Practical Machine Learning

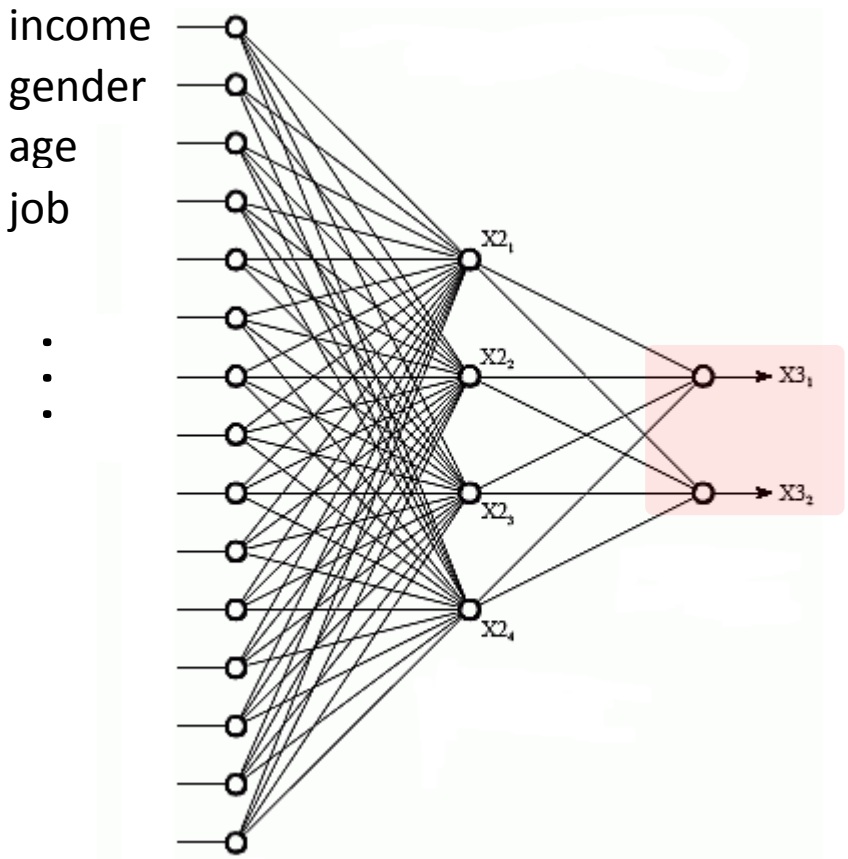
Workshop 3.
Convolutional Neural Network + Style Transfer

Dr. Suyong Eum

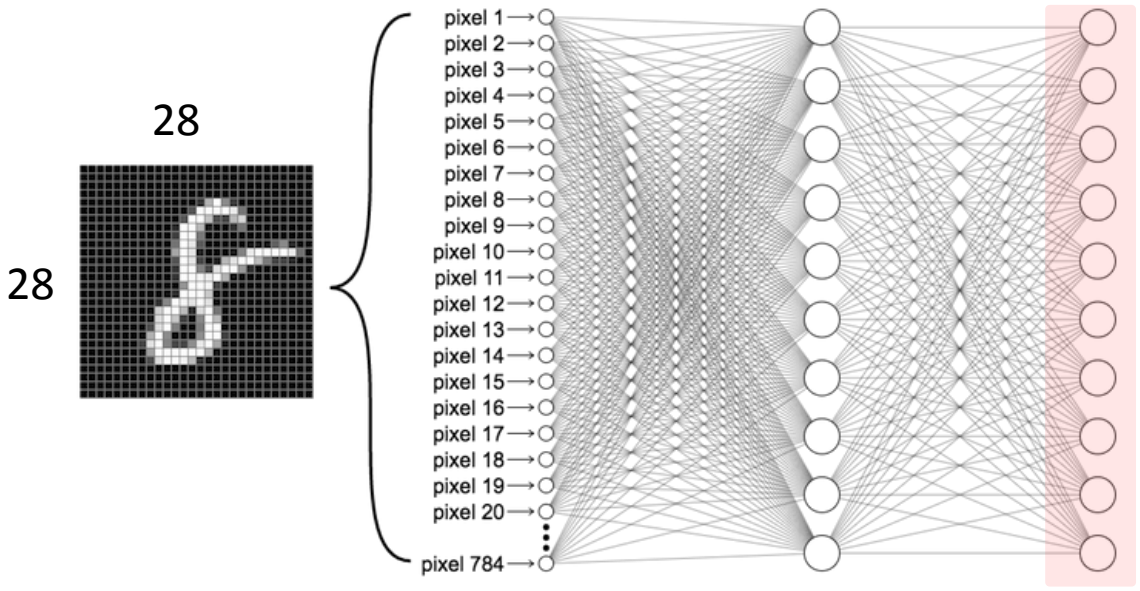
OSAKA UNIVERSITY

Neural Networks

Example structures of neural networks

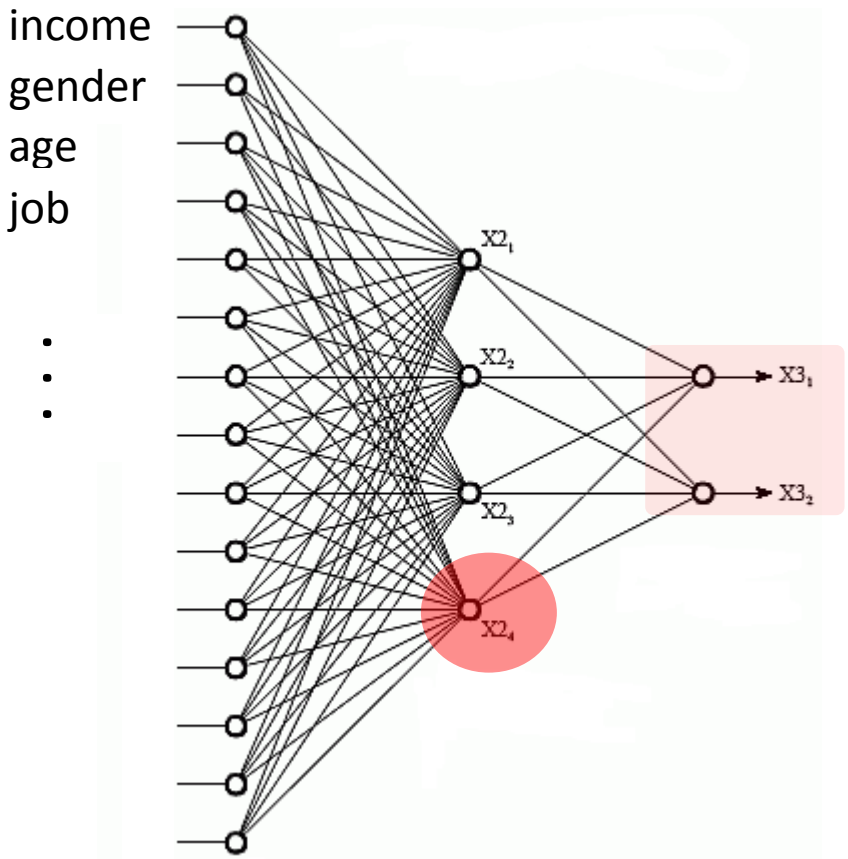


Credit card approval

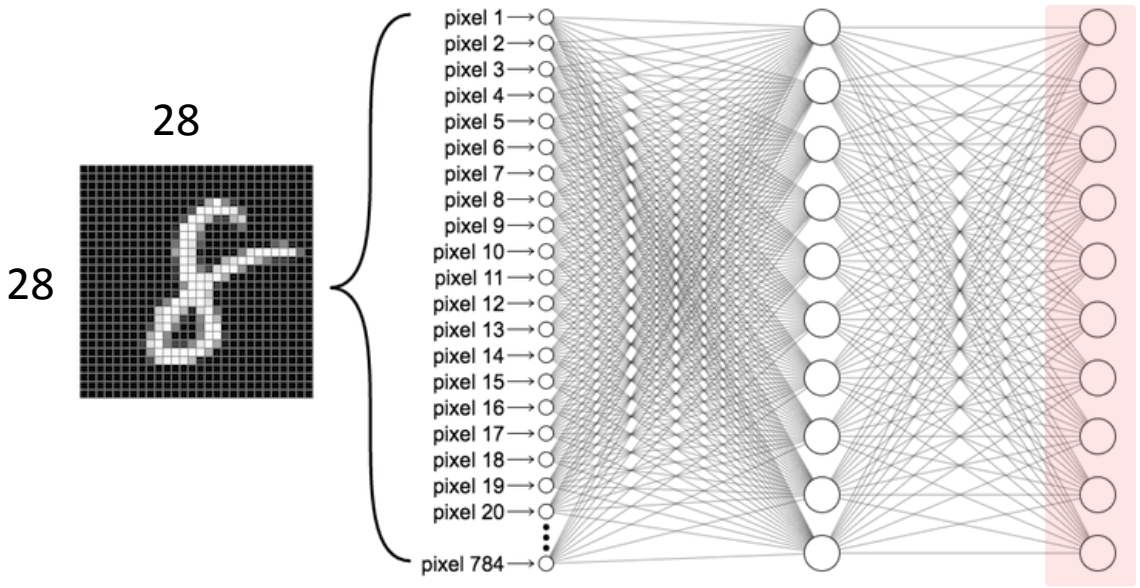


Digit recognition

Example structures of neural networks

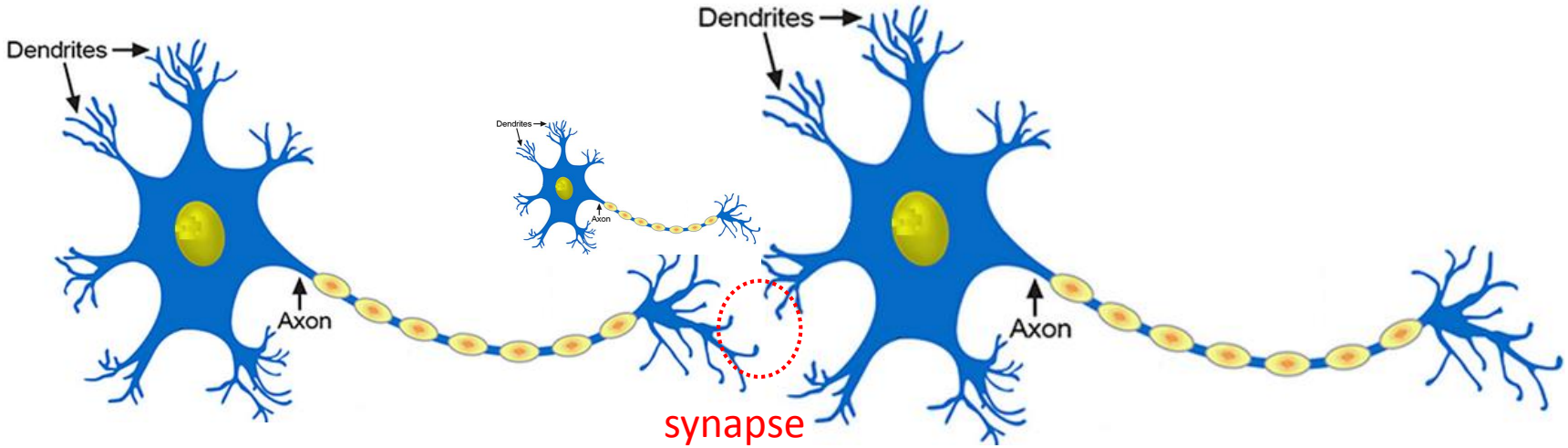


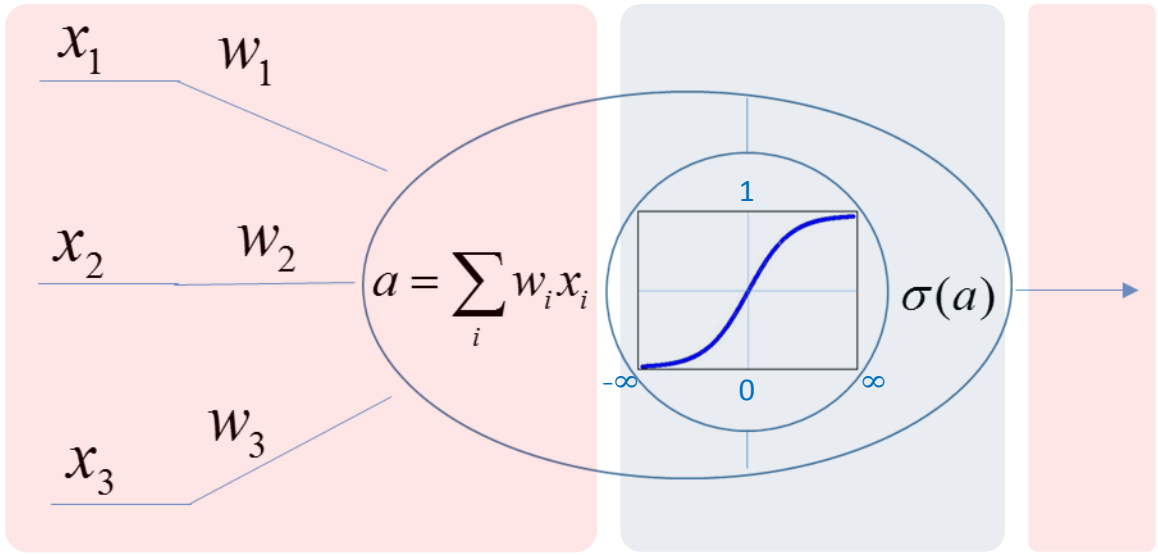
Credit card approval



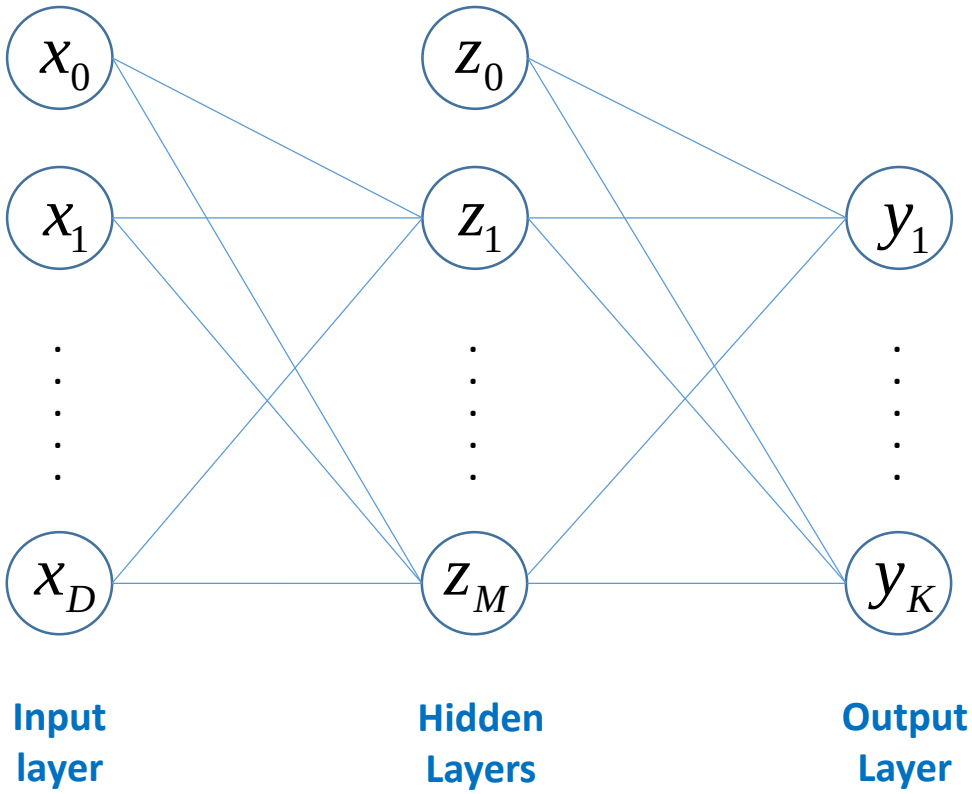
Digit recognition

A bio-inspired approach





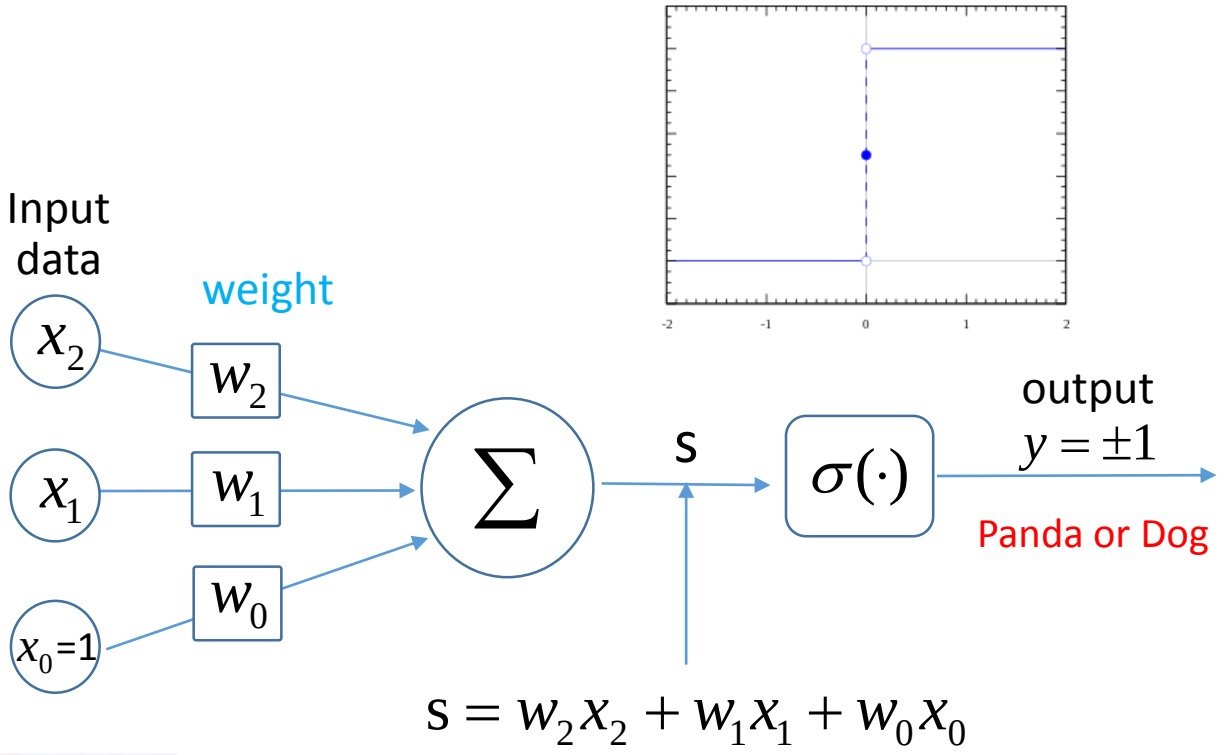
Terminology in neural networks

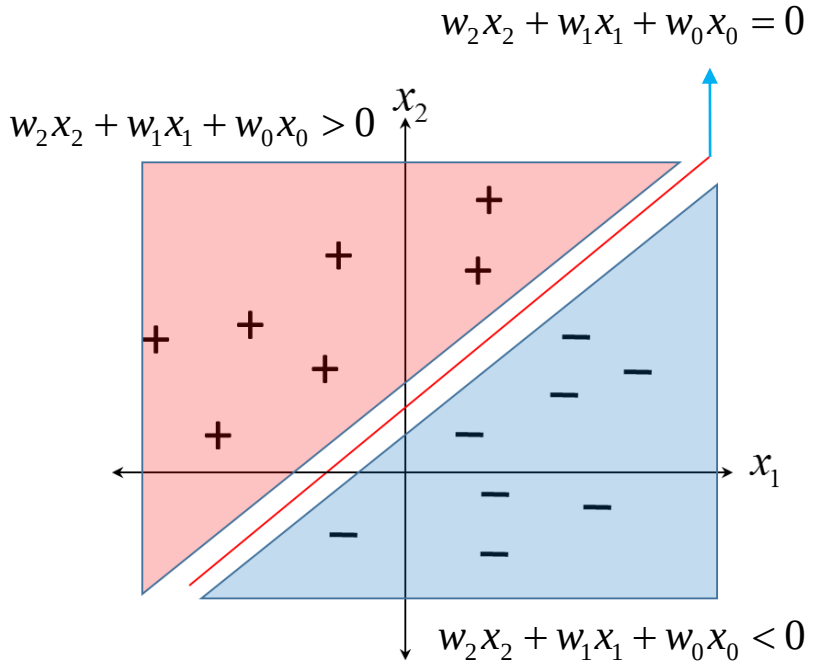


How many layers it has?

One layer neural network (perceptron): architecture

- Simplest form of a neural network used for a linear classification.

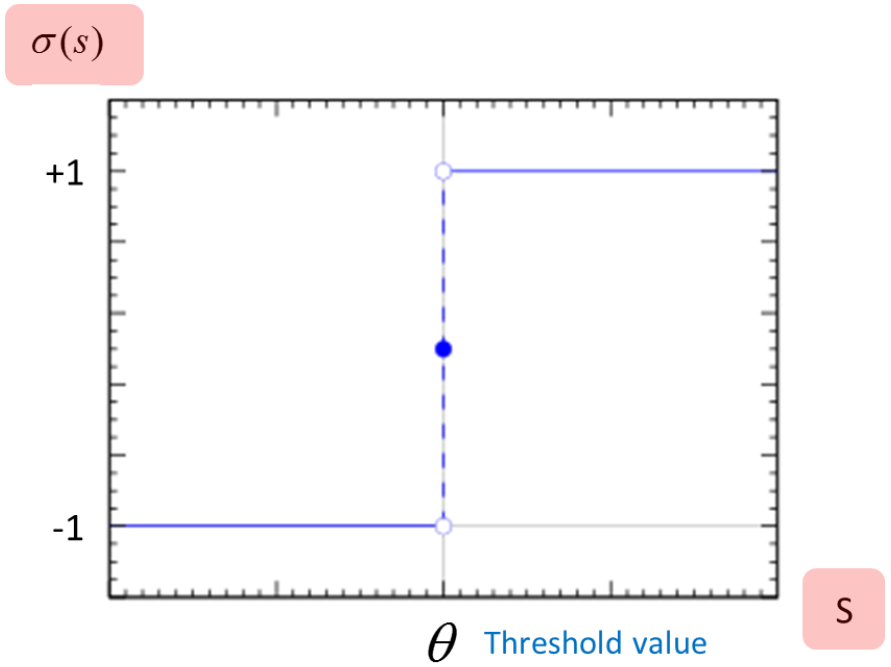




One layer neural network (perceptron): architecture

- It uses a step function as an activation function: Yes/No question.
- The step function is discontinuous: any problem?

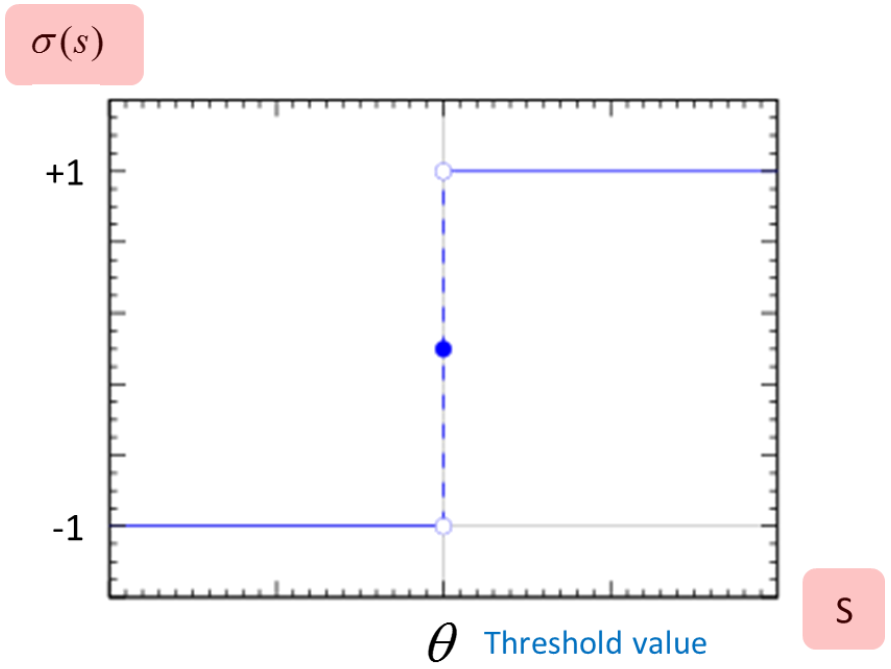
$$\sigma(s) = \begin{cases} +1 & if \ \ s > \theta \\ -1 & otherwise \end{cases}$$

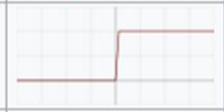
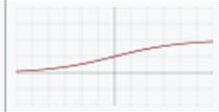
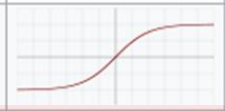


One layer neural network (perceptron): architecture

- It uses a step function as an activation function: Yes/No question.
- The step function is discontinuous: any problem?

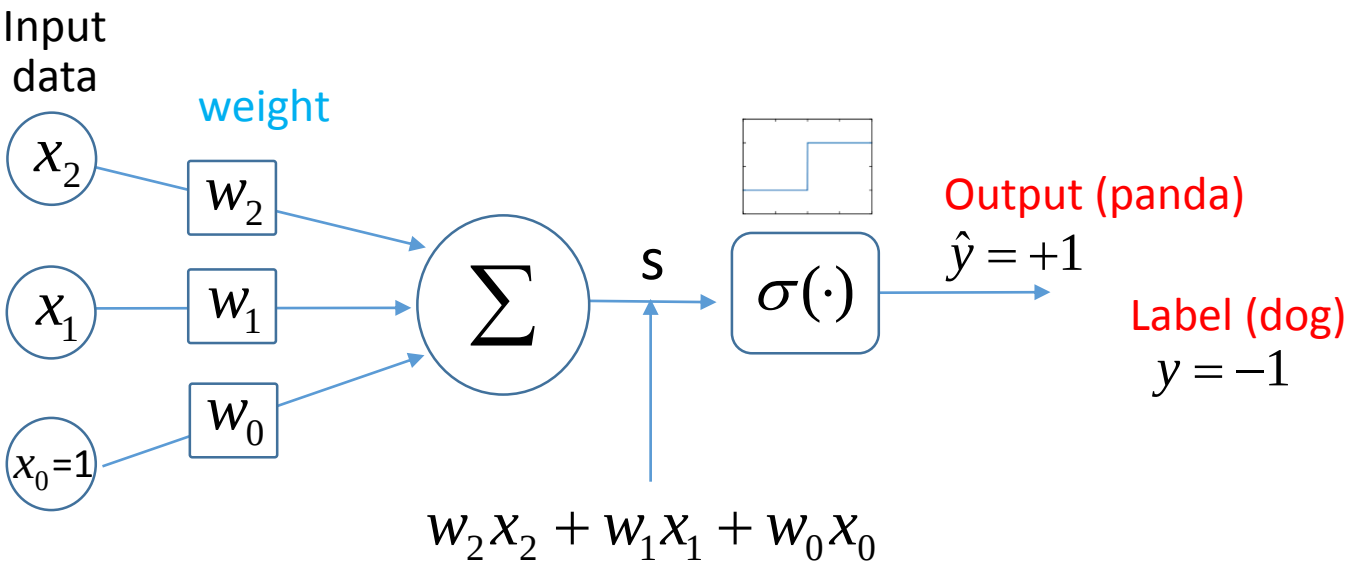
$$\sigma(s) = \begin{cases} +1 & if \ \ s > \theta \\ -1 & otherwise \end{cases}$$

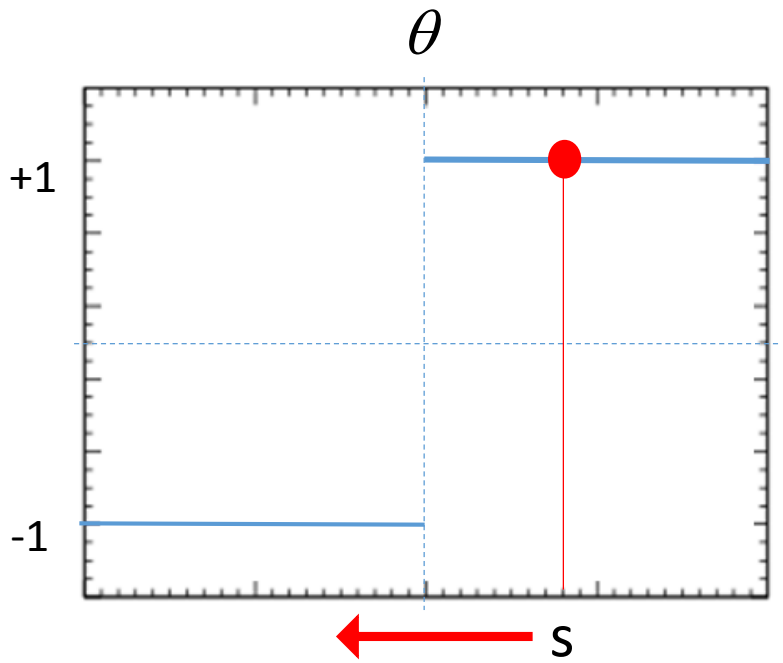


Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Rectified linear unit (ReLU)[11]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[12]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

One layer neural network (perceptron): how to update parameters?

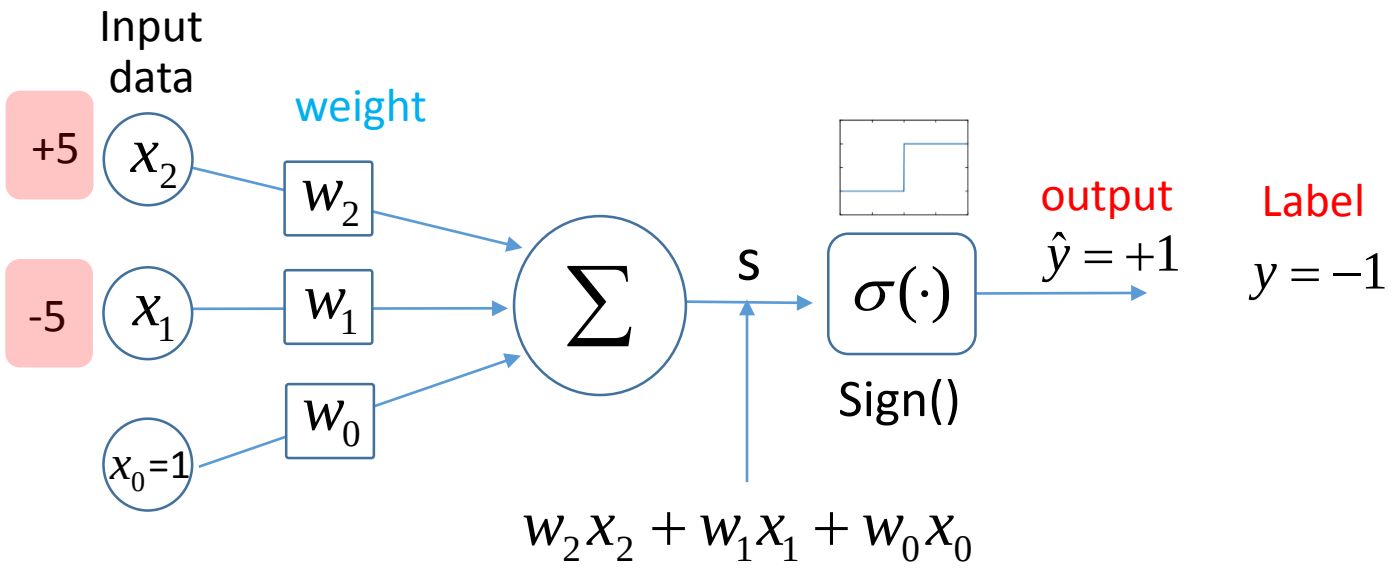
- Assuming that a data point X (*x1, x2*) has a label (-1 : dog)
- Assuming that it is misclassified to be (+1 : panda)
- Weights should be updated.

