Special topics: Convolutional Neural Networks

Giảng viên: TS. Nguyễn Thị Ngọc Diệp

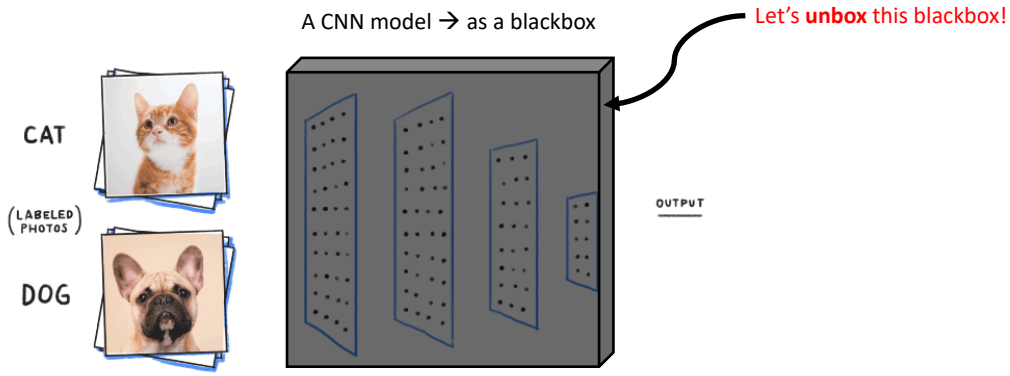
Email: ngocdiep@vnu.edu.vn

Slide & Code: https://github.com/chupibk/INT3414_22

Schedule

Week	Content	Class hour	Self-study hour
1	Introduction CNNs in Computer Vision	2	1
2	~~Foundations of CNNs~~ ~~Case study: Image classification problem~~ Basics of Neural networks Training with backpropagation Implementation with PyTorch	2	2-6
3	~~Training and tuning parameters~~ ~~Data augmentation - Data generator~~ Foundations of CNNs Transfer learning	2	2-6
4	Object detection	2	2-6
5	Segmentation	2	2-6
6, 7	Mid-term presentations	2	2-6
8, 9	Advanced topics using CNNs	2	2-6
10, 11, 12	Final project presentations	1	2-6
13	Class summarization	1-3	open

Recall week 1

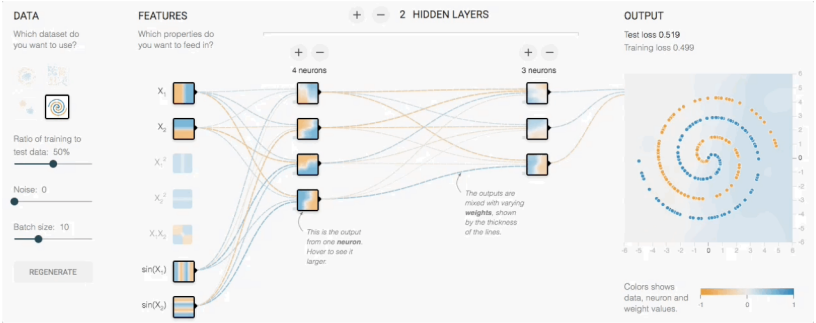


The most important: design the problem to a solvable task using the blackbox

Image credit: https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Recall week 2: Build a neural network

https://playground.tensorflow.org/



m observed data, 4 features

Input $X \in \mathbb{R}^{4\times m}$ → Hidden 1 $H^{(1)} \in \mathbb{R}^{4\times m}$ → Hidden 2 $H^{(2)} \in \mathbb{R}^{3\times m}$ → Output $y \in \mathbb{R}^{1\times m}$

