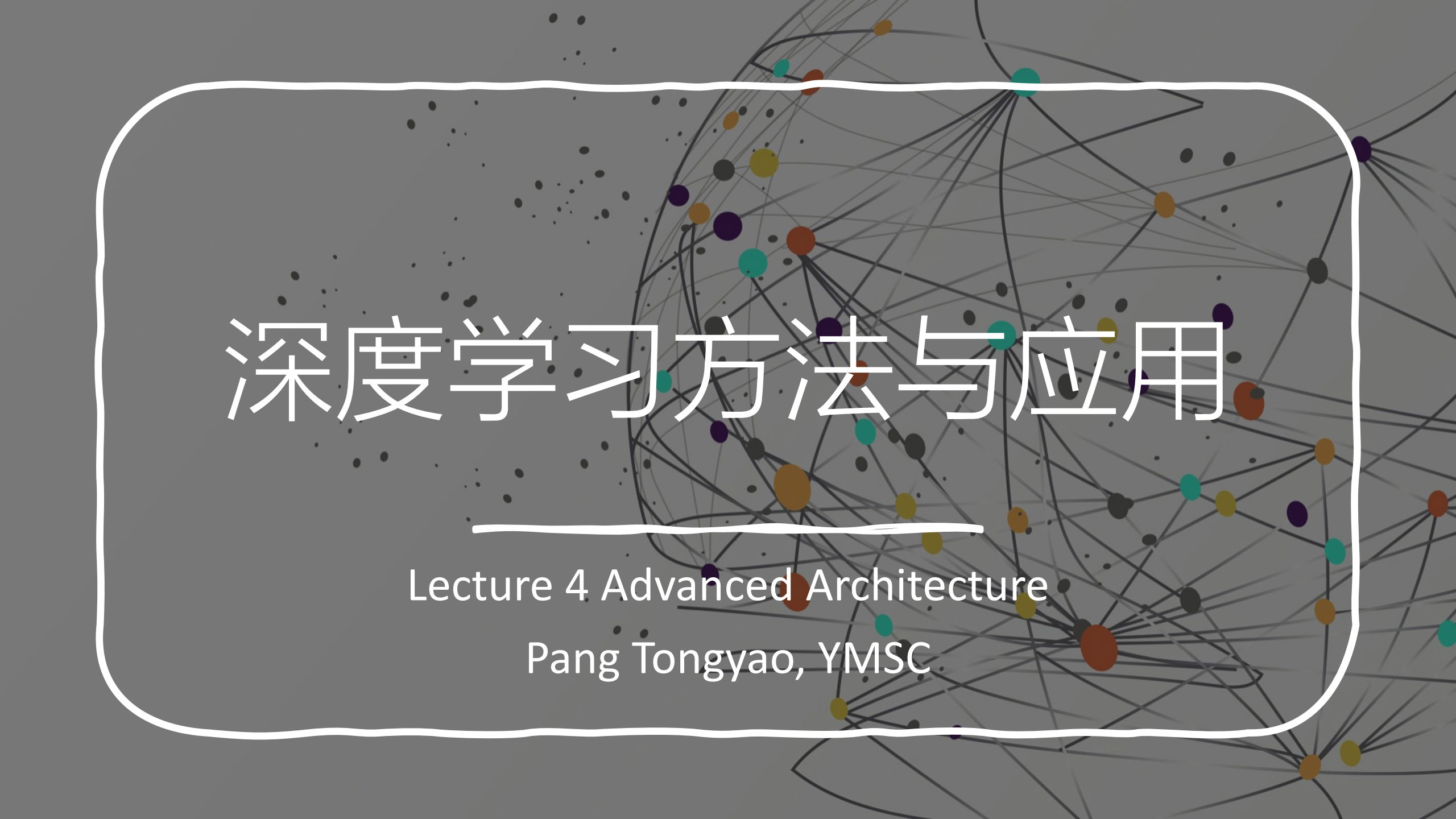


深度学习方法与应用

Lecture 4 Advanced Architecture

Pang Tongyao, YMSC

Last Episode

- MLP

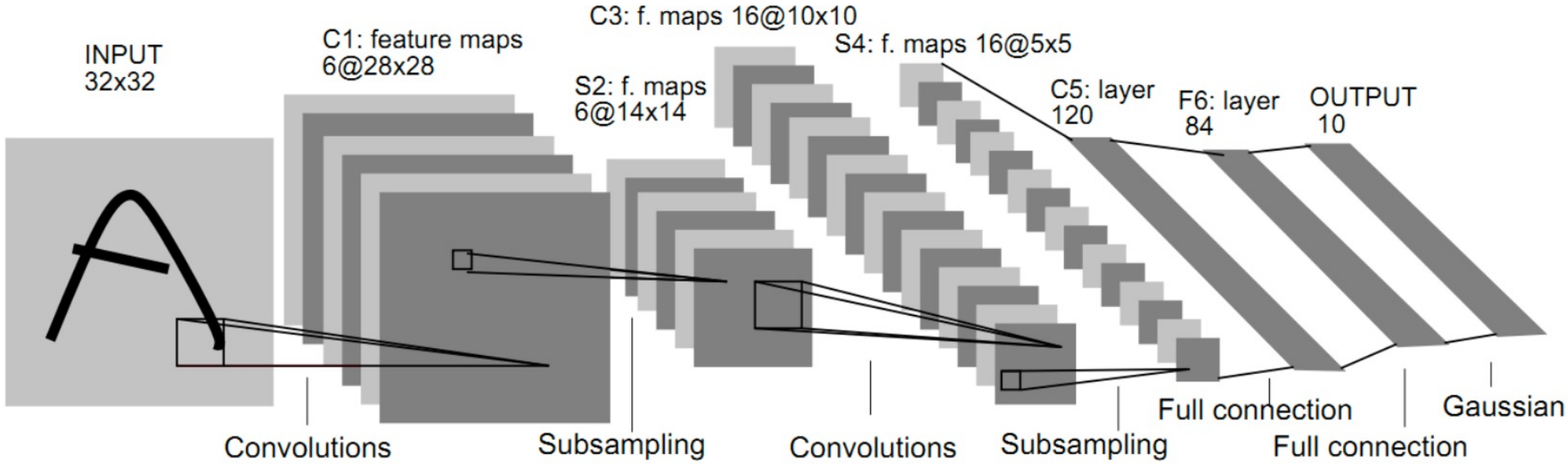
$$x_L = f_{\theta_L}\left(f_{\theta_{L-1}}\left(\cdots f_{\theta_1}(x_0)\right)\right)$$

$$x_i = f_{\theta_i}(x_{i-1}) = \sigma(w_i x_{i-1} + b_i)$$

- Universal Approximation Theory
- Activation function: Sign → Sigmoid → ReLU (LeakyReLU)
- Optimization: SGD,Adam , Initialization, Batch Normalization
- Generalization: Dropout

Convolutional Neural Networks

Convolutional Neural Network (CNN)



LeNet, 1989

"Backpropagation applied to handwritten zip code recognition", LeCun et al. 1989

Filtering for Image Processing

Original Image

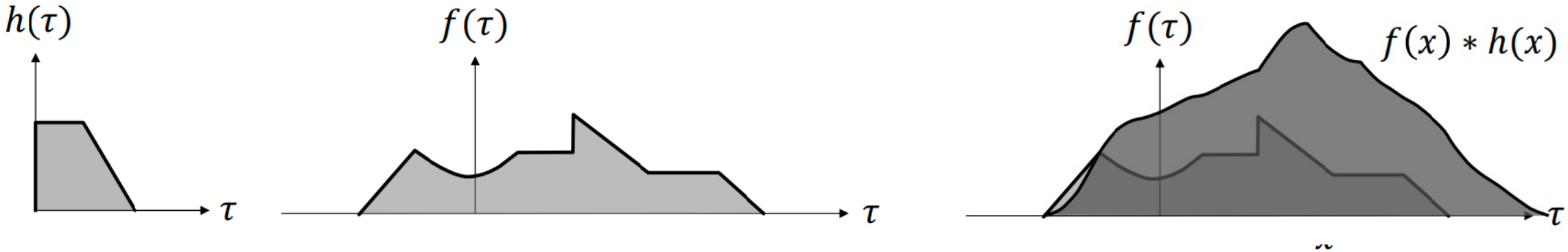
Mean filtering: replacing each pixel value in an image with the mean value of its neighbors

Median filtering: replacing each pixel value in an image with the median value of its neighbors

Linear Filtering: Convolution

Convolution of two functions $f(x)$ and $h(x)$

$$g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} f(\tau)h(x-\tau)\,d\tau$$



Convolution

Discrete Convolution: $g[i] = \sum_{n=1}^{N} f[n]h[i-n]$

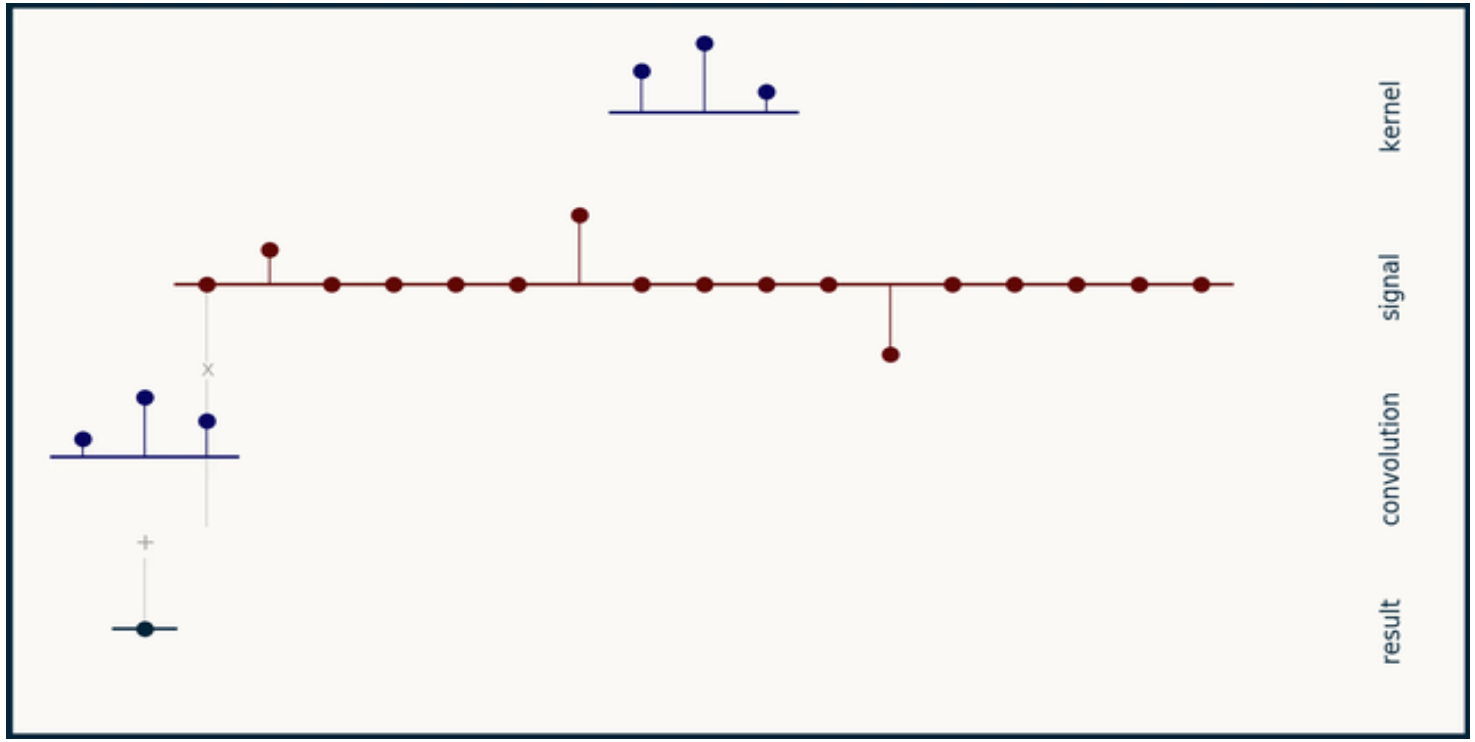


Image credit: https://e2eml.school/convolution_one_d.html

Convolution

- Linearity: $\mathcal{G}(\alpha f_1 + \beta f_2) = \alpha \mathcal{G}(f_1) + \beta \mathcal{G}(f_2)$

Let: $g_1(x) = \int_{-\infty}^{\infty} f_1(\tau) h(x-\tau)\, d\tau$ and $g_2(x) = \int_{-\infty}^{\infty} f_2(\tau) h(x-\tau)\, d\tau$

Then:

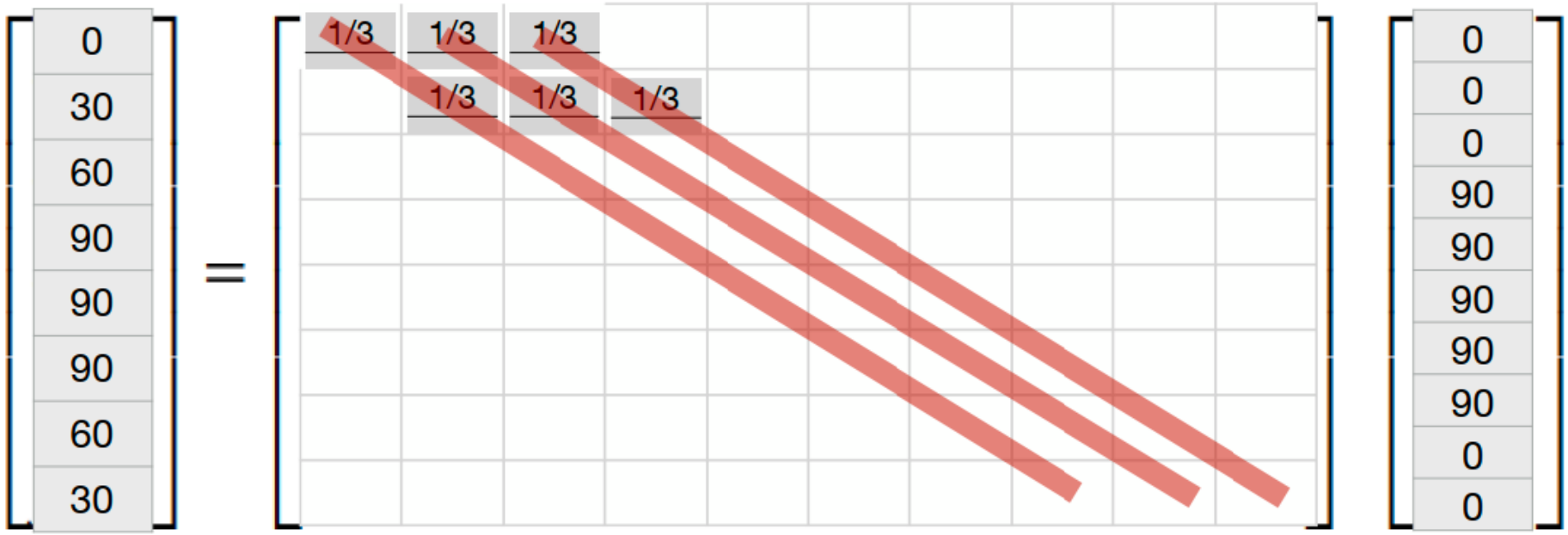
$$\int_{-\infty}^{\infty} (\alpha f_1(\tau) + \beta f_2(\tau)) h(x-\tau)\, d\tau$$

$$= \alpha \int_{-\infty}^{\infty} f_1(\tau) h(x-\tau)\, d\tau + \beta \int_{-\infty}^{\infty} f_2(\tau) h(x-\tau)\, d\tau$$

$$= \alpha g_1(x) + \beta g_2(x)$$

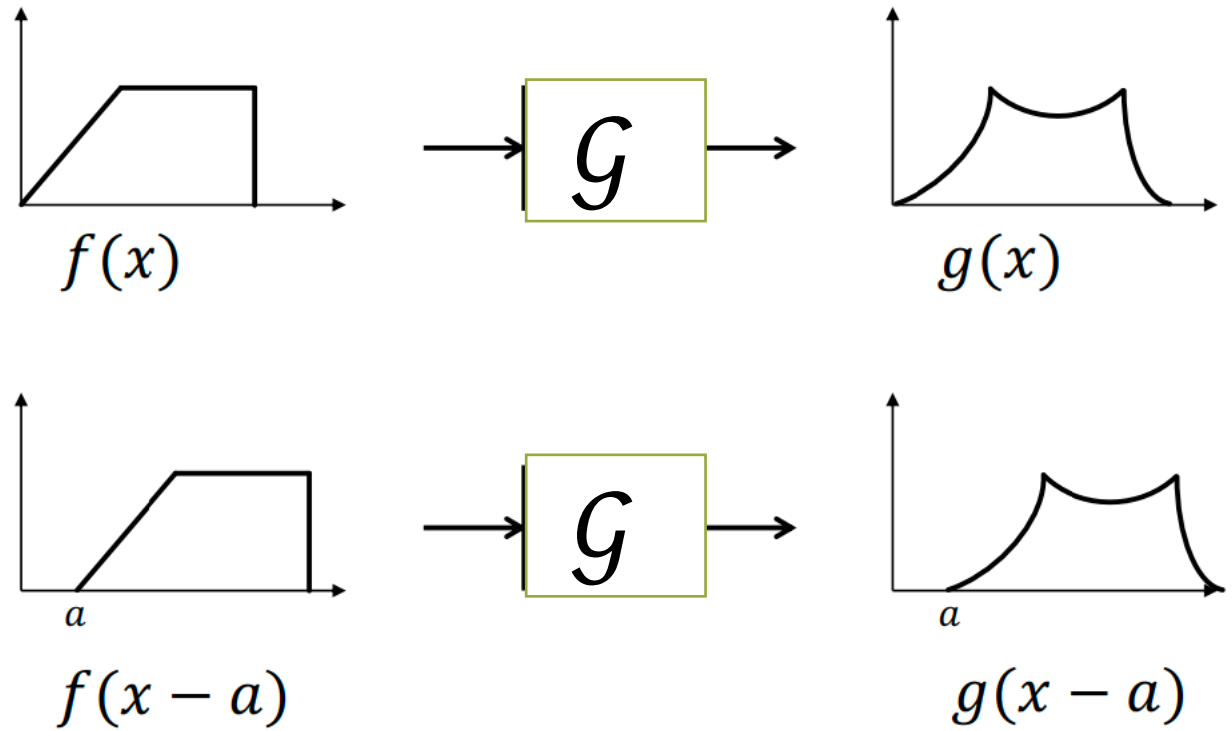
Convolution

- Discrete 1D convolution is matrix multiplication



Convolution

- Shift Invariance: $\mathcal{G}\left(f(\cdot - a)\right)(x) = \mathcal{G}(f(\cdot))(x - a)$

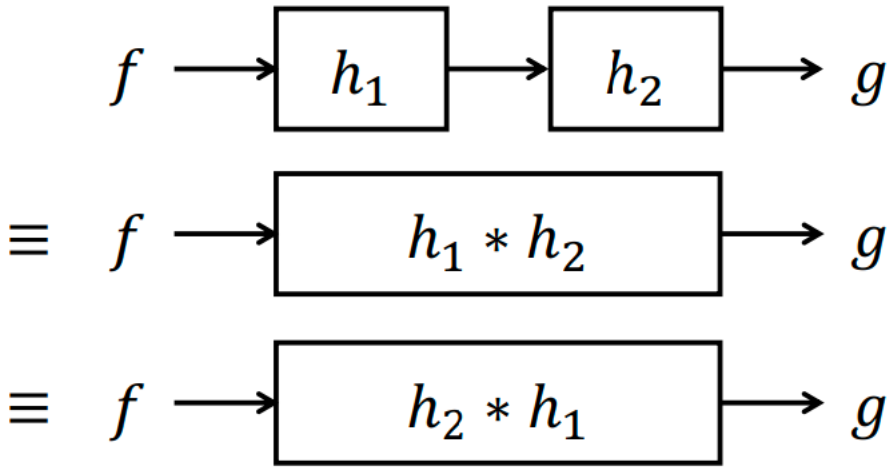


Convolution

Commutative $a * b = b * a$

Associative $(a * b) * c = a * (b * c)$

Cascaded System



2D Convolution

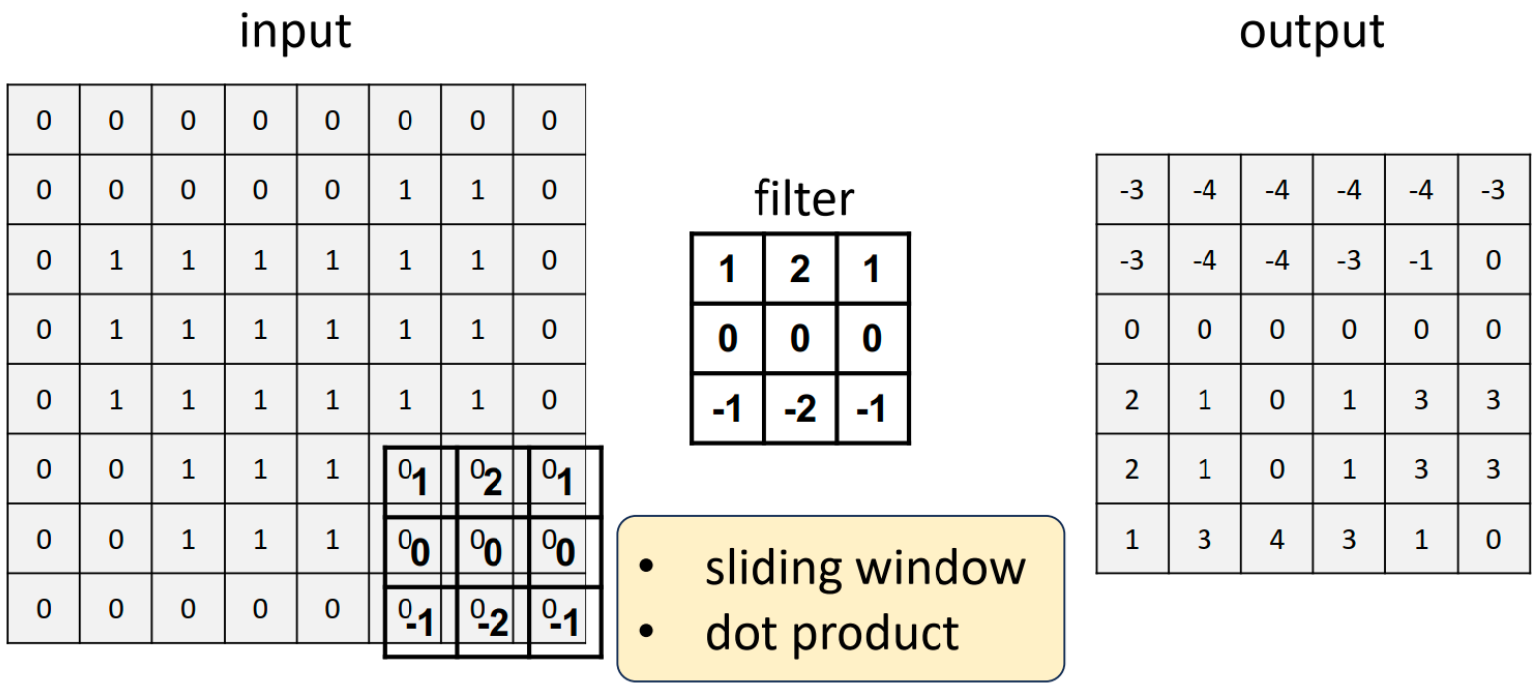
- Continuous 2D Convolution

$$g(x,y) = \iint_{-\infty}^{\infty} f(\tau,\mu)h(x-\tau,y-\mu)\,d\tau d\mu$$

- Discrete 2D Convolution

$$g[i,j] = \sum_{m=1}^{M}\sum_{n=1}^{N} f[m,n]\underbrace{h[i-m,j-n]}_{\text{“Mask,” “Kernel,” “Filter”}}$$

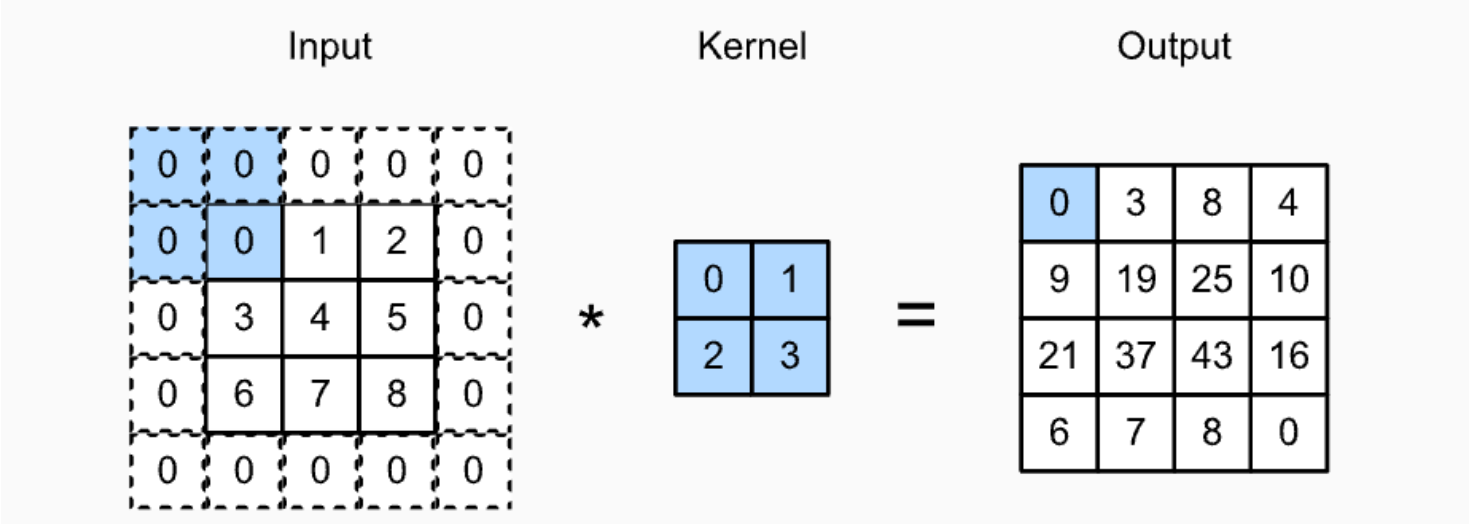
2D Convolution



the input shape: $n_h \times n_w$, the convolution kernel shape $k_h \times k_w$,
the output shape will be $(n_h - k_h + 1) \times (n_w - k_w + 1)$

2D Convolution

- Padding



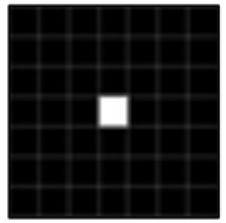
- The input shape: $n_h \times n_w$, the convolution kernel shape $k_h \times k_w$, the padding size (p_h, p_w), the output shape will be $(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$.
- In many cases, we will want to set $p_h = k_h - 1$ and $p_w = k_w - 1$ to give the input and output the same height and width.