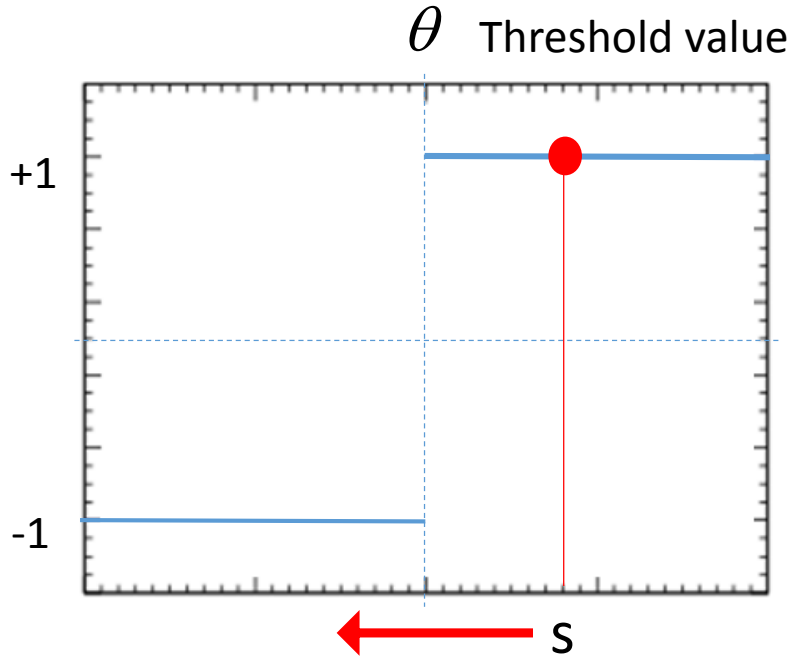




$$\mathrm{w}^{(\mathrm{new})} = \mathrm{w}^{(\mathrm{old})} + \gamma \cdot y_n \cdot \mathrm{x}_\mathrm{n}$$

One layer neural network (perceptron): parameter update

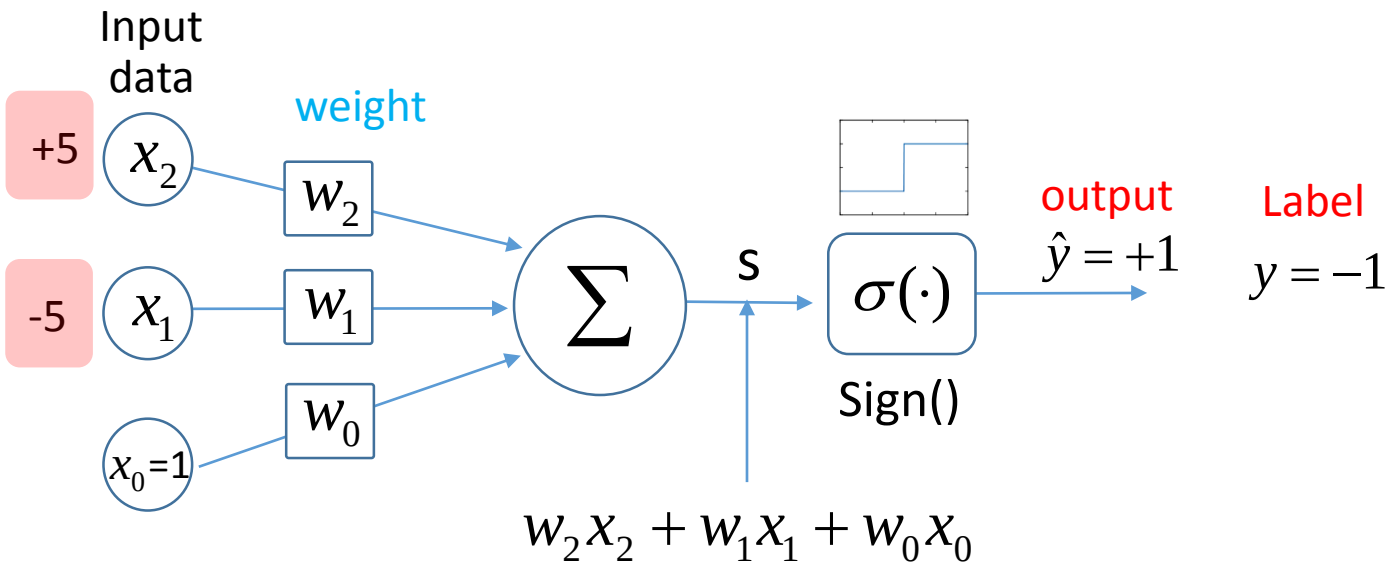


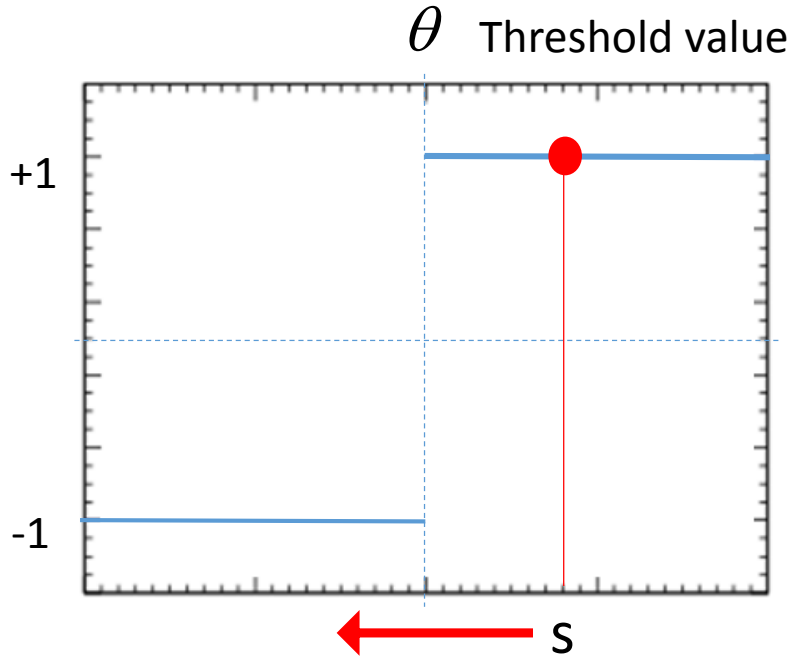


$$w^{(new)} = w^{(old)} + \gamma \cdot y_n \cdot x_n$$

- In the example above, the sum "S" needs to be reduced then how to change the weight values: W_1 and W_2?
  - W_2: make it smaller
  - W_1: make it bigger

One layer neural network (perceptron): parameter update





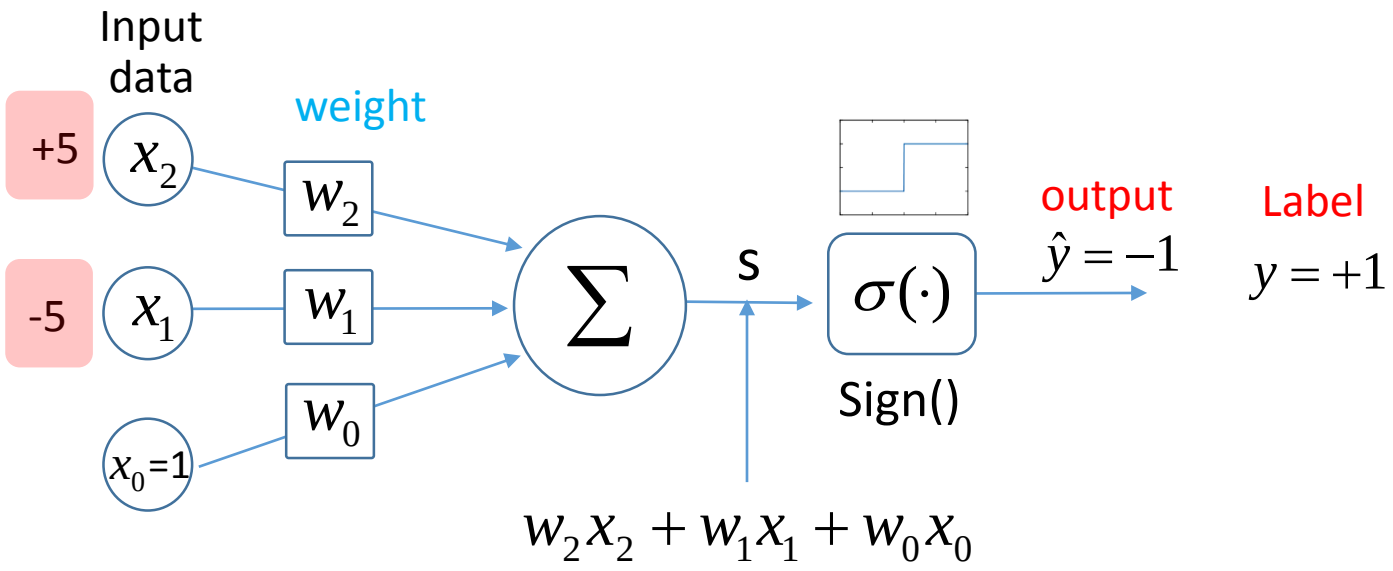
$$\mathrm{w}^{(\mathrm{new})} = \mathrm{w}^{(\mathrm{old})} + \gamma \cdot y_n \cdot \mathrm{x}_\mathrm{n}$$

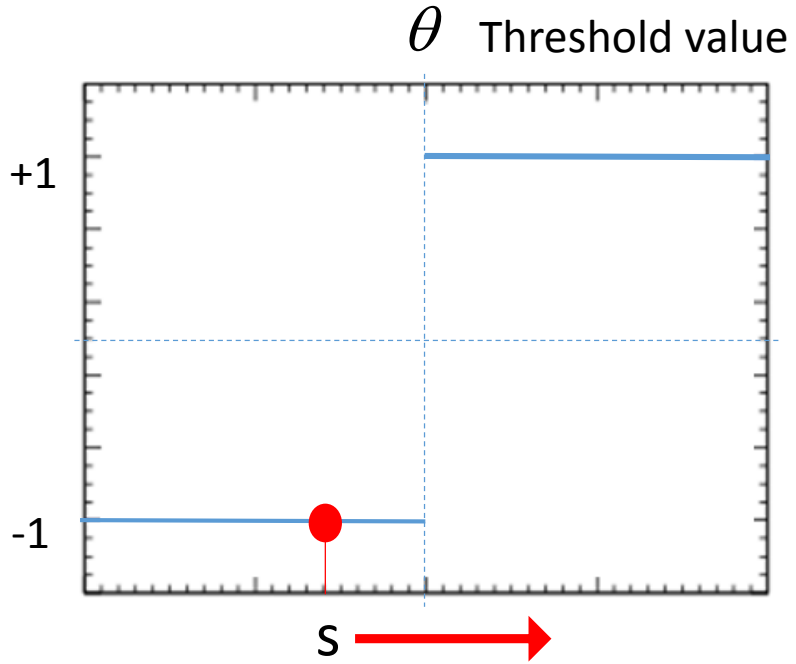
$$w_2^{(\mathrm{new})} = w_2^{(\mathrm{old})} + \gamma \cdot (-1) \cdot (x_2 = 5)$$

$$w_1^{(\mathrm{new})} = w_1^{(\mathrm{old})} + \gamma \cdot (-1) \cdot (x_1 = -5)$$

$$w_0^{(\mathrm{new})} = w_0^{(\mathrm{old})} + \gamma \cdot (-1) \cdot (x_0 = 1)$$

One layer neural network (perceptron): parameter update





$$\mathrm{w}^{(\mathrm{new})} = \mathrm{w}^{(\mathrm{old})} + \gamma \cdot y_n \cdot \mathrm{x}_\mathrm{n}$$

$$w_2^{(\mathrm{new})} = w_2^{(\mathrm{old})} + \gamma \cdot (+1) \cdot (x_2 = 5)$$

$$w_1^{(\mathrm{new})} = w_1^{(\mathrm{old})} + \gamma \cdot (+1) \cdot (x_1 = -5)$$

$$w_0^{(\mathrm{new})} = w_0^{(\mathrm{old})} + \gamma \cdot (-1) \cdot (x_0 = 1)$$

One layer neural network (perceptron): algorithm procedure

1) Random initialization of parameters

$$\text{random} \sim w=\{w_0, w_1, w_2, \ldots, w_d\}$$

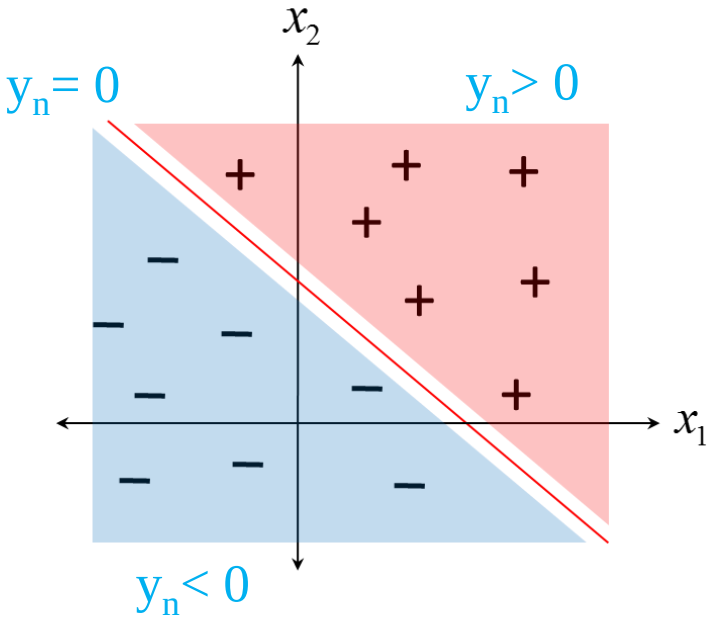
2) Searching misclassified data points using the decision boundary (y_n)

$$y_n = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d$$

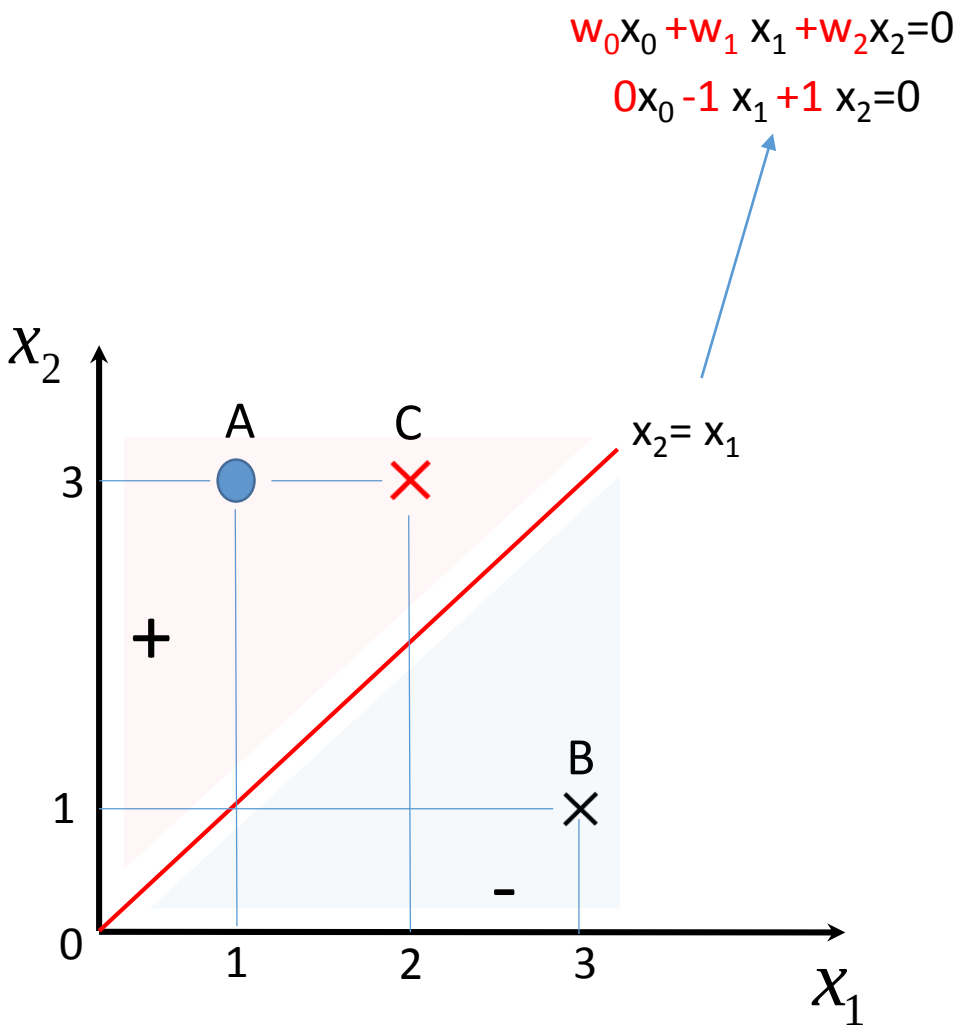
2-1) Updating the parameters using the misclassified data points

$$w^{(new)} = w^{(old)} + \gamma \cdot y_n \cdot x_n$$

2-2) Go to step 2)



One layer neural network (perceptron): example



- Current decision boundary is determined with $w=(0, -1, +1)$
- Data point x_C $(1, 2, 3)$ is misclassified
- Learning rate: $\gamma = 0.1$
- Let's update the parameter

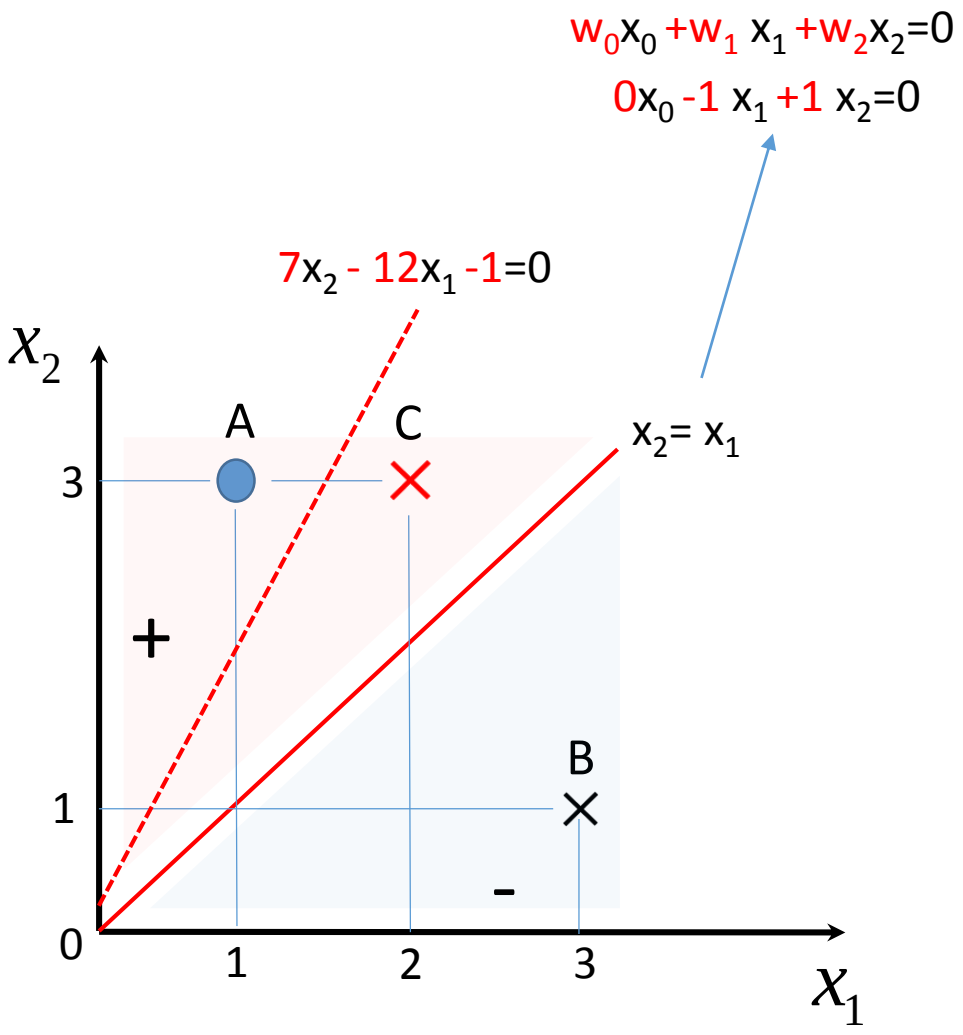
$w=(0, -1, +1)$ $\quad$ $x_c=(1, 2, 3)$

$$\mathrm{w}^{(\mathrm{new})} = \mathrm{w}^{(\mathrm{old})} + \gamma \cdot y_n \cdot \mathrm{x}_\mathrm{n}$$

$$= (0,-1,+1) + (0.1)(-1)(1,2,3)$$

$$= (-0.1, \quad -1.2, \quad 0.7)$$

One layer neural network (perceptron): example



- Current decision boundary is determined with w=(0, -1, +1)
- Data point x_C (1, 2, 3) is misclassified
- Learning rate: $\gamma = 0.1$
- Let's update the parameter

$w=(0, -1, +1)$ $\quad$ $x_c=(1, 2, 3)$

$$\mathrm{w}^{(\mathrm{new})} = \mathrm{w}^{(\mathrm{old})} + \gamma \cdot y_n \cdot \mathrm{x}_\mathrm{n}$$

$$= (0,-1,+1) + (0.1)(-1)(1,2,3)$$

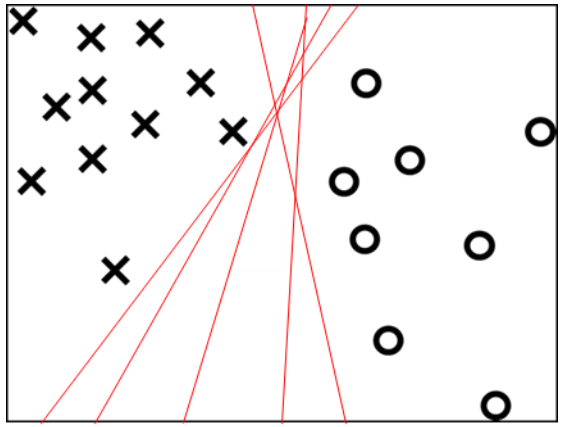
$$= (-0.1, \quad -1.2, \quad 0.7)$$

- New decision boundary is

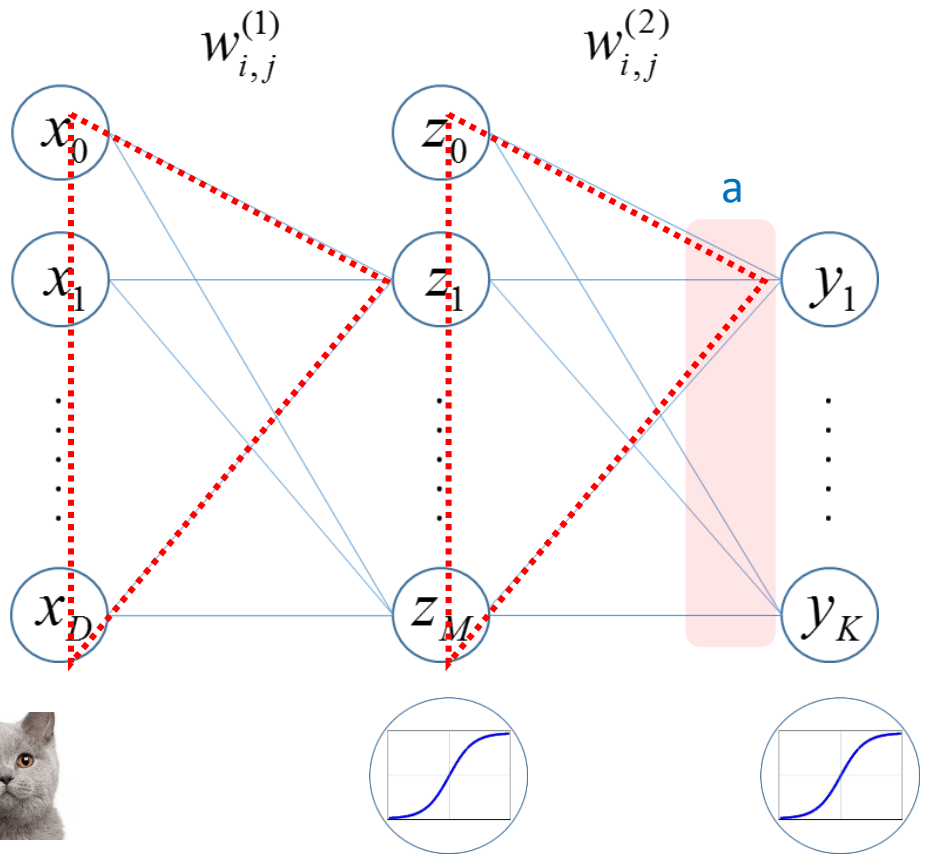
$$7x_2 - 12x_1 - 1 = 0$$

One layer neural network (perceptron): convergence theorem

- The perceptron algorithm is guaranteed to find an exact solution within a finite number of iteration if given data set is linearly separable.
  - Slow convergence: cannot tell its feasibility until it's convergence
  - Does not converge if there is not any solution
  - Many solutions exist: converge to one depending on an initial and the order of data feeding



A neural network model



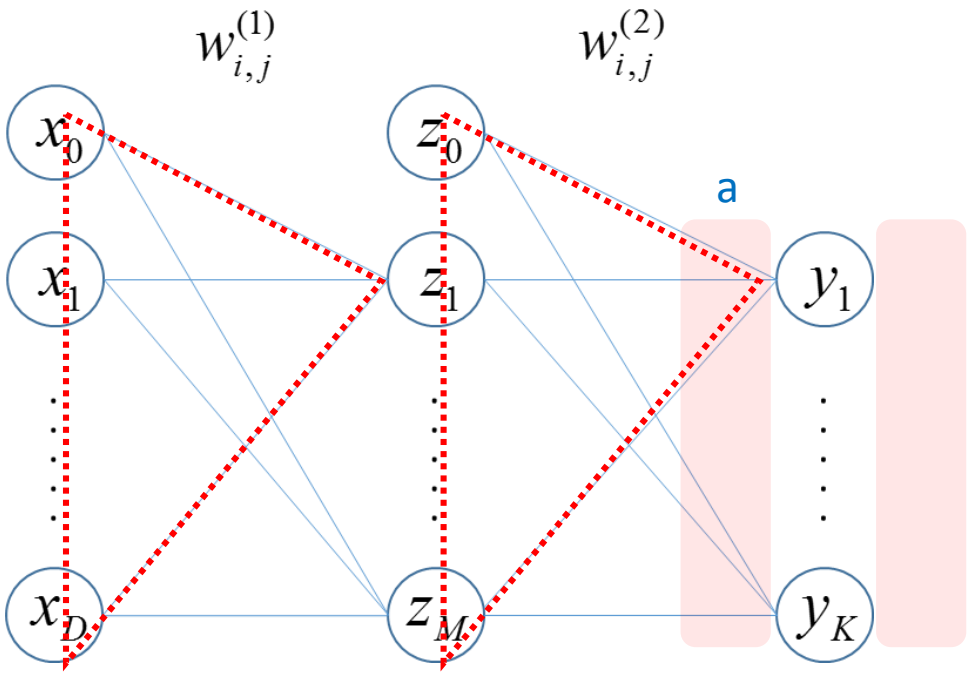
a

output	
-5	frog
-1	bird
1	dog
5	cat

$$w_{1,1}^{(1)}x_1 + w_{2,1}^{(1)}x_2 + w_{0,1}^{(1)}x_0$$

$$w_{1,1}^{(2)}z_1 + w_{2,1}^{(2)}z_2 + w_{0,1}^{(2)}z_0$$

A neural network model



a

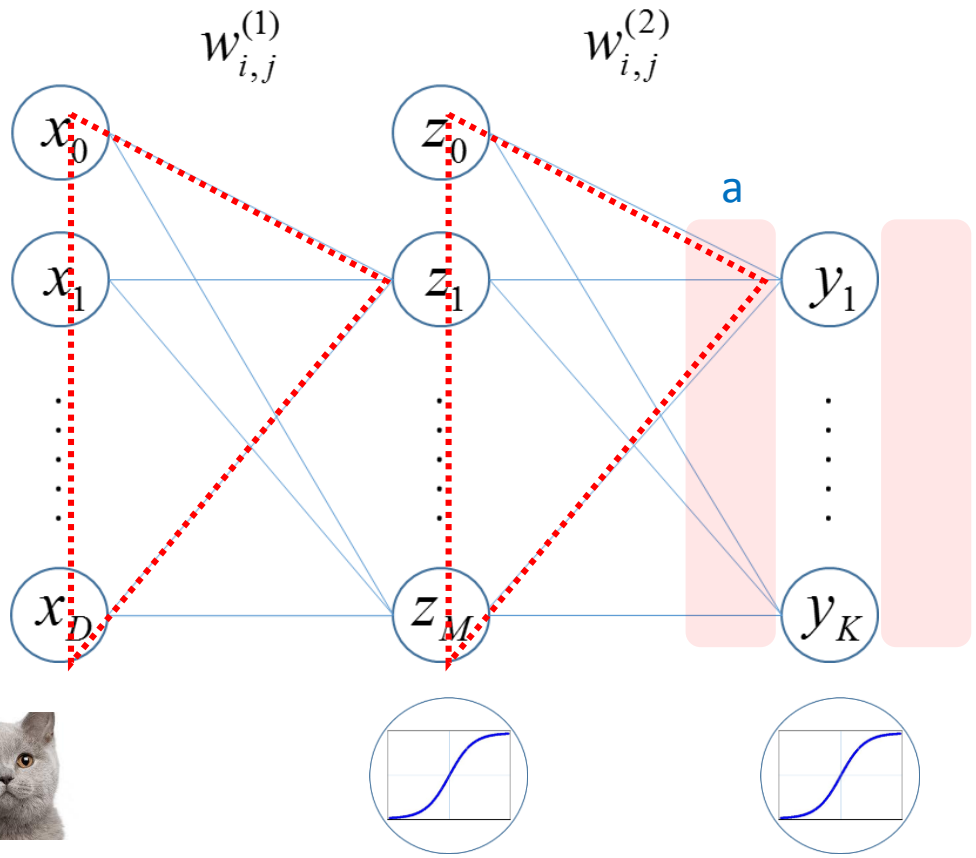
output		Sigmoid
-5	frog	0.00669
-1	bird	0.26894
1	dog	0.73106
5	cat	0.99331

$$\sigma_1(a) = \frac{1}{1+e^{-a}}$$

$$w_{1,1}^{(1)} x_1 + w_{2,1}^{(1)} x_2 + w_{0,1}^{(1)} x_0$$

$$w_{1,1}^{(2)} z_1 + w_{2,1}^{(2)} z_2 + w_{0,1}^{(2)} z_0$$

A neural network model



a

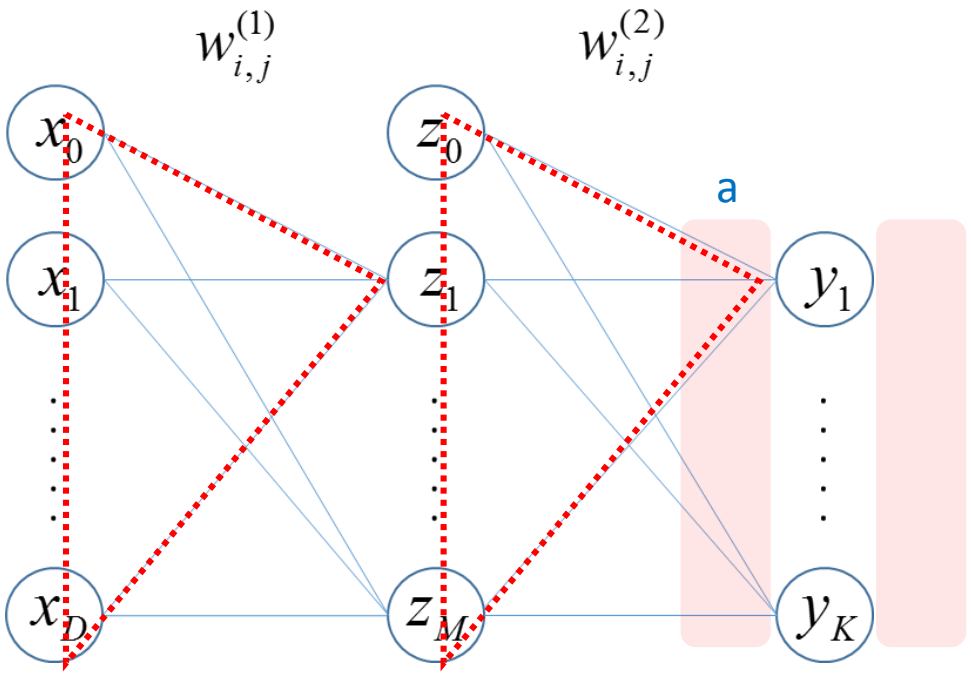
output		Sigmoid	Normalization
-5	frog	0.00669	0.00334
-1	bird	0.26894	0.13447
1	dog	0.73106	0.36553
5	cat	0.99331	0.49666

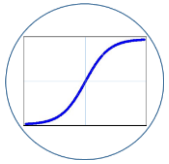
$$\sigma_1(a) = \frac{1}{1+e^{-a}}$$

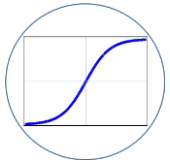
$$w_{1,1}^{(1)} x_1 + w_{2,1}^{(1)} x_2 + w_{0,1}^{(1)} x_0$$

$$w_{1,1}^{(2)} z_1 + w_{2,1}^{(2)} z_2 + w_{0,1}^{(2)} z_0$$

A neural network model: softmax vs normalization







$$w_{1,1}^{(1)}x_1 + w_{2,1}^{(1)}x_2 + w_{0,1}^{(1)}x_0$$

$$w_{1,1}^{(2)}z_1 + w_{2,1}^{(2)}z_2 + w_{0,1}^{(2)}z_0$$

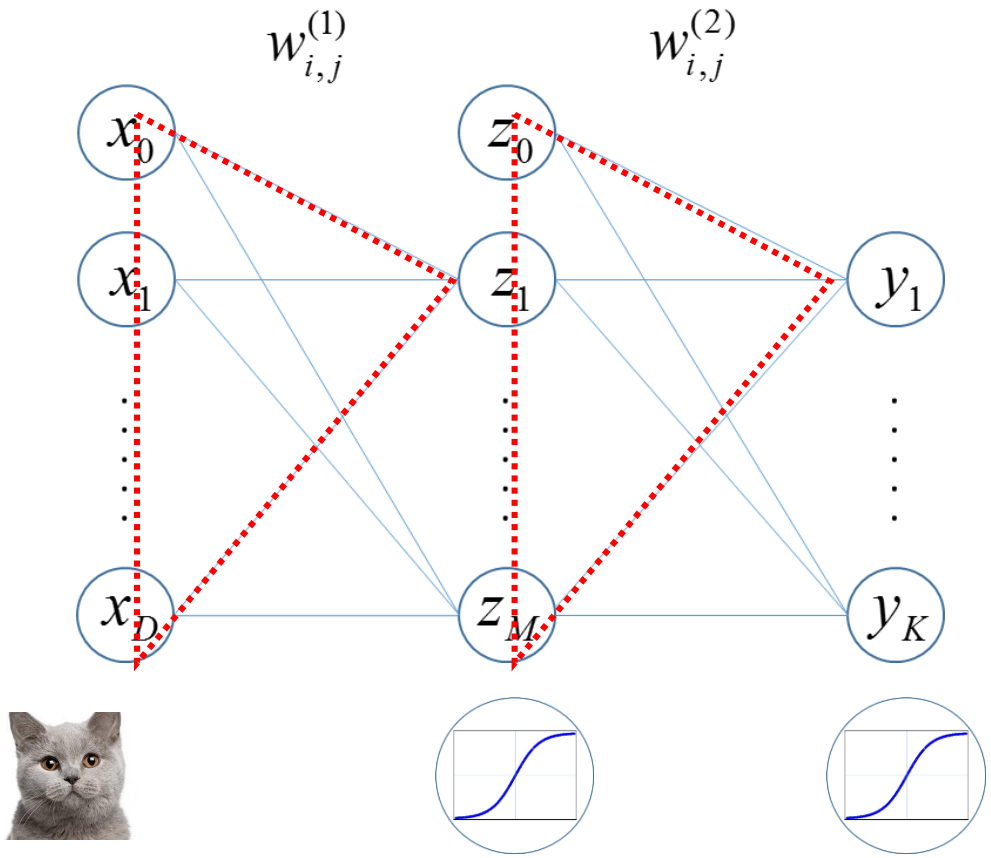
a

output		Sigmoid	Normalization	Softmax
-5	frog	0.00669	0.00334	0.00004
-1	bird	0.26894	0.13447	0.00243
1	dog	0.73106	0.36553	0.01794
5	cat	0.99331	0.49666	0.97959

$$\sigma_1(a) = \frac{1}{1+e^{-a}}$$

$$\sigma_2(a_j) = \frac{e^{a_j}}{\sum_i e^{a_i}}$$

A neural network model: cross entropy with softmax



$$w_{1,1}^{(1)}x_1 + w_{2,1}^{(1)}x_2 + w_{0,1}^{(1)}x_0$$

$$w_{1,1}^{(2)}z_1 + w_{2,1}^{(2)}z_2 + w_{0,1}^{(2)}z_0$$

output		Sigmoid	Normalization	y Softmax
-5	frog	0.00669	0.00334	0.00004
-1	bird	0.26894	0.13447	0.00243
1	dog	0.73106	0.36553	0.01794
5	cat	0.99331	0.49666	0.97959

t

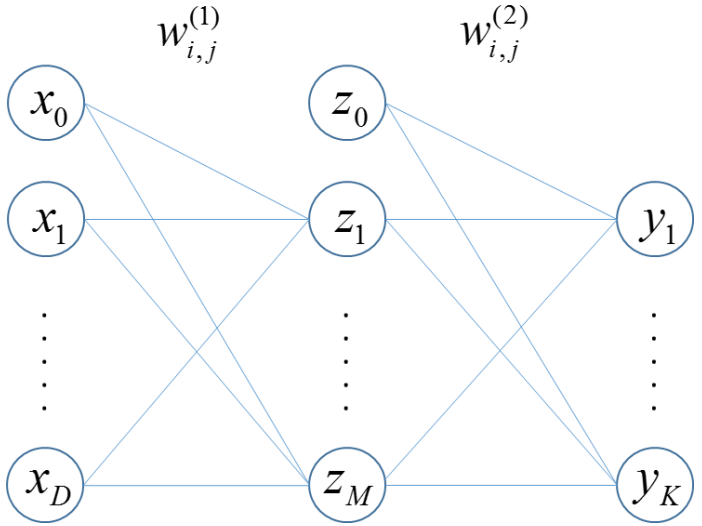
Label
0
0
0
1

$$H(\mathrm{y}) = -\sum_{\mathrm{i}} \mathrm{t_i} \log(y_i) = 0.020621$$

Operation with a toy example of backpropagation

Overview of the operation

Input data (X)

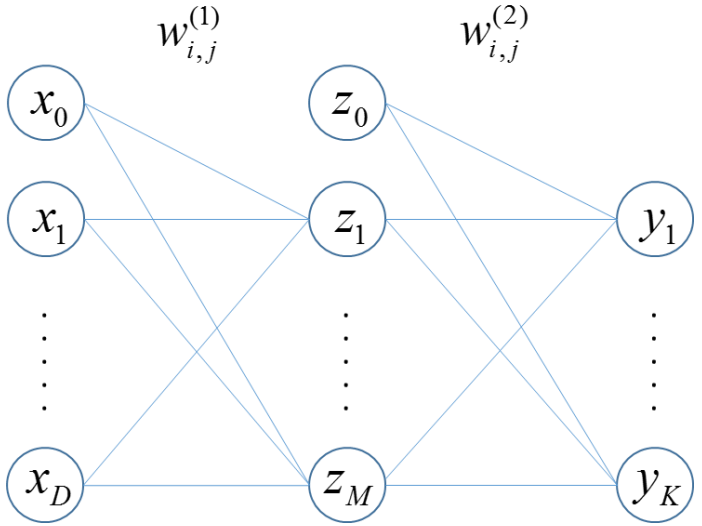


Label (t)

