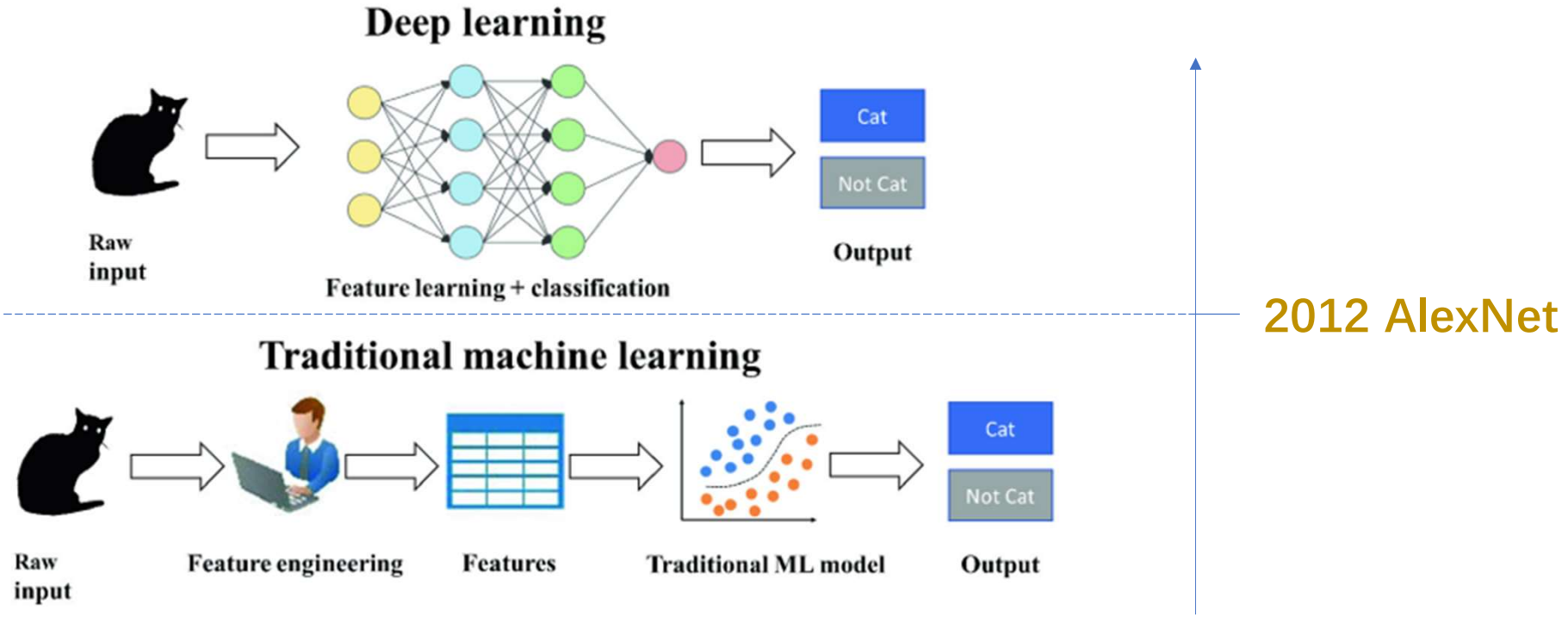
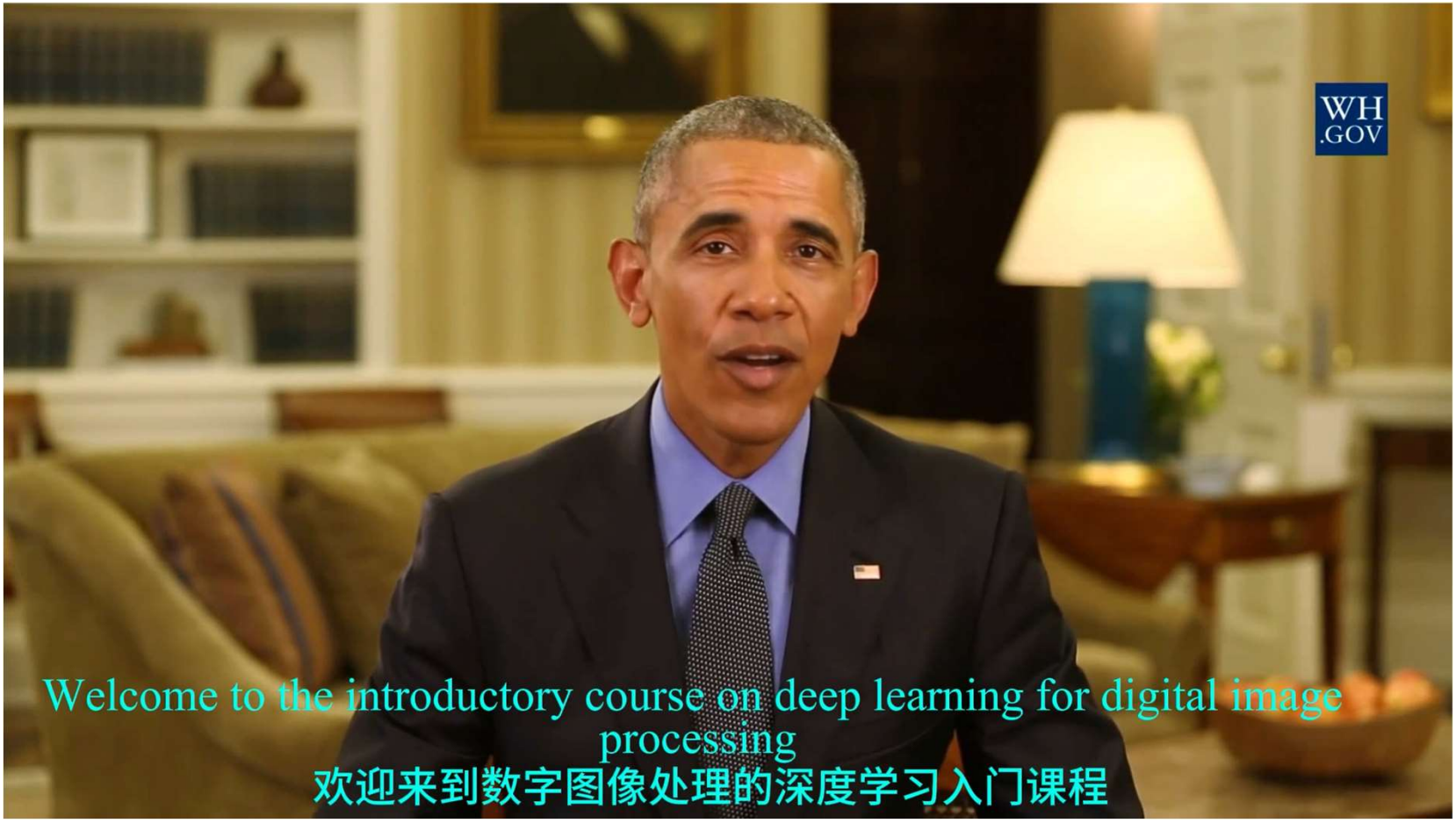
深度学习简介
Introduction to Deep Learning



一些无法用传统算法实现的数字图像处理任务

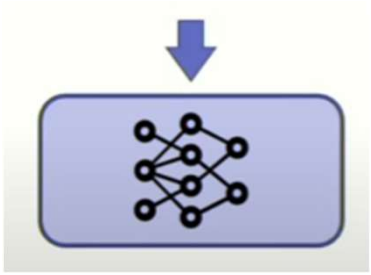
Complex Video Fake

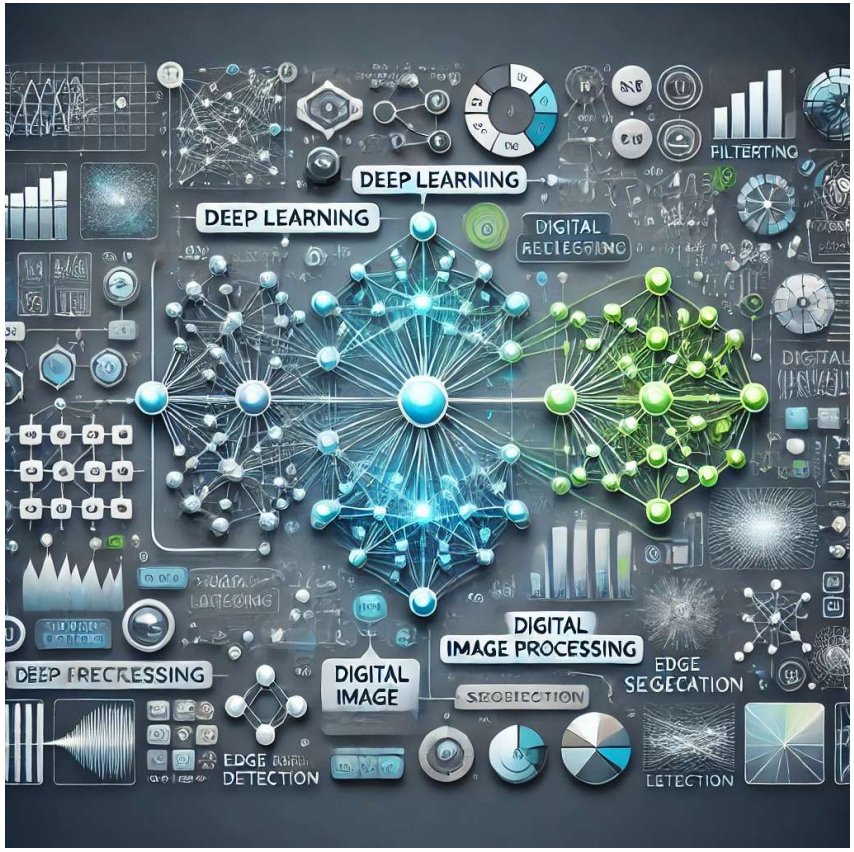


Complex Image Generation

Generating Images from Natural Language

□An□inġăĝê□ộğ□đêêř□ľêăsŋîŋĝ□ăŋđ□
đîĝîʧăľ□inġăĝê□řsộçêșșîŋĝ□ăŋđ□
ʧḥêîs□sêľăʧîộŋșḥîř□

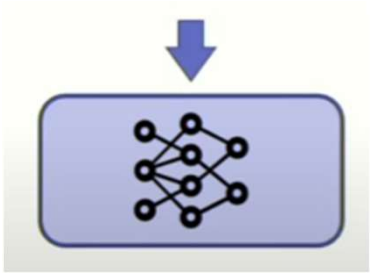




Complex Image Generation

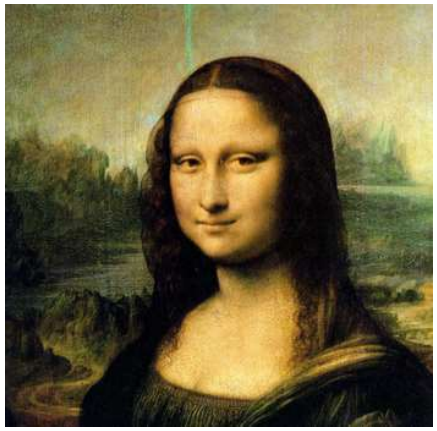
Generating Images from Natural Language

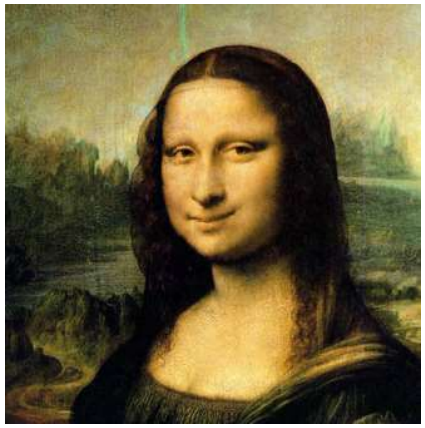
□ắ□řḥộʈộ□ộğ□ṇộŋlêỳ□xsîʈê□đêêř□
ľêắsŋîŋĝ□çộđê□xîʈḥ□řỳʈộsçḥ□

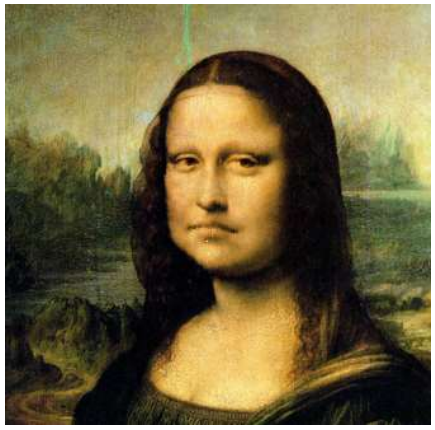


Why these tasks need (deep) learning?

Tasks do not need learning







2 Moving Least Squares Deformation

Here we consider building image deformations based on collections of points with which the user controls the deformation. Let p be a set of control points and q the deformed positions of the control points p. We construct a deformation function f satisfying the three properties outlined in the introduction using Moving Least Squares [Levin 1998]. Given a point v in the image, we solve for the best affine transformation $l_v(x)$ that minimizes

$$\sum_i w_i |l_v(p_i) - q_i|^2 \qquad (1)$$

where p_i and q_i are row vectors and the weights w_i have the form

$$w_i = \frac{1}{|p_i - v|^{2\alpha}}.$$

Because the weights w_i in this least squares problem are dependent on the point of evaluation v, we call this a *Moving Least Squares* minimization. Therefore, we obtain a different transformation $l_v(x)$ for each v.

- 目标很明确，很容易转化为数学优化问题
- 只依赖原始图像本身 和/或 少量用户交互

Why these tasks need (deep) learning?

Is this picture

a dog

or a cat

or a horse ?

Need Knowledge

about what's

cat/dog/horse…

Why these tasks need (deep) learning?

What it would be

like if Obame

Say Chinese?

Need Knowledge

about

the mapping from

audio to mouth shape

Why these tasks need (deep) learning?

They all need to

learn Knowledge (Prior)

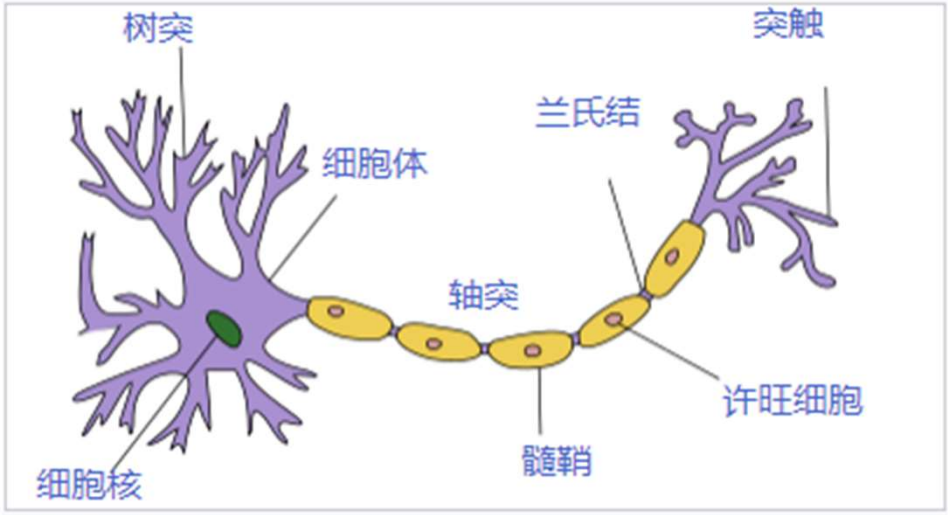
from data

How to Learn Knowledge from data?

模仿大脑的学习过程

“A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain.”

