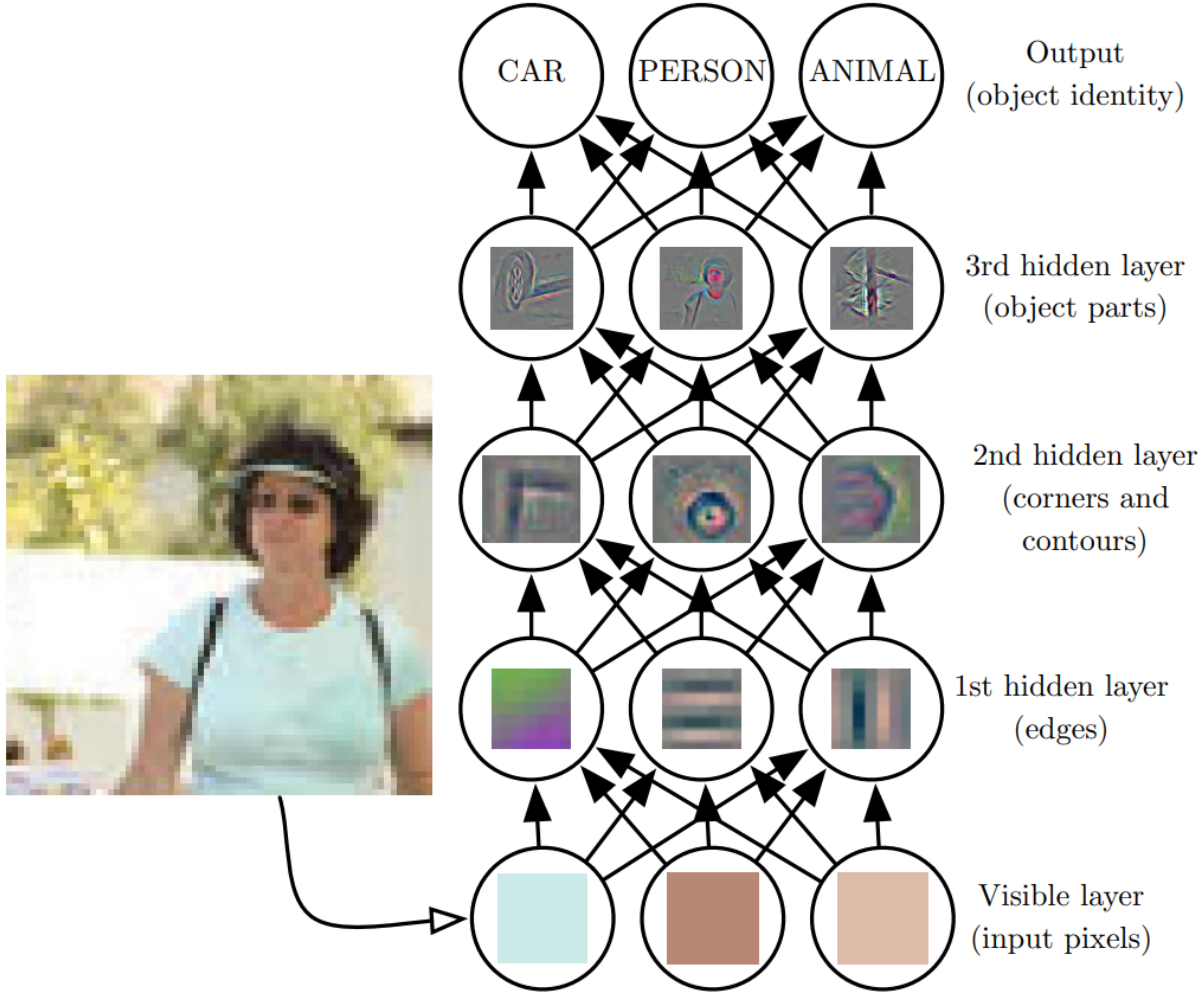
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Imitating Human Perception

- Input: $\boldsymbol{x}$ is the image (RGB)
- 1st hidden layer:
  $\boldsymbol{x}_1 = \boldsymbol{f}_1(\boldsymbol{x}) = \sigma(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$
- 2nd hidden layer:
  $\boldsymbol{x}_2 = \boldsymbol{f}_2(\boldsymbol{x}) = \sigma(\boldsymbol{W}_2\boldsymbol{x}_1 + \boldsymbol{b}_2)$
- ... ...