Overview of the operation

forwarding

Input data (X)

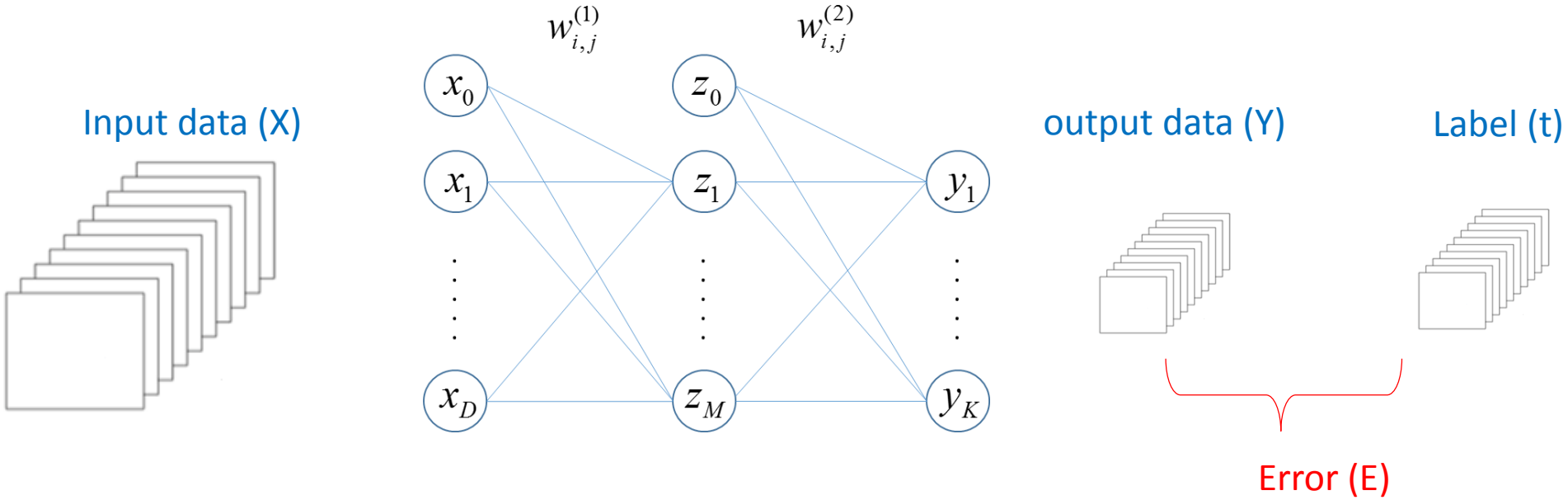


output data (Y)

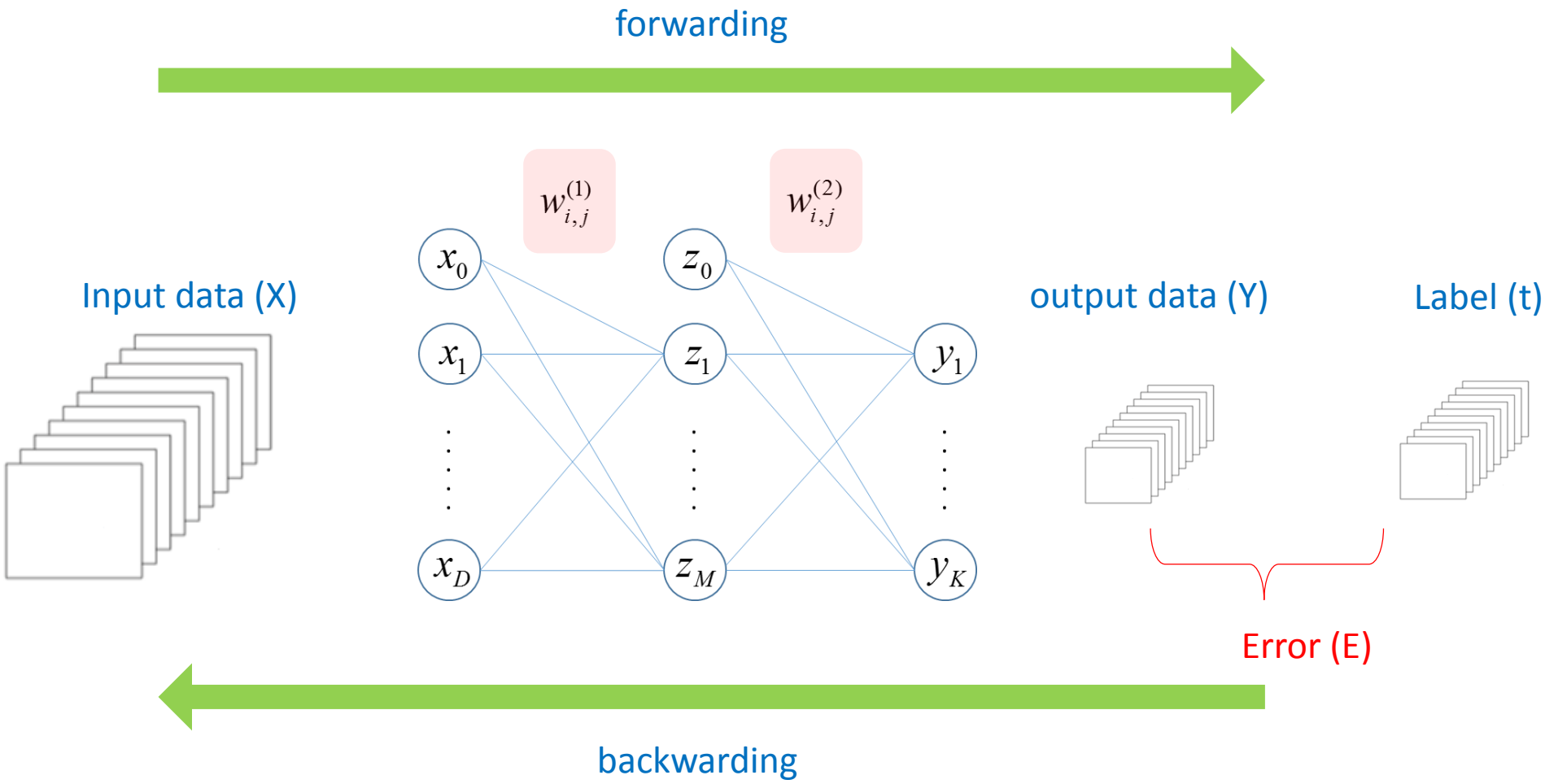
Label (t)

Overview of the operation

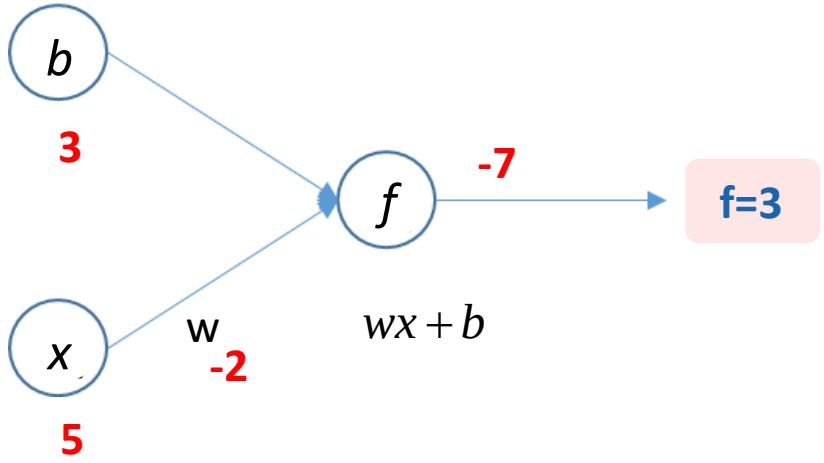
forwarding



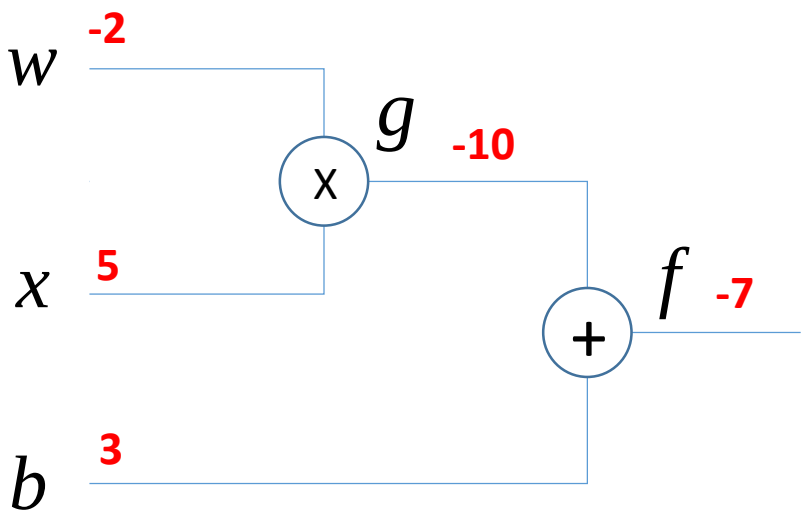
Overview of the operation

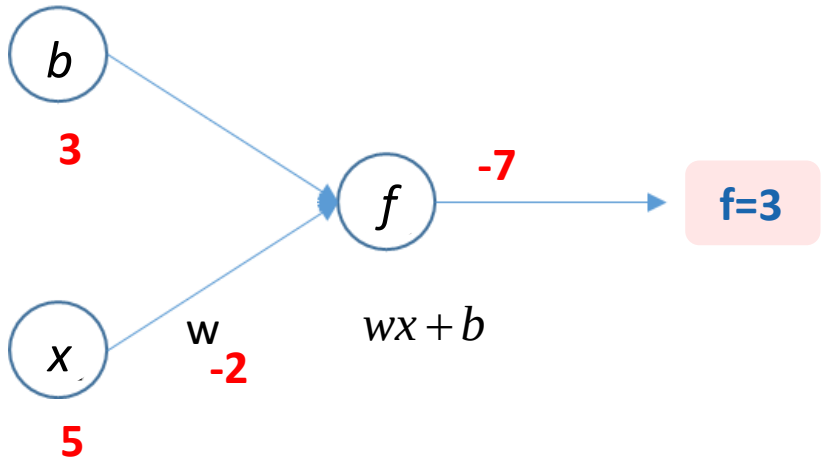


Backpropagation: a toy example

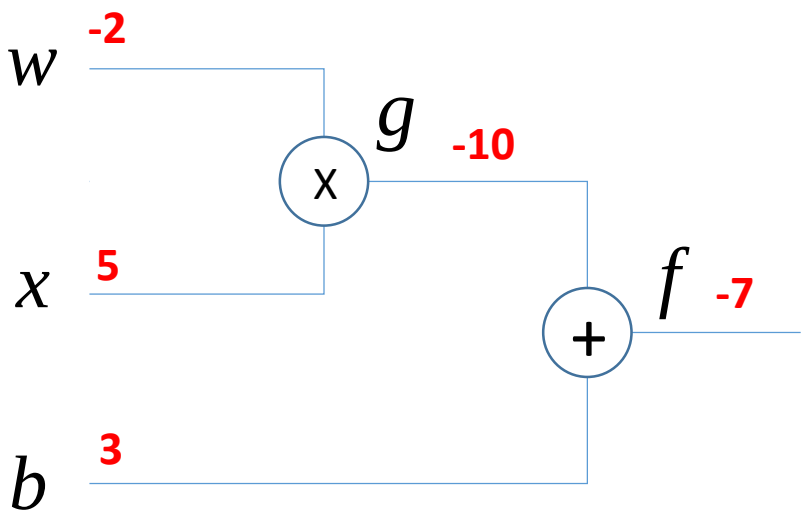


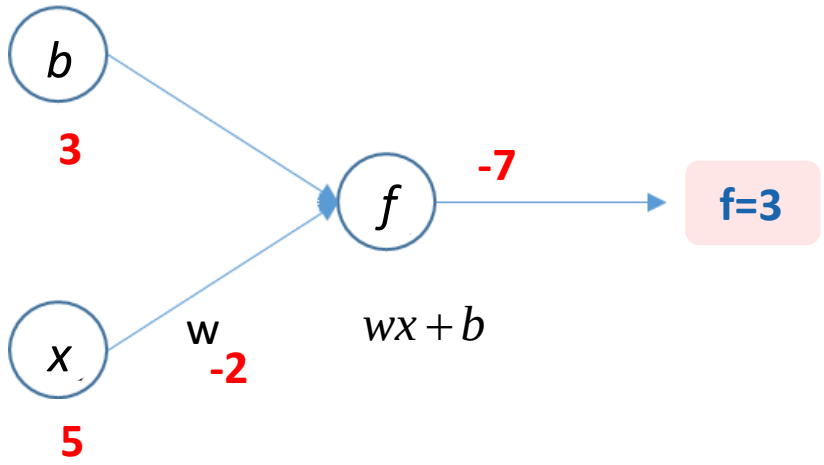
Backpropagation: a toy example





Backpropagation: a toy example

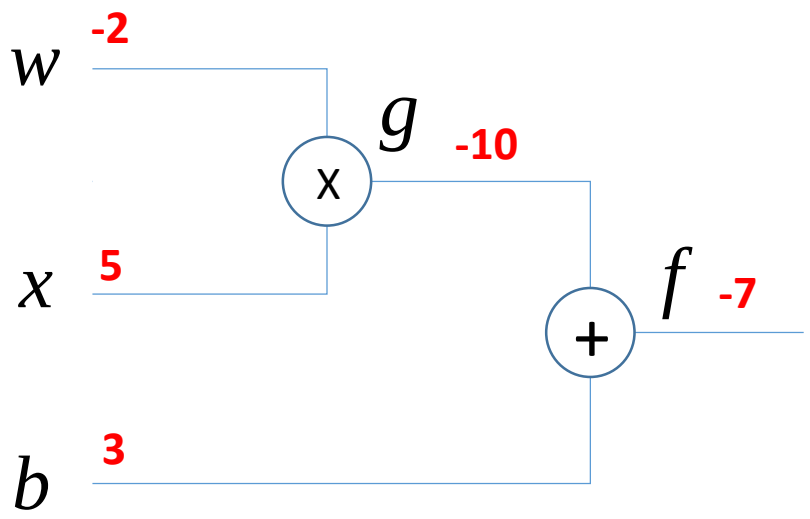




$$f = g + b$$

$$g = wx$$

Backpropagation: a toy example



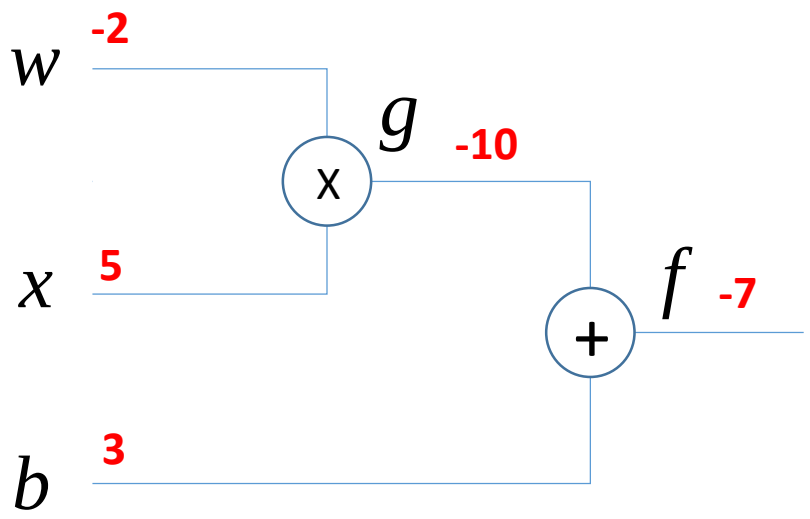
$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w} = x = 5$$

When "w" is changed by 1 unit, it will change the value of "f" by 5 unit.

$$f = g + b$$

$$g = wx$$

Backpropagation: a toy example



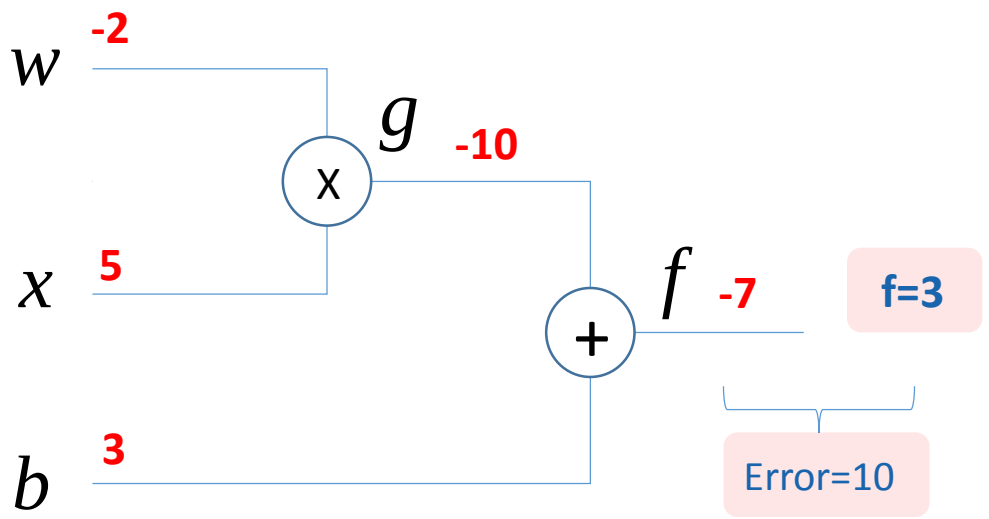
$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w} = x = 5$$

$$\frac{\partial f}{\partial b} = 1$$

$$f = g + b$$

$$g = wx$$

Backpropagation: a toy example



$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w} = x = 5$$

$$\frac{\partial f}{\partial b} = 1$$

$$f = g + b$$

$$g = wx$$

- Assuming that the value of f should be "3".
- How to update variables which you are interested?

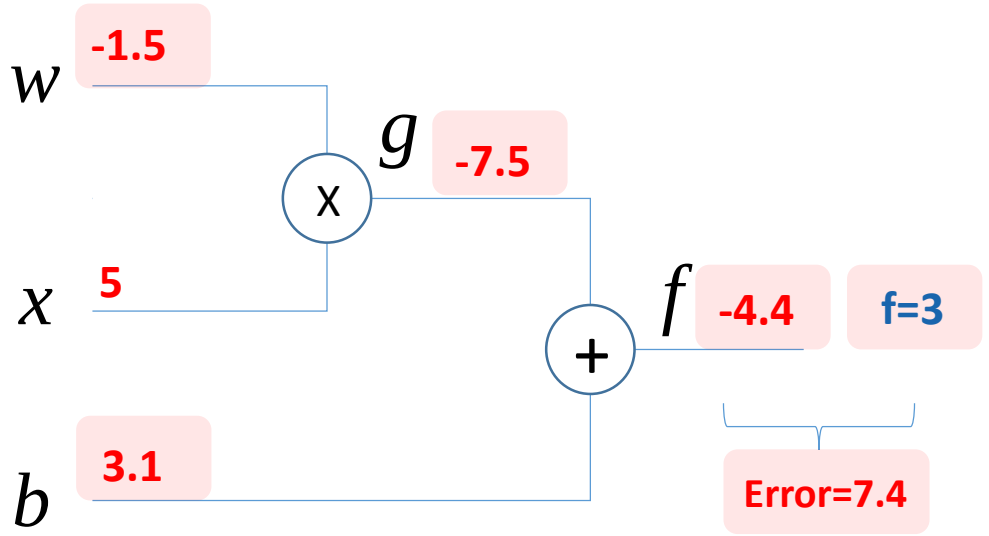
$$W_{new} = W_{old} + \eta \frac{\partial f}{\partial w_i}$$

$$b_{new} = b_{old} + \eta \frac{\partial f}{\partial b}$$

$$W_{new} = -2 + 0.1 \times 5 = -1.5$$

$$b_{new} = 3 + 0.1 \times 1 = 3.1$$

Backpropagation: a toy example



$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w} = x = 5$$

$$\frac{\partial f}{\partial b} = 1$$

$$f = g + b$$

$$g = wx$$

- Assuming that the value of f should be "3".
- How to update variables which you are interested?

$$W_{new} = W_{old} + \eta \frac{\partial f}{\partial w_i}$$

$$b_{new} = b_{old} + \eta \frac{\partial f}{\partial b}$$

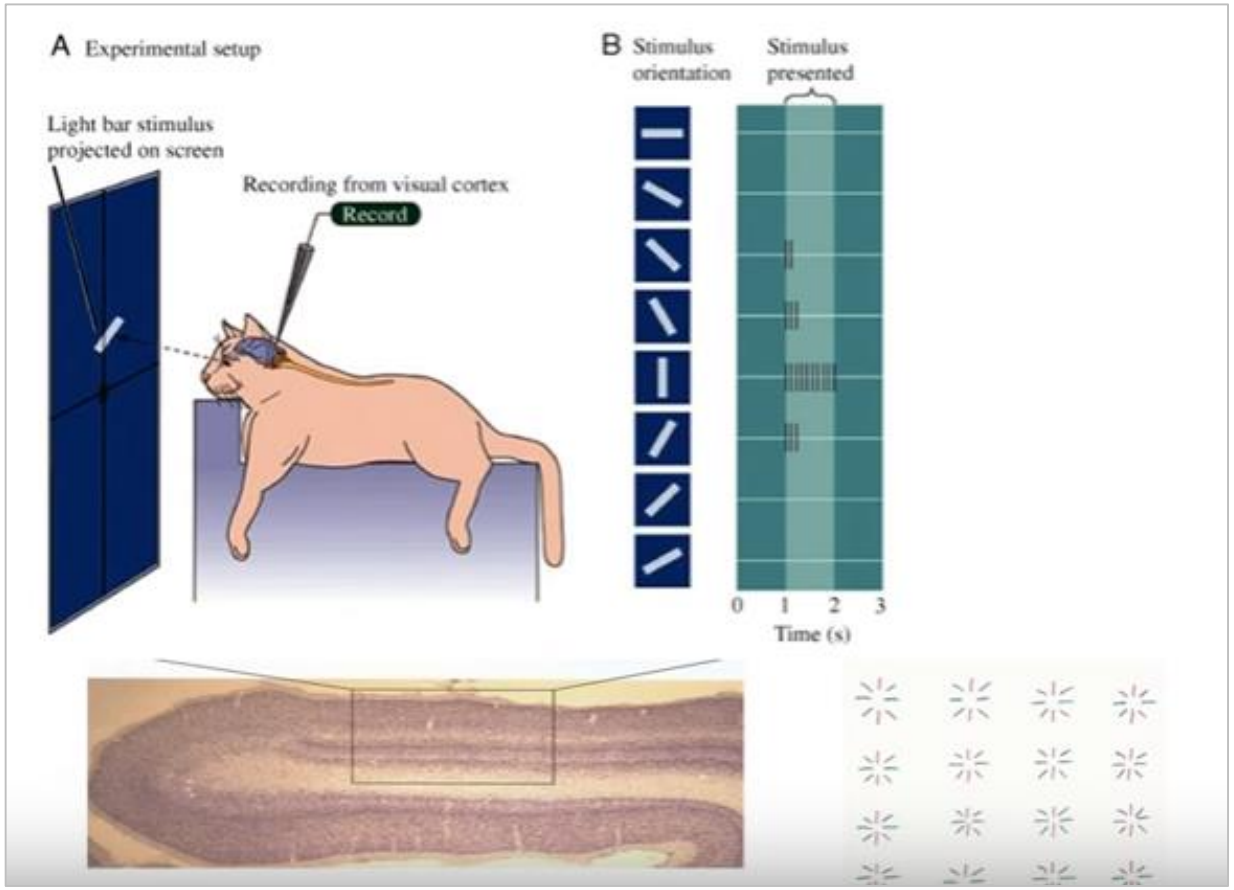
$$W_{new} = -2 + 0.1 \times 5 = -1.5$$

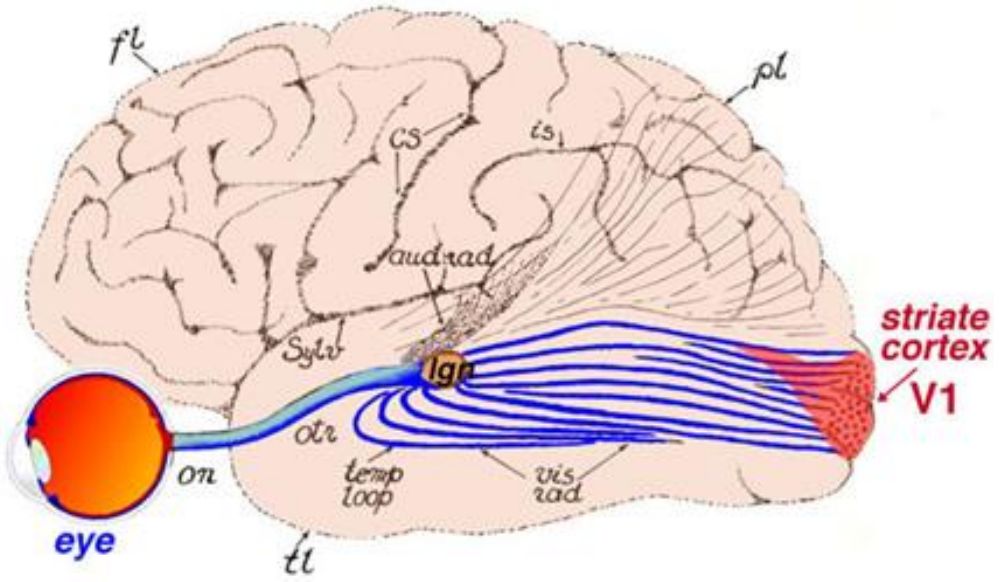
$$b_{new} = 3 + 0.1 \times 1 = 3.1$$

Convolutional Neural Networks (CNN)

History of CNN

- Hubel and Wiesel in 1962





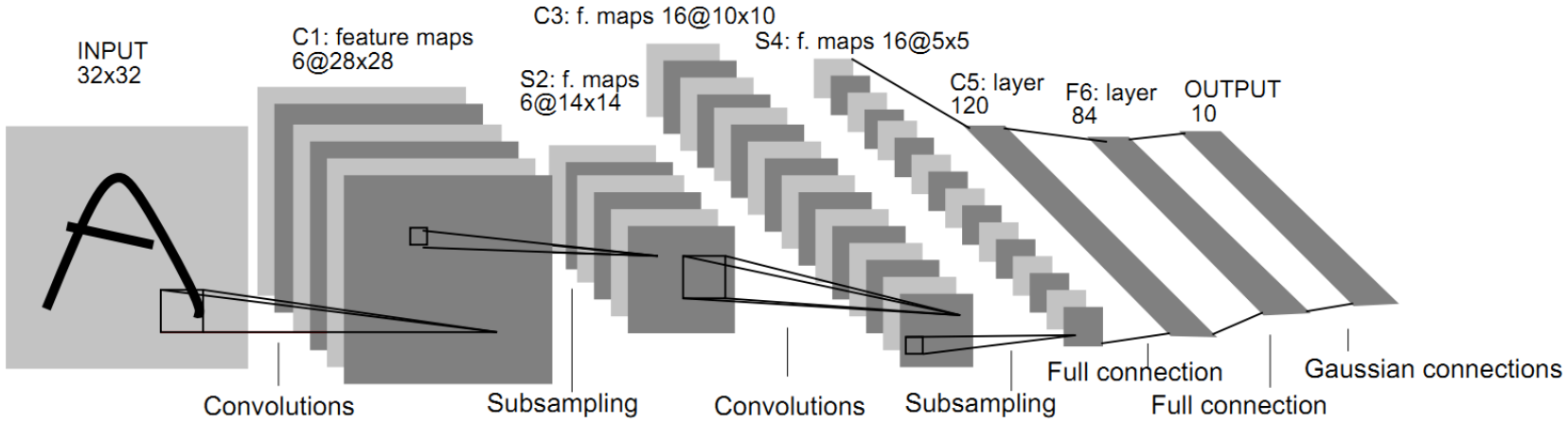
https://www.youtube.com/watch?v=v20-E_2bT2c

History of CNN

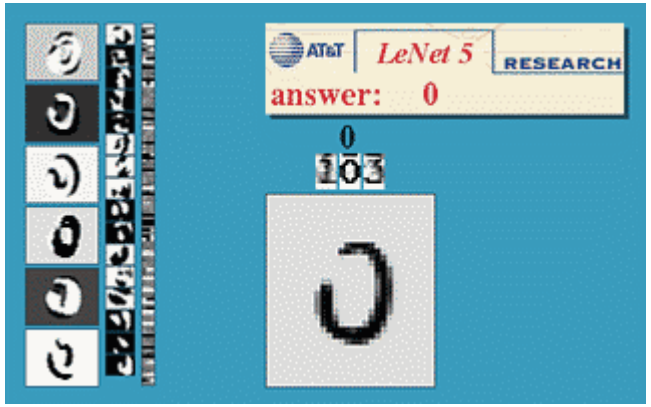
https://www.youtube.com/watch?v=IOHayh06LJ4

History of CNN – 1995

- LeCun et al, “Convolutional Networks for Images, Speech, and Time-Series”
  - http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf



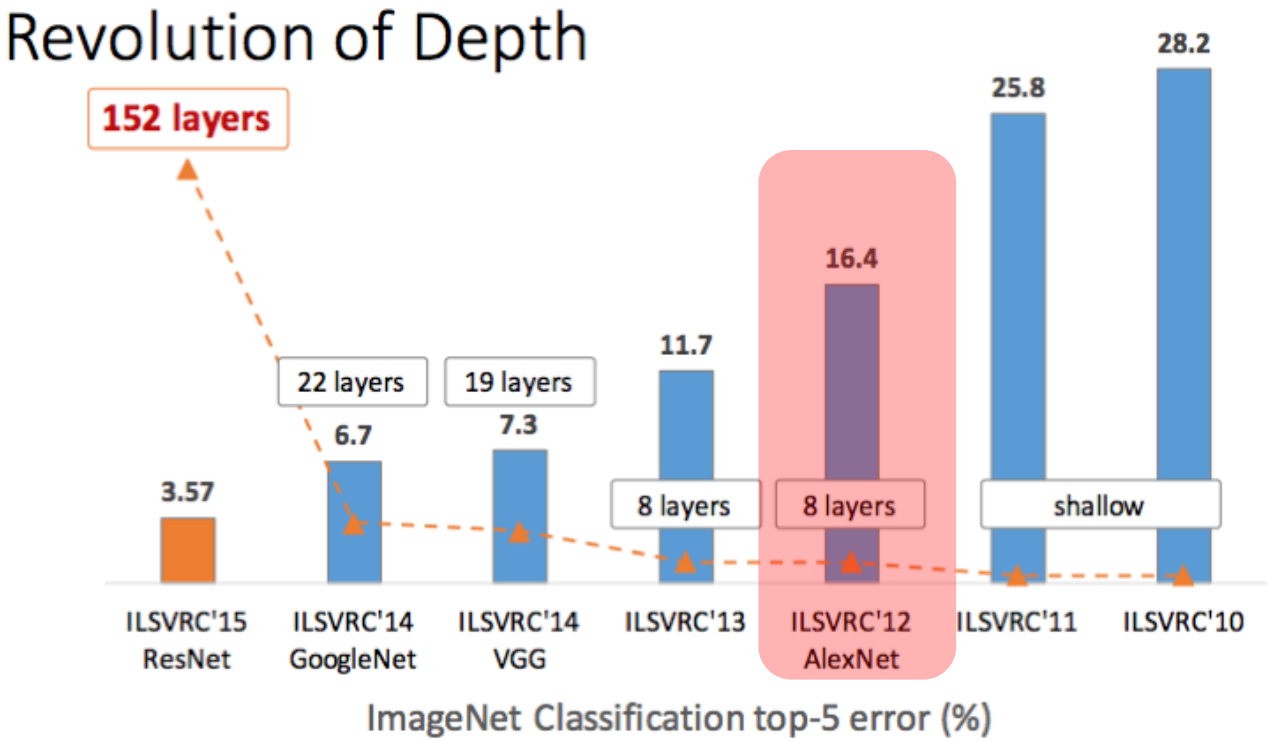
Lecun Yann
Director of AI Research, Facebook

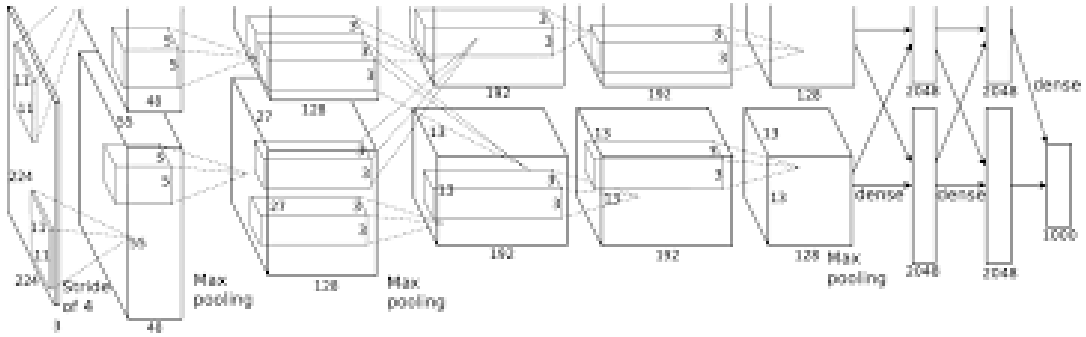


http://yann.lecun.com/

History of CNN – 2012

- Alex Krizhevsky et al, "*ImageNet Classification with Deep Convolutional Neural Networks*" (2012)
  - Win the imageNet competition: annual Olympics of computer vision with astounding results compared to previously existing approaches (26% to 15%).

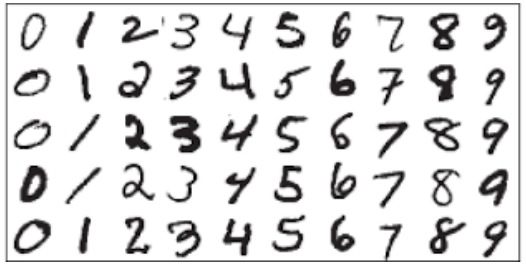


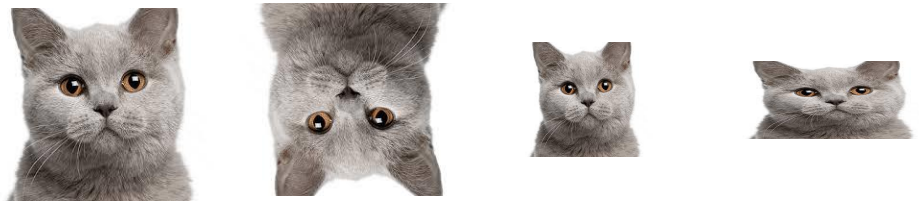


https://medium.com/@Lidinwise/the-revolution-of-depth-facf174924f5

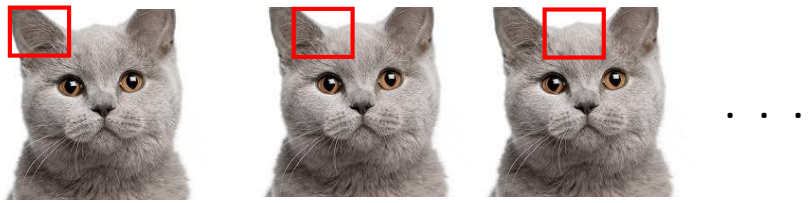
Why Convolutional Neural Networks (CNN)?

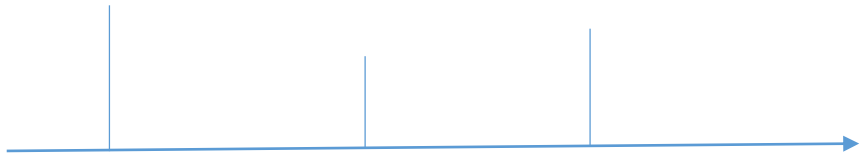
- Invariance to rotation, transformation, size, etc.





