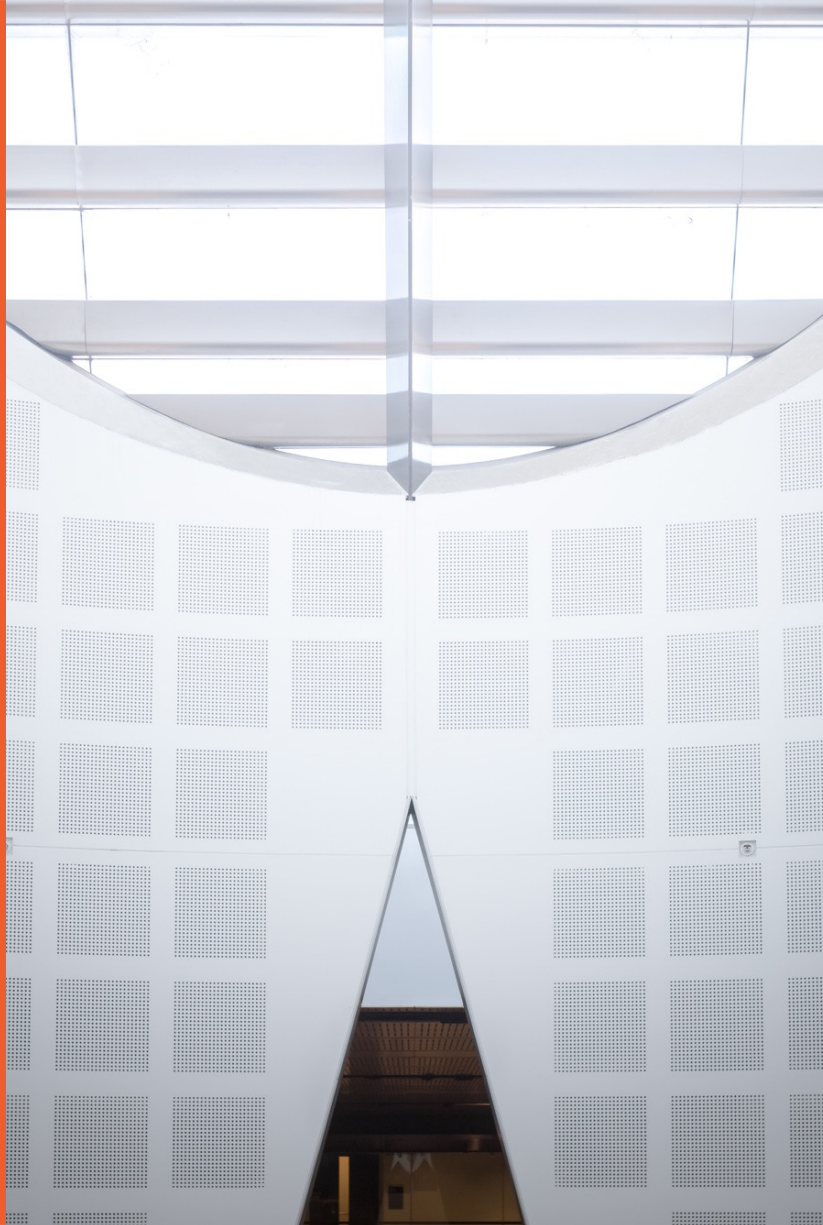
Convolutional Neural Networks

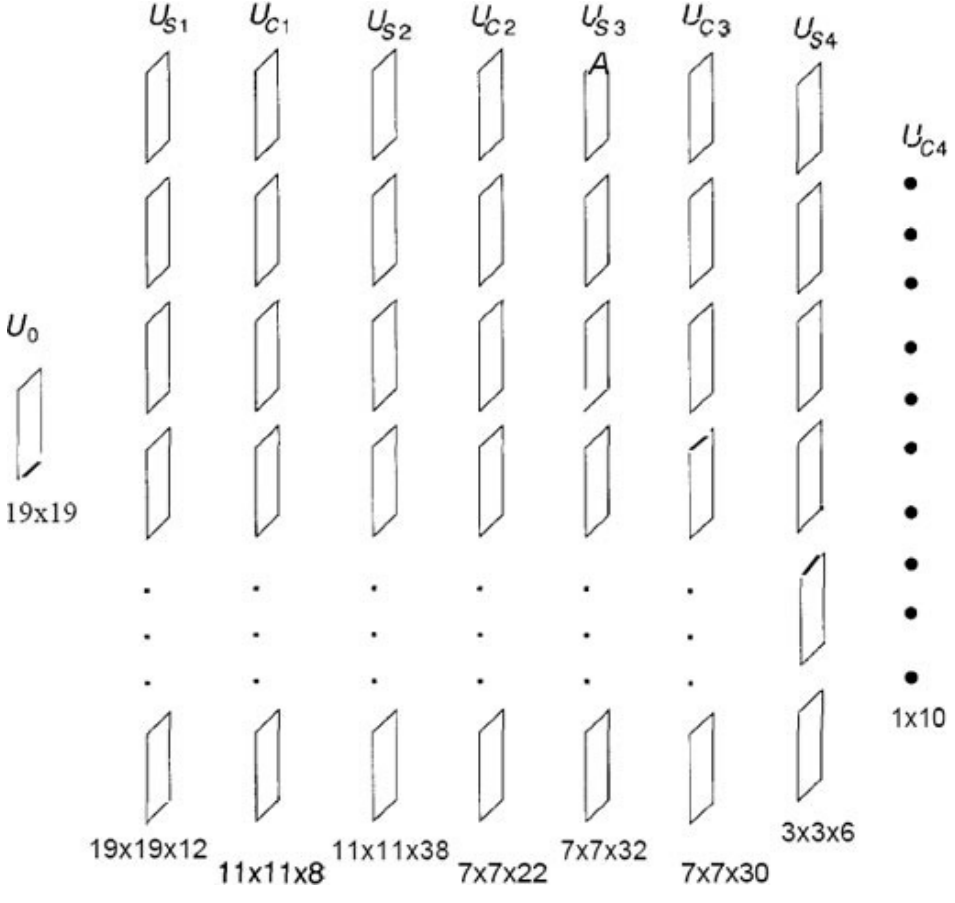
Dr Chang Xu
School of Computer Science

THE UNIVERSITY OF SYDNEY



History of CNNs

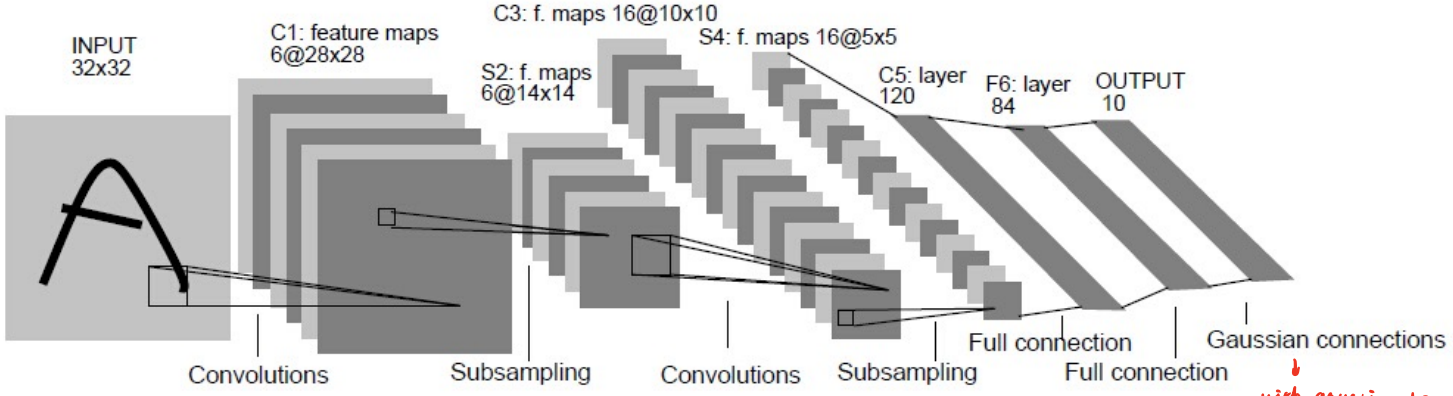
Neocognitron（Kunihiko Fukushima, 1980）



History of CNNs

LeNet-5（LeCun et al, 1998）

- Built the modern framework of CNNs: Convolutional Layer, Pooling Layer, and Fully-Connected Layer
- Trained with the backpropagation algorithm
- Classify handwritten digits. However, it can not perform well on more complex problems, e.g., large-scale image and video classification



History of CNNs

Linear classifier: 8% ~ 12% error

K-nearest-neighbor: 1.x% ~ 5% error

introduce nonlinear kernel

Support Vector Machine: 0.6% ~ 1.4% error

(Conventional) Neural Nets: 1% ~ 5% error

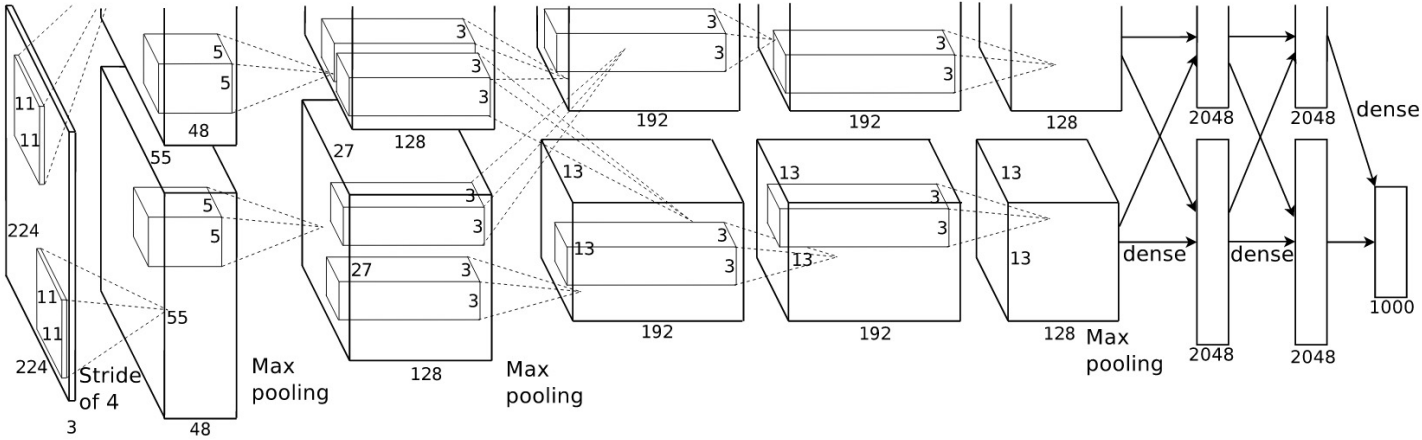
"The MNIST Database"

History of CNNs

AlexNet（Krizhevsky et al, 2012）

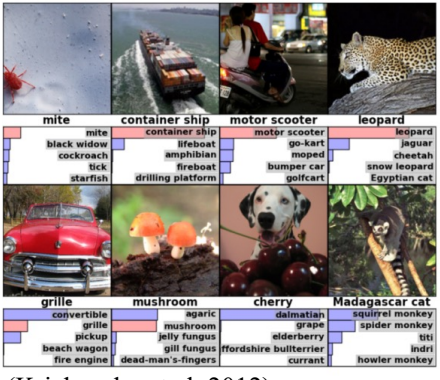
- Significant improvements on the image classification task, ImageNet 2012
- The network achieved a top-5 error of 15.3%, more than 10.8 percentage points ahead of the runner up.
- Basic framework of CNNs with a deeper structure
- Benefit from ImageNet dataset, GPUs, ReLU, Dropout …

5 convolutional layers and 3 fully connected layers

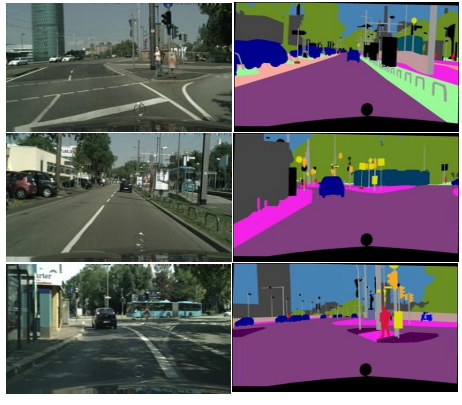


Today, CNNs are everywhere

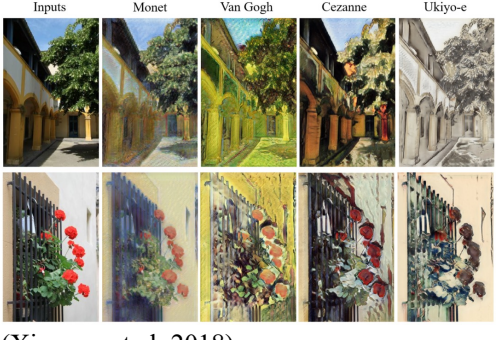
- Image classification, Image segmentation, Pose estimation, Style transfer, Image detection, Image caption …



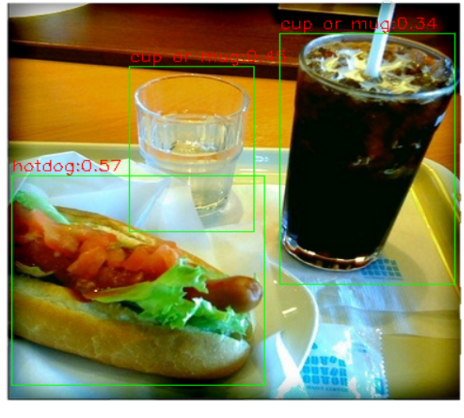
(Krizhevsky et al, 2012)



(Shaoli et al, 2017)



(Xinyuan et al, 2018)



Large black and white dog jumping hurdle

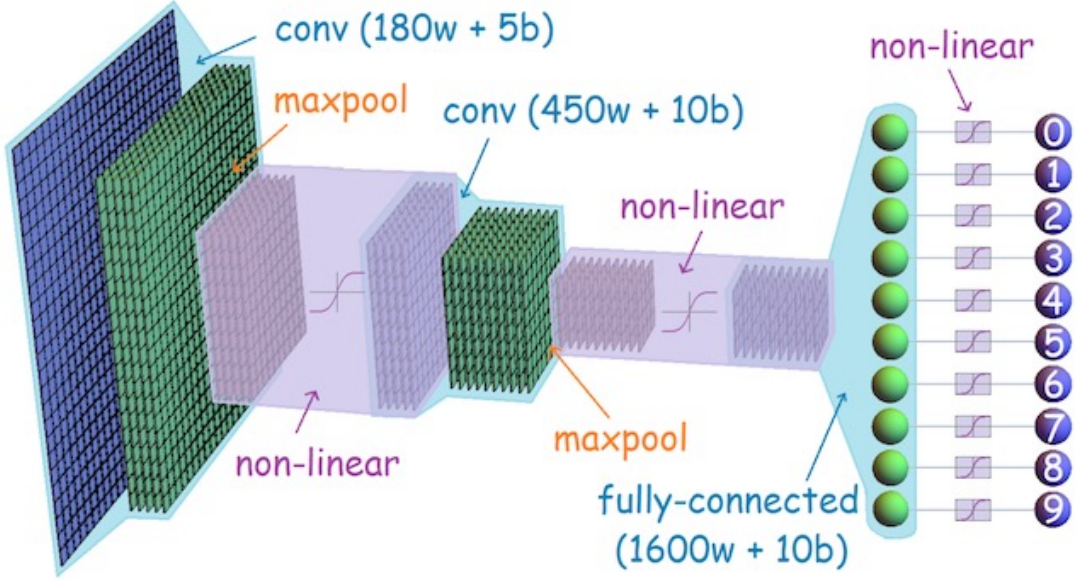
(Jianfeng et al, 2017)

A man in a red shirt is playing a saxophone outside a business

Basic CNNs Components

A general CNN

- **Convolutional Layer**
- **Pooling**
- Fully-connected Layer → MLP



(https://leonardoaraujosantos.gitbooks.io)

A toy example

https://github.com/pytorch/examples/blob/master/mnist/main.py

https://pytorch.org/docs/stable/nn.html#convolution-layers

https://pytorch.org/docs/stable/nn.html#linear

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

Convolution layers in PyTorch

https://pytorch.org/docs/stable/nn.html#convolution-layers

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

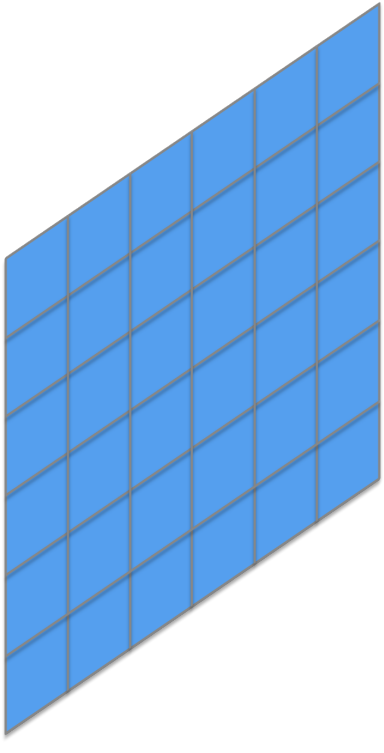
need to optimize ↑ input signal.

