MATH466/MATH766
Math of machine learning
01/27 Lecture 5 Fully Connected Neural Networks and Convolutional Neural Networks

References:

- Ch11 of The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani and Jerome Friedman
- Ch 6, 7 and 9 of Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville
-

Todays contents:

- Fully connected neural networks (feedforward neural network / multilayer perceptron) basics
- fitting neural networks: back-propagation
- common issues
- universal approximation theorem
- convolution and convolutional neural networks

Important concepts:

- back-propagation

Recommend reading:

- Ch 6.3 and 6.6 of Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville
- preface of Ch 7, Ch7.1, Ch7.8 and Ch 7.12 of Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville
- Ch 9.10 and 9.11 of Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville

Warm up: 1. $f, g : \mathbb{R} \to \mathbb{R}$ differentiable

$y = f(\theta)$, $J = g(y)$

$\frac{dJ}{d\theta} =$ _____

2. $f : \mathbb{R} \to \mathbb{R}^d$, $g : \mathbb{R}^d \to \mathbb{R}$, differentiable

$y = f(\theta)$, $J = g(y)$

$\frac{dJ}{d\theta} =$ _____

Goal of supervised learning: learn some function f

Main approaches in previous lectures: parameterize f with $\theta \in \mathbb{R}^{d_\theta}$

$f(x; \theta)$

learn parameters $\theta \in \mathbb{R}^{d_\theta}$

Difficulty: find a "good" parameterization

- has desired property
- has enough representation power
- ... ...

Issue of previous parameterizations: linearity

Want: a simple but general parameterization

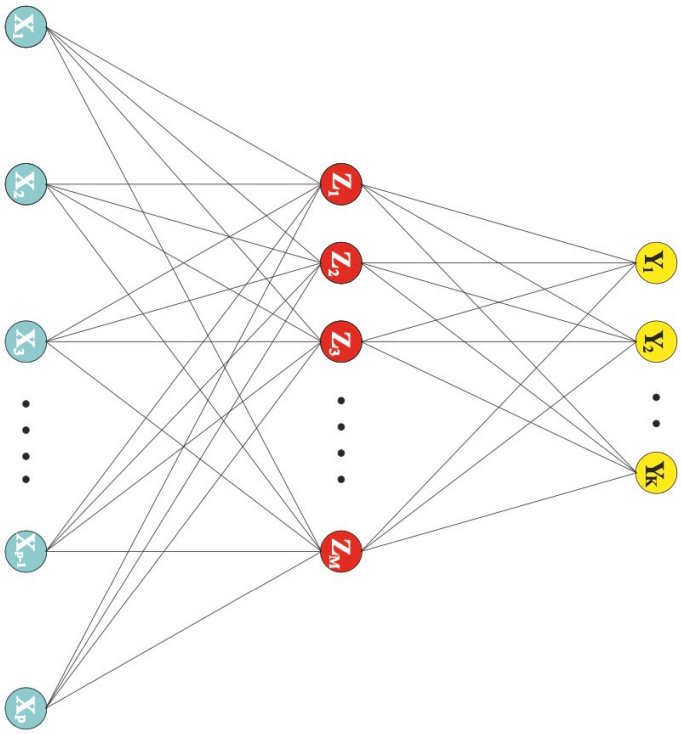
FCNN (FNN, MLP): linear ∘ linear ∘ linear → linear ×

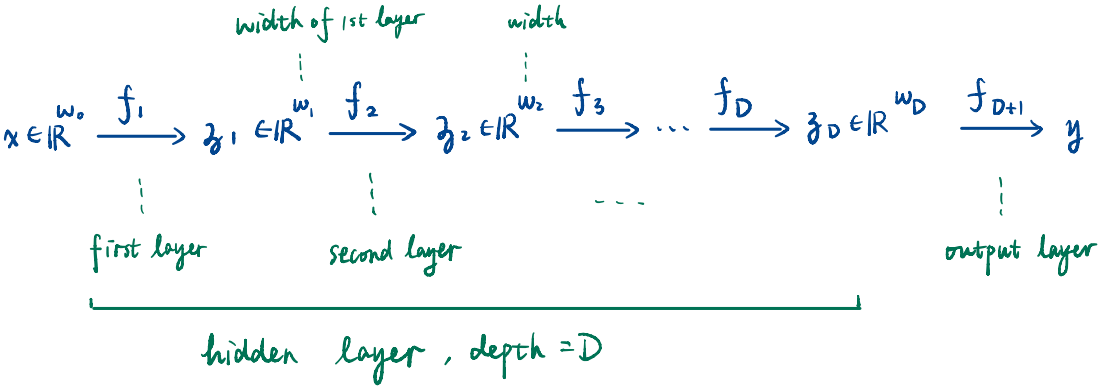
→ nonlinear ! ?

