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

- Convolution and Pooling
  - 1D convolution
  - 2D convolution
  - Pooling
- CNN Architectures and Training techniques
  - AlexNet, VGG, InceptionNet, etc.
  - Activation functions
  - Normalization functions
  - Hyper-parameters
- Applications
  - Classification, Detection, Segmentation, NeuralStyle, Video, FaceRecognition, Sentiment analysis

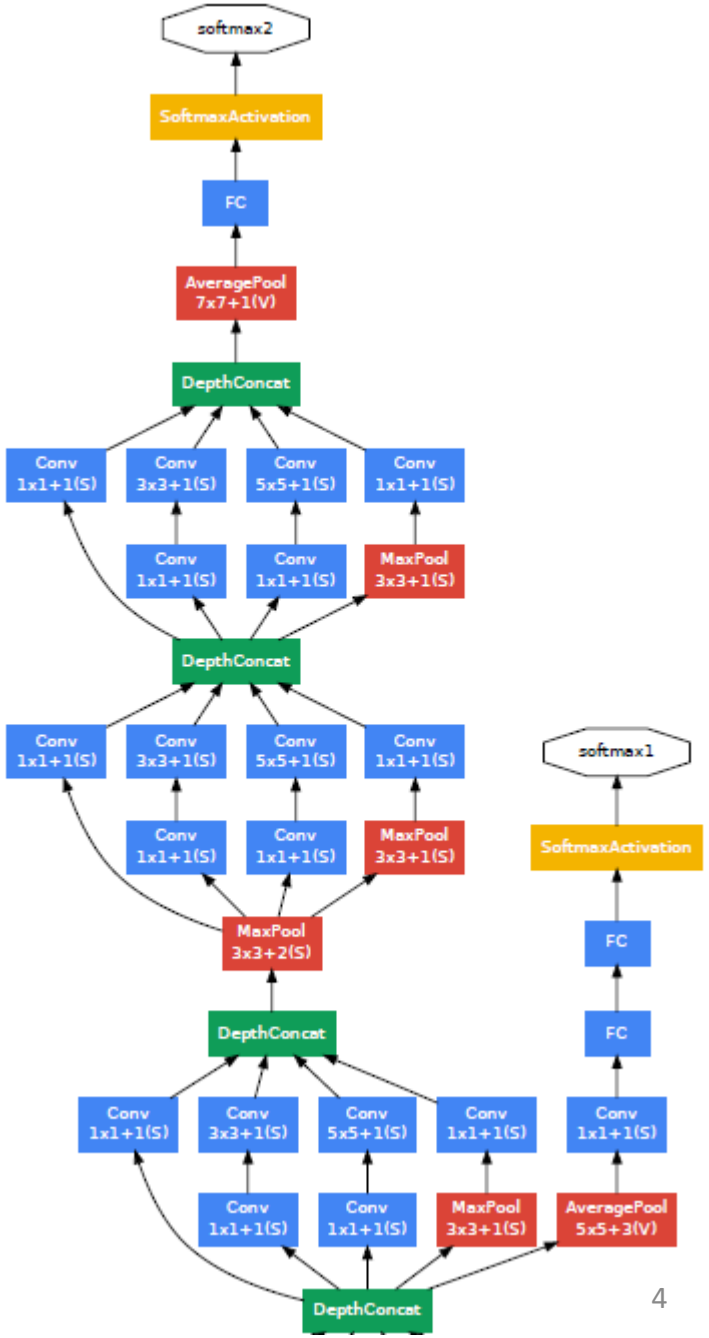
Last Week

LeNet-5	AlexNet	VGG
Convolution + subsampling; Backpropagation	ReLU, dropout, deep structure, large training dataset, GPU	3x3 convolution

GoogLeNet / InceptionNet

- Motivation
  - Different kernels to group features
  - Computation/memory cost

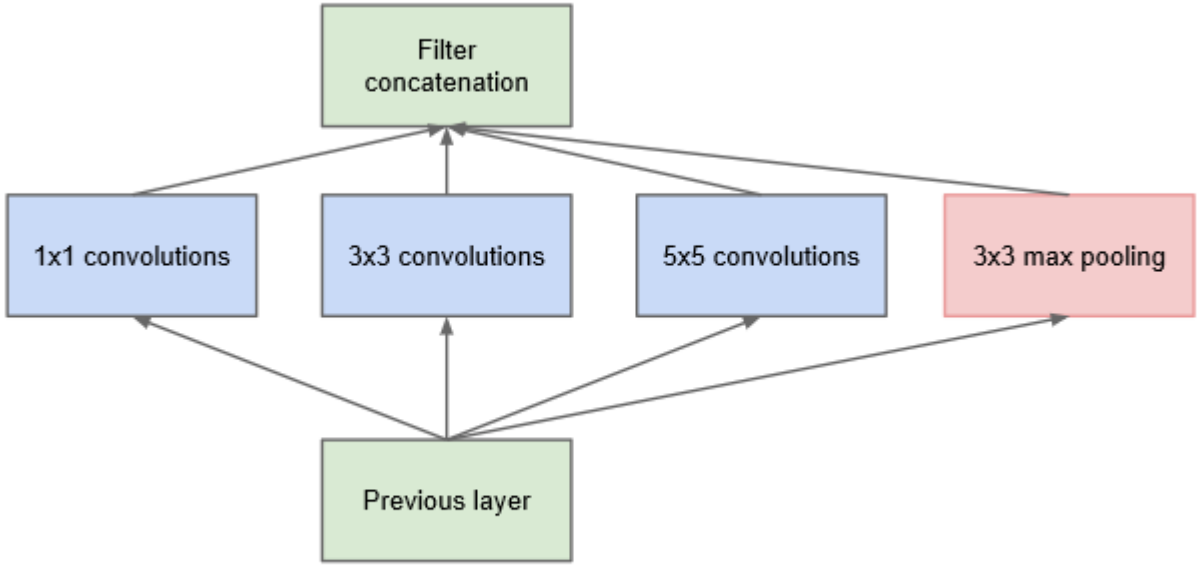
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



Source from [8]

GoogLeNet / InceptionNet

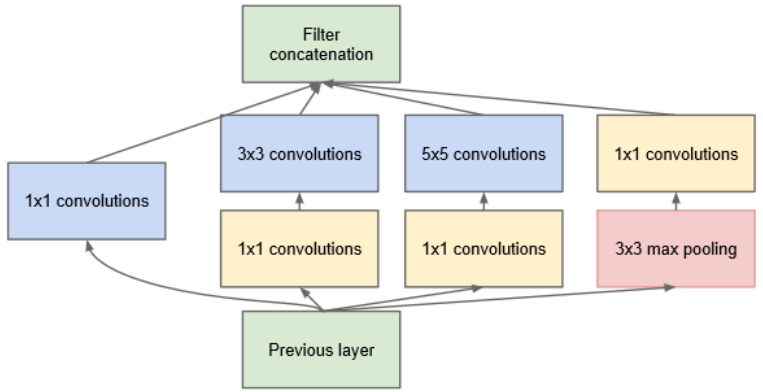
- Each Stack
  - 1x1, 3x3, 5x5 kernels (avoid co-adaption)
  - Wide block



