인공지능 직무전환 과정 3
Deep Learning

Lab 04

연세대학교 컴퓨터과학과
김선주 강재연 조영현

Today

AlexNet / VGG / GoogLeNet / ResNet 구현, 트레이닝 및 테스트

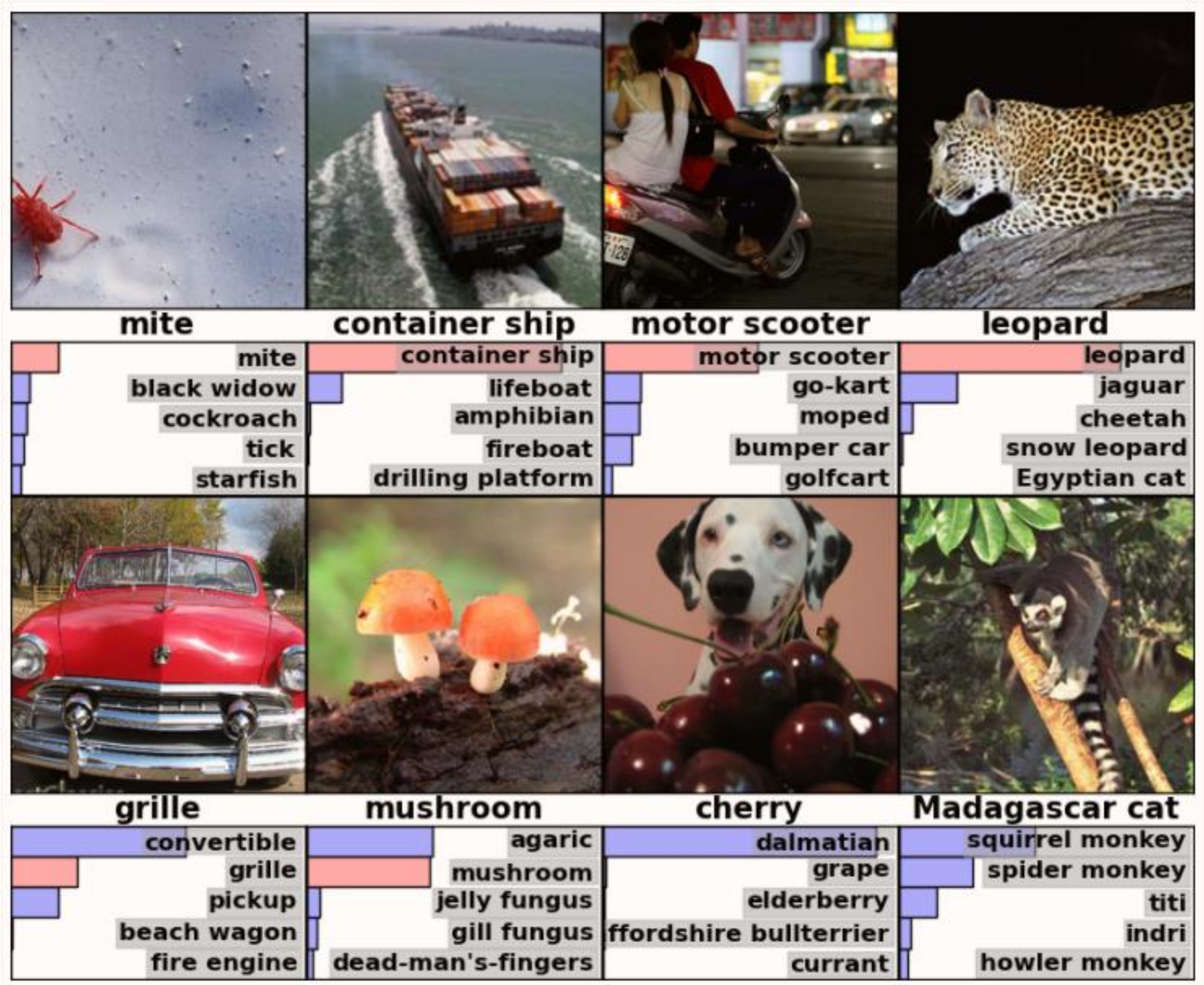
Model ensembles

Data preprocessing / Data augmentation 구현

Optimazations:
SGD / Momentum / Adagrad / RMSProp / Adam 소개

ILSVRC challenge

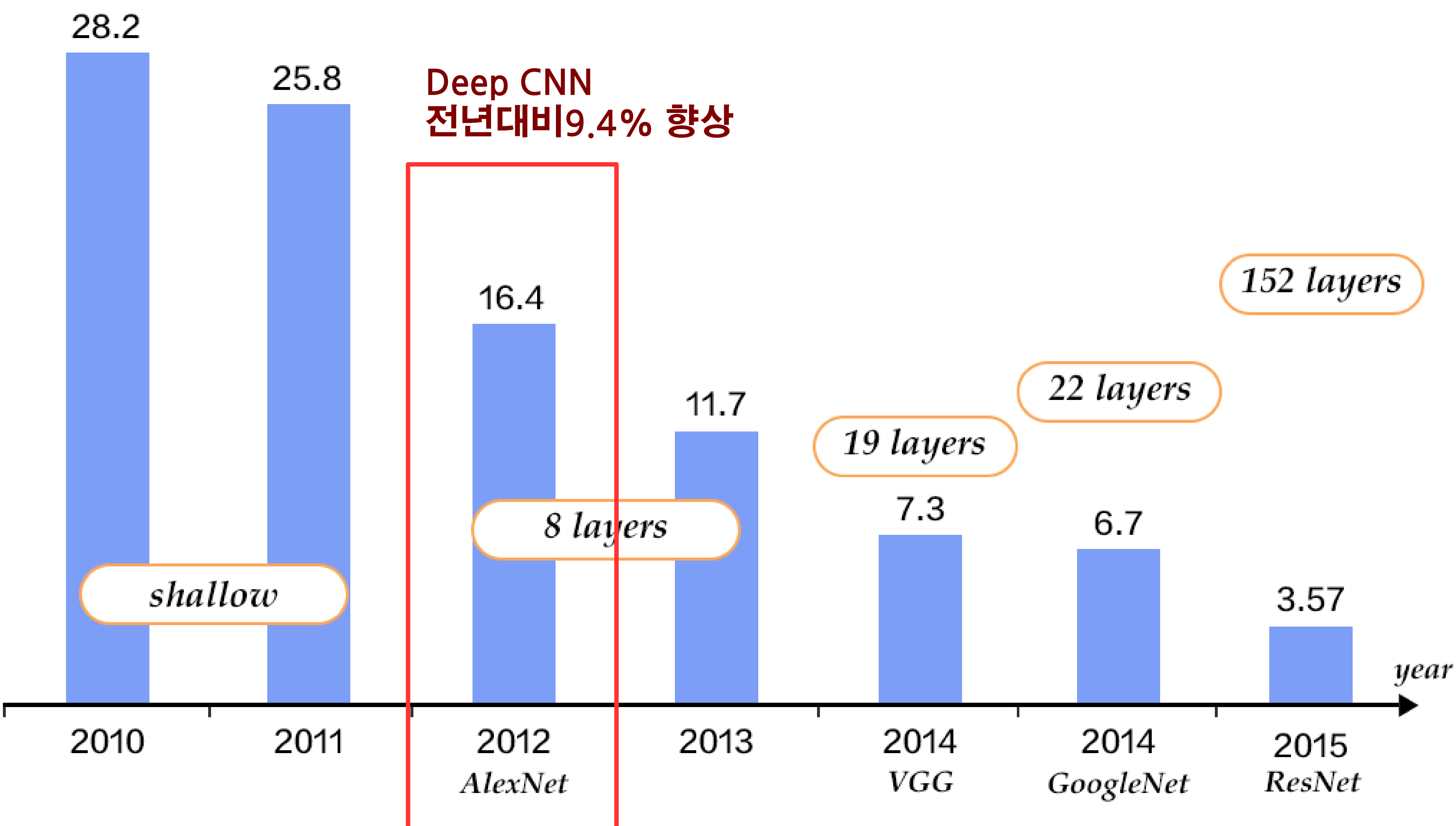
Image classification task http://image-net.org/