nonlinear, nonlinear

CNN: simplify

1. Fully Connected Neural Network





- $f_i = \sigma \circ h_i$ (σ entry-wise composition)

  $h_i : z_{i-1} \mapsto W_i z_{i-1} + b_i$ affine function

  weight $W_i \in \mathbb{R}^{w_i \times w_{i-1}}$

  bias $b_i \in \mathbb{R}^{w_i}$

  $\sigma : \mathbb{R} \to \mathbb{R}$ nonlinear

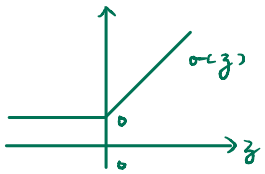
- $f(\cdot, \theta) = f_{D+1} \circ h_D \circ \sigma \circ h_{D-1} \circ \cdots \circ \sigma \circ h_1$

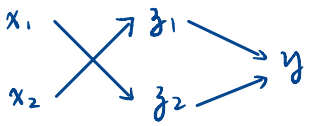
  w.l par (W_D, b_D) (W_{D-1}, b_{D-1}) (W_1, b_1)

$\theta = \{ (W_i, b_i) \mid i = 1, \cdots, D \}$ trainable parameters

D, $\{ w_i \mid i = 1, \cdots, D \}$, σ hyperparameters (pick before training)

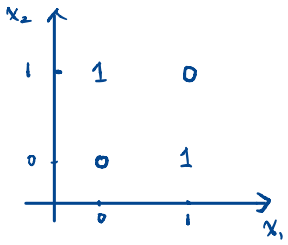
e.g. ($d=2$) $D=1$, $w_1 = 2$, $\sigma(z) = \max\{z, 0\}$.





$y(x; W, b, \underline{w}, c) = \underline{w}^T \max\{0, Wx + b\} + c$

How this can be helpful? Consider XOR problem



x	y
(0, 0)	0
(0, 1)	1
(1, 0)	0
(1, 1)	1

no linear function can learn this w/ only features x_1, x_2.
(but add feature $x_1 x_2$ can help)

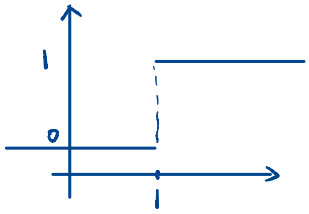
Consider the above network w/

$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $b = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$, $\underline{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, $c = 0$ (check!)

Nonlinear activation function plays an important role

- Some commonly used activation function

① step function (used in traditional perceptron) $\sigma(z) = 1_{\{z > 0\}}$



② ReLU (rectified linear unit) $\sigma(z) = \max\{z, 0\}$

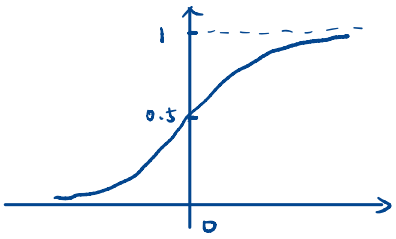
gradient = 0 when $z < 0$, not good for training

Leaky ReLU $\sigma(z) = \max\{0.1z, z\}$

ELU $\sigma(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}$

③

Logistic function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$



s-shape

Sigmoid function

hyperbolic tangent tanh function $\sigma(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$ (shifted and scaled logistic)

2. Universal Approximation Theorem

NNs have strong representation power to approximate a large class of functions.

There are many different versions of universal approximation theorem.

