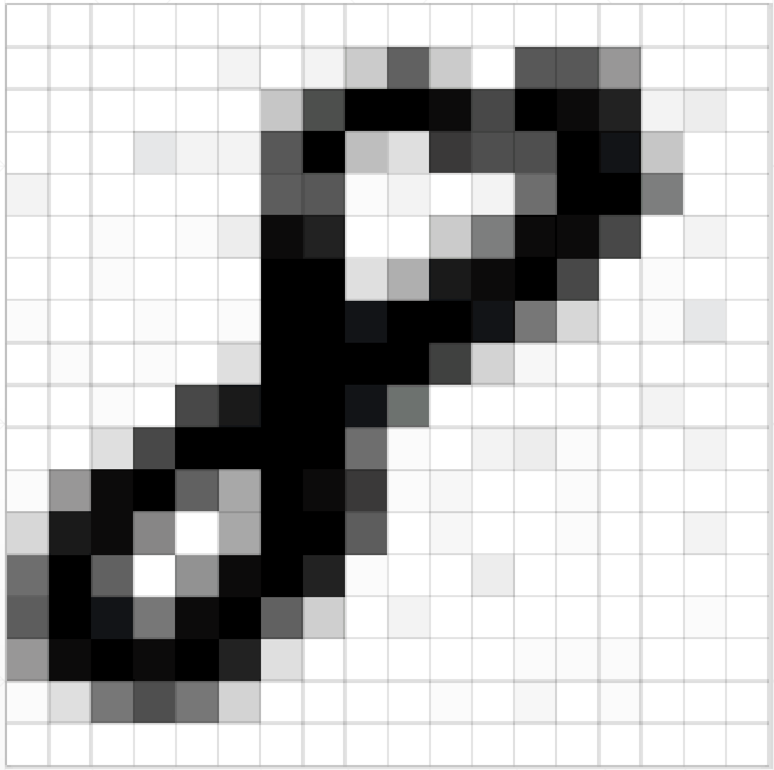
PyTorch

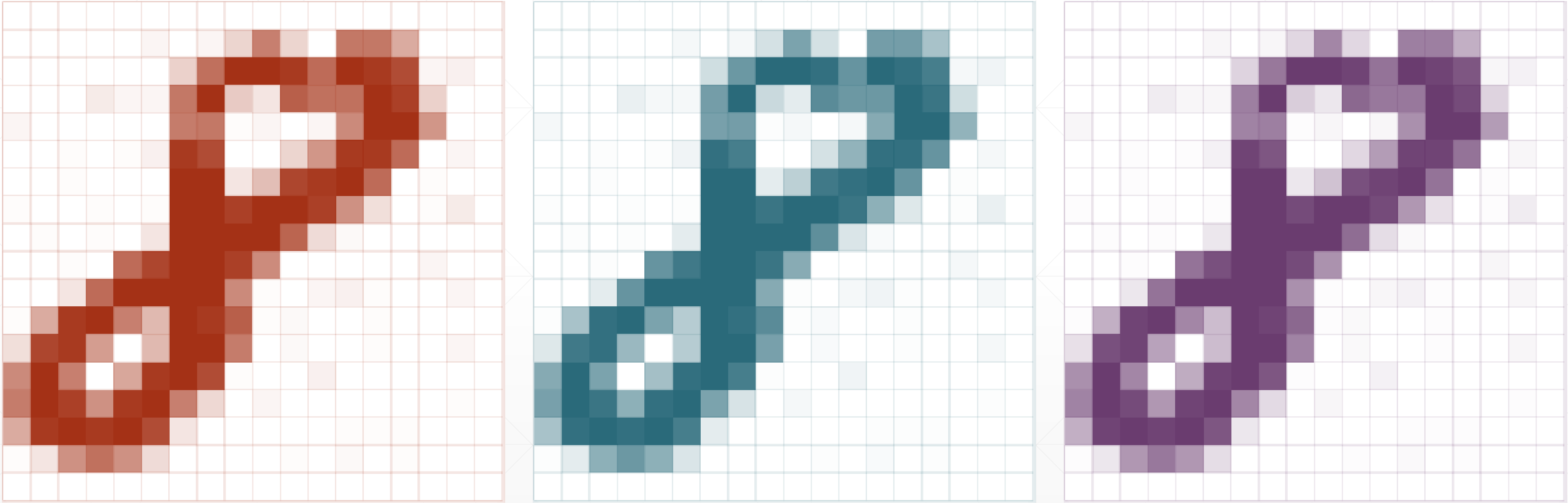
什么是卷积

主讲人：龙良曲

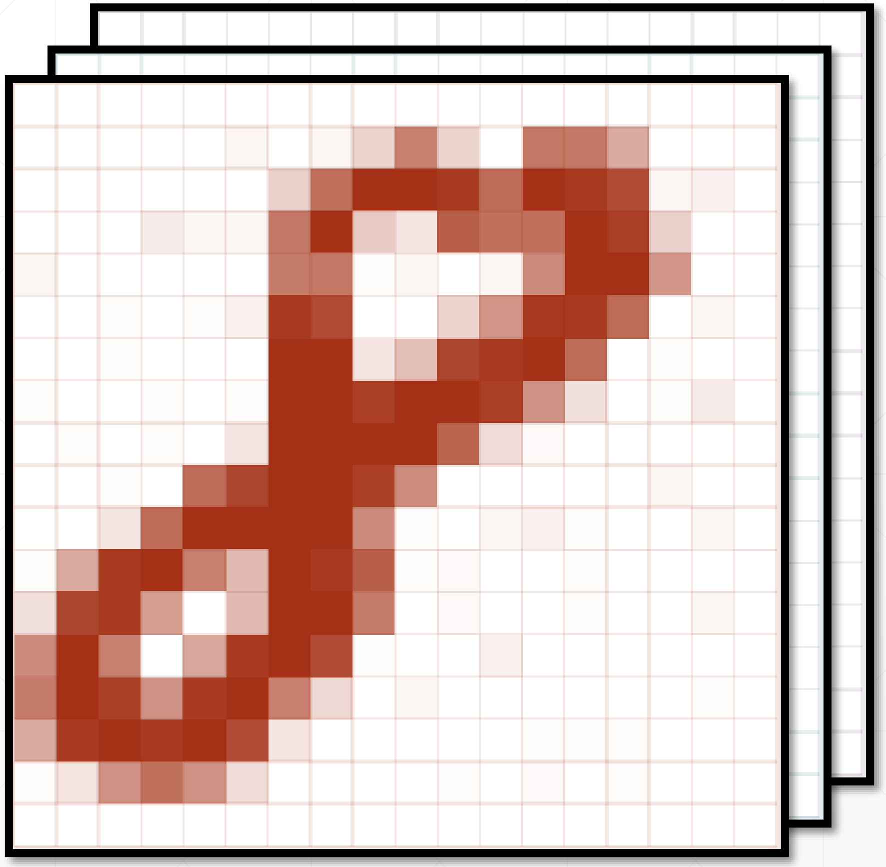
Feature Maps



Feature maps

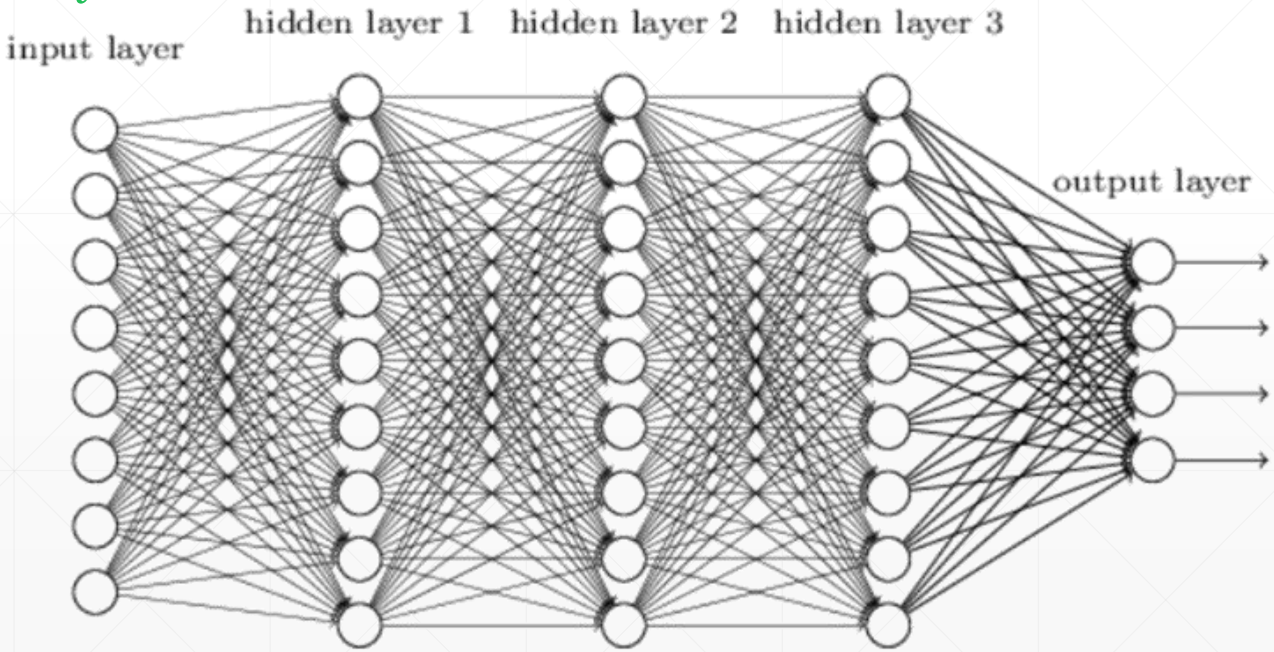


Feature maps



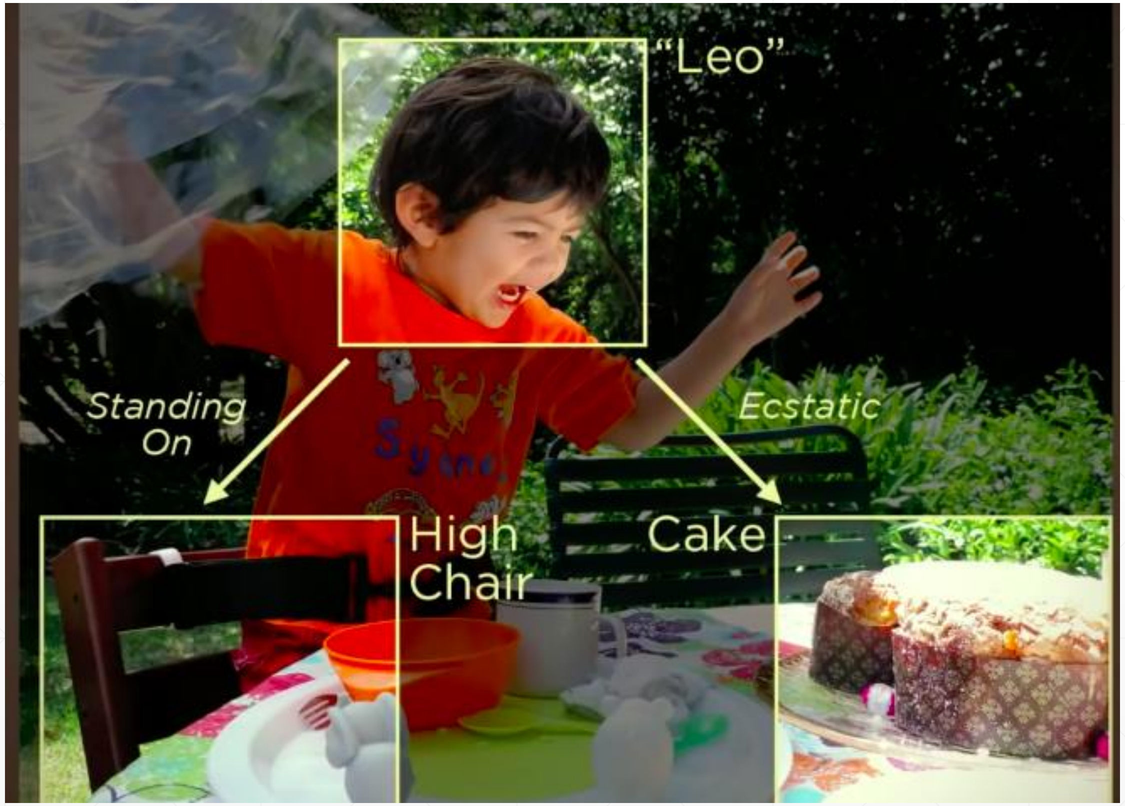
What's wrong with Linear

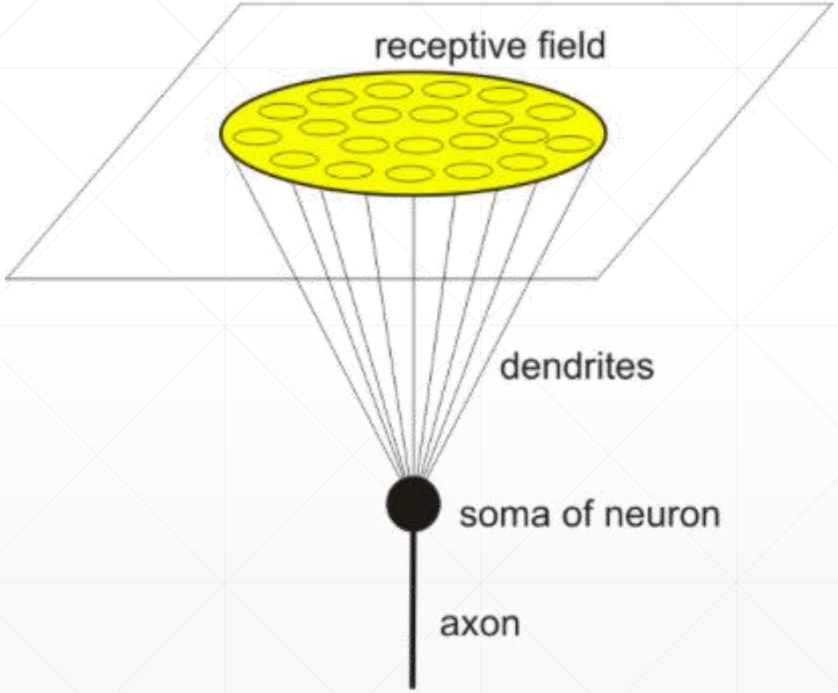
- 4 Hidden Layers: [784, 256, 256, 256, 256, 10]
  - 390K parameters
  - 1.6MB memory
  - 80386



http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf

Receptive Field

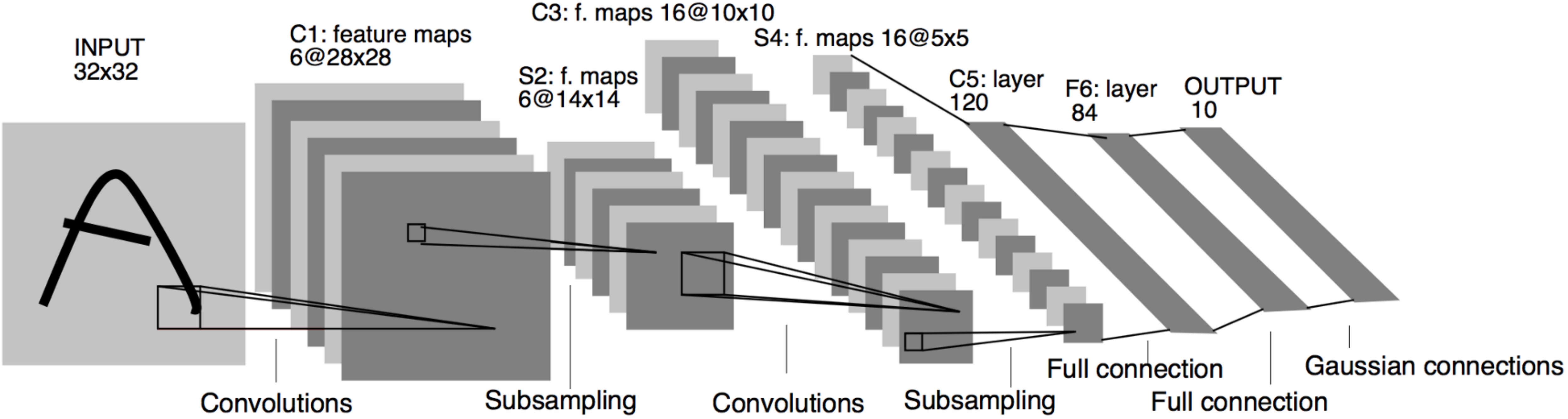




https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

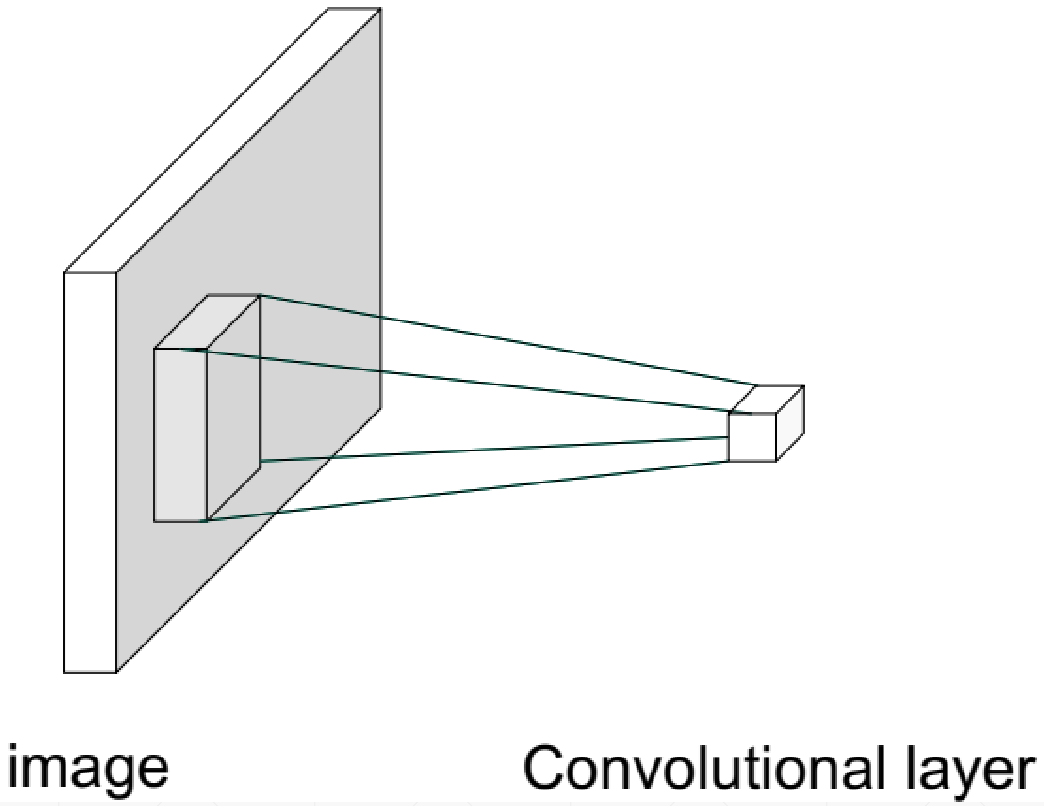
Weight sharing

- ~60k parameters
- 6 Layers

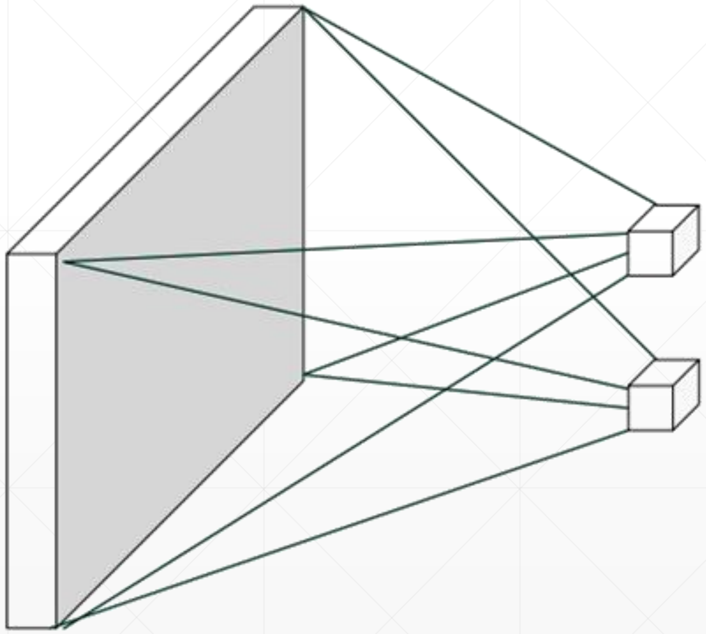


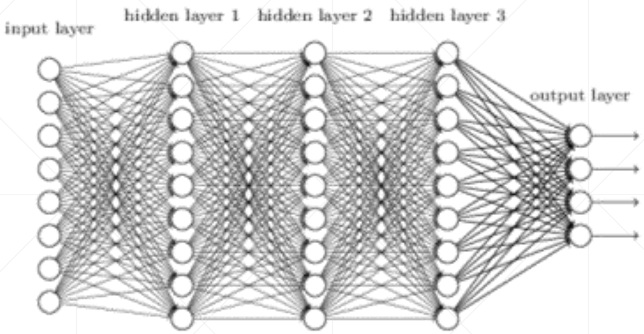
http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf

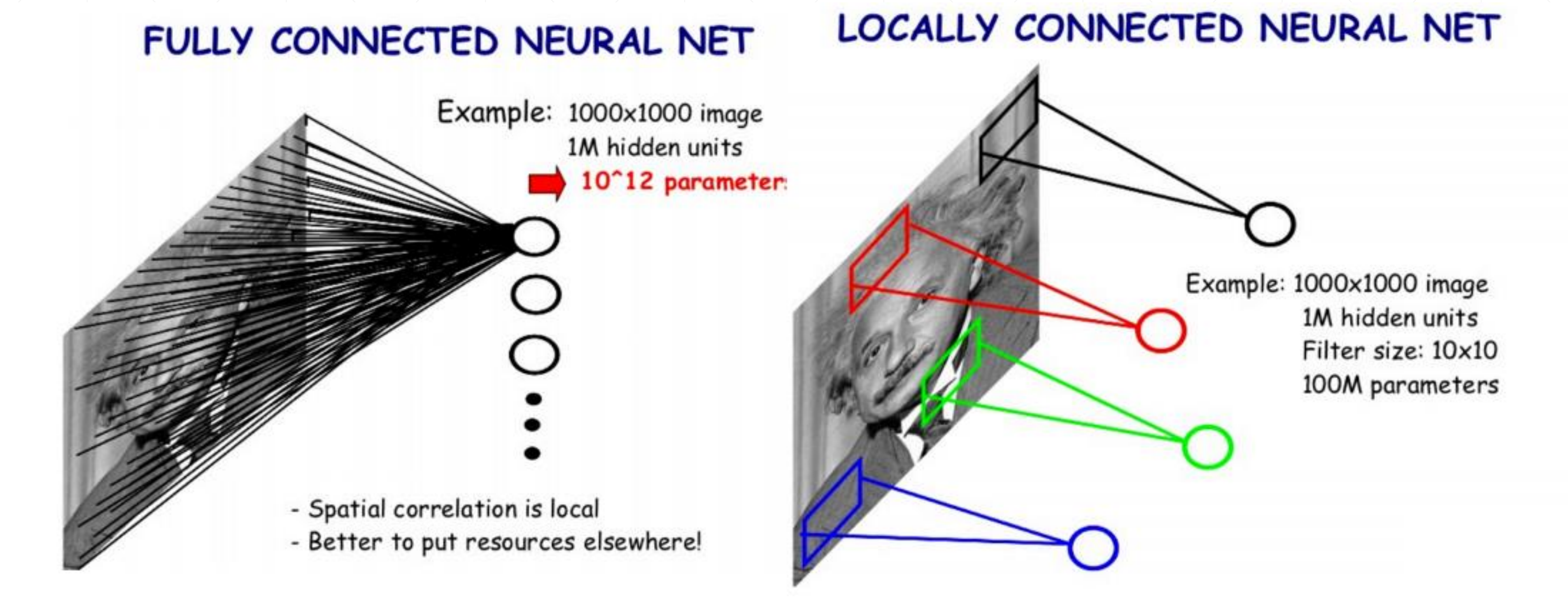
Convolution Operation



Rethink Linear layer

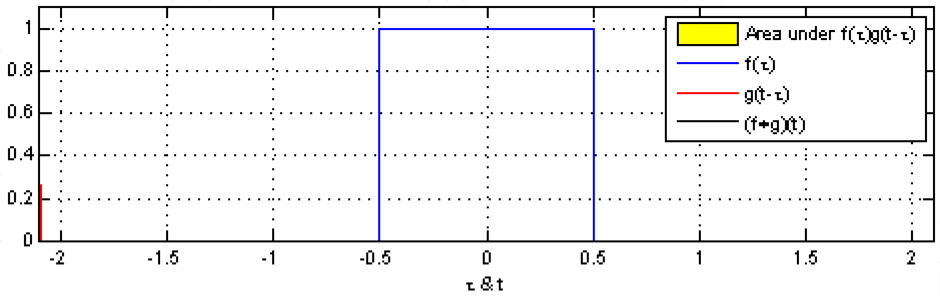


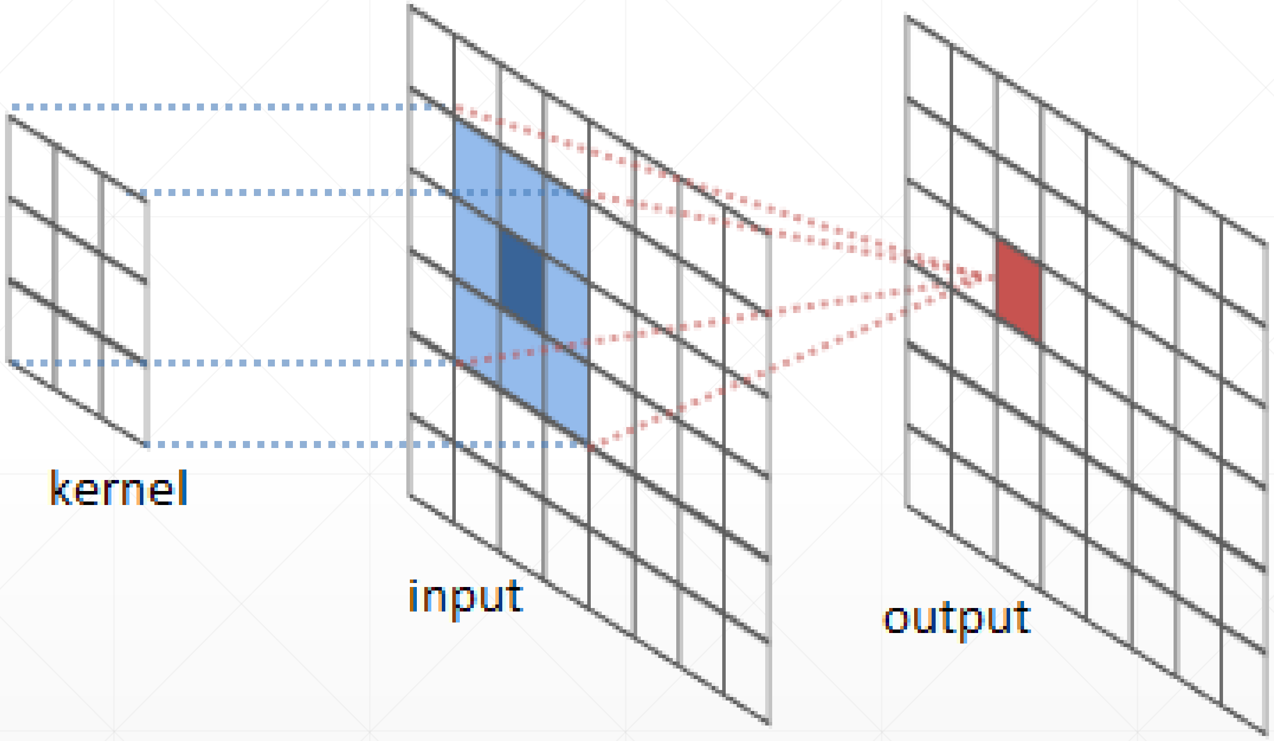



FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameter
- Spatial correlation is local
- Better to put resources elsewhere!
LOCALLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Why call Convolution?

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$




kernel
input
output

Convolution

Sharpen:

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

https://www.superdatascience.com/convolutional-neural-networks-cnn-step-1-convolution-operation/

Convolution

Blur:

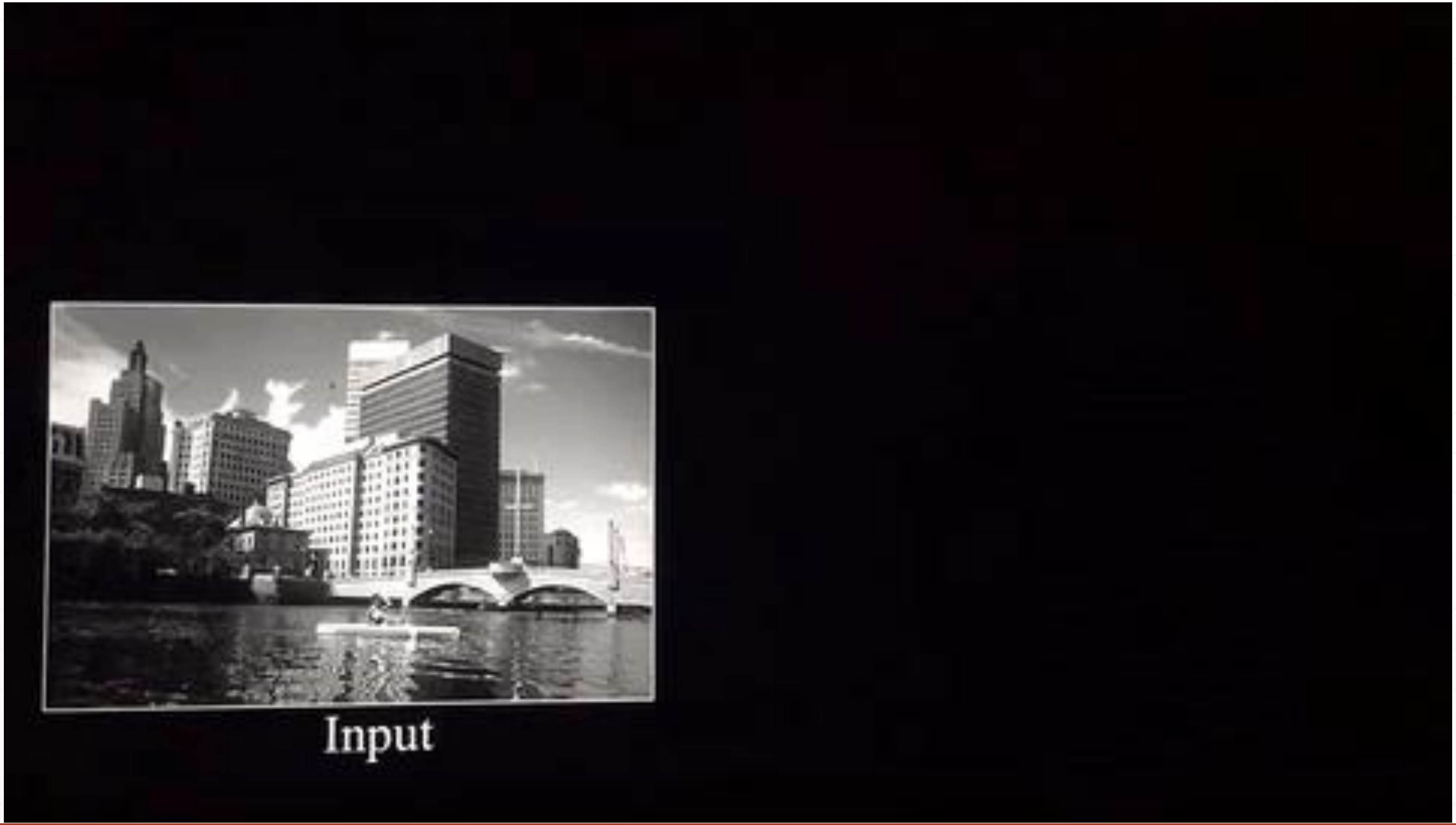
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Convolution

Edge Detect:

	0	1	0	
	1	-4	1	
	0	1	0	

CNN on feature maps



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

下一课时

卷积神经网络

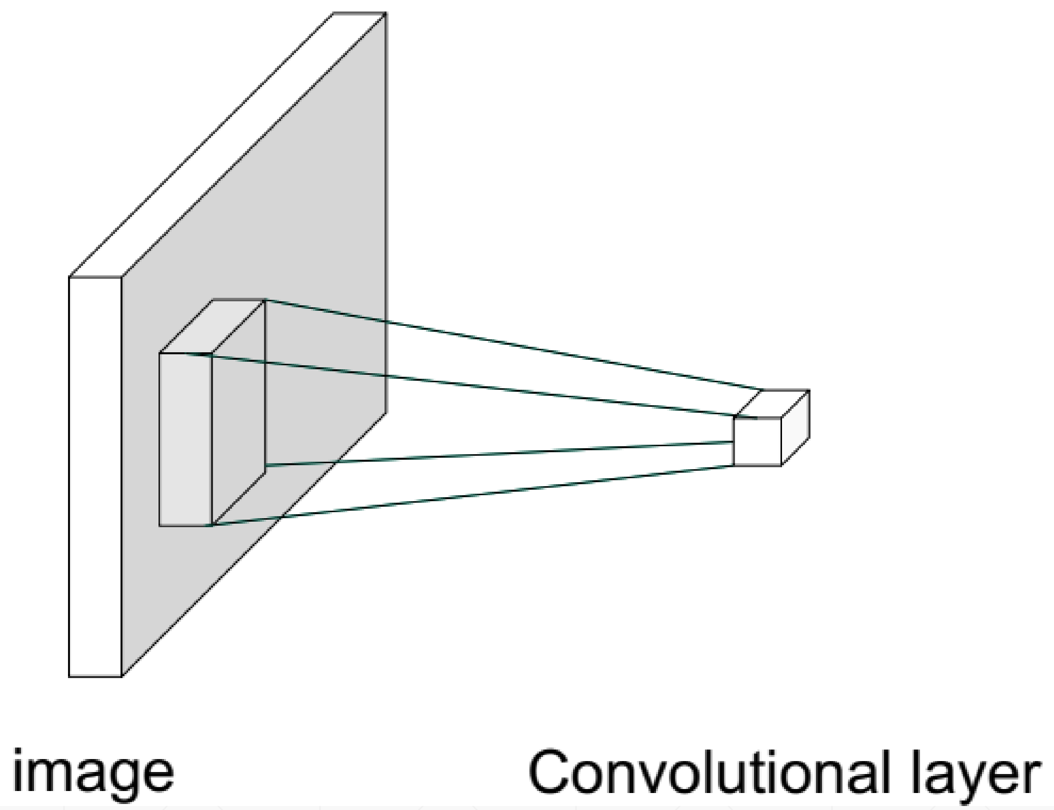
Thank You.