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolutional Layer

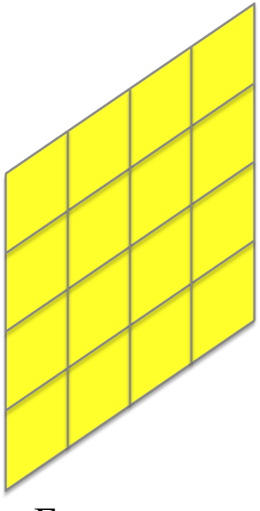
- Give a simple example: take a grayscale image as input



Grayscale Image: $\boldsymbol{X}$

$w_{0,1}$	$w_{0,2}$	$w_{0,3}$
$w_{1,1}$	$w_{1,2}$	$w_{1,3}$
$w_{2,1}$	$w_{2,2}$	$w_{2,3}$

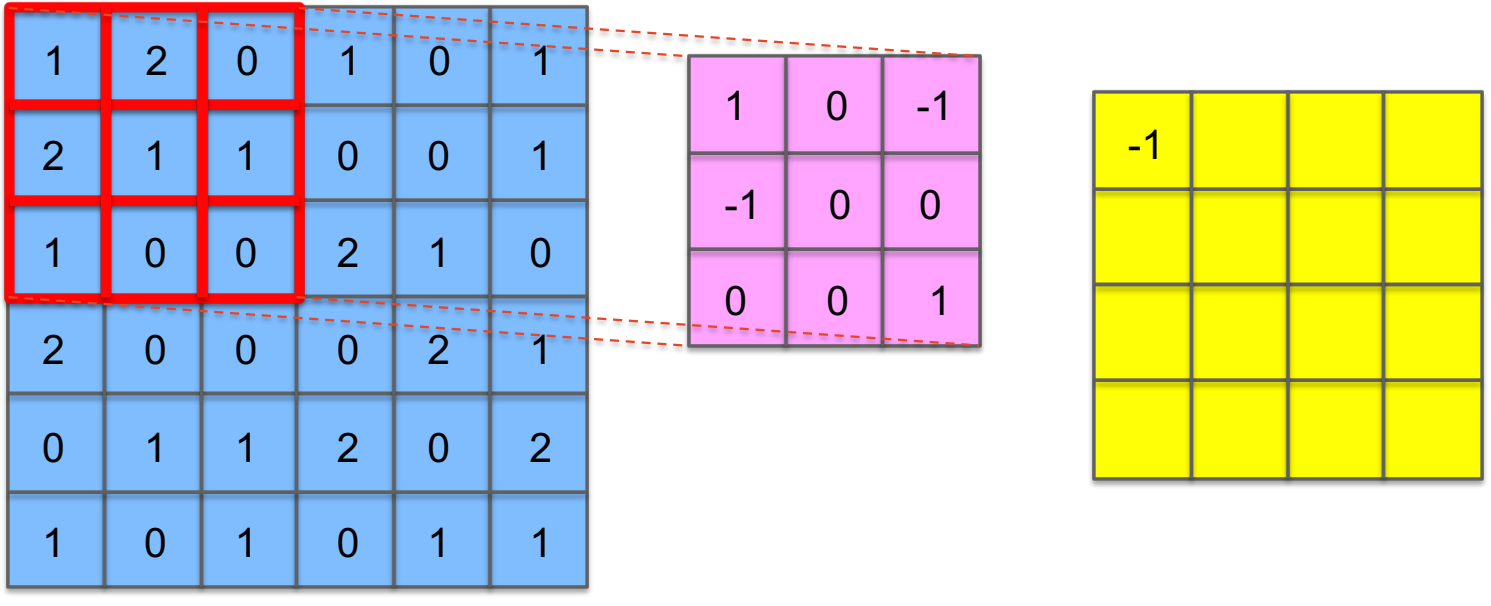
Filter: $\boldsymbol{w}$



Feature

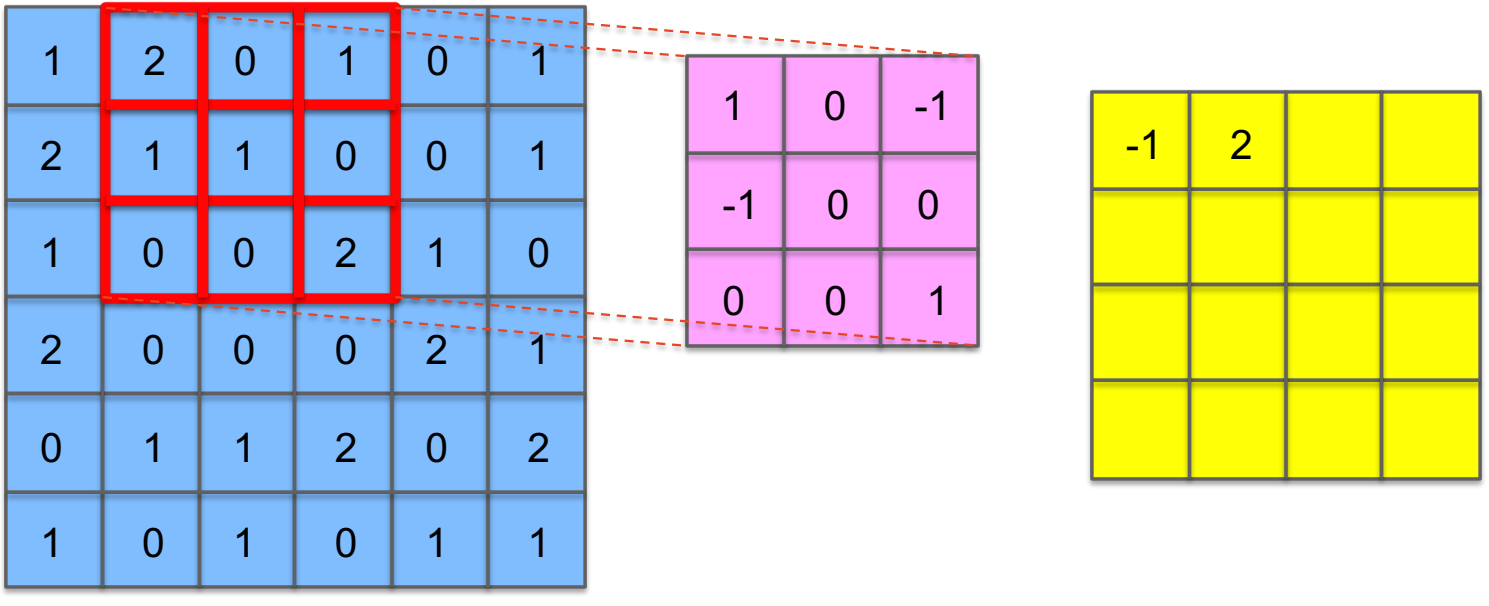
Convolutional Layer

- Convolution



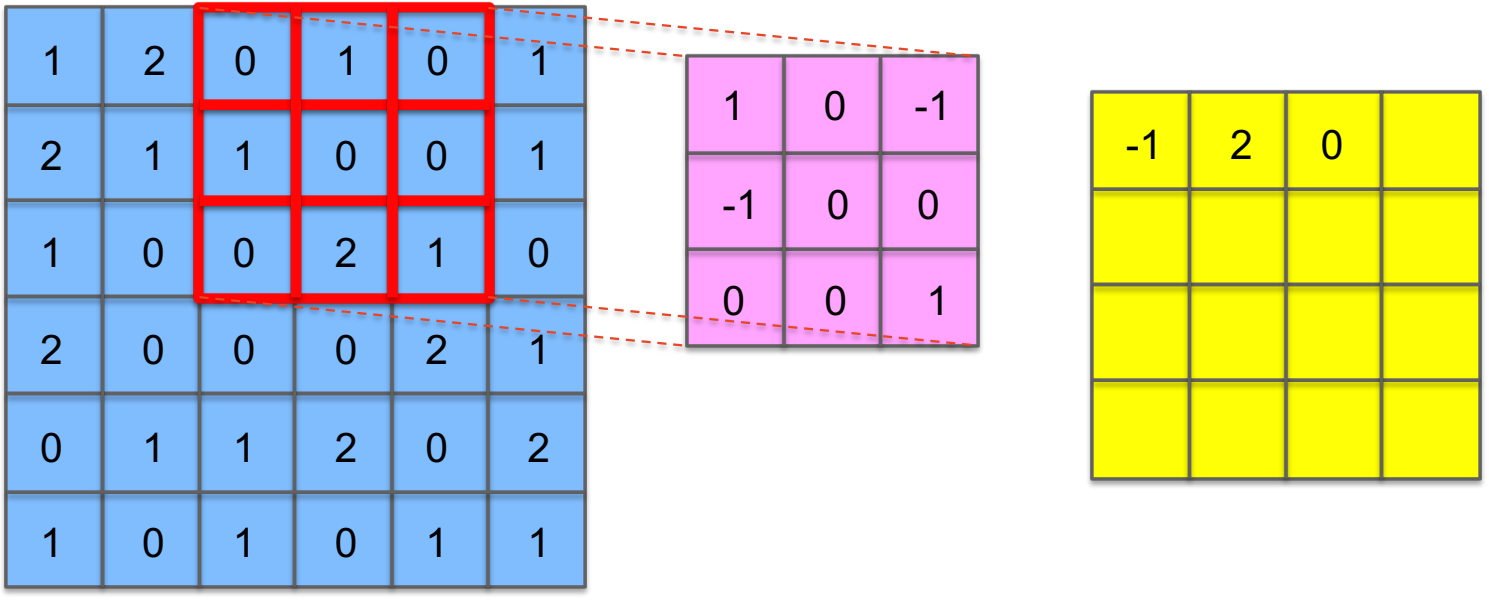
Convolutional Layer

- Convolution



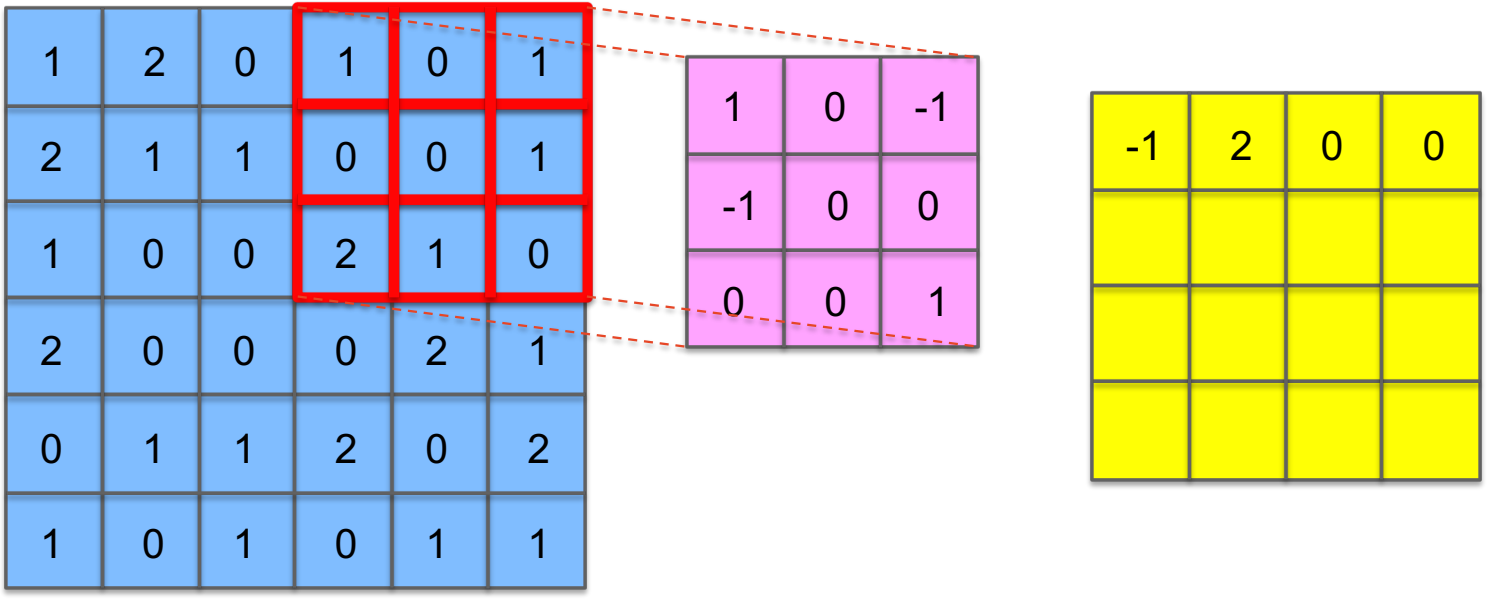
Convolutional Layer

- Convolution



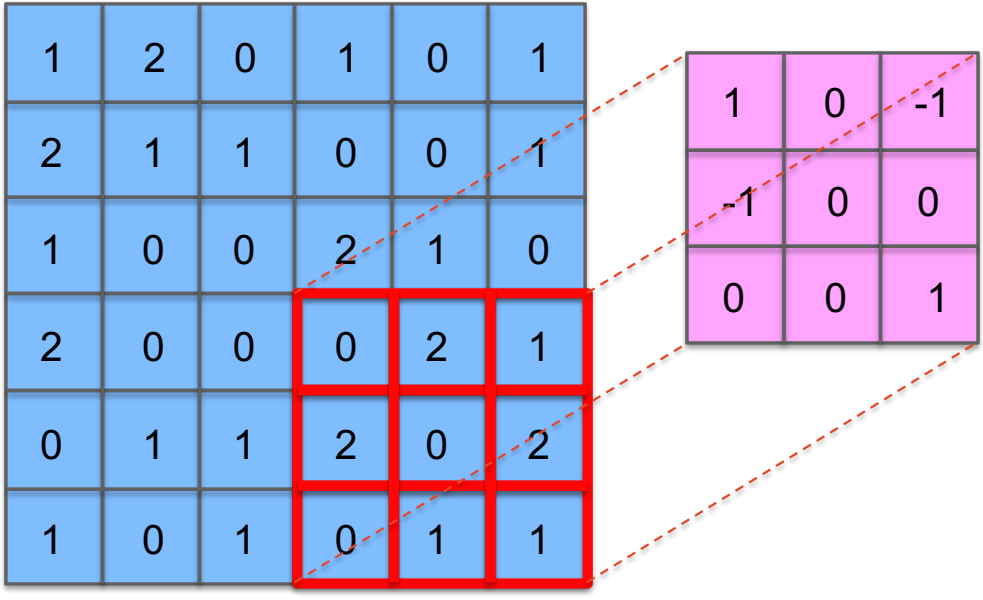
Convolutional Layer

- Convolution



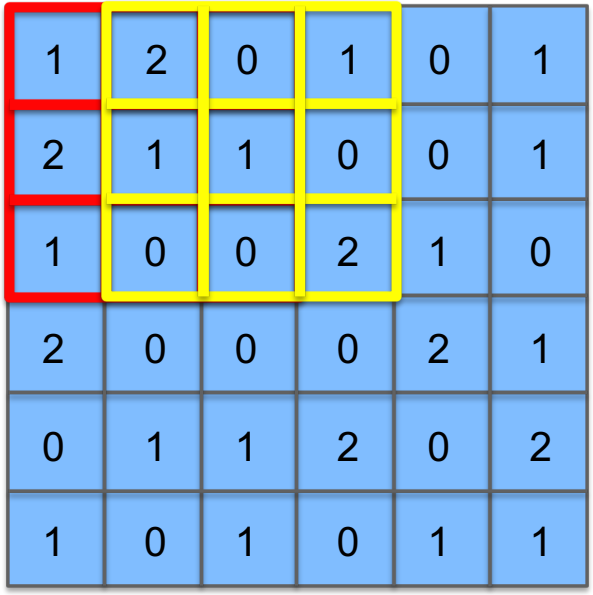
Convolutional Layer

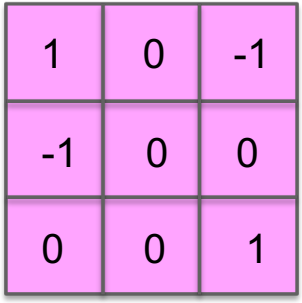