[Cybenko, 1989] Consider $I_n = [0,1]^n$ and $C(I_n)$ being the space of continuous functions on I_n

Let σ be any continuous discriminatory function (e.g. logistic).

Then given any $f \in C(I_n)$ and $\varepsilon > 0$, there exist a finite sums of the form

$$G(x) = \sum_{j=1}^{w} c_j \sigma(a_j^T x + b_j)$$

such that $|G(x) - f(x)| < \varepsilon$ for any $x \in I_n$.

In plain language, 1 layer NN w/ arbitrary width
with sigmoid activation function
can approximate any continuous functions on n-d cube
to arbitrary precision.

There are also other versions w/ different
width constraint, depth constraint, activation function, etc ...

And there are also studies on given a function and NN structure,
what is the depth and width requirement to approximate this.

Strong representation power (large model capacity)

Good! Good?

Recall U-shape curve... How to avoid overfitting?

Other issues?

3. Training a Neural Network

- Loss function $J(\theta)$ $\longrightarrow$ $\min_{\theta} J(\theta)$ (*)

e.g. least square regression. data $\{(x_i, y_i)\}_{i=1}^n$, $y_i \in \mathbb{R}^{d_y}$

$f_{D+1} : z_D \mapsto W_{D+1} z_D + b_{D+1}$, $W_{D+1} \in \mathbb{R}^{d_y \times w_D}$, $b_{D+1} \in \mathbb{R}^{d_y}$

$f = f_{D+1} \circ f_D \circ \cdots \circ f_1$, $\theta = \{$ $\}$

prediction $f(x;\theta) \in \mathbb{R}^{d_y}$

empirical risk $J(\theta) = \frac{1}{2} \sum_{i=1}^n \| y_i - f(x_i;\theta) \|^2$

e.g. k-class classification. data $\{(x_i, y_i)\}_{i=1}^n$, $y_i \in \{1, 2, \cdots, k\}$

step 1. NN approximation:

$z_D \in \mathbb{R}^K$, $f_{D+1} : z_D \mapsto \underline{y} \in \mathbb{R}^K$, $(\underline{y})_k = \frac{e^{z_k}}{\sum_{j=1}^k e^{z_j}}$ (softmax)

$f = f_{D+1} \circ f_D \circ \cdots \circ f_1$, $\theta = \{$ $\}$

prediction $f(x;\theta) \in \Delta^k$, $\Delta^k = \{ p \in \mathbb{R}^k : p_j \geq 0, \sum_{j=1}^k p_j = 1 \}$