神经元的结构

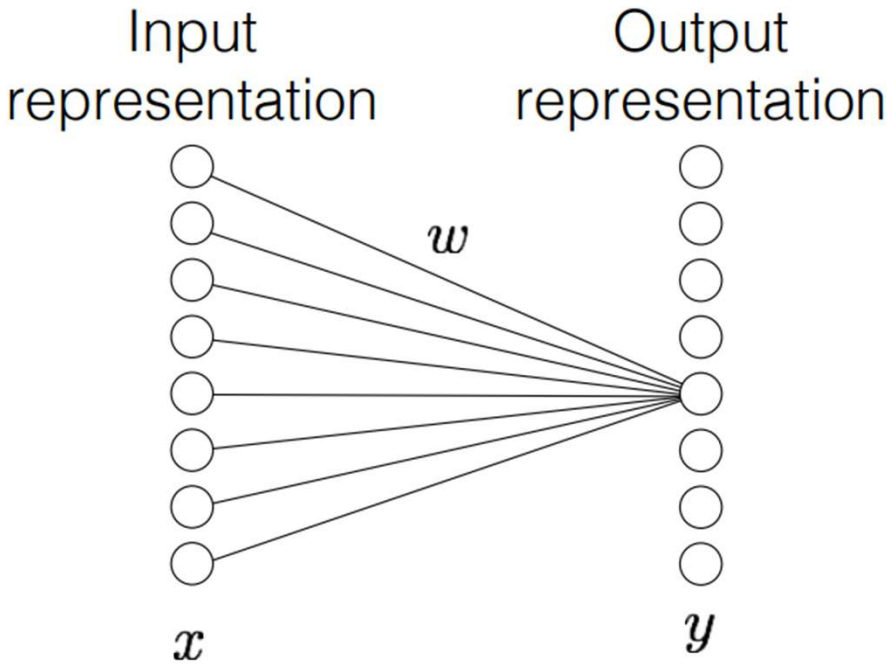


典型神经元的结构

神经元（英语：neuron）又名**神经细胞**（nerve cell），是组成神经系统结构和执行神经功能活动的一大类高度分化细胞，由胞体和胞突（树突和轴突）组成，属神经组织的基本结构和功能单位。神经元大致分为：感觉（传入）神经元，运动（传出）神经元、联络（中间）神经元三类。

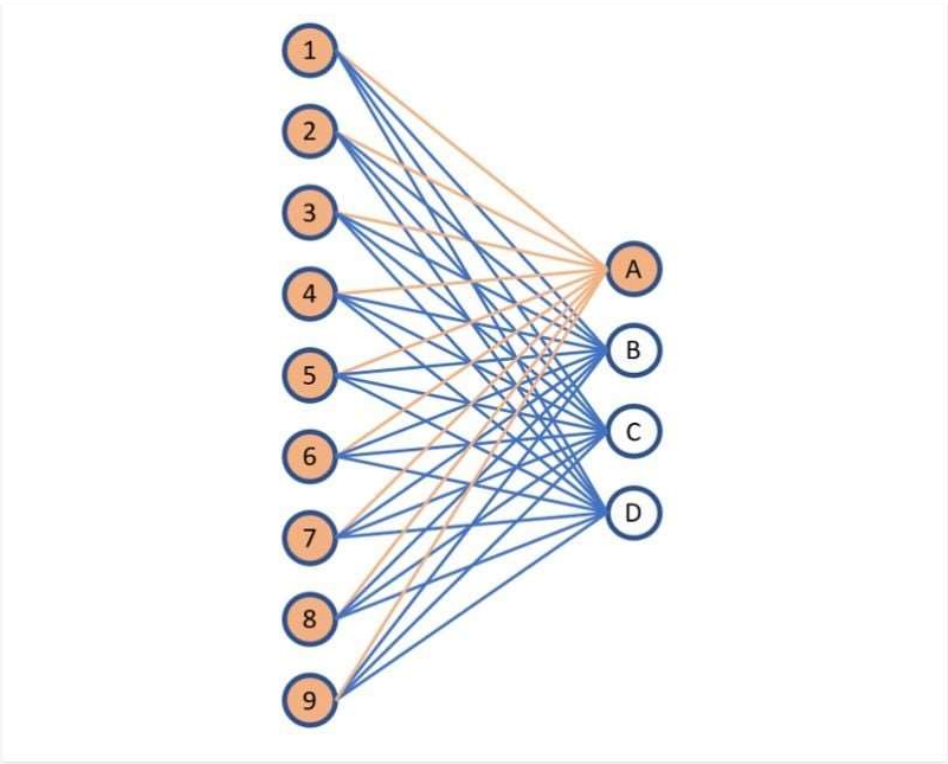
神经元具有感受刺激、整合信息和传导冲动的能力。神经元感知环境的变化后，再将信息传递给其他的神经元，并指令集体做出反应。神经元占了神经系统约一半，其他大部分由神经胶质细胞所构成。神经元的基本构造包括：树突、轴突、髓鞘和细胞核。传递形成电流，在其尾端为受体，借由化学物质（神经传递物质，如多巴胺、乙酰胆碱等）传导，在适当的量传递后在两个突触间形成电流传导。

深度学习中的“神经元”



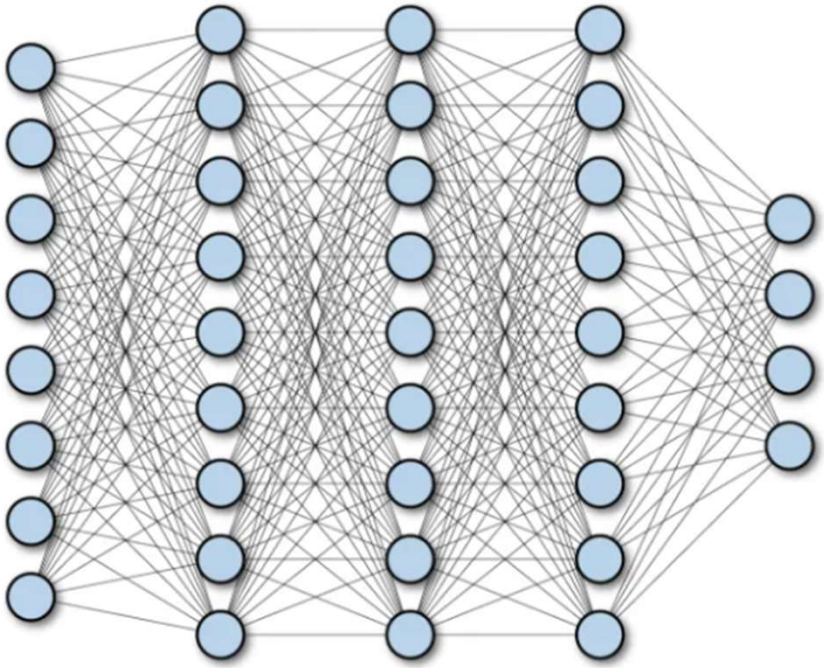
$$y_j = \sum_i w_{ij} x_i$$

i: the i^{th} dimension of x, j; the j^{th} dimension of y

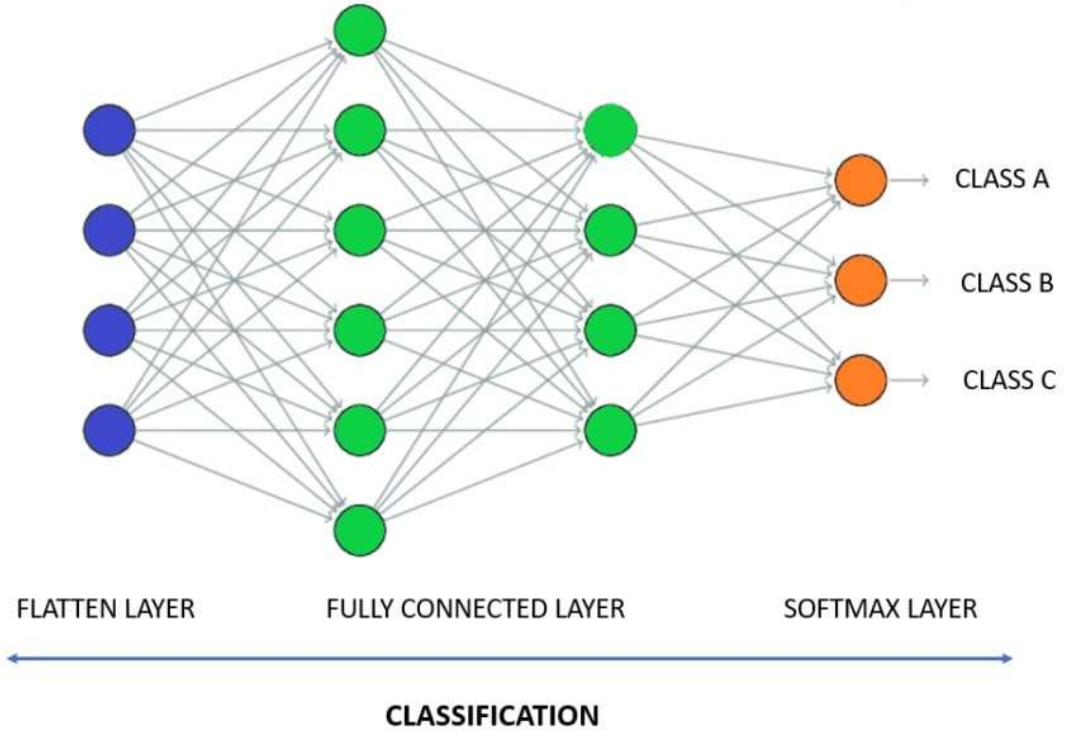


Fully Connected Layer
全连接层

多个“神经元”



Multilayer Deep Fully Connected Network, Image Source



Why this Neural Net could do Classification?

Toy Example: MNIST Dataset

Examples from MNIST dataset [LeCun. 1998]

MNIST Dataset

The MNIST database of handwritten digits (http://yann.lecun.com)

Data Card | Code (183) | Discussion (0) | Suggestions (0)

About Dataset

Context

MNIST is a subset of a larger set available from NIST (it's copied from http://yann.lecun.com/exdb/mnist/)

Content

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. .

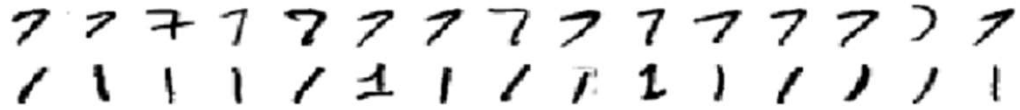
Four files are available:

- train-images-idx3-ubyte.gz: training set images (9912422 bytes)
- train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
- t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
- t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

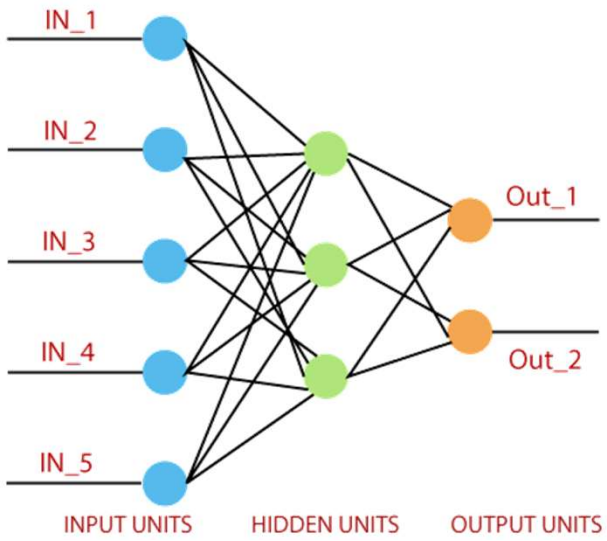
Warm-up Example: Binary Digit Classification

How to Learn the classification of 1 v.s. 7

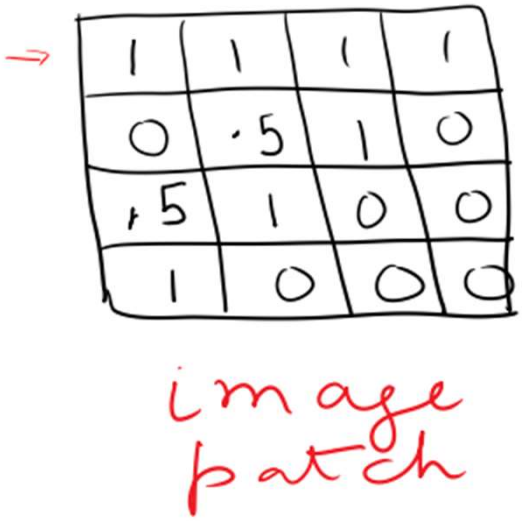
- **Collect Training Images**
  - Positive:
  - Negative:



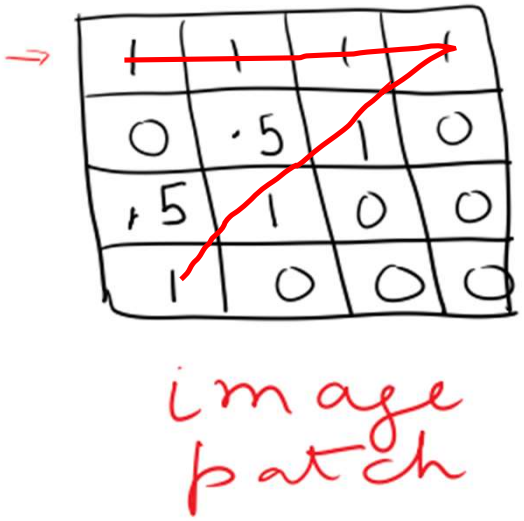
- **Training Time**
  - Compute feature vectors for positive and negative example images
  - Train a classifier
- **Test Time**
  - Compute feature vector on new test image:
  - Evaluate classifier