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

AlexNet

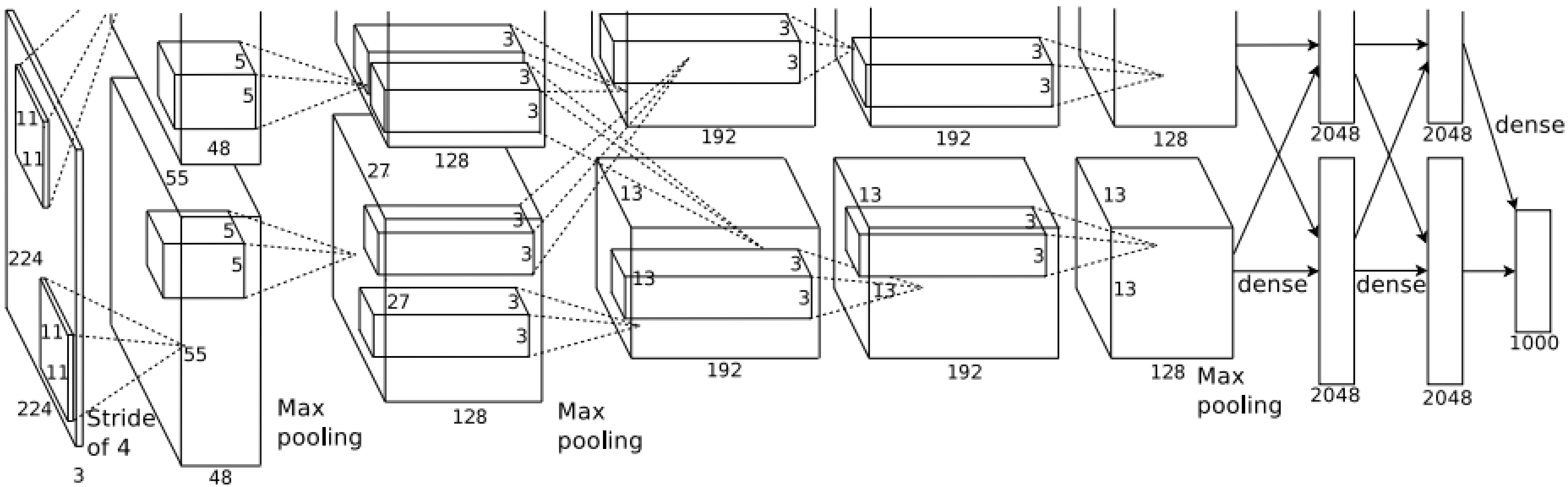
Winner of 2012 ILSVRC image classification challenge



http://book.paddlepaddle.org/03.image_classification/

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

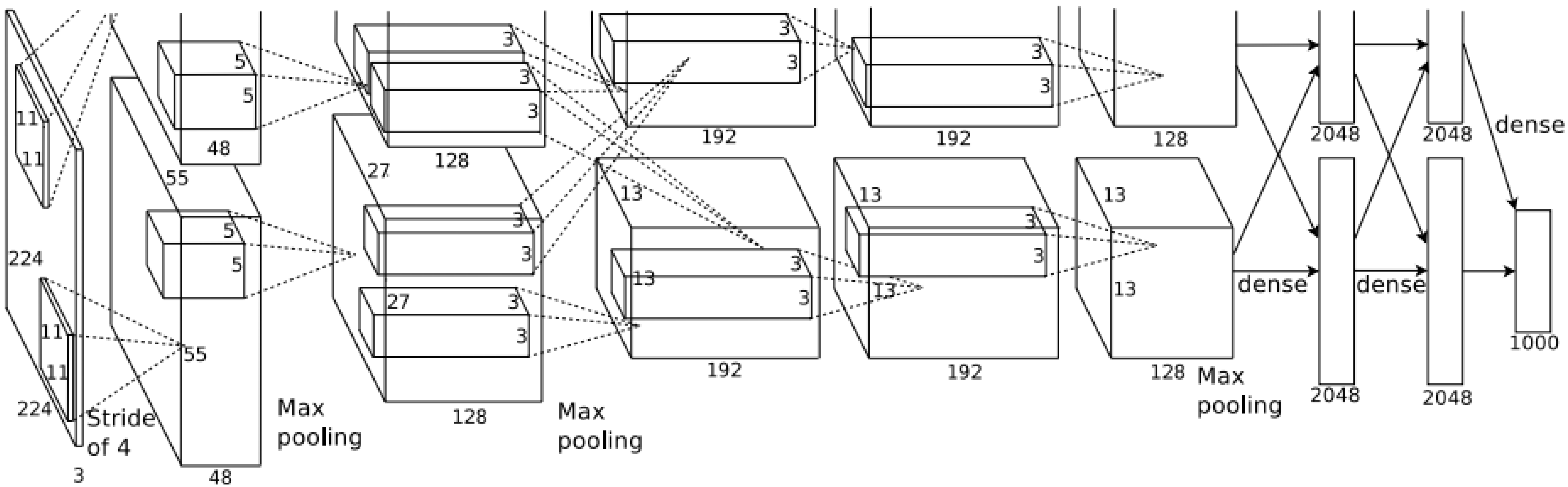
[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

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

# of params: 11*11*3*96 = 35K
Output size: (227-11)/4+1 = 55

[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)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

AlexNet

Two GPUs

Local response normalization $b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a^j_{x,y})^2 \right)^\beta$

ReLU

Data augmentation

Random 227x227 crop from input images

LR flip

RGB color transform

Dropout drop ratio 0.5

SGD

Momentum 0.9

Weight decay 0.0005

7 Models ensemble

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

AlexNet

Local response normalization `tf.nn.lrn`

ReLU `tf.nn.relu`

Data augmentation 직접구현 or

- Random crop `tf.random_crop`
- LR flip `tf.image.random_flip_left_right`
- RGB color transform `tf.image.random_brightness`, `tf.image.random_contrast`

Dropout `tf.nn.dropout`

SGD

Momentum `tf.train.MomentumOptimizer`

Weight decay 직접구현 → L2 regularization on weights

$$L = L_{\text{data}} + \frac{\lambda}{2}||W||_2^2$$

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

AlexNet

Implementation: alexnet.py

Gaussian initialization mean=0, stddev=0.01

Initial learning rate 0.01
  Multiply 0.1 when the validation set accuracy stopped improving

Data preparation
    Image 를 바로 이용할 수 있게 가공할 것인가,
    실시간으로 읽어 이용 할 것인가?

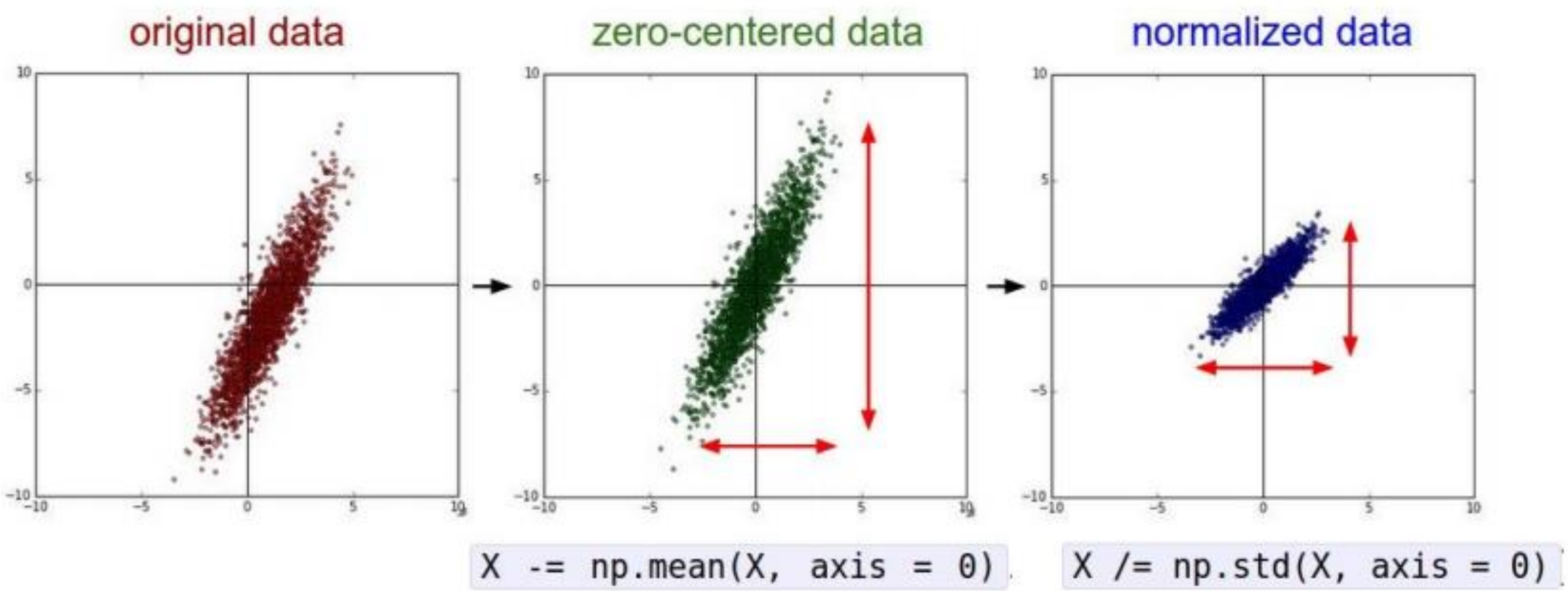
Training / Testing
Save / Restore parameters

