Deep Learning for Visual Recognition Part 1

Injung Kim
Handong Global University

Agenda

- Introduction to Deep Learning
- CNN Architectures
- Training of CNN
- Deep Learning Issues

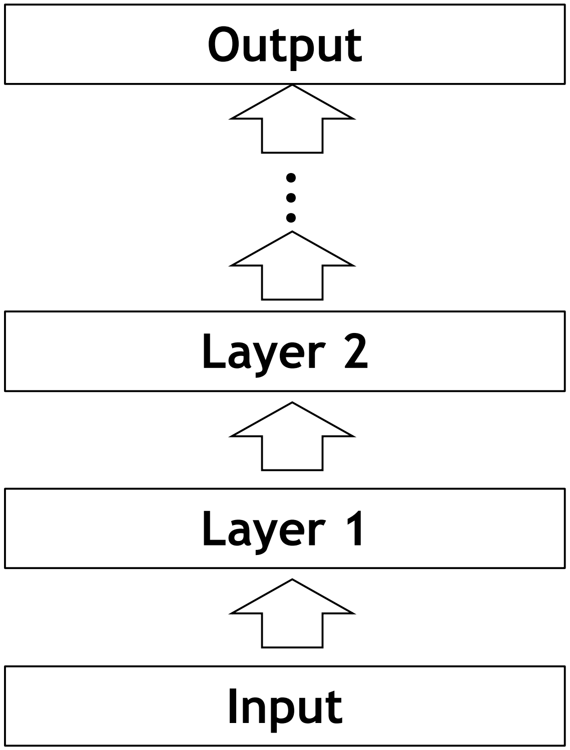
Deep Learning

- A branch of machine learning to model **high-level abstractions in data**, mostly, based on **deep networks**.

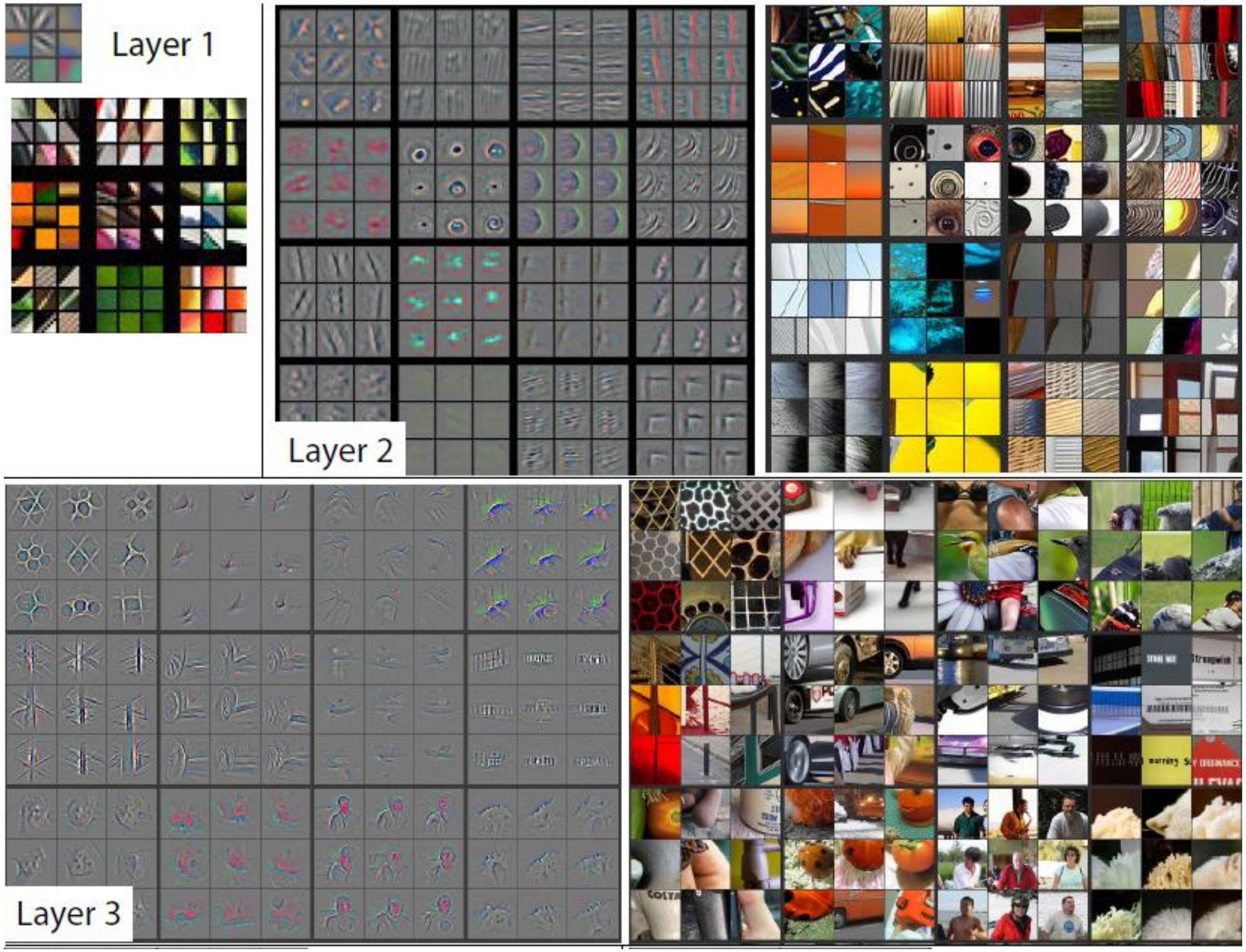
  - Each layer combines input features to produce high-level features.

$$o = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

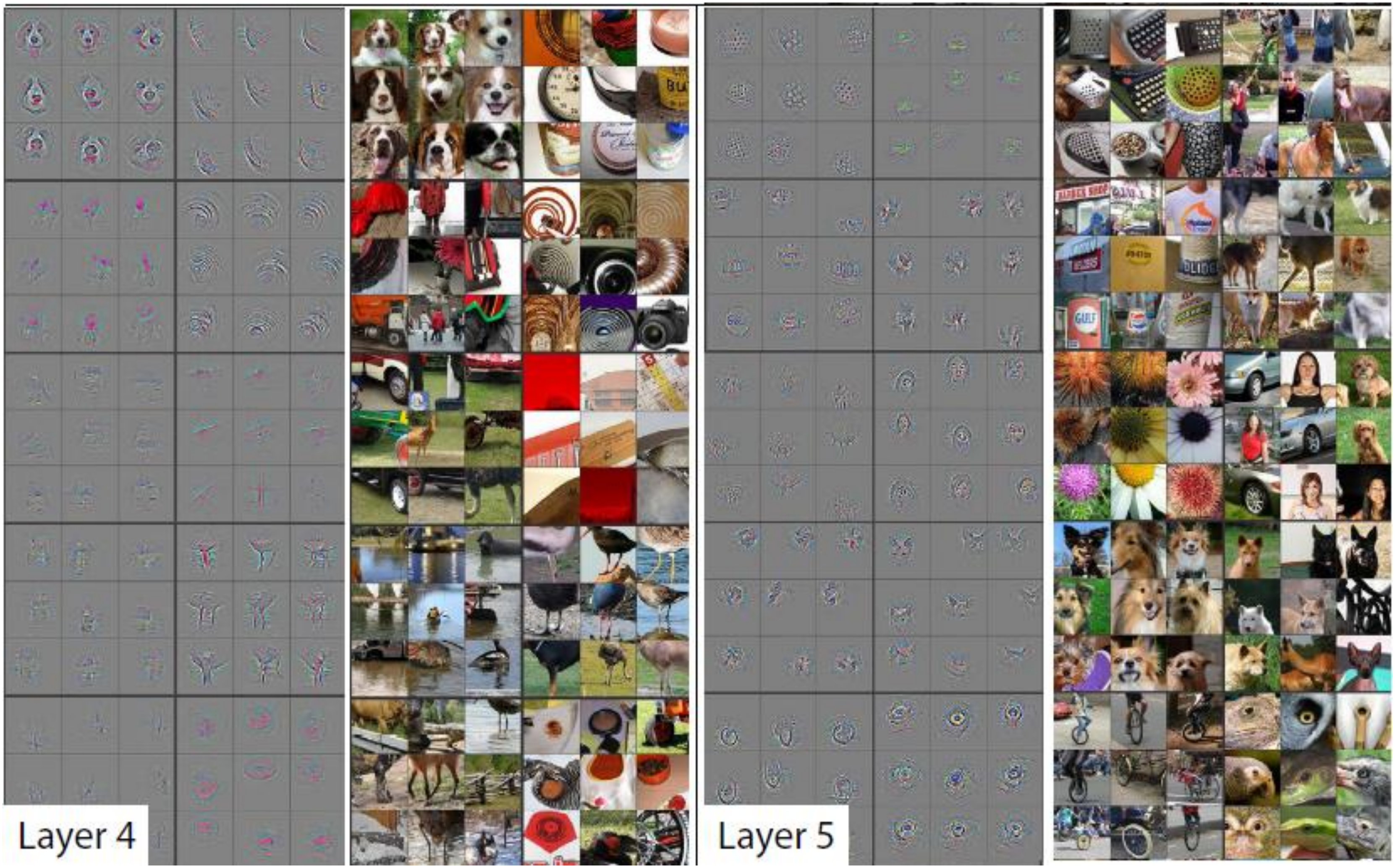
  - A neural network with many layers can learn high-level features.



Visualization of Low-level Layers [Zeiler2013]

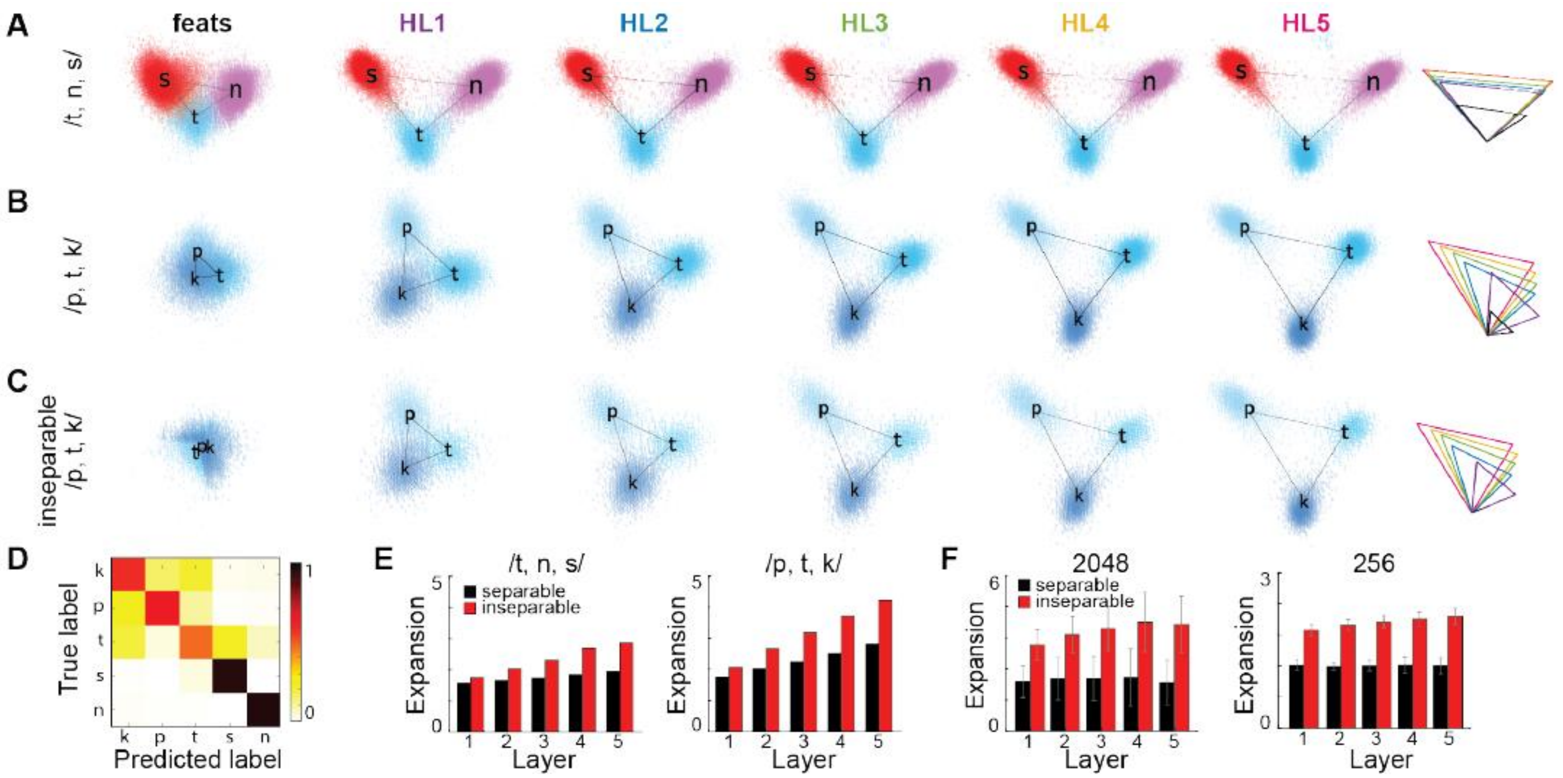


Visualization of High-level Layers [Zeiler2013]



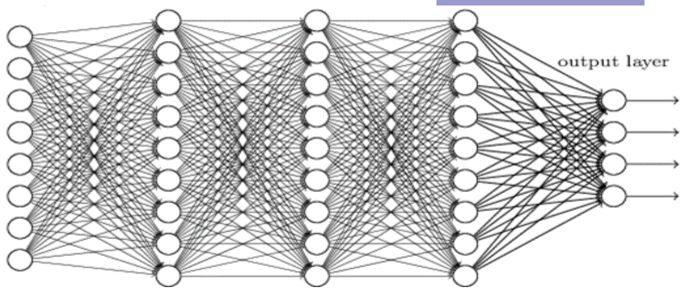
Feature of MLP Layers

- Nagamine and Mesgarani, "Understanding the Representation and Computation of Multilayer Perceptrons: A Case Study in Speech Recognition," 2017.

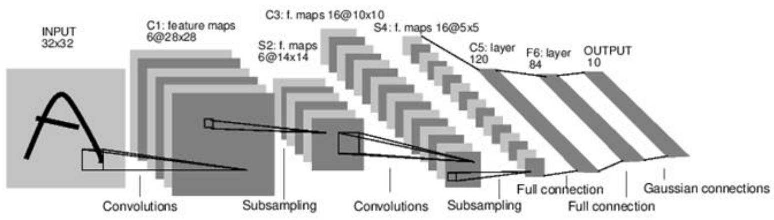


Deep Learning Architectures

- Deep Neural Networks (DNN)
  - MLP, SOM, RBF, …
  - RBM, DBN, DBM, …

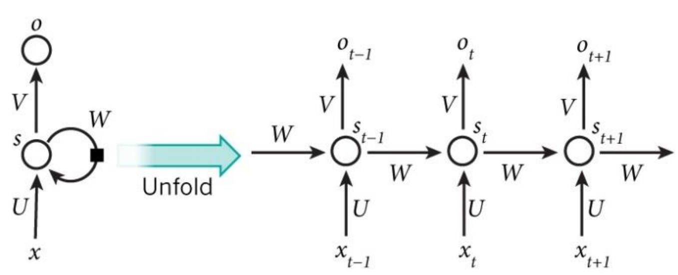


- Convolutional neural networks (CNN)
  - Recognition/processing/generation of images
  - Combines heterogeneous layers (convolution, pooling, etc.)
  - Learns position invariant local features



A Full Convolutional Neural Network (LeNet)

- Recurrent neural networks (RNN)
  - Recognition/processing/generation of time-series data
  - Recurrent connection (memory)
  - Input + context



Deep Learning Architectures

- Hybrid models / combined models
  - Convolutional RBM, Recurrent CNN, Long-term recurrent convolutional network (LSTM + CNN), CBHG
- Deep generative models
  - GAN, VAE, pixel RNN/CNN
- Detection / segmentation models
  - R-CNN, Fast/Faster R-CNN, Mask R-CNN, YOLO, SSD
  - FCN, SegNet, RefineNet, DeepLab, JPP net, etc.
- Attention models
  - Machine translation, ASR/TTS, visual recognition
  - Transformer, non-local nets, self-attention models
- Explicit memory models
  - Memory networks, neural Turing machines (NTM)

CNN Architectures

Biological Discoveries [Hubel&Wiesel 1962]

- Visual area – neurons responds selectively to local feature of a visual patterns

  Ex) lines and edges.

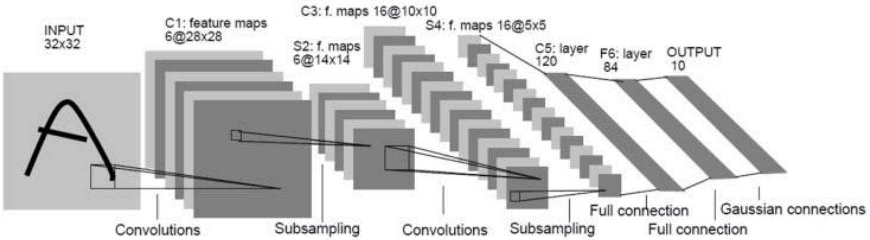
- Higher area – neurons responds selectively to higher level features

  Ex) circles, triangles, squares, human face

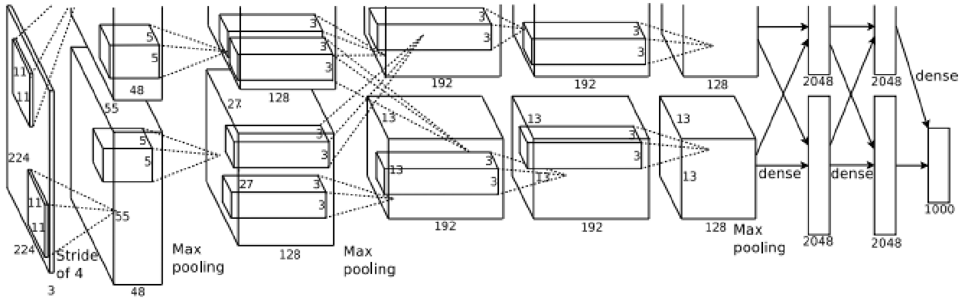
- Neural networks in brain are not complete at birth.
  - They gradually develop, adapting flexibly to circumstances after birth.

Convolutional Neural Networks

- Convolutional Neural networks (CNN): a class of deep feed-forward network designed to mimic human/animal visual systems
  - LeNet 1998, MNIST, CPU

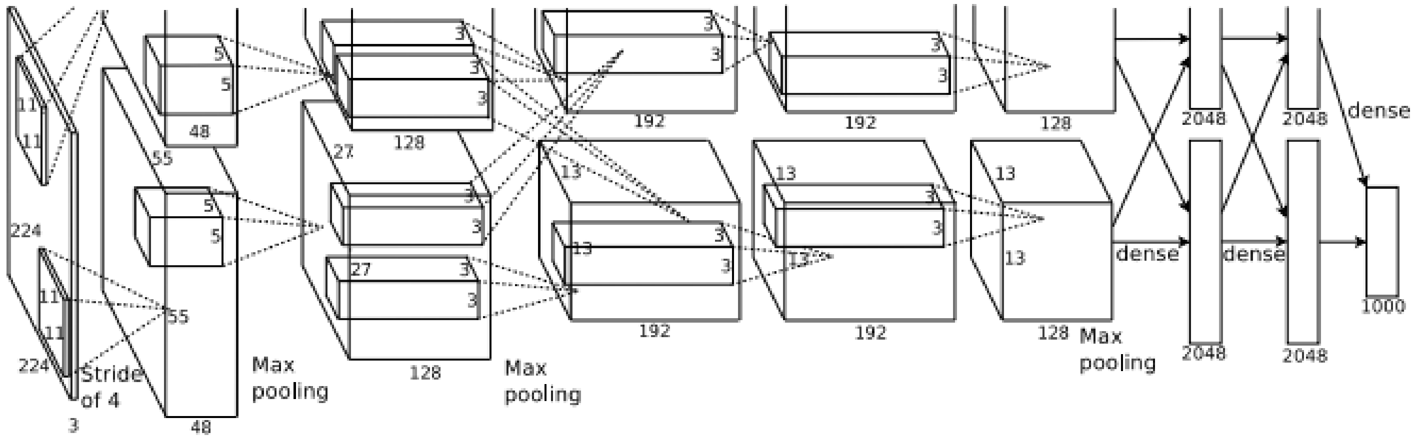


  - AlexNet 2012, ImageNet, GTX 580 x 2



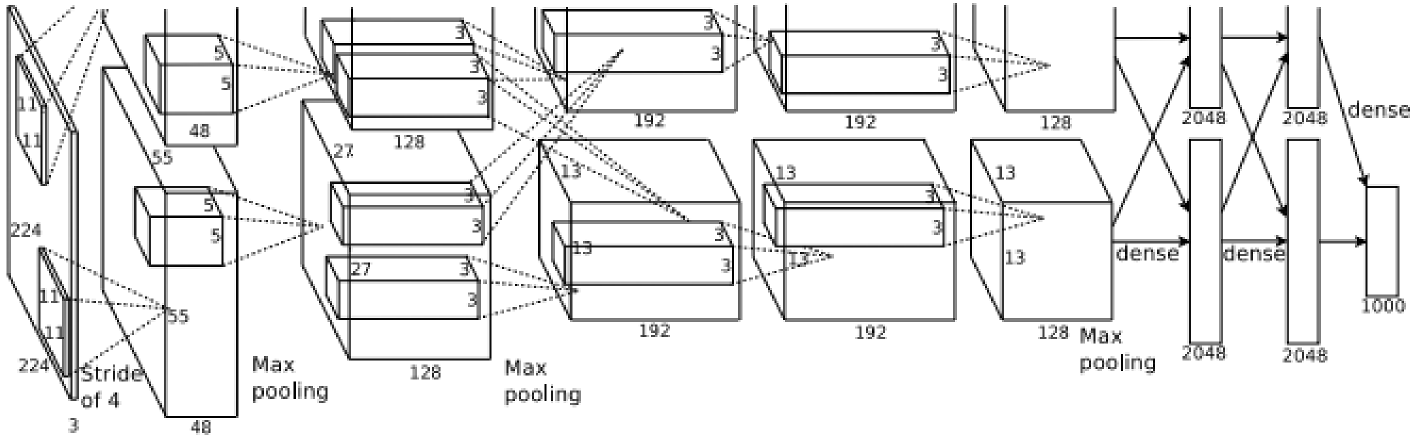
Convolutional Neural Networks (CNN)

- Most CNN layers input/output 3-5D tensors
  - 3D: channels, rows, columns
  - 4D: channels, rows, columns + batch
  - 5D: channels, rows, columns + batch + time
    - Recurrent convolutional layers



Convolutional Neural Networks (CNN)

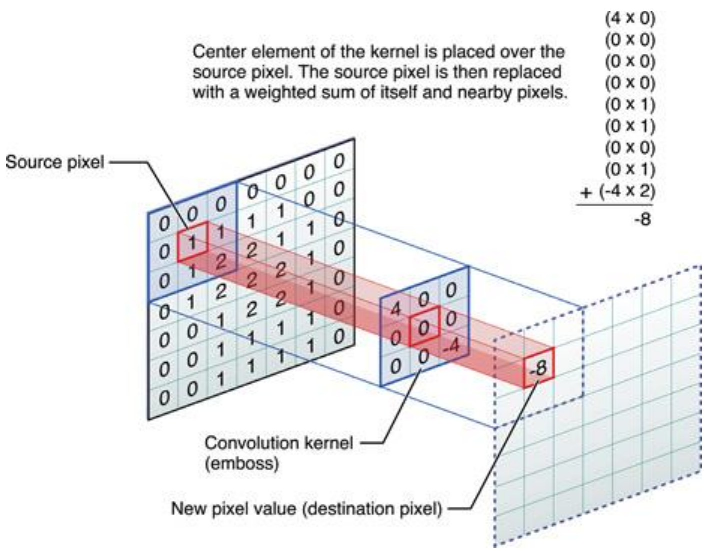
- Composed of heterogeneous layers
  - Convolution (standard, transposed, dilated, separable)
  - Pooling (max, GAP, average, SPP, …)
  - Fully-connected
  - Batch/instance/layer/group normalization, dropout
  - Skip connections (Highway, ResNet, DenseNet, DPN)
  - ROI pooling, RPN, etc.

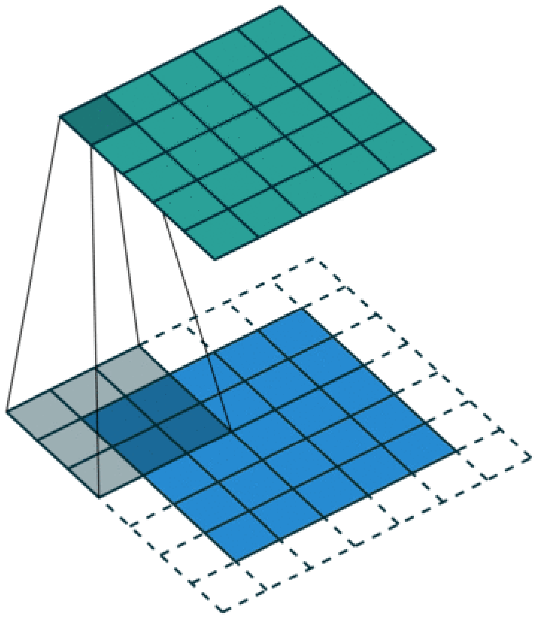


Convolution Layers

- Convolution layers extract **position-invariant local features** by convolution operation

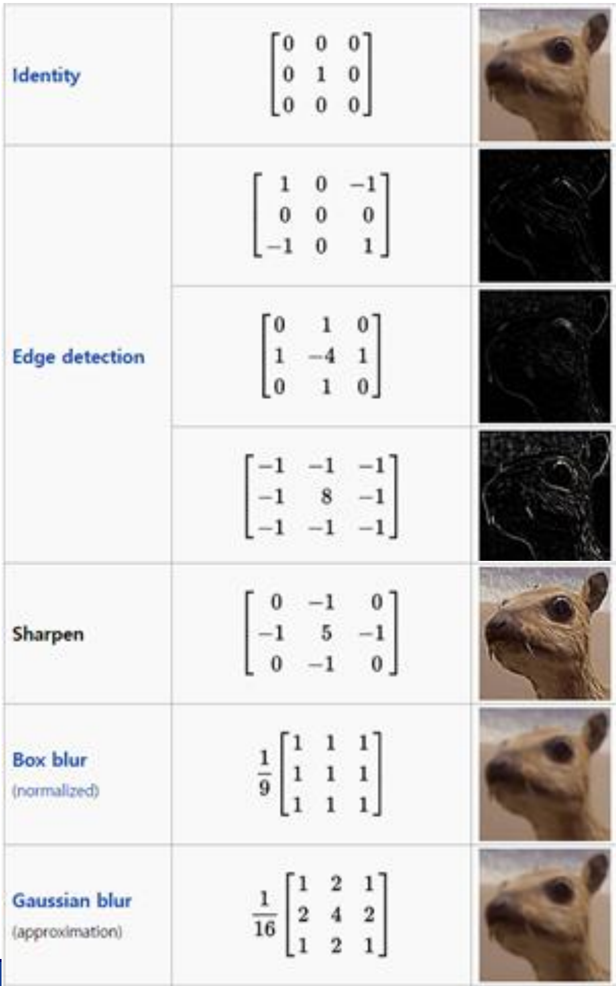
$$X^{n}_{(p,i,j)} = f\left(\sum_{q\in C^{n}_{p}}\sum_{0\le u,v\le M_n-1} w^{n}_{(q,p,u,v)} X^{n-1}_{(q,iS_n+u,jS_n+v)} + \theta^{n}_{p}\right)$$





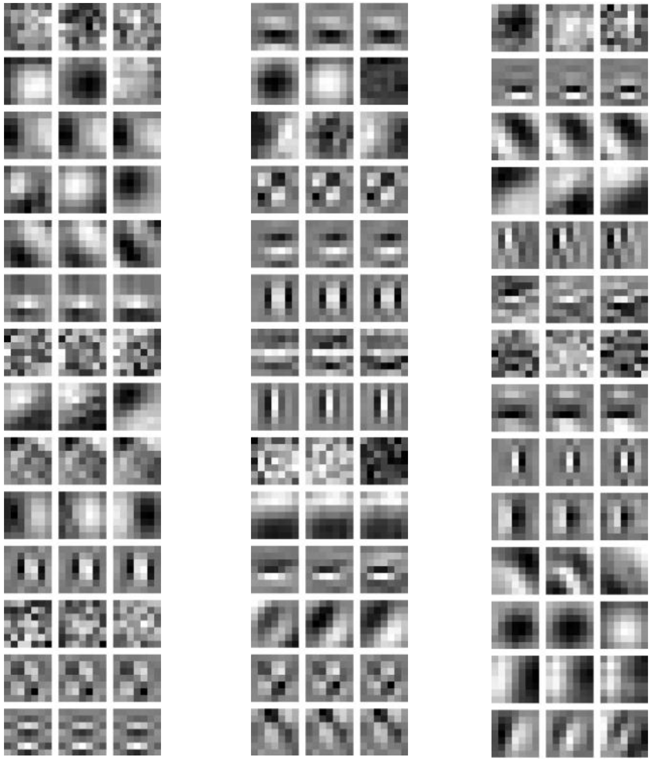
Learning Filters

- Convolution filters

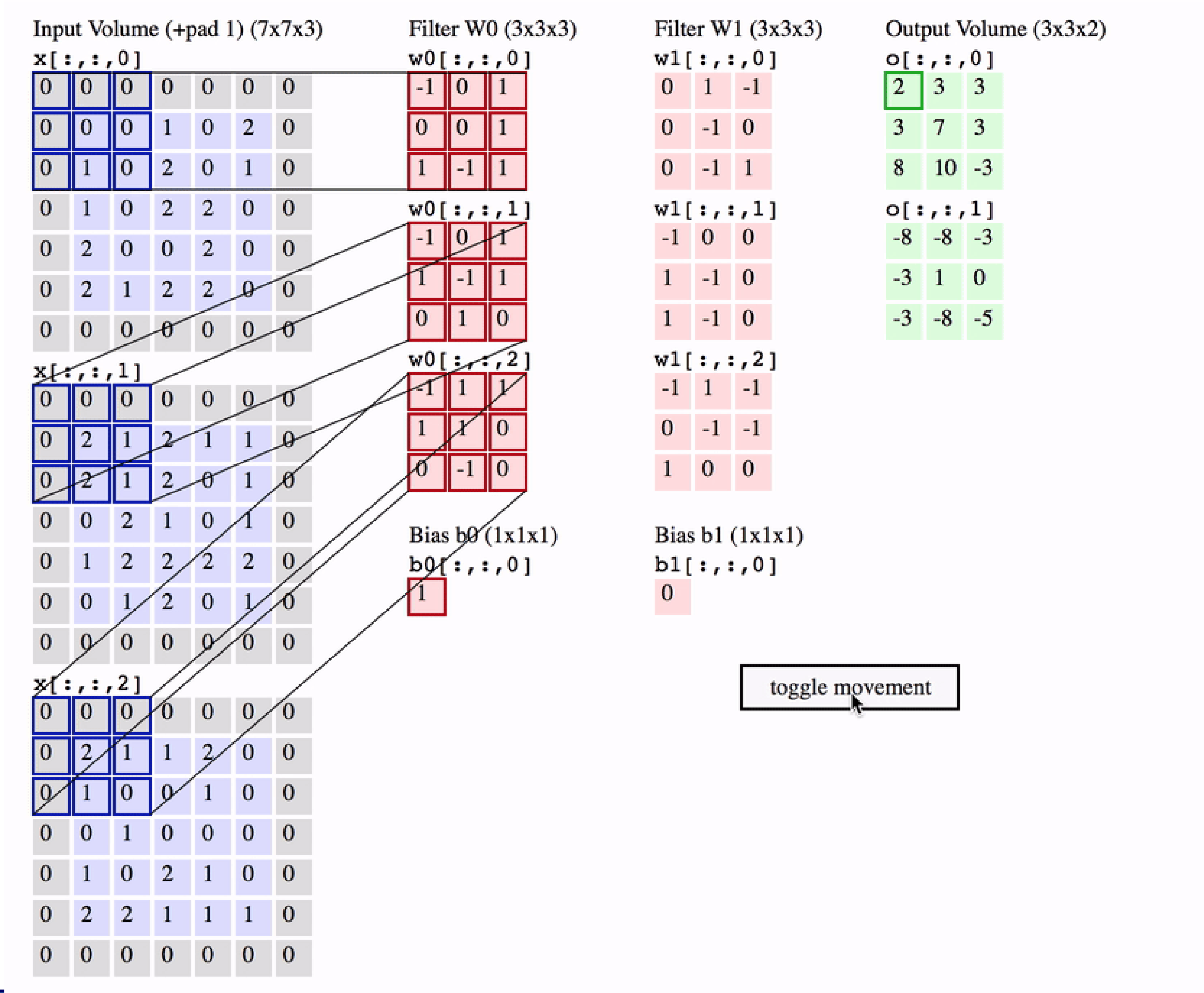


Identity	$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$	
Edge detection	$\begin{bmatrix}1&0&-1\\0&0&0\\-1&0&1\end{bmatrix}$	
	$\begin{bmatrix}0&1&0\\1&-4&1\\0&1&0\end{bmatrix}$	
	$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}$	
Sharpen	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$	

- Filter learning

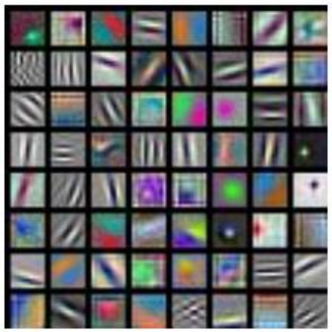


Convolution Layers



Learning Kernels

- Trained kernels of 1st layer

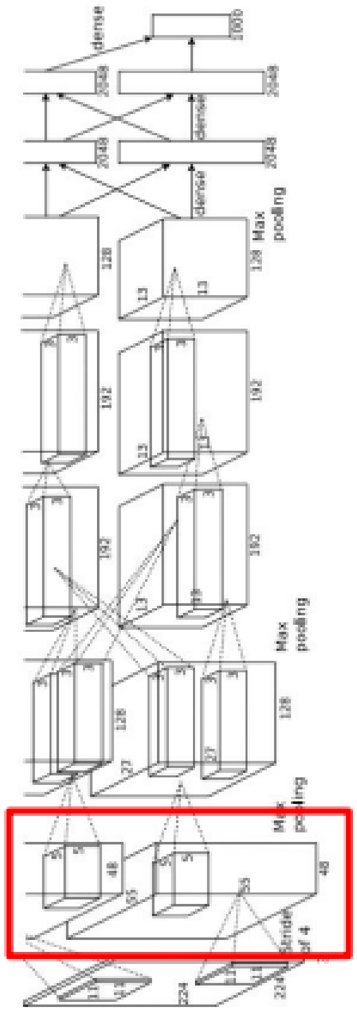


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7



Cf. Gabor functions

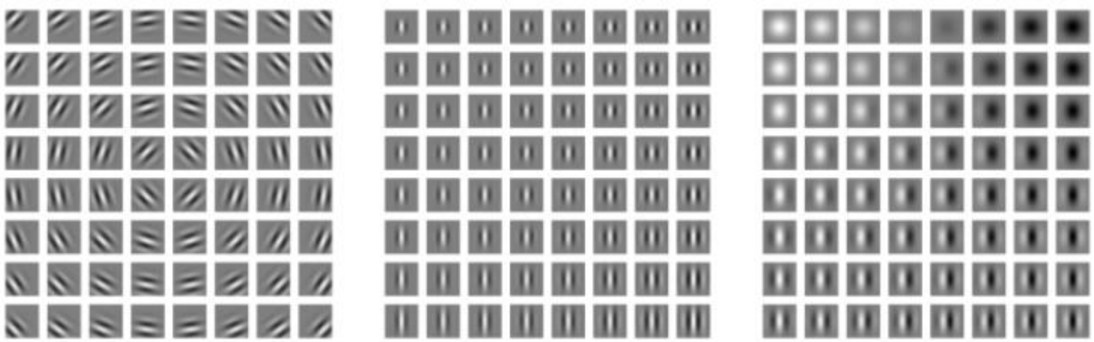


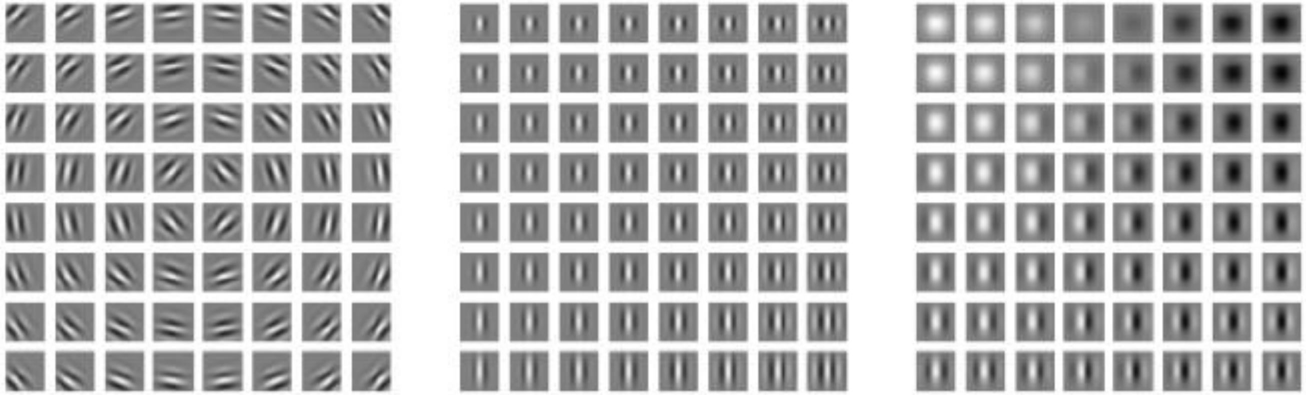
Figure source: Stanford cs231n, The Deep Learning Book

CNN vs. Human Vision

- V1 cells have weights that are described by Gabor functions
  - Response of simple cell

$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x, y) I(x, y)$$

  - $w(x,y)$: Gabor function



Gabor Function

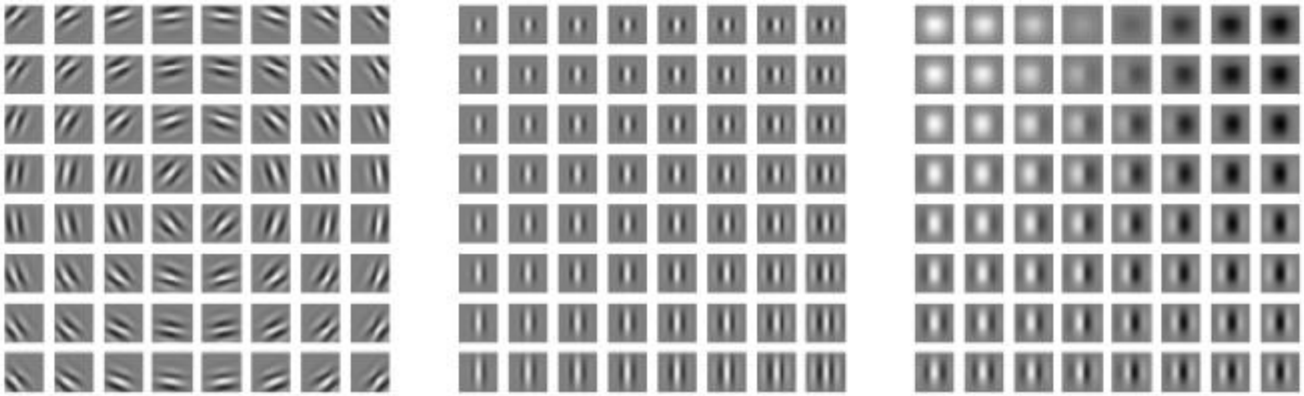
- Gabor Function

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp\left(-\beta_x x'^2 - \beta_y y'^2\right) \cos(f x' + \phi)$$

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

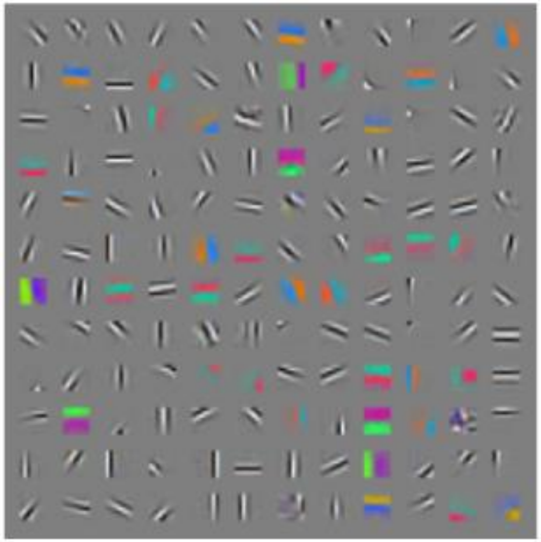
$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau)$$

α, β_x, β_y, f, ϕ, x_0, y_0, and τ are parameters

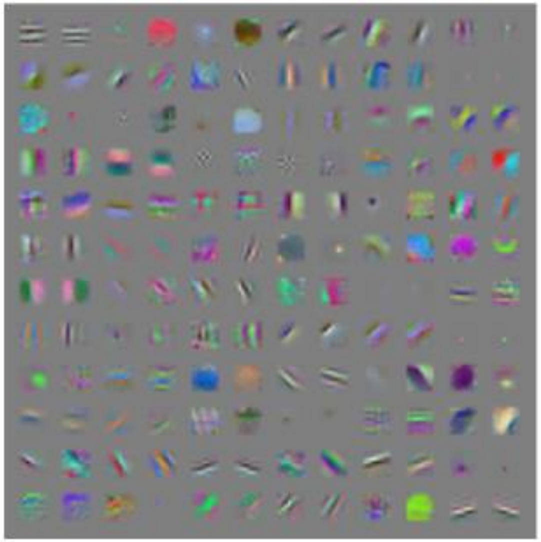


Trained Kernels

- A simple unsupervised learning algorithm, sparse coding, learns features with receptive fields similar to those of simple cells (Gabor-like functions)
- The features learned by machine learning models with those employed by V1



Unsupervised learning



Fully supervised CNN

Convolution Layers

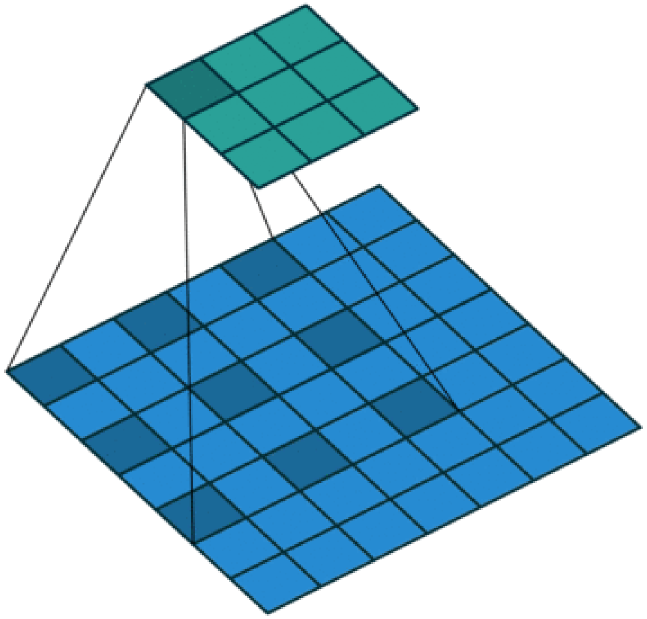
- Propagation

$$X^n_{(p,i,j)} = f\left(\sum_{q \in C^n_p} \sum_{0 \le u,v \le M_n - 1} w^n_{(q,p,u,v)} X^{n-1}_{(q, iS_n + u, jS_n + v)} + \theta^n_p\right)$$

  - q: input plane, p: output plane, M_n: mask width/height
  - C^n_p: # of input planes connected to p^{th} output plane
  - $w^n_{(q,p,u,v)}$: weight at (u,v) on the mask from q^{th} plane to p^{th} plane
  - $X^n_{(p,i,j)}$: feature at (i,j) on p^{th} plane of layer n
  - S_n: stride (horizontal/vertical distance between adjacent windows)
  - θ^n_p: bias

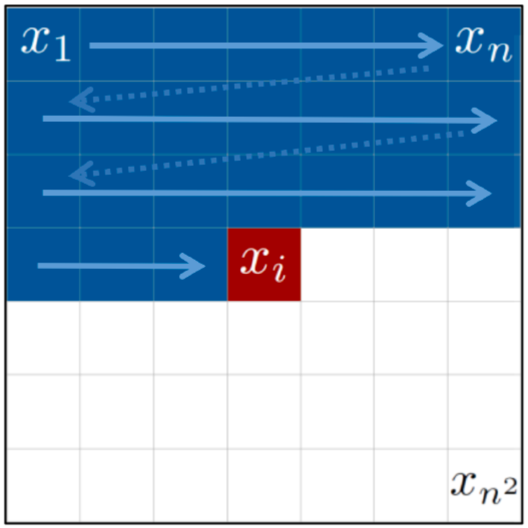
Dilated Convolution

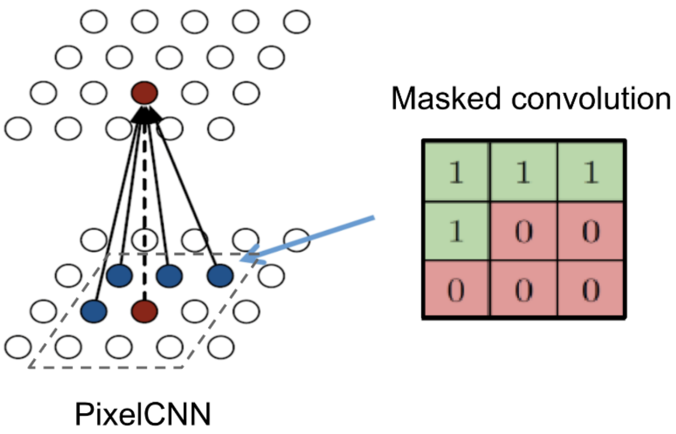
- A.k.a. "atrous convolution"
- Dilated convolutions "inflate" the kernel by inserting spaces between the kernel elements
  - Filter upsampling (reduces computation and parameters)



Masked Convolution

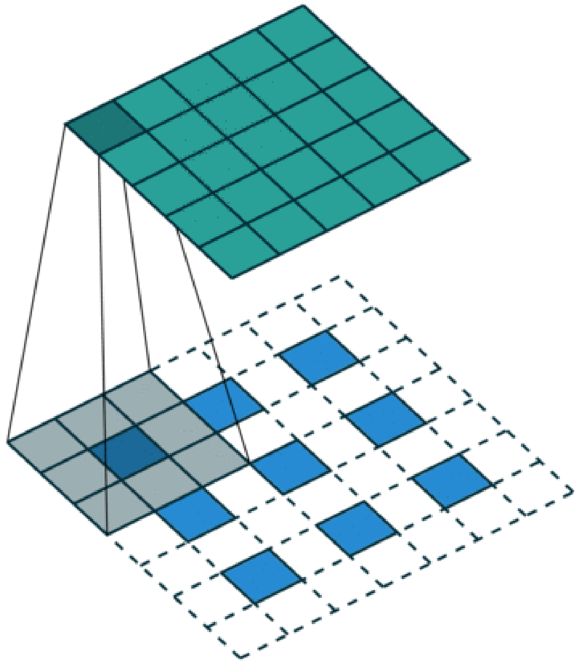
- Convolution on past information
  - Frequently used for autoregressive models (e.g. PixelCNN)





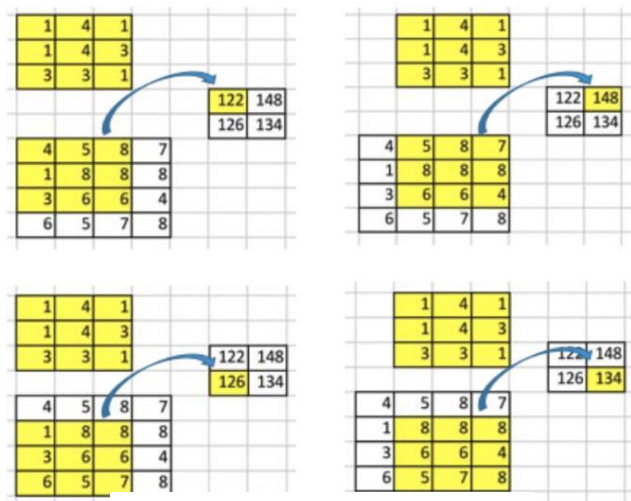
Transposed Convolution

- A.k.a. "fractionally-strided convolution" or "deconvolution(?)"
- Transposed convolutions work by swapping the forward and backward passes of convolution.
  - Feature upsampling

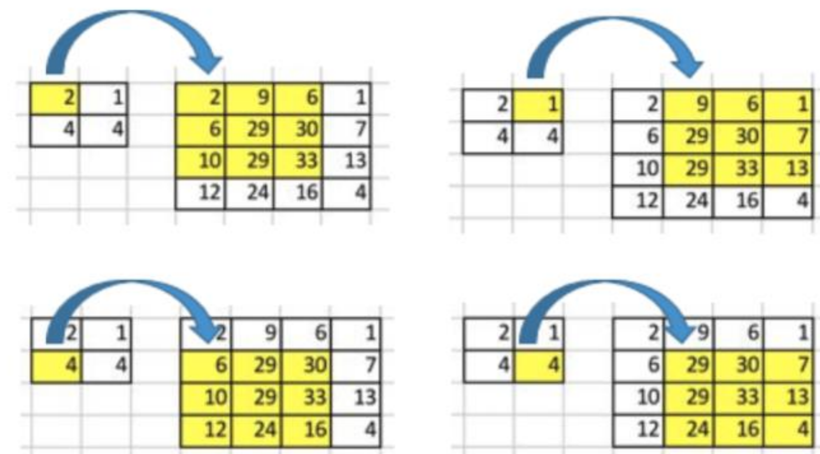


Backward Convolution

- Forward vs. backward convolution

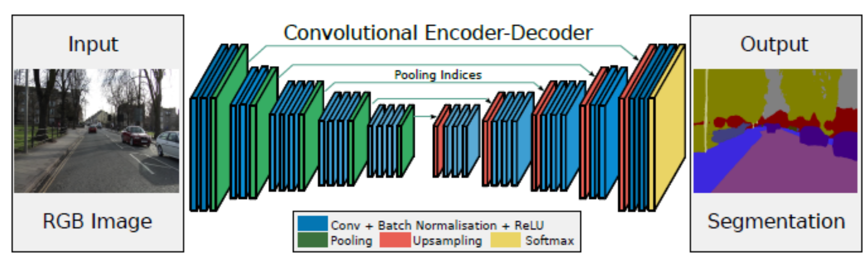


forward convolution



backward convolution

- Back-propagation of convolution layer
- Transposed convolution
  - Up-sampling



Convolution as MatMul

1	2	3
6	5	3
1	4	1

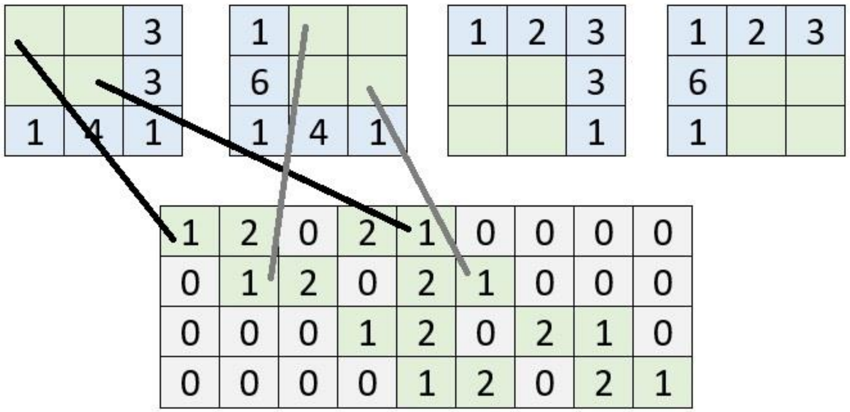
3x3 Input

1	2
2	1

2x2 Kernel

22	21
22	20

2x2 output



$$\begin{bmatrix} 1 & 2 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \\ 6 \\ 5 \\ 3 \\ 1 \\ 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 22 \\ 21 \\ 22 \\ 20 \end{bmatrix}$$

Transposed Convolution as MatMul

1	2
2	1

2x2 Kernel

1	2
2	4

2x2 input
(to add padding)

1	4	4
4	13	10
4	10	4

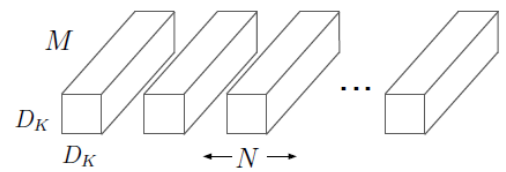
3x3 output

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\times
\begin{bmatrix} 1 \\ 2 \\ 2 \\ 4 \end{bmatrix}
=
\begin{bmatrix} 1 \\ 4 \\ 4 \\ 4 \\ 13 \\ 10 \\ 4 \\ 10 \\ 4 \end{bmatrix}
=
\begin{bmatrix} 1 & 4 & 4 \\ 4 & 13 & 10 \\ 4 & 10 & 4 \end{bmatrix}
$$

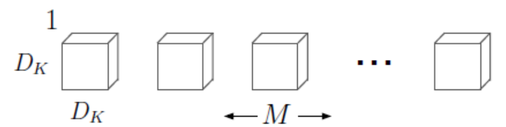
Separable Convolutions

- Separable convolutions consists of two consecutive convolution operations.
  - Depth-wise convolution + point-wise convolution
  - Reduce computation and parameters

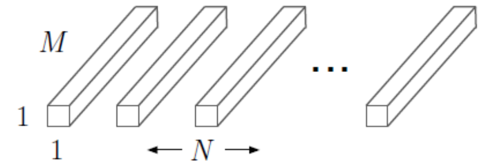
- Computation
  - Standard convolution
    - $D_K D_K M N D_F D_F$
  - ➔ Depthwise convolution
    + Pointwise convolution
    - $D_K D_K M D_F D_F + M N D_F D_F$



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

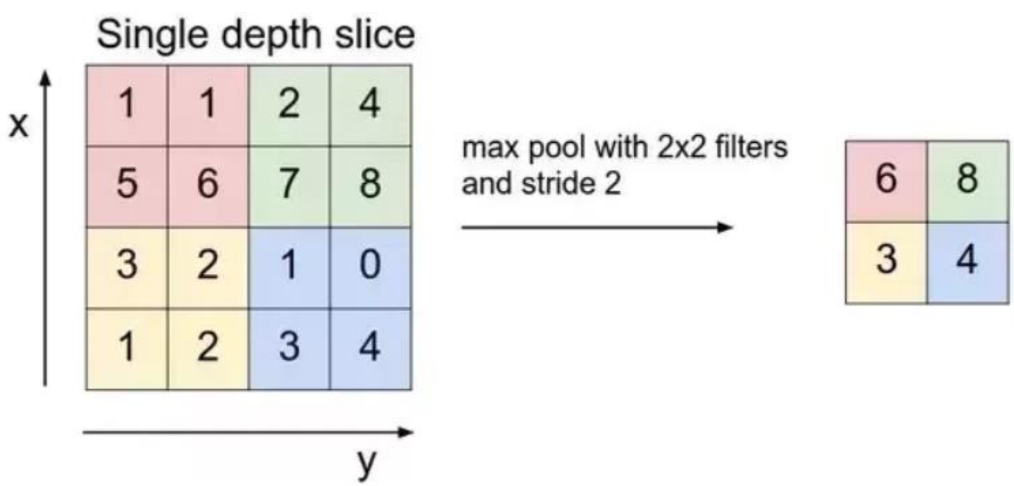


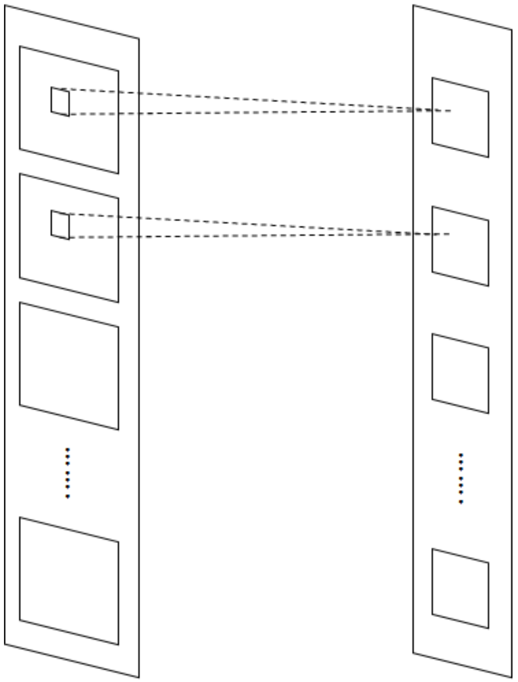
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Max-Pooling Layers

- Max-pooling layers
  - Reduces feature dim.
  - Reduces positional variation

$$X^{n}_{(p,i,j)} = f\left(\max_{0\le u,v\le M_n-1} X^{n-1}_{(p,iS_n+u,jS_n+v)}\right)$$

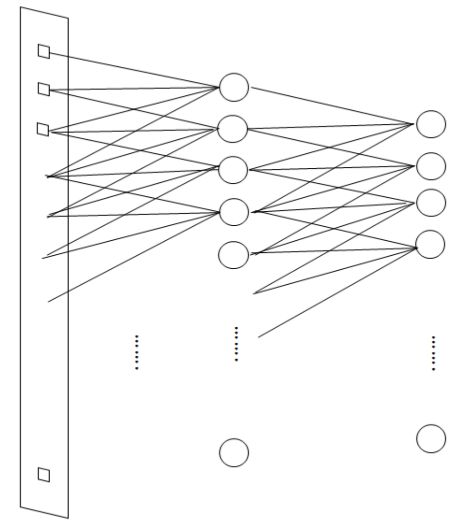




Fully-connected Layers

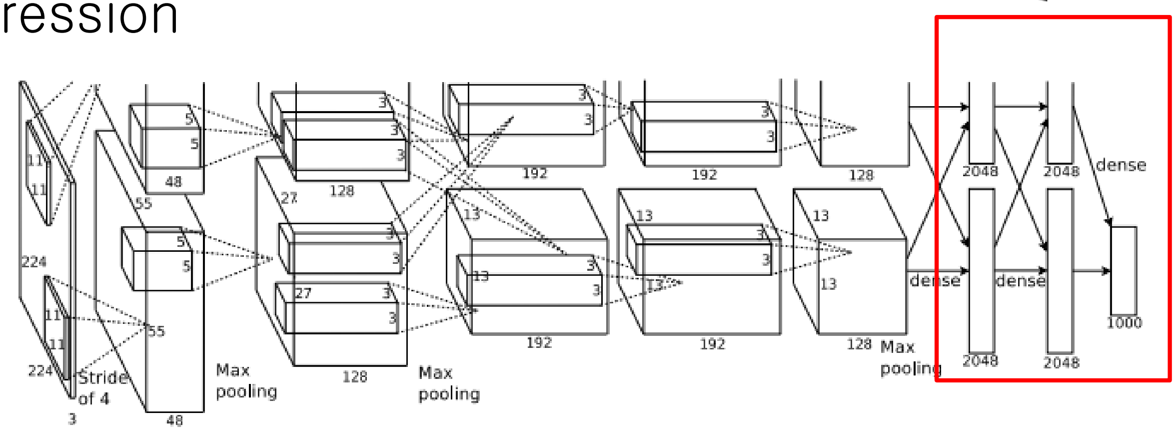
- Propagation

$$X_p^n = f\left(\sum_q w_{(q,p)}^n X_q^{n-1} + \theta_p^n\right)$$



➔ Perform target task with the feature extracted by previous layers

- Classification
- Regression



Why CNN Works Well?

- Effective in learning high-level representation
- Good at catching 2D structures
  - Convolution layers are effective in learning 2D features
  - Tolerates shape variation by pooling and abstraction
- Network structure that less suffers from vanishing gradient problem and overfitting
  - Parameter sharing, sparse connection
- Flexible structure
- Easy to parallelize
- Recent issues: limited context compared to RNN
  - Remedy: non-local neural nets

CNN in PyTorch

- Defining a CNN

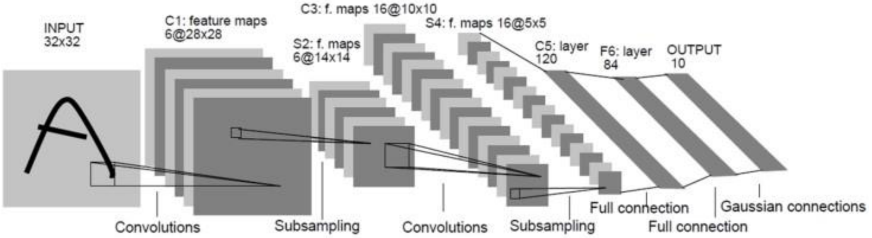
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Names	Layer Types	Hyper-parameters
conv1	convolution	1 -> 6, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
conv2	convolution	6 -> 16, mask = 5x5
pool	max-pooling	window = 2x2, stride = 2
fc1	fully-connected	400 (16 * 5 * 5) -> 120
fc2	fully-connected	120 -> 84
fc3	fully-connected	84 -> 10



CNN in PyTorch

- Defining Loss function and Optimizer

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

  - Cross entropy (with softmax activation)
    - Softmax activation: $X_c^N = \frac{\exp(net_c^N)}{\sum_c \exp(net_c^N)}$

$$E_{CE} = -\sum_c d_c \log(X_c^N)$$

  - Stochastic gradient descent

$$W^{t+1} = W^t + \Delta W^t$$

$$\Delta W^t = -\eta \frac{\partial Loss}{\partial W^t} + m\Delta W^{t-1}$$

CNN in PyTorch

- Training

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)                  # feed input to network
        loss = criterion(outputs, labels)      # compute loss function
        loss.backward()                        # compute gradients
        optimizer.step()                       # update weights

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:                   # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0
```

Deep Learning Issues

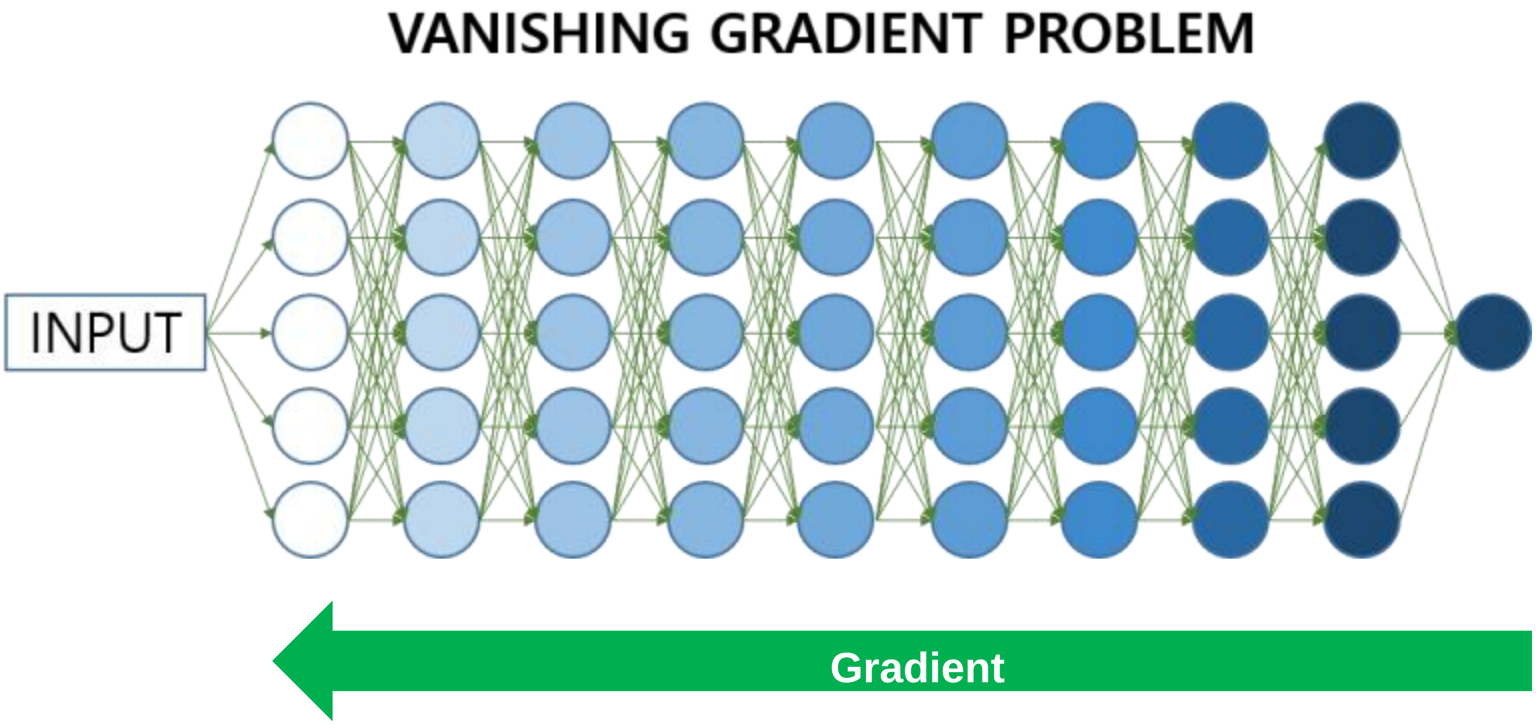
Challenges in Deep Learning

- Difficulties in deep learning
  - Backpropagation algorithm does not work or slow
  - Not better than shallow networks

- Why?
  - The vanishing/exploding gradient problem
  - Local minima, saddle points, plateaus
  - Overfitting
  - Internal covariate shift [Ioffe15]
  - Scattered gradient problem [Balduzzi17]
  - Many unknown reasons

Vanishing Gradient Problem

- Conventional back-propagation algorithm does not work well for deep networks.

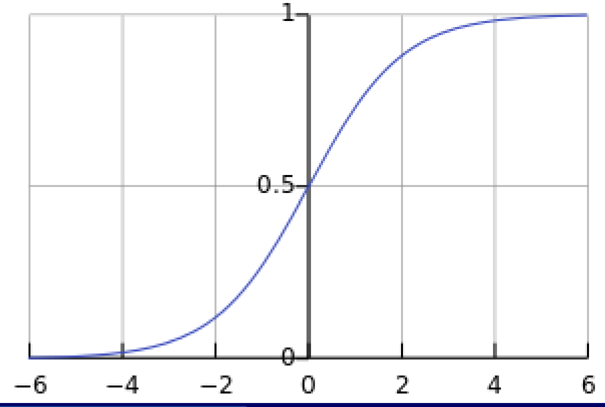


Vanishing Gradient Problem

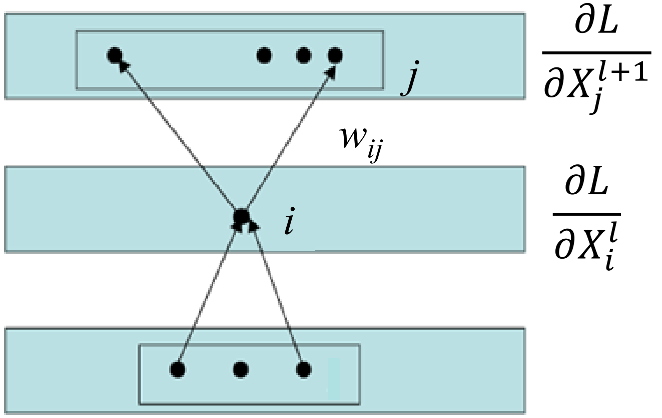
- Conventional back-propagation algorithm does not work well for deep networks.
  - Backpropagation formula

$$\frac{\partial L}{\partial X_i^l} = \frac{\partial L}{\partial X_i^{l+1}} \frac{\partial X_i^{l+1}}{\partial X_i^l} = f'(net_j^{l+1}) \sum_j w_{ij}^{l+1} \frac{\partial L}{\partial X_j^{l+1}}$$

Saturated regime of
Activation functions



Blended gradient

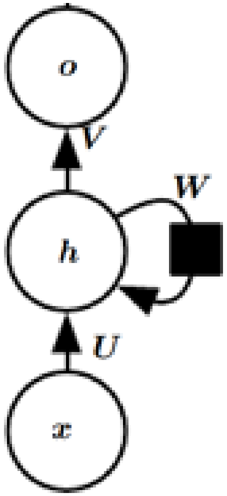


Vanishing/Exploding Gradient on RNN

- One step propagation on RNN

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^\top \boldsymbol{h}^{(t-1)}$$

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^t\right)^\top \boldsymbol{h}^{(0)}$$



- Eigen decomposition of *W*

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^\top \boldsymbol{\Lambda}^t \boldsymbol{Q} \boldsymbol{h}^{(0)}$$

- Gradients propagated over many stages tend to either vanish ($|\lambda_i|<1$) or explode ($|\lambda_i|>1$)

Solutions of Vanishing Gradient Problem

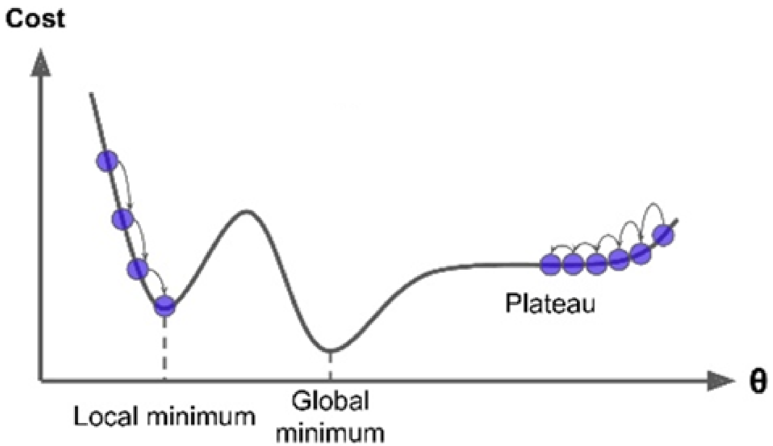
- Layer-wise unsupervised pre-training
  - DBN, stacked auto-encoders

- Architectures to avoid vanishing gradient problem
  - Convolutional neural networks (CNN)
    - Sparse connection, shared weights
  - Gated units (LSTM, GRU, GLU)

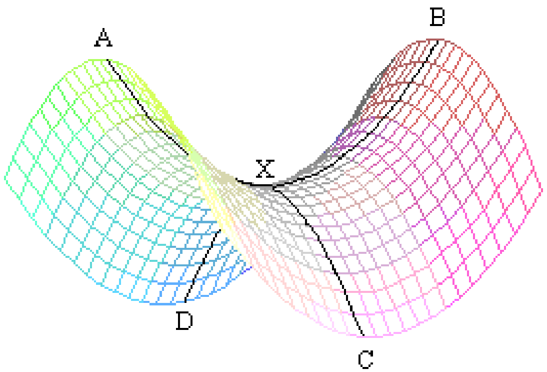
- Improved structures and learning algorithms
  - Piece-wise linear activation functions
    - max-out, ReLU, LReLU, PReLU, ELU, etc.…
  - Skip connection (ResNet, DenseNet, DPN)
  - Batch normalization
  - Xavier initialization, He initialization, LL-initialization
  - Transfer learning, multi-task learning
  - Auxiliary networks, deeply supervised network

Local Minima, Saddle Point, Plateau

- Learning sometimes stops at local minima, saddle points or plateaus
  - Small networks: local minima is major issues
  - Large networks: plateau or saddle points are major issues



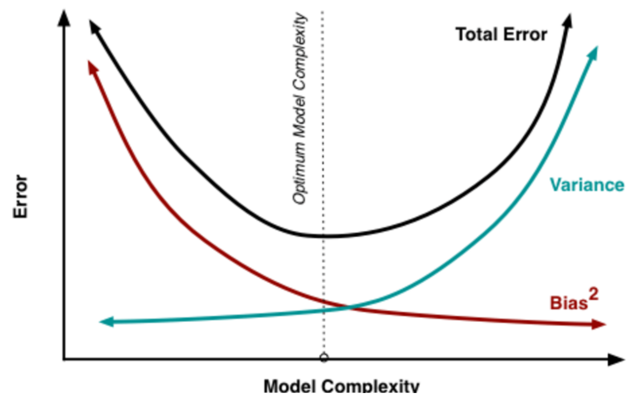
Local minima / Plateau

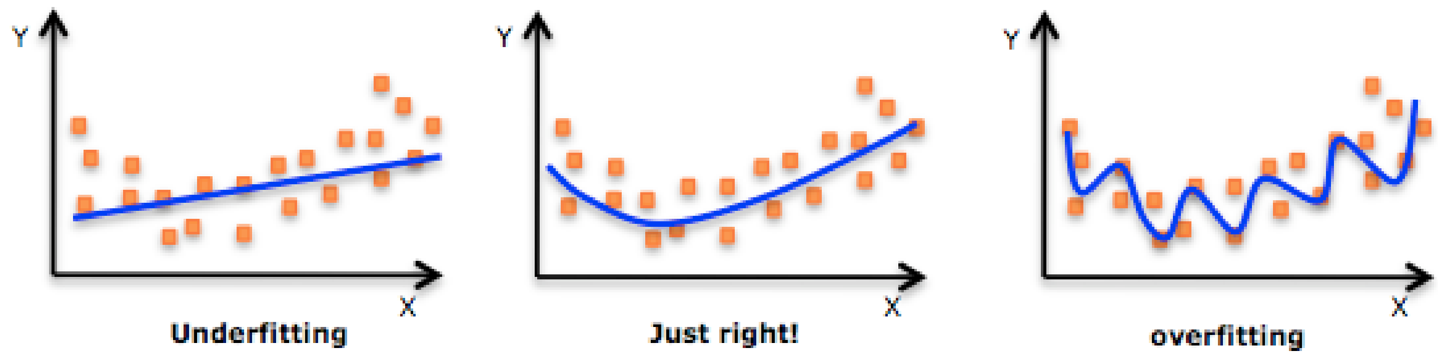


Saddle point

Overfitting

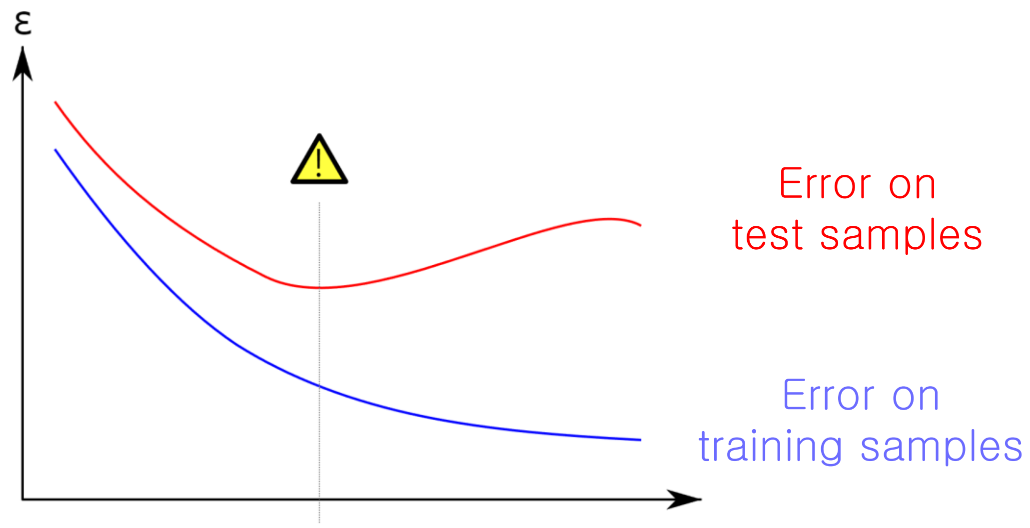
- Overfitting occurs when a model is excessively complex relative to the number of observations.
  - Large capacity model + insufficient data





Overfitting

- Large gab between training and test accuracy



- Remedies
  - More data or simpler model
  - Regularization, transfer learning, batch norm, dropout, etc.

Generalization of Deep Networks

- Traditional knowledge
  - Model with too large capacity does not generalize well

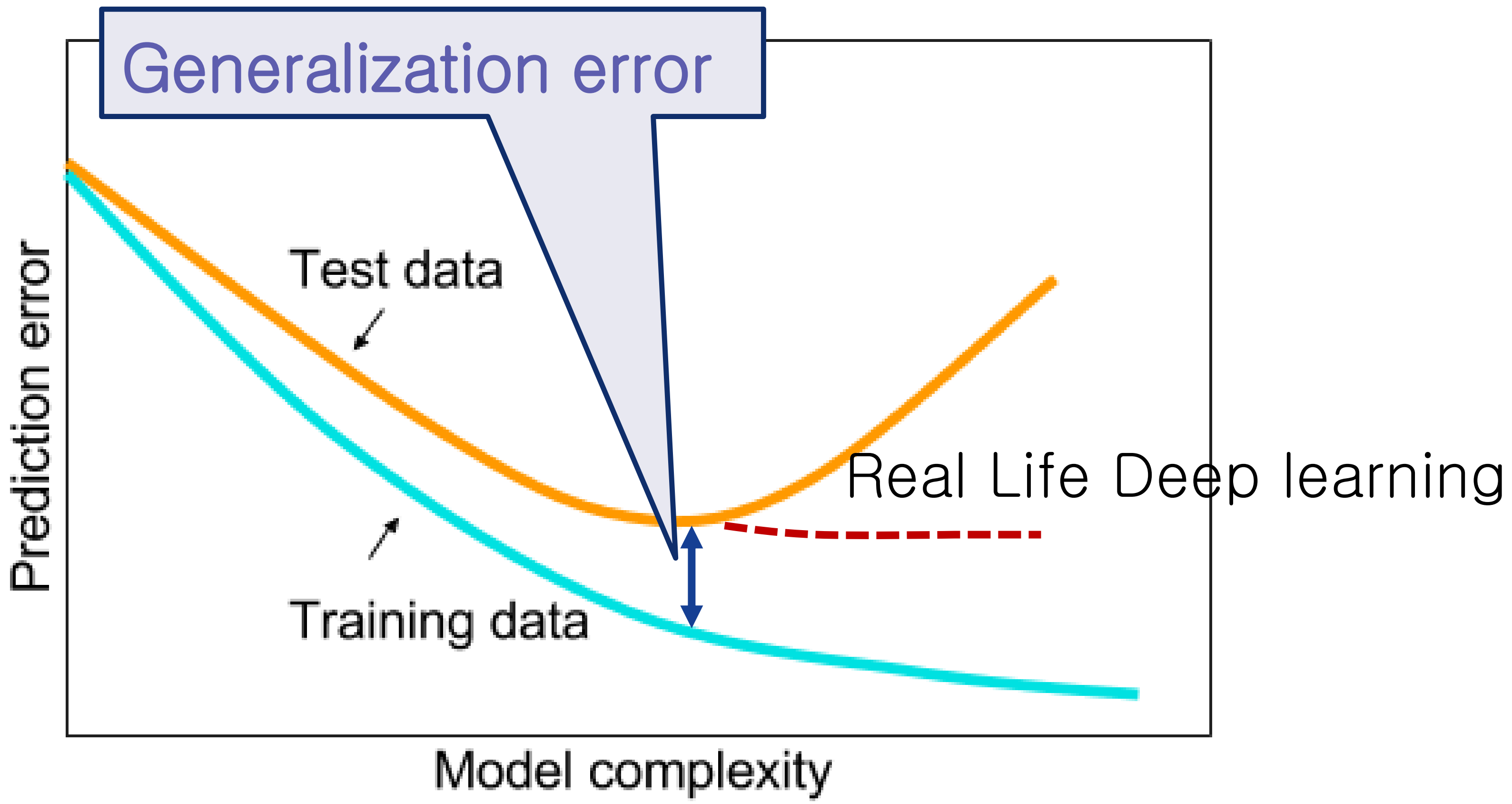
- New observations in deep learning
  - Network depth helps improve generalization
  - Many huge networks generalize well.
    - Train VGG19 (20M parameters) on CIFAR10 (50K samples)

  ➔ Generalization of deep networks is not explainable with conventional knowledge

- Current trend: powerful model + additional techniques
  - Regularization techniques
  - Data augmentation
  - Unsupervised pretraining / semi-supervised learning

Overfitting in Deep Learning

- In deep learning, over-parameterization is often successful



Regularization

- Introduce additional information to solve ill-posed problems or reduce overfitting

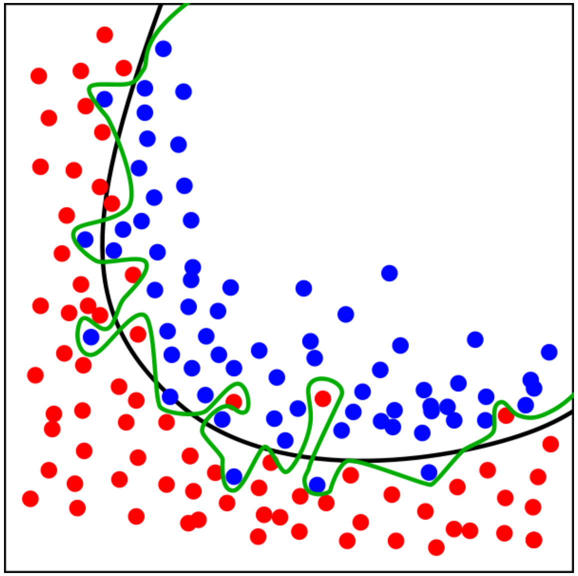
- Add regularization term to loss function

$$E(W) + \lambda\|W\|$$

  - $E(W)$: main loss function
  - λ: regularization factor
  - $\|W\|$: norm of W
    - L2-norm is more popular
    - L1-norm is used for some models (e.g. sparse autoencoder)

- Related topics
  - Support vector machines
  - Prior probability

[Wikipedia]



Q&A

Thank you
for your attention!