Output Function

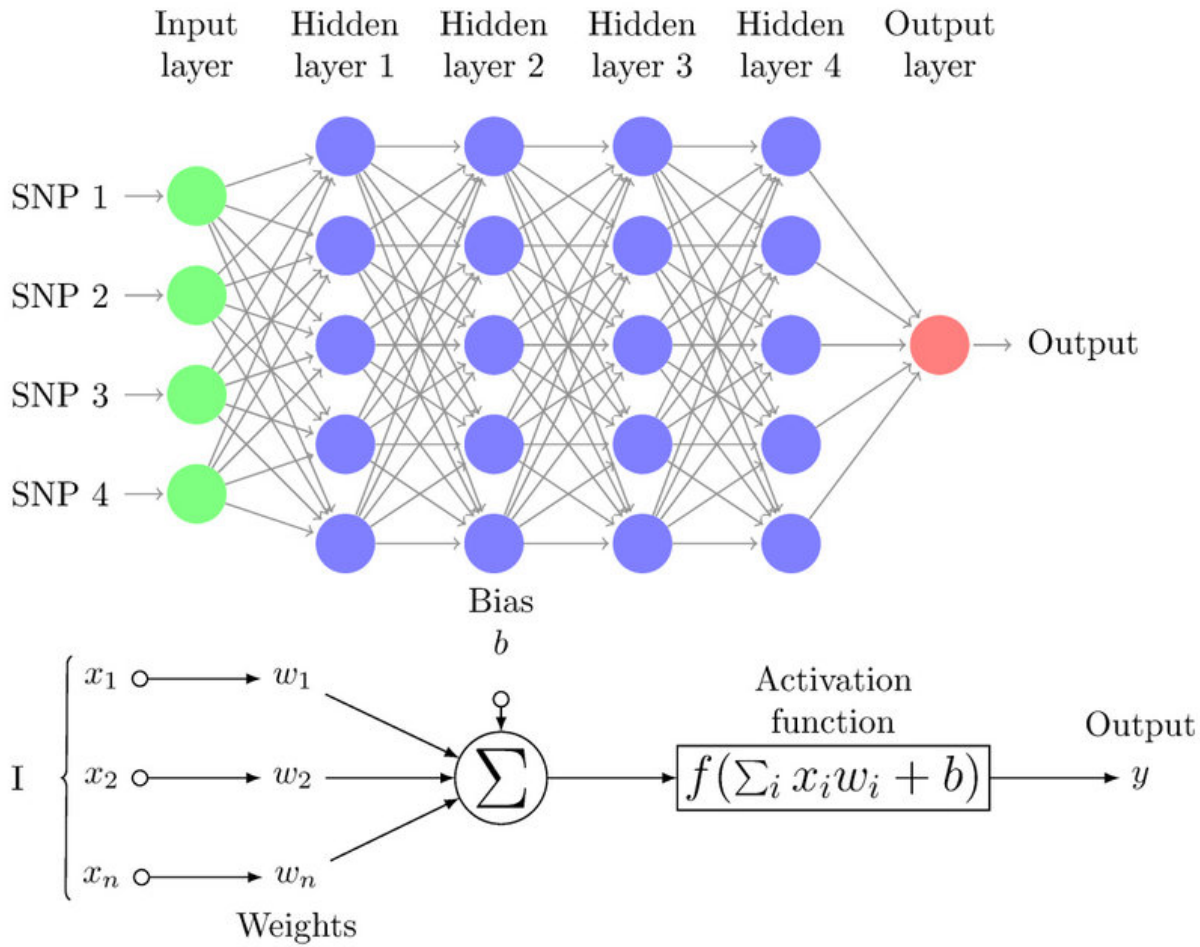
- $\boldsymbol{f}_M(\boldsymbol{x}) = \mathrm{softmax}(\boldsymbol{W}_M \boldsymbol{x}_{M-1} + \boldsymbol{b}_l$
- $softmax(\boldsymbol{x}_M)_i = \frac{x_{M,i}}{\sum_j x_{M,j}}$
- Regarded as probabilities