- Convolution



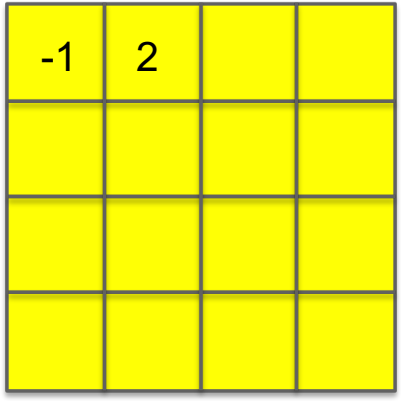
Convolutional Layer

- Stride





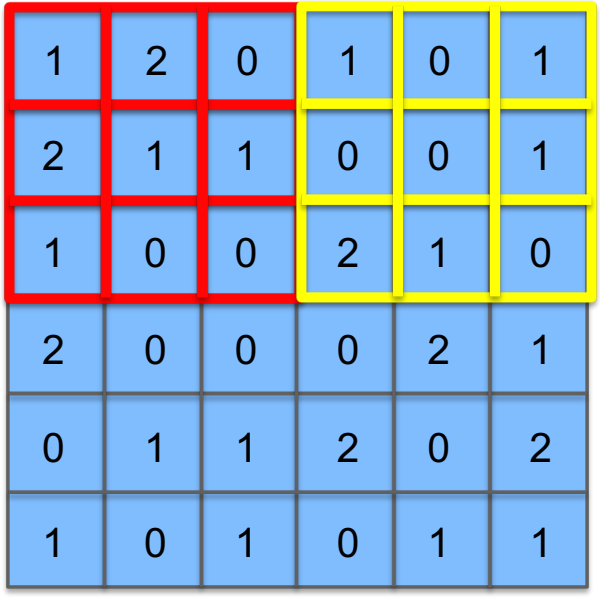
Stride = 1

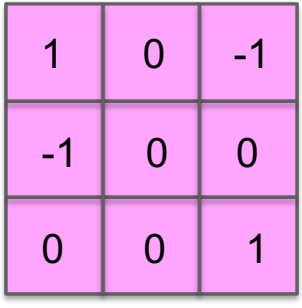


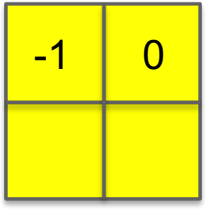
The *stride size* is defined by how much you want to shift your filter at each step.

Convolutional Layer

- Stride







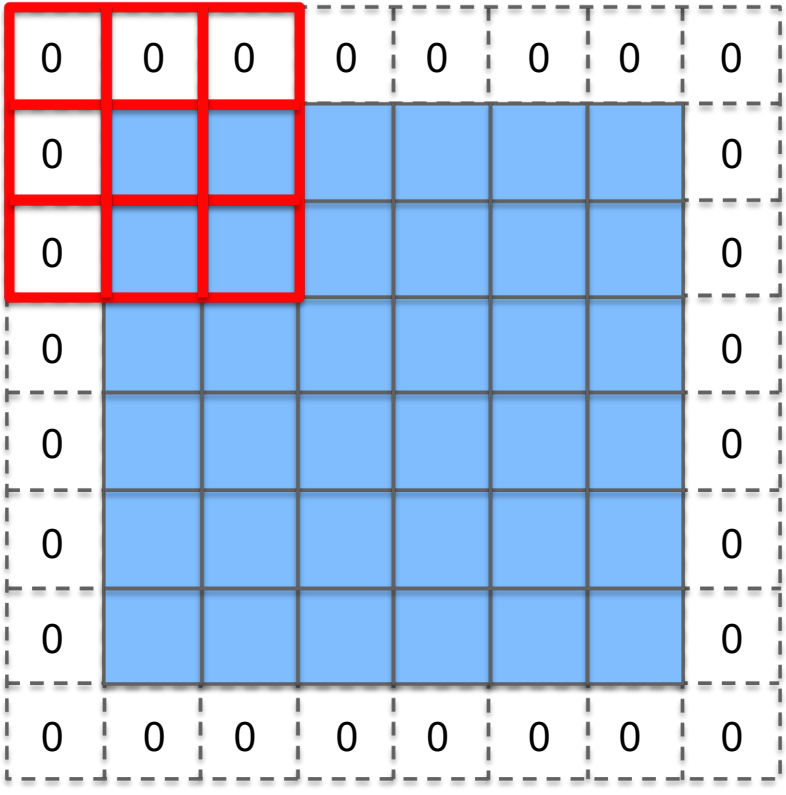
Stride = 3

Convolutional Layer

- Zero padding (pad = 1)

pad with 0 size for pad : 1 for only 1 left, right, top, bottom

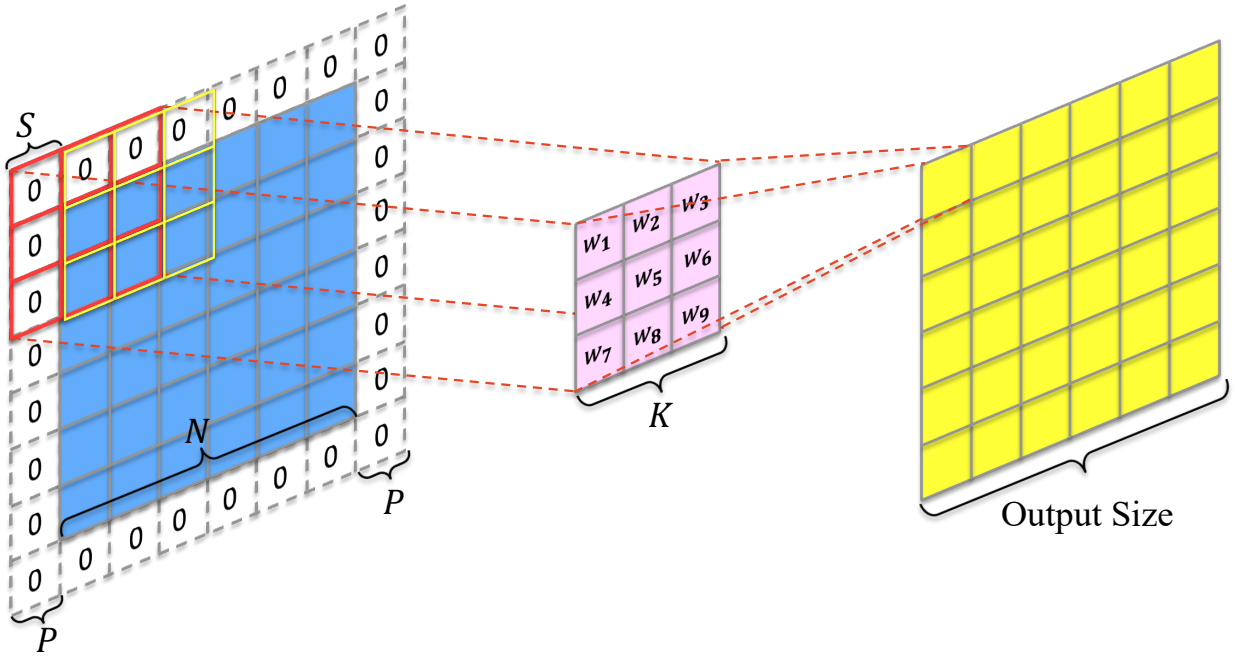
By doing this you can apply the filter to every element of your input matrix.



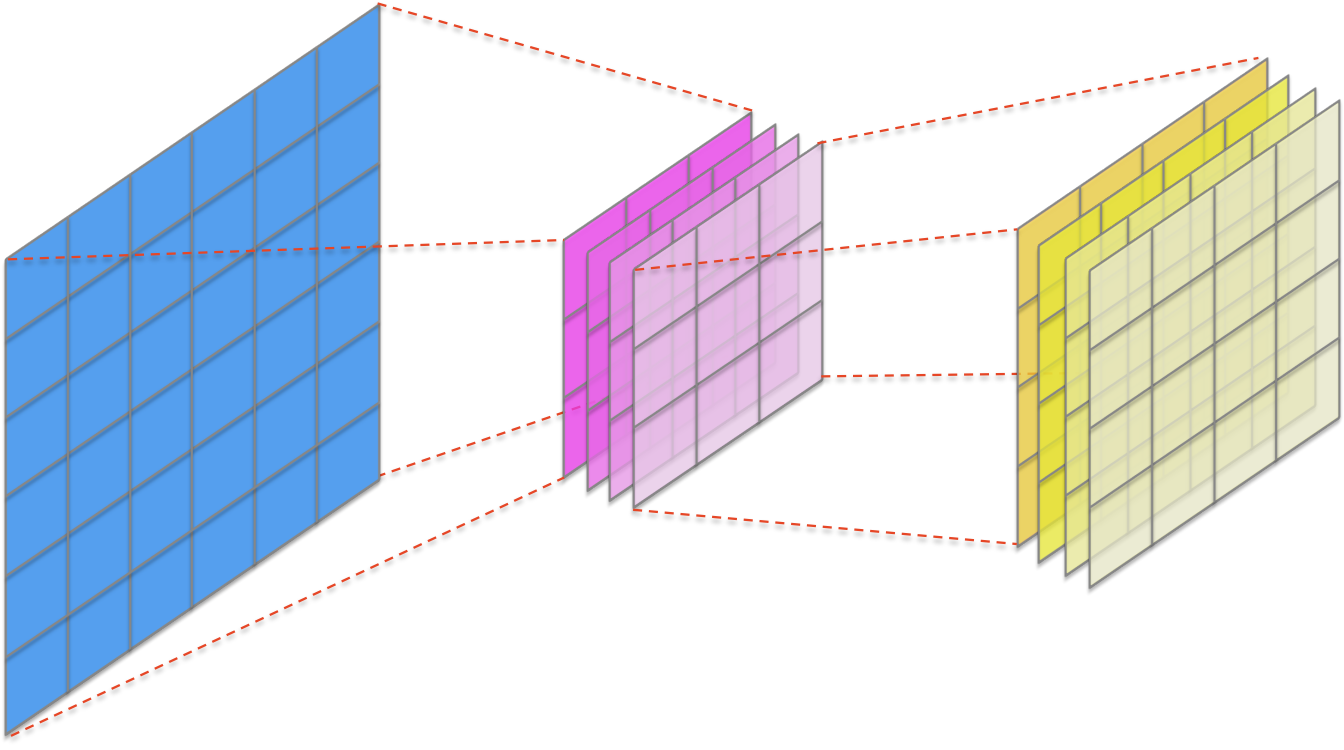
Convolutional Layer

- Output Size

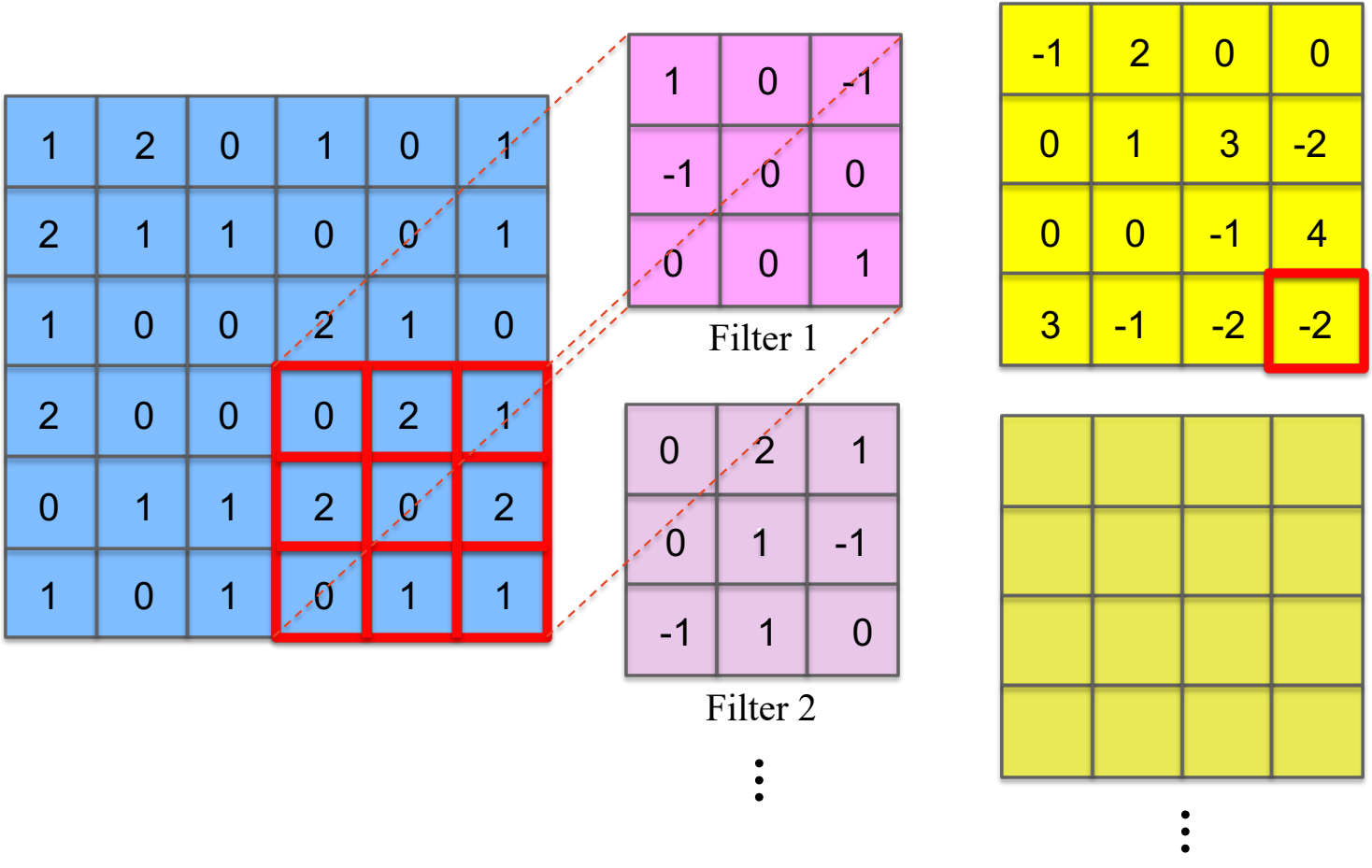
$$\text{Output Size} = \frac{N + 2P - K}{S} + 1$$