PyTorch

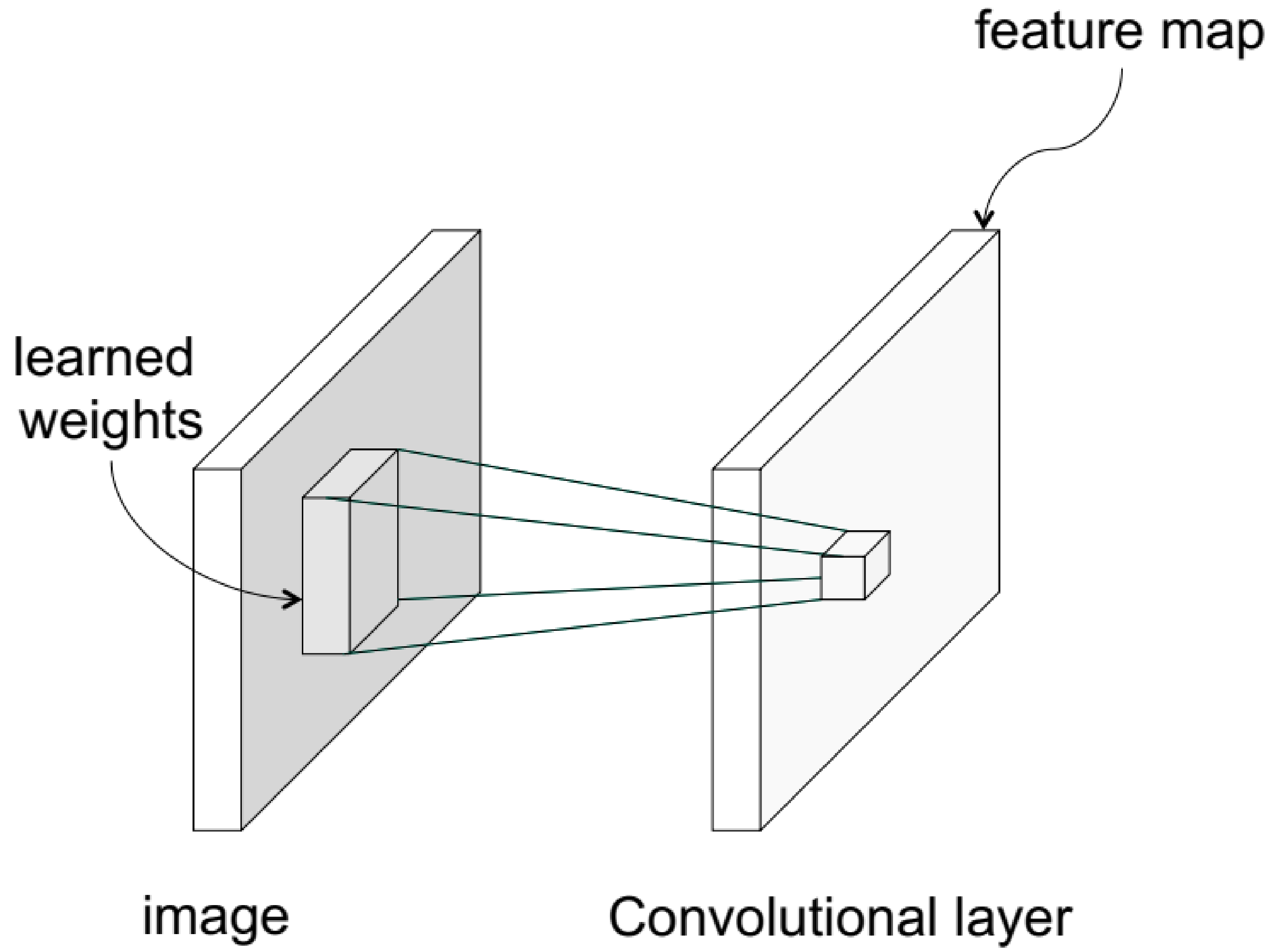
卷积神经网络

主讲人：龙良曲

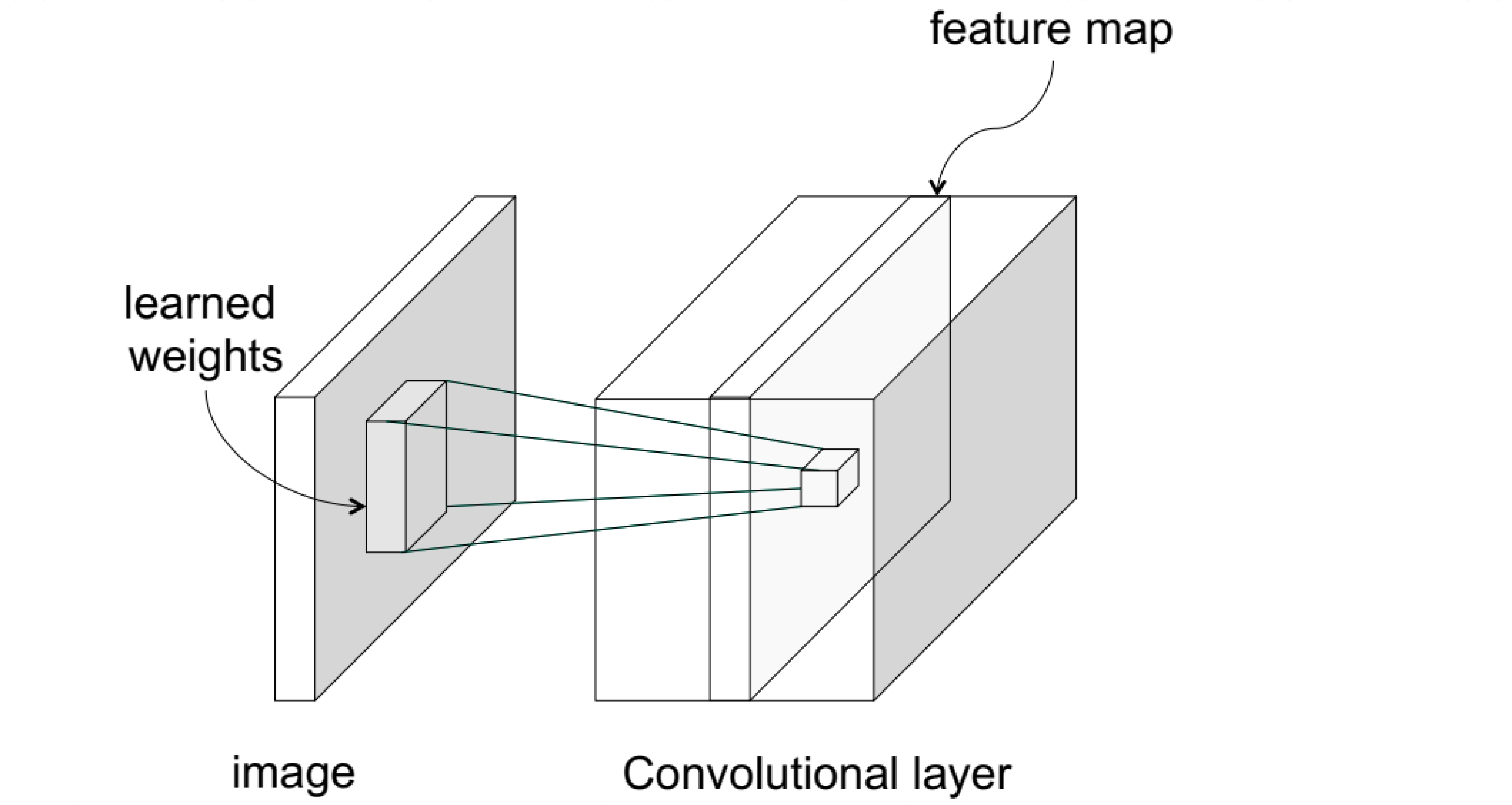
Convolution



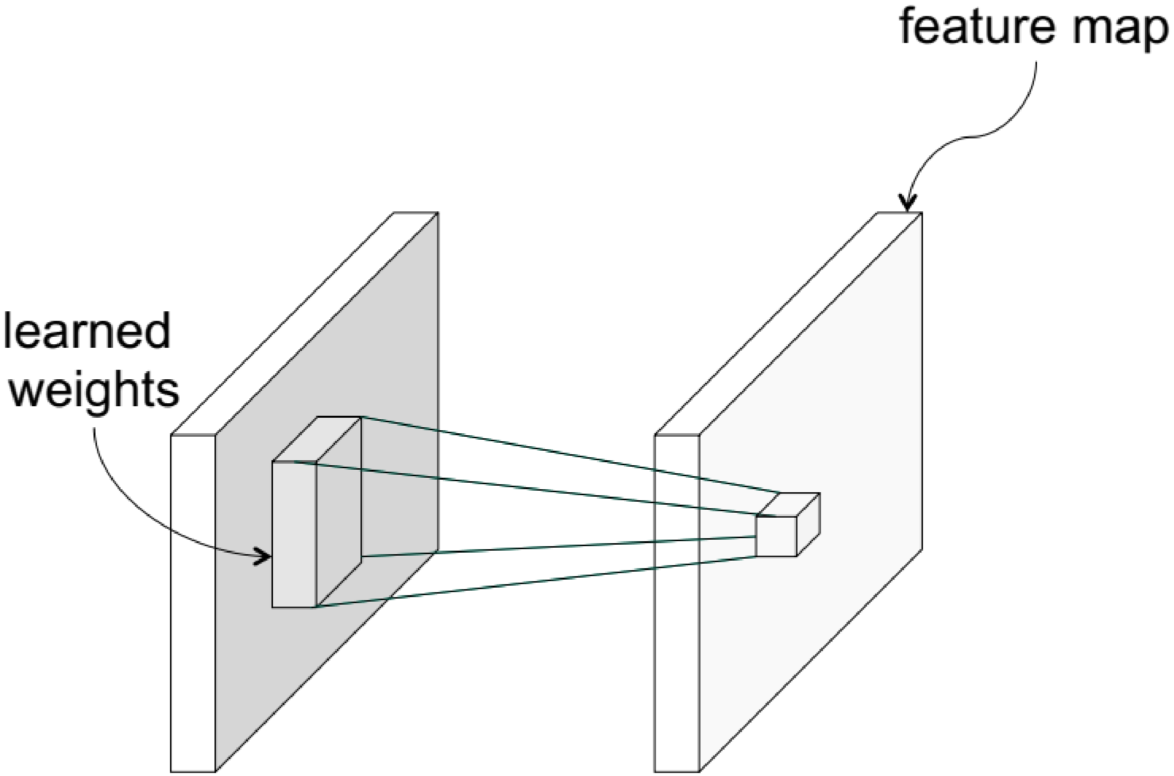
Moving window



Several kernels



Animation

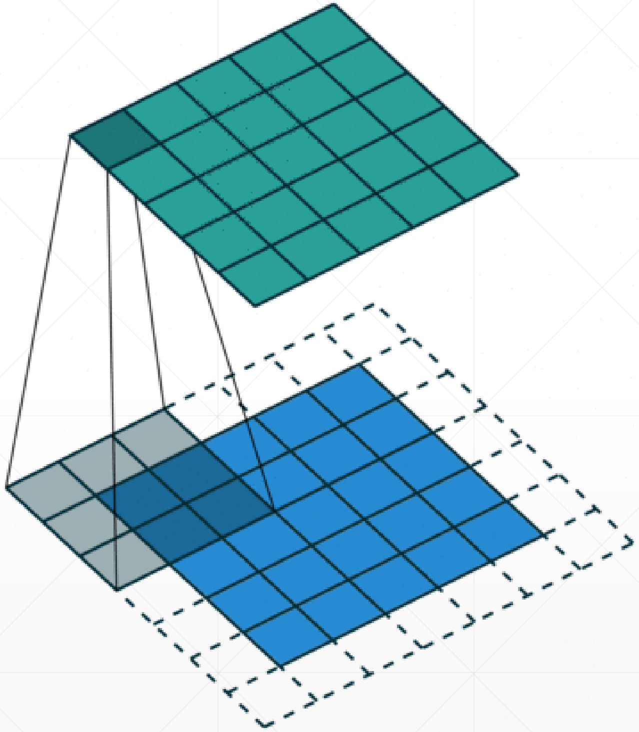


1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

Notation



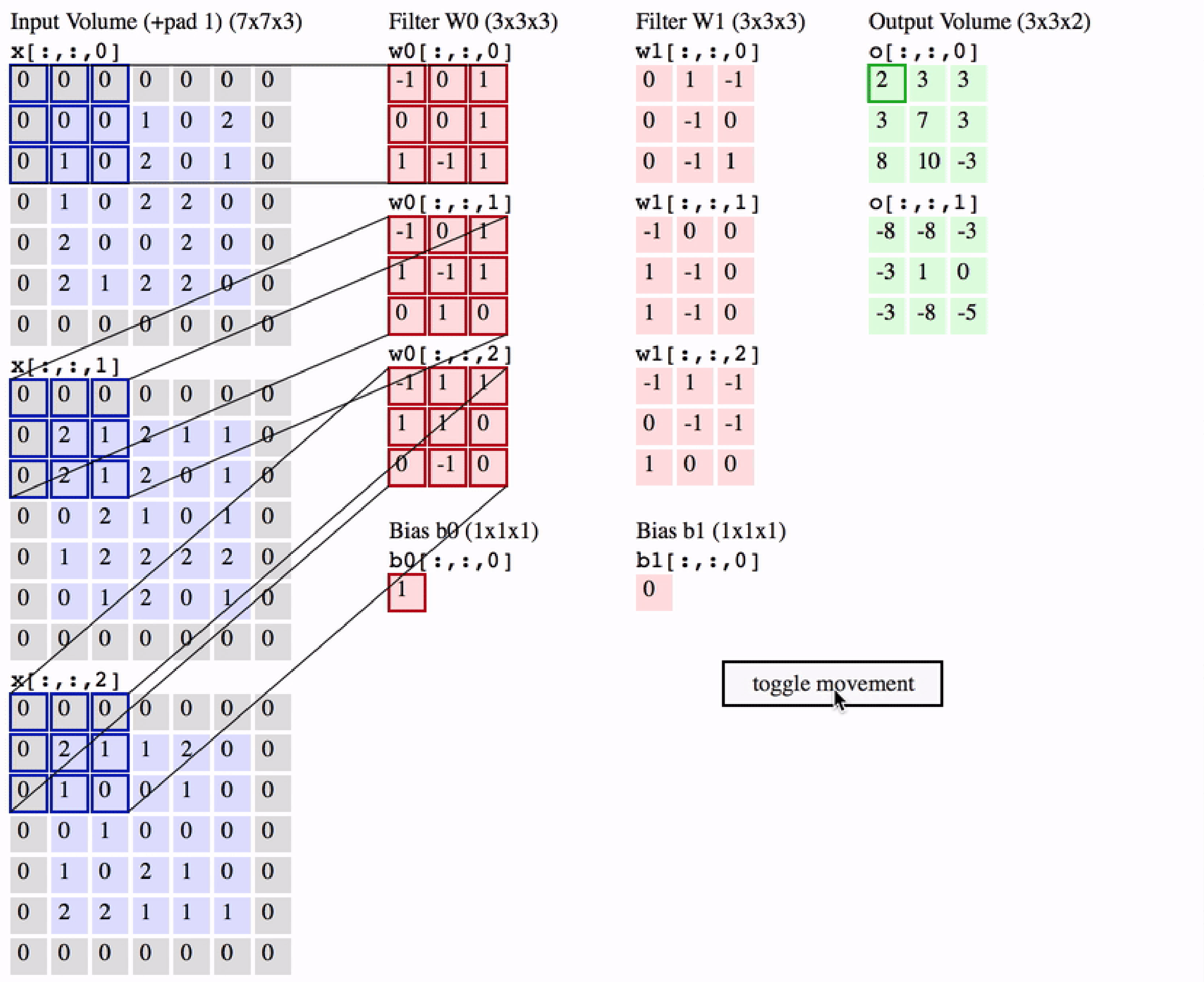
Input_channels:
Kernel_channels: 2 ch
Kernel_size:
Stride:
Padding:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Multi-Kernels



x: [b, 3, 28, 28]

one k: [3, 3, 3]

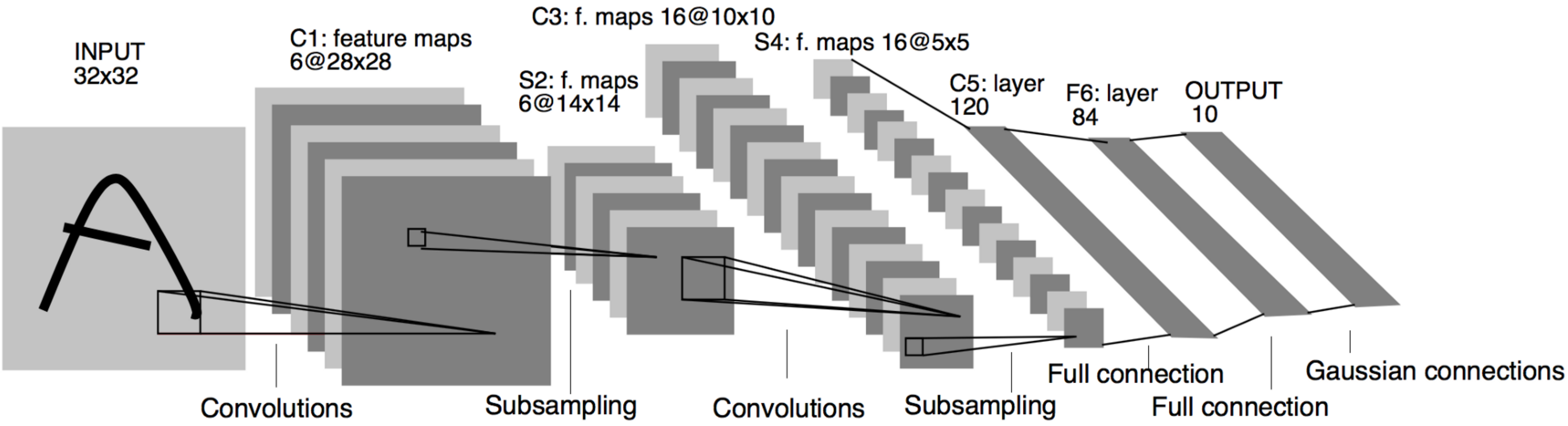
multi-k: [16, 3, 3, 3]

bias: [16]

out: [b, 16, 28, 28]

https://skymind.ai/wiki/convolutional-network

LeNet-5

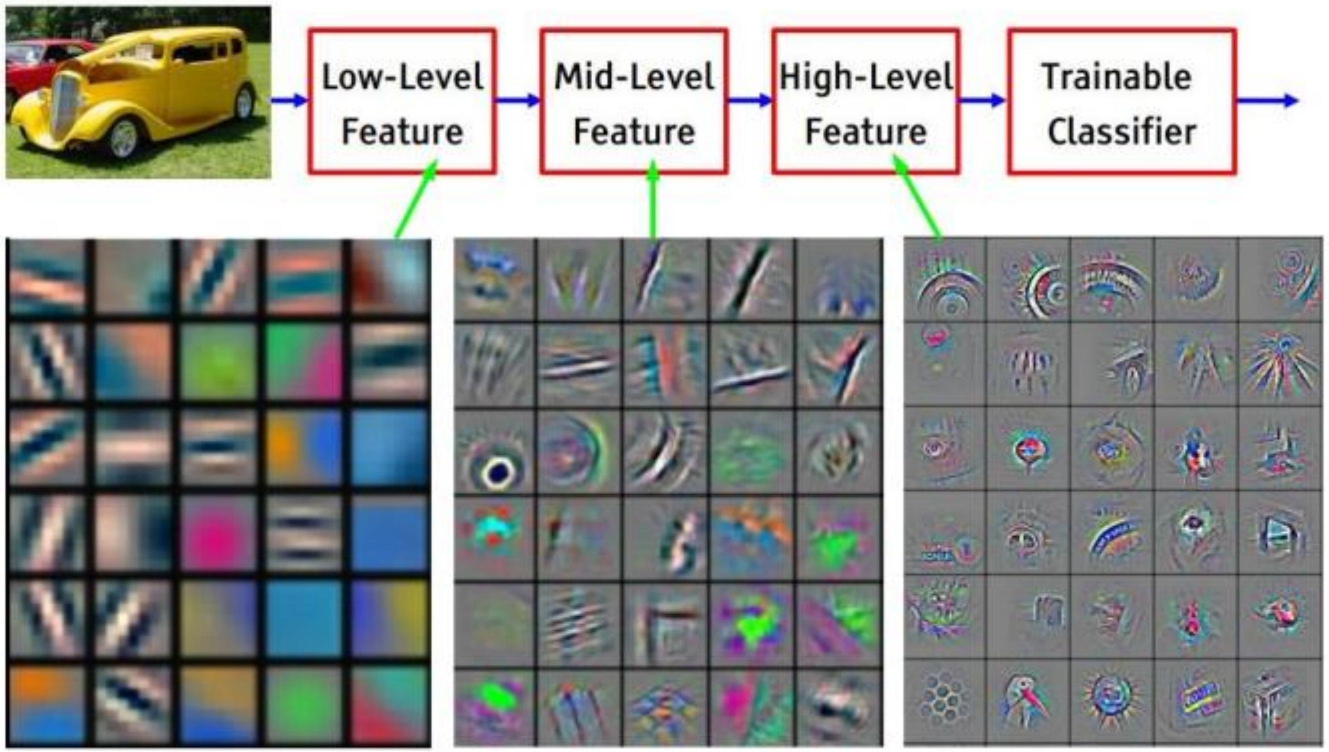


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

Pyramid Architecture

Preview

[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

nn.Conv2d

```
In [9]: layer=nn.Conv2d(1,3,kernel_size=3,stride=1,padding=0)
In [10]: x=torch.rand(1,1,28,28)

In [11]: out=layer.forward(x)
Out[12]: torch.Size([1, 3, 26, 26])

In [13]: layer=nn.Conv2d(1,3,kernel_size=3,stride=1,padding=1)
In [14]: out=layer.forward(x)
Out[15]: torch.Size([1, 3, 28, 28])

In [16]: layer=nn.Conv2d(1,3,kernel_size=3,stride=2,padding=1)
In [17]: out=layer.forward(x)
Out[18]: torch.Size([1, 3, 14, 14])

In [19]: out=layer(x) #__call__
Out[20]: torch.Size([1, 3, 14, 14])
```

Inner weight & bias

```
In [21]: layer.weight
Parameter containing:
tensor([[[[ 0.2727, -0.0923, -0.1530],
          [-0.0664,  0.2896,  0.0593],
          [-0.1967,  0.2786,  0.3163]]],


        [[[ 0.0825,  0.1090,  0.1183],
          [ 0.0857, -0.3036, -0.2539],
          [-0.3169,  0.0118, -0.2634]]],


        [[[ 0.1211,  0.1331, -0.2639],
          [ 0.3033,  0.1766, -0.0017],
          [-0.2050, -0.0187, -0.2170]]]], requires_grad=True)

In [22]: layer.weight.shape
Out[22]: torch.Size([3, 1, 3, 3])

In [23]: layer.bias.shape
Out[23]: torch.Size([3])
```

F.conv2d

```
In [24]: w=torch.rand(16,3,5,5)
In [25]: b=torch.rand(16)

In [26]: out=F.conv2d(x,w,b,stride=1,padding=1)
---------------------------------------------------------------------------
RuntimeError: Given groups=1, weight of size [16, 3, 5, 5],
expected input[1, 1, 28, 28] to have 3 channels, but got 1 channels instead

In [27]: x=torch.randn(1,3,28,28)

In [28]: out=F.conv2d(x,w,b,stride=1,padding=1)
Out[29]: torch.Size([1, 16, 26, 26])

In [30]: out=F.conv2d(x,w,b,stride=2,padding=2)
Out[31]: torch.Size([1, 16, 14, 14])
```

下一课时

池化层

Thank You.

