Convolutional Neural Networks

Dr. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

Up to now,

- Traditional Machine Learning Algorithms
- Deep learning
  - Introduction to Deep Learning
  - Functional view and features
  - Backward and forward computation (including backpropogation and chain rule)

Topics

- convolutional neural networks (CNN)
  - Fully-connected
  - Convolutional Layer
  - Advantage of Convolutional Layer
  - Zero-padding Layer
  - ReLU Layer
  - Pooling
  - Softmax
  - Preprocessing data
- Example
  - LeNet

```
model = Sequential()

model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                        border_mode='valid',
                        input_shape=(1, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Convolutional layer

ReLU layer

Convolutional layer

ReLU layer

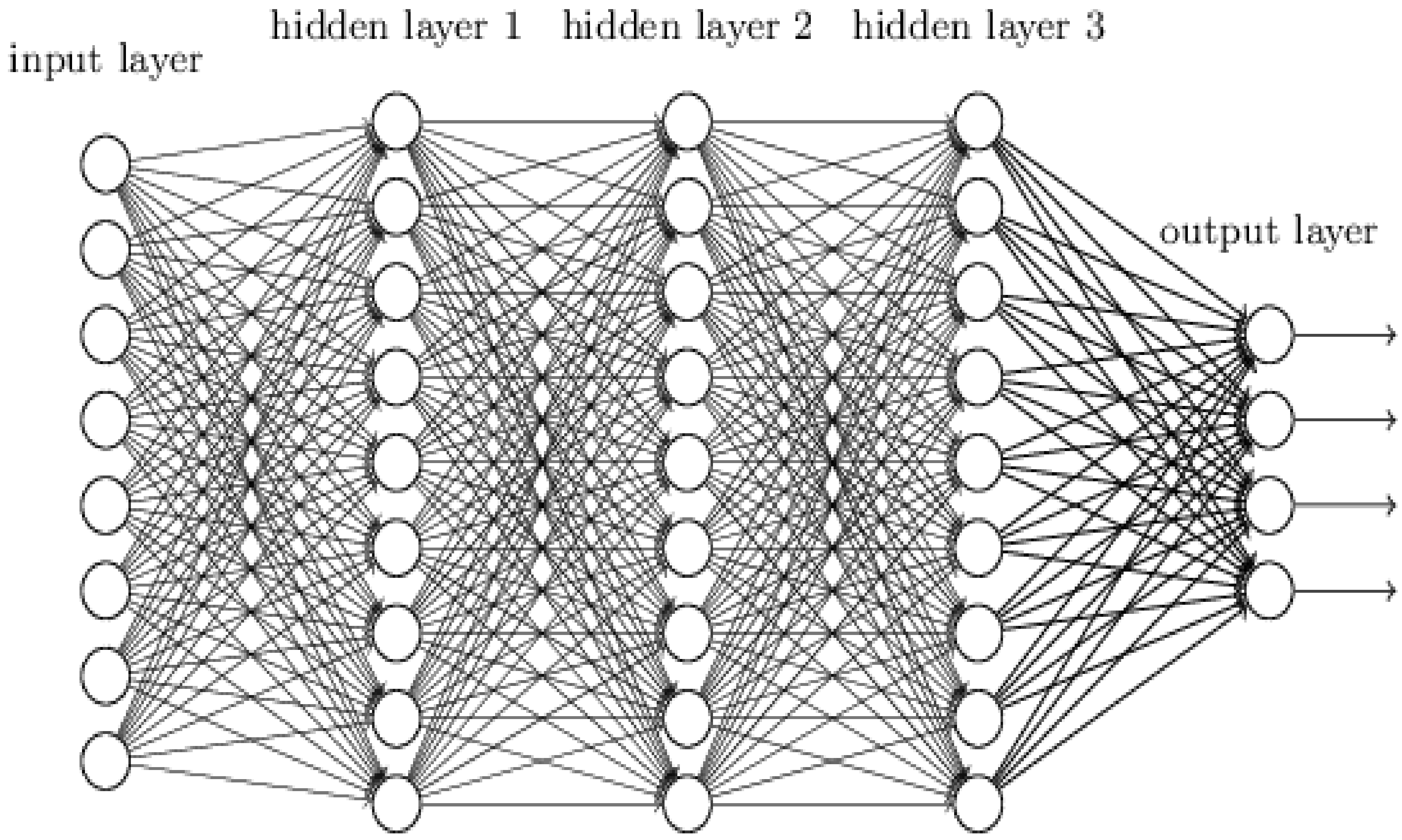
Maxpooling layer

Dropout layer: for regularisation

Flatten layer: from convolutional to fully-connected

