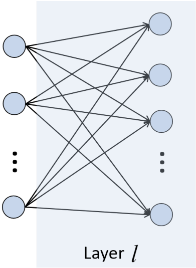
Week 4: Convolutional Neural Networks

Instructor: Ruixuan Wang
wangruix5@mail.sysu.edu.cn

School of Data and Computer Science
Sun Yat-Sen University

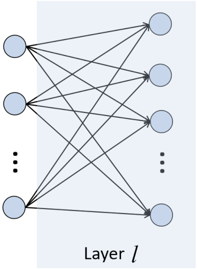
21 March, 2019

Issues in fully connected networks



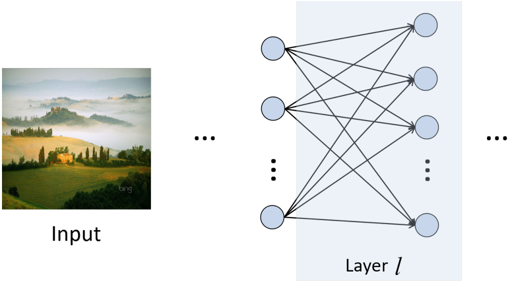
- Layer l: 1000 input signals, 1000 neurons (output signals)
- How many weight parameters at layer l?

Issues in fully connected networks



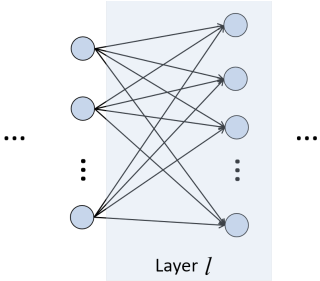
- Layer l: 1000 input signals, 1000 neurons (output signals)
- How many weight parameters at layer l?
- One million (plus 1000 bias parameters)!

Issues in fully connected networks (cont')



- Input: $100 \times 100 \times 3$; first layer: 1000 neurons
- How many weight parameters at first layer?

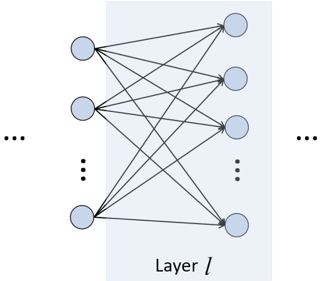
Issues in fully connected networks (cont')



- Input: $100 \times 100 \times 3$; first layer: 1000 neurons
- How many weight parameters at first layer?
- 100*100*3*1000 $=$ 30 million!

Issues in fully connected networks (cont')

Input



- Input: $100 \times 100 \times 3$; first layer: 1000 neurons
- How many weight parameters at first layer?
- 100*100*3*1000 $=$ 30 million!

Fully connected networks are not feasible for image analysis!

Expected operations on images

- To understand an image, need to recognize objects inside
- Objects (e.g., fish) could be everywhere in an image
- Need operations (e.g., fish detector) invariant to translation
- Fully connected networks have no such operations

Expected operations on images

- To understand an image, need to recognize objects inside
- Objects (e.g., fish) could be everywhere in an image
- Need operations (e.g., fish detector) invariant to translation
- Fully connected networks have no such operations

Expected operations on images (cont')


- There already exists image operations invariant to translations
- Edges in the image can be detected wherever they are
- How does such edge detector work? **Convolution!**

Expected operations on images (cont')

- There already exists image operations invariant to translations
- Edges in the image can be detected wherever they are
- How does such edge detector work? **Convolution!**

Convolution

Convolution of func $f(t)$ and $g(t)$

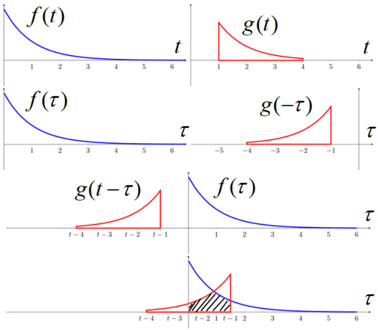
$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

