MLFA-Conv-Net

15-Mar-2023 at 9:52 PM

Convolutional Neural Networks

Motivation :

One strategy to better fit the data is to tune the network architecture to better represent the constraints and traits of the problem

⇒ Inductive bias

Example :

Vision :

⇒ Nearby pixels are correlated

⇒ There are local features like edges and corners

⇒ An object (e.g. handwritten character) can be represented as a combination of such features.

Speech :

⇒ locality in terms of time

⇒ inputs that are temporally close can be grouped

⇒ Combining these primitives, utterances like speech phoneme can be defined.

** Hidden units can be designed not to consider connections from all inputs.

⇒ Hidden unit that define a window over the input space

⇒ Connected to a local subset of inputs

⇒ Decrease in the number of parameters.

This idea of local aggregation can be applied in the successive layers

⇒ Each layer is connected to a small number of local units below

⇒ Combining features from the previous layer to construct more complex features.

Example:

pixels → edges → arcs, corners → simcircle, rectangle
↓
digit

Concern with number of parameters:

⇒ units checking for different parameters in all local patches.

⇒ Too many parameters

⇒ Reduction in no of parameters: Weight sharing

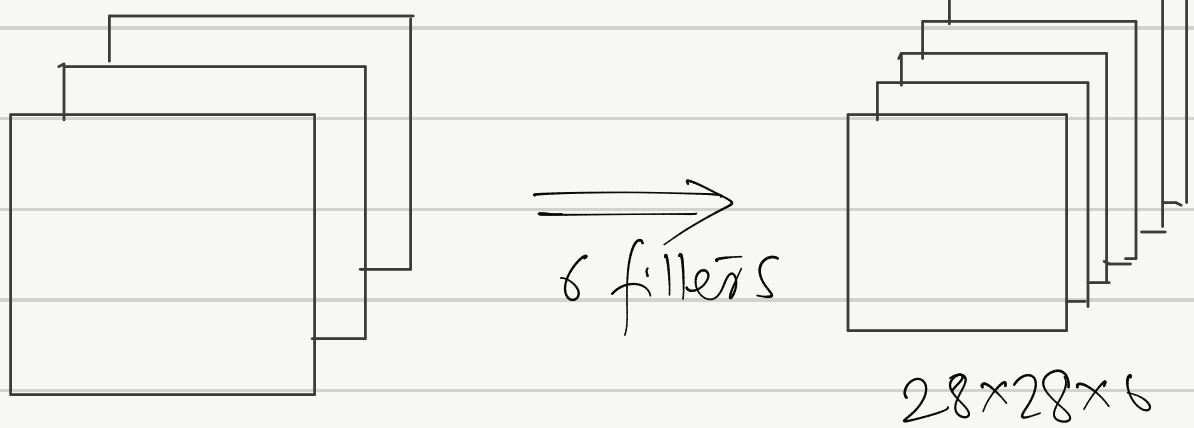
When looking for features like oriented edges, they may be present in different parts of the input space.

Design

⇒ image data is represented as 3D matrices ⇒ TENSOR

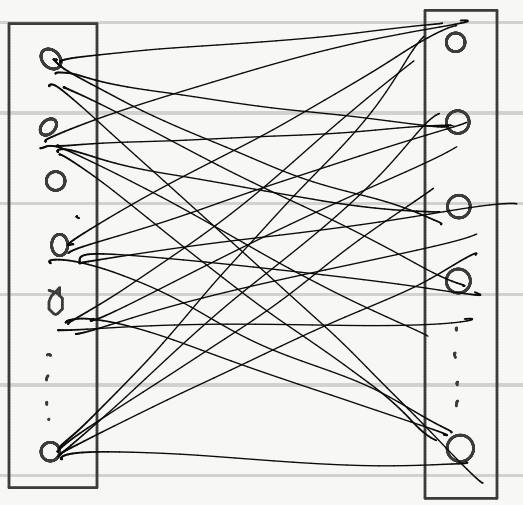
Width × height × no of channels (RGB)

Filters



Parameter sharing

If we had used a fully connected NN



3072 × 4704 > 14 milion parameter only for single layer

Observations: 14 million parameters are not independent from each other

⇒ Each of the 6 filters represents a fixed transformation

⇒ The hidden units for a given filter performs

same kind of computation in every part of an image

⇒ The weights can be kept same across the input space

Sparse Connection

Let us consider a vertical edge detection. A vertical edge may reside in a tiny part of the image and hence detection will depend on a small fraction of the input pixels