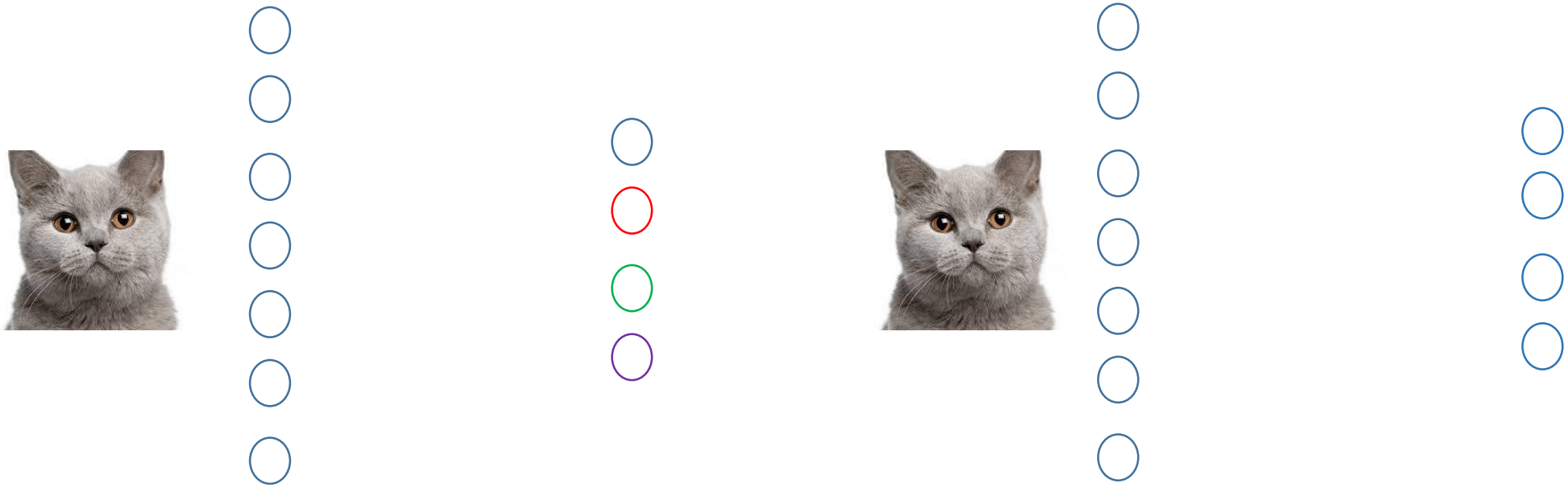
- Three architectural ideas of CNN
  1. Local receptive fields
  2. Weight sharing
  3. Spatial or temporal subsampling





An idea of CNN

- A node takes responsibility for a portion of input data.

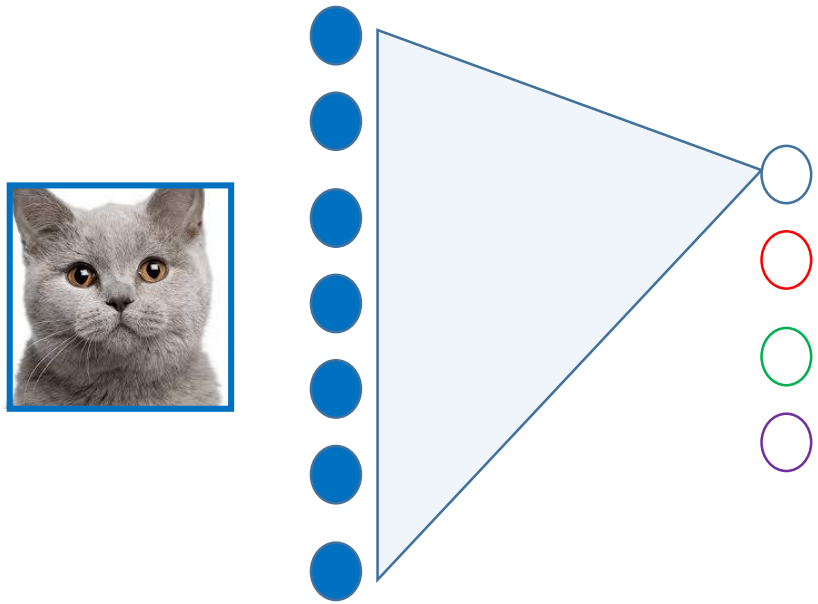


Fully Connected Layer

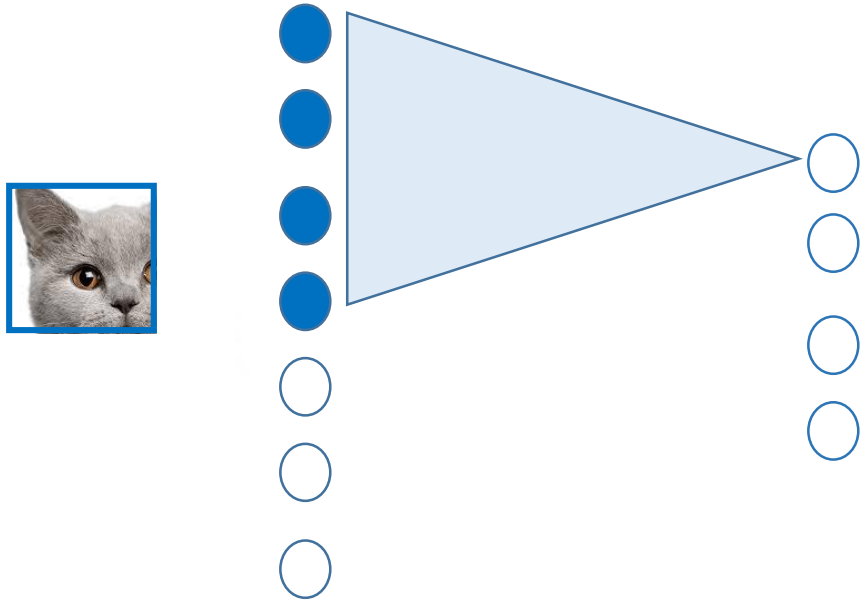
Convolution Layer

An idea of CNN

- A node takes responsibility for a portion of input data.



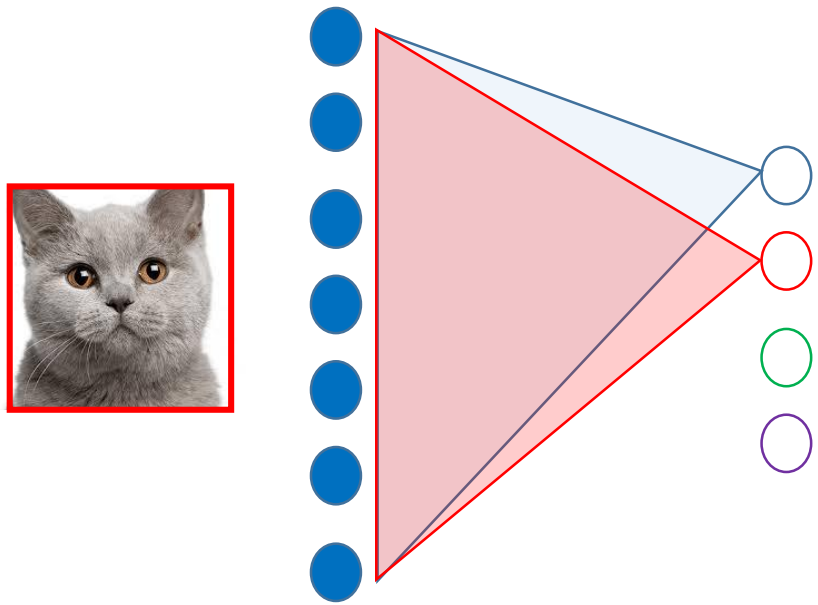
Fully Connected Layer



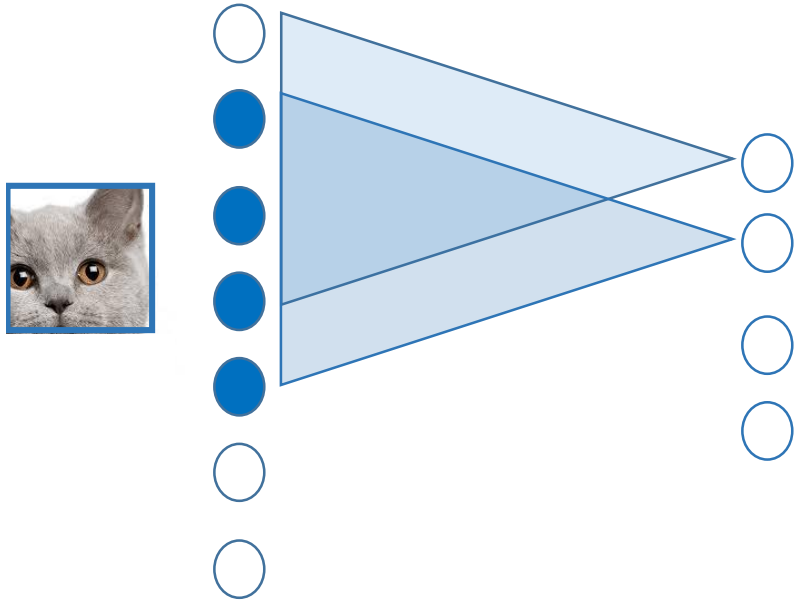
Convolution Layer

An idea of CNN

- A node takes responsibility for a portion of input data.



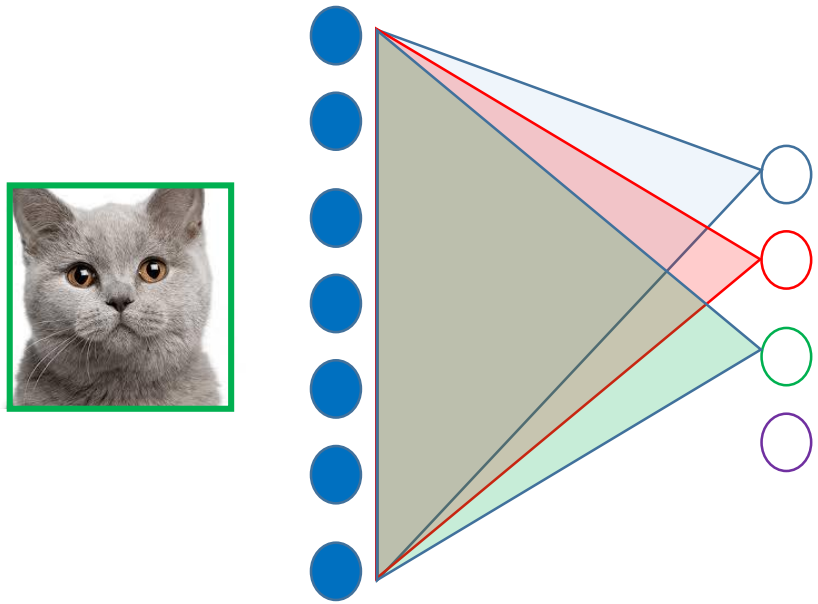
Fully Connected Layer



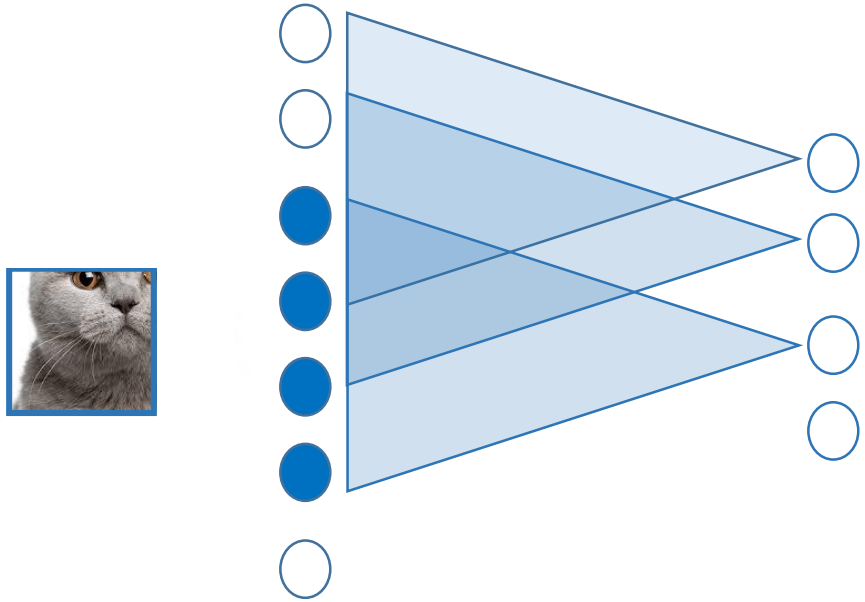
Convolution Layer

An idea of CNN

- A node takes responsibility for a portion of input data.



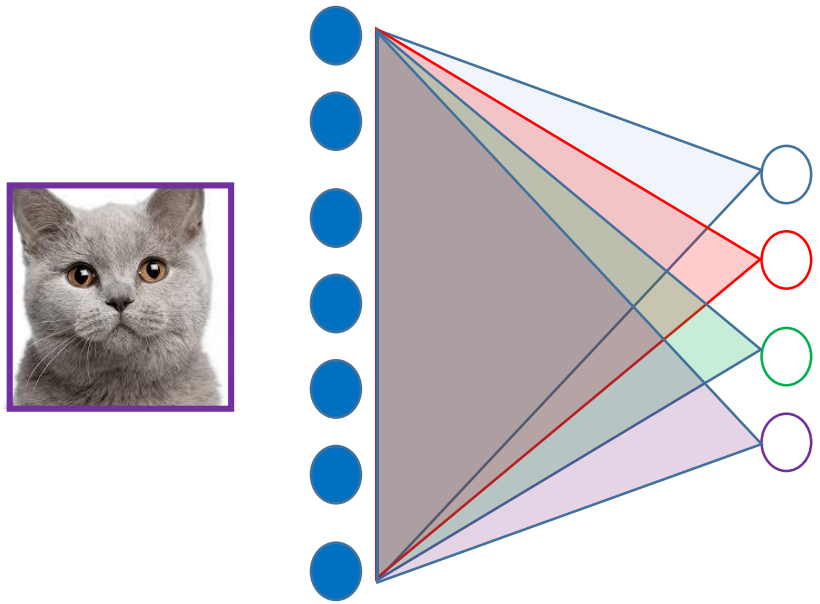
Fully Connected Layer



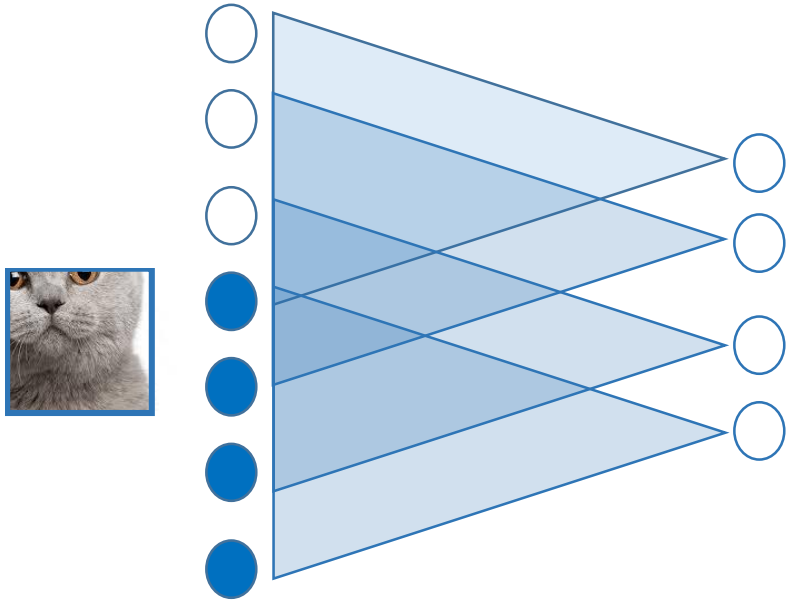
Convolution Layer

An idea of CNN

- A node takes responsibility for a portion of input data.



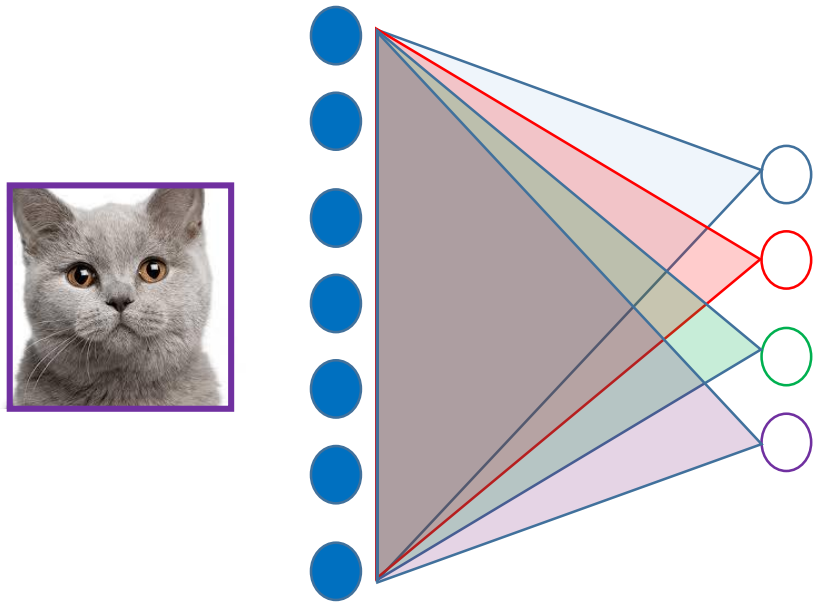
Fully Connected Layer



Convolution Layer

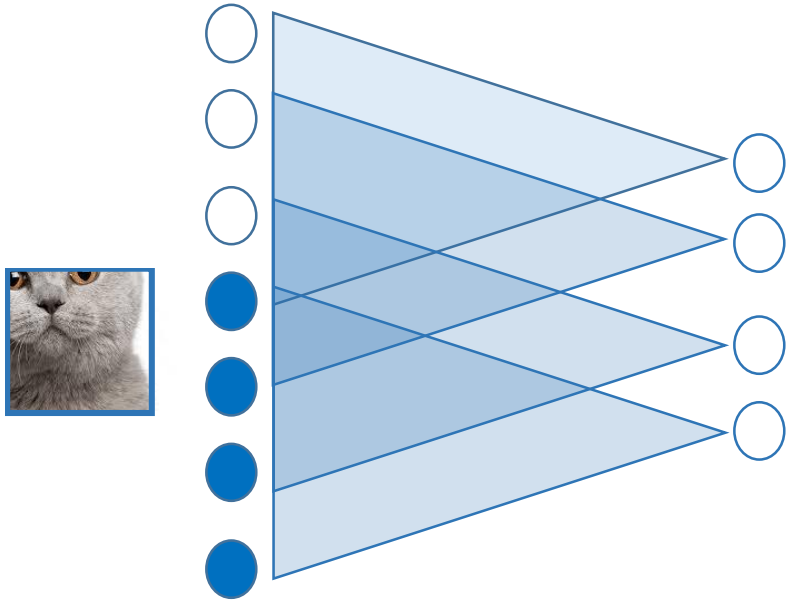
An idea of CNN

- A node takes responsibility for a portion of input data.



Fully connected Layer

- 4 x 7 = 28 weights

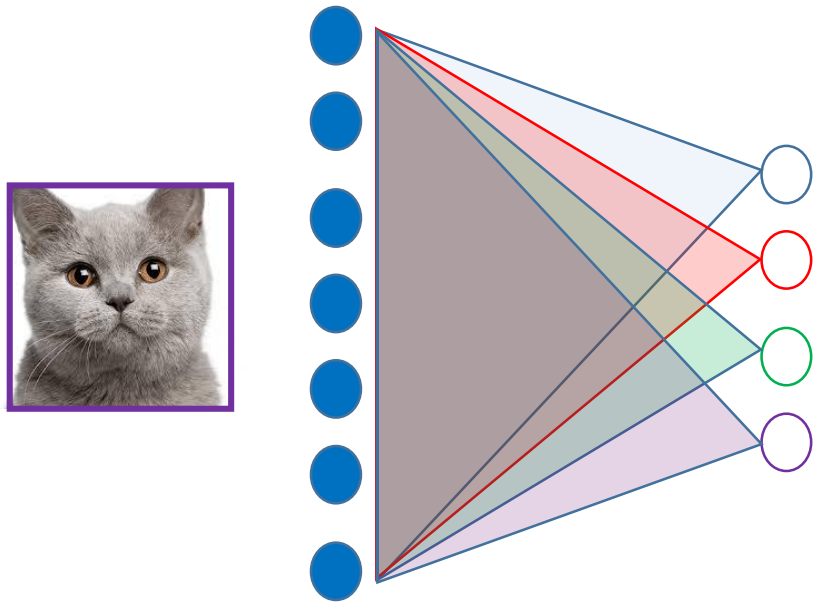


Convolutional Layer

- 4 weights

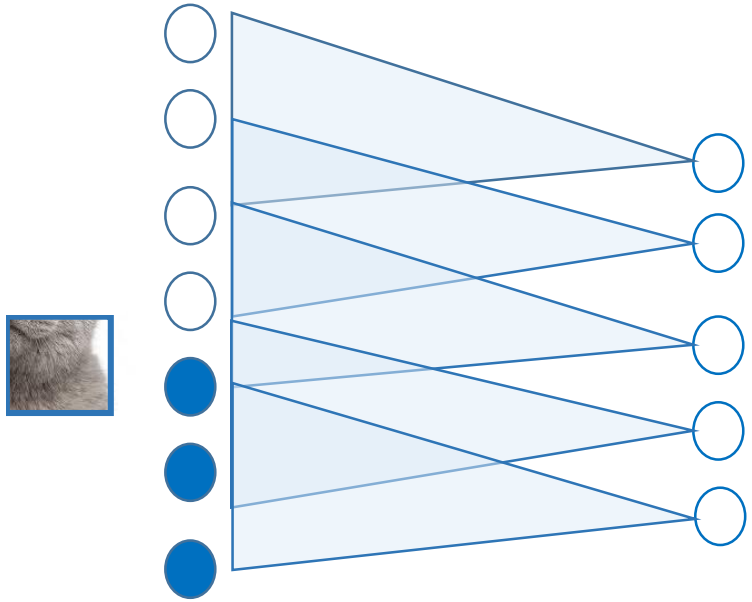
An idea of CNN

- A node takes responsibility for a portion of input data.



Fully connected Layer

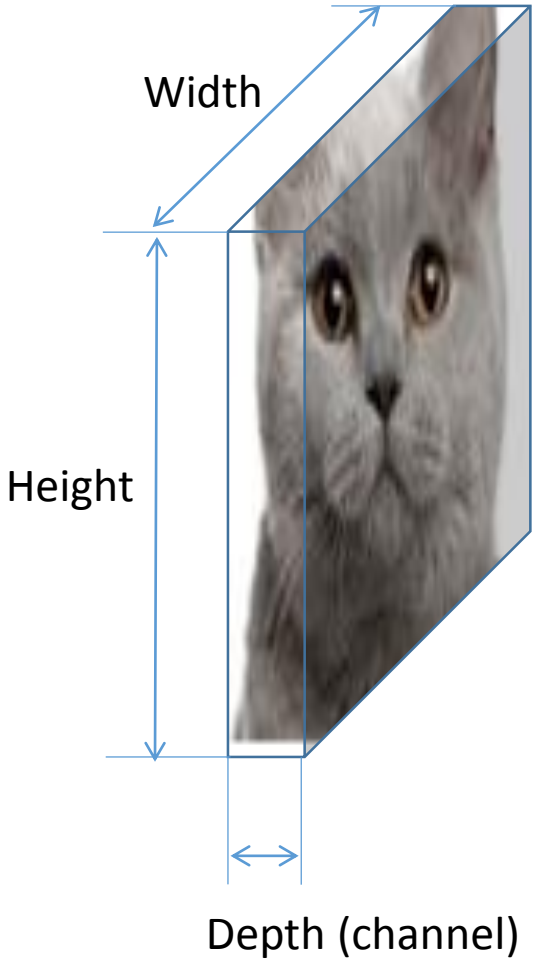
- 4 x 7 = 28 weights



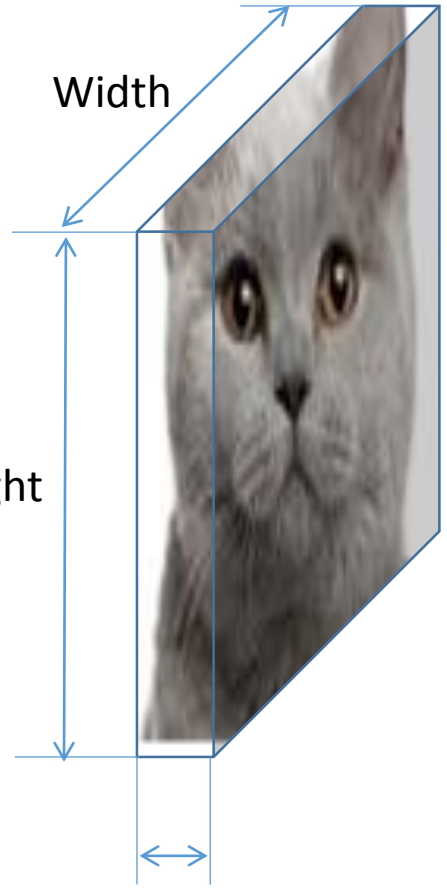
Convolutional Layer

- 3 weights

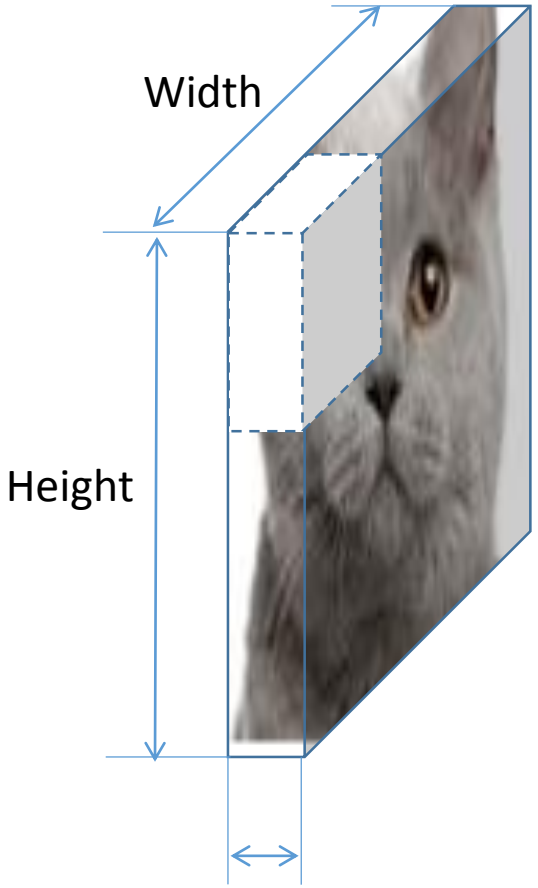
2-D representation vs 1-D representation

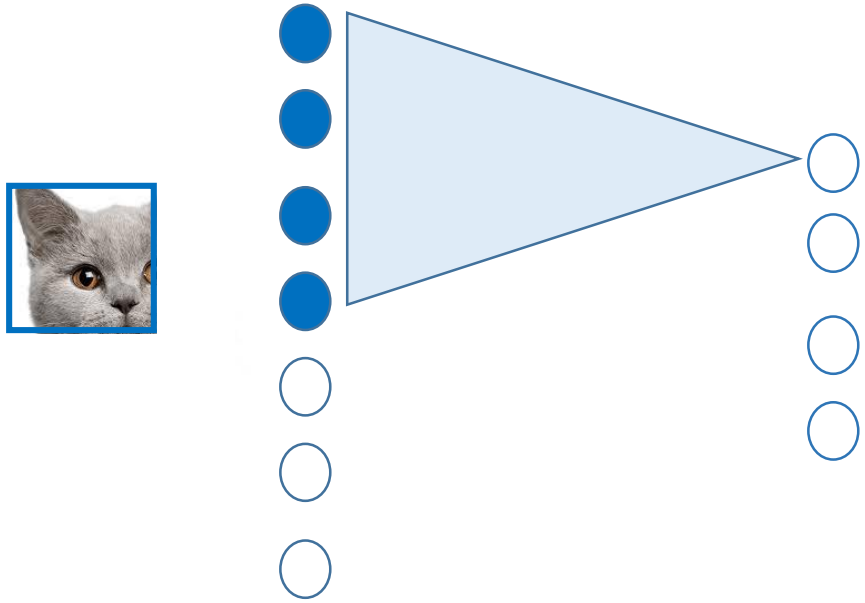


2-D representation vs 1-D representation

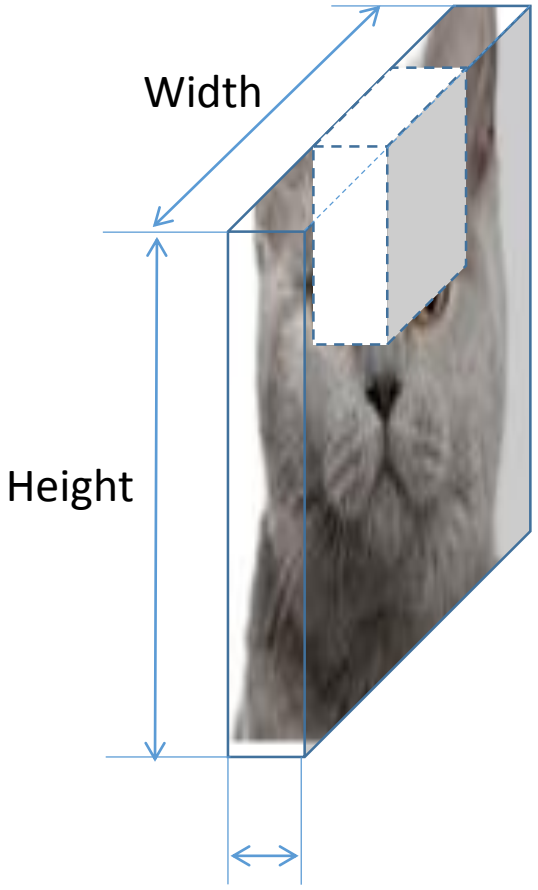


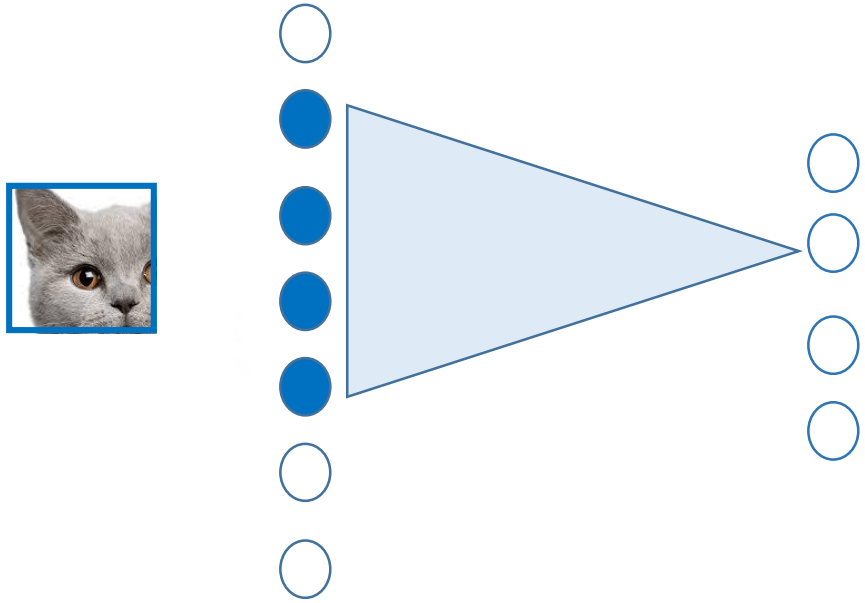
2-D representation vs 1-D representation



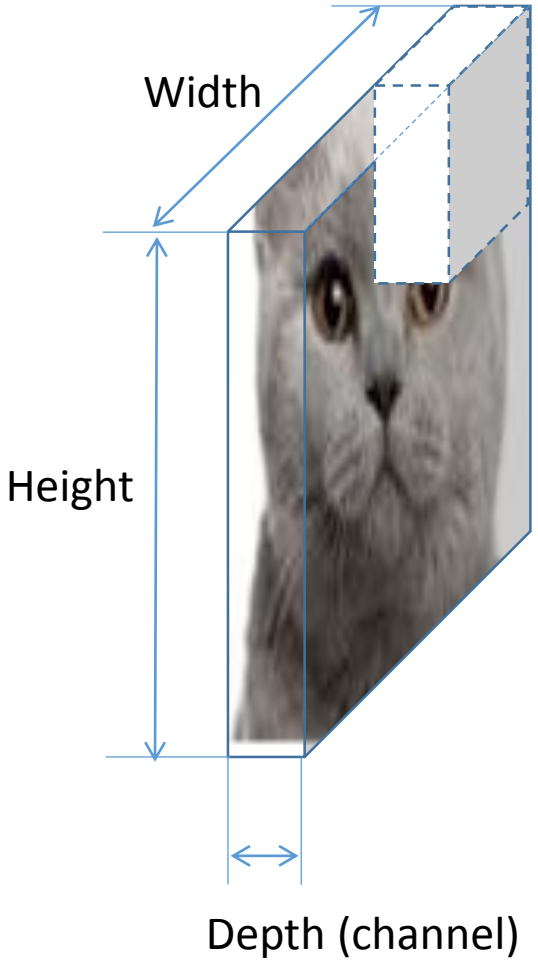


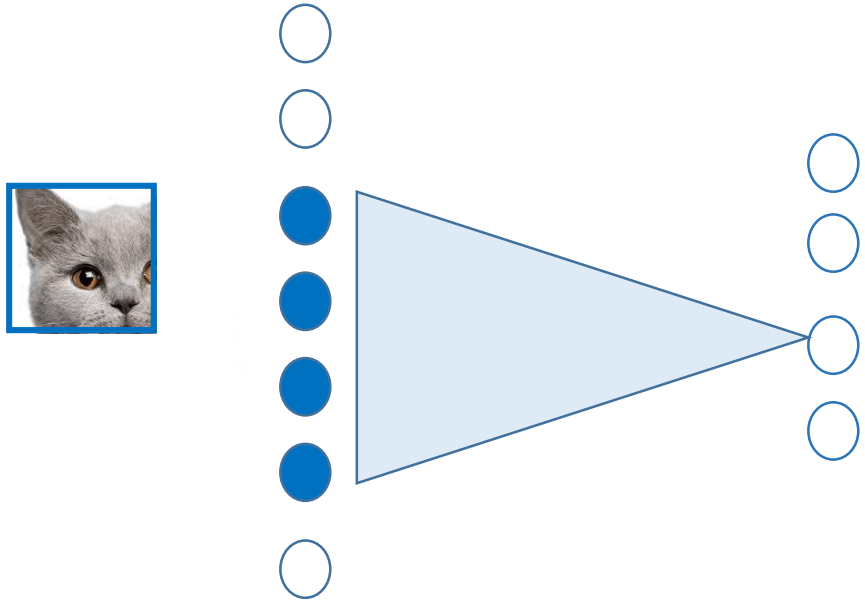
2-D representation vs 1-D representation