figures from https://en.wikipedia.org/wiki/Convolution

Convolution

Convolution of func $f(t)$ and $g(t)$

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

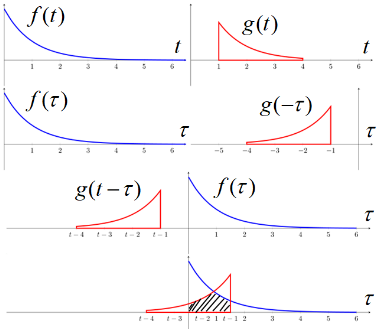


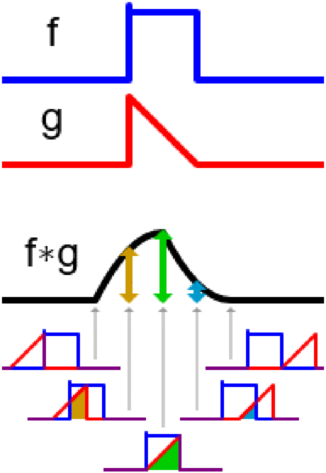
figures from https://en.wikipedia.org/wiki/Convolution

Convolution

Convolution of func $f(t)$ and $g(t)$

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$





figures from https://en.wikipedia.org/wiki/Convolution

Convolution

Discrete convolution

$$(f * g)[i] \equiv \sum_{m=-\infty}^{\infty} f[m]\, g[i-m] = \sum_{m=-\infty}^{\infty} f[i-m]\, g[m]$$

If g[m] $= 0$ when $|m| > M$, then

$$(f * g)[i] = \sum_{m=-M}^{M} f[i-m]\, g[m]$$

When both f and g have two-dimensional input,

$$(f * g)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]\, g[m,n]$$

Convolution

Discrete convolution

$$(f * g)[i] \equiv \sum_{m=-\infty}^{\infty} f[m]\, g[i-m] = \sum_{m=-\infty}^{\infty} f[i-m]\, g[m]$$

If g[m] $= 0$ when $|m| > M$, then

$$(f * g)[i] = \sum_{m=-M}^{M} f[i-m]\, g[m]$$

When both f and g have two-dimensional input,

$$(f * g)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]\, g[m,n]$$

Convolution

Discrete convolution

$$(f * g)[i] \equiv \sum_{m=-\infty}^{\infty} f[m]\, g[i-m] = \sum_{m=-\infty}^{\infty} f[i-m]\, g[m]$$

If g[m] $=$ 0 when $|m| > M$, then

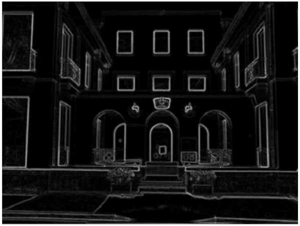
$$(f * g)[i] = \sum_{m=-M}^{M} f[i-m]\, g[m]$$

When both f and g have two-dimensional input,

$$(f * g)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]\, g[m,n]$$

Convolution (cont')

$$*\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} =$$



$f[i,j]$ $g[m,n]$ $(f*g)[i,j]$

- f is an image; g is called **kernel** or **filter**
- Different g's may detect different features, here edge features
- $(f*g)$ is called **feature map**, could be considered as an image

Convolution (cont')

A simple example

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

$f[i,j]$

2	1	0
0	2	2
2	1	0

$g[m,n]$

Convolution (cont')

$$(f * g)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]\, g[m,n]$$

3	3	2	1	0
0_0	0_1	1_2	3	1
3_2	1_2	2_0	2	3
2_0	0_1	0_2	2	2
2	0	0	0	1

$f[i,j]$

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

$(f * g)[m,n]$

- But, feature map and input image have different sizes

Convolution (cont')

$$(f * g)[i, j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]\, g[m, n]$$

3	3	2	1	0
0_{0}	0_{1}	1_{2}	3	1
3_{2}	1_{2}	2_{0}	2	3
2_{0}	0_{1}	0_{2}	2	2
2	0	0	0	1

$f[i, j]$

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

$(f * g)[m, n]$

- But, feature map and input image have different sizes

Convolution at image boundary

- **Padding**: fill image borders, often with 0's, to make feature map have the same size as that of input image

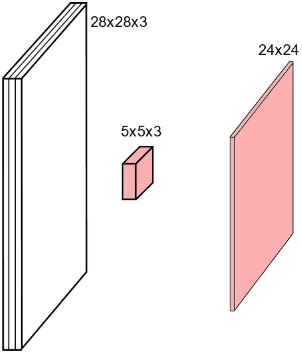
0_0	**0_1**	**0_2**	**0**	**0**	**0**	**0**
0_2	3_2	3_0	2	1	0	**0**
0_0	0_1	0_2	1	3	1	**0**
0	3	1	2	2	3	**0**
0	2	0	0	2	2	**0**
0	2	0	0	0	1	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