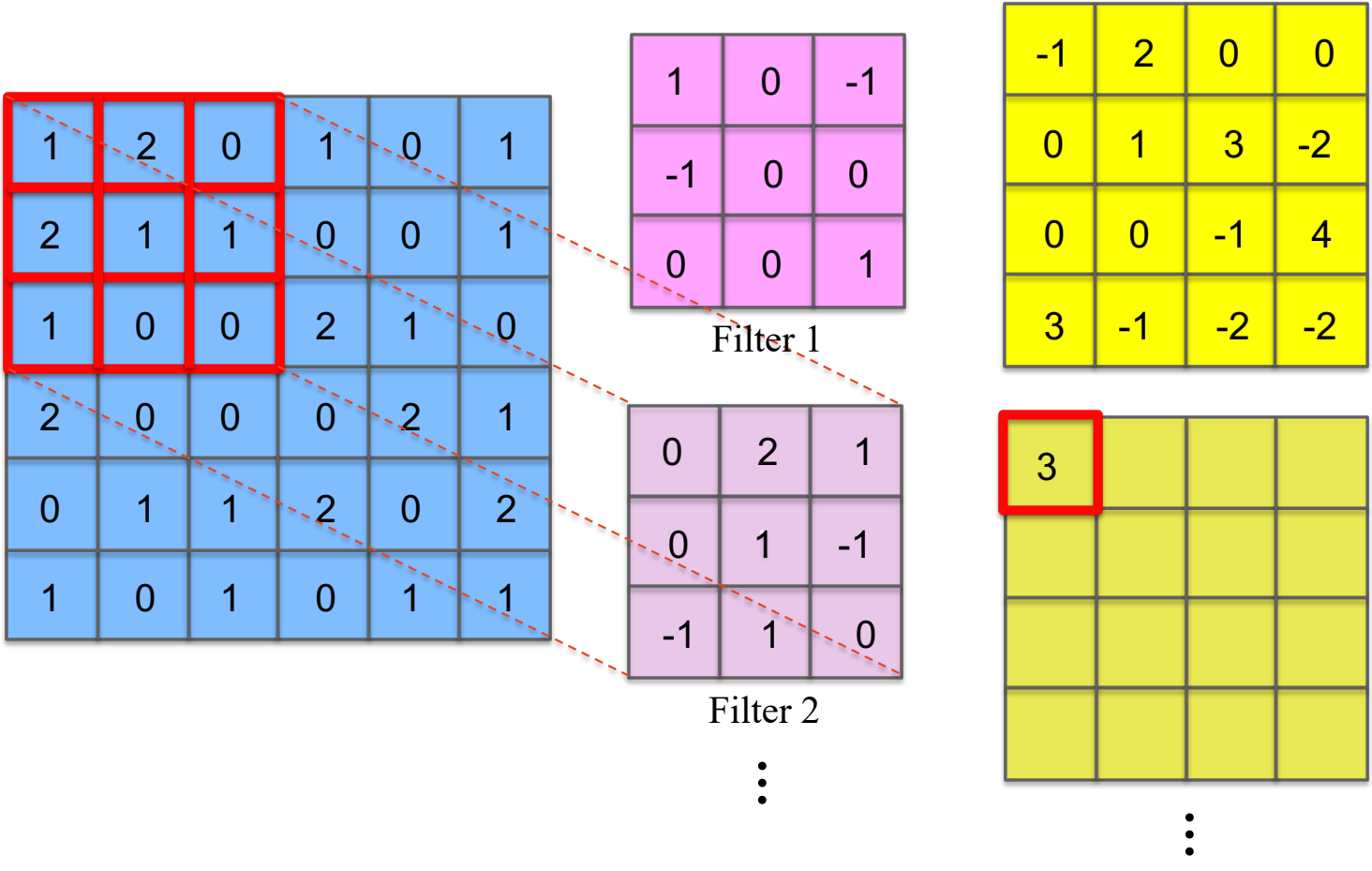
Learn multiple filters



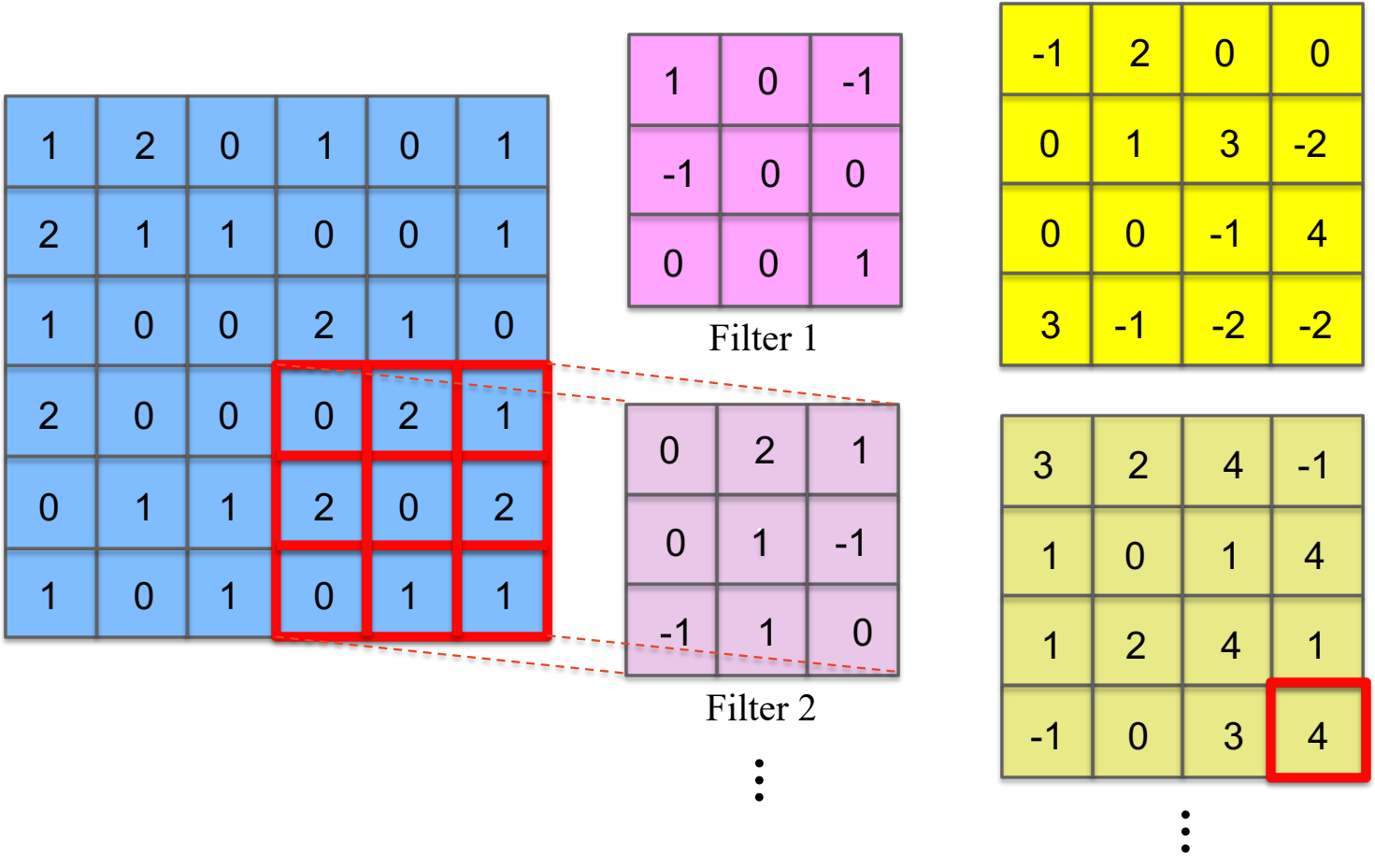
Learn multiple filters



Learn multiple filters

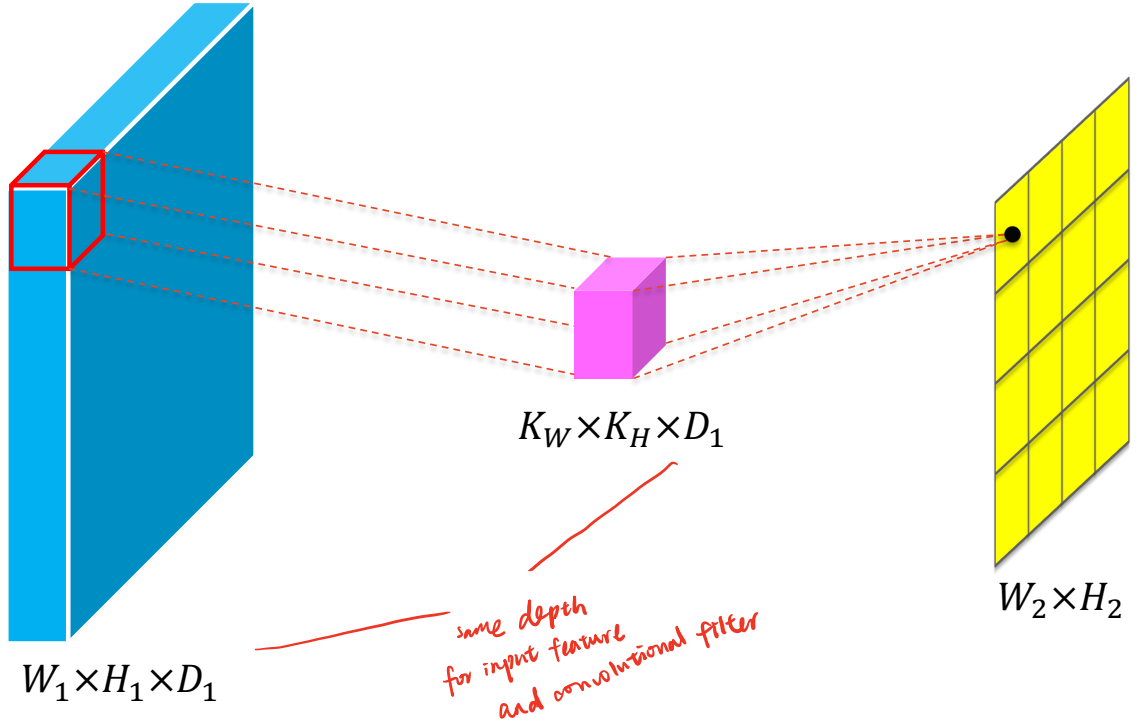


Learn multiple filters

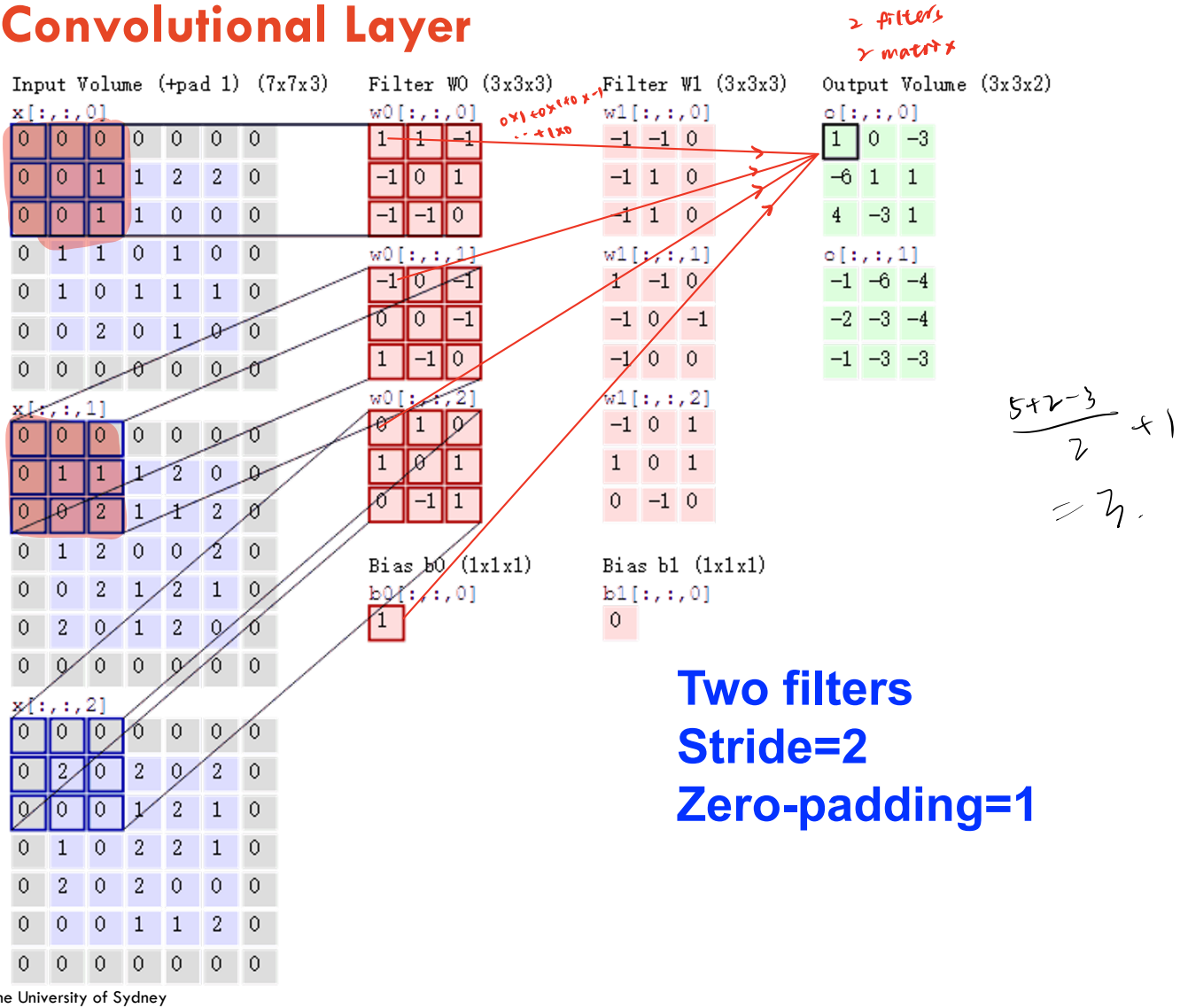


Convolutional Layer

- Above, we have only considered a 2-D image as input
- When the input has depth (e.g. RGB images), the convolution ops should be…