Learning weights and biases

$W^{(1)}X + B^{(1)} = H^{(1)}$, $W^{(1)} \in \mathbb{R}^{4\times 4}$

$W^{(2)}X + B^{(2)} = H^{(2)}$, $W^{(2)} \in \mathbb{R}^{3\times 4}$

$W^{(3)}X + B^{(3)} = y$, $W^{(3)} \in \mathbb{R}^{1\times 3}$

Recall week 2: Optimizers

Updating weights iteratively

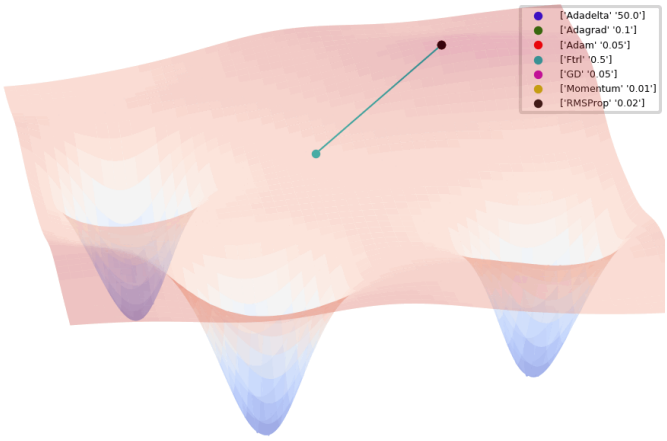
```
for t in range(num_epochs):
    # Forward pass
    y_pred = model(x_train_t)

    # Compute loss
    loss = loss_fn(y_pred, y_train_t)

    # Clear the buffers
    optimizer.zero_grad()

    # Backward pass: compute gradients
    loss.backward()

    # update the parameters
    optimizer.step()
```



Visualization credit: https://github.com/Jaewan-Yun/optimizer-visualization

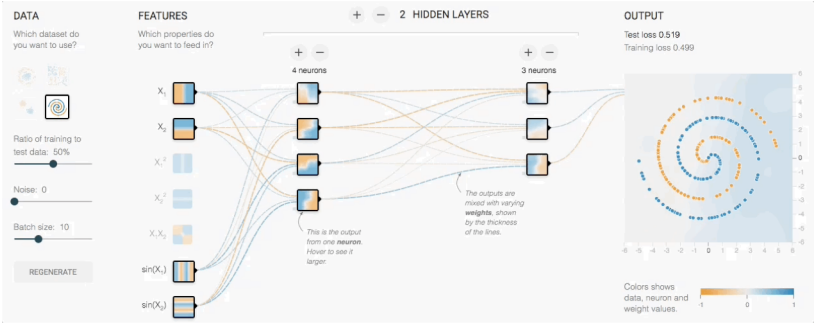
Foundations of CNNs

Week 2

Code @Google Colab

- Link: https://colab.research.google.com/drive/1hLRa43CbzXeE0-DY7UlTrbDUeITsS7Pe

From Neural Networks to Convolutional NNs



m observed data, 4 features

Input $X \in \mathbb{R}^{4\times m}$ → Hidden 1 $H^{(1)} \in \mathbb{R}^{4\times m}$ → Hidden 2 $H^{(2)} \in \mathbb{R}^{3\times m}$ → Output $y \in \mathbb{R}^{1\times m}$

$W^{(1)}X + B^{(1)} = H^{(1)}$, $W^{(1)} \in \mathbb{R}^{4\times 4}$

$W^{(2)}X + B^{(2)} = H^{(2)}$, $W^{(2)} \in \mathbb{R}^{3\times 4}$

$W^{(3)}X + B^{(3)} = y$, $W^{(3)} \in \mathbb{R}^{1\times 3}$

Learning weights and biases

What if input is a 32x32 image?

