CNN in practice

I 고려대학교 산업경영공학과
I 서덕성

INDEX

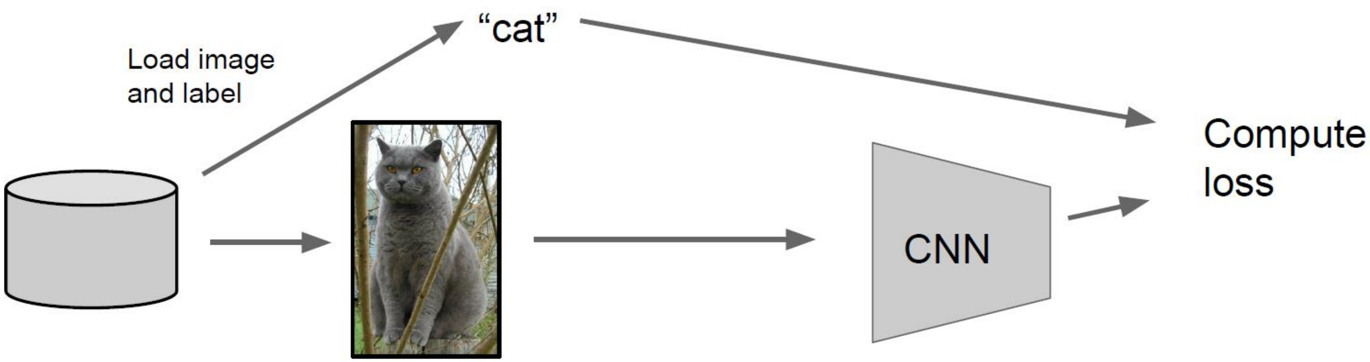
- Making the most of your data
  - Data augmentation
  - Transfer learning
- All about convolutions
  - How to arrange them
  - How to compute them fast
- Implementation details
  - GPU / CPU
  - Bottleneck
  - Floating procedure

Making the most of your data

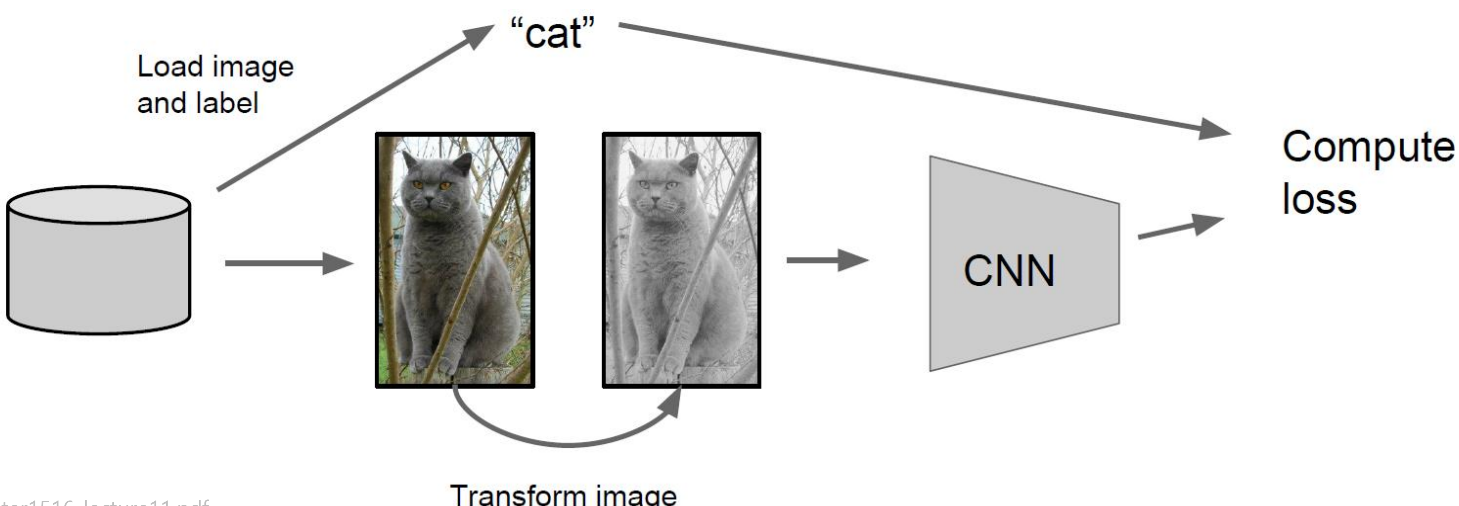
Data augmentation

❖ Original CNN

- 1개의 image와 1개의 label이 모델 학습시 input으로 들어감



- 많은 image는 없지만, 많은 척 하고싶다 -> Data augmentation

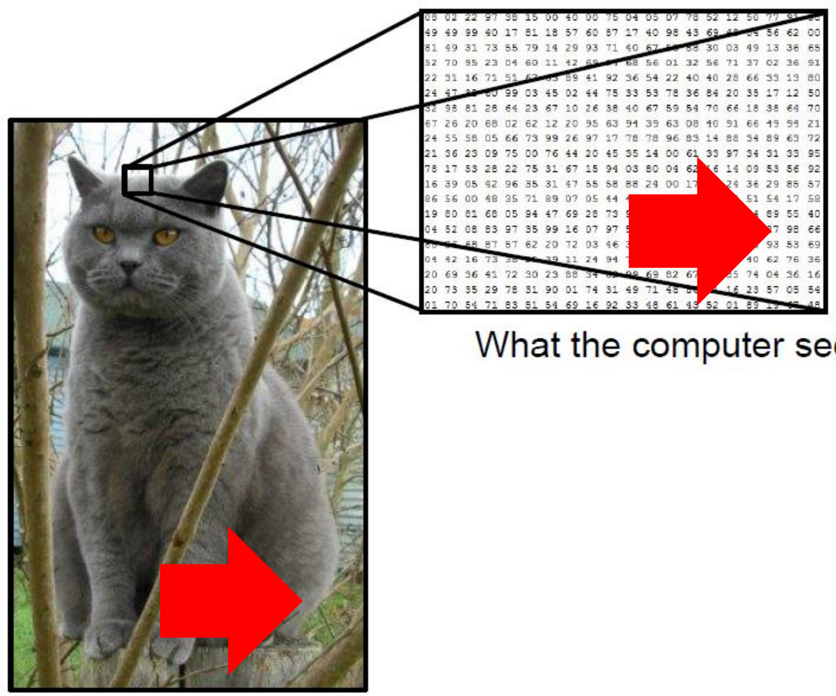


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data augmentation

❖ Concept

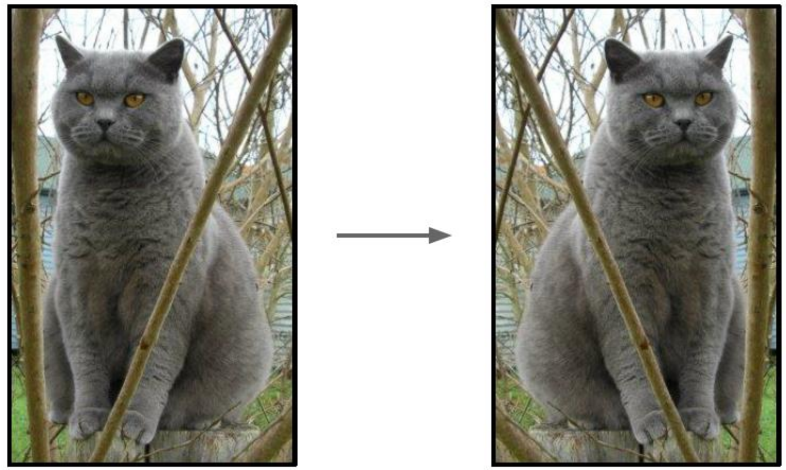
- Label 변화 없이 픽셀값(독립변수)만 변화를 주자
- 변화 준 데이터까지도 학습에 사용하자
- 매우 많이 활용되고 있음



What the computer sees

Data augmentation

❖ Horizontal flips

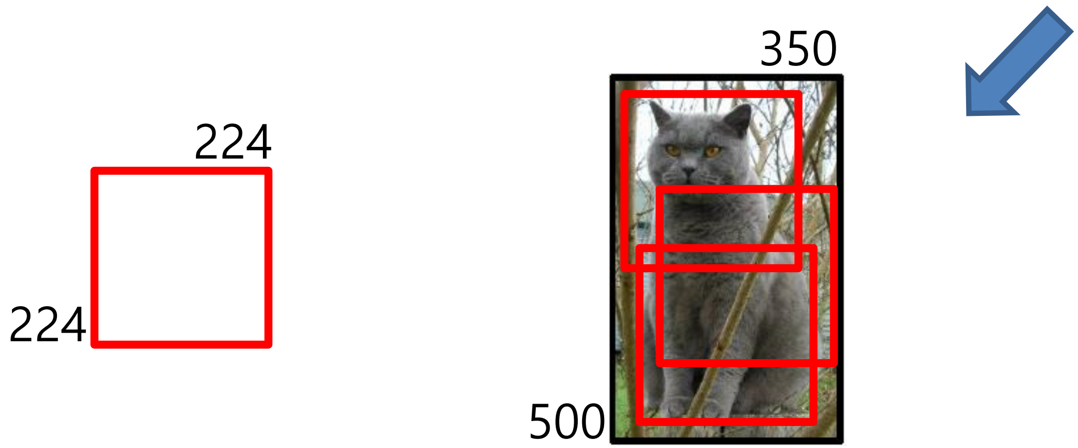


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data augmentation

❖ Random crops/scales

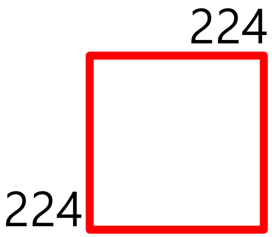
- ResNet 에서 사용한 방법
    - Training : sample random crops / scales
        1. Pick random L in range [256, 480]
        2. Resize training image, short side = L
        3. Sample random 224 x 224 patch