Example

Input

$f(x, y)$

$*$



$\delta(x, y)$

$=$

Output

$f(x, y)$

Impulse Filter: $f * \delta = f$

Example

Input

$*$

$=$

Output

$f(x, y)$ $\delta(x - u, y - v)$ $f(x - u, y - v)$

Shift

Example

Input

$f(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$a(x,y)$

$=$

Output

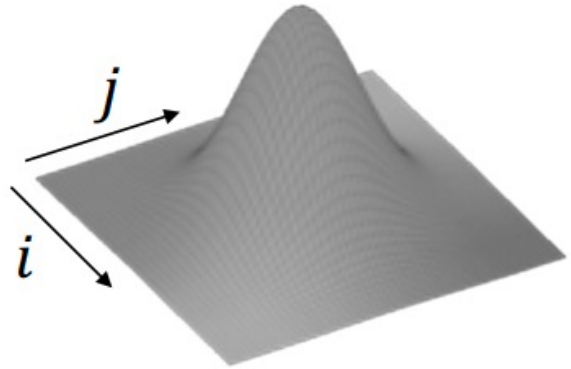
$g(x,y)$

Averaging: mean filtering

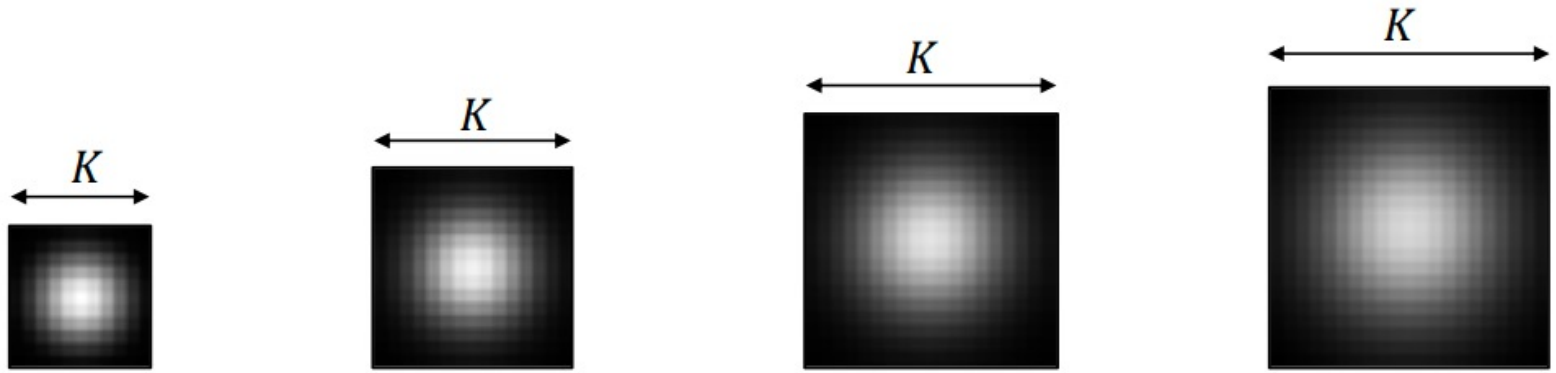
Gaussian Filtering: Smoothing

$$n_\sigma[i,j] = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\left(\frac{i^2+j^2}{\sigma^2}\right)}$$

1



σ^2: Variance



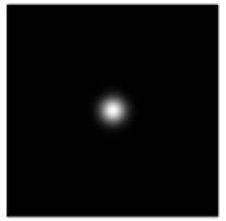
$\sigma = 2$ $\sigma = 3$ $\sigma = 4$ $\sigma = 5$

Gaussian Filtering: Smoothing

Input

$f(x,y)$

$*$



$\sigma = 4$

$n_4(x,y)$

$=$

Output

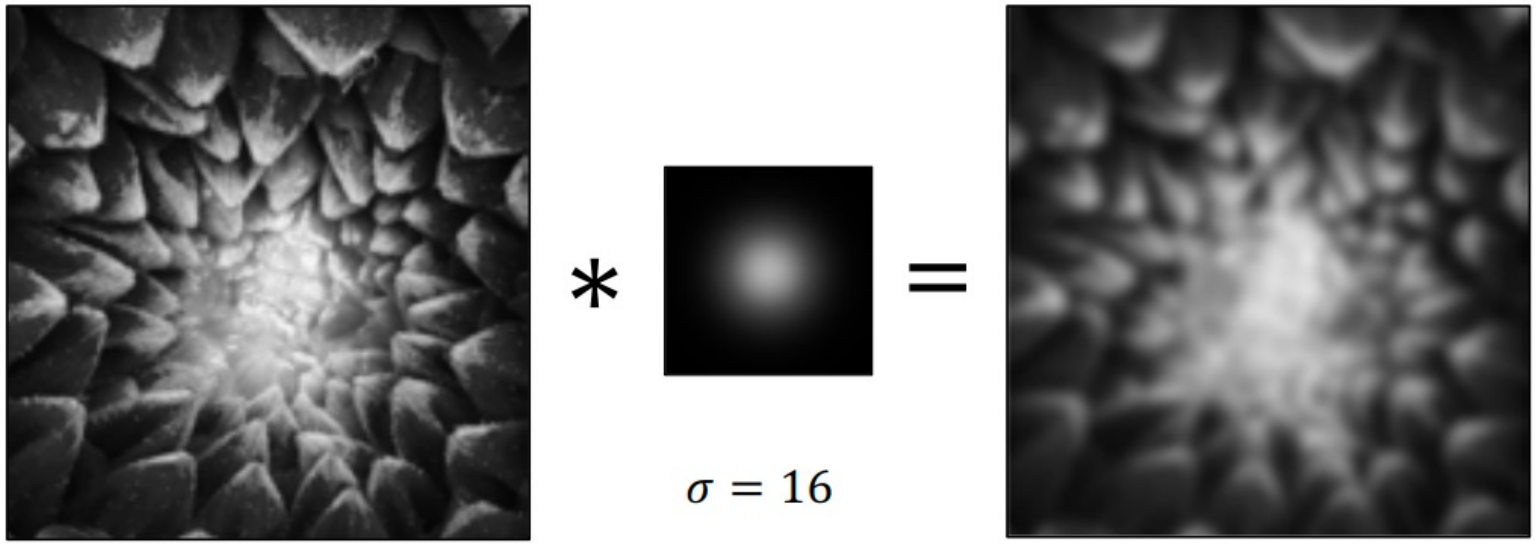
$g(x,y)$

Larger the kernel (or σ), more the blurring

Gaussian Filtering: Smoothing

Input

Output



$*$

$=$

$\sigma = 16$

$f(x, y)$ $n_{16}(x, y)$ $g(x, y)$

Larger the kernel (or σ), more the blurring

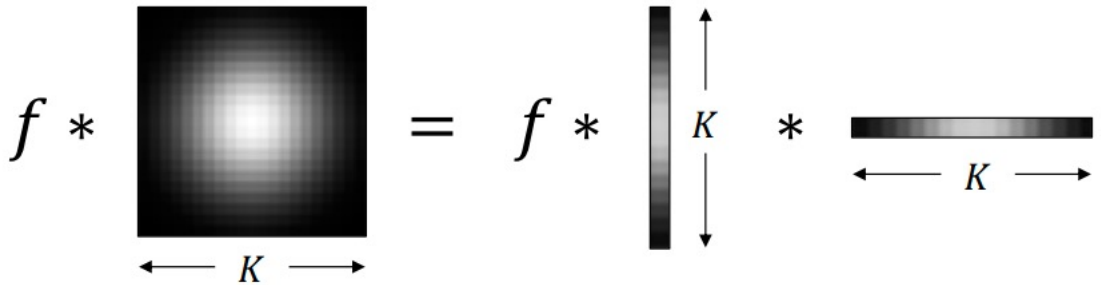
Gaussian Filtering

- Gaussian filter is separable

$$g[i,j] = \frac{1}{2\pi\sigma^2}\sum_{m=1}^{K}\sum_{n=1}^{K} e^{-\frac{1}{2}\left(\frac{m^2+n^2}{\sigma^2}\right)} f[i-m, j-n]$$

$$g[i,j] = \frac{1}{2\pi\sigma^2}\sum_{m=1}^{K} e^{-\frac{1}{2}\left(\frac{m^2}{\sigma^2}\right)} \cdot \sum_{n=1}^{K} e^{-\frac{1}{2}\left(\frac{n^2}{\sigma^2}\right)} f[i-m, j-n]$$

- Using one 2D Gaussian filter = Using two 1D Gaussian filters



$O(N^2K^2)$ vs. $O(N^2K)$

Sobel Filter

filter

$$
* \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} =
$$

Sobel Filter

- Separable

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \otimes [1 \quad 2 \quad 1]$$

High pass filter: edge detection

Low pass filter: smoothing

What does “high(low) pass” mean?

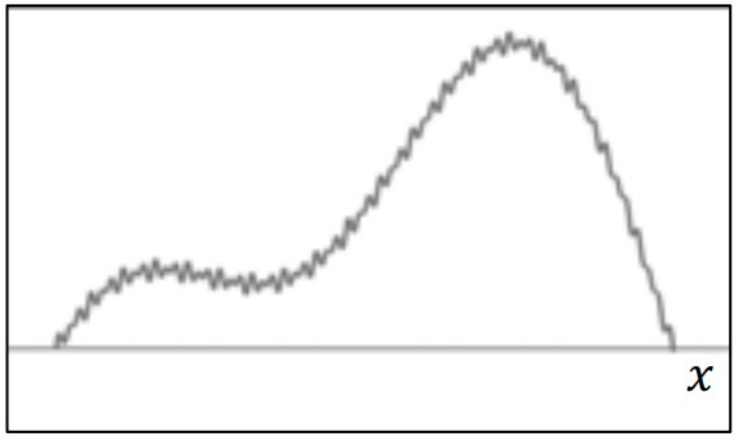
Convolution Theorem

Spatial Domain	Frequency Domain
$g(x) = f(x) * h(x)$	$G(u) = F(u)\,H(u)$
Convolution	Multiplication

$$G(u) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau)h(x-\tau)e^{-i2\pi ux}\,d\tau dx$$

$$G(u) = \underbrace{\int_{-\infty}^{\infty} f(\tau)e^{-i2\pi u\tau}\,d\tau}_{F(u)} \; \underbrace{\int_{-\infty}^{\infty} h(x-\tau)e^{-i2\pi u(x-\tau)}\,dx}_{H(u)}$$