From NNs to CNNs

Input: 32x32

$X \in \mathbb{R}^{1024 \times m}$

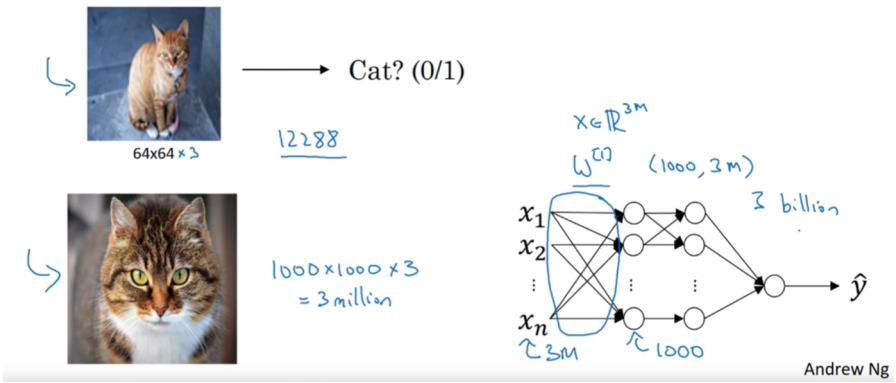
Hidden layer 1: $W^{(1)} \in \mathbb{R}^{1024 \times 4}$

Input: 1024x1024

$X \in \mathbb{R}^{1048576 \times m}$

$W^{(1)} \in \mathbb{R}^{1048576 \times 4}$

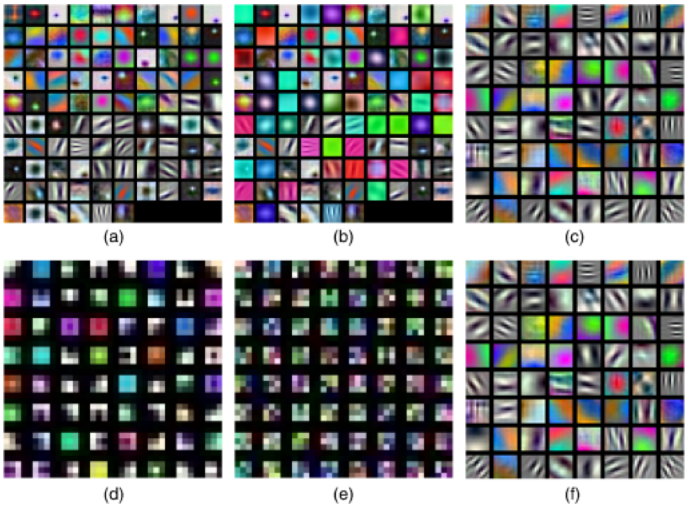
From NNs to CNNs: Color images



Convolutional operations as the fundamental building block of computational neural networks

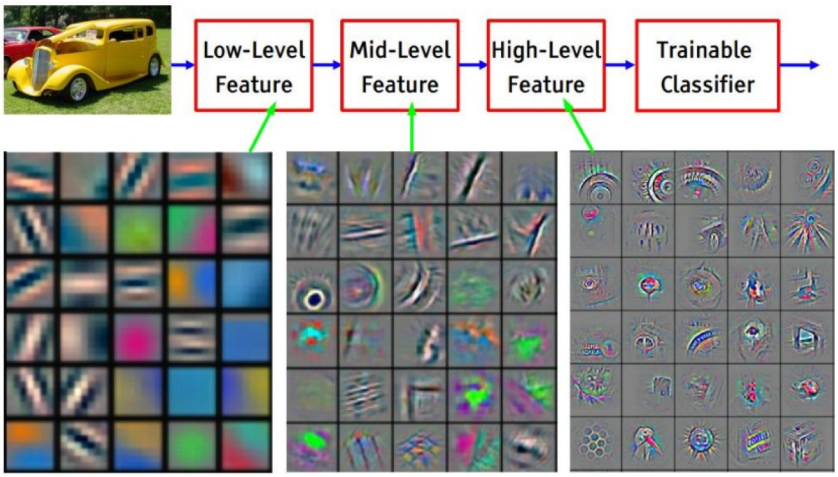
CNNs for image representations

Learned convolutional kernels (first layer)
a) AlexNet (ImageNet)
b) AlexNet (CDB)
c) VGG-f (ImageNet)
d) VGG-16 (ImageNet)
e) VGG-16 (CDB)
f) VGG-f (CDB)



Ref: Georg Wimmer, Andreas Vécsei, Michael Häfner, Andreas Uhl, "Fisher encoding of convolutional neural network features for endoscopic image classification," J. Med. Imag. 5(3) 034504 (24 September 2018)

CNNs for image representations



Source: Internet

Question:

What ”convolution” operation does to make that representation and learning possible?