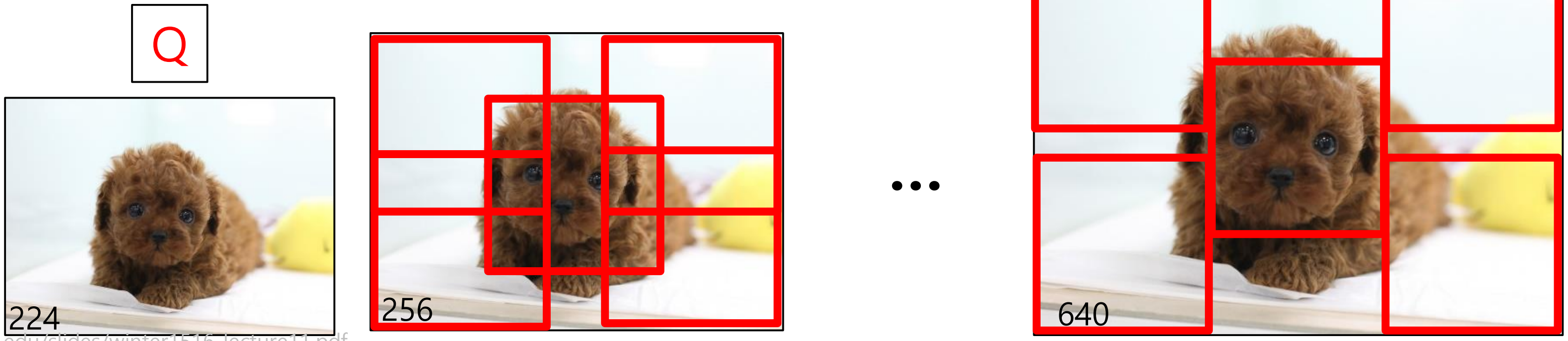


Data augmentation

❖ Random crops/scales

- ResNet 에서 사용한 방법
  - Training : sample random crops / scales
    1. Pick random L in range [256, 480]
    2. Resize training image, short side = L
    3. Sample random 224 x 224 patch
  - Test : average a fixed set of crops
    1. Resize image at 5 scales : {224, 256, 384, 480, 640}
    2. For each size, use 10 224x224 crops
       : 4 corners + center, + flips





Data augmentation

ImageNet Classification with Deep Convolutional Neural Networks

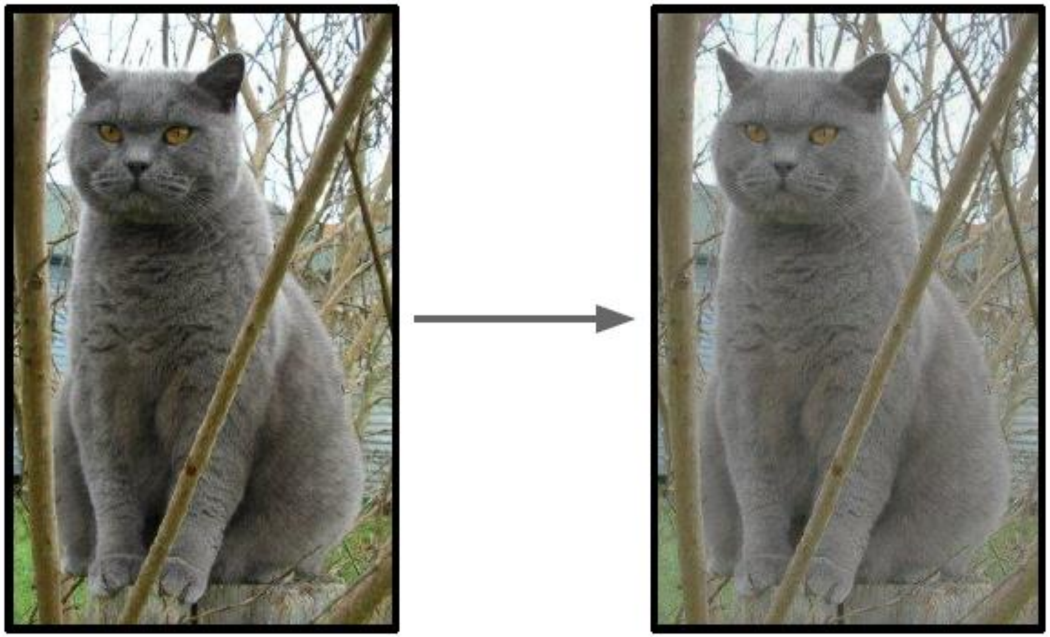
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

❖ Color jittering

- 가장 쉬운 방법 : 명도, 채도 등을 랜덤하게 변화시킴



- 상대적으로 복잡한 방법 : PCA를 이용
  1. [R, G, B] 3층을 가진 모든 training set의 pixel에 대해 PCA 적용
  2. PCA로부터 “color offset”을 추출
  3. 모든 픽셀에 대해 offset을 더함

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Data augmentation

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

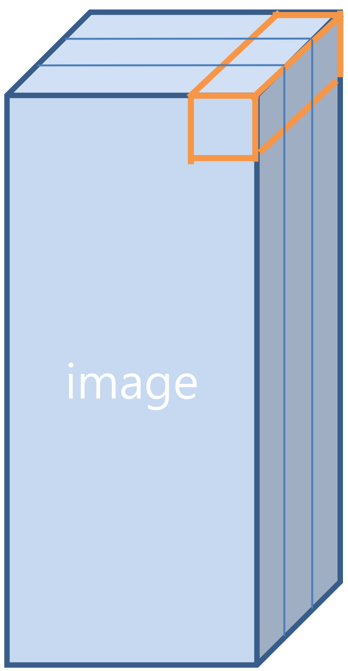
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

❖ Color jittering

- 상대적으로 복잡한 방법 : PCA를 이용
  1. [R, G, B] 3층을 가진 모든 training set의 pixel에 대해 PCA 적용
  2. PCA로부터 “color offset”을 추출
  3. 모든 픽셀에 대해 offset을 더해줌

$$\begin{bmatrix} I^R_{xy} \\ I^G_{xy} \\ I^B_{xy} \end{bmatrix} = \begin{bmatrix} I^R_{xy} \\ I^G_{xy} \\ I^B_{xy} \end{bmatrix} + [p_1\ p_2\ p_3] \begin{bmatrix} \alpha_1\lambda_1 \\ \alpha_2\lambda_2 \\ \alpha_3\lambda_3 \end{bmatrix}$$

Color jittering



All pixels

	R	G	B
1			
2			
3			
…			
N			

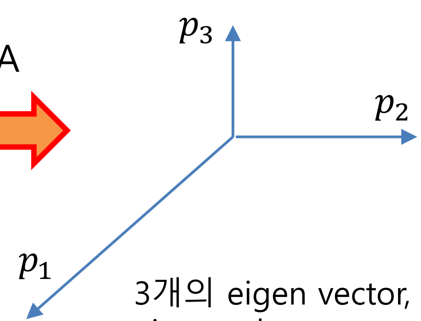
COV. matrix

	R	G	B
R			
G			
B			

p_i: i-th 고유벡터
λ_i: i-th 고유값
α_i: i-th random variable (image마다 뽑음)

$$\alpha_i \sim N(0, 0.1^2)$$

PCA

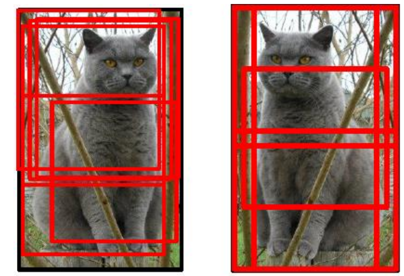


3개의 eigen vector, eigen value

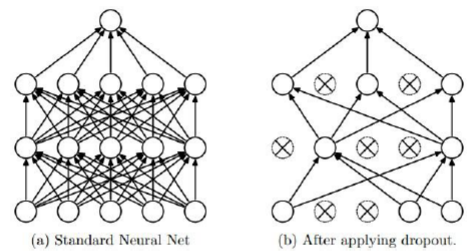
Data augmentation

❖ Get creative

- Random mix/combinations of
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - Lens distortions …(go crazy)



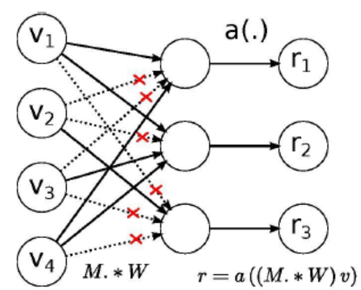
Data Augmentation



Dropout

❖ General theme

- Training : Add random noise (regularizor로써 overfitting 막기 좋음)
  - Data augmentation
  - Dropout
  - DropConnect
  - Batch normalization
  - Model ensembles



DropConnect

- Test : Marginalize over the noise (종합)

Data augmentation

01
02
03
04
05
06
07

❖ Data augmentation summary

- Simple to implement, use it
- Especially useful for small datasets
- Fits into framework of noise / marginalization

Transfer learning

01
02
03
04
05
06
07

❖ CNN 학습시 한계

- 충분한 데이터가 없음
- 학습이 너무 오래 걸림 (VGGNet의 경우 2~3주)
- 최고 성능의 model을 internet에서 얻을 수 있음 -> 이를 활용하자!

❖ 상황별 Transfer learning 적용

- 데이터의 양
- 모델 학습시 사용한 데이터와 내가 가진 데이터의 유사성

	Similar	Difference
Very little data		
Quite a lot of data		

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Transfer learning

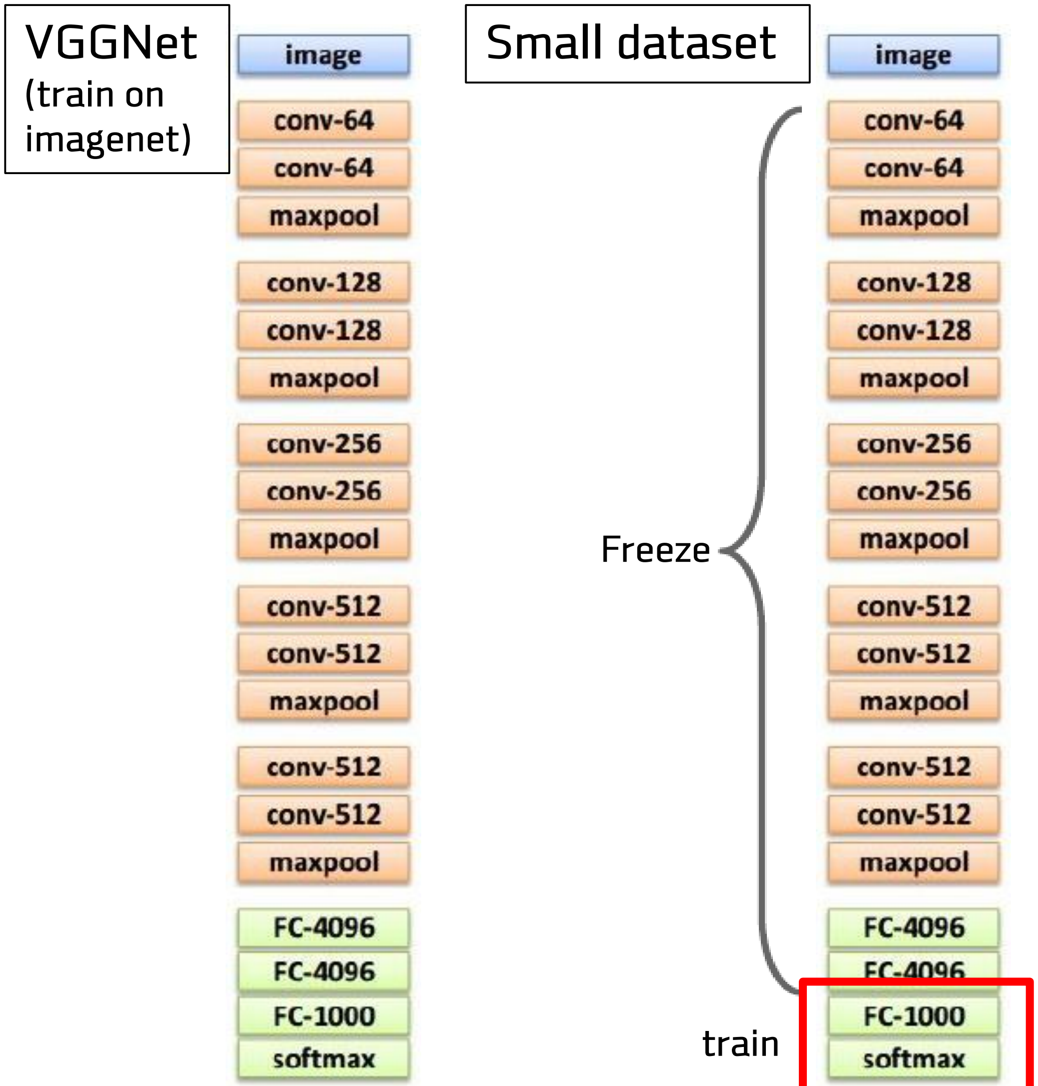
❖ With VGGNet

VGGNet
(train on imagenet)