Convolutional Layer



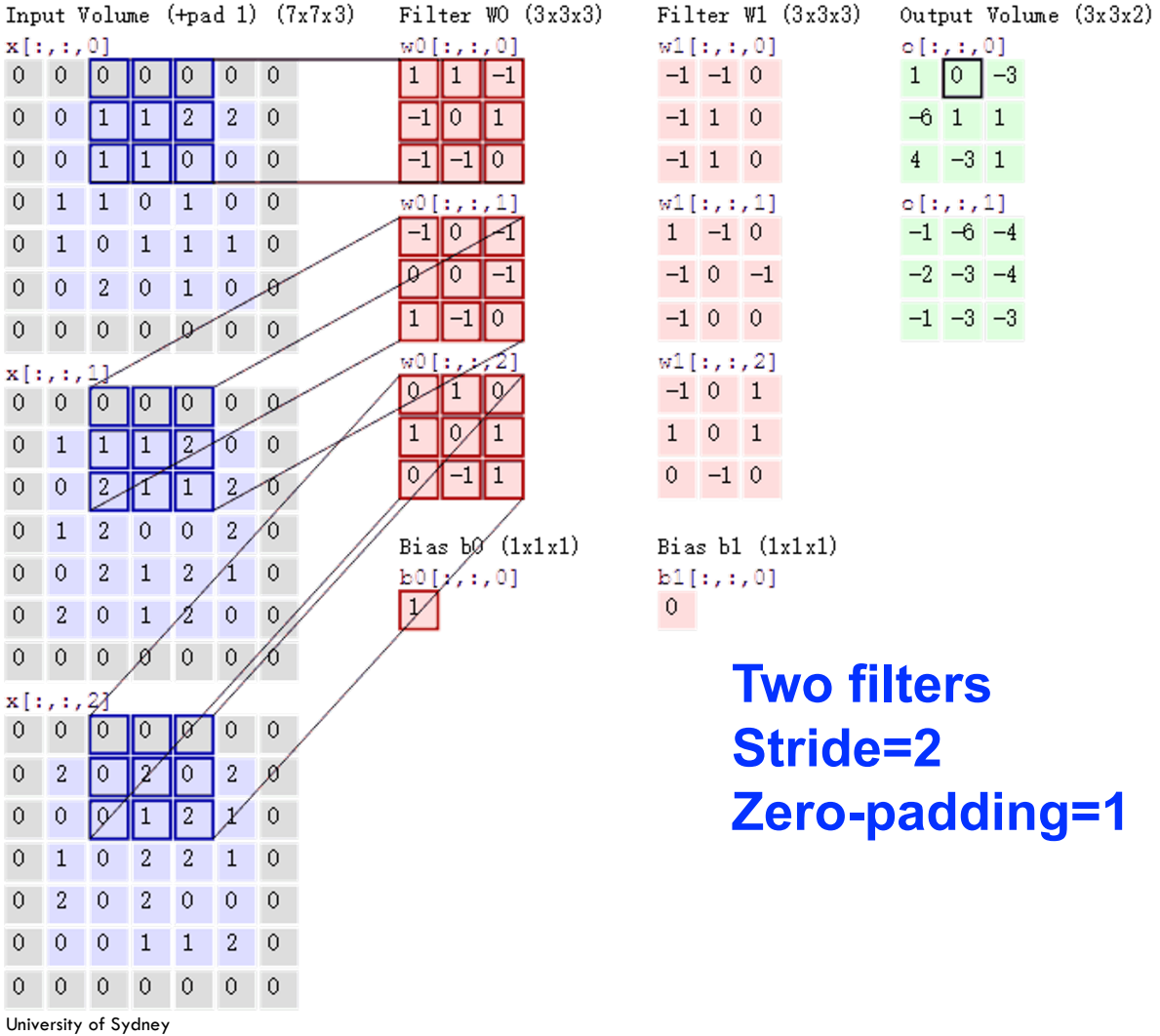
2 filters
2 matrix

$\frac{5+2-3}{2}+1$

$= 3.$

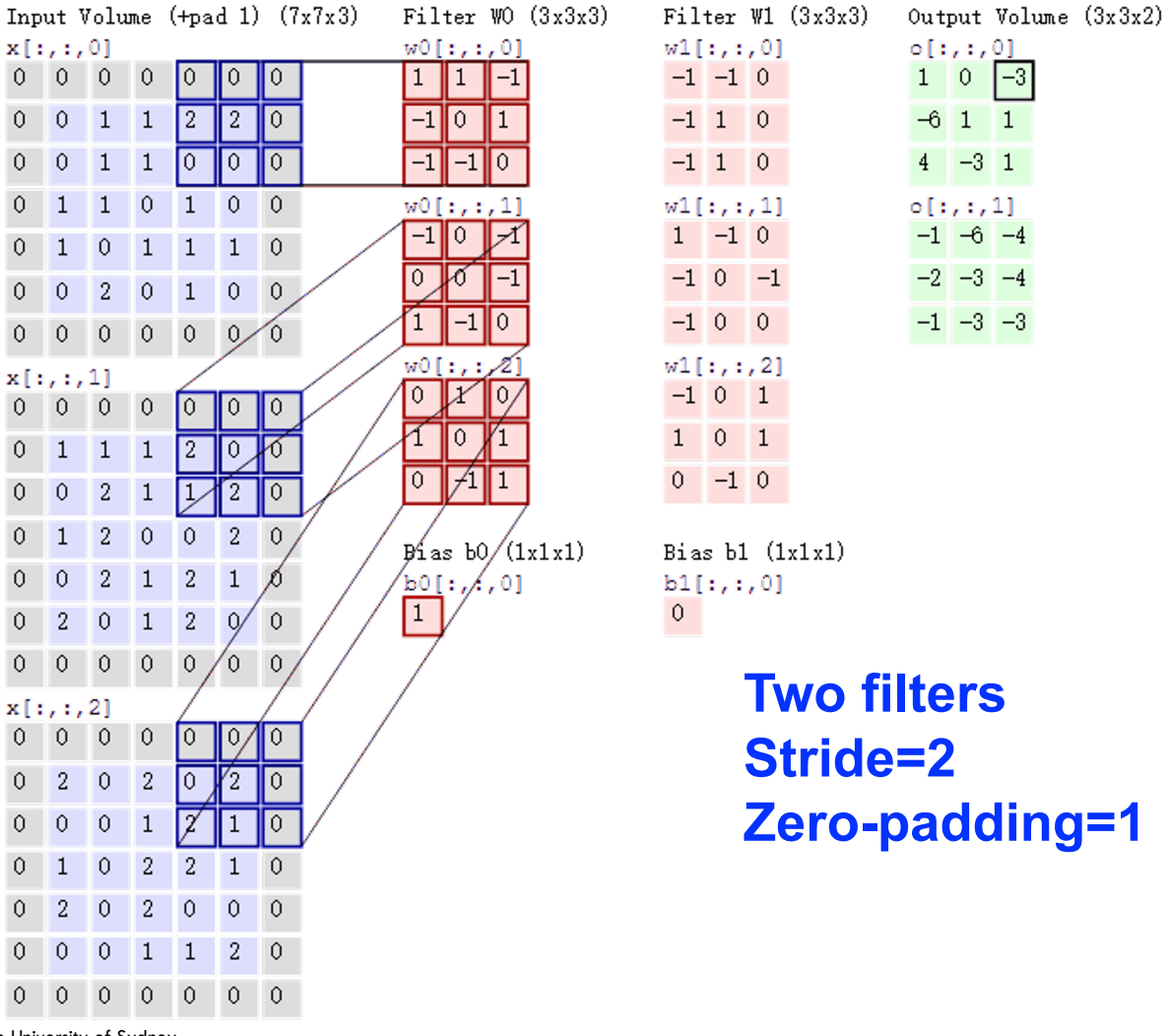
Two filters
Stride=2
Zero-padding=1

Convolutional Layer



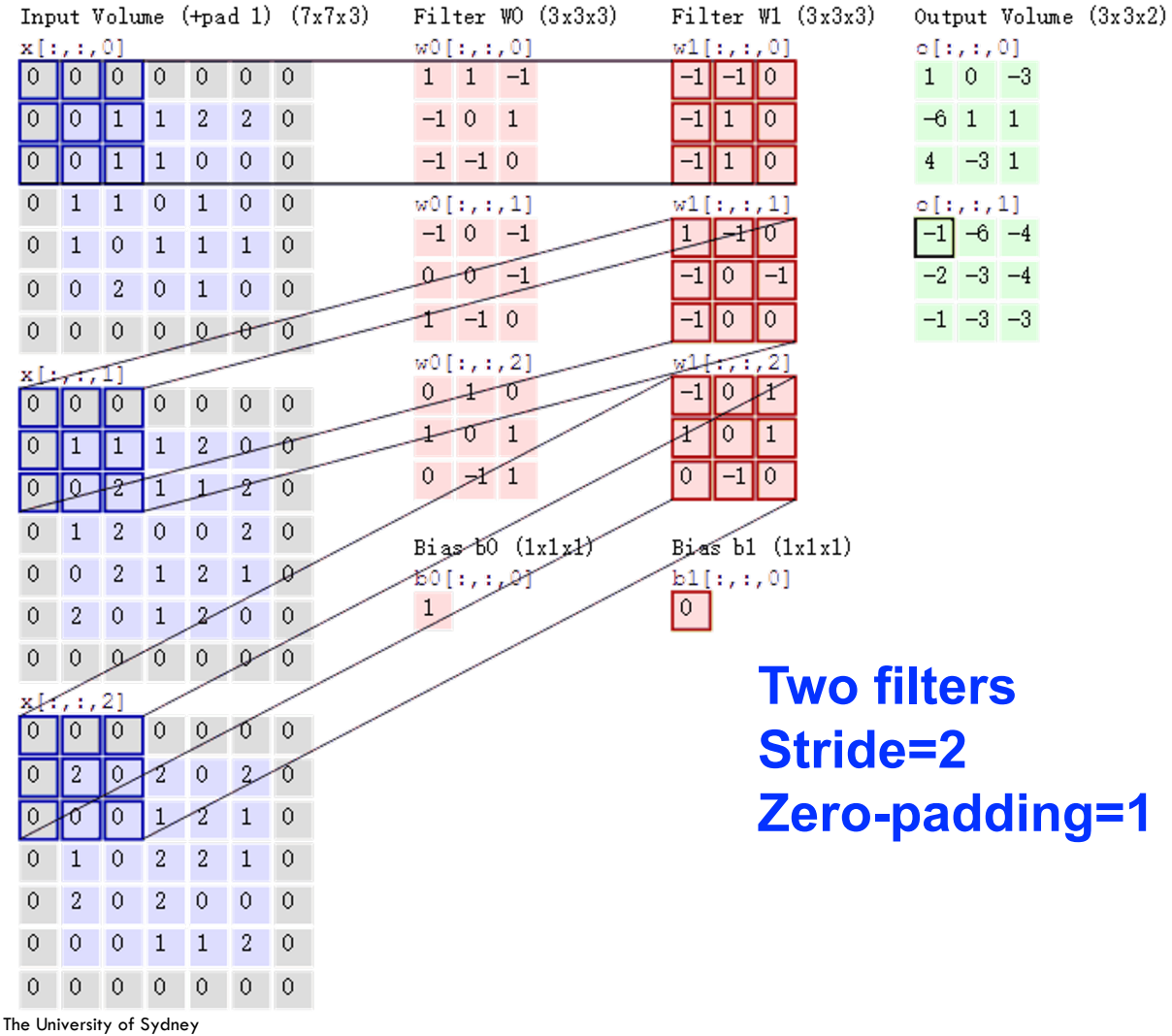
Two filters
Stride=2
Zero-padding=1

Convolutional Layer



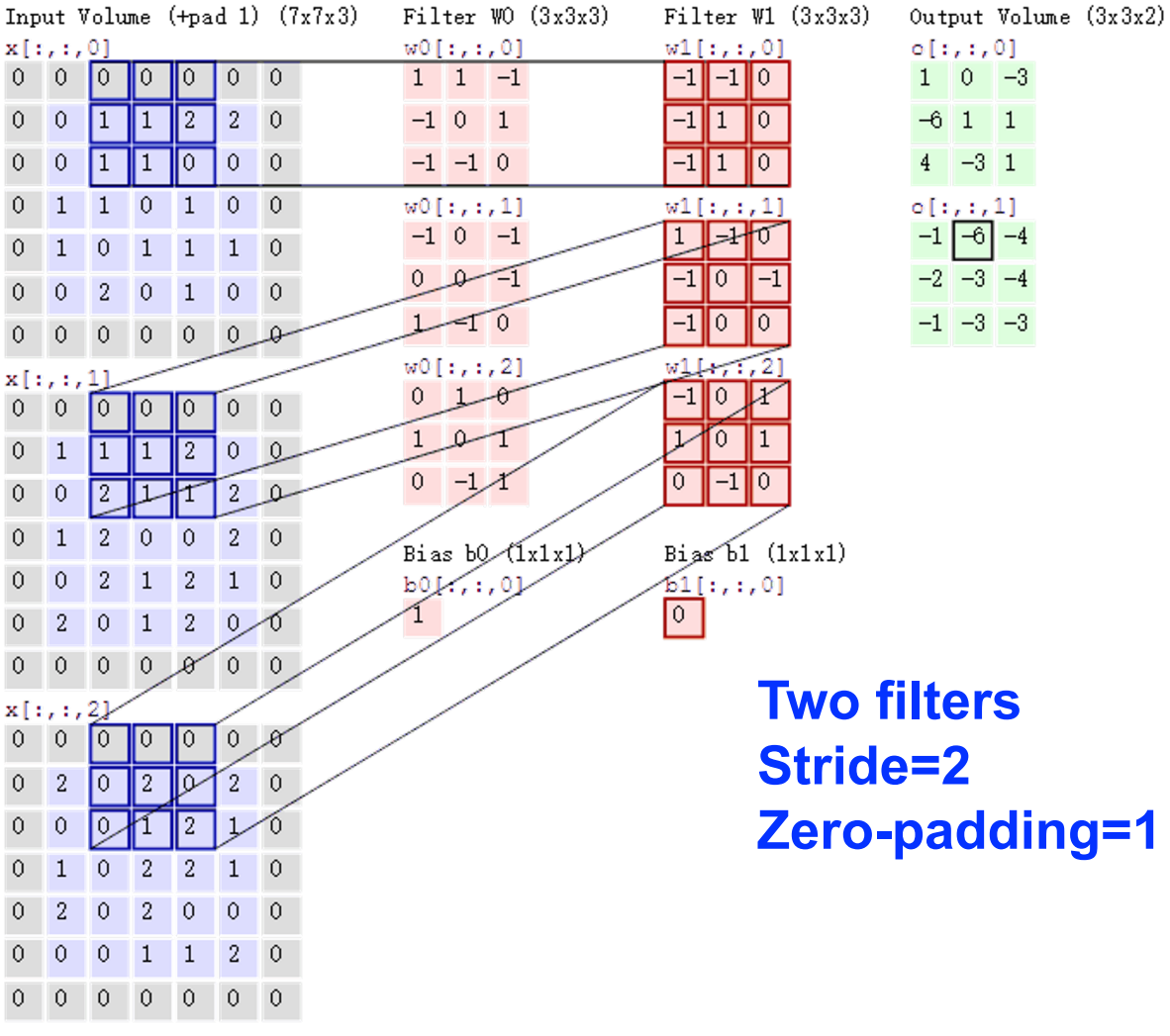
Two filters
Stride=2
Zero-padding=1

Convolutional Layer



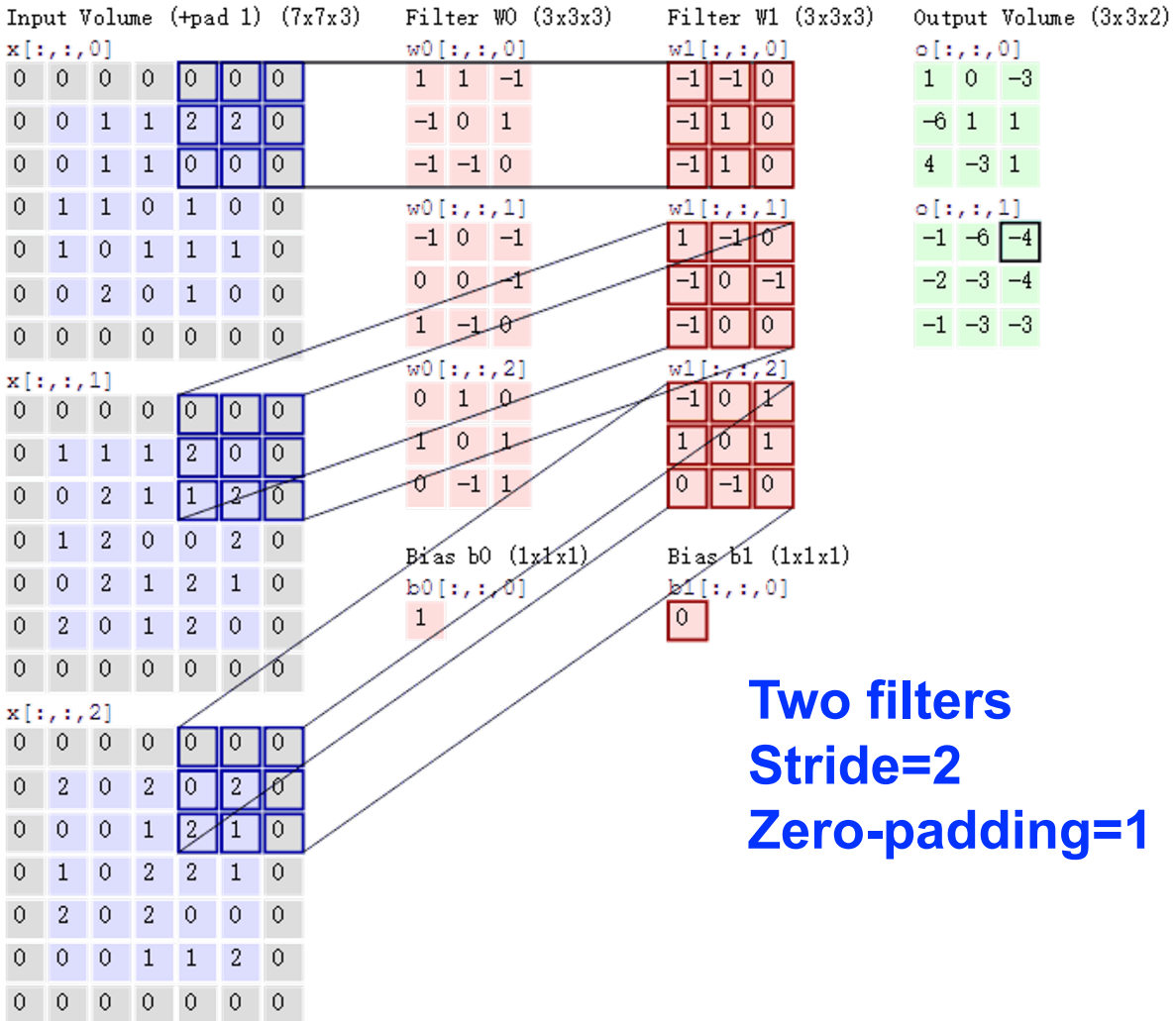
Two filters
Stride=2
Zero-padding=1

Convolutional Layer



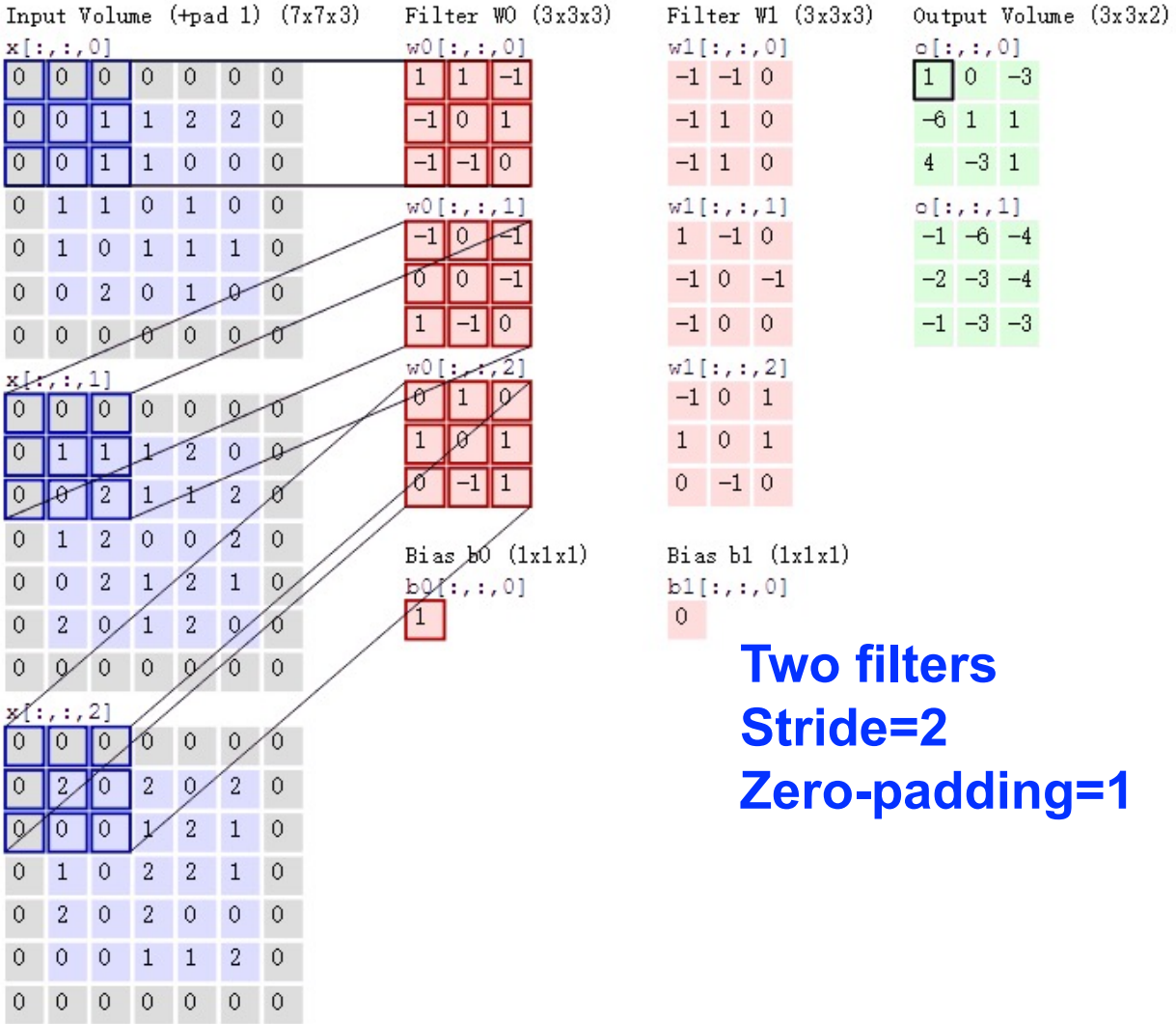
Two filters
Stride=2
Zero-padding=1

Convolutional Layer



Two filters
Stride=2
Zero-padding=1

Convolutional Layer

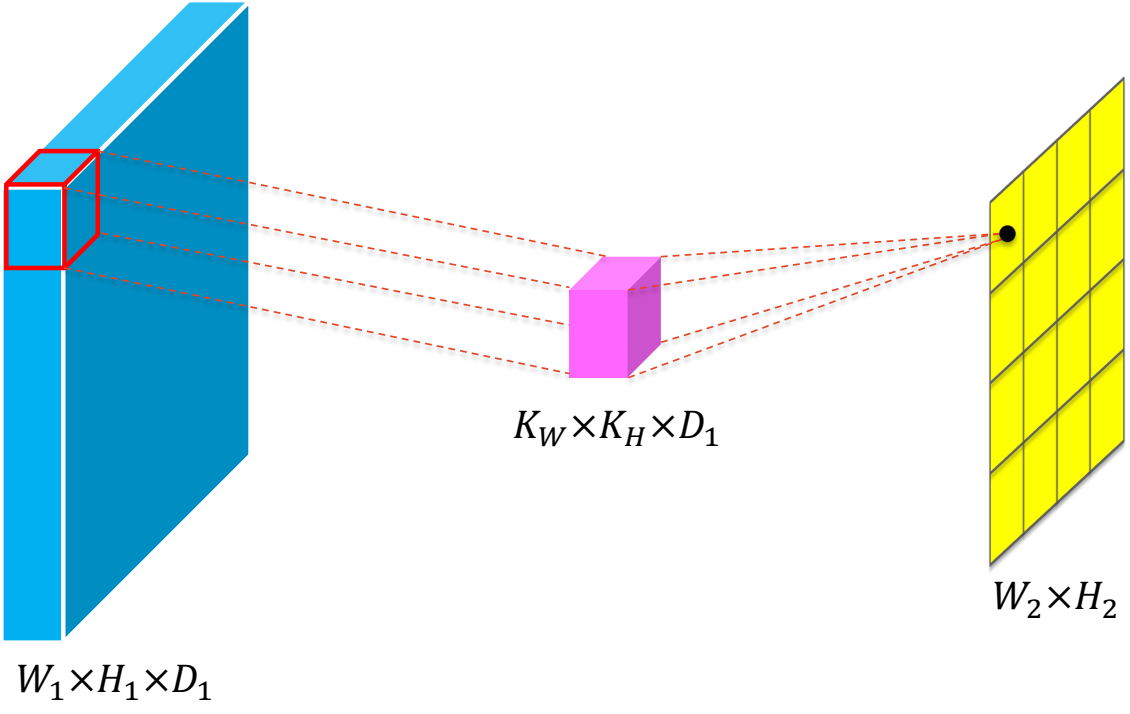


Two filters
Stride=2
Zero-padding=1

Convolutional Layer

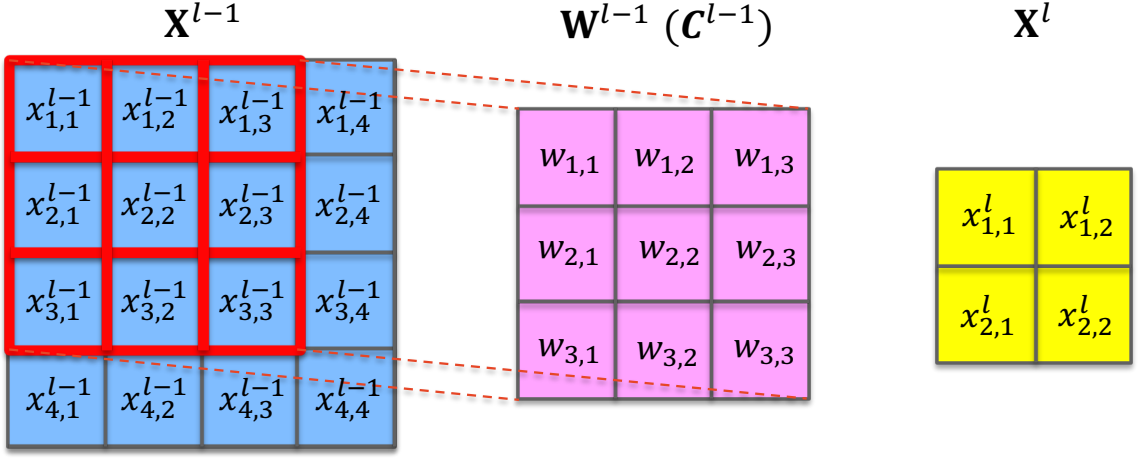
- Suppose stride is (S_W, S_H) and pad is (P_W, P_H)

$$W_2 = \frac{W_1 + 2P_W - K_W}{S_W} + 1 \text{ and } H_2 = \frac{H_1 + 2P_W - K_H}{S_H} + 1$$



Convolution as a matrix operation

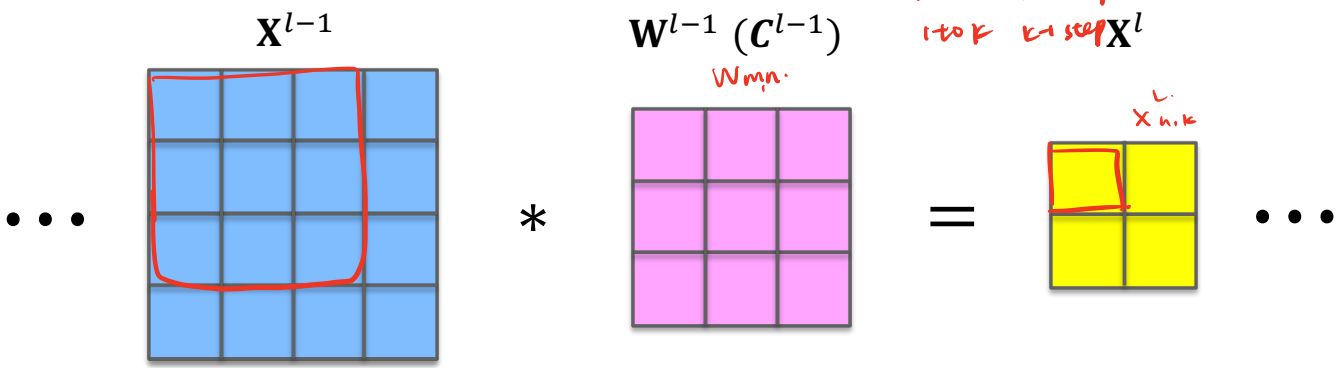
- If the input $\mathbf{X}^{l-1}$and output $\mathbf{X}^{l}$ were to be unrolled into vectors, the convolution could be represented as a sparse matrix$\boldsymbol{C}^{l-1}$where the non-zero elements are the elements $w_{m,n}$ of the kernel.