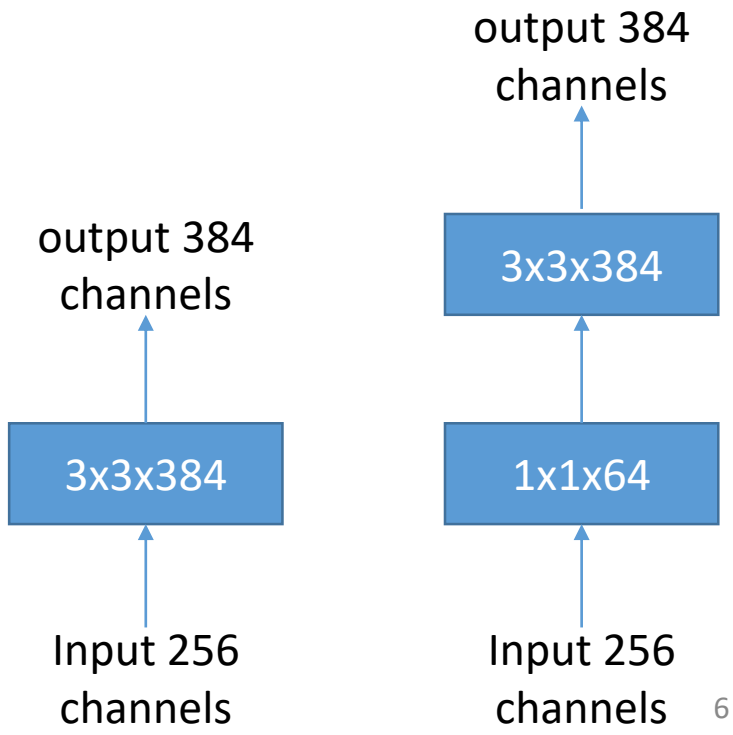
(a) Inception module, naïve version

GoogLeNet / InceptionNetV1

- 1x1 convolution [6]
  - 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?

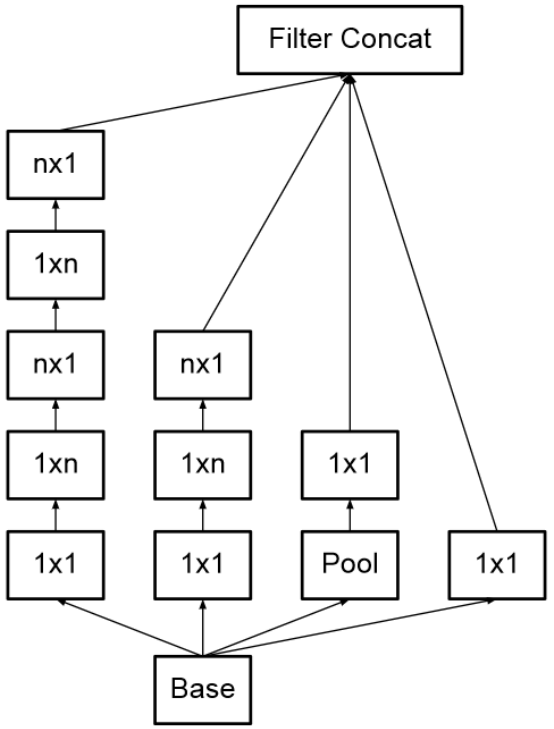


(b) Inception module with dimension reductions

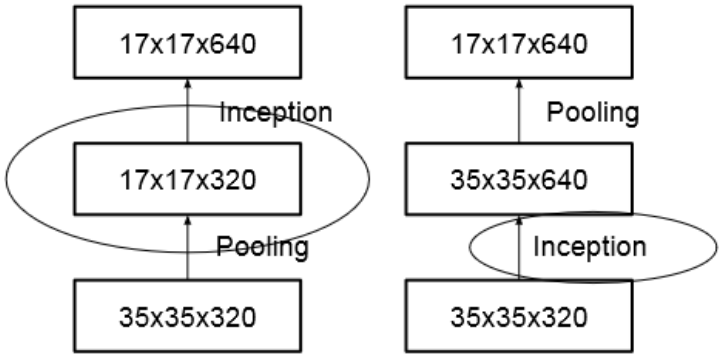


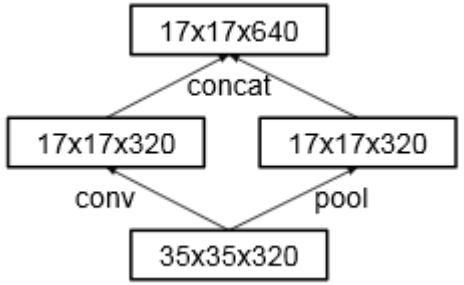
GoogLeNet / InceptionNetV2

- Factorization of the kernel
  - 7x7 -> 1x7 and 7x1 ?