Data preprocessing

Subtract

The mean image or

Per-channel mean value

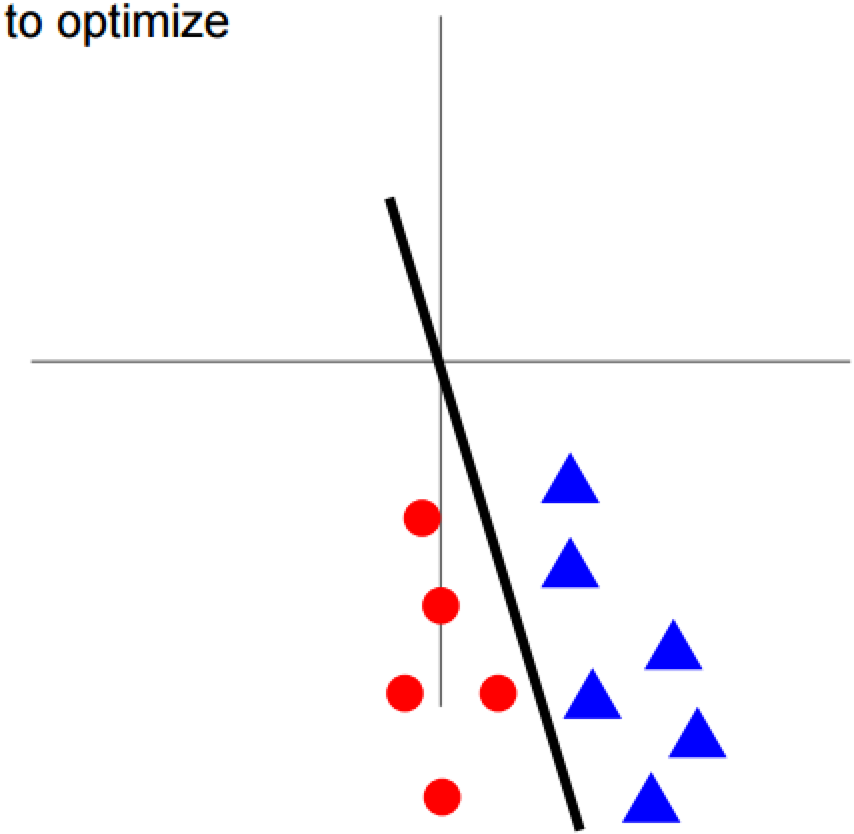


http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

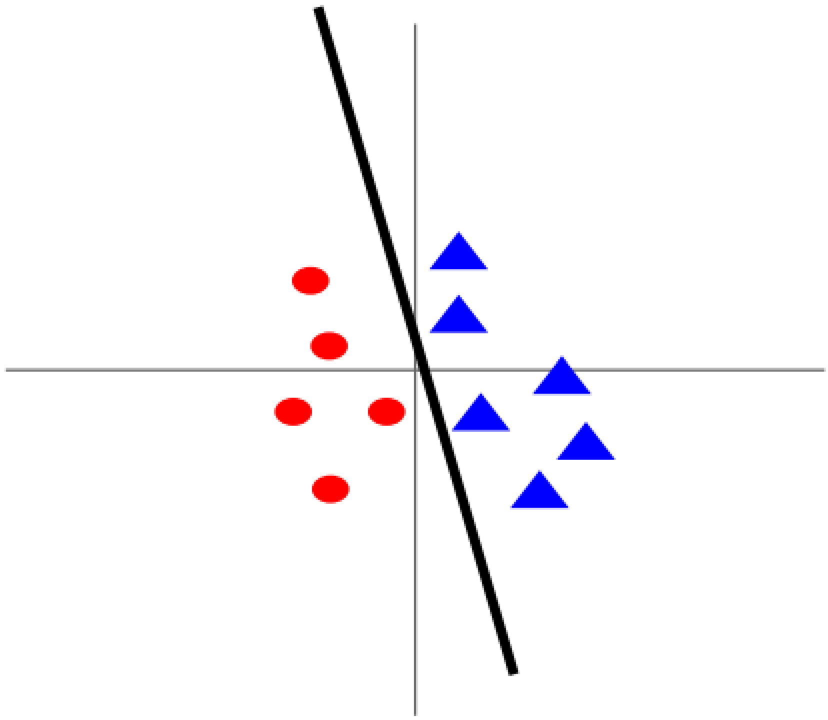
Data preprocessing

Why?

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize



After normalization: less sensitive to small changes in weights; easier to optimize



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

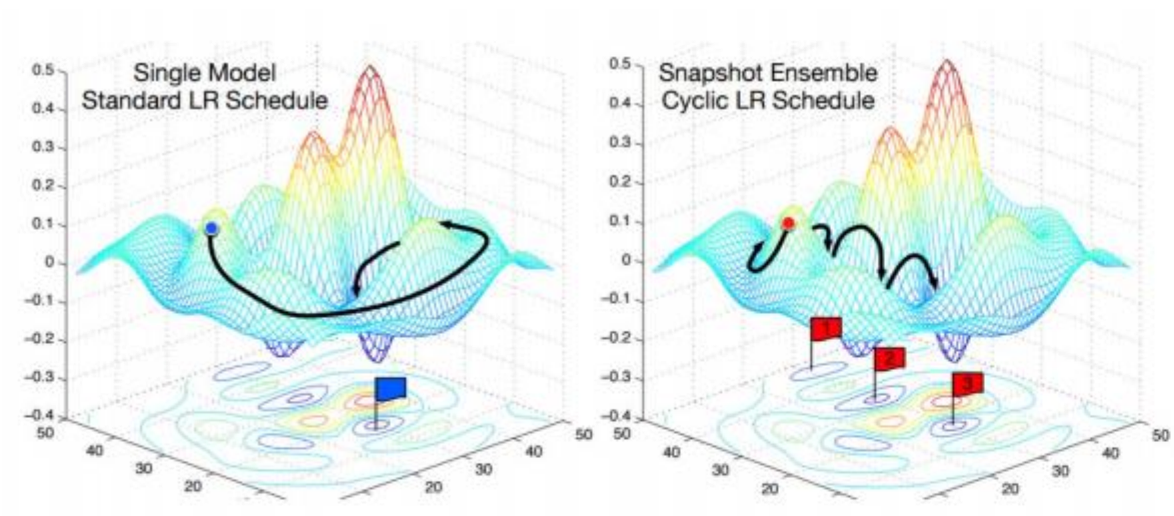
Model ensembles

1. Train multiple independent models
2. At test time average their results

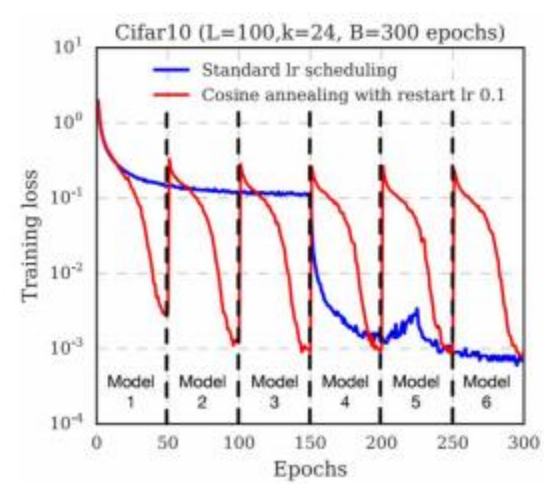
→ **1-2% extra performance**

Trick:

Instead of training independent models,
use multiple snapshots of a single model during training!



Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

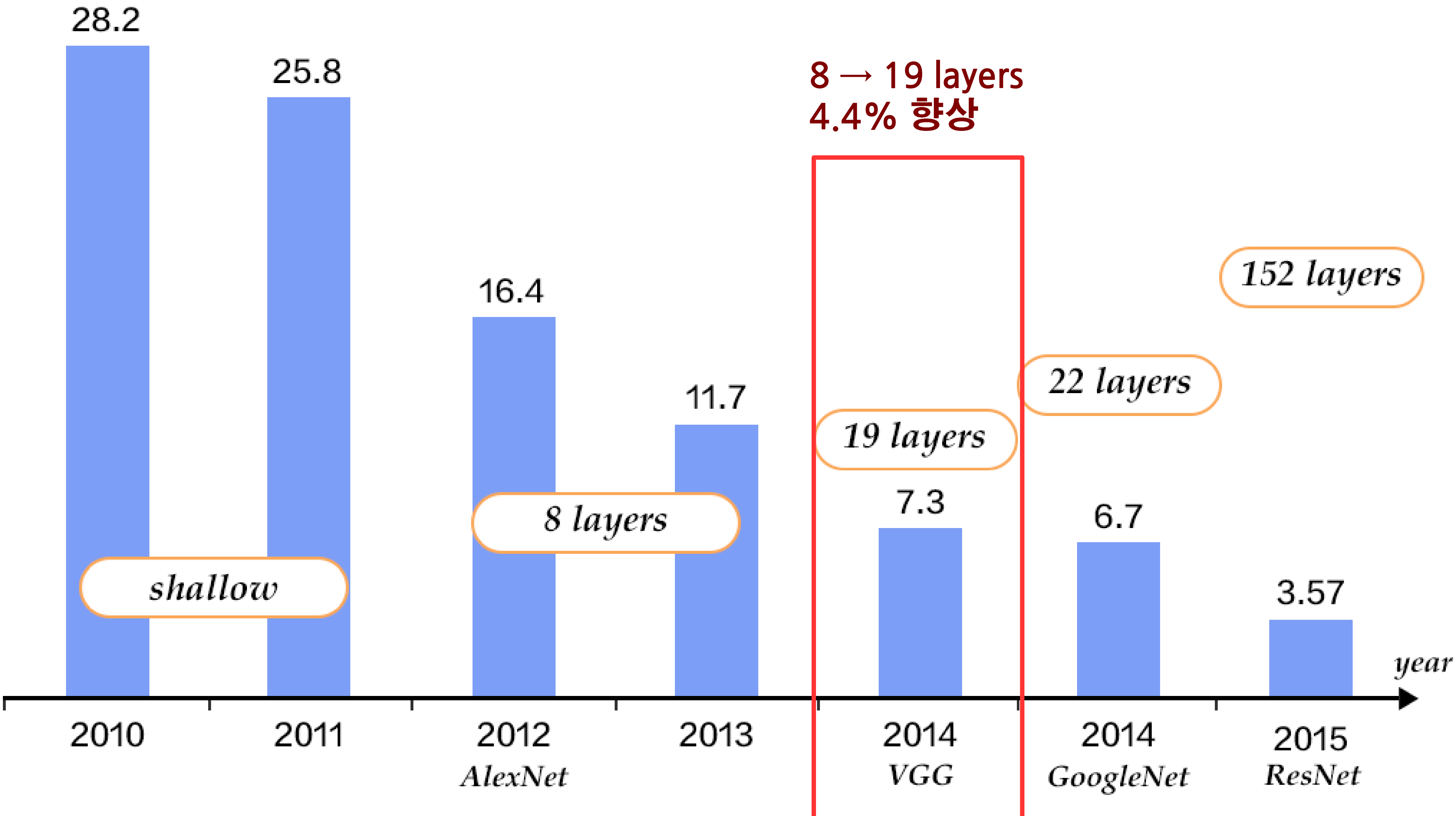


Cyclic learning rate schedules can make this work even better!