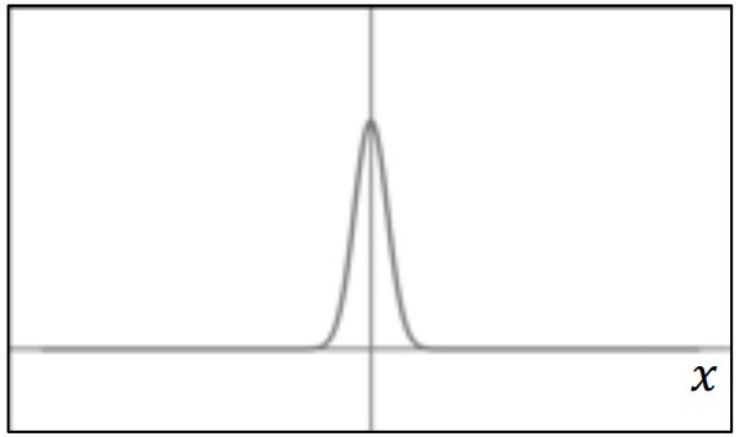
Gaussian Filtering in Fourier Domain



Noisy Signal $f(x)$

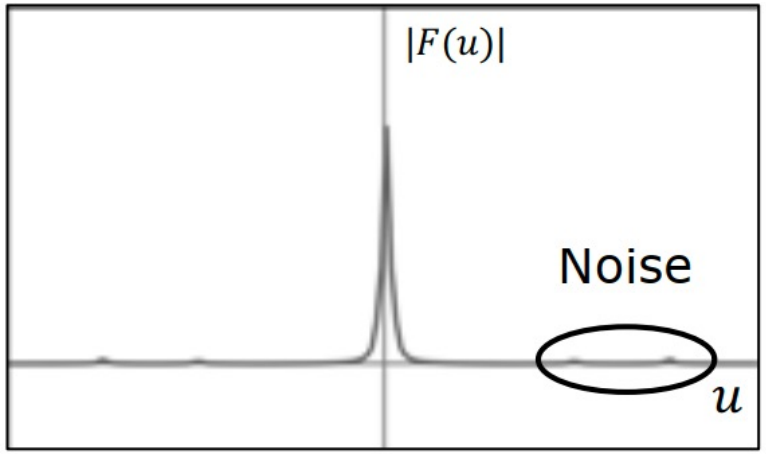
FT

$*$

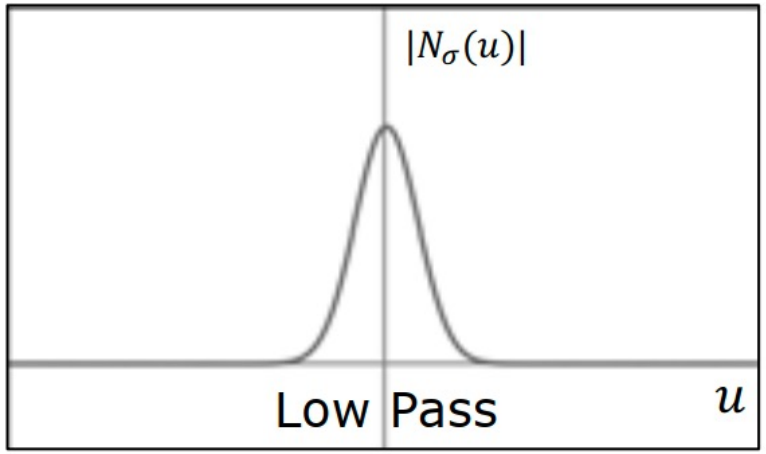


Gaussian Kernel $n_\sigma(x)$

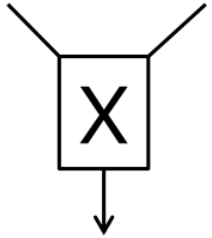
FT

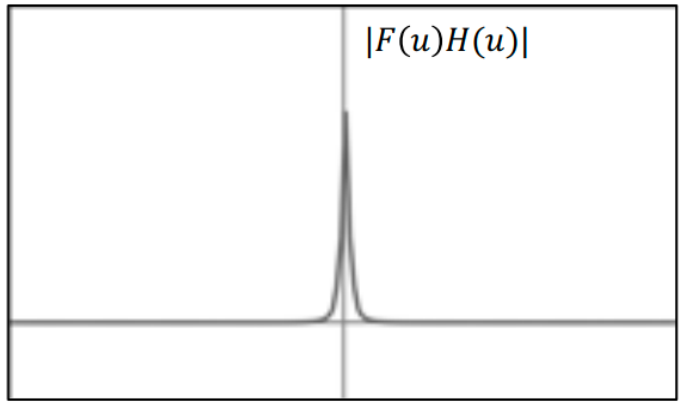


$F(u)$



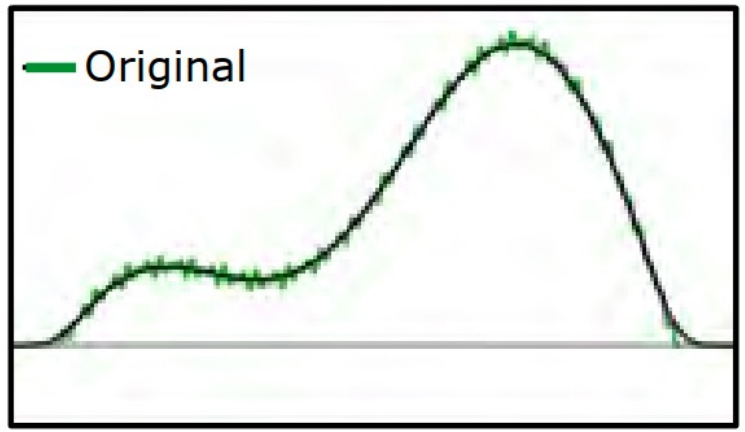
$N_\sigma(u)$



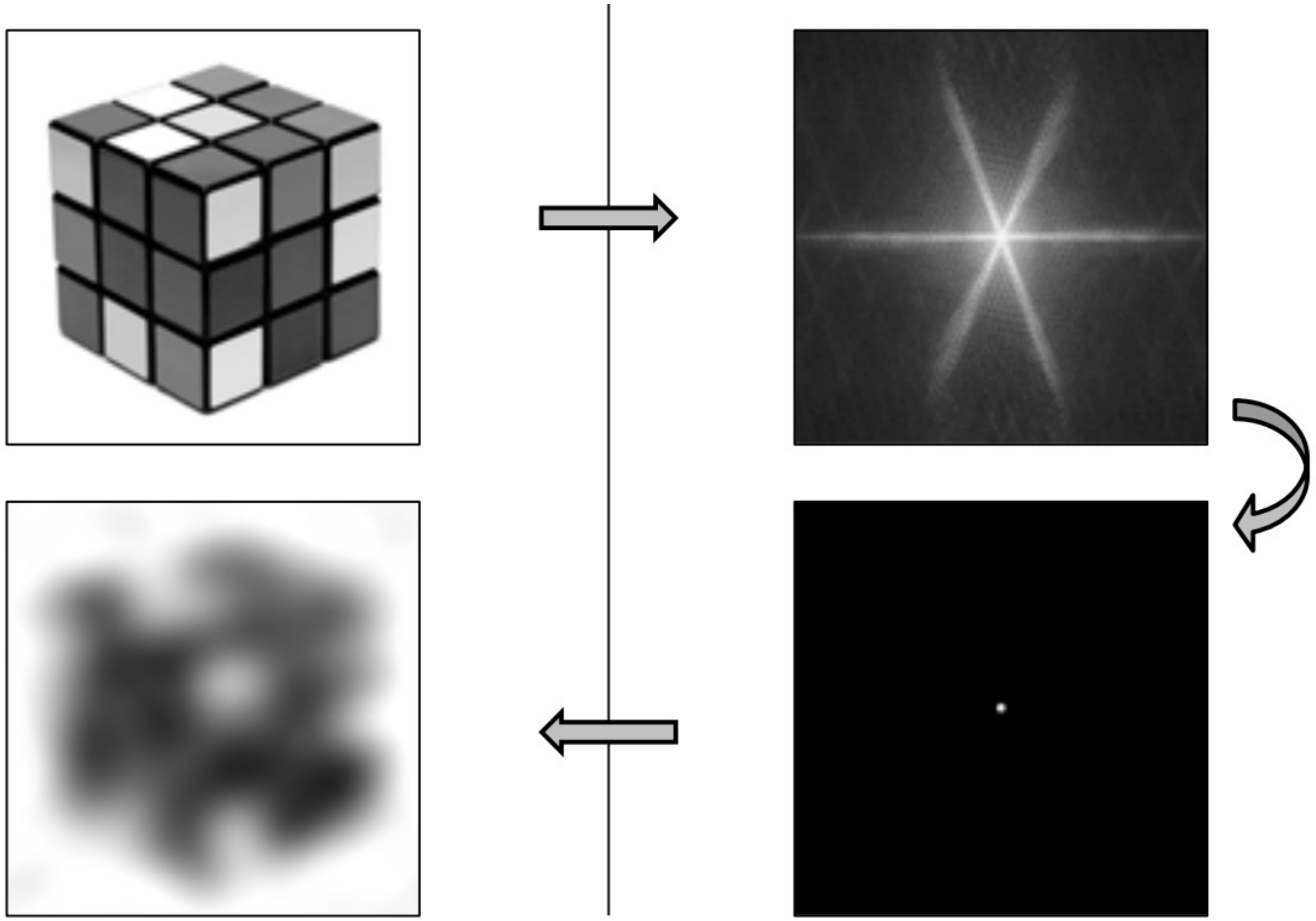


$F(u)H(u)$

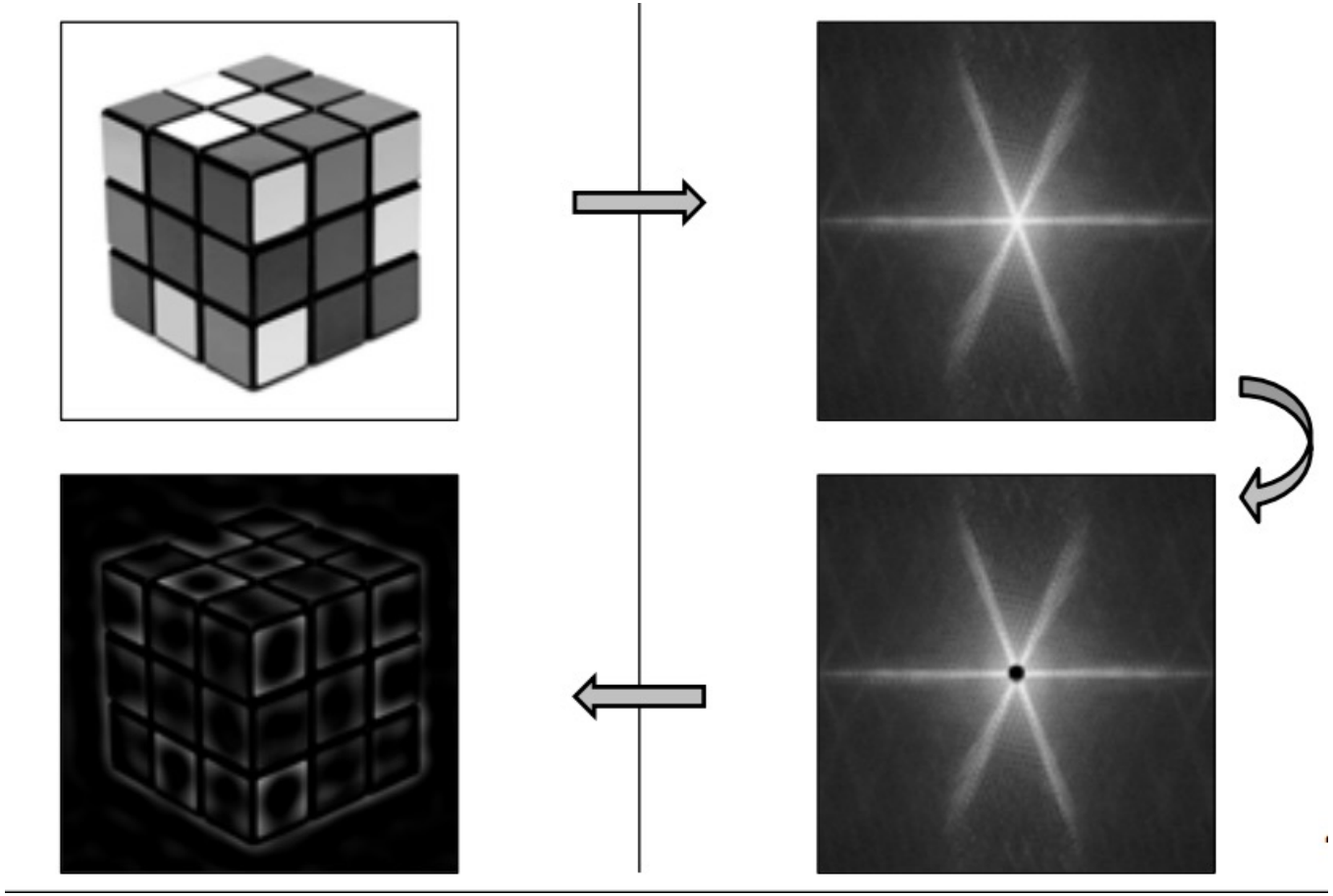
Gaussian smoothed signal



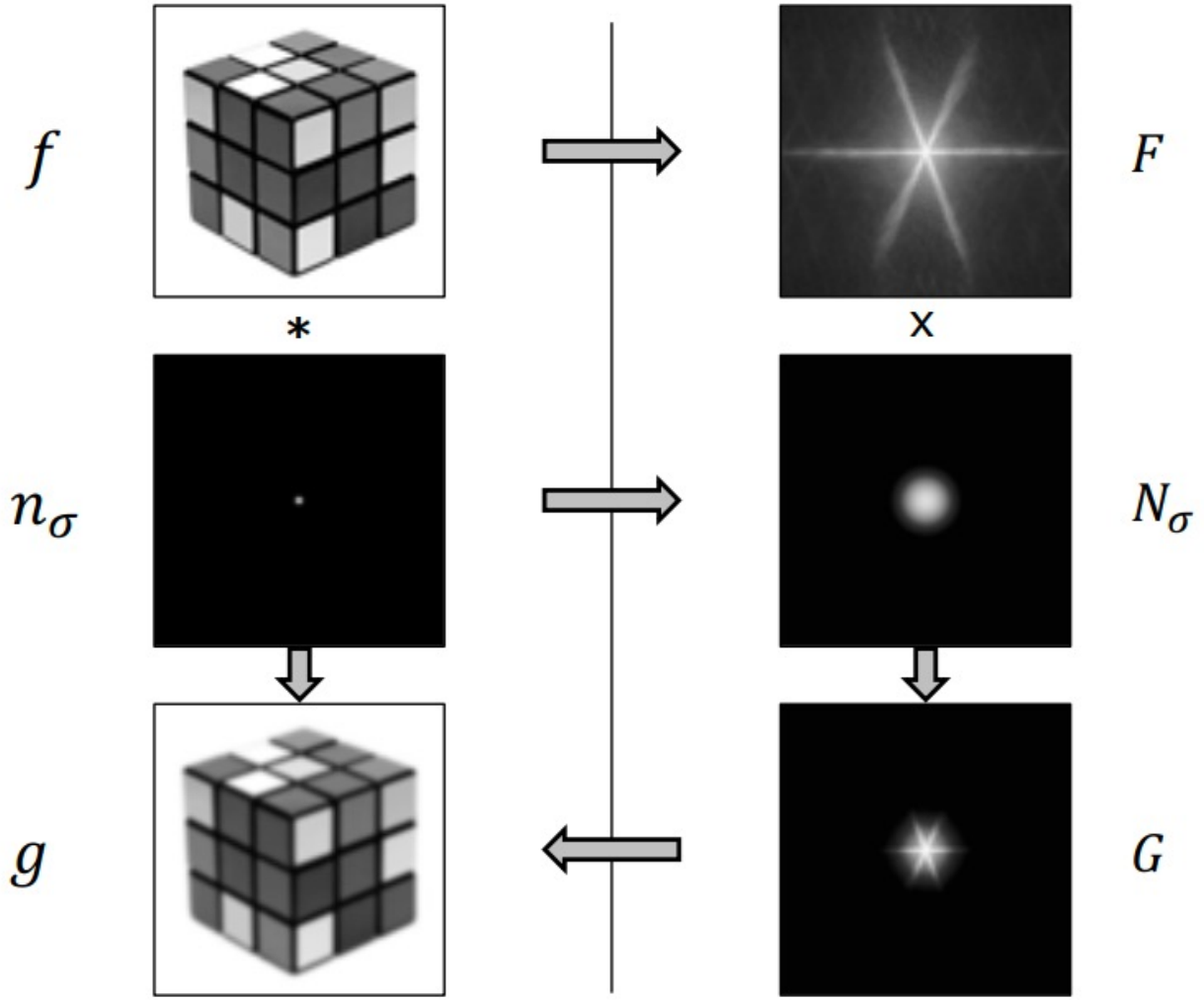
Low Pass Filter



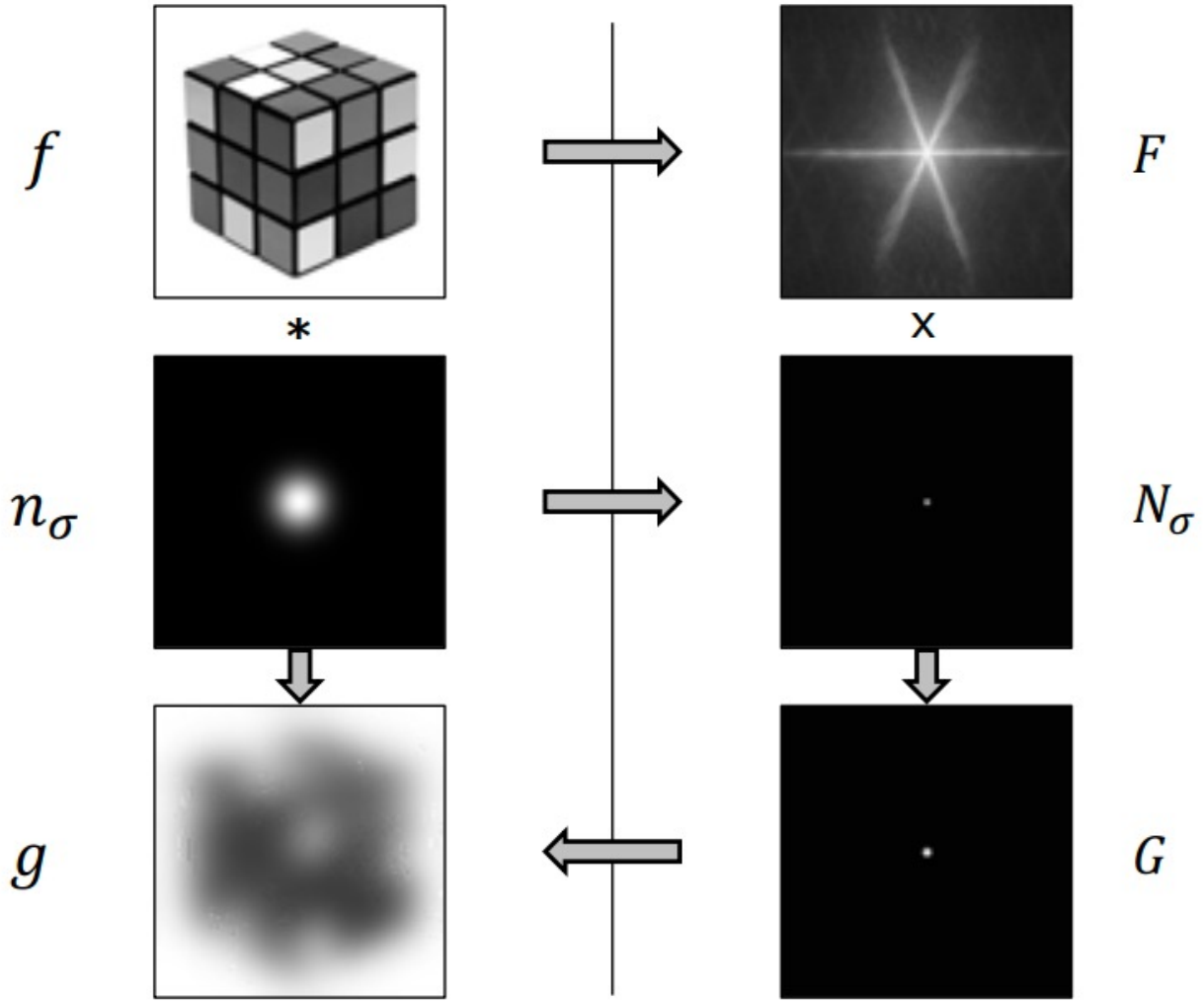
High Pass Filter



Gaussian Filtering



Gaussian Filtering



3D Convolution

RGB Image:3 channels

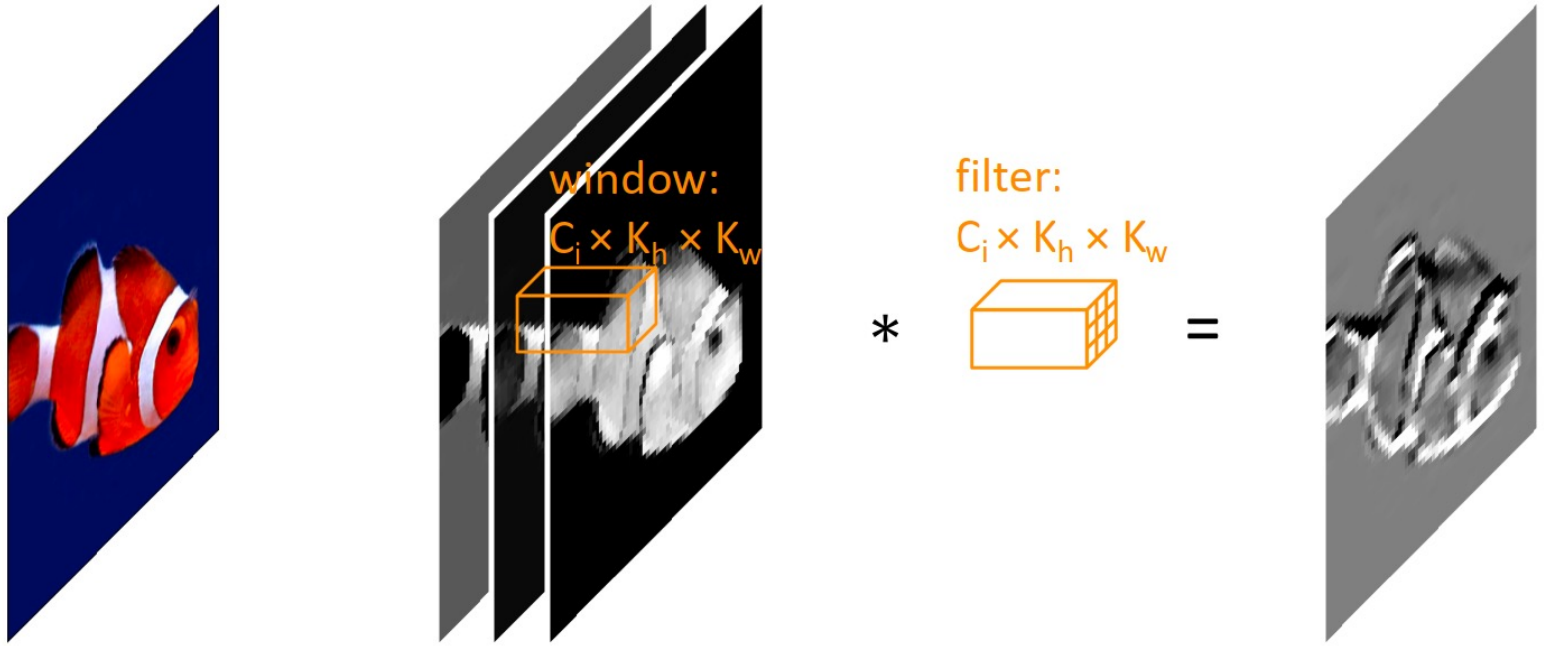
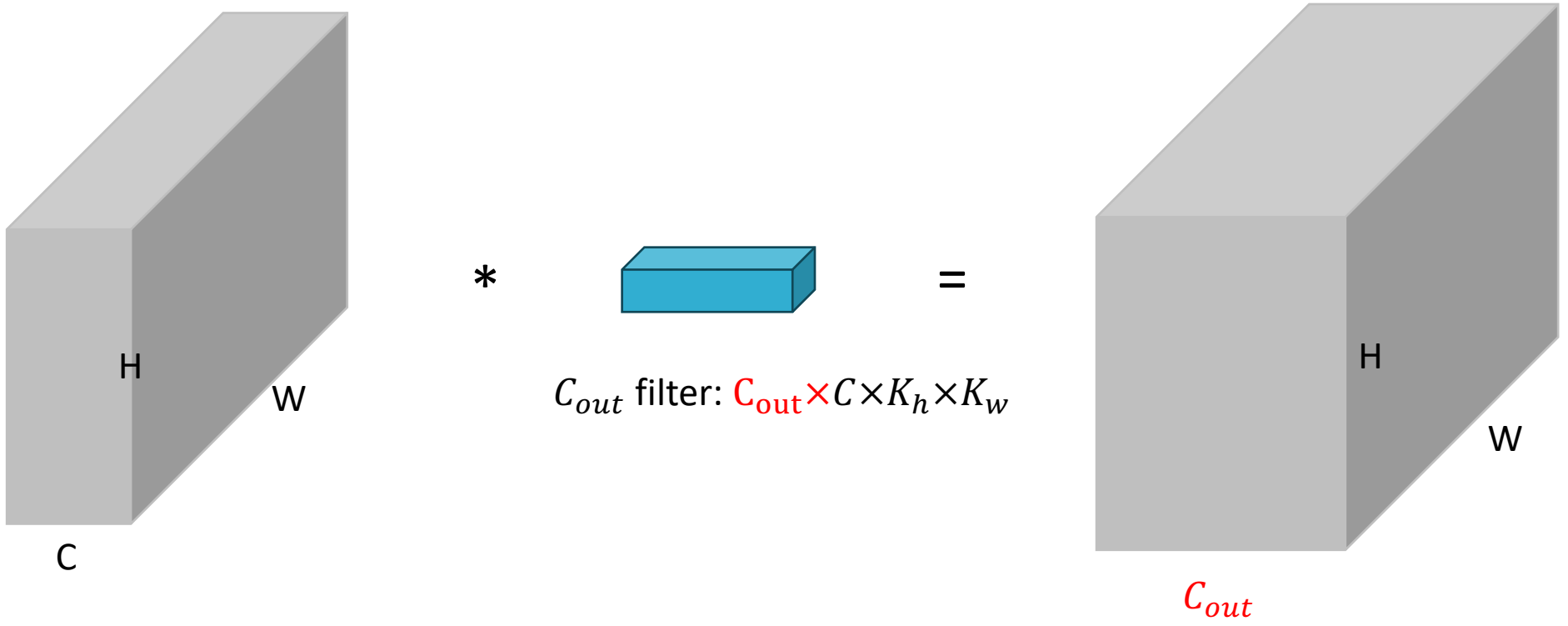


Image Credits: He Kaiming

Tensor Convolution

- Feature tensor: $C \times H \times W$



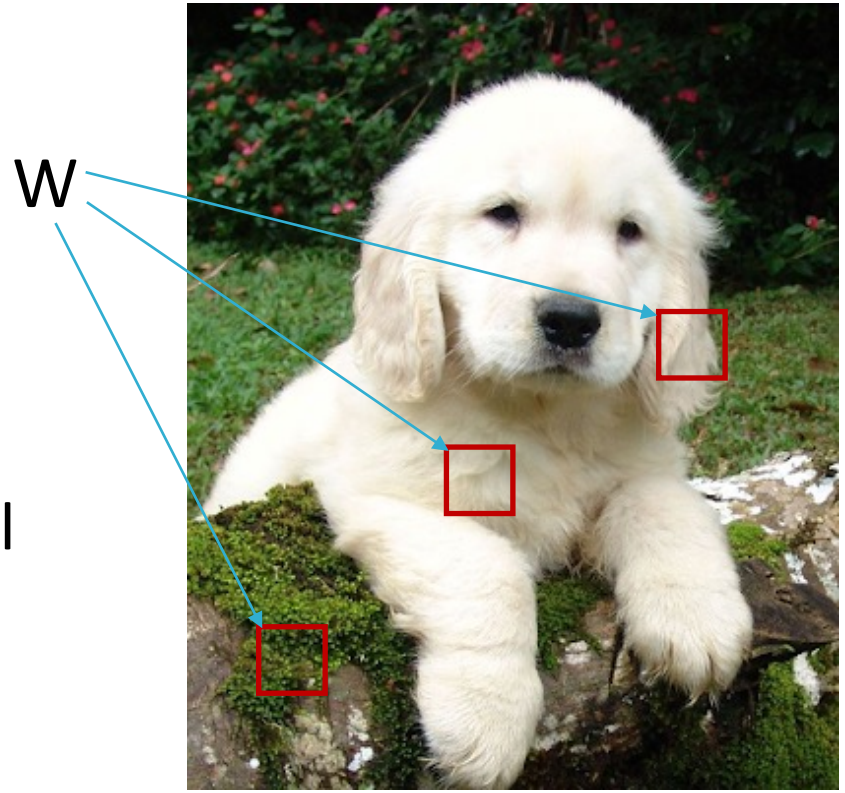
CNN

- Convolutional layer

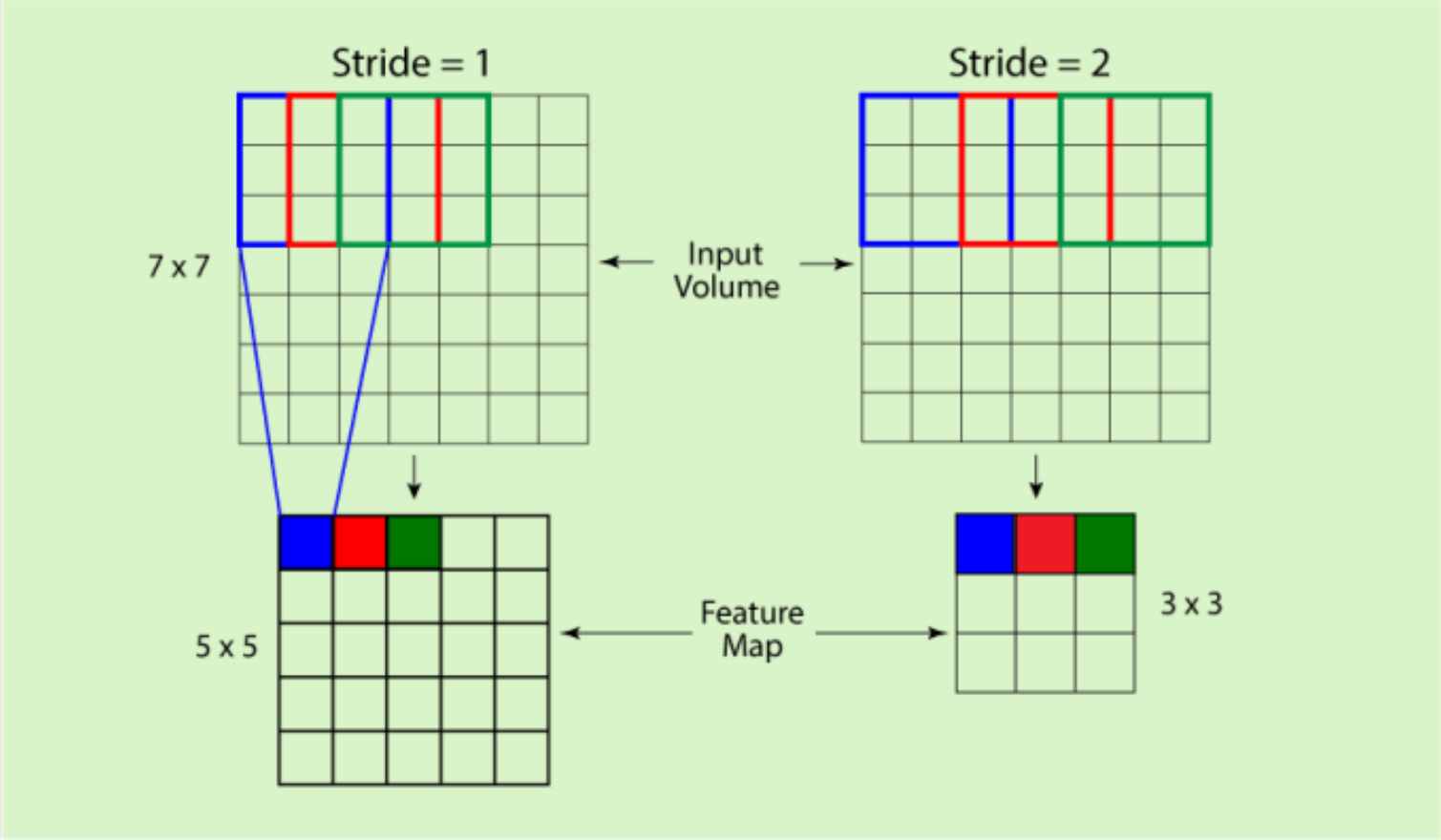
$$y = \mathrm{w} * x + b$$

  - input: $C_{in} \times H \times W$
  - kernel: $C_{out} \times C_{in} \times K_h \times K_w$
  - bias: $C_{out} \times 1 \times 1$
  - output: $C_{out} \times H_{out} \times W_{out}$

- Less parameters due to weight sharing and local connection
  - MLP: weights $C_{out} \times C_{in} \times HW \times H_{\mathrm{out}} W_{out}$
  - MLP+Local connection: $C_{out} \times C_{in} \times K_h K_w \times H_{\mathrm{out}} W_{out}$



Stride



Stride>1: resolution↓

https://developersbreach.com/convolution-neural-network-deep-learning/

CNN

- Receptive field of CNN expands along with the depth of NN.

