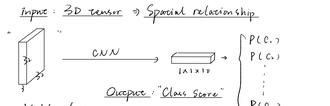
Convolutional Neural Network

① Inspirations from visual cortex

- Many neurons in the visual cortex have a small **local receptive field** reacting only to visual stimuli from a small region.
- Some neurons react to images of lines with different **orientations**.
- Some neurons have larger receptive fields and react to **more complex patterns** that are combinations of the simpler, lower-level patterns. Hierarchical
- These observations led to the **layered model** of the visual cortex in that higher-level neurons are based on the outputs of neighboring lower-level neurons.
- The CNN model is a special type of feedforward neural networks with two new building blocks: **convolutional layers** and **pooling layers**.

CNN

Input : 3D tensor ⇒ Spacial relationship



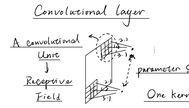
Output : "Class Score"

Hidden layer

1. Convolutional layer.
2. Pooling layer. ⇒ "□ → □"
3. Normalization layer: Accelerate and Stabilize training
4. Fully-connected layer : FNN.

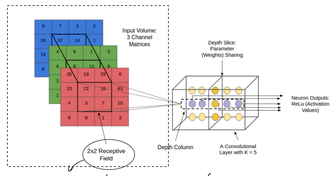
Convolutional layer

A convolutional Unit → Receptive Field



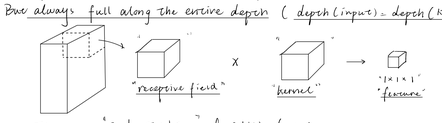
parameter Sharing ⇒ Detect same feature

One kernel (filter) applied to different locations



Receptive is local in space (3x3 input , 2x2 kernel)

But always full along the entire depth (depth(input) = depth(kernel))

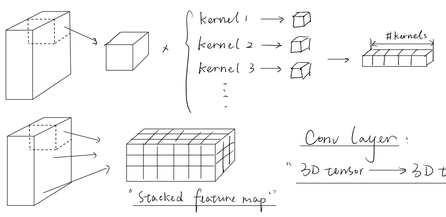


This is an "inductive bias" of CNN: Locality

We assume feature in inputs can be learned locally

(easier to compute, less parameters)

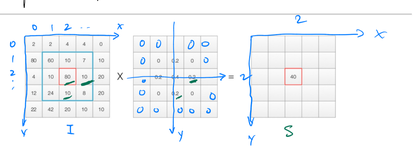
There are multiple filters in one layer



Conv layer :

" 3D tensor ⟶ 3D tensor "

Computation of a Convolutional Unit



$$S(2,2) = I(2,2)\cdot k(0,0) + I(2,3)\,k(0,1) + I(3,1)\,k(1,-1) \dots$$

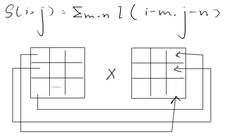
$$= \sum_{m,n} I(2+m, 2+n)\cdot k(m,n)$$

$$S(i,j) = \sum_{m,n\in[-1,0,1]} I(i+m, j+n)\,k(m,n)$$

↖ "Cross-correlation"

Convolution :

$$S(i,j) = \sum_{m,n} I(i-m, j-n)\,k(m,n)$$

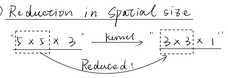


= Cross-correlation of I and "flipped K"

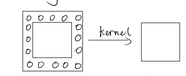
Since learnable parameters, convolution & cross-correlation can be considered the same in deep learning

Technical issues with Convolutional layer

① Reduction in Spatial size

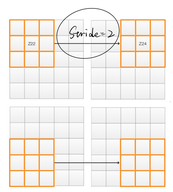


"Padding"



Size stays the same & helps Better preserve information on the edge

② Stride



Math about Convolutional Layer

Input : $W_1 \times H_1 \times D_1$

Kernel : $F \times F$

kernels : k

stride : s

padding : P

⇒

Output : $W_2 \times H_2 \times D_2$ ★ initial position

$$W_2 = \frac{(W_1 + 2P - F)}{S} + 1$$

$$H_2 = \frac{(H_1 + 2P - F)}{S} + 1$$

$$D_2 = k$$

parameters :

Conv layer : $K \cdot (F^2 D_1 + 1)$; Increase! Bias!

Fully connected layer : $(1 + W_1 H_1 D_1)\cdot W_2 H_2 D_2$

FLOPs : ("x" "+")

for each output unit

of "x" : $D_1 \cdot F^2$

of "+" : $D_1 \cdot F^2$

Total : $\geq D_1 \cdot F^2 \times W_2 H_2 D_2$ (# of outputs)

Pooling layer :

Reduce the Spacial size of the feature map

Max Pooling works the best

Another inductive bias in CNN: Shift Invariance

"Shift does not change features"

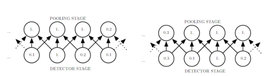
$h_1 > h_2 \Rightarrow h = \max\{h_1, h_2\}$.

When Back Propagation :

$$\frac{dh}{dh_1} = \begin{cases} 1, & h_1 > h_2 \\ 0, & h_1 < h_2 \end{cases}$$

$$\frac{dh}{dh_2} = \begin{cases} 1, & h_2 > h_1 \\ 0, & h_2 < h_1 \end{cases}$$

if we change inputs by a small amount the value of most of pooling layer does not change



Every value in the bottom row has changed, but only half of the values in the top row have changed.