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

VGG

Second winner of 2014 ILSVRC image classification challenge



http://book.paddlepaddle.org/03.image_classification/

VGG

VGG16 VGG19

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

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

VGG

Only 3x3 filters, deeper network

Conv: 3x3 filter, stride 1, pad 1
Max pool: 2x2, stride 2

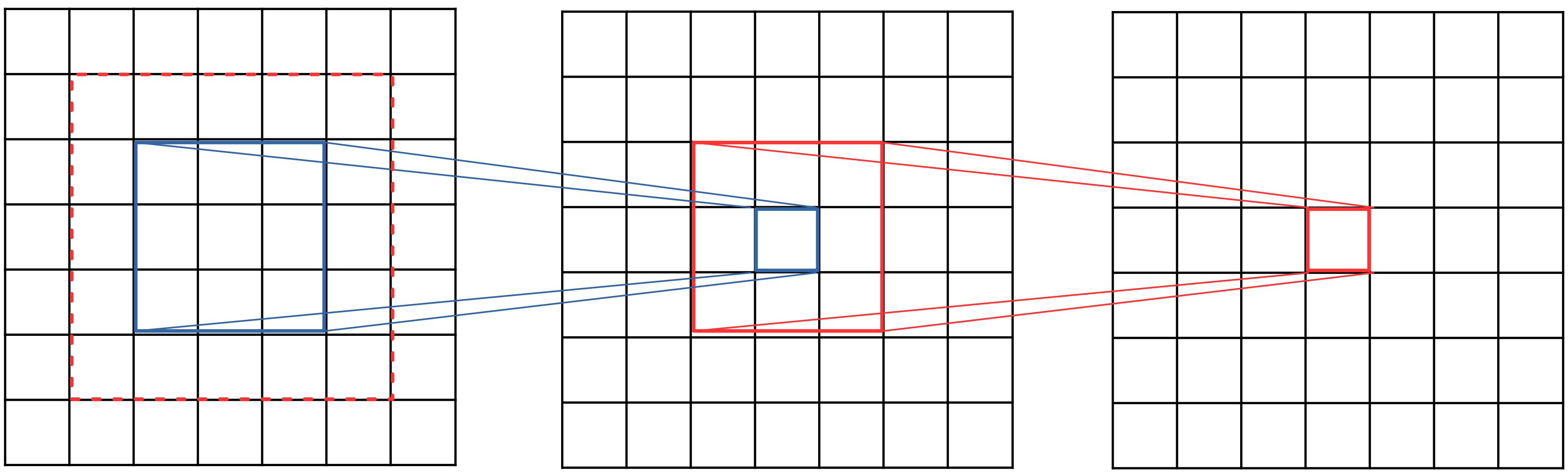
Similar training procedure as AlexNet
No LRN

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

VGG

Why only 3x3 conv layers?

Stack of two 3x3 conv (stride 1) layers has same **effective receptive fie ld** as one 5x5 conv layer



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

VGG

Why only 3x3 conv layers?

Stack of two 3x3 conv (stride 1)

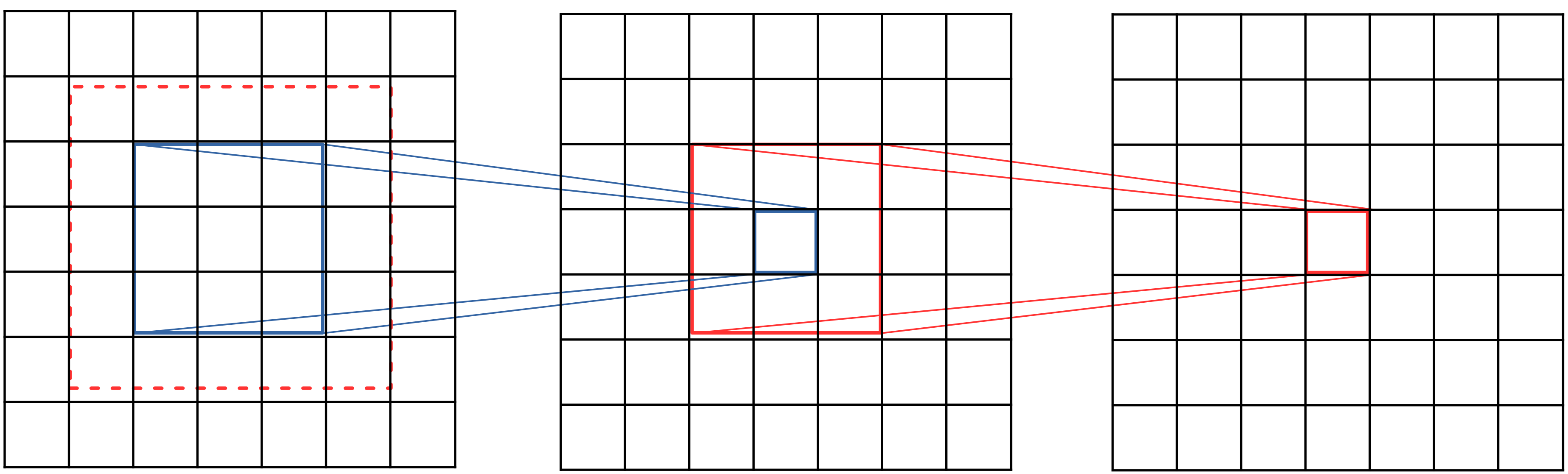
$\# \text{ of params: } 2*(3*3*C_{in}*C_{out})=18*C_{in}*C_{out}$

Fewer params

More nonlinearities

One 5x5 conv layer

$\# \text{ of params: } 5*5*C_{in}*C_{out}=25*C_{in}*C_{out}$



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

VGG

Most memory is used in early Conv layers

(not counting biases)

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

Most params are in FC layers

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

VGG

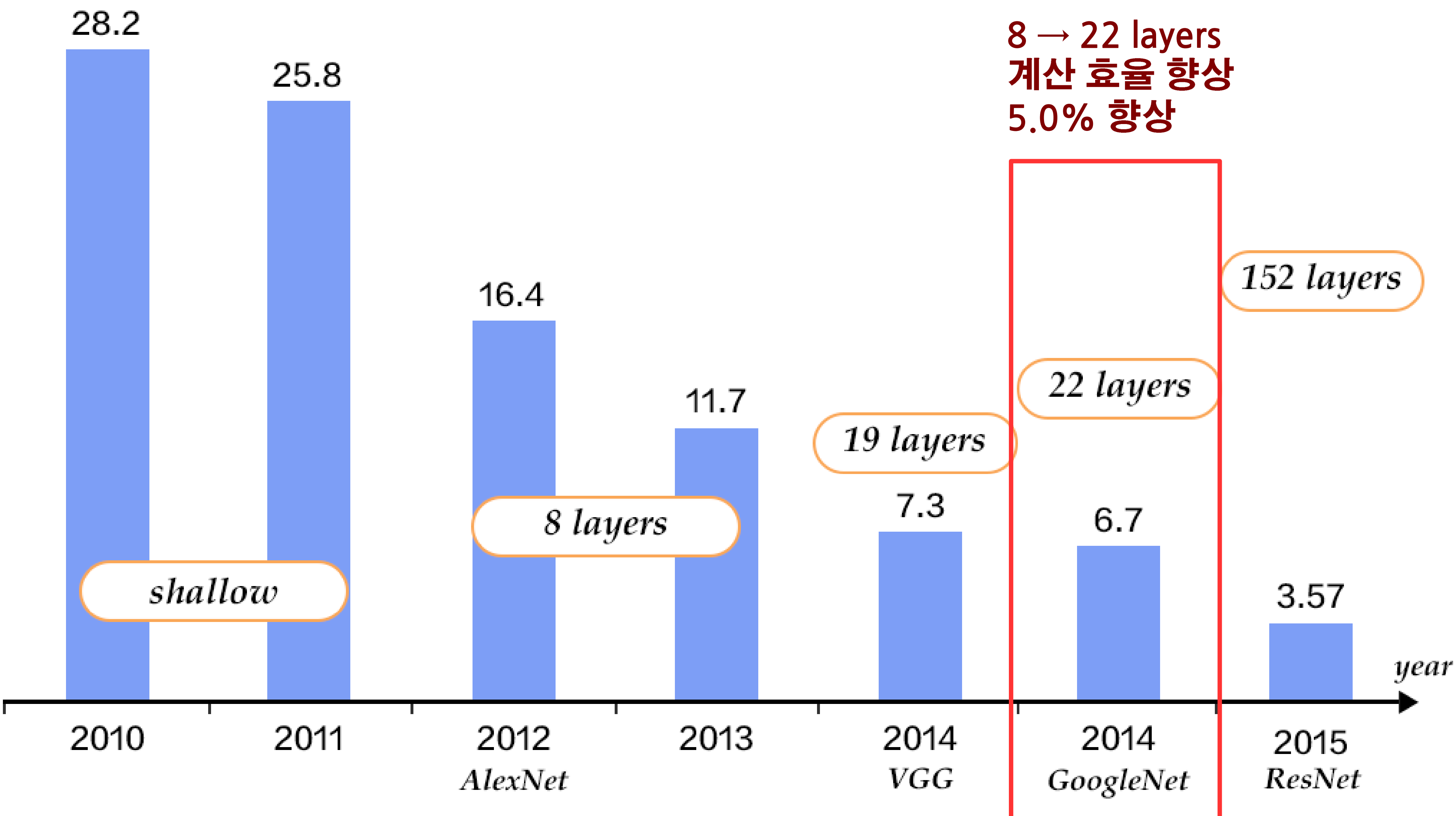
Implementation: vgg16.py

Conv: 3x3 filter, stride 1, pad 1
Max pool: 2x2, stride 2
Gaussian initialization mean=0, stddev=0.01
ReLU
No LRN
Dropout drop ratio 0.5
SGD with Momentum 0.9
Weight decay 0.0005