Convolution example

This kernel is known to detect vertical edge ☺

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

convolve

*

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

=

6x6
Input image

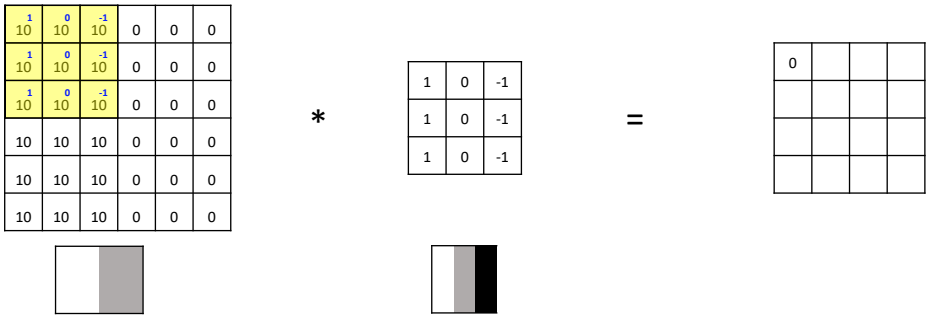
3x3
filter

4x4
output

Why 4x4?

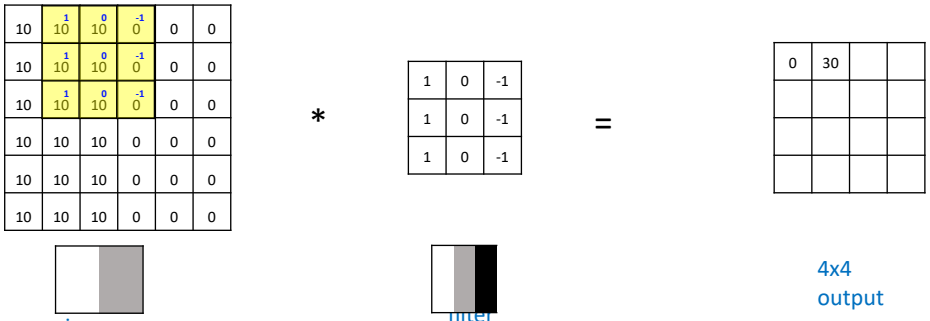
Convolution example

Start from top left...



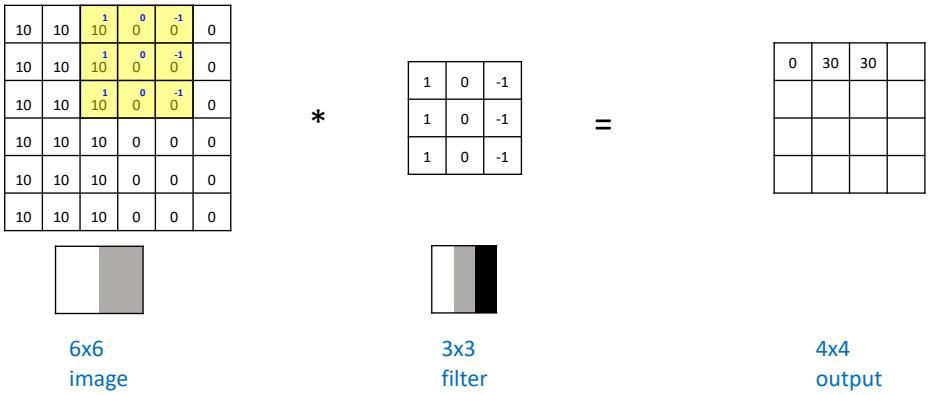
Convolution example

Move one pixel to the next...