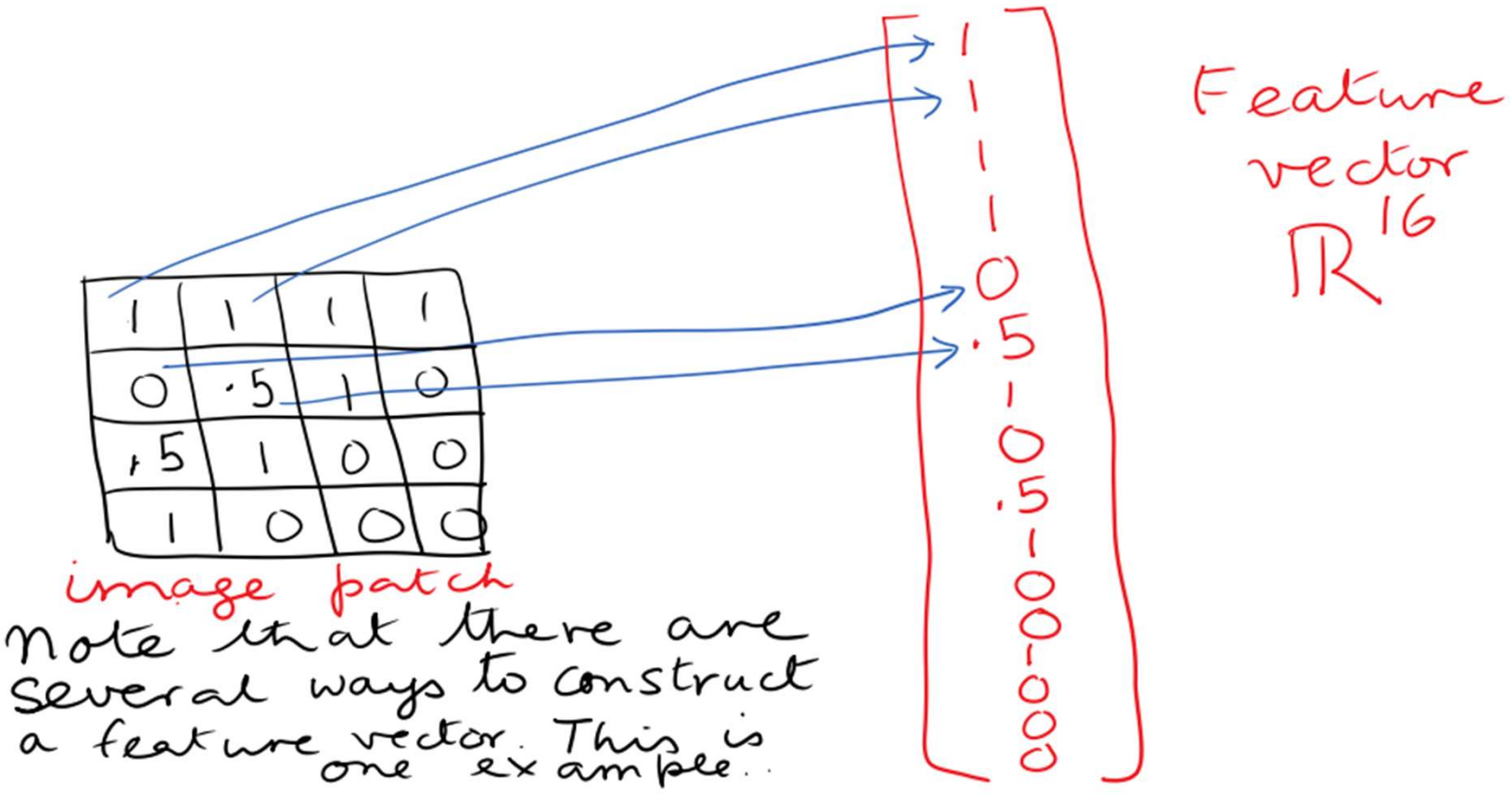
Analyze the process



Analyze the process

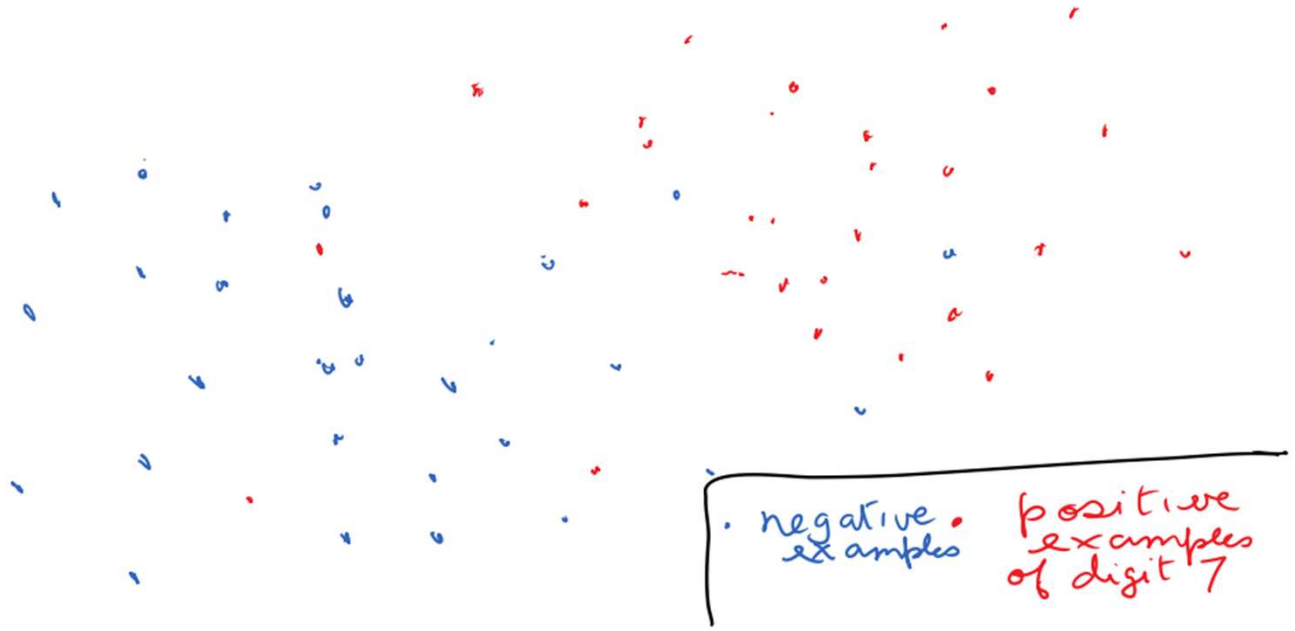


Analyze the process

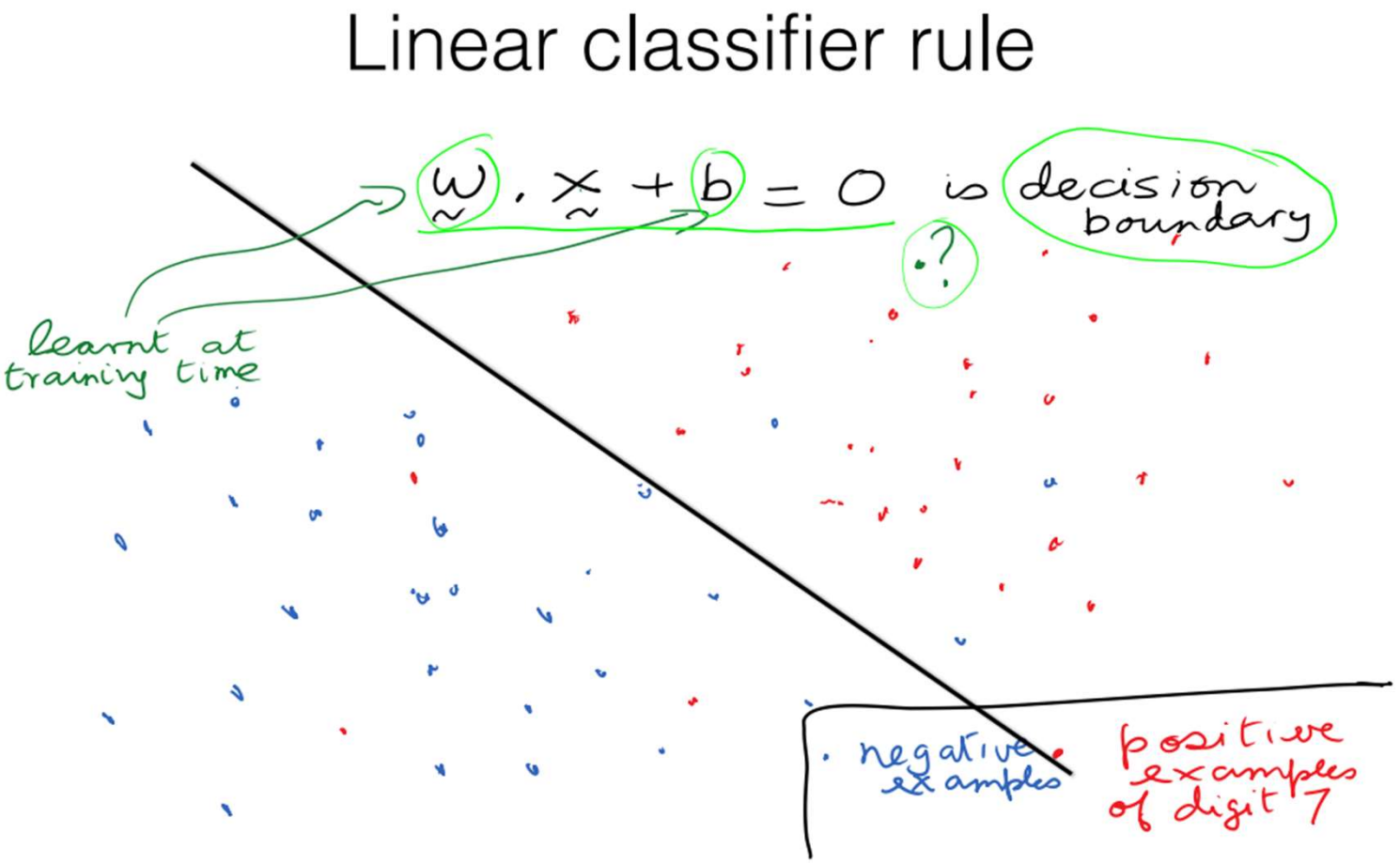


Treat it as high-dimensional points

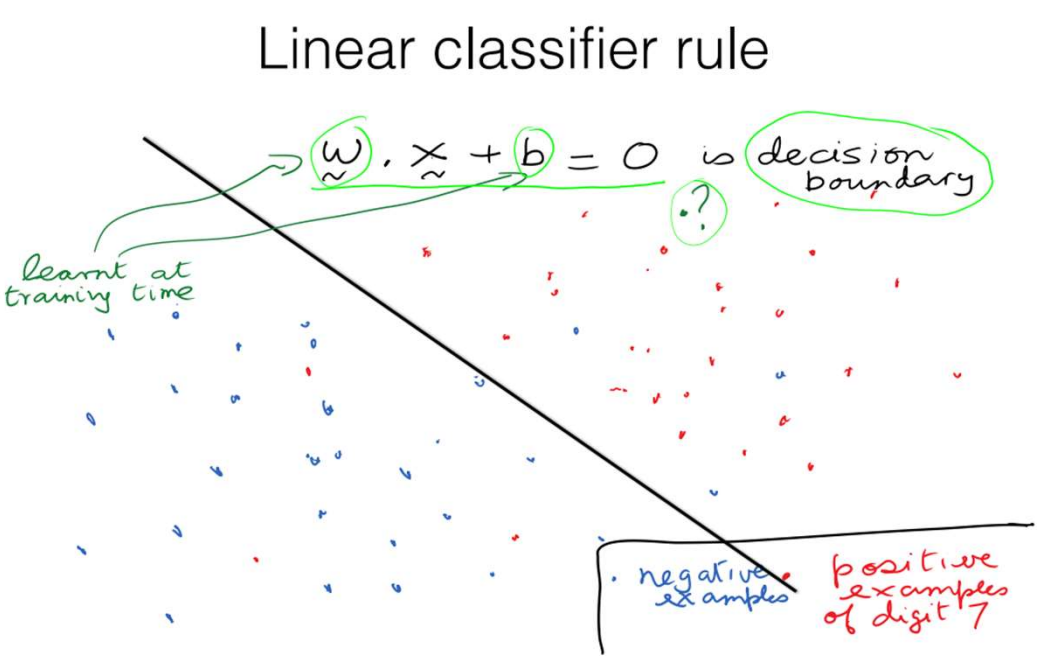
In feature space, positive and negative examples are just points…



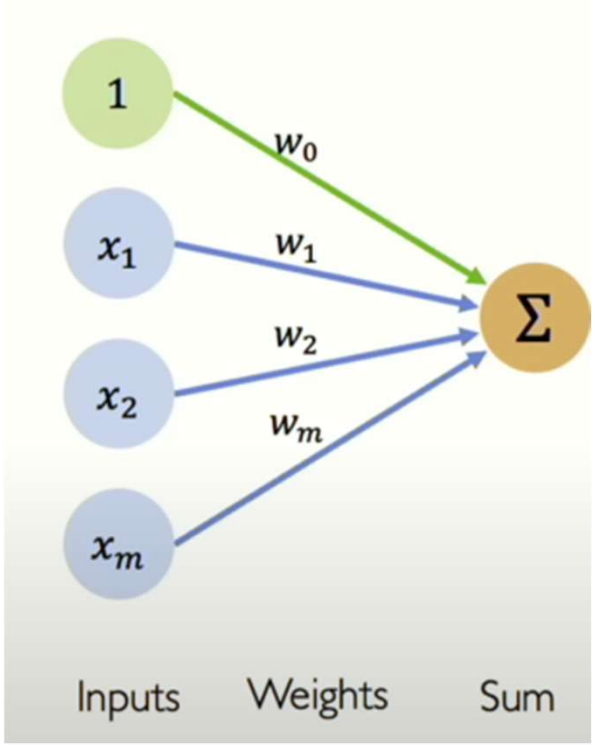
多元回归分析— 超平面


Linear classifier rule
$\underset{\sim}{w} \cdot \underset{\sim}{x} + b = 0$ is decision boundary
learnt at training time
?
negative examples
positive examples of digit 7

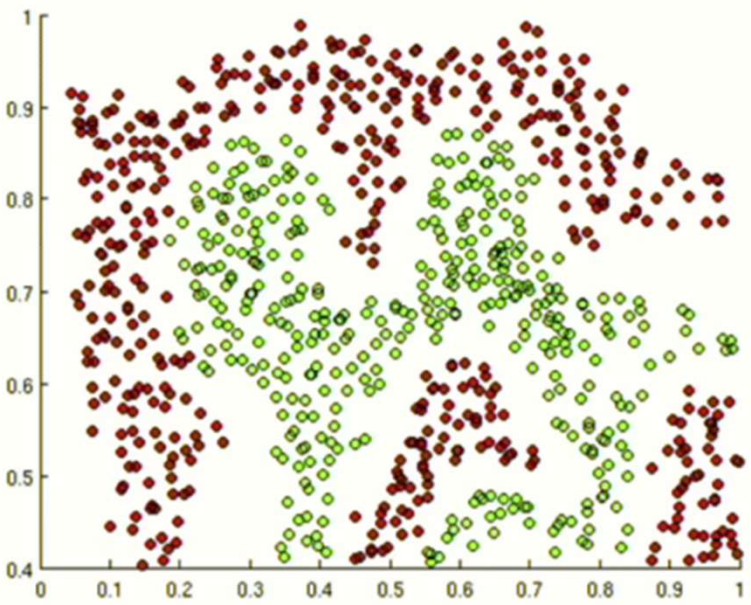
多元线性分类 == Fully Connected Layer

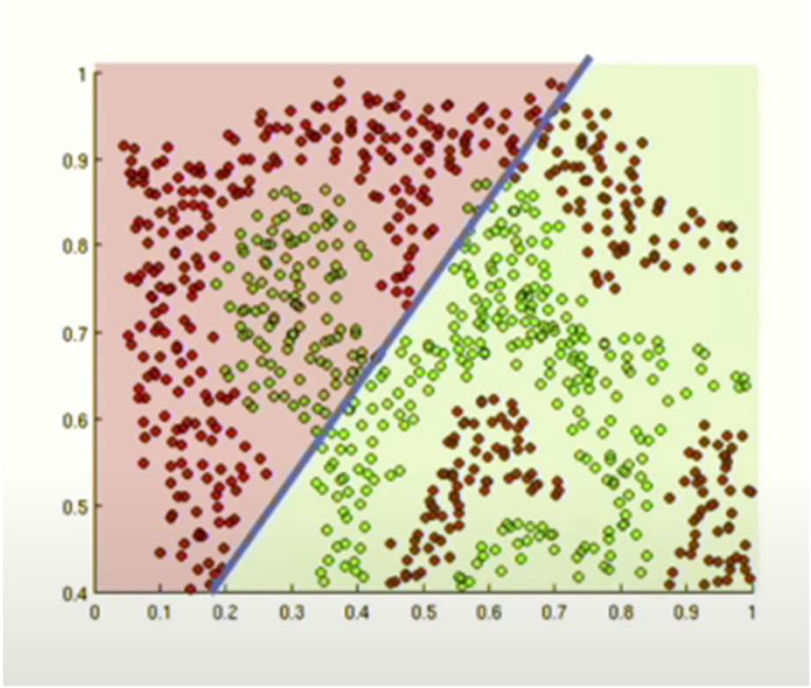


<->

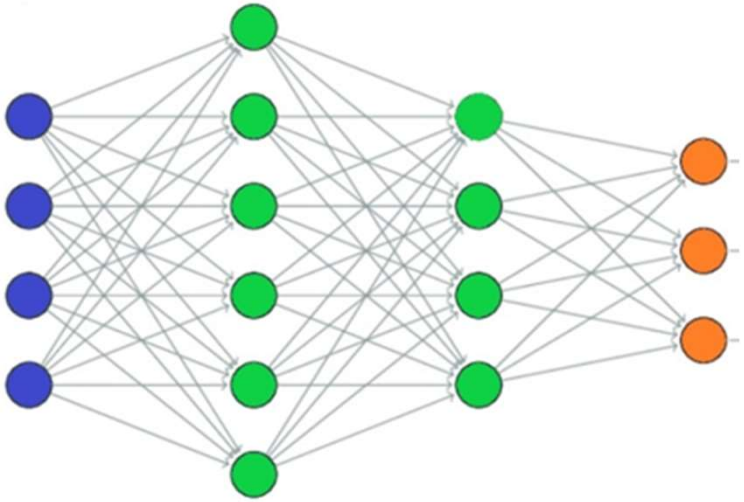


如果Points不是线性可分的?



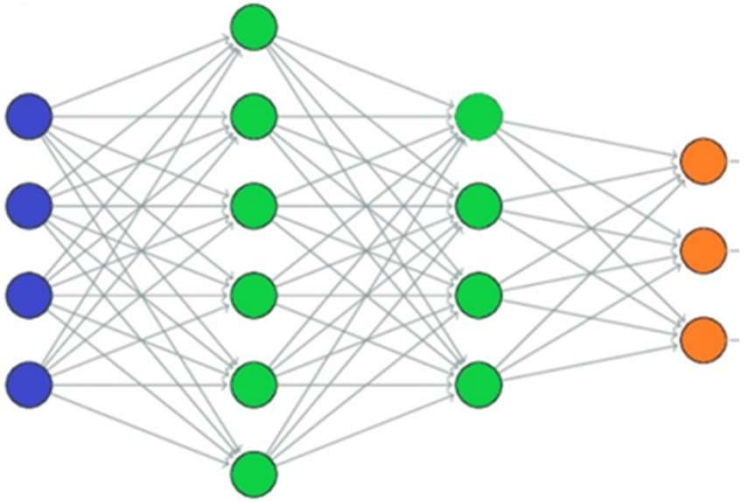


多层级叠加可行吗?



$$output = W_3 \times ((W_2) \times (W_1 \times X_{input}))$$

多层级叠加可行吗?