⇒ ignoring contributions from the other pixels

⇒ sparse connection

Convolution Operation:

Image matrix: $A \in \mathbb{R}^{w \times h}$

Filter or kernel: $F \in \mathbb{R}^{f \times f}$ → learned by the neural net

$f < w, \quad f < h$

$A * F \in \mathbb{R}^{w-f+1,\ h-f+1}$

$$(A * F)_{ij} = \sum_{u=1}^{f} \sum_{v=1}^{f} A_{i+u-1,\ j+v-1} \cdot f_{uv}$$

$1 \le i \le w-f+1$

$1 \le i \le h-f+1$

1	2	3	4	5	6
2	4	6	1	3	5
6	5	4	3	2	1
1	6	2	5	3	4
4	1	5	2	6	3
2	1	6	5	4	3

6×6

*

1	0	-1
1	0	-1
1	0	-1

3×3

=

-4	3	3	-4
-3	6	4	-1
0	2	0	2
-6	-4	0	2

6-3+1, 6-3+1

4×4

$1\times1 + 0\times2 + (-1)\times3 + 1\times2 + 0\times4 + (-1)\times6 + 1\times6 + 0\times5 + (-1)\times4 = -4$

Filter vertical Edge Detection:

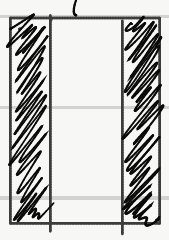
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



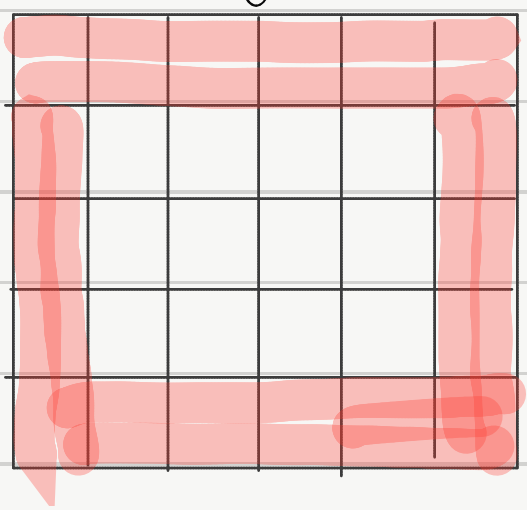
1	1	1
1	1	1
1	1	1

Kernel that blurs an image

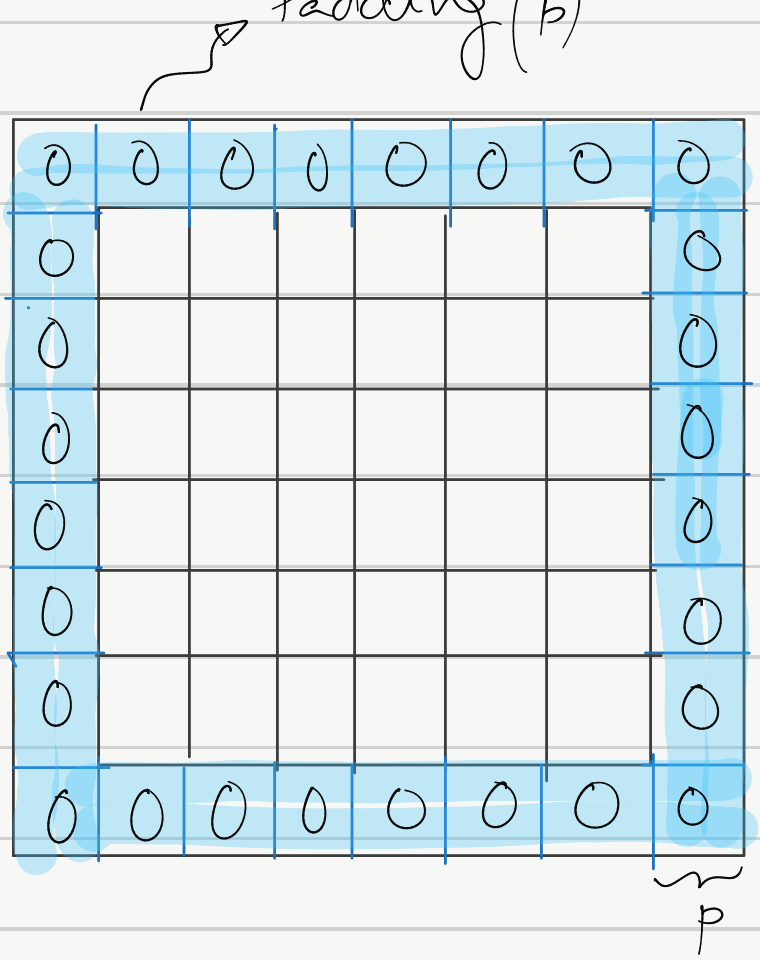
0	-1	0
-1	5	-1
0	-1	0

Kernel that sharpens an image

Padding:



* The convolution operator misses the first and last column or row
  ⇒ if there are important info in these rows and columns they are missed out.



