Jadwal Minggu ke-5 dan 6 IF3270

Kamis 20 Maret 2025: Kuis 1 (offline, closed book/notes/laptop)

Untuk minggu ke-6, jika tidak ada pengumuman dari ITB, maka:
Senin 24 Maret 2025: Kuliah (online)
Kamis 27 Maret 2025: Kuliah (online)

IF3270 Pembelajaran Mesin

Convolutional Neural Network

Tim Pengajar IF3270

Review

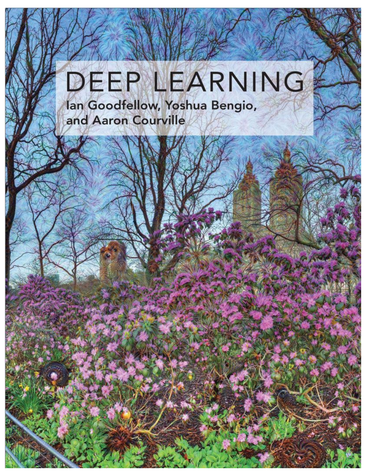
- Machine learning (ML) overview

- Ensemble Methods Supervised Learning

- Perceptron Supervised Learning
- Supervised Learning: ANN: Feed Forward Neural Network

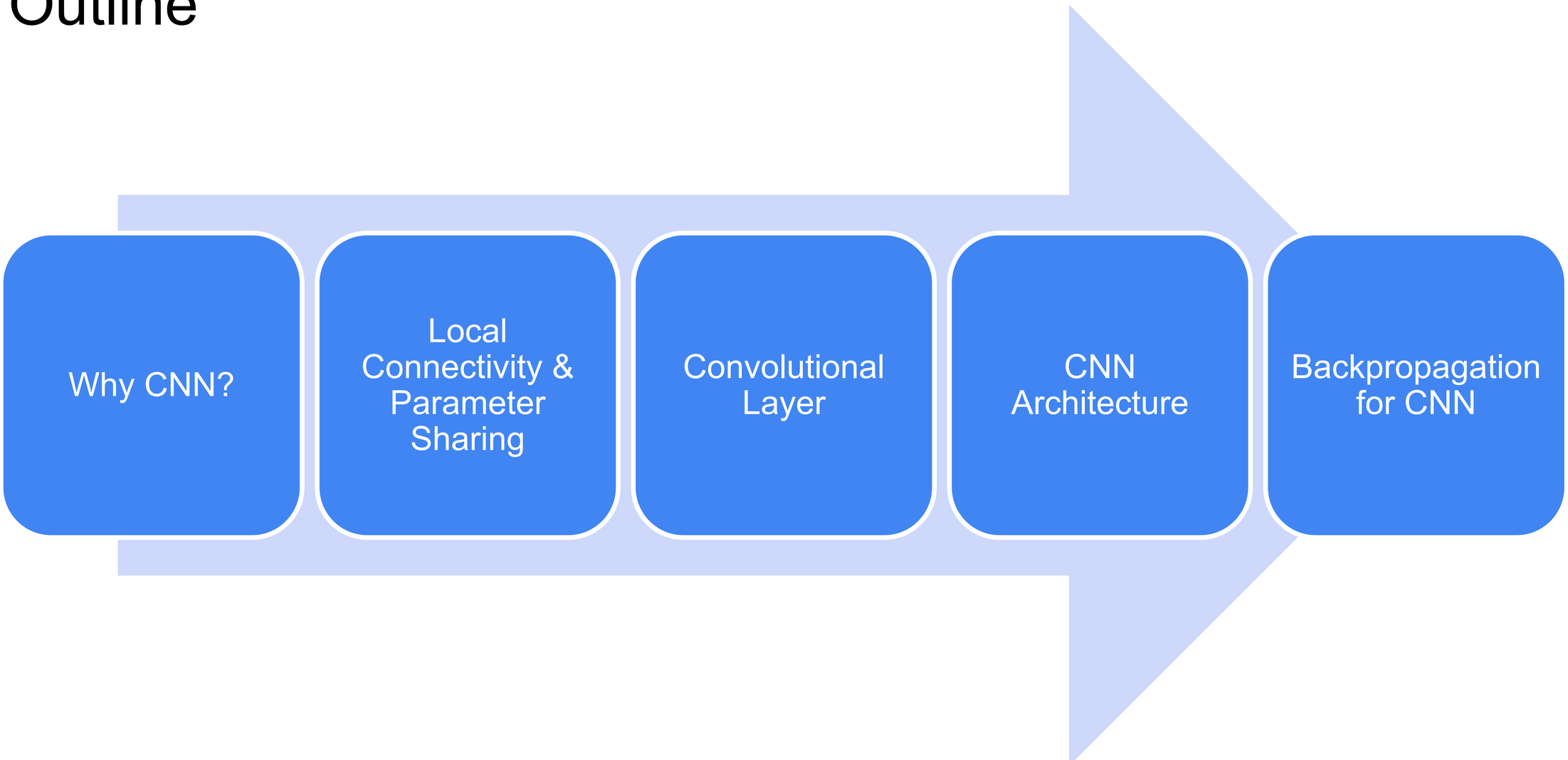
- Today: ANN: Convolutional Neural Network

Reference



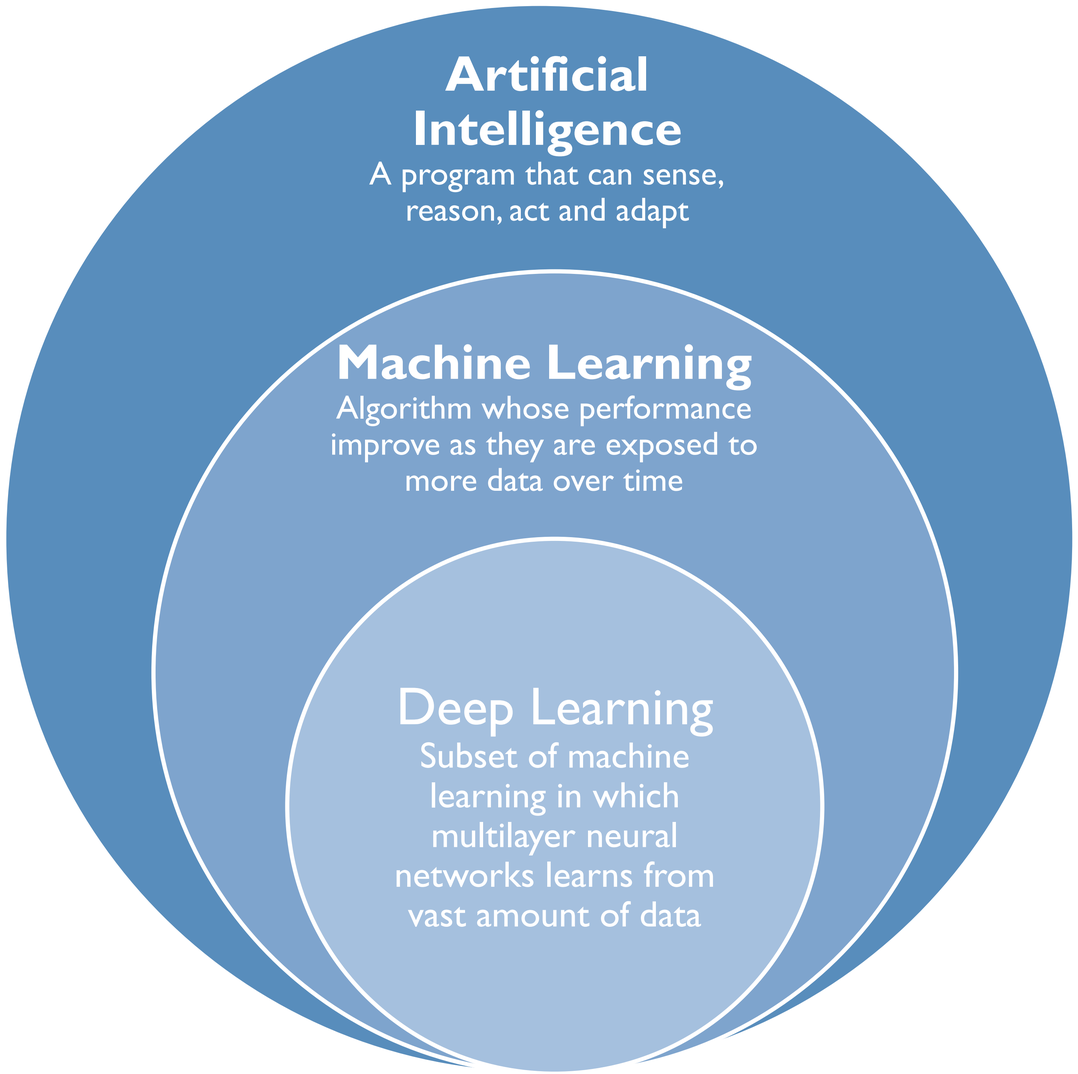
Deep learning. I Goodfellow, Y Bengio, A Courville, Y Bengio. MIT press 1 (2), 2016

Outline


Why CNN?
Local Connectivity & Parameter Sharing
Convolutional Layer
CNN Architecture
Backpropagation for CNN

01 Why CNN?

Artificial Intelligence, Machine Learning and Deep Learning



Why Deep Learning Now?

Larger Datasets

- Image 1000kb/ picture
- Audio: 5000km/ song
- Video: 5,000,000/movie

Better Hardware

- Transistors density doubles every 18months.
- Cost per GB in 1995: $1000.00
- Cost per GB in 2017: $0.02

Smarter Algorithms

- Advances in algorithm innovation, including neural network, leading to better accuracy in training models

Curse of Dimensionality

- Number of possible distinct configuration increases as the number if variables increases

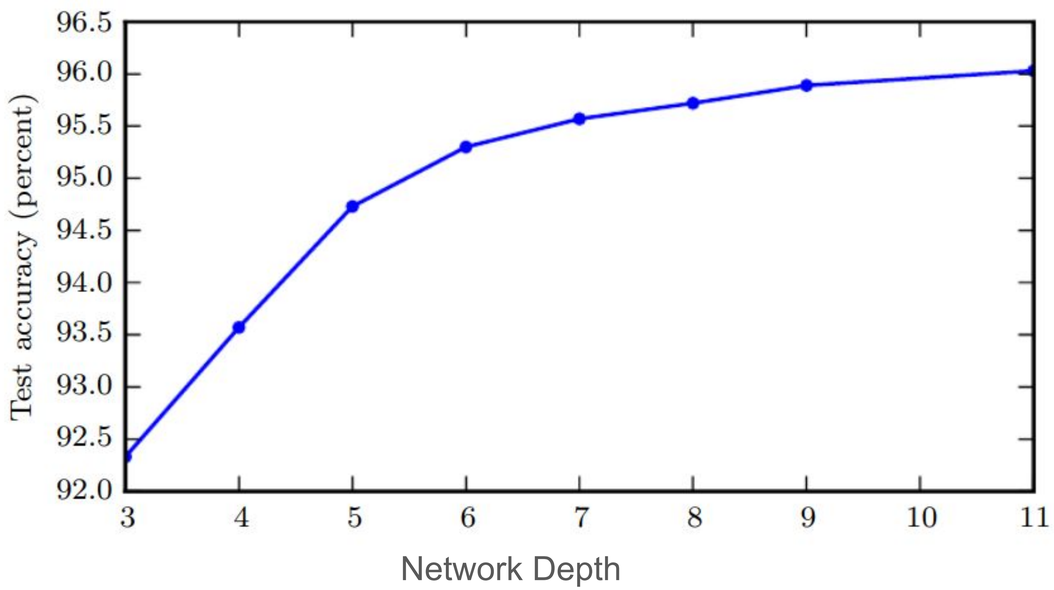
Source: Ian Goodfellow et al, Deep Learning, www.deeplearningbook.org

Universal Approximator Theorem

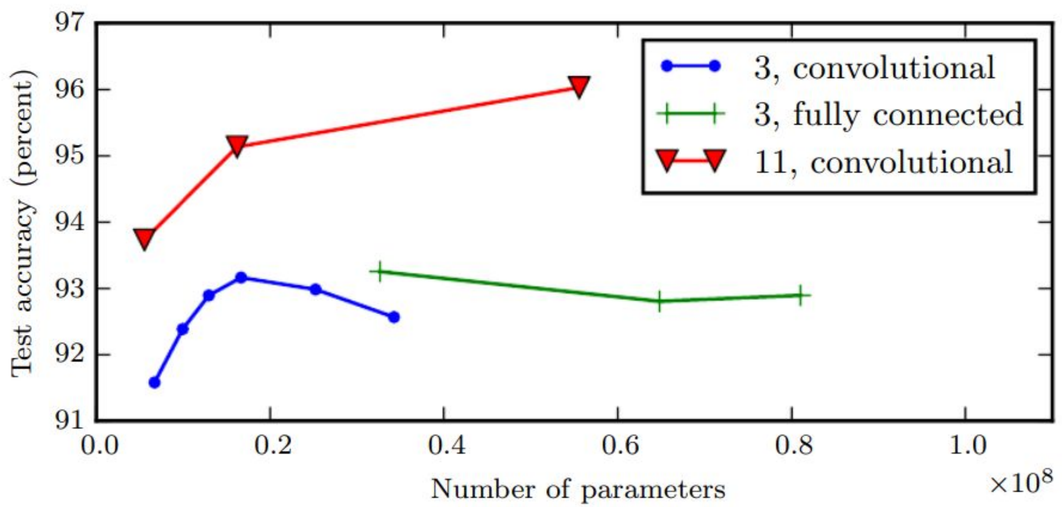
- One hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy
  - guarantees that a feedforward neural network with at least one hidden layer can represent an approximation of any function (within a broad class) to an arbitrary degree of accuracy, provided that it has enough hidden units.
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more

Deep vs Shallow Models

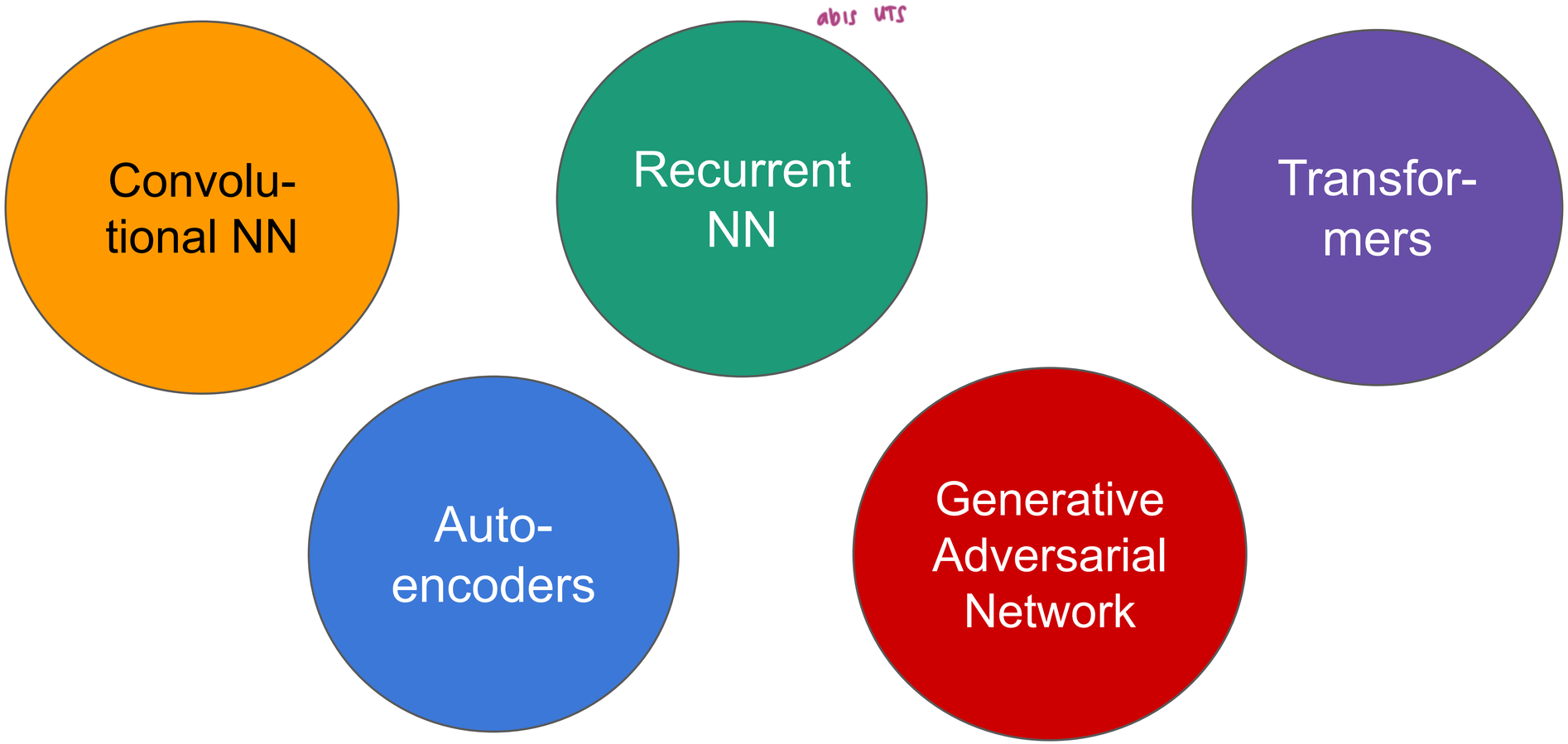
Better Generalization with Greater Depth



Large, Shallow Models Overfit More



Types of Deep Learning Models

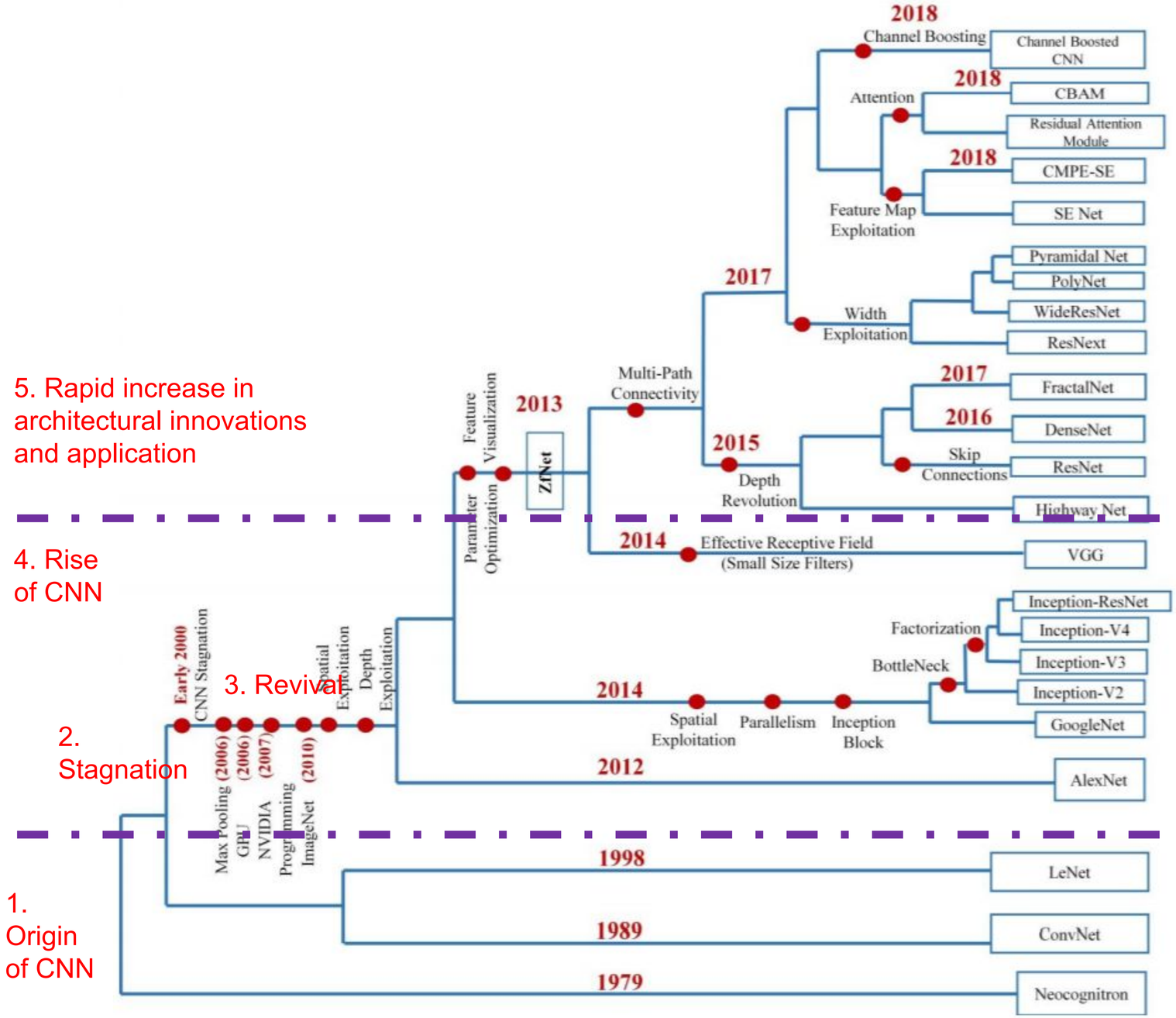


CNN History

First deep learning models to perform well

Most widely used algorithm among biologically inspired AI technique.