$$\begin{bmatrix} \frac{e^{x1}}{e^{x1}+e^{x2}+e^{x3}+e^{x4}} \\ \frac{e^{x2}}{e^{x1}+e^{x2}+e^{x3}+e^{x4}} \\ \frac{e^{x3}}{e^{x1}+e^{x2}+e^{x3}+e^{x4}} \\ \frac{e^{x4}}{e^{x1}+e^{x2}+e^{x3}+e^{x4}} \end{bmatrix} = \begin{bmatrix} e^{x1} \\ e^{x2} \\ e^{x3} \\ e^{x4} \end{bmatrix} /(e^{x1}+e^{x2}+e^{x3}+e^{x4})$$

$$\Longrightarrow e^x / sum(e^x)$$

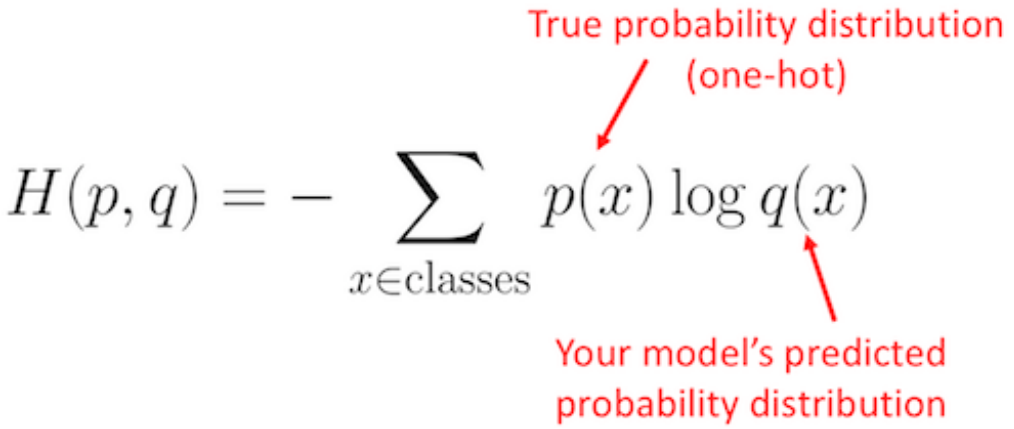
Multi-layer Perceptrons (MLP)

- $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}_M \circ \boldsymbol{f}_{M-1} \circ \ldots\ldots \circ \boldsymbol{f}_1(\boldsymbol{x})$
- $\boldsymbol{W}_1, \boldsymbol{b}_1, \ldots\ldots, \boldsymbol{W}_M, \boldsymbol{b}_M$ are trainable