PyTorch

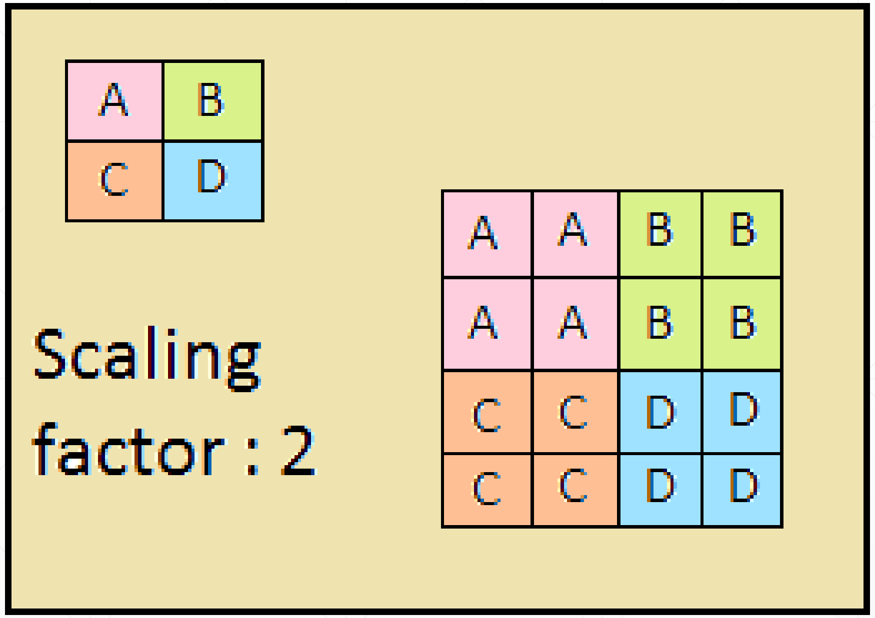
Down/up sample

主讲人：龙良曲

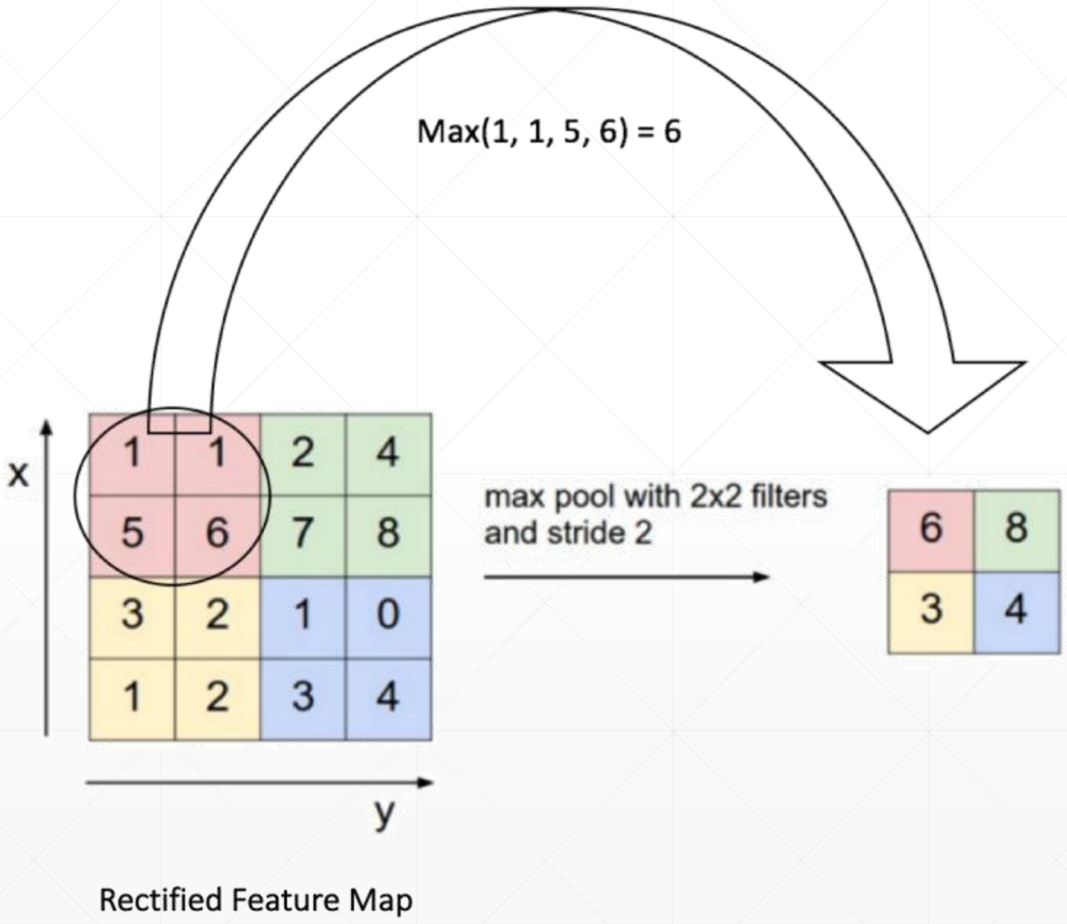
Outline

- Pooling
- upsample
- ReLU

Downsample

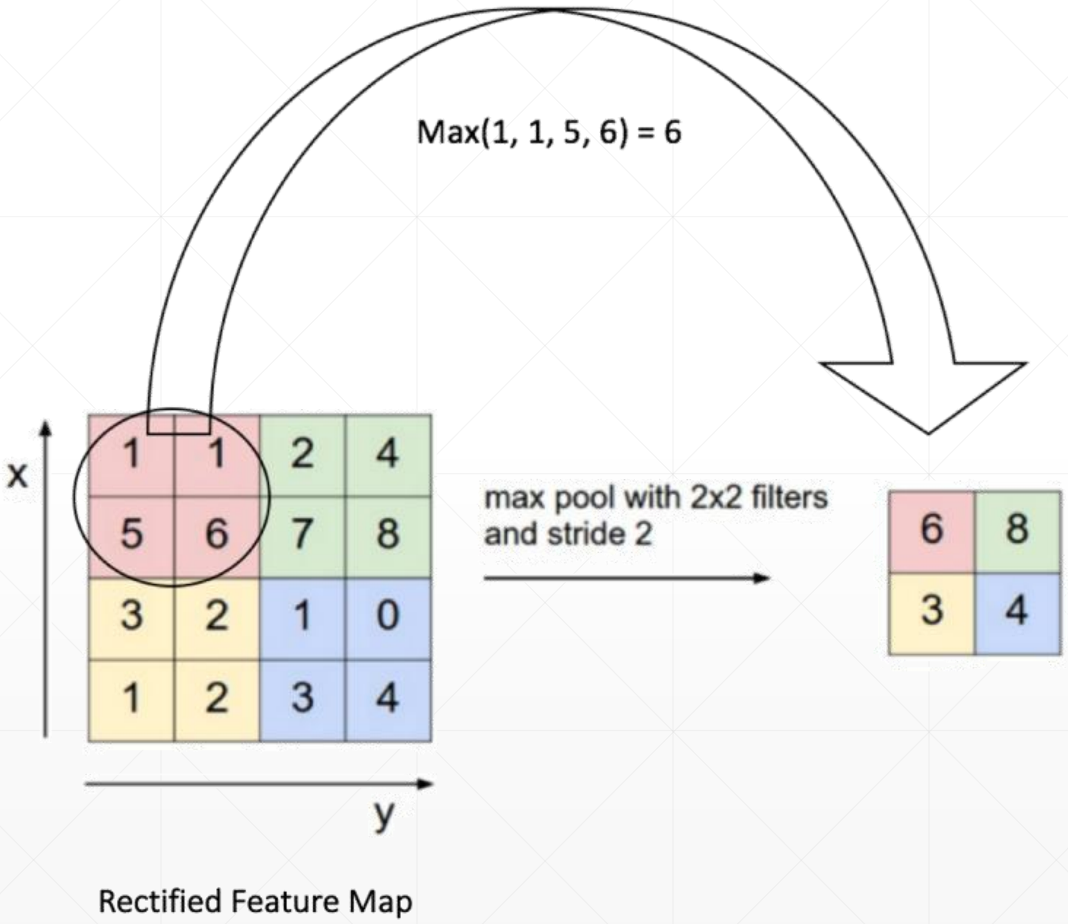


Max pooling


Max(1, 1, 5, 6) = 6
x
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
y
max pool with 2x2 filters
and stride 2
6
8
3
4
Rectified Feature Map

Avg pooling

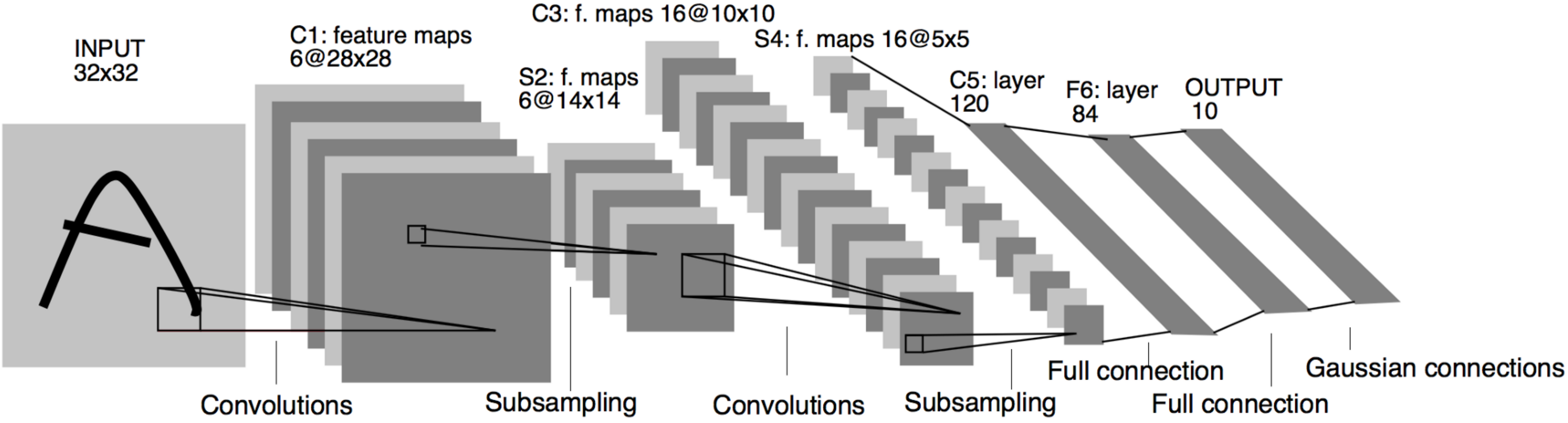
- ?



Max(1, 1, 5, 6) = 6
x
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
y
max pool with 2x2 filters
and stride 2
6
8
3
4
Rectified Feature Map

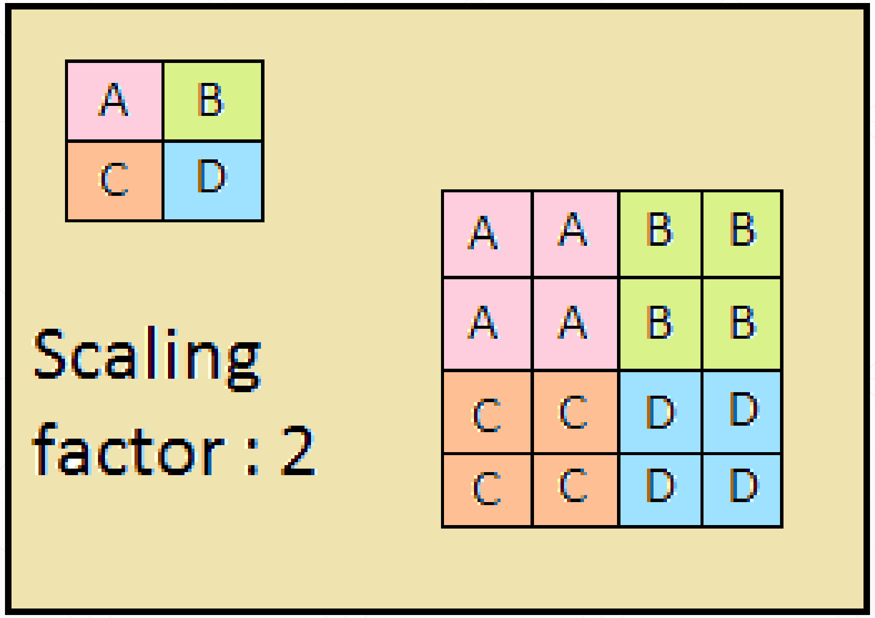
Pooling

- reduce size



```
In [33]: x=out
Out[31]: torch.Size([1, 16, 14, 14])

In [32]: layer=nn.MaxPool2d(2,stride=2)

In [34]: out=layer(x)
Out[35]: torch.Size([1, 16, 7, 7])

In [36]: out=F.avg_pool2d(x,2,stride=2)
Out[37]: torch.Size([1, 16, 7, 7])
```

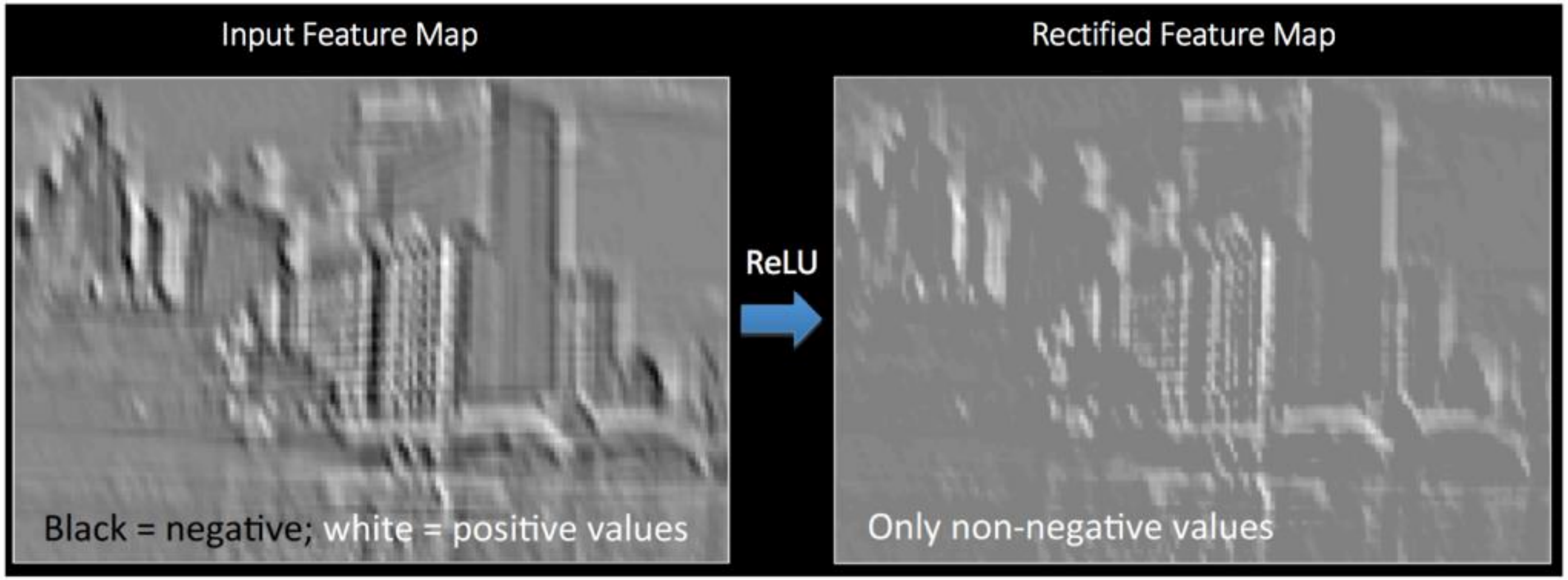
upsample



F.interpolate

```
In [38]: x=out

In [39]: out=F.interpolate(x,scale_factor=2,mode='nearest')

In [40]: out.shape
Out[40]: torch.Size([1, 16, 14, 14])

In [41]: out=F.interpolate(x,scale_factor=3,mode='nearest')

In [42]: out.shape
Out[42]: torch.Size([1, 16, 21, 21])
```

ReLU


Input Feature Map
Rectified Feature Map
ReLU
Black = negative; white = positive values
Only non-negative values

```
In [43]: x.shape
Out[43]: torch.Size([1, 16, 7, 7])

In [44]: layer=nn.ReLU(inplace=True)

In [45]: out=layer(x)
In [46]: out.shape
Out[46]: torch.Size([1, 16, 7, 7])

In [47]: out=F.relu(x)
In [48]: out.shape
Out[48]: torch.Size([1, 16, 7, 7])
```

下一课时

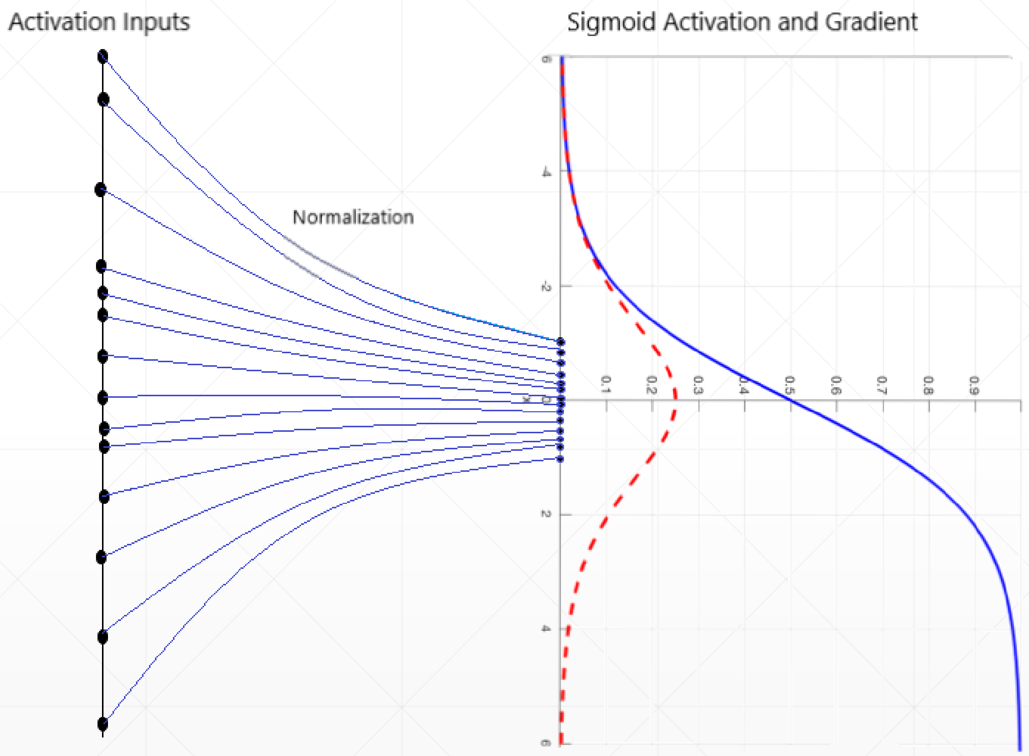
Batch-Norm

Thank You.