- Avoid representation bottleneck



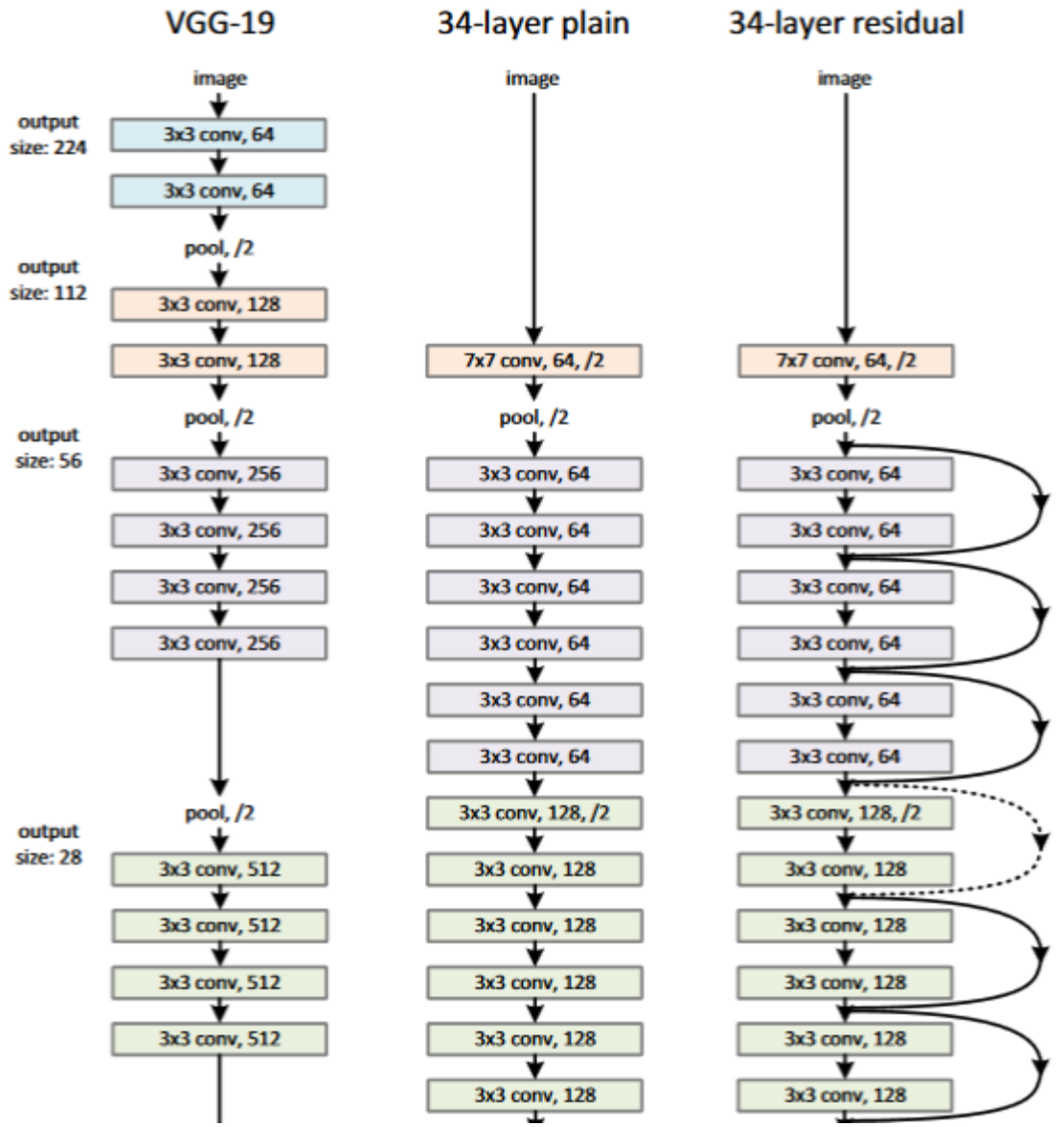


GoogLeNet / InceptionNet

- Remove fully connected layers
  - By average pooling layer with kernel size = input size
  - No parameters
- Auxiliary classifier
  - Generate gradients for lower layers
- Training image augmentation
  - Crop a random patch with size [8%, 100%] of the original image, and
  - aspect ratio is chosen randomly between [3/4, 4/3]
  - photometric distortions [15]
- Test time augmentation
  - Scale the smaller side to 256, 288, 320 and 352;
  - For each scale, take the left, center and right square (or top, center and bottom)
  - For each square, take the 4 corners and the center 224x224, as well as horizontal reflection
  - 144 crops per image -> average the outputs from softmax layer

ResNet [9]

- **Deep** networks are hard to train
  - Optimization challenge
    - Gradient degradation



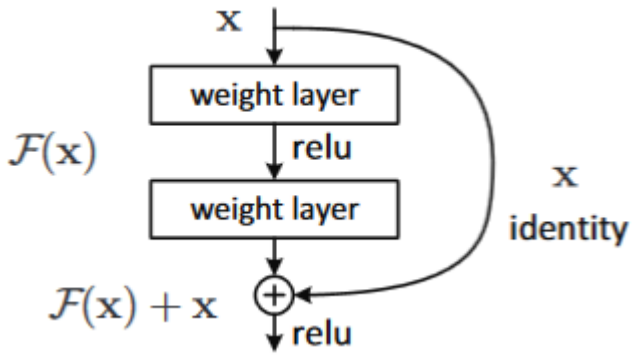
Source from [9]