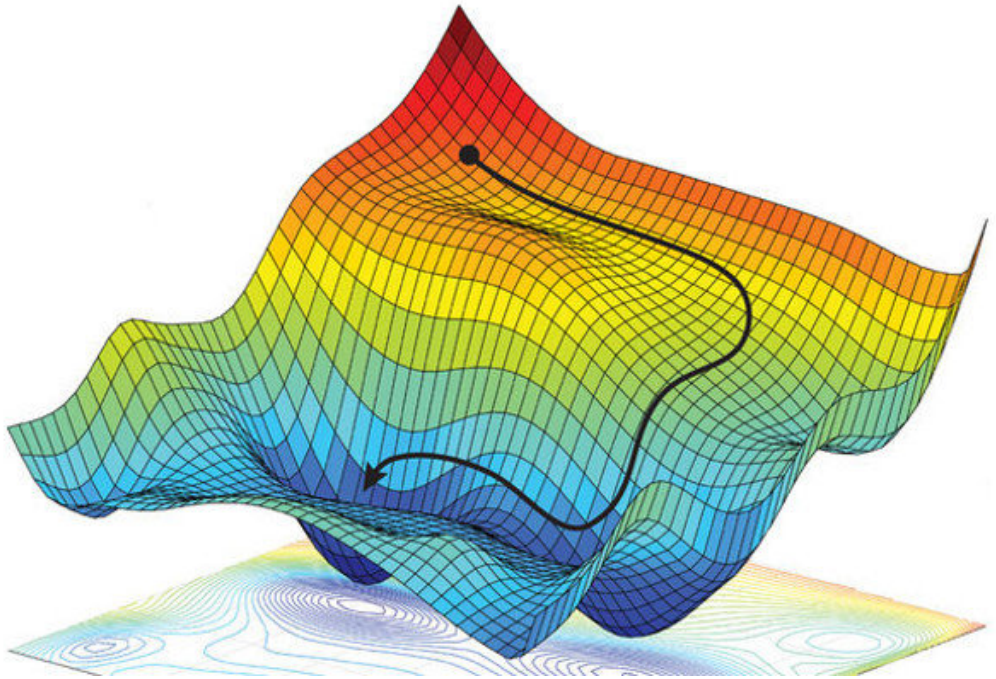
Loss Function

- $L(\overline{\boldsymbol{y}}, \boldsymbol{y})$, indicates the quality
- Smaller is better $\downarrow$
- Cross entropy is for classification
- e.g. $\boldsymbol{p} = [1, 0]; \boldsymbol{q} = [0.8, 0.2]$
- $CE(\boldsymbol{p}, \boldsymbol{q}) = -log(0.8)$



Gradient Descent

- Iterative algorithm
- $\boldsymbol{W}^{(t)} = \boldsymbol{W}^{(t-1)} - \gamma \frac{\partial L}{\partial \boldsymbol{W}^{(t)}}$
- γ is the learning rate
- Chain rule $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

