

COMP5212: Machine Learning

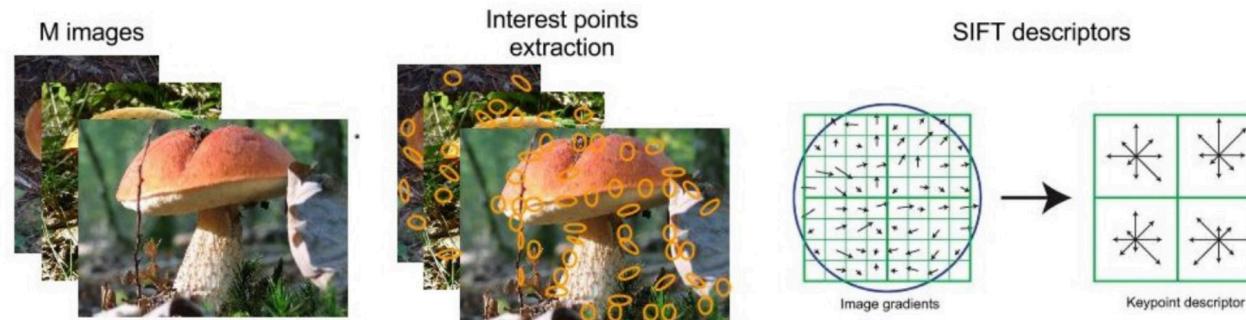
Lecture 13

Minhao Cheng

Convolutional Neural Network

Image classification without CNN

- Input an image
- Extract "interesting points" (e.g., corner detector)
- For each interesting point, extract 128-dimensional SIFT descriptor
- Clustering of SIFT descriptor to get "visual vocabulary"
- Then transform image to a feature vector (bag of visual words)
- Run classification (SVM)



* Mushroom image by Tifred25 (http://commons.wikimedia.org/wiki/File:Bolet_Orange_01.jpg)

Dataset **MNIST**

- Hand-written digits (0 to 9)
 - Total 60,000 samples, 10-class classification

Dataset

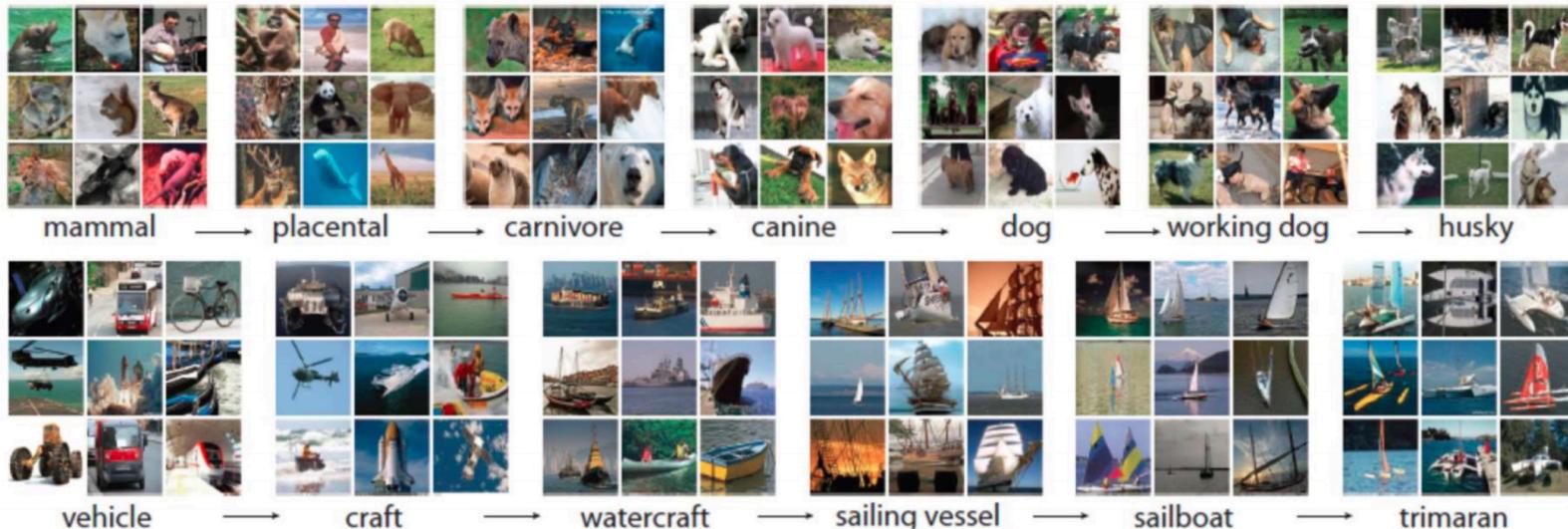
MNIST Classification Accuracy

- See the website by Yann LeCun: <http://yann.lecun.com/exdb/mnist/>

Classifier	Test Error
Linear classifier	12.0 %
SVM, Gaussian kernel	1.4%
SVM, degree 4 polynomial	1.1%
Best SVM result	0.56%
2-layer NN	~ 3.0%
3-layer NN	~ 2.5%
CNN, LeNet-5 (1998)	0.85%
Larger CNN (2011, 2012)	~ 0.3%

Dataset

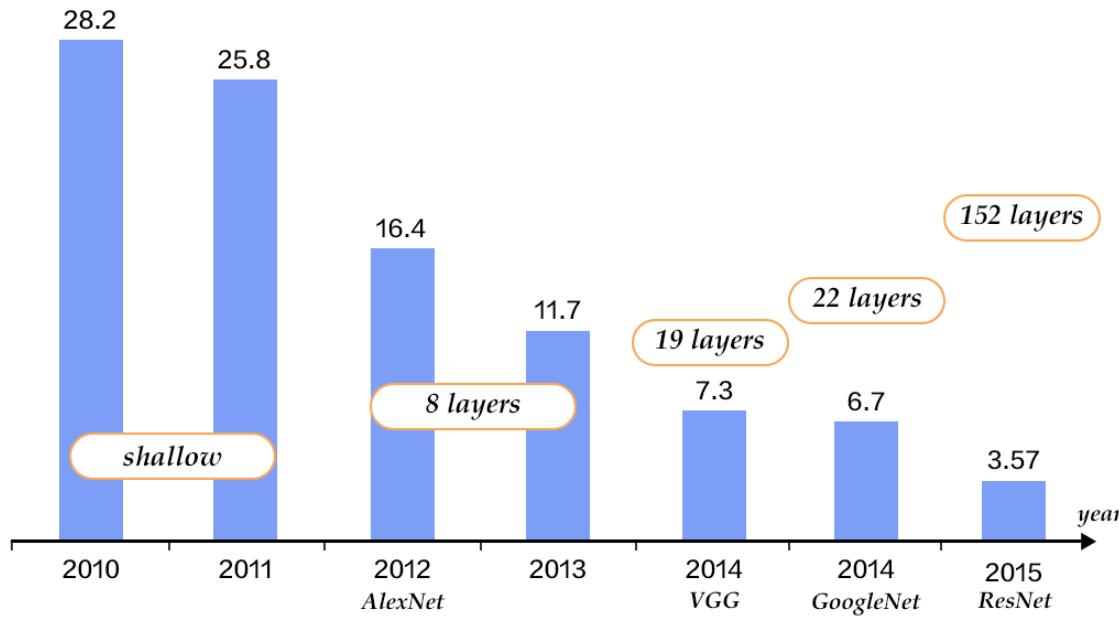
ImageNet Data



- ILSVRC competition: 1000 classes and about 1.2 million images
- Full imangenet: >20,000 categories, each with about a thousand images.