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Transfer learning

❖ With VGGNet



Freeze 구간에서는 W, b를 그대로 사용

- 즉, learning rate = 0
- 초반부 층에서는 점, 선, 색 등
  이미지에 모두 이용가능한 feature가 있음
- 후반부 층에는 training image에 종속적인
  feature가 있음
  -> 유사한 데이터가 조금 있을 때 유용함

train 구간에서는 W, b를 새로 만듦

- 문제에 맞춰서 FC-10이나 SVM으로 변경

Transfer learning

❖ With VGGNet

VGGNet
(train on imagenet)

Train 구간 중 오렌지색(Conv 상위) 부분까지 학습

- 후반부 층에는 training image에 종속적인 feature가 있음
  -> 나도 데이터가 좀 있으니 내 데이터에 적합하자

문제점 :
초반에는 큰 Loss로 인해 오렌지색 부분이 크게 흔들리는 경우가 있음

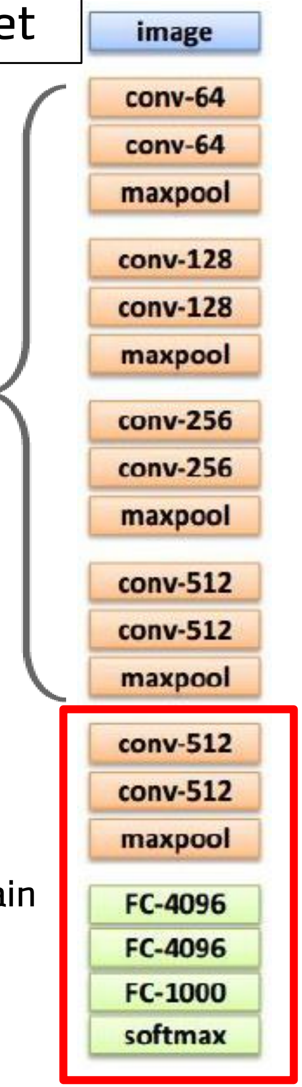
해결방법 :

1. 오렌지 train부분을 잠시 freeze하고
   초록색 부분만 학습한 뒤(Loss 좀 낮추고)
   적절한 타이밍에 freeze를 풀어서 학습
2. (강의에서 준 tip)
   원래 VGGNet의 learning rate값에 대해
   초록색은 $\frac{1}{10}$, 오렌지색은 $\frac{1}{100}$을 주어 학습
   -> 안정화 됨을 경험적으로 확인

Medium dataset

Freeze

train

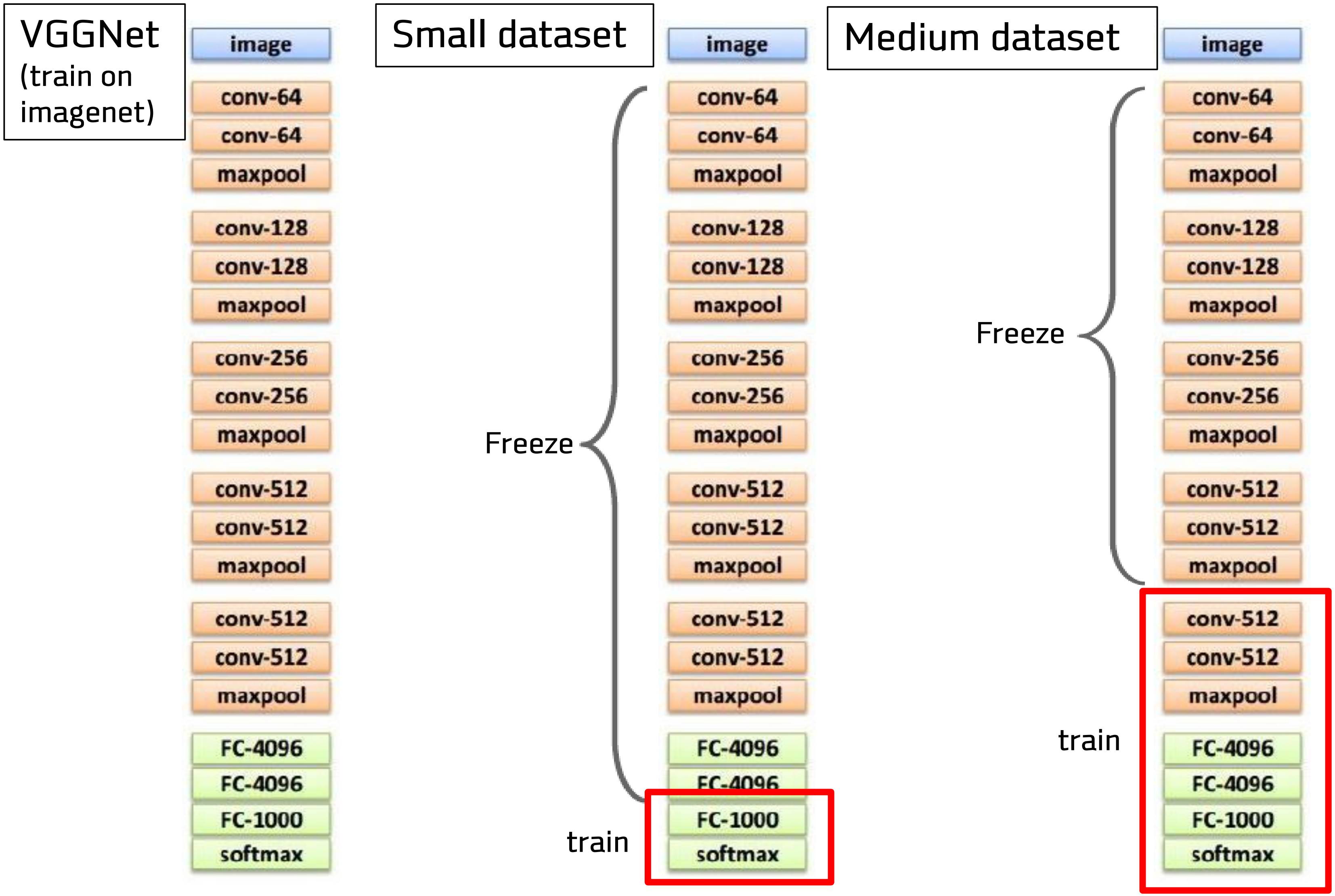


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Transfer learning

01
02
03
04
05
06
07

❖ With VGGNet



http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Transfer learning

❖ 상황별 Transfer learning 적용

- 데이터의 양
- 모델 학습에 사용한 데이터와 내가 가진 데이터의 유사성

	Similar	Difference
Very little data	Use Linear classifier on top layer	In trouble…
Quite a lot of data	Finetune a few layers	Finetune a larger number of layers

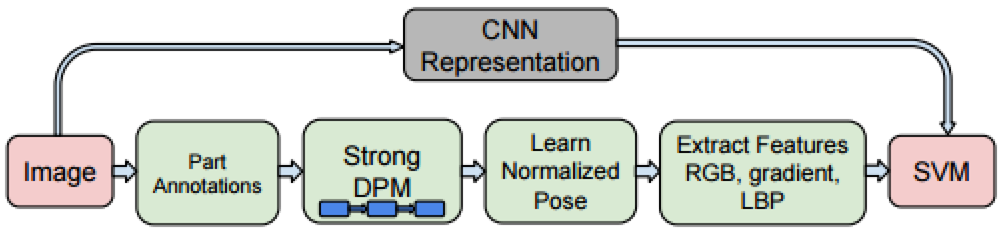
http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

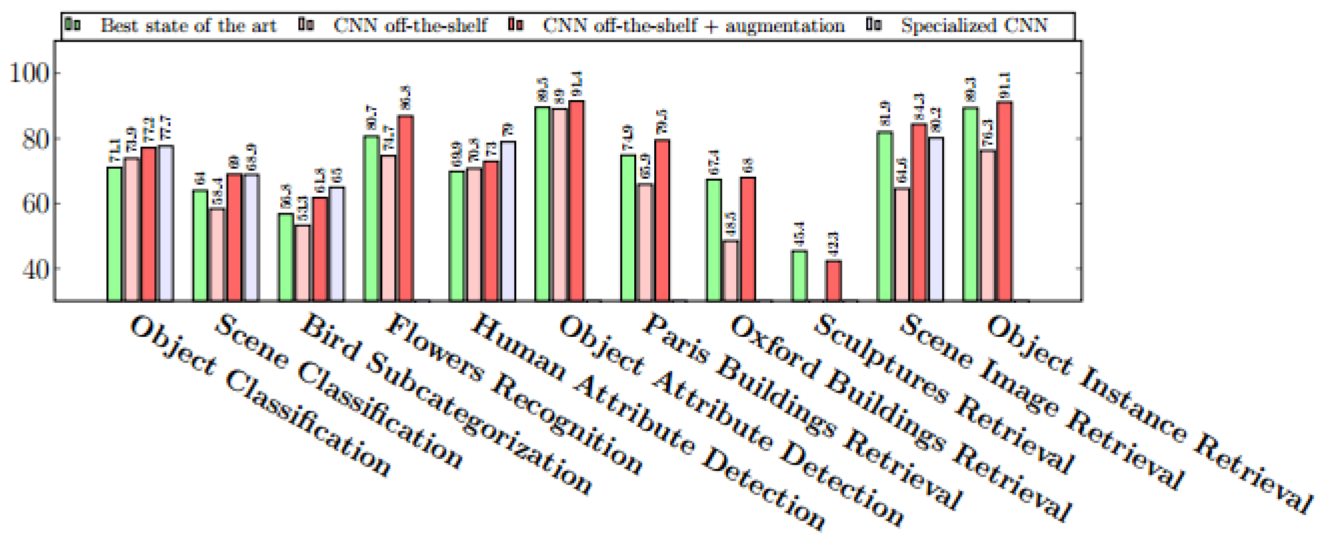
Transfer learning

❖ 성능

- CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

using simple augmentation techniques e.g. jittering. The results strongly suggest that features obtained from deep learning with convolutional nets should be the primary candidate in most visual recognition tasks.