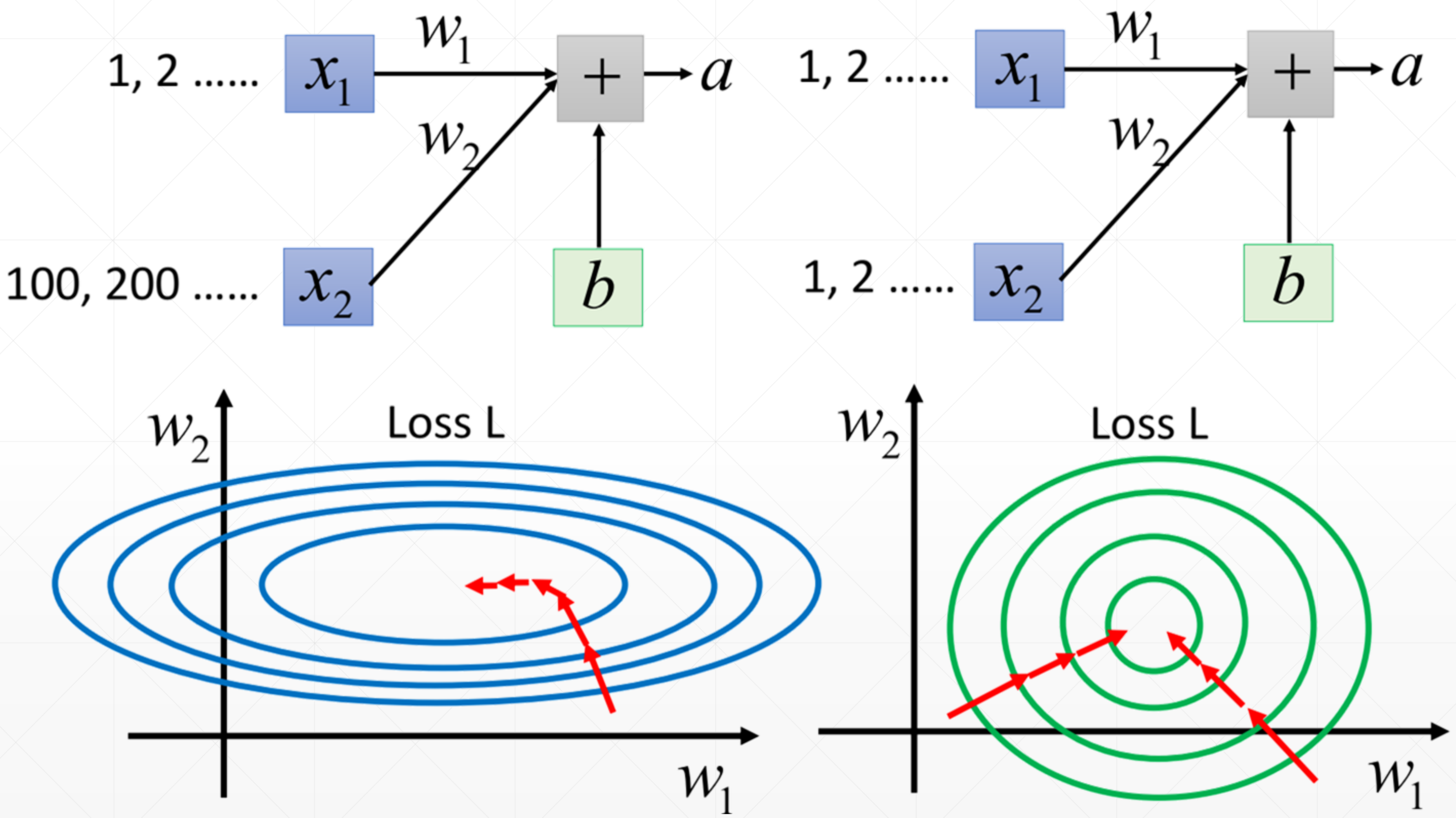
Batch Norm

主讲人：龙良曲

Intuitive explanation



Intuitive explanation



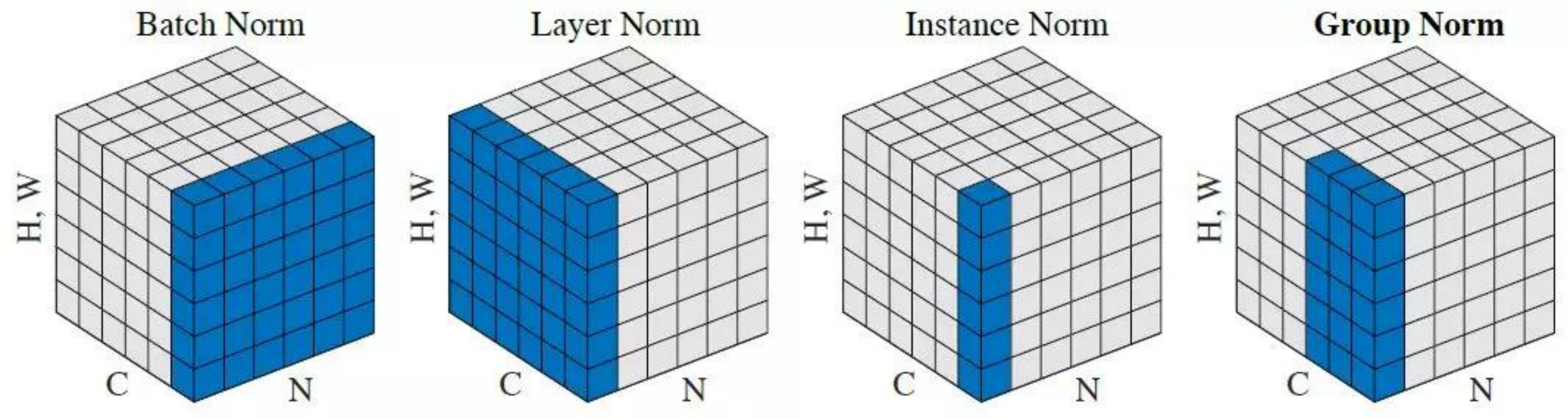
Feature scaling

- Image Normalization

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
```

- Batch Normalization

Batch Norm

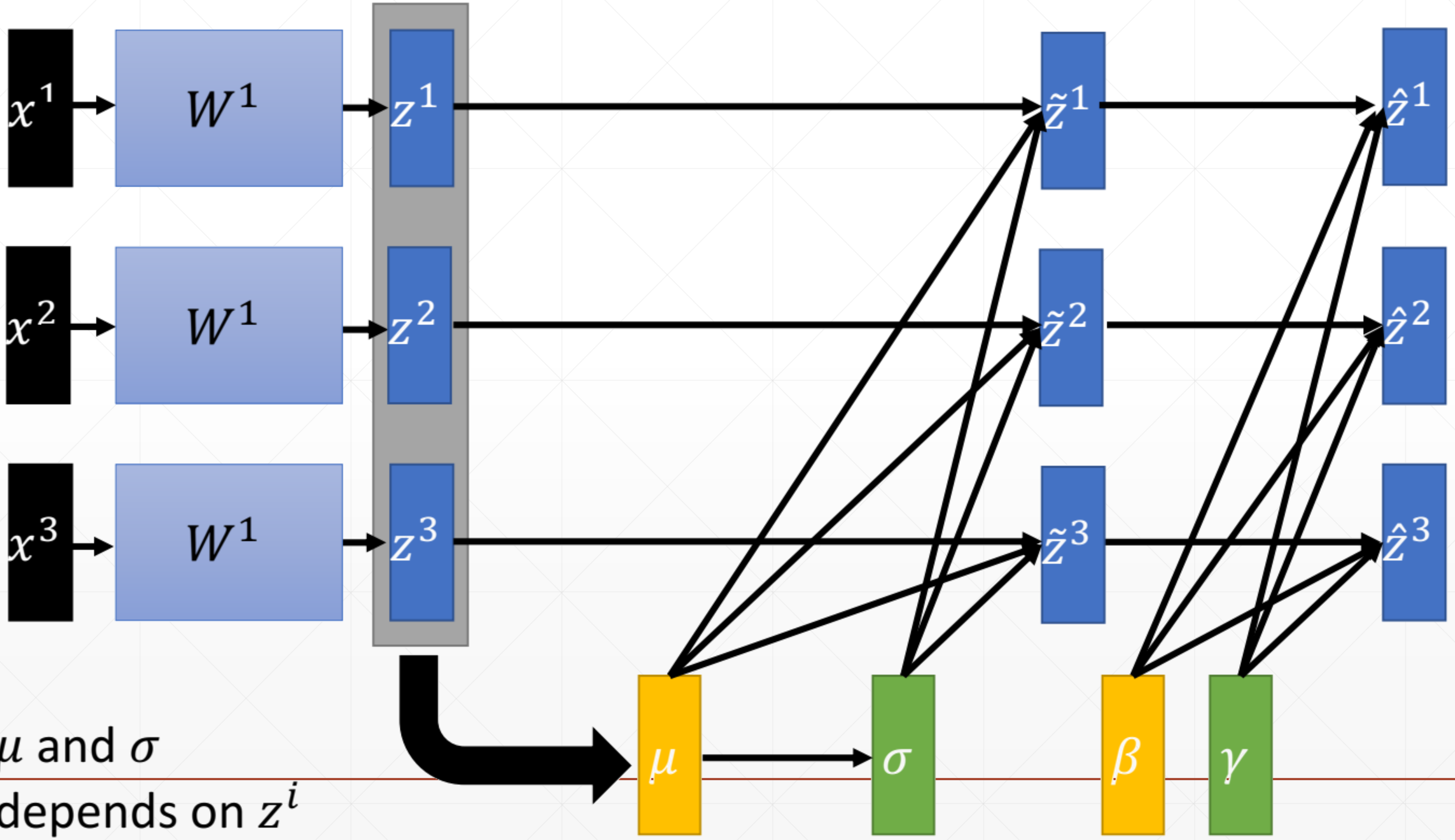


https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Batch normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$



μ and σ depends on z^i

```
In [9]: x=torch.randn(100,16)+0.5
In [10]: layer=torch.nn.BatchNorm1d(16)

In [11]: layer.running_mean,layer.running_var
(tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]))

In [12]: out=layer(x)

In [13]: layer.running_mean,layer.running_var
(tensor([0.0625, 0.0752, 0.0589, 0.0358, 0.0662, 0.0651, 0.0572, 0.0634, 0.0520,
         0.0372, 0.0370, 0.0258, 0.0534, 0.0517, 0.0623, 0.0604]),
 tensor([0.9902, 1.0067, 0.9998, 0.9992, 1.0341, 1.0391, 1.0275, 0.9950, 0.9716,
         0.9829, 0.9820, 1.0044, 0.9921, 0.9941, 0.9935, 1.0153]))
```

```
In [9]: x=torch.randn(100,16)+0.5
In [10]: layer=torch.nn.BatchNorm1d(16)

In [14]: for i in range(100):out=layer(x)

In [15]: layer.running_mean,layer.running_var
Out[15]:
(tensor([0.6253, 0.7521, 0.5887, 0.3578, 0.6616, 0.6512, 0.5725, 0.6337, 0.5202,
         0.3716, 0.3700, 0.2583, 0.5344, 0.5170, 0.6233, 0.6037]),
 tensor([0.9020, 1.0671, 0.9984, 0.9922, 1.3413, 1.3906, 1.2755, 0.9501, 0.7161,
         0.8291, 0.8200, 1.0438, 0.9212, 0.9407, 0.9348, 1.1531]))
```

Pipeline

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

nn.BatchNorm2d

```
In [49]: x.shape
Out[49]: torch.Size([1, 16, 7, 7])

In [50]: layer=nn.BatchNorm2d(16)

In [51]: out=layer(x)
Out[52]: torch.Size([1, 16, 7, 7])

In [53]: layer.weight
Parameter containing:
tensor([0.3119, 0.6959, 0.9881, 0.0130, 0.1879, 0.5179, 0.0464, 0.7868, 0.8371,
        0.4370, 0.9743, 0.7311, 0.5124, 0.5352, 0.5410, 0.1771],
       requires_grad=True)

In [54]: layer.weight.shape
Out[54]: torch.Size([16])
In [55]: layer.bias.shape
Out[55]: torch.Size([16])
```

Class variables

```
In [56]: vars(layer)
 '_buffers': OrderedDict([('running_mean',
               tensor([0.2415, 0.2258, 0.1760, 0.2031, 0.1910, 0.2147, 0.1964, 0.2068, 0.1660,
                       0.2114, 0.2340, 0.1923, 0.2010, 0.1870, 0.1921, 0.1581])),
              ('running_var',
               tensor([1.6709, 1.5211, 1.4196, 1.6144, 1.5087, 1.4599, 1.3999, 1.5254, 1.3087,
                       1.4290, 1.6022, 1.3855, 1.5442, 1.5265, 1.4686, 1.2741])),
              ('num_batches_tracked', tensor(1))]),
 '_modules': OrderedDict(),
 '_parameters': OrderedDict([('weight', Parameter containing:
               tensor([0.3119, 0.6959, 0.9881, 0.0130, 0.1879, 0.5179, 0.0464, 0.7868, 0.8371,
                       0.4370, 0.9743, 0.7311, 0.5124, 0.5352, 0.5410, 0.1771],
                      requires_grad=True)), ('bias', Parameter containing:
               tensor([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
                      requires_grad=True))]),
 '_state_dict_hooks': OrderedDict(),
 'affine': True,
 'eps': 1e-05,
 'momentum': 0.1,
 'num_features': 16,
 'track_running_stats': True,
 'training': True}
```

Test

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

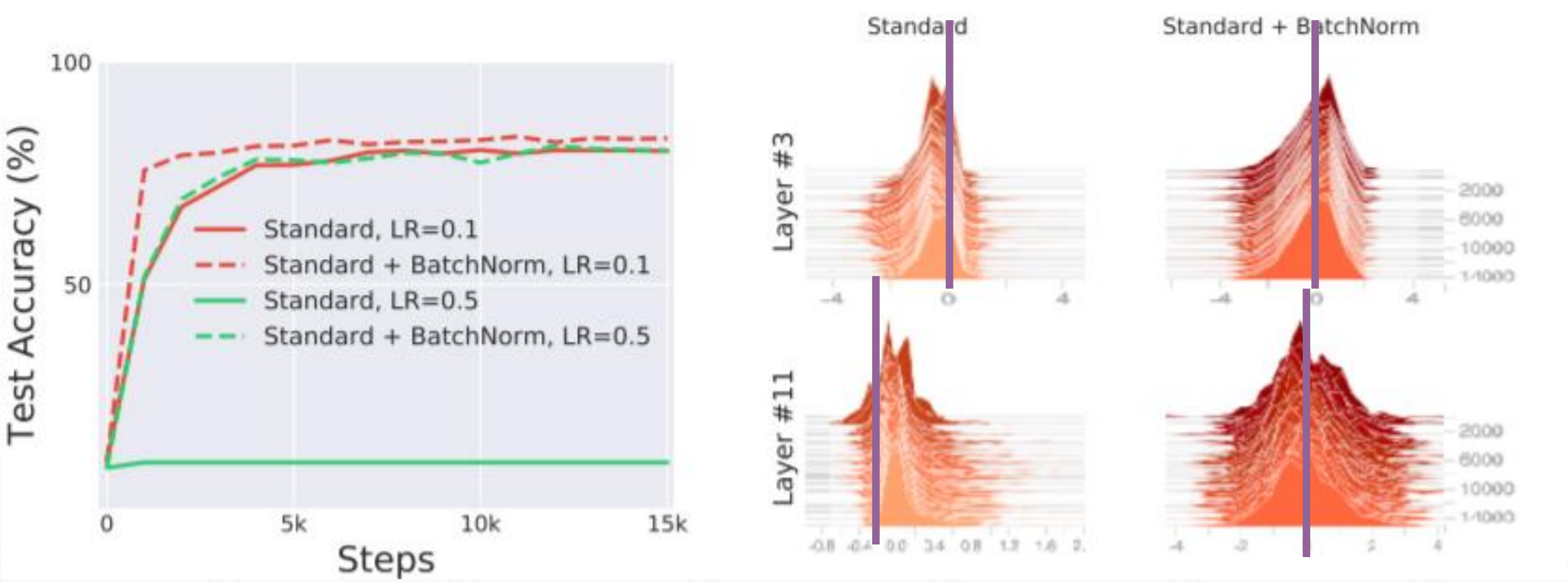
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

```
In [67]: layer.eval()
Out[67]: BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