$f[i,j]$

6.0	**14.0**	**17.0**	**11.0**	**3.0**
14.0	12.0	12.0	17.0	**5.0**
8.0	10.0	17.0	19.0	**9.0**
11.0	9.0	6.0	14.0	**8.0**
6.0	**4.0**	**4.0**	**6.0**	**4.0**

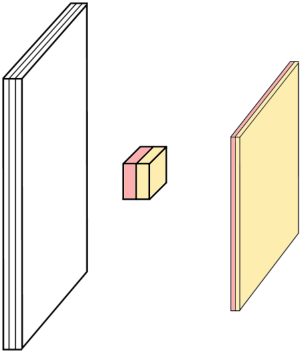
$(f*g)[m,n]$

Convolution with color images



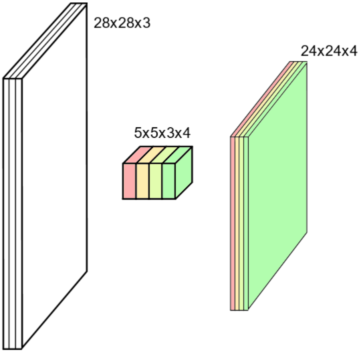
- Color image is a 3D matrix of size (height, width, channels)
- Convolution should be performed across **channels**
- So, a kernel is often 3D, with last dimension size being number of input channels

Convolution with more kernels



- Since more types of features need to be detected, more kernels (filters) are necessary

Convolution with more kernels (cont')



- Kernel size 5 (in this example)
- More kernels result in more feature maps and output channels
- Kernel shape: (kernel size, kernel size,input channels)

Figures from https://m2dsupsdlclass.github.io/lectures-labs/

Not only low-level filters

- Also want to detect mid- to high-level features
- Higher-level features derived from lower-level features
- Human visual system has such hierarchical process

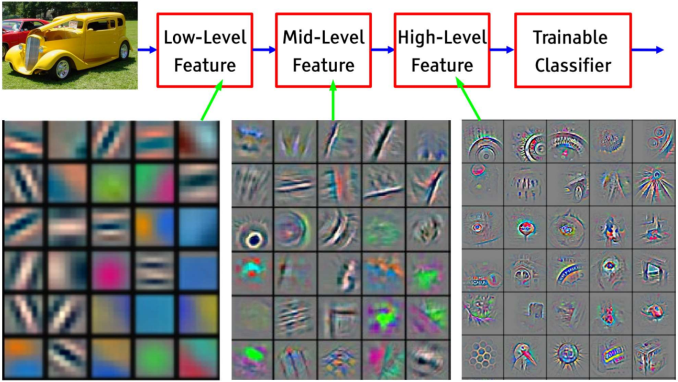
Not only low-level filters

- Also want to detect mid- to high-level features
- Higher-level features derived from lower-level features
- Human visual system has such hierarchical process

figures from https://m2dsupsdlclass.github.io/lectures-labs/

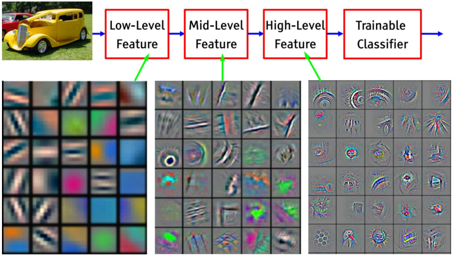
Not only low-level filters

- Also want to detect mid- to high-level features
- Higher-level features derived from lower-level features
- Human visual system has such hierarchical process



figures from https://m2dsupsdlclass.github.io/lectures-labs/

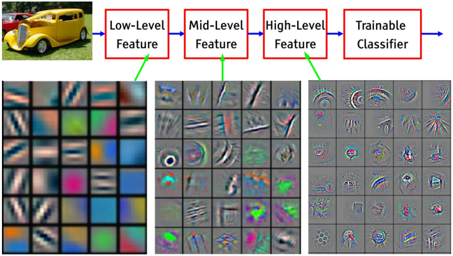
How to design these filters?