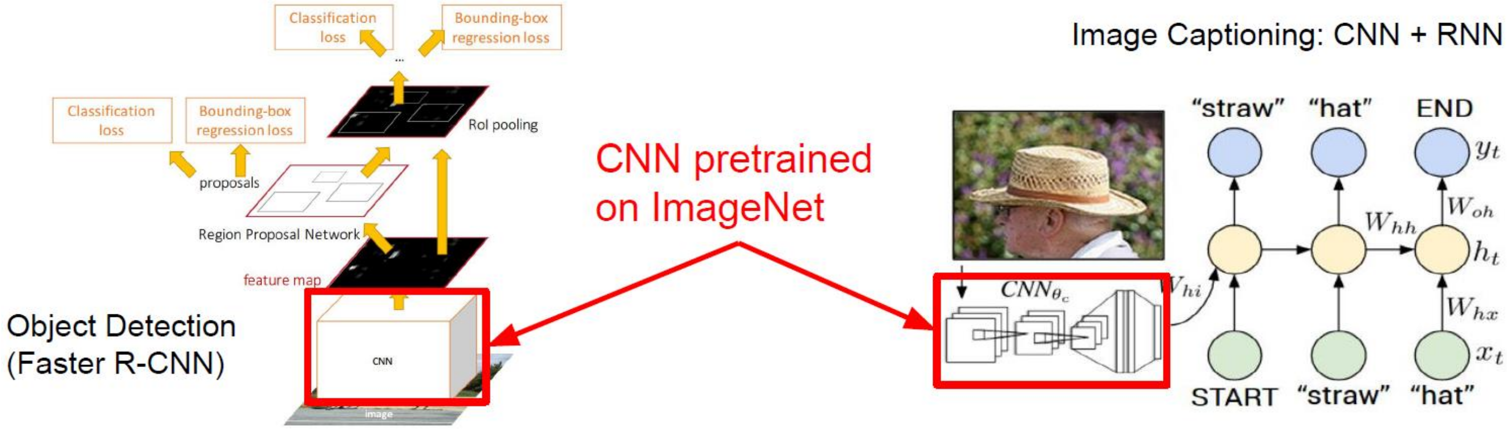




Transfer learning

❖ 사용 예

- Object detection, Image captioning 등 CNN layer에서 활용 가능

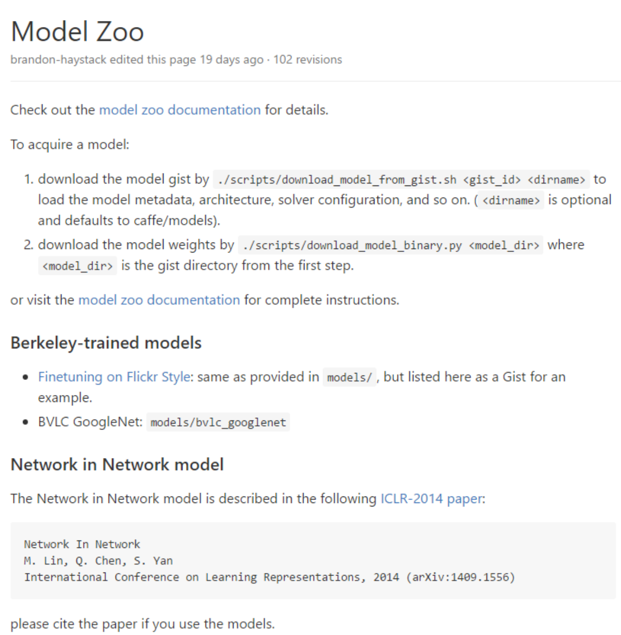


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Transfer learning

❖ Summary (have some dataset < ~ 1M)

- Find a very large dataset that has similar data, train a big ConvNet there
- Transfer learn to your dataset
- Caffe ConvNet library has a **"Model Zoo"** of pretrained models: https://github.com/BVLC/caffe/wiki/Model-Zoo



Model Zoo

brandon-haystack edited this page 19 days ago · 102 revisions

Check out the model zoo documentation for details.

To acquire a model:

1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to caffe/models).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the model zoo documentation for complete instructions.

Berkeley-trained models

- Finetuning on Flickr Style: same as provided in `models/`, but listed here as a Gist for an example.
- BVLC GoogleNet: `models/bvlc_googlenet`

Network in Network model

The Network in Network model is described in the following ICLR-2014 paper:

```
Network In Network
M. Lin, Q. Chen, S. Yan
International Conference on Learning Representations, 2014 (arXiv:1409.1556)
```

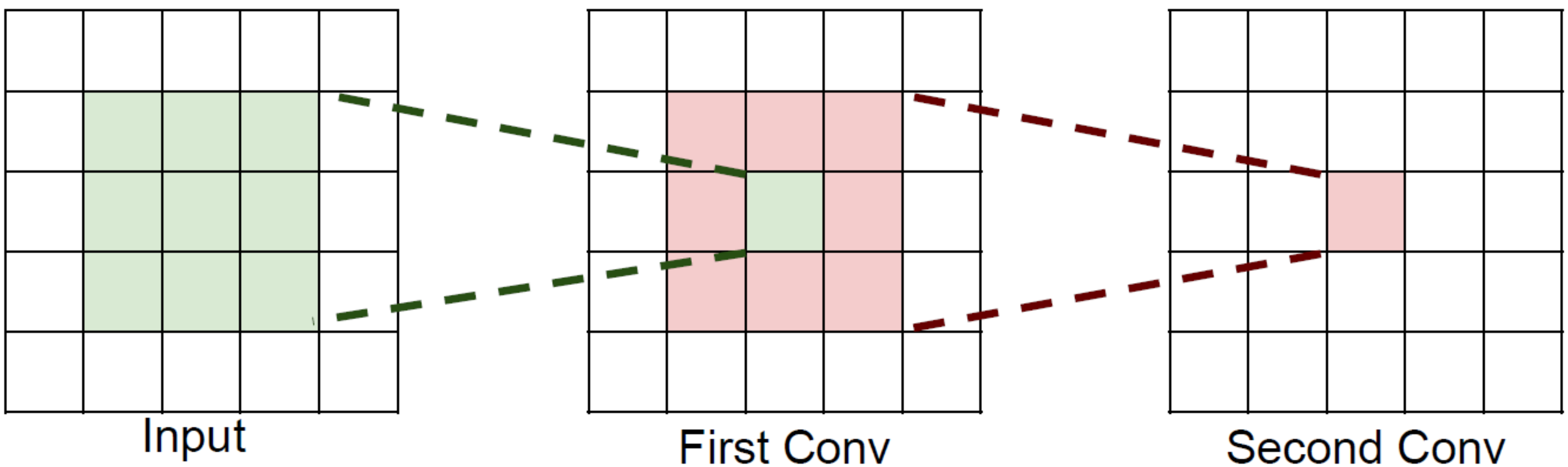
please cite the paper if you use the models.

All about convolutions

How to stack them

❖ Intro

- 3 x 3 filter를 2번 적용하면 1개 node는 몇 개의 pixel 정보를 포함할까?

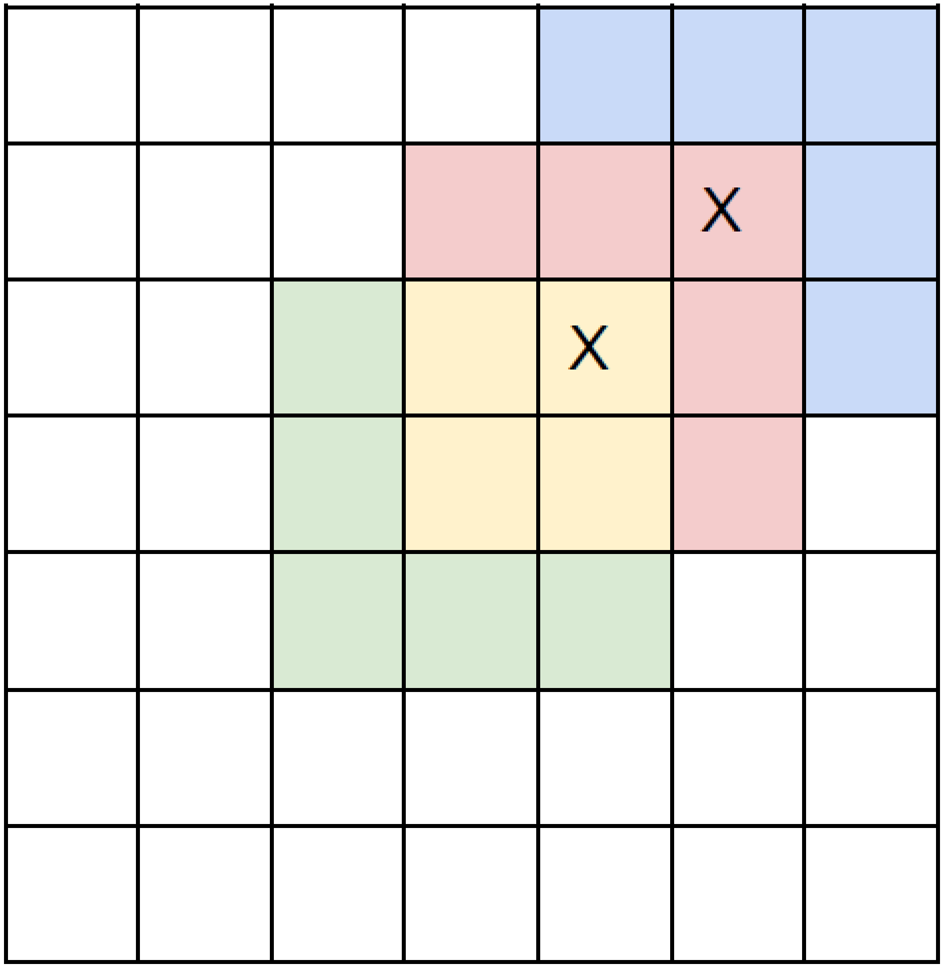


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to stack them

❖ Intro

- 3 x 3 filter를 3번 적용하면 1개 node는 몇 개의 pixel 정보를 포함할까?