Fully-connected layer

Fully-connected



hidden layer 1
hidden layer 2
hidden layer 3
input layer
output layer

Convolution

Convolutional neural networks

- Strong empirical application performance
- Convolutional networks: neural networks that use convolution in place of general matrix multiplication in at least one of their layers

$$h = \sigma(W^T x + b)$$

for a specific kind of weight matrix W

Convolutional layer illustrated

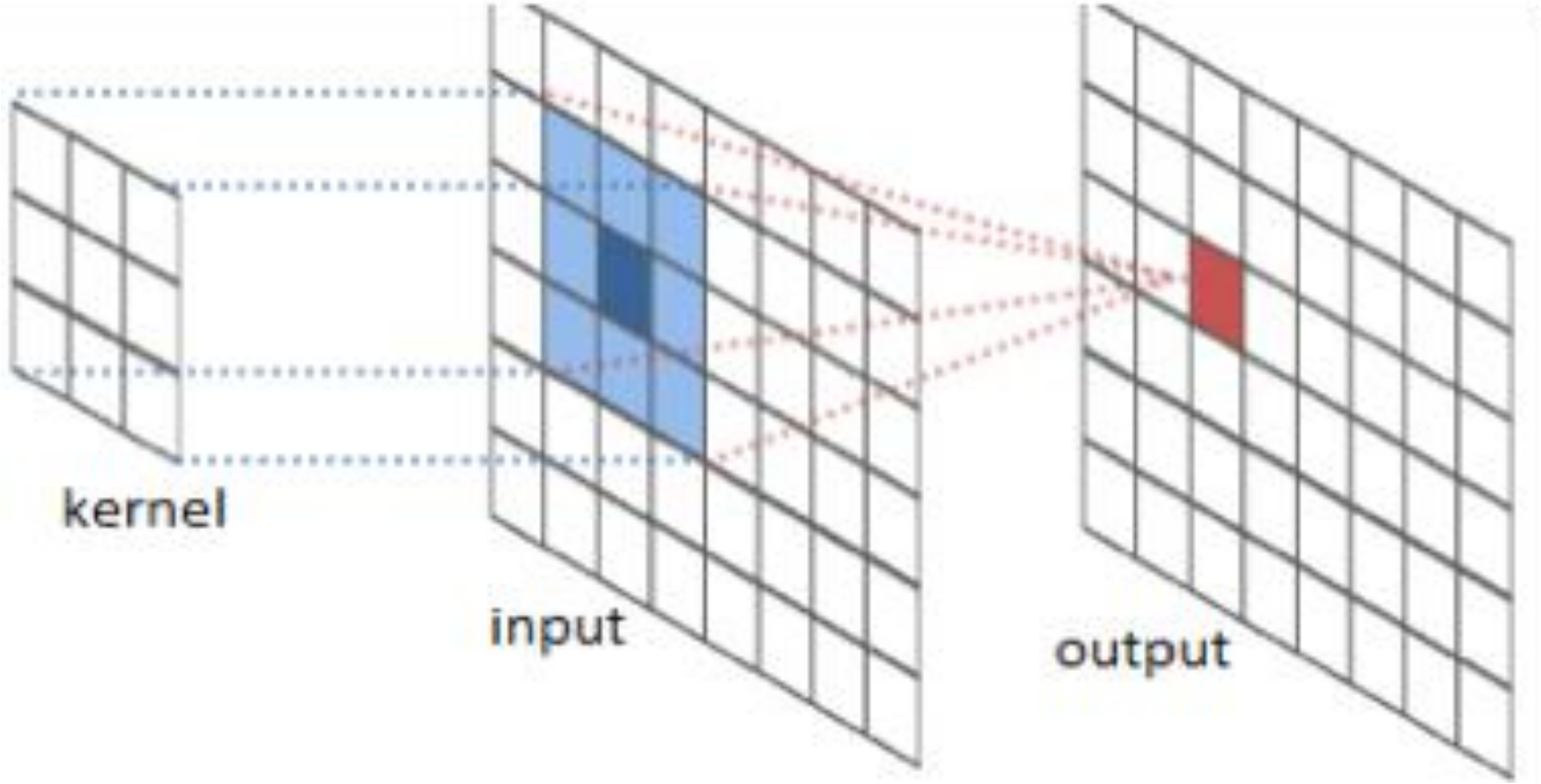


Illustration 1

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

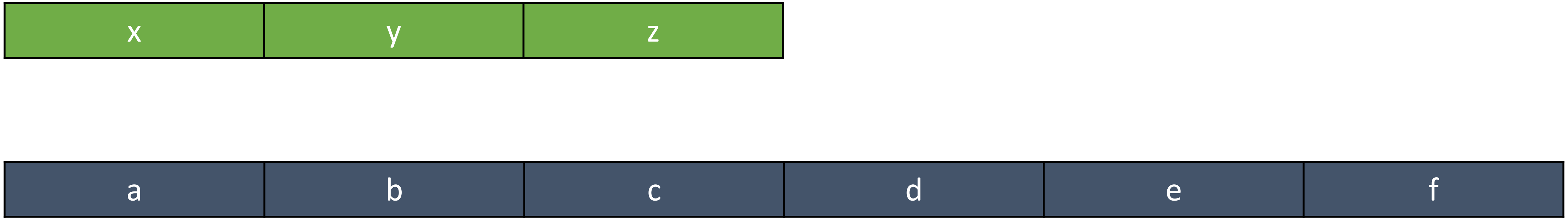


Illustration 1

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

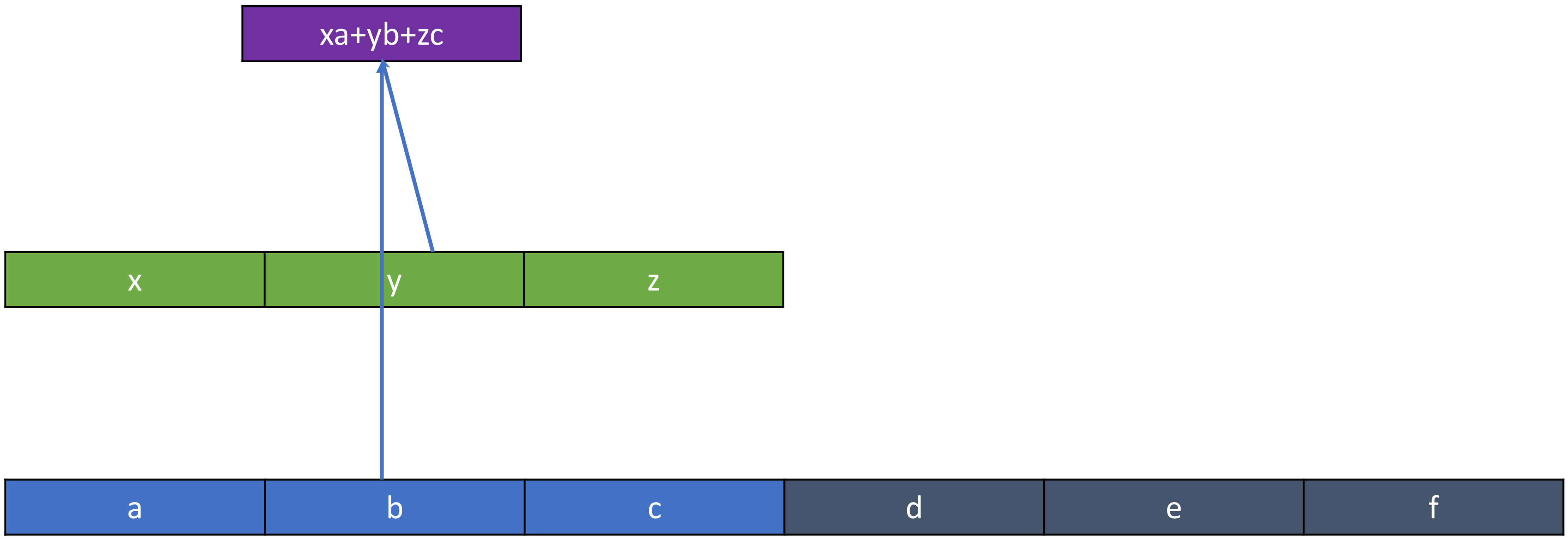


Illustration 1

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

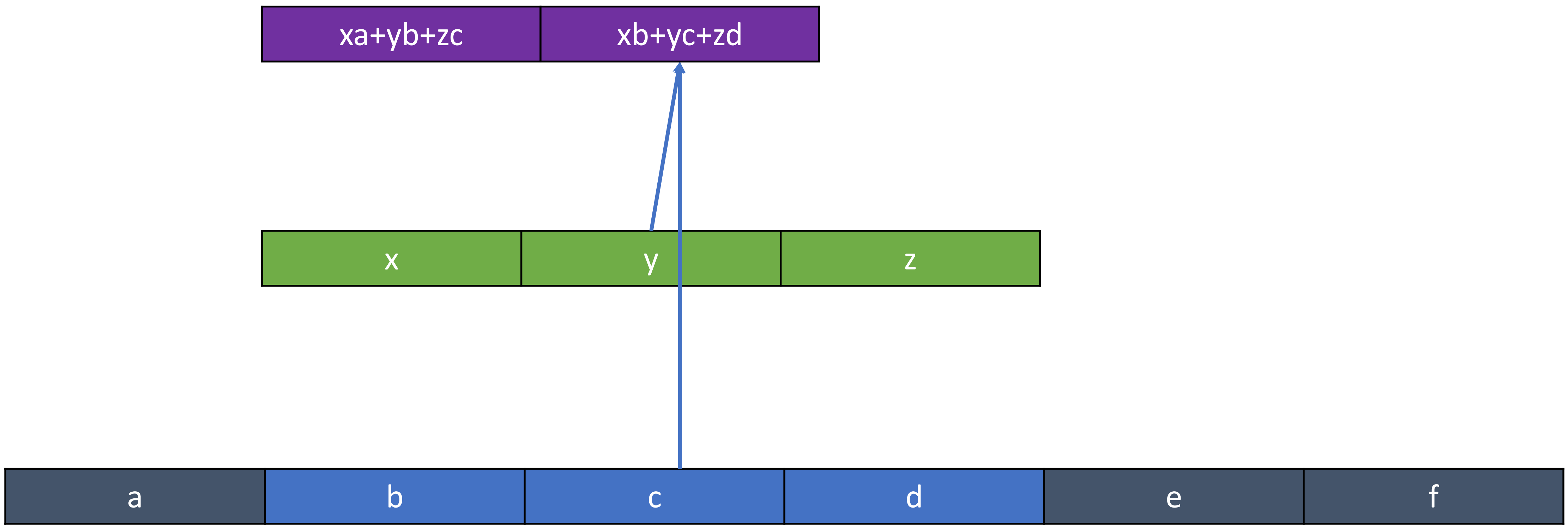


Illustration 1

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

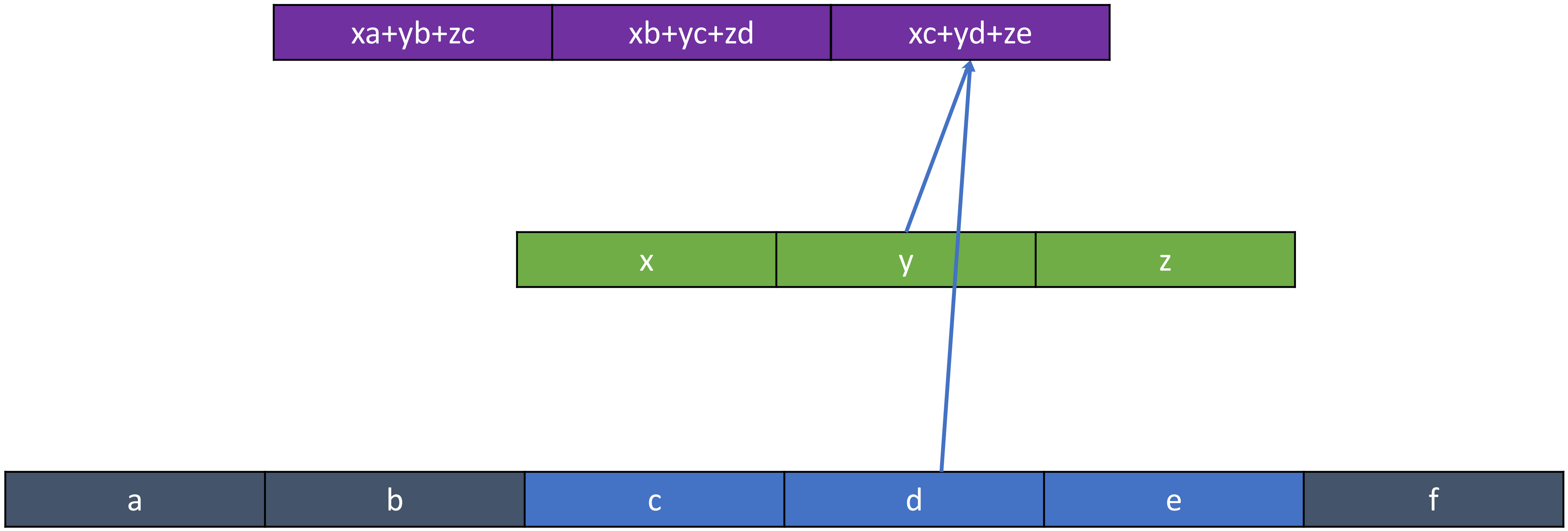


Illustration 1

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