In [68]: out=layer(x)
```

Visualization



Advantages

- Converge faster
- Better performance
- Robust
  - stable
  - larger learning rate

下一课时

经典CNN

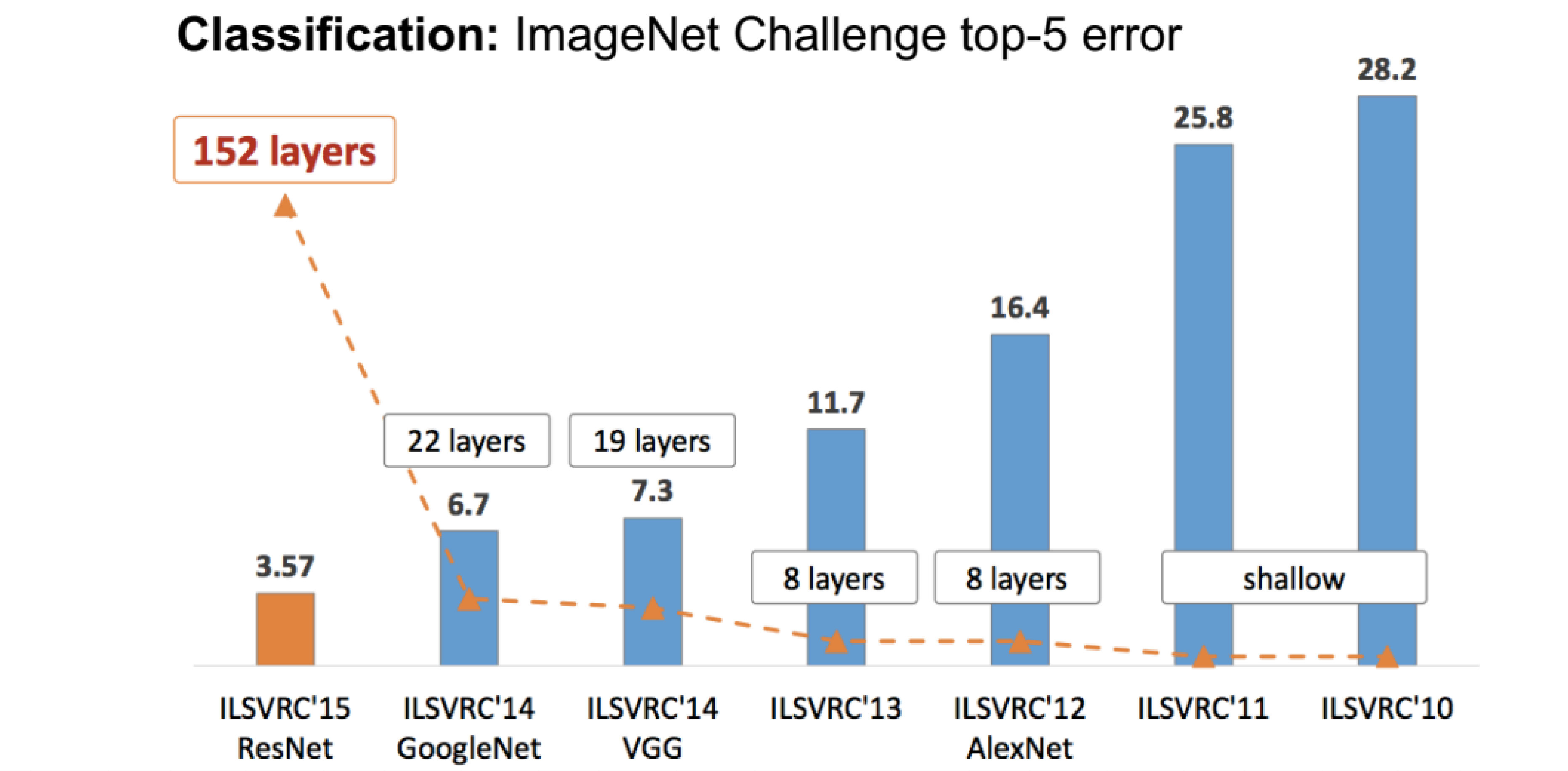
Thank You.

PyTorch

经典卷积网络

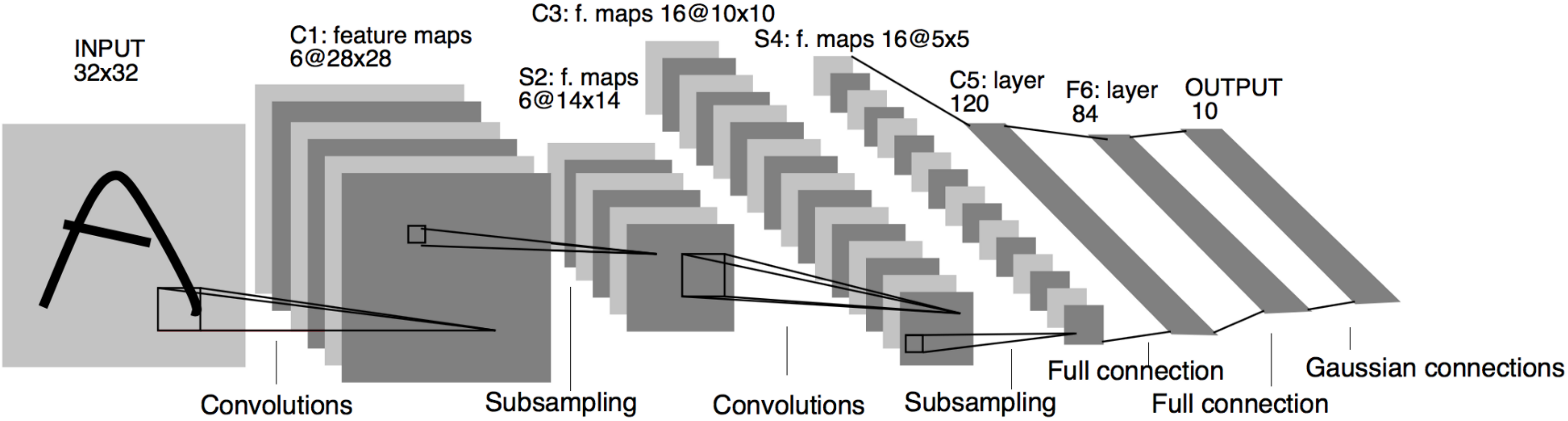
主讲人：龙良曲

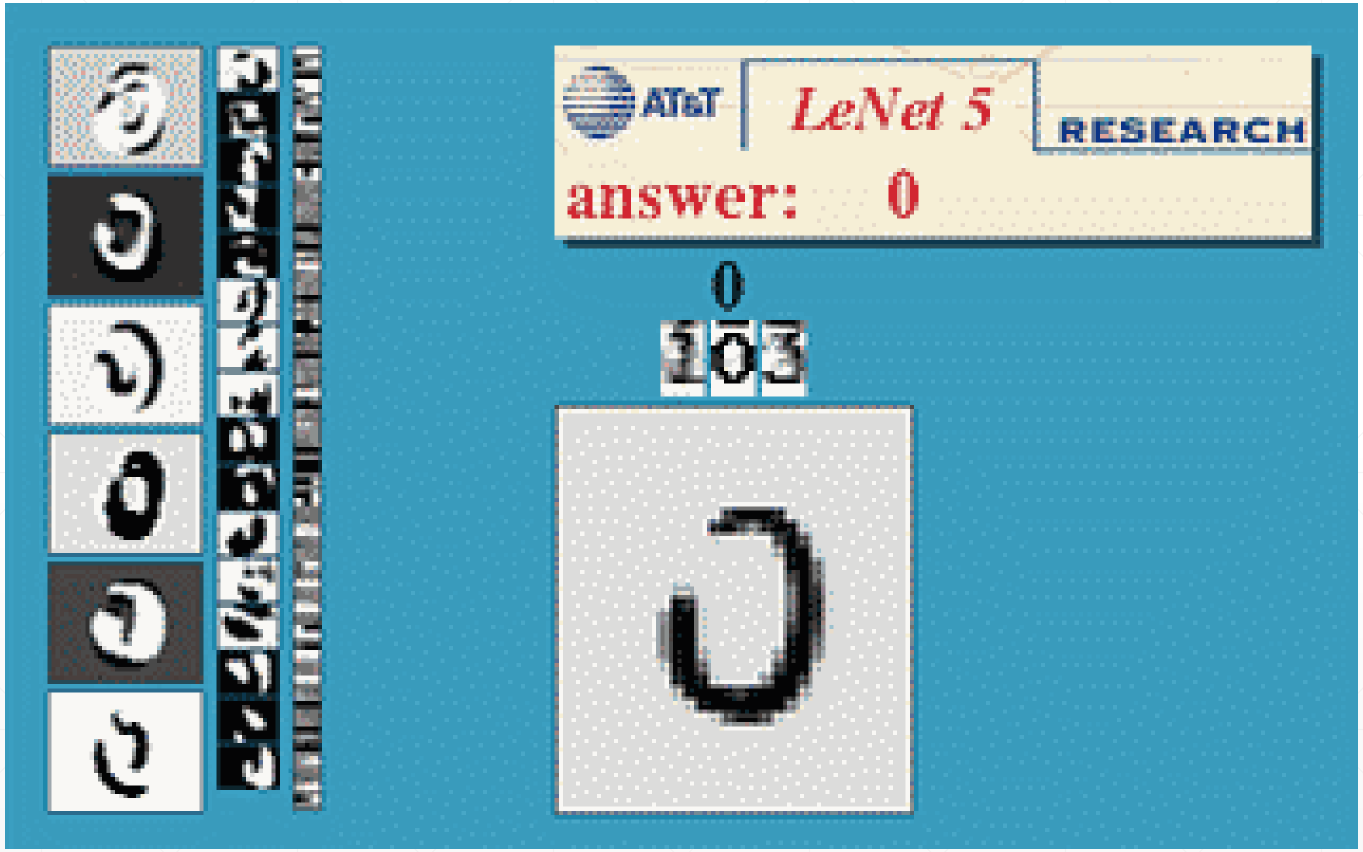
ImageNet


Classification: ImageNet Challenge top-5 error
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

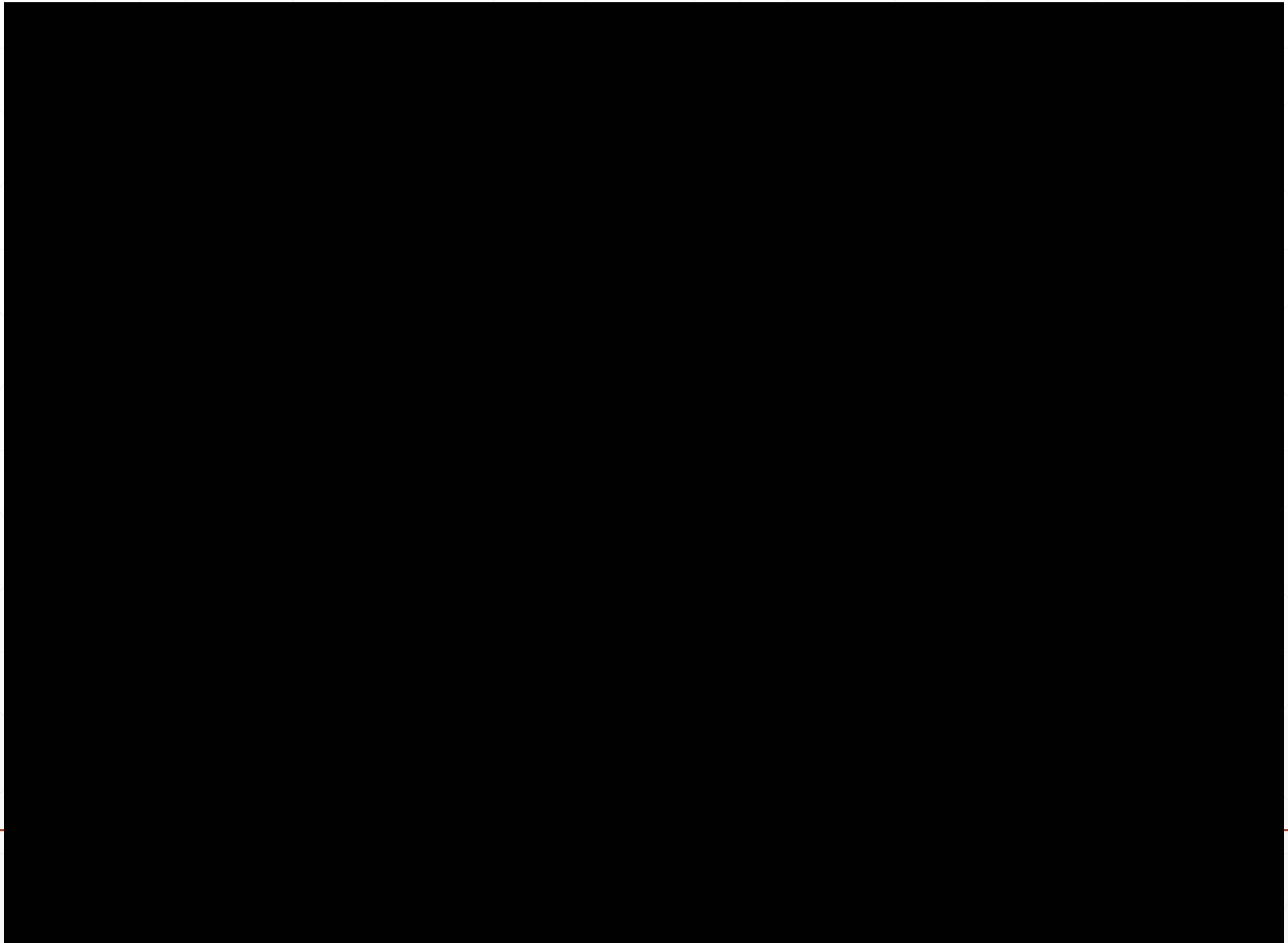
LeNet-5

- 99.2% acc.
- 5/6 layers




AT&T
LeNet 5
RESEARCH
answer: 0
0
103

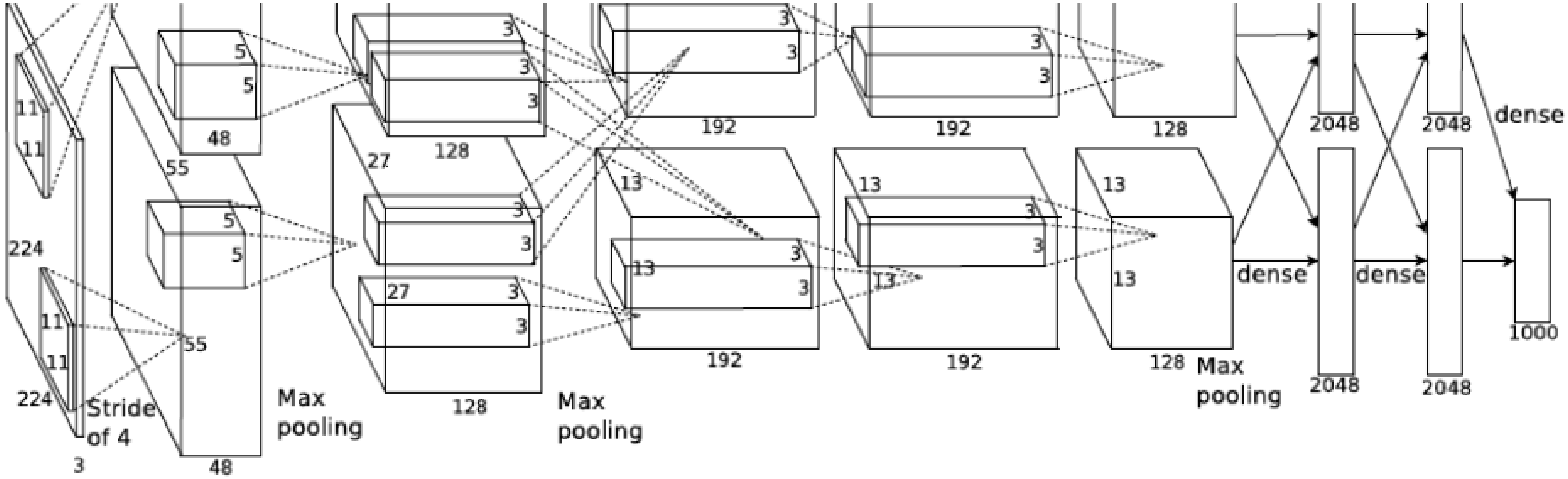
LeNet5 Demo



AlexNet

- GTX 580
  - 3GBx2
- 11x11
- 8 layers

AlexNet: ILSVRC 2012 winner



- Similar framework to LeNet but:
  - Max pooling, ReLU nonlinearity
  - More data and bigger model (7 hidden layers, 650K units, 60M params)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

VGG

- 3x3
- 1x1
- 11-19 layer

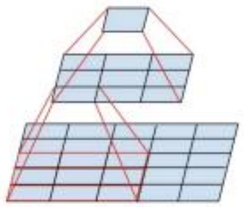
VGGNet: ILSVRC 2014 2^{nd} place

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

- Sequence of deeper networks trained progressively
- Large receptive fields replaced by successive layers of 3x3 convolutions (with ReLU in between)

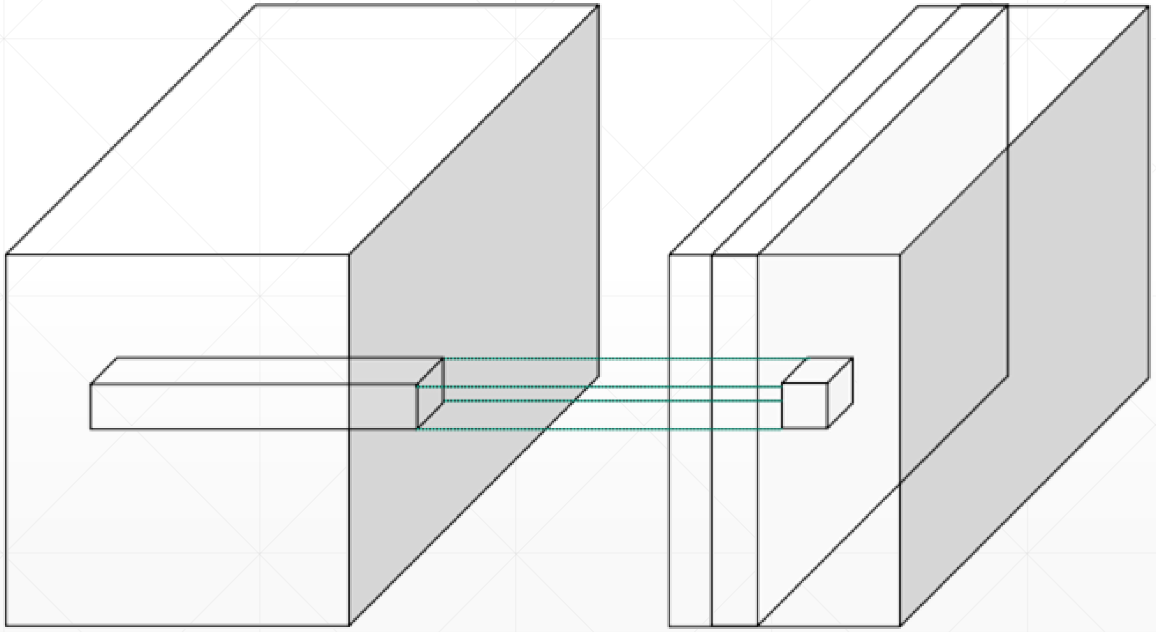


- One 7x7 conv layer with C feature maps needs $49C^2$ weights, three 3x3 conv layers need only $27C^2$ weights
- Experimented with 1x1 convolutions

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

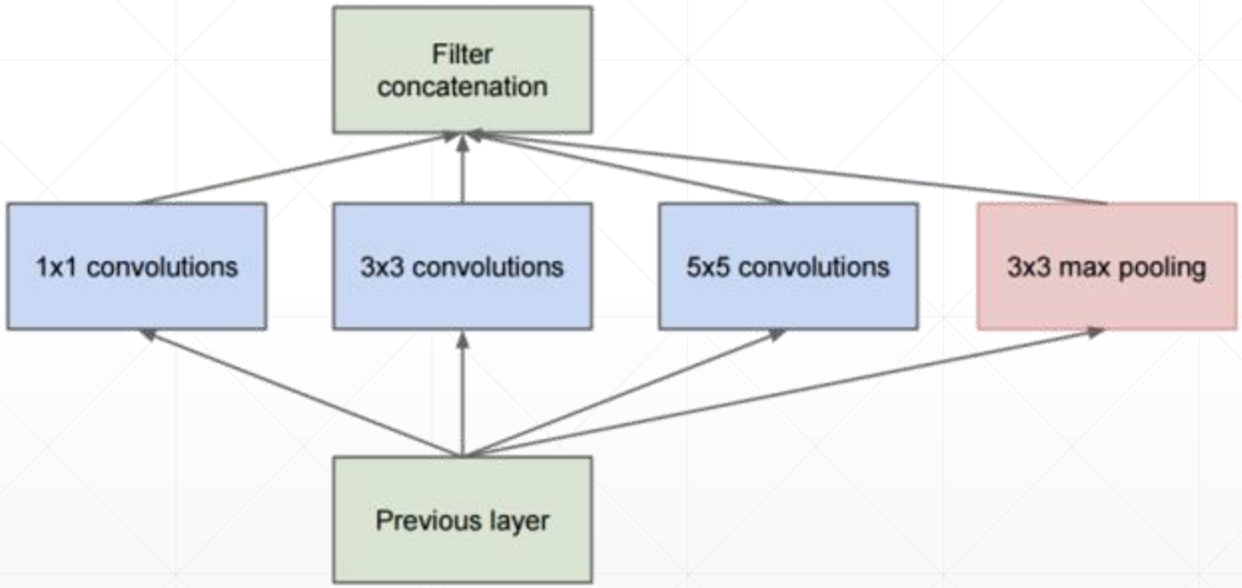
1x1 Convolution

- less computation
- c_in => c_out

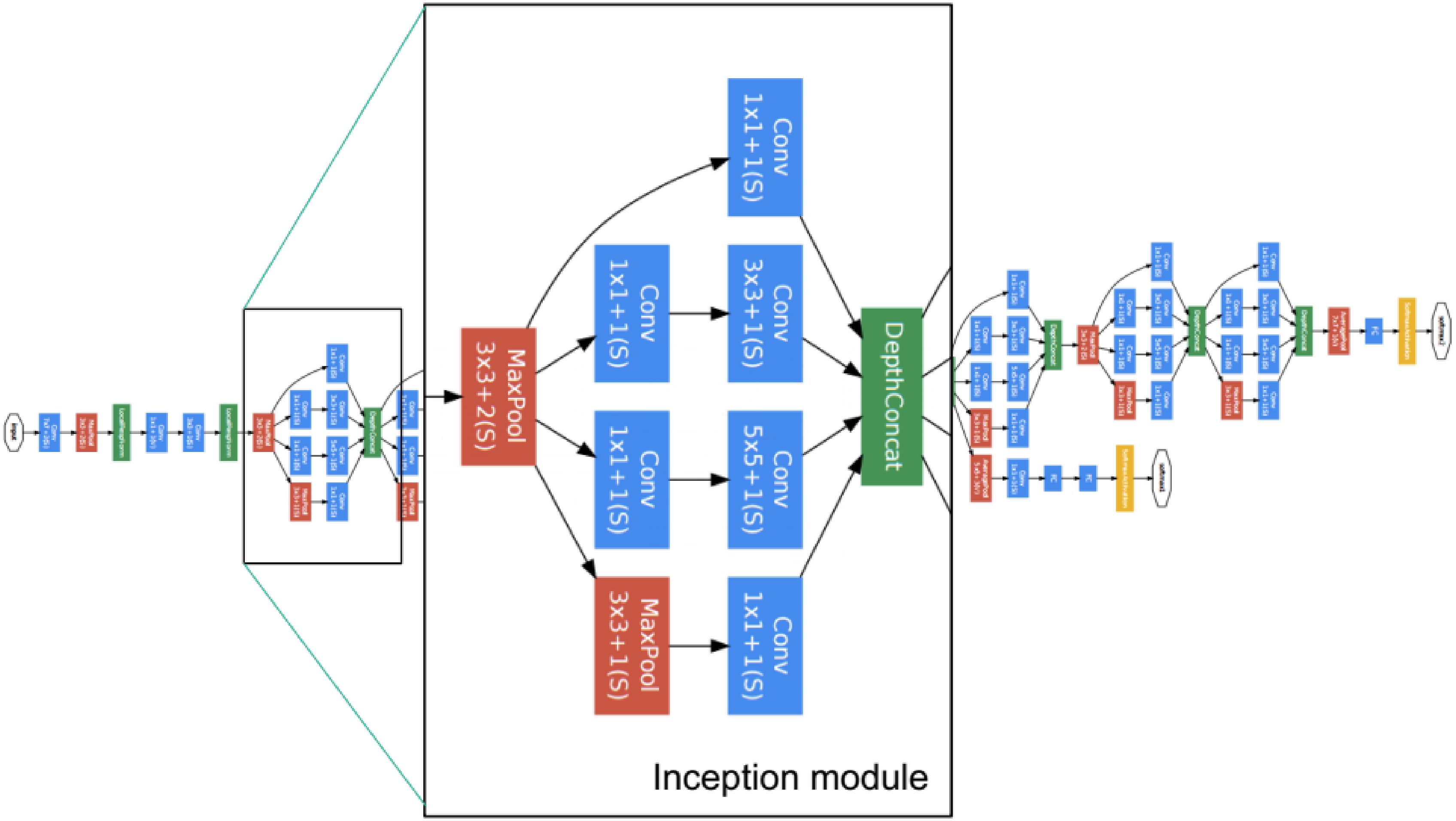


GoogLeNet

- 1st in 2014 ILSVRC
- 22 layers



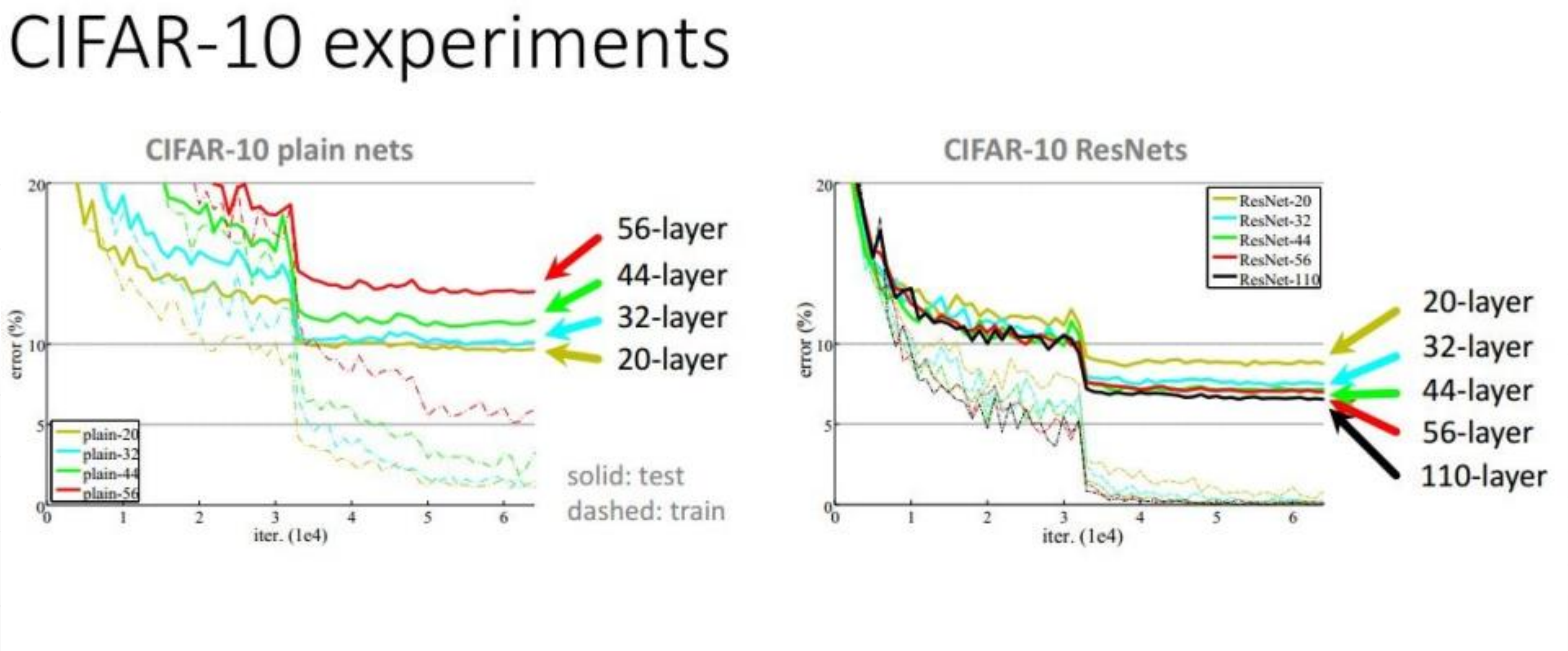
GoogLeNet



C. Szegedy et al., [Going deeper with convolutions](), CVPR 2015

Stack more layers?

- 1000 layers?



下一课时

ResNet

Thank You.

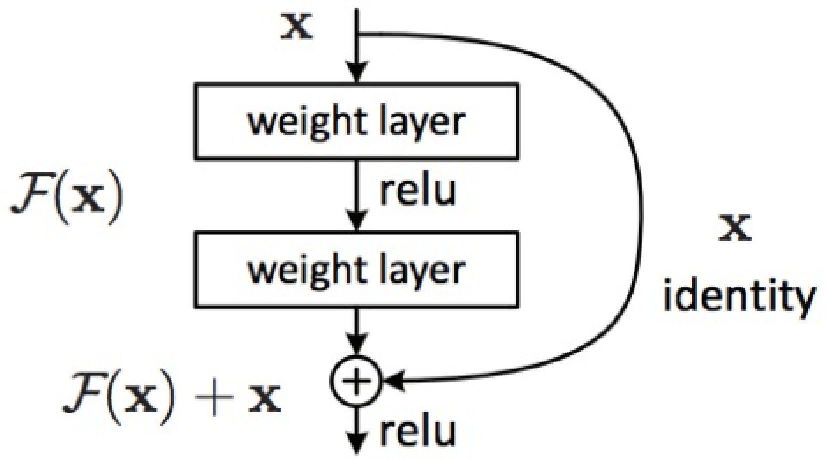
PyTorch

深度残差网络

主讲人：龙良曲

ResNet

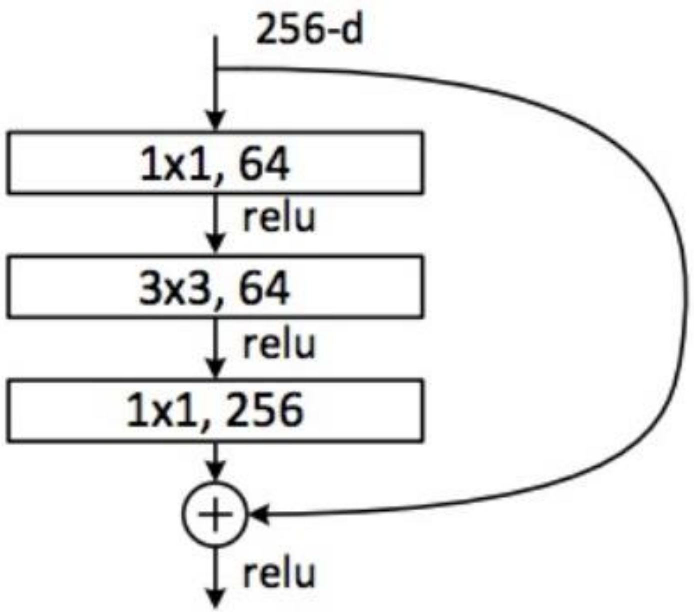
- The residual module
  - Introduce *skip* or *shortcut* connections (existing before in various forms in literature)
  - Make it easy for network layers to represent the identity mapping