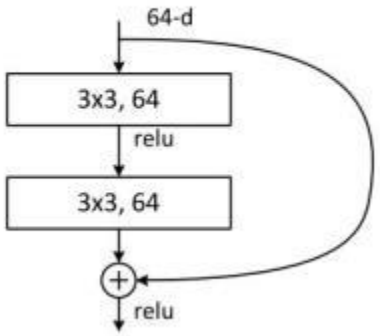
ResNet

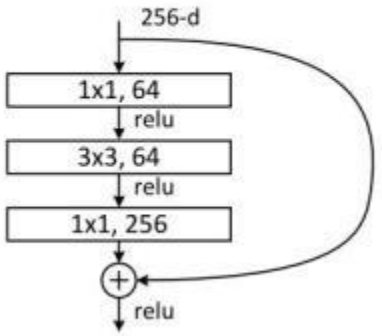
- Skip connection

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

  - Gradient flows without degradation
    - Extreme case is identity connection

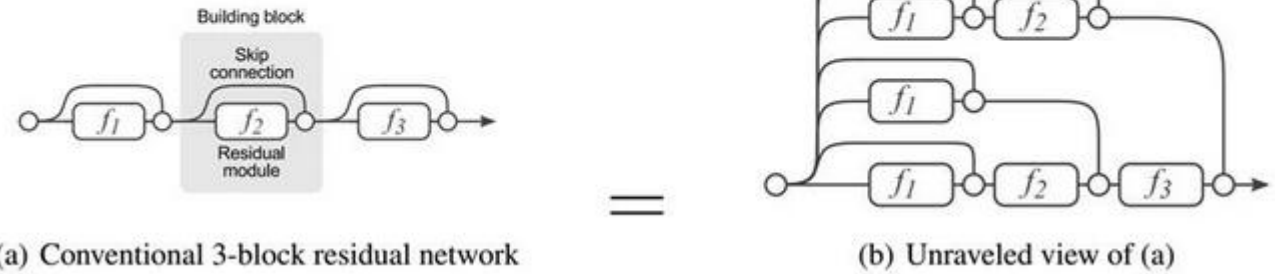






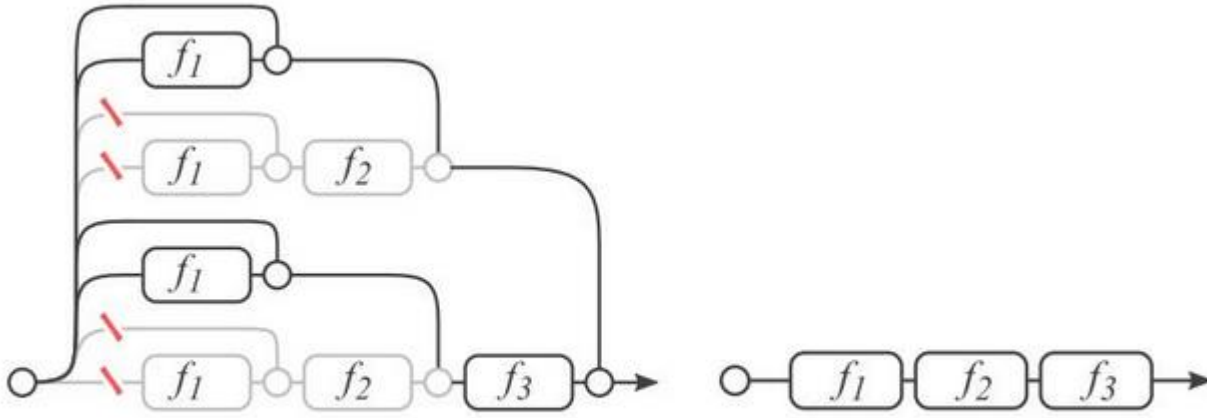
ResNet

- Ensemble



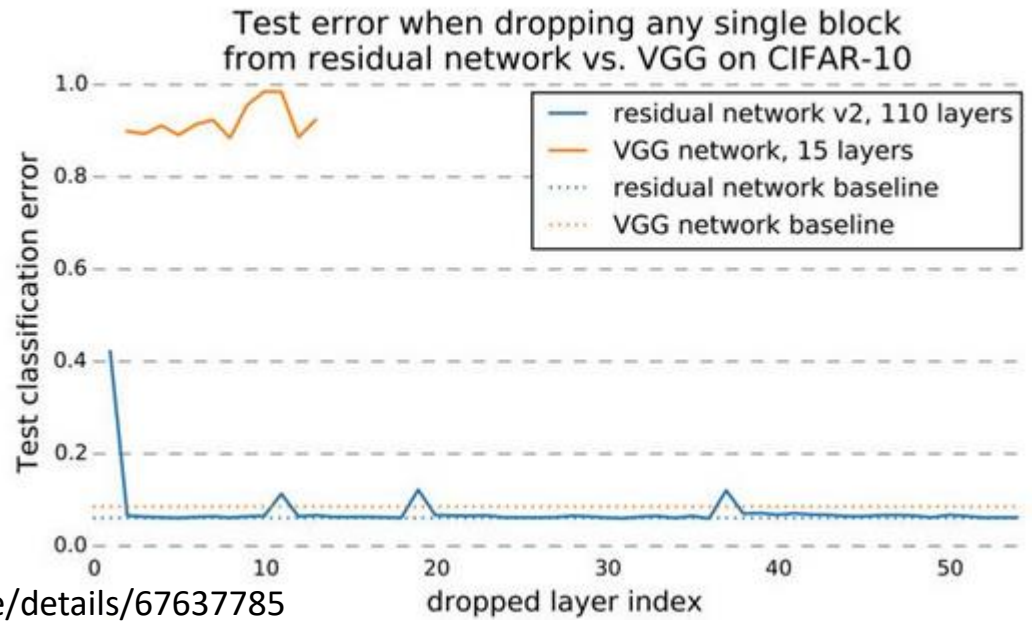
(a) Conventional 3-block residual network

(b) Unraveled view of (a)



(a) Deleting f_2 from unraveled view

(b) Ordinary feedforward network



Source from http://blog.csdn.net/malefactor/article/details/67637785

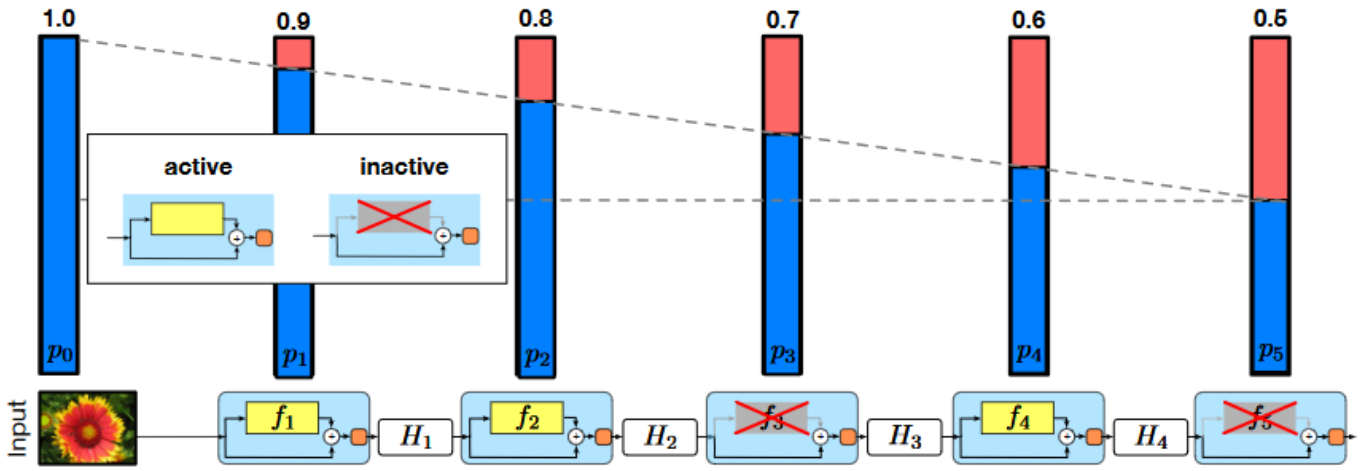
ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

- Extension: **Stochastic ResNet, DenseNet**, FractalNet, WideResNet, etc

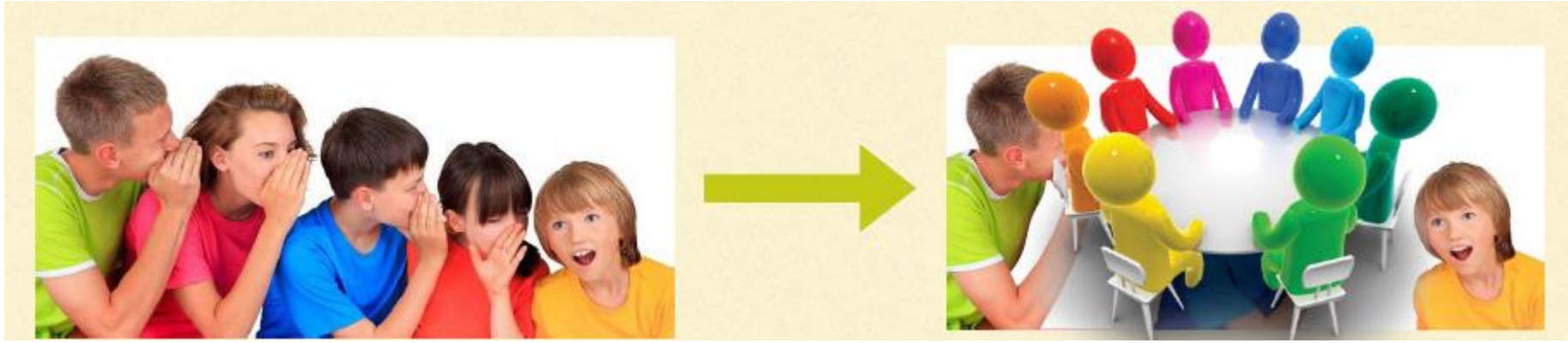
Stochastic ResNet [18]

- Drop one block with probability p
  - Similar to dropout, but applied on block level
  - Model ensemble
  - Rescale the outputs?
- Model is simpler
  - Fewer layers
  - Faster BP



Source from [18]