Useful Property : care more about "feature" than "location"

Batch Normalization

1. Normalize input : $\frac{x-\mu}{\sigma}$

2. affine parameters (learnable) : $y = \delta x + b$.

Example CNN architecture :

ImageNet : A large visual database

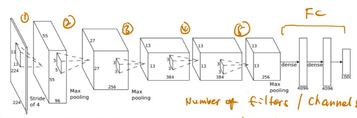
14 million images. hand-annotated (what object is pictured)

Large Scale Visual Recognition Challenge (ILSVRC)

- AlexNet
- ResNet
- VGG
- GoogleNet

① AlexNet

Tensor Shape



Filter size : 11x11, 5x5, 3x3,

5 Convolutional Layers 3 fully-connected layer

60M parameters, mostly in Fc layers

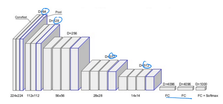
More FLOPs take place in ConV layer

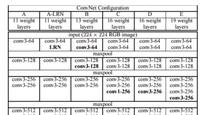
First use of ReLU

Heavy data augmentation

- Dropout: 0.5 ; Batch size: 128; SGD momentum 0.9
- Learning rate: 0.01, reduced by 10 manually when validation accuracy plateaus.
- L2 weight decay; 7 CNN ensemble.

② VGGNet





"Depth ↑" ⇒ Depth is a critical component for good performance

16 layers. always use 3x3 kernel

Totally 138M parameters

③ GoogleNet

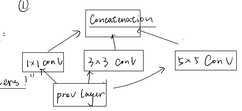
22 layers, "Inception Module" ⇒ dramatically reduced # parameters

(AlexNet: 60M ⟶ GoogleNet: 5M)

Inception Module

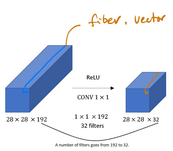
idea: Cover bigger area, but also keep fine resolution on small information ① ②

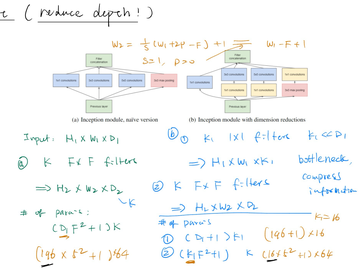
parallelization :



"Too many Parameters!"

Solution. add an 1x1 conv first (reduce depth !)





{ w/o 1x1 ConV Layer : $(D_1 F^2 + 1) \times K$

{ w/ 1x1 ConV Layer : $(D_1 + 1) K_1 + (K_1 F^2 + 1) \cdot K$

Smaller since $K_1 \ll D_1$!

Total 22 layers. starts with some vanilla layers (stem network)

Then 9 stacked inception layers

Auxiliary classification outputs to inject additional gradient at lower layers

No fully-connected layer

BP from the end → may vanishing/exploding...

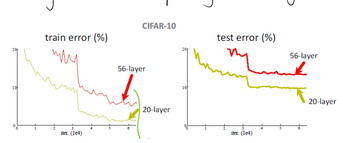
BP from the middle ⟶ propagate to lower layer

④ ResNet

Much more deeper 152 layers

if simply stack more layers

★ Higher model capacity ⟶ Higher in BOTH training/test error !



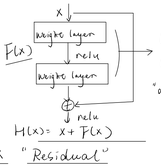
does not work !

Gradient Hard to propagate to lower layers

Plain Modules vs Residual Modules

Plain Net

Residual net



We're using layers to represent the "difference"! (residual, F(x) = H(x) - x)

output = newly learnt + original

$H(x) = x + F(x)$

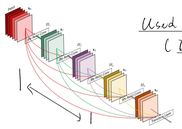
$F(x) = H(x) - x$ "Residual"

$\frac{dH}{dx} = \frac{dF}{dx} + 1$ ⇝ Crucial in BP !

No gradient vanishing since "+1"

⇒ Lower test and training error

Skip Connection



Used in follow-up models (DenseNet)

Using lower layer's output directly as part of input of higher layers

DenseNet

Concatenating features

Layers are slim (12 filters per layer) ⇒ fewer parameters

Final Classifier makes decision based on all feature maps in the network

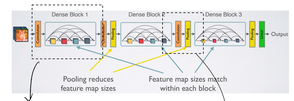
⇓

BP ↑, since each layer has direct access to gradient from loss function

$x_l = F_l([x_0, x_1, \dots, x_{l-1}])$

↓ "concatenation"

Transition layers



In each dense block

Feature maps have same size

"Transition layer"

Change the feature map size

Miscellaneous Issues

Data augmentation :

generate variations from existing ones ⇒ let model know the allowed difference

- Some commonly used operations:
  - **Translation**, **rotation**, or **scaling** by various amounts
  - **Lightning** with various contrasts
  - **Cropping**
  - **Flipping** horizontally (but note that it is not suitable for some image-based applications such as character recognition).

Hierarchical Representation Learning

Lower layers ⟶ higher layers

↓ ↓

Simple feature Complex feature