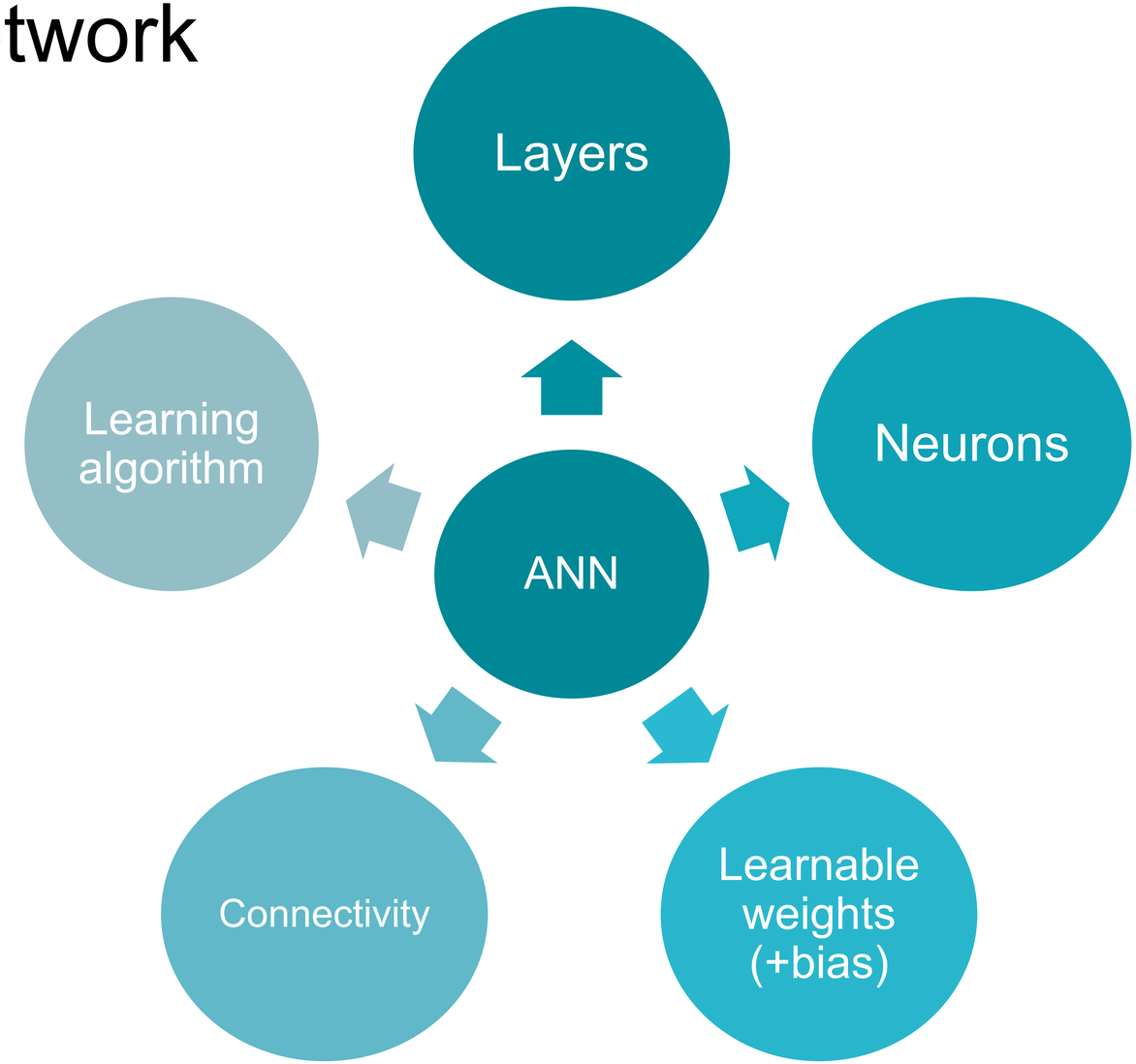
CNN architectures can be categorized into five different eras.



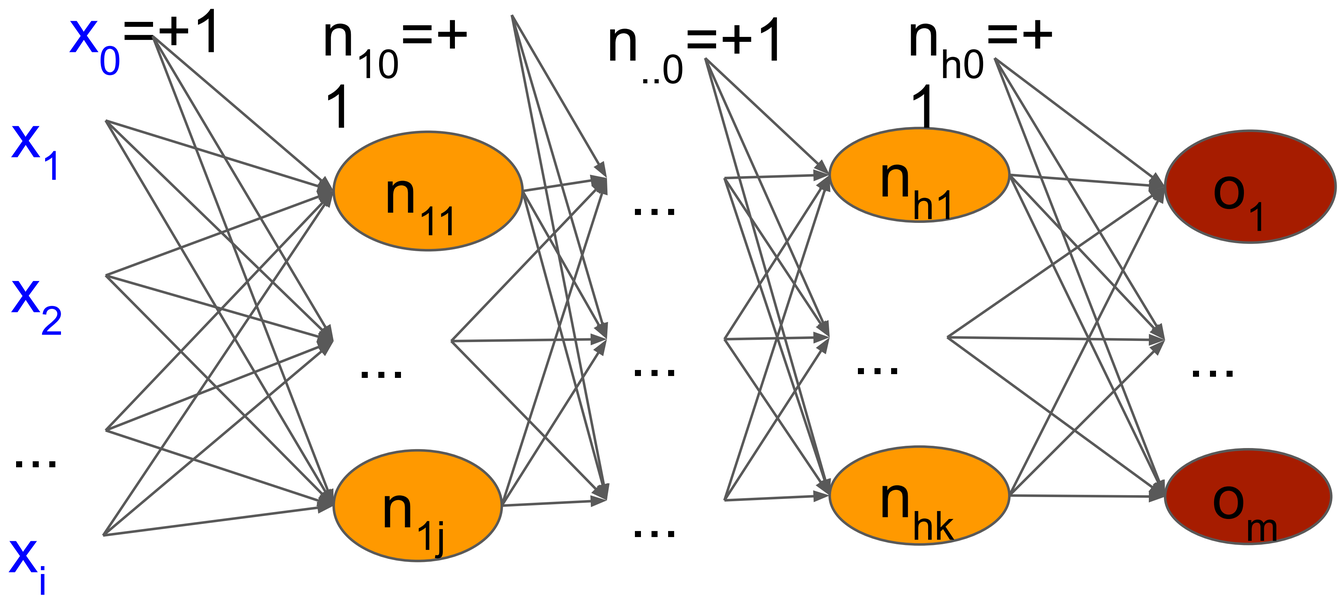
Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 1-62.

CNN is Artificial Neural Network

- Layers: input, hidden*, output
- Forward propagation
- Neurons have learnable weights and biases.
- Every unit connects with some units of previous layer.
- Learning algorithm (i.e.backpropagation)



Layers in Feed Forward Neural Network



Input layer has i input neurons

H hidden layer
Hidden layer 1: j neurons
Hidden layer h: k neurons

Output layer has m neurons

Convolutional Neural Network (CNN) : What

↳ CNN adalah ANN yg memiliki minimal 1 convolutional layer

Convolutional networks (ConvNet)

- Neural network that use convolution in place of general matrix multiplication in at least one of their layers.
- Inputs are data points with grid-like topology. (Explicit assumption that inputs are images)

matriks

Convolution Operation

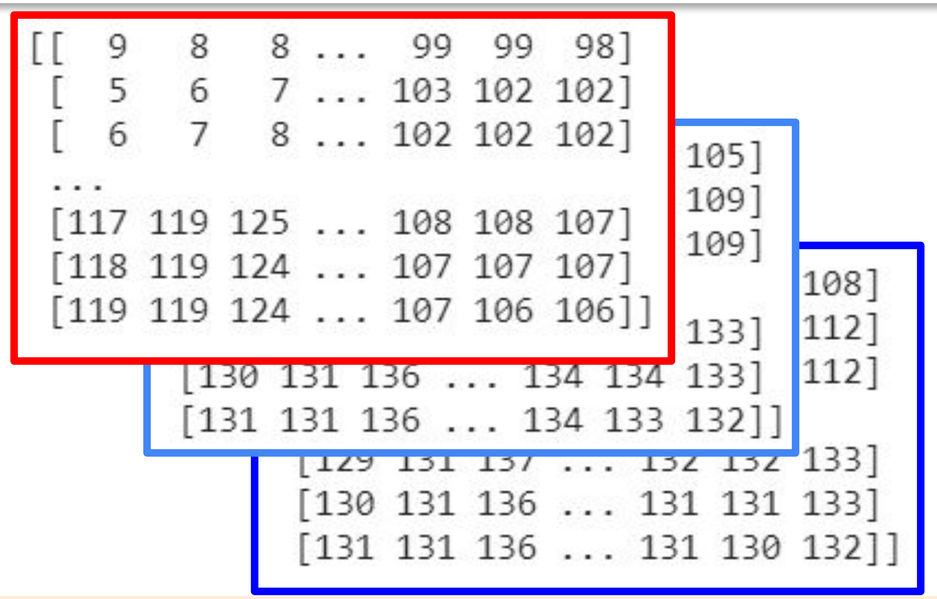
- Convolution: input, kernel feature map
  - Input: multidimensional array of data
  - Kernel: multidimensional array of parameters (matriks)
  - Kernels = filters = weights = parameters

Human Vision vs Computer Vision

What we see

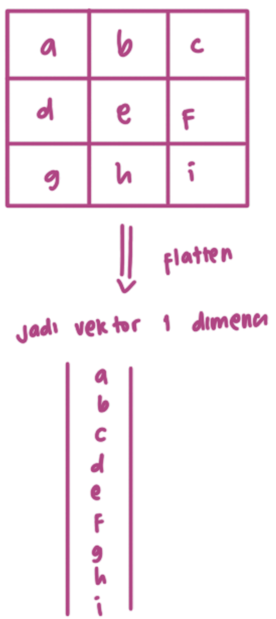
When we look at a picture of a cat, we can classify it as such if the picture has identifiable features such as paws or 4 legs.

What computer see



Computers sees an input image as array of pixels. Based on the image resolution, it will see height x width x dimension. Here is 451 x 731 x 3 array of matrix of RGB.

Challenges with ANN for Image Classification Task



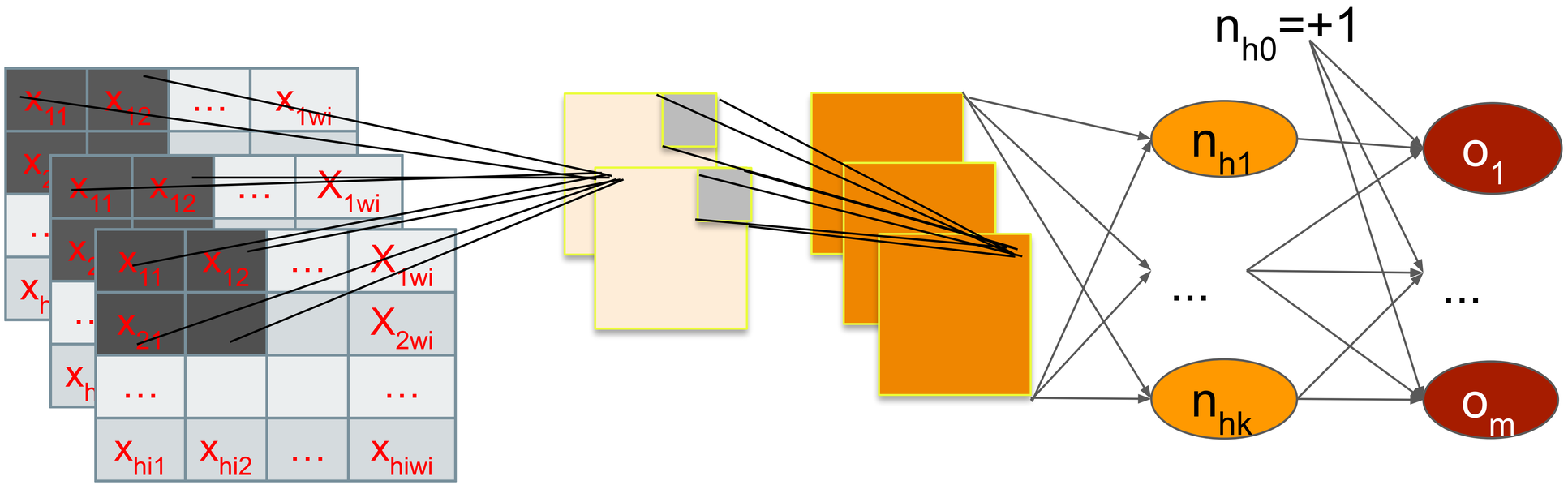
Flatten n-dim data into 1-dim vector

Number of trainable parameters ~ size of the image

ANN loses the spatial features of an image

https://courses.analyticsvidhya.com/courses/take/convolutional-neural-networks-cnn-from-scratch/texts/10846399-why-should-you-use-a-cnn

Layers in Convolutional Neural Network

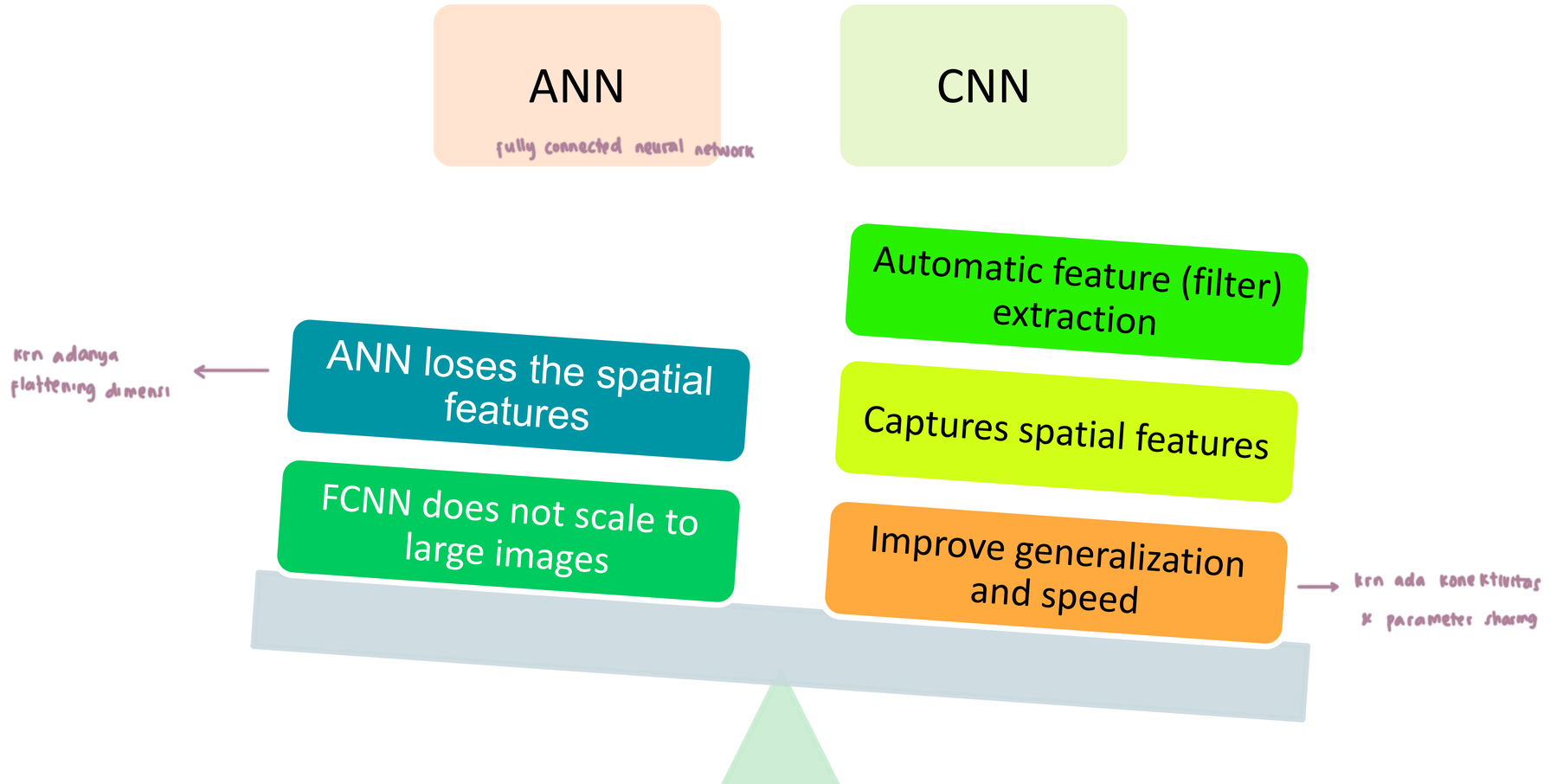


Input layer has hi*wi*di neurons

H hidden layer
Hidden layer 1,..: h*w*d neurons
Hidden layer h: k neurons

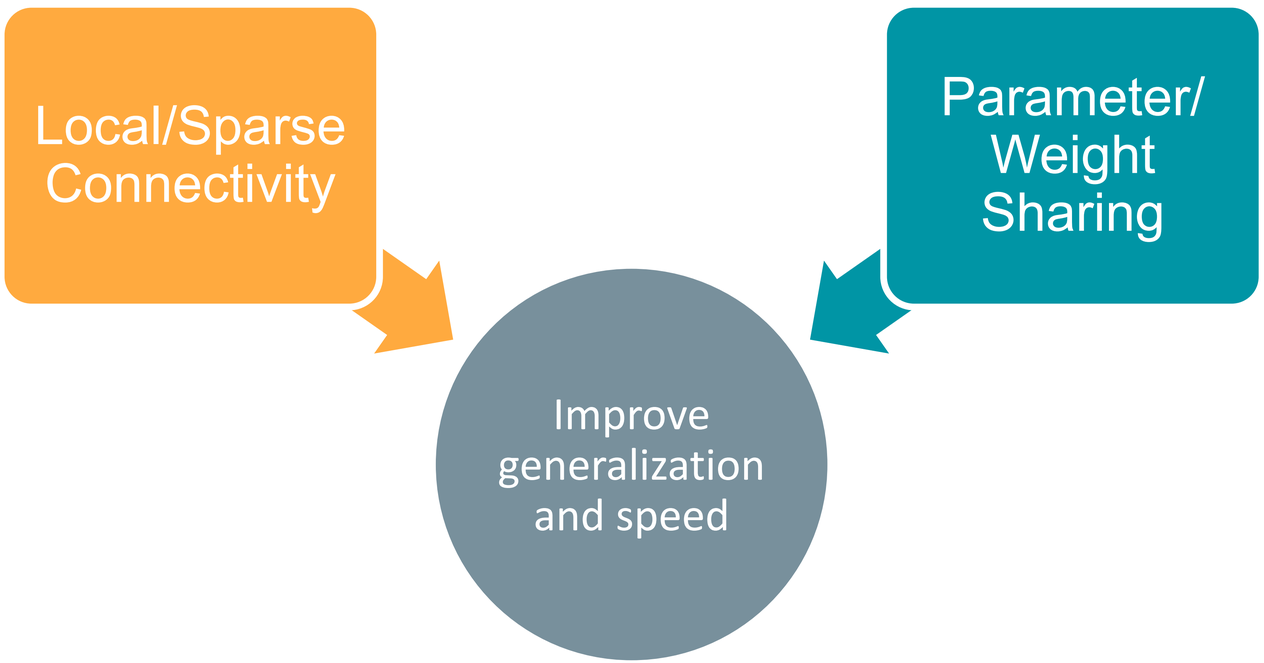
Output layer has m neuron

Convolutional Neural Network (CNN) : Why



02 Local Connectivity & Parameter Sharing

CNN Improve Generalization and Speed



	Acc	Weights
FFNN	87.0%	3240
Locally-connected	88.5%	1226
Local+weight sharing	98.4%	1060

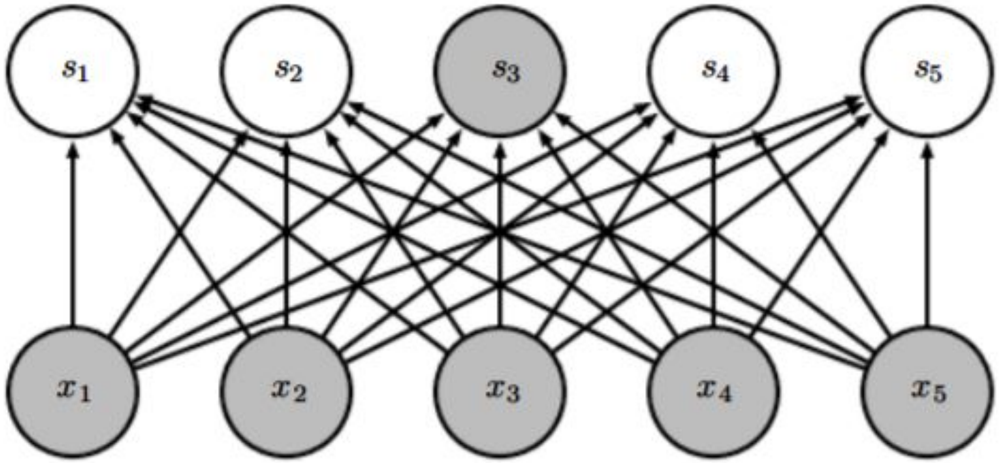
(LeCun, 1989)

Local/Sparse Connectivity/Weights

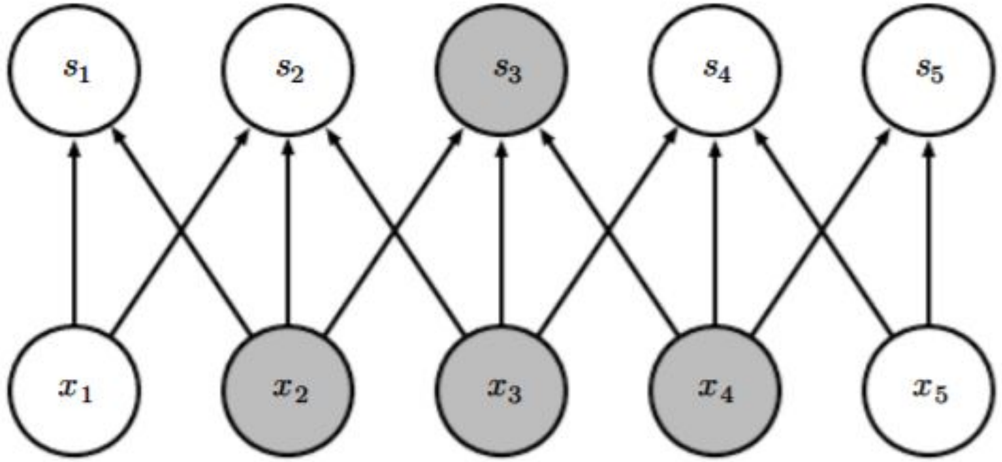
cara itung jumlah neuron : (input + 1) * output

contoh:
output (2) c
↑
hidden (5) b
↑
input (5) a

total = (5 + 1) * 5 + (5+1) * 2 = 42
a b b c



In traditional neural network, every output unit interacts with every input unit. There is a separate parameter describing the interaction between each input unit and each output unit. All of input units affect s3.