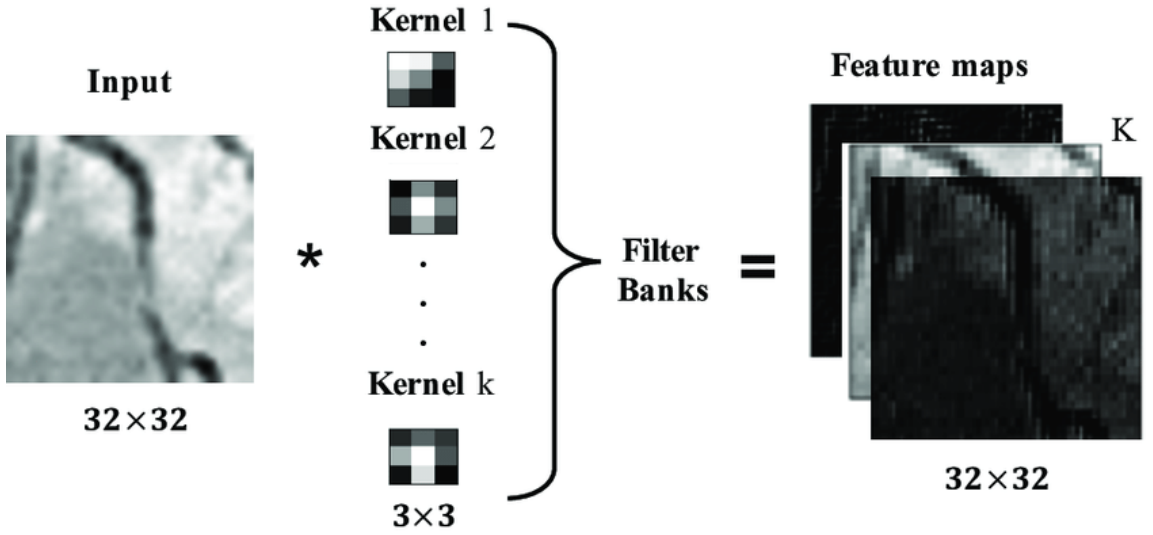


Implementation

- Convolutional Neural Networks
- Training and Testing
- PyTorch

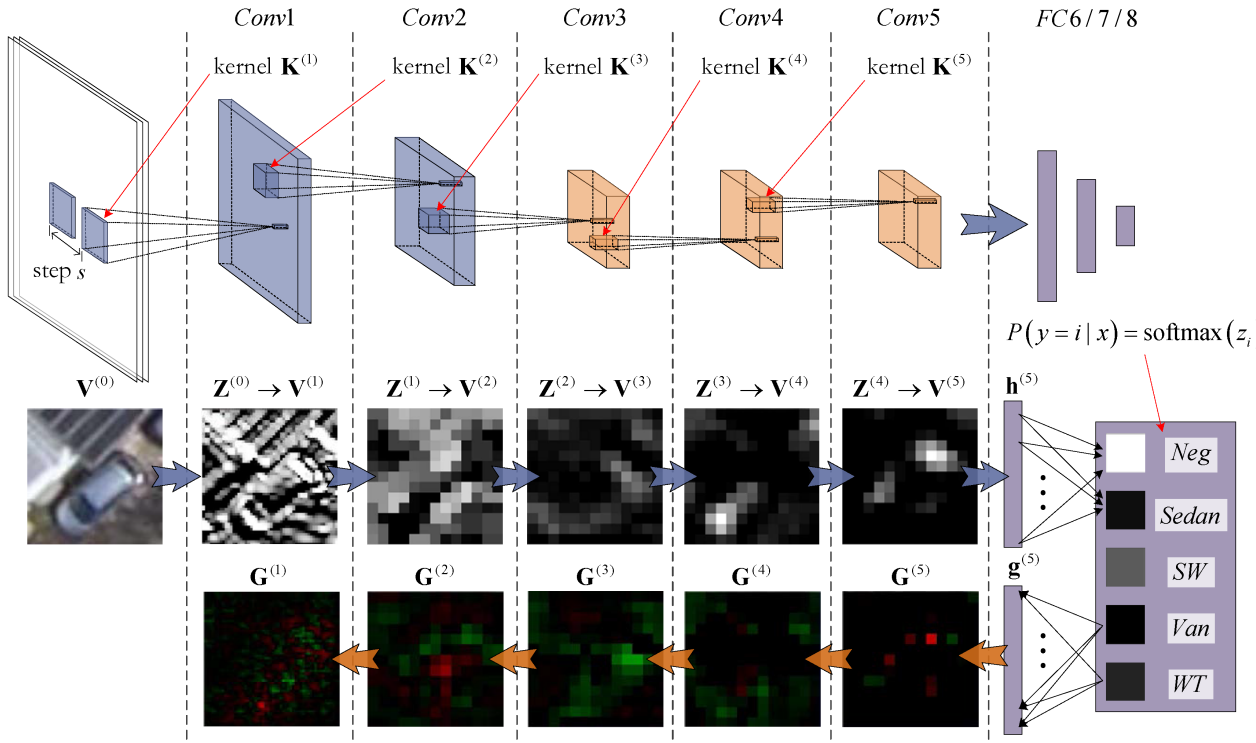
Convolution

- From signal processing
- For feature extraction



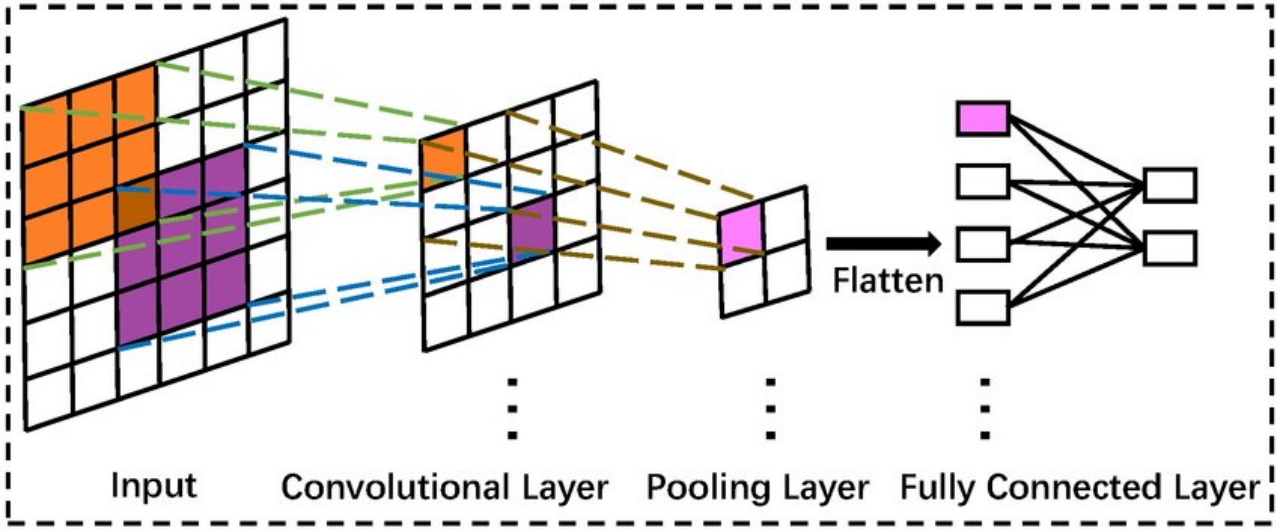