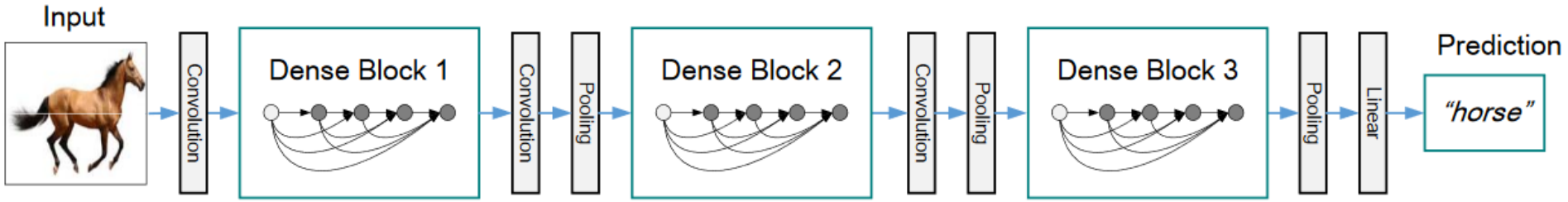
Stochastic ResNet



Source images from Gao Huang

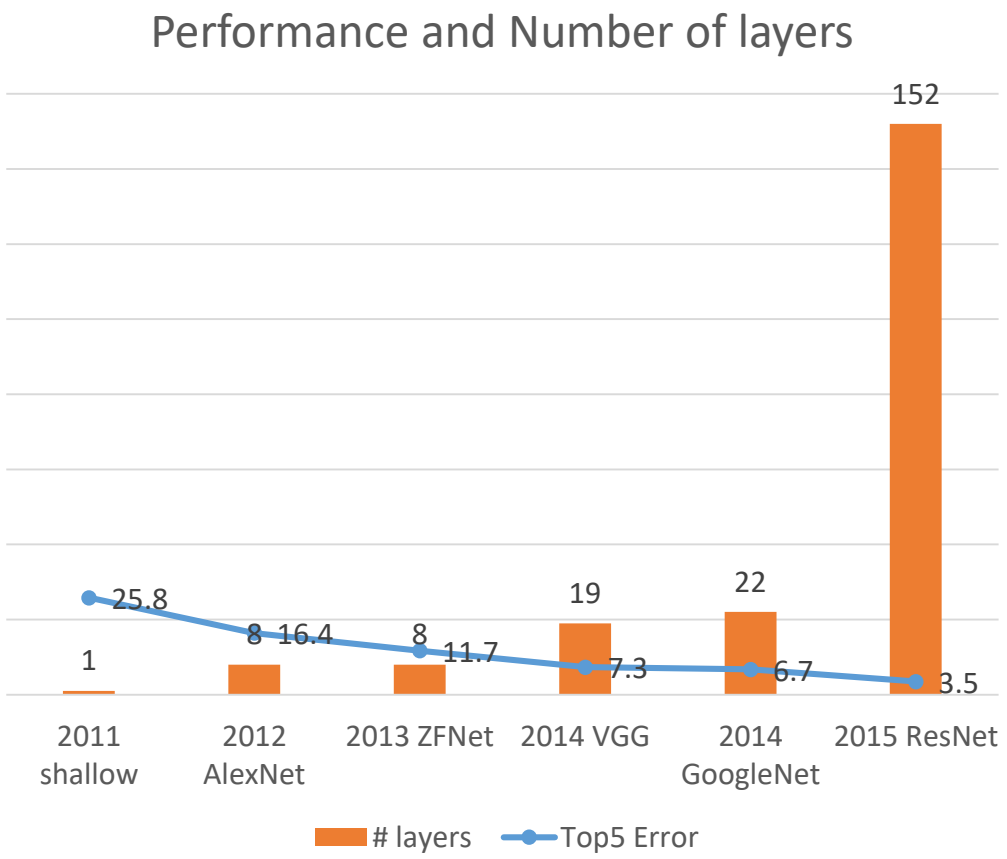
DenseNet [19]

- Extended from Resnet
  - Multiple skip-connections within the block
  - Feature maps from Li are concatenated to Lj (i+1 < j)
- Fewer feature maps at bottom layers
- Feature reuse



Source from [19]

Summary


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

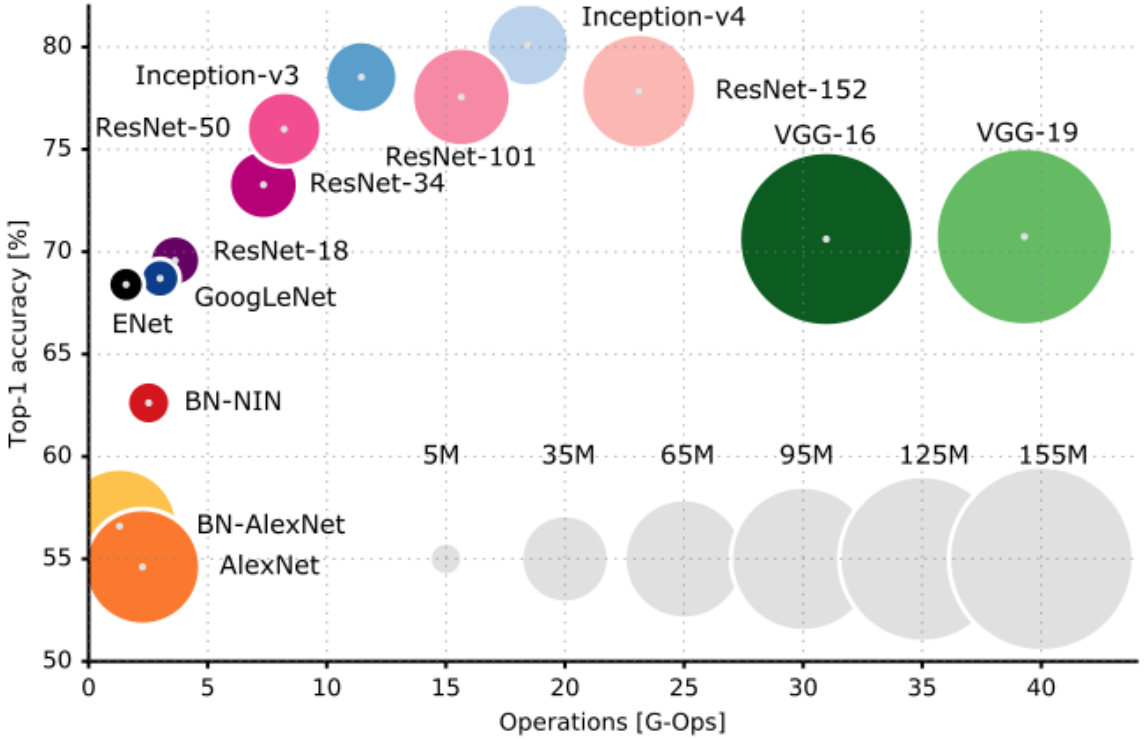
Summary

- Goals of designing CNN architectures
  - Faster
    - Converge faster → optimization
    - Compute faster → complexity
  - More accurate
    - Generalize better → overfitting