$$output = W_3 \times ((W_2) \times (W_1 \times X_{input}))$$

$$output = (W_3 \times W_2 \times W_1) \times X_{input}$$

What's the Problem

$$output = (W_3 \times W_2 \times W_1) \times X_{input}$$

线性运算的复合还是线性运算

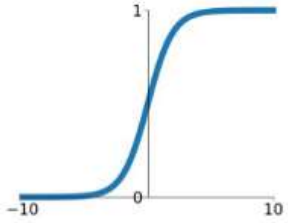
没有引入非线性运算

非线性运算（Activation Functions）

Activation Functions

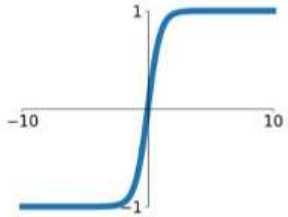
Sigmoid

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



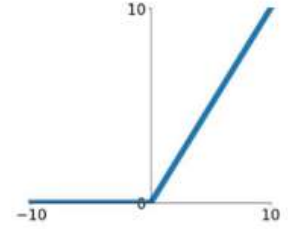
tanh

$$\tanh(x)$$



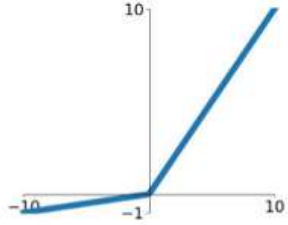
ReLU

$$\max(0, x)$$



Leaky ReLU

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

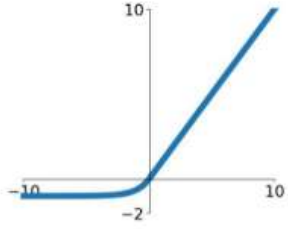


Maxout

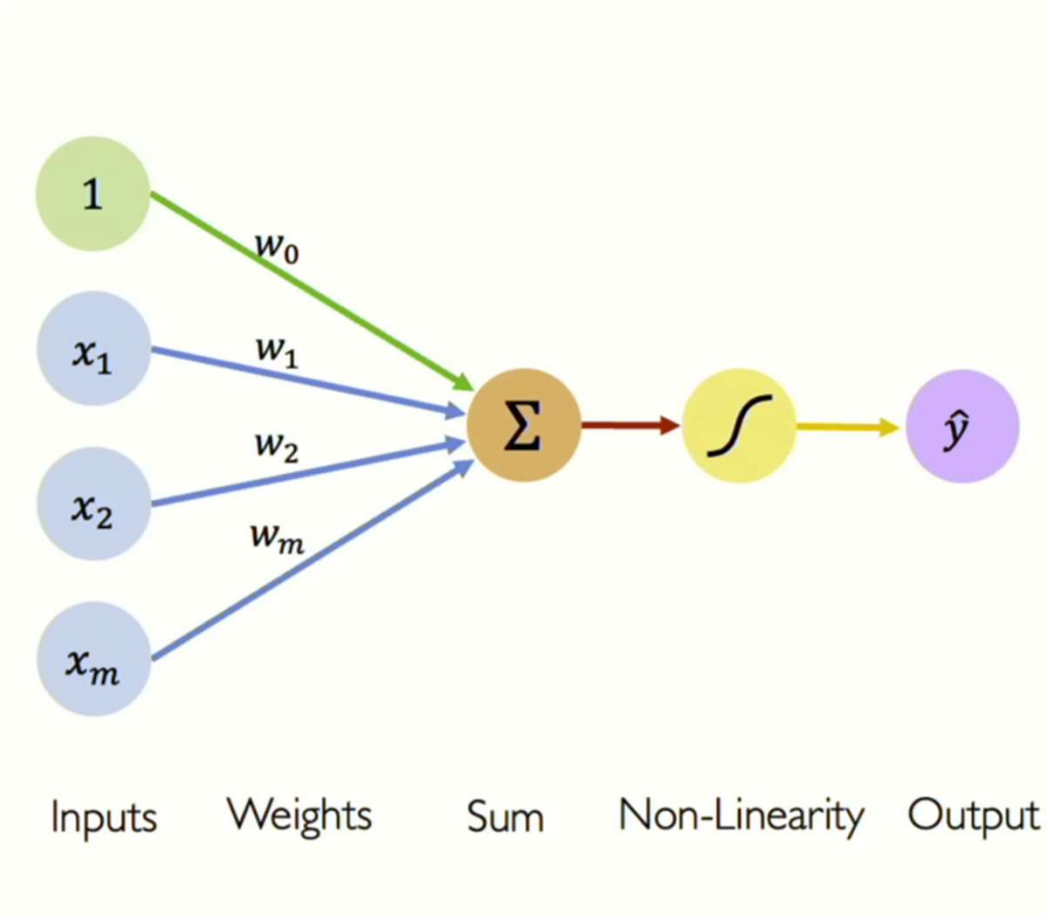
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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



常用的完整的Fully Connected Layer

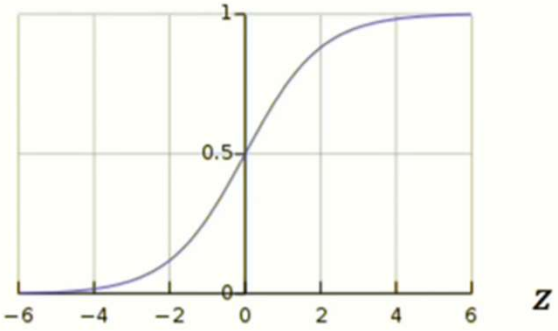


Activation Functions

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

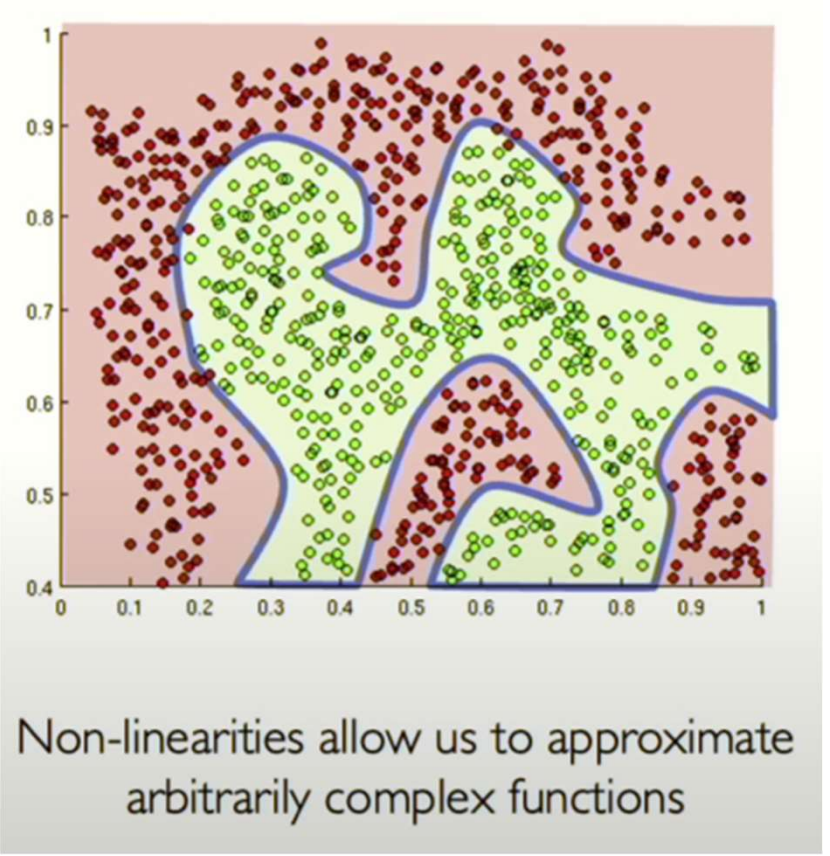
- Example: sigmoid function

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



Implement with PyTorch

```python
import torch
import torch.nn as nn

class Fully_Connected_Layer(nn.Module):
    def __init__(self, in_dim, out_dim):
        super(Fully_Connected_Layer, self).__init__()
        self.Linear_Layer = nn.Linear(in_dim, out_dim)
        self.activation = nn.ReLU()

    def forward(self, x):
        return self.activation(self.Linear_Layer(x))

# 创建全连接层
FC1 = Fully_Connected_Layer(16, 16)
FC2 = Fully_Connected_Layer(16, 16)
FC3 = Fully_Connected_Layer(16, 16)

# 判断层
judge_layer = nn.Linear(16, 1)

# 输入数据
x = torch.randn(1, 16)  # 示例输入

# 前向传播
out = judge_layer(FC3(FC2(FC1(x))))  # out < 0: 1, out >= 0: 7
```



通过多层非线性层复合可以做复杂的分类任务

如何学出每层的W_i（网络权重/Weights）

使其完美解决该任务

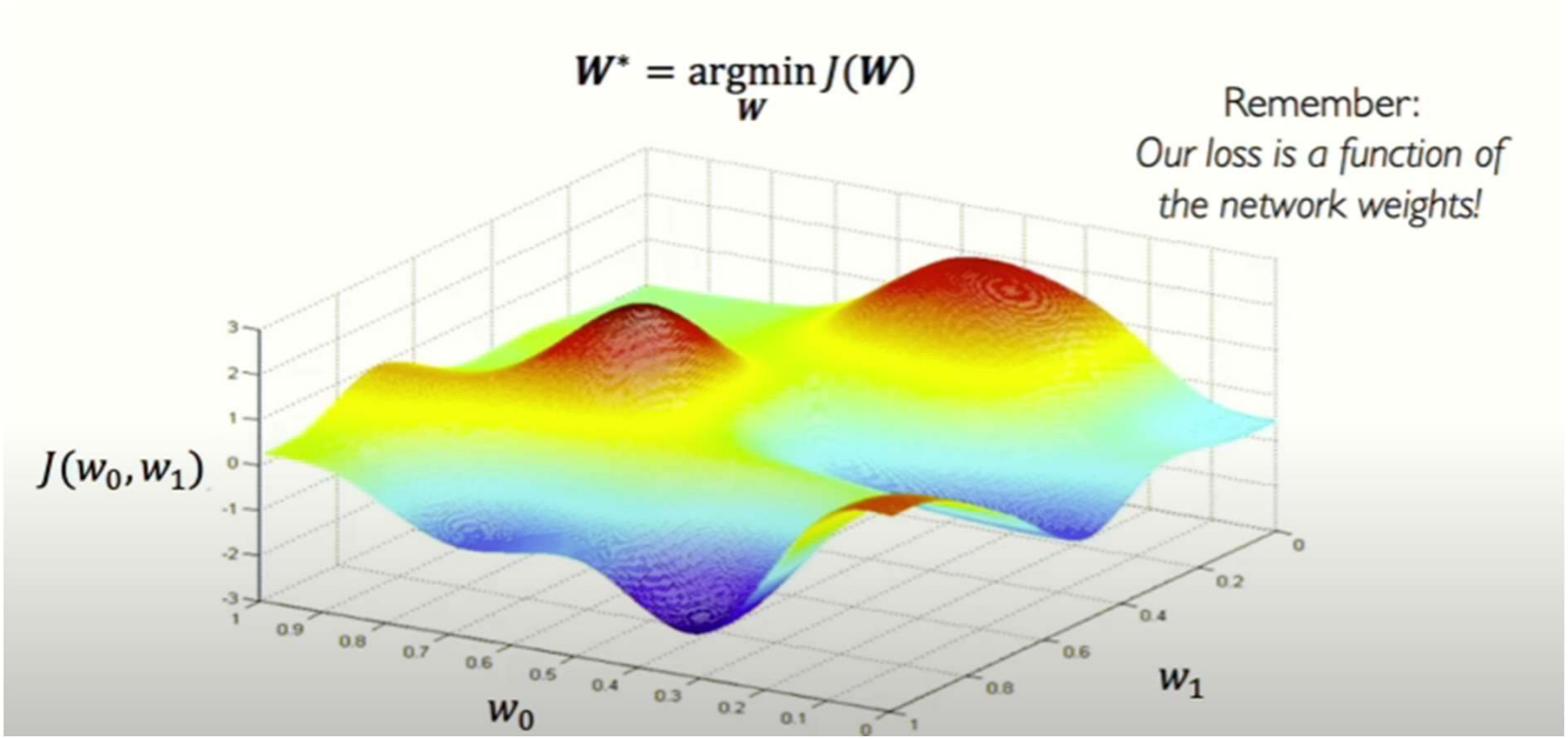
Optimize (Train) with Loss Function

We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Loss Function



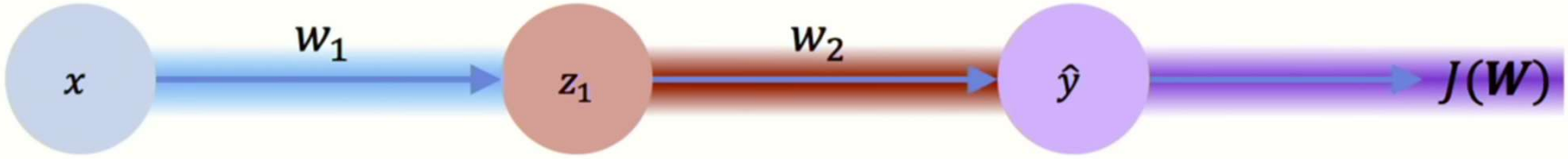
Gradient Descent

One iteration of gradient descent:

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta_t \frac{\partial L(\mathbf{w}^{\mathbf{t}})}{\partial \mathbf{w}}$$

learning rate

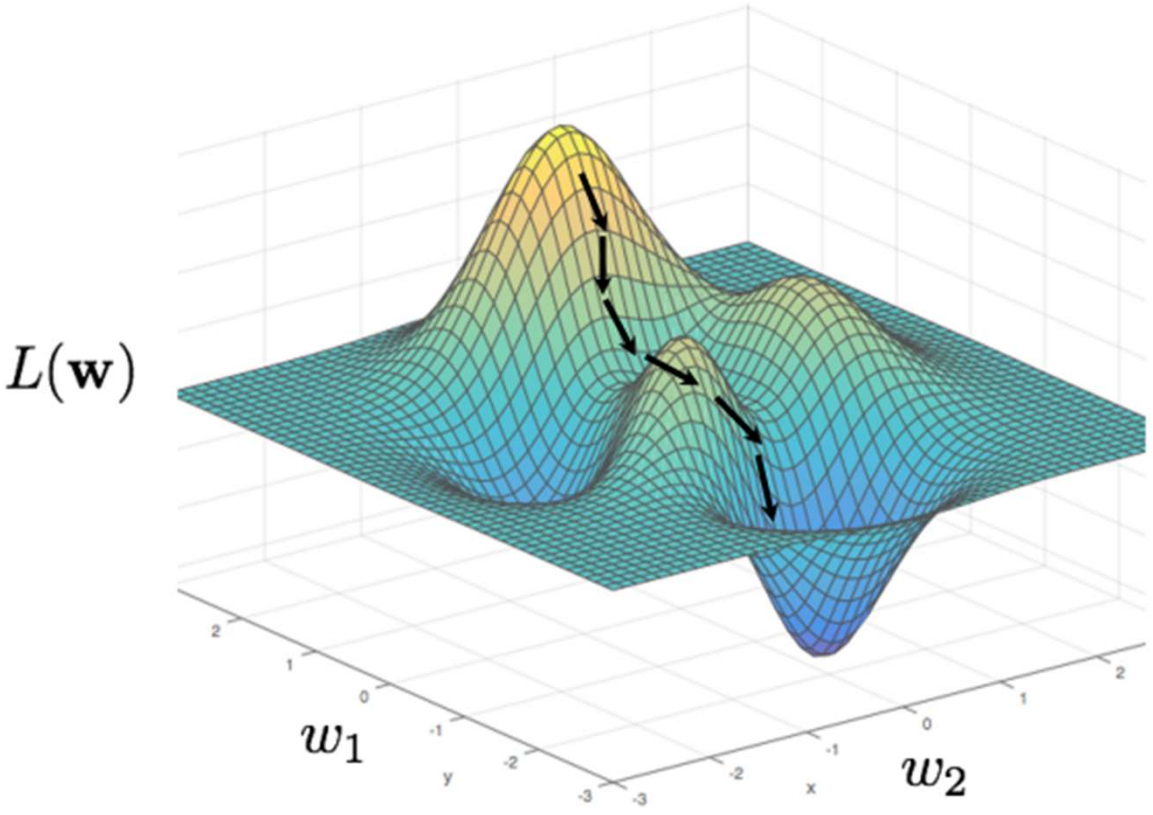
Gradient in Neural Network



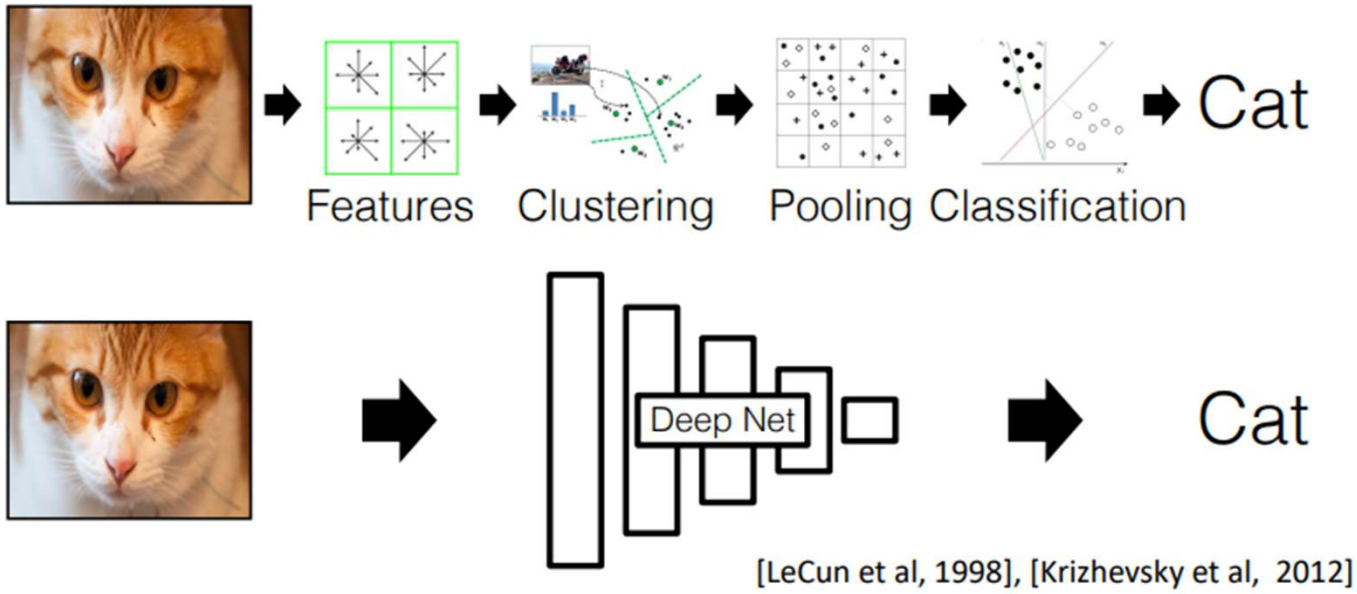
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for **every weight in the network** *using gradients from later layers*

Gradient Descent



Before & After 2012

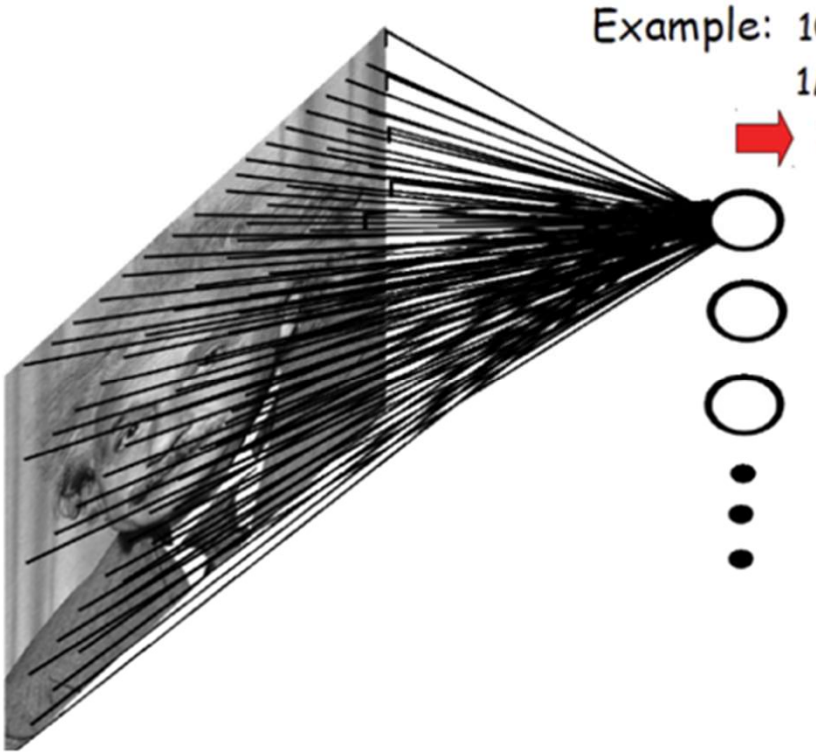


[LeCun et al, 1998], [Krizhevsky et al, 2012]

Thanks to Differentiable Deep Neural Network.