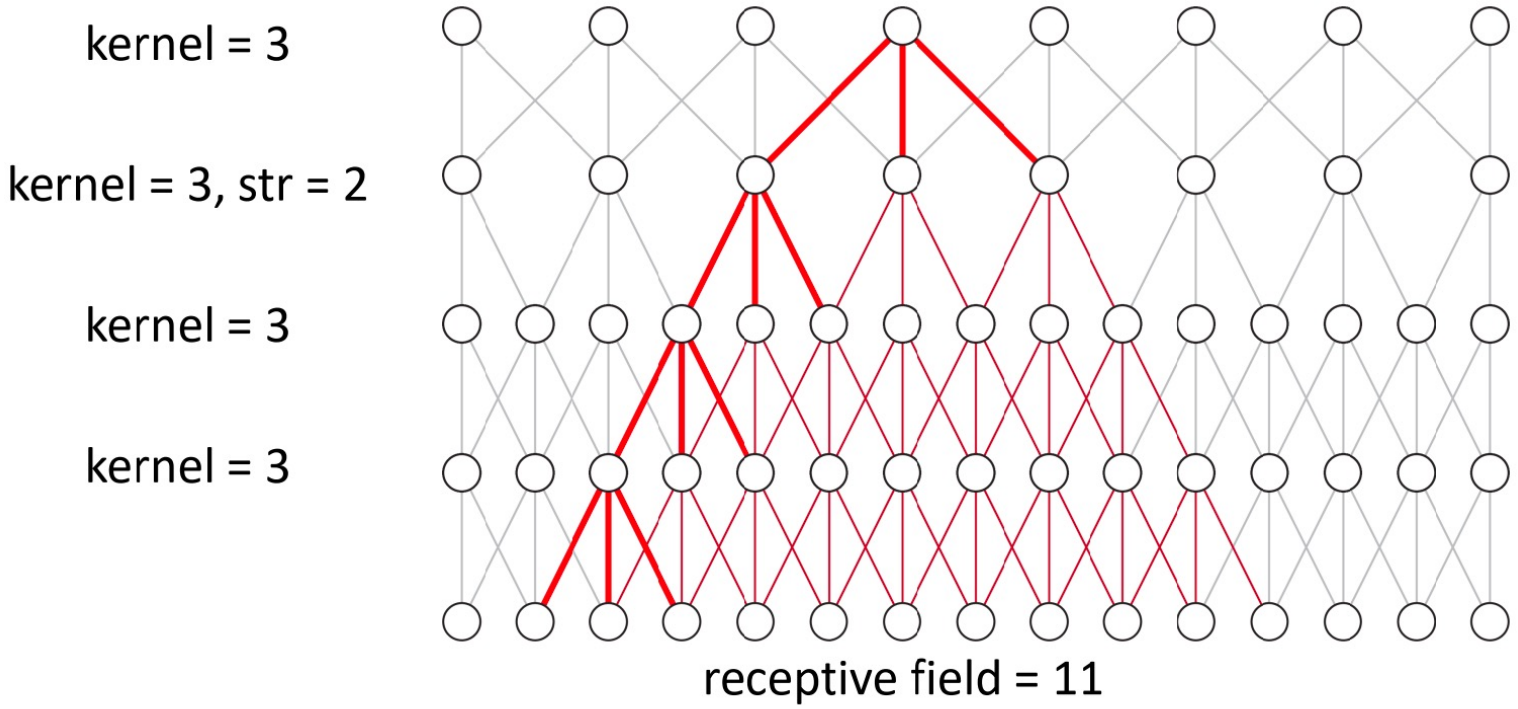


Image Credits: He Kaiming

Pooling Layer

- Max Pooling:

$$y[i] = Max_{\{j \in N(i)\}} x[j]$$

- Average Pooling

$$y[i] = \frac{1}{N(i)} \sum_{\{j \in N(i)\}} x[j]$$

- Similar as convolution: local connection + shift invariance, but non-linear; usually set stride > 2.

Classification

- Features: $\{\boldsymbol{u}(\boldsymbol{x}): \boldsymbol{u} = w^T \phi(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^n, \mathbf{w} \in \mathbb{R}^{m \times n}\}$.
- Multi-classes one-hot coding $\boldsymbol{\mu} = (\mu^1, \mu^2, \dots, \mu^m)^T$

$$\mu^i = \frac{e^{u^i}}{\sum_i e^{u^i}} (\text{softmax function})$$

probability belonging to the i-th categories

- Loss function: cross entropy

$$L(\boldsymbol{y}, f(\boldsymbol{x})) = -\sum_i y^i \log \mu^i$$

The most important element: feature extraction!

Features Should Be Invariant To

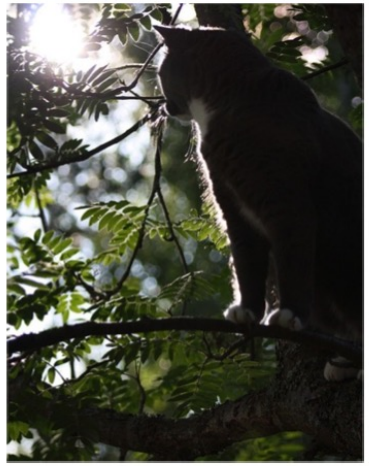
viewpoint variation

occlusion

Features Should Be Invariant To




Deformation



Illumination

Features Should Be Invariant To

Intra-class Variation

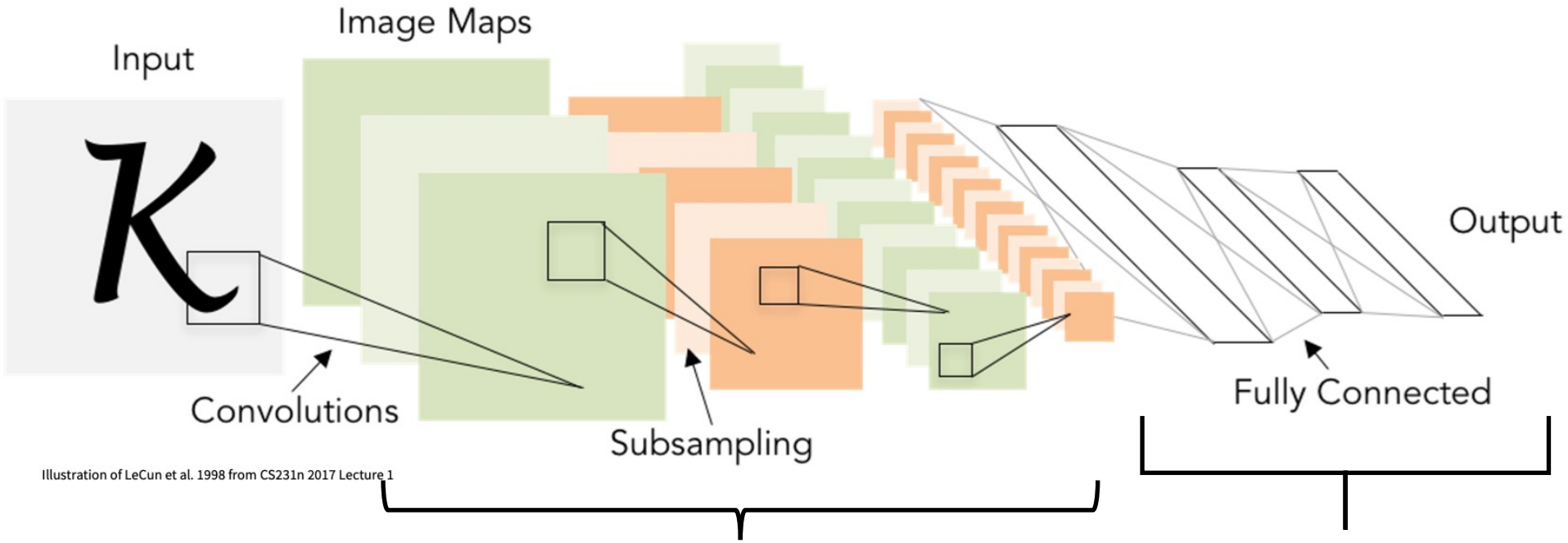
This image is CC0 1.0 public domain

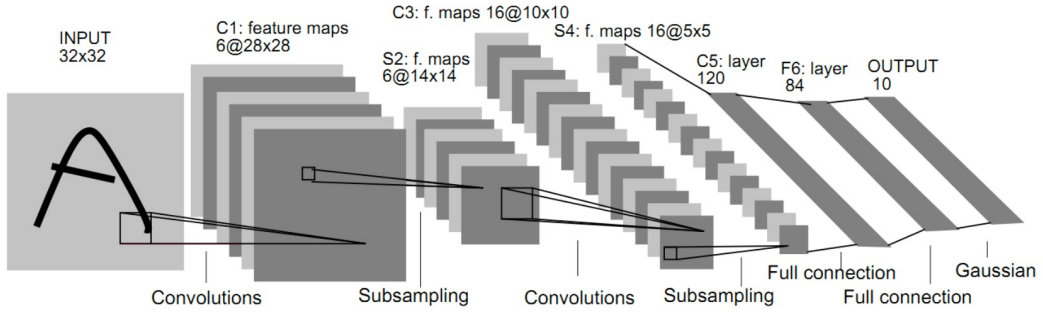
This image is CC0 1.0 public domain

Background Clutter

Deep Neural Network Classifier

- Train a neural network for feature extraction





LeNet,1989

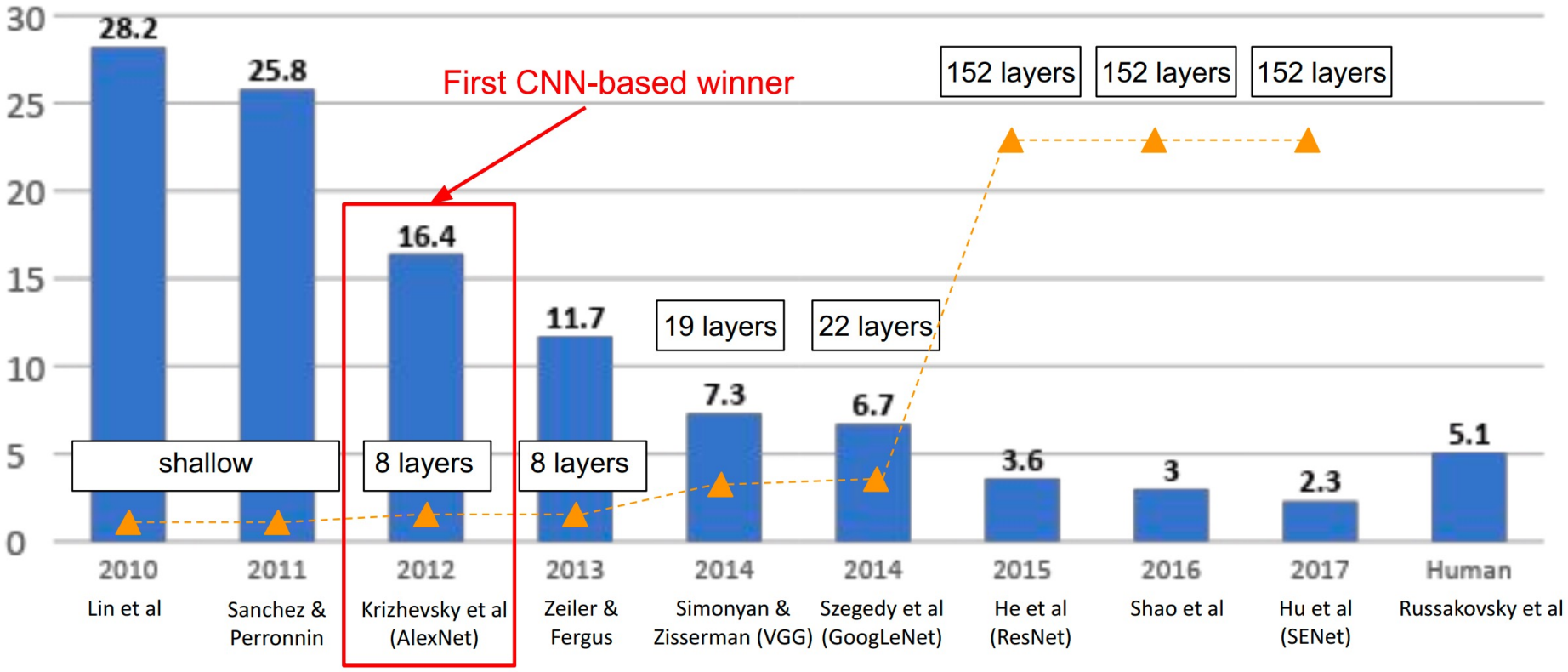
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

torch.nn.Conv2d(*in_channels, out_channels, kernel_size, stride=1, padding=*

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

AlexNet

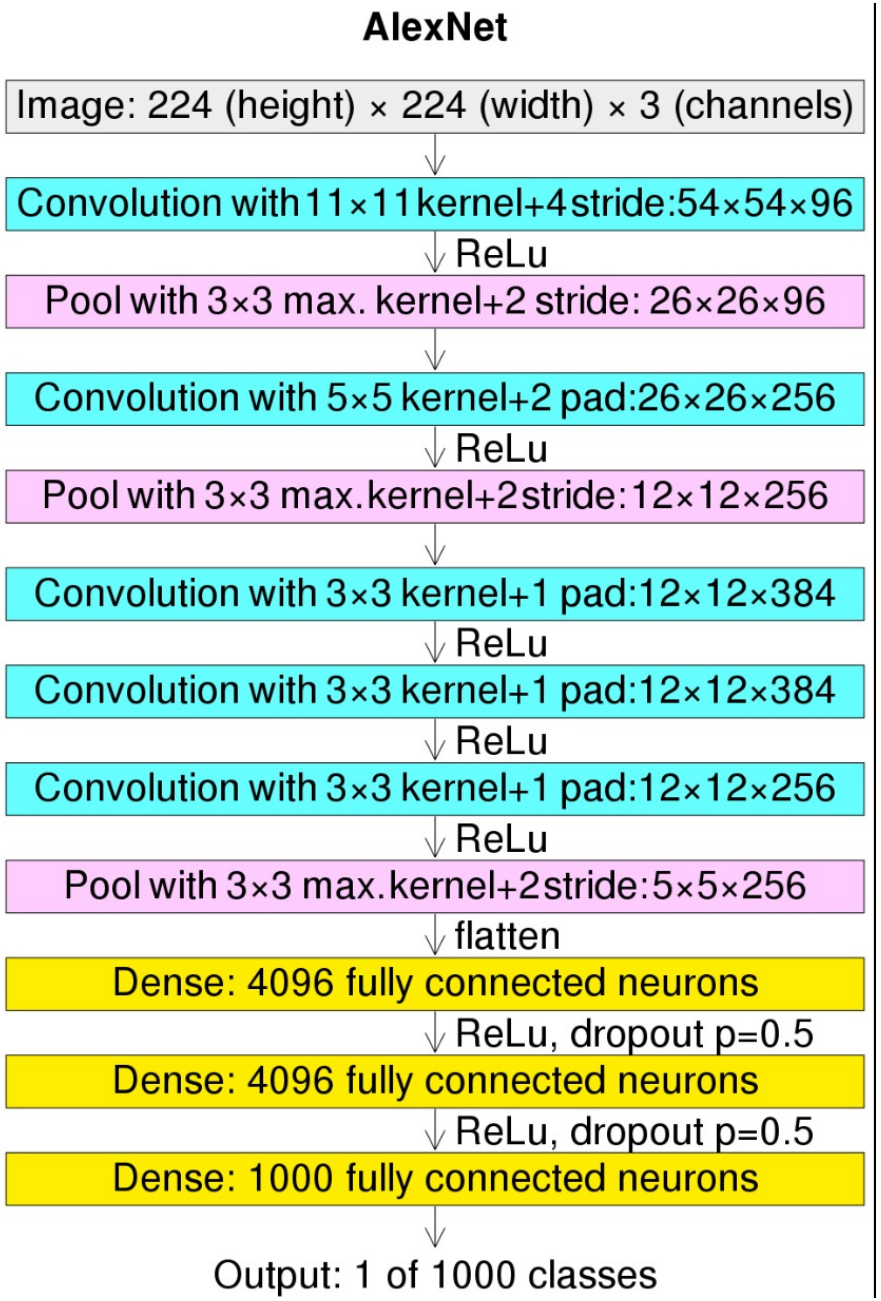
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