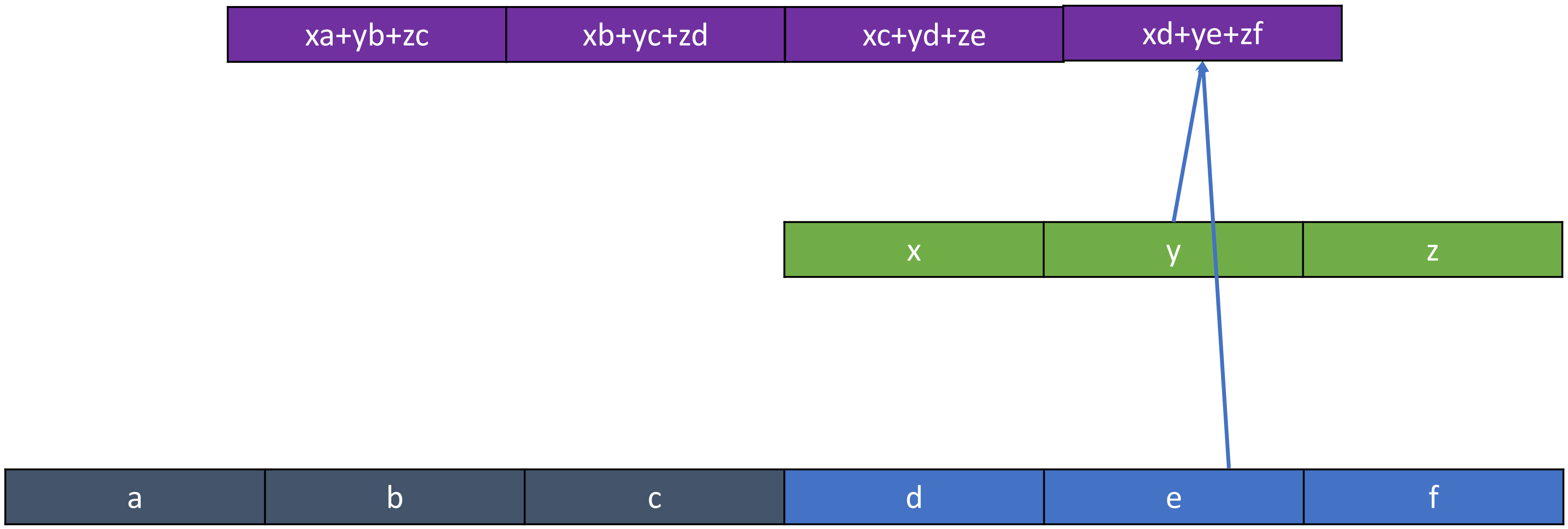


Illustration 1: boundary case

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

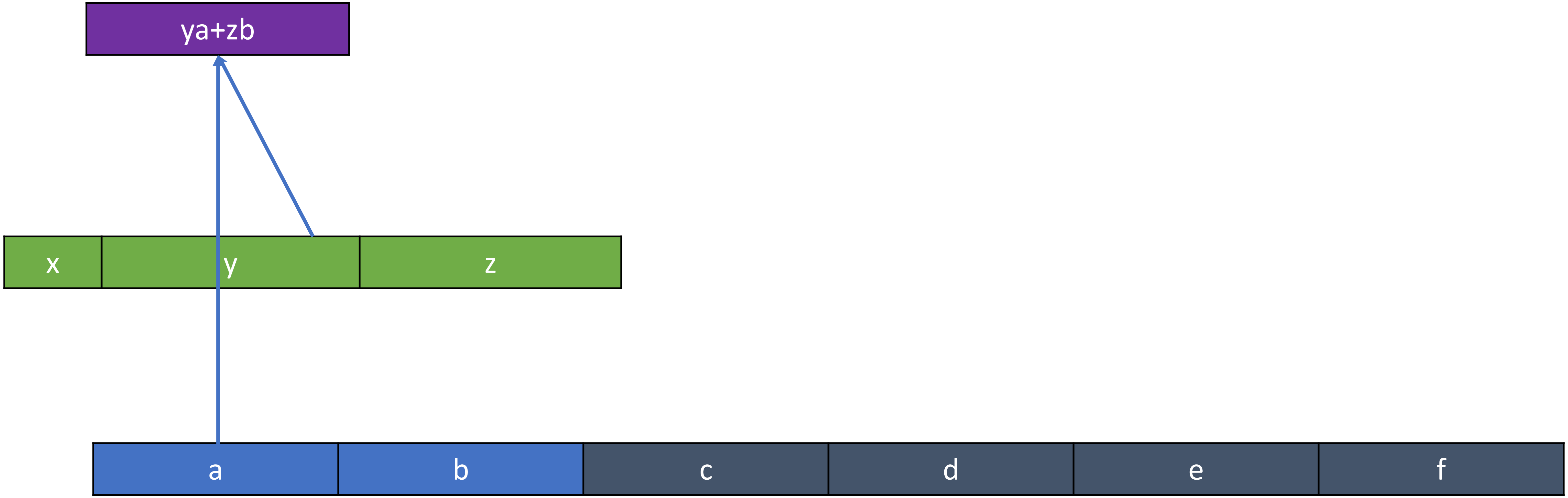


Illustration 1: boundary case

$$w = [z, y, x]$$
$$u = [a, b, c, d, e, f]$$

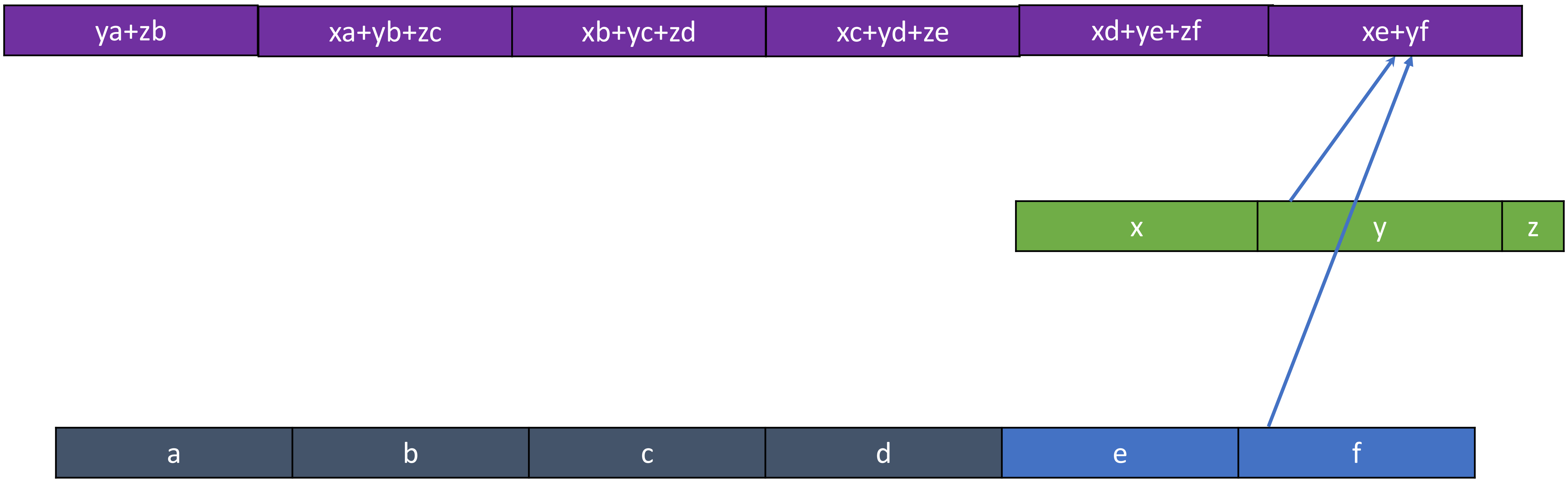


Illustration 1: as matrix multiplication

$$
\begin{bmatrix}
y & z & & & & \\
x & y & z & & & \\
 & x & y & z & & \\
 & & x & y & z & \\
 & & & x & y & z \\
 & & & & x & y
\end{bmatrix}
\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}
\Rightarrow
\begin{bmatrix} ya+zb \\ xa+yb+zc \\ xb+yc+zd \\ xc+yd+ze \\ xd+ye+zf \\ xe+yf \end{bmatrix}
$$

Illustration 2: two dimensional case

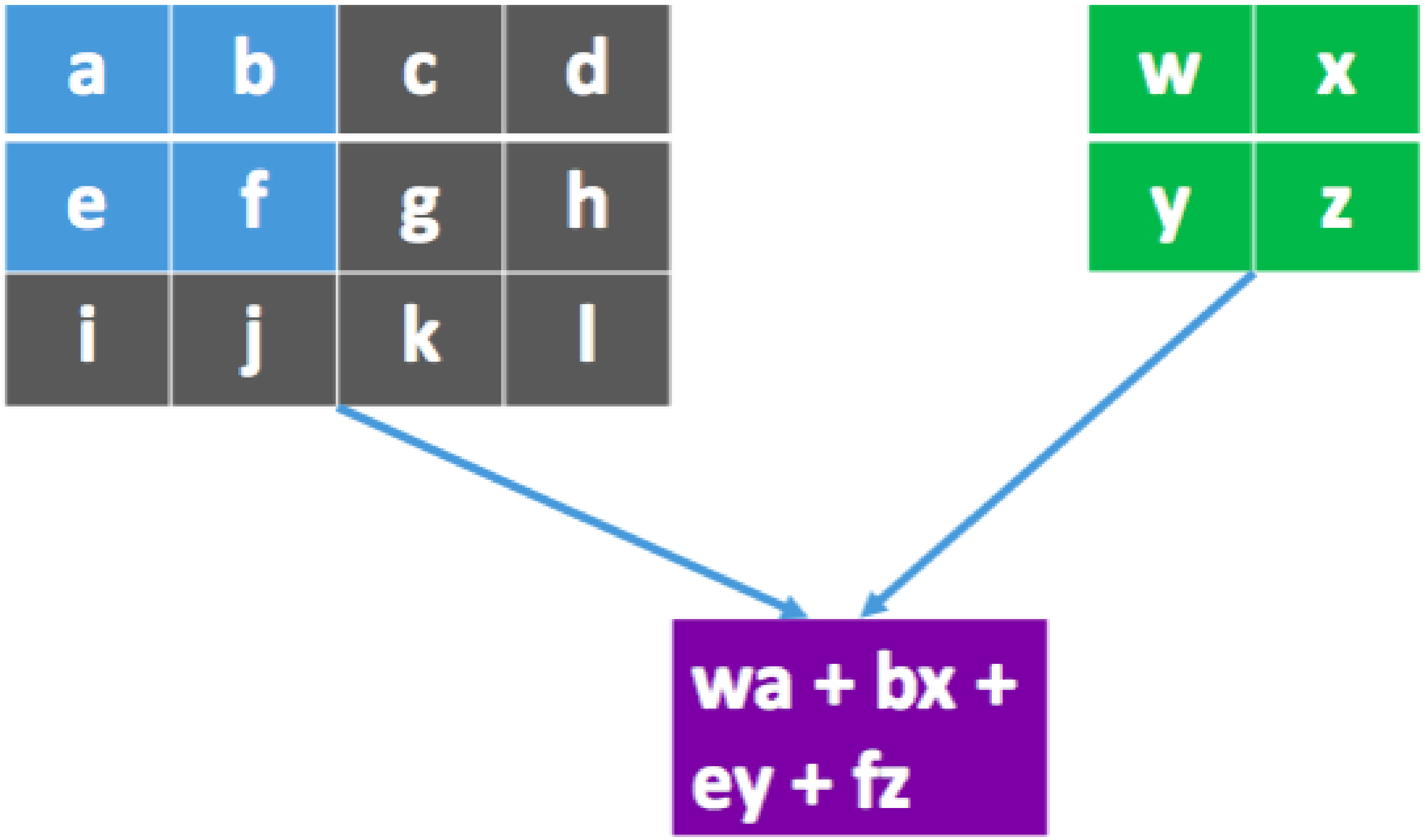


Illustration 2

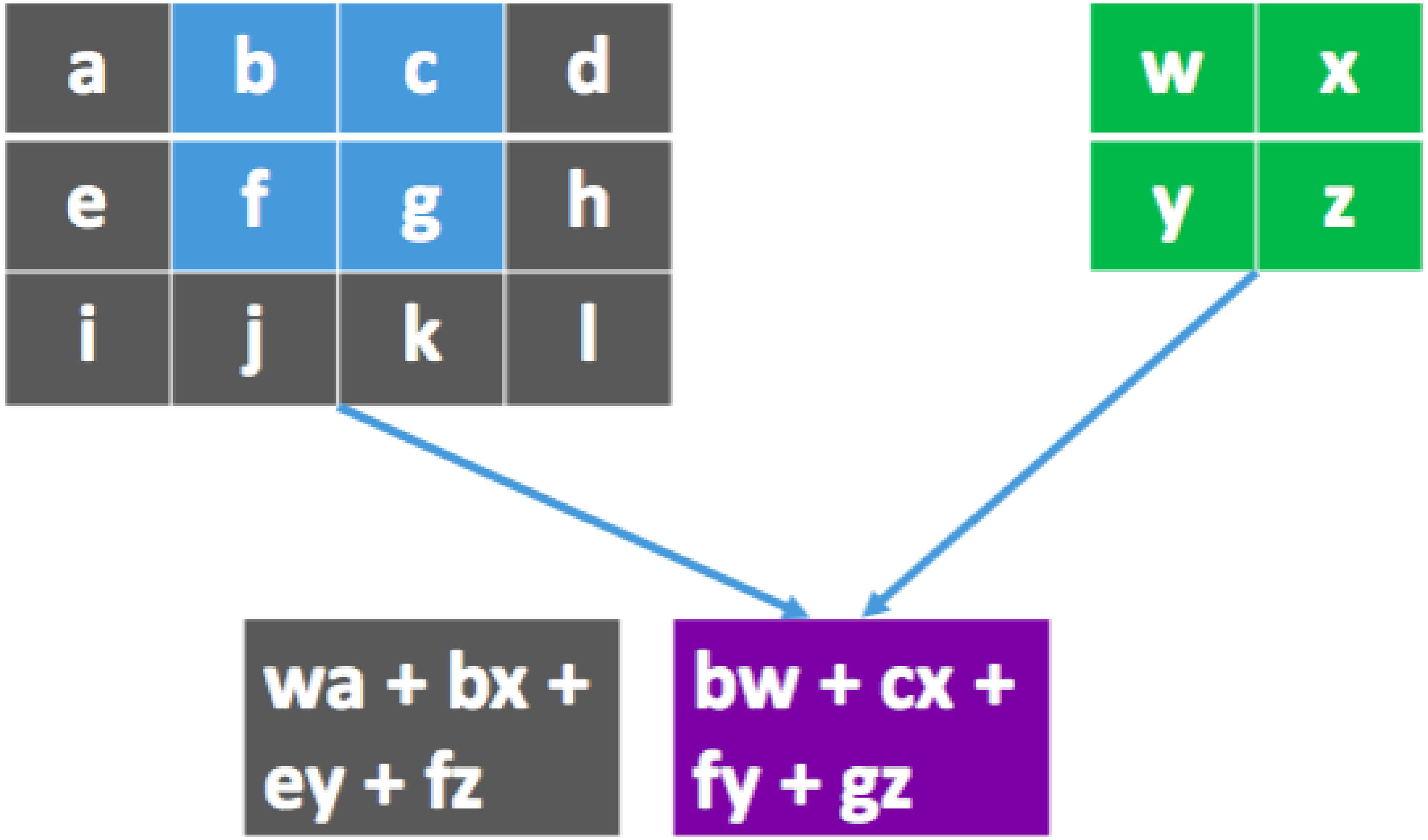
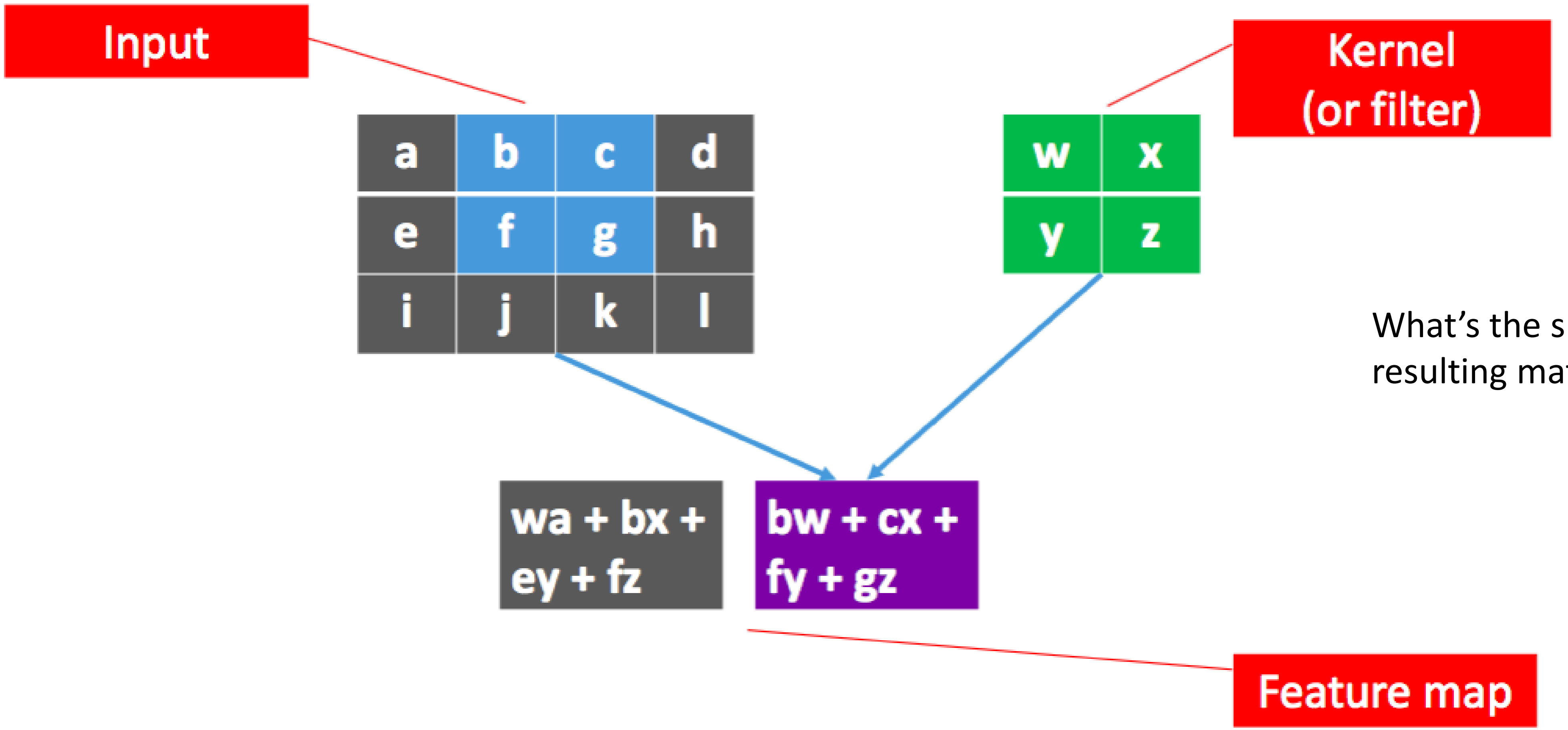
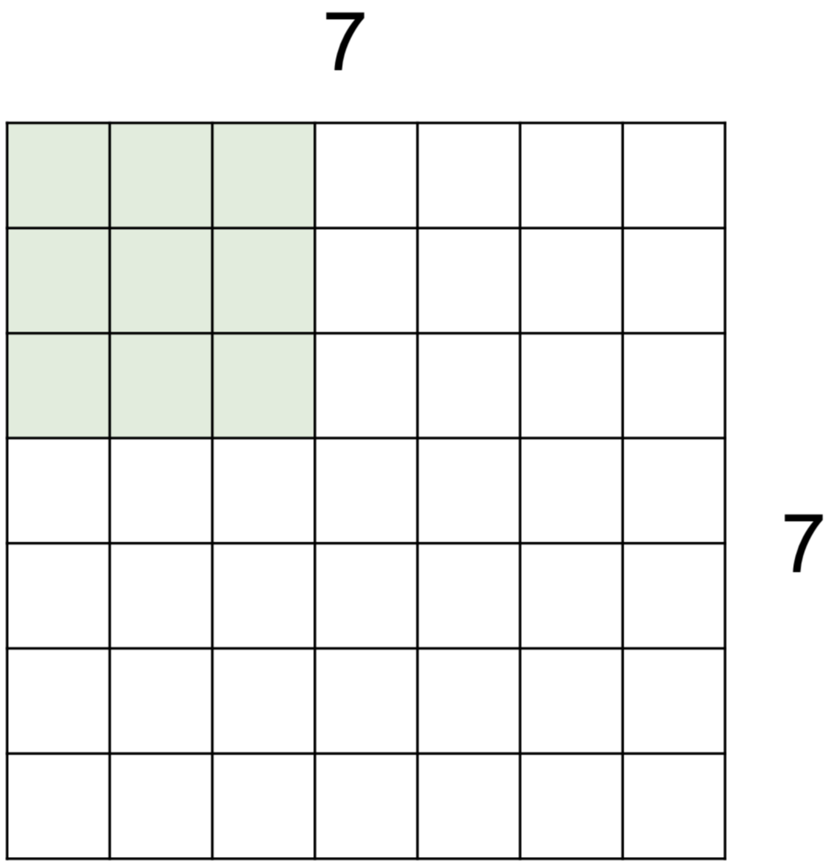


Illustration 2



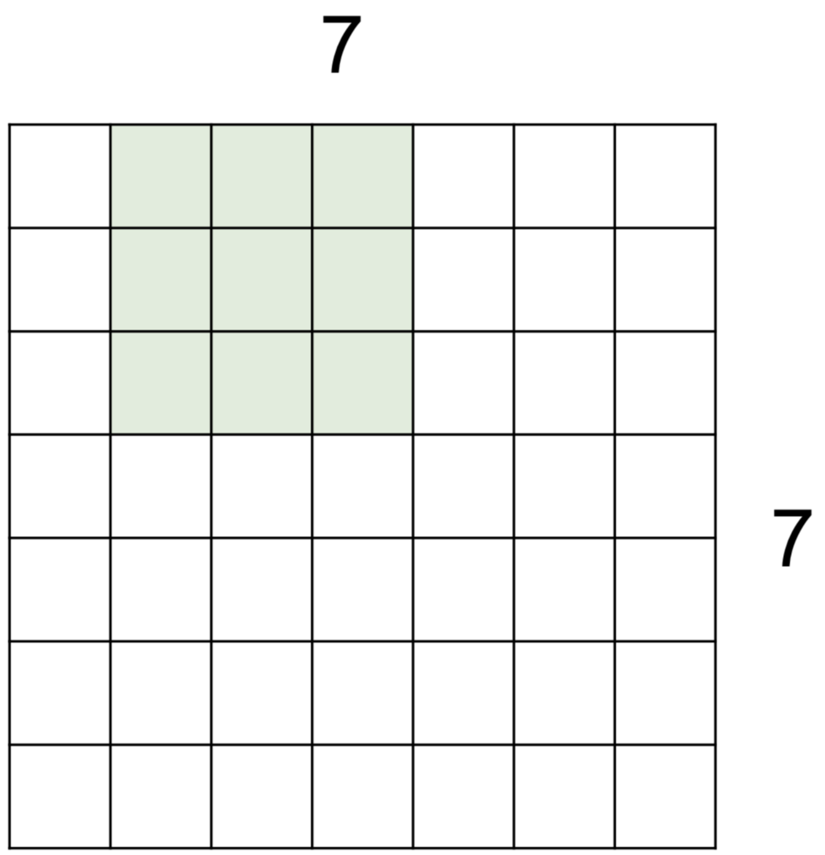
What's the shape of the resulting matrix?

