Network structures: CNN, ResNet, DenseNet

Shikui Tu

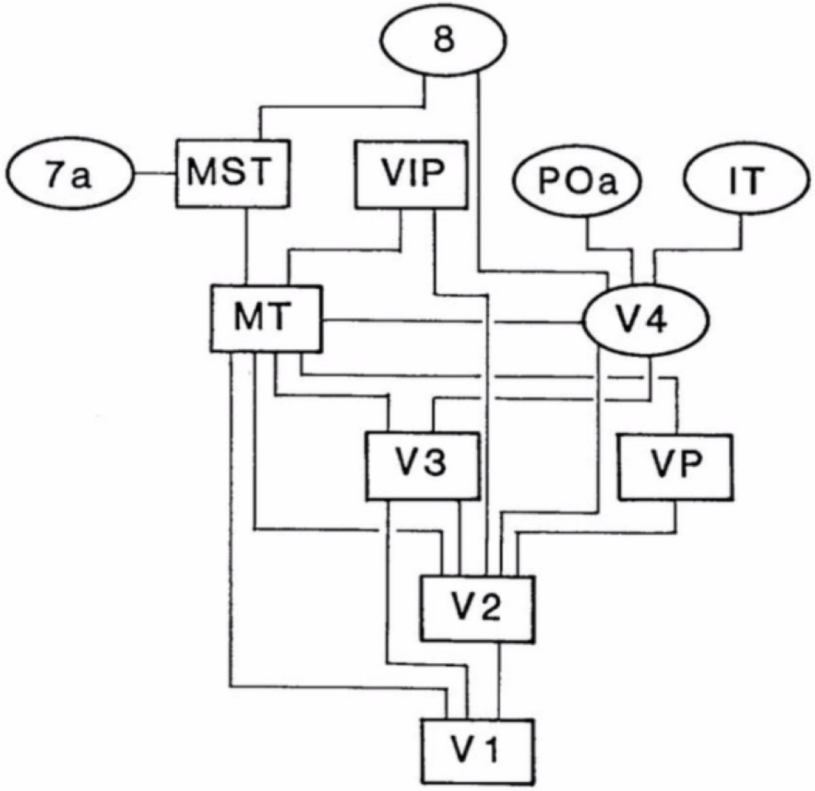
Department of Computer Science and Engineering, Shanghai Jiao Tong University

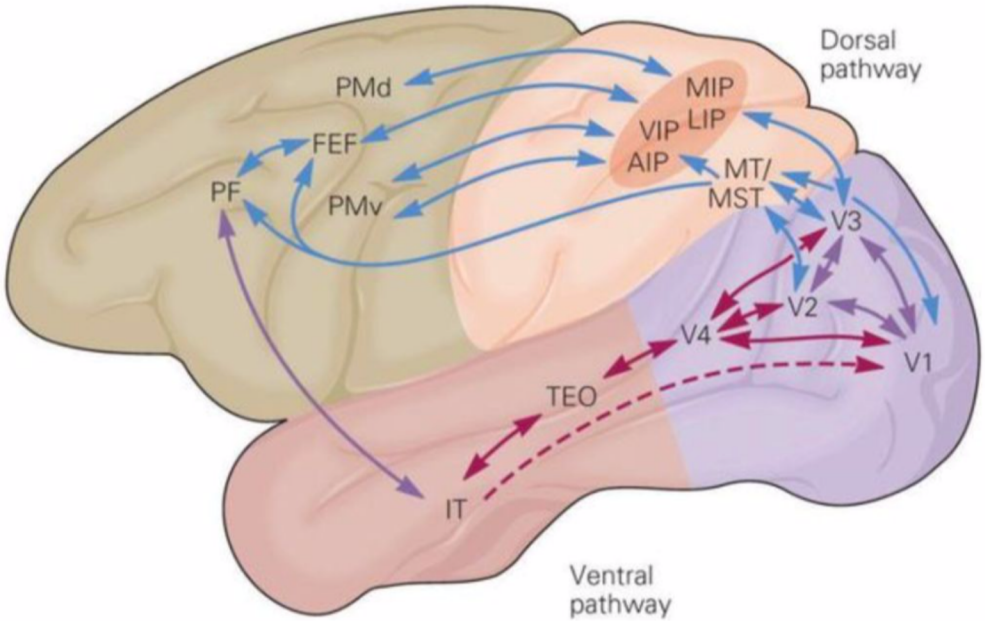
2019-05-21

Outline

- Background introduction
- Convolutional neural networks (CNN)
  - Overview
  - Network details: convolution, pooling, activation, loss functions
- ResNet
- DenseNet
- RNN

Visual cortex





Starting from V1 primary visual cortex, visual signal is transmitted upwards, becoming more complicated and abstract.

视觉信息处理过程

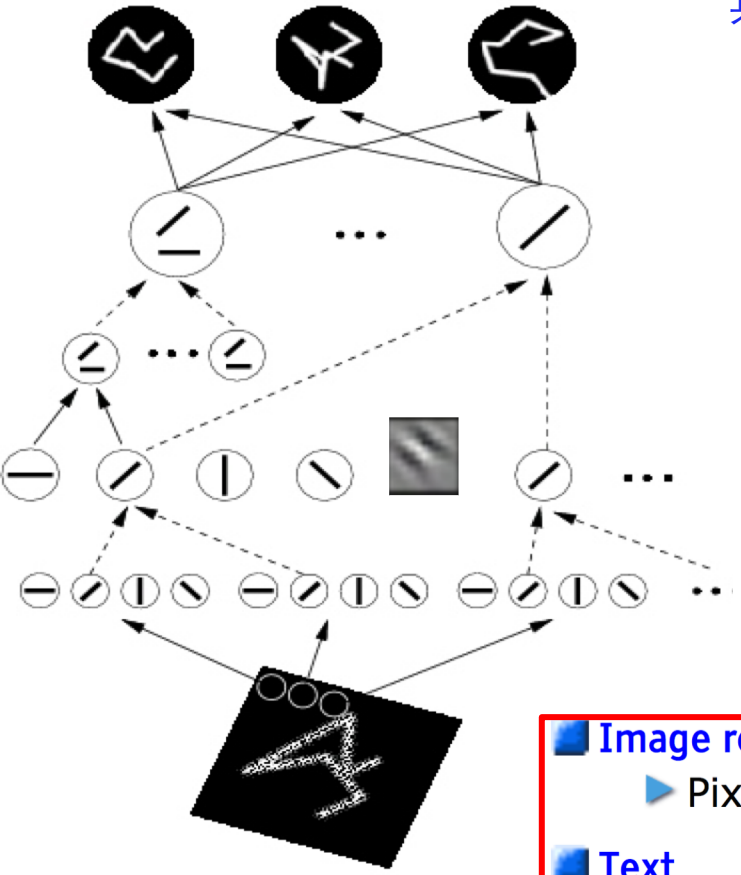
- 生物视觉信息处理的大致过程
  - –0ms: 光刺激发生
  - –10ms: 视网膜光电转换和编码
  - –35ms: 视网膜“模数“转换和神经脉冲
  - –45ms: 外侧膝状体(LGN)中继转发
  - –55ms: V1 方向选择细胞响应
  - –85ms: V4 大范围整合
  - –100ms: 人脸选择细胞响应
  - –160-220ms: 对象分类识别

“只因在人群中多看了你一眼”

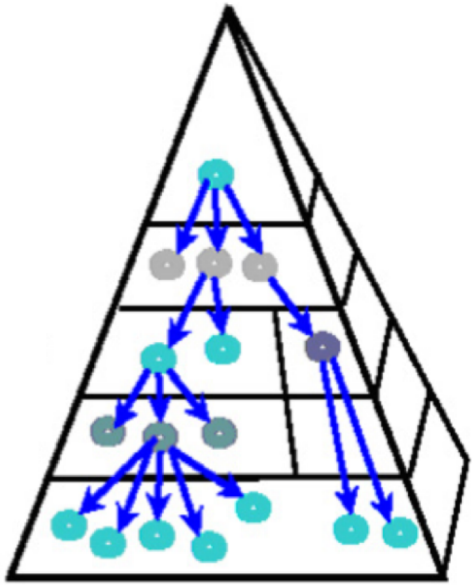
Maunsell and Gibson 1992; Raiguel et al. 1989; Nowak et al. 1995; Schmolesky et al. 1998; Thorpe, Fize& Marlot 1996