* Size of the output is smaller than the input matrix
  ⇒ shrinking
  ⇒ after few conv layers the image shrinks significantly

Input : $w \times h$ || Padding : p
input after padding : $w+2p \times h+2p$
Filter : $f \times f$
output : $(w+2p-f+1) \times (h+2p-f+1)$

Choosing padding size:

⇒ "valid" convolution : $p=0$
⇒ "same" convolution
  ⇒ p is chosen such that input and output size is same.

Input size : $w \times h$

output size $= (w + 2p - f + 1) \times (h + 2p - f + 1)$

$$w + 2p - f + 1 = w$$

$$p = \frac{f-1}{2}$$

Typically f is chosen to be odd so that $\frac{f-1}{2}$ becomes an integer division.

Strides in convolutions:

⇒ Shifting filter by s columns or rows

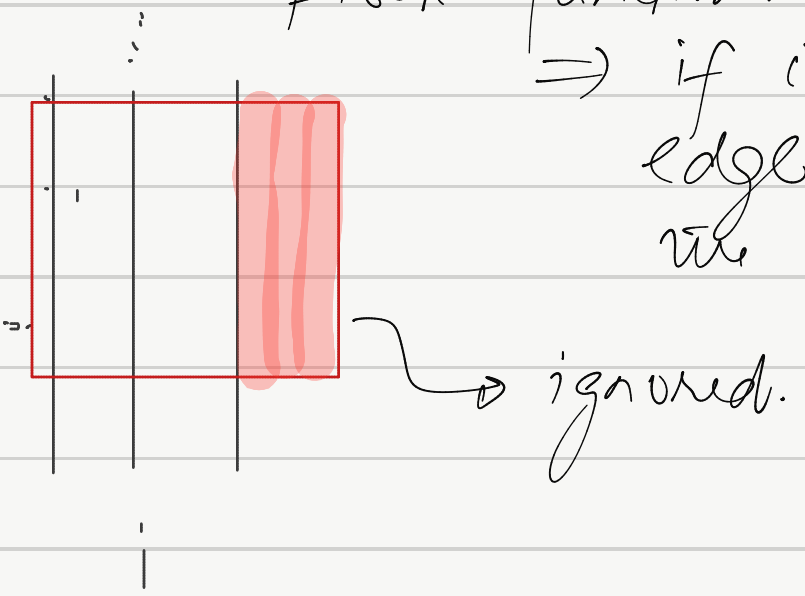
Input : $w \times h$, padding : p . Filter : $f \times f$