A closer look at spatial dimensions:



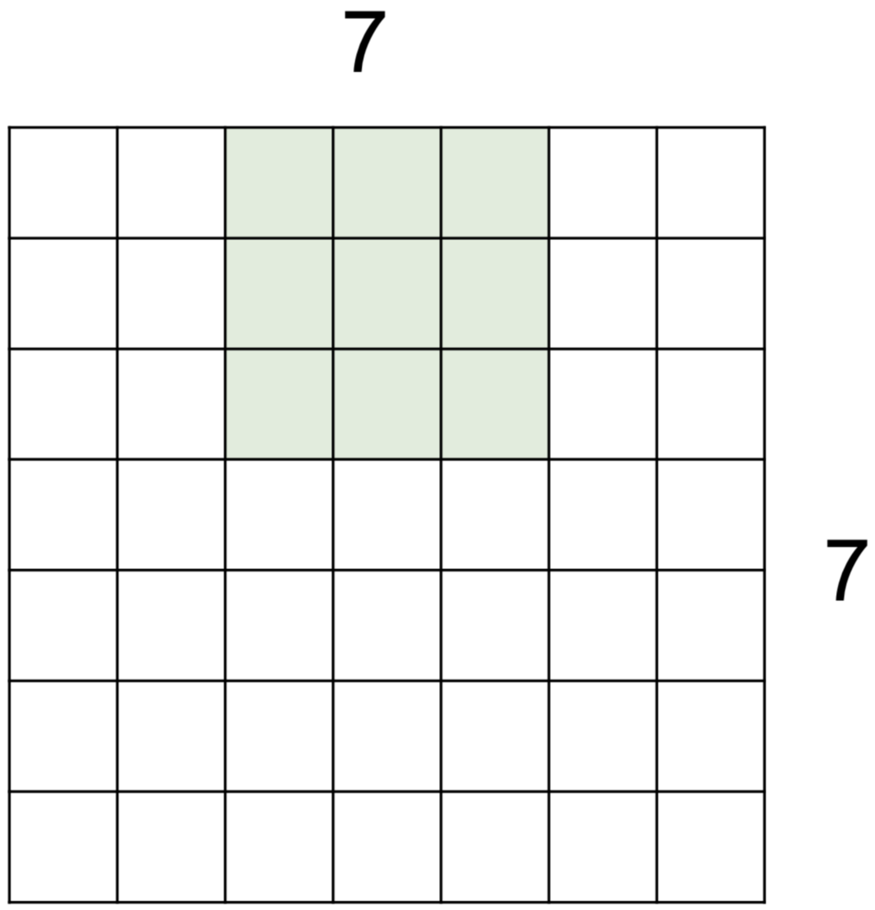
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



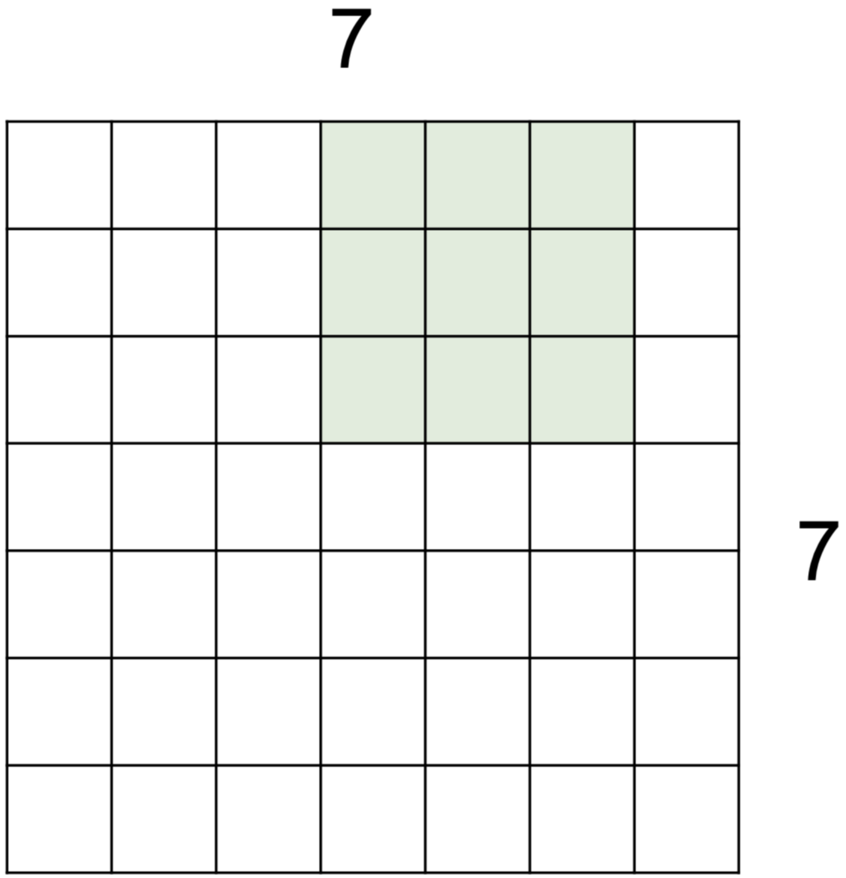
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



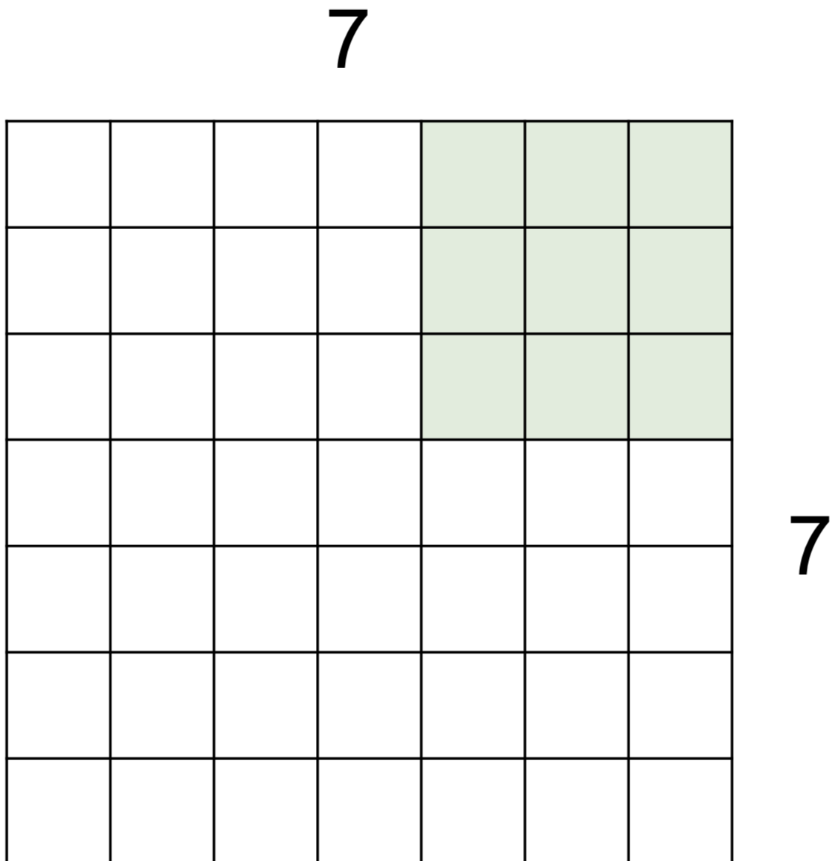
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

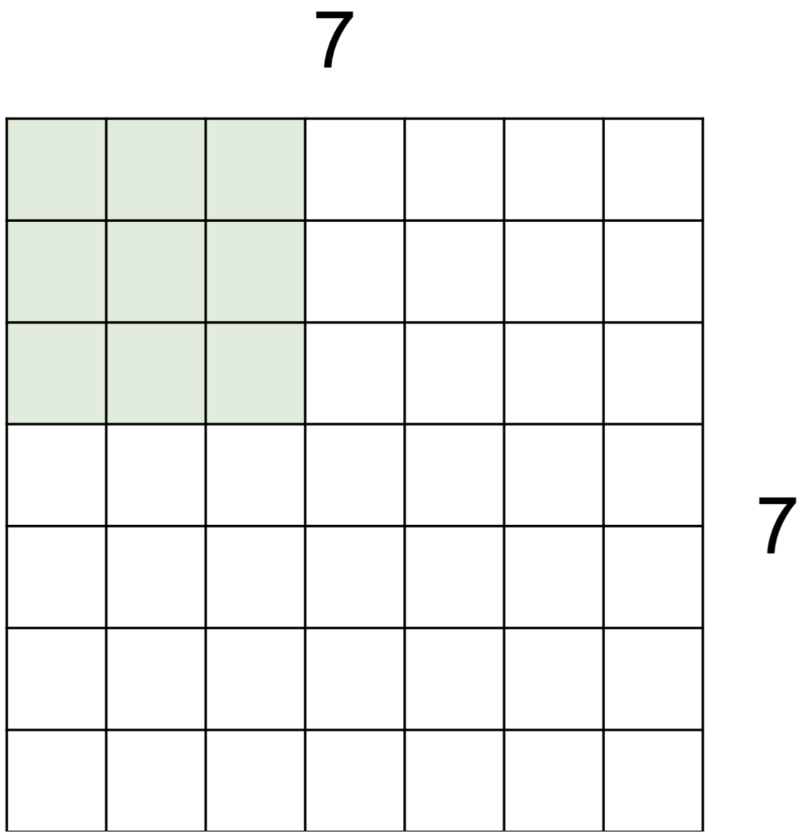
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

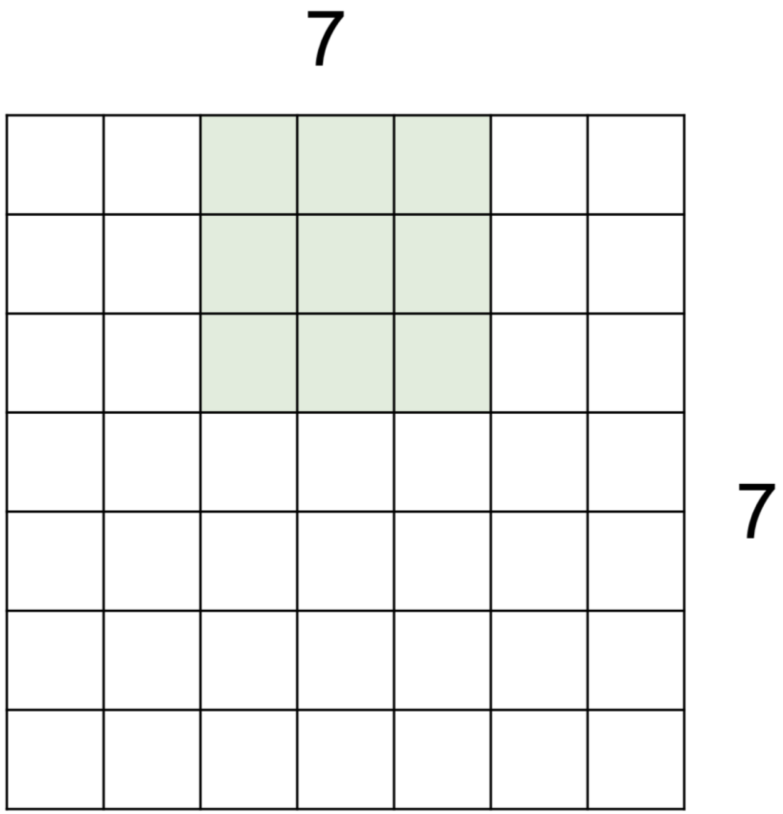
=> 5x5 output

A closer look at spatial dimensions:



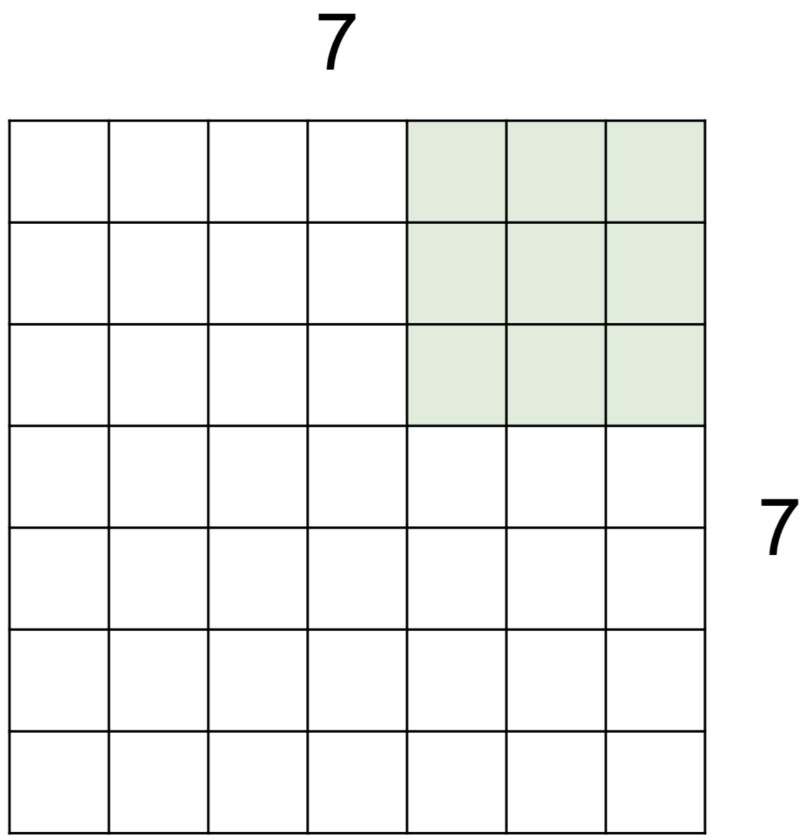
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



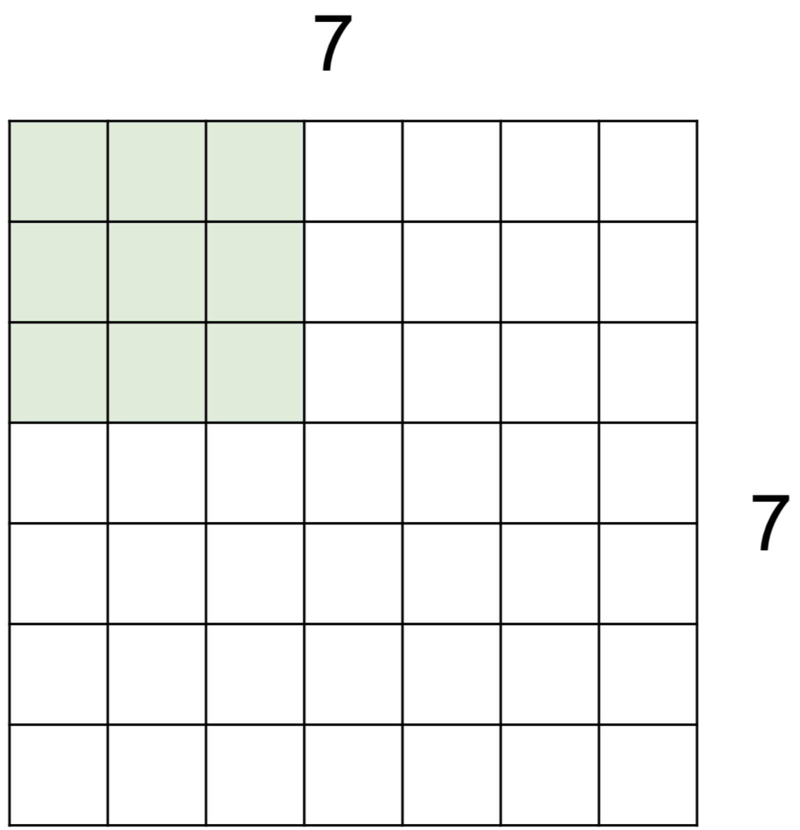
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



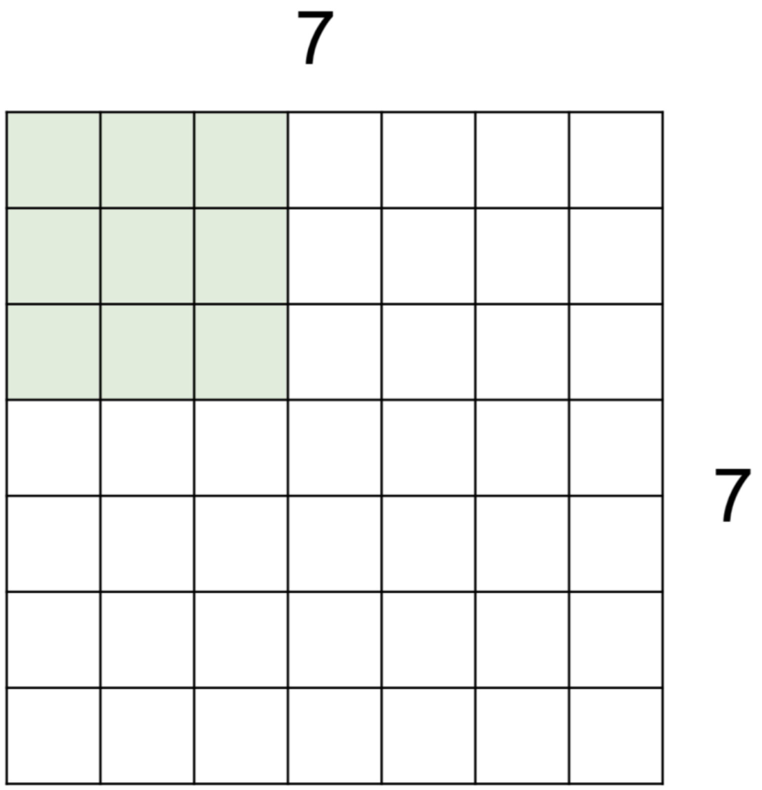
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

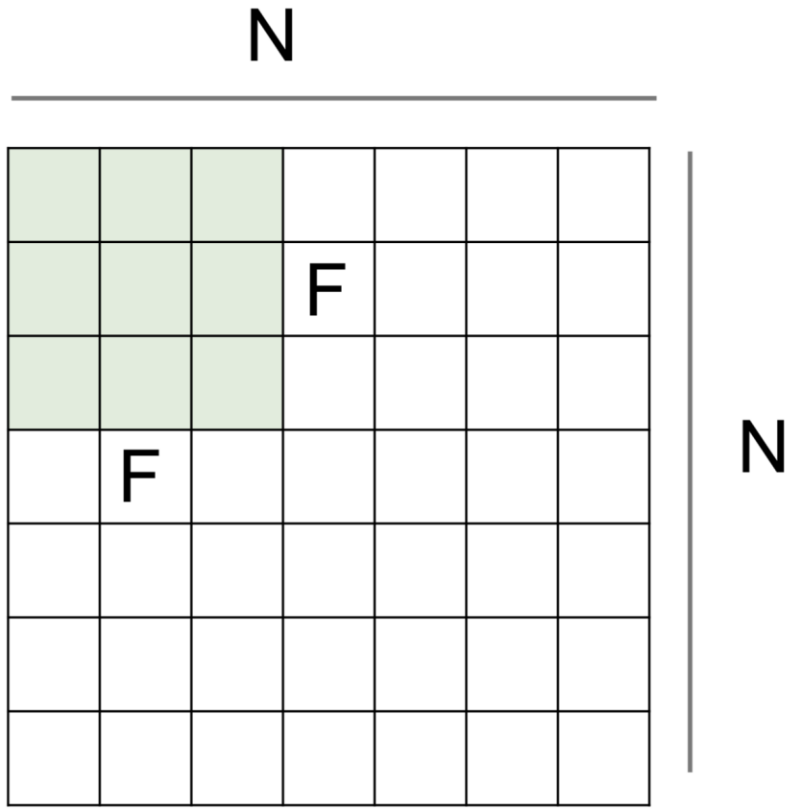
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

A closer look at spatial dimensions:



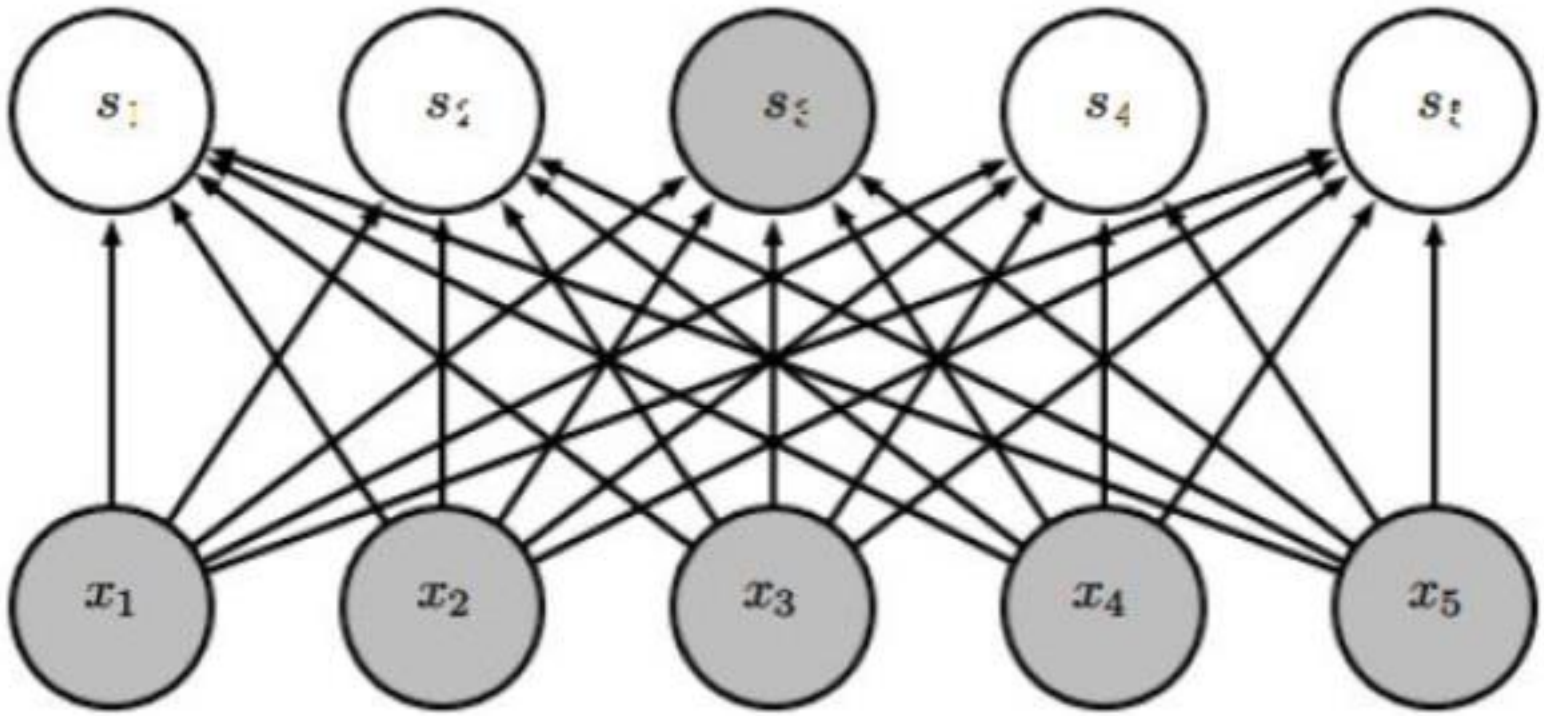
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Advantage of Convolutional Layer

Advantage: sparse interaction

Fully connected layer, $m \times n$ edges

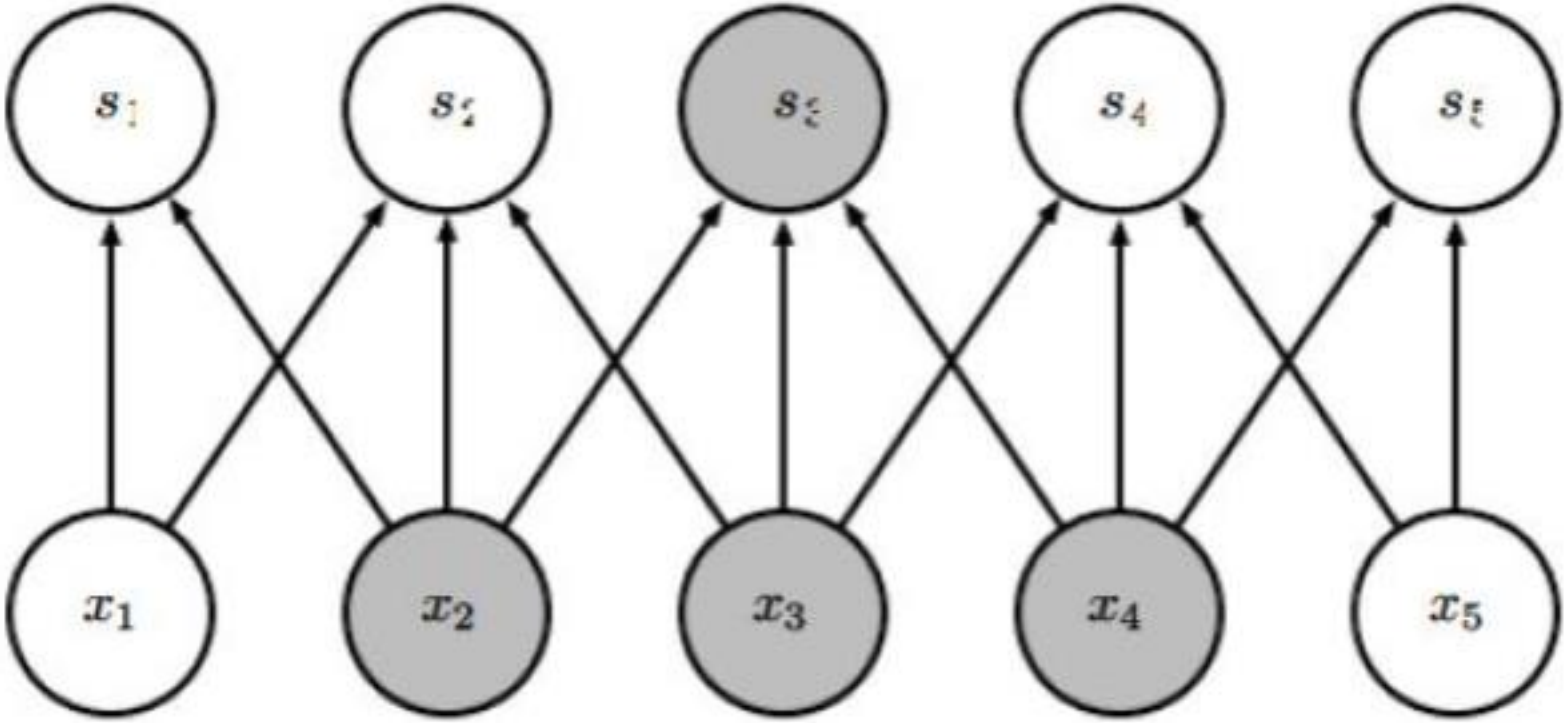


m output nodes

n input nodes

Advantage: sparse interaction

Convolutional layer, $\leq m \times k$ edges



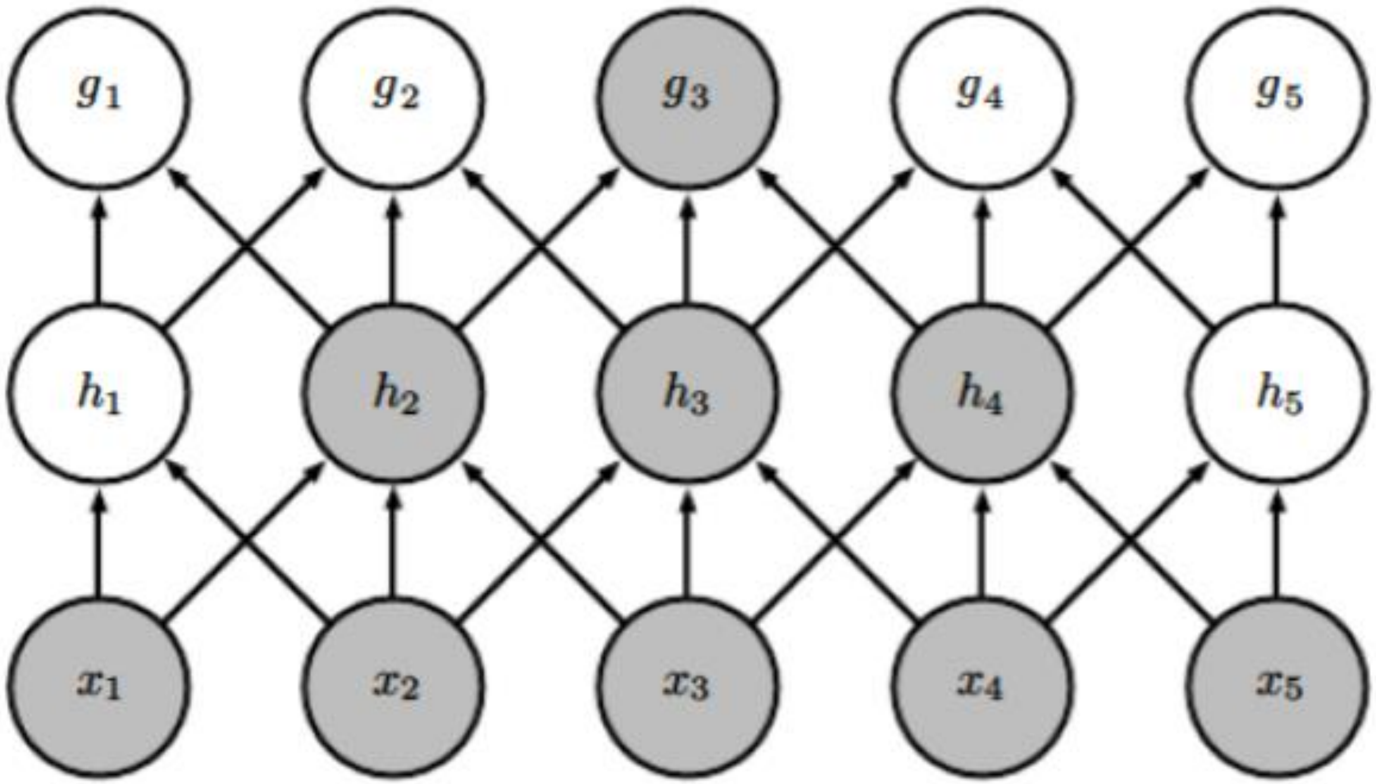
m output nodes

k kernel size

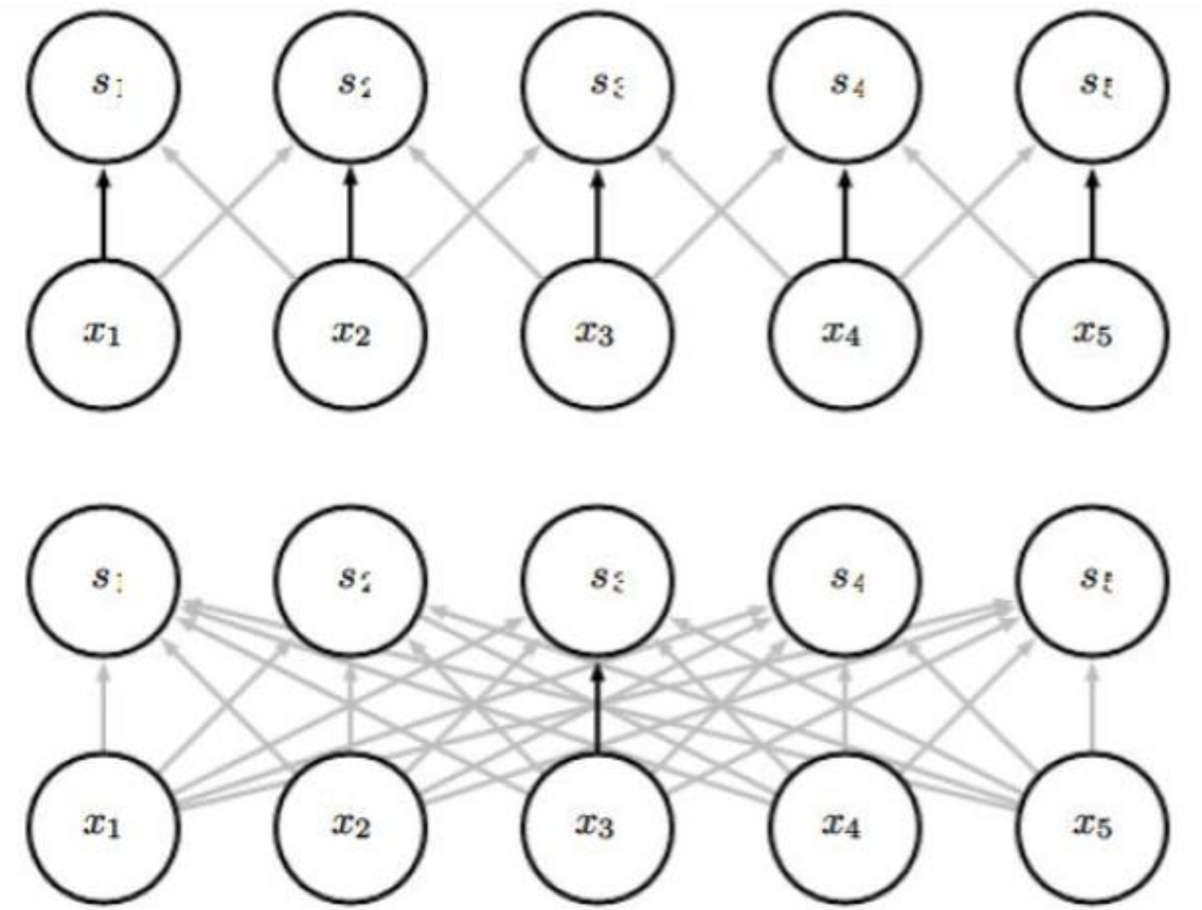
n input nodes

Advantage: sparse interaction

Multiple convolutional layers: larger receptive field



Advantage: parameter sharing/weight tying



The same kernel are used repeatedly. E.g., the black edge is the same weight in the kernel.

Figure from *Deep Learning,* by Goodfellow, Bengio, and Courville

Advantage: equivariant representations

- Equivariant: transforming the input = transforming the output

- Example: input is an image, transformation is shifting