- We do not design these filters.
- All levels' kernels/filters are automatically learned!
- Feature learning: learn filters to detect features
- One end is 'input', the other end is 'output', all others automatically learned, so called '**end-to-end learning**'

This is the power of CNN or Deep Learning!

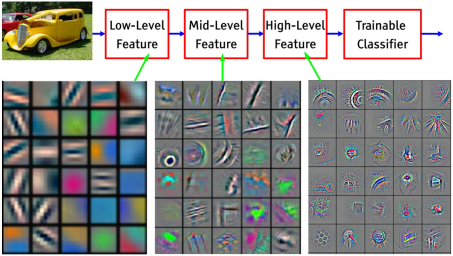
How to design these filters?



- We do not design these filters.
- All levels' kernels/filters are automatically learned!
- Feature learning: learn filters to detect features
- One end is 'input', the other end is 'output', all others automatically learned, so called '**end-to-end learning**'

This is the power of CNN or Deep Learning!

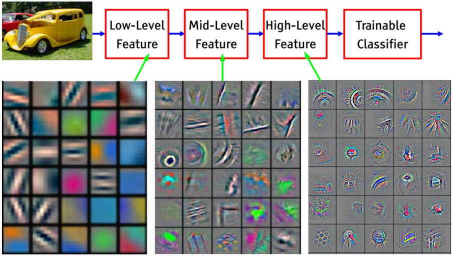
How to design these filters?



- We do not design these filters.
- All levels' kernels/filters are automatically learned!
- Feature learning: learn filters to detect features
- One end is 'input', the other end is 'output', all others automatically learned, so called '**end-to-end learning**'

This is the power of CNN or Deep Learning!

How to design these filters?



- We do not design these filters.
- All levels' kernels/filters are automatically learned!
- Feature learning: learn filters to detect features
- One end is 'input', the other end is 'output', all others automatically learned, so called '**end-to-end learning**'

This is the power of CNN or Deep Learning!

Higher-level filter learning

- Higher-level filters detect semantic object regions in images

(1) So, higher-level filters should cover larger image regions, i.e., have larger receptive field

- Higher-level features often omit fine details of objects

(2) So, should not simply enlarge kernel size

- Then, how?

Higher-level filter learning

- Higher-level filters detect semantic object regions in images

(1) So, higher-level filters should cover larger image regions, i.e., have larger receptive field

- Higher-level features often omit fine details of objects

(2) So, should not simply enlarge kernel size

- Then, how?

Higher-level filter learning

- Higher-level filters detect semantic object regions in images

(1) So, higher-level filters should cover larger image regions, i.e., have larger receptive field

- Higher-level features often omit fine details of objects

(2) So, should not simply enlarge kernel size

- Then, how?

Higher-level filter learning (cont')

Answer: higher-level filter on size-reduced lower-level feature maps

How to reduce feature map size?

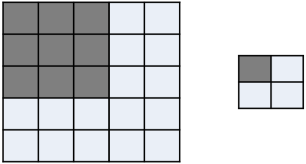
- **Stride**: step size for convolution
- Example below: kernel size 3, stride 2

Higher-level filter learning (cont')

Answer: higher-level filter on size-reduced lower-level feature maps

How to reduce feature map size?

- **Stride**: step size for convolution
- Example below: kernel size 3, stride 2

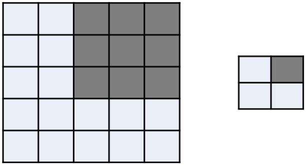


Higher-level filter learning (cont')

Answer: higher-level filter on size-reduced lower-level feature maps

How to reduce feature map size?

- **Stride**: step size for convolution
- Example below: kernel size 3, stride 2

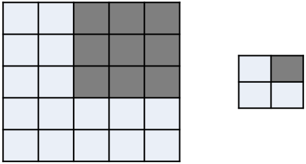


Higher-level filter learning (cont')

Answer: higher-level filter on size-reduced lower-level feature maps

How to reduce feature map size?