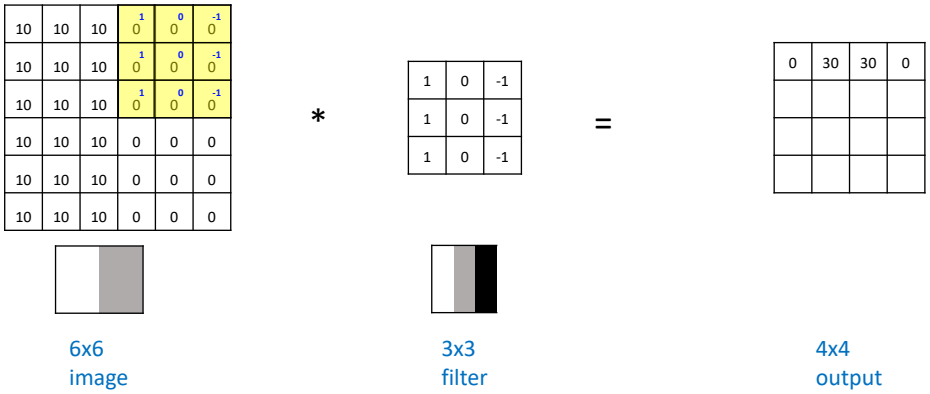
Convolution example

Next in the row...



Convolution example

Next to the right...



Convolution example

Move down one pixel
and begin from the leftmost...

10	10	10	0	0	0
1 10	0 10	-1 10	0	0	0
1 10	0 10	-1 10	0	0	0
1 10	0 10	-1 10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6x6
image

*

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

3x3
filter

=

0	30	30	0
0			

4x4
output

Convolution example

Patiently
to the bottom end...

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	1 0	0 0	-1 0
10	10	10	1 0	0 0	-1 0
10	10	10	1 0	0 0	-1 0

6x6
image

*

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

3x3
filter

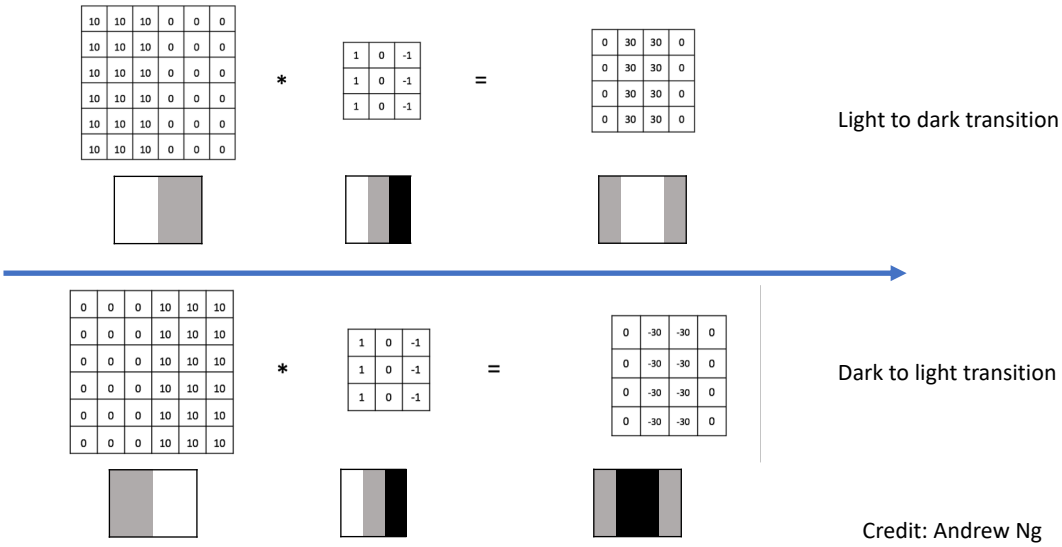
=

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

4x4
output

The thick line found is just the artifact because
the input is only 6x6 pixels.
If the image is larger, the found edge will be very small.