Initial learning rate 0.01
    Multiply 0.1 when the validation set accuracy stopped improving

GoogLeNet

Winner of 2014 ILSVRC image classification challenge

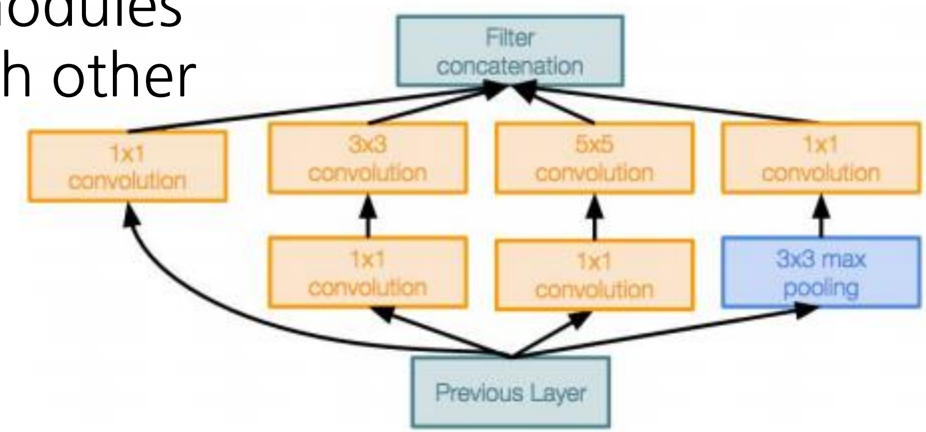


http://book.paddlepaddle.org/03.image_classification/

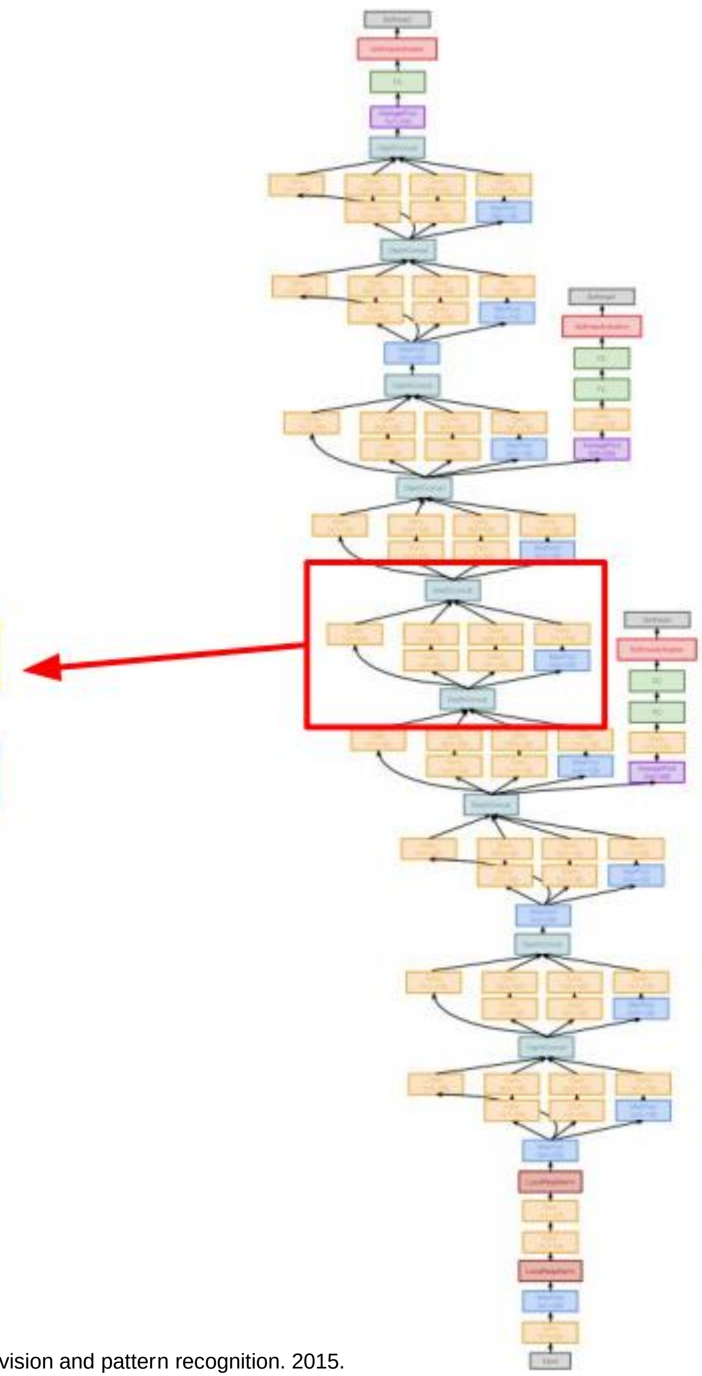
GoogLeNet

No fully connected layers
# of params: 5M (VGG: 138M)

Inception module:
design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module



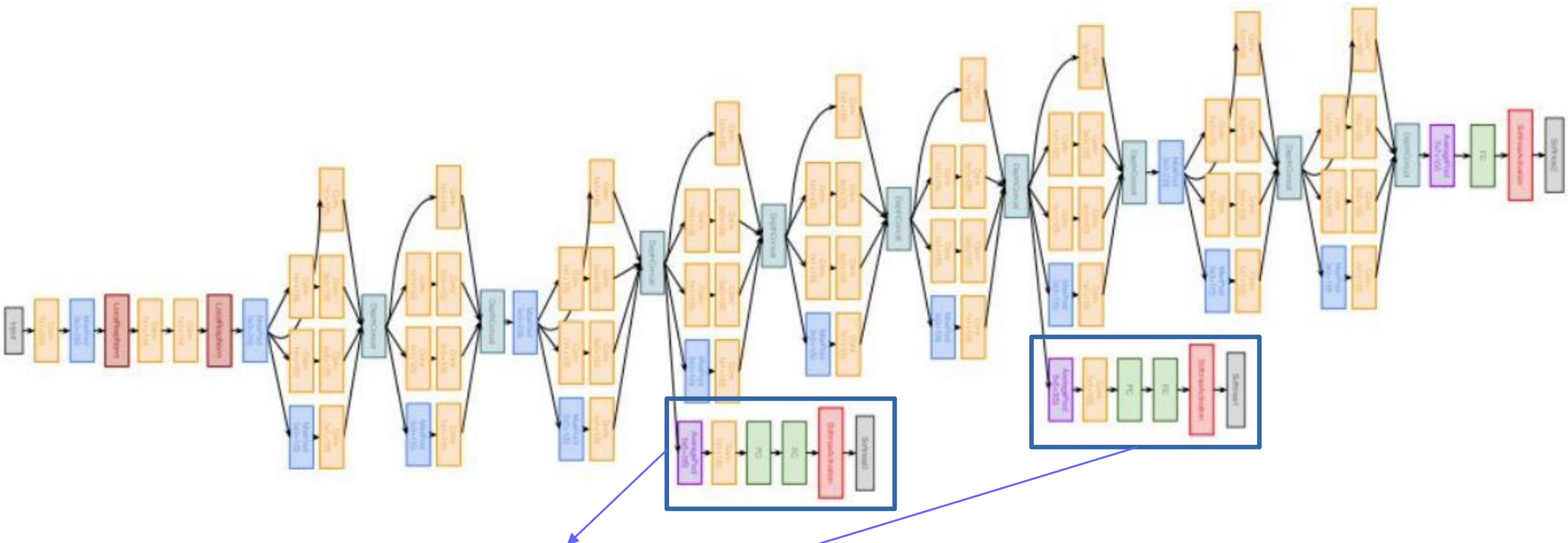
Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

GoogLeNet

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								

Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

GoogLeNet



5x5 AvgPool, stride 3
1x1 Conv 128 filters
FC 1024 units
FC 1024 units
Dropout drop ratio 0.7
Softmax

Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

GoogLeNet

Implementation: googlenet.py

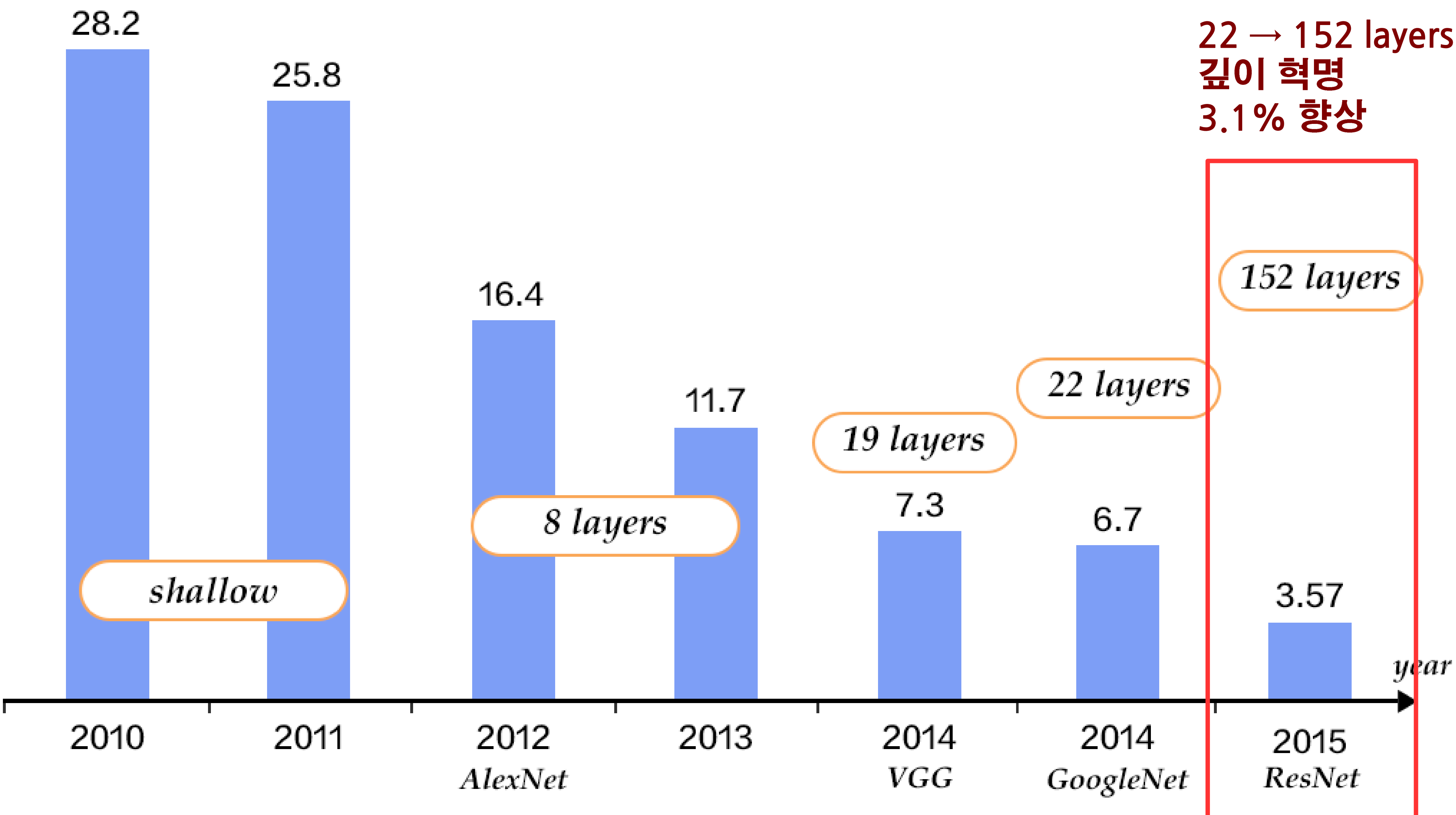
ReLU
Dropout
SGD with Momentum 0.9
Weight decay (may) 0.0002

Initial learning rate (may) 0.01
Multiply 0.96 every 8 epochs

ResNet

Winner of 2015 ILSVRC image classification challenge

22 → 152 layers
깊이 혁명
3.1% 향상



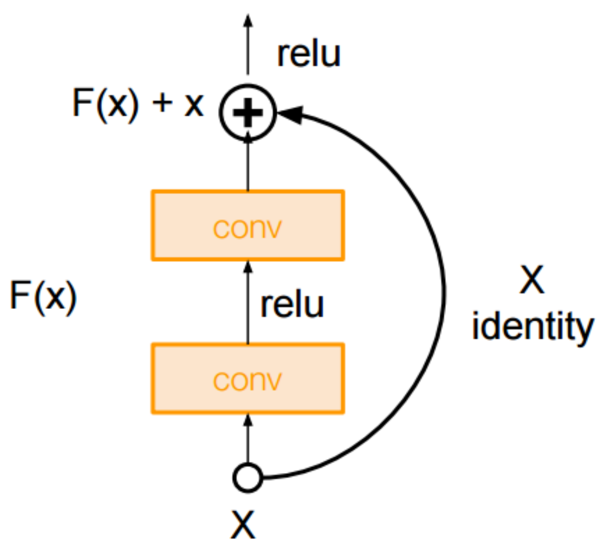
http://book.paddlepaddle.org/03.image_classification/

ResNet

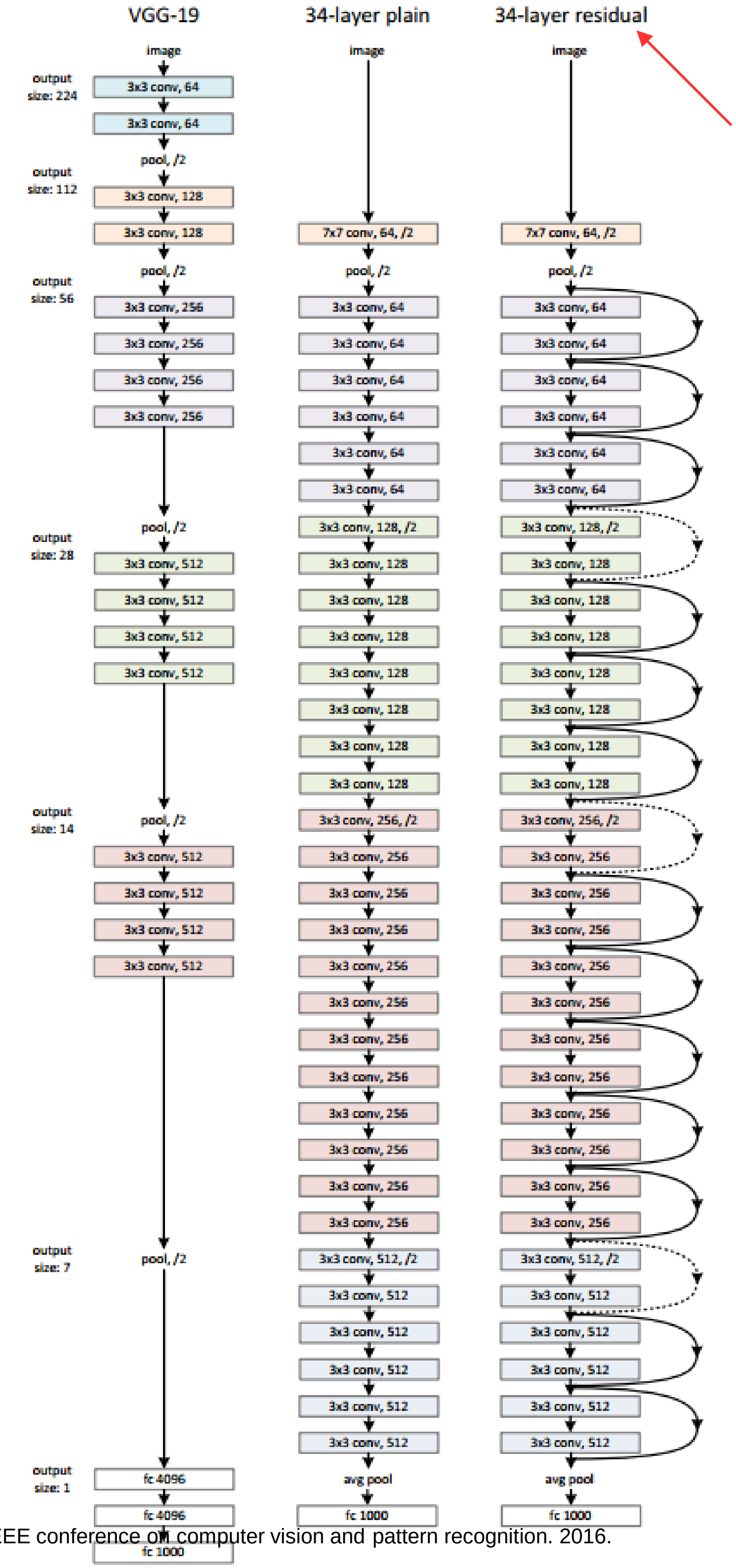
34,50,101,152-layer models

Intuitively, deeper models should perform better than shallower models
But hard to optimize!