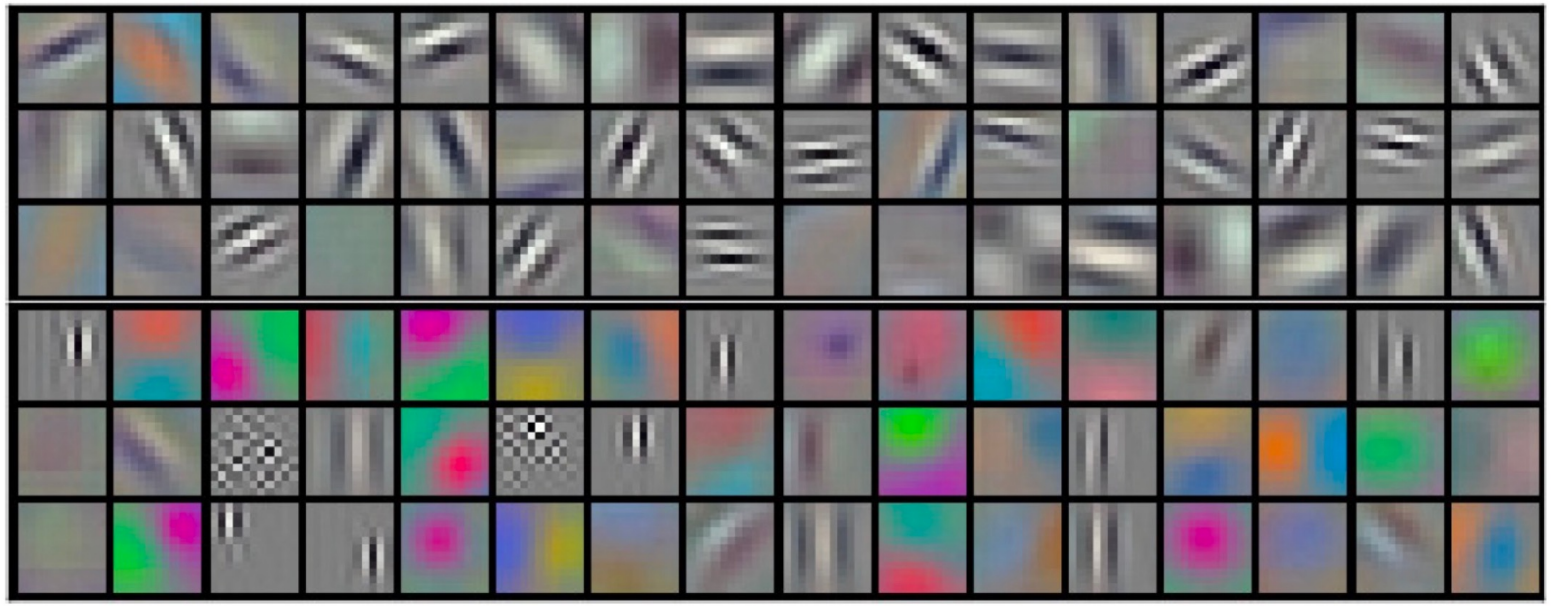
- Stanford CS231n

```python
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        _log_api_usage_once(self)
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

ImageNet: 1000 classes



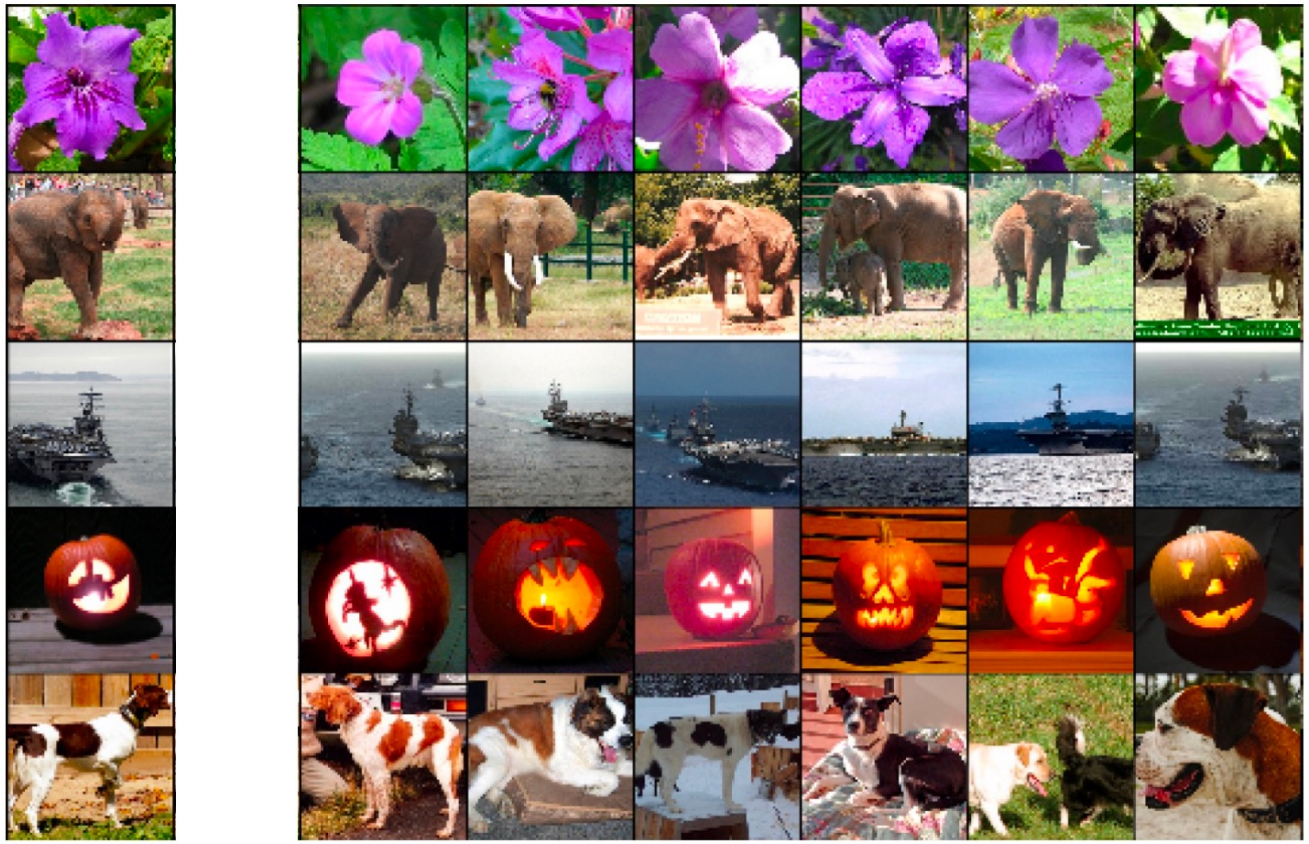
AlexNet



Filters at the first layer

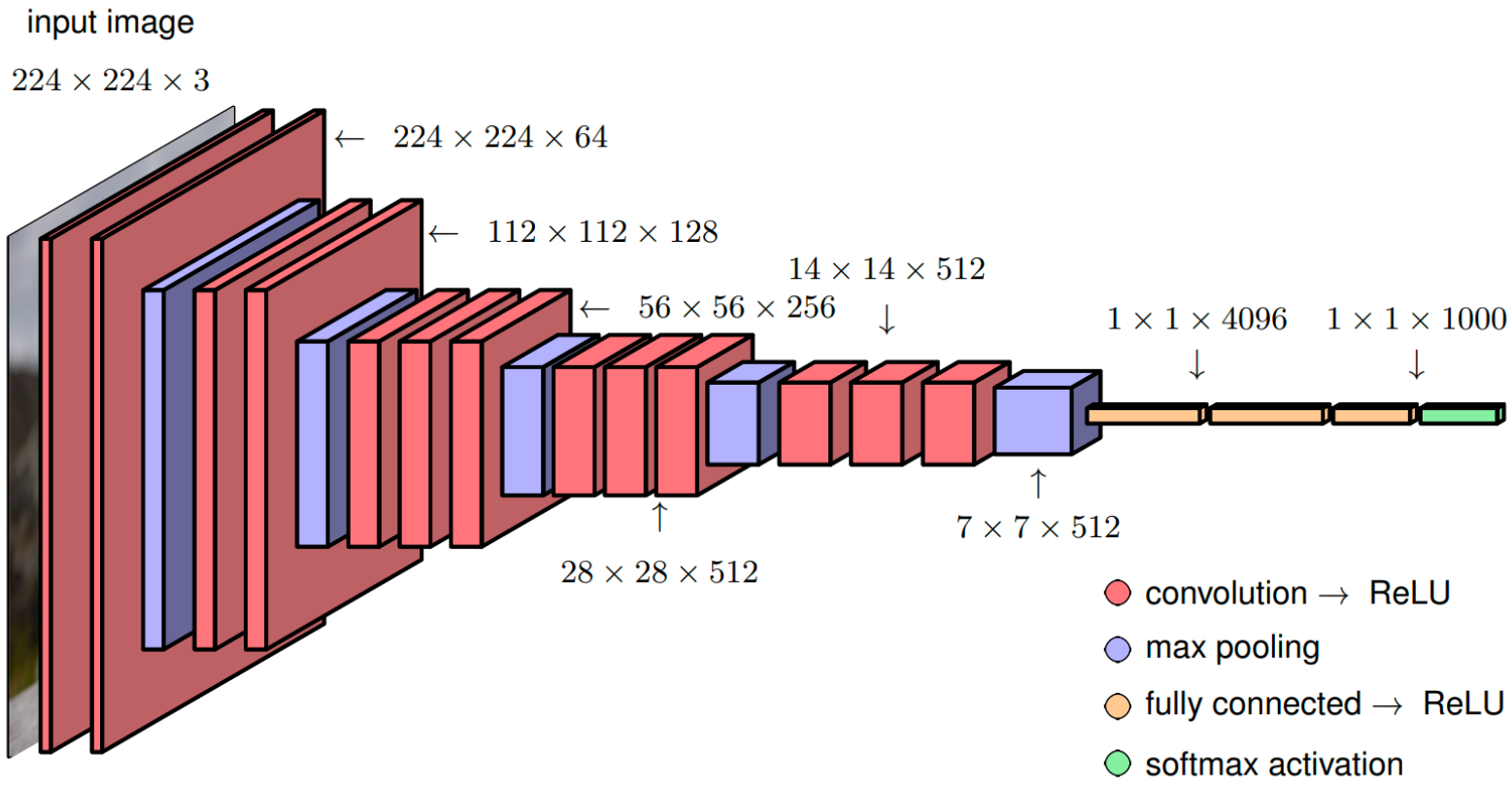
"ImageNet Classification with Deep Convolutional Neural Networks", Krizhevsky, Sutskever, Hinton. NeurIPS 2012

AlexNet



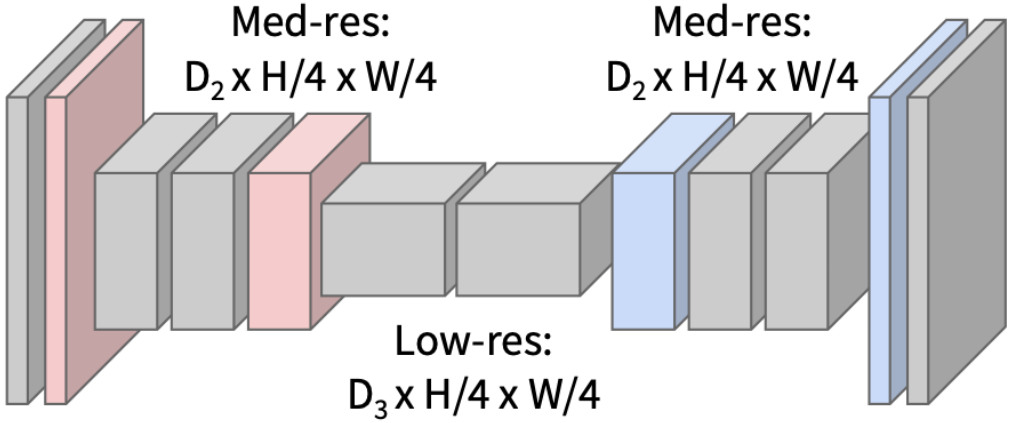
Closest images to the reference in the feature space

VGG-16


input image
224 × 224 × 3
← 224 × 224 × 64
← 112 × 112 × 128
← 56 × 56 × 256
14 × 14 × 512
1 × 1 × 4096
1 × 1 × 1000
28 × 28 × 512
7 × 7 × 512
convolution → ReLU
max pooling
fully connected → ReLU
softmax activation

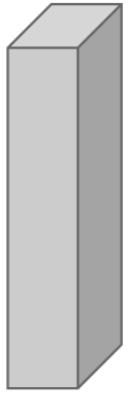
Unet for Segmentation

Input:
3 x H x W



High-res:
D_1 x H/2 x W/2

High-res:
D_1 x H/2 x W/2



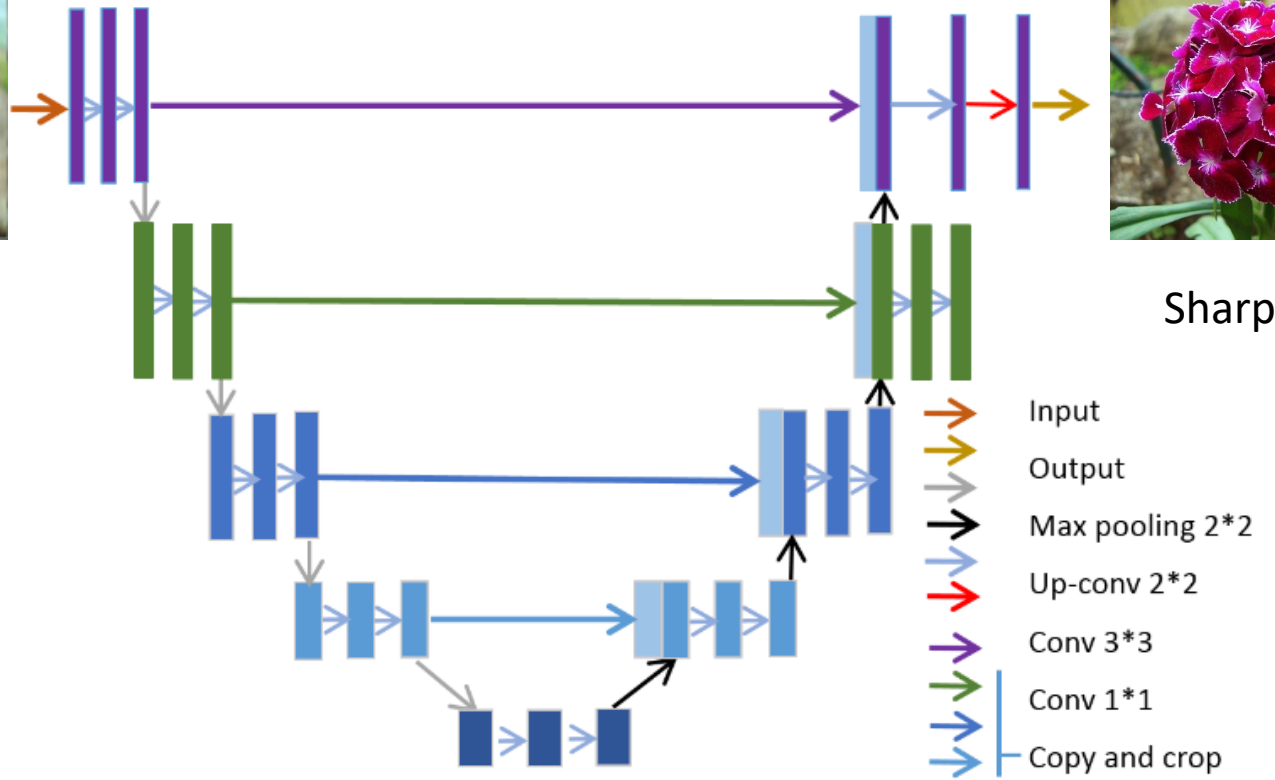
C x H x W

Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015 Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Unet for Image Restoration

Blurred Image



Multiscale Architecture

Sharp Image

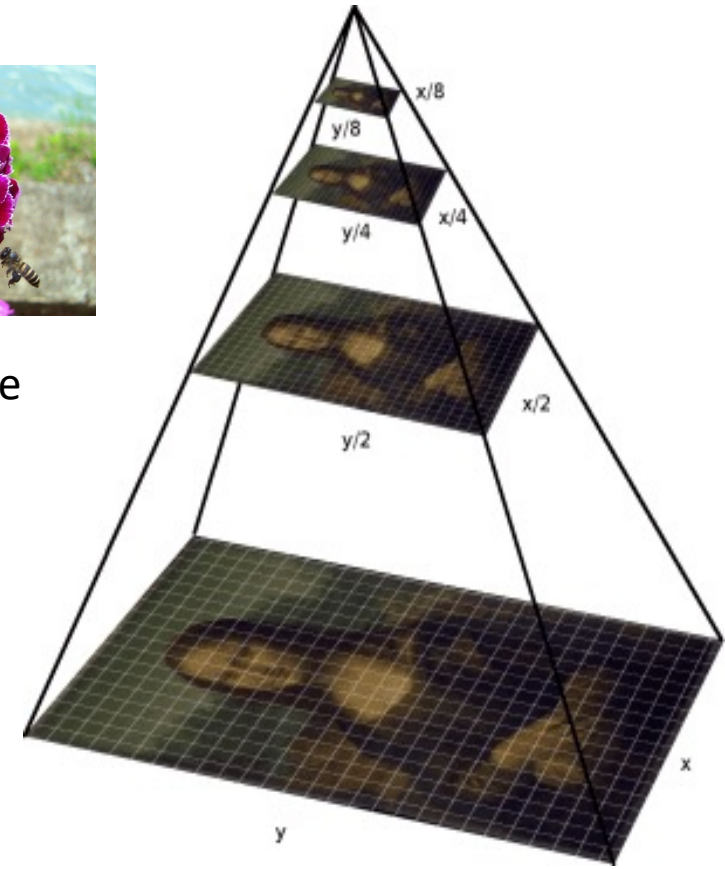
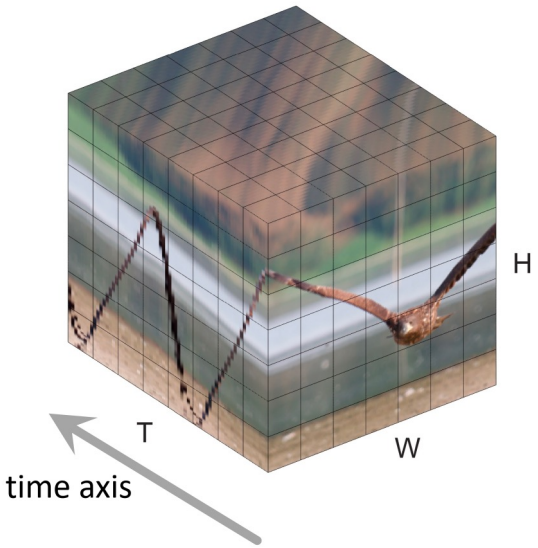


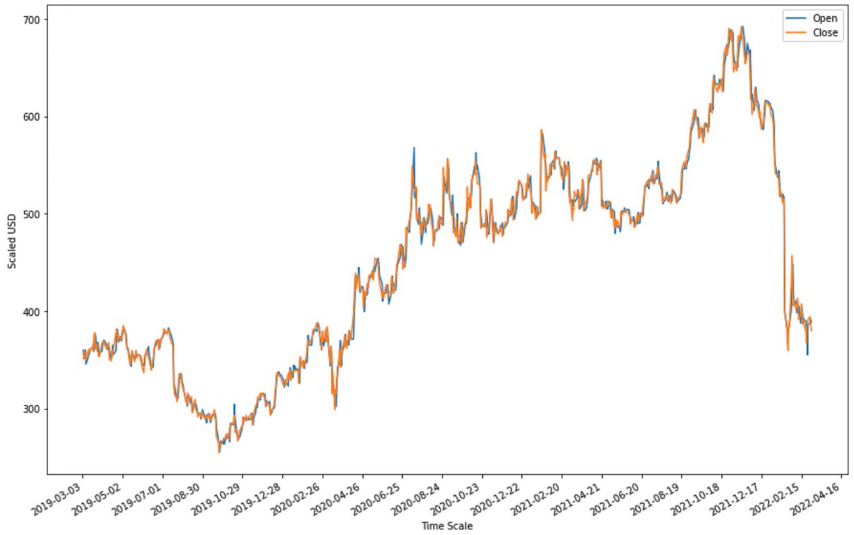
Image Pyramid

U-Net is widely used in image-to-image tasks, e.g. **denoising, super-resolution, and segmentation**.

Recurrent Neural Networks

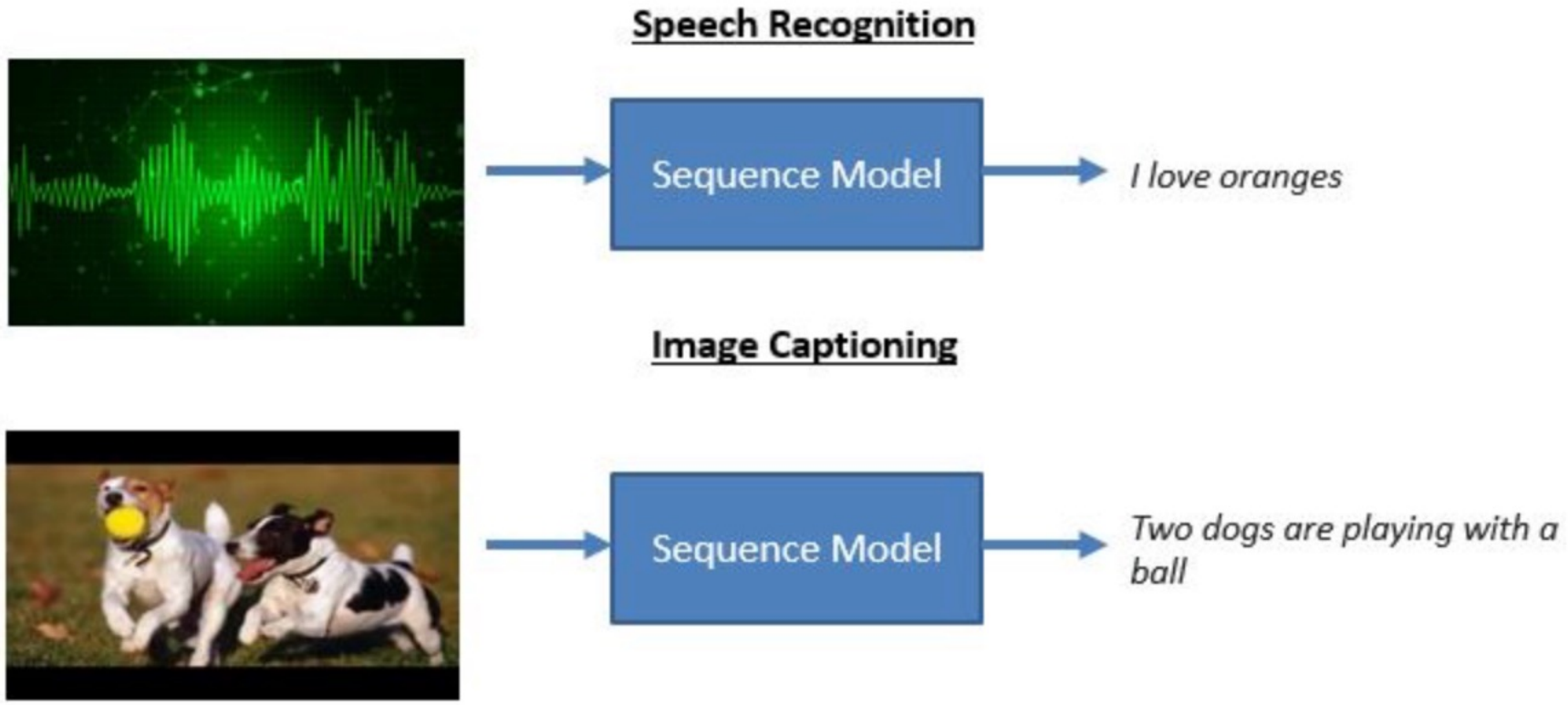
Sequential Data





I love skiing !

Sequence Models



https://www.analyticsvidhya.com/blog/2018/04/sequence-modelling-an-introduction-with-practical-use-cases/

Sequence Models

- DALL-E: text → image

An astronaut riding a horse
in photorealistic style

Sequence Models

- Stock price prediction

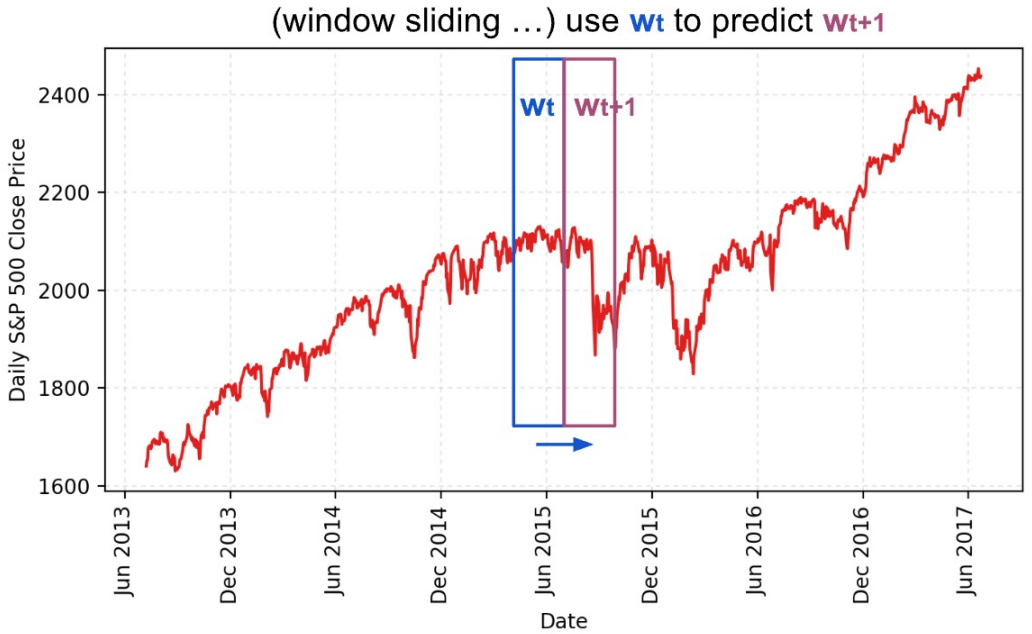
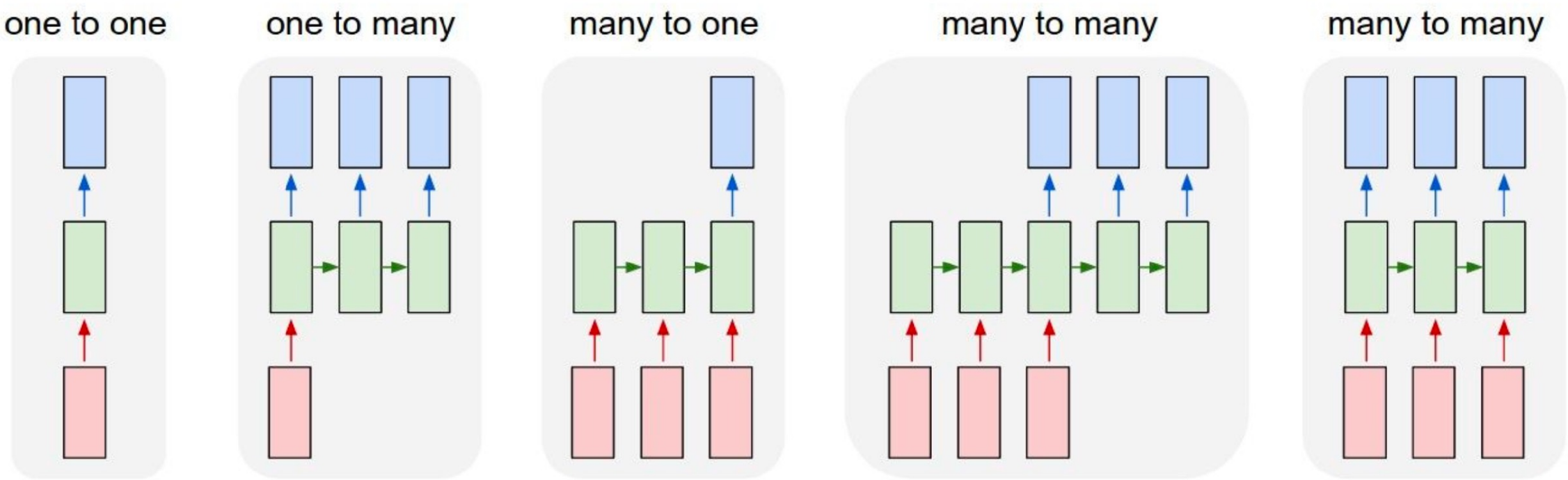


Fig. 1. The S&P 500 prices in time. We use content in one sliding windows to make prediction for the next, while there is no overlap between two consecutive windows.

-- from lilian weng's blog

Sequence Models



Vanilla

image captioning

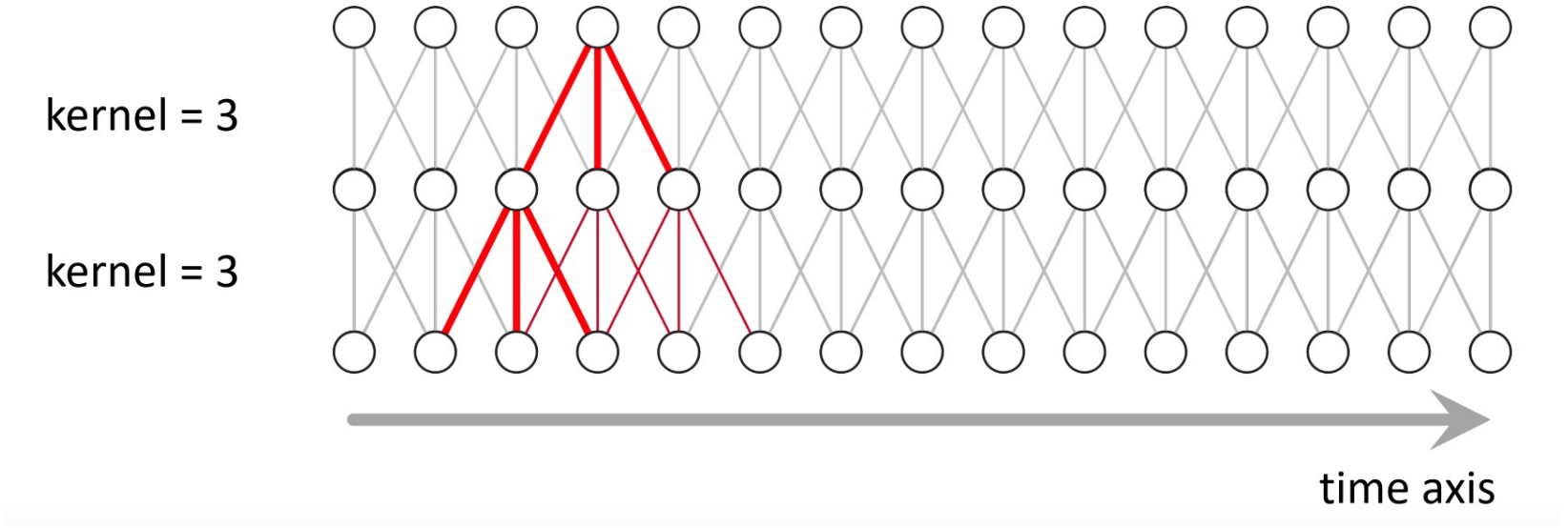
image generation

video captioning

translation

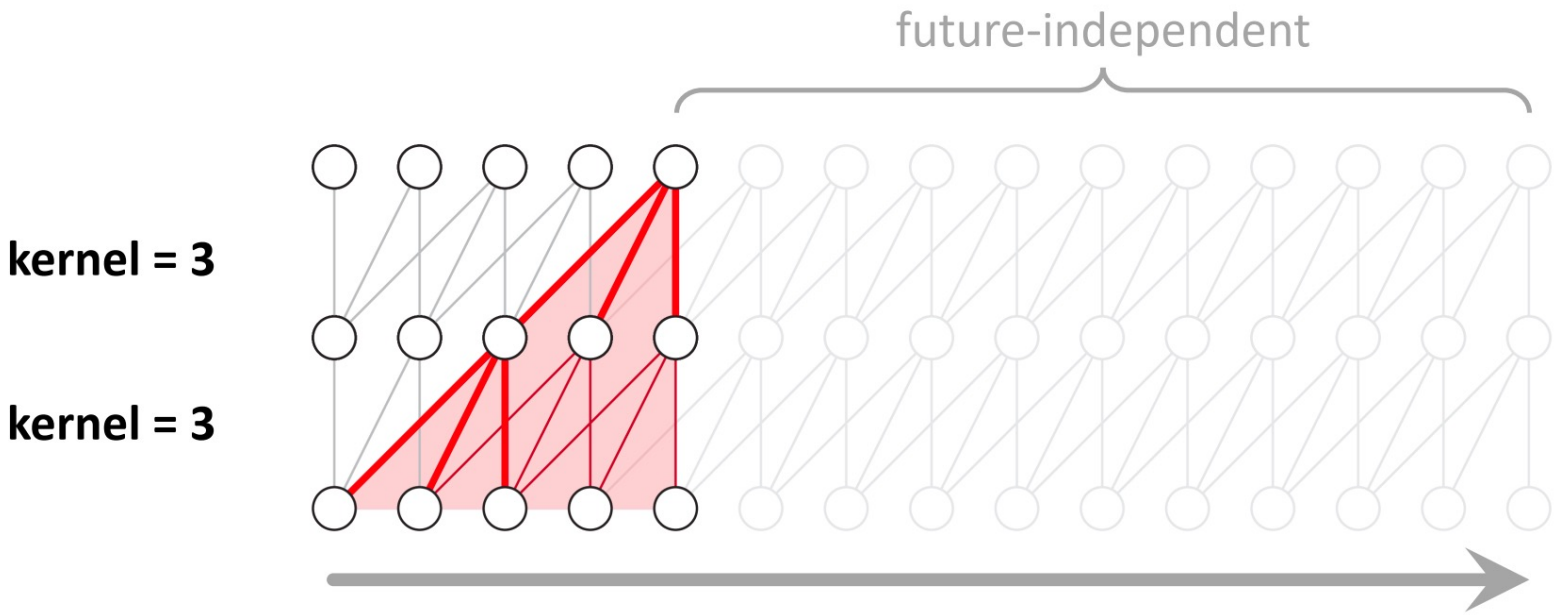
CNNs

- convolution along time axis



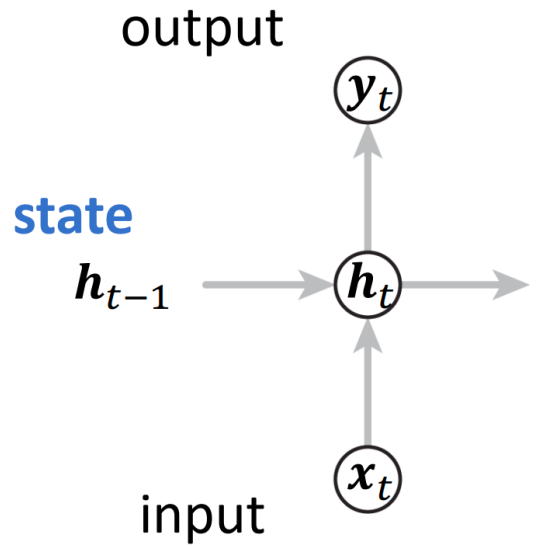
CNNs

- Partial Convolution: current states only depend on the previous states



RNNs

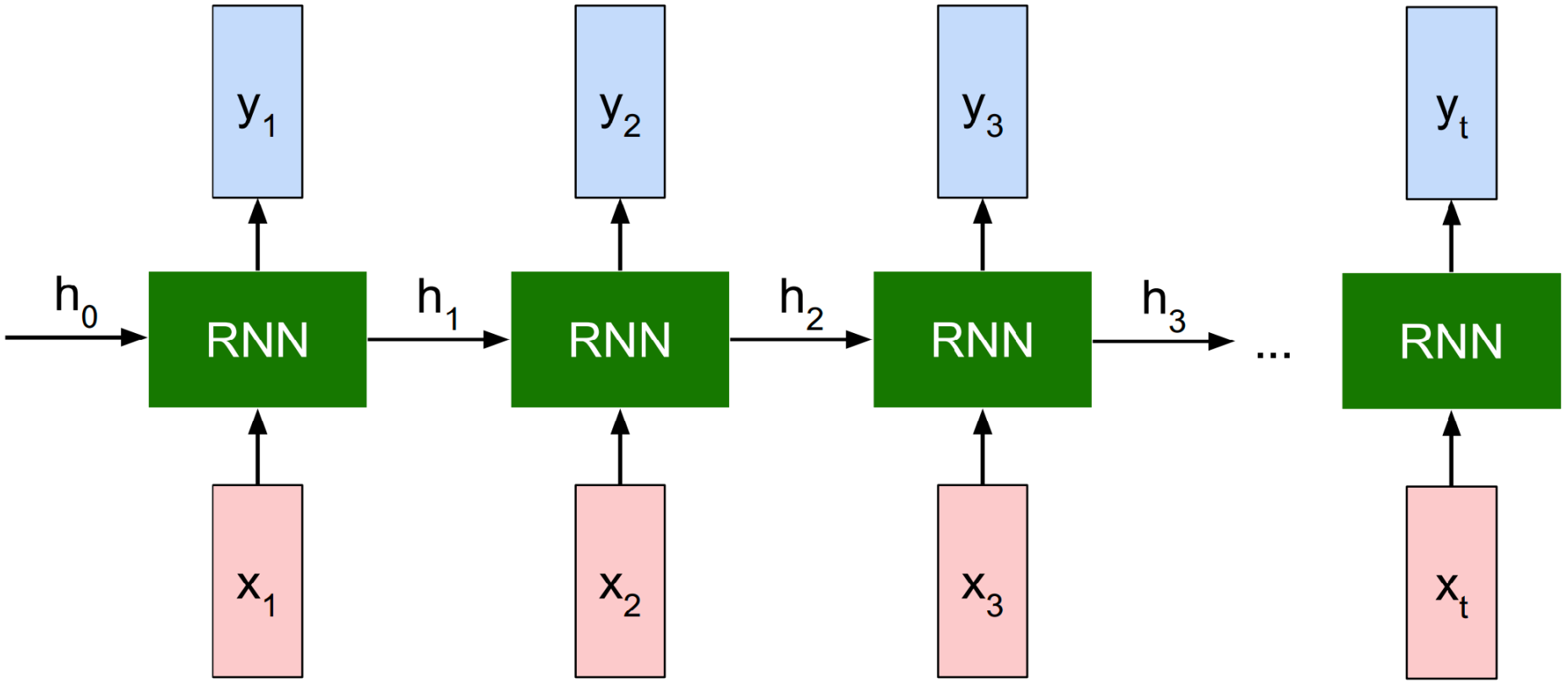
- CNNs are hard to handle long sequences: receptive field enlarges with depth.
- Recurrent Neural Network (RNN): introduce hidden state h to stores previous information.



$$h_t = f(h_{t-1}, x_t)$$
$$y_t = g(h_t)$$

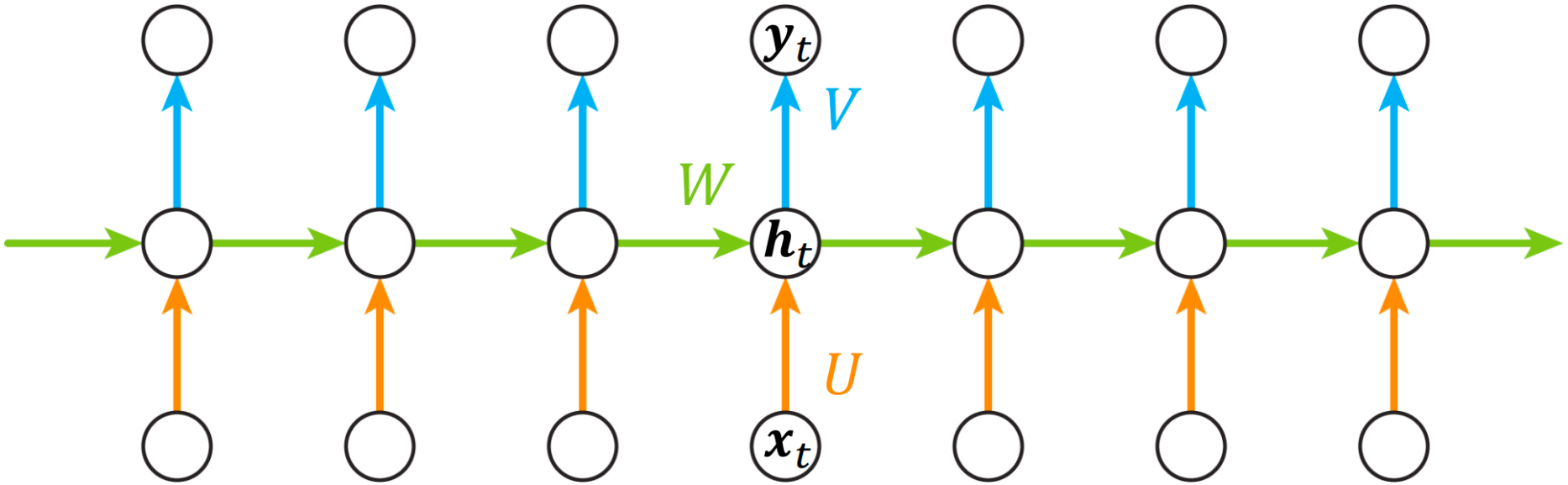
Recurrent: same f, g for different t

RNNs



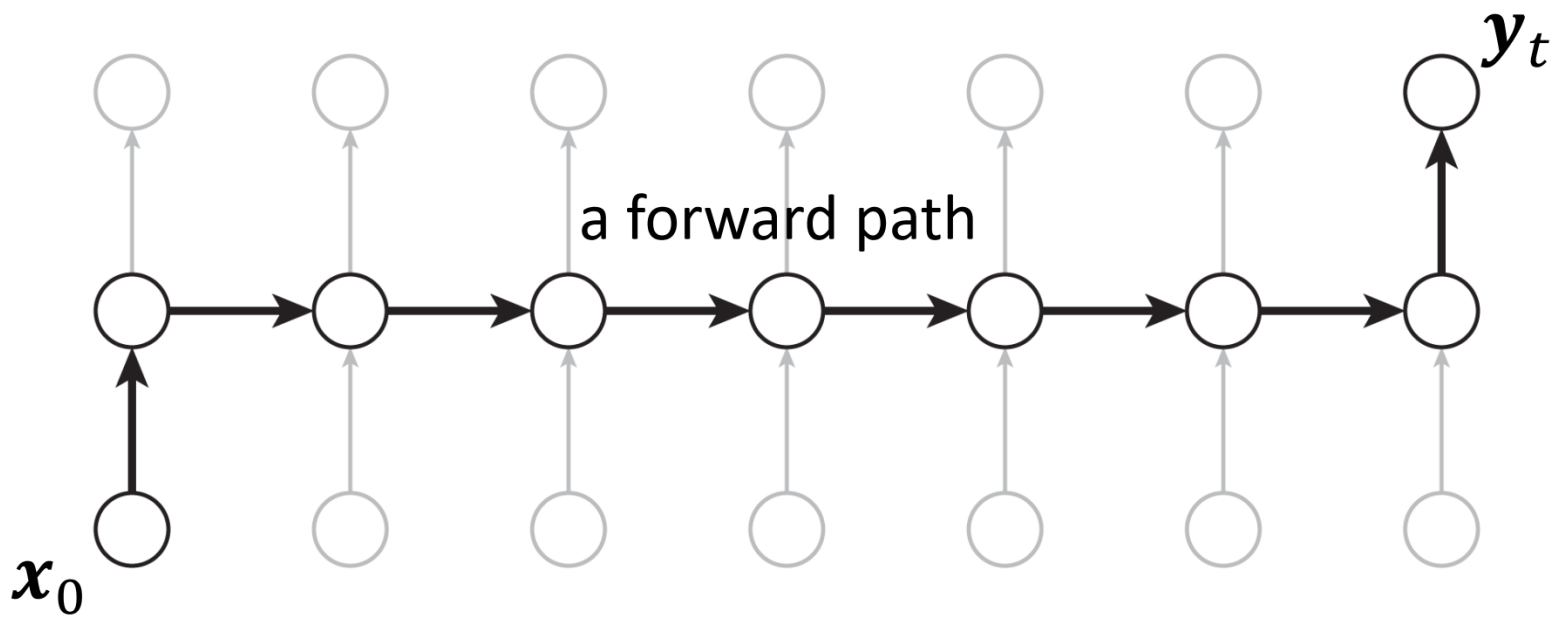
Vanilla RNN

$$\boldsymbol{h}_t = \tanh(W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t)$$
$$\boldsymbol{y}_t = V\boldsymbol{h}_t$$

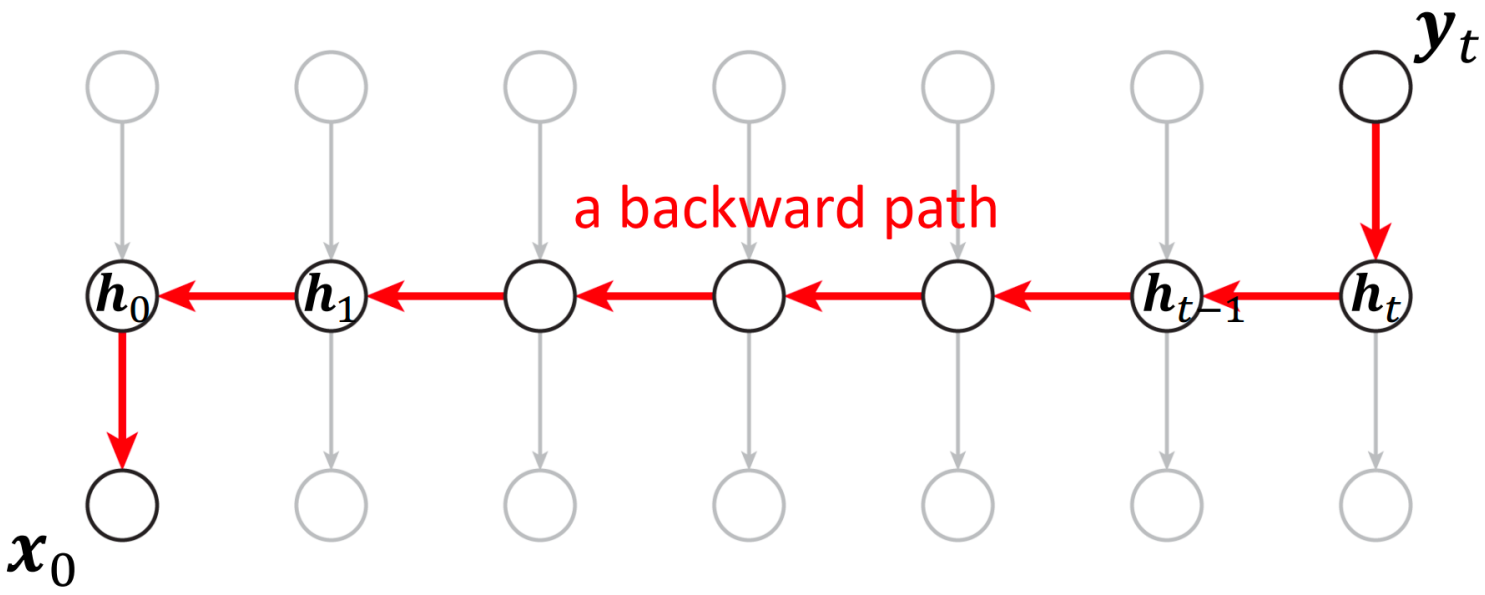


RNNs

- t-layer deep net



RNNs: Back Propagation



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0}\frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