Can we only use Full-Connected Neural Network?

Problem with Fully Connected Layers



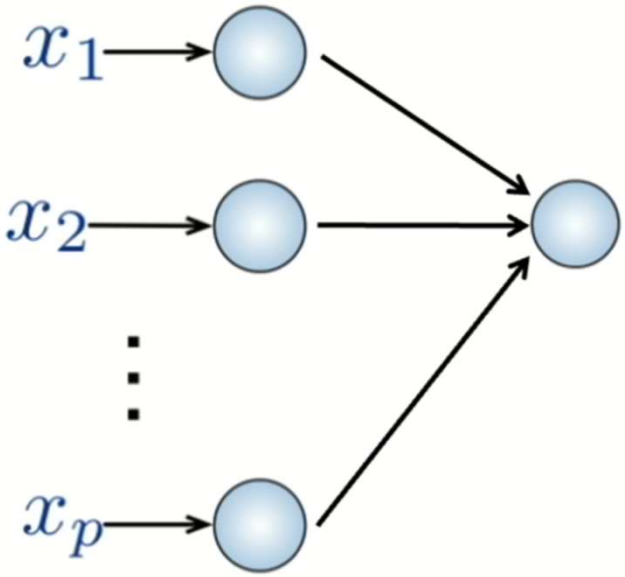
$$n^2 \rightarrow n^2 \text{: } n^4 \text{ parameters}$$

$$10^{12}B \approx 10^9K \approx 10^6M \approx 10^3G \approx 1T$$

Problem with Fully Connected Layers

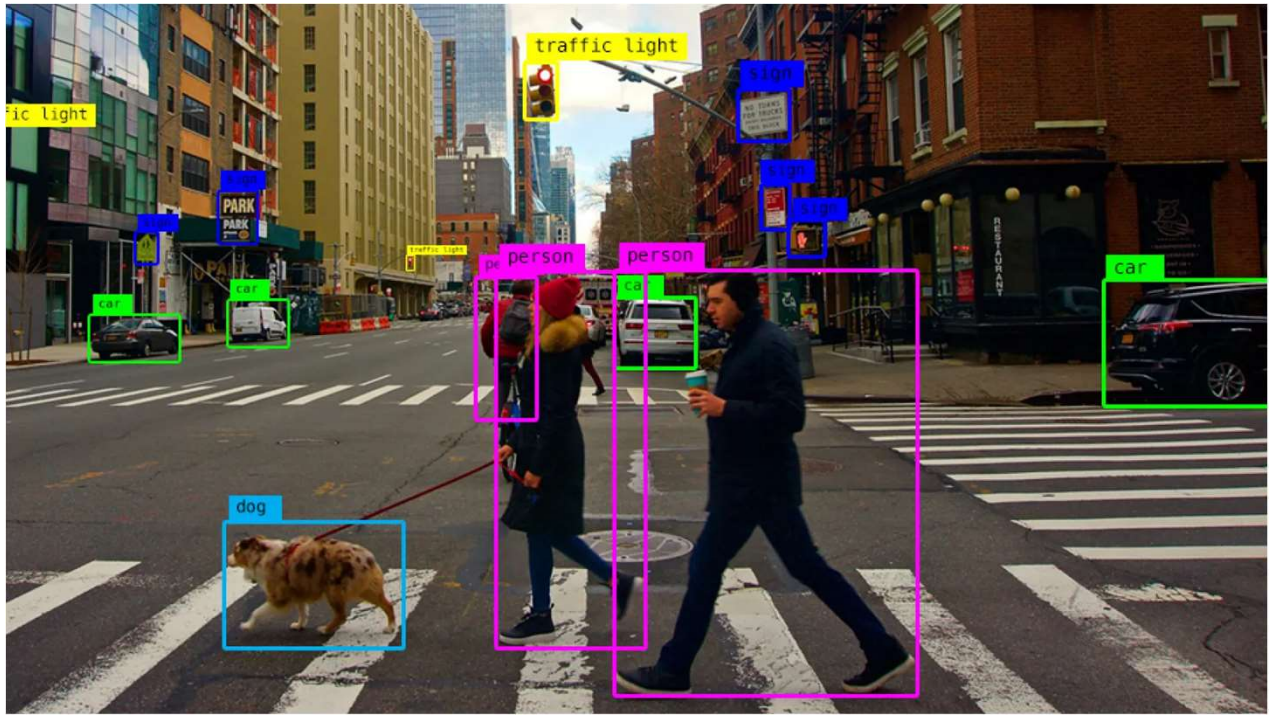
Input:

- 2D image
- Vector of pixel values



- Flatten all pixels to a vector
- Connect all neurons in input layer
- No spatial information

Problem with Fully Connected Layers

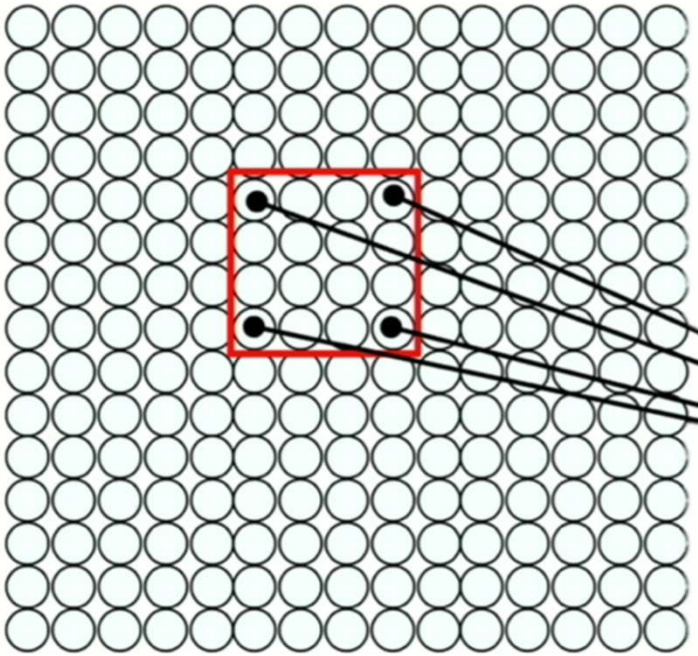


We need spatial (local) information to

efficiently detect objects

Using Spatial Structure

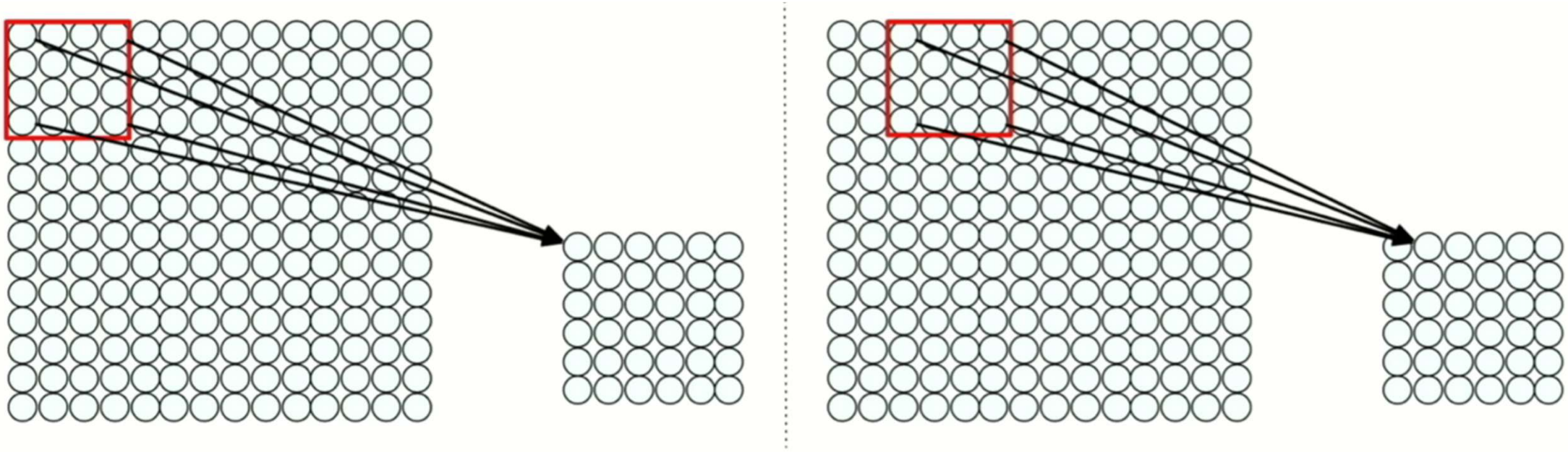
Input: 2D image.
Array of pixel values



Idea: connect patches of input to neurons in hidden layer.

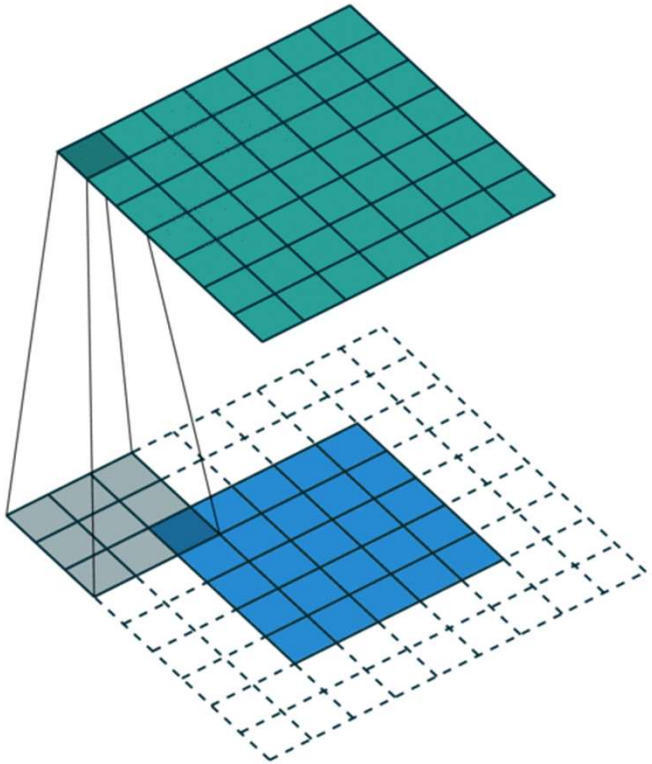
Neuron connected to region of input. Only "sees" these values.

Sliding window to get all info



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.

Previously Learned Image Convolution



1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

CNN for (Simple) Digital Image Processing

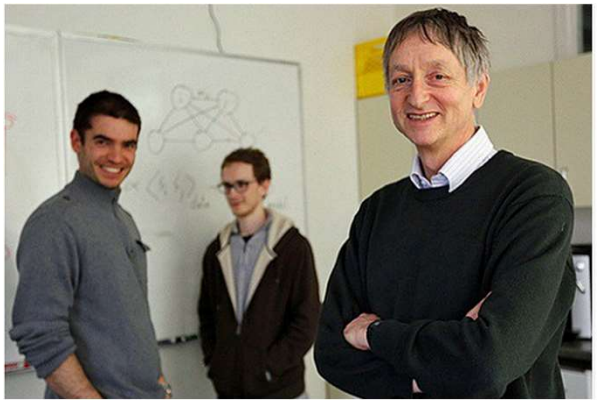
Can we use these networks to do complex tasks?

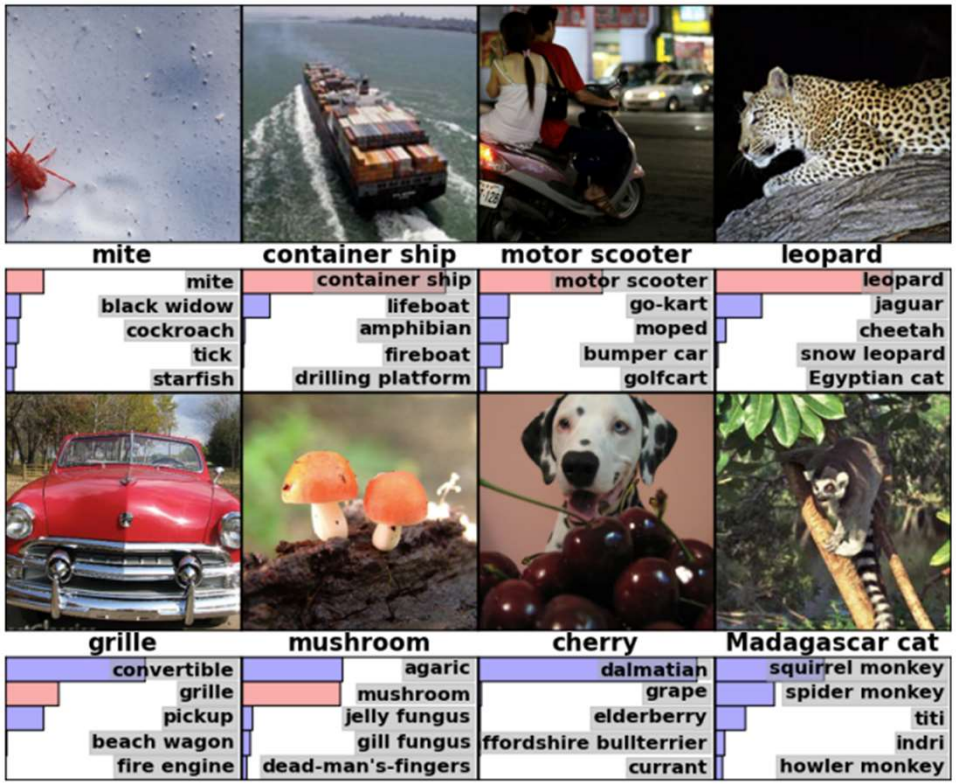
ImageNet Classification with Deep Convolutional Neural Networks

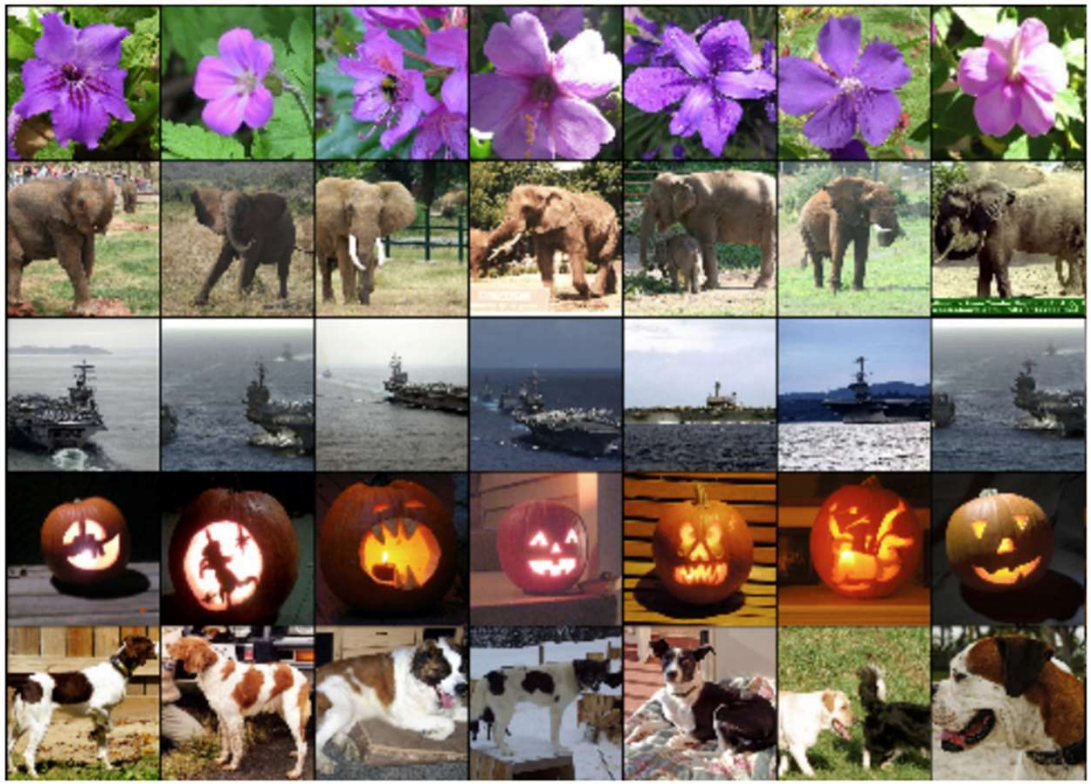
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca







AlexNet Structure

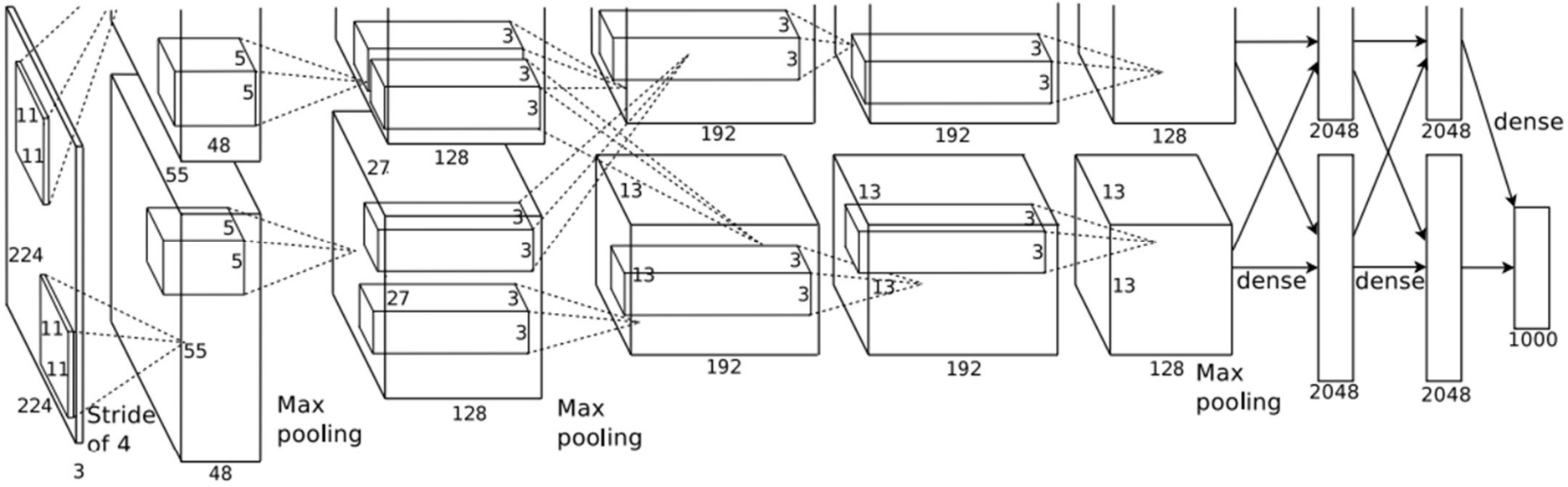
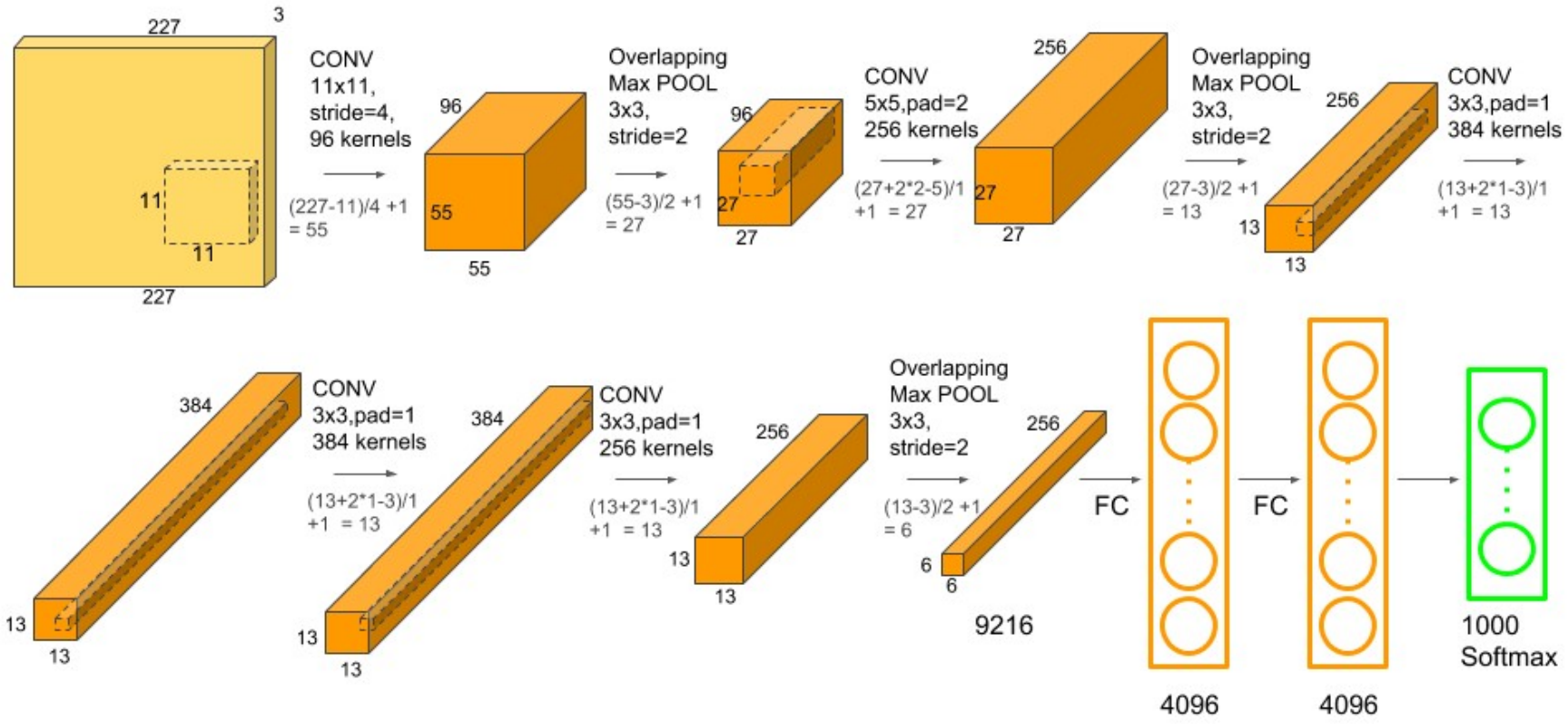
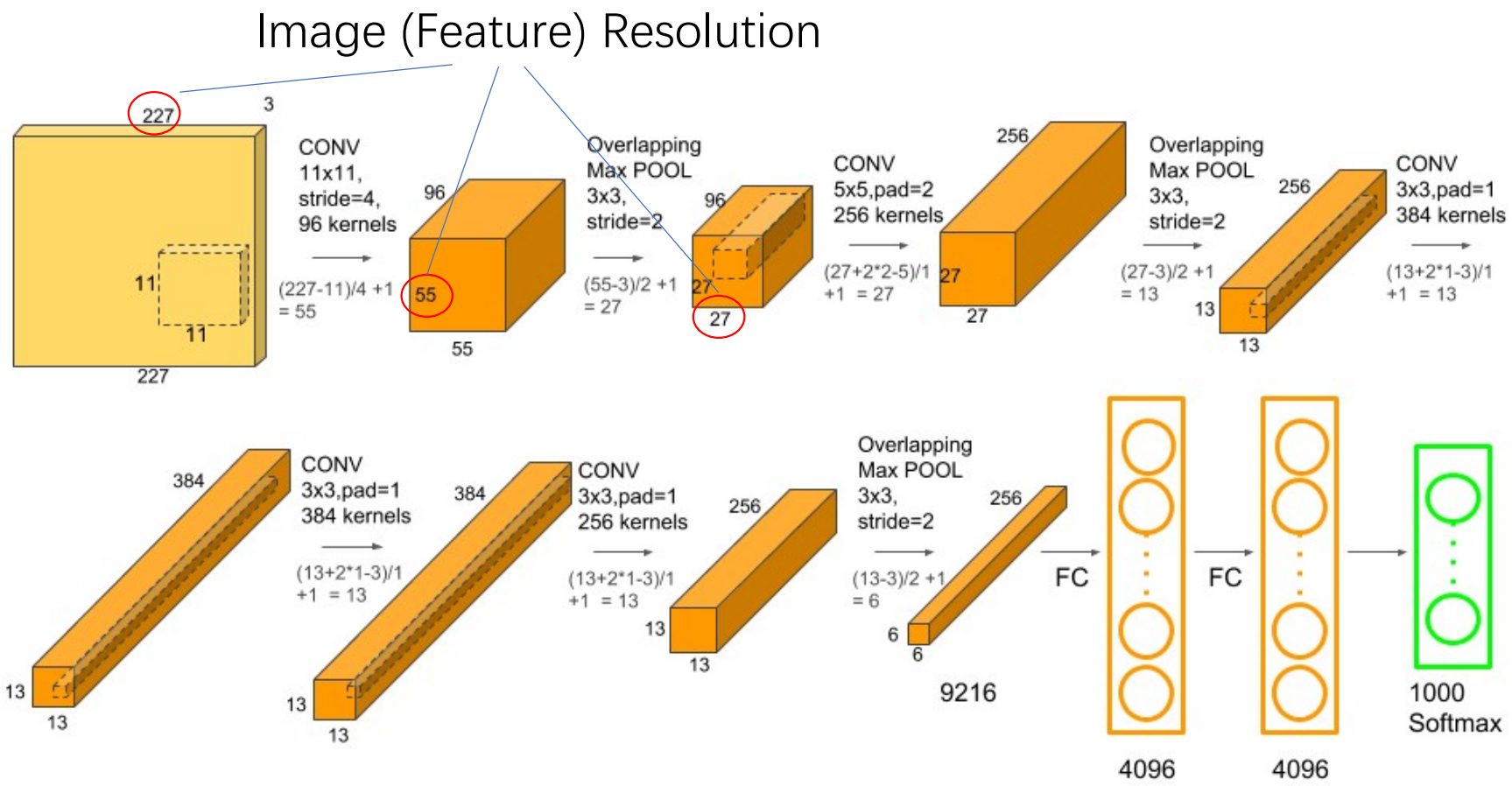