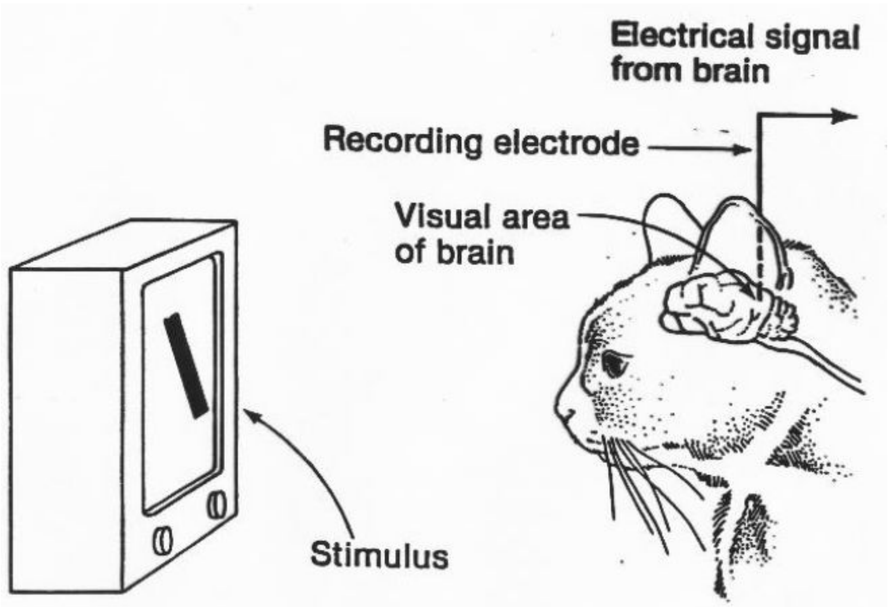
All input units in x that affect output unit s3 are known as the receptive field of s3. When s is formed by convolution with a kernel of width 3, only three inputs affect s3.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Receptive Field: Biological Visual Cortex

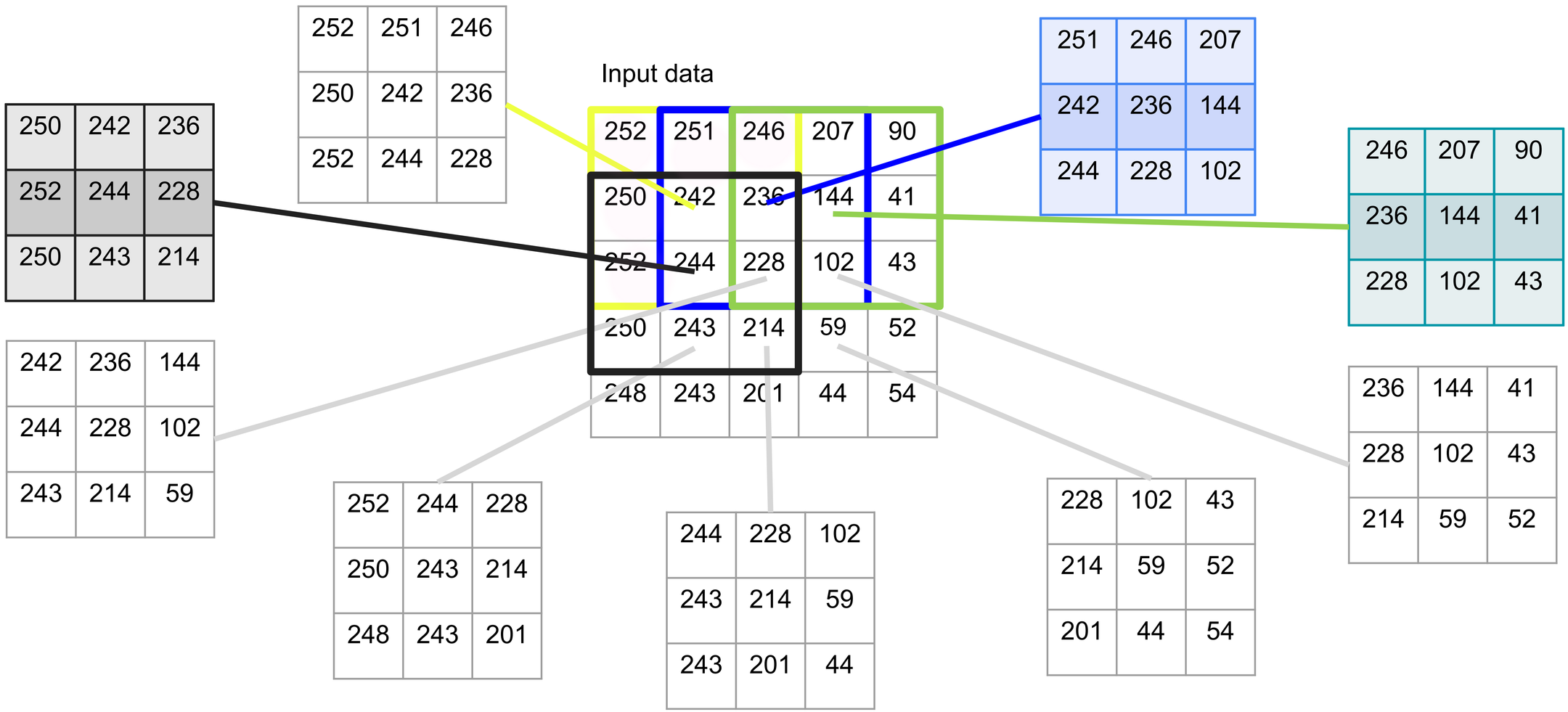
Small regions of cells are sensitive to specific regions of visual field.

Hubel and Wiesel (1959) found out that all of these neurons were organized in a columnar architecture and that together, they were able to produce visual perception.

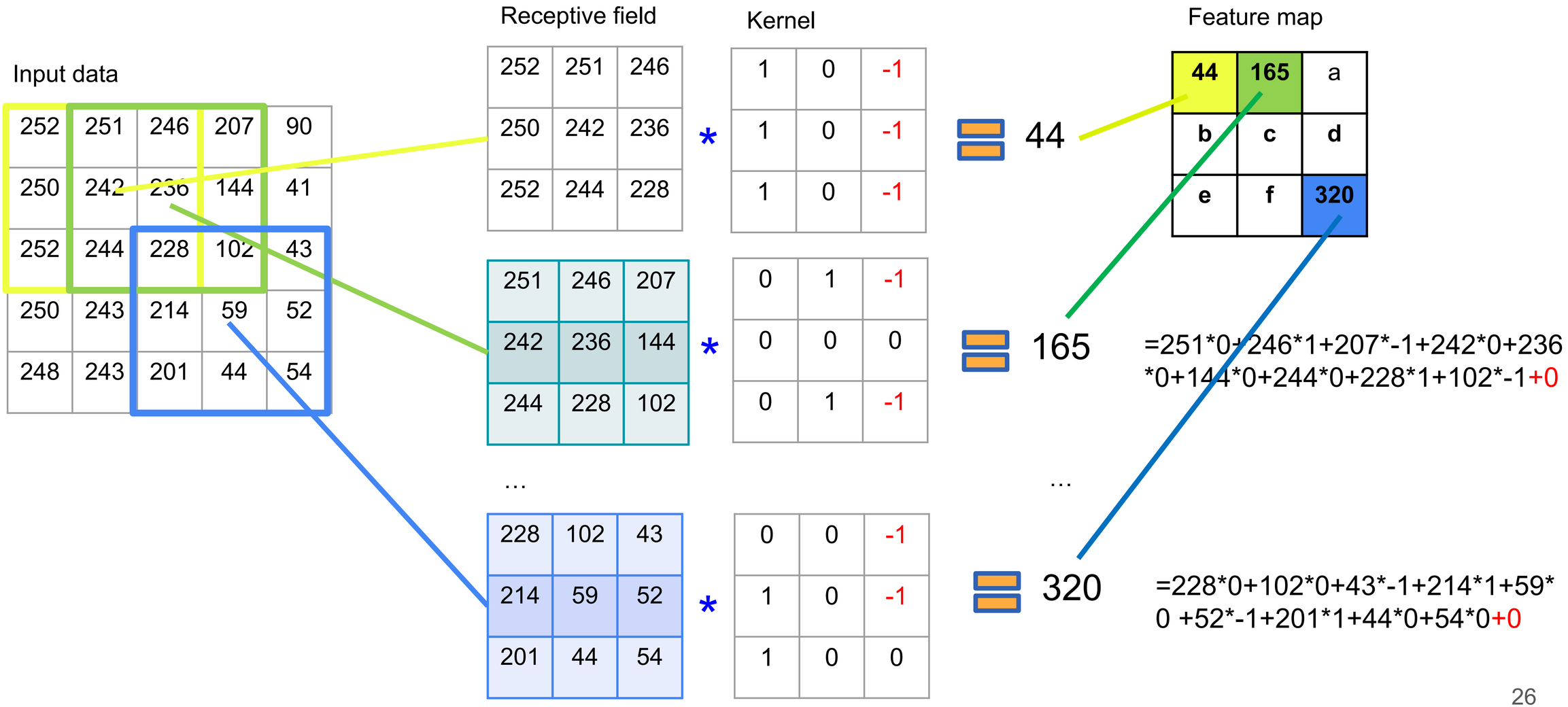


Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, *148*(3), 574.
Figure:
https://www.esantus.com/blog/2019/1/31/convolutional-neural-networks-a-quick-guide-for-newbies

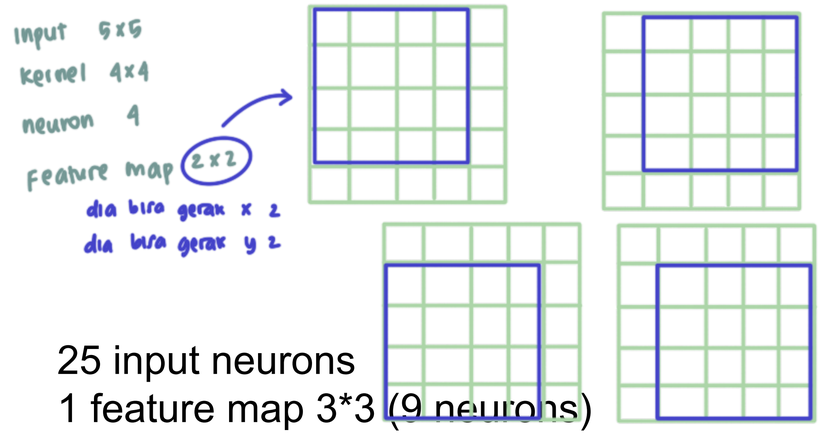
Receptive Field for Grid-like Data with Kernel 3*3



Convolution Operation without Parameter Sharing



Locally Connected Layer (No Sharing)



25 input neurons
1 feature map 3*3 (9 neurons)
Number of weight:
9*(3*3*1+1)=90

bias

nb Chanel = 1

Kernel

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

Input data

252	251	246	207	90
250	242	236	144	41
252	244	228	102	43
250	243	214	59	52
248	243	201	44	54

…

0	0	-1
1	0	-1
1	0	0

Feature map

44	**165**	a
b	**c**	**d**
e	**f**	**320**

Number of neurons (conv layer)
= *outputHeight * outputWidth * nbFeatureMaps*

Number of weights without weight sharing (conv layer):
= *nbofNeurons * ((filterLength * filterWidth * nbChannel) + 1 bias)*

Locally Connected Architecture (LeCun, 1989)

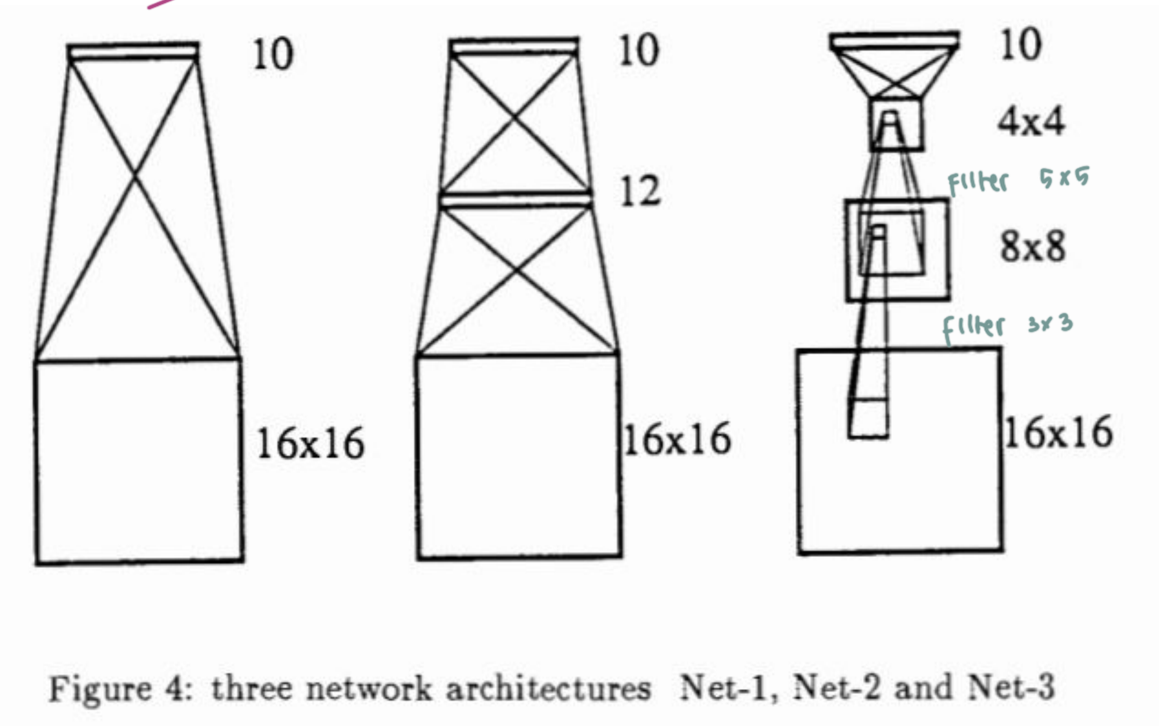


Figure 4: three network architectures Net-1, Net-2 and Net-3

10 (16x16x1 + 1)

Net-1: 2570 weights, (256+10) neurons

12 (10 (16x16x1 + 1))

Net-2: 3214 weights, (256+12+10) neurons

Net-3: 1226 weights, (256+64+16+10) neurons
Filter 1: 3*3; Filter 2: 5*5
Weights: 8*8*(3*3+1) + 4*4*(5*5+1) + (16+1)*10
= 640+416+170 =1226

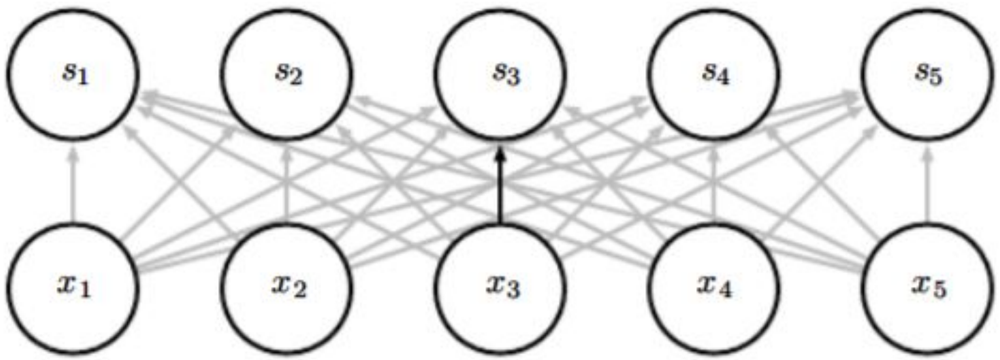
Number of neurons (conv layer)
= *outputHeight * outputWidth * nbFeatureMaps*

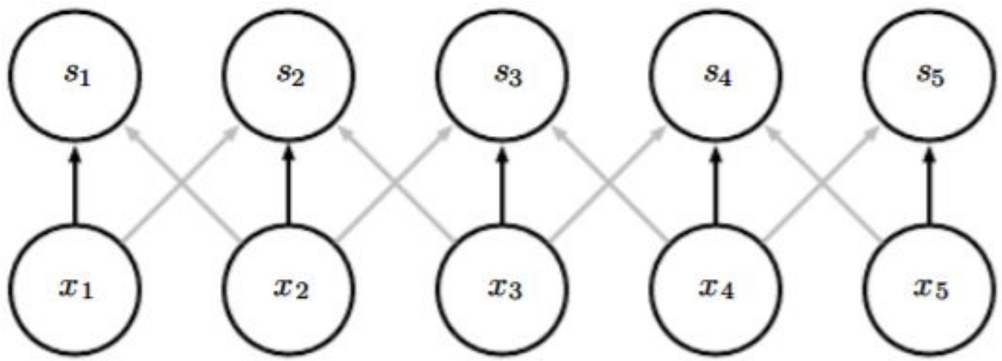
Number of weights without weight sharing (conv layer):
= *nbofNeurons * ((filterLength * filterWidth * nbChannel) + 1 bias)*

$$V = \frac{W - F + 2P}{S} + 1$$

Figure: LeCun, Y. (1989). Generalization and network design strategies. *Connectionism in perspective*, *19*, 143-155.
Formula:
https://towardsdatascience.com/understanding-parameter-sharing-or-weights-replication-within-convolutional-neural-networks-cc26db7b645a

Parameter Sharing



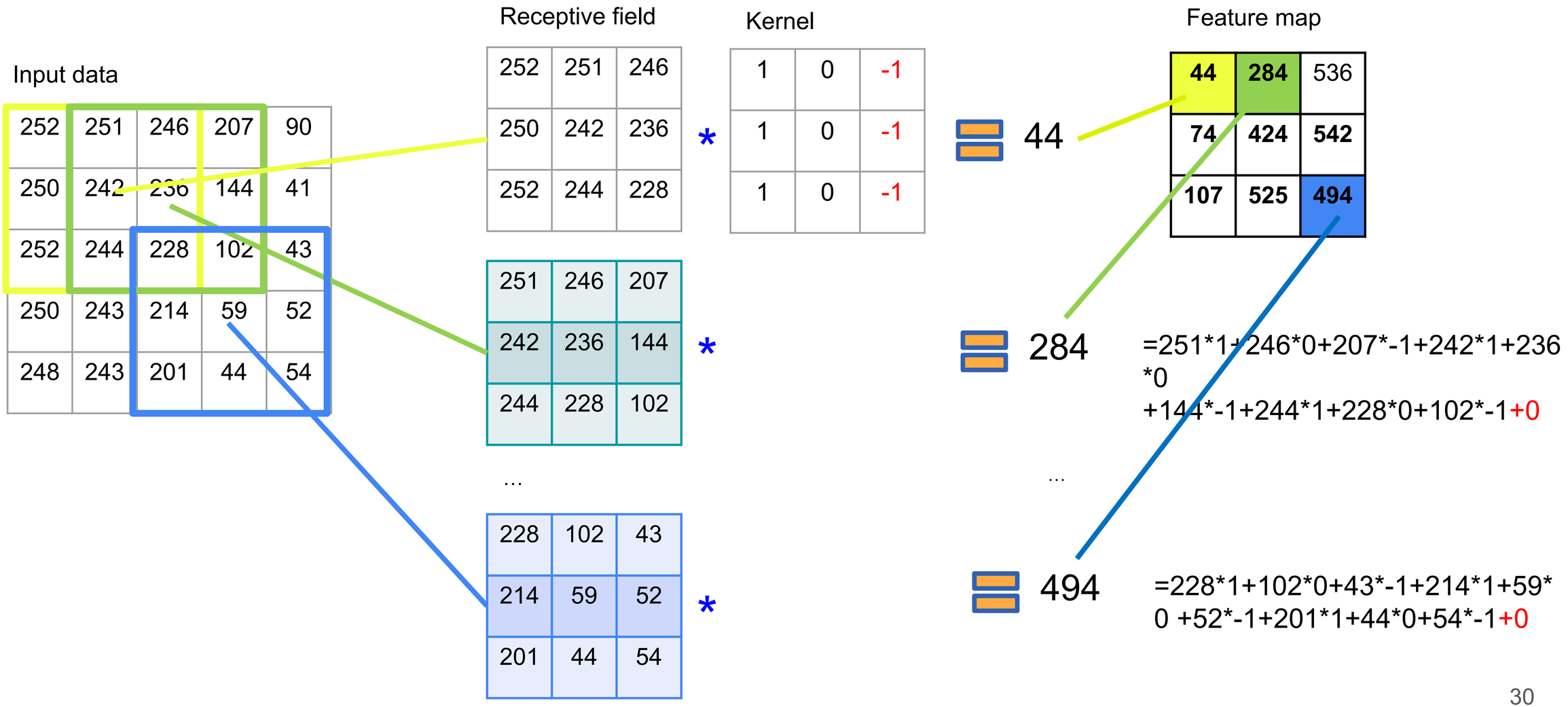


In traditional neural network, if there are m inputs and n outputs, there are (m+1)*n parameters, including bias.

Convolution shares the same parameters across all spatial locations. If kernel width is k for m inputs and n outputs, then there are k*1 parameters.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Convolution Operation with Parameter Sharing



Input Image

252	251	246	207	90
250	242	236	144	41
252	244	228	102	43
250	243	214	59	52
248	243	201	44	54

Feature map

Kernel

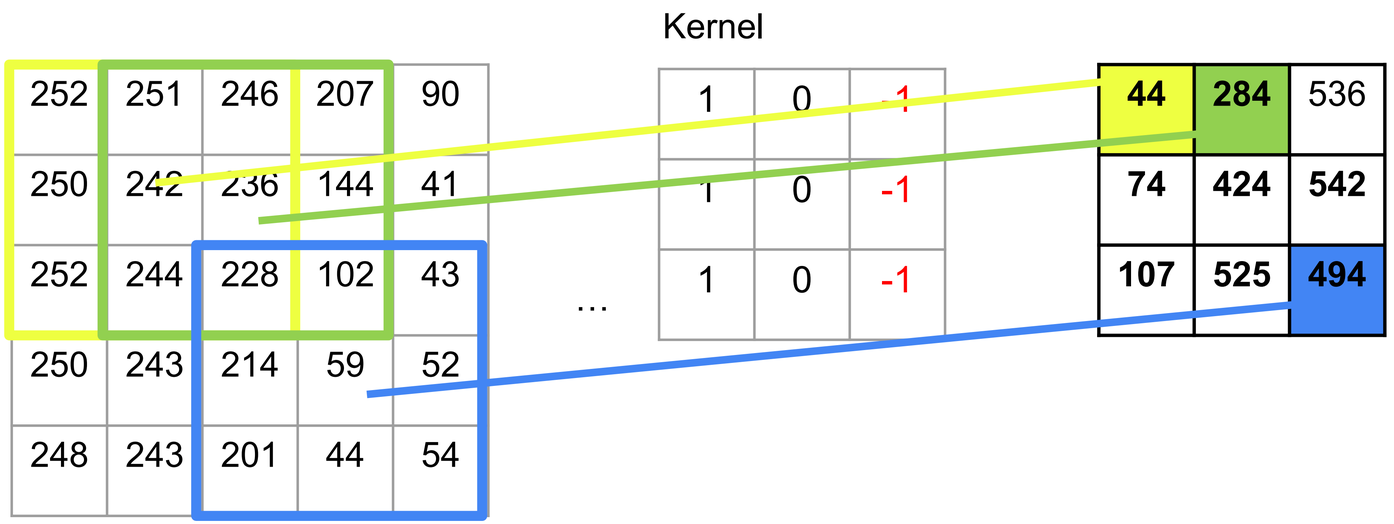
1	0	-1
1	0	-1
1	0	-1

Receptive field

X

=

Convolution Layer with Local Connectivity + Parameter Sharing

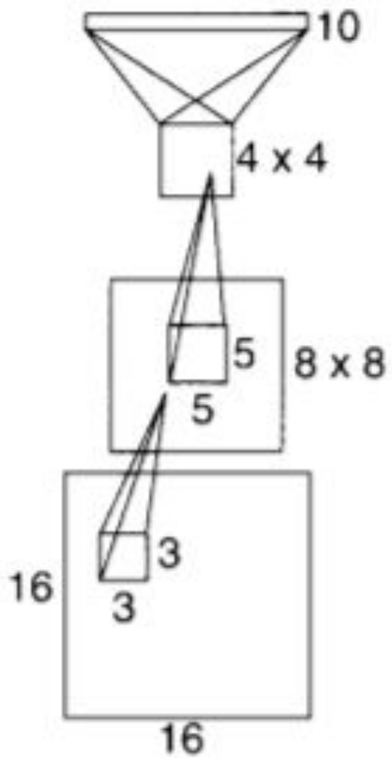


25 input neurons
1 feature map 3*3 (9 neurons)
Number of weight:
1*(3*3*1+1)=10 (with sharing)
Number of weight:
9*(3*3*1+1)=90 (without sharing)

Number of weights without weight sharing (conv layer):
*= nbofNeurons * ((filterLength * filterWidth * nbChannel) + 1 bias)*

Number of weights with weight sharing:
*= nbFeatureMaps * ((filterLength * filterWidth * nbChannel) + 1 bias)*

Parameter Sharing: Example Net3



Net3: 1226 weights, (256+64+16+10) neurons

Weights (without sharing):
64*(3*3*1+1) + 16*(5*5*1+1) + (16+1)*10 =1226

Weights (with sharing):
1*(3*3*1+1) + 1*(5*5*1+1) + (16+1)*10 =10+26+170=206

Number of weights without weight sharing (conv layer):
= *nbofNeurons * ((filterLength * filterWidth * nbChannel) + 1 bias)*

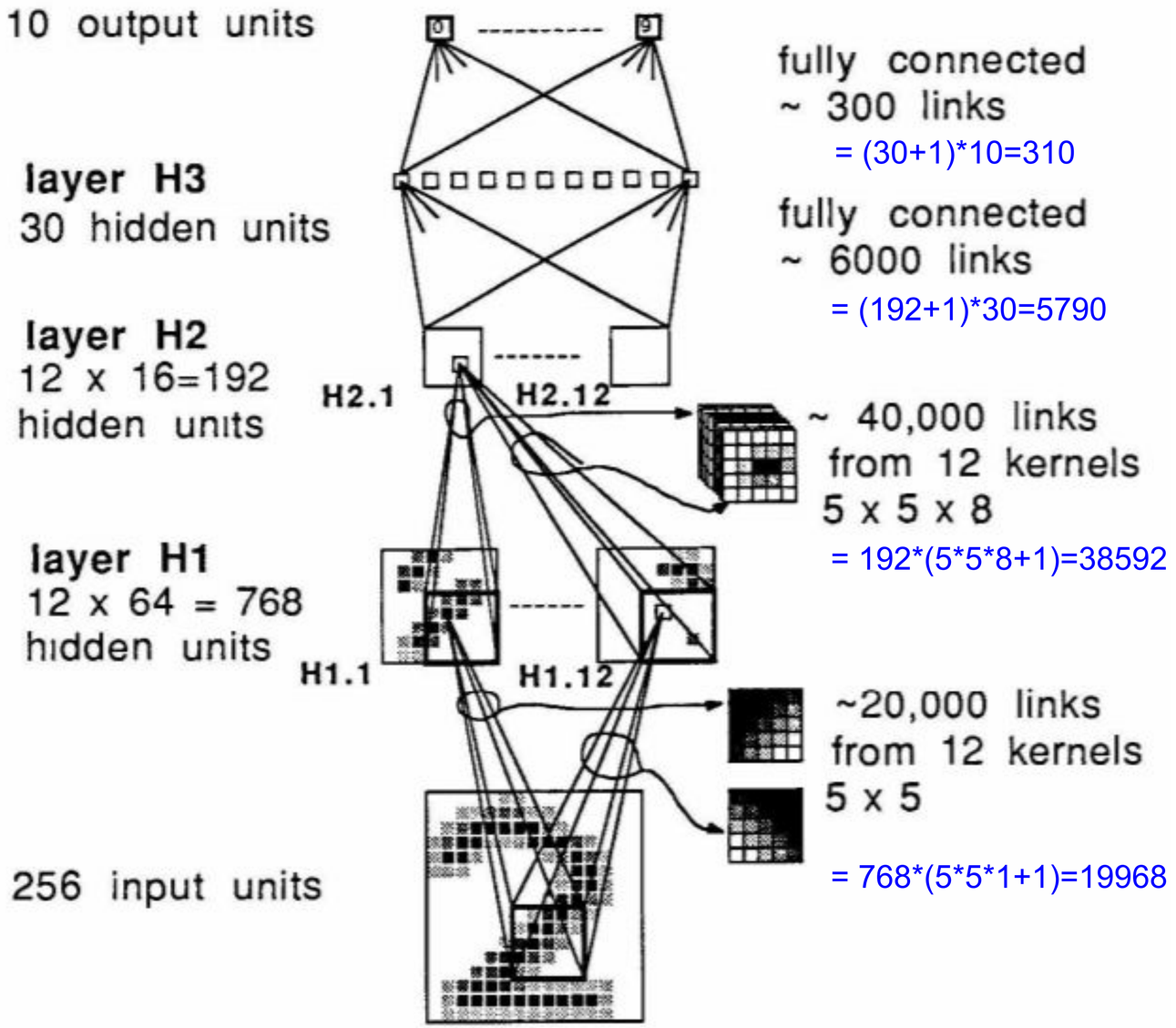
Number of weights with weight sharing:
= *nbFeatureMaps * ((filterLength * filterWidth * nbChannel) + 1 bias)*

Figure: Drucker, H., & Le Cun, Y. (1992). Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, *3*(6), 991-997.
Formula:
https://towardsdatascience.com/understanding-parameter-sharing-or-weights-replication-within-convolutional-neural-networks-cc26db7b645a

Locally Connected Architecture (LeCun et al., 1989)

Weights (without sharing):
19.968+38.592+5.790+310=64.660

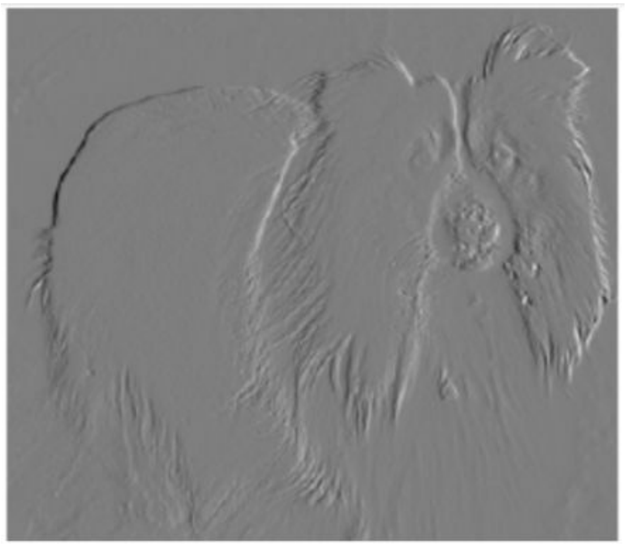
Weights (with sharing):
12*(5*5*1+1) + 12*(5*5*8+1) +5.790+310
=8824



LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541-551.

Convolution More Efficient Computationally

Edge detection:
raw data (320,280) □
feature map (319,280)



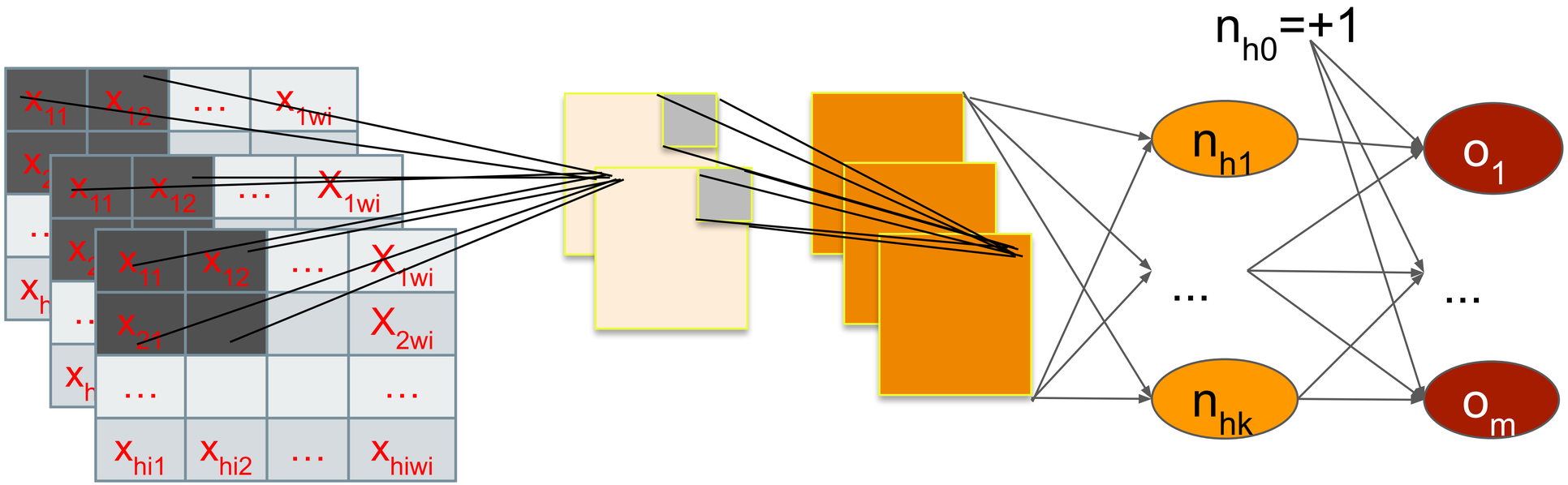
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

FFNN input-output layer requires 320*280*319*280 > 8e9 operations in matrix multiplication.

Convolution using kernel 1*2 <1,-1> requires 319*280*3 = 267.960 operations in matrix multiplication (roughly 60.000 times more efficient)

03 Convolutional Layer

Convolutional Neural Network

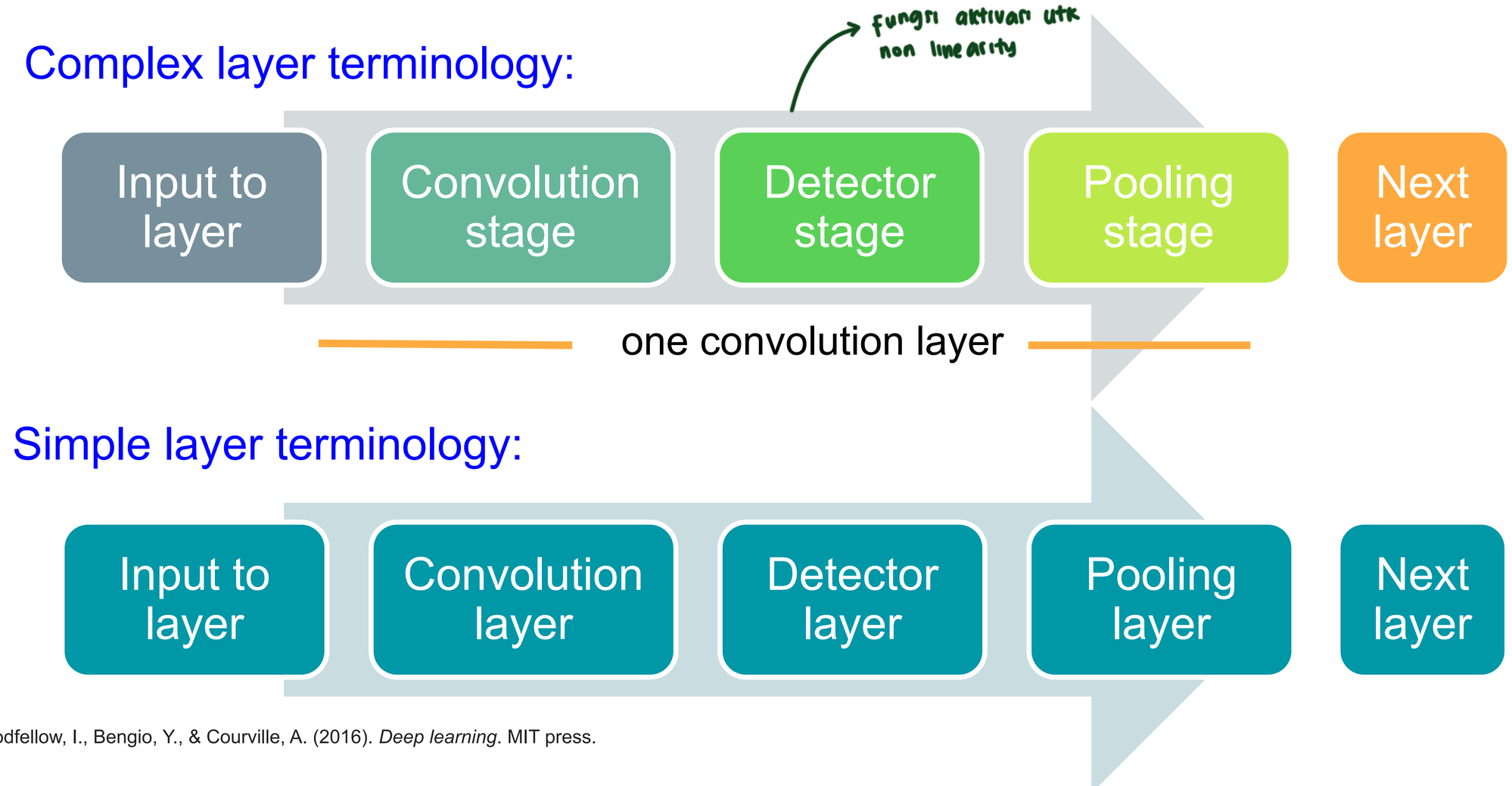


Input layer has hi*wi*di neurons

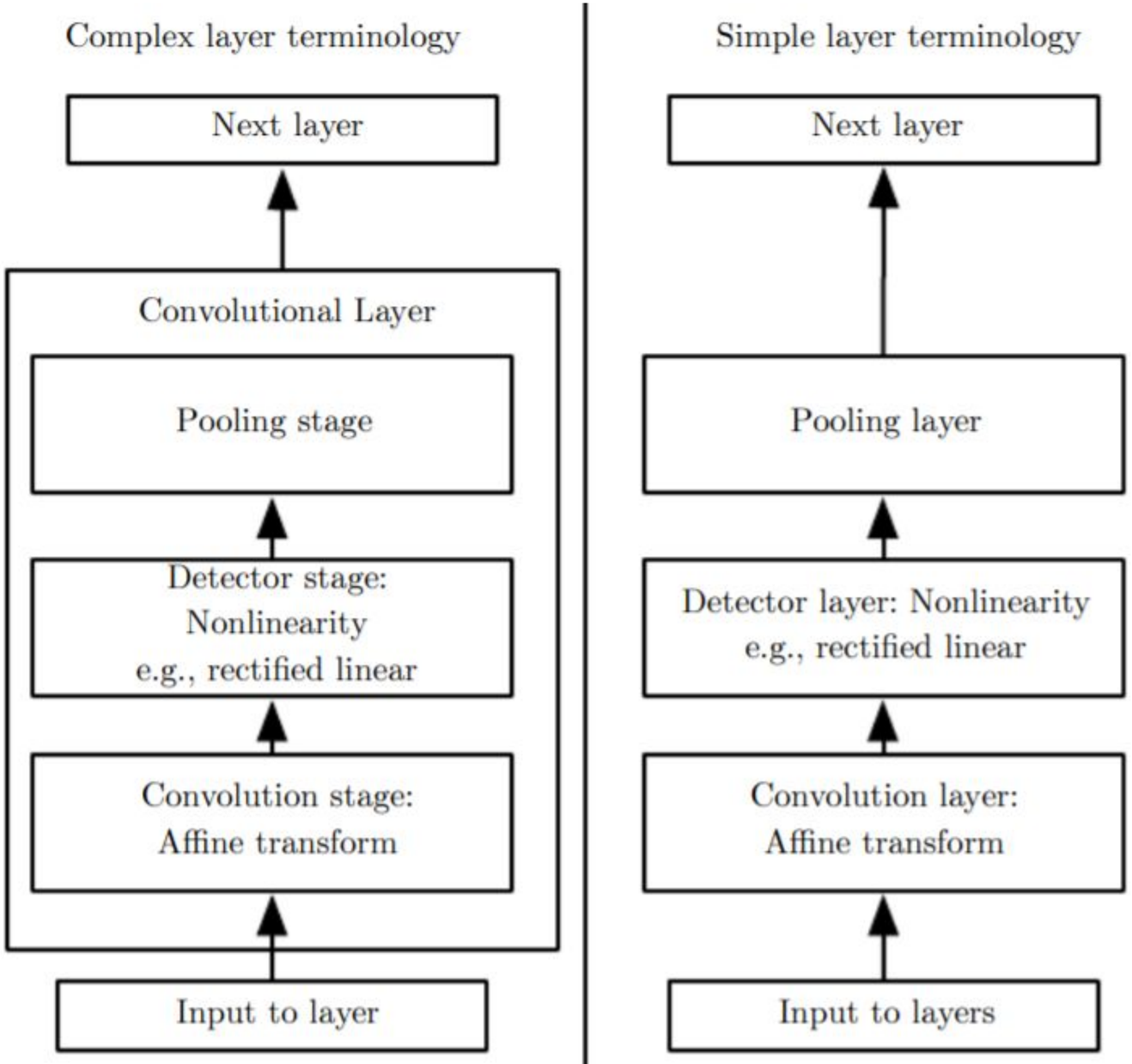
H hidden layer
Hidden layer 1,..: h*w*d neurons
Hidden layer h: k neurons

Output layer has m neuron

Convolutional Layer: Terminology



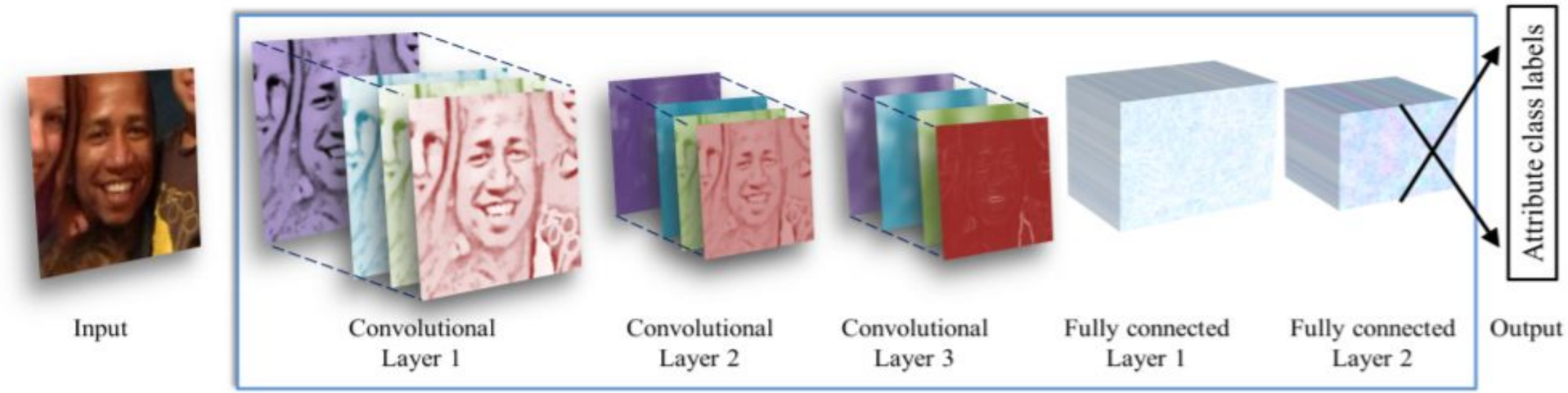
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.