$$h_t = \sigma(\mathrm{Wh_{t-1}} + \mathrm{Ux_t})$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \sigma'^{(\mathrm{Wh_{t-1}}+\mathrm{Ux_t})} W$$

shared W for all time t!

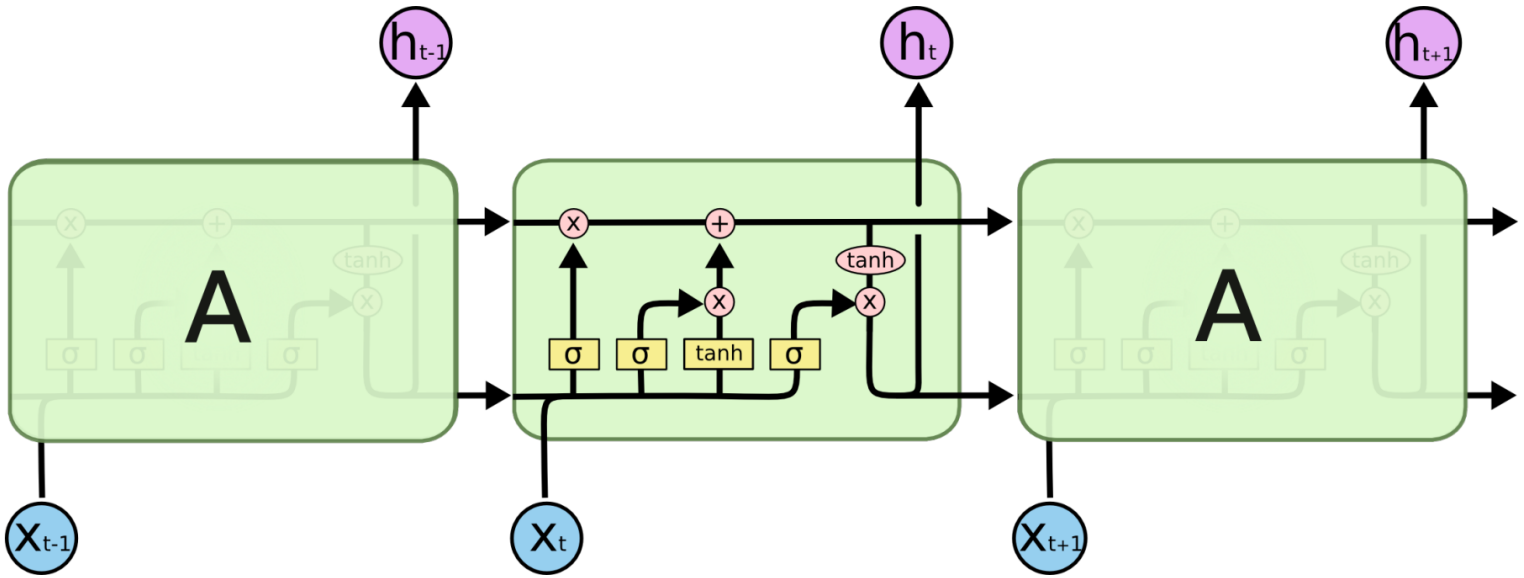
Omitting non-linear layer:

$$\frac{\partial y_t}{\partial x_0} = \frac{\partial y_t}{\partial h_t} W^{t-1} \frac{\partial h_0}{\partial x_0}$$

Gradient exploding or vanishing!

LSTMs

- Long Short Term Memory (LSTM): capable of learning long-term dependencies

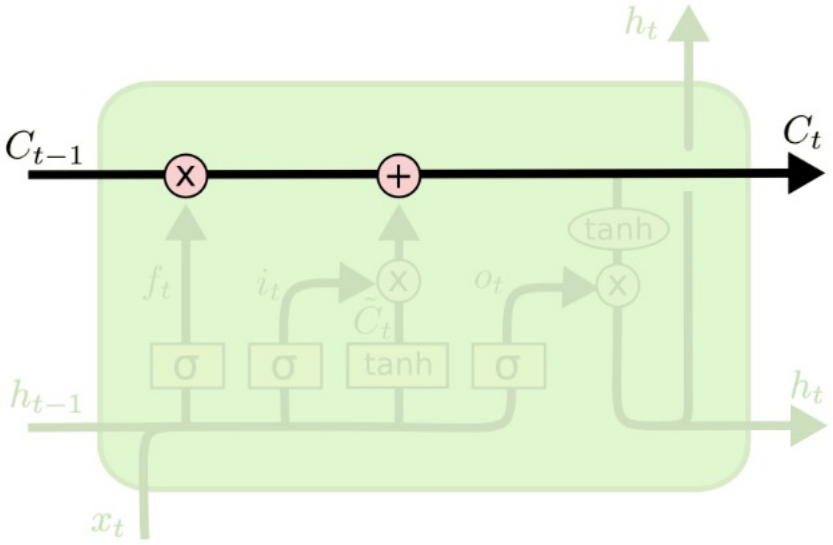


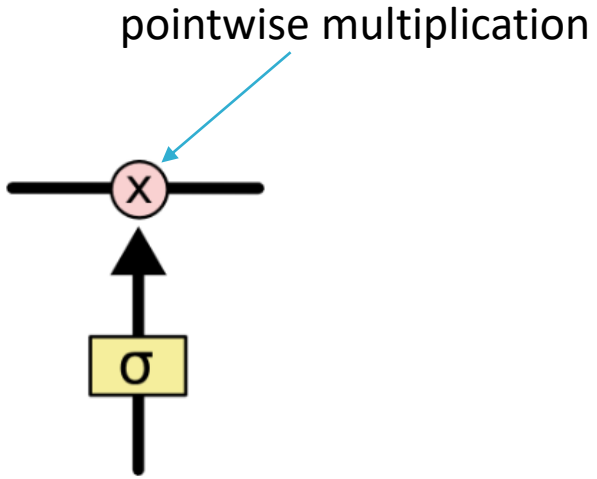
Hochreiter and J. Schmidhuber. Long short-term memory,1995

Picture from Blog: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Core Idea behind LSTMs

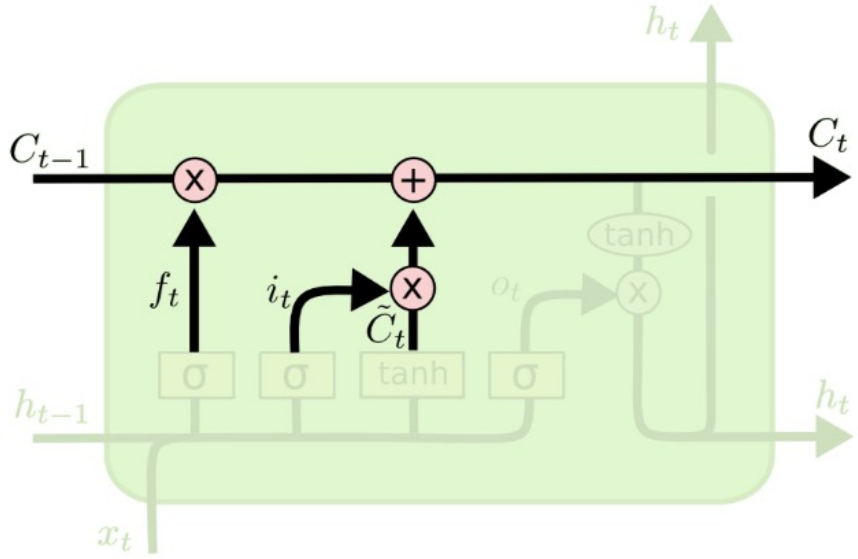
- **The cell state:** like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.
- **The Gates:** optionally let information through.





Blog: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

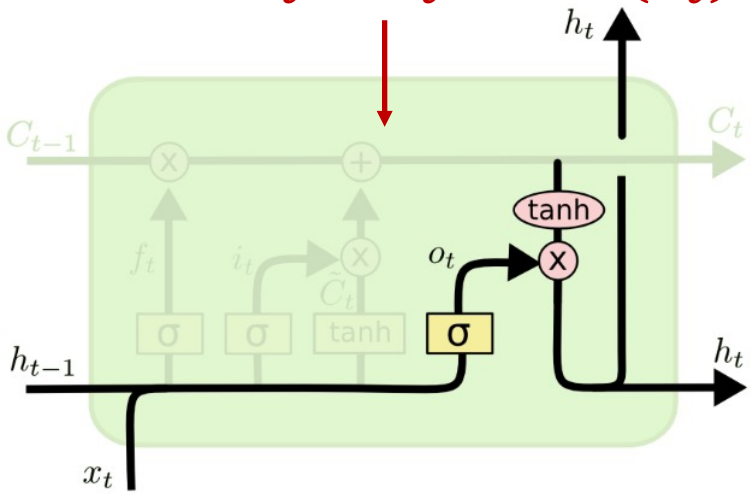
LSTM



$$\begin{pmatrix} i \\ f \\ o \\ \tilde{C}_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} \begin{pmatrix} W_i \\ W_f \\ W_o \\ W_c \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$



Gate i, f, o: control the information flow

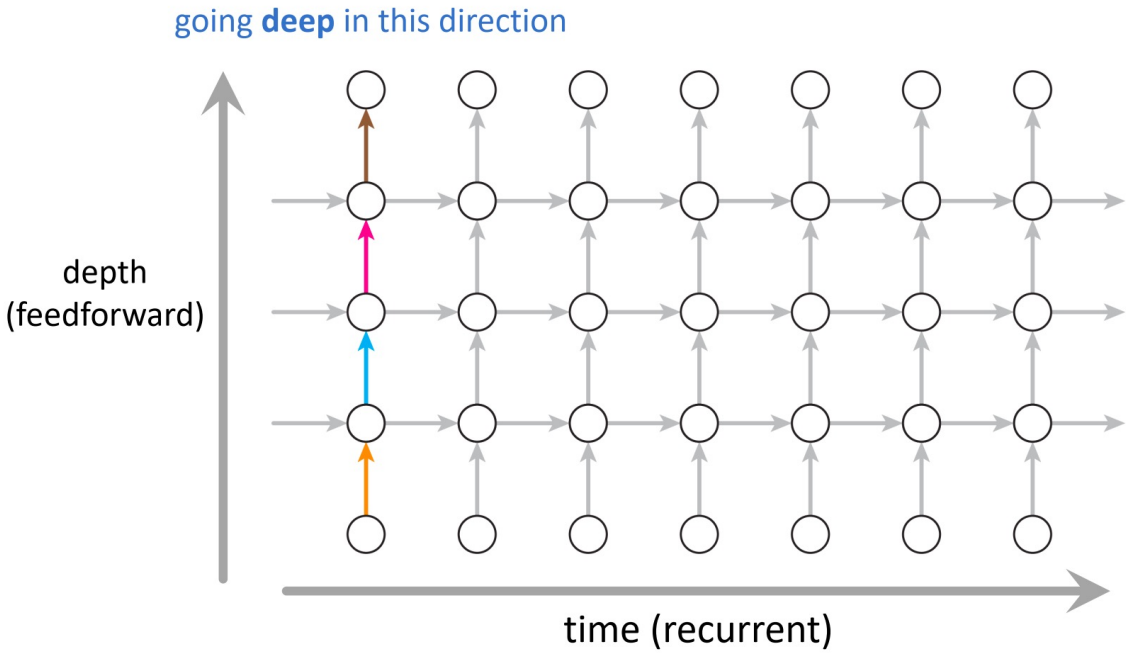
Input gate i

forget gate f: $f_t = 1$, like ResNet!

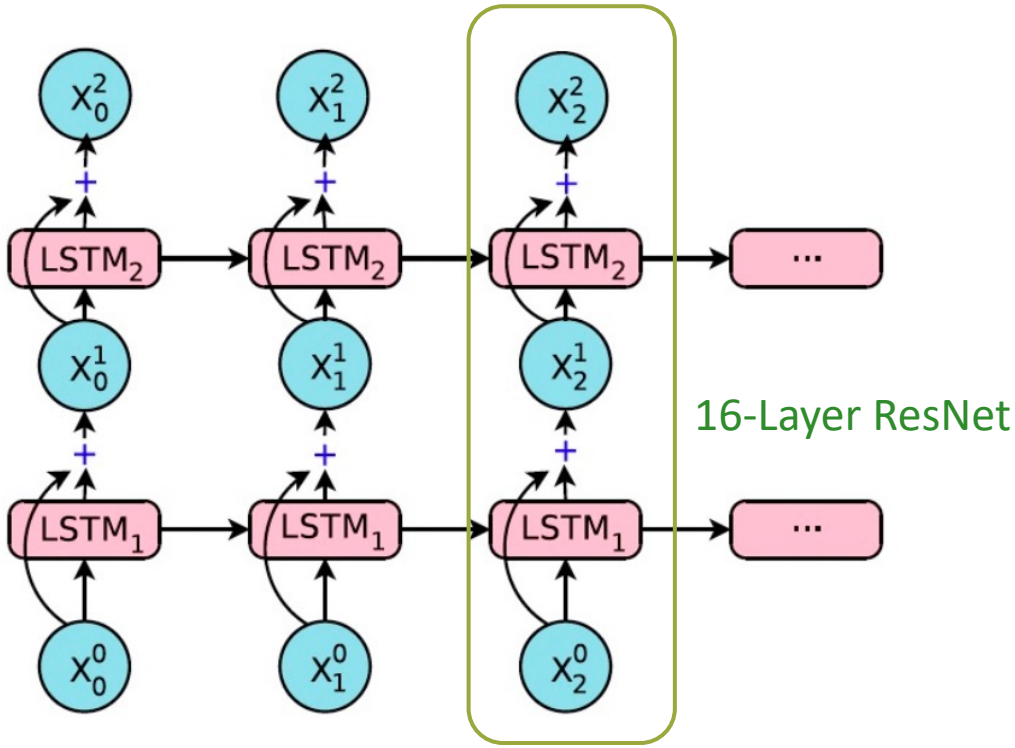
output gate o

Blog: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Deep RNN

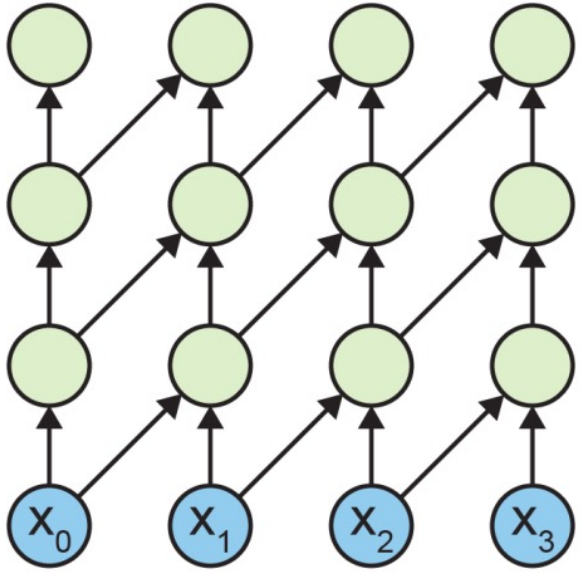


Google's Neural Machine Translation System



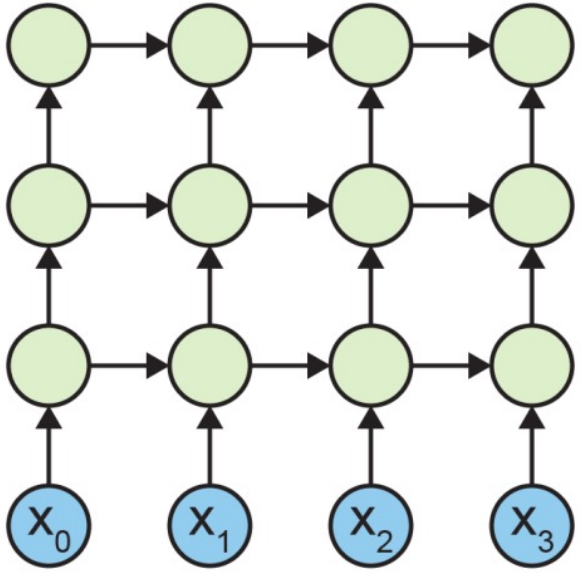
"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", Wu, et al., 2016

CNN vs. RNN



CNN

Short context
Parallel computing



RNN

Long context
Computing states sequentially, not friendly for parallel computing.

Transformer

Vision Attention

- **V1 (Primary Visual Cortex):** Processes basic visual features (edges, orientation)
- **V1 Saliency Hypothesis**: V1 transforms the visual inputs into a saliency map of the visual field to guide visual attention or direction of gaze.

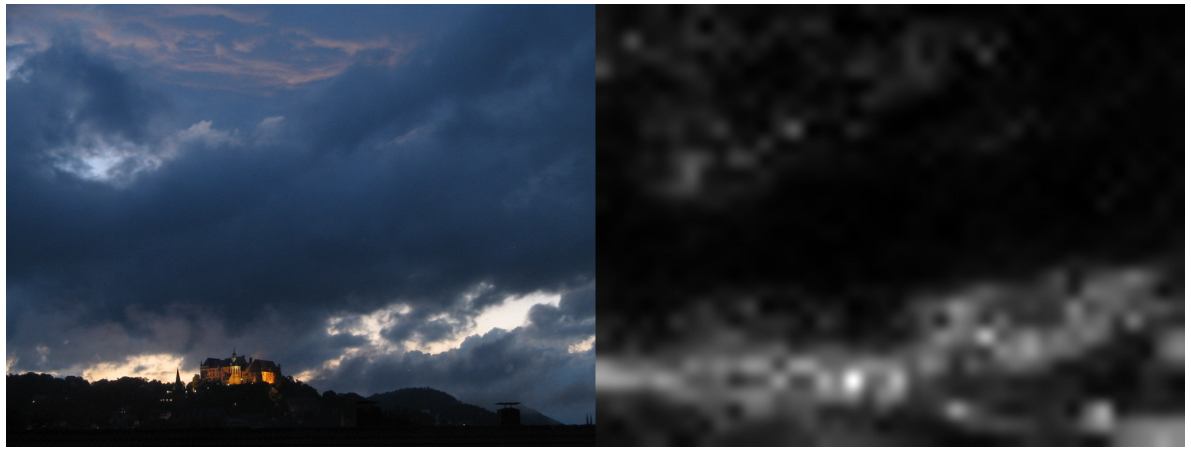
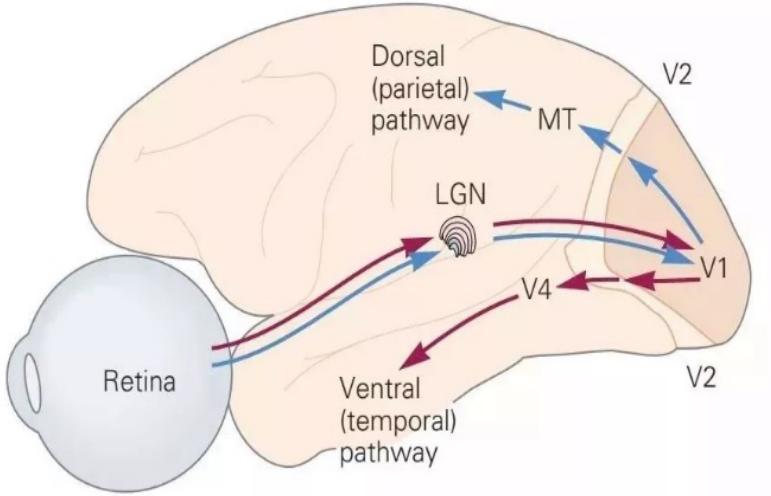


Illustration: Saliency map in biological view

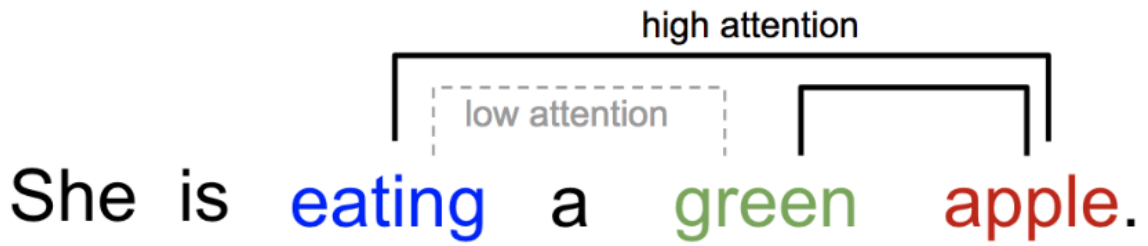


人类视觉系统中的主要成像通路

（图片来自*Principles of Neural Science*第五版）

Context Attention

- One word "attends" to other words in the same sentence differently.



https://lilianweng.github.io/posts/2018-06-24-attention/

- Context information in different sentence is not of fixed length.

Attention

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

- Attention in deep learning: a vector of importance weights.
- First proposed by Bahdanau et al. for translation: automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word

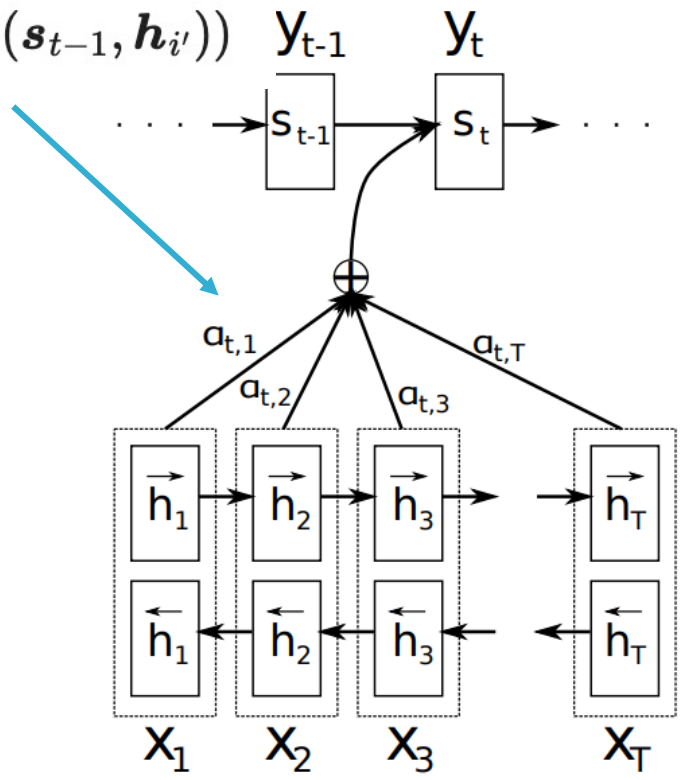


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

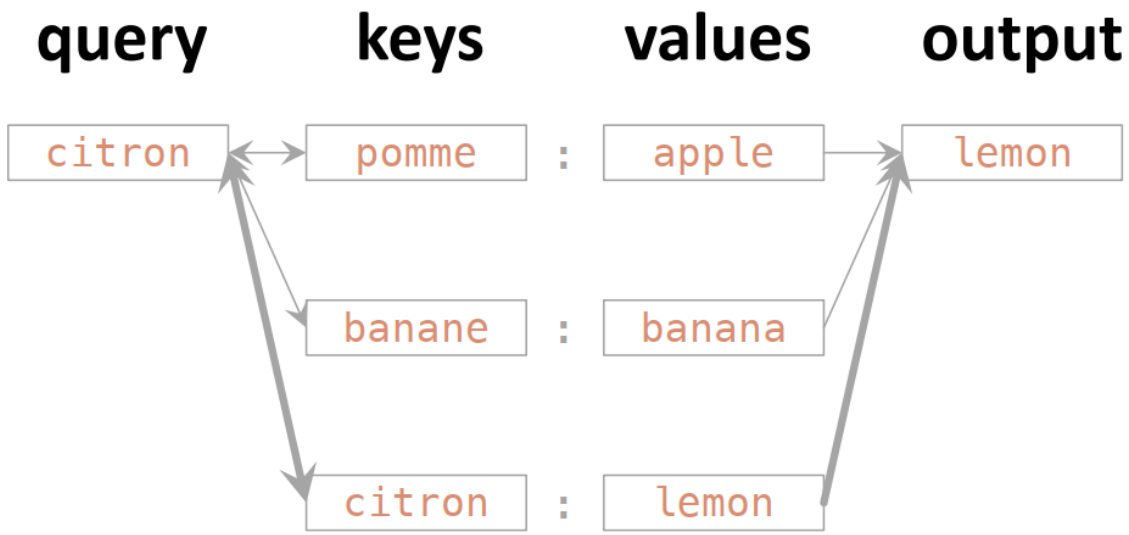
Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate.” ICLR 2015.

Transformer

- Ashish Vaswani, et al. “Attention is all you need.” NIPS 2017.

Attention