$$\boldsymbol{C}^{l-1}\times(x_{1,1}^{l-1},\cdots,x_{4,4}^{l-1})^{\mathsf{T}}=(x_{1,1}^{l},\cdots,x_{2,2}^{l})^{\mathsf{T}}$$

$$\boldsymbol{C}^{l-1}=\begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{1,1} & w_{1,2} & w_{1,3} & 0 & w_{2,1} & w_{2,2} & w_{2,3} & 0 & w_{3,1} & w_{3,2} & w_{3,3} \end{pmatrix}$$

Back-propagation in convolutional layer



$$\boldsymbol{C}^{l-1}\times(x_{1,1}^{l-1},\cdots,x_{4,4}^{l-1})^{\intercal}=(x_{1,1}^{l-1},\cdots,x_{2,2}^{l-1})^{\intercal}$$

- Backward pass

$$\frac{\partial Loss}{\partial w_{m,n}}=\sum_{h,k}\frac{\partial Loss}{\partial x_{h,k}^{l}}\frac{\partial x_{h,k}^{l}}{\partial w_{m,n}},$$

,where $$\frac{\partial x_{h,k}^{l}}{\partial w_{m,n}}=x_{h+m-1,k+n-1}^{l-1},$$

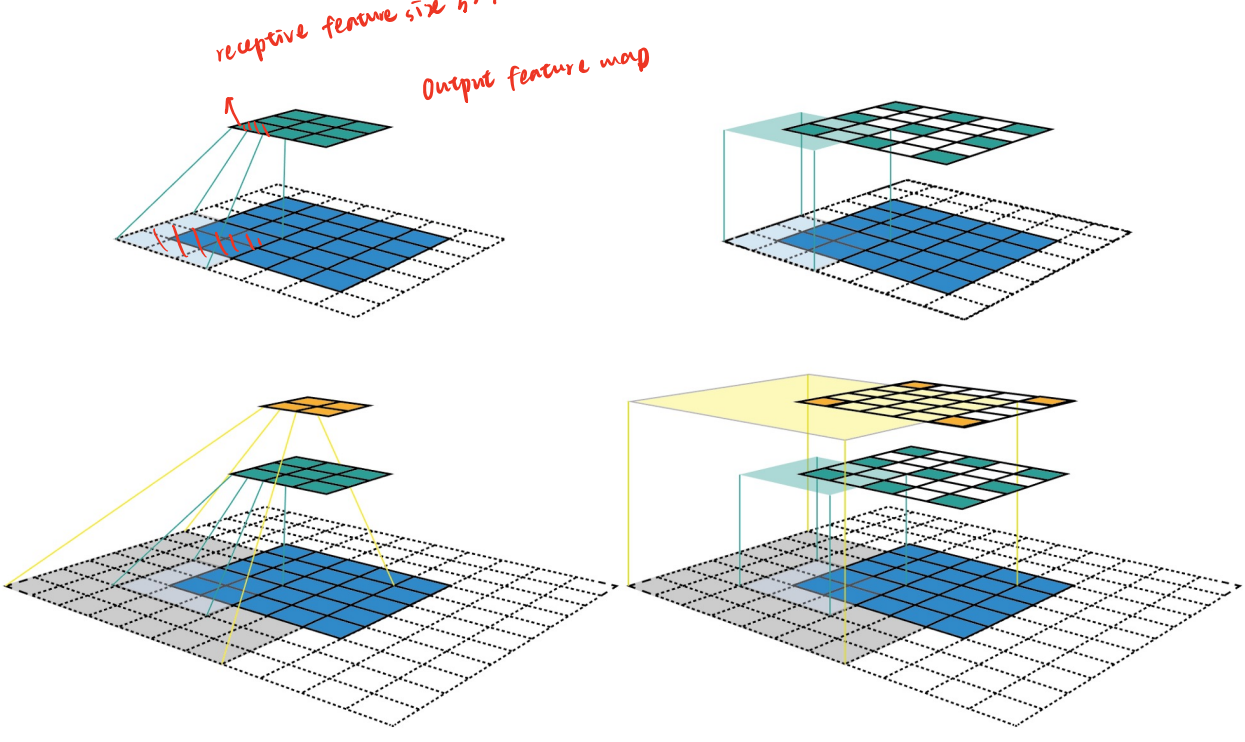
$$\frac{\partial Loss}{\partial x_{h,k}^{l}}=\frac{\partial Loss}{\partial x_{i}^{l}}=\sum_{j}\frac{\partial Loss}{\partial x_{j}^{l+1}}\frac{\partial x_{j}^{l+1}}{\partial x_{i}^{l}}=\sum_{j}\frac{\partial Loss}{\partial x_{j}^{l+1}}\boldsymbol{C}_{j,i}^{l}=\frac{\partial Loss}{\partial \mathbf{x}^{l+1}}\boldsymbol{C}_{*,i}^{l}=\boldsymbol{C}_{*,i}^{l\ \intercal}\frac{\partial Loss}{\partial \mathbf{x}^{l+1}}.$$

(Note that x_i^l represent i-th element in the $\mathbf{X}^l$. Here, $i=(h-1)\times H+k$.