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

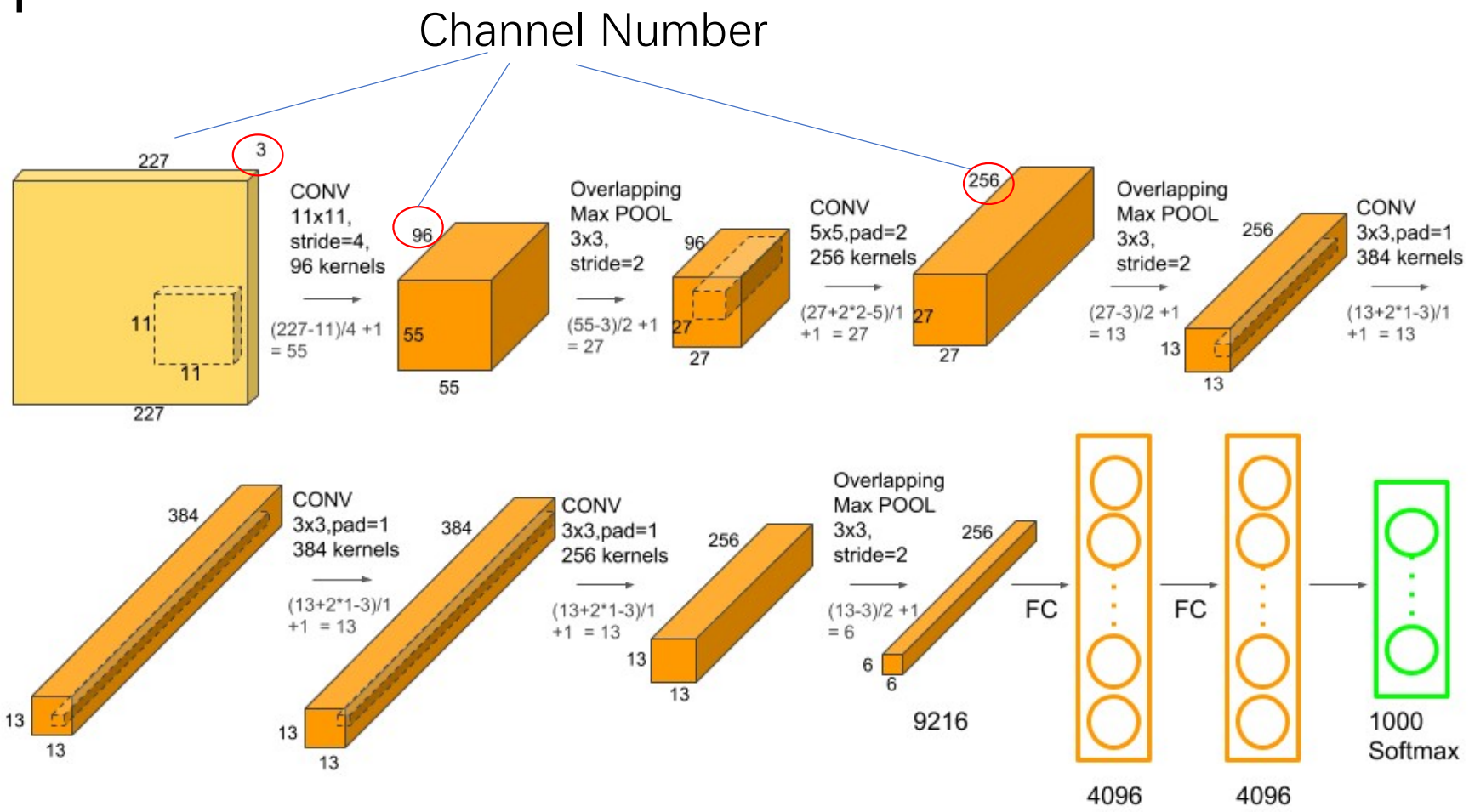
Better Visualization



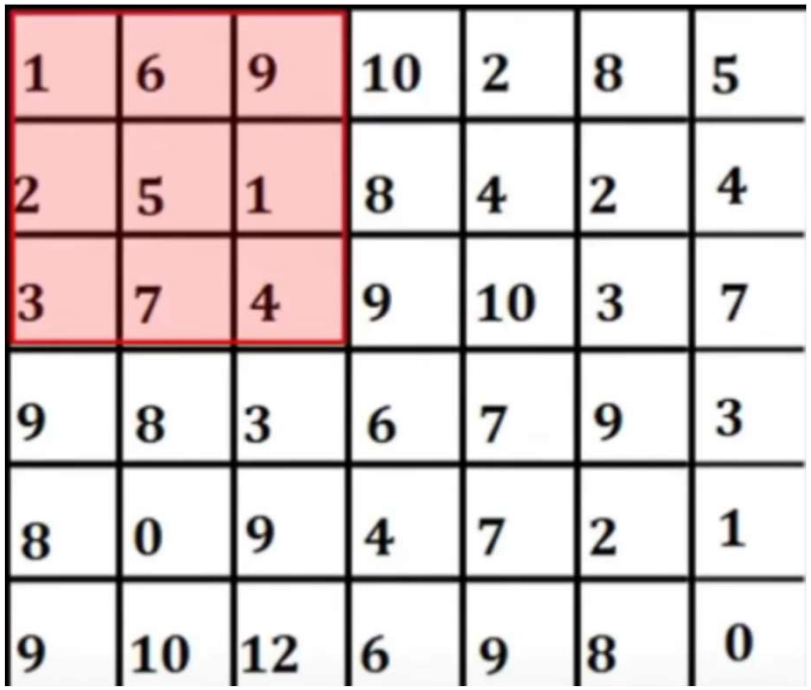
Explanation



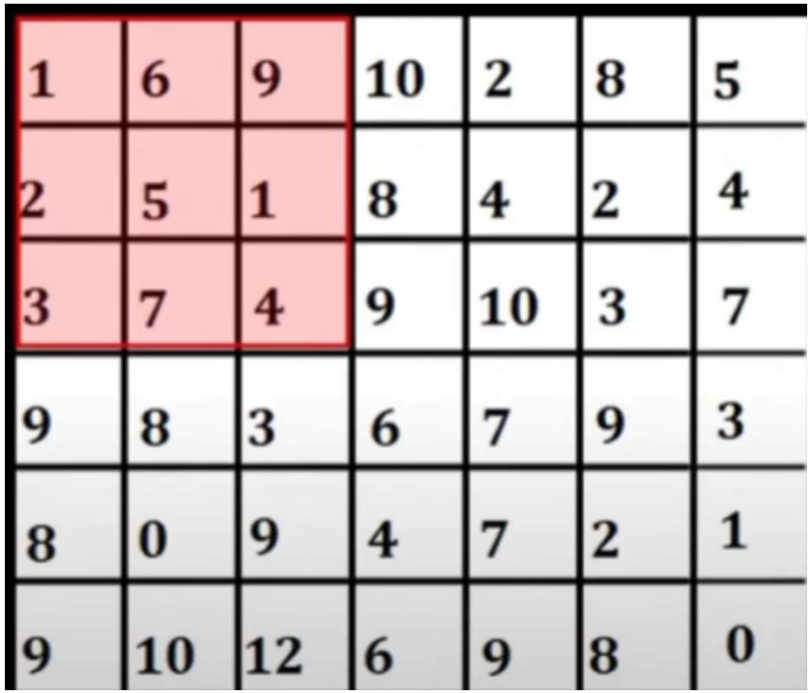
Explanation



Stride



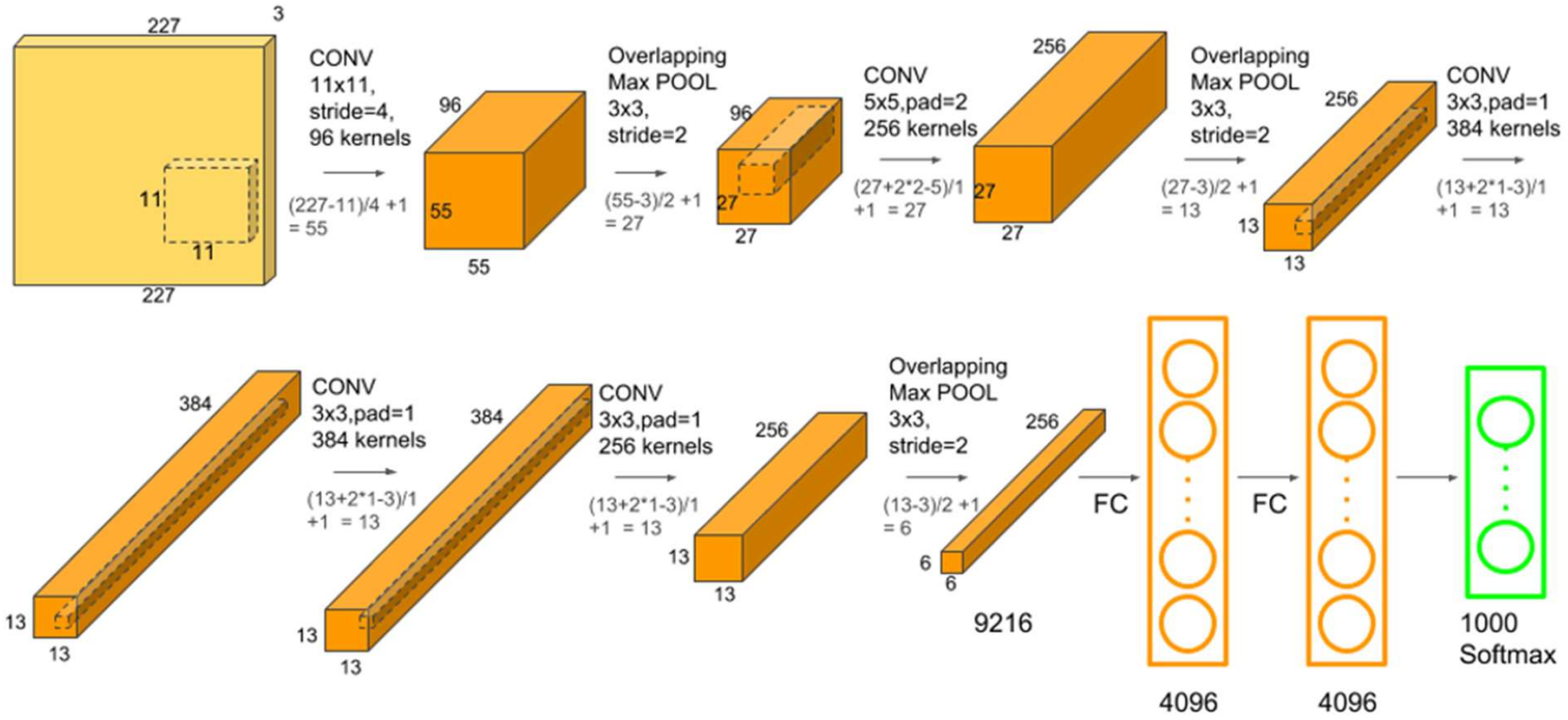
Stride = 1



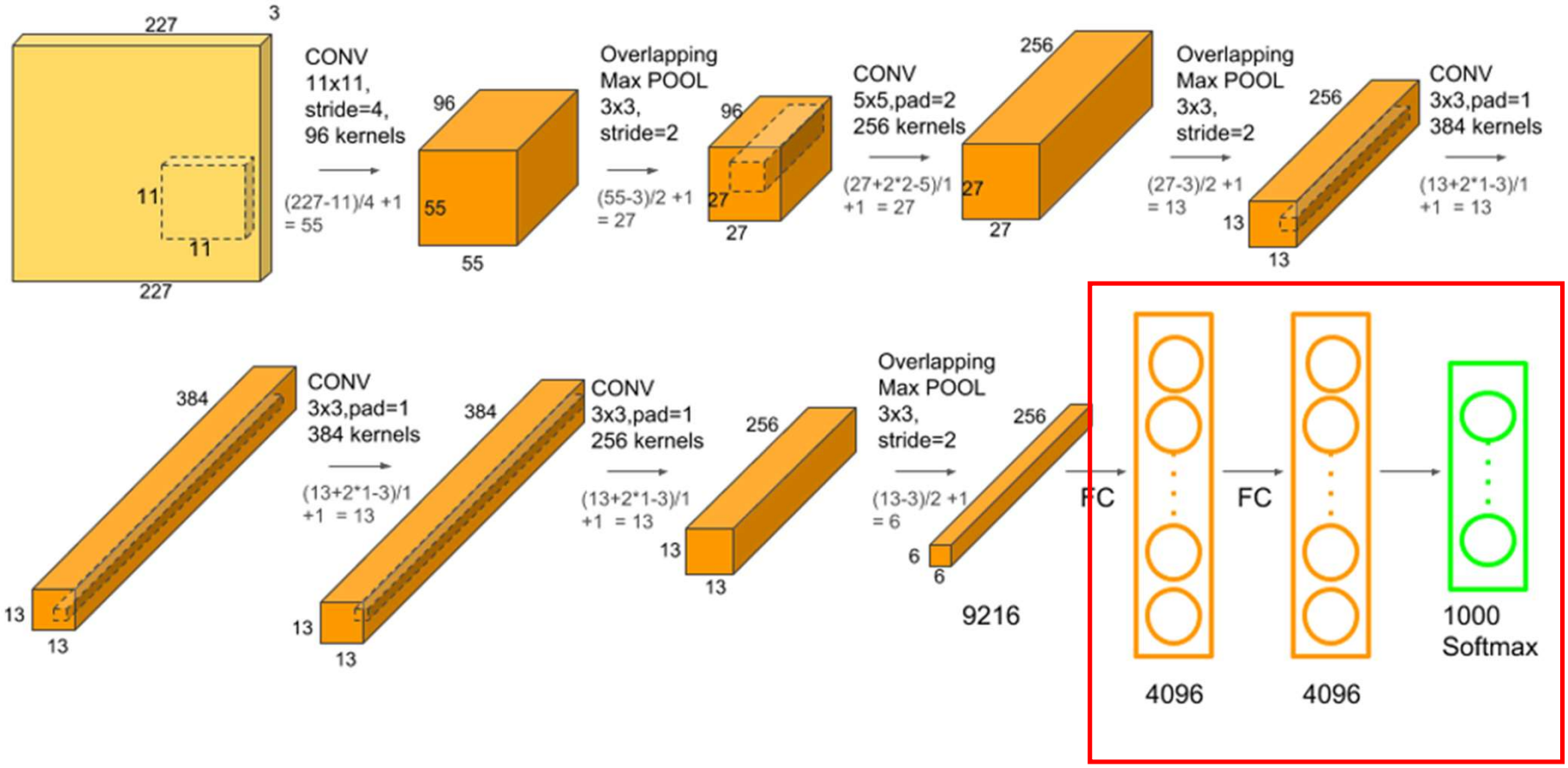
Stride = 2

Down Resolution

$$\text{output size} = \left[\frac{(\text{input size}) + 2*\text{padding} - (\text{kernel size} - 1) - 1}{\text{stride}} + 1 \right]$$

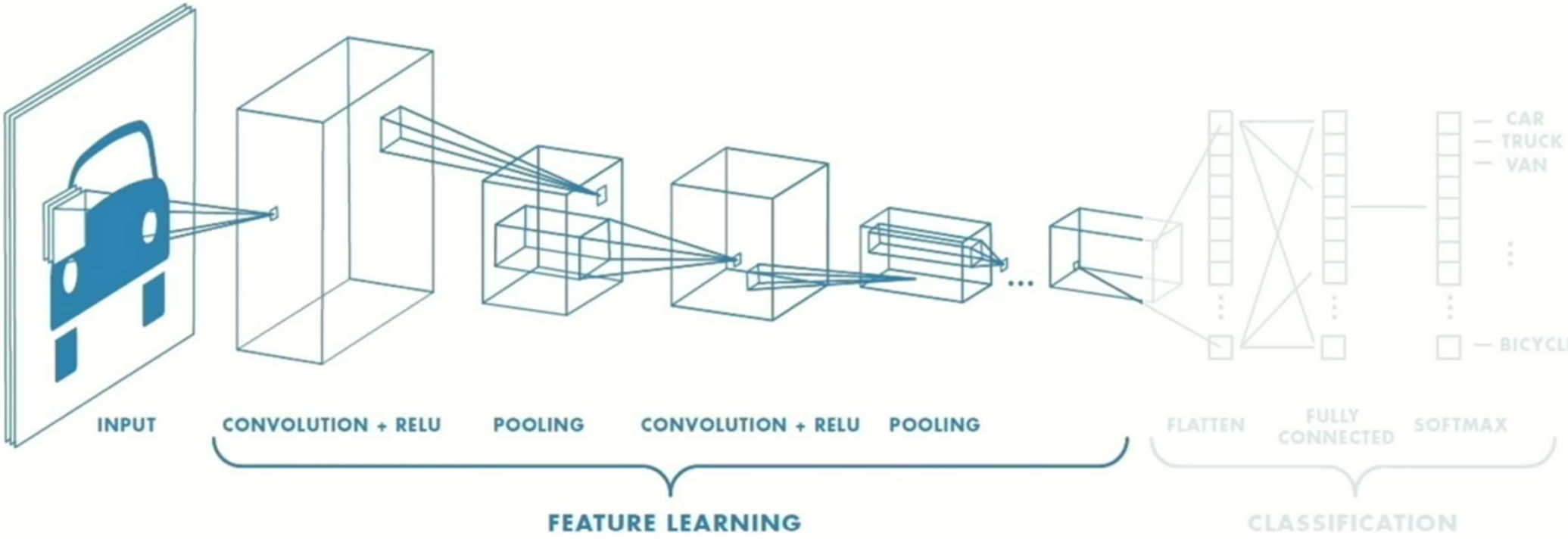


Final Output



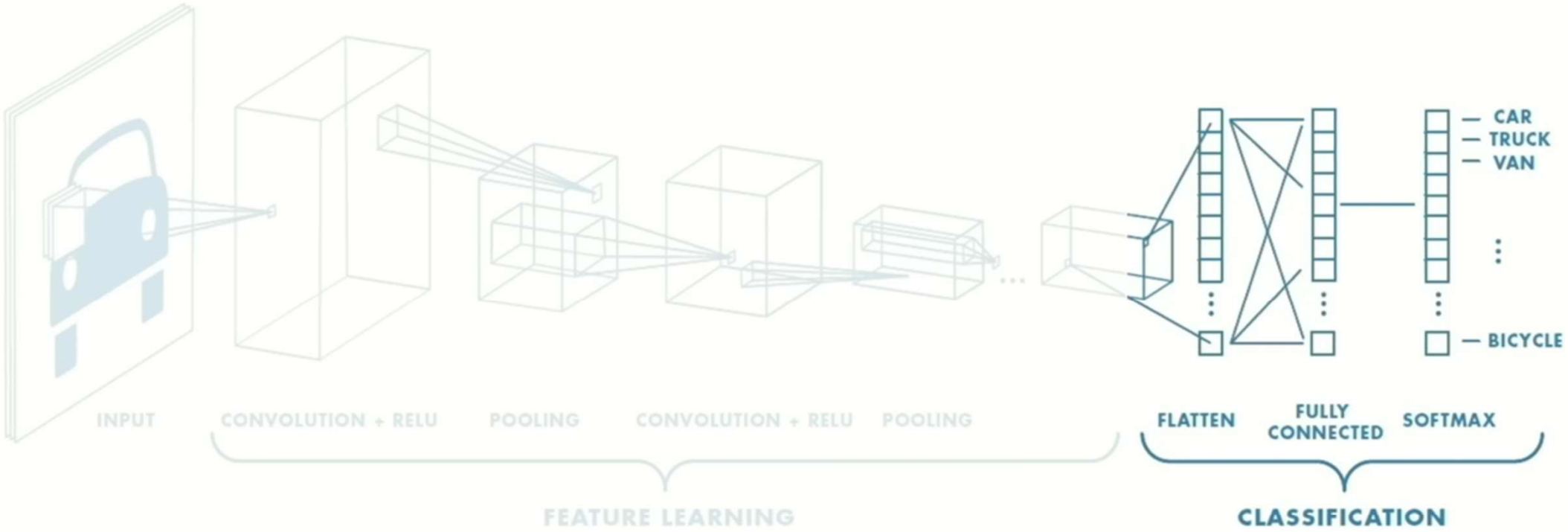
Fully Connected Layer output probabilities belonging to each class

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

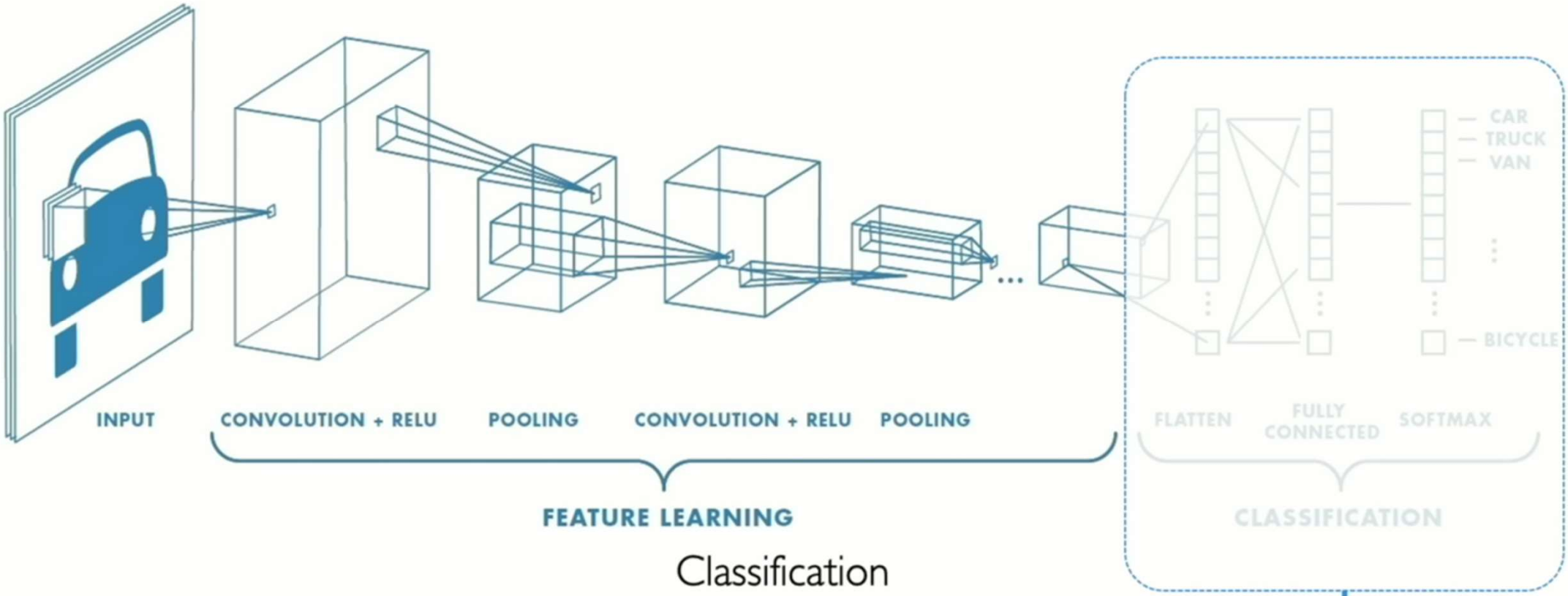
CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

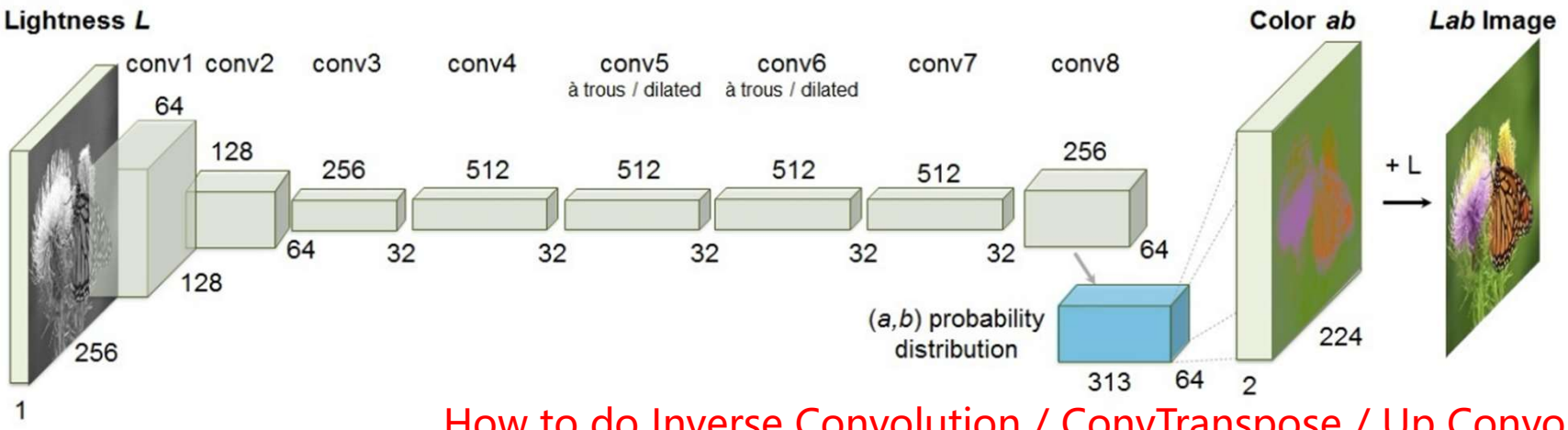
An Architecture for Many Applications



Classification
Object detection
Segmentation
Probabilistic control

How to do Complex DIP tasks with CNNs?