Complexity

- Bottleneck layer
  - Reduce feature maps
- Smaller kernels
  - 11x11, 7x7, 5x5->3x3
  - 5x5 -> 1x5 and 5x1
- Group convolution/depth-wise convolution
- Reduce fully connected layers

Complexity


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
Top-1 accuracy [%]
Operations [G-Ops]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40

Optimization

- Activation function
  - ReLU, PReLU, LeakyReLU
- Skip connection, dense connection
- Auxiliary loss
- Data normalization and parameter initialization
- Batch-normalization

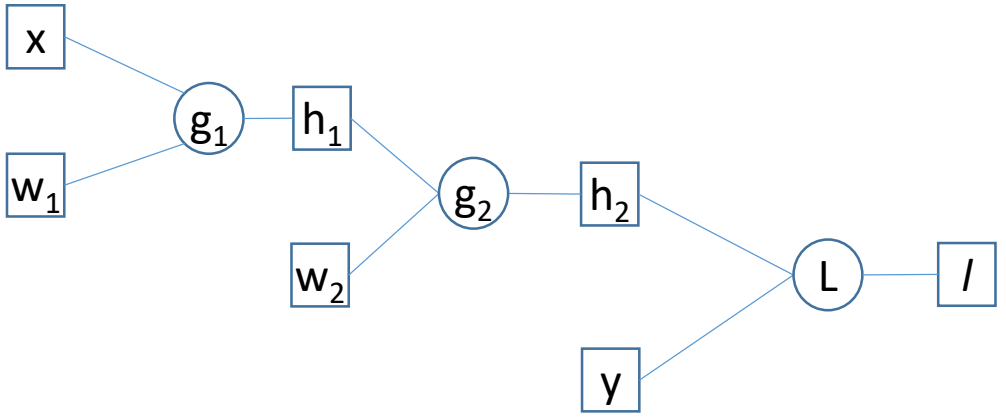
Generalization

- Dropout
- Stochastic connection
- Residual block/skip connection
- Inception block
- Data augmentation

Optimization techniques for CNN

Back-propagation

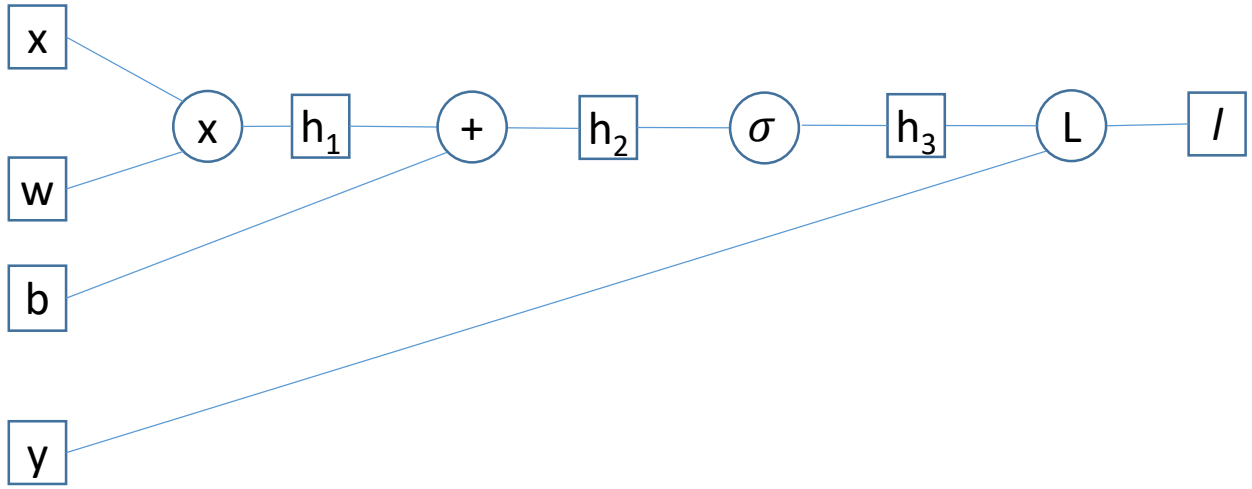
- Computation graph view
  - $l = L(g_2(g_1(x, w_1), w_2), y)$



  - $l = L\,(y, \sigma(x \times w + b))$
  - $= -y \times log\frac{1}{1+e^{-(x \times w+b)}} - (1-y) \times log\frac{1}{1+e^{-(x \times w+b)}}$
  - Graph ?

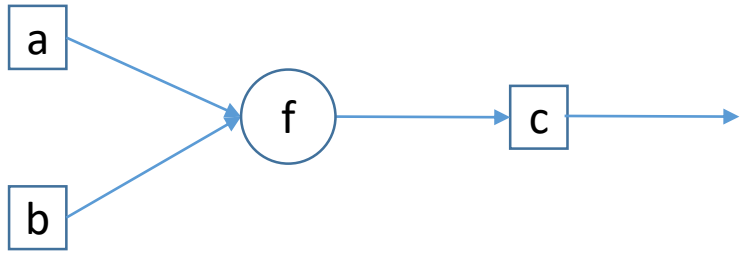
Back-propagation

- Example
  - $l = L\left(y, \sigma(x \times w + b)\right)$
  - $= -y \times log \frac{1}{1+e^{-(x \times w+b)}} - (1-y) \times log \frac{1}{1+e^{-(x \times w+b)}}$
  - x=1.2, w=0.5, b=-0.6, y = 1
  - *l*?
  - $\frac{\partial l}{\partial b}, \frac{\partial l}{\partial w}$?

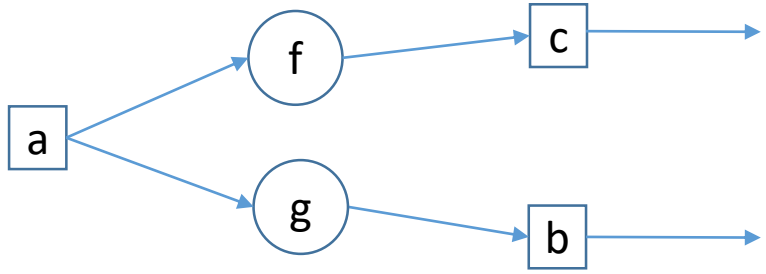


Back-propagation

- For any subgraph I
  - $\frac{\partial l}{\partial a} = \frac{\partial l}{\partial c} \times \frac{\partial c}{\partial a} = \frac{\partial l}{\partial c} \times f_{a'}$
  - $\frac{\partial l}{\partial b} = \frac{\partial l}{\partial c} \times \frac{\partial c}{\partial b} = \frac{\partial l}{\partial c} \times fb'$
- For any subgraph II
  - $\frac{\partial l}{\partial a} = \frac{\partial l}{\partial c} \times \frac{\partial c}{\partial a} + \frac{\partial l}{\partial b} \times \frac{\partial b}{\partial a} = \frac{\partial l}{\partial c} \times f' + \frac{\partial l}{\partial b} \times g'$