Receptive Field

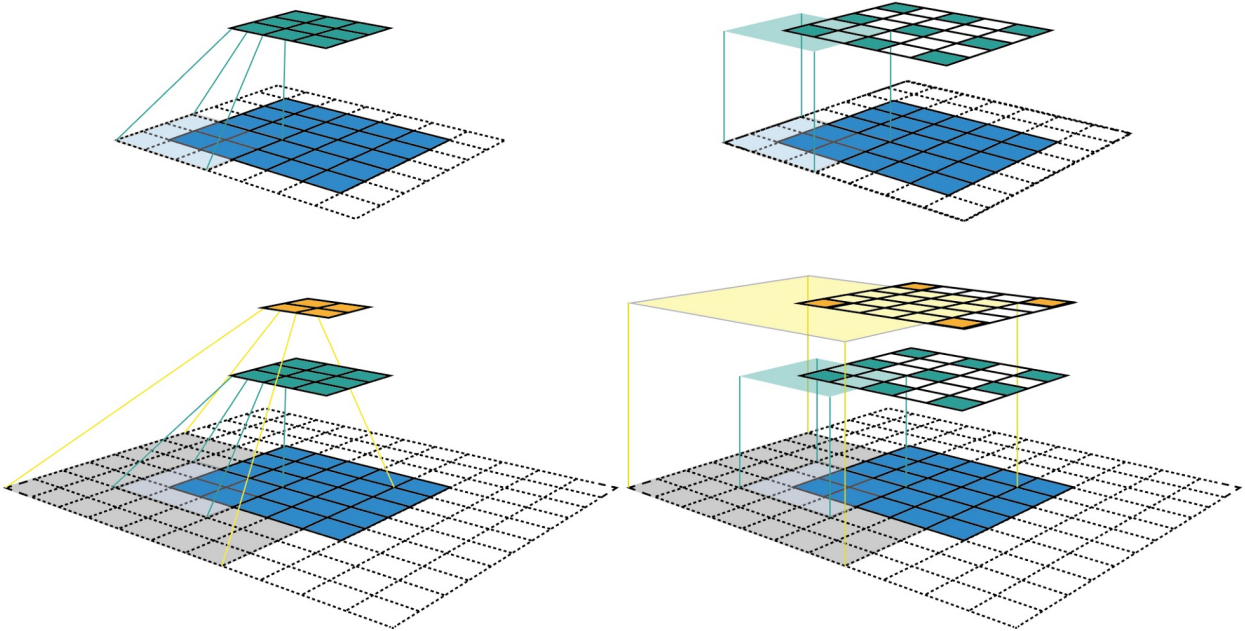
Receptive Field

- The receptive field in Convolutional Neural Networks (CNN) is the region of the input space that affects a particular unit of the network.
- In this example, we use the convolution filter $\mathbf{W}$ with size $K = 3\times3$, padding $P = 1$, stride $S = 2\times2$.



Receptive Field

- From the left column, we are hard to tell the receptive filed size, especially for deep CNNs.
- The right column shows the fixed-sized CNN visualization, which solves the problem by keeping the size of all feature maps constant and equal to the input map. Each feature is then marked at the center of its receptive field location.



Convolution layers in PyTorch

https://pytorch.org/docs/stable/nn.html#convolution-layers

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

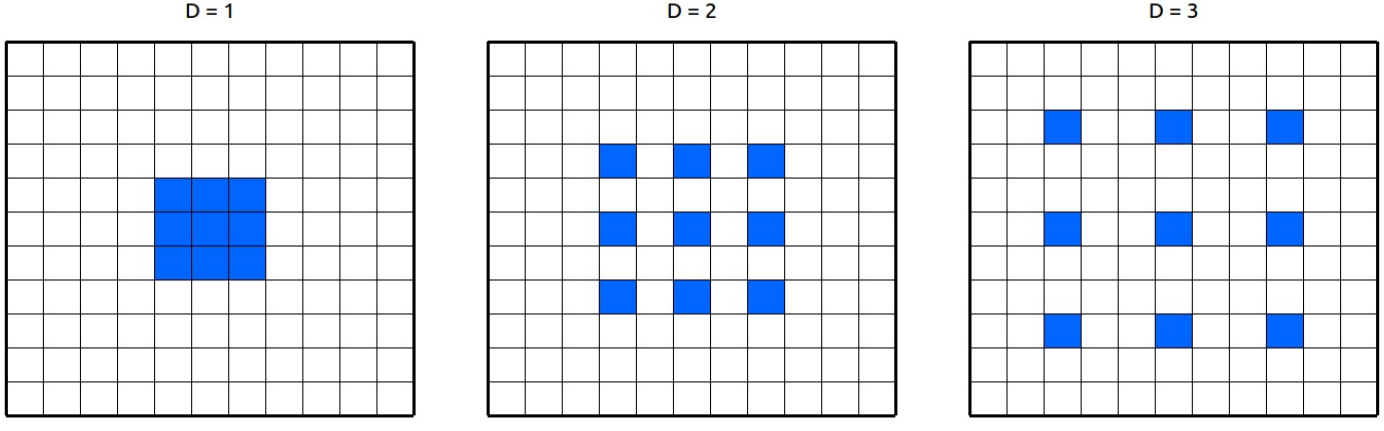
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Dilated Convolution

- In simple terms, dilated convolution is just a convolution applied to input with defined gaps.
- Dilation: Spacing between kernel elements. Default: 1.
- D=2 means skipping one pixel per input
- The receptive filed grows exponentially while the number of parameters grows linearly.

global understanding
extracting information



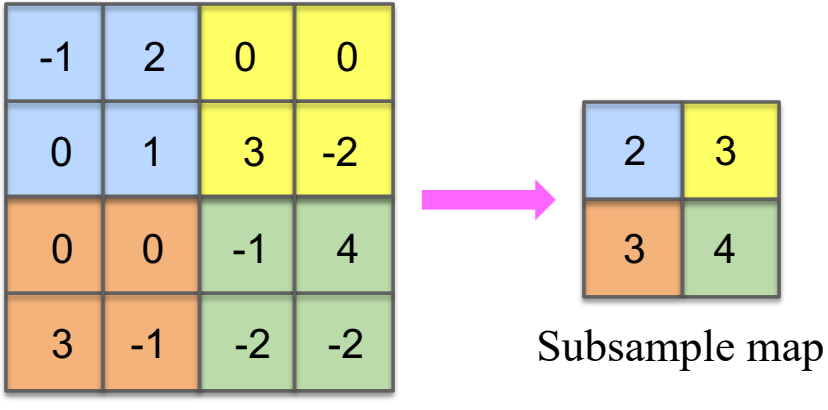
(Yu et al, 2015)

Pooling

Pooling

Max pooling

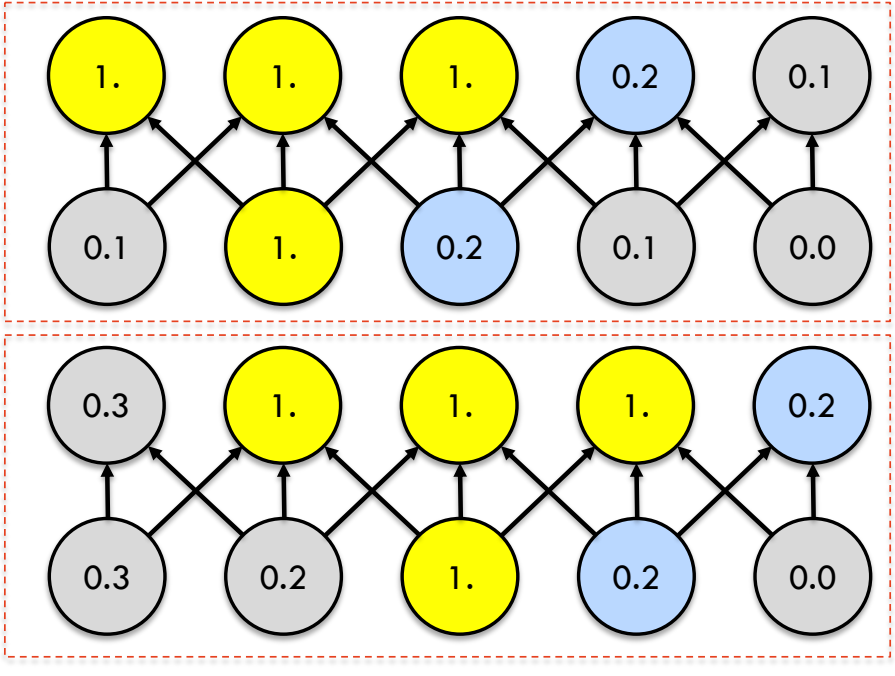
- Filter size: (2,2)
- Stride: (2,2)
- Pooling ops: max(·)



Motivation: Pooling

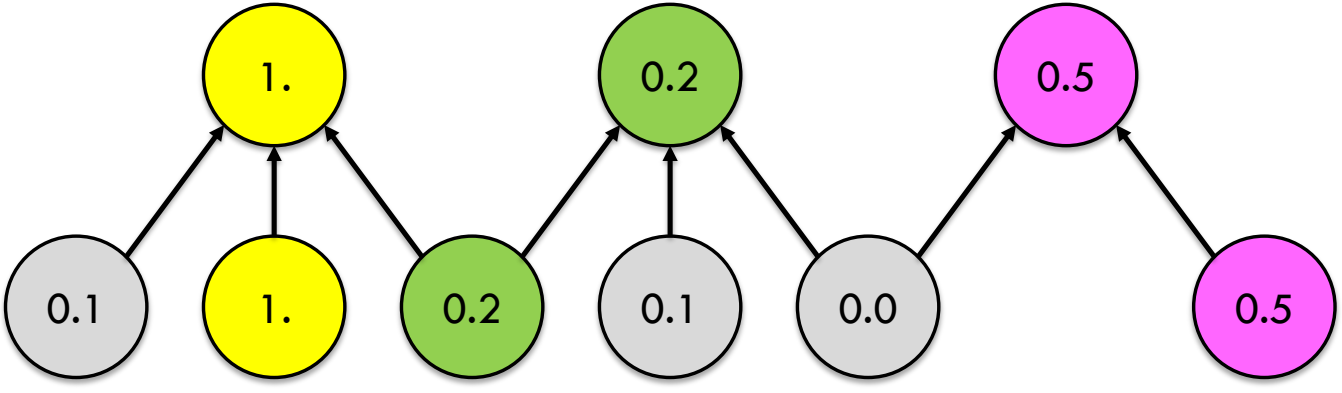
- Pooling helps the representation become slightly invariant to small translations of the input
  - Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is
  - Taking max pooling as an example:

not care about input size



Motivation: Pooling

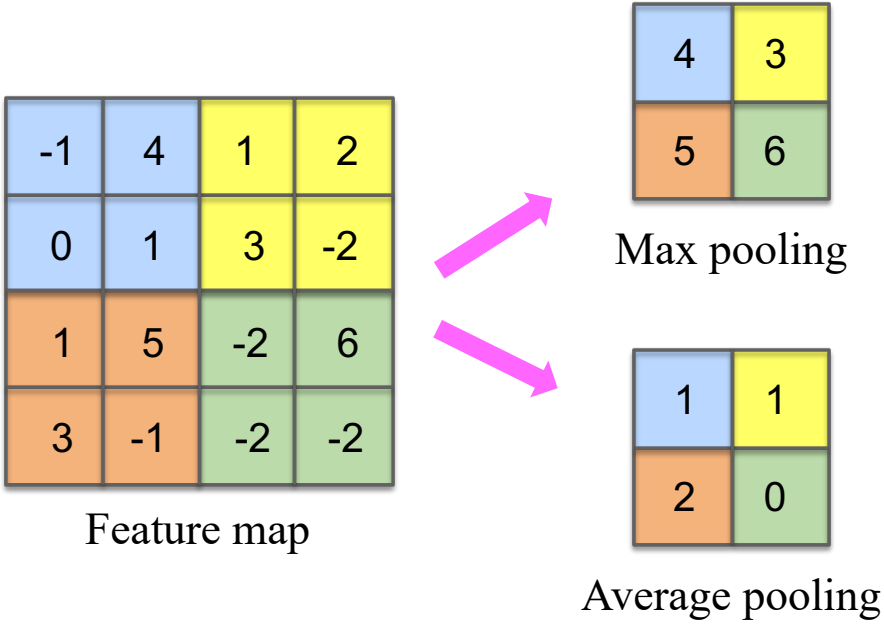
- Because pooling summarizes the responses over a whole neighbourhood, it is possible to use fewer pooling units than detector units
- Since pooling is used for down sampling, it can be used to handle inputs of varying sizes



Pooling

Average pooling

- Filter size: (2,2)
- Stride: (2,2)
- Pooling ops: mean(·)

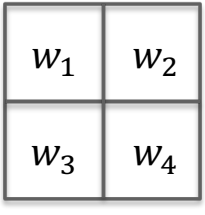


Pooling

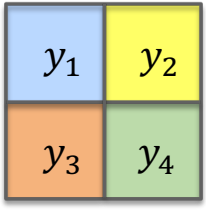
L_2 norm pooling

- Filter size (Gaussian kernel size): (2,2)
- Stride: (2,2)
- Pooling ops: $y_i = \sqrt{\sum_j w_j x_{i,j}{}^2}$