Convolutional Layers

- Convolutional layer
- Preserve 2D structure
- Local perception
- Less weights



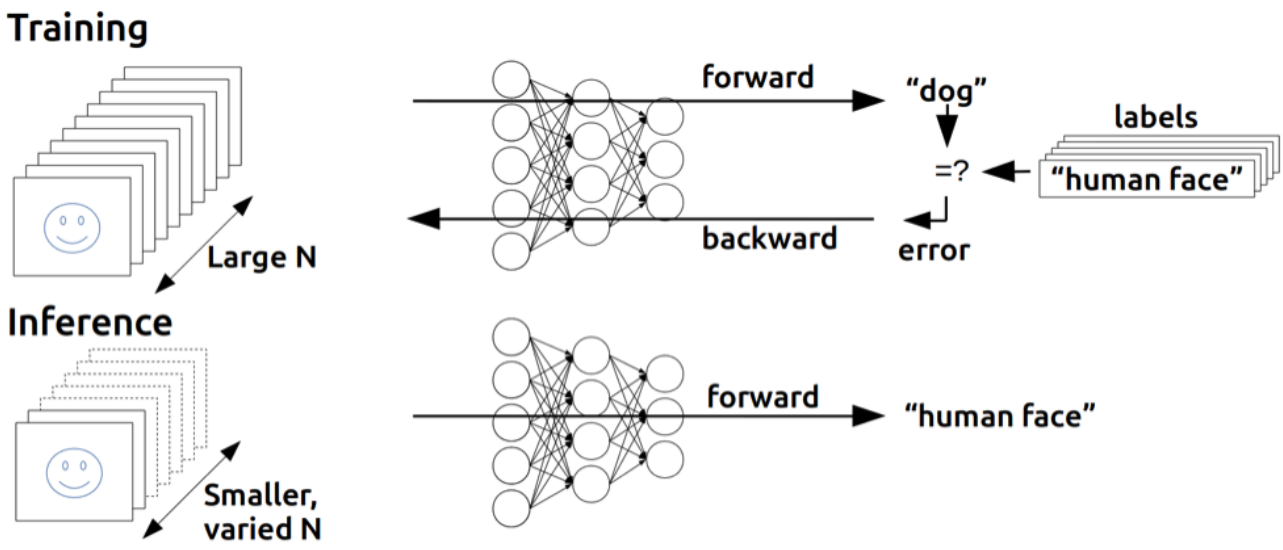
Convolutional Neural Networks

- Convolutional layer
- Max/Avg. pooling layer
- Fully connected layer (MLP)