  - For some reason, need to skip at least two layers



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
[Deep Residual Learning for Image Recognition](), CVPR 2016 (Best Paper)

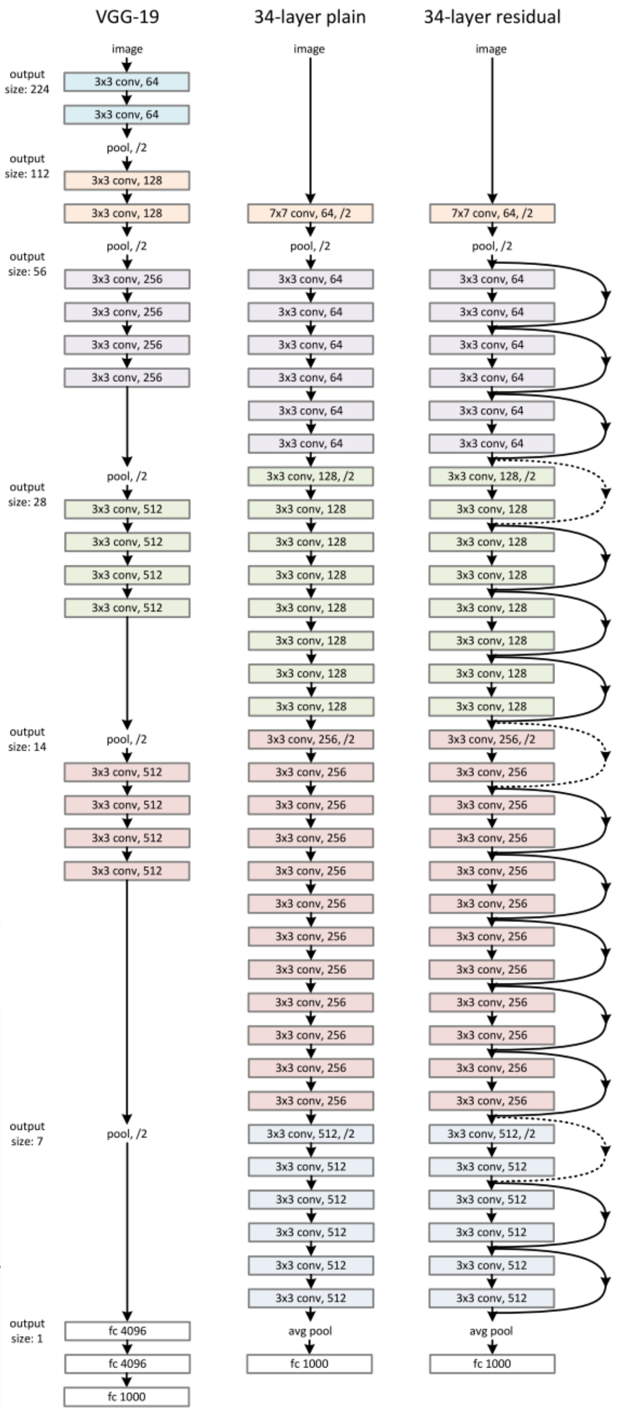
ResNet

Deeper residual module (bottleneck)



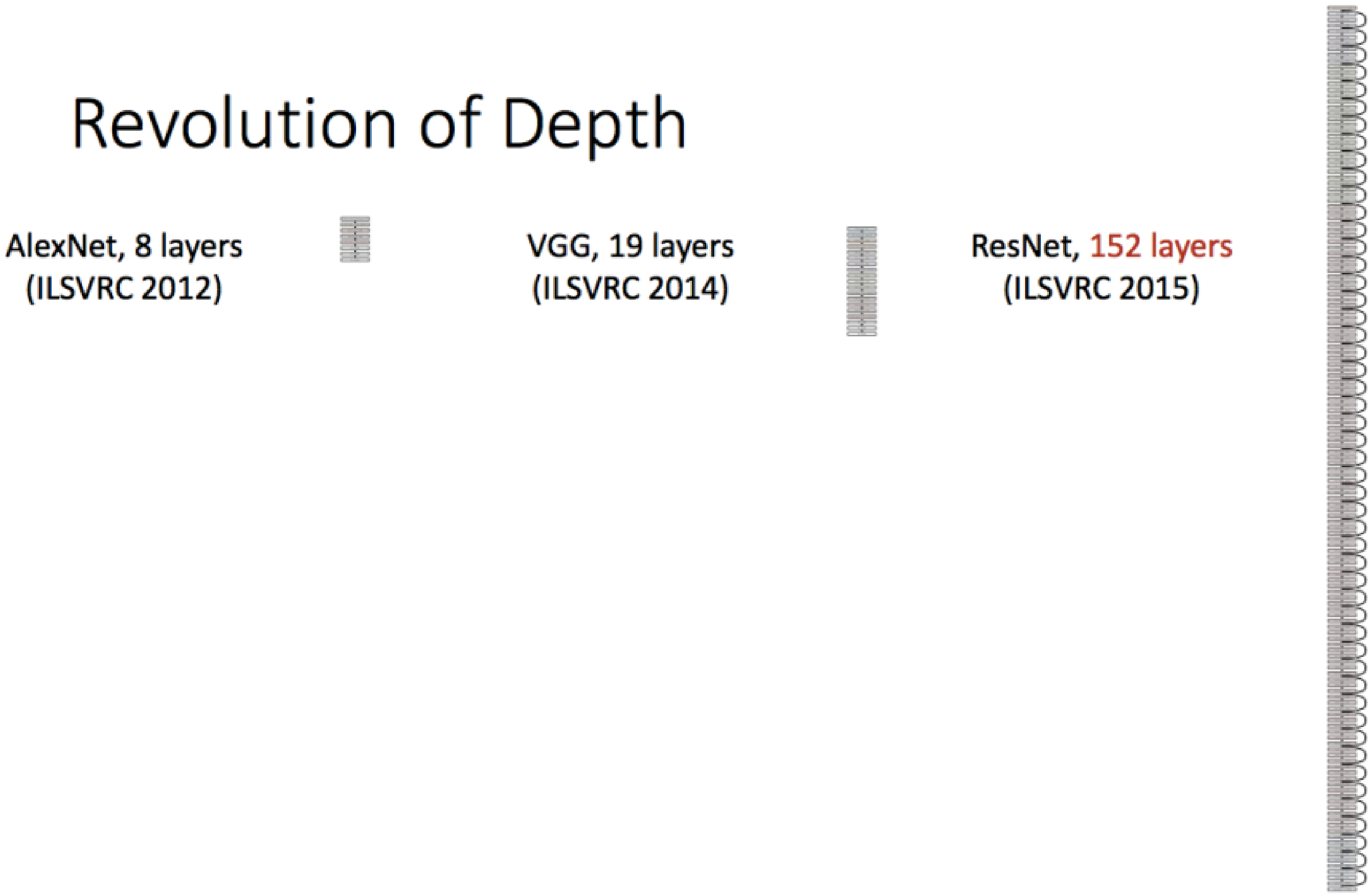
- Directly performing 3x3 convolutions with 256 feature maps at input and output:
  256 x 256 x 3 x 3 ~ 600K
  operations
- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  256 x 64 x 1 x 1 ~ 16K
  64 x 64 x 3 x 3 ~ 36K
  64 x 256 x 1 x 1 ~ 16K
  Total: ~70K

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)


VGG-19
34-layer plain
34-layer residual
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000


I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com

ResNet: ILSVRC 2015 winner



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016

BOOM!

Microsoft Research

MSRA @ ILSVRC & COCO 2015 Competitions

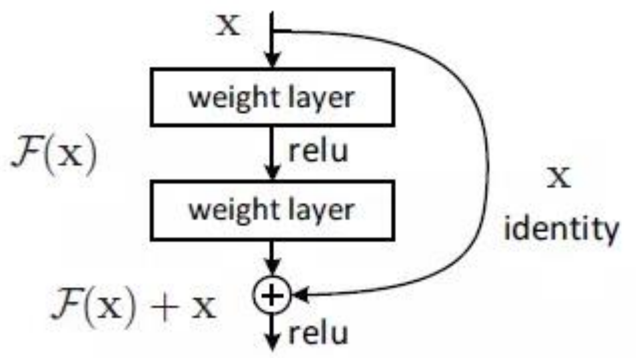
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

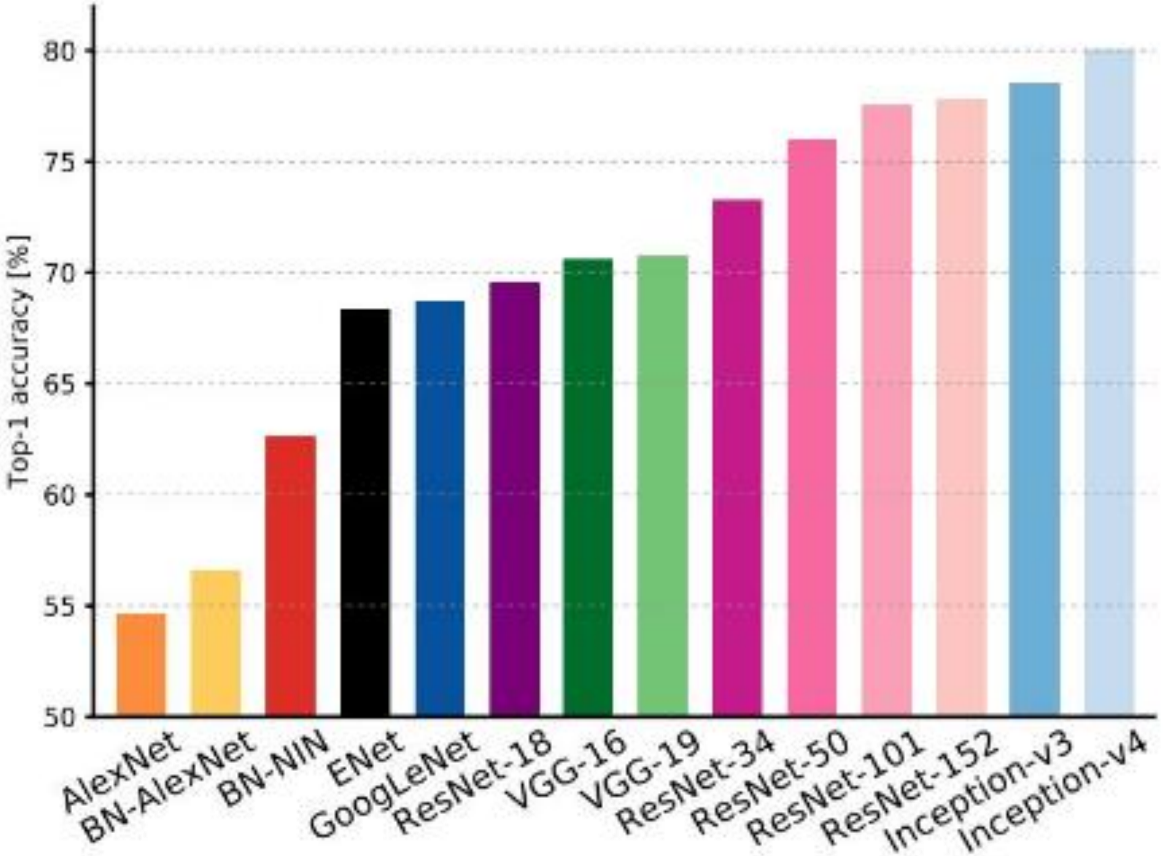
ICCV15

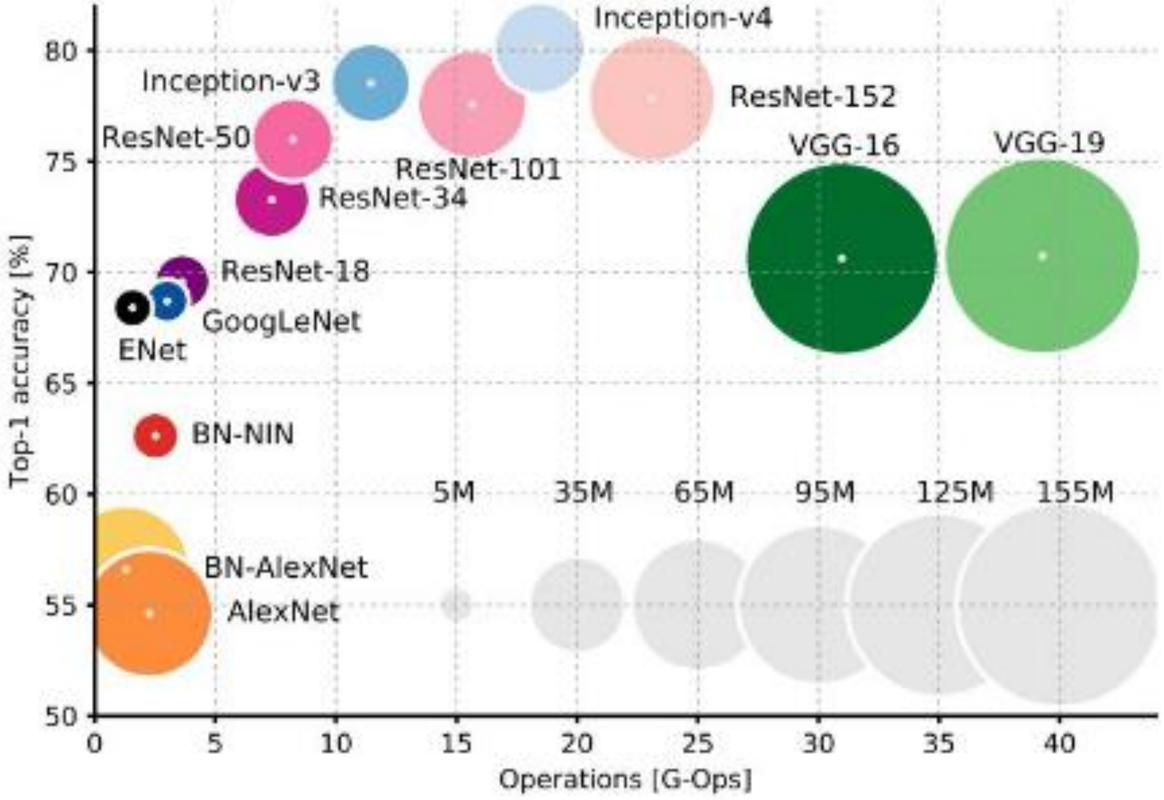
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Why call Residual?

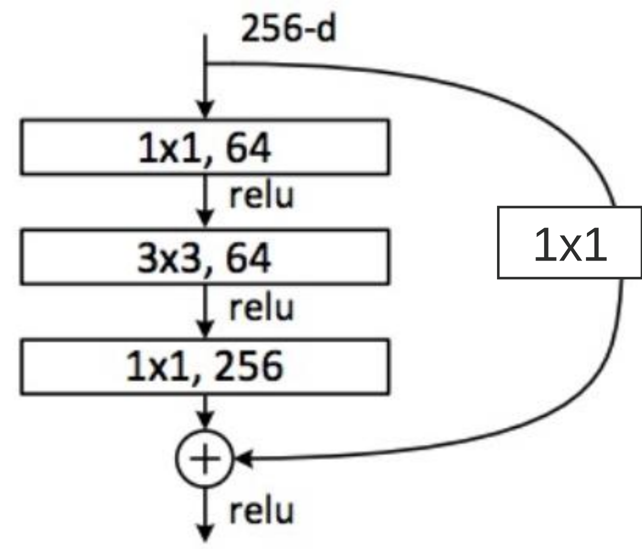


$$\mathcal{F}(\mathrm{x}) := \mathcal{H}(\mathrm{x}) - \mathrm{x}$$

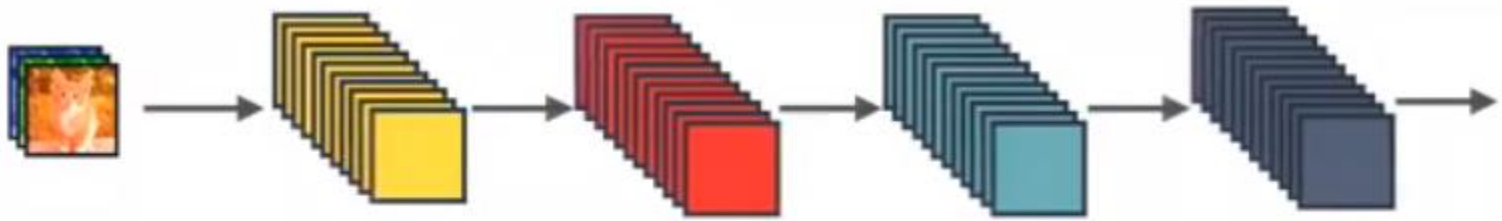

Top-1 accuracy [%]
80
75
70
65
60
55
50
AlexNet
BN-AlexNet
BN-NIN
ENet
GoogLeNet
ResNet-18
VGG-16
VGG-19
ResNet-34
ResNet-50
ResNet-101
ResNet-152
Inception-v3
Inception-v4

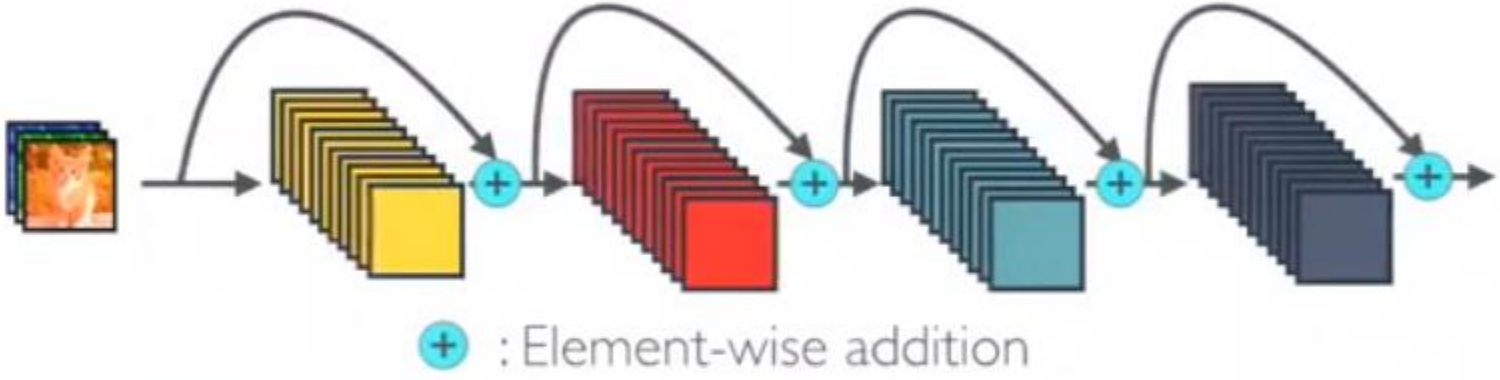

Top-1 accuracy [%]
Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
VGG-16
VGG-19
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
5M
35M
65M
95M
125M
155M
BN-AlexNet
AlexNet
Operations [G-Ops]
0
5
10
15
20
25
30
35
40

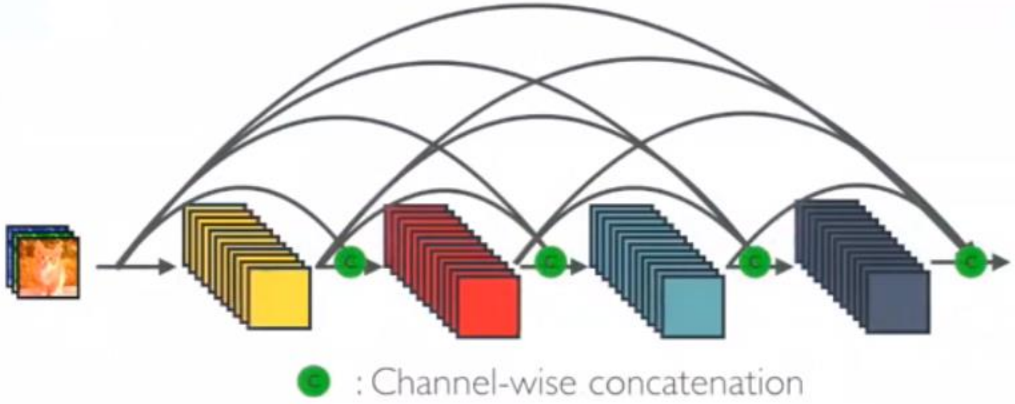
```python
class ResBlk(nn.Module):
    def __init__(self, ch_in, ch_out):
        self.conv1 = nn.Conv2d(ch_in, ch_out, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(ch_out)
        self.conv2 = nn.Conv2d(ch_out, ch_out, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(ch_out)