the k-th entry $(f(x;\theta))_k$ predicts $\mathbb{P}(y = k \mid x)$

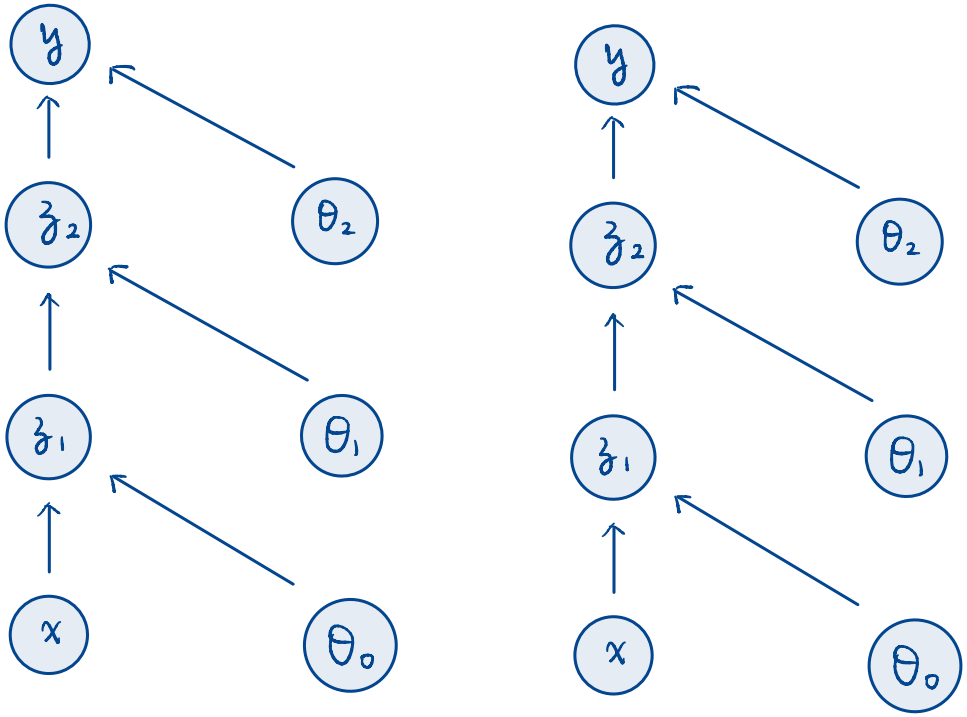
step 2 objective / loss

① Regression p.o.v. convert data label y_i to a prob. vector $\underline{y}_i$

② MLE p.o.v

- Back - propagation (to solve (*))

key: evaluate $\nabla_\theta J(\theta)$ (assume that J is differentiable)



Example $x \in \mathbb{R}^d$, $y \in \mathbb{R}$. $f(x;\theta) = \sum_{j=1}^{w} c_j \sigma(a_j^T x + b_j)$, σ differentiable

$$\theta = \{(a_j, b_j, c_j) \in \mathbb{R}^{d+1+1} \mid j = 1, \dots, w\}$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i;\theta))^2$$

$$\frac{\partial J}{\partial c_j} =$$

$$\frac{\partial J}{\partial b_j} =$$

$$\frac{\partial J}{\partial a_j} =$$

4. Convolutional Neural Network

4.1 Convolution and Sparse Connectivity

- Continuous : $f : \mathbb{R}^d \to \mathbb{R}$, $k : \mathbb{R}^d \to \mathbb{R}$

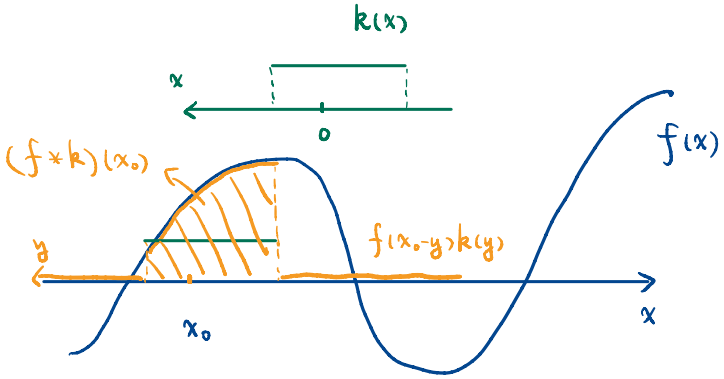
feature map $\to (f * k)(x) := \int_{\mathbb{R}^d} f(y)\, k(x-y)\, dy$ (assume $< +\infty$)

(f: input, k: kernel)

$= \int_{\mathbb{R}^d} f(x-y)\, k(y)\, dy$

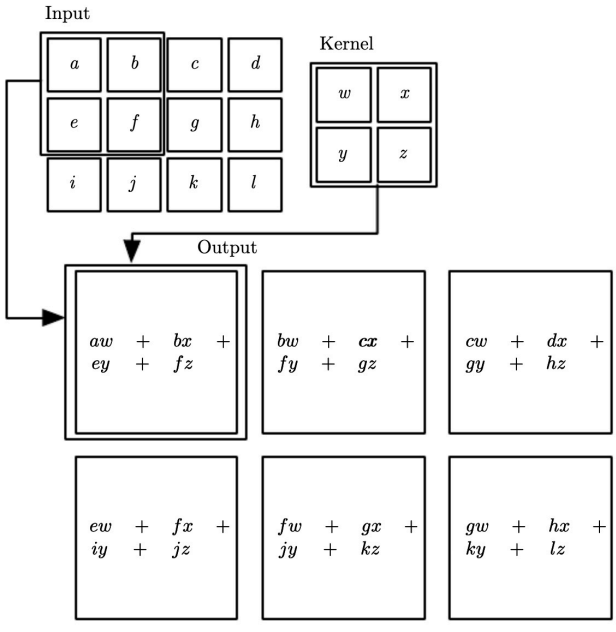
e.g. assume $\exists\, R > 0$ s.t. if $|k(y)| > 0$, then $\|y\|_2 \leq R$