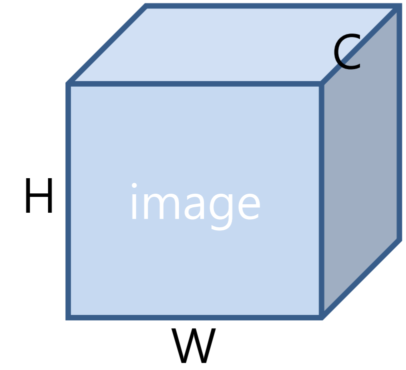


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

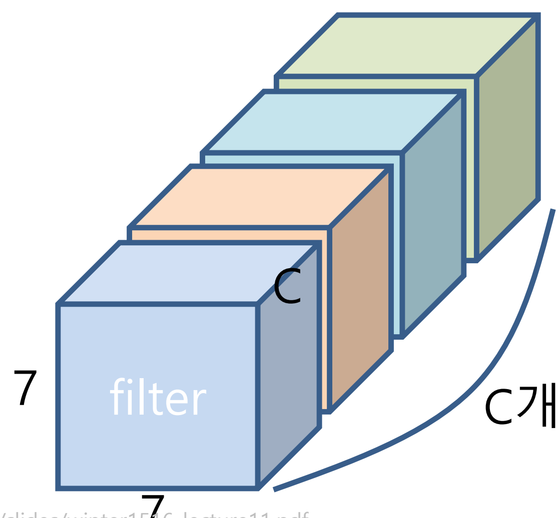
How to stack them

❖ The power of small filters

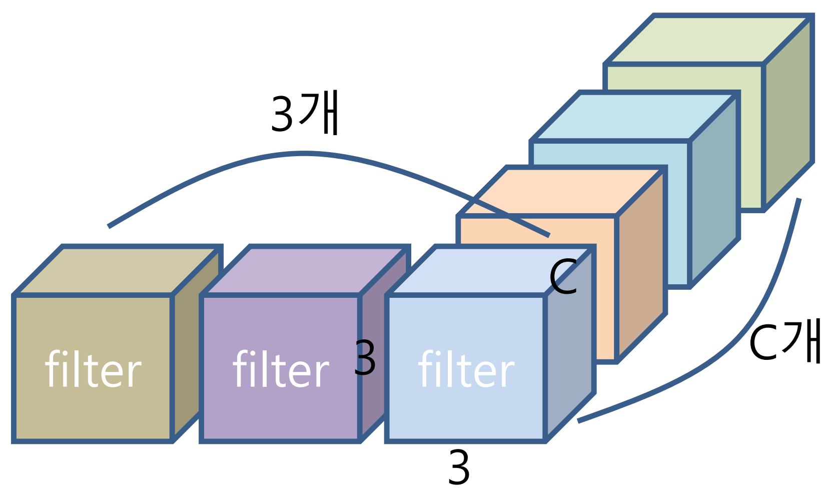
- Condition
  - Input : H x W x C
  - C개의 filter를 사용해서 conv 지나도 다시 depth = C
  - Stride = 1, padding 사용해서 H, W 유지



- <u>One CONV</u> with 7x7 filters
  - Number of weights
    = C x (7 x 7 x C)
    = $49C^2$

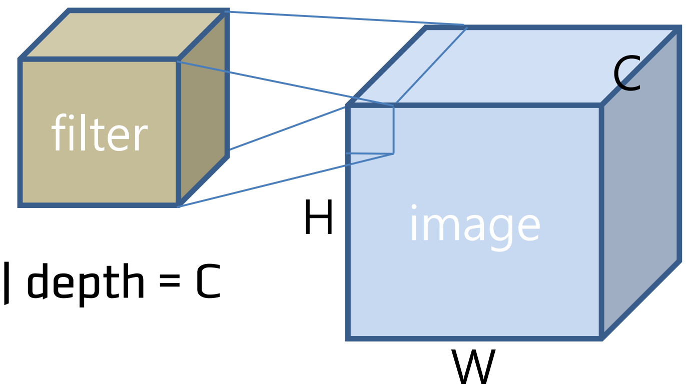


- <u>Three CONV</u> with 3x3 filters
  - Number of weights
    = 3 x [C x (3 x 3 x C)]
    = $27C^2$



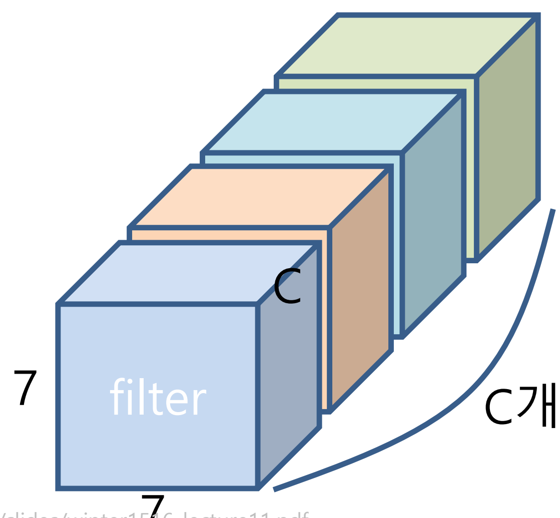
How to stack them

❖ The power of small filters

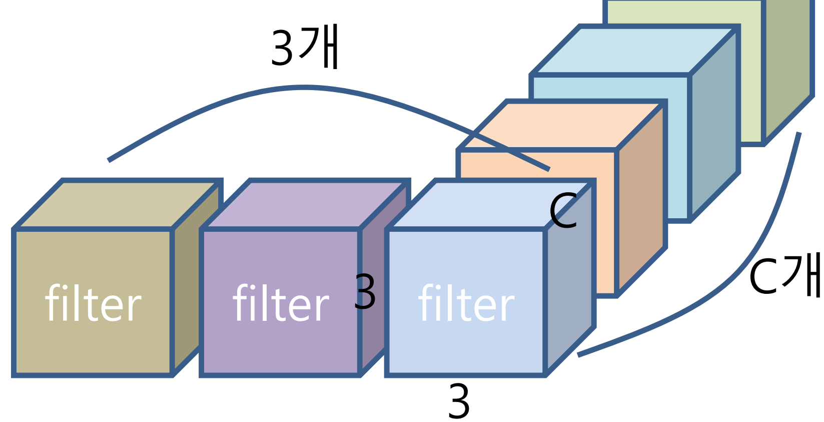


- Condition
  - Input : H x W x C
  - C개의 filter를 사용해서 1층 지나도 다시 depth = C
  - Stride = 1, padding 사용

- One CONV with 7x7 filters
  - Number of multiply-adds
    = (H x W) x C x (7 x 7 x C)
    = $49HWC^2$



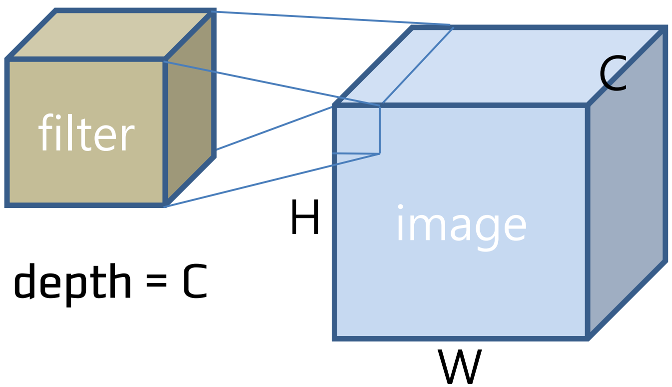
- Three CONV with 3x3 filters
  - Number of multiply-adds
    = (H x W) x 3 x [C x (3 x 3 x C)]
    = $27HWC^2$



http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to stack them

❖ The power of small filters



- Condition
  - Input : H x W x C
  - C개의 filter를 사용해서 1층 지나도 다시 depth = C
  - Stride = 1, padding 사용

- One CONV with 7x7 filters
  - Number of multiply-adds
    = (H x W) x C x (7 x 7 x C)
    = $49HWC^2$

- Three CONV with 3x3 filters
  - Number of multiply-adds
    = (H x W) x 3 x [C x (3 x 3 x C)]
    = $27HWC^2$

- Less compute, more nonlinearity = GOOD!

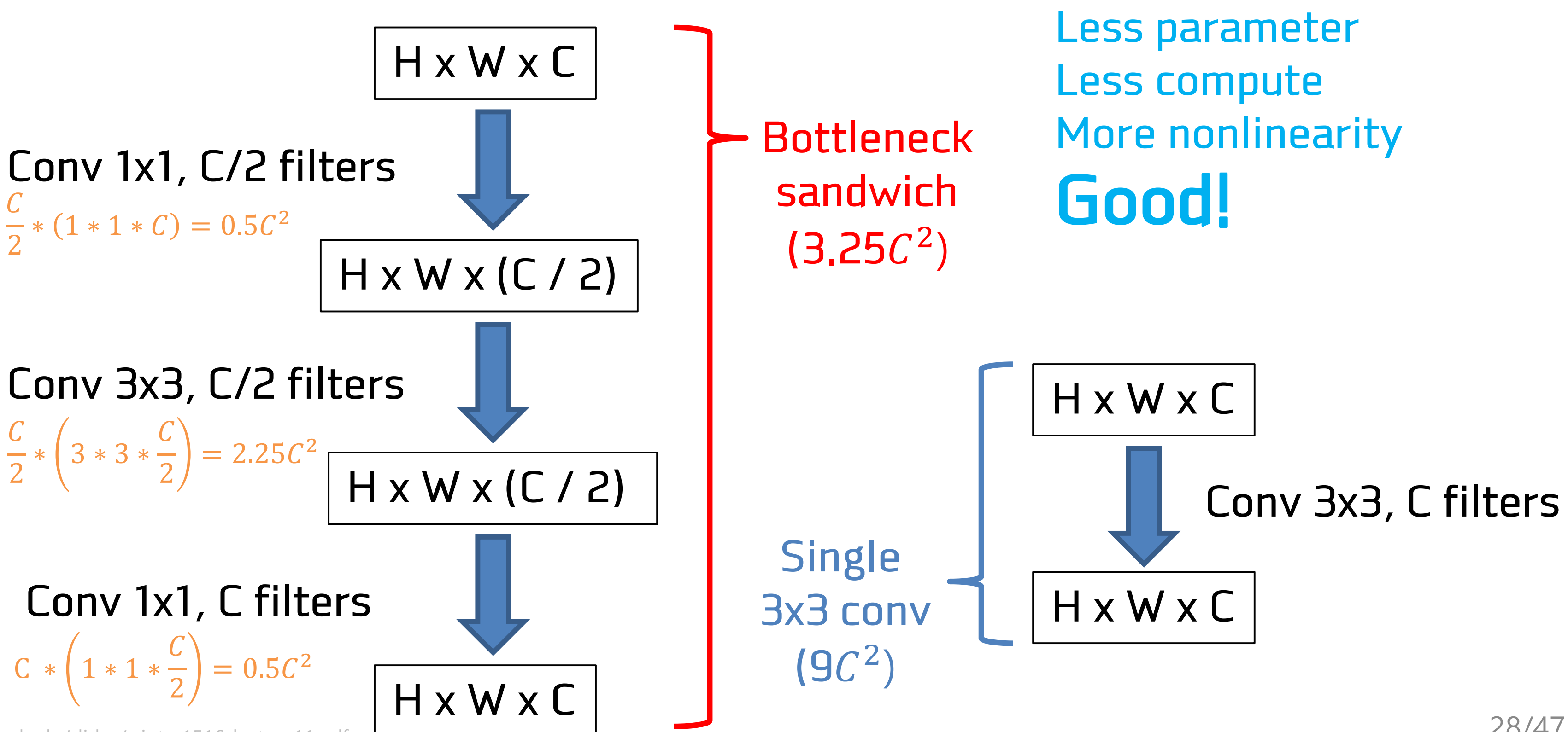
http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to stack them

❖ Why not try 1 x 1 filters?