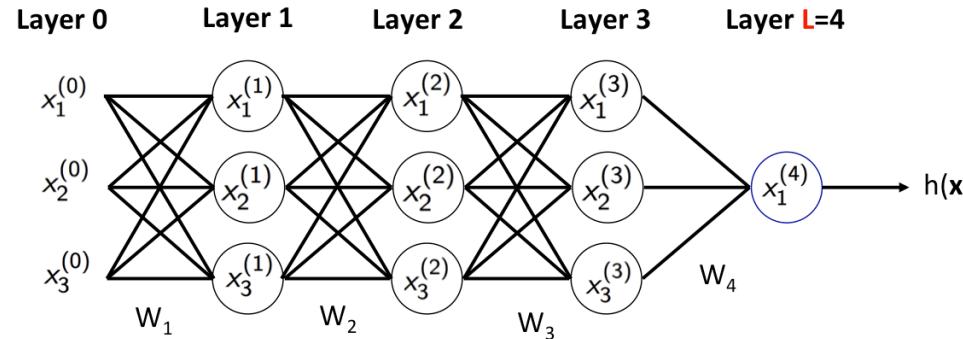
Dataset

ImageNet Results



Convolutional Neural Network

Neural Networks



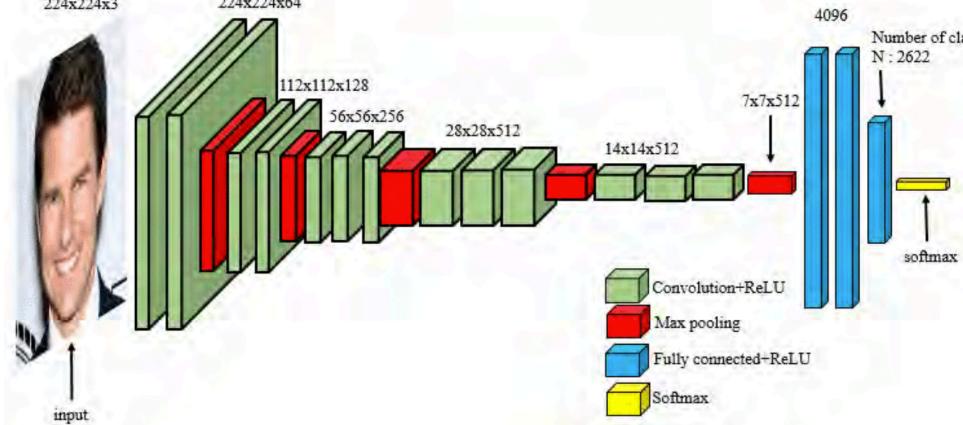
$$\begin{aligned} h(\mathbf{x}) &= x_1^{(4)} = \theta(W_4 \mathbf{x}^{(3)}) = \theta(W_4 \theta(W_3 \mathbf{x}^{(2)})) \\ &= \dots = \theta(W_4 \theta(W_3 \theta(W_2 \theta(W_1 \mathbf{x})))) \end{aligned}$$

- Fully connected networks \Rightarrow doesn't work well for computer vision applications

Convolutional Neural Network

The structure of CNN

- Structure of VGG



- Two important layers:

- Convolution
- Pooling

64x64x3

Convolutional Neural Network

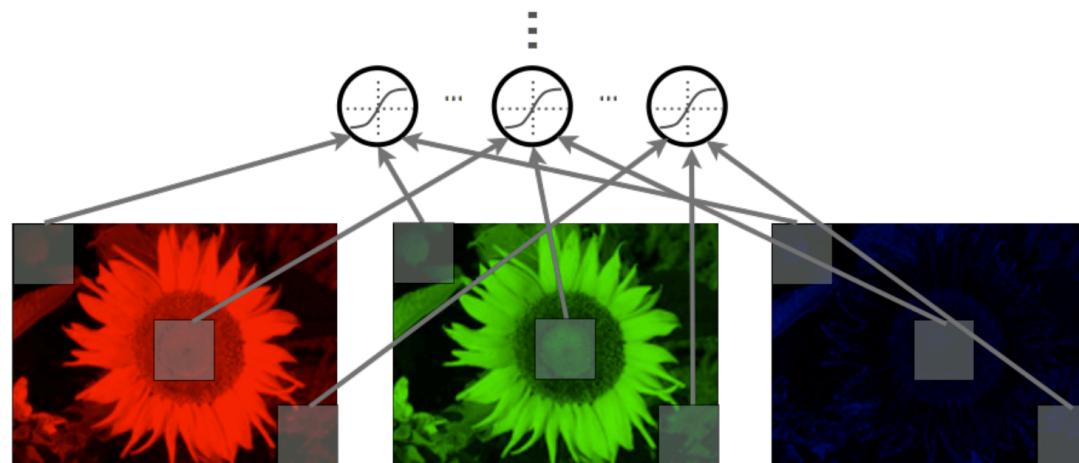
Convolution Layer

- Fully connected layers have too many parameters
 - ⇒ poor performance
- Example: VGG first layer
 - Input: $224 \times 224 \times 3$
 - Output: $224 \times 224 \times 64$
 - Number of parameters if we use fully connected net:
 - $(224 \times 224 \times 3) \times (224 \times 224 \times 64) = 483 \text{ billion}$
 - Convolution layer leads to:
 - Local connectivity
 - Parameter sharing

Convolutional Neural Network

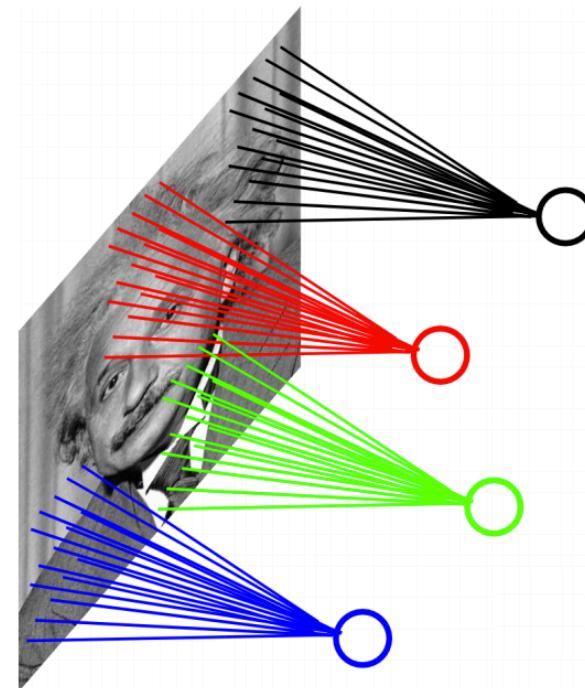
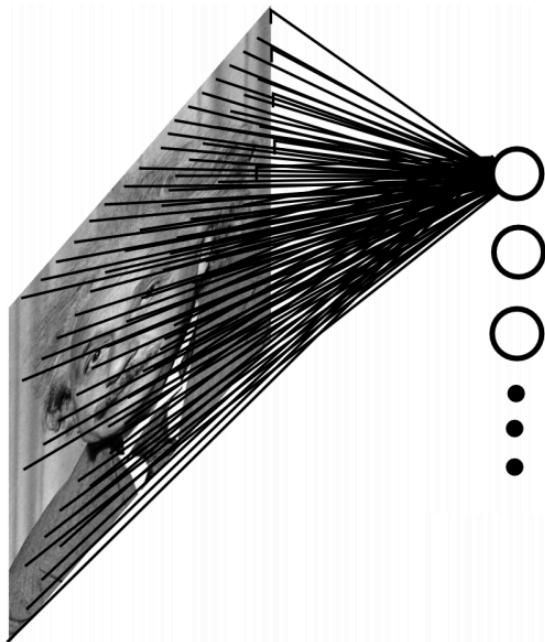
Local connectivity

- Each hidden unit is connected only to a sub-region of input
- It is connected to all channels (R, G, B)



Convolutional Neural Network

Local connectivity



Convolutional Neural Network

Parameter Sharing

- Making a reasonable assumption:
 - If one feature is useful to compute at some spatial position (x, y) ,
 - then it should also be useful to compute at a different position (x_2, y_2)
- Using the [convolution operator](#)

Convolutional Neural Network

Convolution

- The convolution of an image x with a kernel k is computed as

$$\bullet (x * k)_{ij} = \sum_{pq} x_{i+p,j+q} k_{p,q}$$

$$\begin{array}{|c|c|c|}\hline 1 & 0.5 & 20 \\ \hline 0.25 & 0 & 0 \\ \hline 0 & 0 & 20 \\ \hline\end{array} \quad * \quad \begin{array}{|c|c|}\hline 1 & 0.5 \\ \hline 0.25 & 0 \\ \hline\end{array} \quad = \quad \begin{array}{|c|c|}\hline & \\ \hline & \\ \hline\end{array}$$

Convolutional Neural Network

Convolution

$$1*1 + 0.5*0.2 + 0.25*0.2 + 0*0 = 1.15$$

The diagram illustrates a convolution operation. On the left, a 3x3 input matrix is shown with values: top row [1, 0.2, 0.5], middle row [0.2, 0, 0], bottom row [0.25, 0, 0]. The bottom-right element of the input is highlighted with a gray box. To its right is a multiplication symbol (*). Next is a 2x2 kernel matrix with values: top row [1, 0.2], bottom row [0.2, 0]. To its right is an equals sign (=). Finally, the result of the convolution is shown as a 2x2 output matrix with a single value: [1.15].

1	0.2	0.5	20
0.2	0	0	0
0.25	0	0	20

*

1	0.2
0.2	0

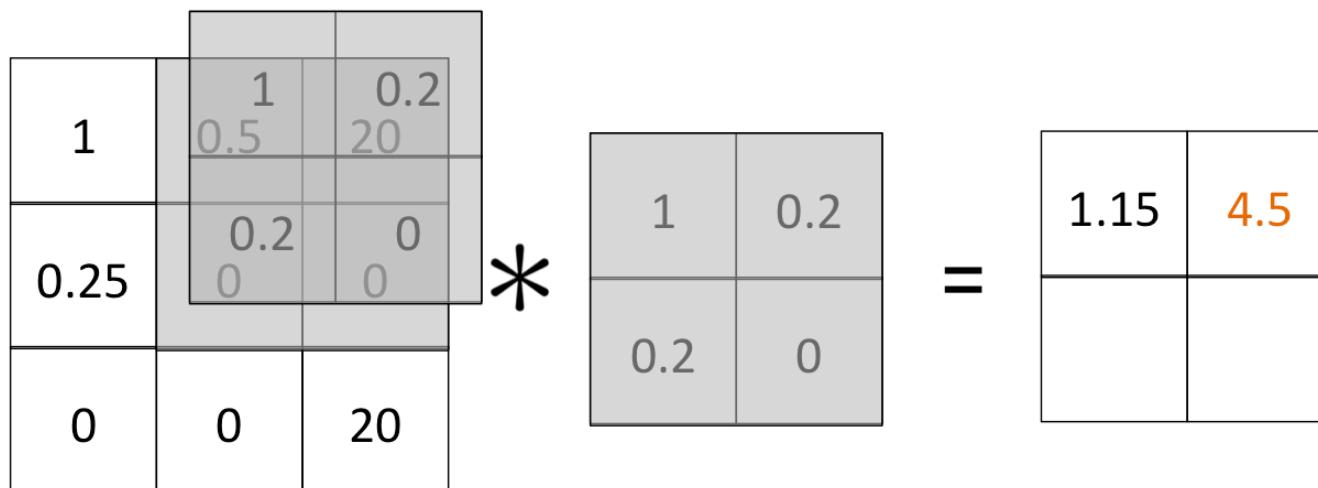
=

1.15	

Convolutional Neural Network

Convolution

$$0.5*1 + 20*0.2 + 0*0.2 + 0*0 = 4.5$$



Convolutional Neural Network

Convolution

$$0.25 * 1 + 0 * 0.2 + 0 * 0.2 + 0 * 0 = 0.25$$

The diagram illustrates a convolution operation. On the left, a 3x3 input matrix is shown with values: top row [1, 0.5, 20], middle row [0.25, 0, 0], bottom row [1, 0.2, 20]. A 2x2 kernel matrix is shown in the center, with values: top row [1, 0.2], bottom row [0.2, 0]. An asterisk (*) indicates the multiplication of the input and kernel matrices. An equals sign (=) indicates the result. On the right, the resulting 2x2 output matrix is shown with values: top row [1.15, 4.5], bottom row [0.25, 0]. The value 0.25 is highlighted in orange.

1	0.5	20
0.25	0	0
1	0.2	20

*

1	0.2
0.2	0

=

1.15	4.5
0.25	

Convolutional Neural Network

Convolution

$$0*1 + 0*0.2 + 0*0.2 + 20*0 = 0$$

1	0.5	20								
0.25	<table border="1"><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>0.2</td></tr></table>	0	0	1	0.2	<table border="1"><tr><td>0</td><td>0.2</td></tr><tr><td>0.2</td><td>0</td></tr></table>	0	0.2	0.2	0
0	0									
1	0.2									
0	0.2									
0.2	0									
0	<table border="1"><tr><td>0</td><td>20</td></tr><tr><td>0.2</td><td>0</td></tr></table>	0	20	0.2	0	<table border="1"><tr><td>1.15</td><td>4.5</td></tr><tr><td>0.25</td><td>0</td></tr></table>	1.15	4.5	0.25	0
0	20									
0.2	0									
1.15	4.5									
0.25	0									

*

1	0.2
0.2	0

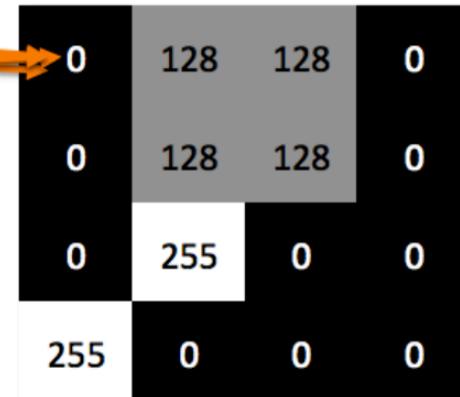
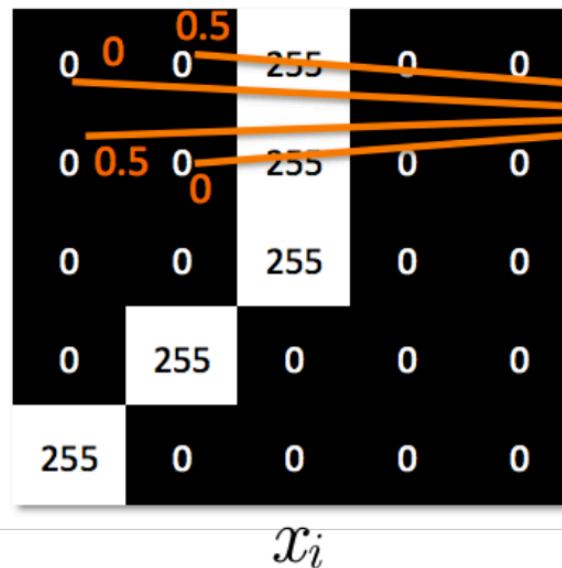
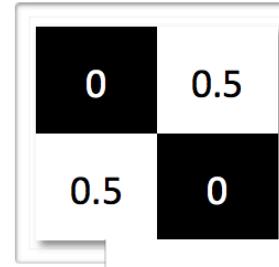
=

1.15	4.5
0.25	0

Convolutional Neural Network

Convolution

$$x * k_{ij}, \text{ where } W_{ij} = \tilde{W}_{ij}$$



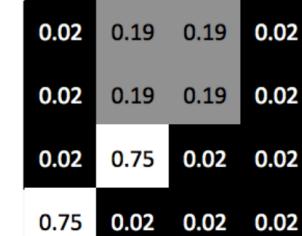
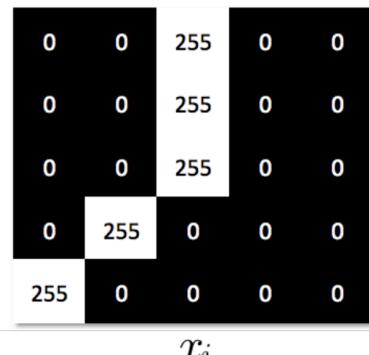
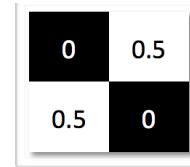
$$x_i * k_{ij}$$

Convolutional Neural Network

Convolution

- Element-wise activation function after convolution
 - \Rightarrow detector of a feature at any position in the image

$$x * k_{ij}, \text{ where } W_{ij} = \tilde{W}_{ij}$$

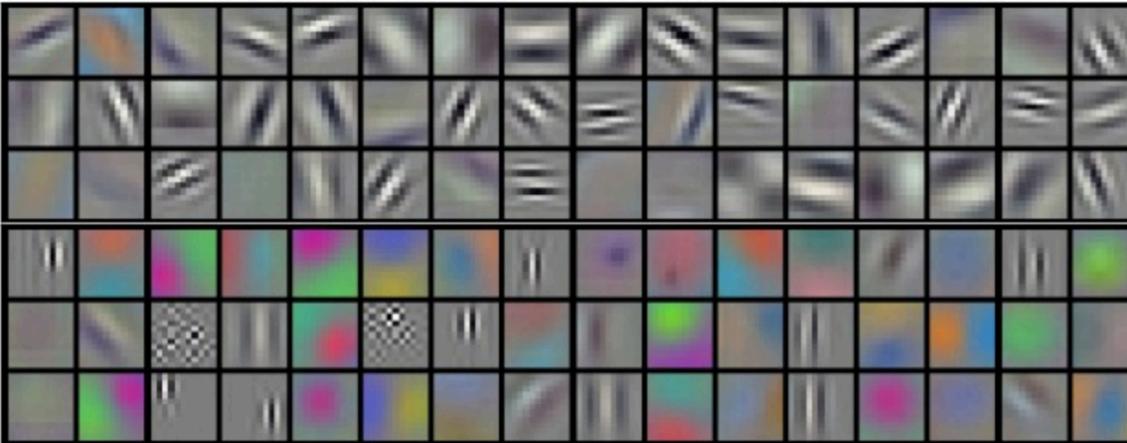


$$\text{sigm}(0.02 \cdot x_i * k_{ij} - 4)$$

Convolutional Neural Network

Learned Kernels

- Example kernels learned by AlexNet

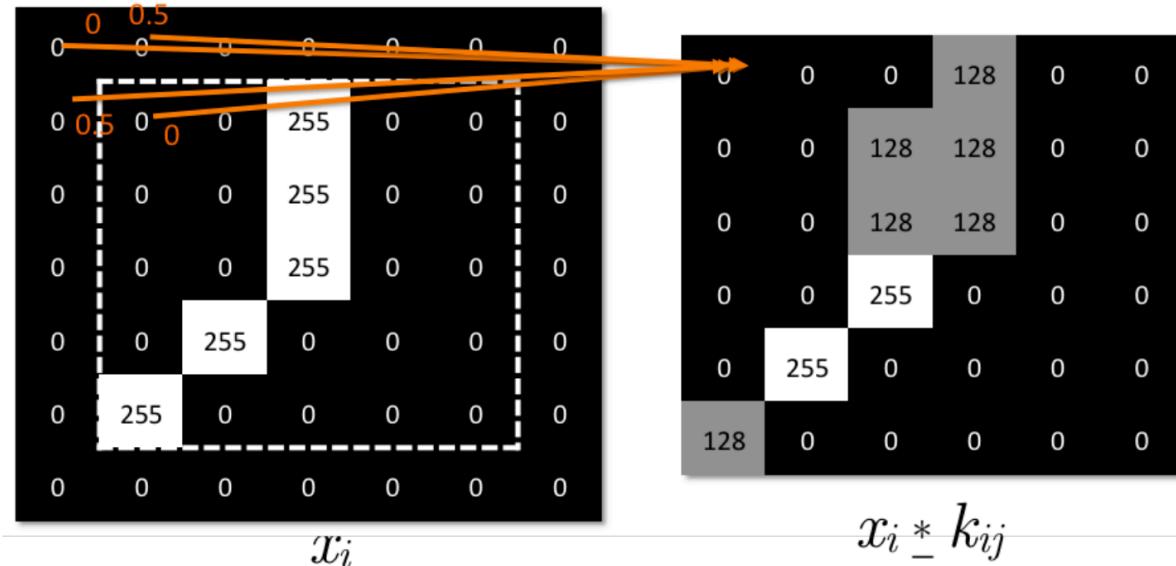


- Number of parameters:
 - Example: 200×200 image, 100 kernels, kernel size 10×10
 - $\Rightarrow 10 \times 10 \times 100 = 10K$ parameters

Convolutional Neural Network

Padding

- Use **zero padding** to allow going over the boundary
 - Easier to control the size of output layer



Convolutional Neural Network

Strides

- Stride: The amount of movement between applications of the filter to the input image
- Stride (1,1): no stride

$3 \times 32 \times 32$

kernel-size: 5×5

kernel-no : 66

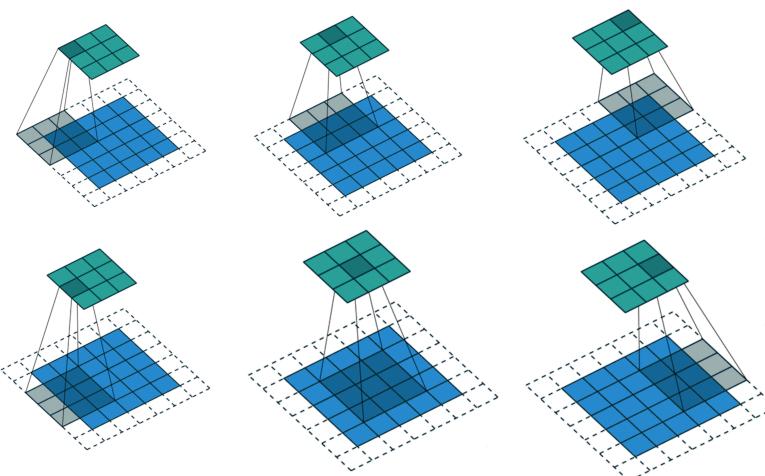
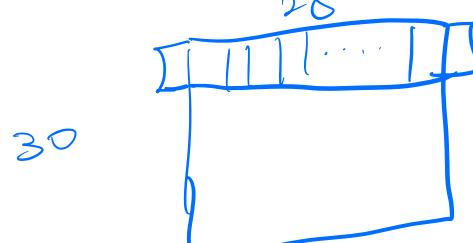
$66 \times 28 \times 28$

kernel-size : 5×5

kernel-no : 128

(stride=2
padding=1)

$128 \times 13 \times 13$



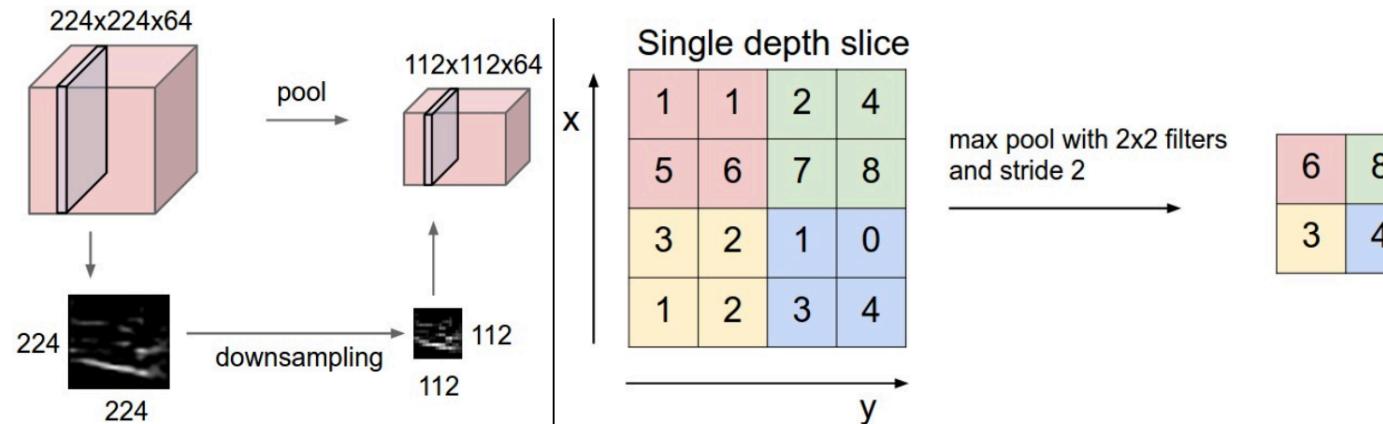


Convolutional Neural Network

Pooling

$26 \sim 30$

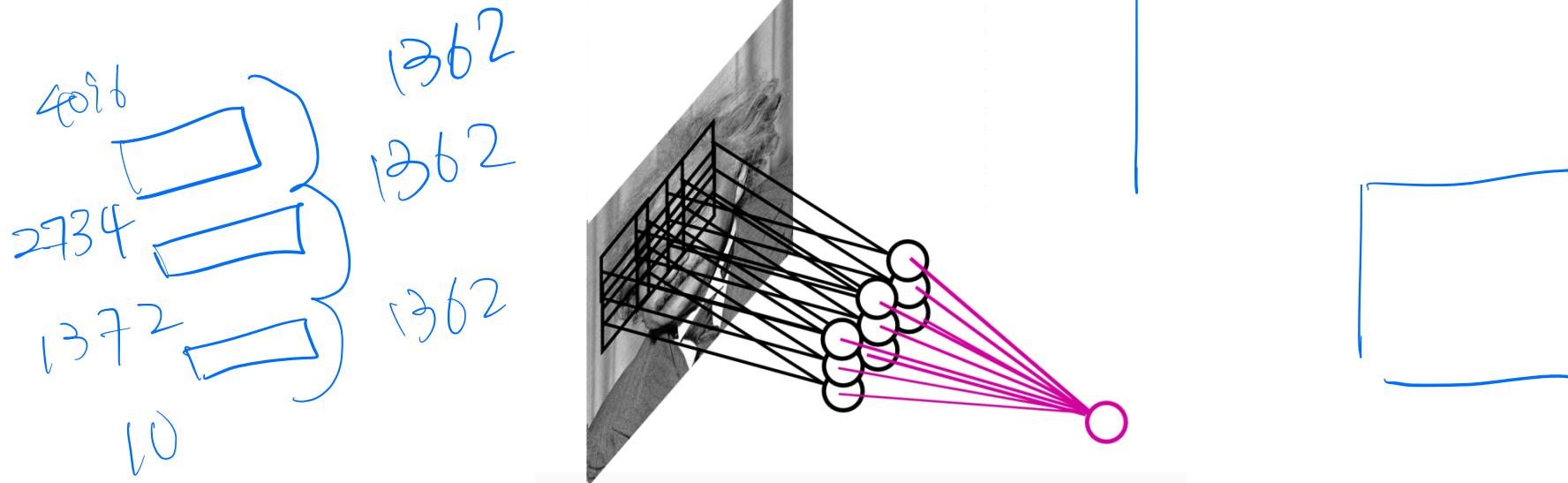
- It's common to insert a **pooling layer** in-between successive convolutional layers
- Reduce the size of presentation, down-sampling
- Example: **Max pooling**



Convolutional Neural Network

Pooling

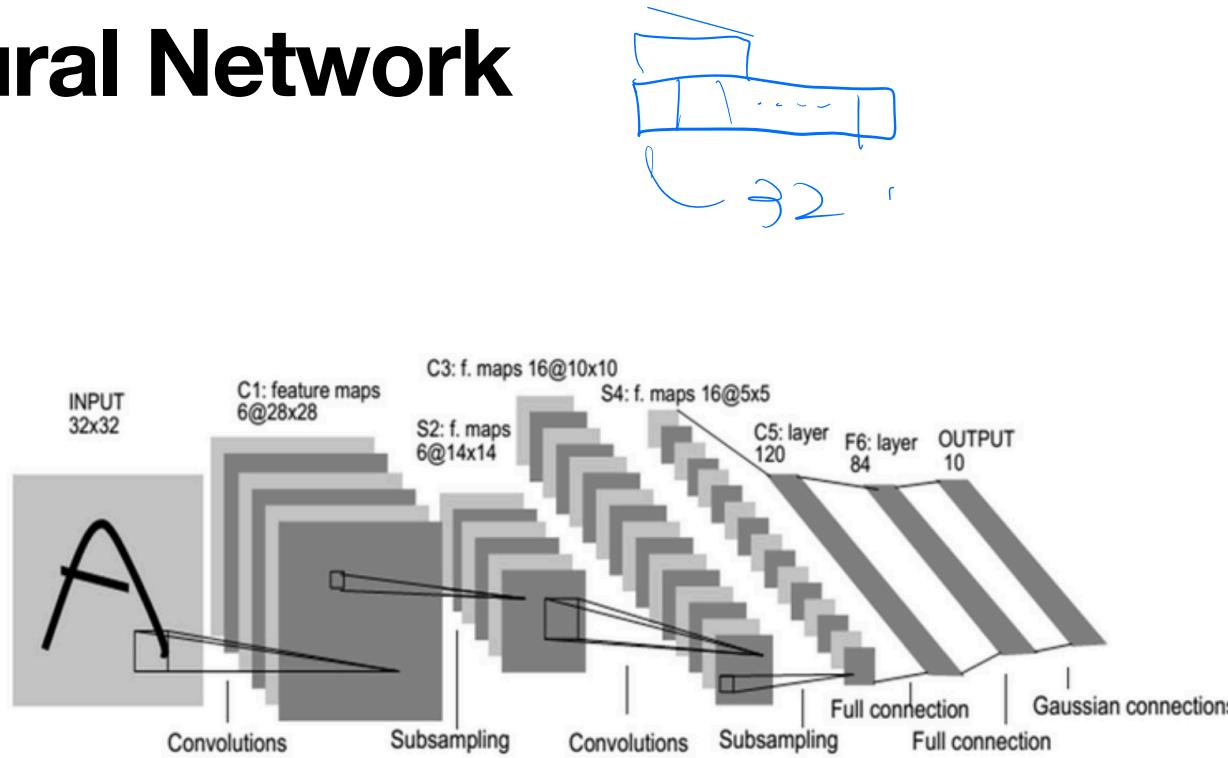
- By **pooling**, we gain robustness to the exact spatial location of features



Convolutional Neural Network

Example: LeNet5

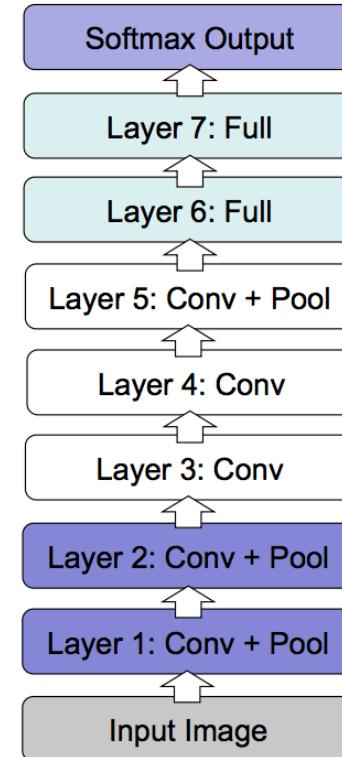
- Input: 32×32 images (MNIST)
- Convolution 1: 6 5×5 filters, stride 1
 - Output: 6 28×28 maps
- Pooling 1: 2×2 max pooling, stride 2
 - Output: 6 14×14 maps
- Convolution 2: 16 5×5 filters, stride 1
 - Output: 16 10×10 maps
- Pooling 2: 2×2 max pooling with stride 2
 - Output: 16 5×5 maps (total 400 values)
- 3 fully connected layers: 120 \Rightarrow 84 \Rightarrow 10 neurons



Convolutional Neural Network

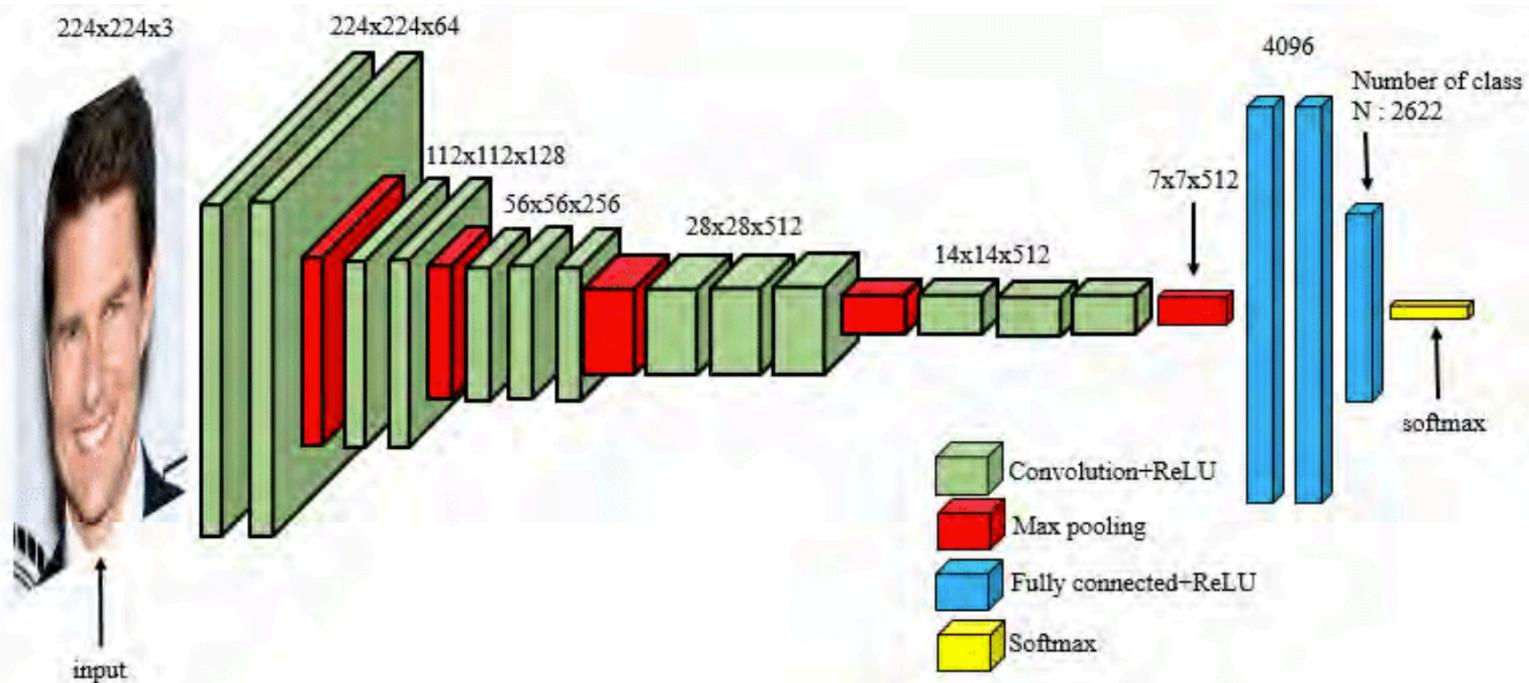
AlexNet

- 8 layers in total, about 60 million parameters and 650,000 neurons.
- Trained on ImageNet dataset
- 18.2% top-5 error
- "ImageNet Classification with Deep Convolutional Neural Networks", by Krizhevsky, Sutskever and Hinton, NIPS 2012.



Convolutional Neural Network

VGG Network



Convolutional Neural Network

VGG Network

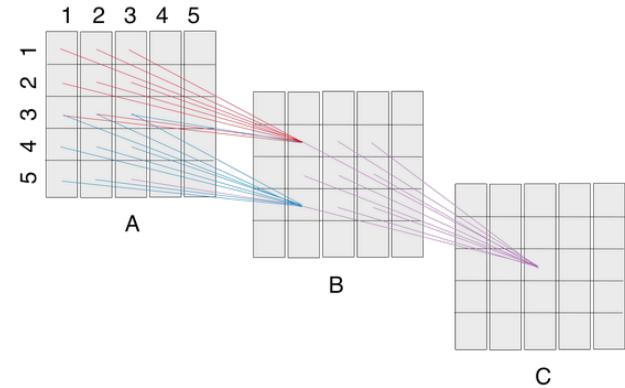
- Output provides an estimate of the conditional probability of each class

```
INPUT: [224x224x3]           memory: 224*224*3=150K  weights: 0
CONV3-64: [224x224x64]      memory: 224*224*64=3.2M  weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]      memory: 224*224*64=3.2M  weights: (3*3*64)*64 = 36,864
POOL2: [112x112x64]         memory: 112*112*64=800K  weights: 0
CONV3-128: [112x112x128]    memory: 112*112*128=1.6M  weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]    memory: 112*112*128=1.6M  weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]          memory: 56*56*128=400K  weights: 0
CONV3-256: [56x56x256]      memory: 56*56*256=800K  weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]      memory: 56*56*256=800K  weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]      memory: 56*56*256=800K  weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]          memory: 28*28*256=200K  weights: 0
CONV3-512: [28x28x512]      memory: 28*28*512=400K  weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]      memory: 28*28*512=400K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]      memory: 28*28*512=400K  weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]          memory: 14*14*512=100K  weights: 0
CONV3-512: [14x14x512]      memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]      memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]      memory: 14*14*512=100K  weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]             memory: 7*7*512=25K   weights: 0
FC: [1x1x4096]               memory: 4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]               memory: 4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]               memory: 1000  weights: 4096*1000 = 4,096,000
```

Convolutional Neural Network

What do the filters learn?

- The **receptive field** of a neuron is the input region that can affect the neuron's output
- The receptive field for a first layer neuron is its neighbors (depending on kernel size) \Rightarrow capturing very local patterns
- For higher layer neurons, the receptive field can be much larger \Rightarrow capturing global patterns



Convolutional Neural Network

Training

- Training:
 - Apply SGD to minimize in-sample training error
 - Backpropagation can be extended to convolutional layer and pooling layer to compute gradient!
 - Millions of parameters ⇒ easy to overfit

Convolutional Neural Network

Revisit Alexnet

- Dropout: 0.5 (in FC layers)
- A lot of data augmentation
- Momentum SGD with batch size 128, momentum factor 0.9
 - It can be linked directly to the L2 regularization
- L2 weight decay (L2 regularization)
 - by transferring the regularization term directly to the update term
- Learning rate: 0.01, decreased by 10 every time when reaching a stable validation accuracy

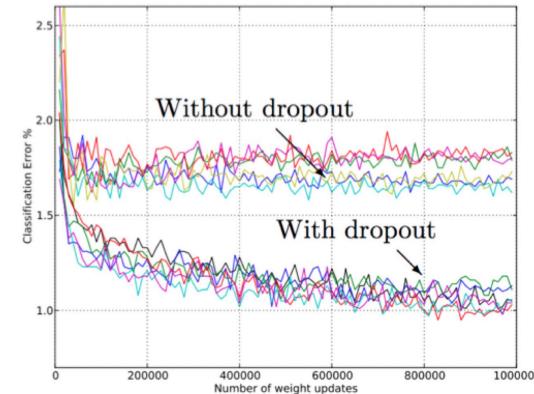
Convolutional Neural Network

Dropout

- One of the most effective regularization for deep neural networks

Method	CIFAR-10	CIFAR-100
Conv Net + max pooling (hand tuned)	15.60	43.48
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	15.13	42.51
Conv Net + max pooling (Snoek et al., 2012)	14.98	-
Conv Net + max pooling + dropout fully connected layers	14.32	41.26
Conv Net + max pooling + dropout in all layers	12.61	37.20
Conv Net + maxout (Goodfellow et al., 2013)	11.68	38.57

Table 4: Error rates on CIFAR-10 and CIFAR-100.

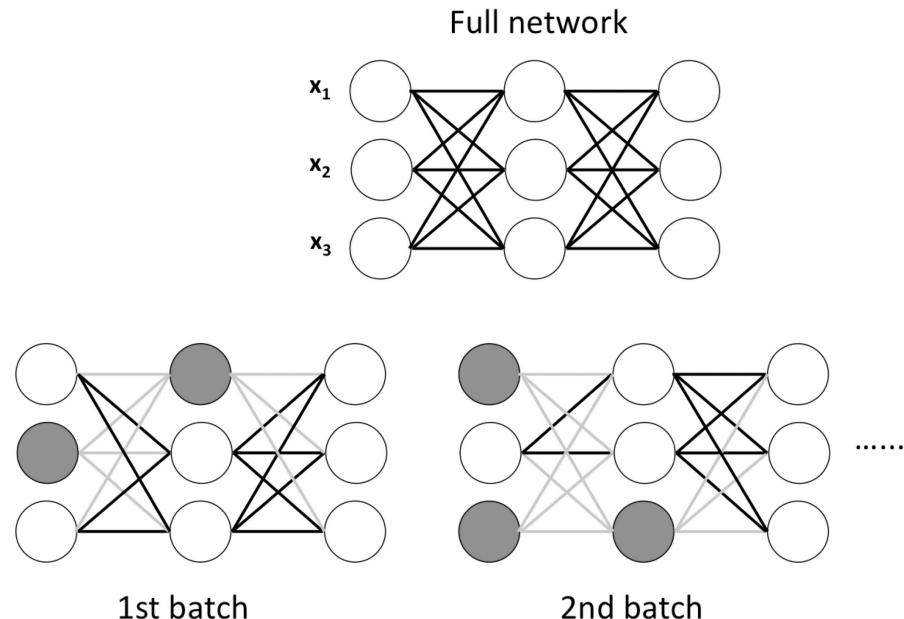


Srivastava et al, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, 2014.

Convolutional Neural Network

Dropout(**training**)

- Dropout in the **training** phase:
 - For each batch, turn off each neuron (including inputs) with a probability $1 - \alpha$
 - Zero out the removed nodes/edges and do backpropogation



Convolutional Neural Network

Dropout(test)

- The model is different from the full model:

- Each neuron computes

- $x_i^{(l)} = B\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$

- Where B is Bernoulli variable that takes 1 with probability α

- The expected output of the neuron:

- $E[x_i^{(l)}] = \alpha\sigma(\sum_j W_{ij}^{(l)} x_j^{(l-1)} + b_i^{(l)})$

- Use the **expected output** at test time \Rightarrow multiply all the weights by α

Convolutional Neural Network

Explanations of dropout

- For a network with n neurons, there are 2^n possible sub-networks
- Dropout: randomly sample over all 2^n possibilities
Avoid overfitting
- Can be viewed as a way to learn Ensemble of 2^n models

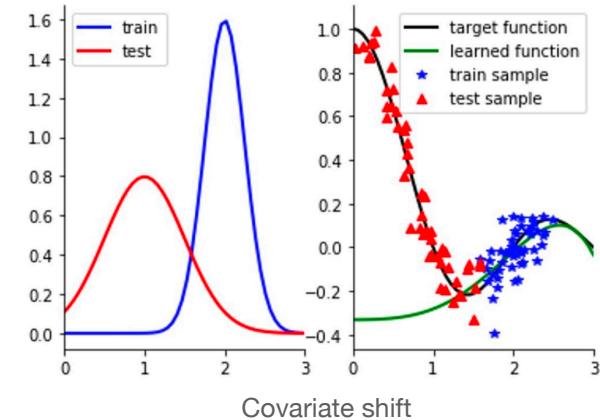
Convolutional Neural Network

Batch Normalization

- Initially proposed to reduce co-variate shift

$$O_{b,c,x,y} \leftarrow \gamma \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta \quad \forall b, c, x, y,$$

- $\mu_c = \frac{1}{|B|} \sum_{b,x,y} I_{b,c,x,y}$: the mean for channel c , and σ_c standard deviation.
- γ and β : two learnable parameters



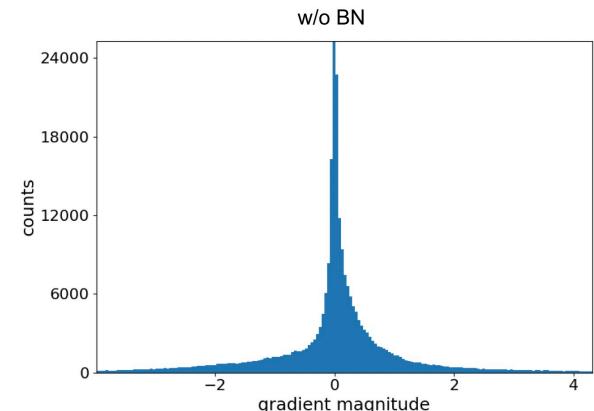
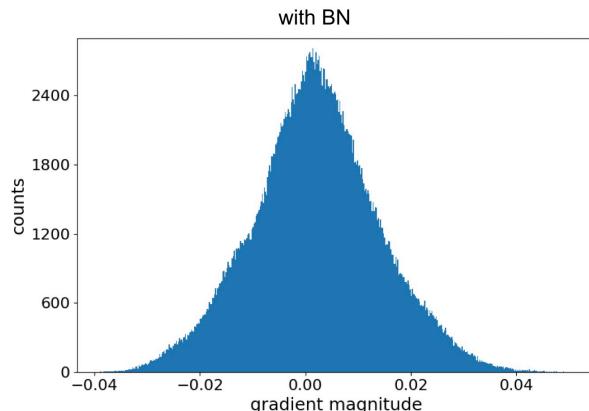
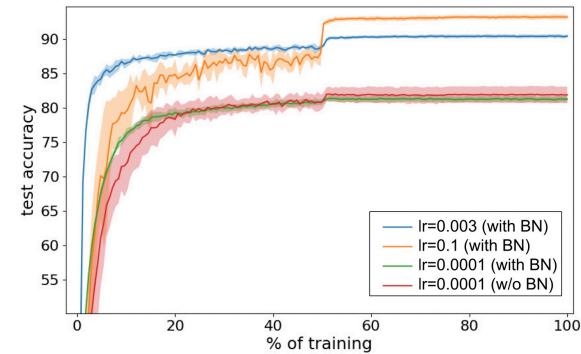
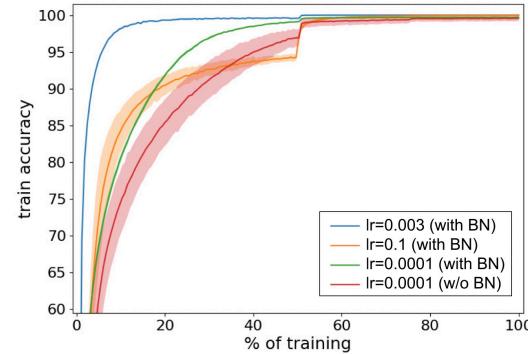
Convolutional Neural Network

Batch Normalization

- Couldn't reduce covariate shift (Ilyas et al 2018)
- Allow larger learning rate
 - Constraint the gradient norm

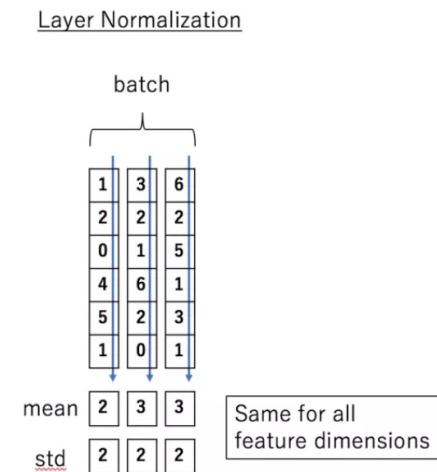
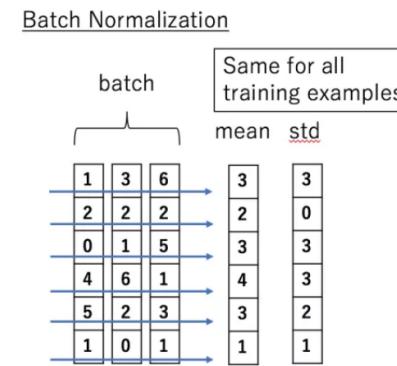
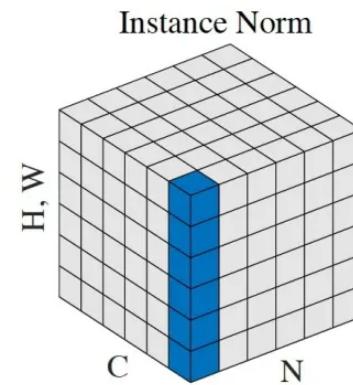
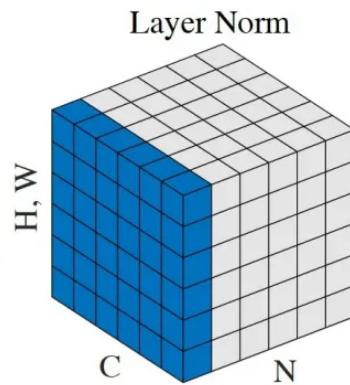
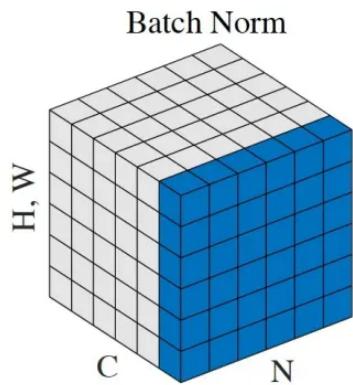
gradient will not be too large

BN is usually placed frequently in the entire model network



Convolutional Neural Network

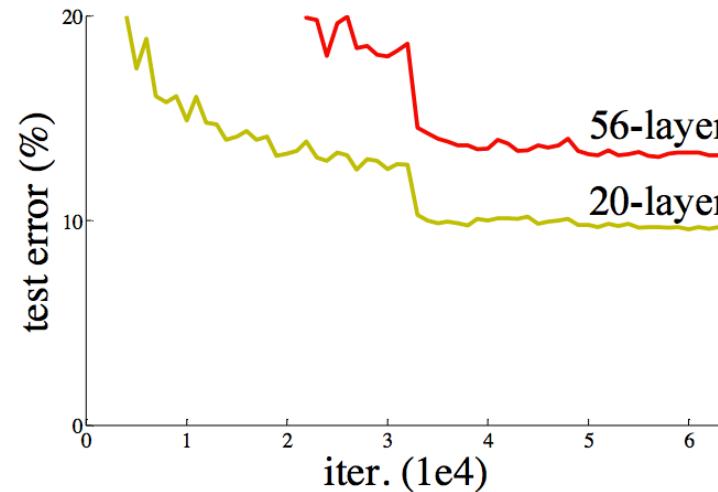
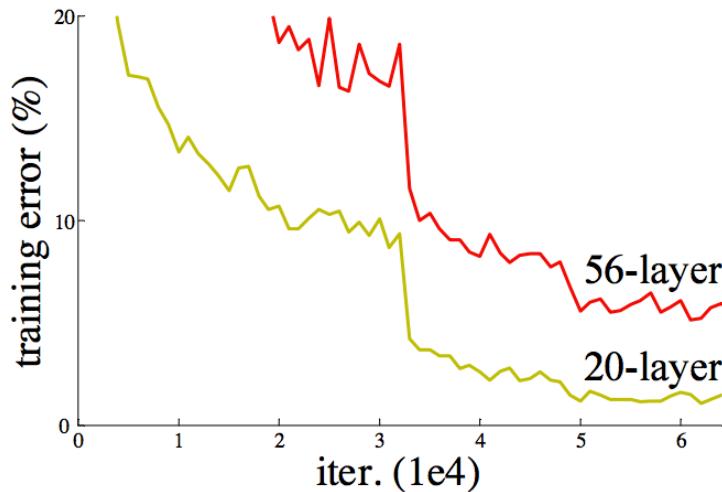
Other normalization



Convolutional Neural Network

Residual Networks

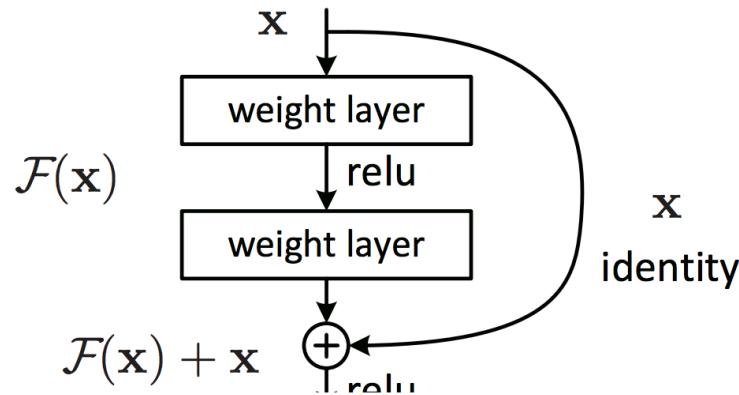
- Very deep convnets do not train well – vanishing gradient problem



Convolutional Neural Network

Residual Networks

- Key idea: introduce “pass through” into each layer



- Thus, only residual needs to be learned

Convolutional Neural Network

Residual Networks

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43 [†]
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	19.38	4.49

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except [†] reported on the test set).