(k has compact support)



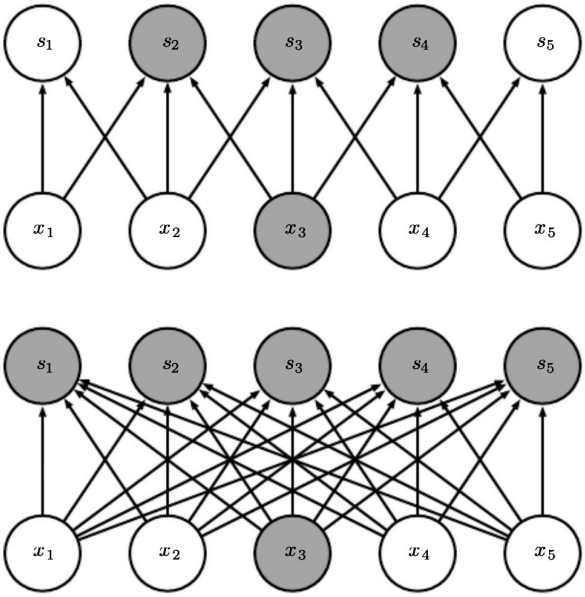
- Discrete (without flipping the kernel, 2D for example)

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



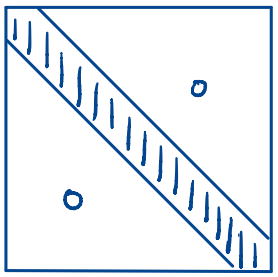
- Sparse connectivity and parameter sharing

$W^T x + b$



CNN

W, sparse, non-zero entries of each column are the same



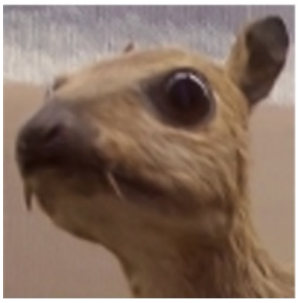
FCNN.

W, dense, no special structure

convolutional layers improve efficiency

What about effectiveness?

① Feature extraction by convolution in classical image processing



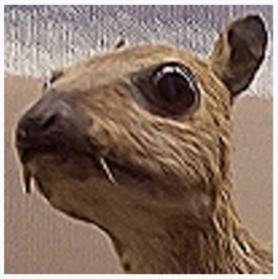
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