Vertical edge detection examples



Vertical and horizontal edge detection

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

Vertical

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

Horizontal

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

6x6

*

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

3x3

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

4x4

Learning kernels

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

Vertical

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

Horizontal

-1	0	1
-2	0	2
-1	0	-1

Sobel

3	0	3
10	0	-10
3	0	-3

Scharr

Any many more

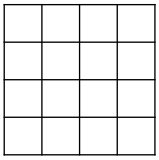
Learning values of kernels automatically

3	0	10	5	24	2
8	4	5	6	10	7
23	3	1	0	0	0
21	4	0	0	0	23
23	0	0	0	45	23
0	3	0	12	34	32

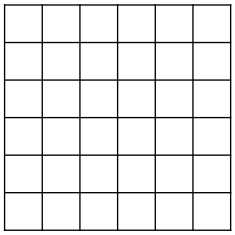
*

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

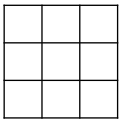
=



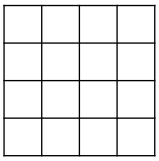
Dealing with boundaries



*



=



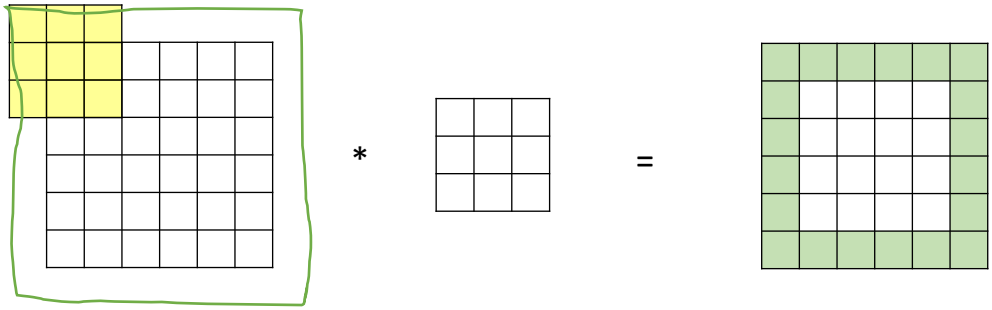
nxn * fxf => (n - f + 1)x(n – f + 1)

6x6 * 3x3 => 4x4

Two problems

1. After convolution, image is shrinked
2. Losing information at the boundaries of the image

Padding



p padding pixel around the boundaries

nxn * fxf => (n – f + 2p + 1)x(n – f + 2p + 1)

“Valid” and “same” convolutions

”Valid” = no padding

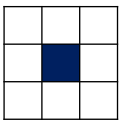
nxn * fxf => (n – f + 2p + 1)x(n – f + 2p + 1)
6x6 * 3x3 => (6 – 3 + 0 + 1) x (6 – 3 + 0 +1) = 4x4

”Same” = Pad so that output size is the same as the input size

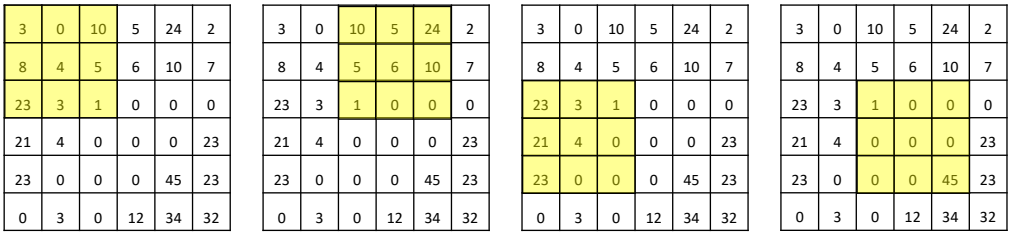
nxn * fxf => (n – f + 2p + 1)x(n – f + 2p + 1)