- Convolution(shift(input)) = shift(Convolution(input))

- Useful when care only about the existence of a pattern, rather than the location

Zero-Padding

Zero-Padding

filter

w	x
y	z

Input

a	b	c	d
e	f	g	h
i	j	k	l

0	0	0	0	0	0
0	a	b	c	d	0
0	e	f	g	h	0
0	i	j	k	l	0
0	0	0	0	0	0

What's the shape of the resulting matrix?

ReLU

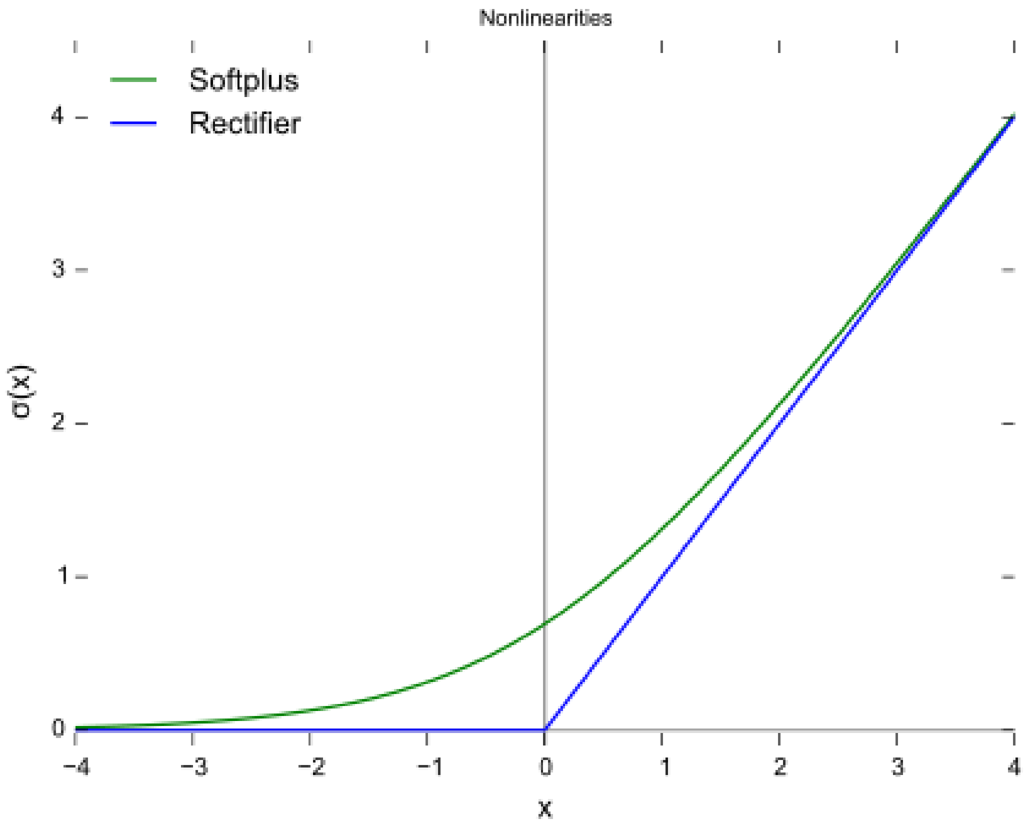
ReLU (**rectified linear unit**)

- **rectifier** is an activation function defined as the positive part of its argument

$$f(x) = \max(0,x)$$

- A smooth approximation to the rectifier is the analytic function

$$f(x) = \log(1+e^{x})$$

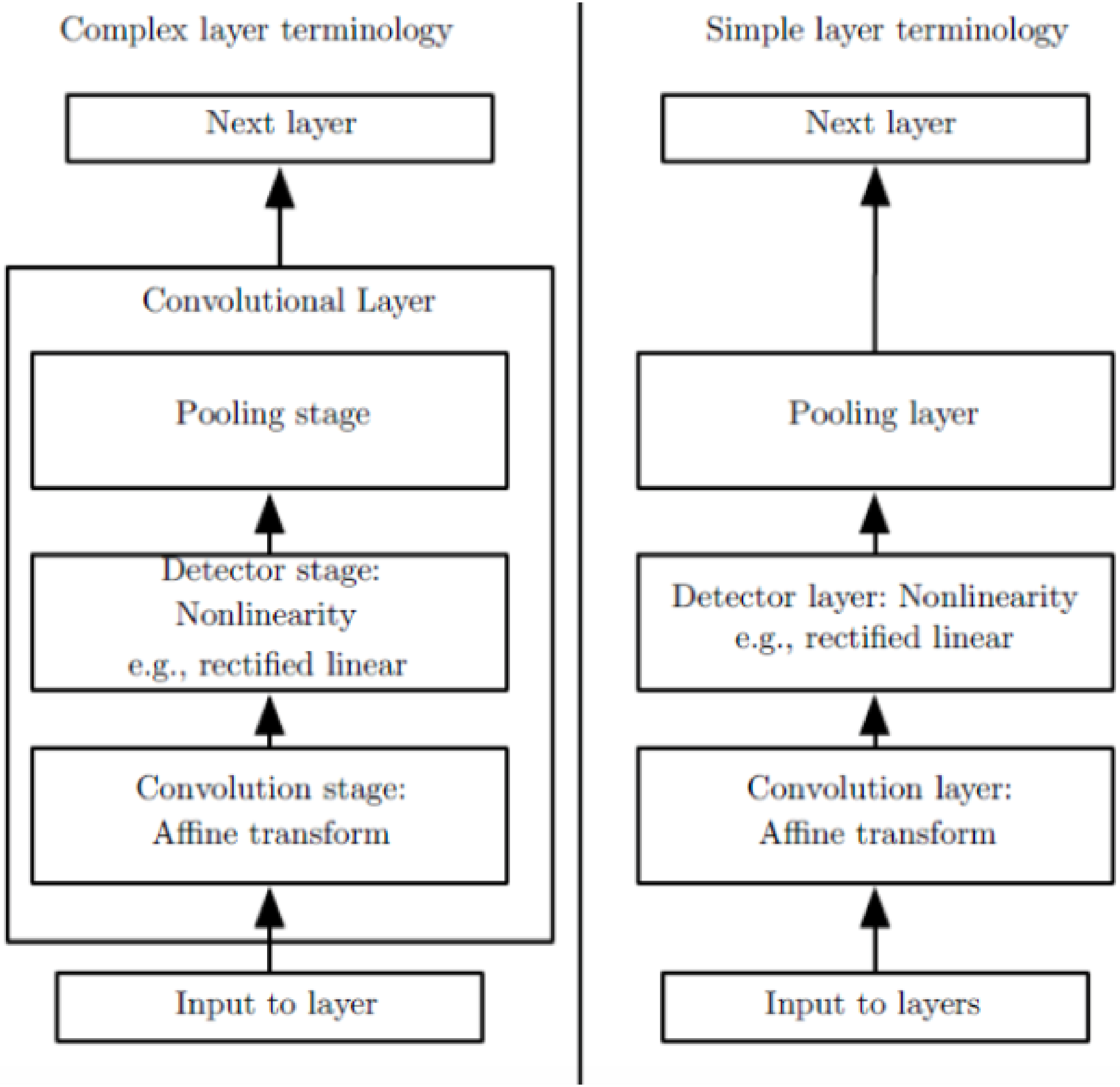


Pooling

Pooling

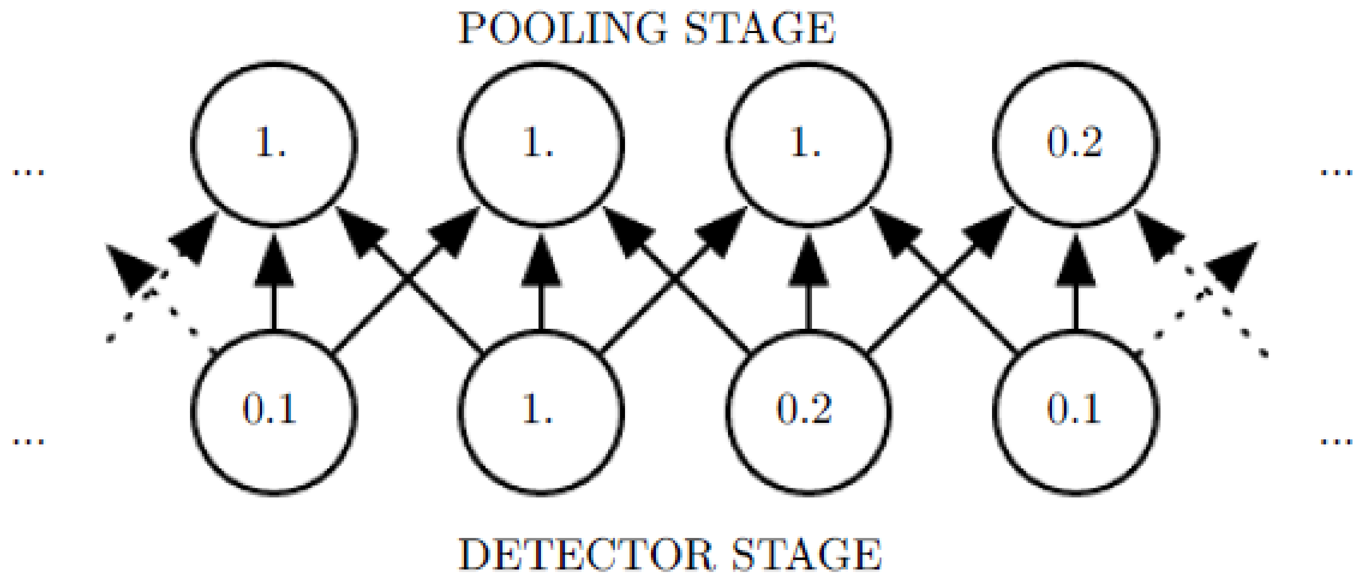
- Pooling layer is frequently used in convolutional neural networks with the purpose to progressively reduce the spatial size of the representation to reduce the amount of features and the computational complexity of the network.