- Receptive 영역이 없어서 다른 pixel(input)까지 포괄하여 정보 저장 불가
  - 1 x 1 filter 단독으로는 절대 앞의 효과를 볼 수 없음

❖ 1 x 1을 활용하여 parameter 수를 줄이자 (Bottleneck sandwich)

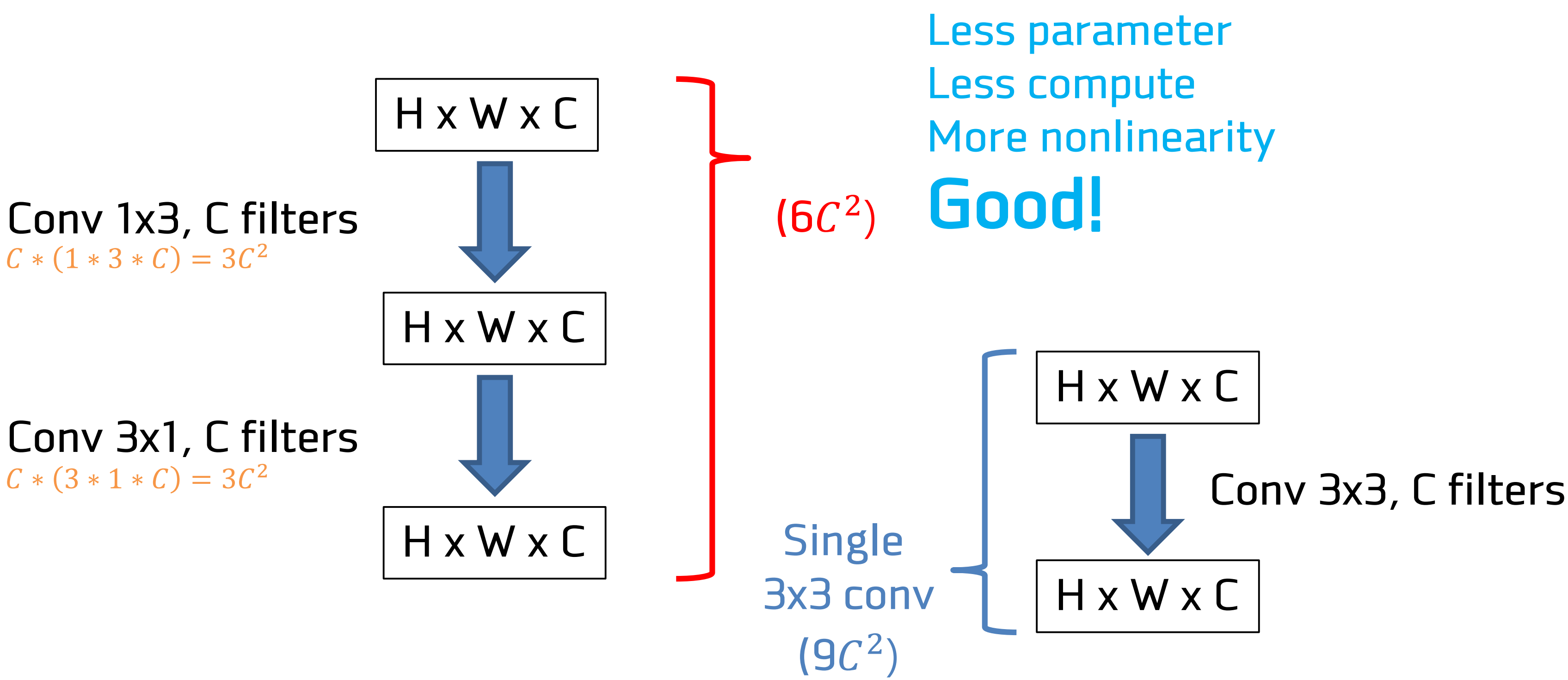


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to stack them

❖ Still using 3 x 3 filters … can we break it up?

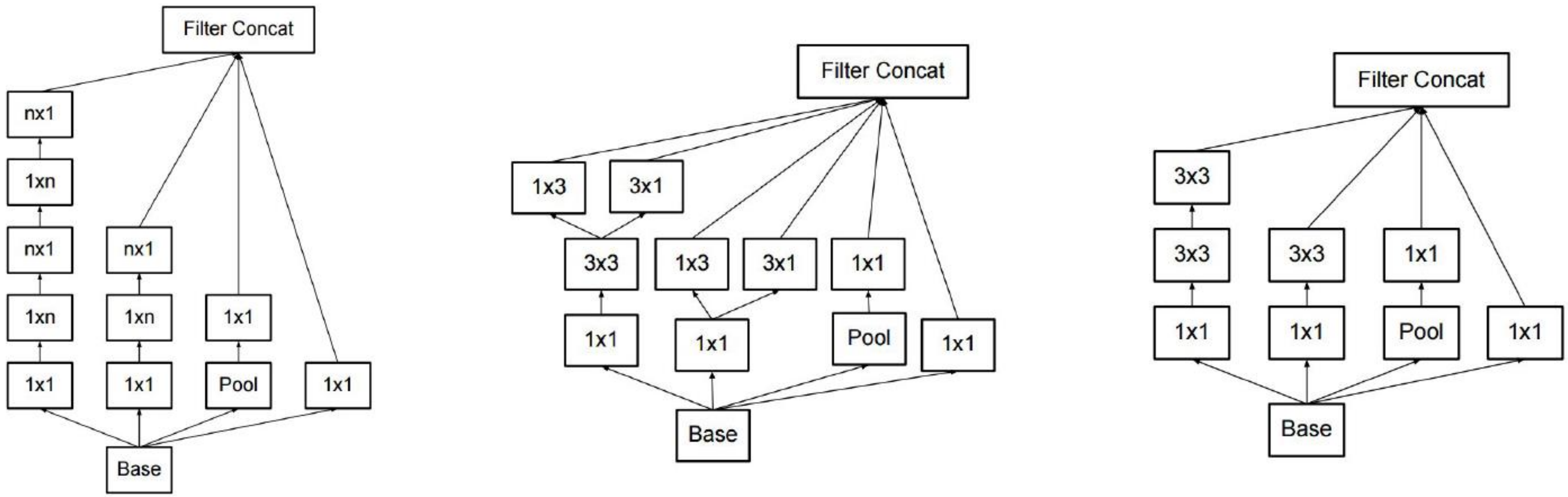
- 정사각형 모양을 탈피하자
- 1 x 3 filter, 3 x 1 filter를 조합해서 3 x 3을 만듦



How to stack them

❖ Example

- Latest version of GoogLeNet incorporates all these ideas



Szegedy et al, "Rethinking the Inception Architecture for Computer Vision"

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to stack them

❖ Summary

- Replace large conv (5 x 5, 7 x 7) with stacks of 3 x 3 conv
- 1 x 1 “bottleneck” conv are very efficient
- Can factor N x N conv into 1 x N and N x 1
- All of the above give

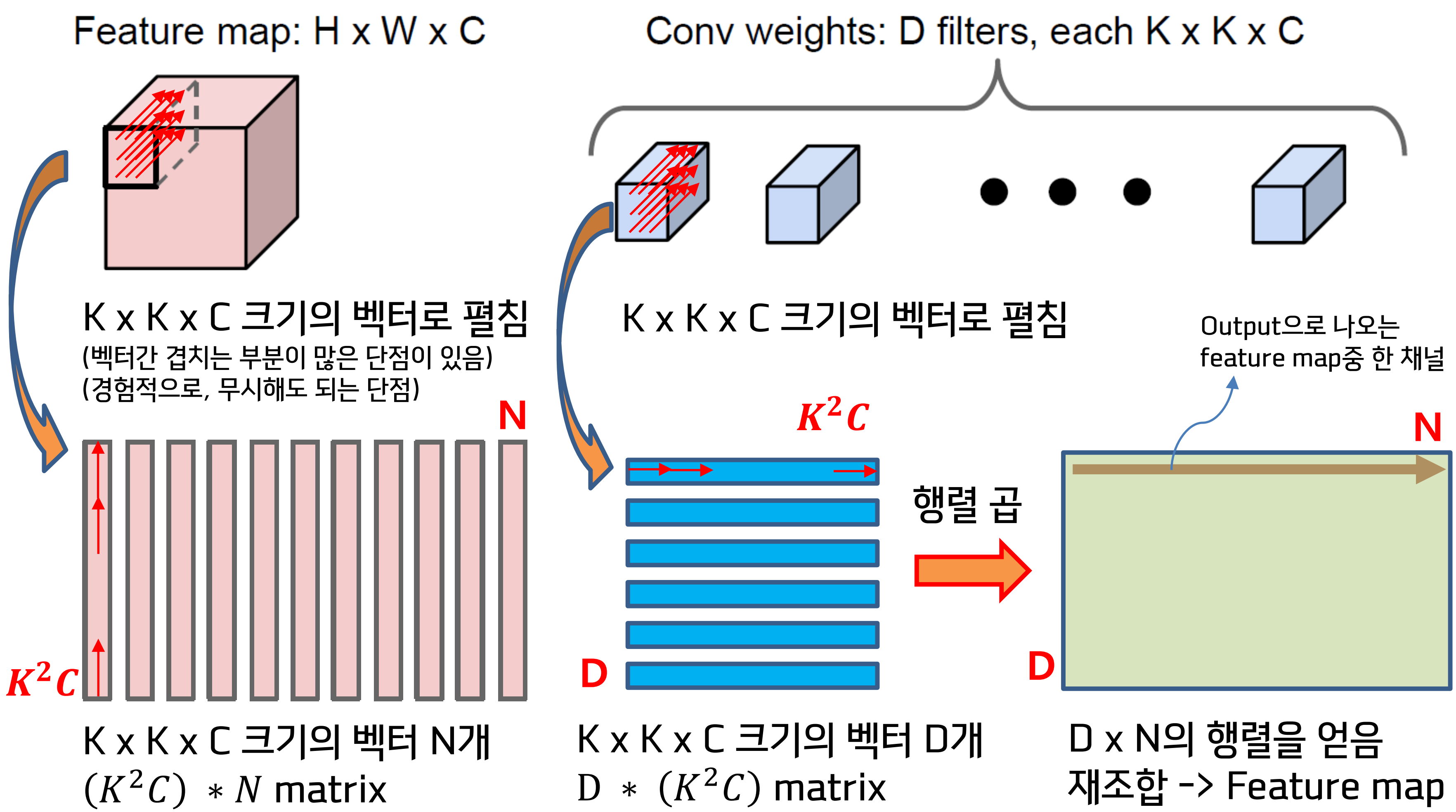
  fewer parameters, less compute, more nonlinearity