Components of Typical CNN Layer

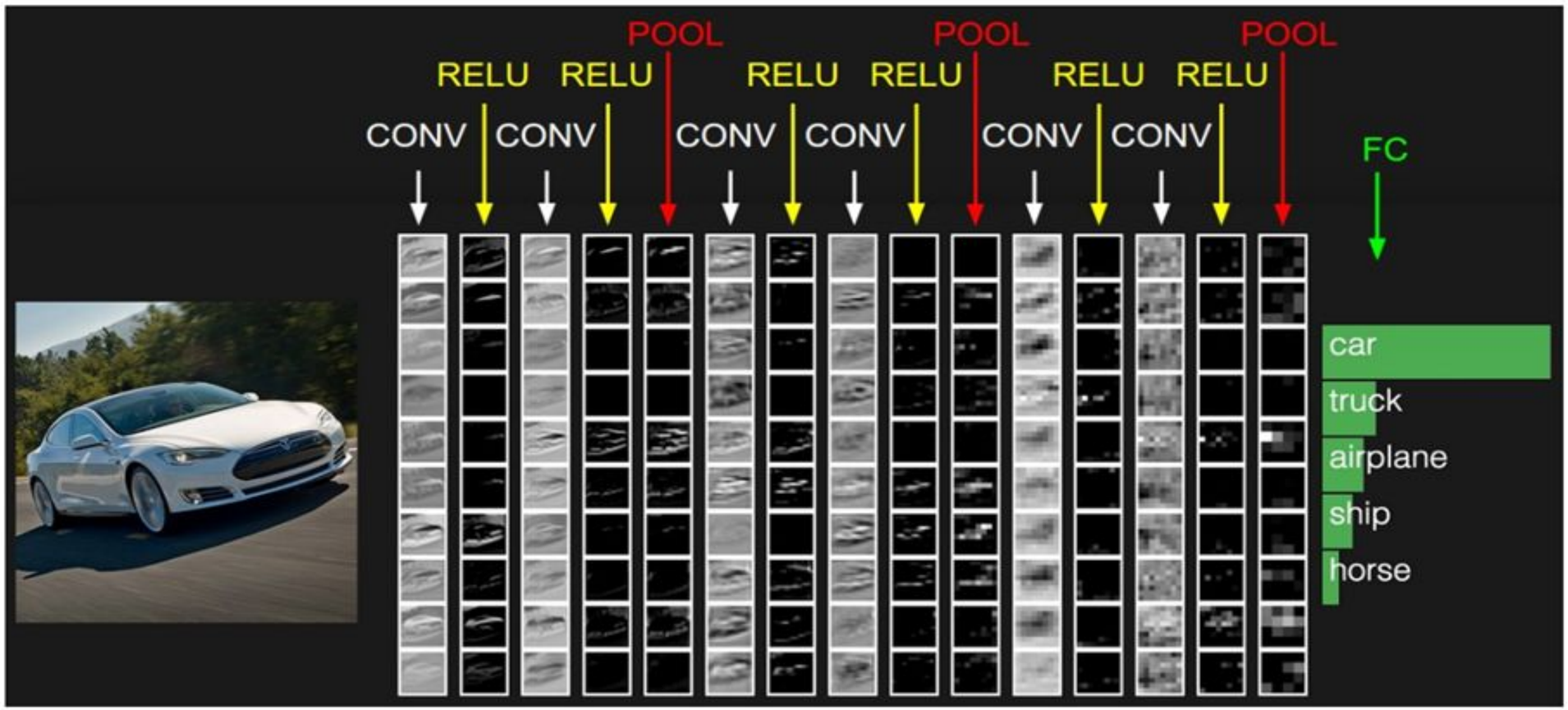
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

CNN Architecture: Example 1



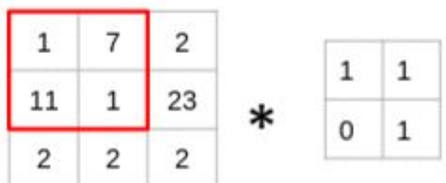
Levi, G., & Hassner, T. (2015). Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 34-42).

CNN Architecture: Example 2



https://cs231n.github.io/convolutional-networks/

Convolution Stage



Kernel moving across input such that all data are covered at least once.

(1x1 + 7x1 + 11x0 + 1x1) = 9

(7x1 + 2x1 + 1x0 + 23x1) = 32

(11x1 + 1x1 + 2x0 + 2x1) = 14

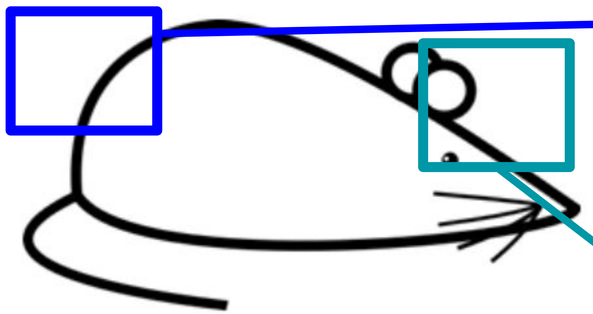
(1x1 + 23x1 + 2x0 + 2x1) = 26

https://courses.analyticsvidhya.com/courses/take/convolutional-neural-networks-cnn-from-scratch/texts/10846511-the-convolutional-layer

Convolution Stage: Feature Extraction

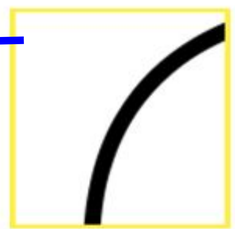
Kernel extracts certain features from input data, especially spatial features.

Input data



Original image

Visualization of the receptive field



0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

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

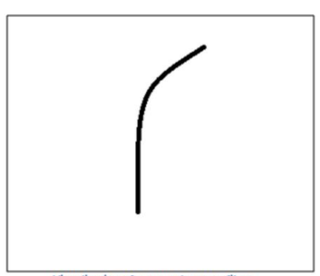
Pixel representation of filter

= 6600

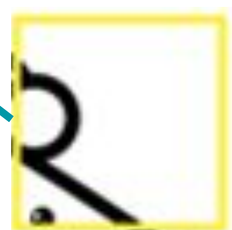
Kernel

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

Pixel representation of filter



Visualization of a curve detector filter



0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

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

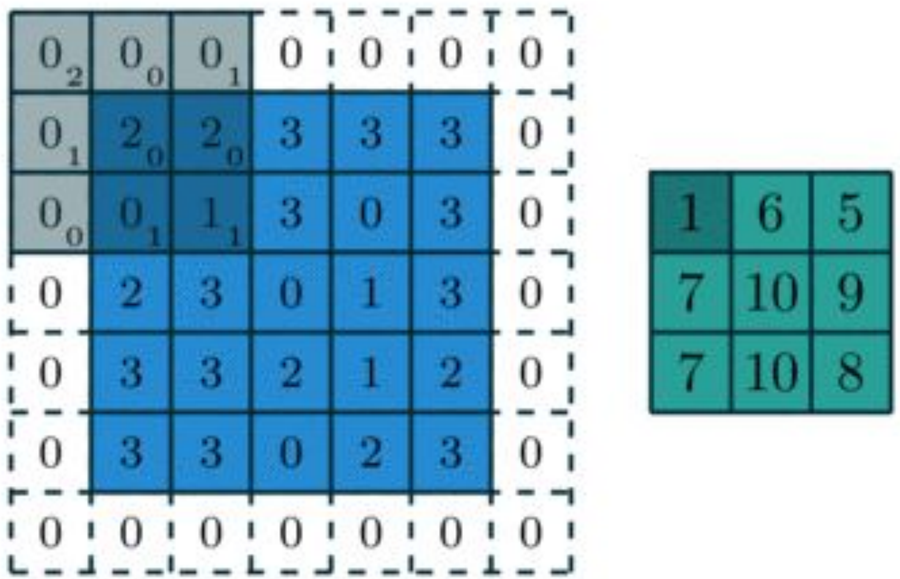
Pixel representation of filter

= 0

https://medium.com/@harsh21476/why-convolution-in-cnn-why-not-something-else-5f707c9e75f1

Convolution Stage: Stride & Padding

Example of stride of 2



1	6	5
7	10	9
7	10	8

$$V = \frac{5-3+2*1}{2} + 1 = 3$$

- Stride: number of pixel moving at a time.
  Stride value ~ reduction size of image.

- Padding: layer of zeros around input.
  It preserve the size of the input image to feature map size.
  If a single zero padding is added, a single stride filter movement would retain the size of the original image.

$$V = \frac{5-3+2*1}{1} + 1 = 5$$

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

Convolution Stage: Output Dimension

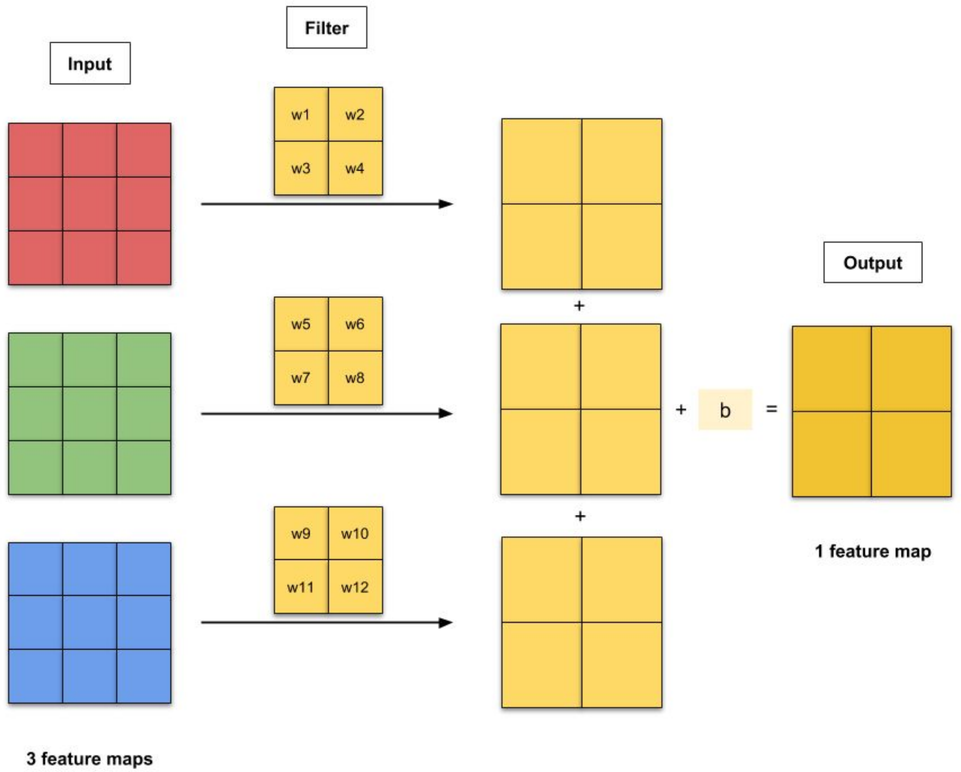
The spatial size of output: V*V*K

$$V = \frac{W-F+2P}{S} + 1$$

input size w*w*d	Number (K) and size of kernel (F*F*d)
Stride value (S)	Number of padding applied (P)

Example 1: Image size 3*3*3 (W=3); 1 filter 2*2*3
(K=1, F=2); moving stride S=1; input padding P=0:
V=([3-2+2*0]/1)+1=2
Output volume: 2*2*1

Convolution of RGB image with 2*2*1 filter to output 1 channel



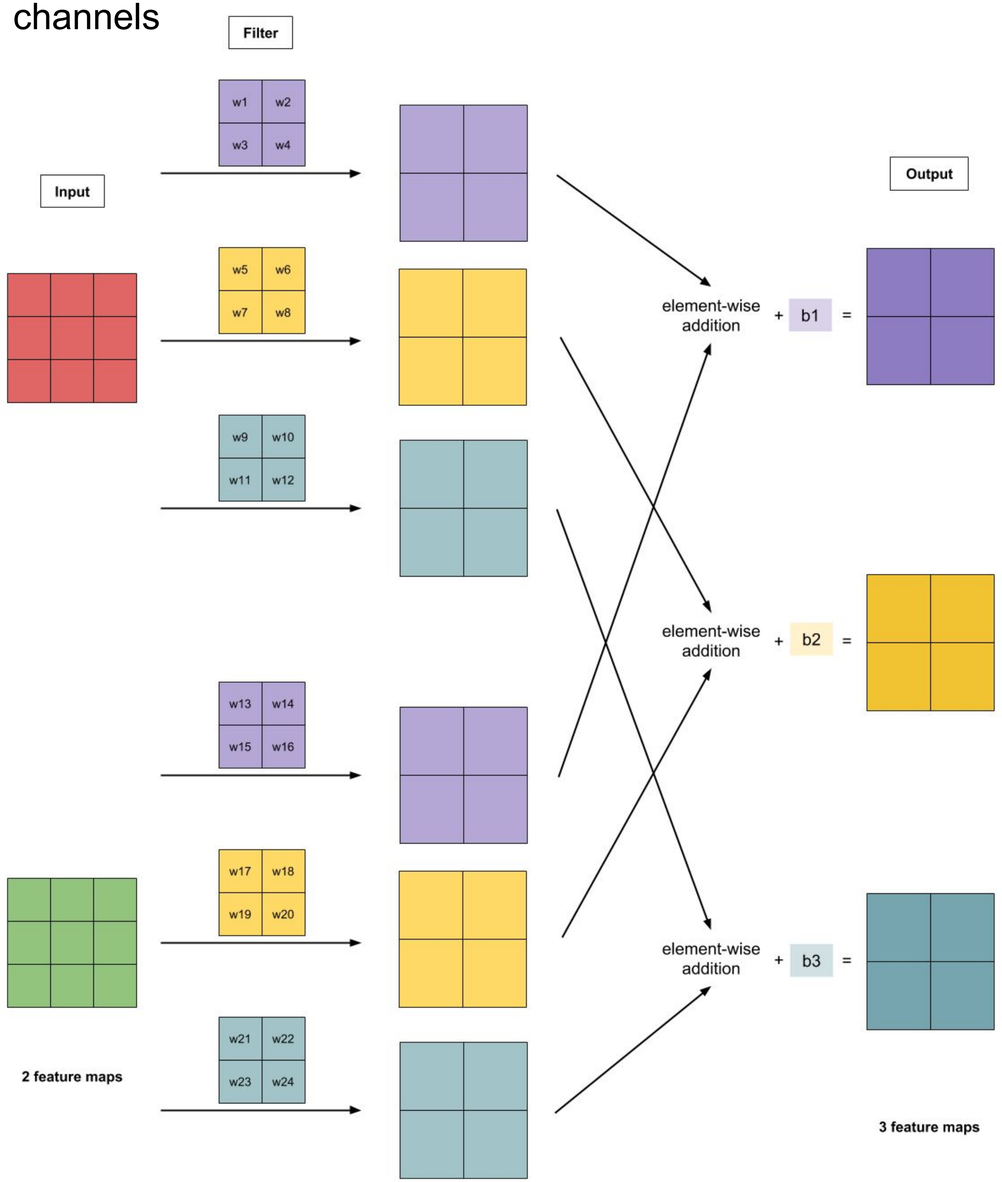
https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889#5137

Convolution Stage: Example

Example 2: input size 3*3*2 (W=3); 3 filters 2*2*2 (K=3, F=2); moving stride S=1; input padding P=0:
V=([3-2+2*0]/1)+1=2
Output volume: 2*2*3

Example 3: input size 32*32*3; 10 filters of size 5*5*3; single stride; and no zero padding W=32, K=10, F=5, P=0, S=1:
V=([32-5+0]/1)+1 = 28.
Output volume: 28*28*10.

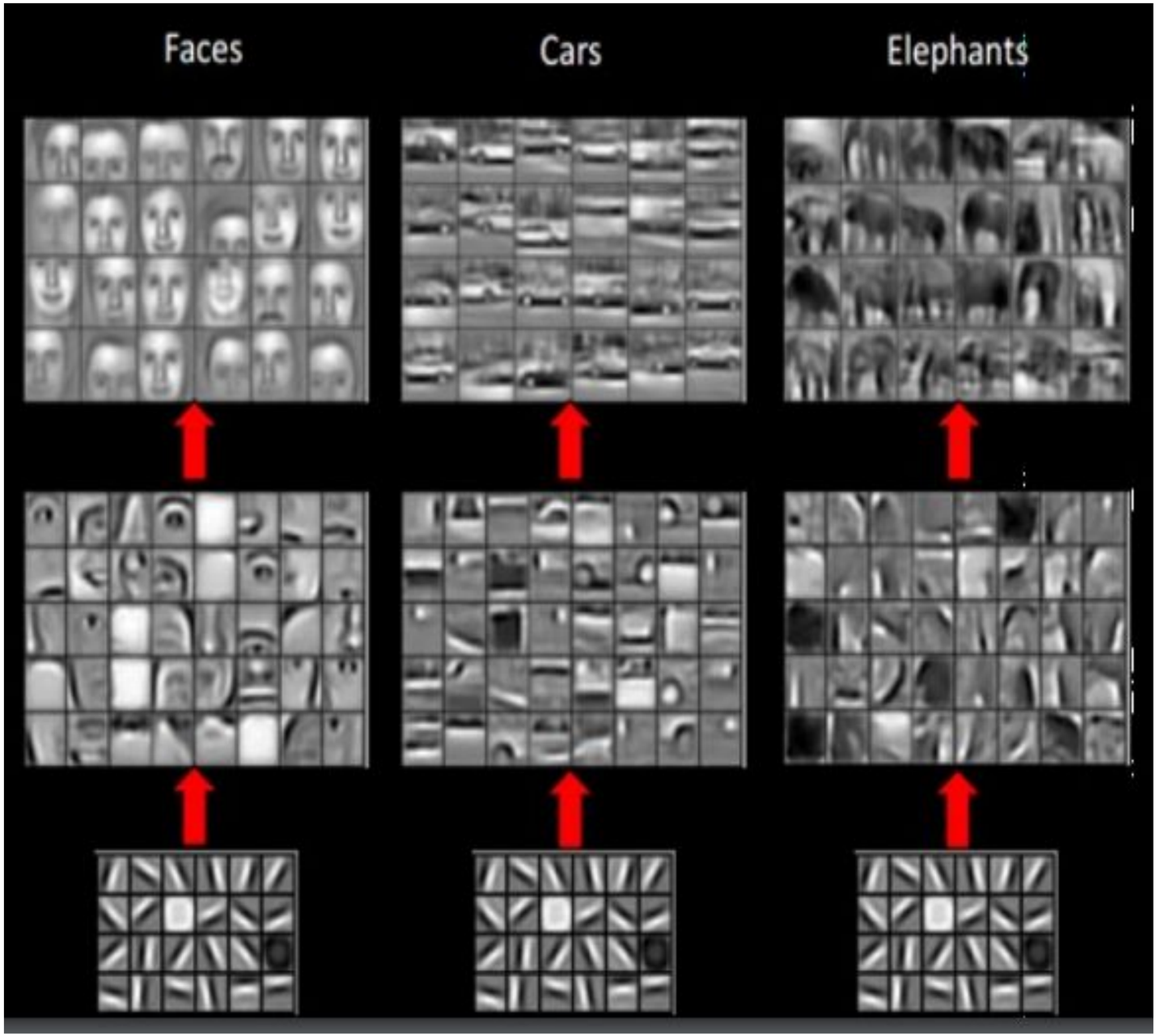
Convolution of 2-channel image with 2×2 filter to output 3 channels



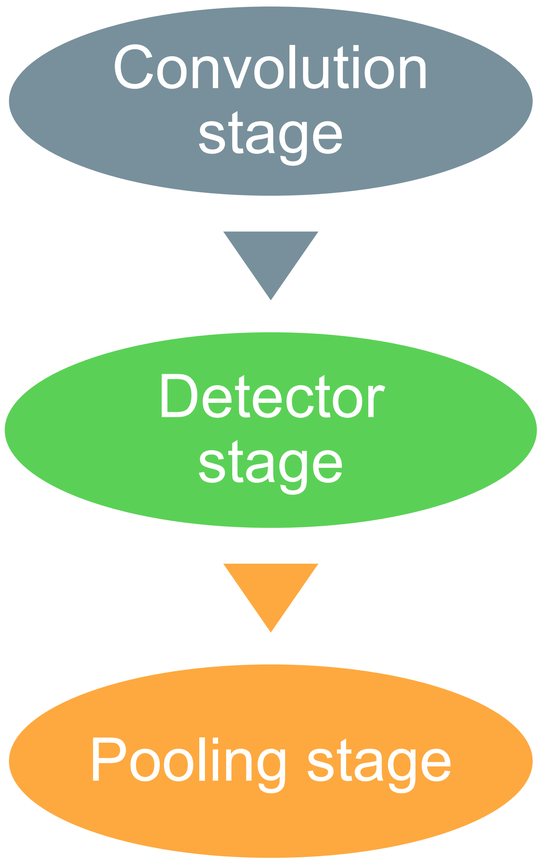
https://towardsdatascience.com/counting-no-of-parameters-in-deep-learning-models-by-hand-8f1716241889#5137

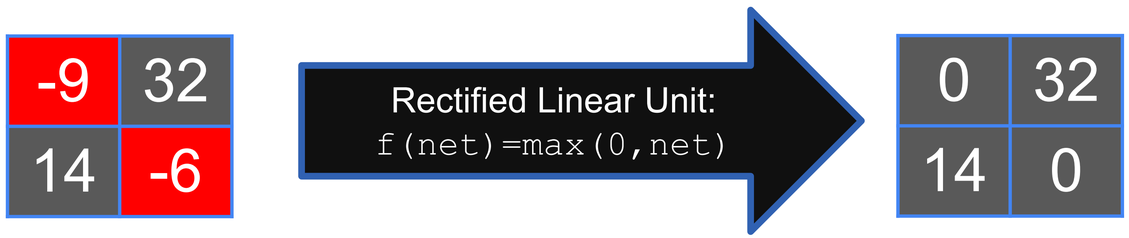
MultiFilter

More conv layer → more complex extracted feature



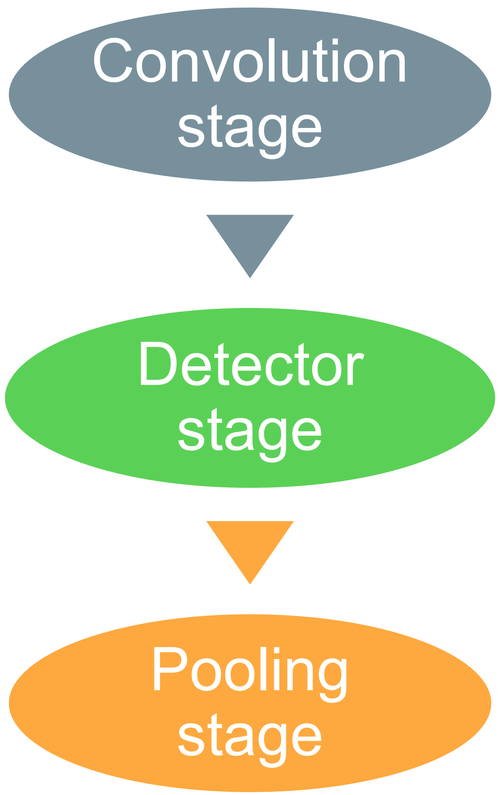
Detector Stage: Activation Function



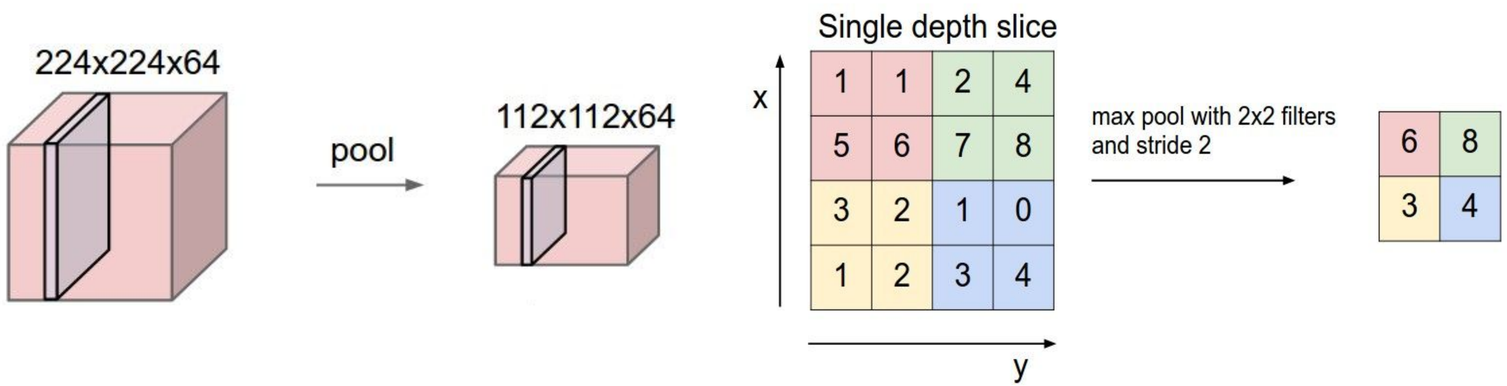


- This stage introduces nonlinearity to a system that basically has just been computing linear operations during the convolution stage.
- In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that **ReLU** (rectified linear unit) **stage** work far better.

Pooling Stage: Downsampling

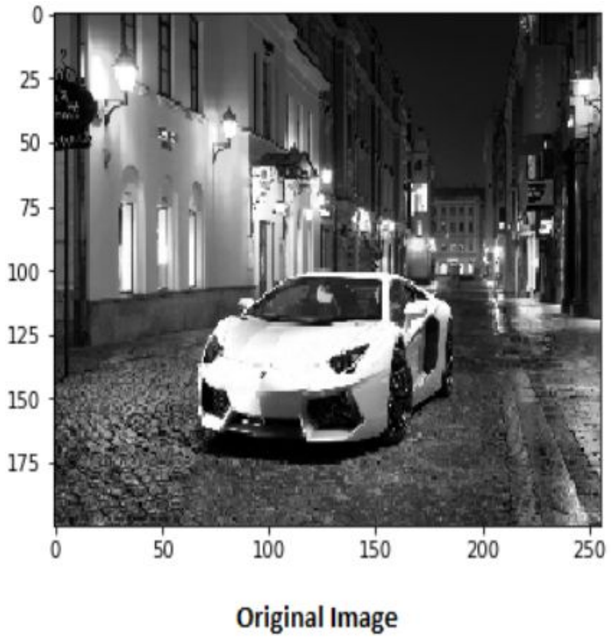


- Pooling aims to reduce spatial size of output from conv stage so filters can explore bigger parts of the image (handling overfitting).
- The most common of pooling is max pooling, average pooling, or L2-norm pooling.



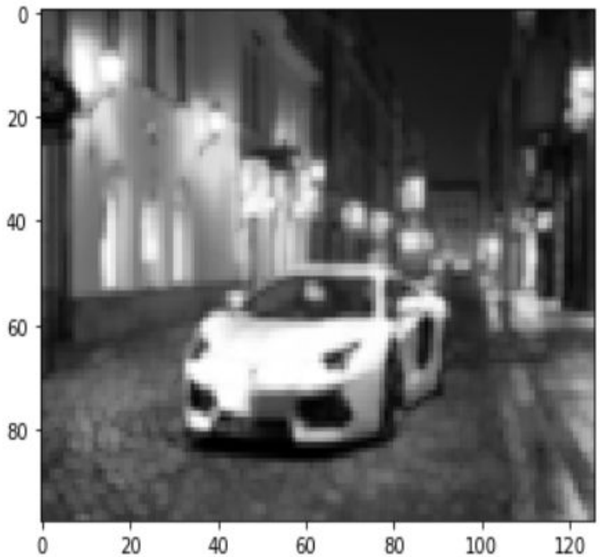
https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

Max pooling on a Real Image



Original Image

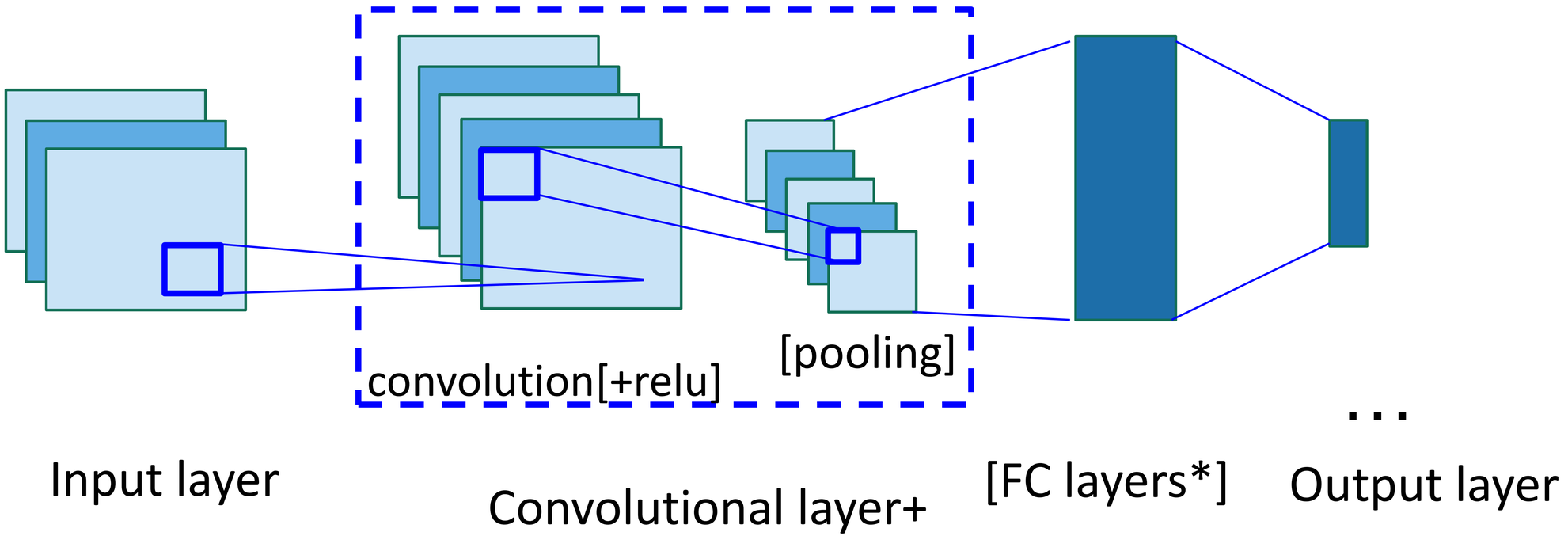
Convoluted image



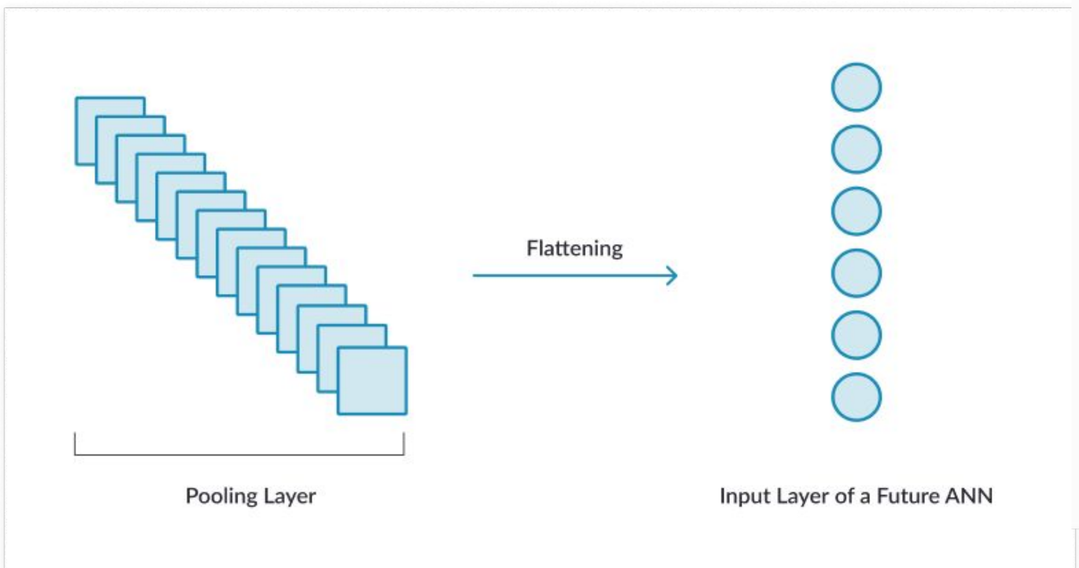
MAX Pooling

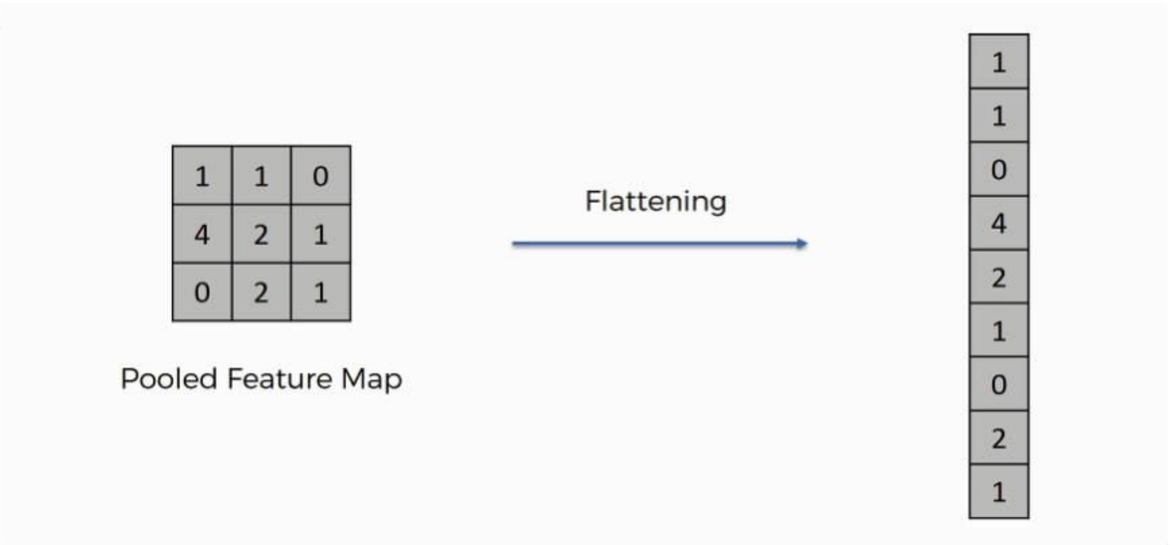
04 CNN Architecture

CNN Architecture



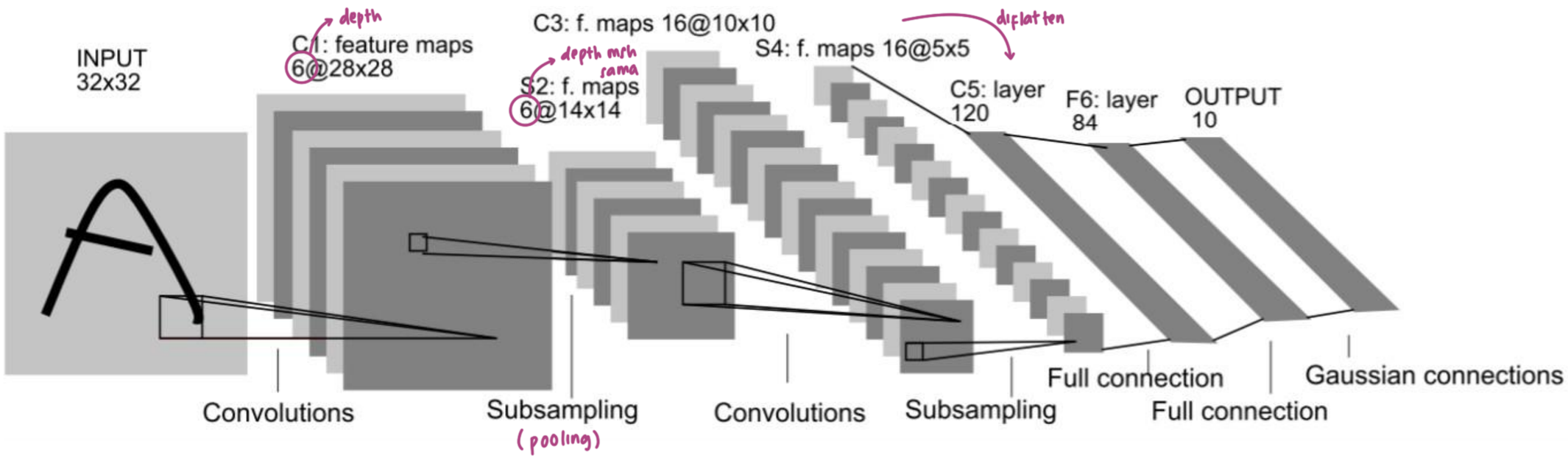
Flattening





https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

LeNet-5: LeCun (1998) to read digits

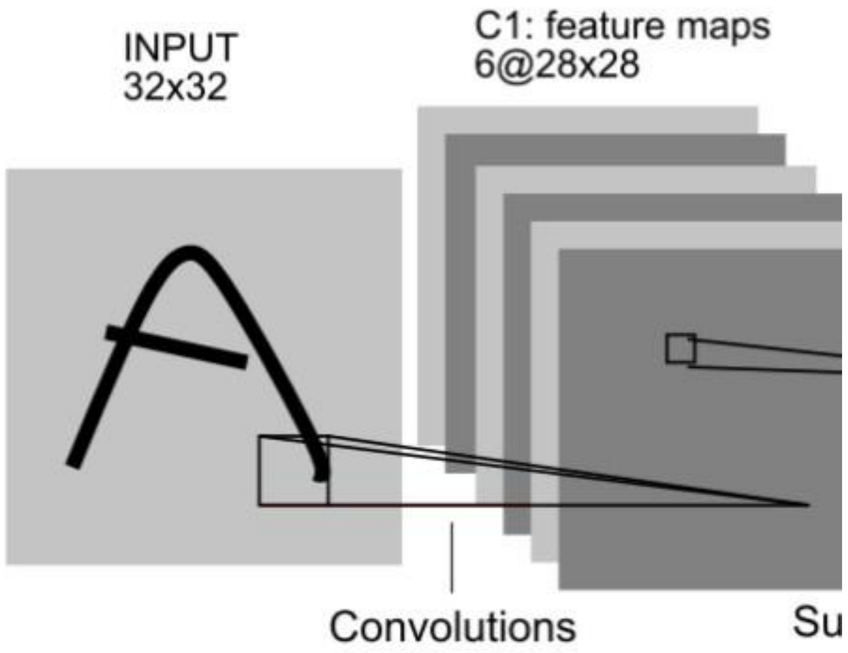


6 filter 5*5 2*2 (avg) 16 filter 5*5

convolution layer — conv. layer — FC

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

LeNet-5: Convolution Stage C1



6 filter 5*5

Nb of filter: 6

Filter size: 5*5

W=32; F=5; S=1; P=0:

V=[(32-5+2*0)/1]+1=28

Feature maps: 28*28*6

$$V = \frac{[W - F + 2P]}{S}$$

Trainable parameter:

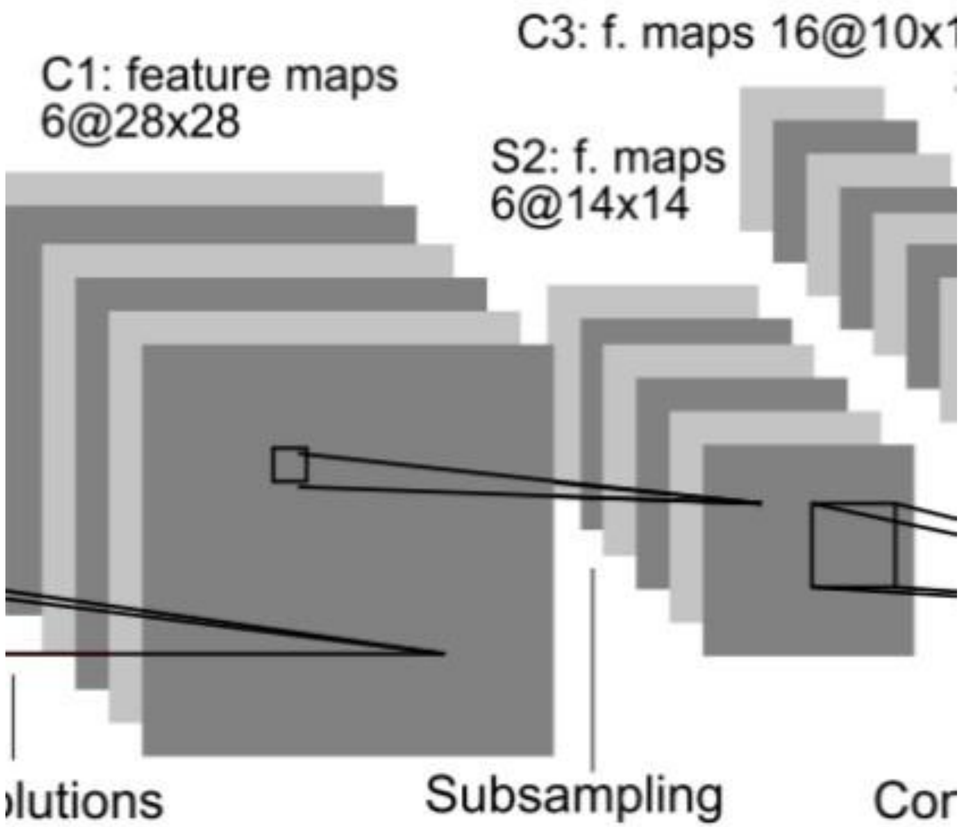
6*(5*5+1)=156 → *1 (krn jml input nya 1)

Connection: per kalian antar elemen yg terjadi

28*28*(5*5+1)*6= 122.304

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

LeNet-5: Pooling Stage S2



2*2 (avg)

Receptive field: 2*2 non-overlap

W=28; F=2; S=2; P=0:

V=[(28-2+2*0)/2]+1 =14

Feature maps: 14*14*6

Trainable parameter: 0 (Keras)

gaperlu pake bobot kalo di keras

Trainable parameter (LeCun):

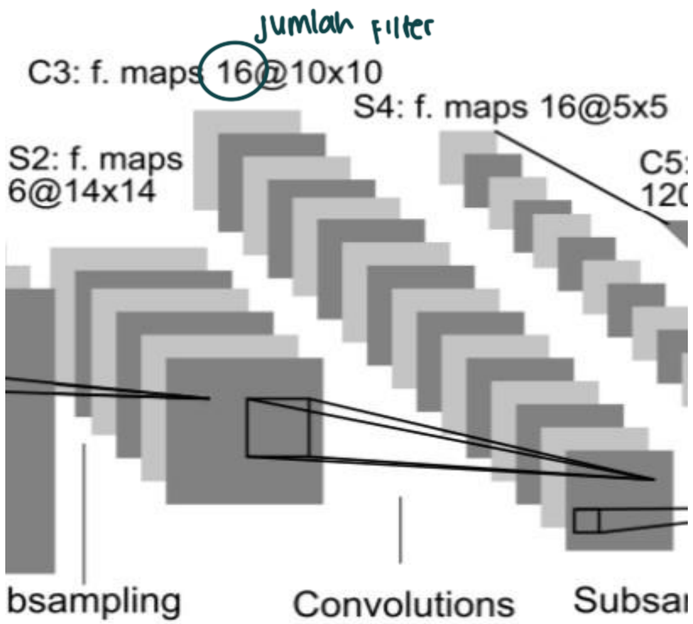
6*2=12

1 bias + 1 koefisien

Connections: 14*14*(2*2+1)*6=5880

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

LeNet-5: Convolution Stage C3



16 filter 5*5

Nb of filter: 16

Filter size: 5*5

W=14; F=5; S=1; P=0:

V=[(14-5+2*0)/1]+1=10

Feature maps: 10*10*16 → ada 16 filter

W : ukuran feature map sblmny
F : ukuran lebar filter
S : ukuran stride
P : ukuran padding

Trainable parameter:

16*(5*5*6+1) =2416 (all feature maps) (+1 dr input stage sblmnya)

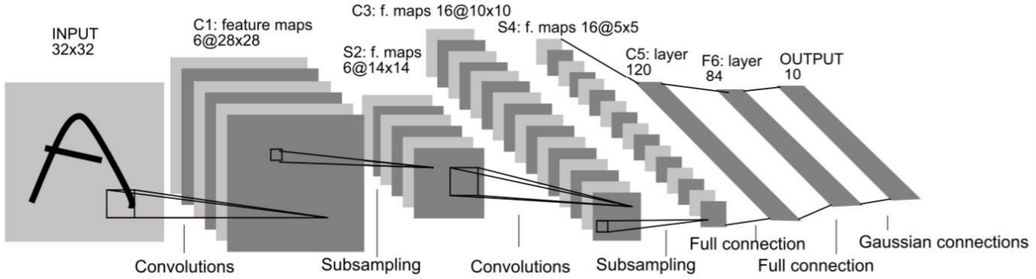
5*5*(6*3 + 9*4 +1*6) + 16 = 1516(subset) → optimasi (dia gapake semua feature map)

Connection:

10*10*(5*5*6+1) *16= 241600 (all)

156.000 (subset)

Implementation LeNet-5




```
from keras.models import Sequential
from keras import layers

model = Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(5, 5),   activation='
          relu', input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3),  activation='
          relu'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='relu'))
model.add(layers.Dense(units=84, activation='relu'))
model.add(layers.Dense(units=10, activation = 'softmax'))
```

https://medium.com/@mgazar/lenet-5-in-9-lines-of-code-using-keras-ac99294c8086

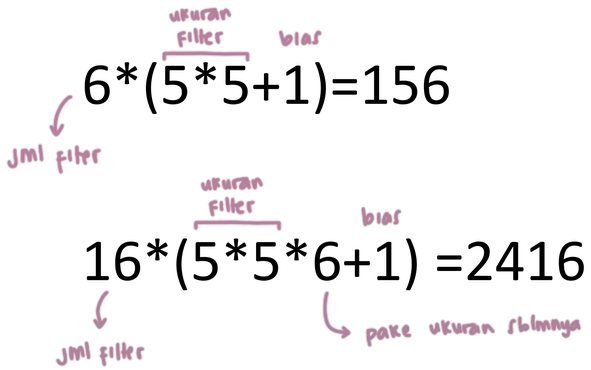
Implementation LeNet-5: Model Summary

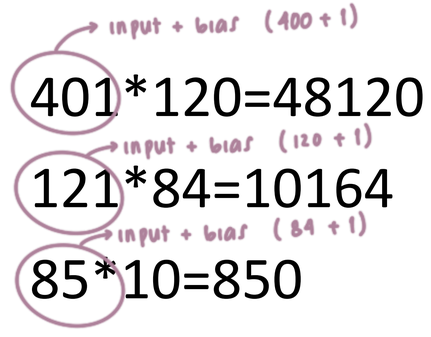
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 6)	156
average_pooling2d (AveragePo	(None, 14, 14, 6)	0
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
average_pooling2d_1 (Average	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48120
dense_1 (Dense)	(None, 84)	10164
dense_2 (Dense)	(None, 10)	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

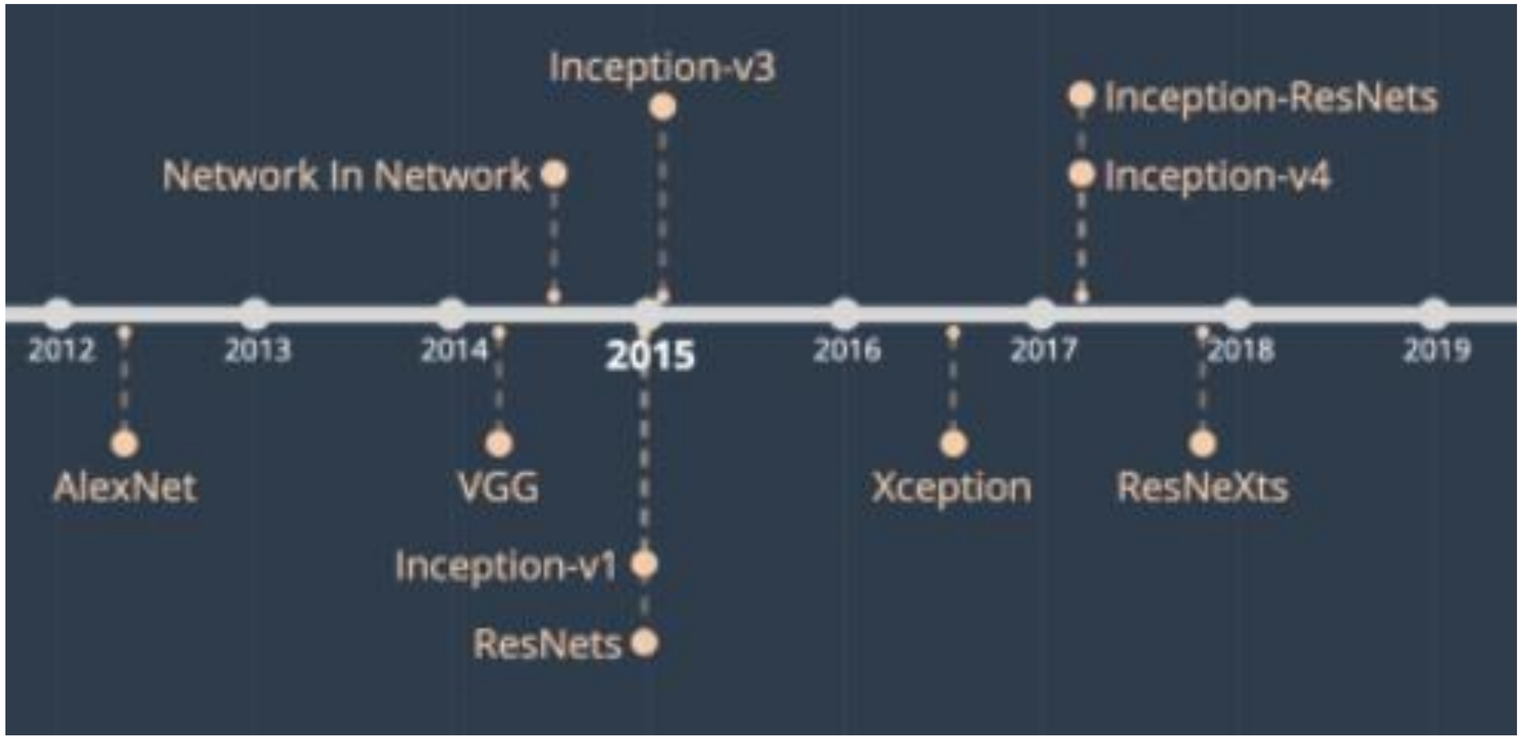
pooling 0 krn di keras

pooling 0 krn di keras

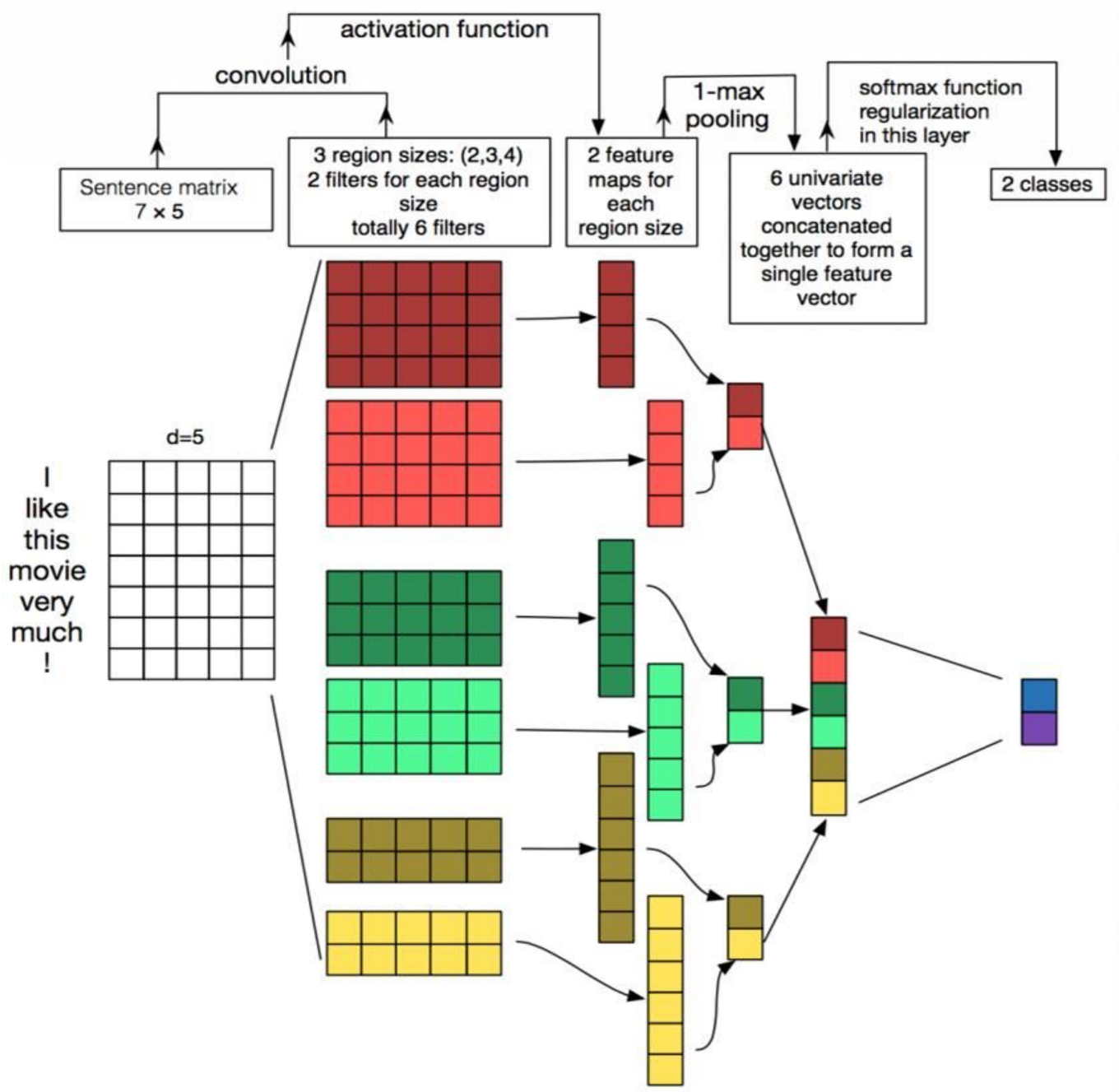




Common Architecture



Text Classification using CNN



CNN-XGBoost Architecture

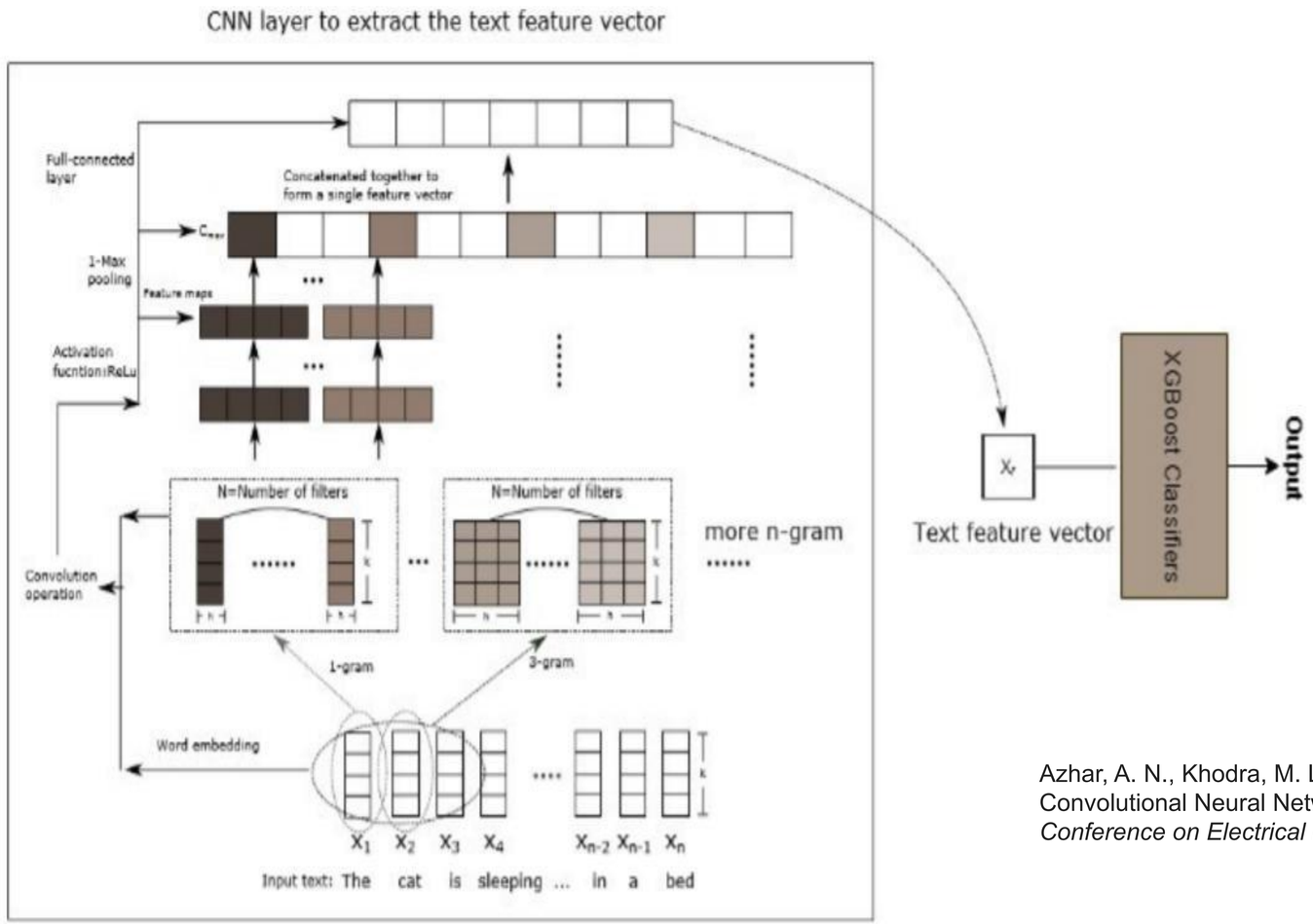


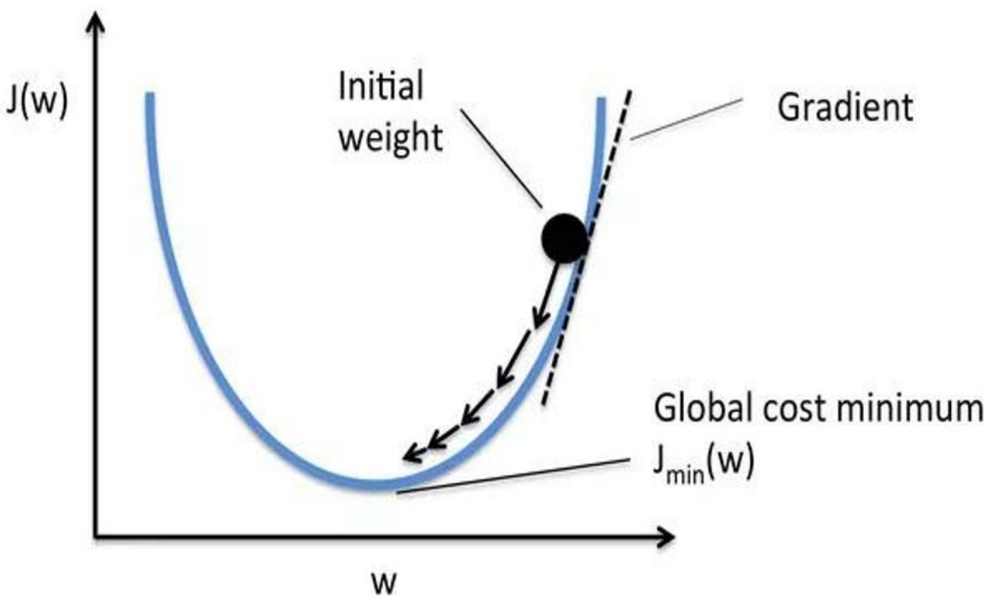
Fig. 1. Proposed Method (CNN-XGBoost) Architecture

Azhar, A. N., Khodra, M. L., & Sutiono, A. P. (2019, July). Multi-label Aspect Categorization with Convolutional Neural Networks and Extreme Gradient Boosting. In *2019 International Conference on Electrical Engineering and Informatics (ICEEI)* (pp. 35-40). IEEE.

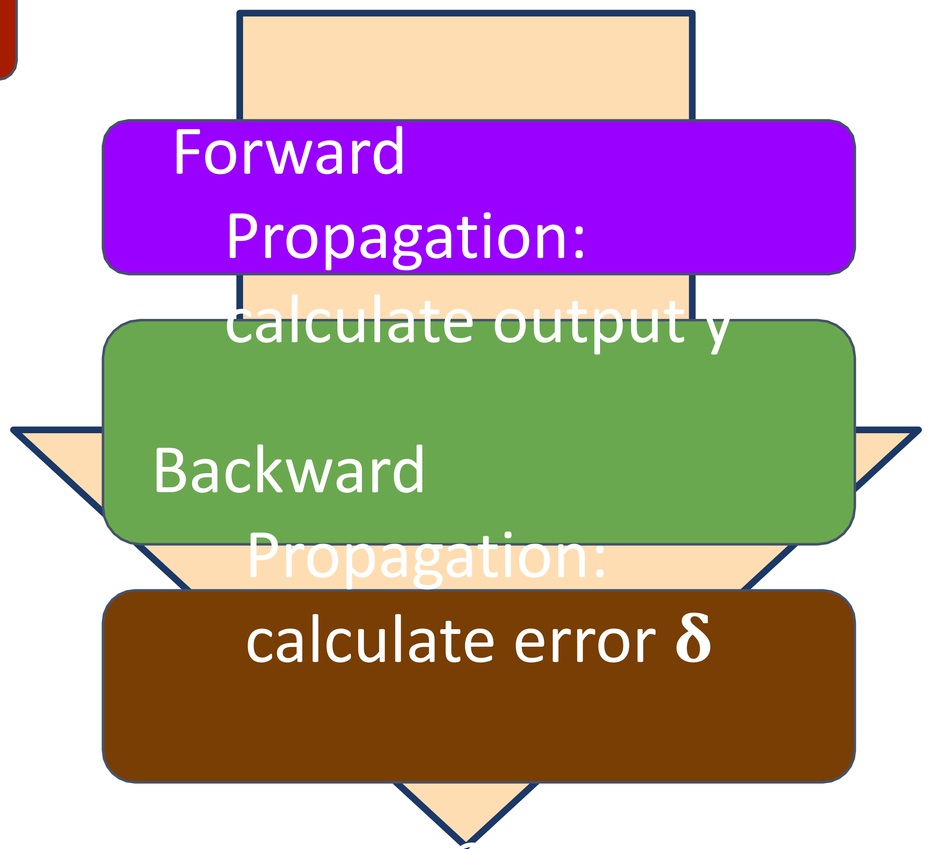
05 Backpropagation for CNN

Backpropagation Overview

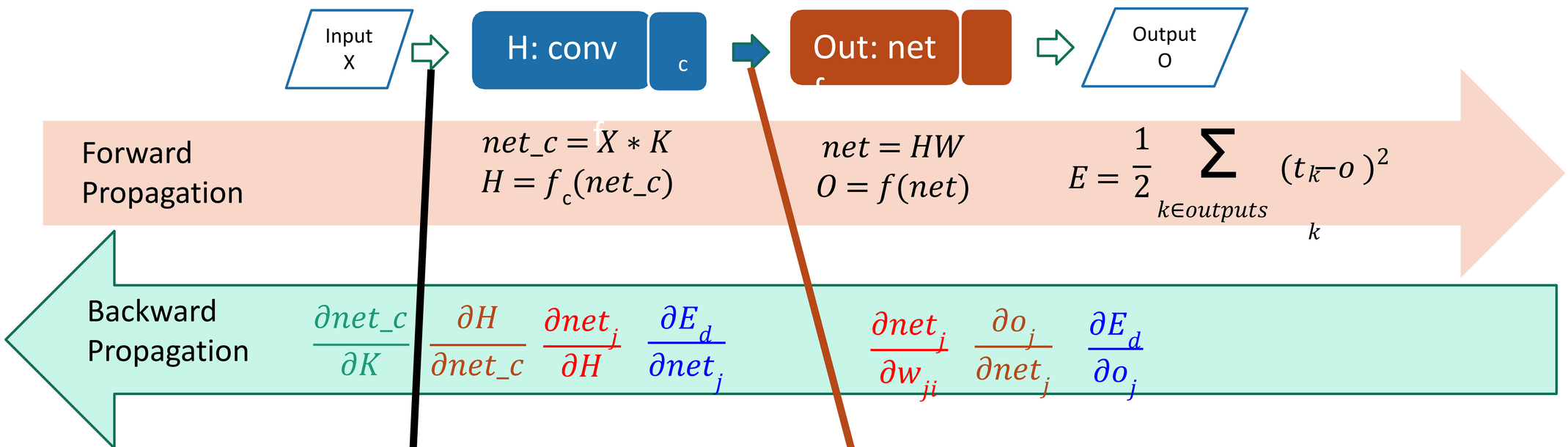
Goal: set weight to minimize error



https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c



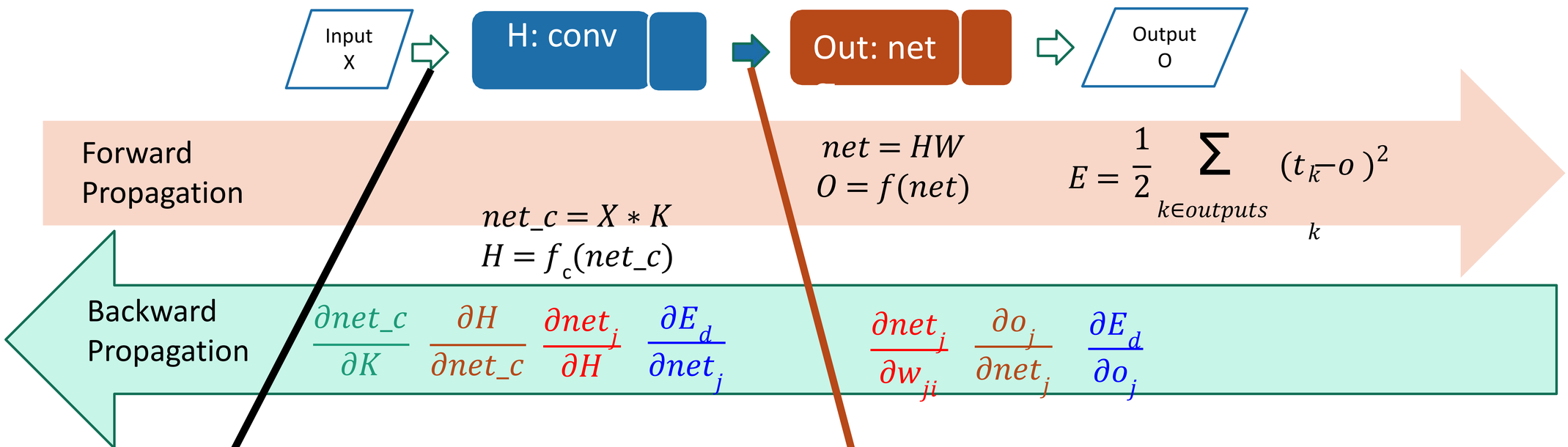
Backpropagation for CNN



$$K = K - \eta \frac{\partial E_d}{\partial K} = K - \eta [\frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial H} \frac{\partial H}{\partial net\_c} \frac{\partial net\_c}{\partial K}]$$

$$w_{ji} = w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}} = w_{ji} - \eta [\frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}]$$

Backpropagation for CNN: Example



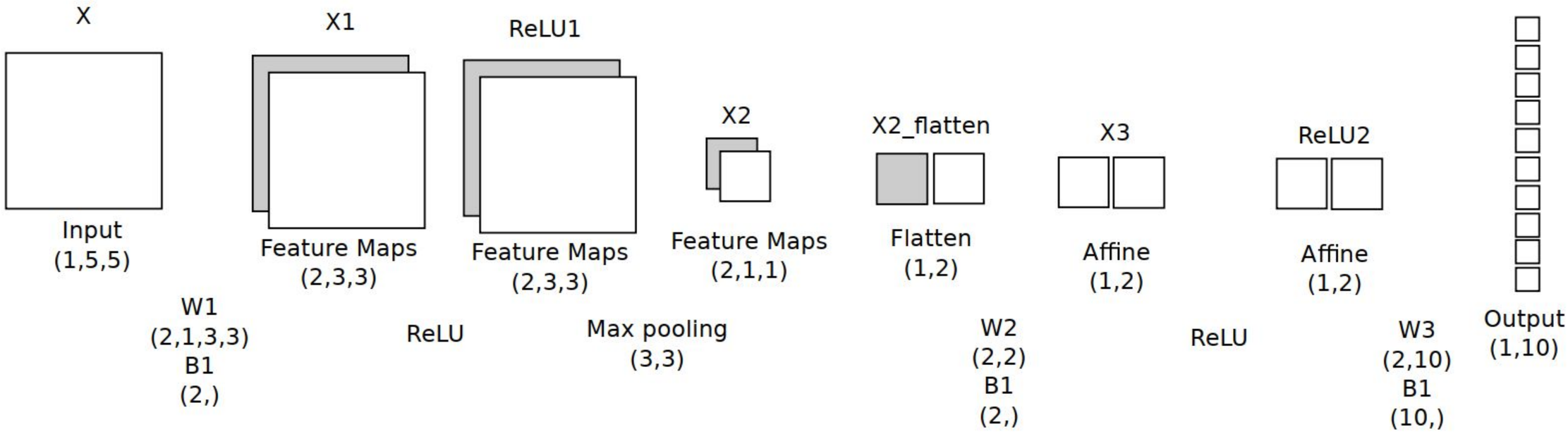
$$K = K - \eta \frac{\partial E_d}{\partial K} = K - \eta \left[\frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial H} \frac{\partial H}{\partial net\_c} \frac{\partial net\_c}{\partial K} \right]$$

$$= K - \eta[-(t_j - o_j)\ o_j(1 - o_j).W.H(1\text{-}H)*X]$$

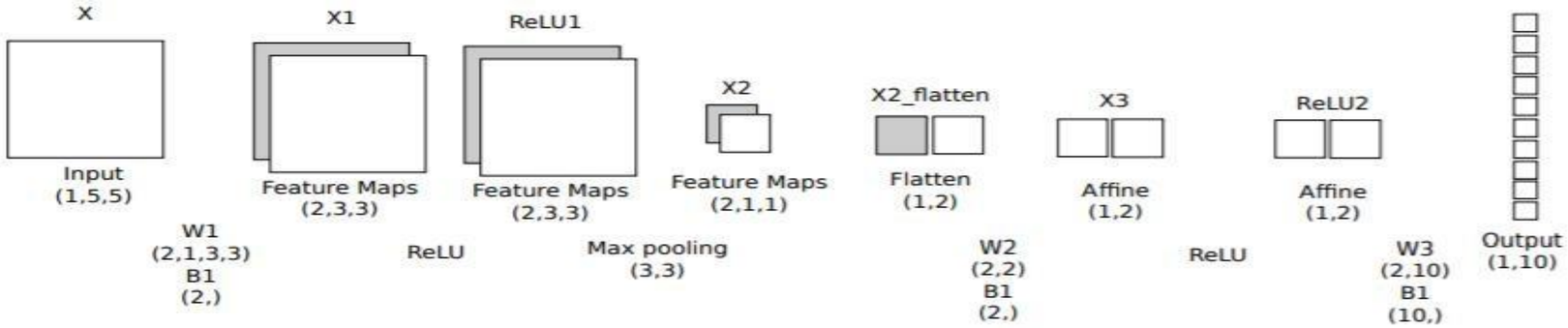
$$w_{ji} = w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}} = w_{ji} - \eta \left[\frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \right]$$

$$= w_{ji} - \eta[-(t_j - o_j)\ o_j(1 - o_j)\ H]$$

Example


X
Input
(1,5,5)
W1
(2,1,3,3)
B1
(2,)
X1
Feature Maps
(2,3,3)
ReLU
ReLU1
Feature Maps
(2,3,3)
Max pooling
(3,3)
X2
Feature Maps
(2,1,1)
X2_flatten
Flatten
(1,2)
W2
(2,2)
B1
(2,)
X3
Affine
(1,2)
ReLU
ReLU2
Affine
(1,2)
W3
(2,10)
B1
(10,)
Output
(1,10)

Example: Forward Propagation



X

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

X1

-85	76	64
109	-1	10
118	71	67

-144	-291	66
-347	102	-192
-239	-52	-162

ReLU1

0	76	64
109	0	10
118	71	67

0	0	66
0	102	0
0	0	0

X2

118
102

X3

424	-172

ReLU2

424	0

Output.T

9.855923e-01
1.419998e-02
2.045877e-04
2.947614e-06
4.246805e-08
6.118620e-10
8.815459e-12
6.118620e-10
4.246805e-08
1.829901e-15

W1

1	2	3
4	7	5
3	-32	25

12	18	12
18	-74	45
-92	45	-18

B1

0
0

W2

1	2
3	-4

B2

0
0

W3.T

0.09	0.02
0.08	0.03
0.07	0.03
0.06	0.02
0.05	0.01
0.04	0.02
0.03	0.07
0.04	0.08
0.05	0.05
0.01	0.01

B3.T

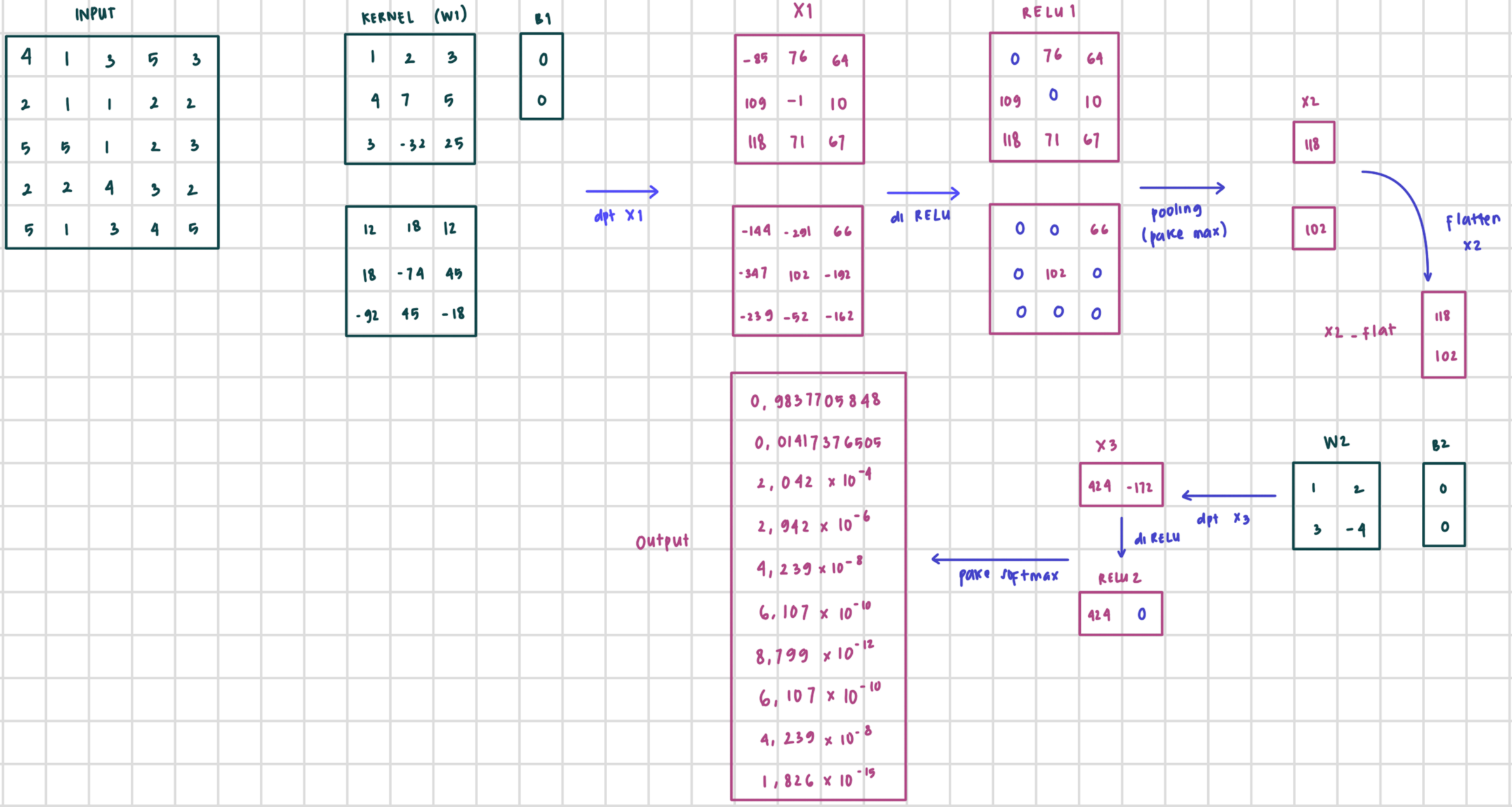
0
0
0
0
0
0
0
0
0
0

softmax

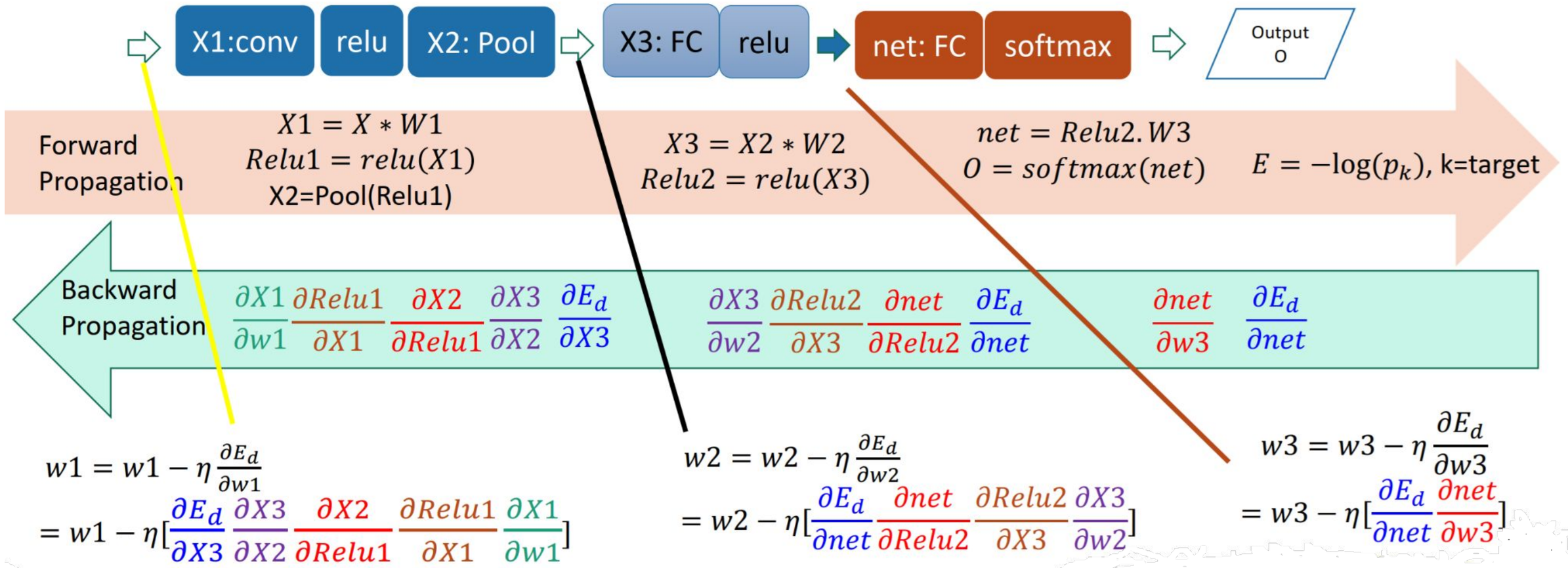
$$\Gamma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

buat case ini

$$\sum_{j=1}^{K} e^{z_j} = 3.8 \times 10^{16}$$


INPUT
4 1 3 5 3
2 1 1 2 2
5 5 1 2 3
2 2 4 3 2
5 1 3 4 5
KERNEL (W1)
1 2 3
4 7 5
3 -32 25
12 18 12
18 -74 45
-92 45 -18
B1
0
0
dpt X1
X1
-85 76 64
109 -1 10
118 71 67
-144 -201 66
-347 102 -192
-239 -52 -162
di RELU
RELU 1
0 76 64
109 0 10
118 71 67
0 0 66
0 102 0
0 0 0
pooling (pake max)
X2
118
102
Flatten X2
X2 - flat
118
102
W2
1 2
3 -4
B2
0
0
dpt X3
X3
424 -172
di RELU
RELU 2
424 0
pake softmax
Output
0,983770598 48
0,01417376505
2,042 x 10^-4
2,942 x 10^-6
4,239 x 10^-8
6,107 x 10^-10
8,799 x 10^-12
6,107 x 10^-10
4,239 x 10^-8
1,826 x 10^-15

Example: Backward Propagation



Computing Gradient for Output Unit Weights

Chain rule to compute gradient:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$net = Relu2.W3$$
$$O = softmax(net)$$

$$E = -\log(p_k), \text{ k=target}$$

$$\frac{\partial net_j}{\partial w3_{ji}} = Relu2_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = \begin{cases} p_j, j \neq targetClass \\ -(1-p_j), j = targetClass \end{cases}$$

Computing $\frac{\partial E_d}{\partial net}$: Example

Output.T
9.855923e-01
1.419998e-02
2.045877e-04
2.947614e-06
4.246805e-08
6.118620e-10
8.815459e-12
6.118620e-10
4.246805e-08
1.829901e-15

correct label was 9 (0-based idx)

Error=-log(p_{target})
=-log(1.83e-15)
=33.93

$\frac{\partial E_d}{\partial net_j}$

$\frac{\partial E_d}{\partial net}$

9.855923e-01
1.419998e-02
2.045877e-04
2.947614e-06
4.246805e-08
6.118620e-10
8.815459e-12
6.118620e-10
4.246805e-08
-1

$$\frac{\partial E_d}{\partial net_j} = \begin{cases} p_j, j \neq targetClass \\ -(1-p_j), j = targetClass \end{cases}$$

Computing $\frac{\partial E_d}{\partial w3}$: Example

$$\frac{\partial E_d}{\partial w3} = \frac{\partial E_d}{\partial net}\frac{\partial net}{\partial w3} = \frac{\partial E_d}{\partial net} Relu2$$

$\frac{\partial E_d}{\partial net}$

9.855923e-01
1.419998e-02
2.045877e-04
2.947614e-06
4.246805e-08
6.118620e-10
8.815459e-12
6.118620e-10
4.246805e-08
-1

Relu2
1
424
0

bias

$\frac{\partial E_d}{\partial w3}$ (dW3.T & dB3.T)

dE/dw3_b3	dE/dw3_1	dE/dw3_2
9.86E-01	4.18E+02	0.00E+00
1.42E-02	6.02E+00	0.00E+00
2.05E-04	8.67E-02	0.00E+00
2.95E-06	1.25E-03	0.00E+00
4.25E-08	1.80E-05	0.00E+00
6.12E-10	2.59E-07	0.00E+00
8.82E-12	3.74E-09	0.00E+00
6.12E-10	2.59E-07	0.00E+00
4.25E-08	1.80E-05	0.00E+00
-1.00E+00	-4.24E+02	0.00E+00

Computing $\frac{\partial E_d}{\partial w2}$: Example

$$\frac{\partial E_d}{\partial w2} = \frac{\partial E_d}{\partial net}\frac{\partial net}{\partial Relu2}\frac{\partial Relu2}{\partial X3}\frac{\partial X3}{\partial w2} = \frac{\partial E_d}{\partial net} w3 \frac{\partial Relu2}{\partial X3} X2$$

$\frac{\partial E_d}{\partial net}$

9.855923e-01
1.419998e-02
2.045877e-04
2.947614e-06
4.246805e-08
6.118620e-10
8.815459e-12
6.118620e-10
4.246805e-08
-1

	w3	
0	0.09	0.02
0	0.08	0.03
0	0.07	0.03
0	0.06	0.02
0	0.05	0.01
0	0.04	0.02
0	0.03	0.07
0	0.04	0.08
0	0.05	0.05
0	0.01	0.01

$\frac{\partial E_d}{\partial X3}$ (dRelu2)

dE/dX3	X2
0	1
0.079854	118
0	102

$\frac{\partial E_d}{\partial w2}$ (dW2.T)

dE/dw2		
0	0.0799	0
0	9.4227	0
0	8.1451	0

Computing $\frac{\partial E_d}{\partial w1}$: Example

$$\frac{\partial E_d}{\partial w1} = \frac{\partial E_d}{\partial X3}\frac{\partial X3}{\partial X2}\frac{\partial X2}{\partial Relu1}\frac{\partial Relu1}{\partial X1}\frac{\partial X1}{\partial w1}$$

$(B_2.T)$ $\frac{\partial E_d}{\partial X3}$

dE/dX3
0
0.079854
0

dX_2 flatten .T

w2		
0	1	2
0	3	-4

$\frac{\partial E_d}{\partial X1}$ $dRelu1 = dX_1$

0	0	0
0	0	0
0.07985382	0	0

0	0	0
0	0.23956146	0
0	0	0

$\frac{\partial E_d}{\partial w1}$

0.3992691	0.3992691	0.07985382
0.15970764	0.15970764	0.31941528
0.3992691	0.07985382	0.23956146

0.23956146	0.23956146	0.47912292
1.1978073	0.23956146	0.47912292
0.47912292	0.95824584	0.71868438

δB1
sum of the error for
channel in δX1

0.07985382	0.23956146

References

1. Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1, No. 2). Cambridge: MIT press, https://www.deeplearningbook.org/
2. Akhil Suri, Introduction to Deep Learning, https://www.slideshare.net/slideshow/introduction-to-deep-learning-how-why-deep-learning/266273570, last accessed: 22 March 2024
3. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.
4. Y. LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition," in Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.
5. E.E.J. Lepe and A. M. Vazquez, A beginner's tutorial for CNN, 2017

Thank you