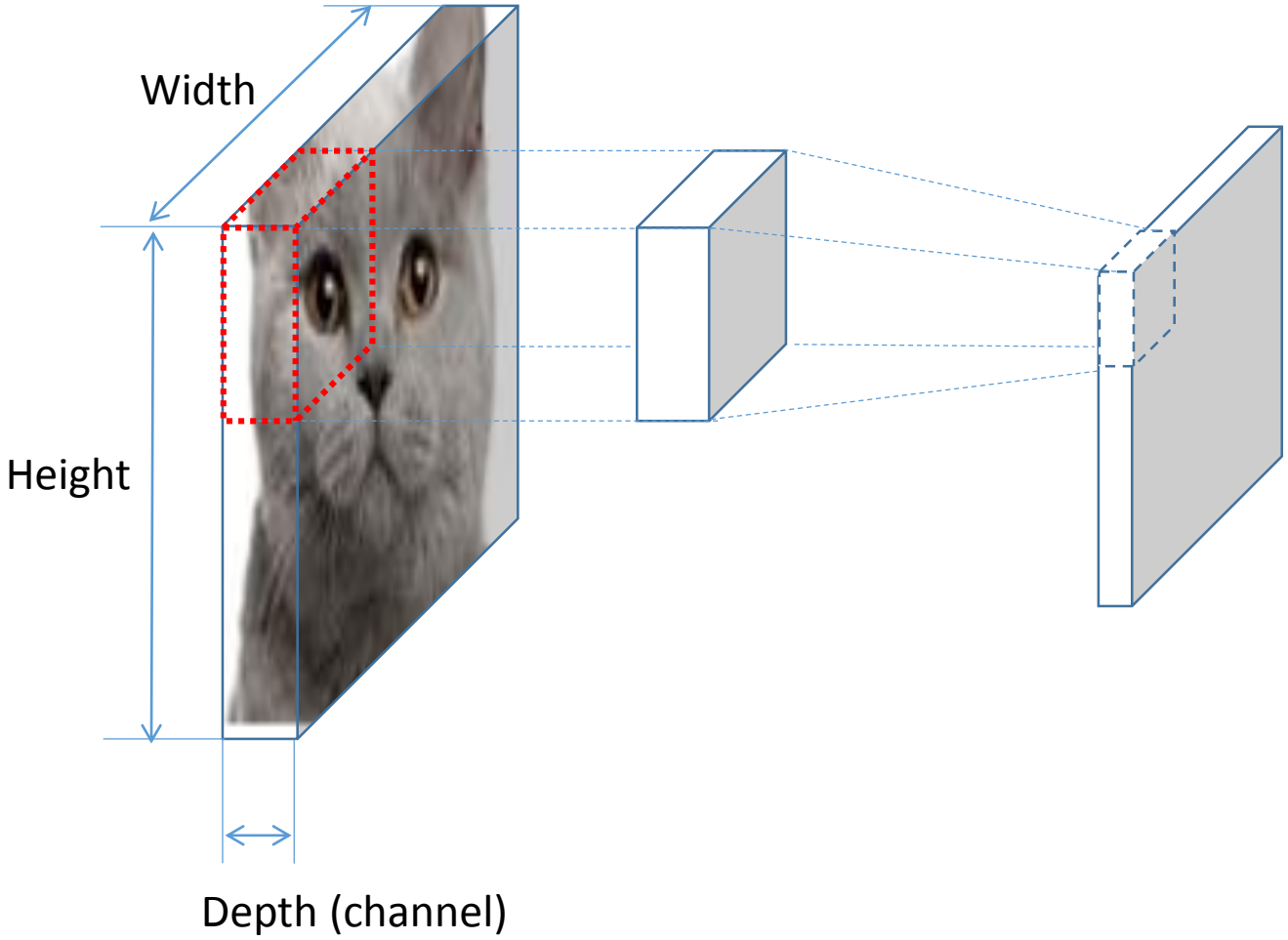


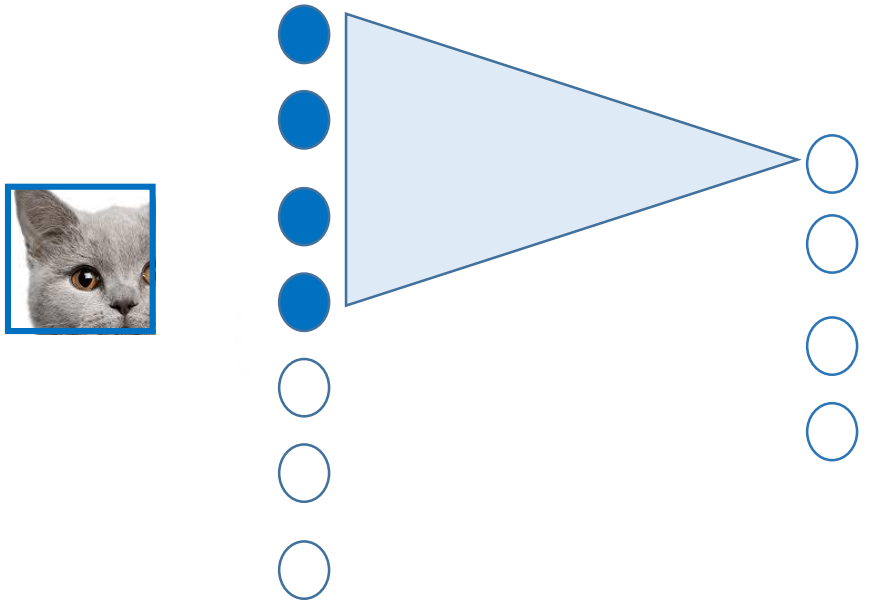
2-D representation vs 1-D representation



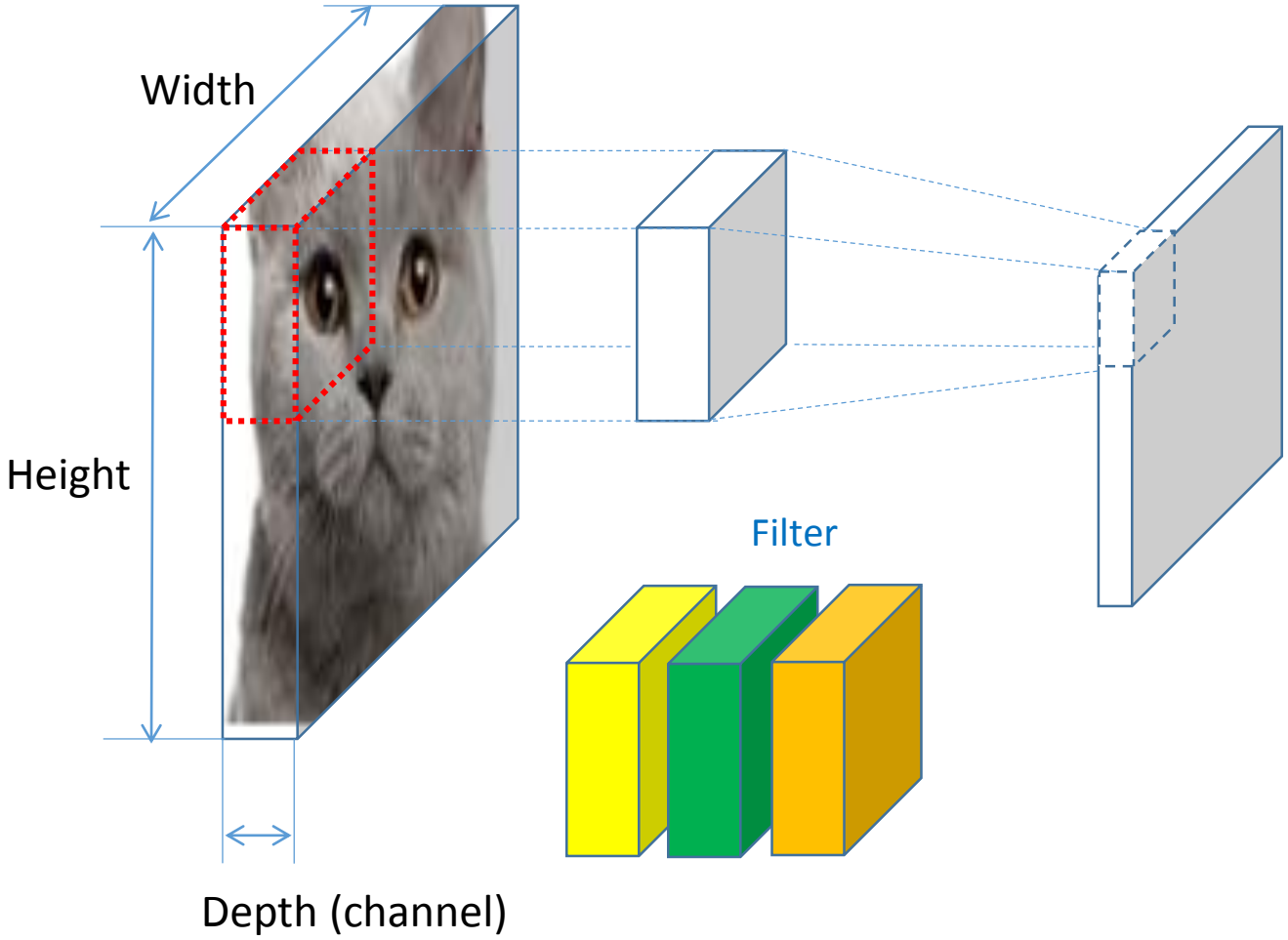


2-D representation vs 1-D representation

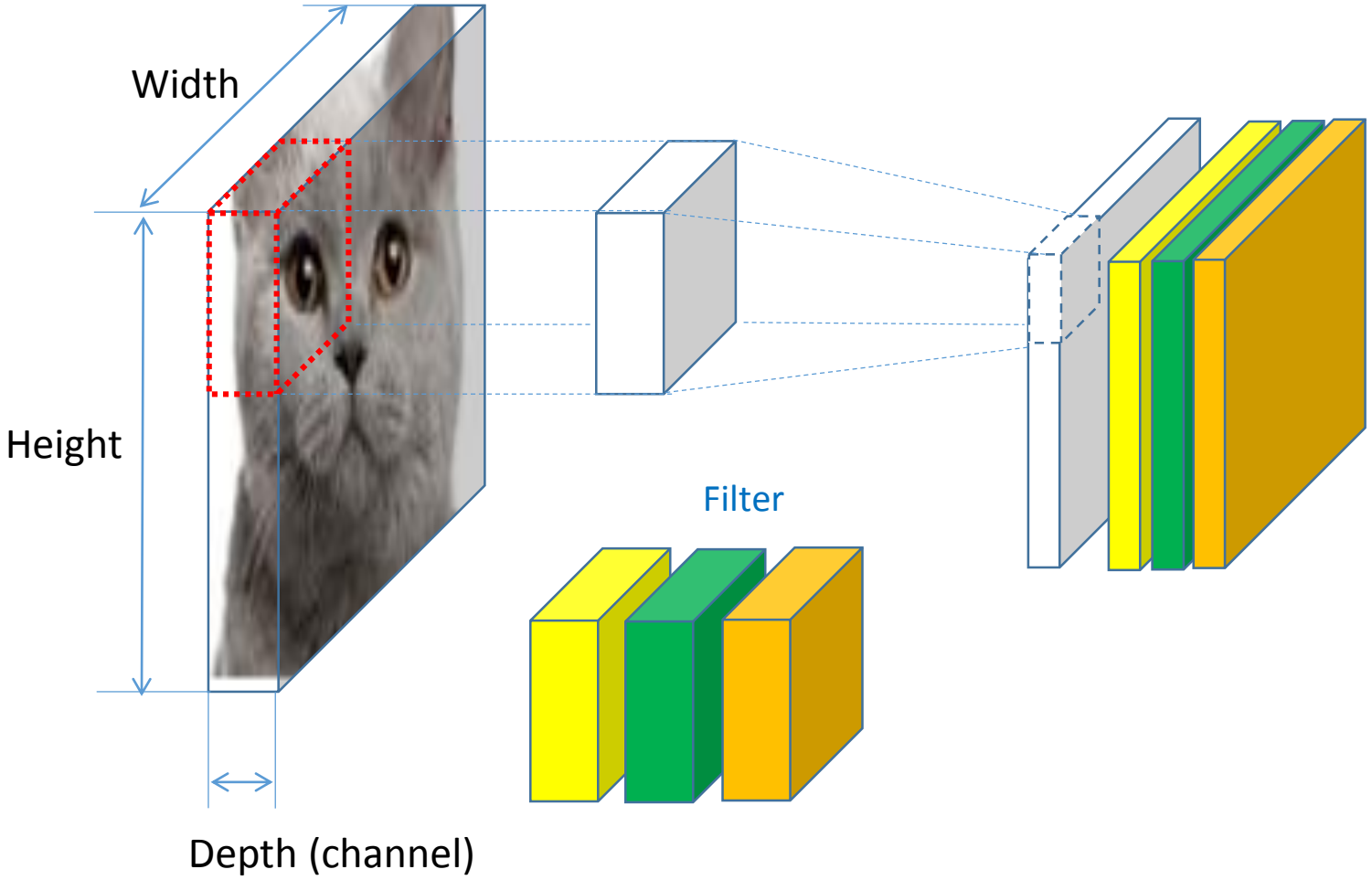




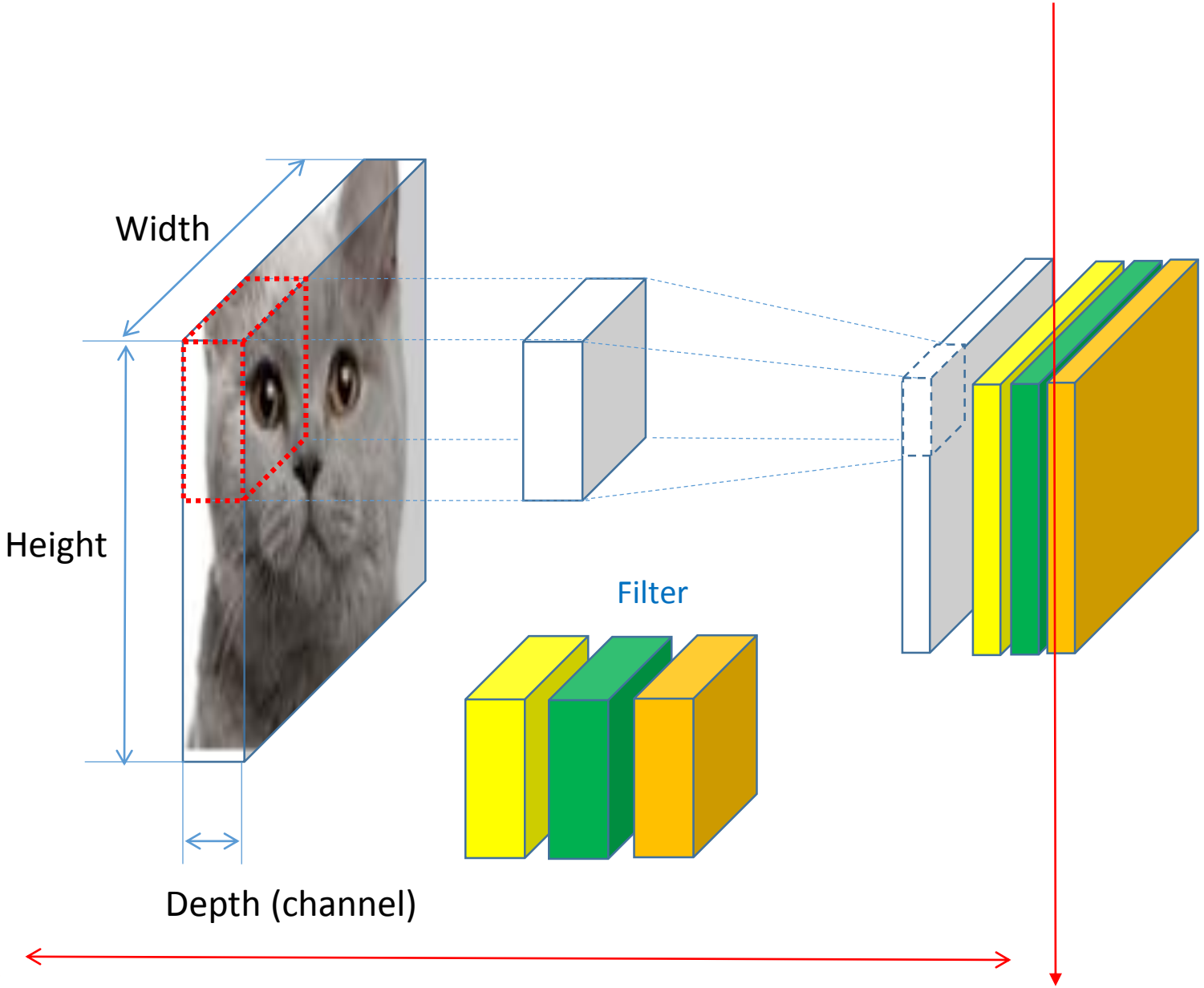
2-D representation of CNN: How does it work?



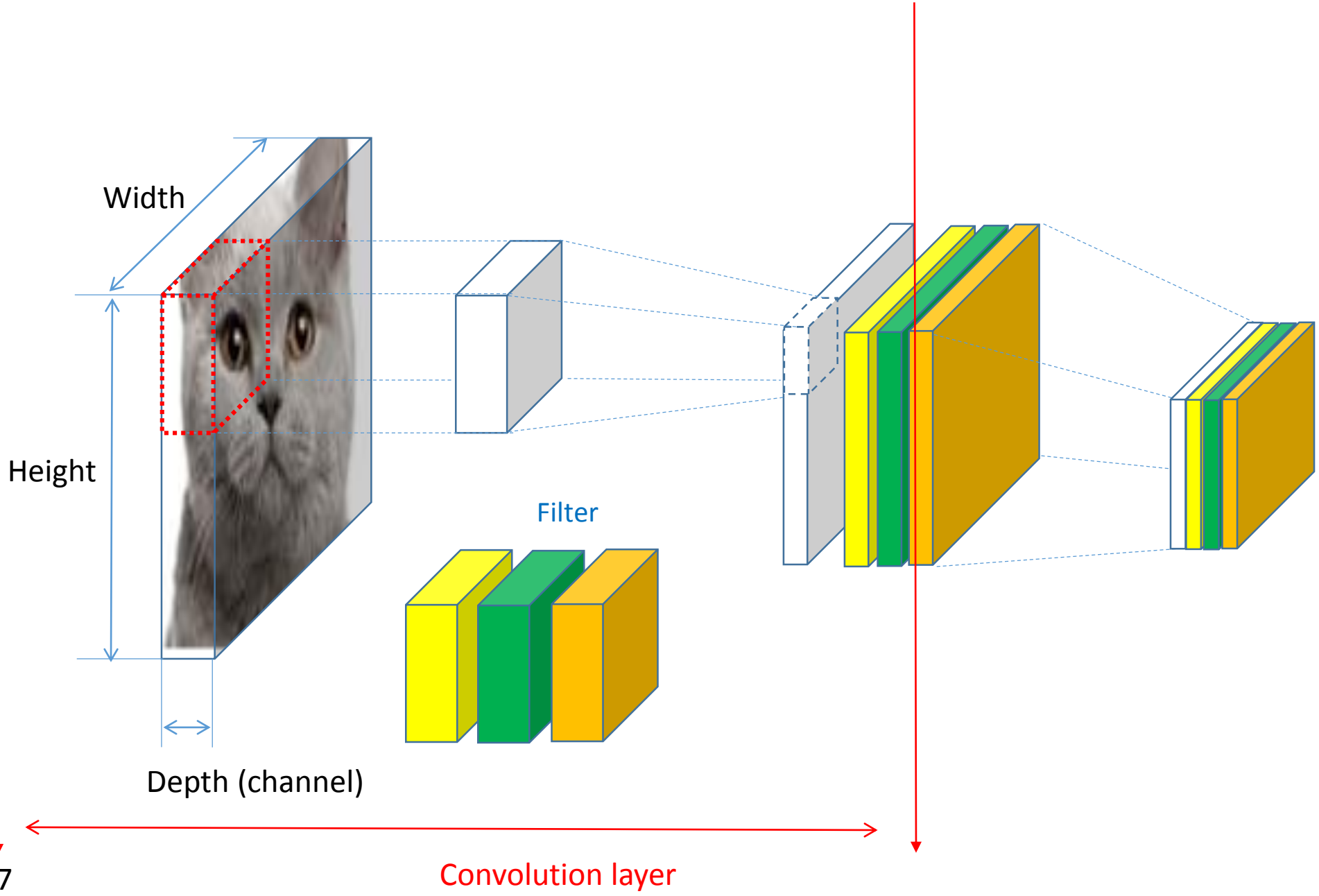
2-D representation of CNN: How does it work?



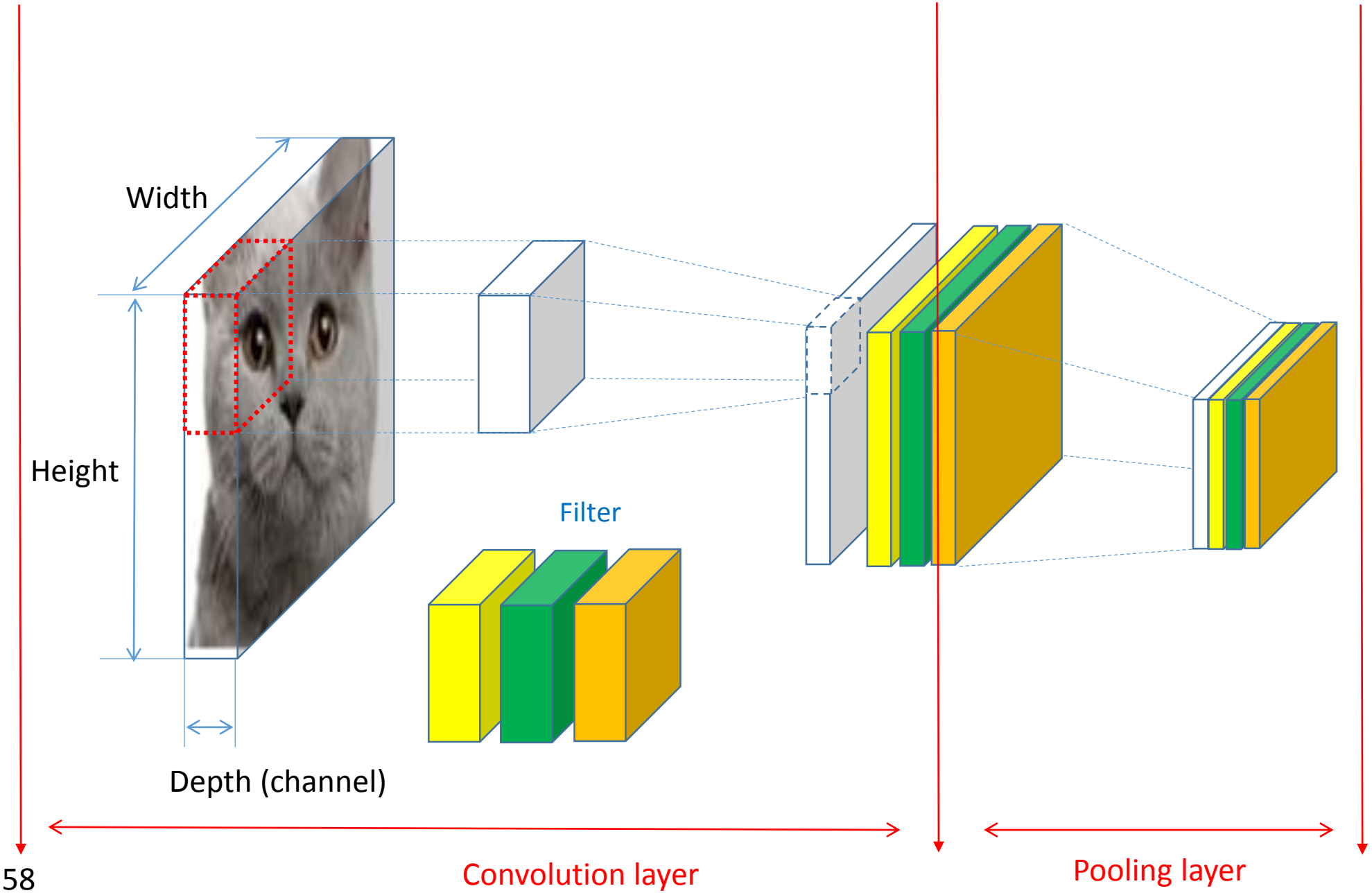
2-D representation of CNN: How does it work?



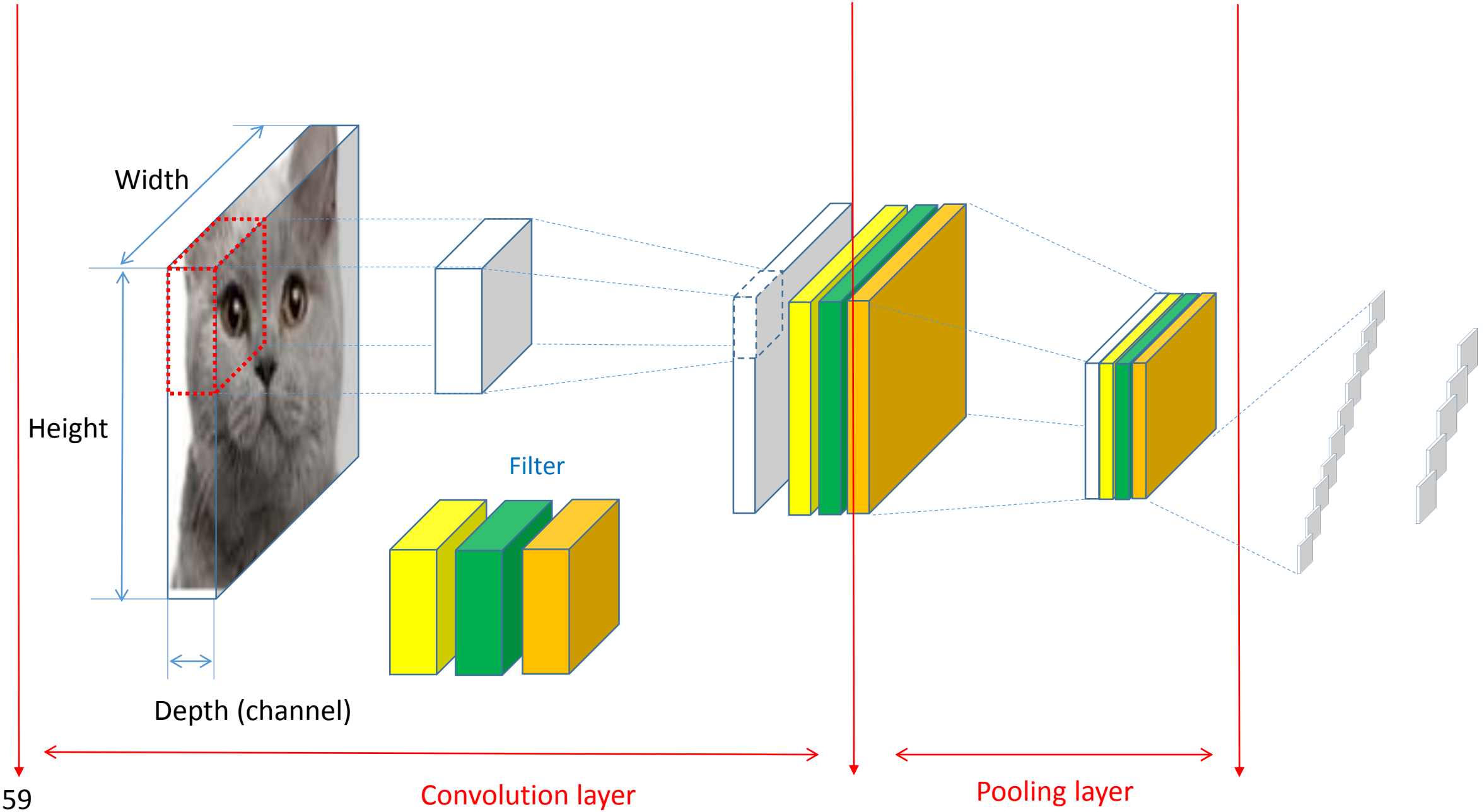
2-D representation of CNN: How does it work?



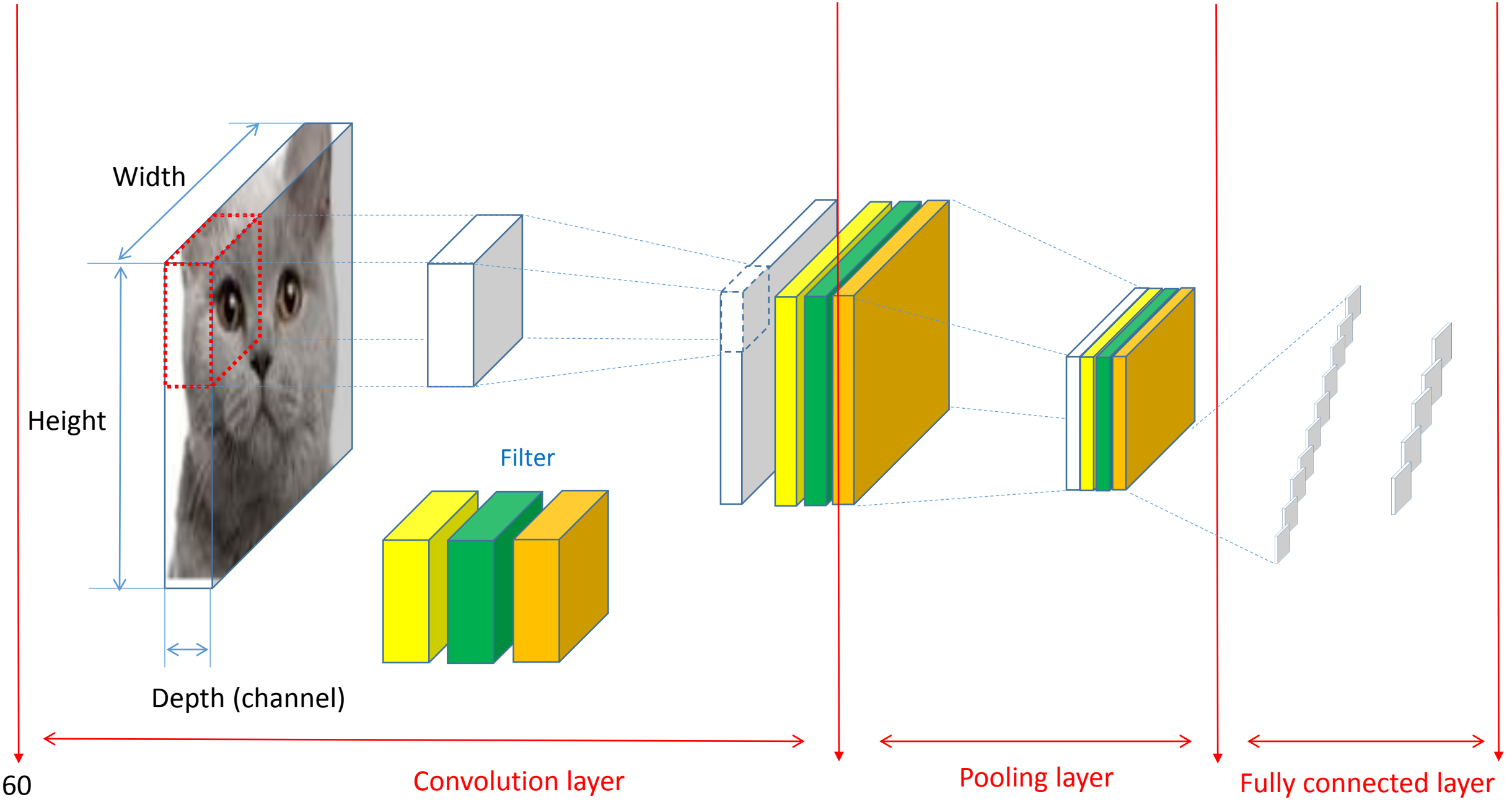
2-D representation of CNN: How does it work?



2-D representation of CNN: How does it work?

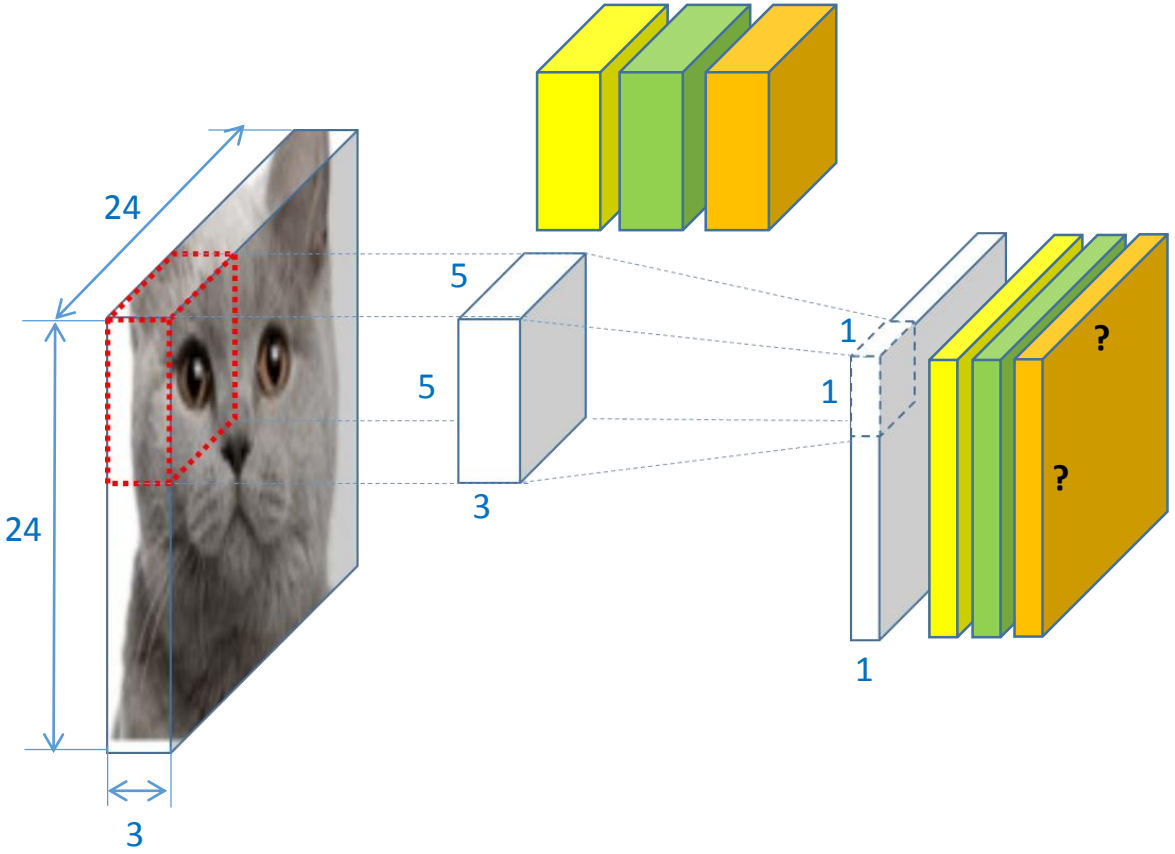


2-D representation of CNN: How does it work?



Convolution Layer

Convolutional Layer

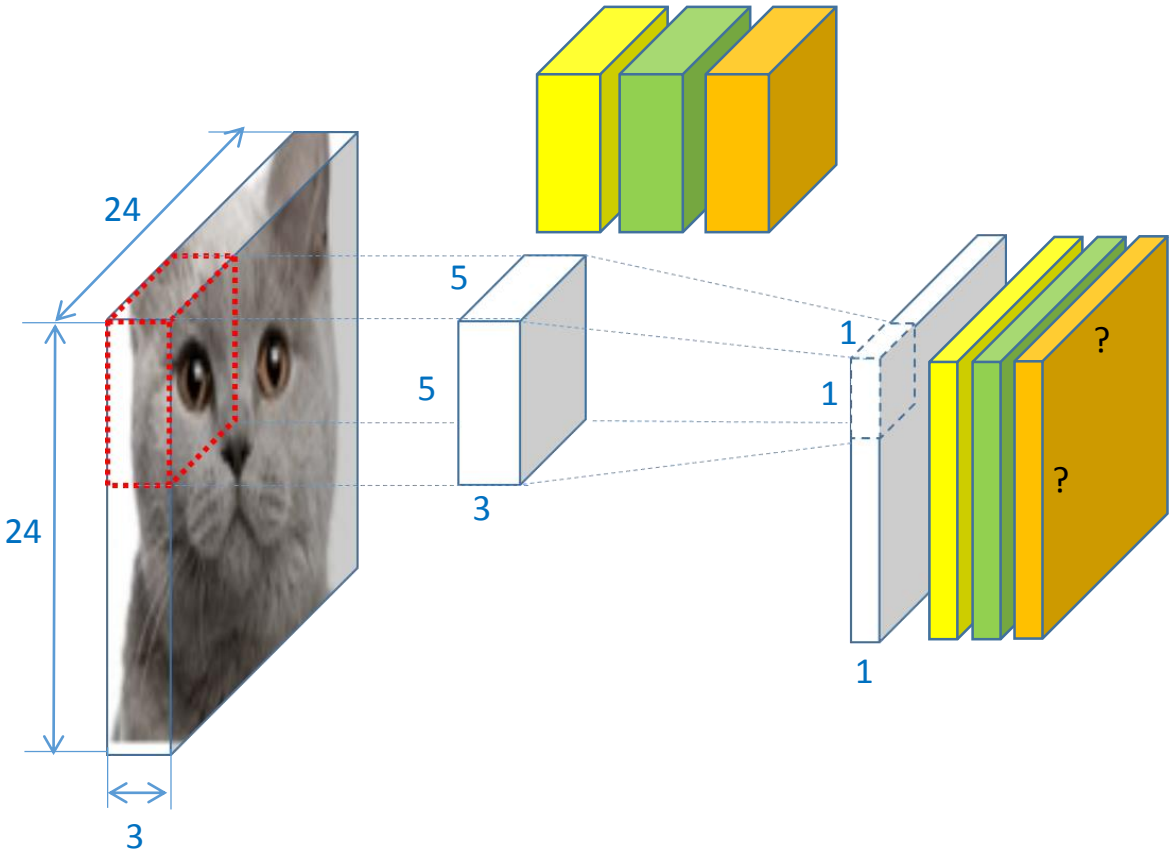


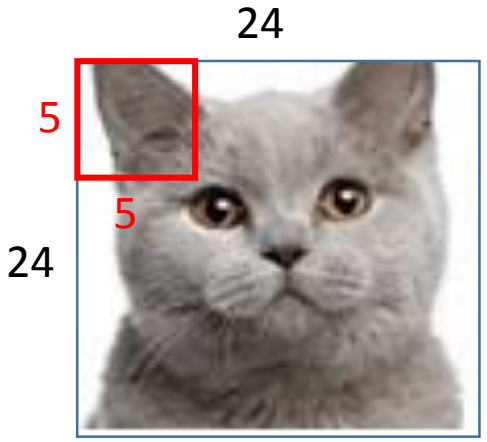
Input layer

Filters
Kernels

Activation maps
Feature maps

Convolutional Layer





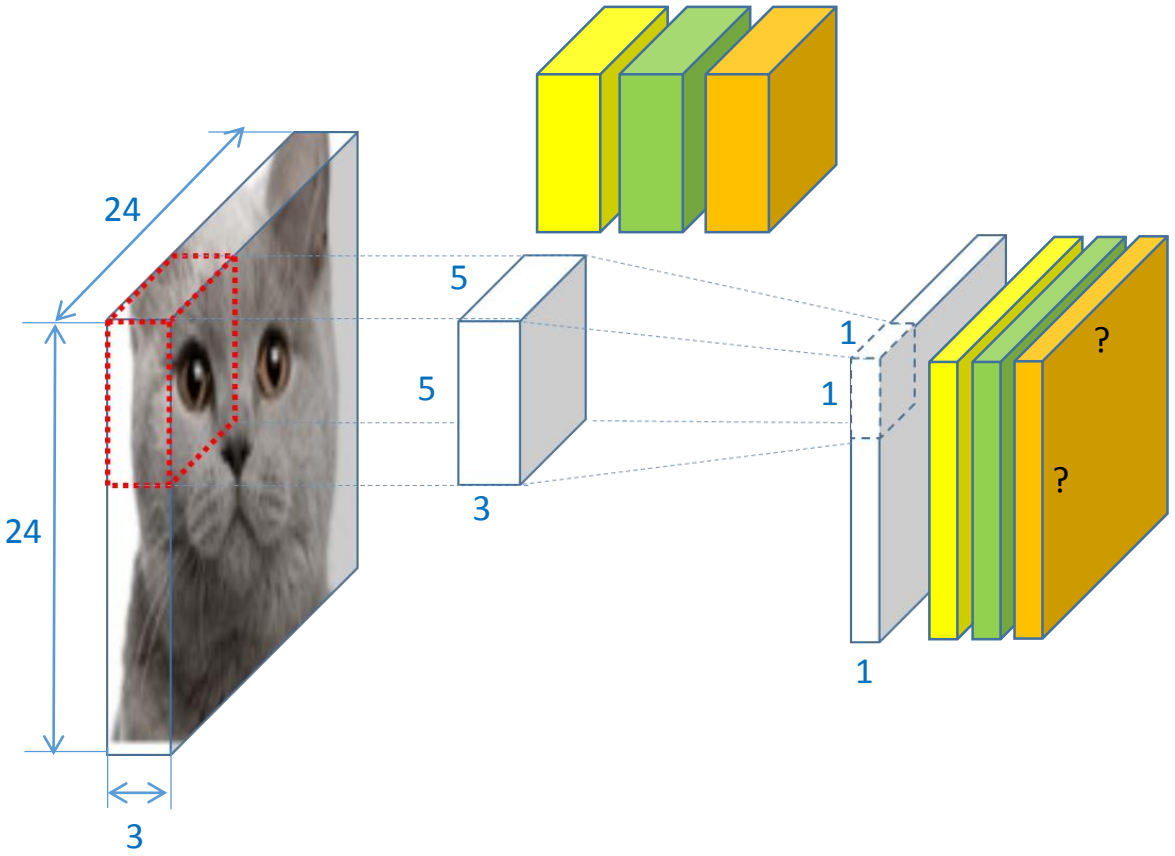
slide= 1

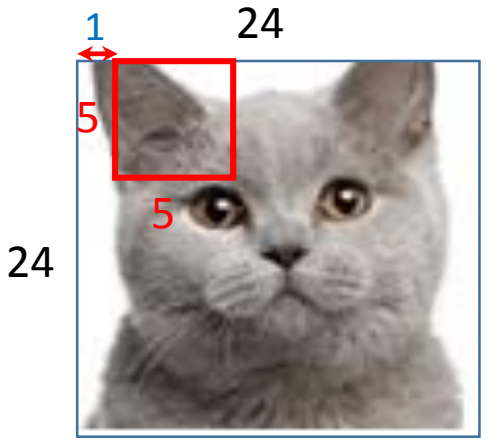
Input layer

Filters
Kernels

Activation maps
Feature maps

Convolutional Layer





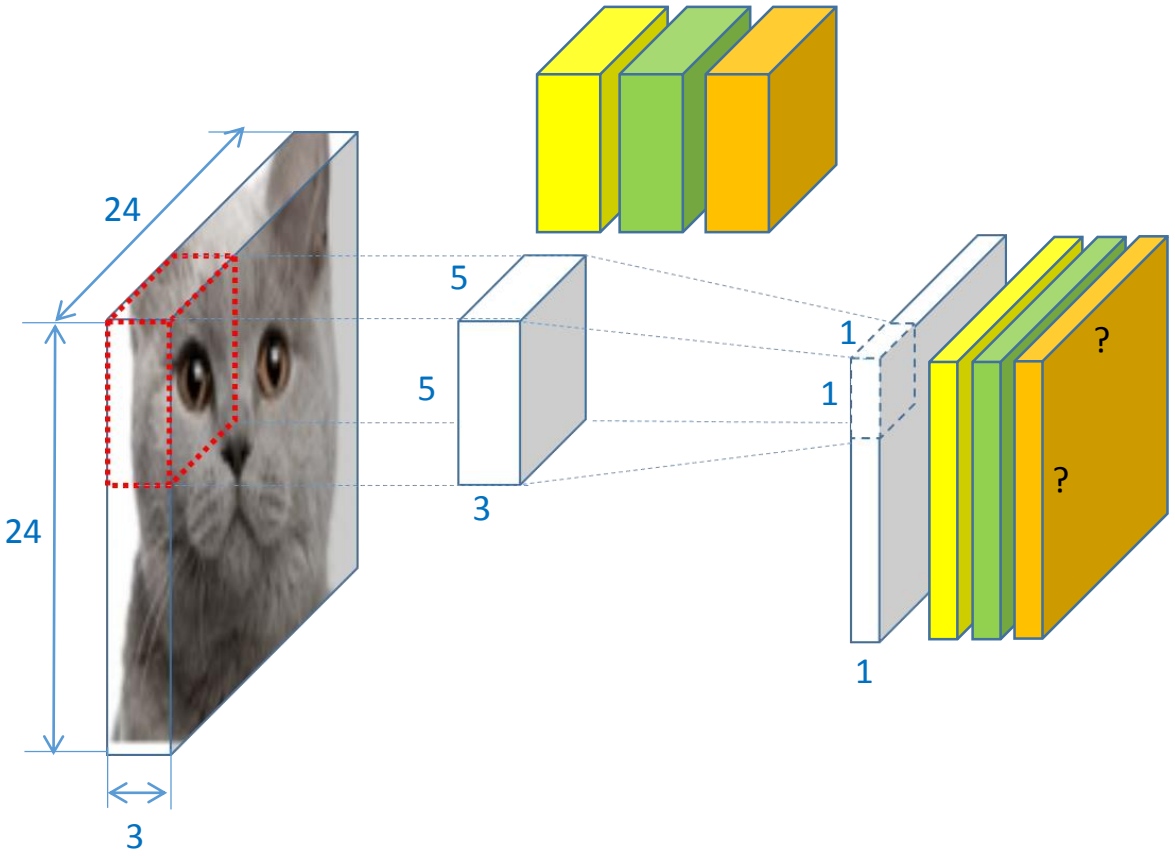
slide= 1

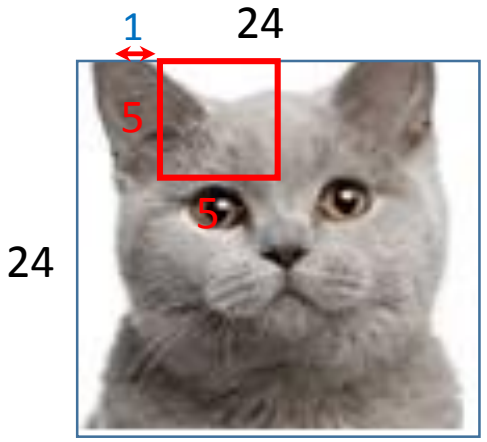
Input layer

Filters
Kernels

Activation maps
Feature maps

Convolutional Layer





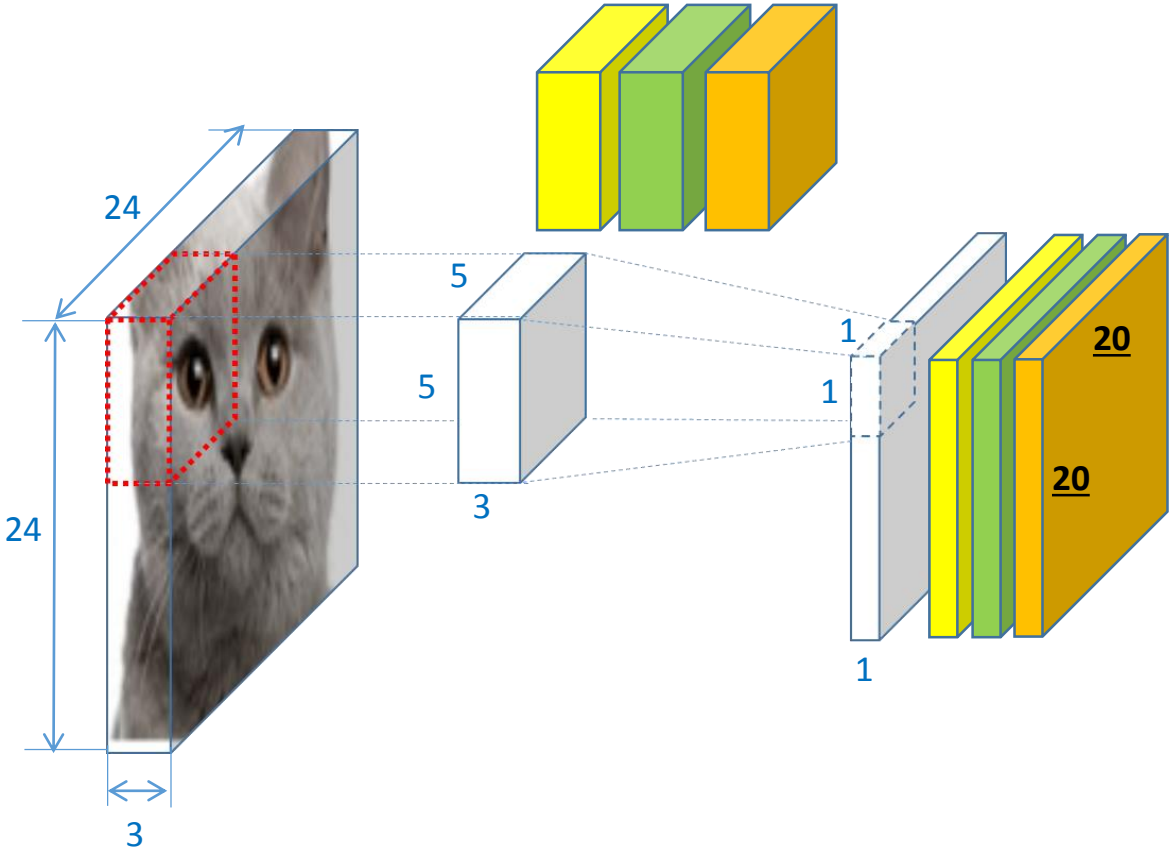
slide= 1

Input layer

Filters
Kernels

Activation maps
Feature maps

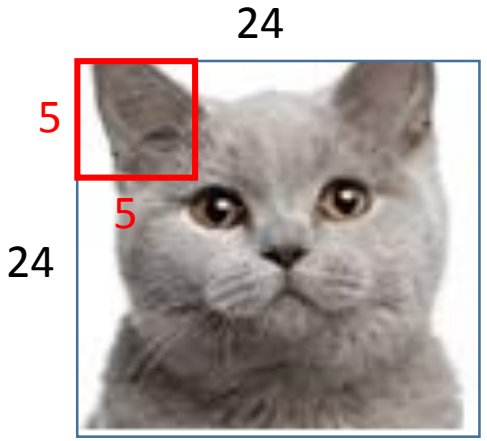
Convolutional Layer

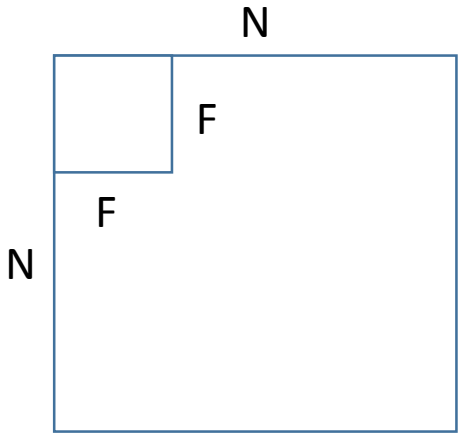


Input layer

Filters
Kernels

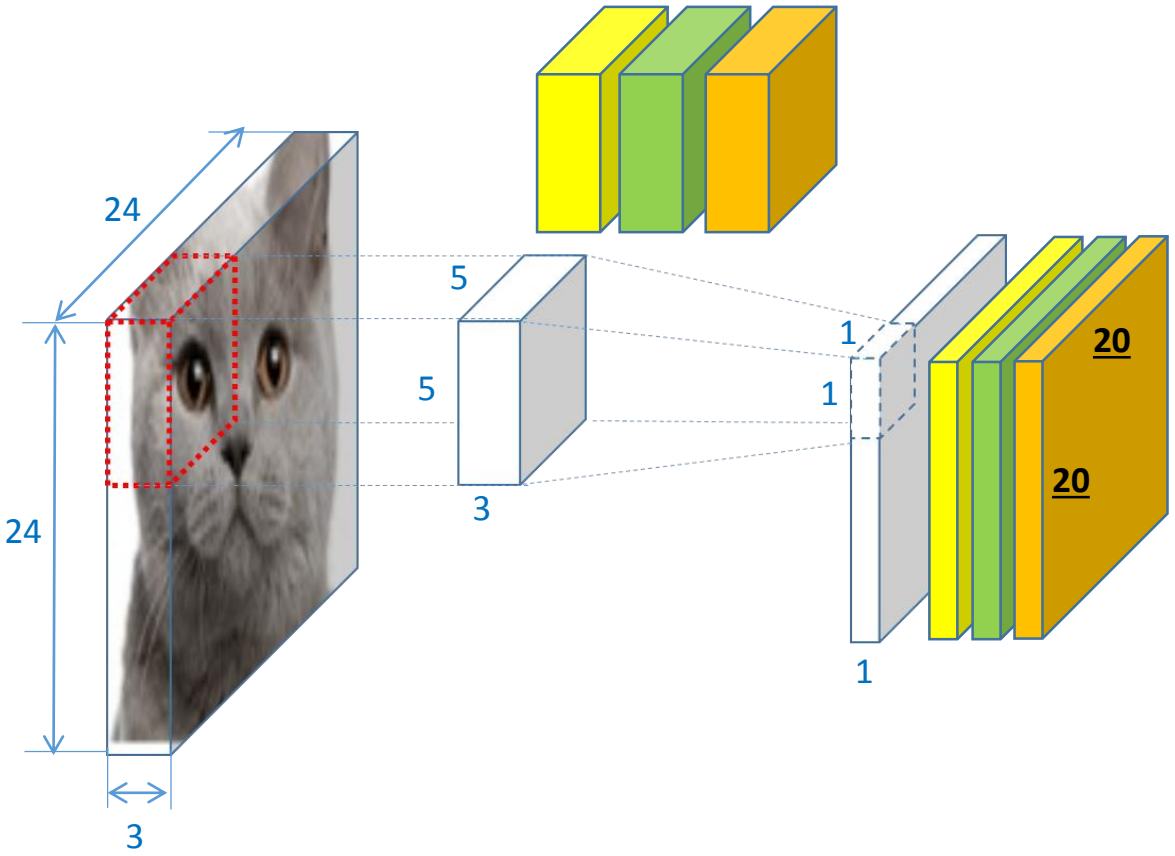
Activation maps
Feature maps





$$\text{Size of activation map} = \frac{(N - F)}{stride} + 1$$

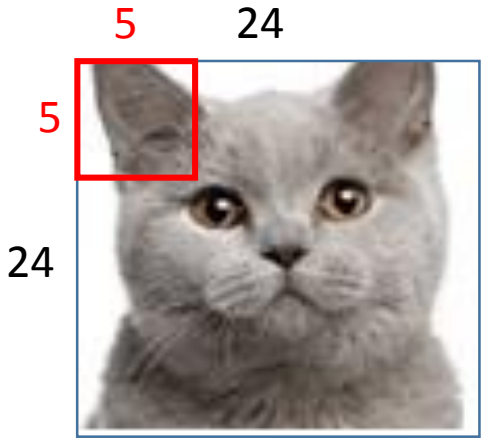
Convolutional Layer

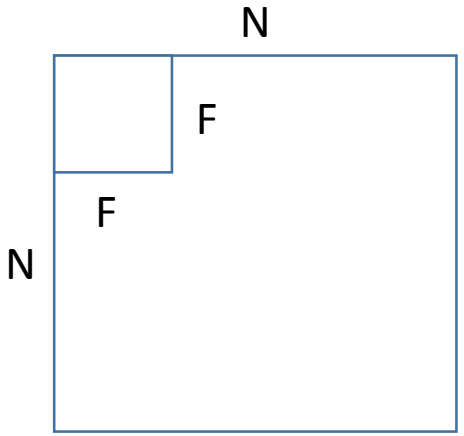


Input layer

Filters
Kernels

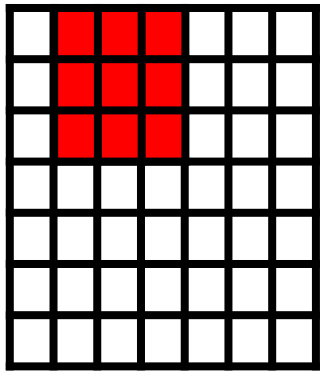
Activation maps
Feature maps



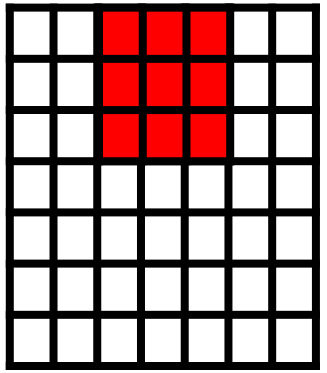


Size of activation map $= \frac{(N-F)}{stride} + 1$

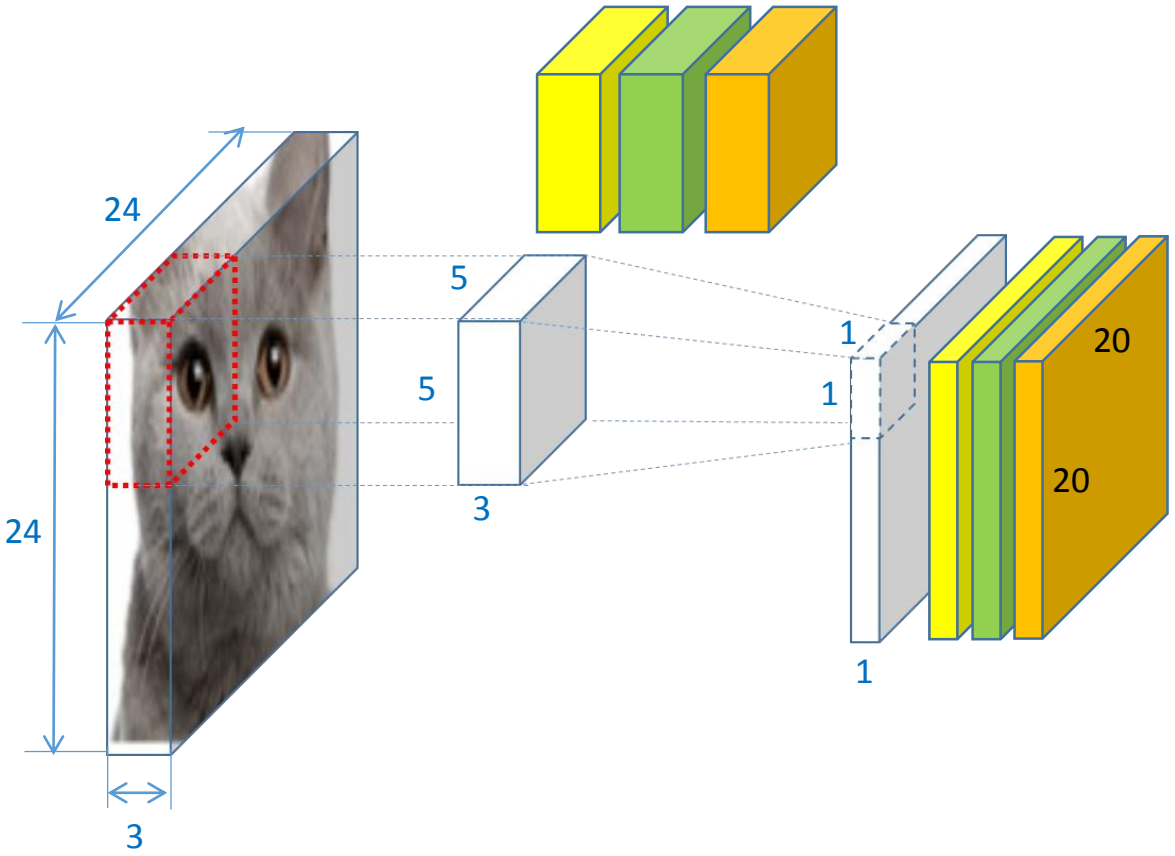
stride 1



stride 2



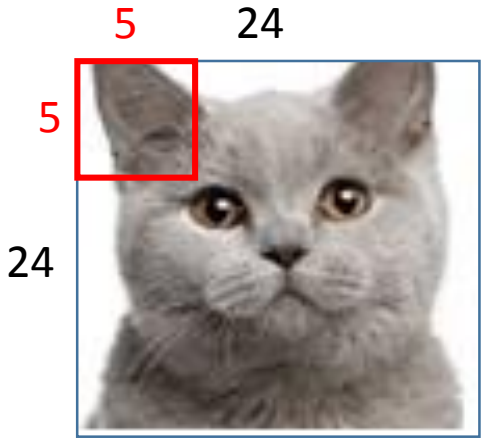
Convolutional Layer

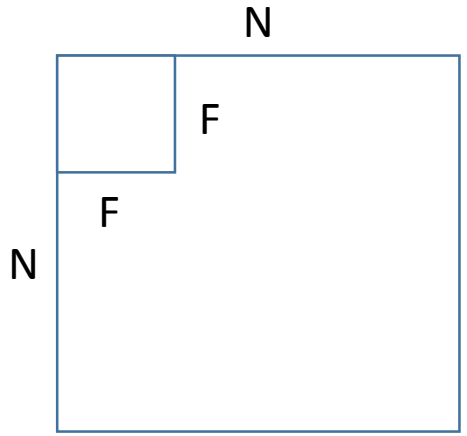


Input layer

Filters
Kernels

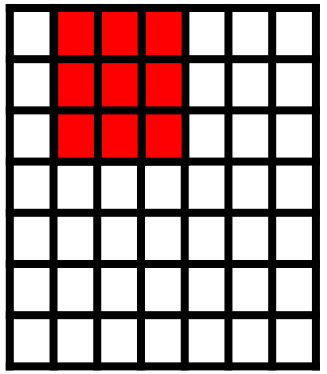
Activation maps
Feature maps





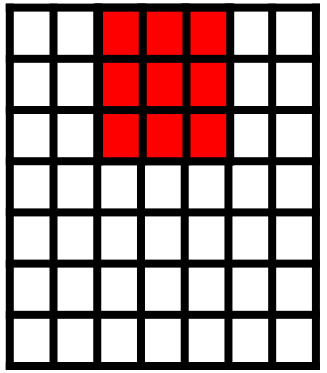
Size of activation map $= \frac{(N - F)}{stride} + 1$

stride 1



Activation map
(5 x 5 matrix)

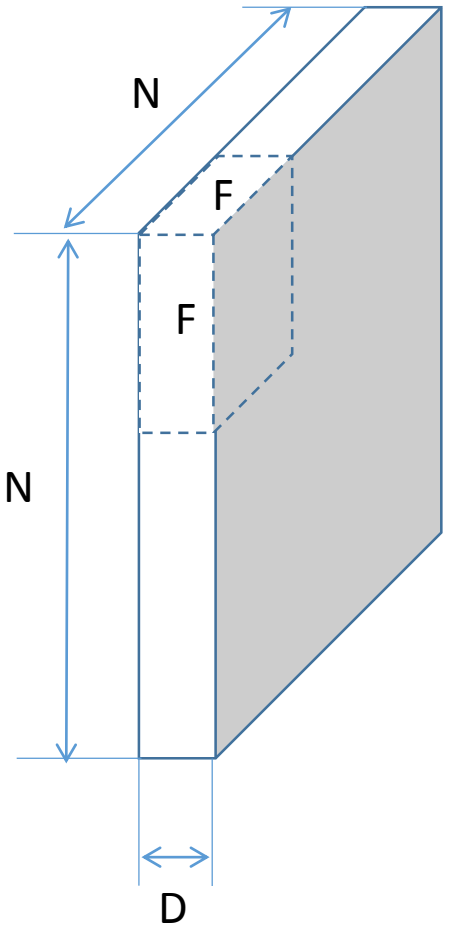
stride 2

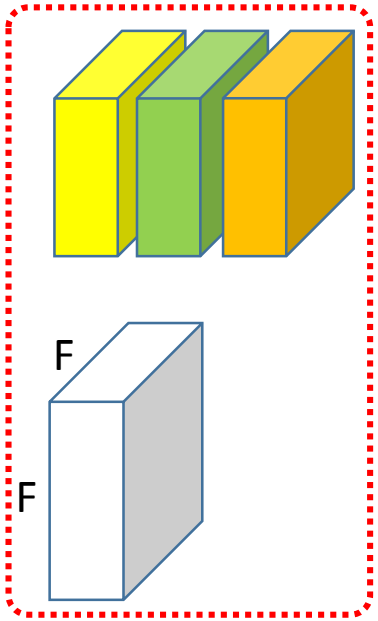


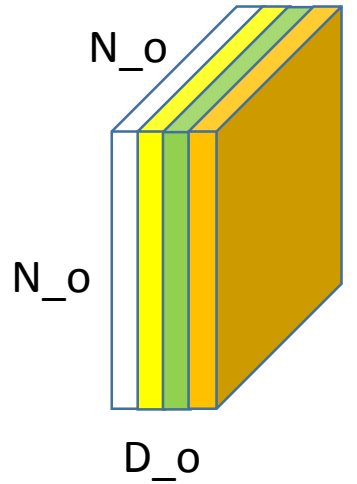
Activation map
(3 x 3 matrix)

Convolutional Layer

Hyper parameters	Symbols
Number of filters	K
Size of the filter	F
Stride	S
Padding	P

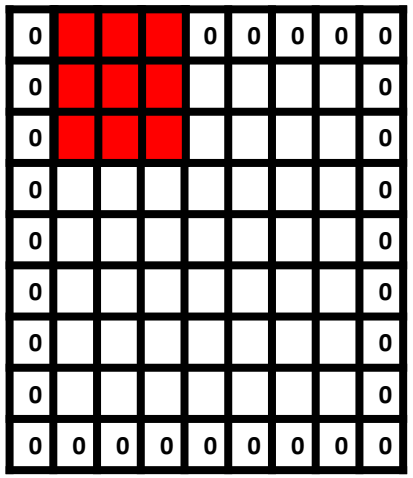






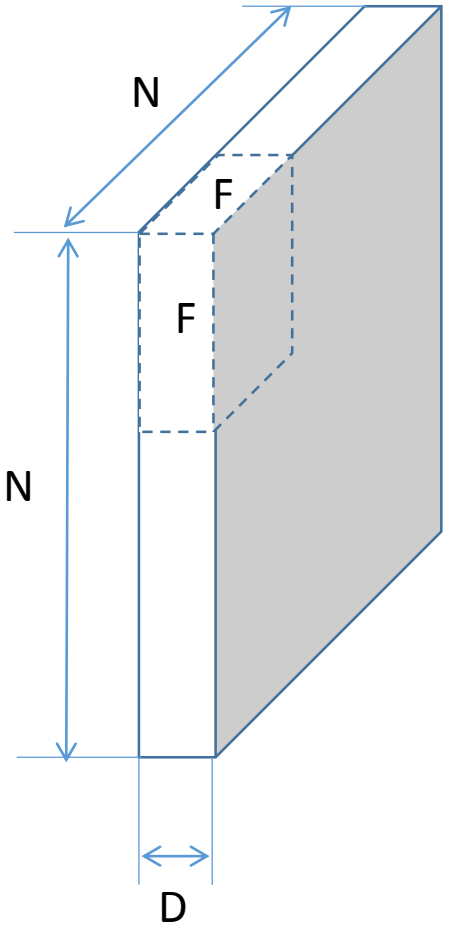
Convolutional Layer

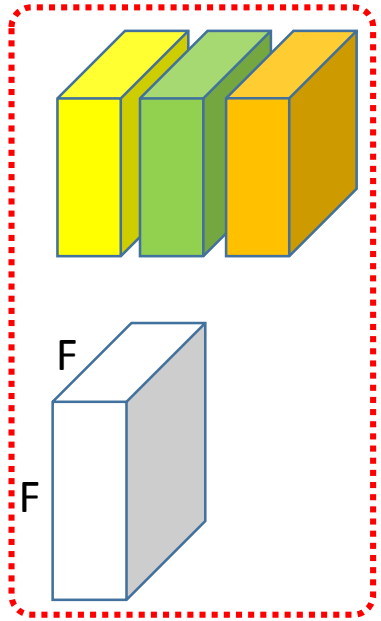
Hyper parameters	Symbols
Number of filters	K
Size of the filter	F
Stride	S
Padding	P



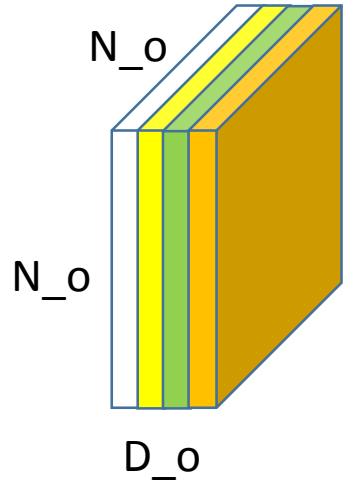
- Padding: P=1
- Stride: S=1
- activation map becomes is 7 x 7 matrix

- Padding aims to maintain the original dimension of the original data.



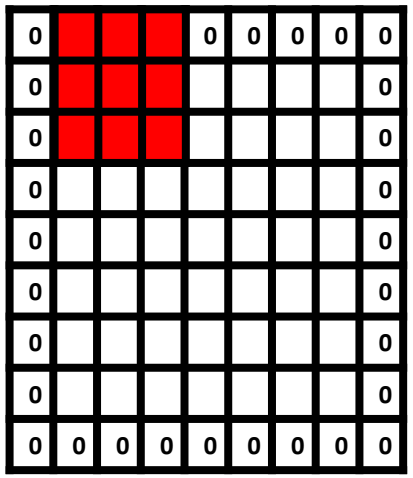


Number of filters: K



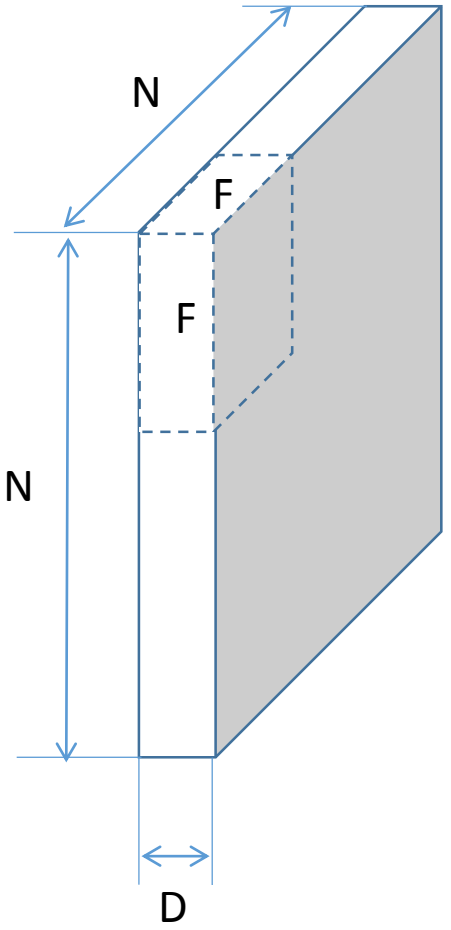
Convolutional Layer

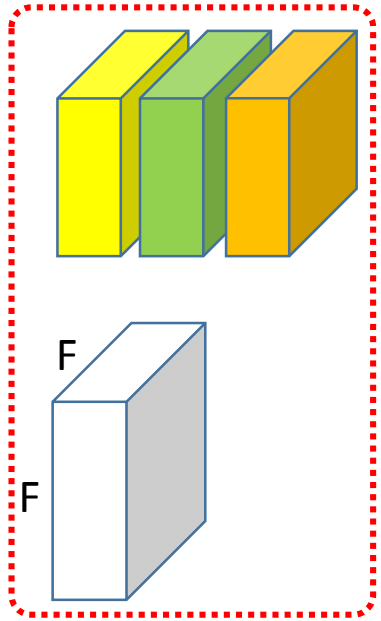
Hyper parameters	Symbols
Number of filters	K
Size of the filter	F
Stride	S
Padding	P



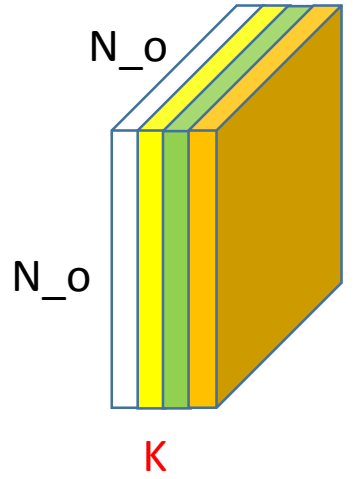
- Padding: P=1
- Stride: S=1
- activation map becomes is 7 x 7 matrix

- Padding aims to maintain the original dimension of the original data.





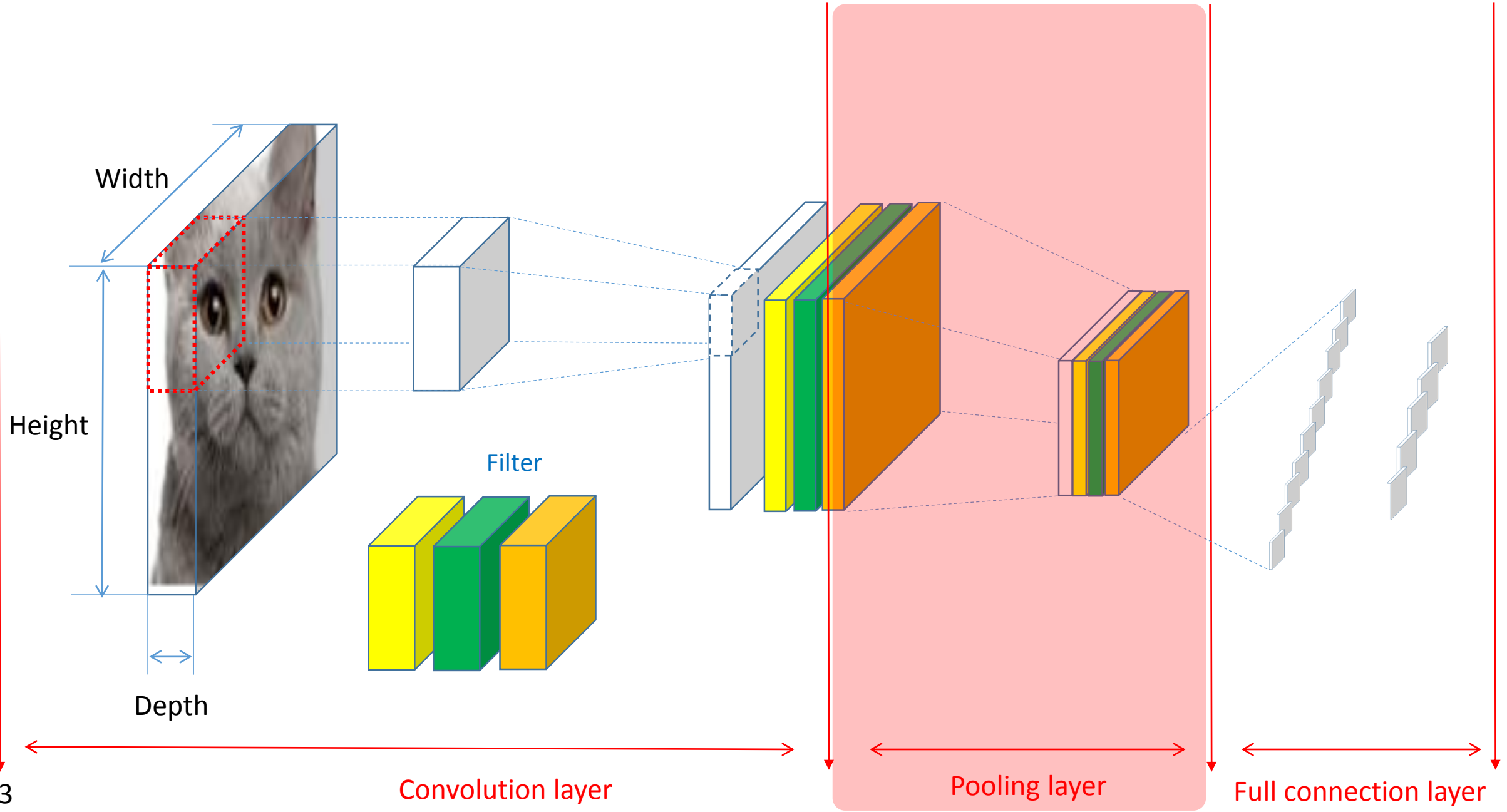
Number of filters: K



$$N\_o = \frac{(N - F + 2P)}{S} + 1$$

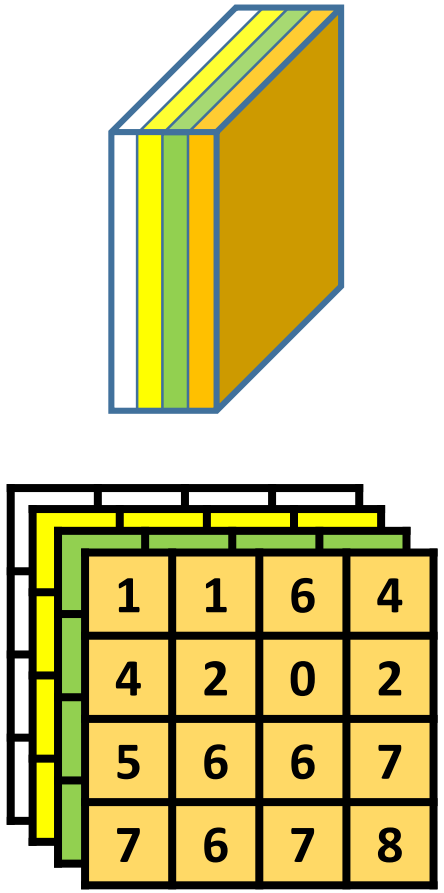
Pooling Layer

2-D representation of CNN: How does it work?



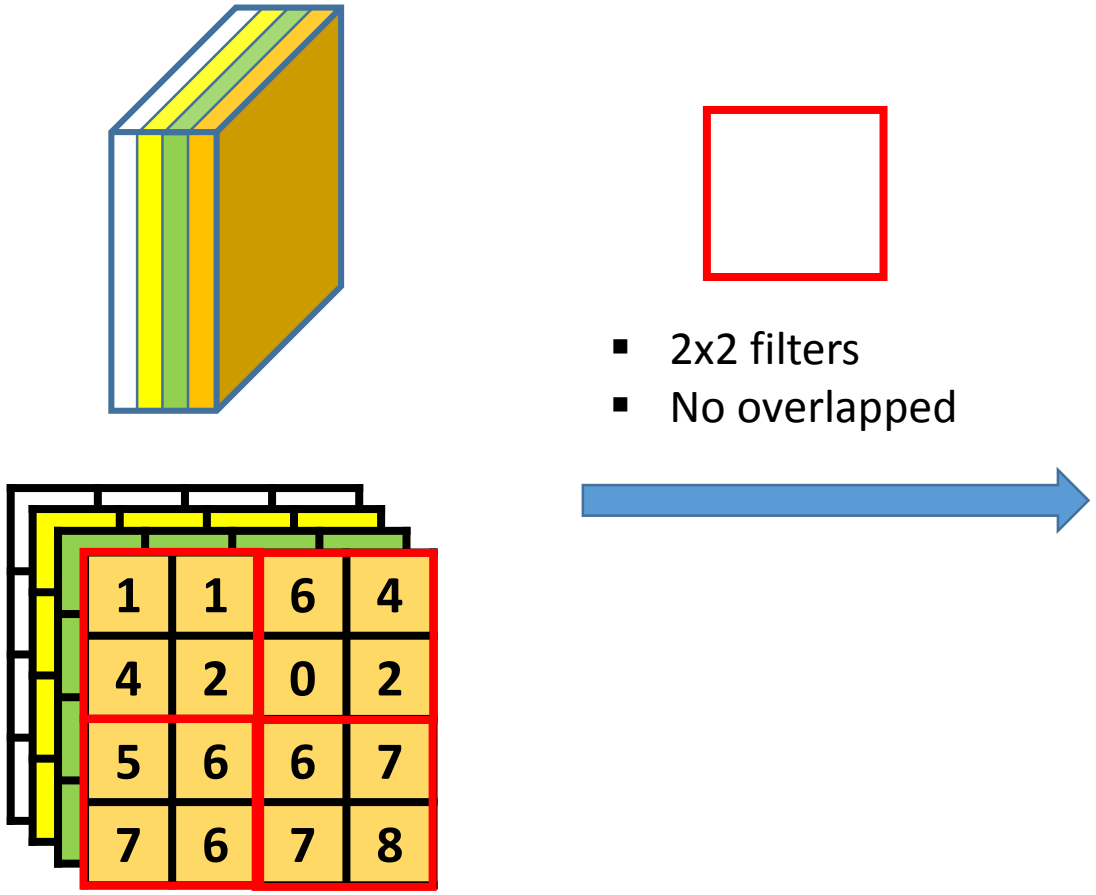
Pooling Layer

- This layer aims to reduce the dimension of each activation map.



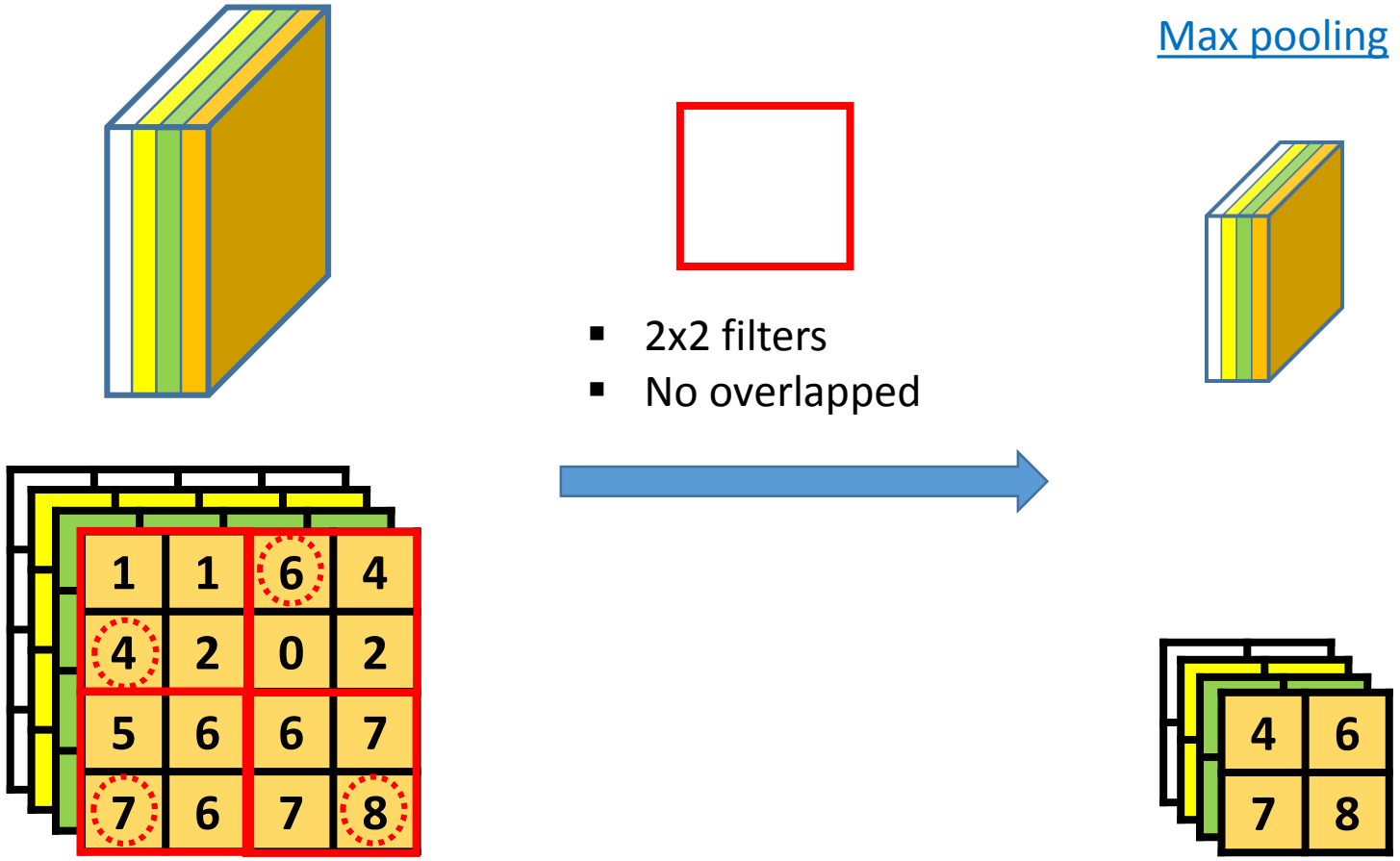
Pooling Layer

- This layer aims to reduce the dimension of each activation map.



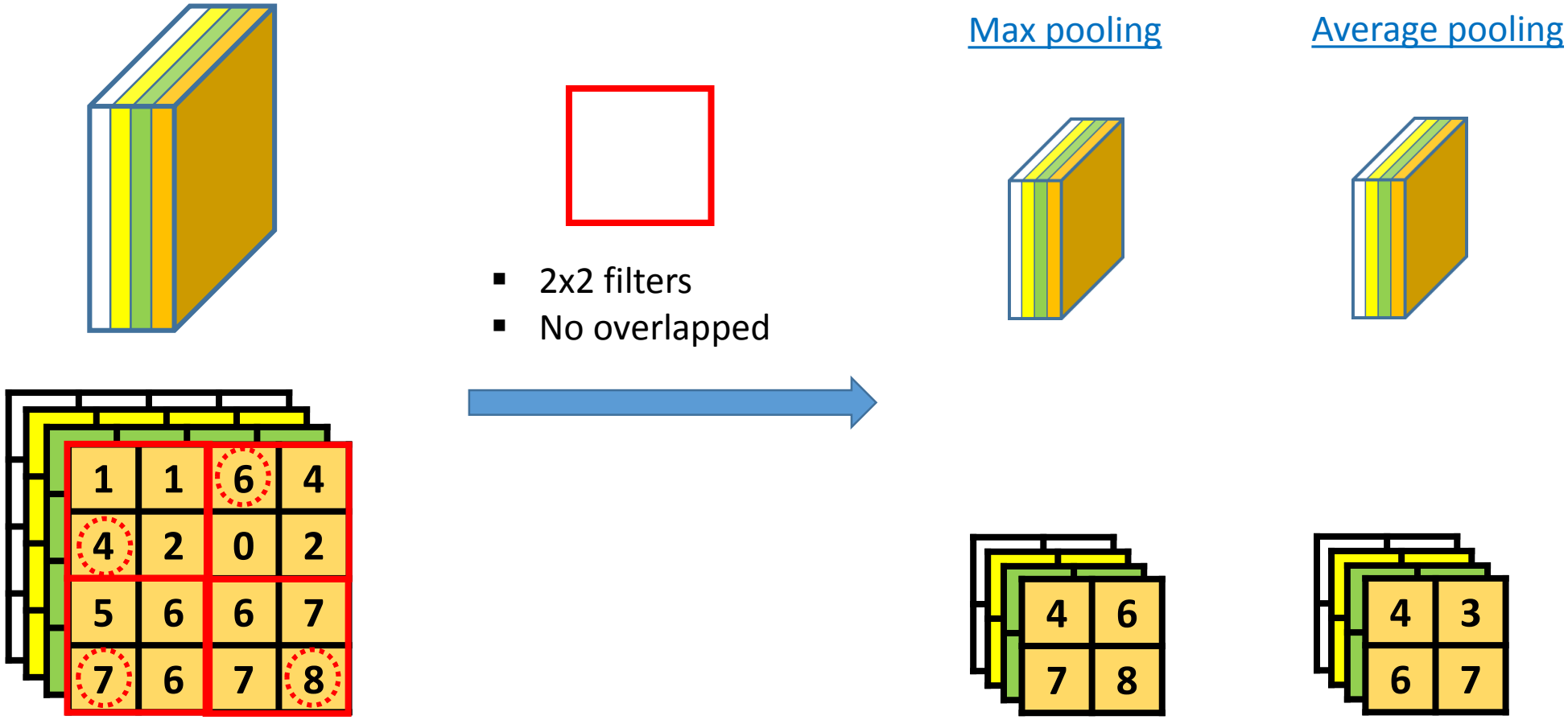
Pooling Layer

- This layer aims to reduce the dimension of each activation map.



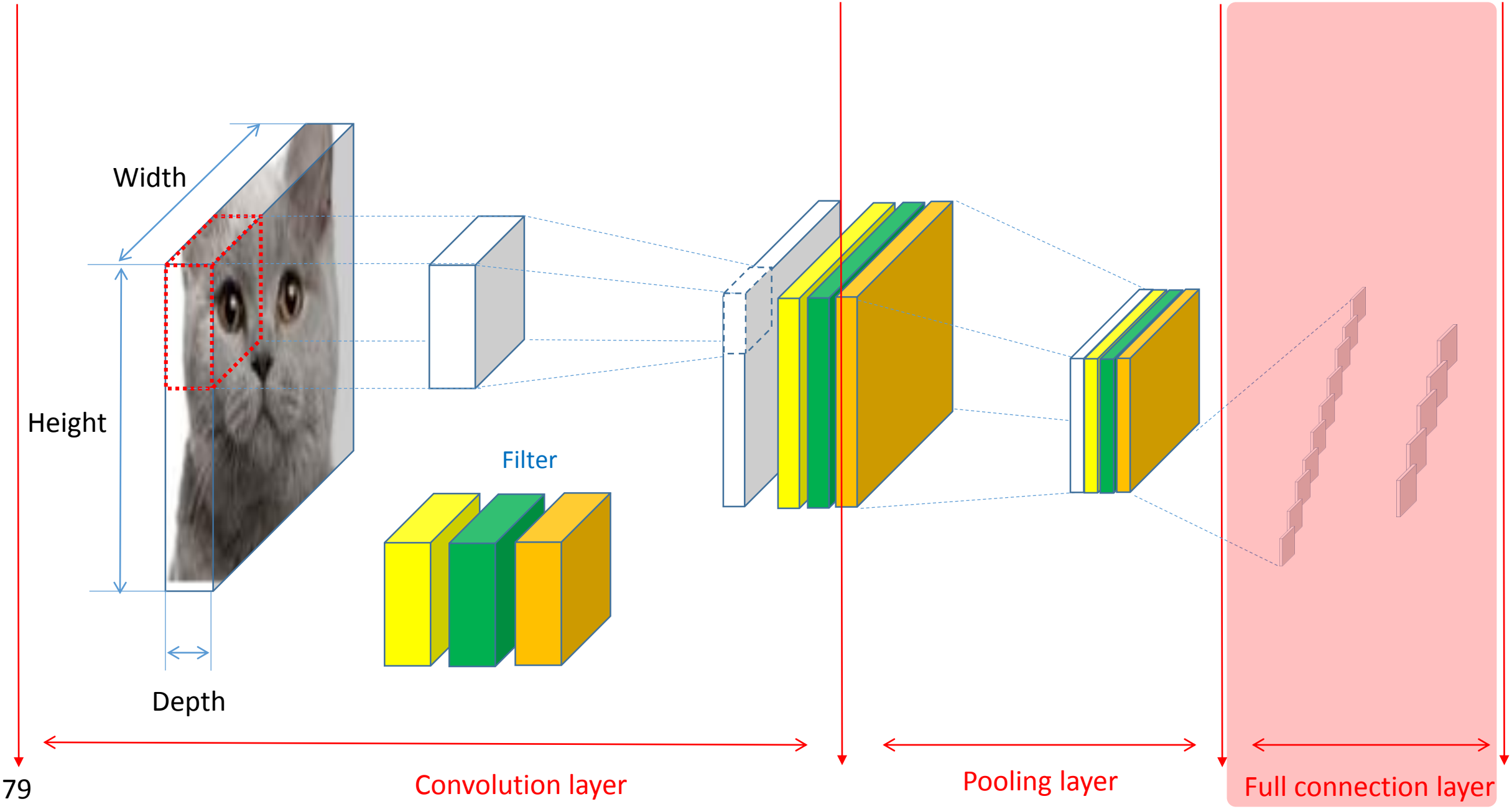
Pooling Layer

- This layer aims to reduce the dimension of each activation map.

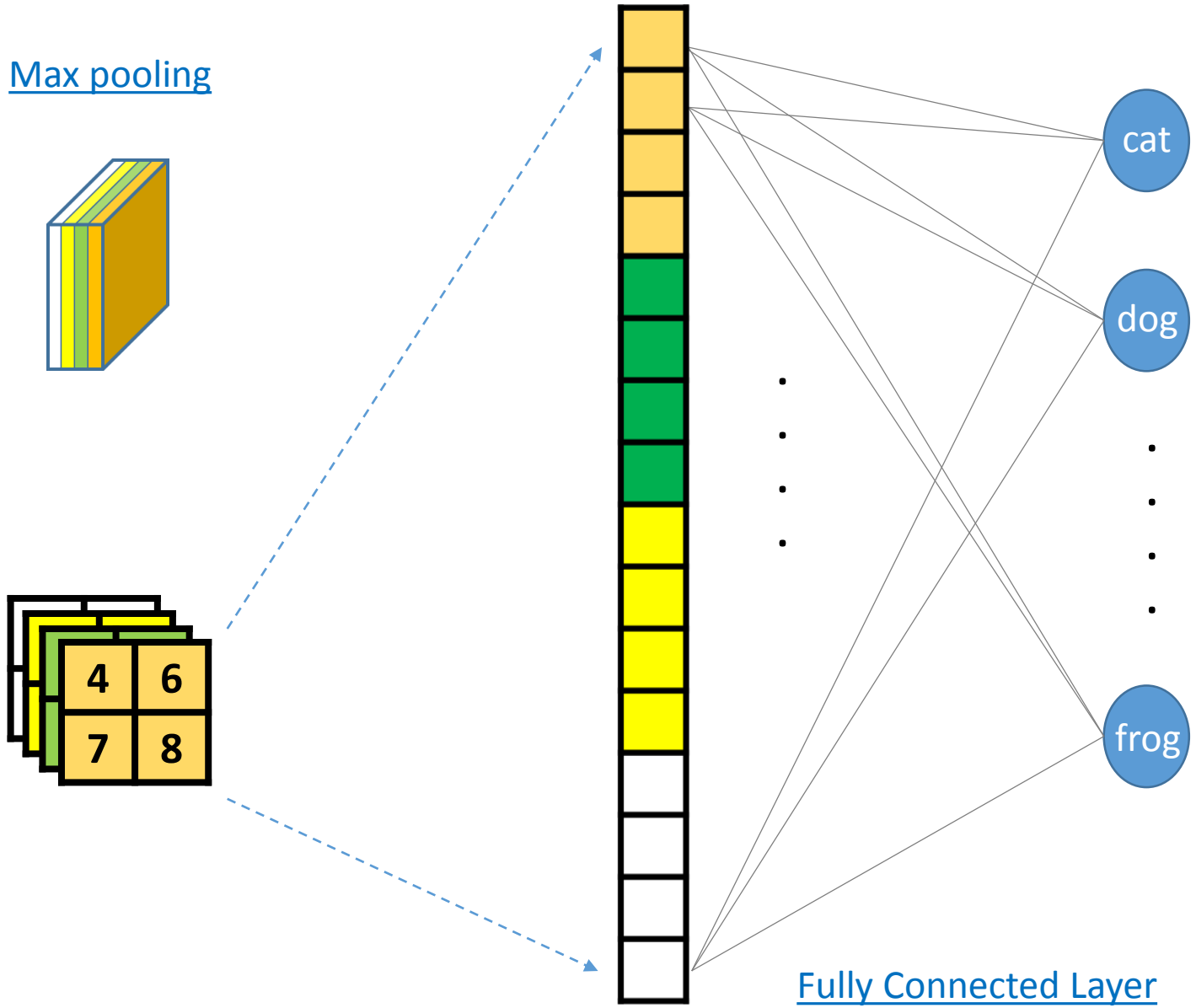


Fully Connected Layer

2-D representation of CNN: How does it work?

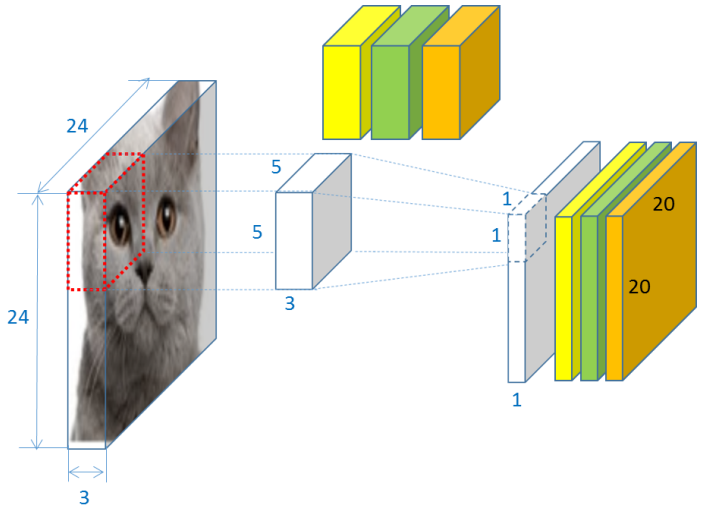


Fully Connected Layer



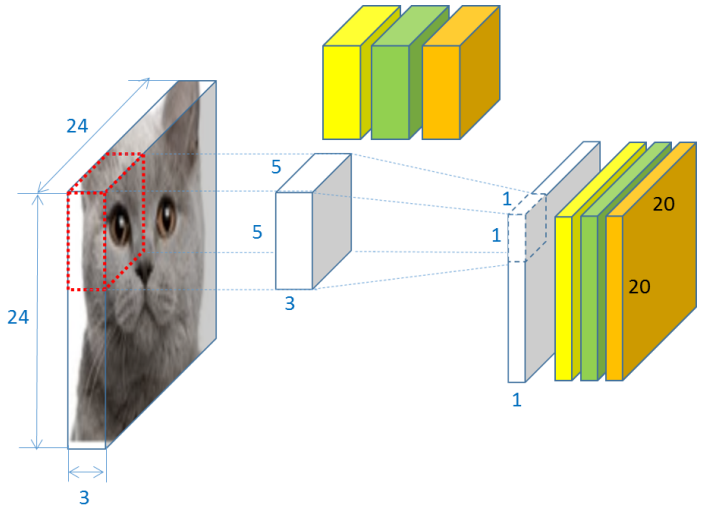
Summary

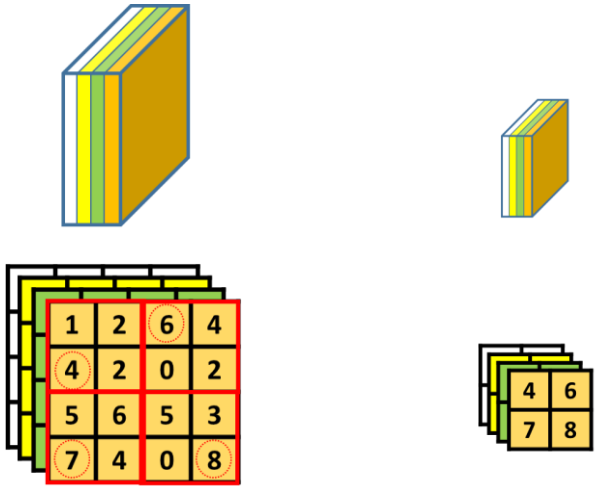
Summary



Convolution layer

Summary

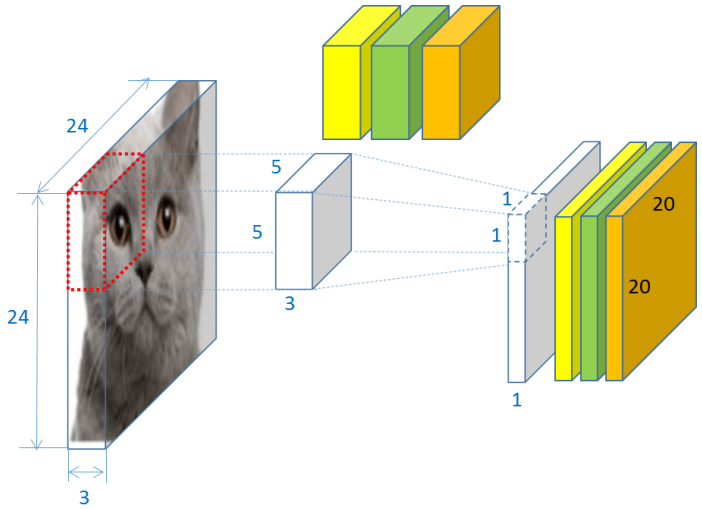


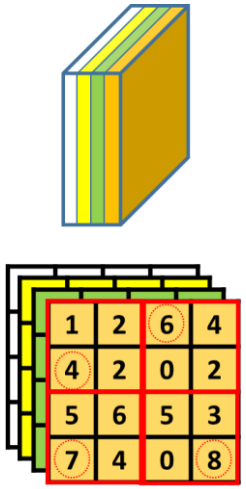


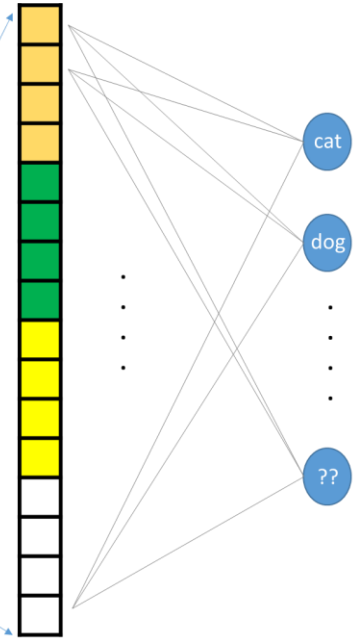
Convolution layer

Pooling layer

Summary







Convolution layer

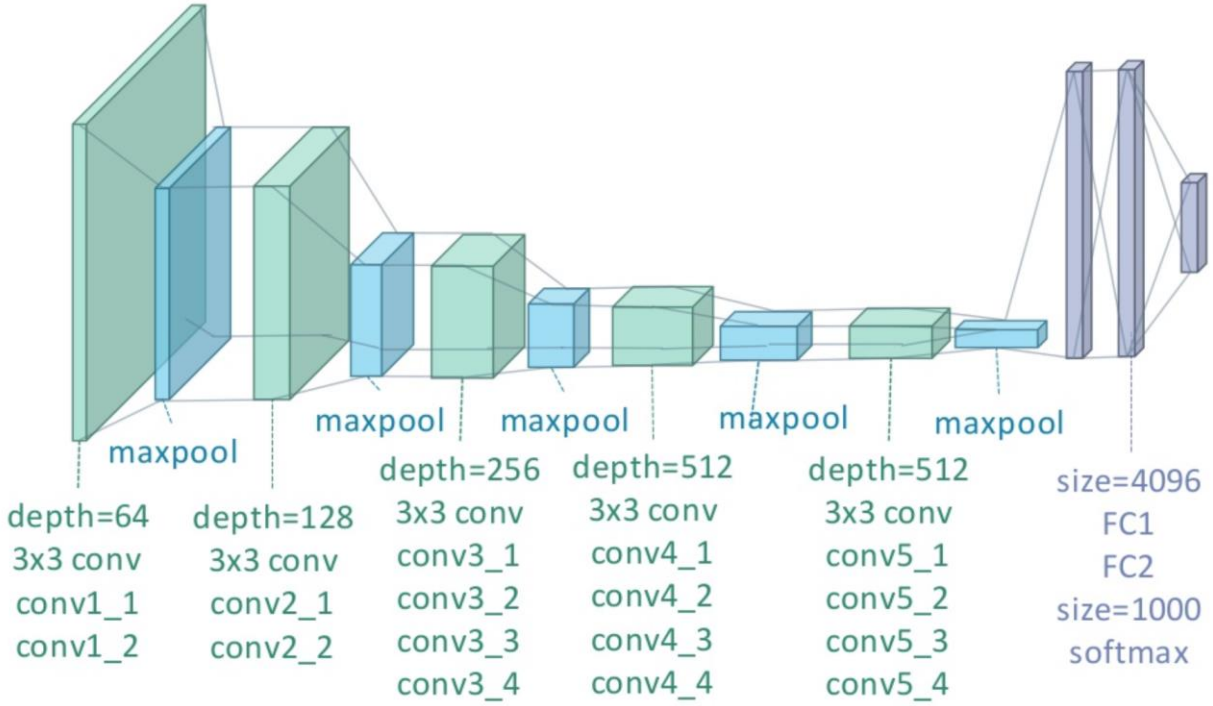
Pooling layer

Full connection layer

Image Style Transfer Using Convolutional Neural Networks

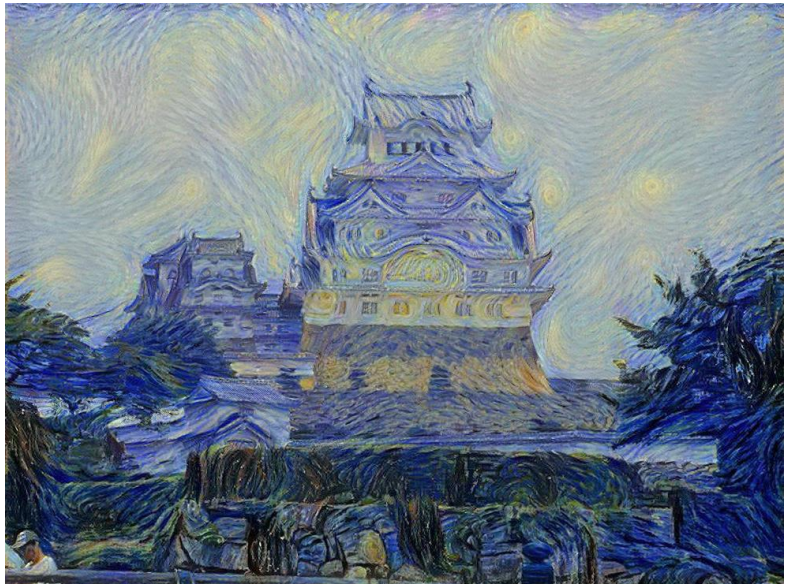
Style Transfer

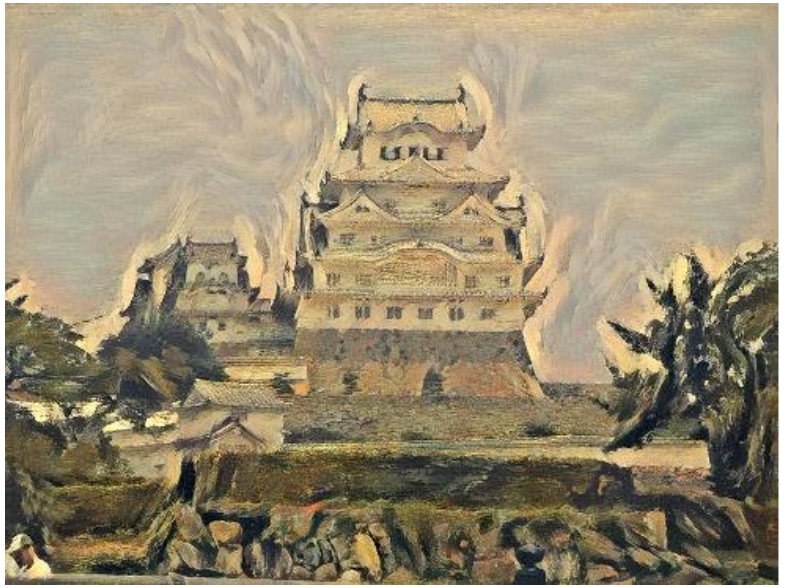
- Image Style Transfer Using Convolutional Neural Networks
  - https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf
- CNN is used to extract the style that a photo has and combine the style with the other photos.
- A pre-trained model
  - Runner-up of 2014 ImageNet Challenge competition
  - http://www.vlfeat.org/matconvnet/pretrained/



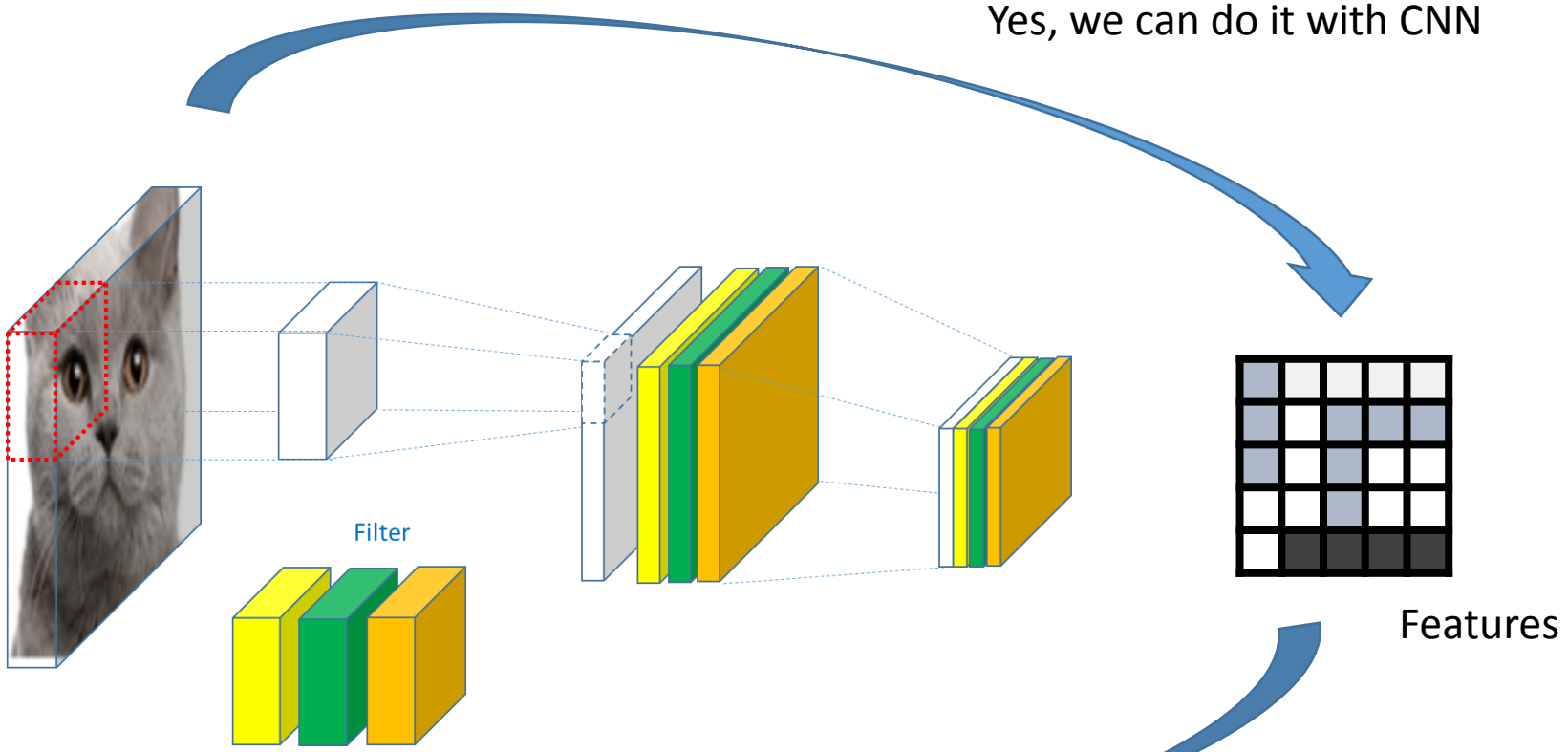
Style Transfer examples

Original image

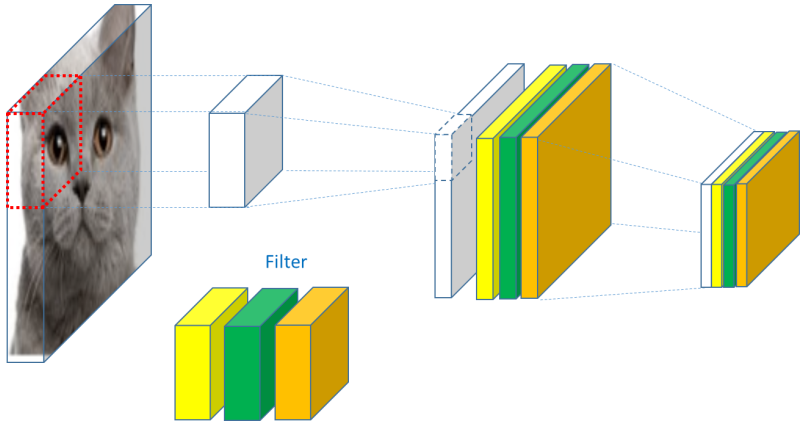




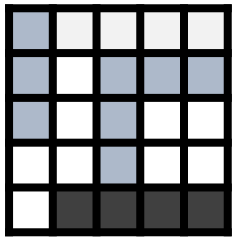
How does it work?

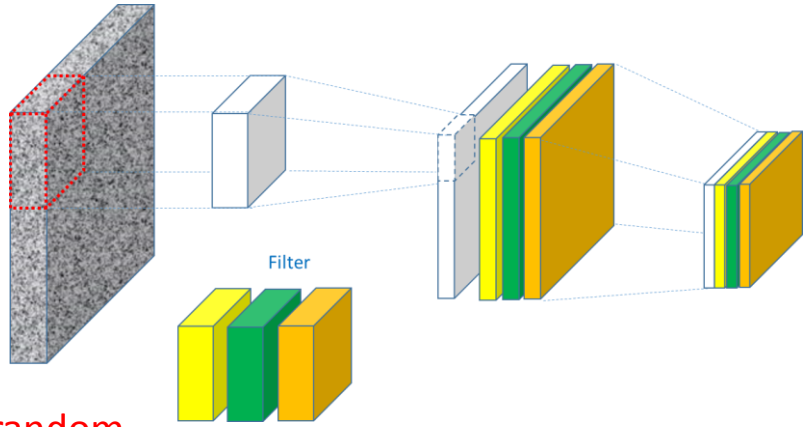


How does it work?



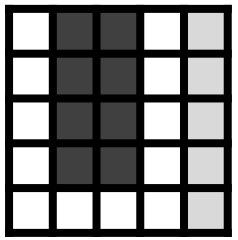
Feature (A)





random input

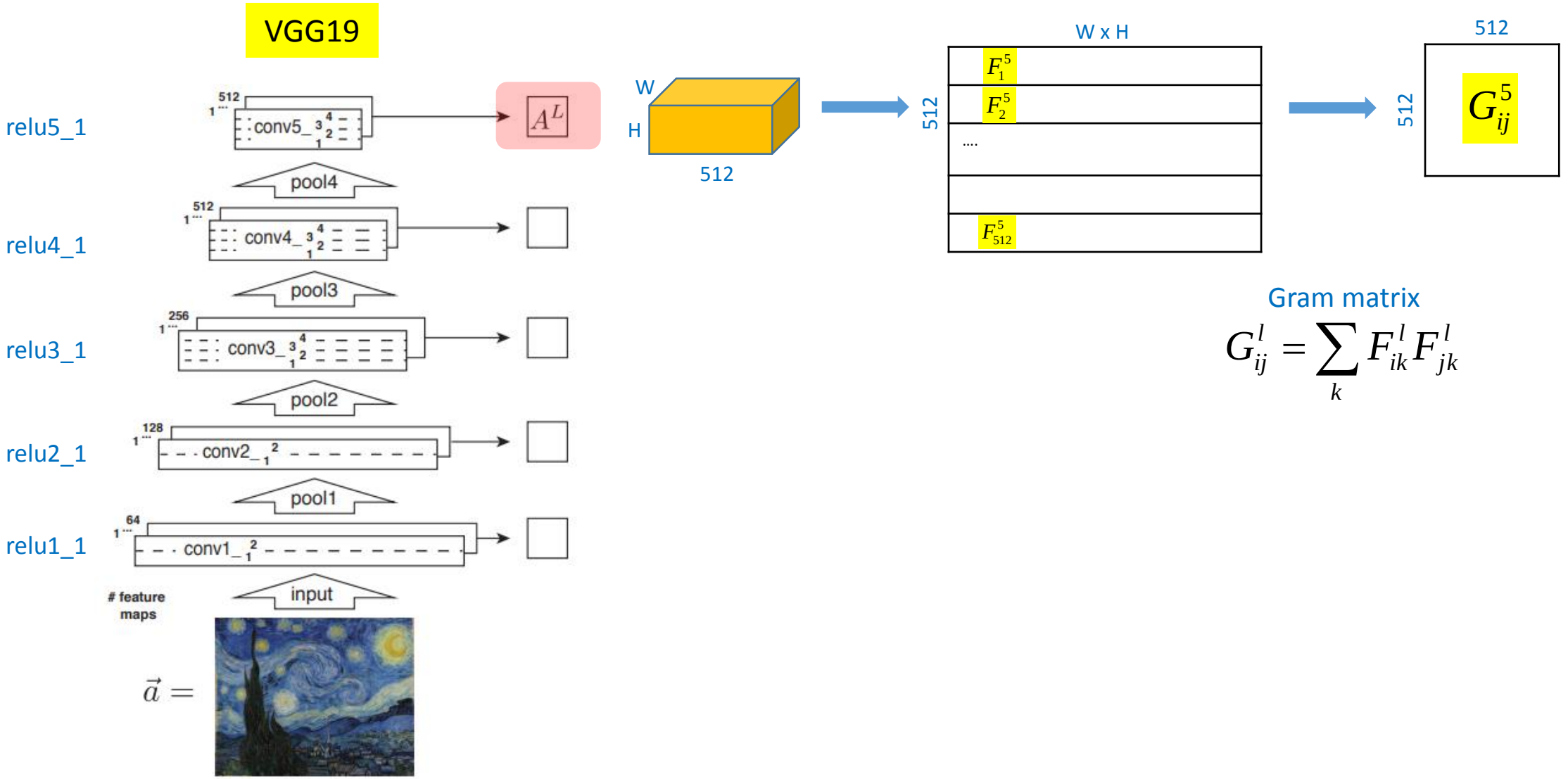
Feature (B)



- Given the random input, aim to evolve(?) the random input in a way that the features (A) and (B) become close (?).
  - If the feature (A) is good (representing the CAT well), any feature close to the feature (A) should be able to produce an image that is similar to the CAT image.
- The CNN is already trained one and so we expect that the feature well represents the input image.

Implementation

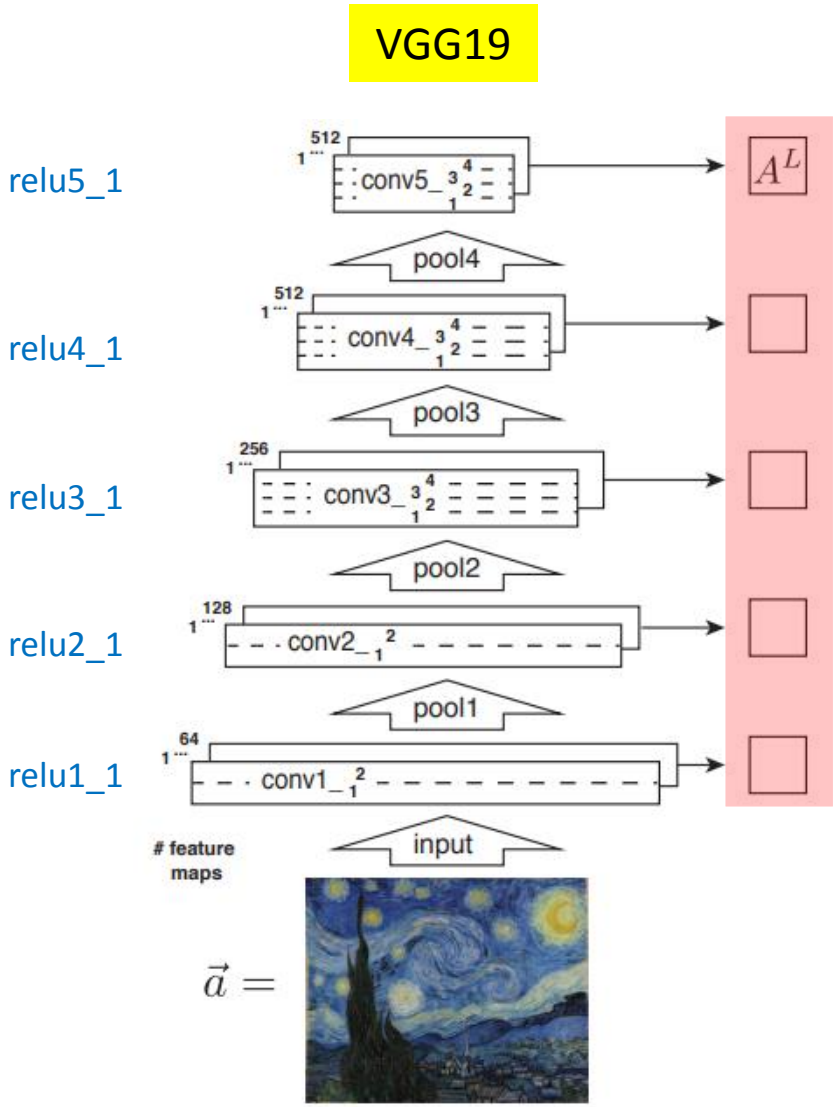
Visualization of features

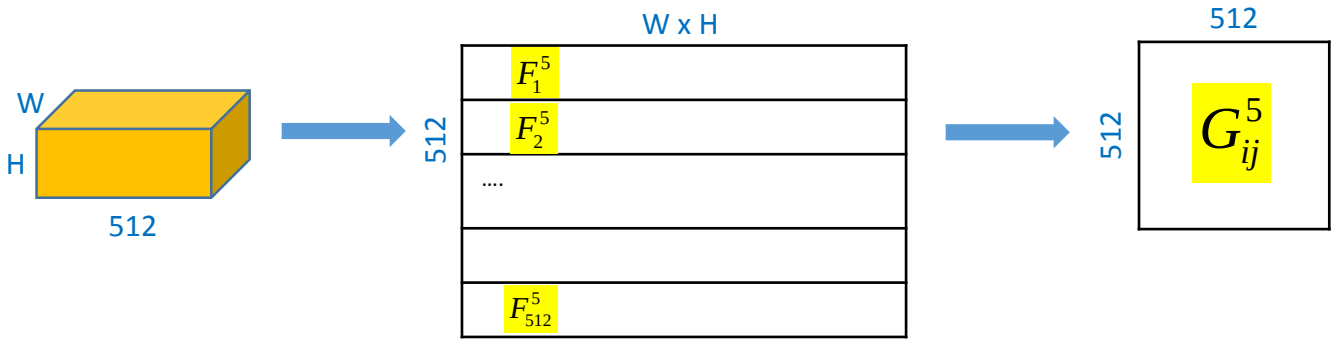


Gram matrix

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

Visualization of features





Gram matrix

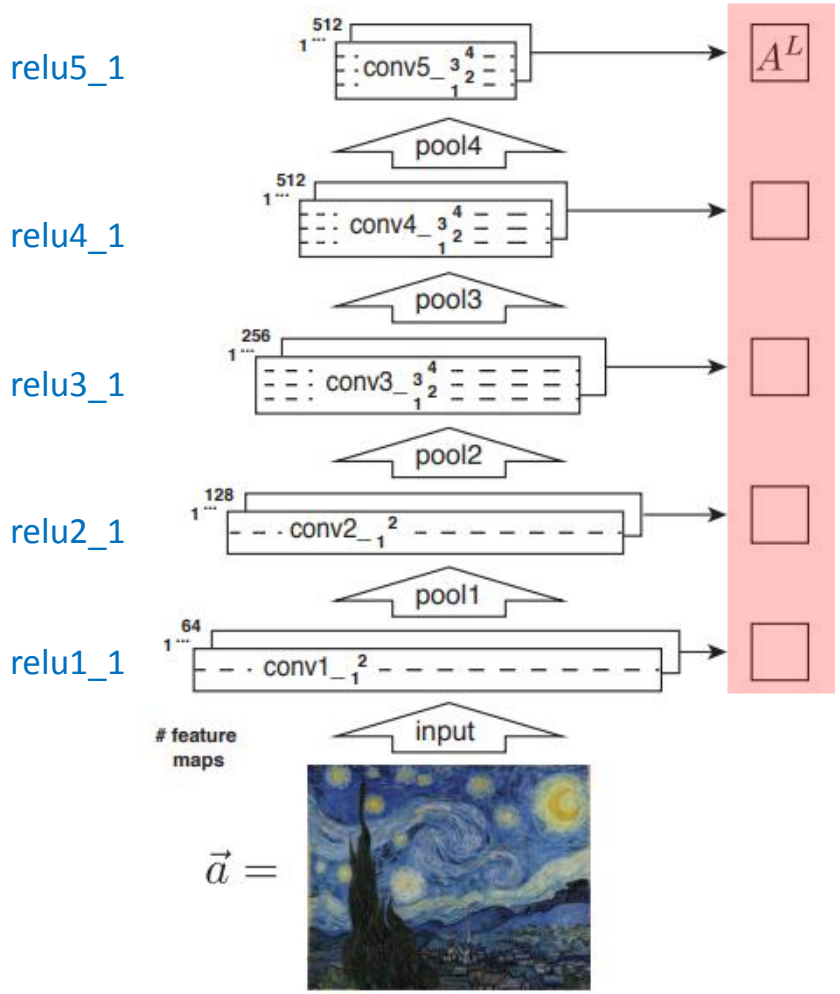
$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

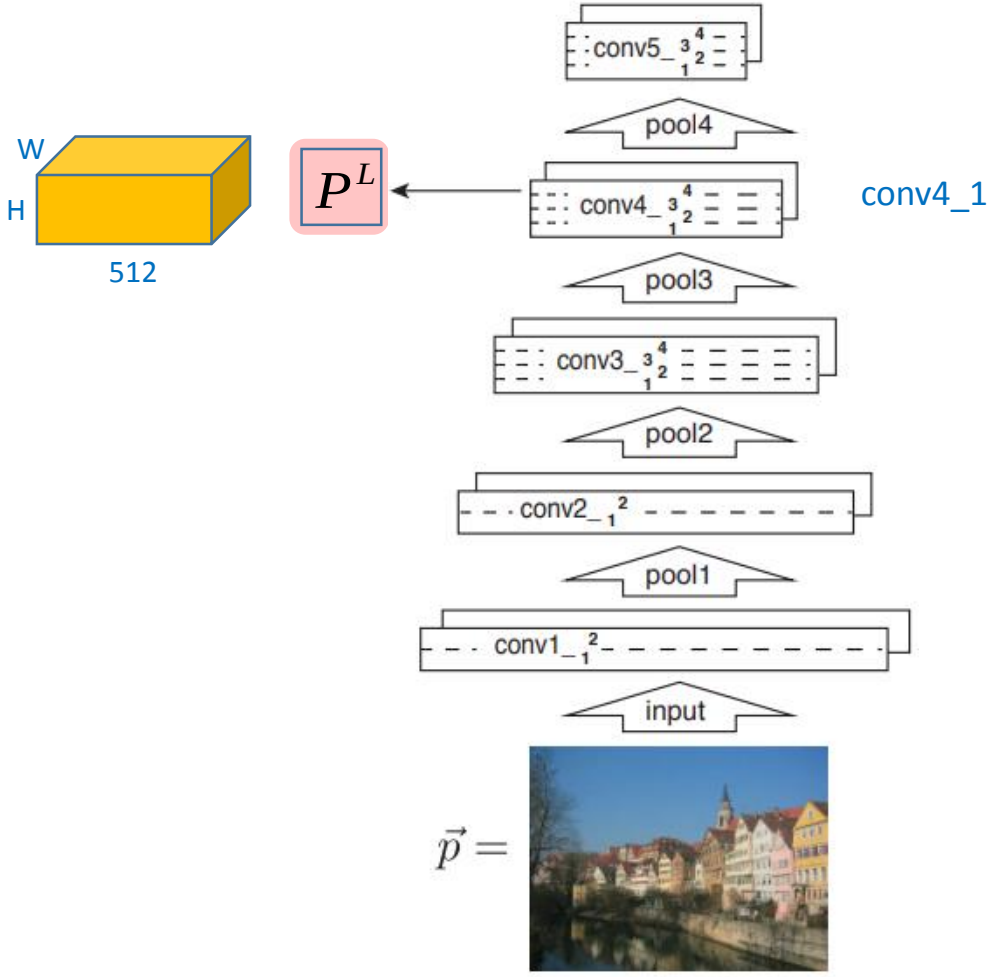
Why Gram matrix?

- Capturing correlation among features in the feature map.
- Strong correlation -> High value.

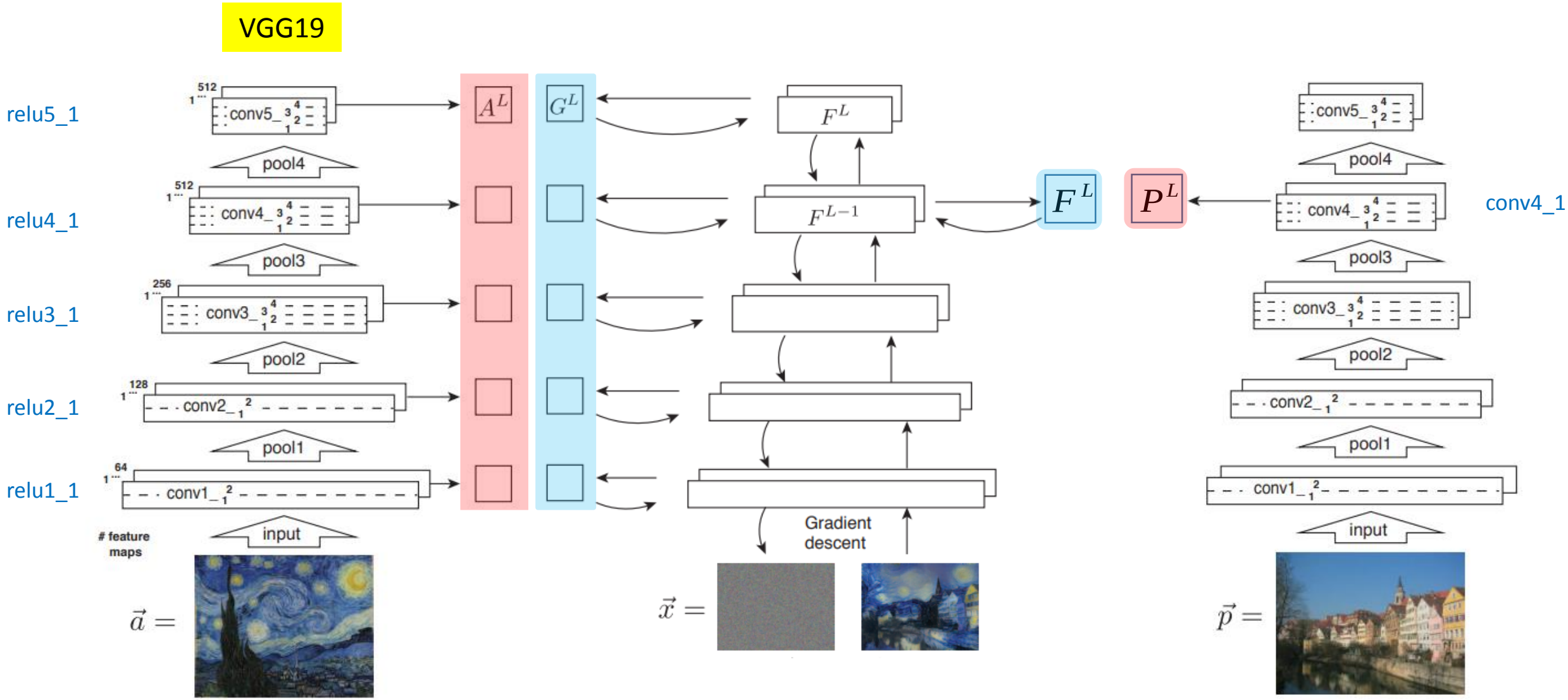
Visualization of features

VGG19





Visualization of features



Visualization of features

VGG19

$$E_S = \sum \left(G^L - A^L\right)^2$$

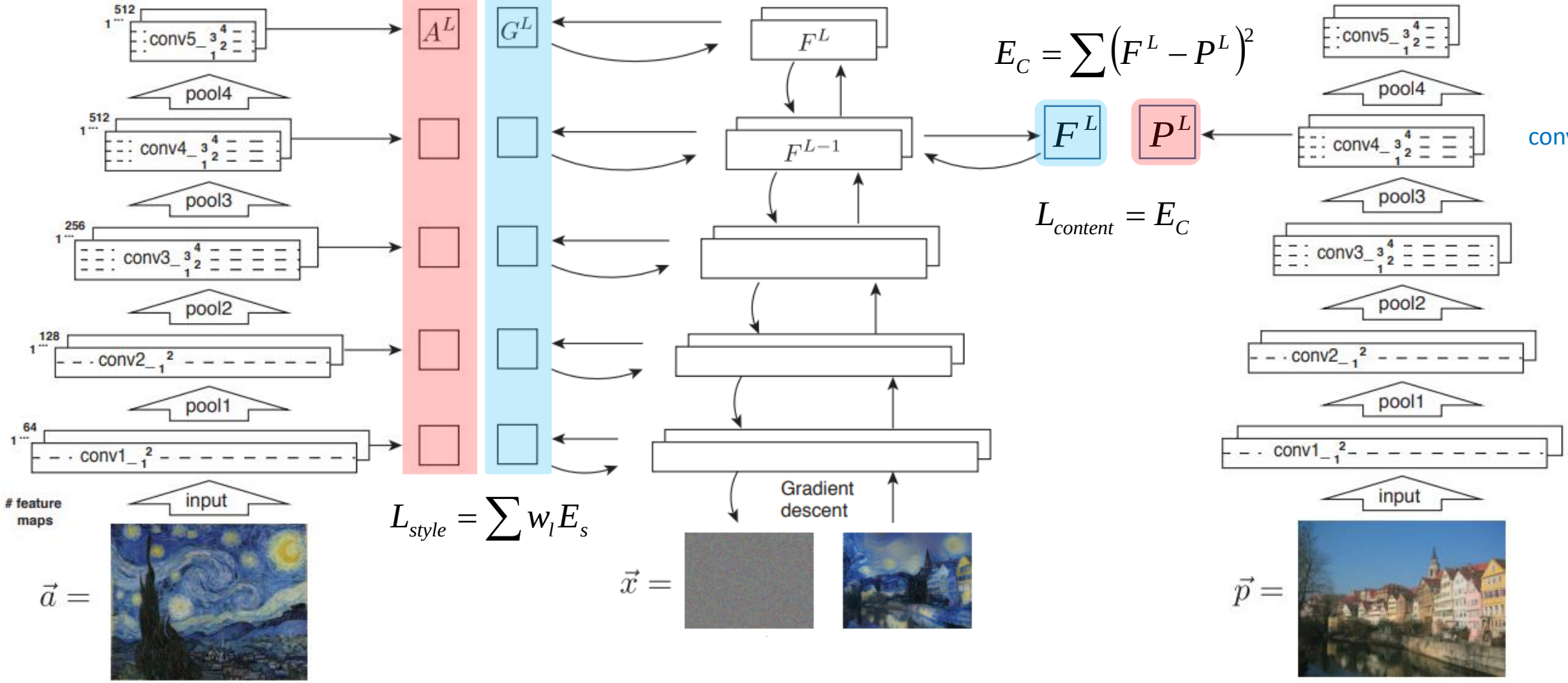
relu5_1

relu4_1

relu3_1

relu2_1

relu1_1



$$L_{style} = \sum w_l E_s$$

$$E_C = \sum \left(F^L - P^L\right)^2$$

$$L_{content} = E_C$$

conv4_1

$$L_{total} = \alpha L_{content} + \beta L_{style} = \sum w_l E_s + E_c$$

Visualization of features

VGG19

$$E_S = \sum \left(G^L - A^L\right)^2$$

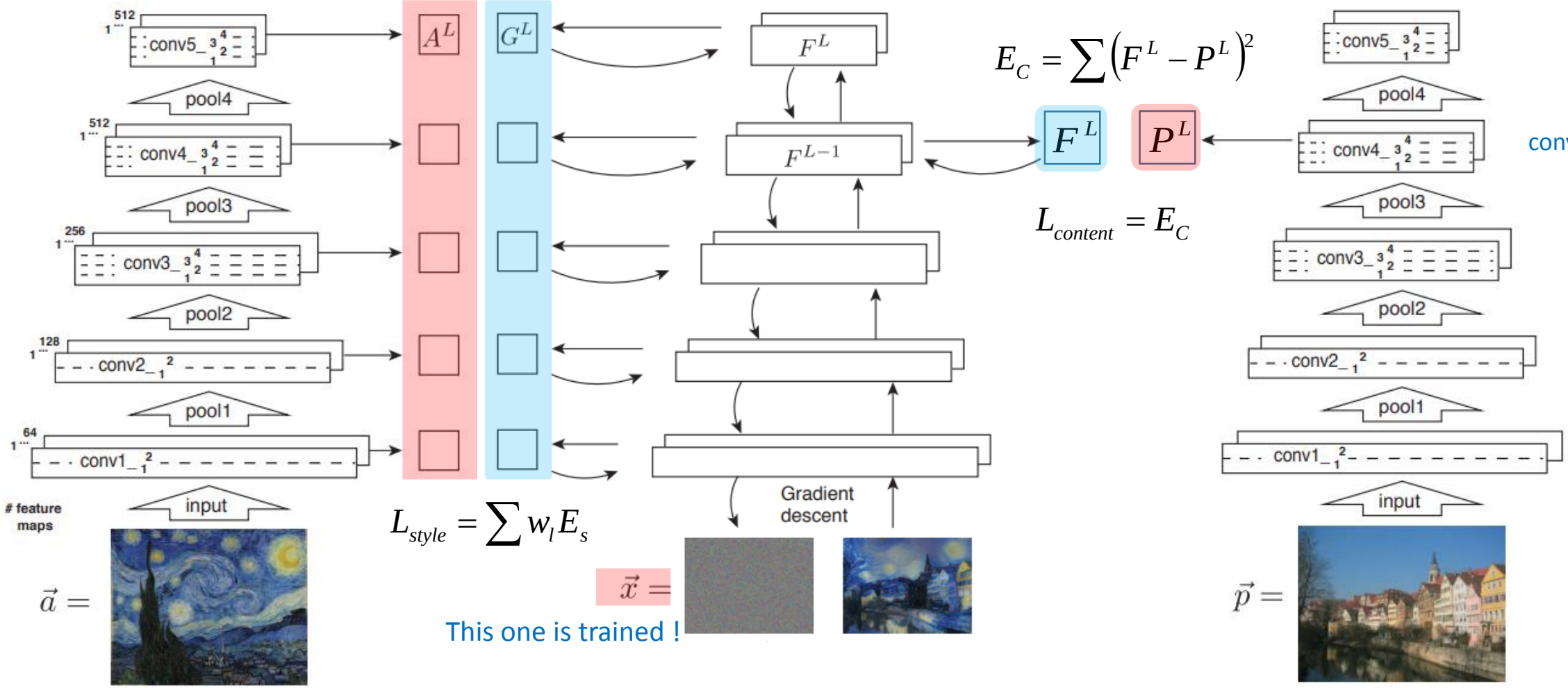
relu5_1

relu4_1

relu3_1

relu2_1

relu1_1



$$L_{style} = \sum w_l E_s$$

This one is trained !

$$E_C = \sum \left(F^L - P^L\right)^2$$

$$L_{content} = E_C$$

conv4_1

$$L_{total} = \alpha L_{content} + \beta L_{style} = \sum w_l E_s + E_c$$

Visualization of features

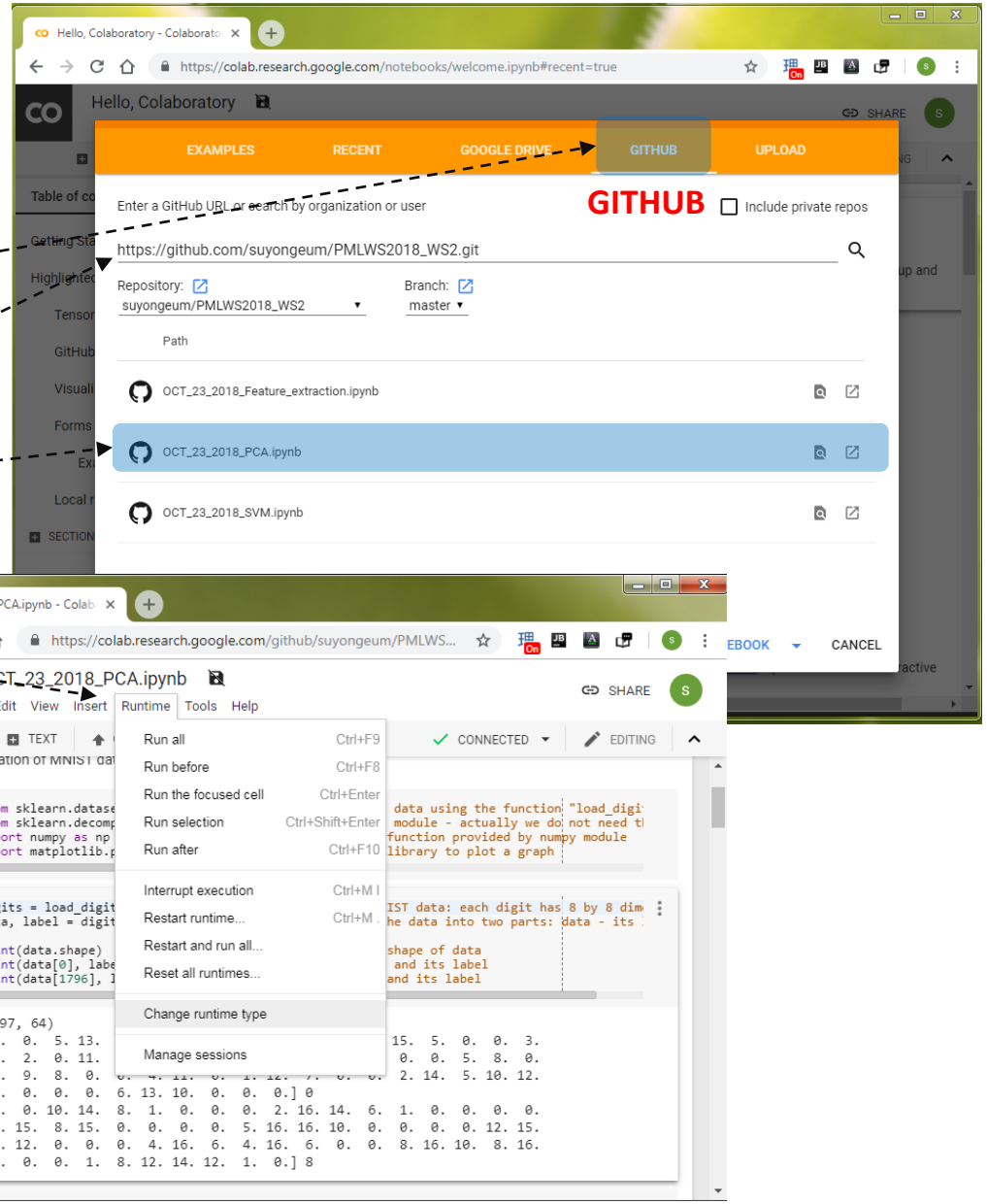
Figure 1: Original image and the reconstructed versions from maxpool layer 1,2 and 3 of Alexnet generated using tf_cnnvis.

- https://github.com/InFoCusp/tf_cnnvis

Hand-on Experience

Colab: Style Transfer

1) Go to the Colab
   - https://colab.research.google.com
2) Select “GITHUB” and copy the link below into
   - https://github.com/suyongeum/PMLWS2018_WS3.git
3) Select the notebook in the list
   - Nov_6_2018_StyleTransfer.ipynb
4) Go to “Runtime” – “Change runtime type”
   - Python 3
   - GPU
5) Save it into your gdrive
   - “File” - “Save a copy in Drive …”



Practices

- GPU vs CPU
  - Running time measurement
- Hyper parameters
  - Location of the layers
  - Weight between content and style layers
  - Weights of style layers
  - Etc…
- Using your photo or different style photo

```
import datetime
before = datetime.datetime.now().timestamp()

…

after = datetime.datetime.now().timestamp()
print( “Time taken:”, after – before)
```