- **Stride**: step size for convolution
- Example below: kernel size 3, stride 2

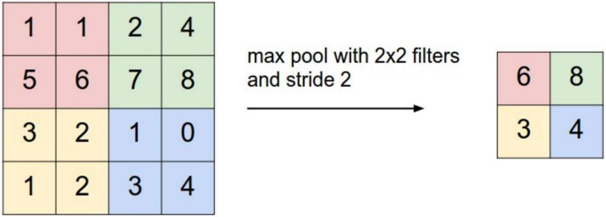


- When stride=1, output feature map is similar to input in size
- when stride=2, output (spatial) size is about half of input

Higher-level filter learning (cont')

Another way to reduce feature map size:

- **Pooling**: max or average over each 2×2 (in general) on each feature map



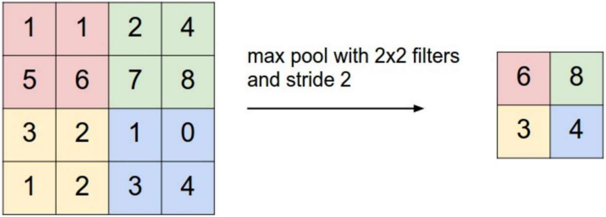
- To learn nonlinear relationship: activation function

figure from http://cs231n.github.io/convolutional-networks

Higher-level filter learning (cont')

Another way to reduce feature map size:

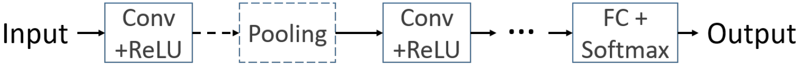
- **Pooling**: max or average over each 2×2 (in general) on each feature map



- To learn nonlinear relationship: activation function

figure from http://cs231n.github.io/convolutional-networks

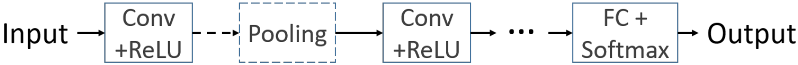
All components together: CNN



Convolutional nerual networks (CNN)

- Convolution + activation as the main operation at each layer
- Model parameters: parameters in all kernels/filters (plus bias)
- CNN Training: find optimal kernel parameters by minimizing a loss function with training dataset

All components together: CNN



Convolutional nerual networks (CNN)

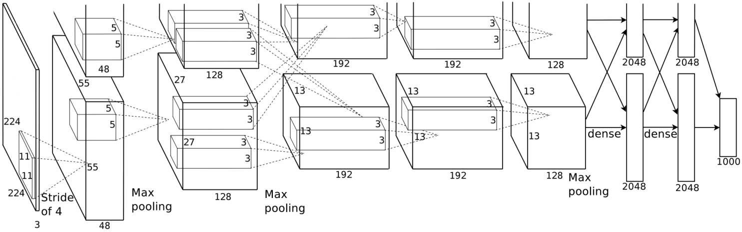
- Convolution + activation as the main operation at each layer
- Model parameters: parameters in all kernels/filters (plus bias)
- CNN Training: find optimal kernel parameters by minimizing a loss function with training dataset

Next...

Let's see a few famous CNN models!

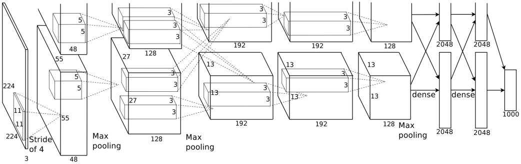
content of slides below are mainly from
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf
and https://m2dsupsdlclass.github.io/lectures-labs/

AlexNet




```
INPUT:      [227x227x3]
CONV1:      [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1:  [27x27x96]      3x3   filters at stride 2
CONV2:      [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2:  [13x13x256]     3x3   filters at stride 2
CONV3:      [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV4:      [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV5:      [13x13x256] 256 3x3   filters at stride 1, pad 1
MAX POOL3:  [6x6x256]       3x3   filters at stride 2
FC6:        [4096]      4096 neurons
FC7:        [4096]      4096 neurons
FC8:        [1000]      1000 neurons (softmax logits)
```