Colorization

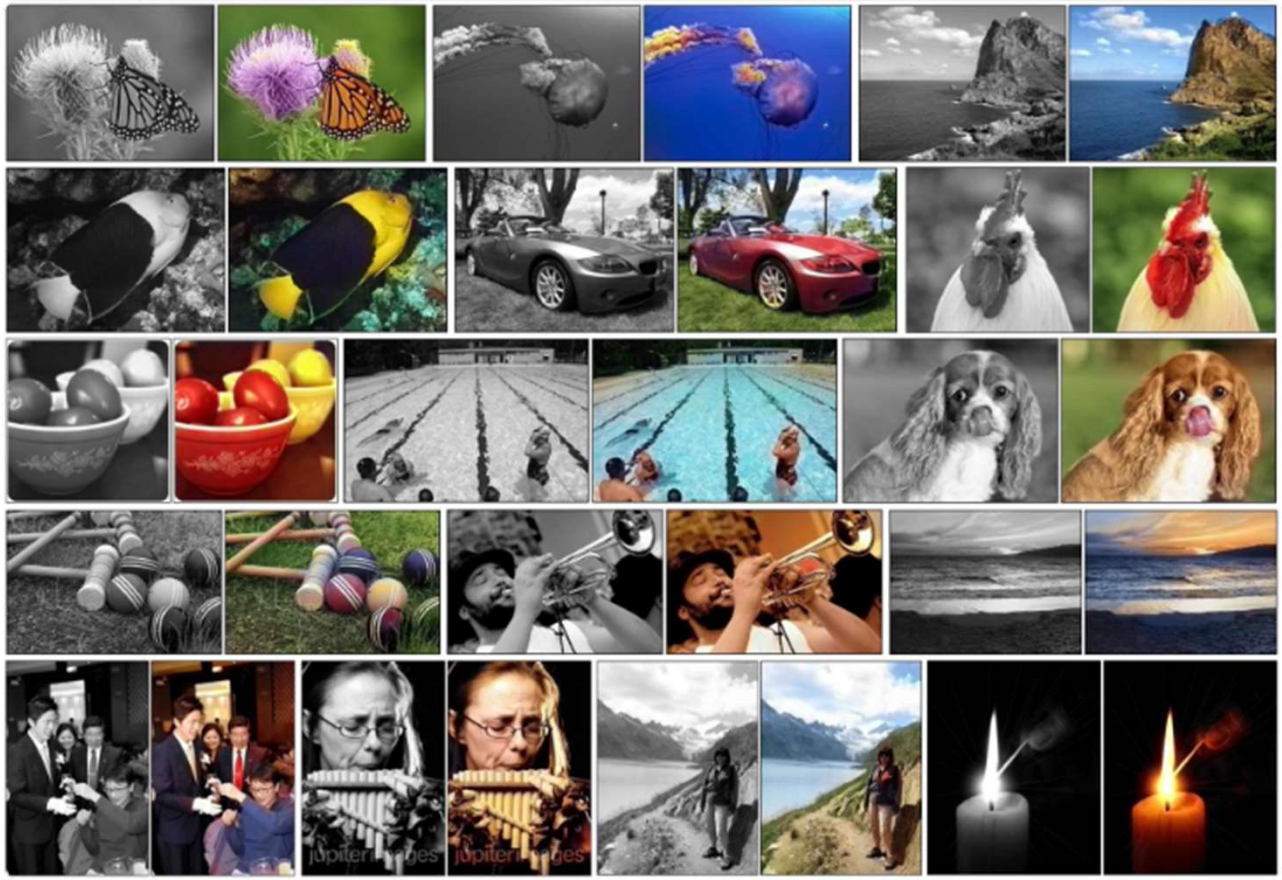


How to do Inverse Convolution / ConvTranspose / Up Convolution?

Fig. 2. Our network architecture. Each `conv` layer refers to a block of 2 or 3 repeated `conv` and `ReLU` layers, followed by a `BatchNorm` [30] layer. The net has no `pool` layers. All changes in resolution are achieved through spatial downsampling or upsampling between `conv` blocks.

Colorful Image Colorization. ICCV 2015.

Optimize with Data



Learn Knowledge

about

what color is proper

from

100000+ colorful images…

Colorful Image Colorization. ICCV 2015.

Semantic Segmentation

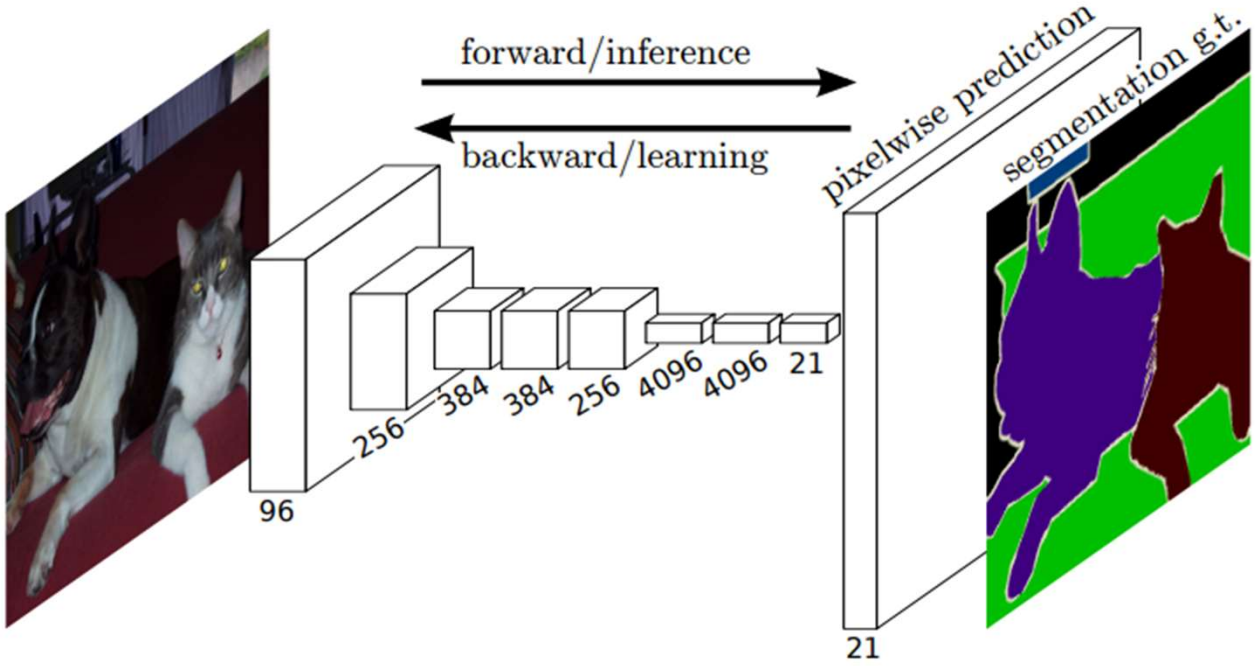


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

Fully Convolutional Networks for Semantic Segmentation. CVPR 2015.

Image Translation

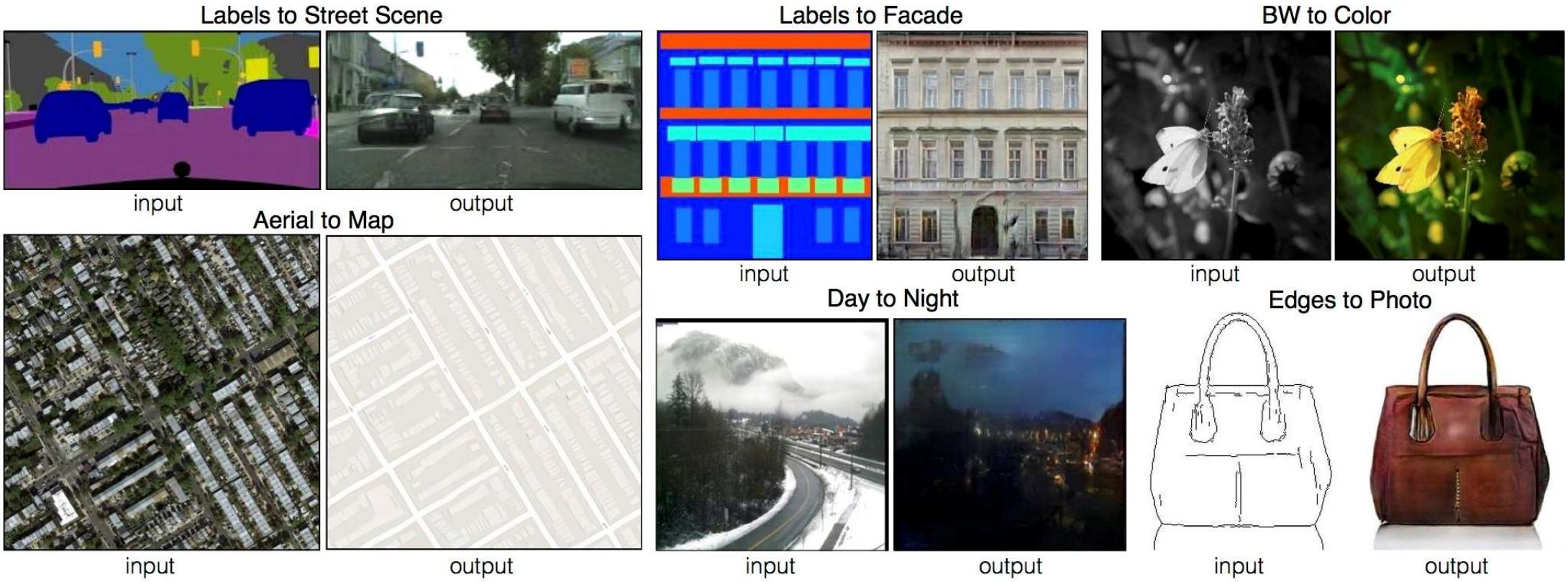


Image-to-Image Translation with Conditional Adversarial Nets. CVPR 2017.

Image Translation

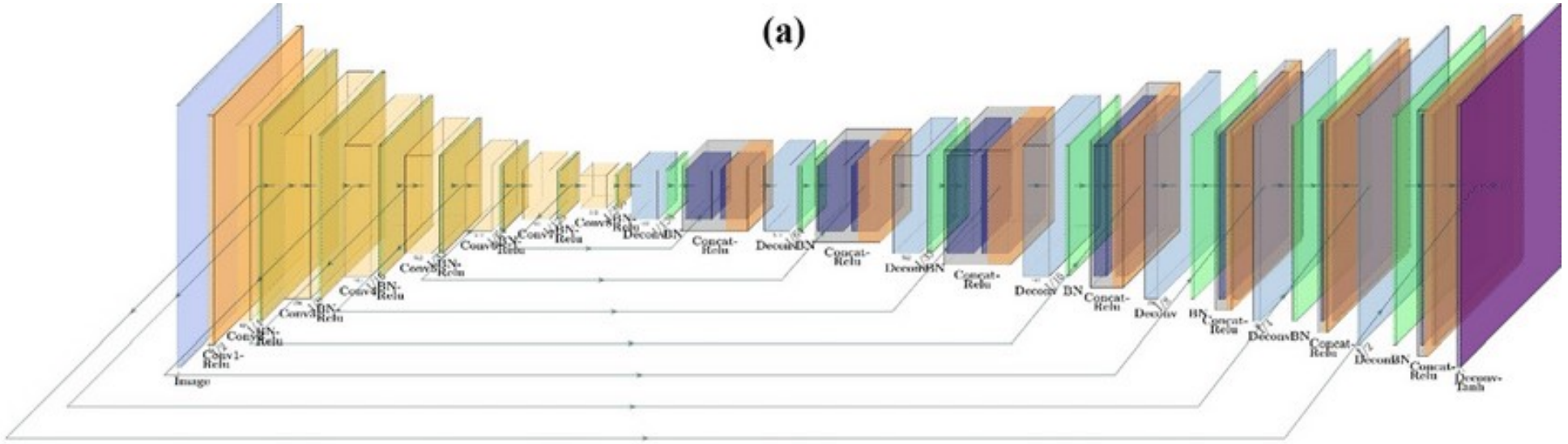


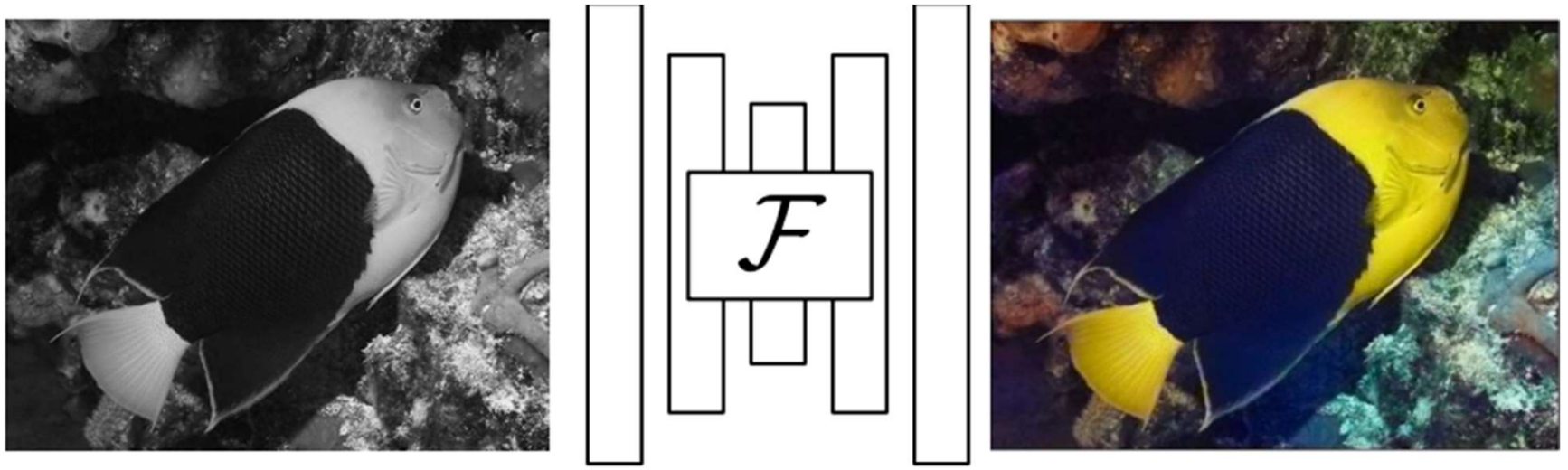
Image-to-Image Translation with Conditional Adversarial Nets. CVPR 2017.

Course Summary

一些复杂的图像 处理/生成 任务目前只能用深度学习

深度学习是一些运算的复合：Fully-Connected & Convolution

Deep Learning适合各种任务的关键在于强大的表达能力和基于可微的方便求解



中国科学技术大学

University of Science and Technology of China

谢谢观看!