Feature map



Gaussian window



Output

Pooling

- Other pooling
  - **L_p pooling** (preserves the class-specific spatial/geometric information in the pooled features)

$$y_i = \left(\sum_j w_j {x_{i,j}}^p \right)^{1/p}$$

  - **Mixed pooling** (addresses the over-fitting problem)

$$y_i = a \max(x_{i,1}, \dots, x_{i,n}) + (1 - a)\text{mean}(x_{i,1}, \dots, x_{i,n})$$

  - **Stochastic pooling** (hyper-parameter free, regularizes large CNNs)

$$y_i = x_l, \text{ where } l \sim P(p_1, \cdots, p_n) \text{ and } p_j = \frac{x_{i,j}}{\sum_j x_{i,j}}$$

  - **Spectral pooling** (preserves considerably more information per parameter than other pooling strategies)

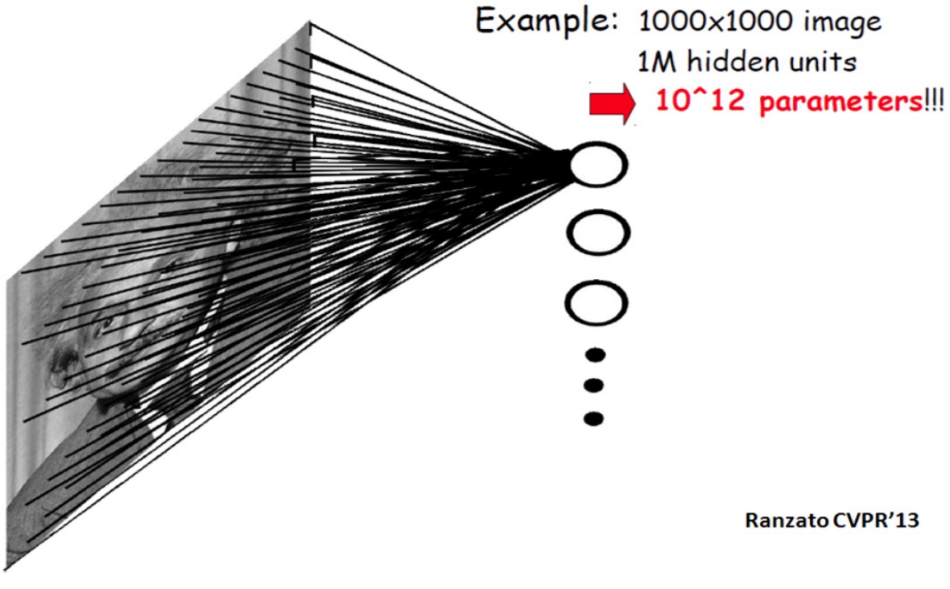
$$\mathbf{y} = \mathcal{F}(\mathbf{x}) \in \mathbb{C}^{M \times N}, \hat{\mathbf{y}} = \mathcal{F}^{-1} (\mathbf{y} \in \mathbb{C}^{H \times W})$$

  - …

Why CNNs ?

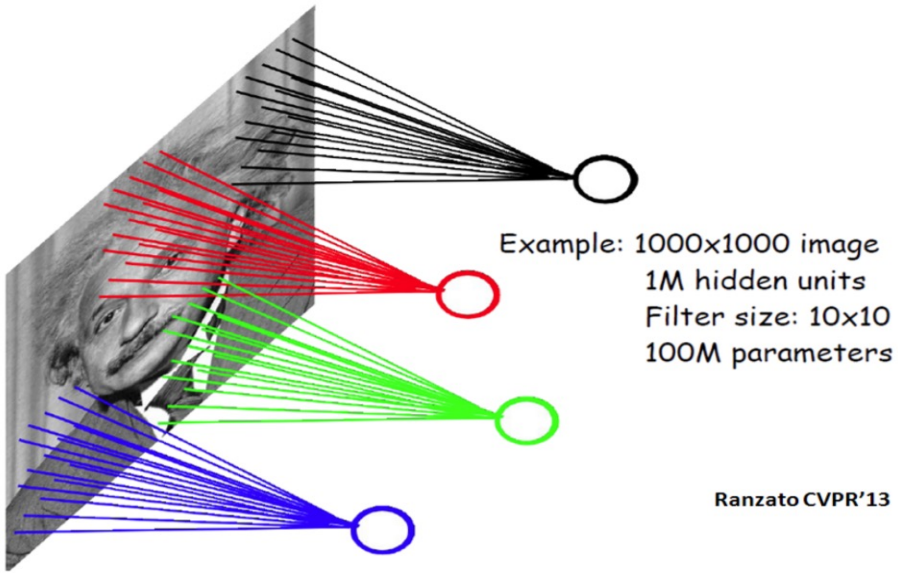
Motivation: convolution

- Problems of fully connected neural networks
  - Every output unit interacts with every input unit
  - The number of weights grows largely with the size of the input image
  - Pixels in distance are less correlated



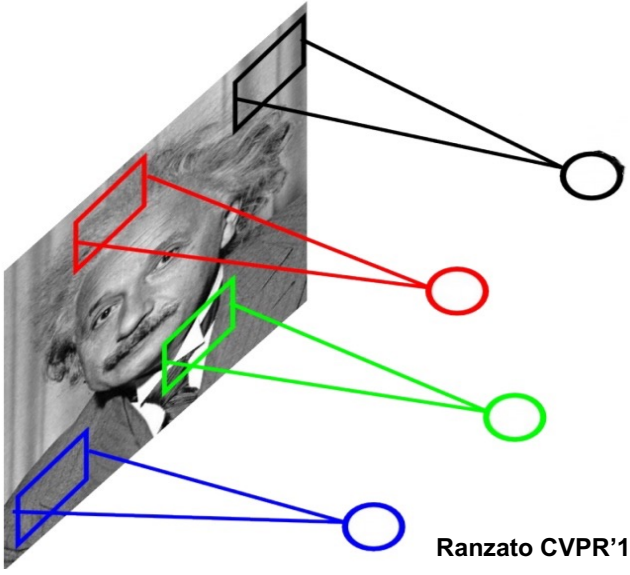
Motivation: convolution

- Locally connected neural net
  - Sparse connectivity: a hidden unit is only connected to a local patch
  - It is inspired by biological systems, where a cell is sensitive to a small sub-region, called a receptive field.
  - Here, the receptive field can be called as filter or kernel

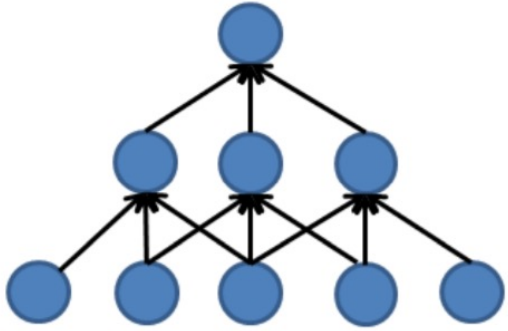


Motivation: convolution

- Problems of Locally connected neural net
  - The learned filter is a spatially local pattern
  - A hidden node at a higher layer has a larger receptive field in the input
  - Stacking many such layers leads to “filters”(not anymore linear) which become increasingly “global”

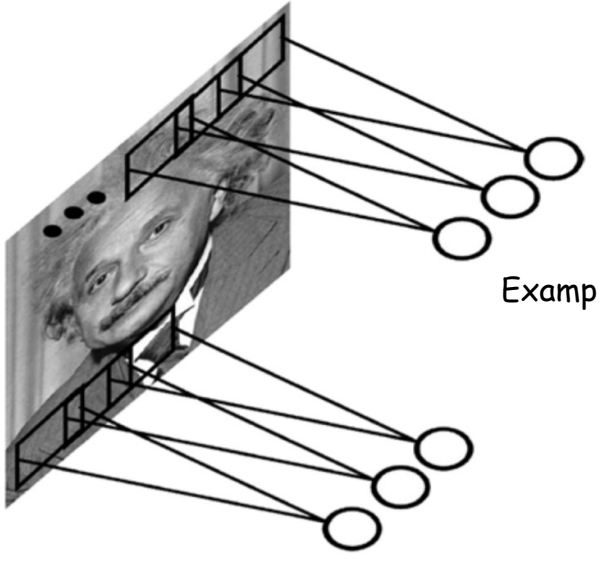


Ranzato CVPR’13



Motivation: convolution

- Shared weights
  - Translation invariance: capture statistics in local patches and they are independent of locations
  - Hidden nodes at different locations share the same weights. It greatly reduces the number of parameters to learn

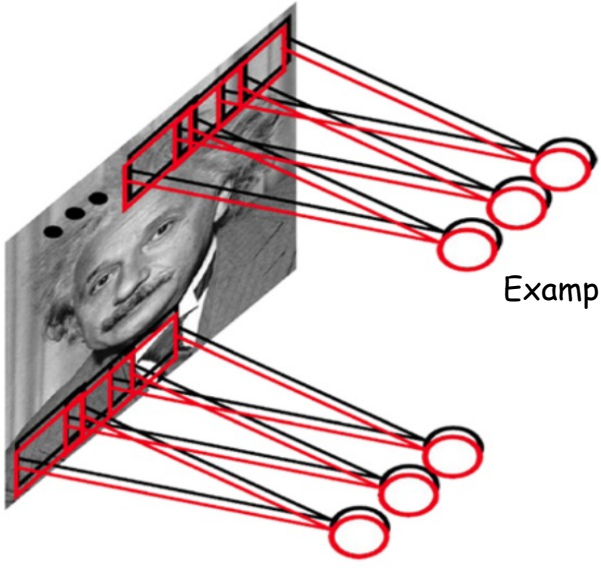


Example: 1000x1000 image
1 Filters
Filter size: 10x10
100 parameters

Ranzato CVPR'13

Motivation: convolution

- Multiple filters
  - Multiple filters provide the probability of detecting the spatial distributions of multiple visual patterns
  - One filter can build a feature map, multiple filters will build a stack of feature maps



Example: 1000x1000 image
100 Filters
Filter size: 10x10
10k parameters

Ranzato CVPR'13

Motivation: convolution

- Multiple filters: intuitive examples

Input

Image blurring

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Edge detection

0	0	0	0	0
0	0	1	0	0
0	1	-4	1	0
0	0	1	0	0
0	0	0	0	0

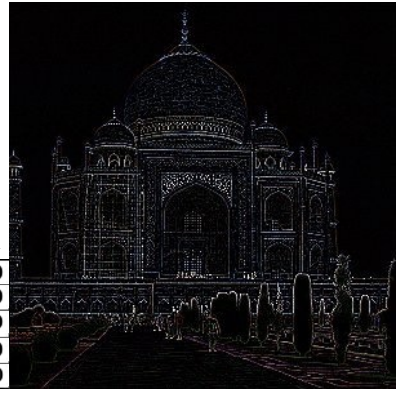
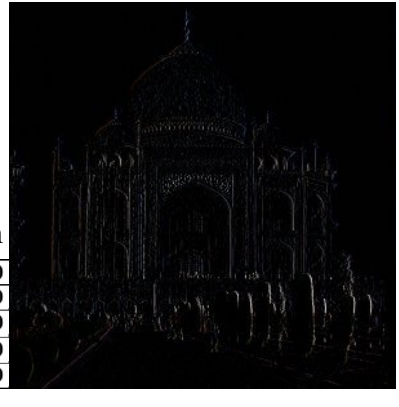


Image enhancement

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

Vertical detection

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



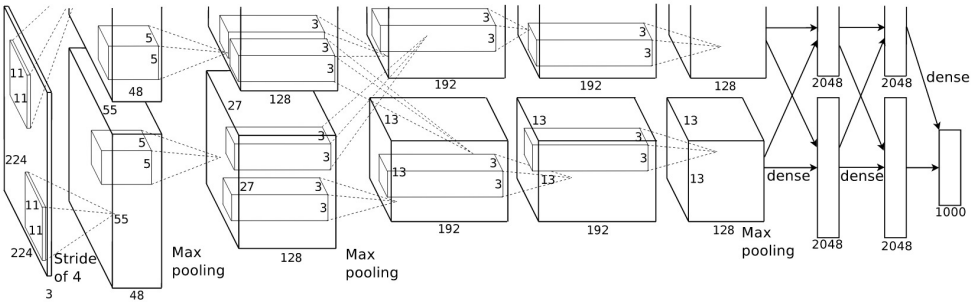
Visualize features

Visualize features

- Why CNNs work so well?

Hierarchical Convolution,
Nonlinear operations (ReLU, max pooling…)

Input image

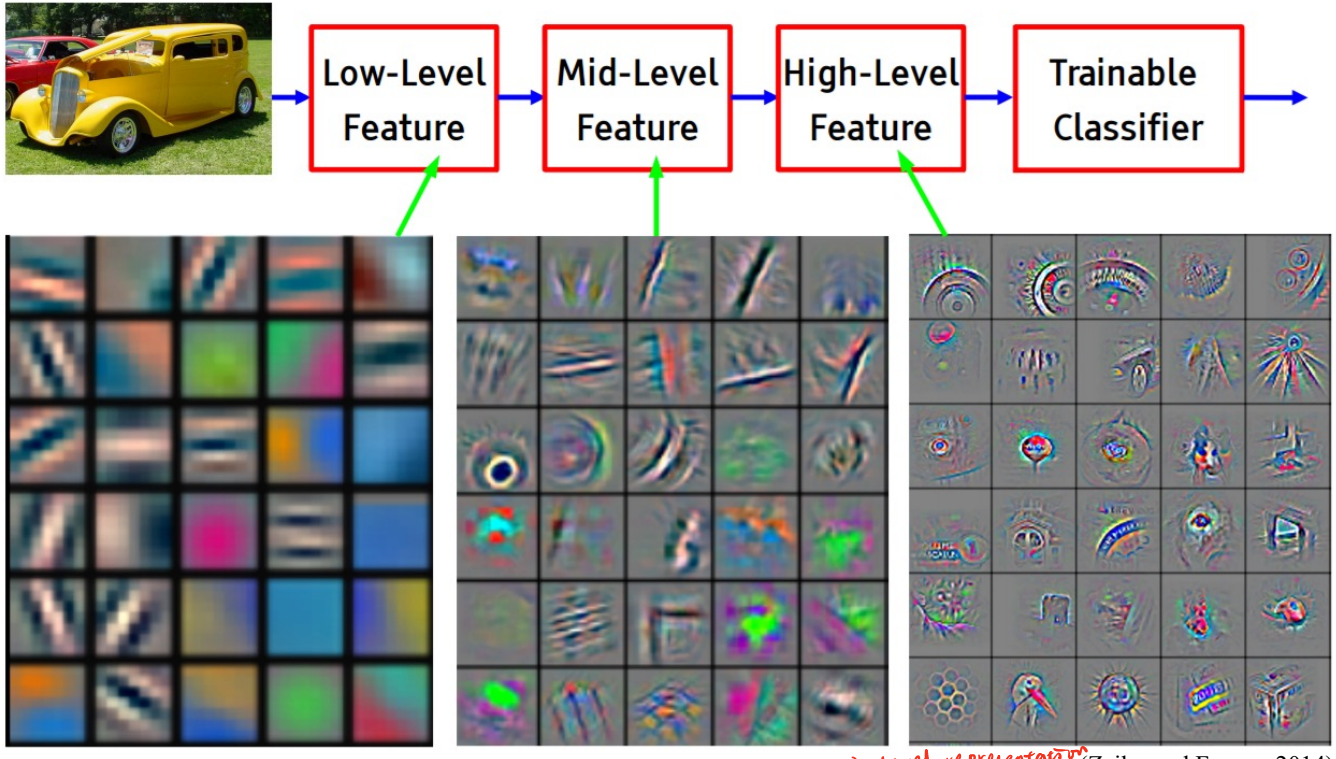


Class Scores:
1000 numbers

What happens inside hidden layers?

Visualize features

- Give insight into the function of intermediate feature layers and the operation of the classifier



(Zeiler and Fergus, 2014)

high-level representation
pre-train
↓
for other downstream

Thank you!