Solution: **Residual block**



Residual block



He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

ResNet

Periodically, double the number of filters and downsample spatially us
ing stride 2

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

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

ResNet

Implementation: resnet34.py

Batch normalization after every Conv layer
He initialization
ReLU
No Dropout
SGD with Momentum 0.9
Weight decay 0.00001

Initial learning rate 0.1
    Multiply 0.1 when the validation set accuracy stopped improving

SGD

Cost function: $J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \phi(\mathbf{w}^T\mathbf{x})^{(i)}\right)^2$

Batch gradient descent

The gradient is calculated from the whole training set

$$\Delta\mathbf{w} = -\eta\nabla J = \eta\sum_{i=1}^{N}\left(y^{(i)} - \phi(\mathbf{w}^T\mathbf{x})^{(i)}\right)\mathbf{x}^{(i)}$$

Stochastic gradient descent

The gradient is calculated from a single sample

$$\Delta\mathbf{w} = -\eta\nabla J = \eta\left(y^{(i)} - \phi(\mathbf{w}^T\mathbf{x})^{(i)}\right)\mathbf{x}^{(i)}$$

Mini-batch gradient descent

The gradient is calculated from a mini-batch (more than one training sample)

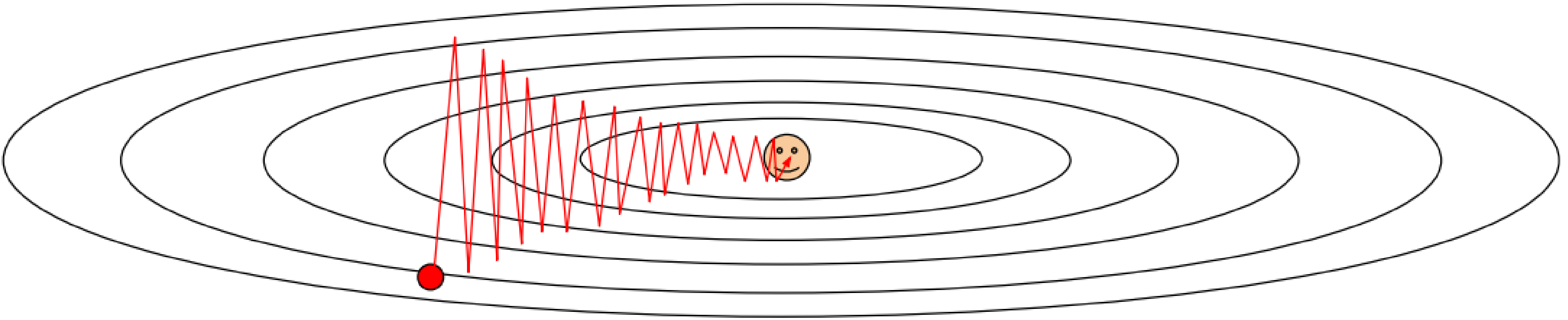
1. Choose an initial vector of parameters $\mathbf{w}$ and learning rate η.
2. Repeat until an approximate minimum is obtained:
   1. Randomly shuffle examples in the training set.
   2. For $i = 1, 2, \ldots, n$ do:

      $w := w + \Delta\mathbf{w}$

http://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

Problems with SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



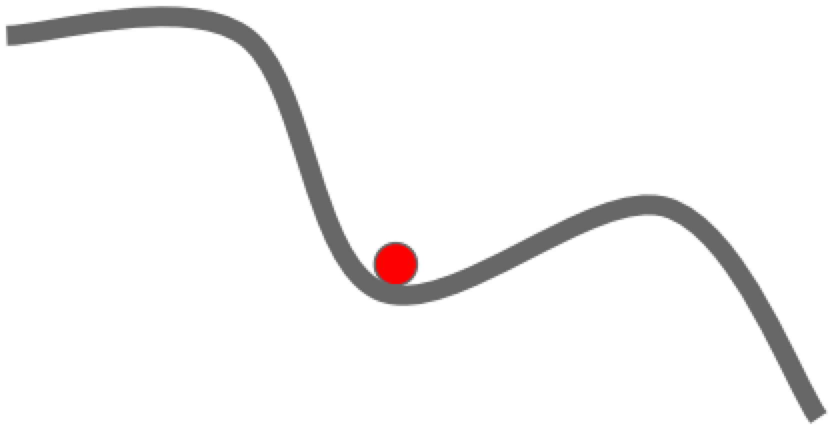
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

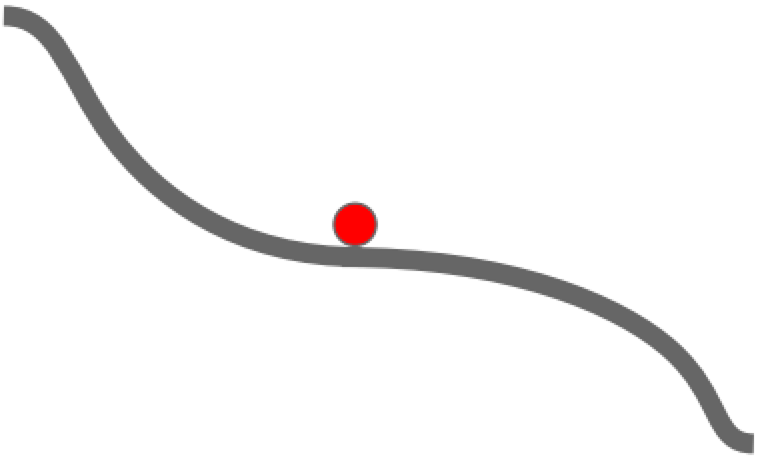
Problems with SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

Saddle points much more common in high dimension





http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

SGD + Momentum

관성

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

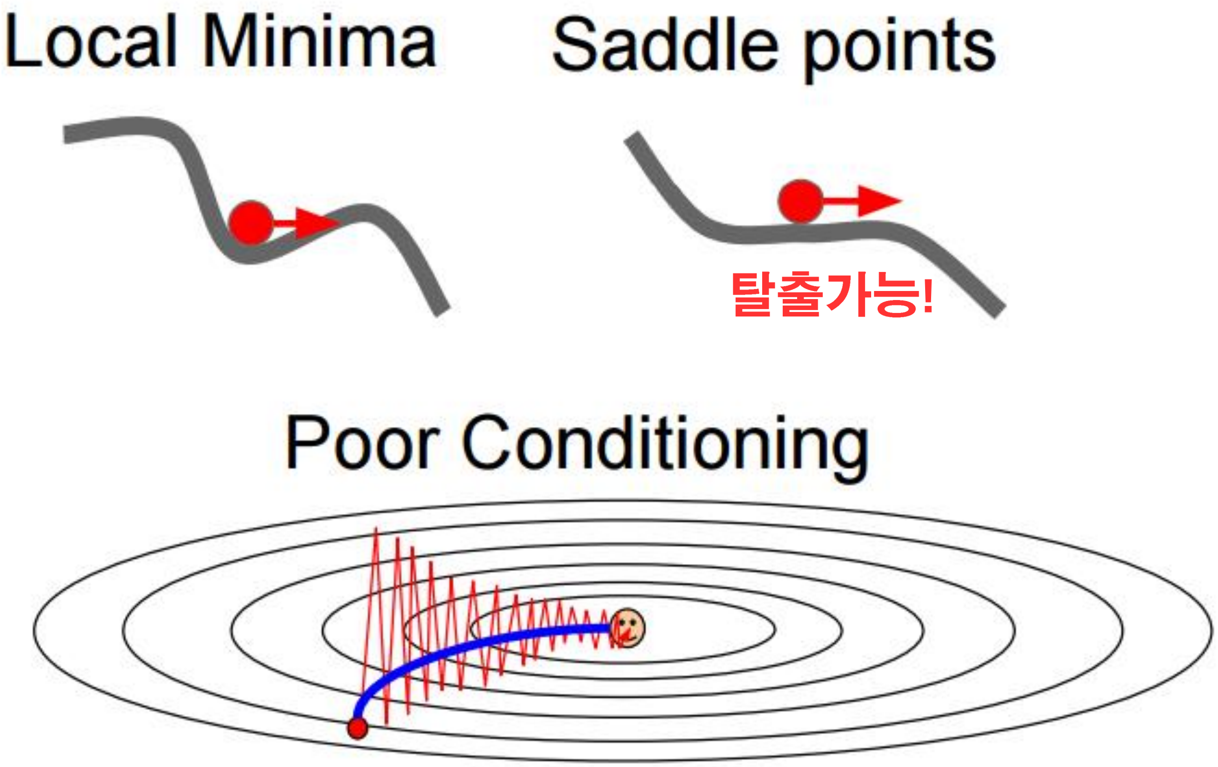
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

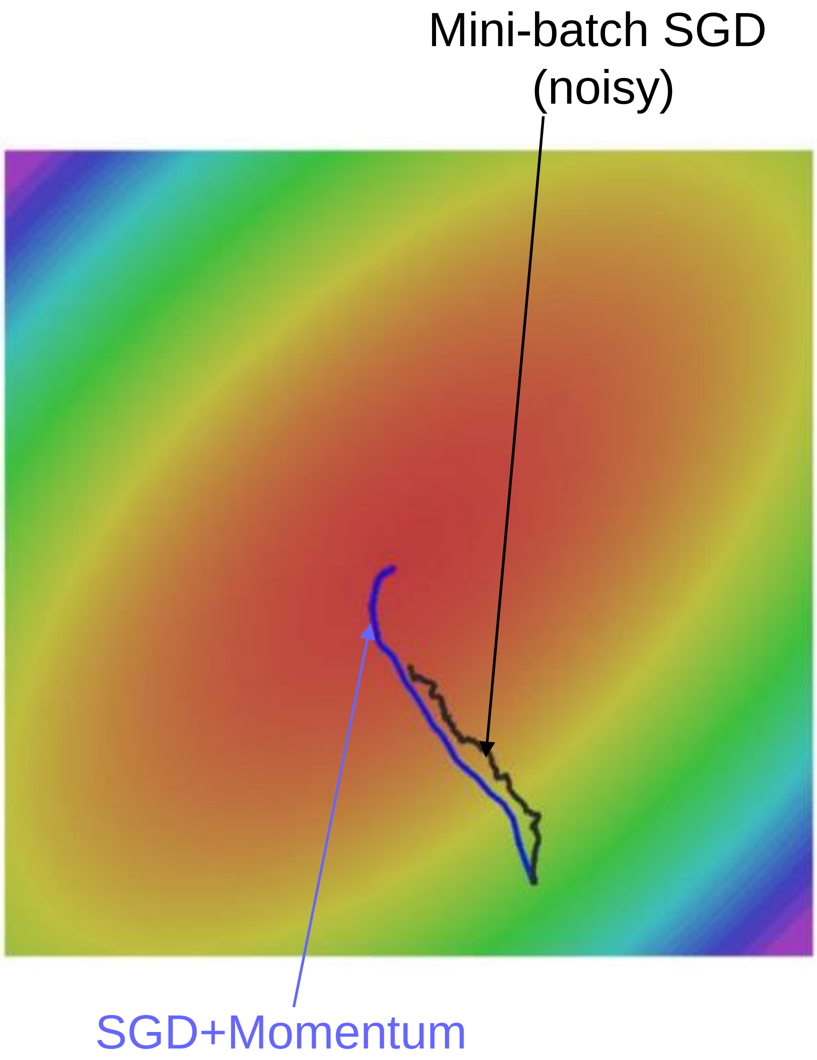
```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

SGD + Momentum





http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Nesterov momentum

SGD

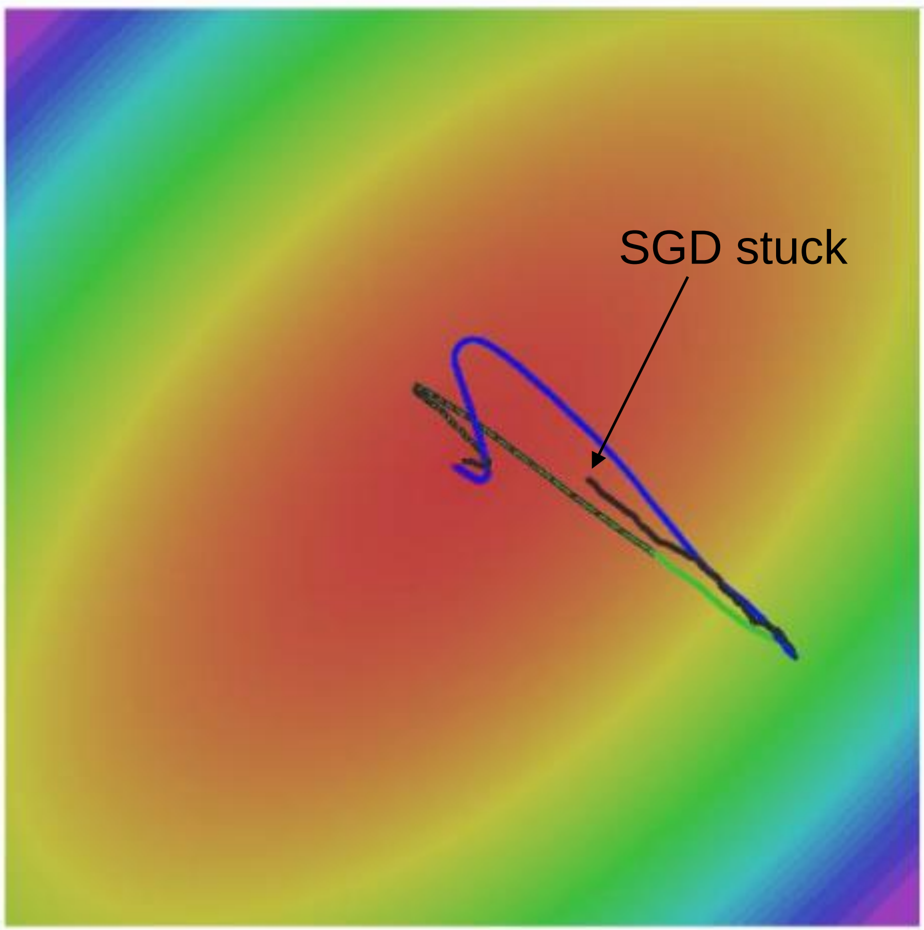
$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Nesterov

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$



Momentum 방식의 경우 멈춰야 할 시점에서도 관성에 의해 훨씬 멀리 갈수도 있다는 단점이 존재하는 반면, Nesterov 방식의 경우 일단 모멘텀으로 이동을 반정도 한 후 어떤 방식으로 이동해야할 지를 결정한다. 따라서 Momentum 방식의 빠른 이동에 대한 이점은 누리면서도, 멈춰야 할 적절한 시점에서 제동을 거는 데에 훨씬 용이하다고 생각할 수 있을 것이다.

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Adagrad

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

Adaptive gradient

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

‘지금까지 많이 변화하지 않은 변수들은(G_t ↑) step size를 크게 하고 ,
지금까지 많이 변화했던 변수들은(G_t ↓) step size를 작게 하자 ’

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

Learning rate decay 를 신경쓰지 않아도 된다.
But, 학습을 오래하면 step size가 너무 작아진다.

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

RMSProp T. Tieleman, and G. Hinton. RMSProp: Divide the gradient by a running average of its recent magnitude.

Adagrad의 단점을 보완

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2 \longrightarrow G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

G 가 무한정 커지는 것을 방지
감마의 기본값은 0.99

http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf

Adam

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Momentum + RMSProp + Bias correction

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

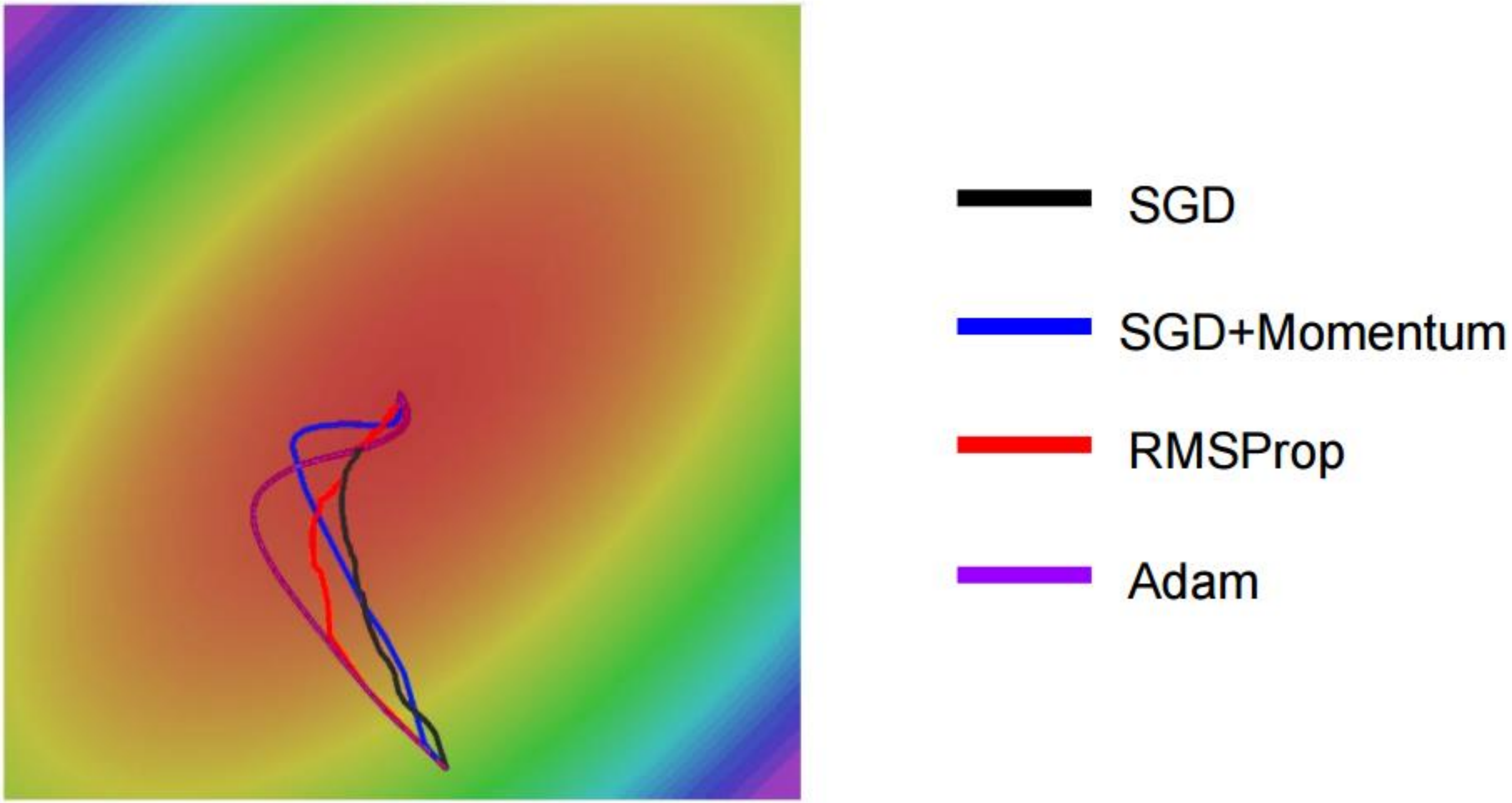
AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!

Adam

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015



Animation: http://cs231n.github.io/neural-networks-3/

Optimizers in TF

Optimizers

The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.

You never instantiate the Optimizer class itself, but instead instantiate one of the subclasses.

- `tf.train.Optimizer`
- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer` ← Most popular choice nowadays
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`

https://www.tensorflow.org/api_guides/python/train#Optimizers