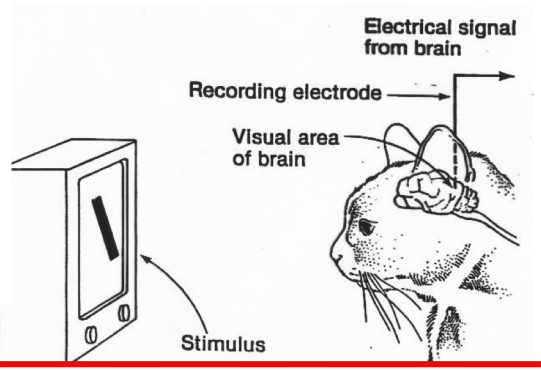
Feature Detection Theory 特征检测理论（1960）

Hubel Wiesel



具备广泛的普遍性！

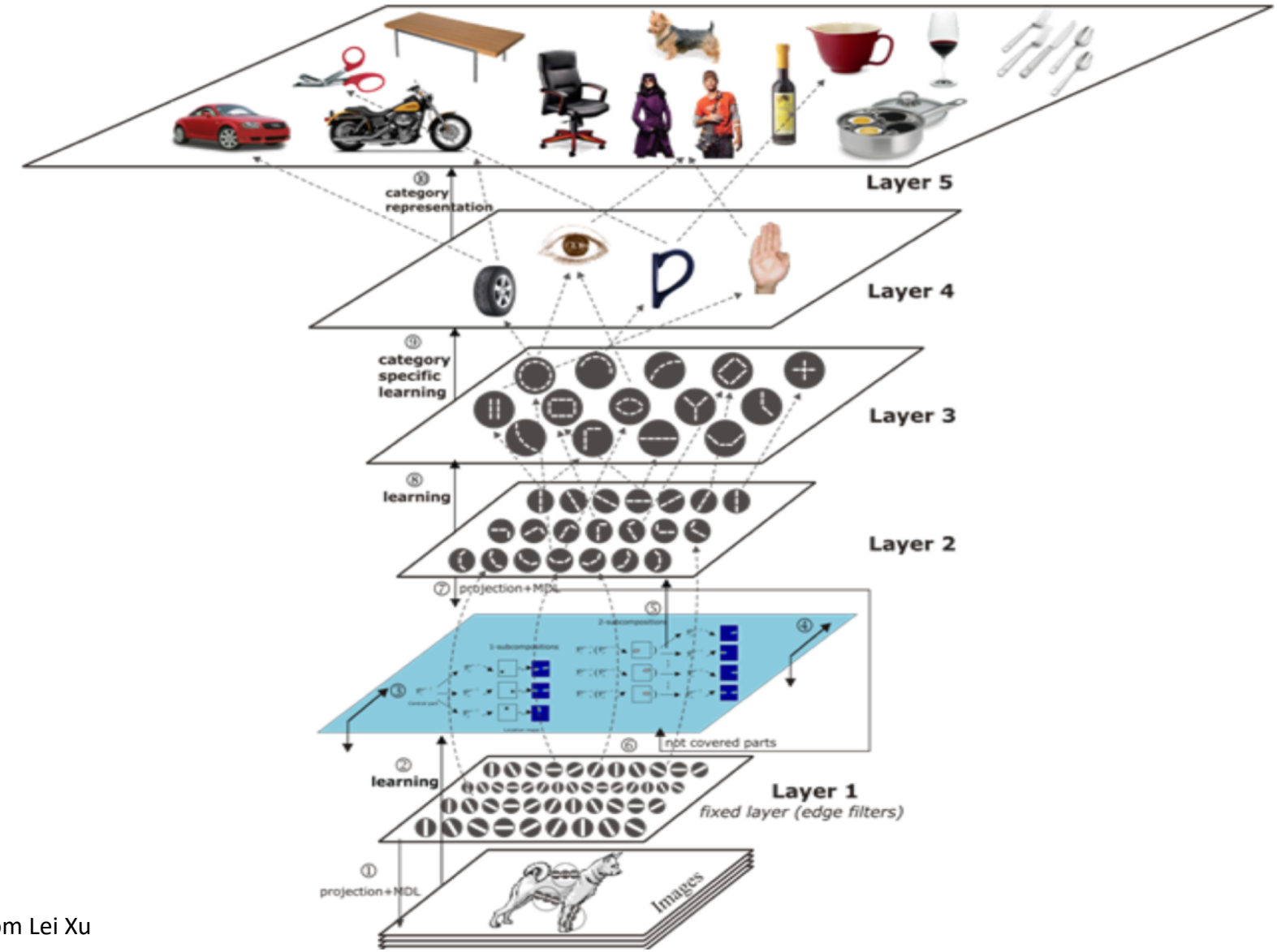




- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word →

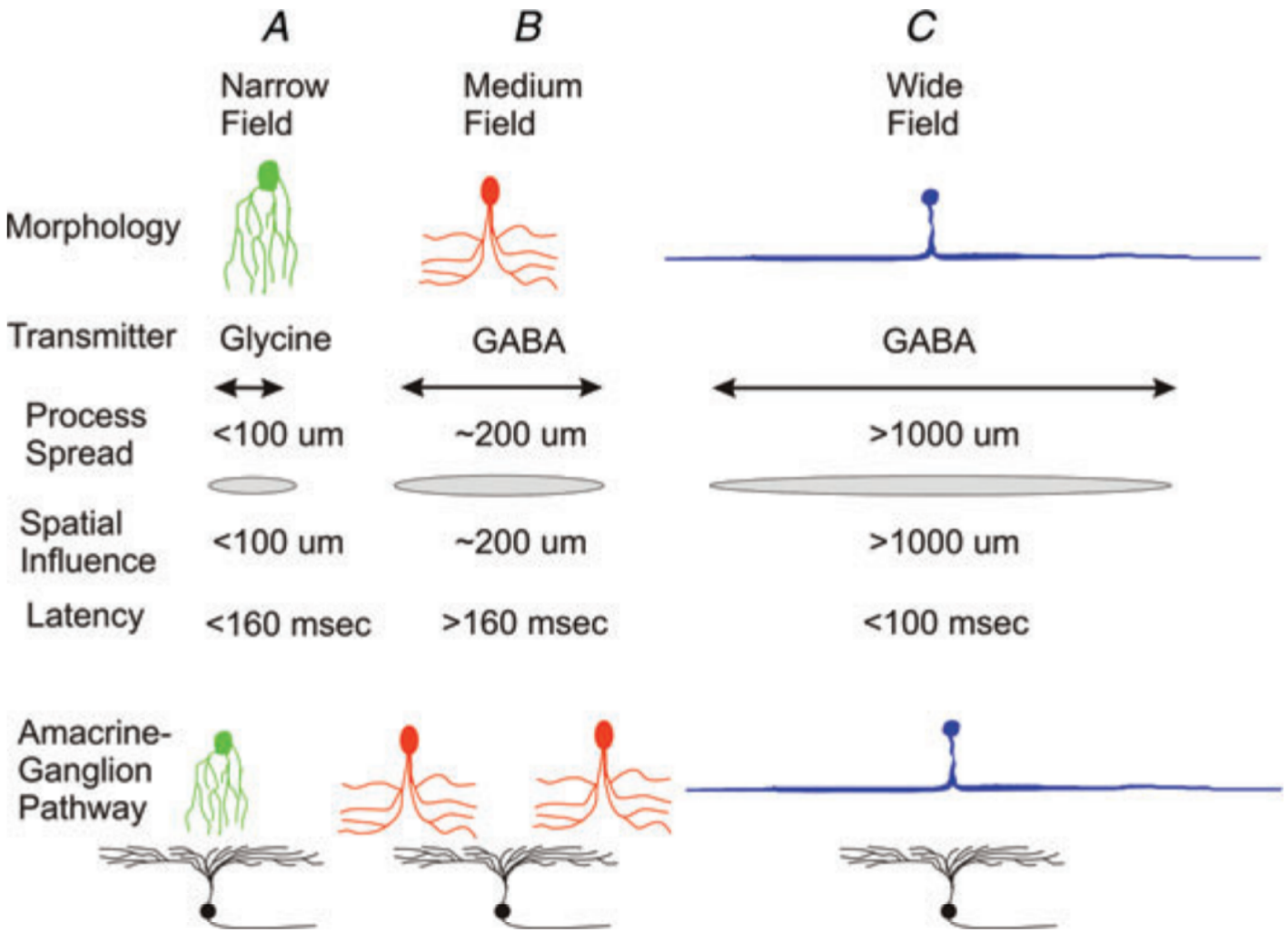
Slide from Lei Xu

Hierarchical representations



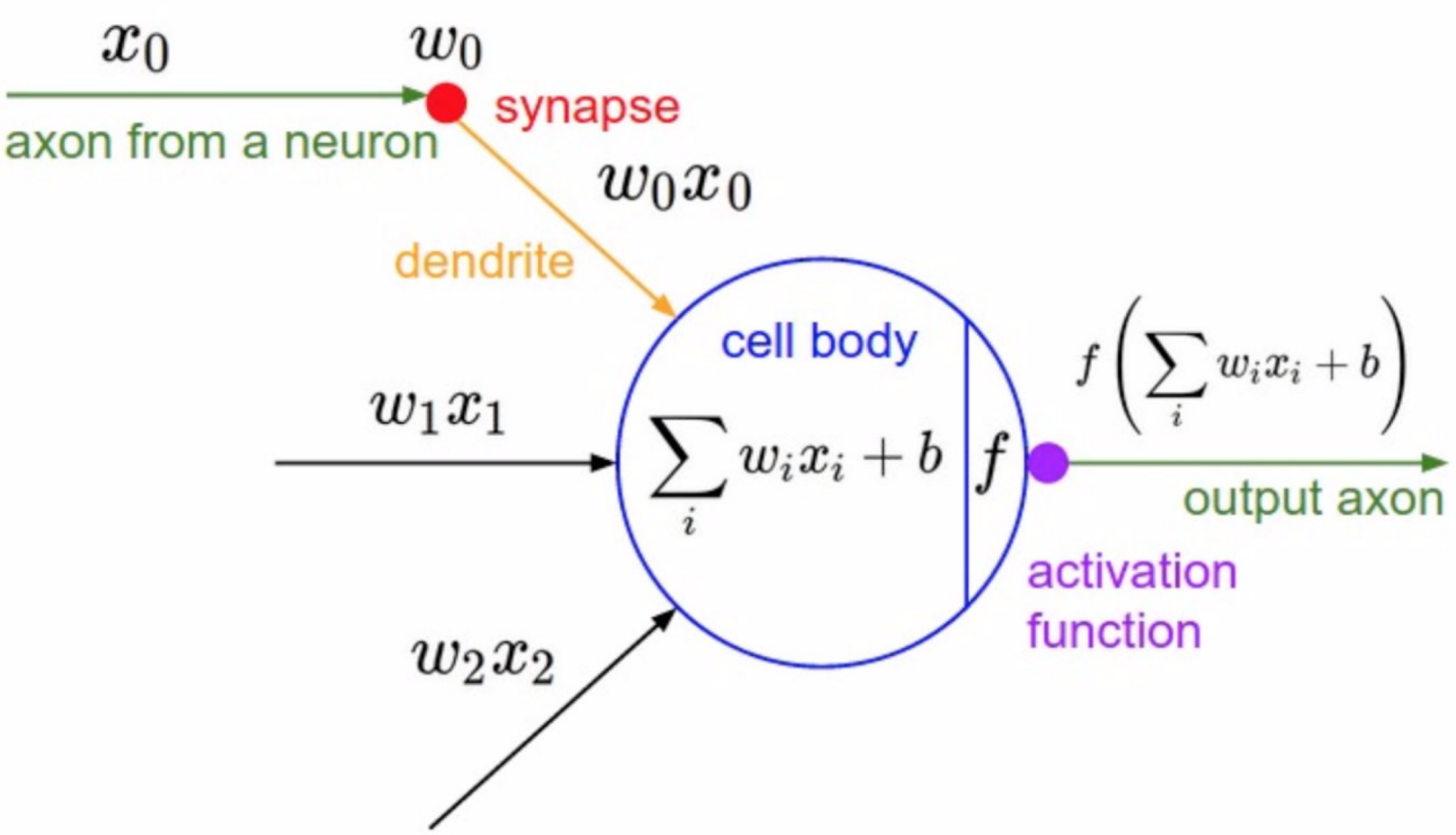
Slide from Lei Xu

narrow field, medium field and wide field amacrine cells



Frank S. Werblin, J Physiol 589.15 (2011) pp 3691–3702

The neuron model



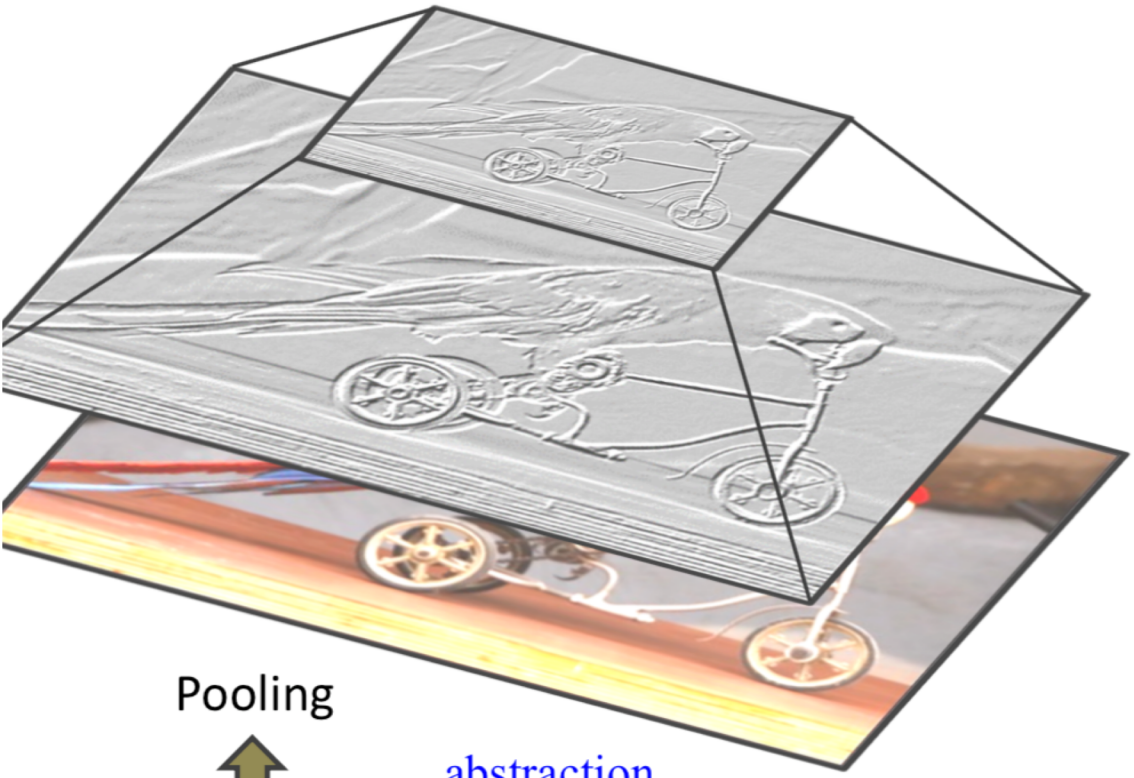
Outline

- Background introduction

- **Convolutional neural networks (CNN)**
  - Overview
  - Network details: convolution, pooling, activation, loss functions

- ResNet
- DenseNet
- RNN

A Basic Module of the **Convolutional Neural Nets**



Pooling

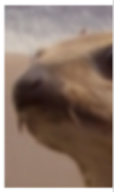
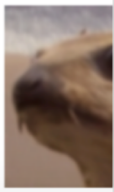
abstraction

Convolution

summary

Image

Slide from Lei Xu

Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Basic functions: *constant, successor, projection*

$$f(x_1,\cdots,x_k)=n,\quad f(x)=x+1,\quad P_i(x_1,\cdots,x_k)=x_i$$

Composition operator

$$h\circ(g_1,\ldots,g_m)\overset{\text{def}}{=}f(x_1,\ldots,x_k)=h(g_1(x_1,\ldots,x_k),\ldots,g_m(x_1,\ldots,x_k)).$$

Primitive recursion operator $\rho(g,h)\overset{\text{def}}{=}f(y,x_1,\ldots,x_k)$ where

$$f(0,x_1,\ldots,x_k)=g(x_1,\ldots,x_k)$$

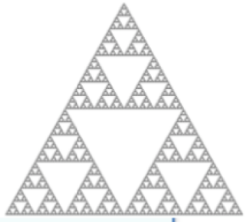
$$f(y+1,x_1,\ldots,x_k)=h(y,f(y,x_1,\ldots,x_k),x_1,\ldots,x_k)$$

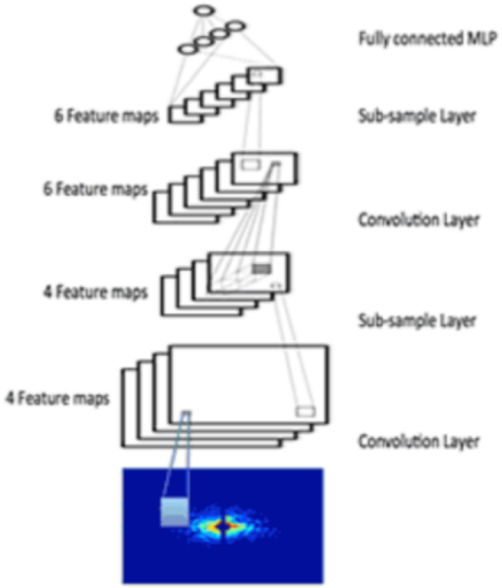
Minimization operator

Given a (k+1)-ary total function $f(y,x_1,\ldots,x_k)$

$$\mu(f)(x_1,\ldots,x_k)=z \overset{\text{def}}{\iff} f(z,x_1,\ldots,x_k)=0 \quad \text{and}$$

$$f(i,x_1,\ldots,x_k)>0 \quad \text{for } i=0,\ldots,z-1.$$





$$f(x,0)=g(x)$$

$$f(x,y+1)=h(x,y,f(x,y))$$

$$h(x,y,x)=z+x^y x$$

$$div(x,y)=\begin{cases}\text{integer portion of } x/y \text{ if } y\neq 0\\ \text{underfined if } y=0\end{cases}$$

$$f(\bar{x},3)=h(\bar{x},2,f(\bar{x},2))$$

$$f(\bar{x},2)=h(\bar{x},1,f(\bar{x},1))$$

$$f(\bar{x},1)=h(\bar{x},0,f(\bar{x},0))$$

$$f(\bar{x},0)=g(\bar{x})$$

Slide from Lei Xu

Early CNN: LeNet-5

LeNet-5, a pioneering 7-level convolutional network by LeCun et al. in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks digitized in 32x32 pixel images.

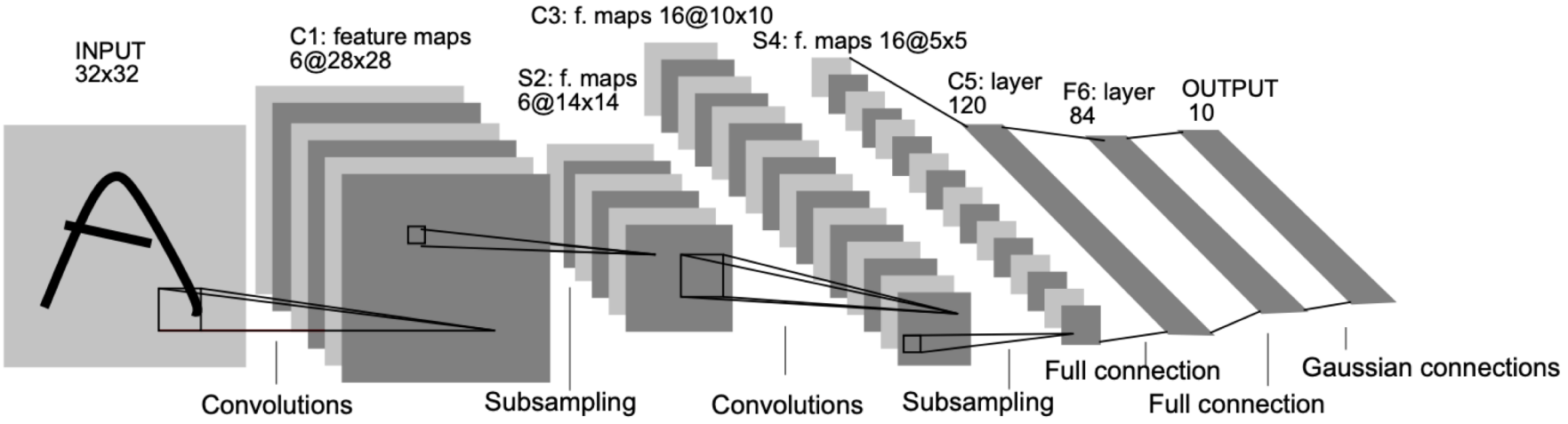


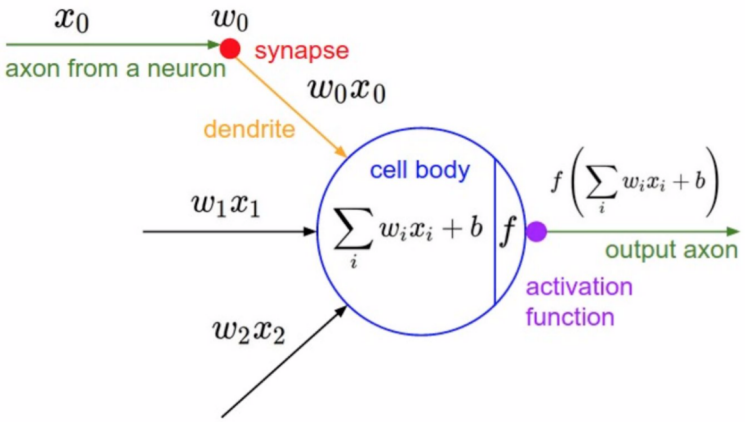
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324.

Affine transformations

- A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a non-linearity).

- Applicable to any type of input
  - flattened into a vector
  - Image, sound clip, etc.



V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Affine transformations are not good for image-like structured data

- Images, sound clips and many other similar kinds of data have an intrinsic structure.
- They are stored as multi-dimensional arrays.
- They feature one or more axes for which ordering matters (e.g., width and height axes for an image, time axis for a sound clip).
- One axis, called the channel axis, is used to access different views of the data (e.g., the red, green and blue channels of a color image, or the left and right channels of a stereo audio track).

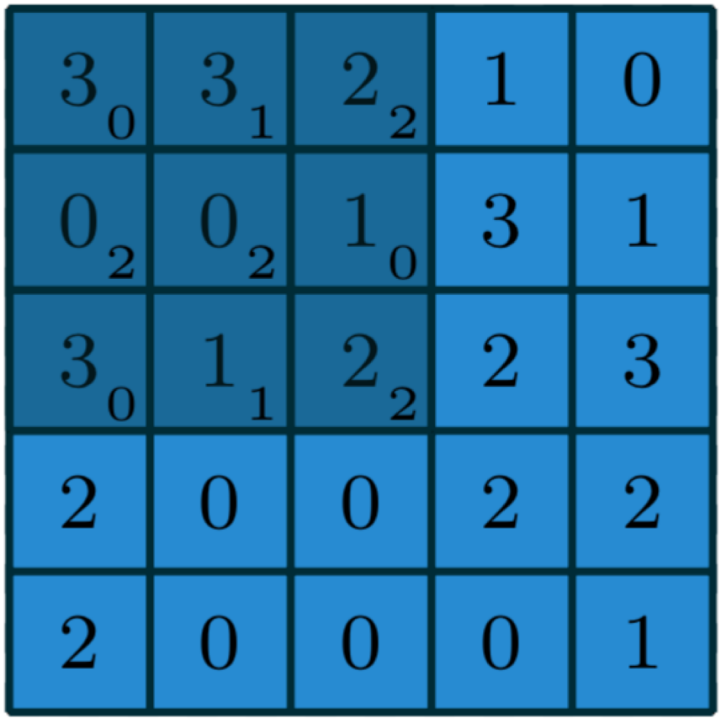
V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Discrete convolution

Kernel (3x3)

0	1	2
2	2	0
0	1	2

Input feature map



Output feature map

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

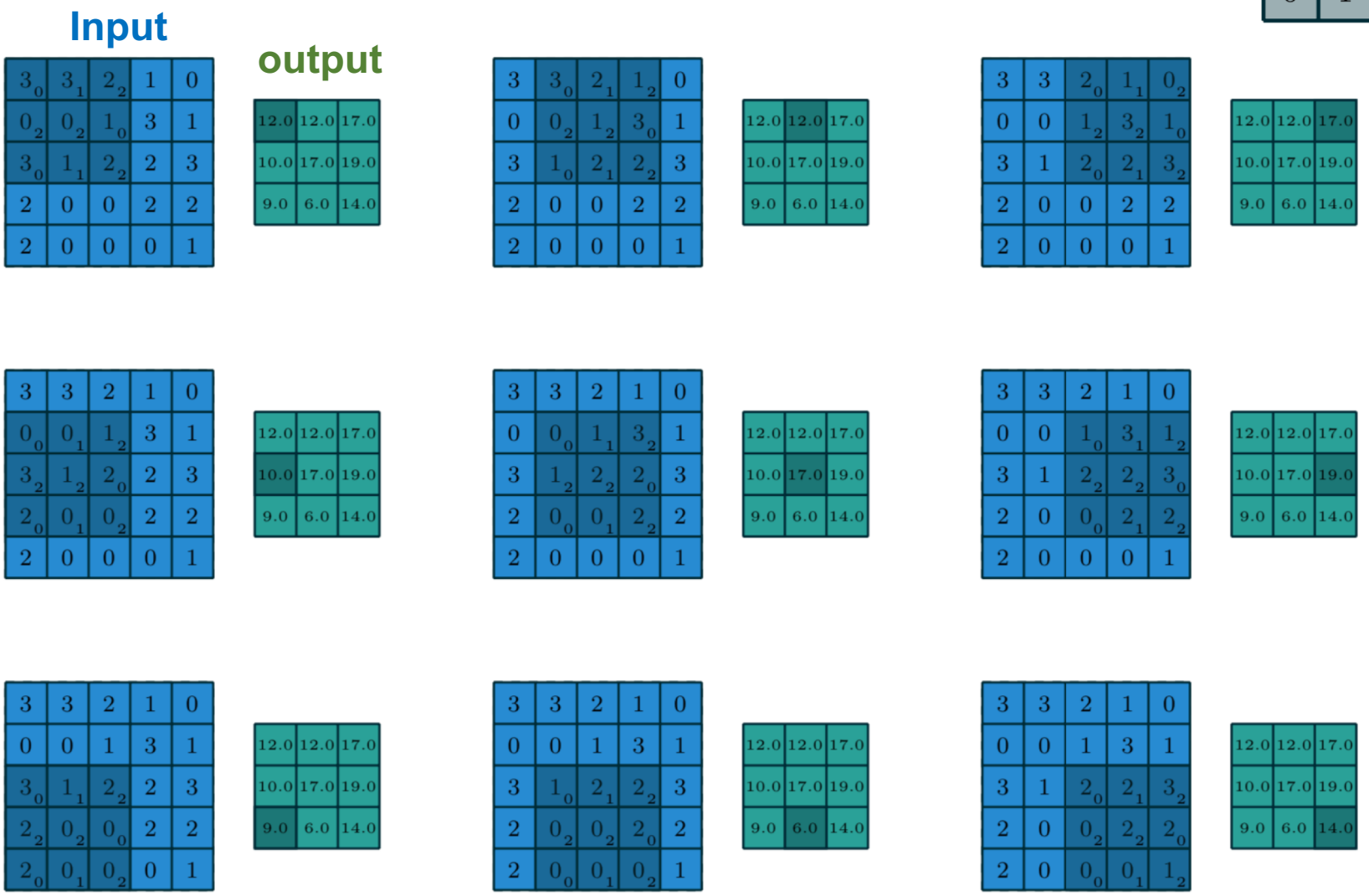
0 x 3 + 1 x 3 + 2 x 2
+ 2 x 0 + 2 x 0 + 0 x 1
+ 0 x 3 + 1 x 1 + 2 x 2
= 12.0

V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

Discrete convolution

Kernel (3x3)

0	1	2
2	2	0
0	1	2



V. Dumoulin and F. Visin, arXiv:1603.07285, 2016

N-D convolution (N=2,3,…)

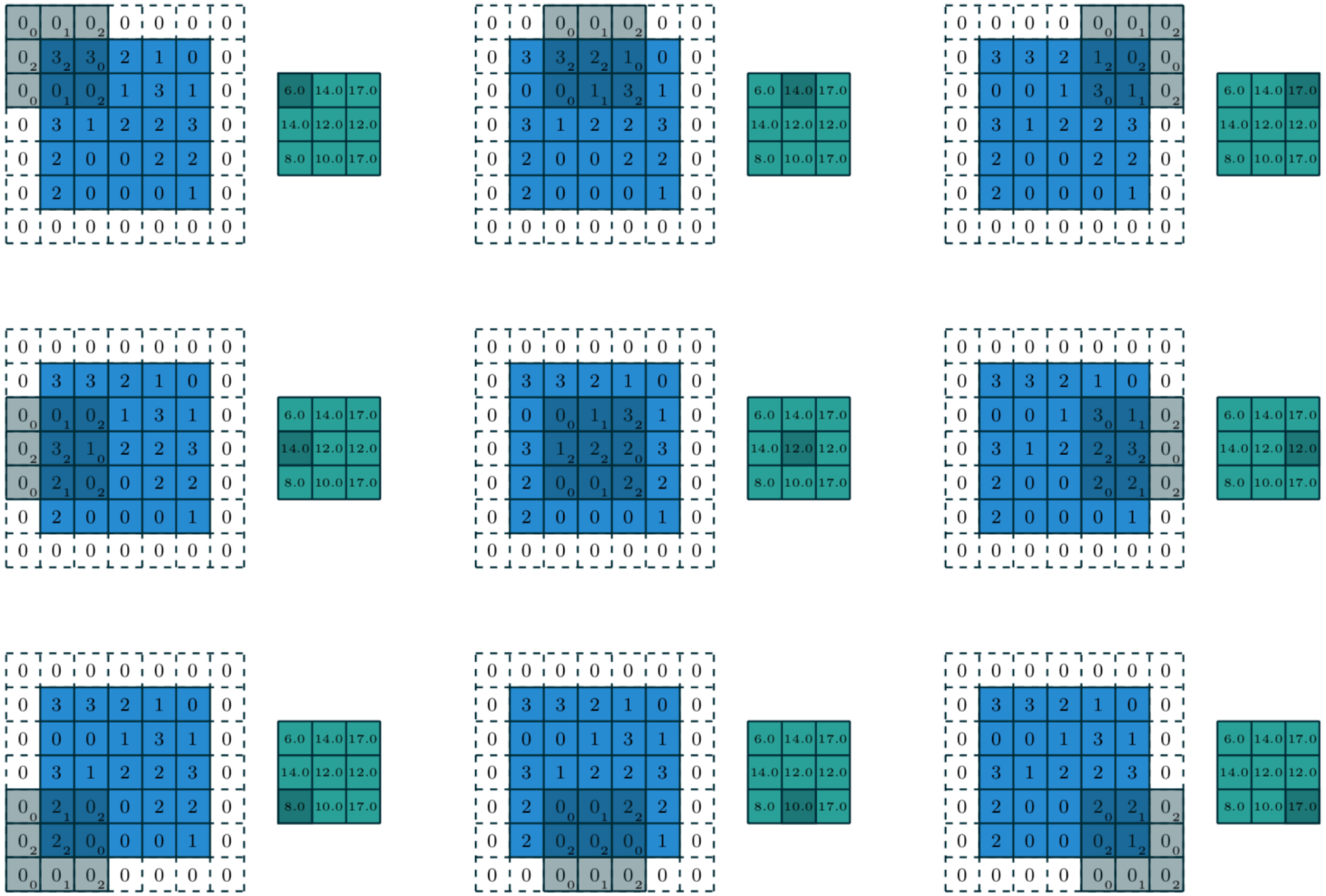
axis $j = 1, 2, \ldots, N$.

$$n \equiv \text{number of output feature maps},$$
$$m \equiv \text{number of input feature maps},$$
$$k_j \equiv \text{kernel size along axis } j.$$

- i_j: input size along axis j,
- k_j: kernel size along axis j,
- s_j: stride (distance between two consecutive positions of the kernel) along axis j,
- p_j: zero padding (number of zeros concatenated at the beginning and at the end of an axis) along axis j.

Example with zero paddings

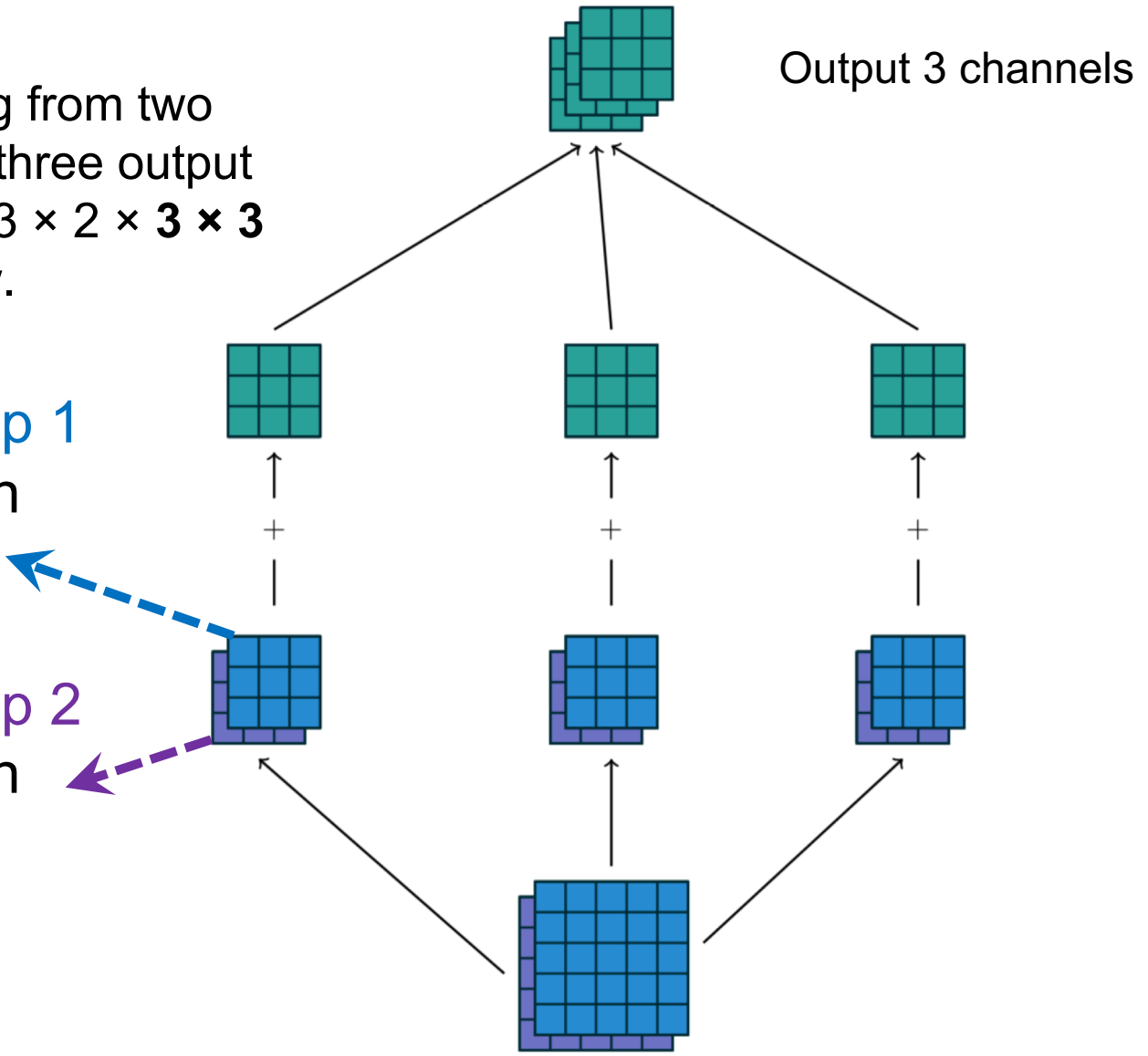
for $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, and $p_1 = p_2 = 1$.



Example with three output feature maps

A convolution mapping from two input feature maps to three output feature maps using a $3 \times 2 \times \mathbf{3} \times \mathbf{3}$ collection of kernels $\mathbf{w}$.

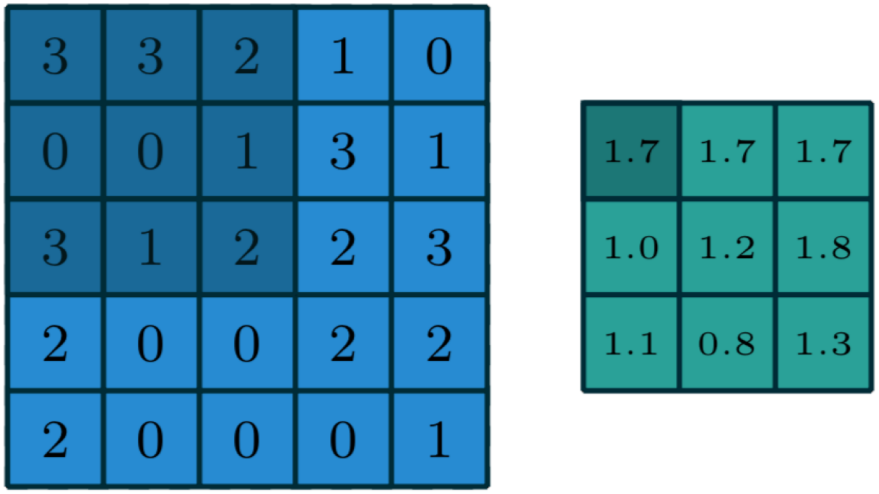
- input feature map 1 is convolved with kernel $\mathbf{w}_{1,1}$.
- input feature map 2 is convolved with kernel $\mathbf{w}_{1,2}$.



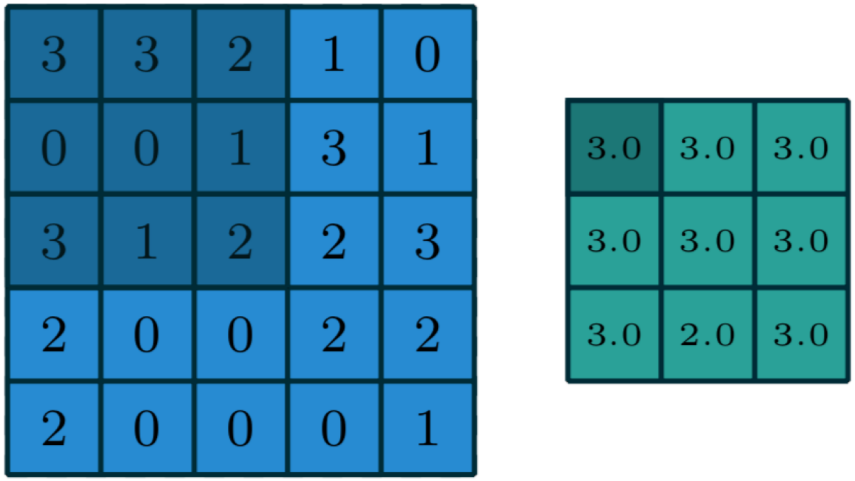
Pooling

- Pooling operations reduce the size of feature maps by using some function to summarize subregions, such as taking the **average** or the **maximum** value.

Average pooling

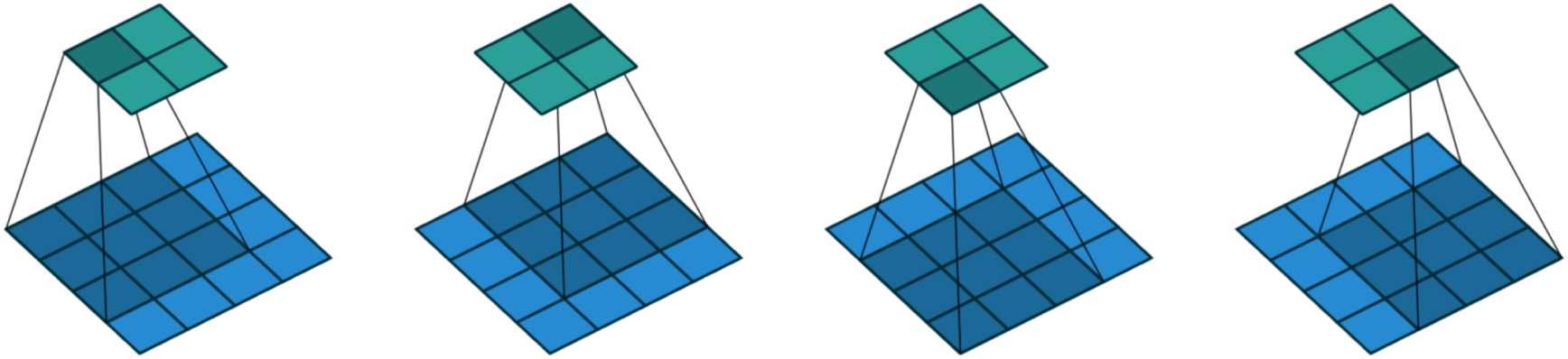


Maximum pooling



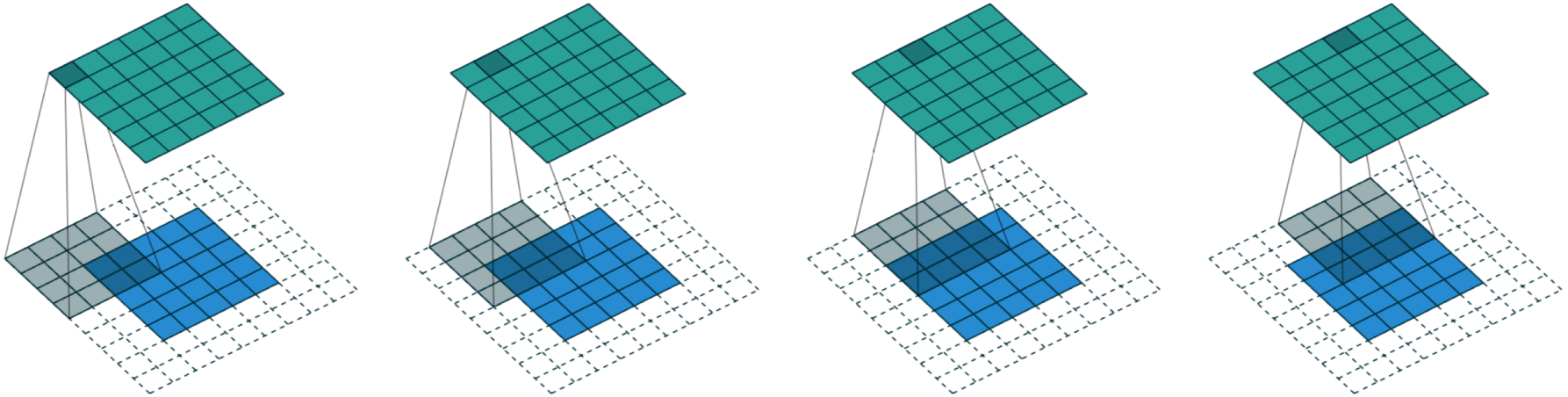
Example -- No padding, no strides

Convolving a 3 × 3 kernel over a 4 × 4 input using unit strides (i.e., i = 4, k = 3, s = 1 and p = 0).

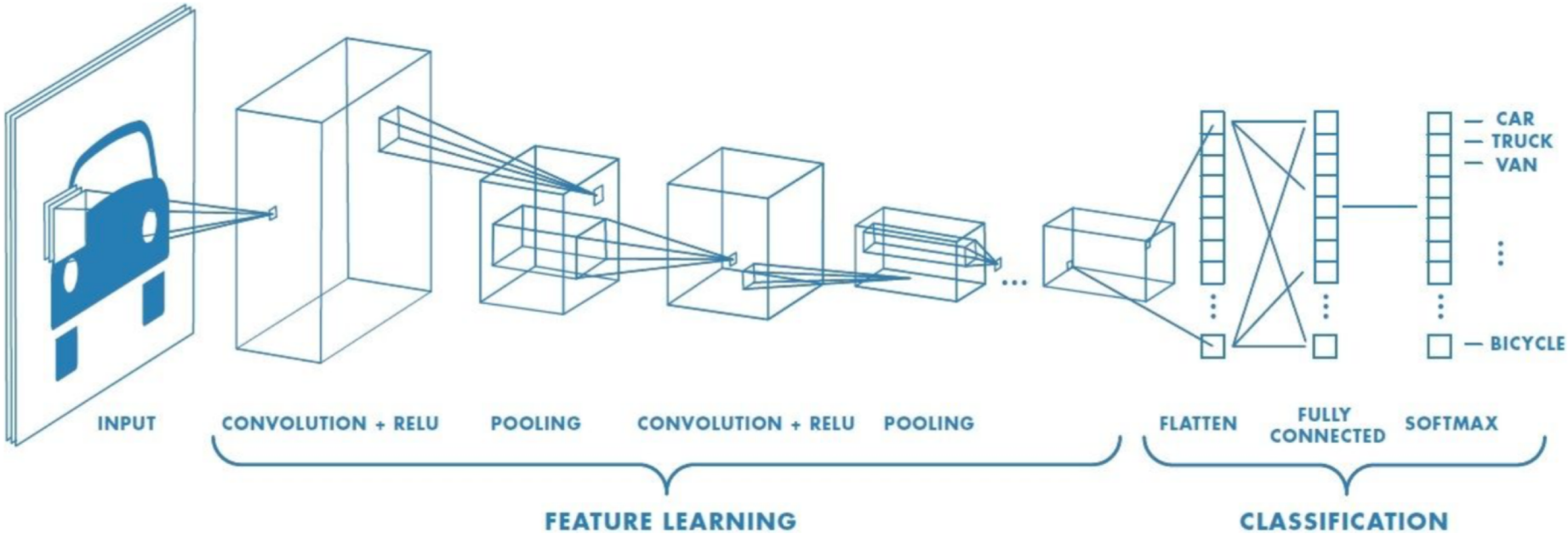


Example -- Arbitrary padding, no strides

Convolving a 4 × 4 kernel over a 5 × 5 input padded with a 2 × 2 border of zeros using unit strides (i.e., i = 5, k = 4, s = 1 and p = 2).



Sample architecture of CNN

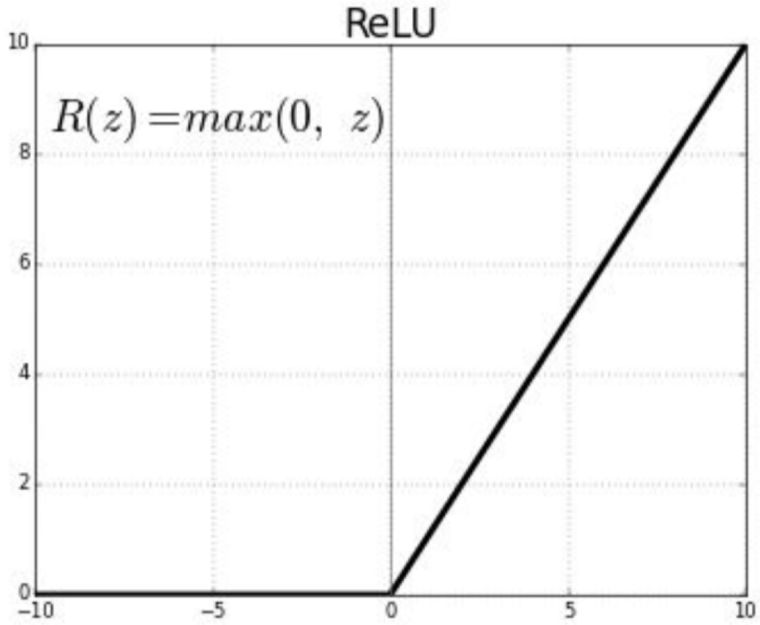


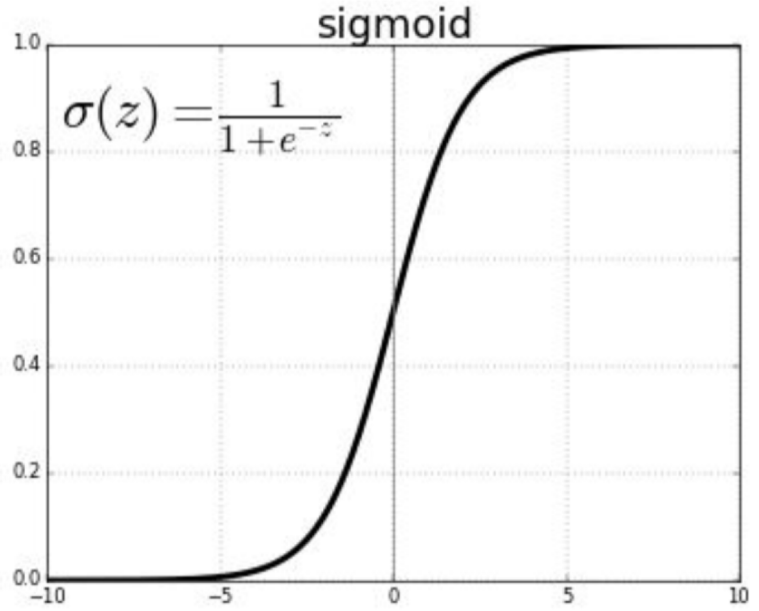
Activation layer

- Used to increase non-linearity of the network without affecting receptive fields of conv layers
- Prefer **ReLU**, results in faster training
- **LeakyReLU** addresses the vanishing gradient problem

Other types:

- Leaky ReLU,
- Randomized Leaky ReLU,
- Parameterized ReLU
- Exponential Linear Units (ELU),
- Scaled Exponential Linear Units Tanh,
- hardtanh, softtanh, softsign, softmax, softplus...





Softmax

- A special kind of activation layer, usually at the end of FC layer outputs
- Can be viewed as a fancy **normalizer** (a.k.a. Normalized exponential function)
- Produce a **discrete probability distribution** vector
- Very convenient when combined with cross-entropy loss

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- Sample vector **x**
- Weight vector **w**

Loss layer

- **L1, L2** loss
- **Cross-Entropy** loss (works well for classification, e.g., image classification)
- **Hinge** Loss
- **Huber** Loss, more resilient to outliers with smooth gradient
- **Minimum Squared Error** (works well for regression task, e.g., Behavioral Cloning)

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

$$p(y_i|x_i) = [h_\theta(x_i)]^{(y_i)}[1 - h_\theta(x_i)]^{(1-y_i)}$$

$$p(y_i = 1|x_i) = h_\theta(x_i) \qquad p(y_i = 0|x_i) = 1 - h_\theta(x_i)$$

$$\sum_i max(0, 1 - y_i * h_\theta(x_i))$$

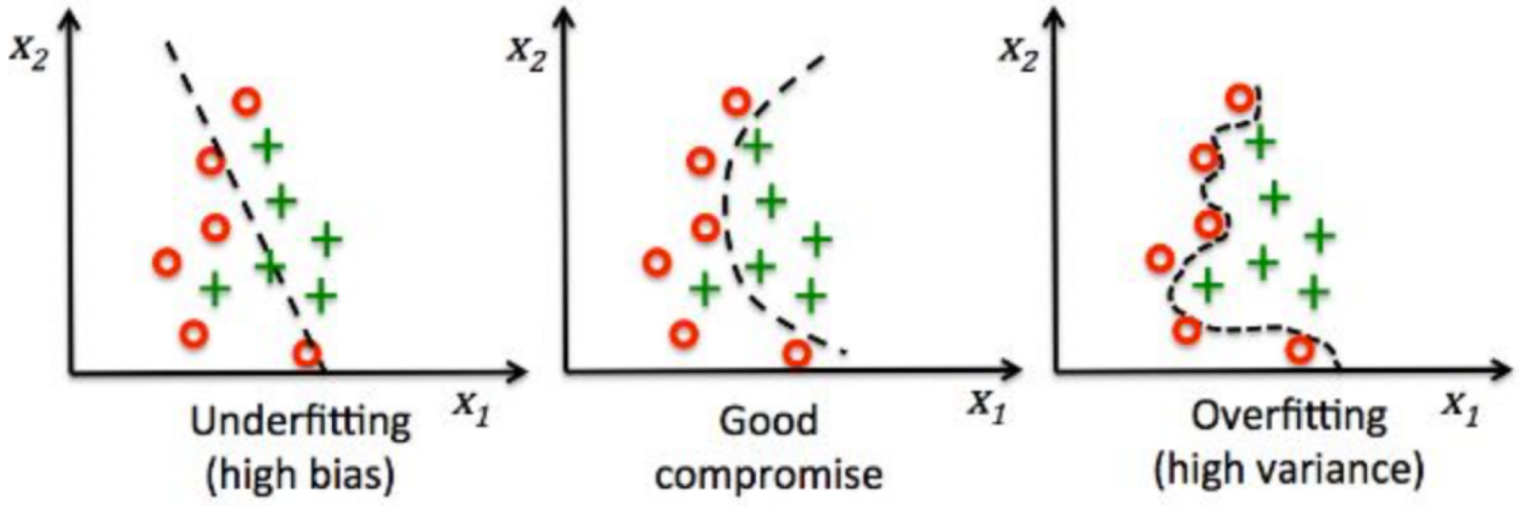
$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))^2$$

https://rohanvarma.me/Loss-Functions/

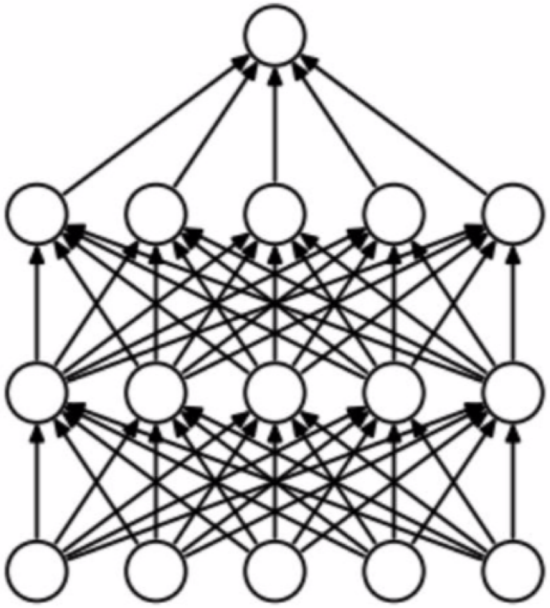
Regularization

- L1 / L2
- Dropout
- Batch norm
- Gradient clipping
- Max norm constraint

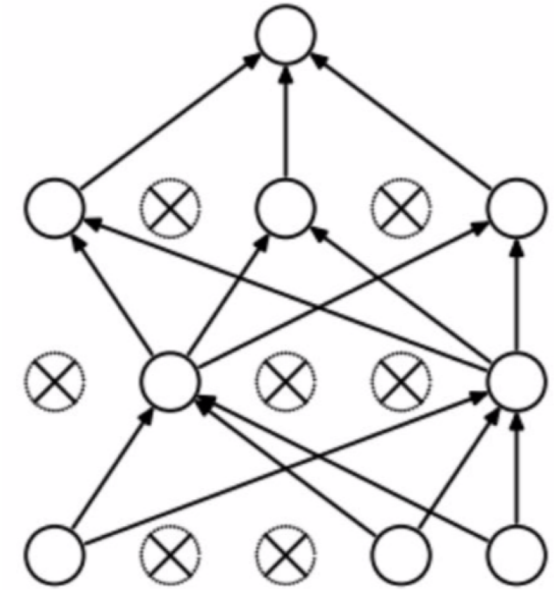


Dropout

- During training, randomly ignore activations by probability ***p***
- During testing, use all activations but scale them by ***p***
- Effectively prevent overfitting by reducing correlation between neurons



(a) Standard Neural Net



(b) After applying dropout.

Batch Normalization

- Makes networks robust to bad initialization of weights
- Usually inserted right before activation layers
- Reduce covariance shift by normalizing and scaling inputs
- The scale and shift parameters are trainable to avoid losing stability of the network

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

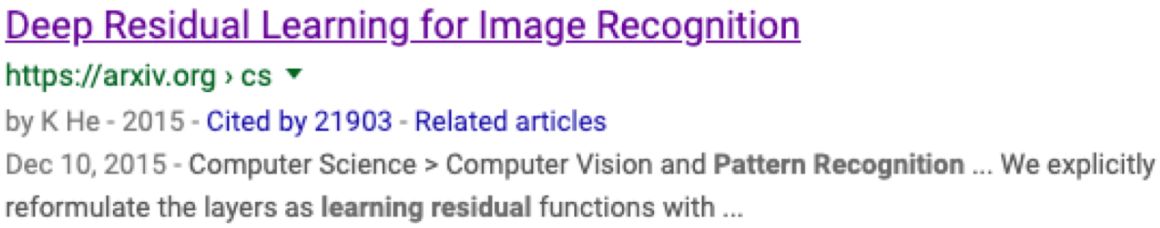
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Outline

- Background introduction
- Convolutional neural networks (CNN)
  - Overview
  - Network details: convolution, pooling, activation, loss functions
- ResNet
- DenseNet
- RNN

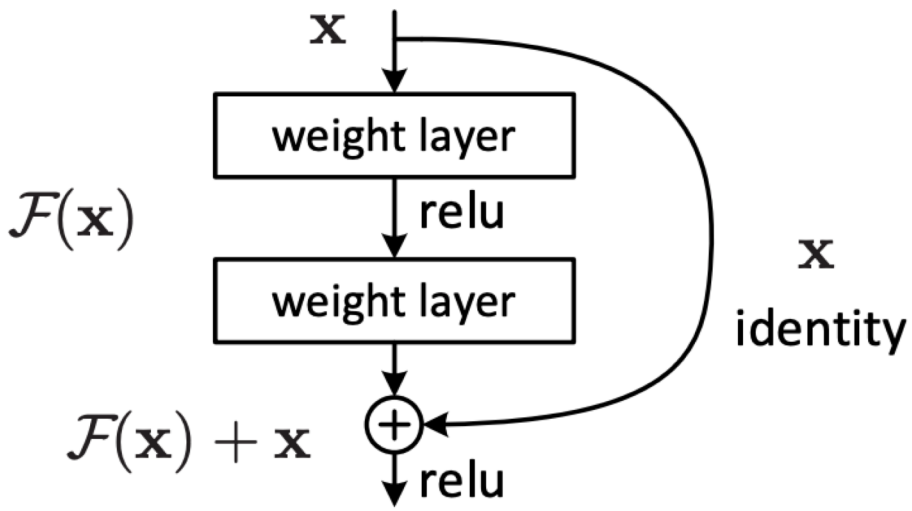
ResNet

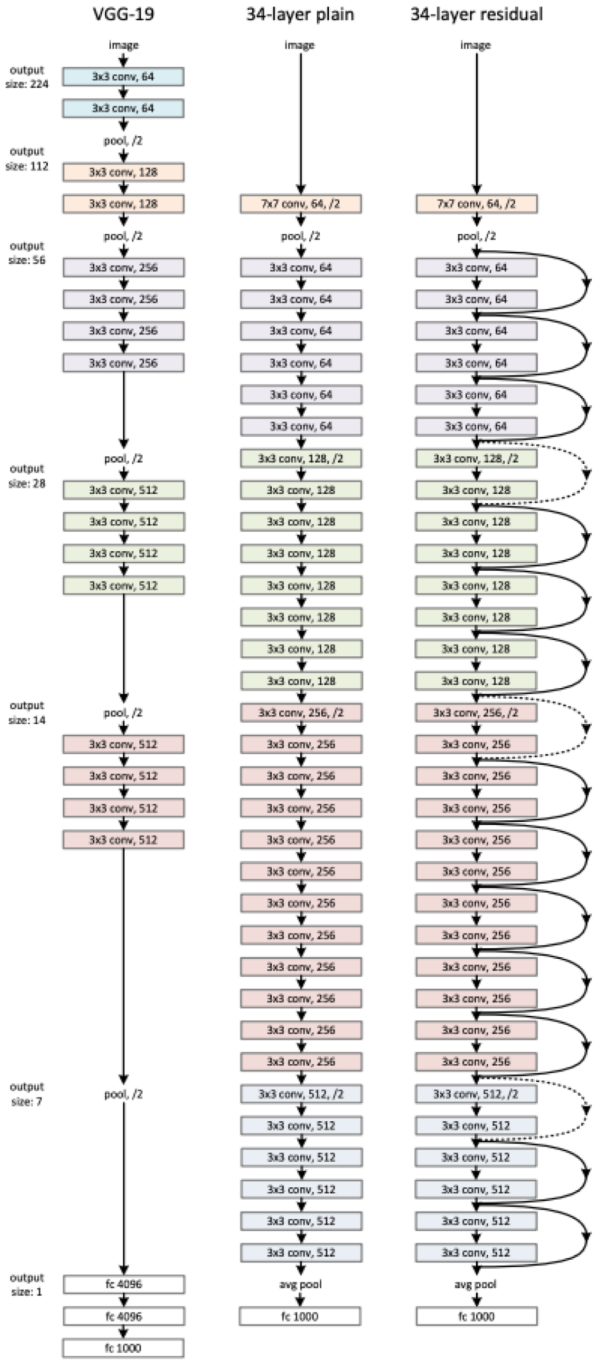
- Residual Network, by Kaiming He (2015)
- Heavy usage of "**skip connections**" which are similar to RNN Gated Recurrent Units (GRU)
- Commonly used as visual feature extractor in all kinds of learning tasks, ResNet50, ResNet101, ResNet152
- 3.57% Top-5 accuracy, **beats human**



ResNet architecture

Residual learning: a building block





DenseNet

Densely Connected Convolutional Networks
https://arxiv.org › cs
by G Huang - 2016 - Cited by 3591 - Related articles
In this paper, we embrace this observation and introduce the **Dense Convolutional Network** (DenseNet), which **connects** each layer to every other layer in a ...

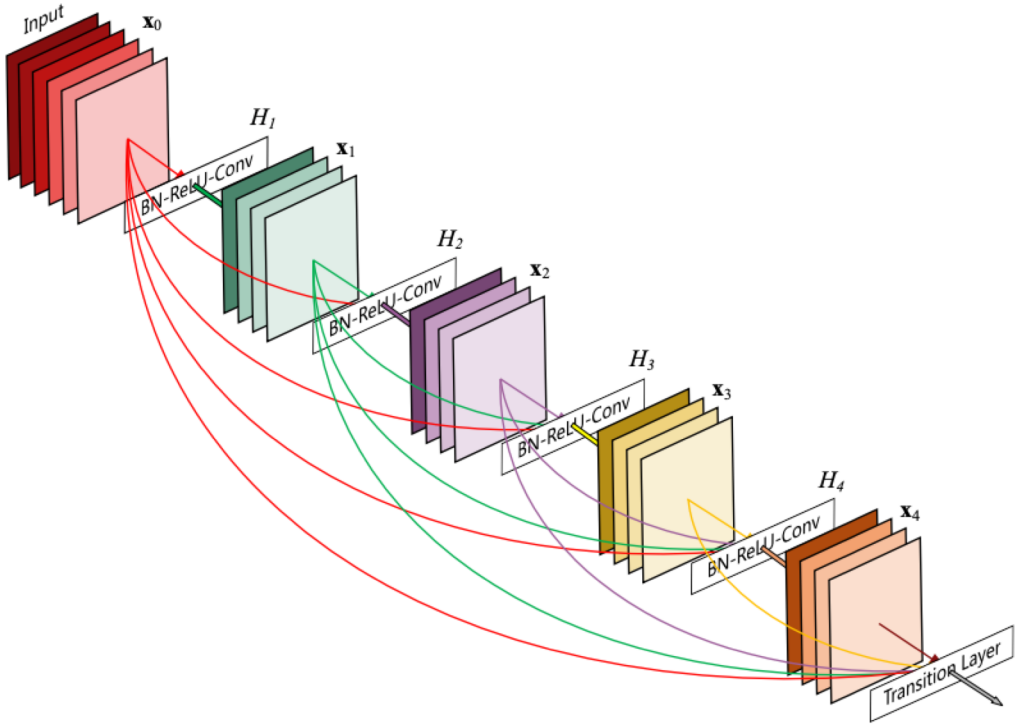
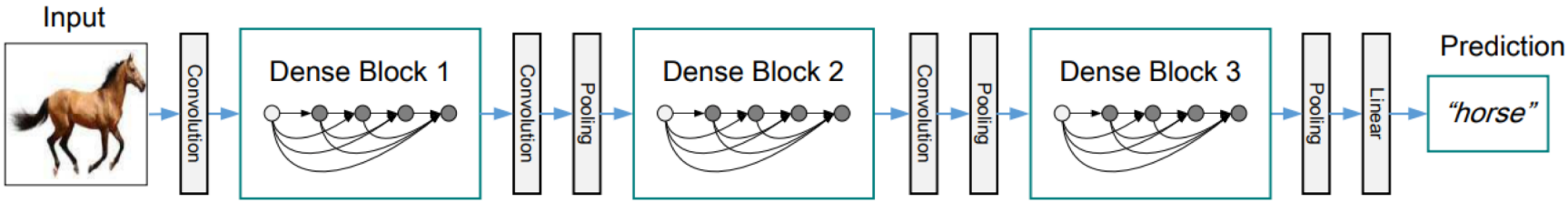
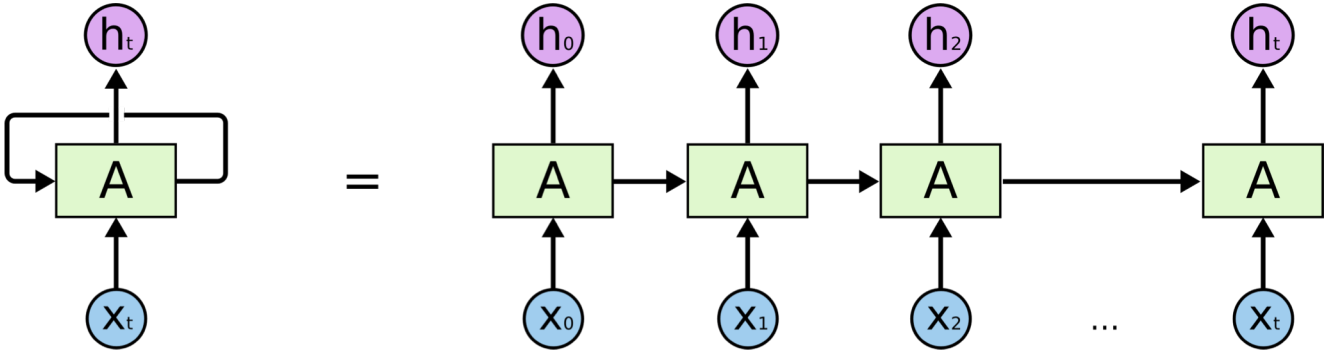


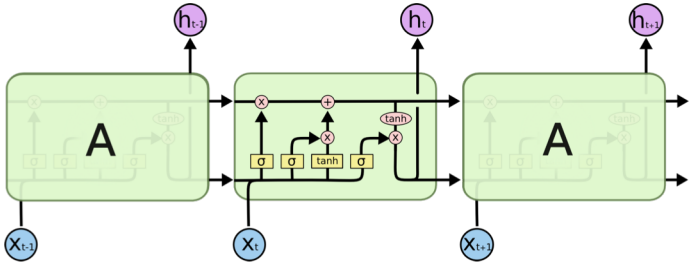
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

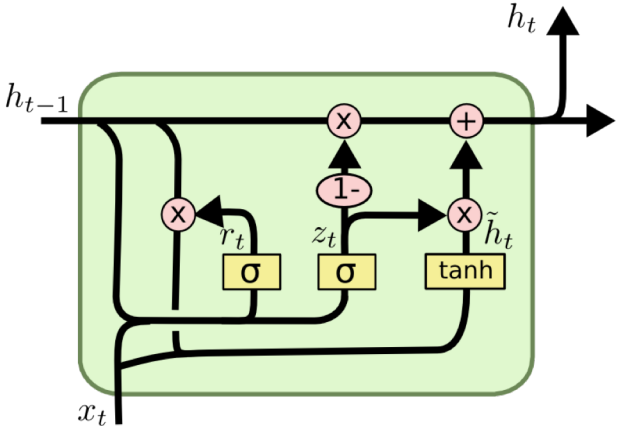


Recurrent neural networks



Long Short Term Memory networks (LSTM)





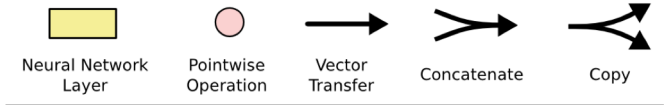
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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



Thank you!