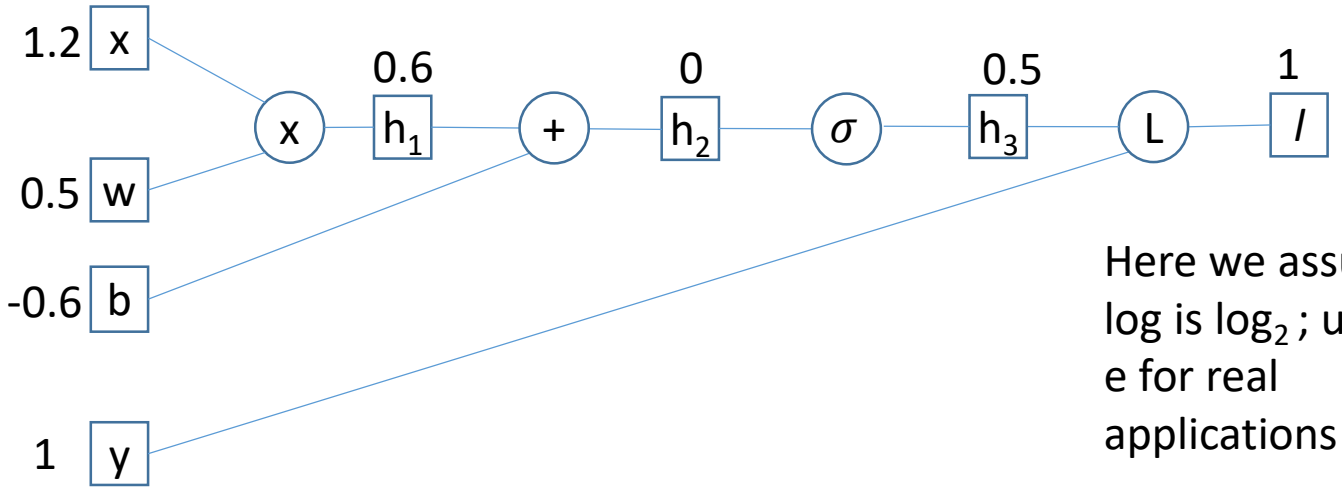
Subgraph I



Subgraph II

Back-propagation

- Example
  - l=, $\frac{\partial l}{\partial b}$, $\frac{\partial l}{\partial w}$?
  - $L'_{h_3} = -\frac{y}{h_3} - \frac{1-y}{1-h_3}$
  - $\sigma'_{h2} = h_3(1-h_3)$
  - $+'_{h1} = 1, +'_b = 1$
  - $\times'_w = x$



Here we assume log is $\log_2$; use log e for real applications

Back-propagation

- Vectorization
  - $x \in R^d, W \in R^{d \times z}, b \in R^z$
  - $x \in R^{n \times d}, W \in R^{d \times z}, b \in R^z$
  - Gradients and variables have the same shape: $\frac{\partial l}{\partial W} \in R^{d \times z}, \frac{\partial l}{\partial b} \in R^z$
- Debugging
  - print shapes of all variables and their gradients
  - Print average l1 norm (average of absolute value) of all variables and check their value range

Back-progation

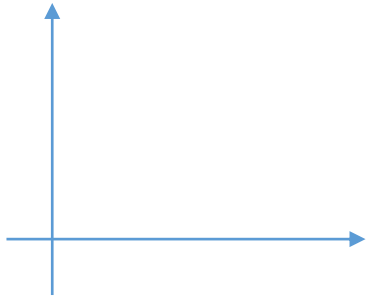
- API

Implement
f(x, w, b) = x*w + b

```
class Add:
    define forward(a, b):
        return a + b

    define backward(dc):
        return dc, dc


class Multiply:
    define forward(x, W):
        return np.dot(x, W)

    define backward(dy, x, W):
        dW = np.dot(x.T(), dy)
        dx = np.dot(dy, W.T())
        return dx, dW
```

Data preprocessing

- Normalization
  - Centering around 0
    - Subtract mean; mean per pixel or mean per channel
  - Standardization
    - Subtract mean / std; (mean, std) per channel.



$$f = \sum w_i x_i + b$$

$$\frac{df}{dw_i} = x_i$$

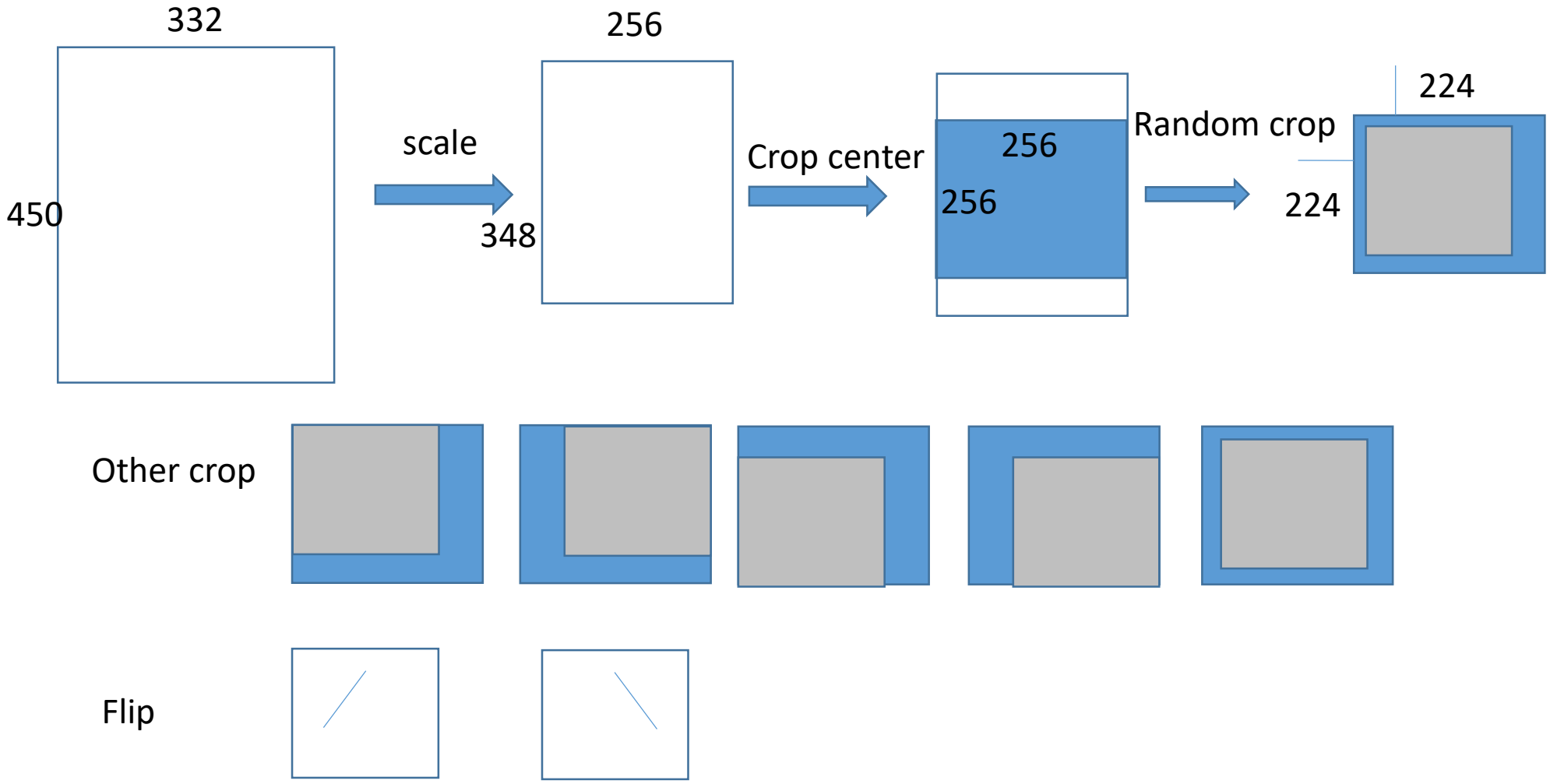
$$\frac{dL}{dw_i} = \frac{dL}{df}\frac{df}{dw_i} = \frac{dL}{df}x_i$$