```
dict_fr2en = {
    "pomme": "apple",
    "banane": "banana",
    "citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



Attention

Attention: soft inquiry

1. query-key matching

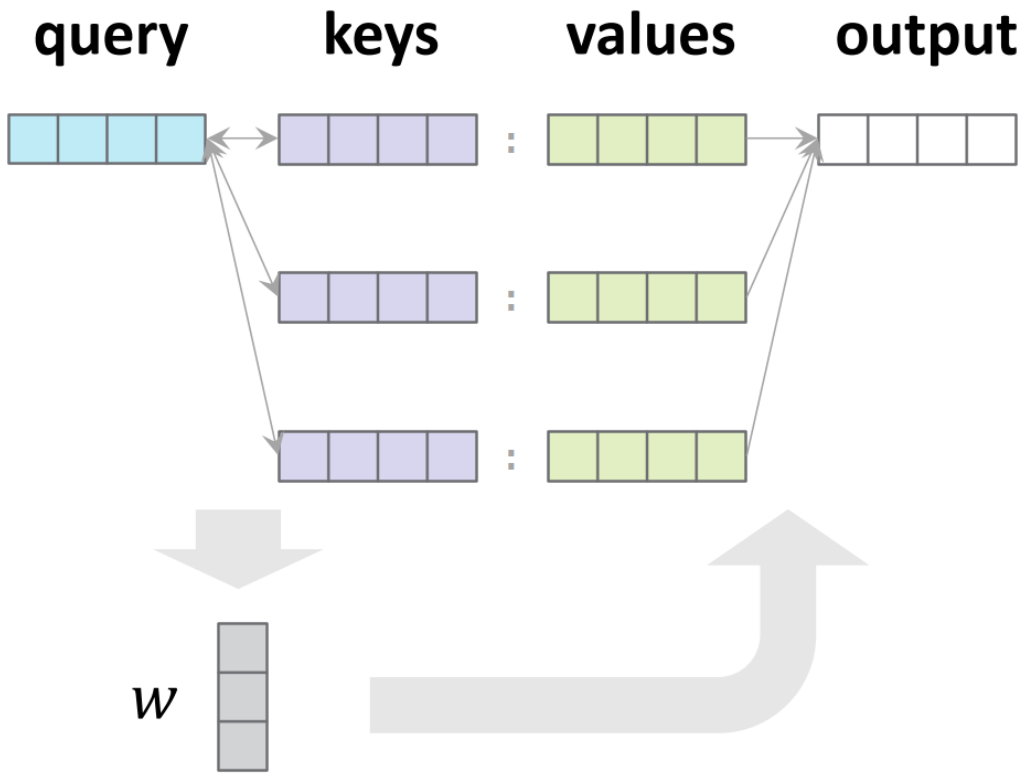
$$s_i = q_i \cdot k_i$$

2. compute weights(attention)

$$\mathrm{w}_i = softmax(s_i)$$

3. weighted averaging

$$z = \sum_i w_i v_i$$



Attention

- multiple queries

1. query-key matching

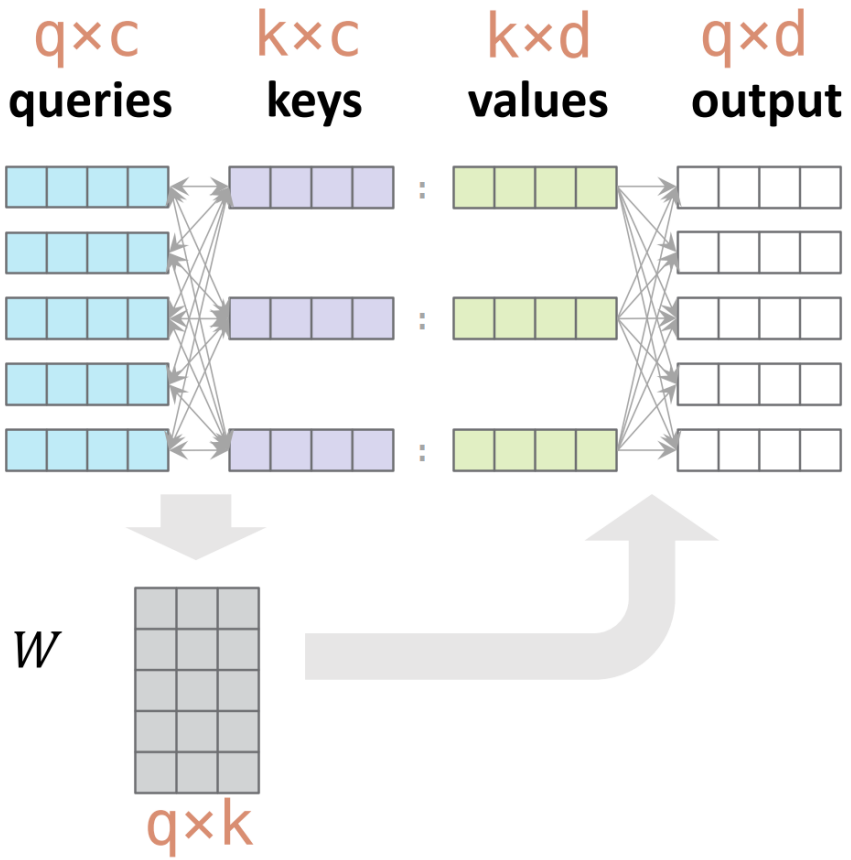
$$S = QK^T$$

2. compute weights(attention)

$$W = softmax(\frac{S}{\sqrt{c}})$$

3. weighted averaging

$$Z = WV$$



Scaled Dot-product: for independent q_i, k_i, with mean 0 and variance 1, scaled dot-product also has mean 0 and variance 1.

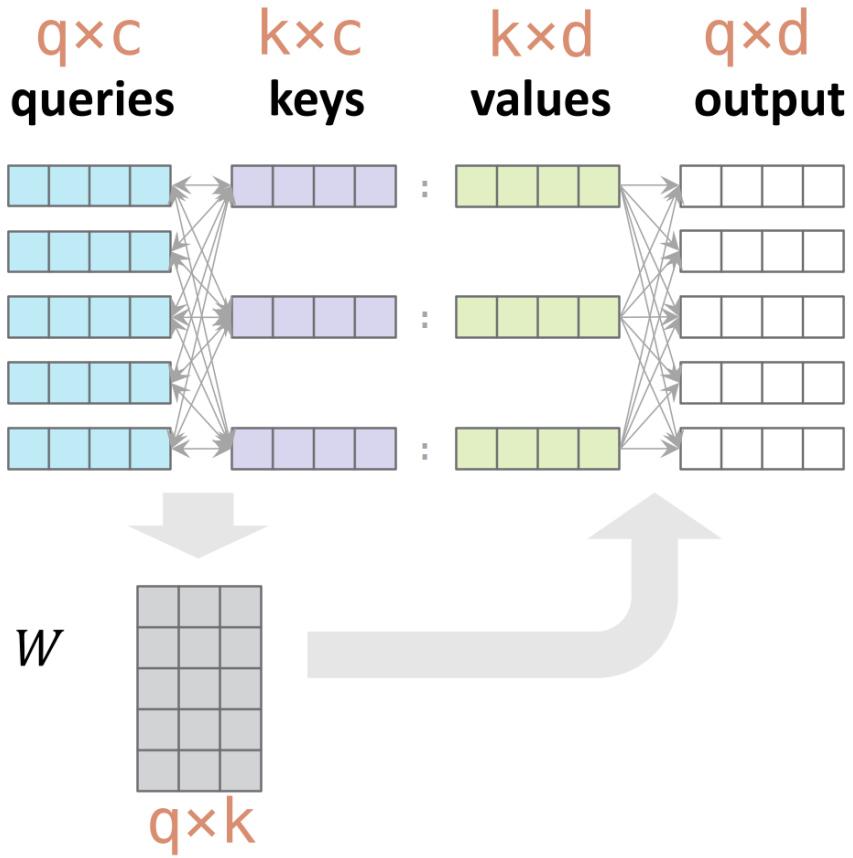
Attention

- no parameters
- # queries = # output

  # keys = # values

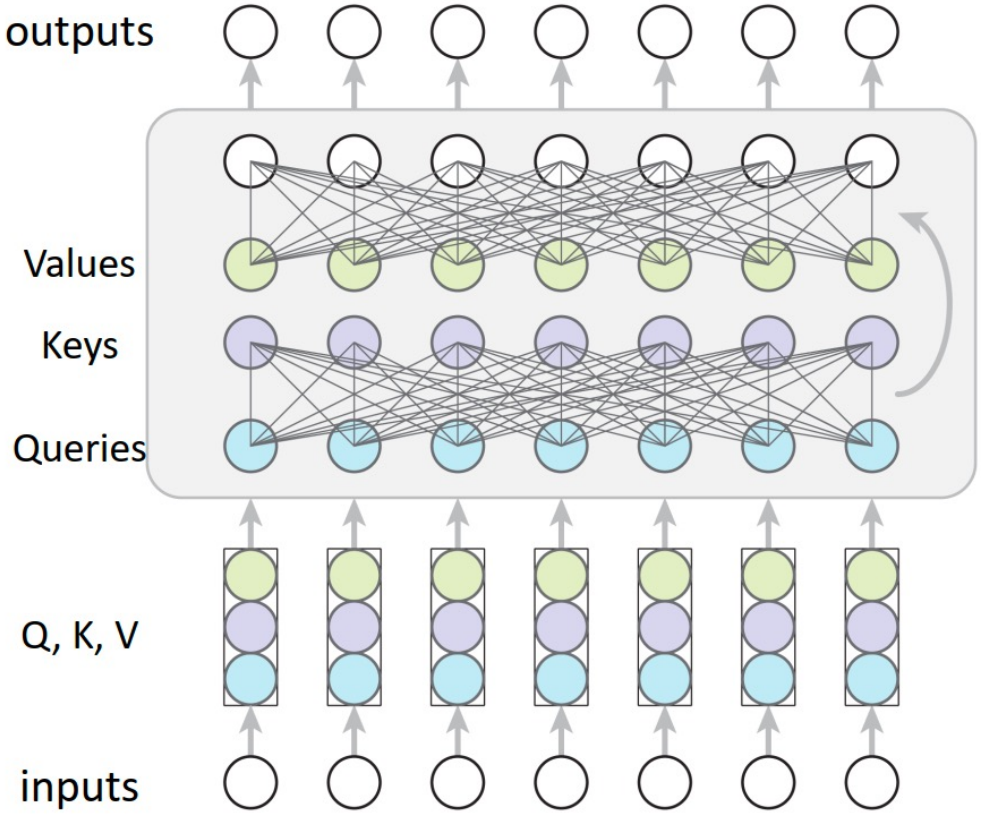
  key channels = query channels

  value channels = output channels



Self-attention

- Q, K, V are computed from one input sequence.