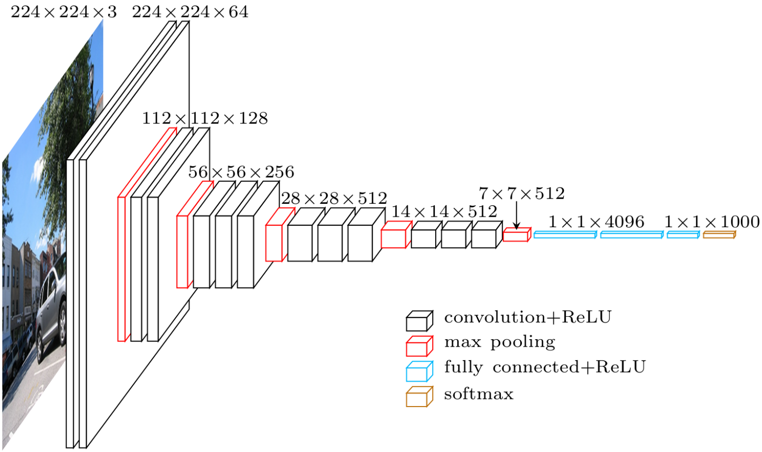
AlexNet



- first use of ReLU
- data augmentation
- dropout 0.5
- batch size 128
- ensemble: 7 CNN classifiers
- SGD momentum 0.9
- learning rate 0.01, divided by 10 for a few times
- L2 regularization

Krizhevsky, Alex, Sutskever, Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012

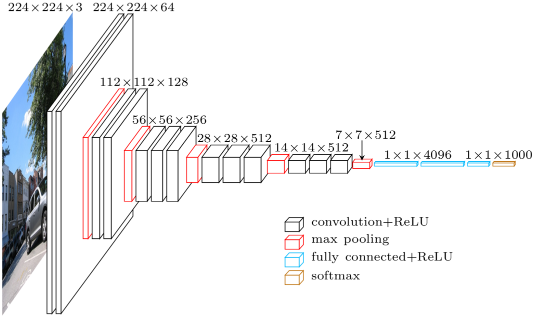
VggNet



- small-size (3×3) filters, fewer model parameters
- more layers therefore more non-linearities

Simonyan, Karen, Zisserman, "Very deep convolutional networks for large-scale image recognition", 2014.

VggNet (cont')



	Activation maps		Parameters		
INPUT:	[224x224x3]	= 150K	0		
CONV3-64:	[224x224x64]	= 3.2M	(3x3x3)x64	=	1,728
CONV3-64:	[224x224x64]	= 3.2M	(3x3x64)x64	=	36,864
POOL2:	[112x112x64]	= 800K	0		

figure from https://m2dsupsdlclass.github.io/lectures-labs/

VggNet (cont')

	Activation maps		Parameters		
INPUT:	[224x224x3]	= 150K	0		
CONV3-64:	[224x224x64]	= 3.2M	(3x3x3)x64	=	1,728
CONV3-64:	[224x224x64]	= 3.2M	(3x3x64)x64	=	36,864
POOL2:	[112x112x64]	= 800K	0		
CONV3-128:	[112x112x128]	= 1.6M	(3x3x64)x128	=	73,728
CONV3-128:	[112x112x128]	= 1.6M	(3x3x128)x128	=	147,456
POOL2:	[56x56x128]	= 400K	0		
CONV3-256:	[56x56x256]	= 800K	(3x3x128)x256	=	294,912
CONV3-256:	[56x56x256]	= 800K	(3x3x256)x256	=	589,824
CONV3-256:	[56x56x256]	= 800K	(3x3x256)x256	=	589,824
POOL2:	[28x28x256]	= 200K	0		
CONV3-512:	[28x28x512]	= 400K	(3x3x256)x512	=	1,179,648
CONV3-512:	[28x28x512]	= 400K	(3x3x512)x512	=	2,359,296
CONV3-512:	[28x28x512]	= 400K	(3x3x512)x512	=	2,359,296
POOL2:	[14x14x512]	= 100K	0		
CONV3-512:	[14x14x512]	= 100K	(3x3x512)x512	=	2,359,296
CONV3-512:	[14x14x512]	= 100K	(3x3x512)x512	=	2,359,296
CONV3-512:	[14x14x512]	= 100K	(3x3x512)x512	=	2,359,296
POOL2:	[7x7x512]	= 25K	0		
FC:	[1x1x4096]	= 4096	7x7x512x4096	=	102,760,448
FC:	[1x1x4096]	= 4096	4096x4096	=	16,777,216
FC:	[1x1x1000]	= 1000	4096x1000	=	4,096,000

TOTAL activations: 24M x 4 bytes ~= 93MB / image (x2 for backward)
TOTAL parameters: 138M x 4 bytes ~= 552MB (x2 for plain SGD, x4 for Adam)

VggNet (cont')

Both feature maps and model parameters consume GPU memory!