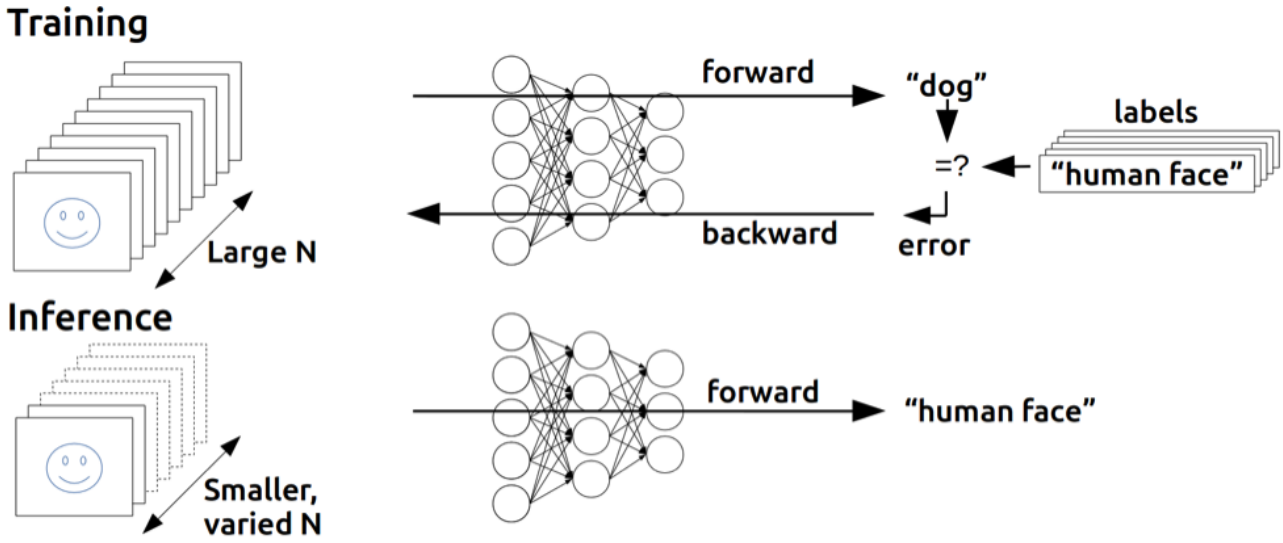
Training Neural Networks

- Sample a batch of data
- Forward: compute the loss
- backward: update the weights
- $\boldsymbol{W}^{(t)} = \boldsymbol{W}^{(t-1)} - \gamma \frac{\partial L}{\partial \boldsymbol{W}^{(t)}}$



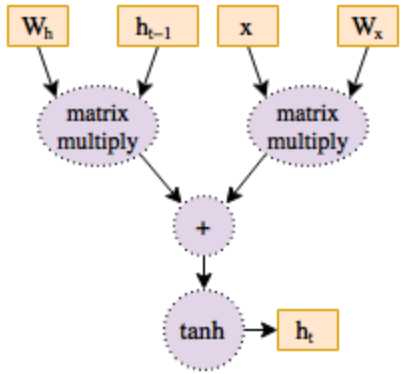
Testing Neural Networks

- Sample a batch of data
- Compute the loss and accuracy



PyTorch

- PyTorch is famous
- Computation graph



$$h_t = \tanh\left(W_h h_{t-1}^\top + W_x x^\top\right)$$