stride : s

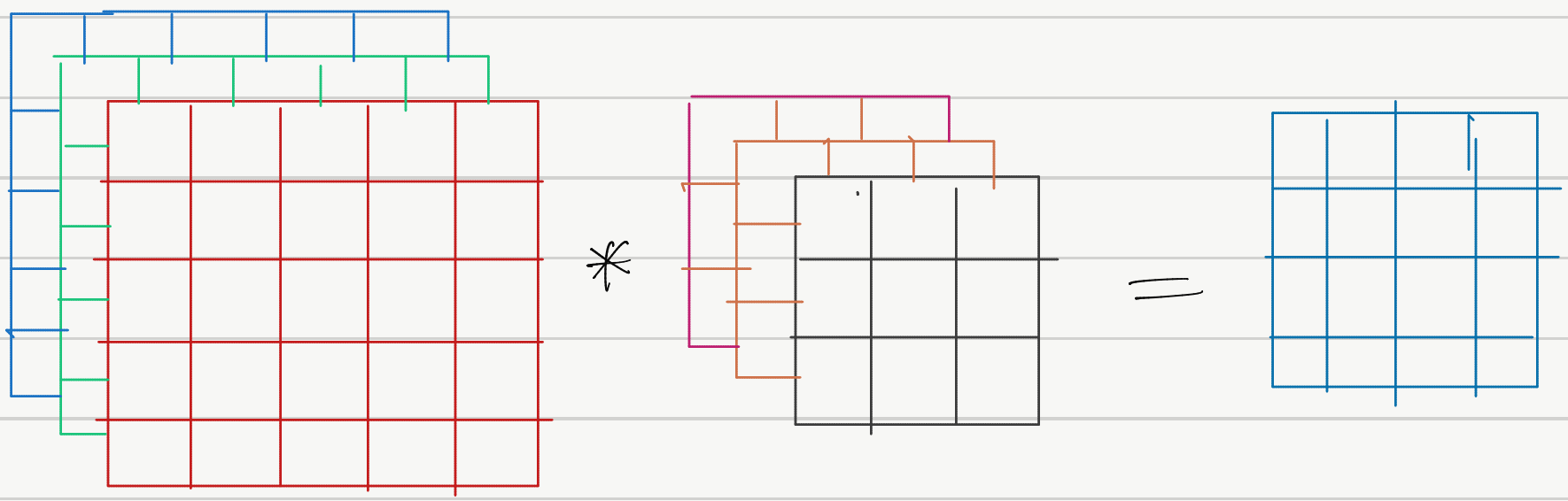
output size : $\left(\left\lfloor \frac{w + 2p - f}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{h + 2p - f}{s} \right\rfloor + 1\right)$

Floor function:

⇒ if the filter matrix is hanging at the edge, discard the computation of the hanging part



Convolution on Tensors:

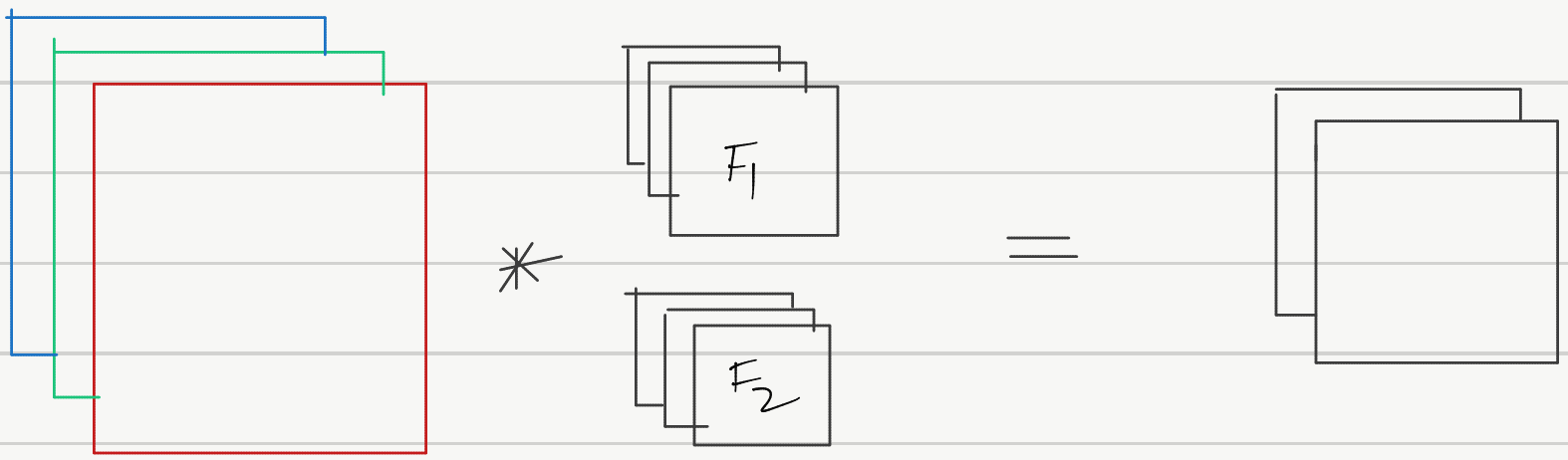


Input: $A \in \mathbb{R}^{w \times h \times c}$, Filter: $F \in \mathbb{R}^{f \times f \times c}$

$$A * F \in \mathbb{R}^{(w-f+1) \times (w-f \times 1)}$$

$$(A * F)_{ij} = \sum_{u=1}^{f} \sum_{v=1}^{f} \sum_{k=1}^{c} A_{i+u-1, j+v-1, k} \cdot F_{u,v,k}$$