        self.extra = nn.Sequential()
        if ch_out != ch_in:
            # [b, ch_in, h, w] => [b, ch_out, h, w]
            self.extra = nn.Sequential(
                nn.Conv2d(ch_in, ch_out, kernel_size=1, stride=1),
                nn.BatchNorm2d(ch_out)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out = self.extra(x) + out
        return out
```



DenseNet







下一课时

nn.Module

Thank You.

PyTorch

nn.Module

主讲人：龙良曲

```python
class MyLinear(nn.Module):

    def __init__(self, inp, outp):
        super(MyLinear, self).__init__()

        # requires_grad = True
        self.w = nn.Parameter(torch.randn(outp, inp))
        self.b = nn.Parameter(torch.randn(outp))

    def forward(self, x):
        x = x @ self.w.t() + self.b
        return x
```

Magic

- Every Layer is nn.Module
  - nn.Linear
  - nn.BatchNorm2d
  - nn.Conv2d
- nn.Module nested in nn.Module

1. embed current layers

- Linear
- ReLU
- Sigmoid
- Conv2d
- ConvTransposed2d
- Dropout
- etc.

2. Container

- net(x)

```
self.net = nn.Sequential(
    nn.Conv2d(1, 32, 5, 1, 1),
    nn.MaxPool2d(2, 2),
    nn.ReLU(True),
    nn.BatchNorm2d(32),

    nn.Conv2d(32, 64, 3,  1, 1),
    nn.ReLU(True),
    nn.BatchNorm2d(64),

    nn.Conv2d(64, 64, 3,  1, 1),
    nn.MaxPool2d(2, 2),
    nn.ReLU(True),
    nn.BatchNorm2d(64),

    nn.Conv2d(64, 128, 3, 1, 1),
    nn.ReLU(True),
    nn.BatchNorm2d(128)
)
```

3. parameters

```
In [80]: net=nn.Sequential(nn.Linear(4,2),nn.Linear(2,2))
In [81]: list(net.parameters())[0].shape
Out[81]: torch.Size([2, 4])
In [82]: list(net.parameters())[3].shape
Out[82]: torch.Size([2])
In [83]: list(net.named_parameters())[0]
('0.weight', Parameter containing:
 tensor([[ 0.3389, -0.0053, -0.0499, -0.0407],
         [ 0.4691, -0.0466, -0.4306,  0.3315]], requires_grad=True))
In [84]: list(net.named_parameters())[1]
('0.bias', Parameter containing:
 tensor([0.0780, 0.1454], requires_grad=True))

In [87]: dict(net.named_parameters()).items()
dict_items([('0.weight', Parameter containing:
tensor([[ 0.3389, -0.0053, -0.0499, -0.0407],
        [ 0.4691, -0.0466, -0.4306,  0.3315]], requires_grad=True)),
           ('0.bias', Parameter containing:
tensor([0.0780, 0.1454], requires_grad=True)),
            ('1.weight', Parameter containing:
tensor([[ 0.0924, -0.2787],
        [-0.4831, -0.3320]], requires_grad=True)),
            ('1.bias', Parameter containing:
tensor([-0.2160,  0.0170], requires_grad=True))])

In [90]: optimizer=optim.SGD(net.parameters(),lr=1e-3)
```

4. modules

- modules: all nodes
- children: direct children

```python
class BasicNet(nn.Module):
    def __init__(self):
        super(BasicNet, self).__init__()
        self.net = nn.Linear(4, 3)

    def forward(self, x):
        return self.net(x)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.net = nn.Sequential(BasicNet(),
                                 nn.ReLU(),
                                 nn.Linear(3, 2))

    def forward(self, x):
        return self.net(x)
```

```
parameters: net.0.net.weight torch.Size([3, 4])
parameters: net.0.net.bias torch.Size([3])
parameters: net.2.weight torch.Size([2, 3])
parameters: net.2.bias torch.Size([2])

children: net Sequential(
  (0): BasicNet(
    (net): Linear(in_features=4, out_features=3, bias=True)
  )
  (1): ReLU()
  (2): Linear(in_features=3, out_features=2, bias=True)
)
```

```
modules:  Net(
  (net): Sequential(
    (0): BasicNet(
      (net): Linear(in_features=4, out_features=3, bias=True)
    )
    (1): ReLU()
    (2): Linear(in_features=3, out_features=2, bias=True)
  )
)
modules: net Sequential(
  (0): BasicNet(
    (net): Linear(in_features=4, out_features=3, bias=True)
  )
  (1): ReLU()
  (2): Linear(in_features=3, out_features=2, bias=True)
)
modules: net.0 BasicNet(
  (net): Linear(in_features=4, out_features=3, bias=True)
)
modules: net.0.net Linear(in_features=4, out_features=3, bias=True)
modules: net.1 ReLU()
modules: net.2 Linear(in_features=3, out_features=2, bias=True)
```

5. to(device)

```
device = torch.device('cuda')
net = Net()
net.to(device)
```

6. save and load

```
device = torch.device('cuda')
net = Net()
net.to(device)

net.load_state_dict(torch.load('ckpt.mdl'))

# train...

torch.save(net.state_dict(), 'ckpt.mdl')
```

7. train/test

```
device = torch.device('cuda')
net = Net()
net.to(device)

# train
net.train()
...

# test
net.eval()
...
```

8. implement own layer

```python
class Flatten(nn.Module):
    def __init__(self):
        super(Flatten, self).__init__()

    def forward(self, input):
        return input.view(input.size(0), -1)

class TestNet(nn.Module):

    def __init__(self):
        super(TestNet, self).__init__()
        self.net = nn.Sequential(nn.Conv2d(1, 16, stride=1, padding=1),
                                 nn.MaxPool2d(2, 2),
                                 Flatten(),
                                 nn.Linear(1*14*14, 10))


    def forward(self, x):
        return self.net(x)
```

8. own linear layer

```
class MyLinear(nn.Module):

    def __init__(self, inp, outp):
        super(MyLinear, self).__init__()

        # requires_grad = True
        self.w = nn.Parameter(torch.randn(outp, inp))
        self.b = nn.Parameter(torch.randn(outp))

    def forward(self, x):
        x = x @ self.w.t() + self.b
        return x
```

下一课时

Data Argumentation

Thank You.

PyTorch

数据增强

主讲人：龙良曲

Big Data

- The key to prevent Overfitting

Sample more data?

Limited Data

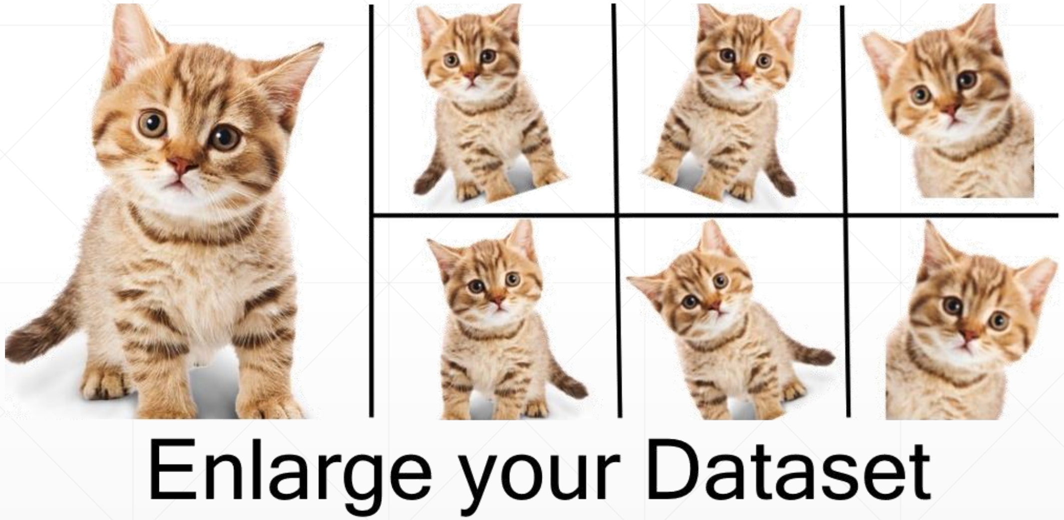
- Small network capacity
- Regularization
- Data argumentation

Recap

```python
cifar_train = datasets.CIFAR10('cifar', True, transform=transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor()
]), download=True)
cifar_train = DataLoader(cifar_train, batch_size=batchsz, shuffle=True)

cifar_test = datasets.CIFAR10('cifar', False, transform=transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor()
]), download=True)
```

Data argumentation

- Flip
- Rotate
- Random Move & Crop
- GAN

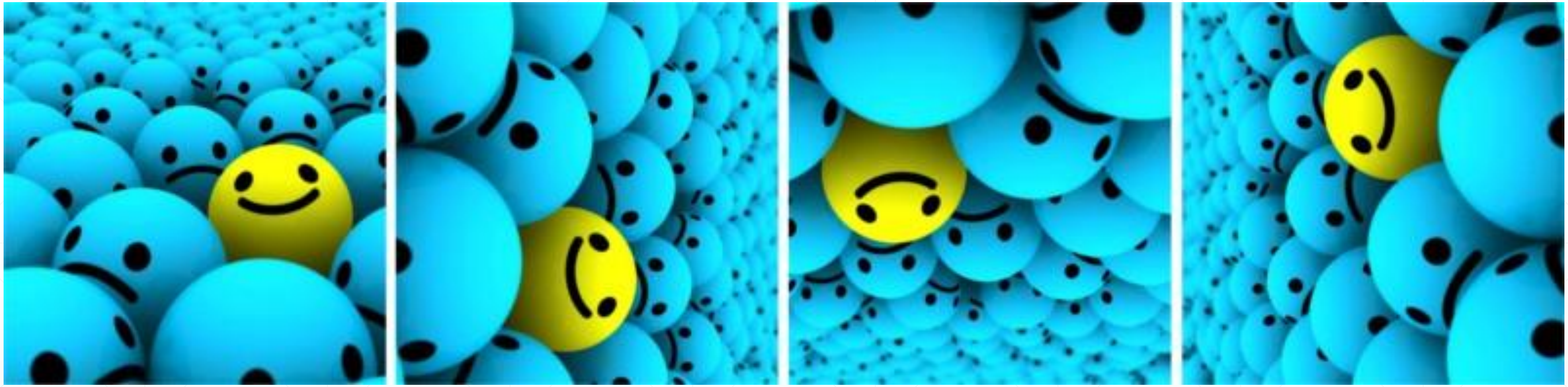


https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced

Flip

```python
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.RandomVerticalFlip(),
                       transforms.ToTensor(),
                       # transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True)
```

Rotate



Rotate

```python
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.RandomVerticalFlip(),
                       transforms.RandomRotation(15),
                       transforms.RandomRotation([90, 180, 270]),
                       transforms.ToTensor(),
                       # transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True)
```

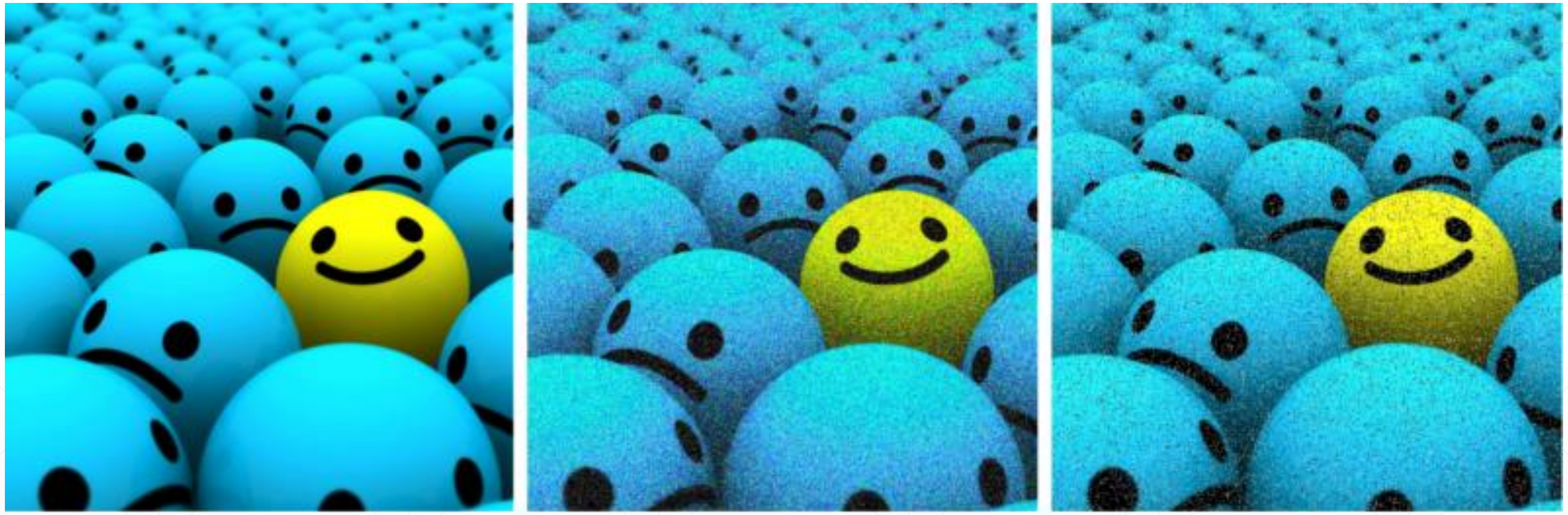
Scale

```python
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.RandomVerticalFlip(),
                       transforms.RandomRotation(15),
                       transforms.RandomRotation([90, 180, 270]),
                       transforms.Resize([32, 32]),
                       transforms.ToTensor(),
                       # transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True)
```

Crop Part

```python
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.RandomHorizontalFlip(),
                       transforms.RandomVerticalFlip(),
                       transforms.RandomRotation(15),
                       transforms.RandomRotation([90, 180, 270]),
                       transforms.Resize([32, 32]),
                       transforms.RandomCrop([28, 28]),
                       transforms.ToTensor(),
                       # transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True)
```

Noise



- Data argumentation will help
- But not too much

下一课时

艺术风格迁移

Thank You.