n – f + 2p + 1 = n
⇒ p = (f-1)/2

⇒ Kernels is often odd!
⇒ 3x3, 5x5, 7x7



Strided convolution



Stride = 2
Padding = 0

(6– 3) /2 + 1 ➔ output: 2x2

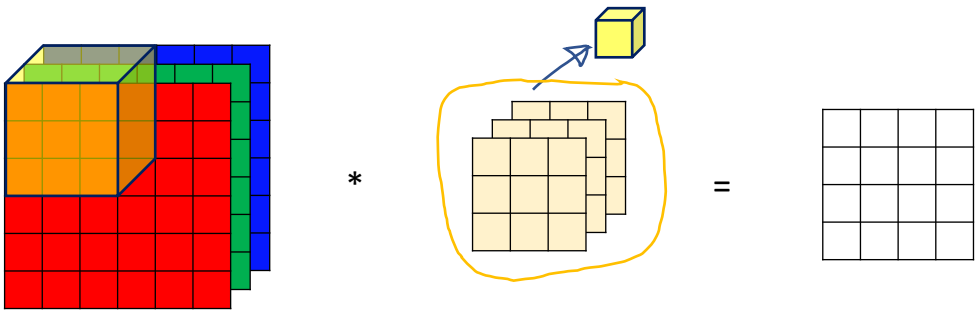
Summary of convolutions

$n \times n$ image $\quad f \times f$ filter

padding p $\quad$ stride s

Size of output: $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

Convolutions over volumes



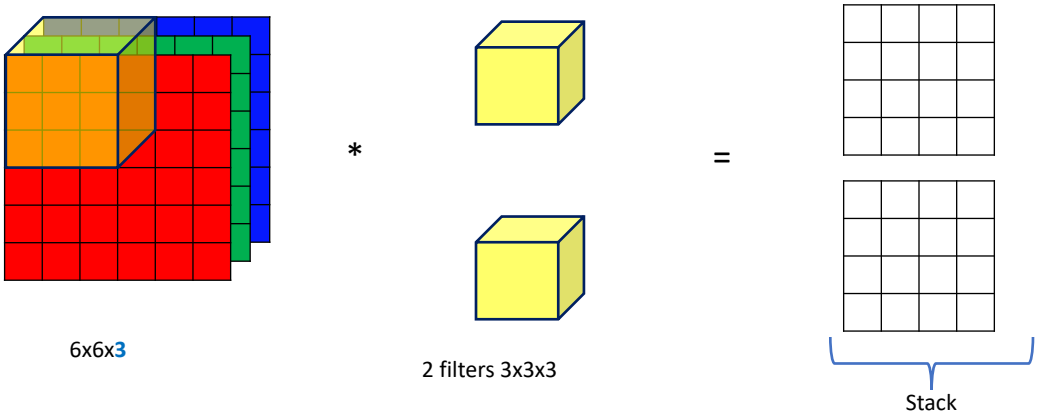
6x6x3

#channels

3x3x3

4x4

Multiple filters



6x6x3

2 filters 3x3x3

Stack

4x4x2

#channels of output

Parameters of a Convolution layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters: K,
  - Spatial extent (filter size): F,
  - Stride: S,
  - Amount of zero padding: P.
- Produces a volume of size $W_2 \times H_2 \times D_2$

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

- Introduces $F \cdot F \cdot D_1$ weights per filter
- For a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases

Reference: http://cs231n.github.io/convolutional-networks/

Types of layer in a CNN

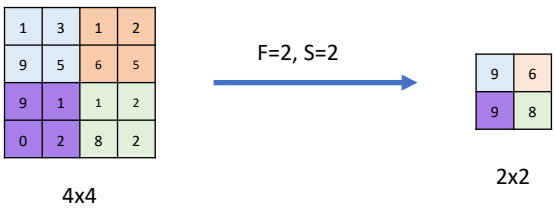
- Convolution (Conv)
- Pooling (Pool)
- Fully connected (FC)

Pooling layer: Max pooling

1	3	1	2
9	5	6	5
9	1	1	2
0	2	8	2

4x4

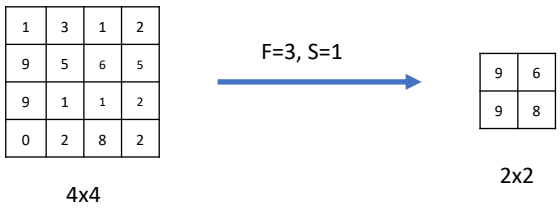
Pooling layer: Max pooling



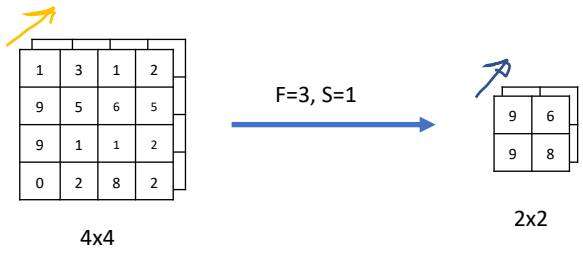
4x4

2x2

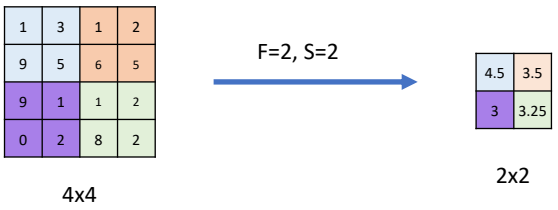
Pooling layer: Max pooling



Pooling layer: Max pooling over volume



Pooling layer: Average pooling



4x4

2x2

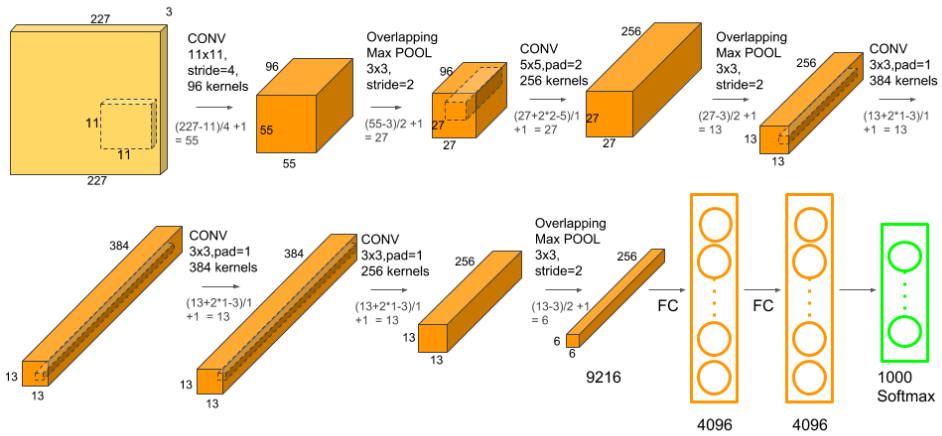
Summary of pooling

- Only hyperparameters:
  - F: filter size
  - S: stride
  - Not usually using padding P
- No parameters to learn!

Advantages of CNNs

- Parameter sharing or translation invariant
  - Once learnt, the filter can apply at any position
- Local connectivity or sparsity connection
  - One neuron can only relate to its neighborhood
  - More effective than using FC

CNN example: AlexNet



Implementation

- Link: https://colab.research.google.com/drive/1hLRa43CbzXeE0-DY7UlTrbDUeITsS7Pe

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

Mid-term assignment

- Individual report:
  - Each student chooses a paper and write a report of 2-page A4
    - The architecture
    - The reproduction
- Group presentation
  - Students choose a same paper will automatically form a group
  - Present your “discovery” at the class ☺

8 groups

- VGG-16 & VGG19
- MobileNet
- ResNet-50 & ResNet-152
  - He et al., 2015. Deep residual networks for image recognition
- ShuffleNet

- GoogLeNet & Inception-v3
- Xception
- ResNext50
- EfficientNet