Terminology



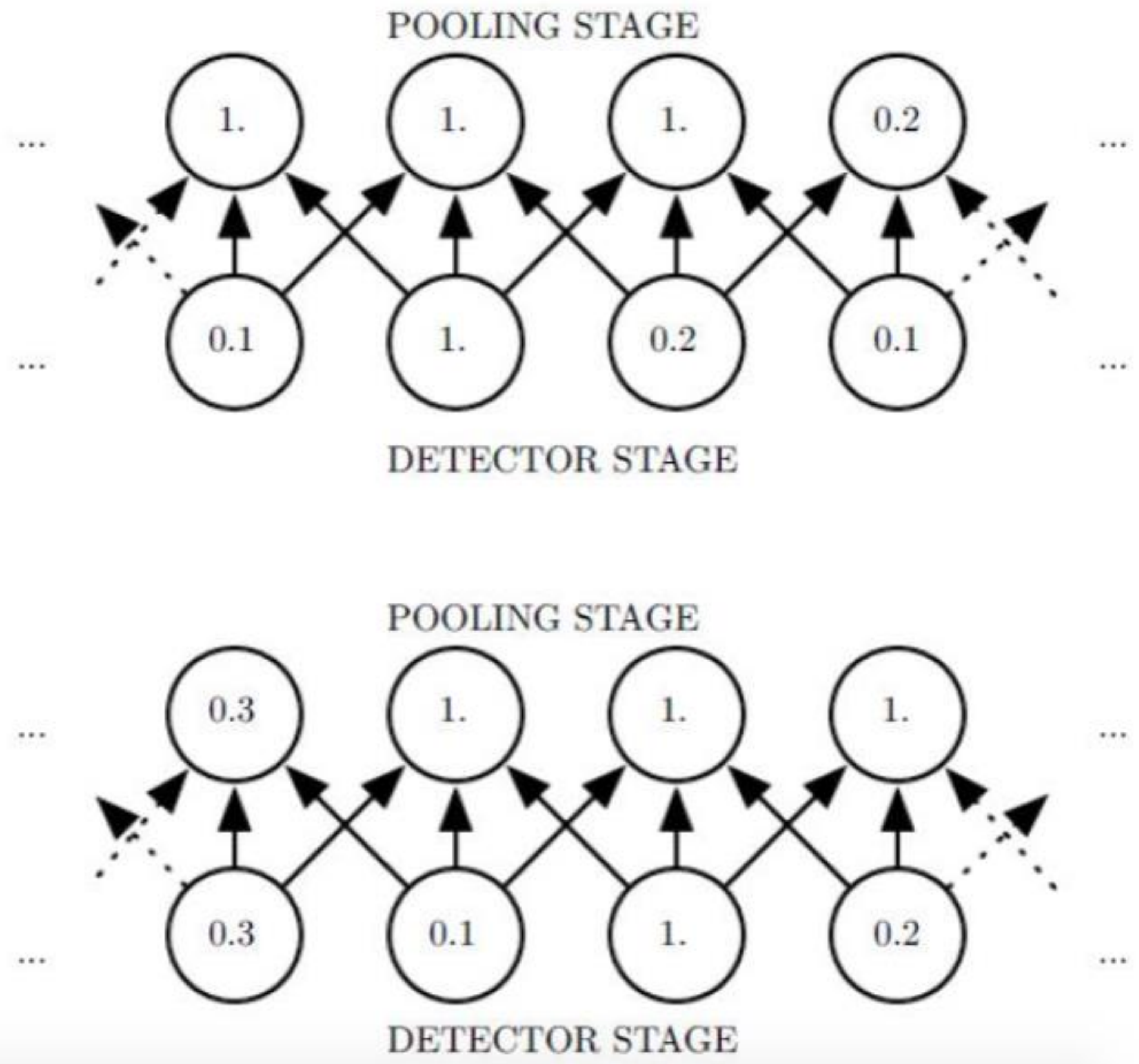
Pooling

- Summarizing the input (i.e., output the max of the input)



Advantage

- Induce invariance



Example: Max-pooling

max{1,3,7,2} = 7

max{4,5,1,9} = 9

1	3	4	5
7	2	9	1
9	2	4	7
4	3	6	2

7	9
9	7

max{9,2,3,4} = 9

max{4,7,6,2} = 7

Motivation from neuroscience

- David Hubel and Torsten Wiesel studied early visual system in human brain (V1 or primary visual cortex), and won Nobel prize for this

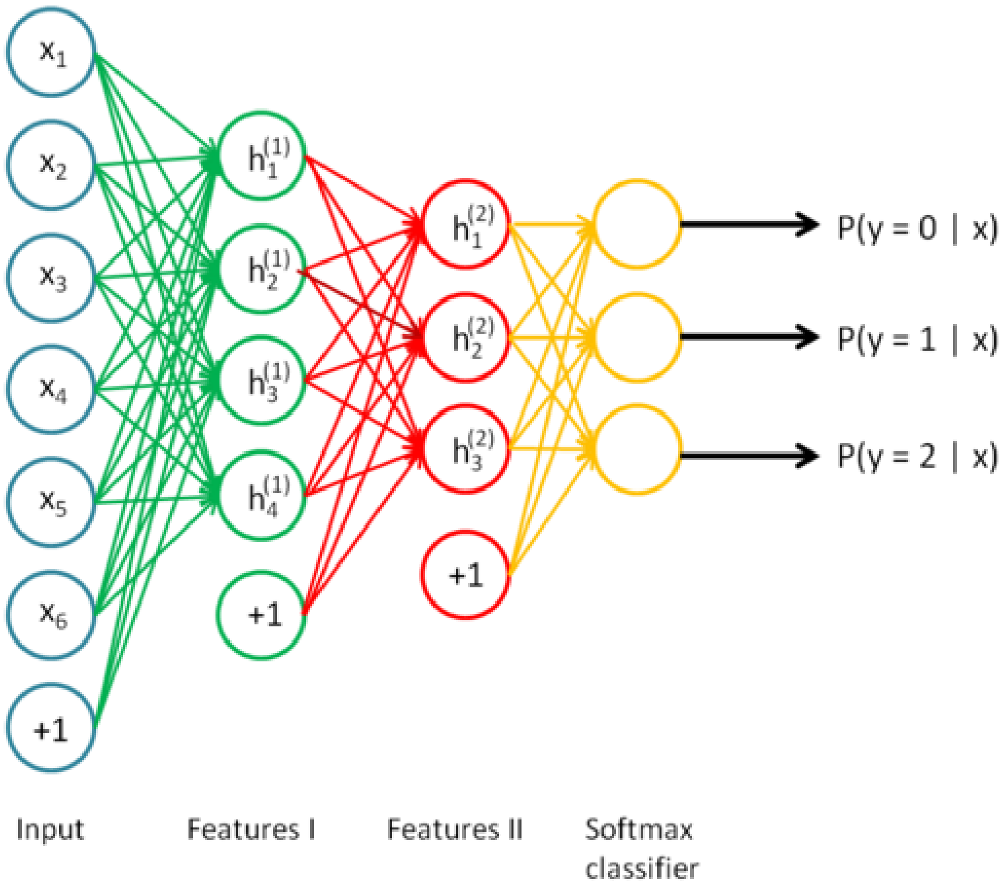
- V1 properties
  - 2D spatial arrangement
  - Simple cells: inspire convolution layers
  - Complex cells: inspire pooling layers

Softmax

Softmax

- Recall that logistic regression produces a decimal between 0 and 1.0. For example, a logistic regression output of 0.8 from an email classifier suggests an 80% chance of an email being spam and a 20% chance of it being not spam. Clearly, the sum of the probabilities of an email being either spam or not spam is 1.0.
- **Softmax** extends this idea into a multi-class world. That is, Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would.

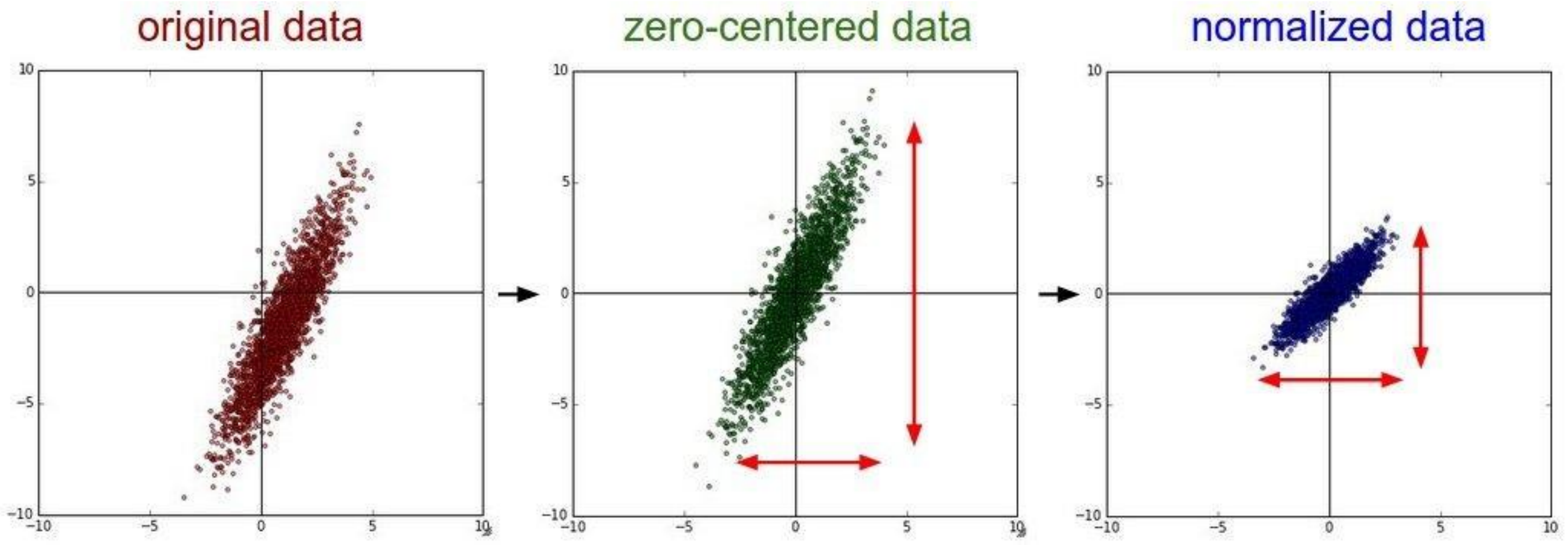
Softmax



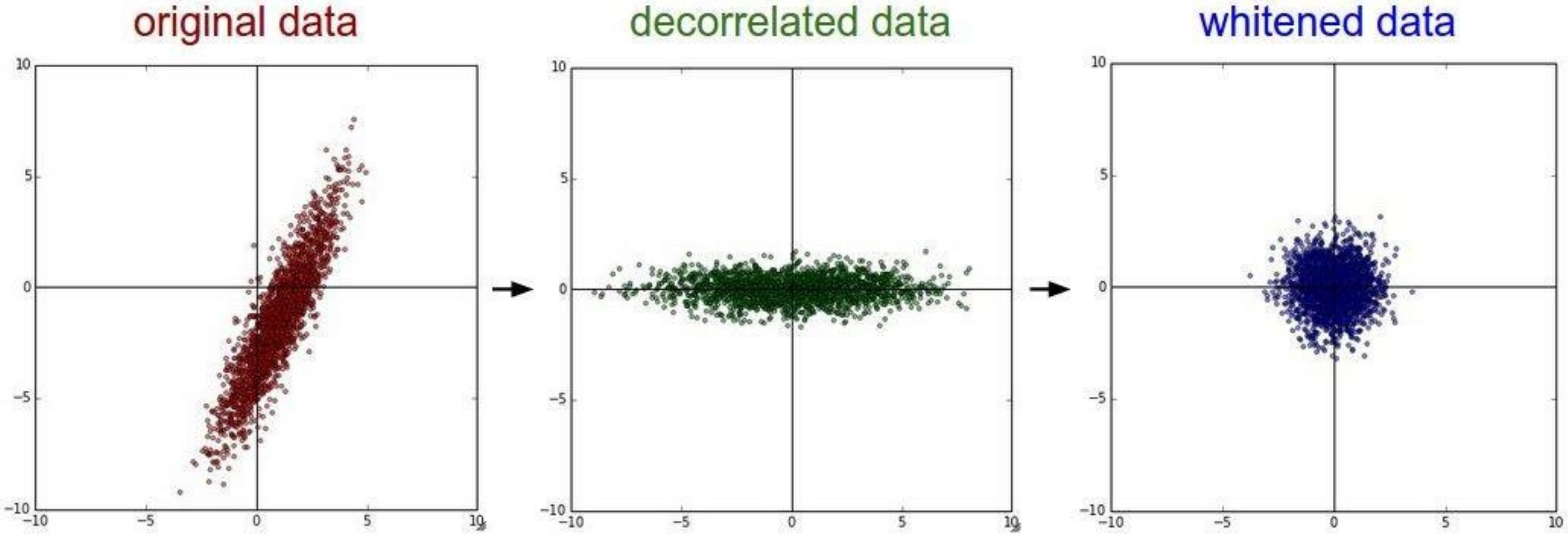
x1
x2
x3
x4
x5
x6
+1
h(1)1
h(1)2
h(1)3
h(1)4
+1
h(2)1
h(2)2
h(2)3
+1
P(y = 0 | x)