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

❖ im2col

- 두 행렬의 원소단위 곱을 벡터단위 내적으로 접근 -> fast



http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

❖ FFT (Fast Fourier Transform)

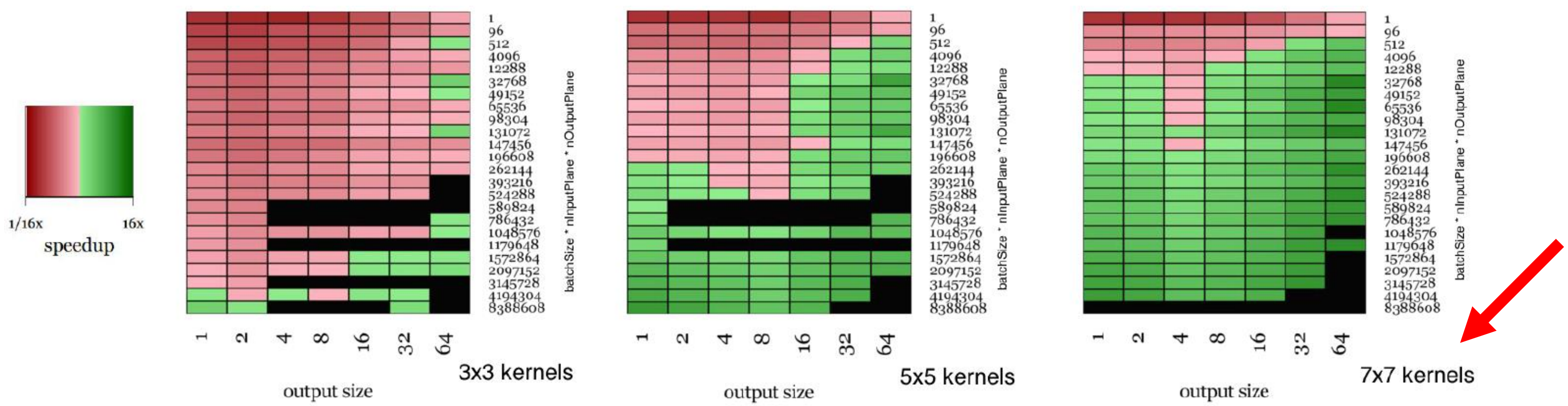
- 이산 푸리에 변환(Discrete Fourier transform)
  - 정의
    - 이산적인 복소수 값 $x_0, x_1, \ldots, x_{N-1}$을 복소수 값 $X_0, X_1, \ldots, X_{N-1}$로 변환하는 식
    - $X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}kn}$, (역연산)$x_k = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{-\frac{2\pi i}{N}kn}$, $k, n = 0, \ldots, N-1$
- FFT
  - 이산 푸리에 변환을 빠르게 수행하는 효율적인 알고리즘
  - DFT는 $O(n^2)$의 연산이 필요. FFT는 $O(nlogn)$의 연산만으로도 가능
- Convolution
  - 정의
    - 두 함수 f, g에 대해, $f * g(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$
  - Fourier transform 적용
    - $F(f * g) = F(f) * F(g)$
- 기존 연산 : $O(n^2)$의 연산. 본 접근법 : $O(nlogn)$의 연산

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

❖ FFT (Fast Fourier Transform)

- Implementing convolutions : FFT
  - Compute FFT of weights : $F(W)$
  - Compute FFT of image : $F(X)$
  - Compute elementwise product : $F(W) * F(X)$
  - Compute inverse FFT :
    $\mathrm{Y} = \mathrm{F}^{-1}\big(F(W) * F(X)\big) = F^{-1}\big(F(W * X)\big) = W * X$
- Performance
  - Large filter에서만 좋은 성능



Vasilache et al, Fast Convolutional Nets With fbfft: A GPU Performance Evaluation

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

❖ Fast algorithms

- **Naive matrix multiplication** : Computing product of two N x N matrices takes $O(N^3)$ operations
- **Strassen's Algorithm** : Use clever arithmetic to reduce complexity to $O(N^{log_2 7}) \sim O(N^{2.81})$

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\begin{aligned}
\mathbf{M}_1 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\
\mathbf{M}_2 &:= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1} \\
\mathbf{M}_3 &:= \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\
\mathbf{M}_4 &:= \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\
\mathbf{M}_5 &:= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \\
\mathbf{M}_6 &:= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\
\mathbf{M}_7 &:= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})
\end{aligned}$$

$$\begin{aligned}
\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\
\mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\
\mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\
\mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6
\end{aligned}$$

- 덧셈이 많아졌는데?
  - 큰 행렬의 경우, 행렬의 곱셈이 덧셈보다 더 많은 시간을 필요로 하기 때문에 덧셈을 더 하는 대신 곱셈을 덜 하는 것이 전체적으로 더 효율적이다.

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

01
02
03
04
05
06
07

❖ Fast algorithms

- 3 x 3 filter에 대해 fast algorithm을 적용
- 작은 filter에 대해 좋은 성능을 보임

N	cuDNN msec	cuDNN TFLOPS	F(2x2,3x3) msec	F(2x2,3x3) TFLOPS	Speedup
1	12.52	3.12	5.55	7.03	2.26X
2	20.36	3.83	9.89	7.89	2.06X
4	104.70	1.49	17.72	8.81	5.91X
8	241.21	1.29	33.11	9.43	7.28X
16	203.09	3.07	65.79	9.49	3.09X
32	237.05	5.27	132.36	9.43	1.79X
64	394.05	6.34	266.48	9.37	1.48X

Table 5. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp32 data. Throughput is measured in Effective TFLOPS, the ratio of direct algorithm GFLOPs to run time.

N	cuDNN msec	cuDNN TFLOPS	F(2x2,3x3) msec	F(2x2,3x3) TFLOPS	Speedup
1	14.58	2.68	5.53	7.06	2.64X
2	20.94	3.73	9.83	7.94	2.13X
4	104.19	1.50	17.50	8.92	5.95X
8	241.87	1.29	32.61	9.57	7.42X
16	204.01	3.06	62.93	9.92	3.24X
32	236.13	5.29	123.12	10.14	1.92X
64	395.93	6.31	242.98	10.28	1.63X

Table 6. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp16 data.

http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Lavin_Fast_Algorithms_for_CVPR_2016_paper.pdf
http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

How to compute them (fast)

❖ Summary

- Im2col : Easy to implement, but big memory overhead
- FFT : Big speedups for small kernels
- “Fast Algorithms” seem promising, not widely used yet

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Implementation details

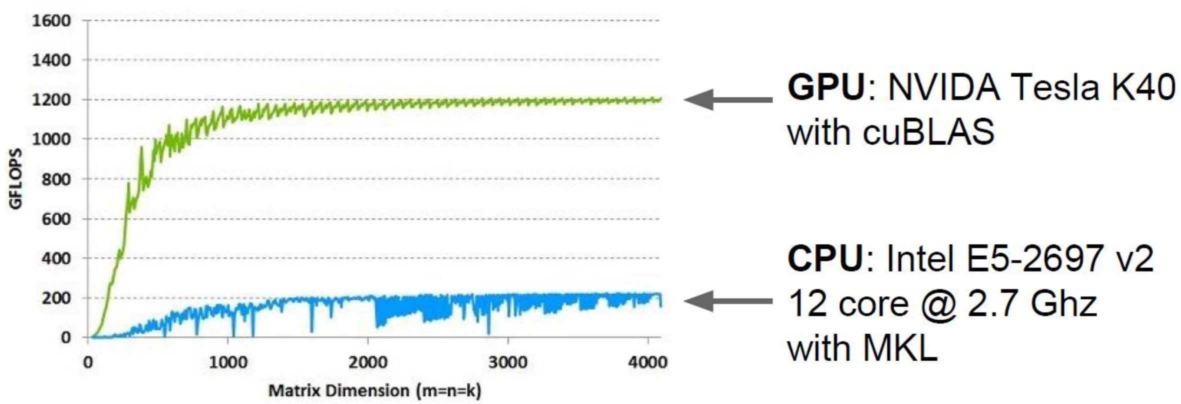
GPU / CPU

01
02
03
04
05
06
07

❖ CPU / GPU

- CPU
    - Few, fast cores (1 – 16)
    - Good at sequential processing
- GPU
    - Many, slower cores (thousands)
    - Originally for graphics
    - Good at parallel computation