box blur

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

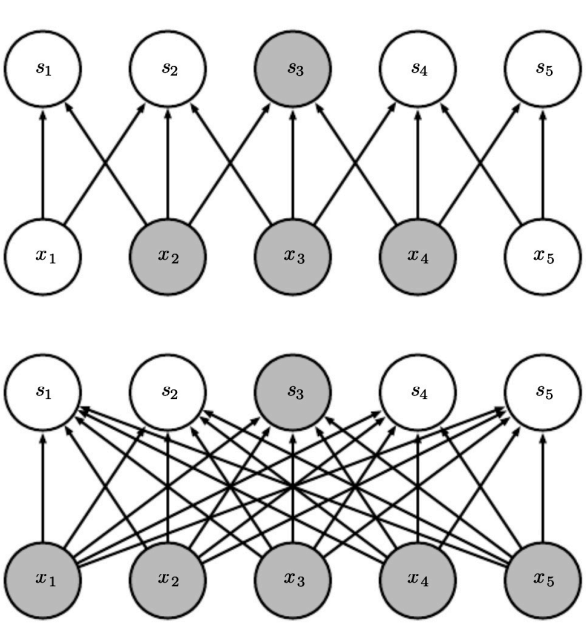
edge detection

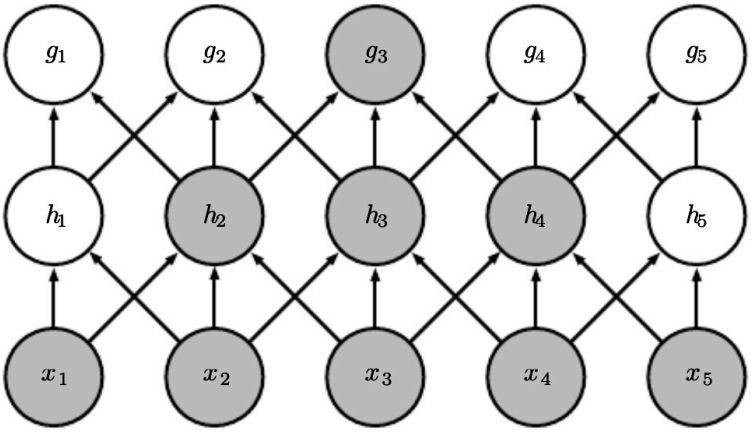
$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$



sharpen

② deep convolution can describe complicated interactions from simple sparse interacting building blocks

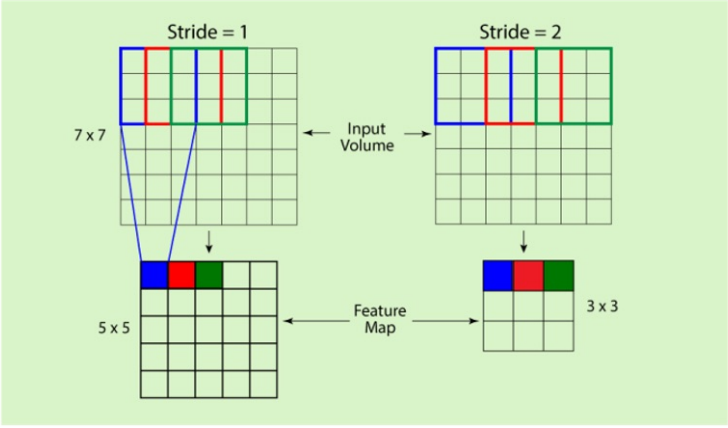




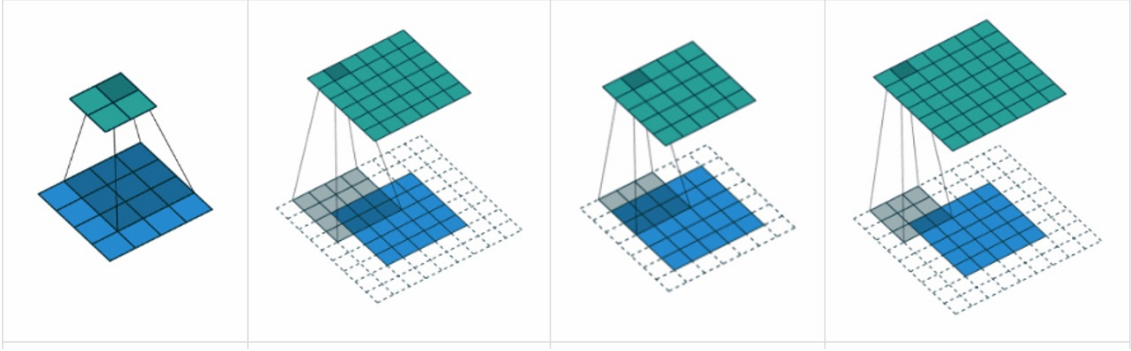
4.2. convolutional layer = convolution + activation + pooling