P(y = 1 | x)
P(y = 2 | x)
Input
Features I
Features II
Softmax classifier

Preprocessing data

Preprocessing data



Preprocessing data



Preprocessing data

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet) (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet) (mean along each channel = 3 numbers)

Example: LeNet

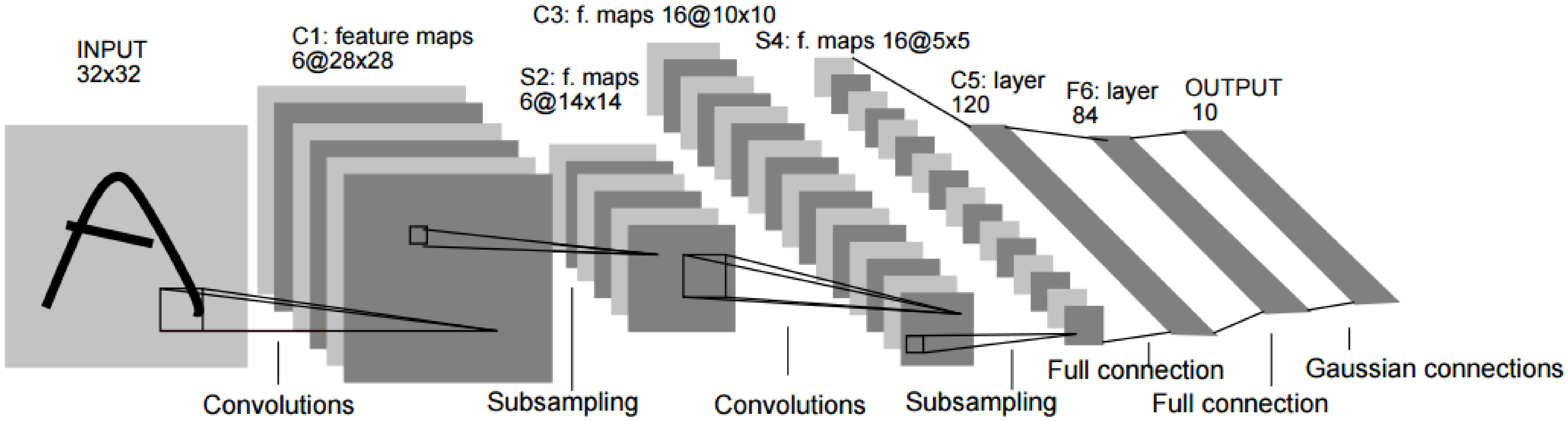
LeNet-5

- Proposed in *"Gradient-based learning applied to document recognition" , by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998*

- Apply convolution on 2D images (MNIST) and use backpropagation

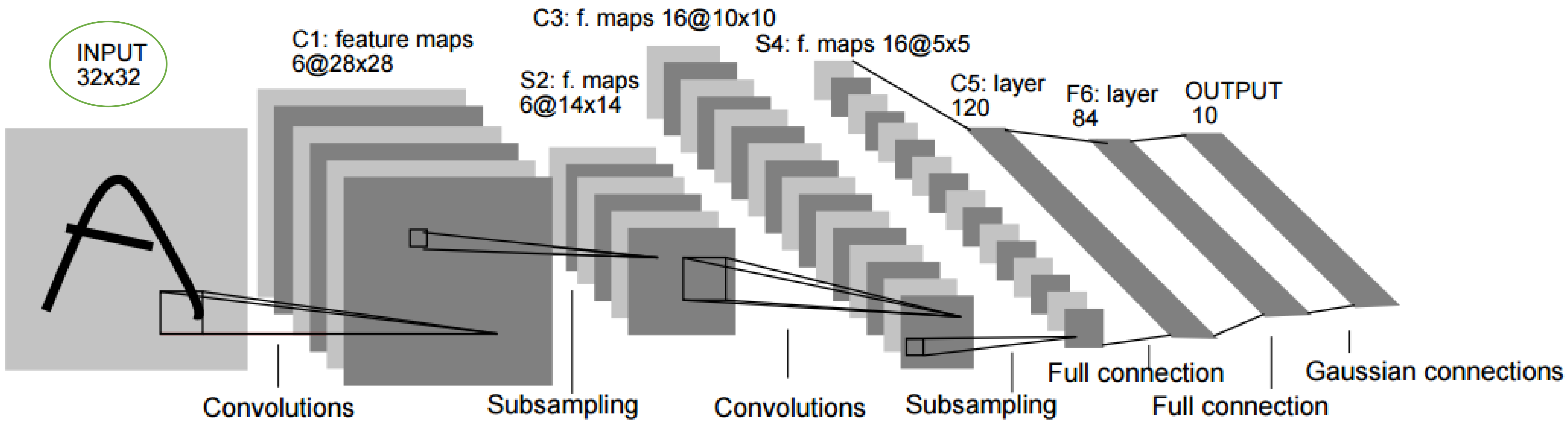
- Structure: 2 convolutional layers (with pooling) + 3 fully connected layers
  - Input size: 32x32x1
  - Convolution kernel size: 5x5
  - Pooling: 2x2

LeNet-5



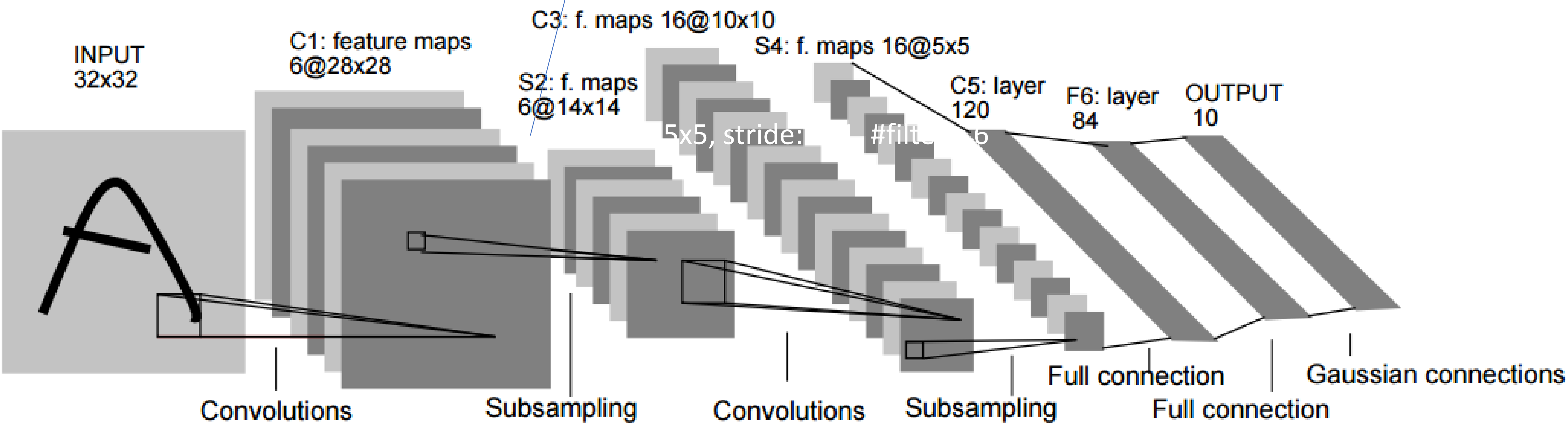
INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

LeNet-5


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

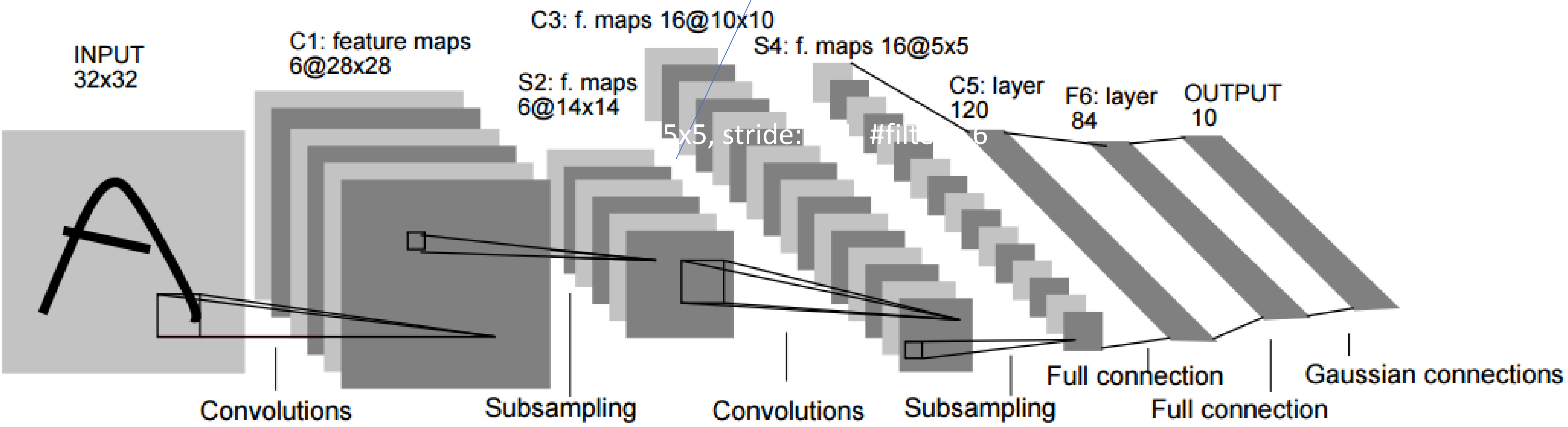
LeNet-5

Pooling: 2x2, stride: 2



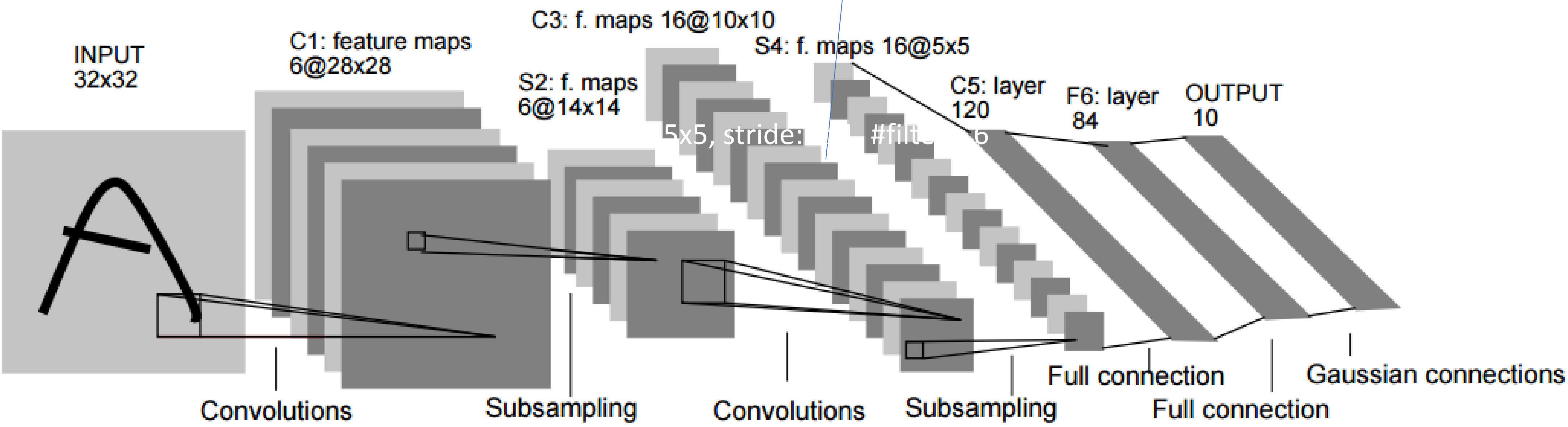
LeNet-5

Filter: 5x5x6, stride: 1x1,
#filters: 16



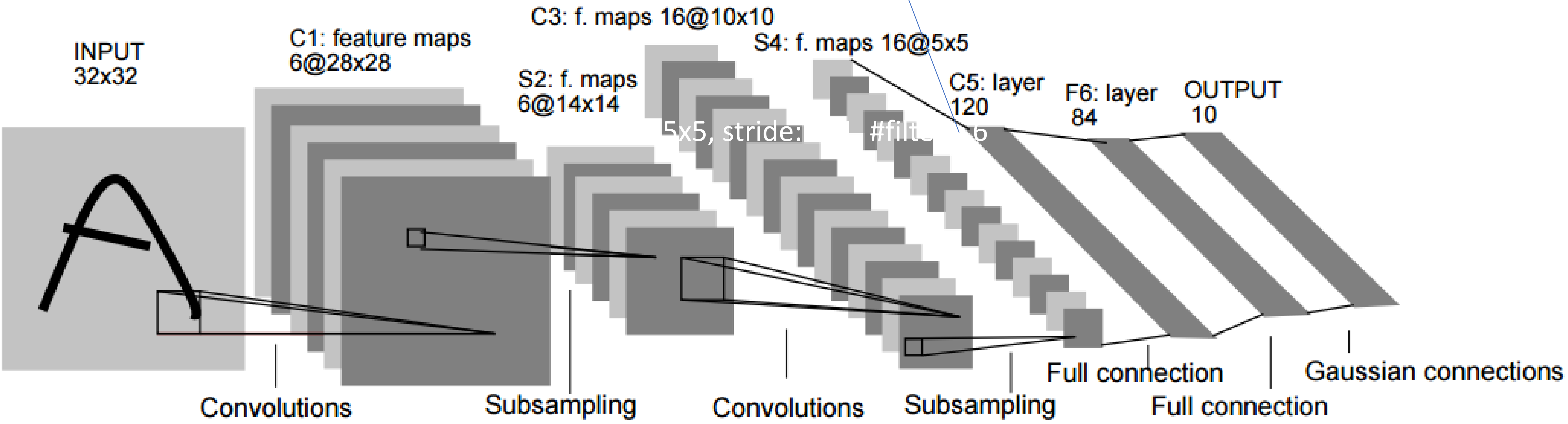
LeNet-5

Pooling: 2x2, stride: 2



LeNet-5

Weight matrix: 400x120



LeNet-5

Weight matrix: 120x84 Weight matrix: 84x10