When GPU memory is an issue:

VggNet (cont')

Both feature maps and model parameters consume GPU memory!

When GPU memory is an issue:

- reduce batch size
- reduce input data size
- reduce number of FC layers
- reduce neurons in FC layers
- reduce number of input to FC layers
- or use more GPUs, etc.

Problem of deeper networks

How to reduce number of input to 1st FC layer?

- Reduce size of feature map of the last conv layer, e.g., by increasing more conv layers (with pooling)

figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

Problem of deeper networks

How to reduce number of input to 1st FC layer?

- Reduce size of feature map of the last conv layer, e.g., by increasing more conv layers (with pooling)
- However, more layers caused larger **training** and test error!

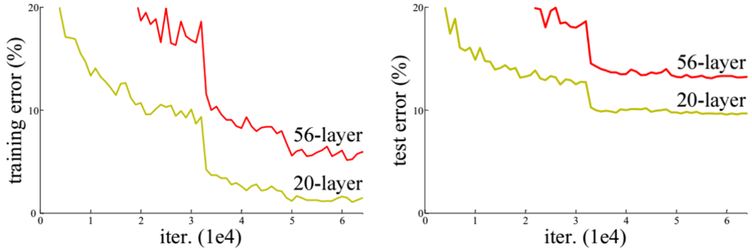
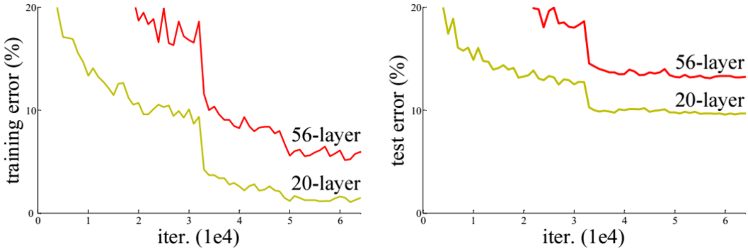


figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

Problem of deeper networks

How to reduce number of input to 1st FC layer?

- Reduce size of feature map of the last conv layer, e.g., by increasing more conv layers (with pooling)
- However, more layers caused larger **training** and test error!



- Deeper network not overfitting, but harder to optimize!

figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

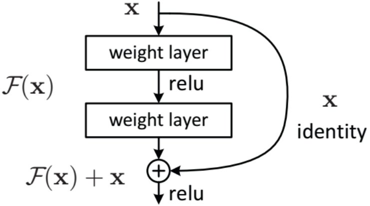


Figure 2. Residual learning: a building block.

- Learning residual between desired mapping $\mathcal{H}(\mathrm{x})$ and input x

$$\begin{aligned}\mathcal{H}(\mathrm{x}) &= \mathcal{F}(\mathrm{x}) + \mathrm{x} \\ \mathcal{F}(\mathrm{x}) &= \mathcal{H}(\mathrm{x}) - \mathrm{x}\end{aligned}$$

- If $\mathcal{H}(\mathrm{x})$ is identity mapping, it is easier to push residual to zero than to fit an identity mapping by a stack of nonlinear layers

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

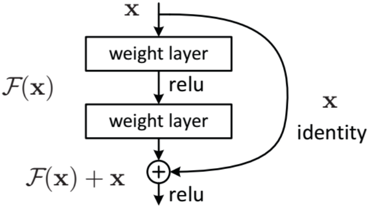


Figure 2. Residual learning: a building block.

- Learning residual between desired mapping $\mathcal{H}(\mathbf{x})$ and input $\mathbf{x}$

$$\begin{aligned} \mathcal{H}(\mathbf{x}) &= \mathcal{F}(\mathbf{x}) + \mathbf{x} \\ \mathcal{F}(\mathbf{x}) &= \mathcal{H}(\mathbf{x}) - \mathbf{x} \end{aligned}$$

- If $\mathcal{H}(\mathbf{x})$ is identity mapping, it is easier to push residual to zero than to fit an identity mapping by a stack of nonlinear layers

ResNet

Solution: use network layer to learn residual mapping rather than directly to learn a desired underlying mapping!