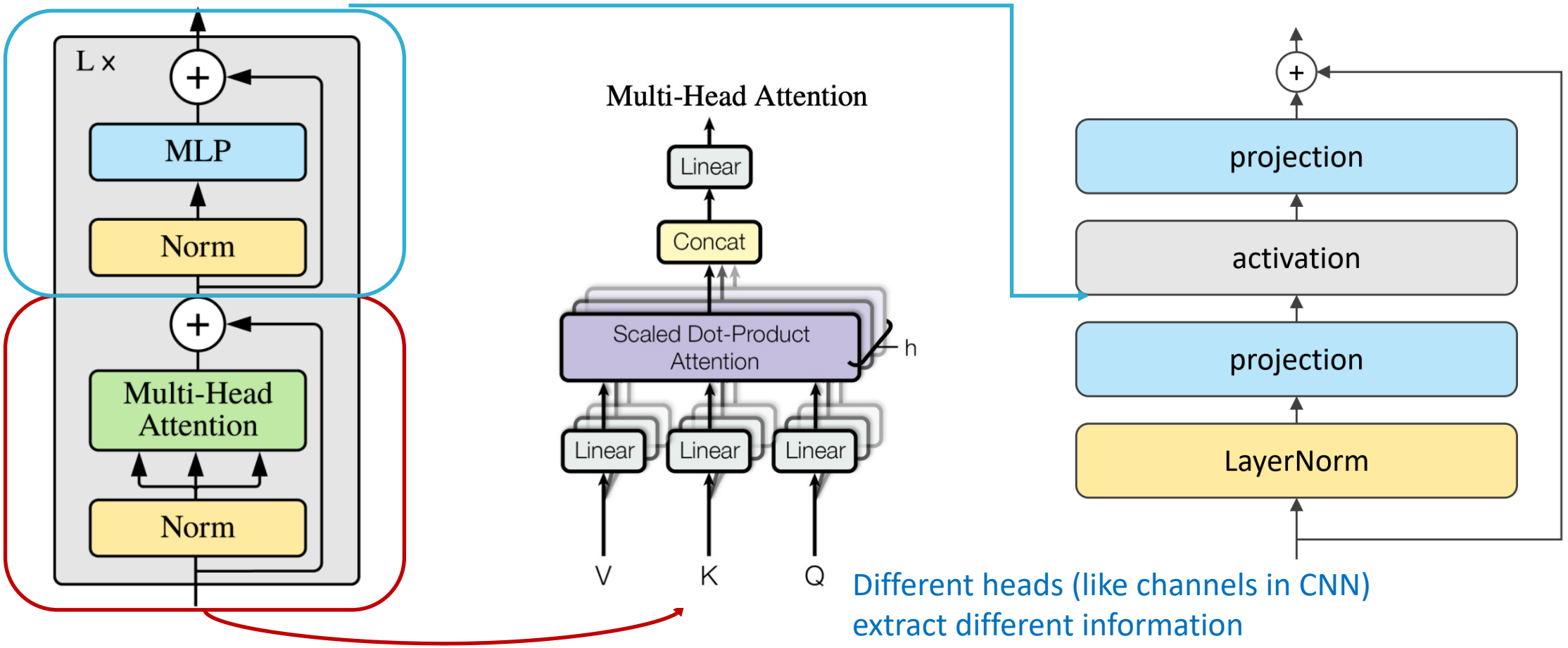


Input: $\mathrm{X} = (X_t)_t$;

- Long Context: compute attention across the whole sequence.
- Weight: W_q, W_k, W_v are shared by all X_t

$$Q_t = \mathrm{X_t W_q} \in \mathbb{R}^{n \times m}$$
$$K_t = X_t W_k \in \mathbb{R}^{n \times m}$$
$$V_t = \mathrm{X_t W}_v \in \mathbb{R}^{n \times d}$$

Transformer



Different heads (like channels in CNN) extract different information

Vaswani, et al., “Attention is all you need”, 2017

Transformer

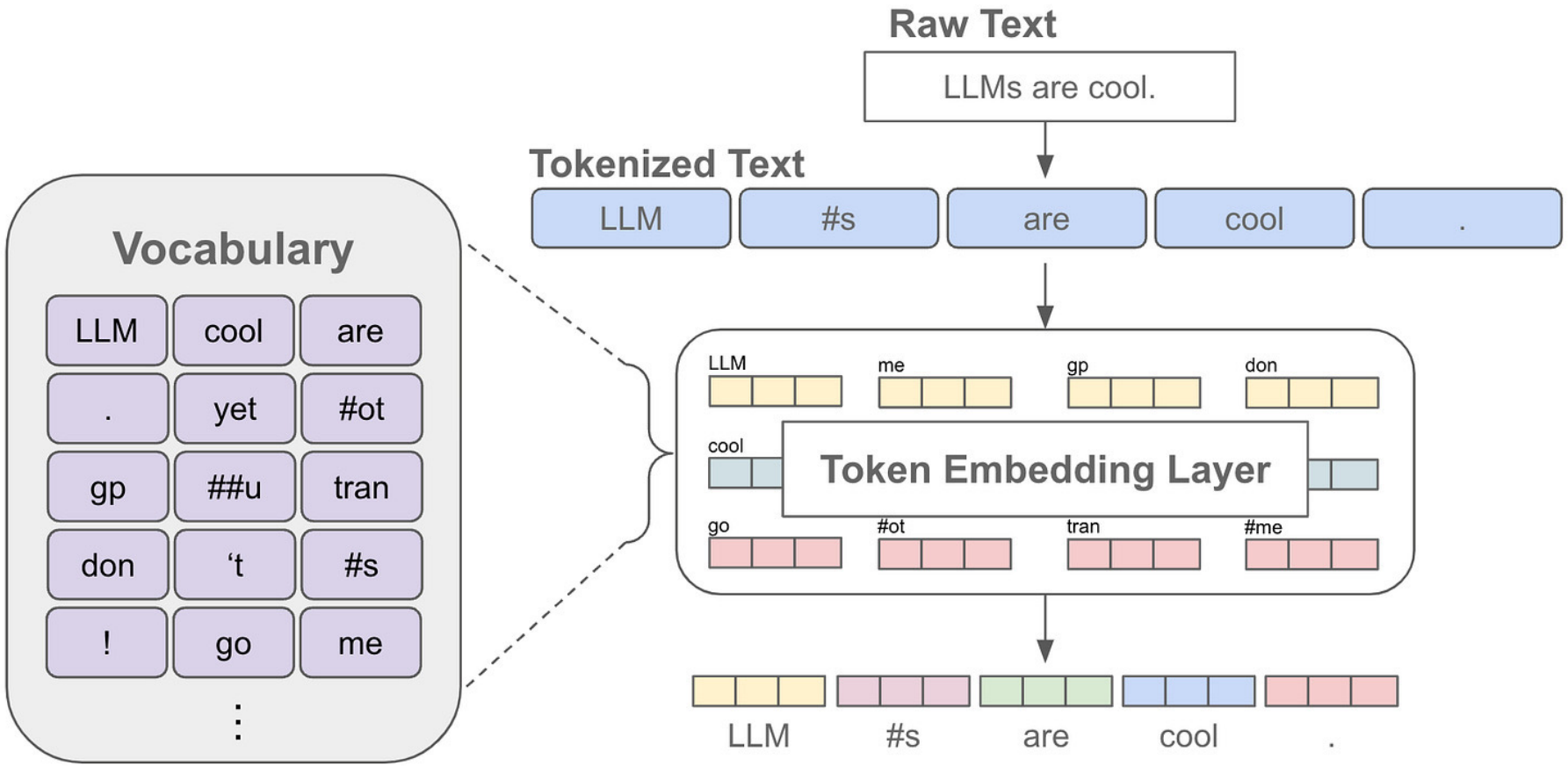
- Attention: global connection
- Projections: compute matrix multiplication within each token; sharing the matrixes across tokens

$$Q = XW_q, K = XW_k, V = XW_v$$

W_q, W_k, W_v are shared for all tokens

- Allow for parallel computation!

Token Embedding



https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

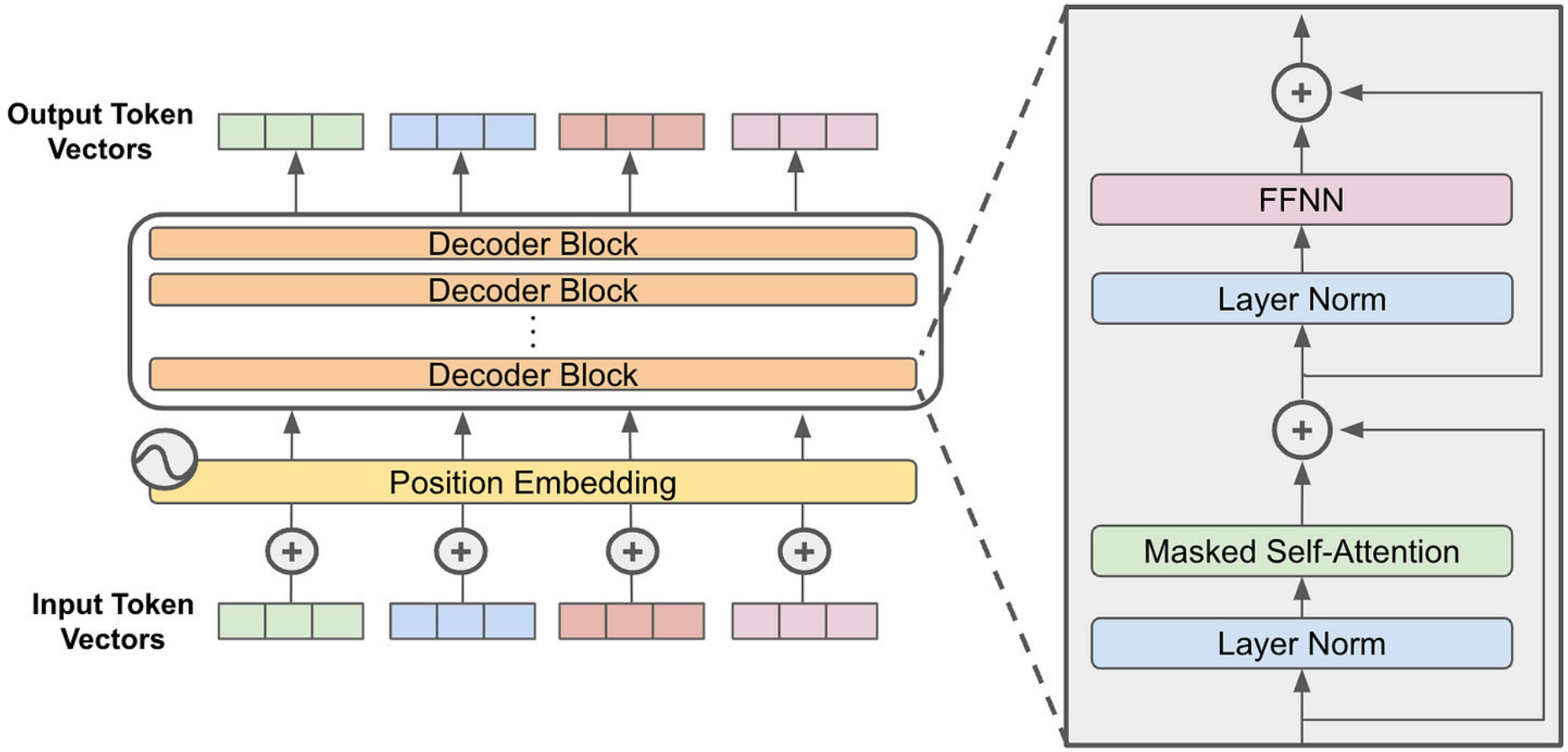
Position Embedding

- Transformer contains no recurrence and no convolution
- In order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence

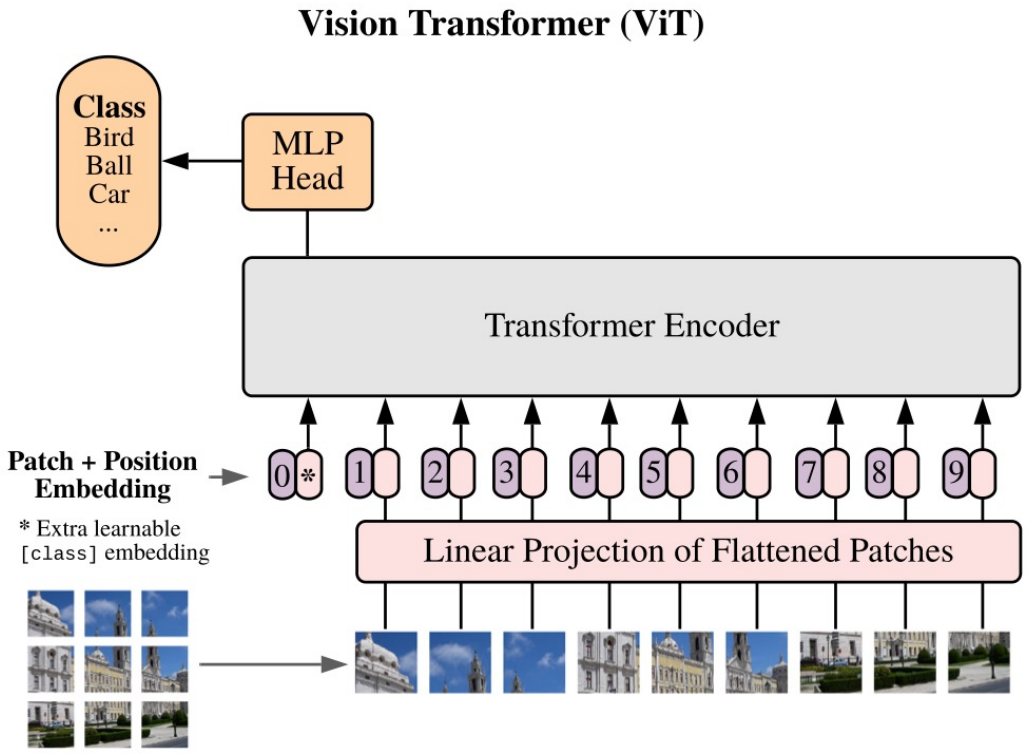
$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right),$$
$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right).$$

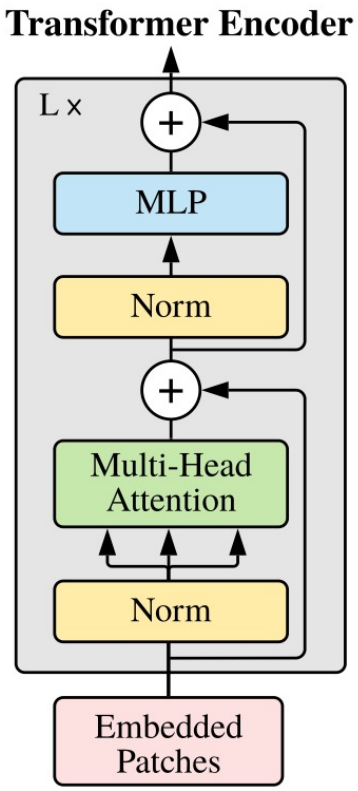
$P \in \mathbb{R}^{n \times d}$: n *tokens*, d *dimensional embedding*

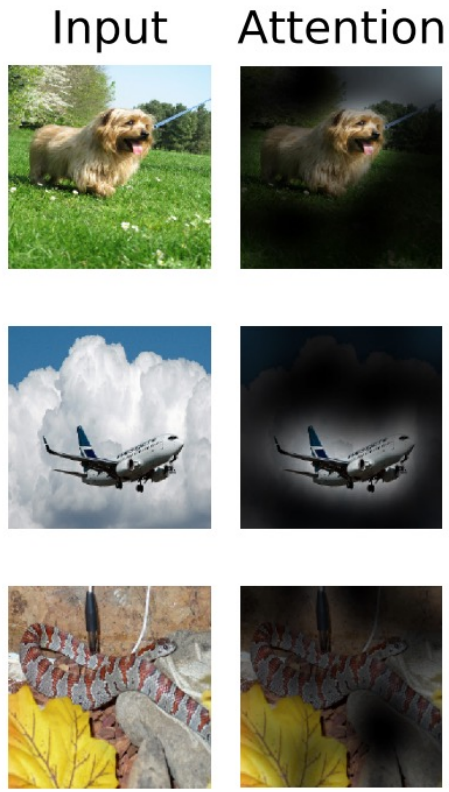
Decoder-only Transformer



Vision Transformer





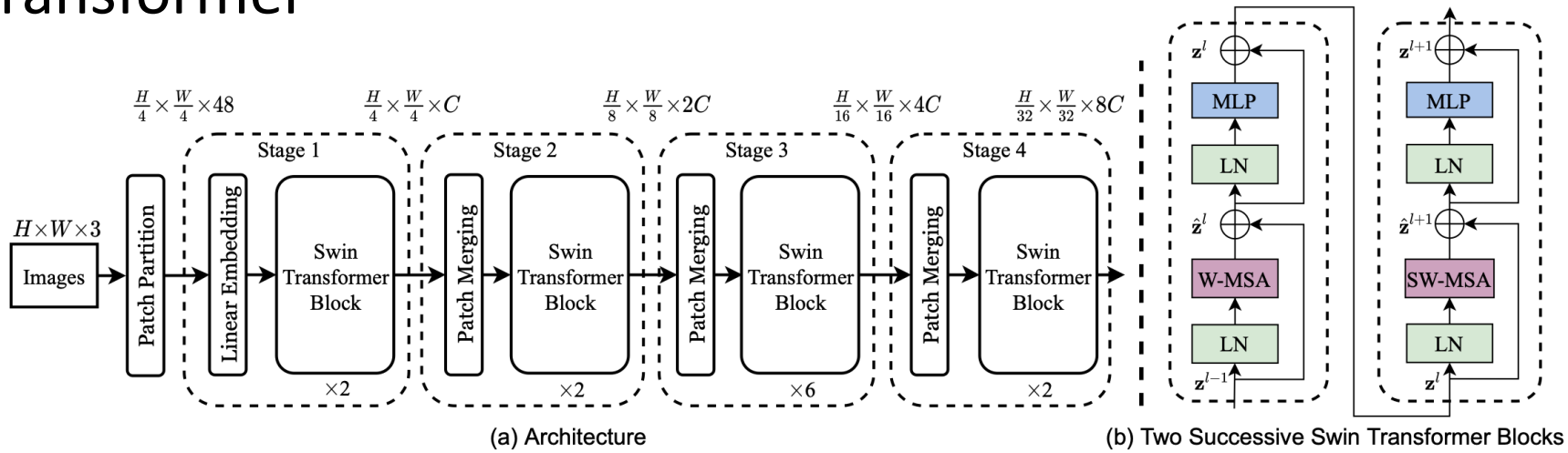


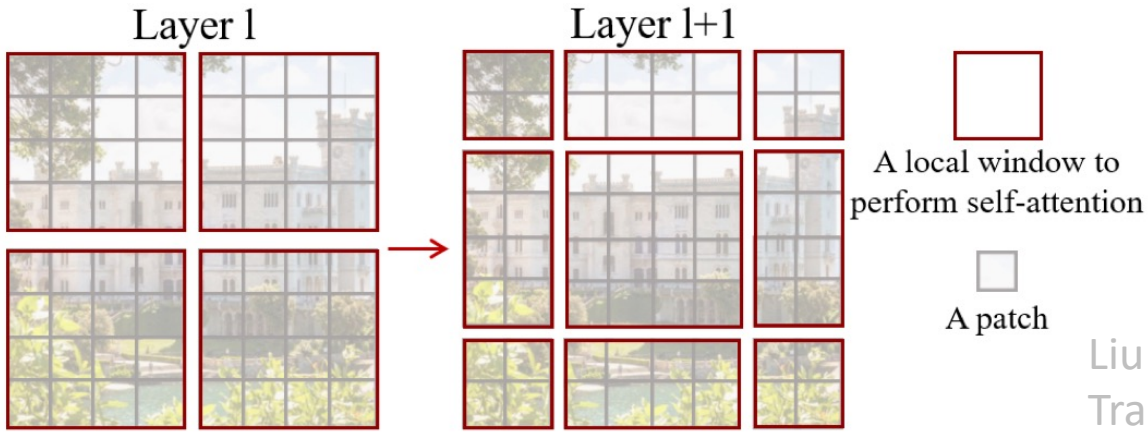
Too many tokens in one image; the computation is slow.

Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020

Vision Transformer

- Swin Transformer



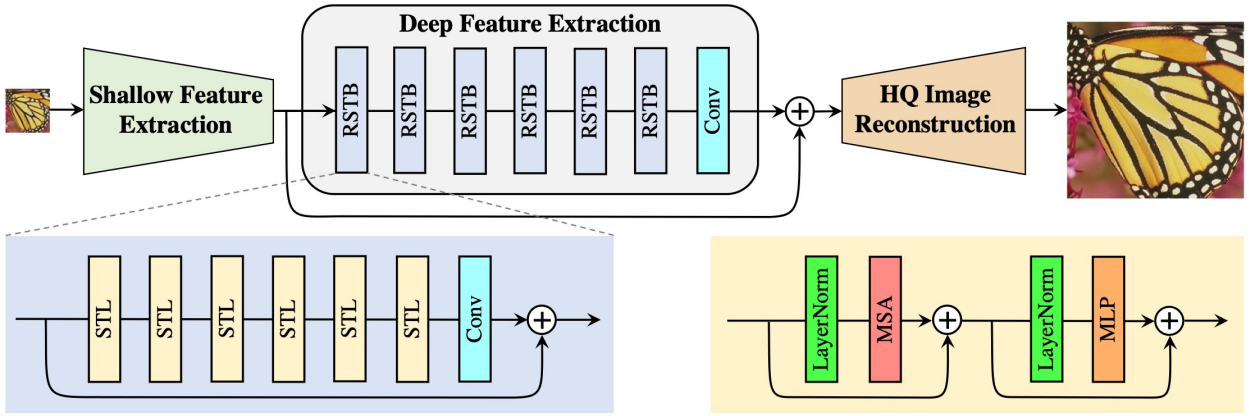


- Only perform self-attentions within each window
- By shifting the window, boundary patches can interact with more patches to compute attentions.

Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021

Vision Transformer

- SwinIR (Swin for image restoration)



(a) Residual Swin Transformer Block (RSTB)

(b) Swin Transformer Layer (STL)

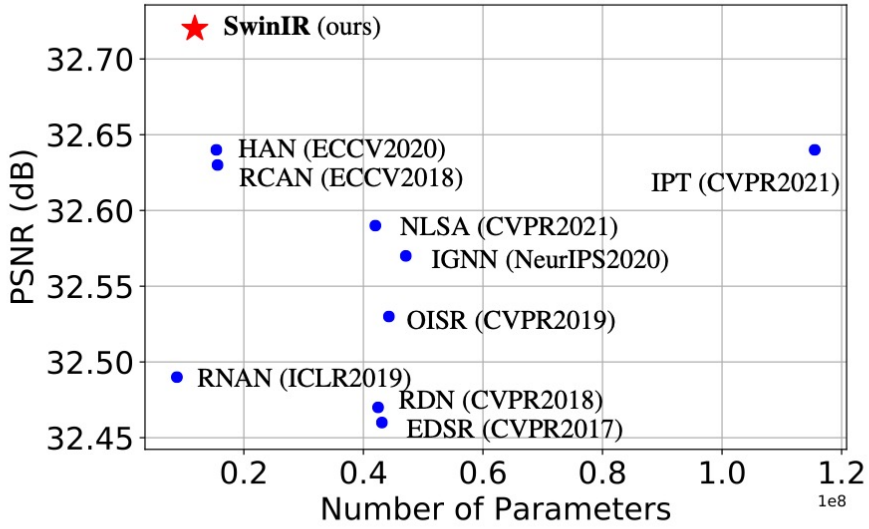


Figure 1: PSNR results v.s the total number of parameters of different methods for image SR ($\times 4$) on Set5 [3].

Liang et al. "SwinIR: Image Restoration Using Swin Transformer", arXiv, 2021