- other terms in convolution

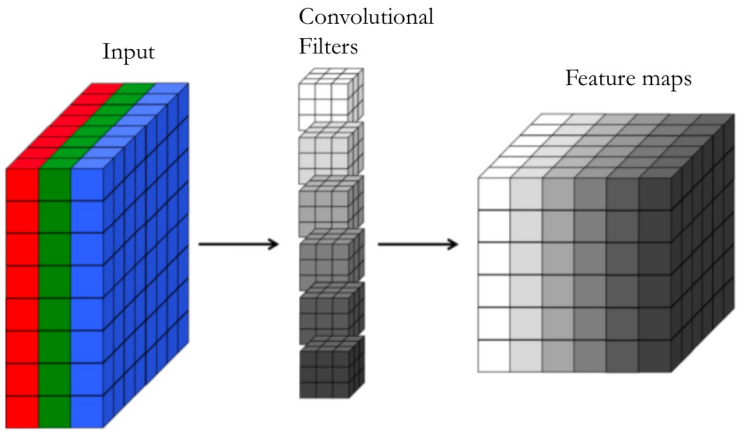
stride



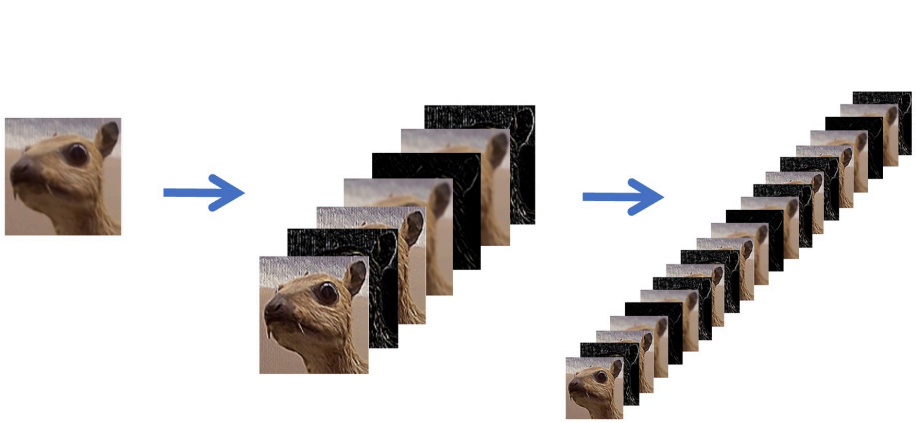
padding

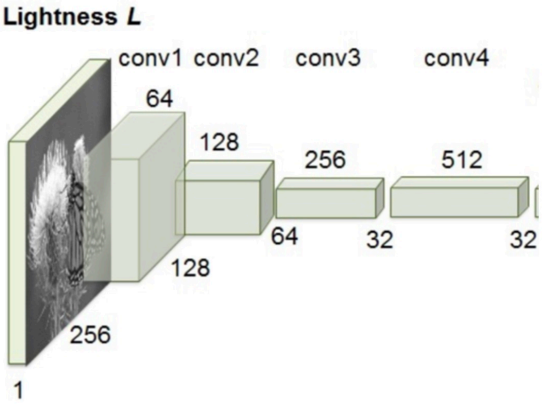


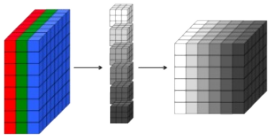
channels



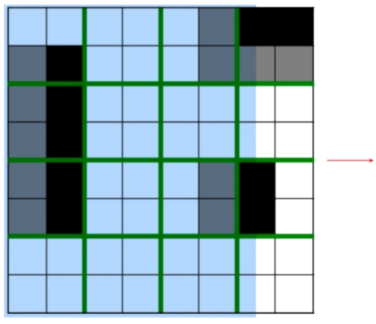
- pooling : spatial down-sampling







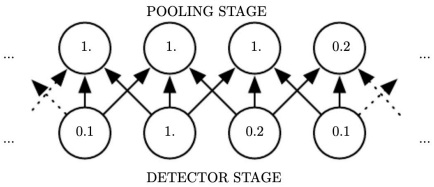
- # of channels increases to retrieve multiple features
- Dimension quickly grows if spatial resolution stays the same

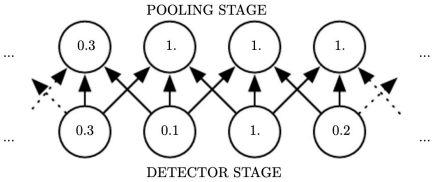


avg-pooling or max-pooling

pooling reduces computational burden

In addition, pooling introduces invariance





Remark 1. A typical structure of CNN

VGGNet

convolutional layers

fully-connected layers

Remark 2. Fully-connected layers and convolutional layers only feed information forward in the network.

A different class of NN layer: recurrent neural network

like $x \longrightarrow z \circlearrowleft f$

include feedback connection.

Remark 3. A recently popular structure: attention, transformer