because $x_i > 0$, the gradient $\frac{dL}{dw_i}$ always has the same sign as $\frac{dL}{df}$ (all positive or all negative).

Source from: https://stats.stackexchange.com/questions/237169/why-are-non-zero-centered-activation-functions-a-problem-in-backpropagation

Data augmentation

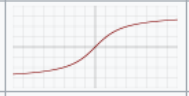
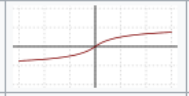
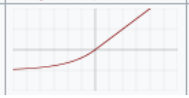
- Scaling
- Cropping
- Flipping
- Intensity
- Rotation



Activation function

- Gradient flow
- Computation overhead

Source from: https://en.wikipedia.org/wiki/Activation_function

Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$
Softsign [7][8]		$f(x) = \frac{x}{1 + \lvert x \rvert}$	$f'(x) = \frac{1}{(1 + \lvert x \rvert)^2}$	$(-1, 1)$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky rectified linear unit (Leaky ReLU)[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parameteric rectified linear unit (PReLU)[11]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Randomized leaky rectified linear unit (RReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \text{ [1]} \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit (ELU)[13]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$

Batch-normalization [17]

- Internal covariate-shift
  - $P(y|x) \sim Ps(x) \quad vs \quad Pt(x)$
  - $P(y|x) \sim P(y|x, \theta)$
  - $P(y|x) \sim P(y|h_k) \;\; P(h_k)$
- Normalize h_k $\quad \widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}} \qquad y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}.$
- 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
- Converge faster

Parameter initialization

- Gaussian, N(0, 0.01)
- Uniform, U(-0.05, 0.05)
- Glorot/Xavier [20]
  - Uniform U(-sqrt(6/(fan_in + fan_out)), +sqrt(6/(fan_in + fan_out)))
  - **Gaussian N(0, sqrt(2/(fan_in + fan_out))**
- He/MSRA [21]
  - Uniform U(-sqrt(6/fan_in), +sqrt(6/fan_in))
  - **Gaussian N(sqrt(2/fan_in))**

Regularization

- Dropout
- L2 norm
  - $l = L(\quad) + \ \alpha \times ||W||_2^2$
- L1 norm
  - $l = L(\quad) + \ \alpha \times ||W||_1$

Summary

- Preprocess the data (zero center)
- Proper initialization (HeGaussian)
- Use Batch-Normalization + Dropout
- Start with large learning rate and small weight decay
  - Decrease the learning rate if numeric error (NaN)
  - Decrease the learning rate if validation error stops dropping
- Model ensemble by training multiple models with
  - Different architectures
  - Different subsets of training data
  - Different initialization approaches
- Hyper-parameter tuning
  - **Learning rate**, weight decay, number of layers and size of each layer,

Administrative

- Assignment 1 due date: Sept. 28th, 11:59PM
- Assignment 2
  - Announcement date: Sept. 29th
  - Due date: Oct. 19th, 11:59PM

Assignment 2

- 2D Convolution
  - Multiple input channels, multiple output channels
  - Forward and backward
    - Better use the matrix multiplication implementation
- 2D Max Pooling
  - Forward and backward
- Dropout
  - Forward and backward
- The grading is based on
  - the outputs for given inputs
  - Report explaining the operations (using mathematic equations)
    - E.g. how to do the backward for convolution layer?
    - dW=outer(dy, x), what is dy and what is x?
    - Do not copy the code into the report; be concise.

Assignment 2

- Combine above layers into a CNN
  - Use any data preprocessing, augmentation, learning rate scheme, SGD, parameter initialization approaches.
  - Plot the figures for training and validation loss VS training iterations
  - Compare the results for different settings.
- Grading is based on
  - the code (no need to submit any outputs files)
  - The report
    - Including the settings (data preprocessing, augmentation, etc.)
    - Results

References

- [1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998
- [2] https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- [3] https://www.coursera.org/learn/neural-networks/lecture/68Koq/another-diversion-the-softmax-output-function-7-min
- [4] https://arxiv.org/pdf/1311.2901.pdf
- [5] https://medium.com/mlreview/experiments-with-a-new-loss-term-added-to-the-standard-cross-entropy-85b080c42446
- [6] M. Lin, Q. Chen, and S. Yan, Network in network, ICLR 2014
- [7] https://arxiv.org/pdf/1409.1556.pdf
- [8] C. Szegedy et al., Going deeper with convolutions, CVPR 2015
- [9] Kaiming He, Xiangyu Zhang, Shaoqing Ren , and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016
- [10] https://arxiv.org/abs/1608.06993
- [11] https://arxiv.org/abs/1610.02357
- [12] C. Szegedy et al., Rethinking the inception architecture for computer vision, CVPR 2016
- [13] https://arxiv.org/abs/1602.07261 (inception v4)
- https://arxiv.org/pdf/1702.08591.pdf

- [14] Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proc. AISTATS*, volume 9, pp. 249–256, 2010.
- [15] Andrew G. Howard. Some improvements on deep convolutional neural network based image classification. CoRR, abs/1312.5402, 2013.
- [16] http://iamaaditya.github.io/2016/03/one-by-one-convolution/
- [17] 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
- [18] Deep Networks with Stochastic Depth. Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, Kilian Weinberger. 2016
- [19] Densely Connected Convolutional Networks. Gao Huang, Zhuang Liu, Kilian Q. Weinberger, Laurens van der Maaten. 2017
- [20] **http**://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf
- [21] http://arxiv.org/abs/1502.01852
- https://mp.weixin.qq.com/s?__biz=MzA3Mjk0OTgyMg==&mid=2651123524&idx=1&sn=0546ceca3d88e2ff1e66fbecc99bd6a7
- https://github.com/vdumoulin/conv_arithmetic
- http://www.alexirpan.com/2017/04/26/perils-batch-norm.html
- https://arxiv.org/pdf/1702.08591.pdf