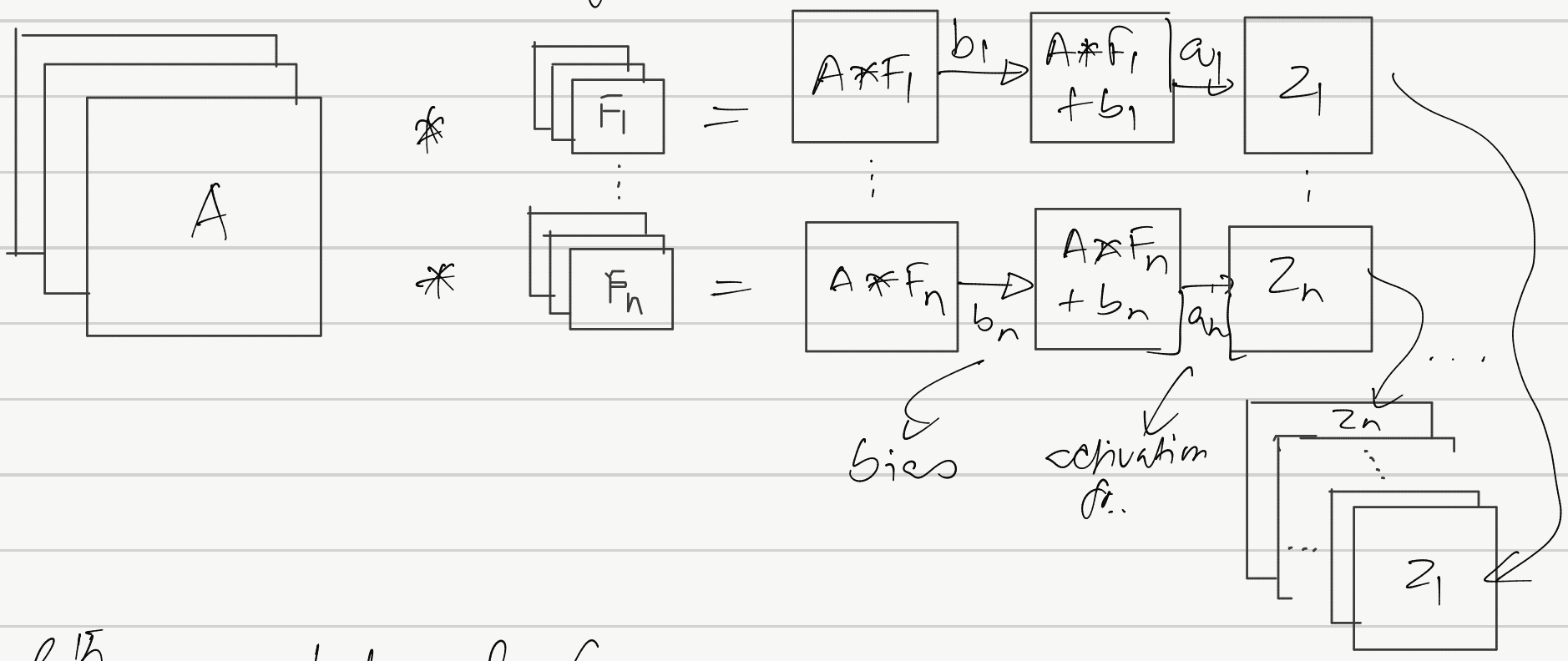
Multiple filters:



Input: $A \in \mathbb{R}^{w \times h \times c}$ $F_i \in \mathbb{R}^{f \times f \times c}$ $F = [F_1, \cdots F_{nf}]$

$$A * F \in \mathbb{R}^{(w-f+1) \times (h-f+1) \times n_f}$$

Convolutional Layers:



l^{th} convolutional Layer

- Filter size : $f^{(l)}$
- No of fillers : $n_f^{(l)}$
- padding size : $p^{(l)}$
- Stride length : $s^{(l)}$

Input : $w^{(l-1)} \times h^{(l-1)} \times c^{(l-1)}$

output : $w^{(l)} \times h^{(l)} \times c^{(l)}$

$$w^{(l)} = \left\lfloor \frac{w^{(l-1)} + 2p^{(l)} - f^{(l)}}{s^{(l)}} \right\rfloor + 1$$

$h^{(l)} \rightarrow$ similar

$$c^{(l)} = n_f^{(l)}$$

Parameters:

- For each filter : $f^{(l)} \cdot f^{(l)} \cdot c^{(l)}$
- For each filter : 1 bias
- Total : $n_f^{(l)} [f^{(l)} \cdot f^{(l)} \cdot c^{(l-1)} + 1]$