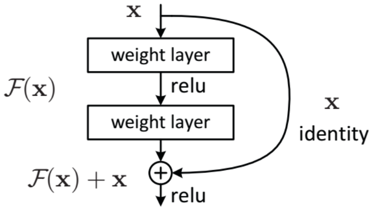


Figure 2. Residual learning: a building block.

- Learning residual between desired mapping $\mathcal{H}(\mathbf{x})$ and input $\mathbf{x}$

$$\begin{aligned}\mathcal{H}(\mathbf{x}) &= \mathcal{F}(\mathbf{x}) + \mathbf{x} \\ \mathcal{F}(\mathbf{x}) &= \mathcal{H}(\mathbf{x}) - \mathbf{x}\end{aligned}$$

- If $\mathcal{H}(\mathbf{x})$ is identity mapping, it is easier to push residual to zero than to fit an identity mapping by a stack of nonlinear layers

ResNet (cont')

Why ResNet works better?

- train/update each layer easier
- an ensemble of CNN models with different architectures

ResNet properties:

- stack residual blocks
- double filters periodically
- downsample maps by stride 2
- gobal avg pooling on last conv layer
- no FC layers until output
- parameters 25M (ResNet-50) vs 138M (VggNet)
- computation 3.8B vs 15.3B flops

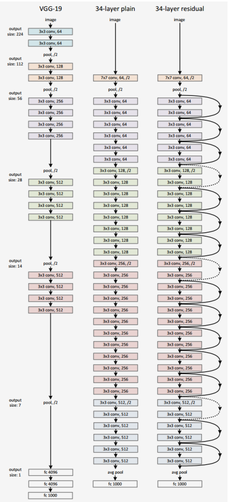


figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

ResNet (cont')

Why ResNet works better?

- train/update each layer easier
- an ensemble of CNN models with different architectures

ResNet properties:

- stack residual blocks
- double filters periodically
- downsample maps by stride 2
- gobal avg pooling on last conv layer
- no FC layers until output
- parameters 25M (ResNet-50) vs 138M (VggNet)
- computation 3.8B vs 15.3B flops

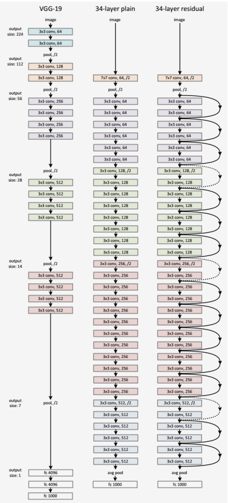
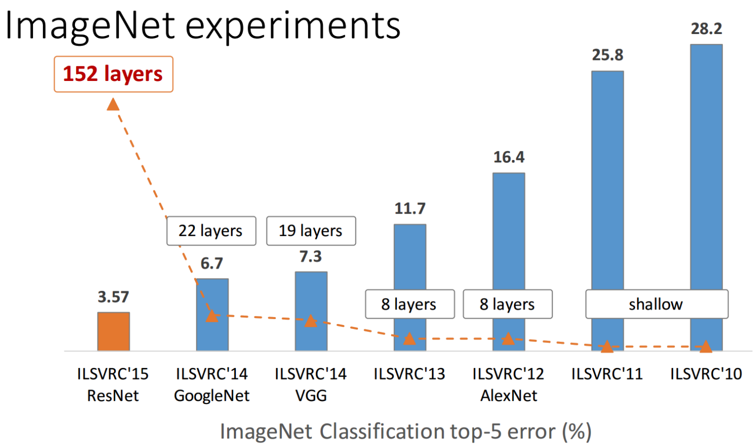


figure from He, Zhang, Ren, Sun, "Deep residual learning for image recognition", CVPR 2016

ResNet (cont')



- Better than humans on 1000-class image classification!

figure from Kaiming He, "Deep residual learning for image recognition", tutorial on ICML 2016

Summary

- Core operation in CNN is convolution
- CNN can hierarchically extract low- to high-level features
- Power of deep learning is to learn filters end-to-end!
- Resnet outperforms humans in multiple tasks

Further reading:

- Chapter 9 in textbook "Deep learning", http://www.deeplearningbook.org/
- cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf