Convolutional Neural Networks

CS5242

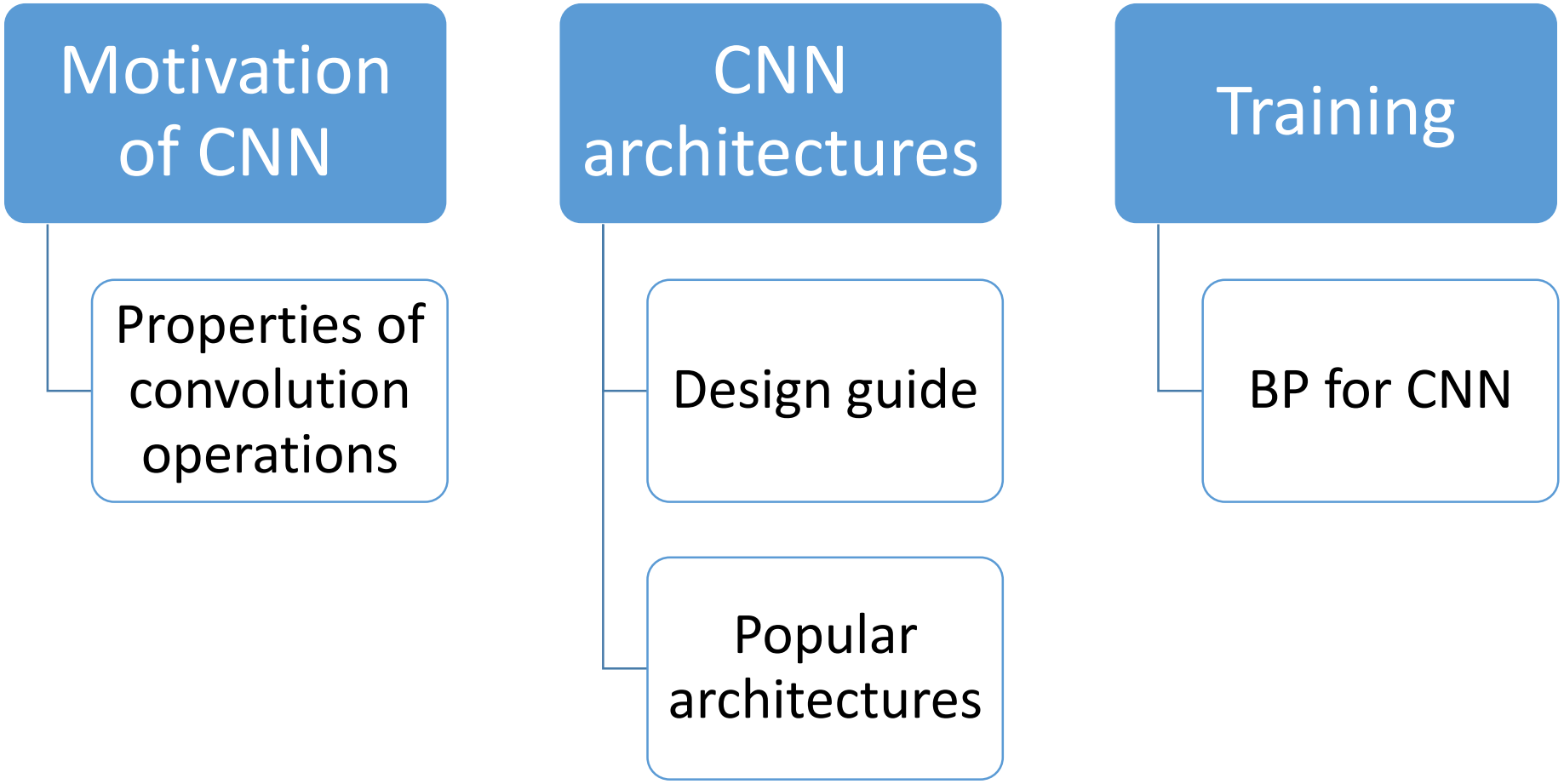
Lee Hwee Kuan & **Wang Wei**

Teaching assistant:

Connie Kou Khor Li, Ji Xin, Ouyang Kun

cs5242@comp.nus.edu.sg

Roadmap


Motivation of CNN
Properties of convolution operations
CNN architectures
Design guide
Popular architectures
Training
BP for CNN

Intended learning outcome

01

Know the properties of CNN and its advantages compared with MLP

02

Understand the similarities and difference of different architectures

03

Implement BP of CNN for image classification

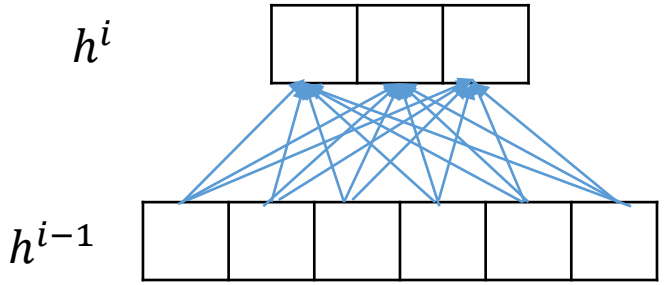
Motivation of CNN

Why

- Why do we use CNN?
  - Good performance!
- Why CNN has good performance?
- Where do we use CNN?
  - Mainly for compute vision

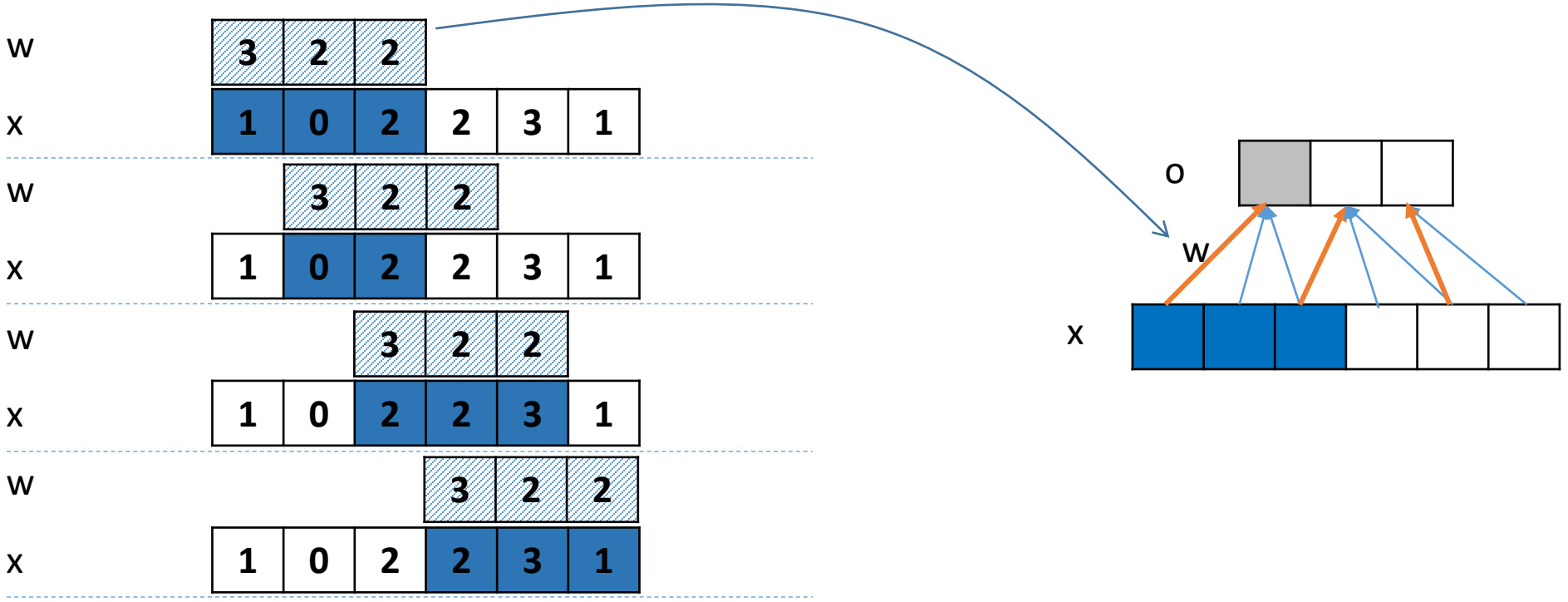
Recap

- MLP
  - Universal approximator
    - Many hidden neurons
    - >= One single hidden layer
  - $h^i = relu\left(h^{i-1}W^i + b^i\right)$
  - $W^i \in R^{|h^{i-1}|\times|h^i|}, b^i \in R^{|h^i|}, h^0 = x$
  - Problems?
    - Fully connected → many parameters
    - Hard to optimize without any constraint/regularization
      - To find a good mapping (from input to prediction)



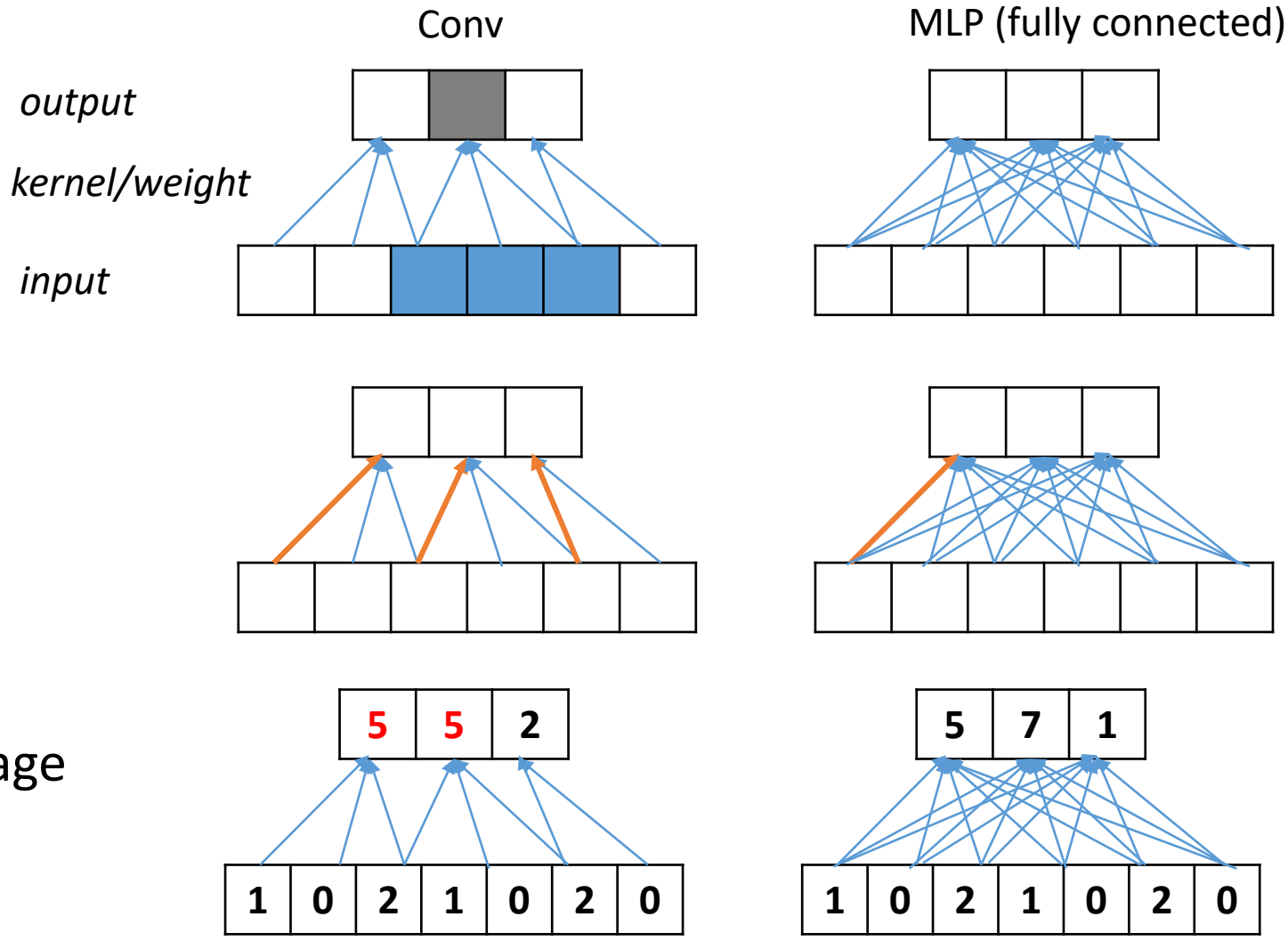
Recap

- Convolution



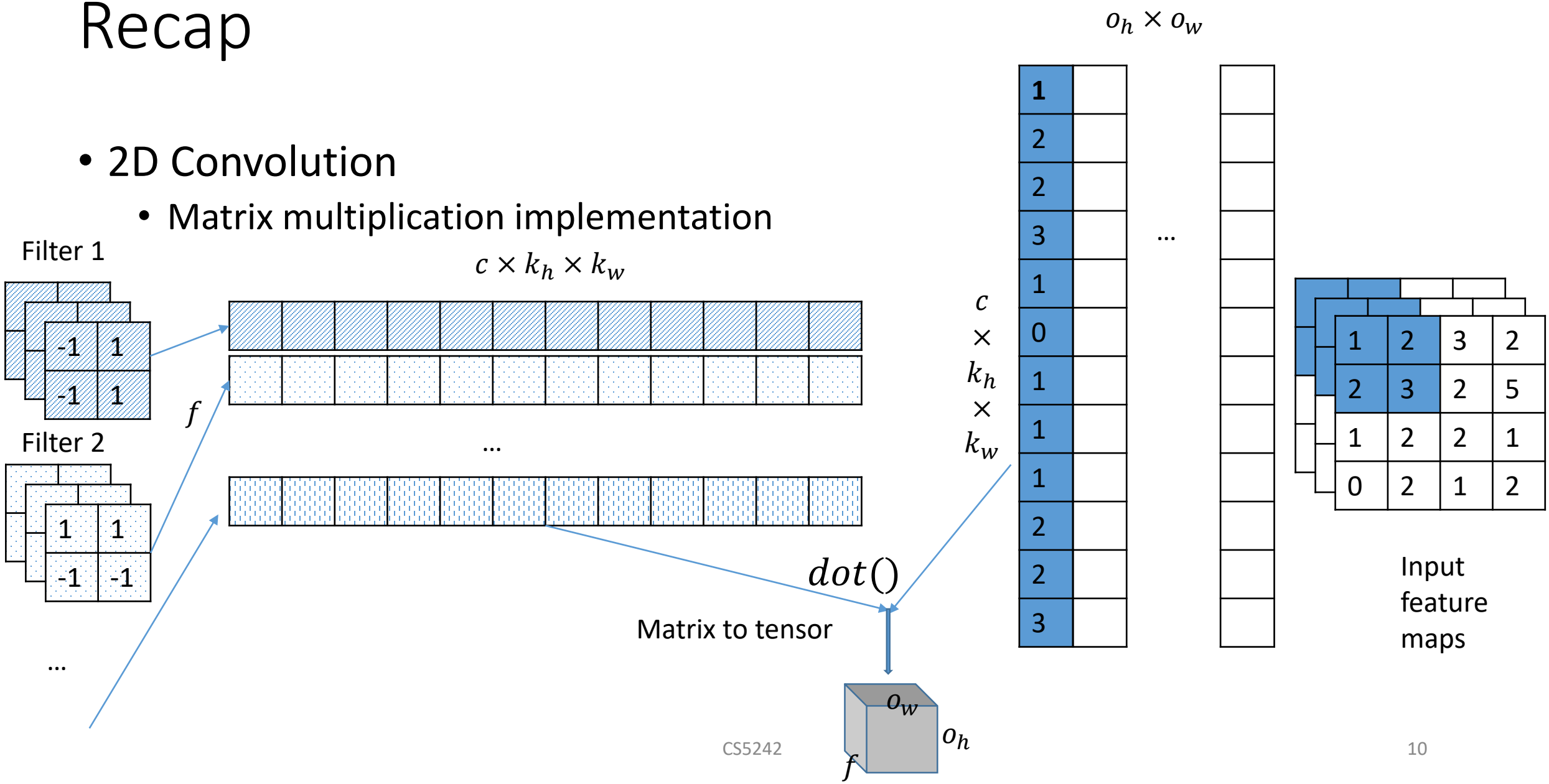
Properties (Why Convolution better?)

- Sparse connection
  - Fewer parameters
  - Less overfitting
- Weight sharing
  - Regularization
  - Less overfitting
- Location invariant
  - Robust to object position in the image



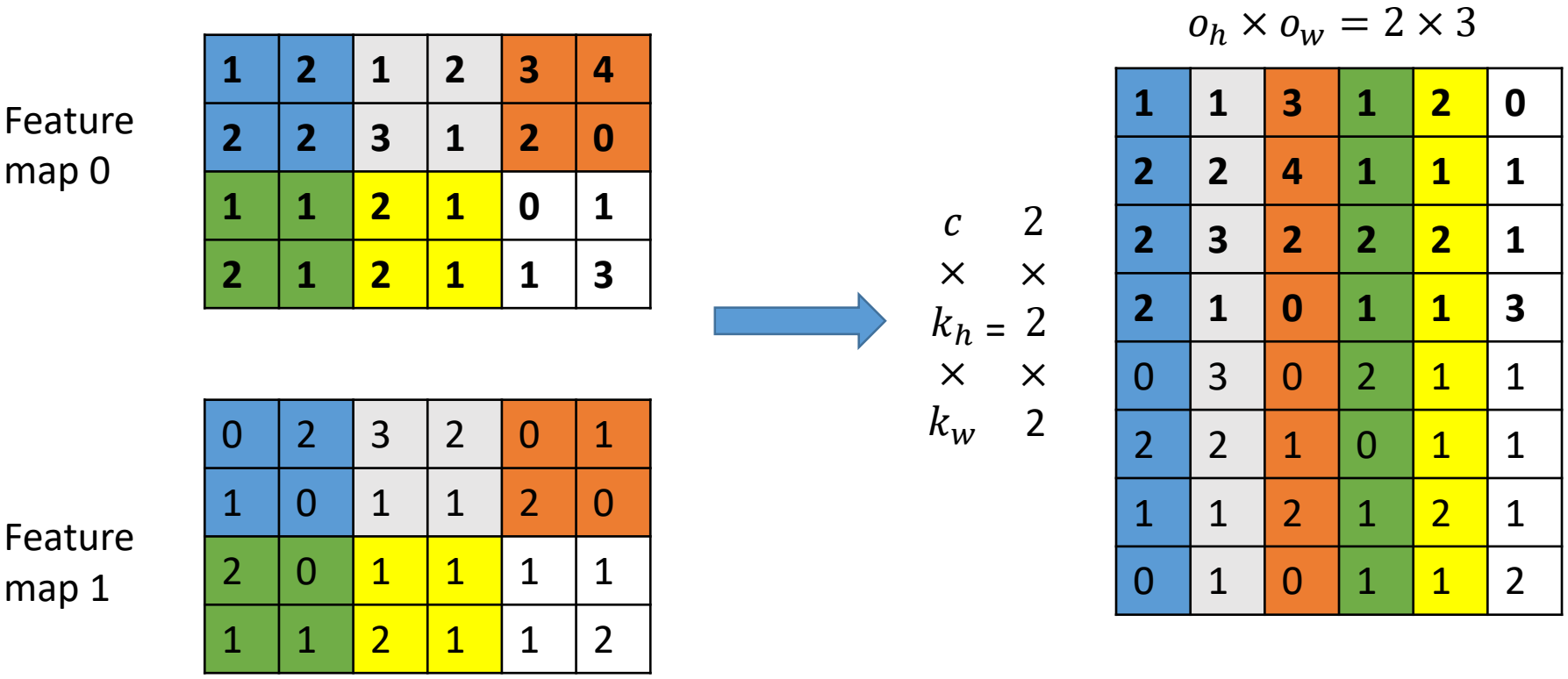
Recap

- 2D Convolution
  - Matrix multiplication implementation



Recap

- 2D Convolution
  - Img2col: receptive fields across feature maps are concatenated into to one column



Recap

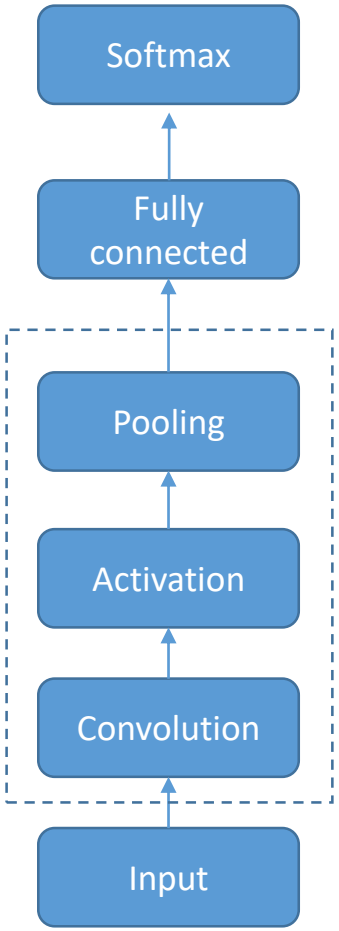
- 2D Convolution
  - Create filter matrix
    - W=np.random.rand(f, c*k*k)
  - Forward
    - Convert input feature maps into matrix X (img2col)
    - $Y = WX$
  - Backward
    - Given $\frac{\partial L}{\partial Y}$
    - Compute $\frac{\partial L}{\partial W}$= $\frac{\partial L}{\partial Y} X^T$, $\frac{\partial L}{\partial X}$=$W^T \frac{\partial L}{\partial Y}$

CNN architectures

Design guide

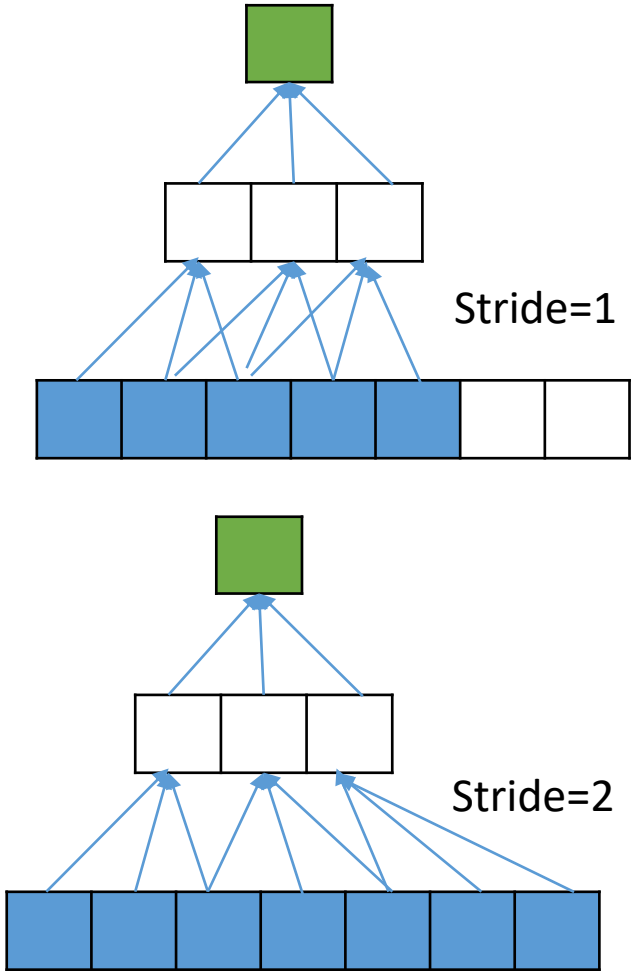
- Faster
  - Converge faster (Numeric optimization, SGD)
    - Learning rate
    - Gradient
  - Compute faster
    - Computation complexity
- More accurate
  - Large representation capacity -> overfitting
  - Generalize better

Basic architecture



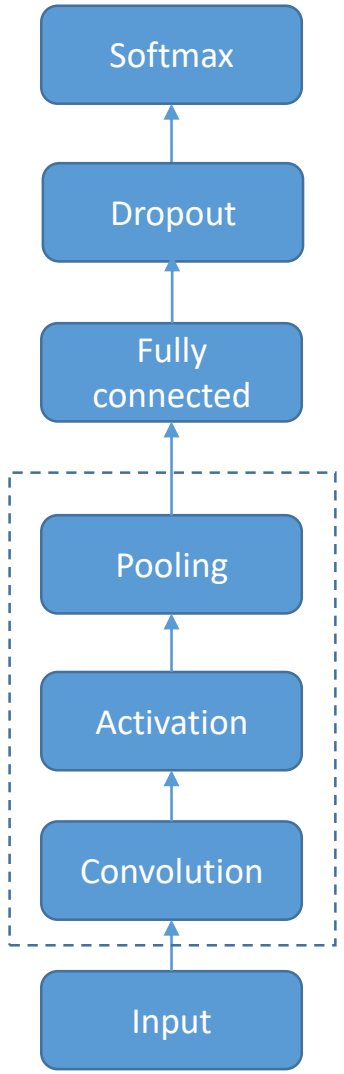
Basic architecture

- Convolution and non-linear activation
  - If no activation
    - collapse all conv layers into one
    - $h_1 = W_1 x, h_2 = W_2 h_1, h_3 = W_3 h_2 \ldots$
    - → $h_k = W_k W_{k-1} \ldots W_1 h_{k-1}$
  - ReLU [6]
    - Avoid gradient vanishing problems caused by sigmoid
    - Easy to optimize → converge faster, better
- Pooling layer
  - Subsampling
    - Stride > 1
    - Increase the receptive field of each neuron
    - == Convolutional layer with stride > 1
- Repeat the C-A-P block to increase the depth
  - Increase the model capacity → better accuracy



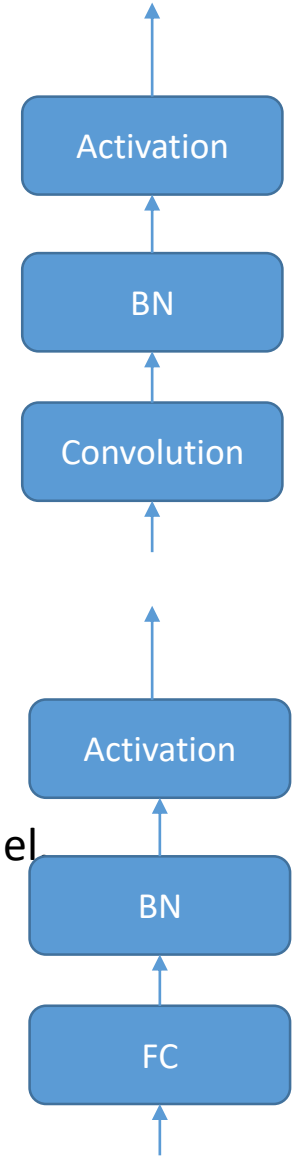
Basic architecture

- Fully connected layer
  - Convert the feature vector into a specified length
    - e.g. of labels
  - Optional
    - Replaced by 1x1 convolutional layer with # filters = # labels
- Softmax layer
  - For classification problems
  - Generate a probability for each candidate label



Basic architecture

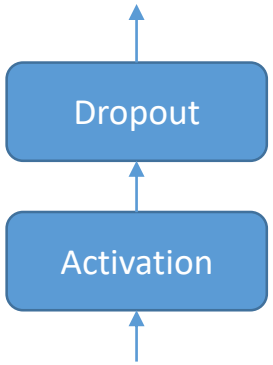
- Normalization layer
  - To avoid numeric exploding
  - Make training more stable and converge faster
  - BatchNorm (BN) [7]
    - Normalize h_k to avoid internal covariate-shift
    - Applied after convolution/fully connected before activation
      - After convolution layer. Compute mean and var per channel; one (γ, β) per channel
    - Running smooth to get global mean and var for inference
    - Improves gradient flow through the network
      - Allows larger learning rate;

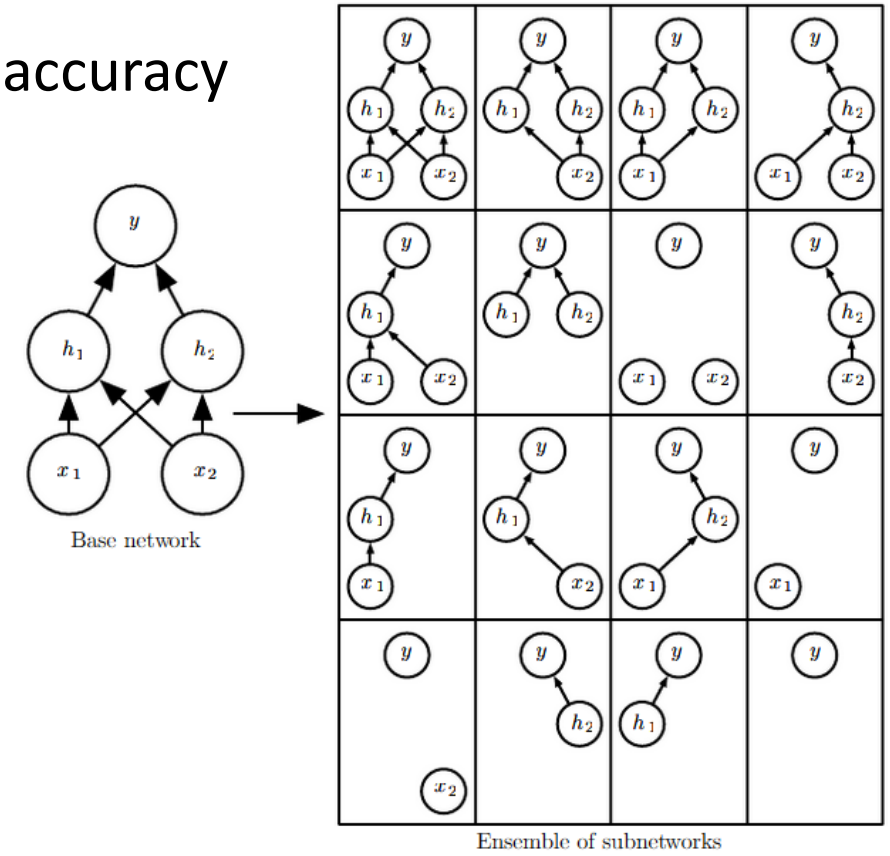


$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \qquad y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$

Basic architecture

- Dropout
  - Model ensemble→ generalize well → improve accuracy
  - Dropout ratio p
    - Training
      - set each neuron to 0 with probability p
      - Multiply each neuron with 1/(1 – p)





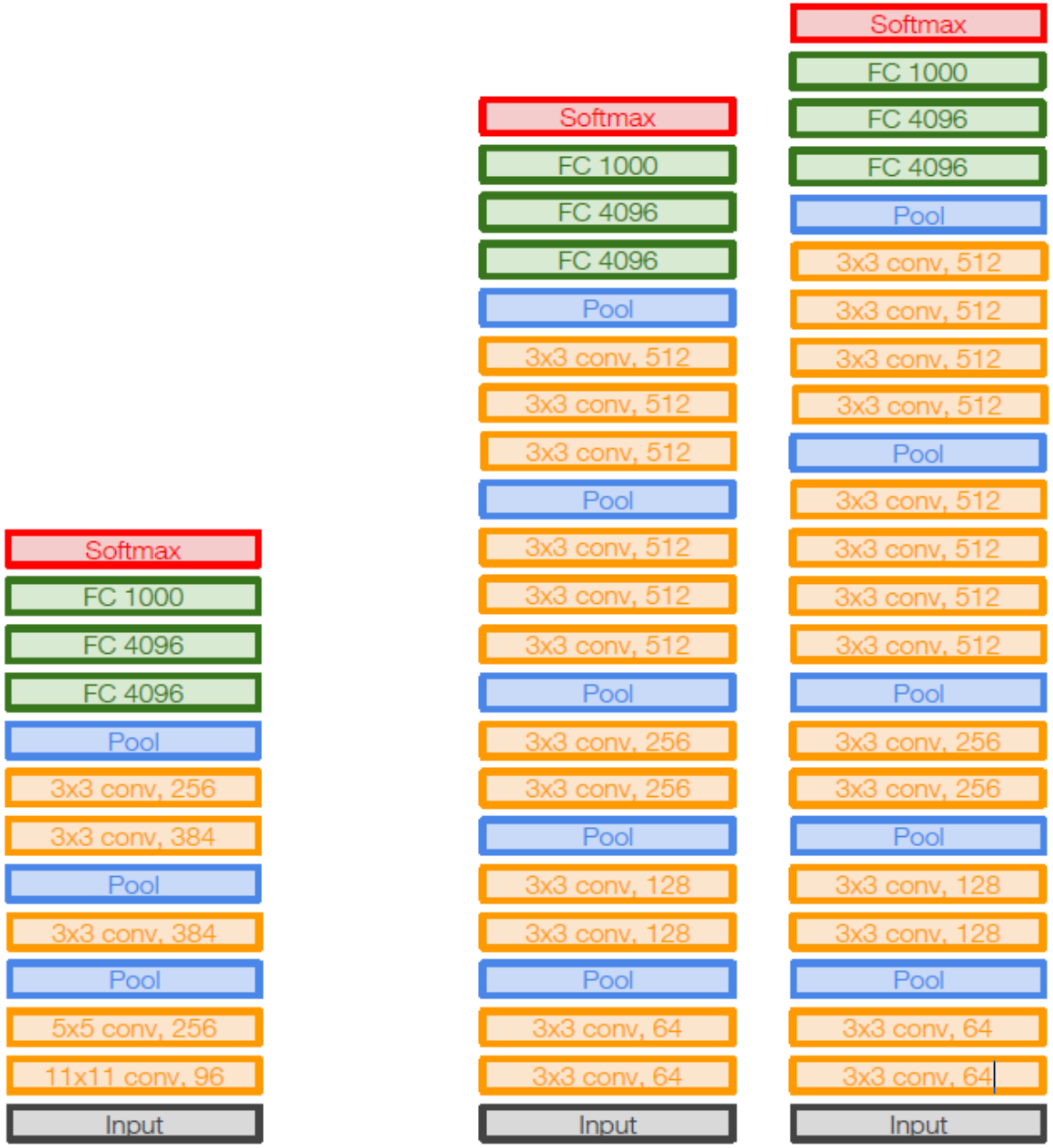
AlexNet 2012 [7]

- http://ethereon.github.io/netscope/#/preset/alexnet
- AlexNet
  - Original
    - two GPUs, filters of some conv layers are separate into two groups
    - # filters = 96-> 48, 48
  - Caffe implementation
    - One GPU, a single filter group
    - # filters = 96
  - Similar performance
    - 18% error, 7 CNN ensemble: 18.2% -> 15.4%
- Characteristics
  - 5 convolutional layers (Deep architecture)
    - bigger model size (parameters) -> large capacity
    - large number of filters (96, 256, 384, 384, 256)
    - Mixed kernel size, 11x11, 5x5, 3x3; stride size, 4x4, 1x1;
  - Dropout → model ensemble → generalize well → improve accuracy
  - ReLu → avoid gradient vanishing

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

From cs231n

VGG 2014 [8]



From cs231n

VGG 2014

- http://ethereon.github.io/netscope/#/preset/vgg-16
- Characteristics
  - Deeper structure (16, 19 convolution layers)
    - More model parameters compared with AlexNet
      - 130+ Million VS 60 Million
    - More non-linear transformation; Large capacity
  - Unified kernel size
    - 3x3 ?
    - Stacking 2 (or 3) 3x3 conv layers == 5x5 (7x7) kernel size in terms of receptive field size
    - Computationally cheaper
      - $f \times (c \times k \times k) \times (h \times w)$, k=3,5,7
      - Run faster
  - 2nd of 2014 classification task,
  - Widely used for other applications, via transfer learning (FC7 generalize well)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Conv layers fewer parameters, most computation

FC layer most parameters, less computaiton

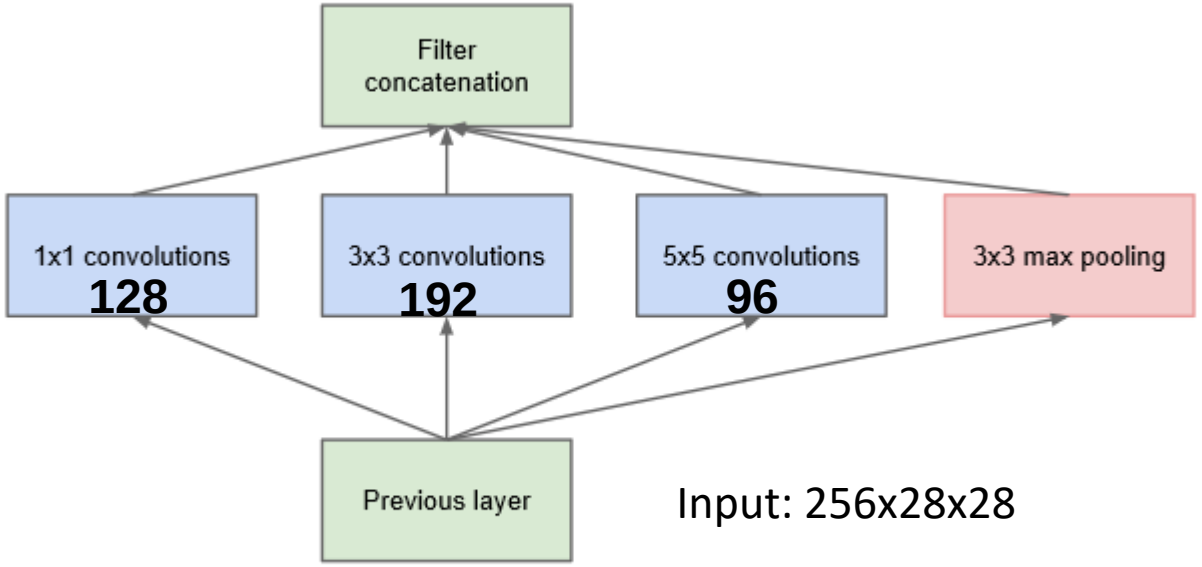
From cs231n

InceptionNet 2014 [9]

- http://ethereon.github.io/netscope/#/preset/googlenet
- Characteristics
  - Inception block
    - More complex structure compared with AlexNet and VGG (single path)
      - One block has multiple paths, each using different conv kernel size, #filters
    - Includes complex feature transformation→ Larger capacity
  - Average pooling
    - Receptive filed size = feature map size
      - Output feature map size = 1x1
    - Remove fully connected layers
      - Reduce model size → less overfitting
      - Reduce time complexity
  - 5 million parameters, 1st 2014 classification task, 6.7% error (top-5)
  - 22 layers

InceptionNet 2014

- Each inception block
  - 1x1, 3x3, 5x5 kernels
    - avoid co-adaption as all branches have different kernel sizes
  - Wide block

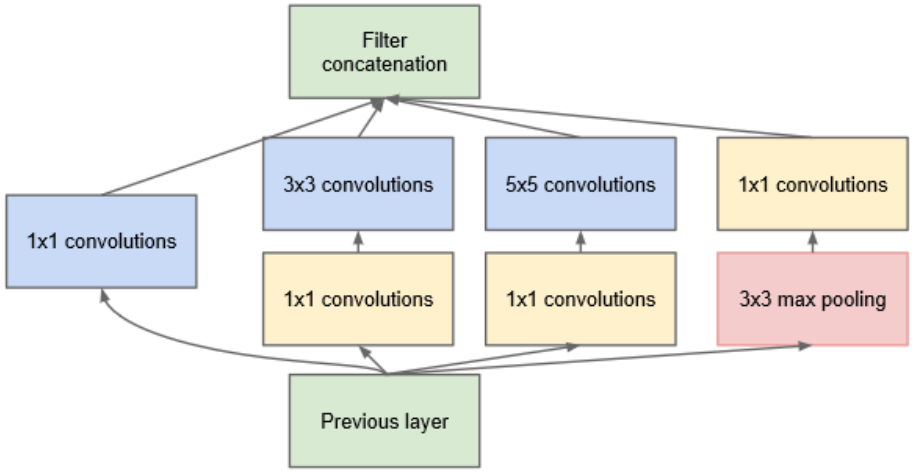


(a) Inception module, naïve version

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

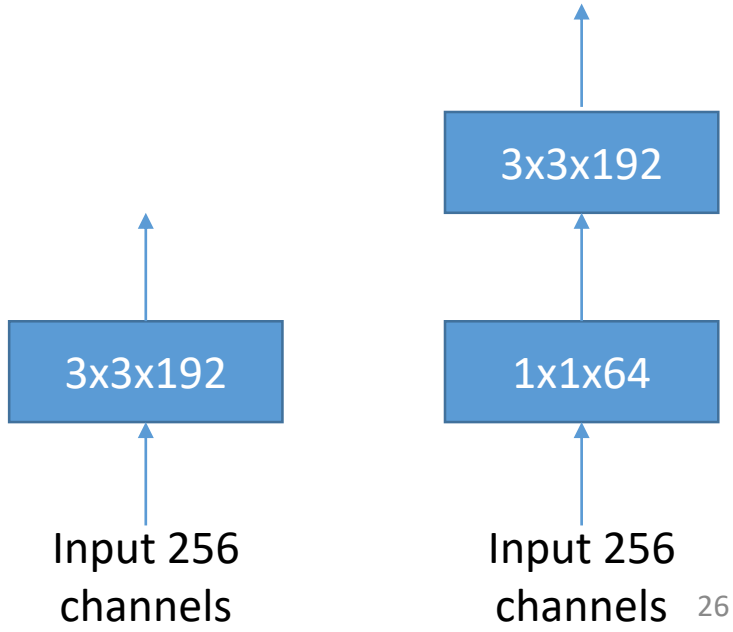
InceptionNet

- 1x1 convolution
  - Fusion of feature maps
    - dimension reduction;
    - remove redundant features;
    - cross channel information learning [16]
  - 1x1 convolution cost, f x c x 1 x 1 x h x w
  - Cost saving?



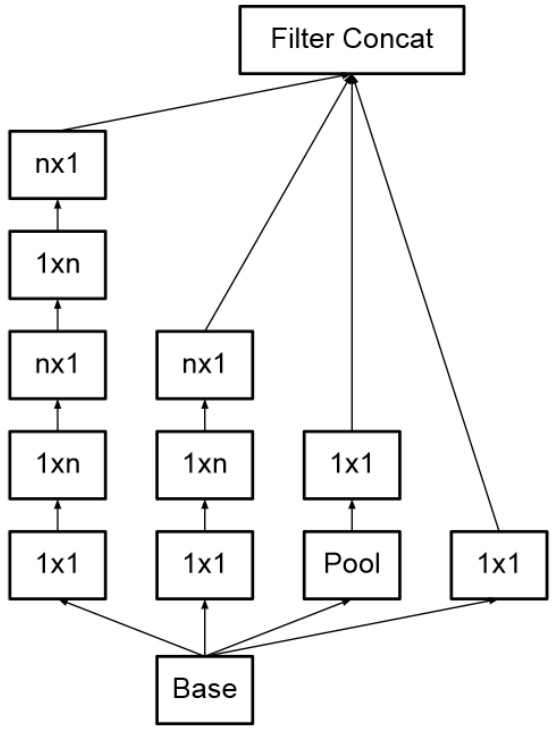
(b) Inception module with dimension reductions

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops



InceptionNet [10]

- Factorization of the kernel
  - 7x7 -> 1x7 and 7x1?
  - Reduce computation cost → train faster



ResNet [11]

- Can we go much deeper?
  - Overfitting?
    - No
  - Optimization?
    - Difficult for deep net

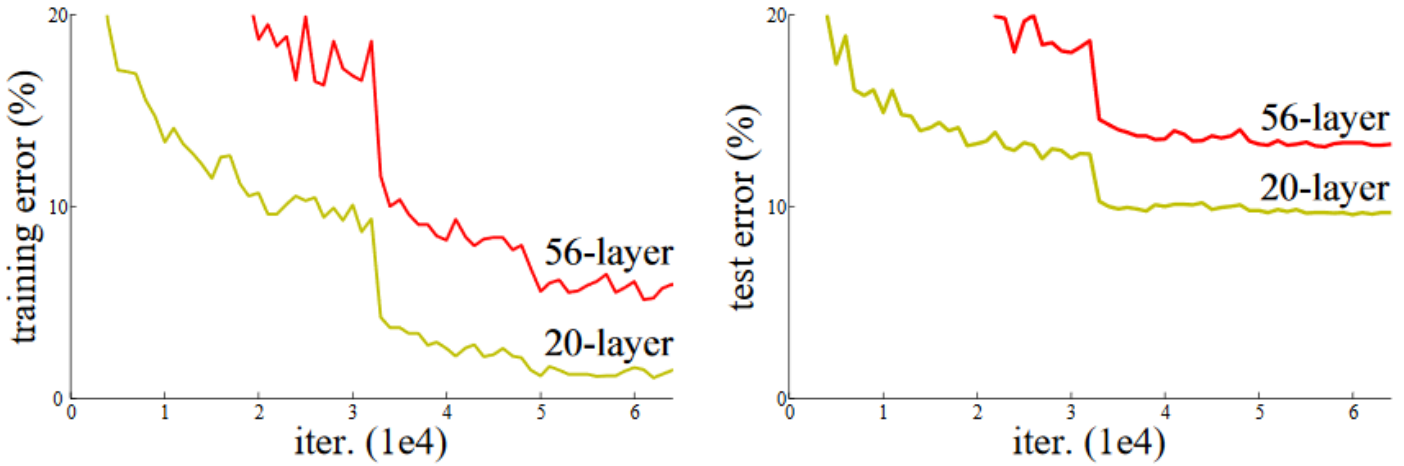
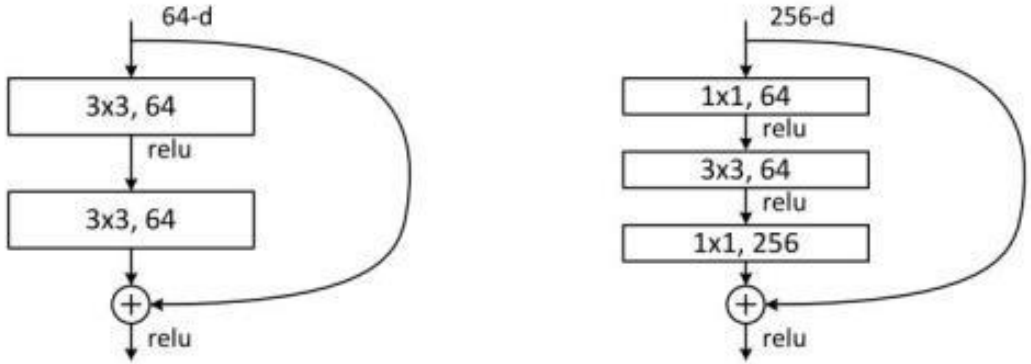


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

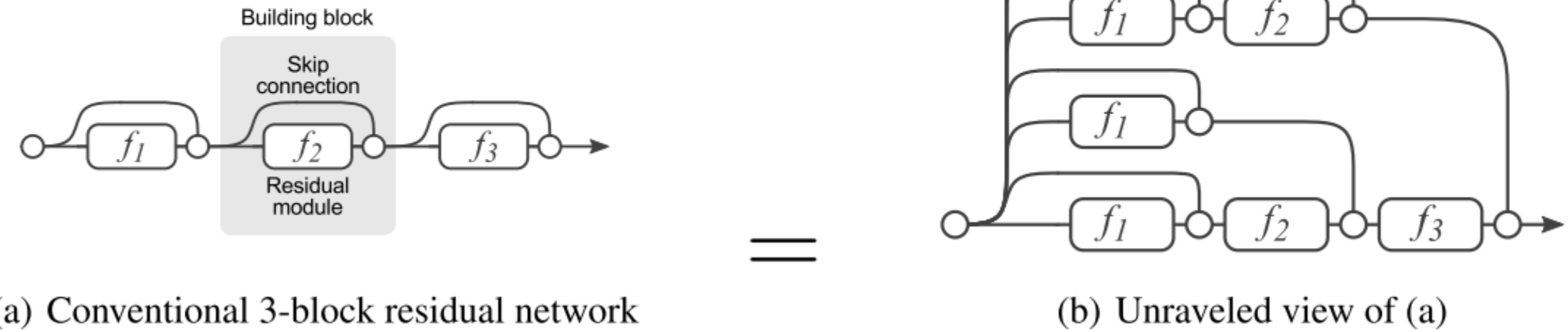
ResNet

- http://ethereon.github.io/netscope/#/gist/d38f3e6091952b45198b
- 152-layers, 3.57% top 5 error, winner for classification, detection, etc.
- Skip/residual connection
  - $y = x + f(x|W)$
  - Gradient flows without degradation
    - Extreme case is identity connection
    - $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} + \frac{\partial L}{\partial y}\frac{\partial f}{\partial x}$
  - Easy to optimize
    - Converge faster



ResNet 2015

- Unravel [1]
  - K skip-connections→2^k paths
  - Model ensemble
    - Generalize better → accuracy

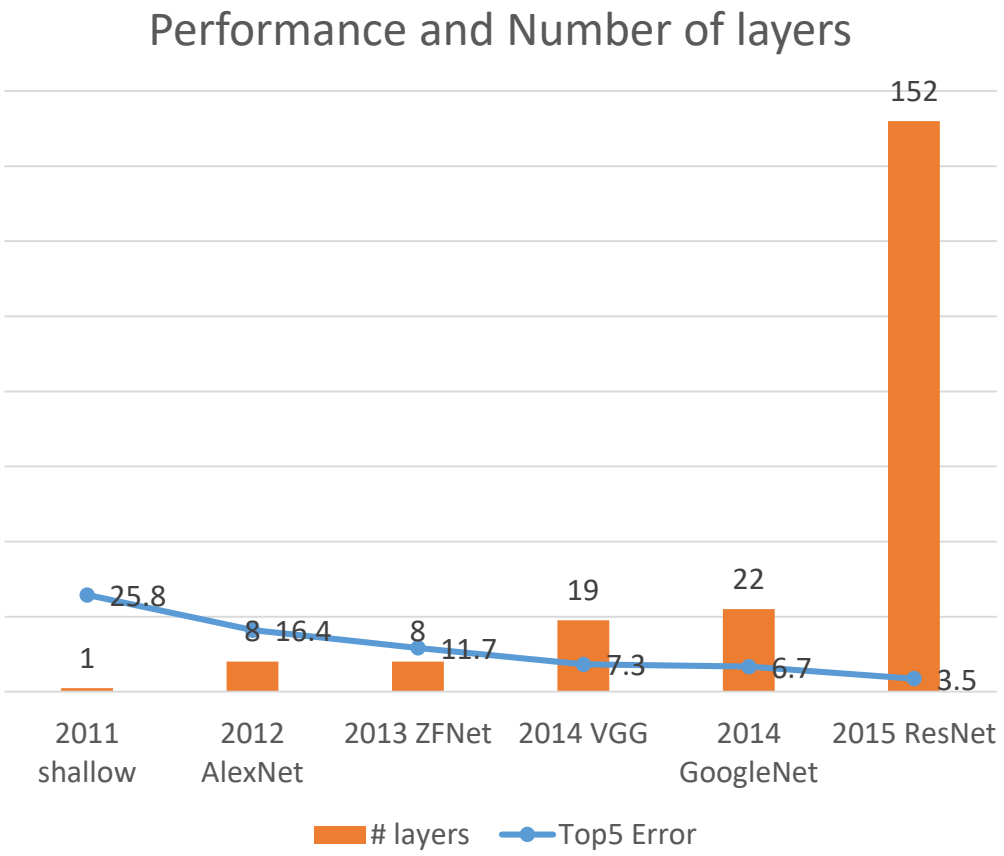


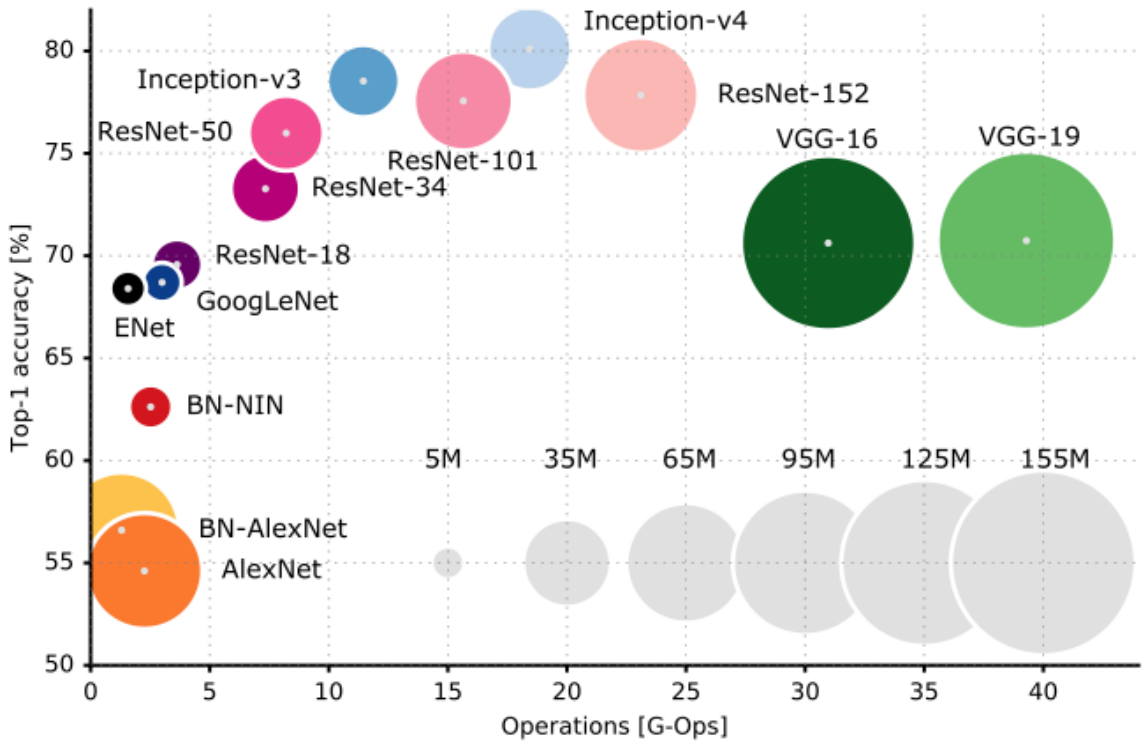
(a) Conventional 3-block residual network