Pooling Layers:

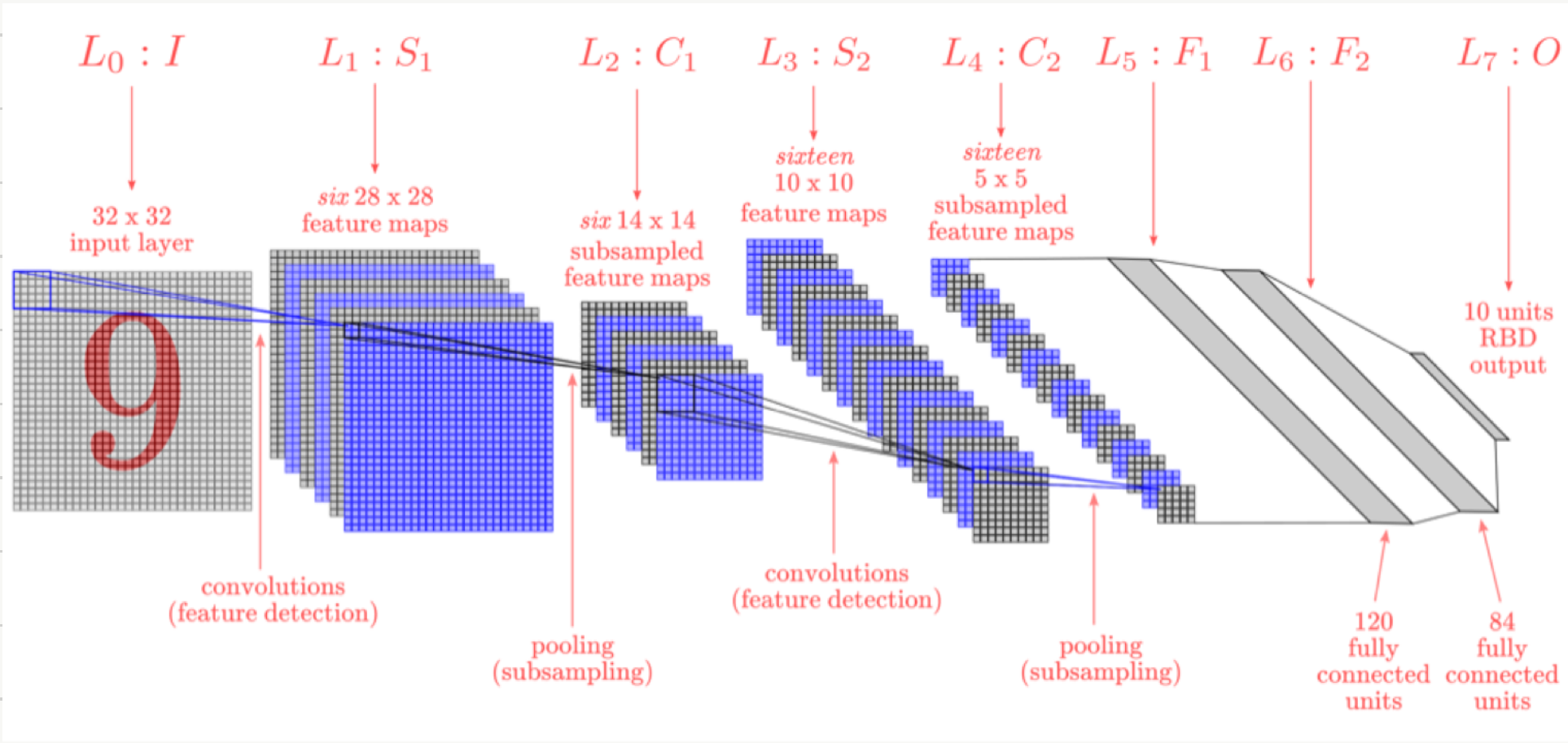
→ Similar to applying filter in convolution layer
→ MAX, AVG pooling
→ used to shrink the input size

Filter size: f
Stride length: s
Type of pooling: MAX or AVG
Padding: p [typically 0]

input: $n \times h \times c$
output: $\left(\left\lfloor \frac{n-f}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{h-f}{s} \right\rfloor + 1\right) \times c$

Pooling layer has fixed computation
⇒ no parameters to learn

Architecture of CNN:
LeNet-5 (LeCun et al 1998)



Parameters ~ 60K

Training a CNN:

→ Similar to training a fully connected network