- GPUs are really good at matrix multiplication



GPU / CPU

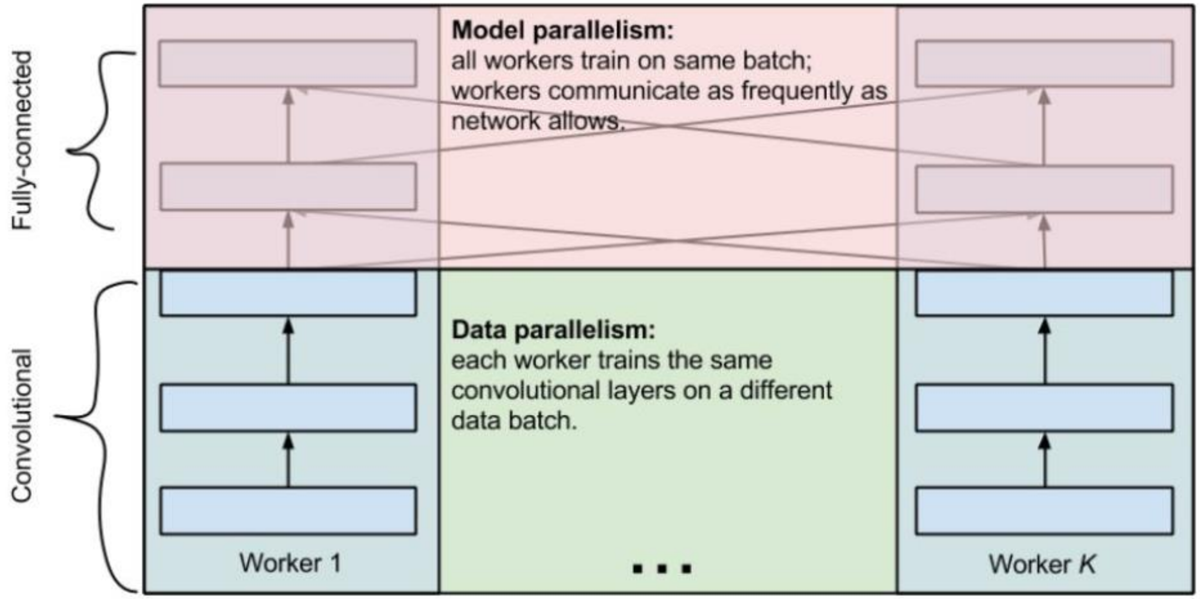
❖ GPU programming

- CUDA (NVIDIA only)
  - Write C code that runs directly on the GPU
  - Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - Similar to CUDA, but runs on anything
  - Usually slower :(
- Udacity: Intro to Parallel Programming
  - https://www.udacity.com/course/cs344
  - For deep learning just use existing libraries

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

❖ 분산 학습

- 개념
  - 여러 개의 GPU(혹은 CPU)에 나눠서 학습시키자
- 다수의 GPU와 컴퓨터를 이용한 분산 학습을 지원한 프레임워크 출현
  - Google의 Tensorflow
  - Microsoft의 CNTK(Computational Network Toolkit)
- Multi-GPU training: More complex

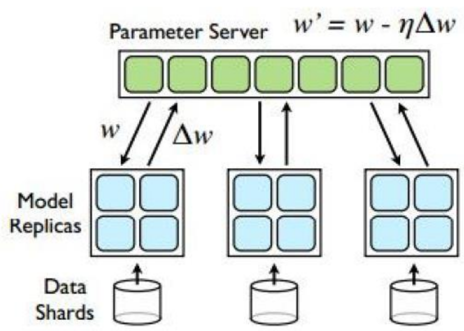


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

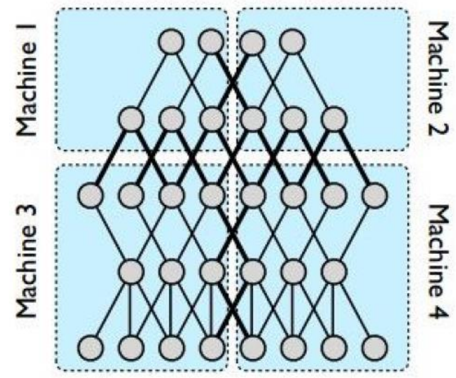
GPU / CPU

❖ 분산 학습

- Google: Distributed CPU training

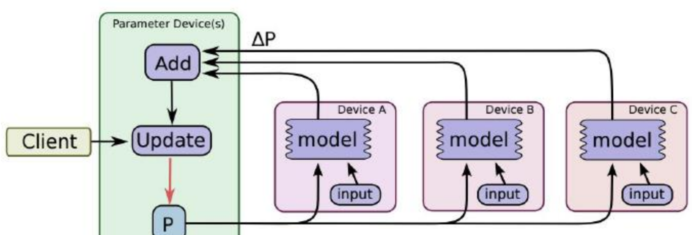


Data parallelism

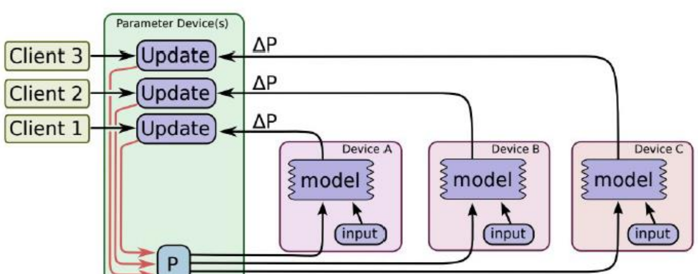


Model parallelism

- Google: Synchronous vs Asynchronous



Synchronous Data Parallelism



Asynchronous Data Parallelism

Bottleneck

❖ 분산 학습

- GPU – CPU bottleneck
  - CPU data prefetch+augment thread running while GPU performs forward/backward pass
  - Allocate task efficiently

- CPU – disk bottleneck
  - SSD >> HDD

- GPU memory bottleneck
  - More GPU(expensive…)

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Floating Point Precision

❖ 실수 표현 정밀도

- 컴퓨터는 bit 단위로 수를 표현함
  - CNN에서는 32bit single을 많이 사용함

- 추세
  - 32bit -> 16bit
    - 16bit로 바뀌면서 실수 정밀도를 잃지만, 성능 손실을 최대한 줄이는 방법을 개발함 (Stochastic rounding, Gupta et al, “Deep Learning with Limited Numerical Precision”, ICML 2015)
    - 이미 cuDNN에서 지원 중
    - Nervana fp16 kernels이 현재 가장 빠름

AlexNet (One Weird Trick paper) - Input 128x3x224x224

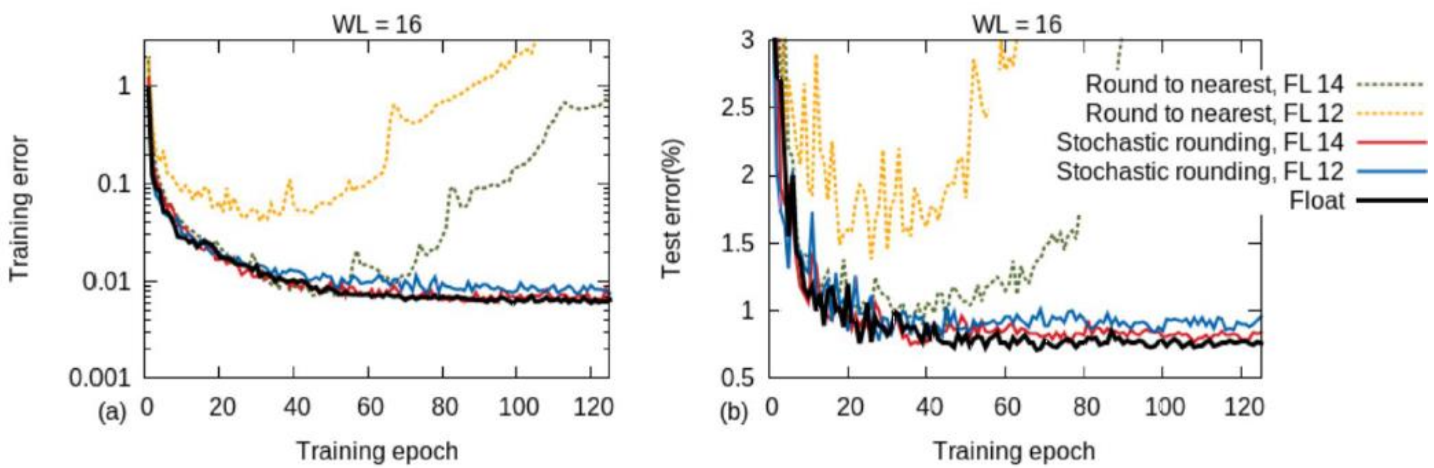
Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**92**	**29**	**62**
CuDNN[R3]-fp16 (Torch)	cudnn.SpatialConvolution	96	30	66
CuDNN[R3]-fp32 (Torch)	cudnn.SpatialConvolution	96	32	64

OxfordNet [Model-A] - Input 64x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**529**	**167**	**362**
Nervana-fp32	ConvLayer	590	180	410
CuDNN[R3]-fp16 (Torch)	cudnn.SpatialConvolution	615	179	436

GoogleNet V1 - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**283**	**85**	**197**
Nervana-fp32	ConvLayer	322	90	232
CuDNN[R3]-fp32 (Torch)	cudnn.SpatialConvolution	431	117	313

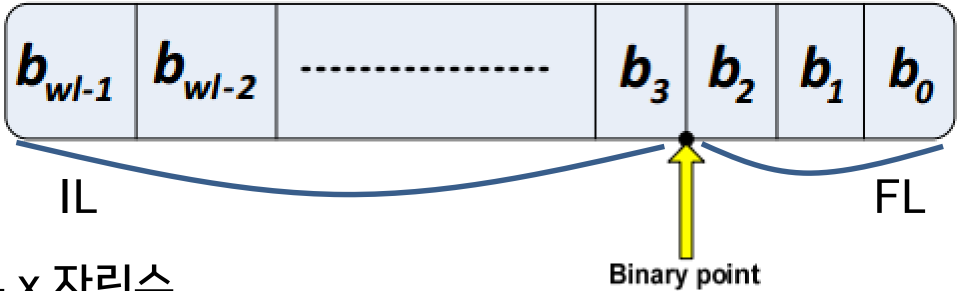


http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Floating Point Precision

❖ 실수 표현 정밀도

- Stochastic rounding
  - 확률에 근거해 rounding
  - Unbiased rounding scheme
  - Expected rounding error is zero, $E(Round\ (x, < IL, FL >)) = x$



  - 정수부 x 자릿수
    - 예) $ab.c\ =\ abc.*\ 2^{-1}$, $a.bc = abc.*\ 2^{-2}$
  - IL, FL : 정수부의 bit 수, 자릿수 담당 bit 수 (둘 다 설정해주는 값)
  - ϵ : 매우 작은 양의 정수 $2^{-\mathrm{FL}}$

$$Round\left(x, \langle \mathrm{IL}, \mathrm{FL} \rangle\right) = \begin{cases} \lfloor x \rfloor & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

http://www.jmlr.org/proceedings/papers/v37/gupta15.pdf
http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Floating Point Precision

❖ 실수 표현 정밀도

- 도전
  - 10bit -> 1bit
    - Courbariaux and Bengio, February 9 2016: "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1"
    - 속도 향상을 위해 극한의 상황까지 도달
    - Activation, weight = +1 or -1
    - 빠른 계산
    - Gradient를 구할 때는 조금 큰 bit를 사용

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Floating Point Precision

01
02
03
04
05
06
07

❖ Summary

- GPUs much faster than CPUs
- Distributed training is sometimes used
  - Not needed for small problems
- Be aware of bottlenecks: CPU / GPU, CPU / disk
- Low precision makes things faster and still works
  - 32 bit is standard now, 16 bit soon
  - In the future: binary nets?

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf

Q & A

01
02
03
04
05
06
07

http://cs231n.stanford.edu/slides/winter1516_lecture11.pdf