(b) Unraveled view of (a)

Source from [1]

Summary


Performance and Number of layers
152
25.8
16.4
11.7
7.3
6.7
3.5
1
8
8
19
22
2011 shallow
2012 AlexNet
2013 ZFNet
2014 VGG
2014 GoogleNet
2015 ResNet
layers
Top5 Error


Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
VGG-16
VGG-19
ResNet-34
ResNet-18
GoogLeNet
ENet
BN-NIN
BN-AlexNet
AlexNet
5M
35M
65M
95M
125M
155M
Top-1 accuracy [%]
Operations [G-Ops]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40

CNN Training

Training algorithm

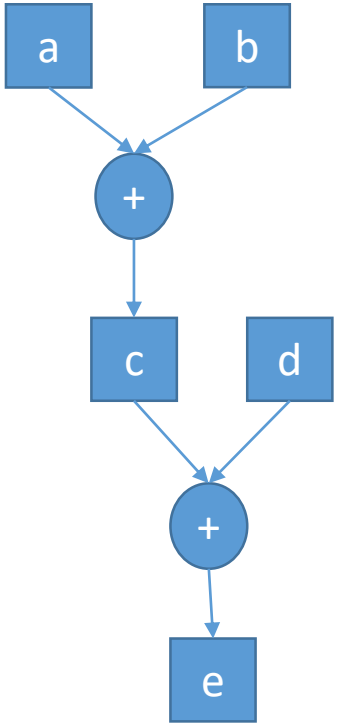
- Training objective
  - Reflected by the loss function
  - Softmax → P(cat|image) VS P(dog|image)
  - Cross-entropy
    - $L = \sum_i t_i log P_i$ $t_i = 1$ if the truth is the i-th label; otherwise 0. Pi=P(i-th label|image)
    - == maximum log-likelihood P(cat|image)

Training algorithm

- Sample a batch of images
  - Run **back-propagation algorithm** to compute the gradients of the **loss** function w.r.t. each parameter
  - **Update** the parameter values in the opposite direction of the gradient
    - $W = W - \alpha \times \frac{\partial L}{\partial W}$

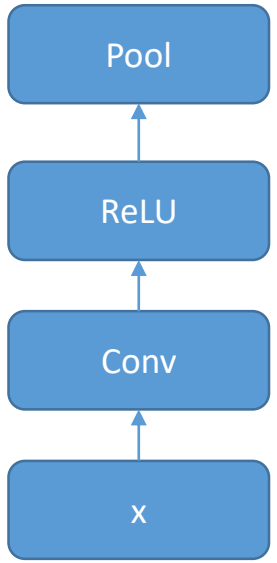
Back-propagation

- Computation graph
  - c = a + b, e = c * d
  - With Add(), Mul()
    - c = Add(a, b)
    - e = Mul(c, d)



Back-propagation

- Computation graph
  - y = ConvFwd(x, W)
  - z = ReLUFwd(y)
  - o =PoolFwd(z)



- Chain Rule
  - $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial W} = \frac{\partial L}{\partial o}\frac{\partial o}{\partial z}\frac{\partial z}{\partial y}\frac{\partial y}{\partial W}$
  - $\frac{\partial L}{\partial z}$ =PoolBwd($\frac{\partial L}{\partial o}$)
  - $\frac{\partial L}{\partial y}$ =ReLuBwd($\frac{\partial L}{\partial z}$)
  - $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W} \leftarrow$ ConvBwd($\frac{\partial L}{\partial y}$)

Back-propagation

- Modular API
  - y=layerFwd(x, W, args)
  - dx=layerBwd(dy, W, args)
- Back-propagation using the modular API
  - For each layer in order
    - Get W and args
    - X=layerFwd(x, W, args)
  - g=ComputeLoss(x, truth)
  - For each layer in reverse order
    - Get W, args
    - g=layerBwd(g, W, args)
    - dW ← g
    - Update W=W-$\alpha\ dW$

Training Practice

- Data normalization
  - Zero mean, unit standard deviation
  - Per channel, 3 values for mean, 3 values for std
  - Compute the mean for each channel over all training images
- Data augmentation
  - Training
    - Random scale, random crop, random flip
  - Testing
    - Scale to fixed size, crop at fixed positions into fixed size, flip and no-flip
- Use adaptive learning rate
  - http://ruder.io/optimizing-gradient-descent/
  - Start with large learning rate and then decreases it 1/10 when the validation loss plateau
- Weight initialization
  - Xavier[12], MSRA [13]
    - Based on fan-in, fan-out
  - Use popular architectures and pre-trained parameters for ImageNet

Projects

Food Image Classification

- Transfer learning
  - Using popular CNNs as the model
    - Change the input size
    - Change the output layer (1000->132)
    - Use the pretrained model parameters over ImageNet to initialize the new model weights
  - Data augmentation
  - Model ensemble
    - Different architectures: VGG+ResNet
    - Different augmentation
    - Different checkpoint files
  - Structure pruning/change
- Training from scratch

Question answering (reading comprehension)

Chloroplasts' main role is to conduct photosynthesis, where the photosynthetic pigment chlorophyll captures the energy from sunlight and converts it and stores it in the energy-storage molecules ATP and NADPH while freeing oxygen from water. They then use the ATP and NADPH to make organic molecules from carbon dioxide in a process known as the Calvin cycle. Chloroplasts carry out a number of other functions, including fatty acid synthesis, much amino acid synthesis, and the immune response in plants. The number of chloroplasts per cell varies from 1 in algae up to 100 in plants like Arabidopsis and wheat.

What is the primary purpose of chloroplasts?
Ground Truth Answers: conduct photosynthesis

- Context {ci}
- Question {qj}
- Answer (a, b)
  - The starting and ending positions of the words in the context

Modelling

- Generate candidate answers
  - In total L^2 possible answers
  - Filter some answers
- For each answer, extract features f with the question considered
  - Span of the answer
  - Relative position in the context
  - Similarity w.r.t the question
- Logistic regression or MLP [4]
  - One candidate answer as a training sample
  - The ground truth answer as the label

Modelling

- P(a|Q, C) and P(b|Q, C)
  - For each a in range(0, end):
- for each b in range(a+1, end):
- estimate P(a|Q, C) and P(b|Q, C)
- Select a and b with the maximum probability
- Core
  - Context representation ← e.g. RNN
    - Context word
  - Question representation ← e.g. RNN
    - Question word
  - P(a|Q, C) = P(W_a|Q, C)
    - e.g. design f(W_a, Q) to measure the relationship of W_a and Q
- [2,3,4,5]

Reference

- https://medium.com/towards-data-science/an-overview-of-resnet-and-its-variants-5281e2f56035
- [1] A. Veit, M. Wilber and S. Belongie. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. arXiv:1605.06431v2,2016.
- [2] https://web.stanford.edu/class/cs224n/reports/2762048.pdf
- [3] https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf
- [4] https://arxiv.org/pdf/1606.05250.pdf
- [5] https://arxiv.org/abs/1611.01604
- [6] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- [7]] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of The 32nd International Conference on Machine Learning, pages 448–456, 2015
- [8] Very Deep Convolutional Networks for Large-Scale Image Recognition. Karen Simonyan, Andrew Zisserman. https://arxiv.org/abs/1409.1556
- [9] C. Szegedy et al., Going deeper with convolutions, CVPR 2015
- [10] Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. https://arxiv.org/abs/1602.07261
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren , and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016
- [12] http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf
- [13] Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. https://arxiv.org/abs/1502.01852