GOING DEEPER WITH CONVOLUTIONS

분석 16기 이지혜 - SEGMENTATION

Contents

1. Introduction
2. Improving Traditional Neural Networks
3. Inception Architecture
4. GoogLeNet
5. Polyack Average + Asynchronous SGD
6. Conclusion

1. Introduction

About Neural Networks

- Object detection
- Given 2 images of wolves, can identify sub species
- Speech recognition
- Identify how people respond to different stimuli in various environments
- Requires a large amount of resources to run smoothly
- Mostly constant

(a) Siberian husky

(b) Eskimo dog

2. Improving Traditional Neural Networks

Problem : Increasing the size of the network

1. Overfitting
2. More Computational Resources

Solution

1. Including Sparsity in the architecture
- Replacing Fully-Connected Layers to Sparse Layers
- Mimic biological systems

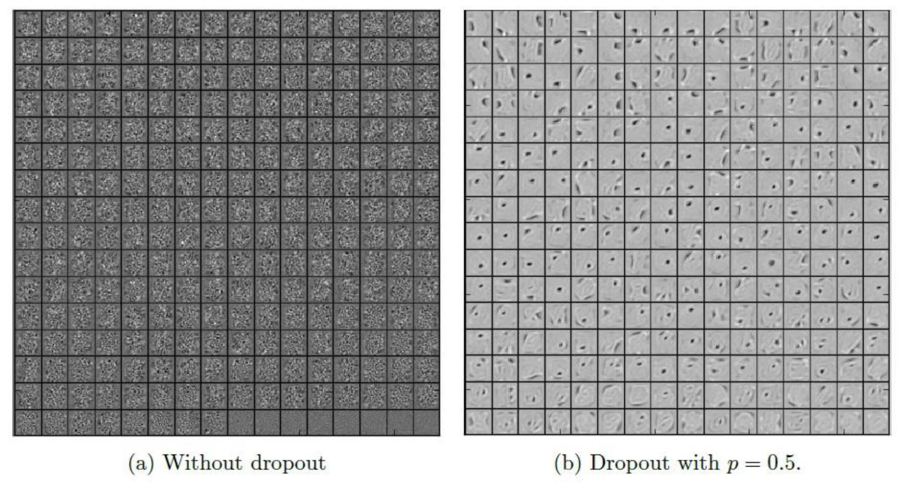
2. How?
- Utilizing computations on dense matrices
- Inception Architecture

2-1. Including Sparsity

Sparsity?

1. 직/간접 선택 / dropout / 모델의 자연스러운 학습 등 방법으로 발생한 deactivated node를 다음 학습에 반영 X

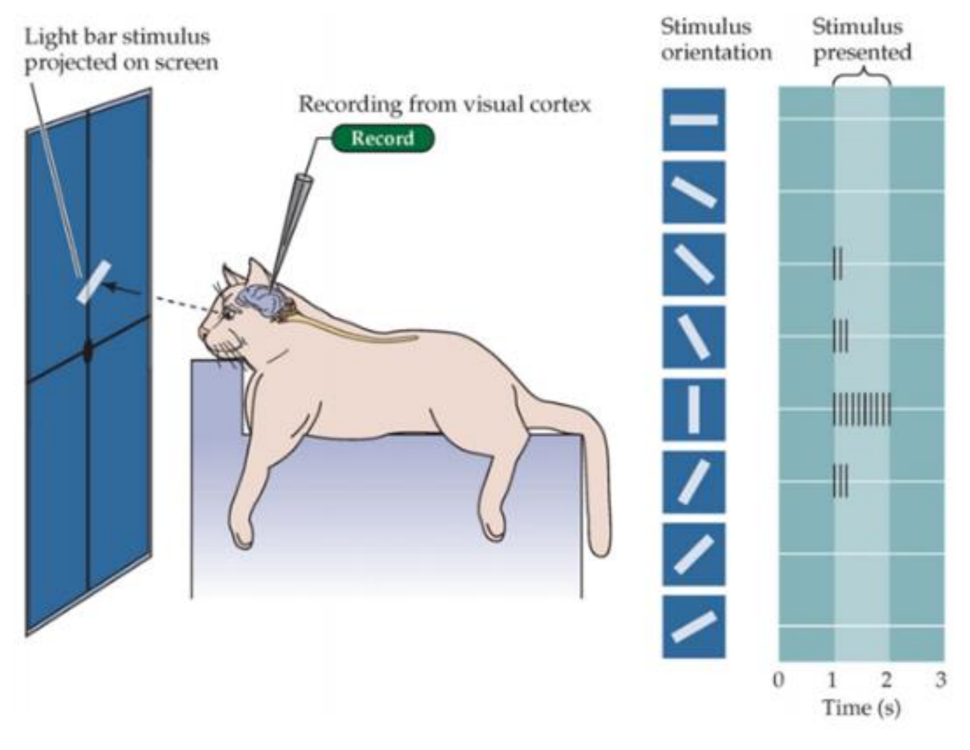
- 대부분의 node를 비활성화
- Strict condition에 따라 상관관계 분석 -> 이후 activate할 node 결정



(a) Without dropout (b) Dropout with $p = 0.5$.

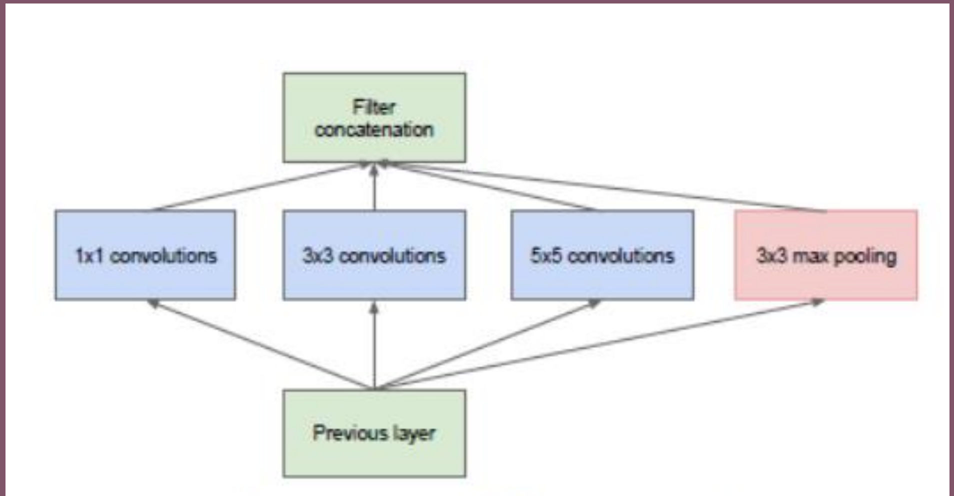
2. 즉, 학습 효과 증대를 위해 depth + width 를 모두 증가시키는 것을 목적으로 둔다.

2-2. Biological Systems?



- CNN의 배경 -> 실험체의 시신경들이 특정 패턴에 뉴런이 반응하는 것 관찰
- 각 패턴에 반응하는 뉴런의 집합은 전체에 비해 소수
- 즉, 반응하는 뉴런은 SPARSE한 구조

3-1. Inception Architecture : Naïve Version



(a) Inception module, naïve version

Input -> [Various Conv Layers + Max Pool]

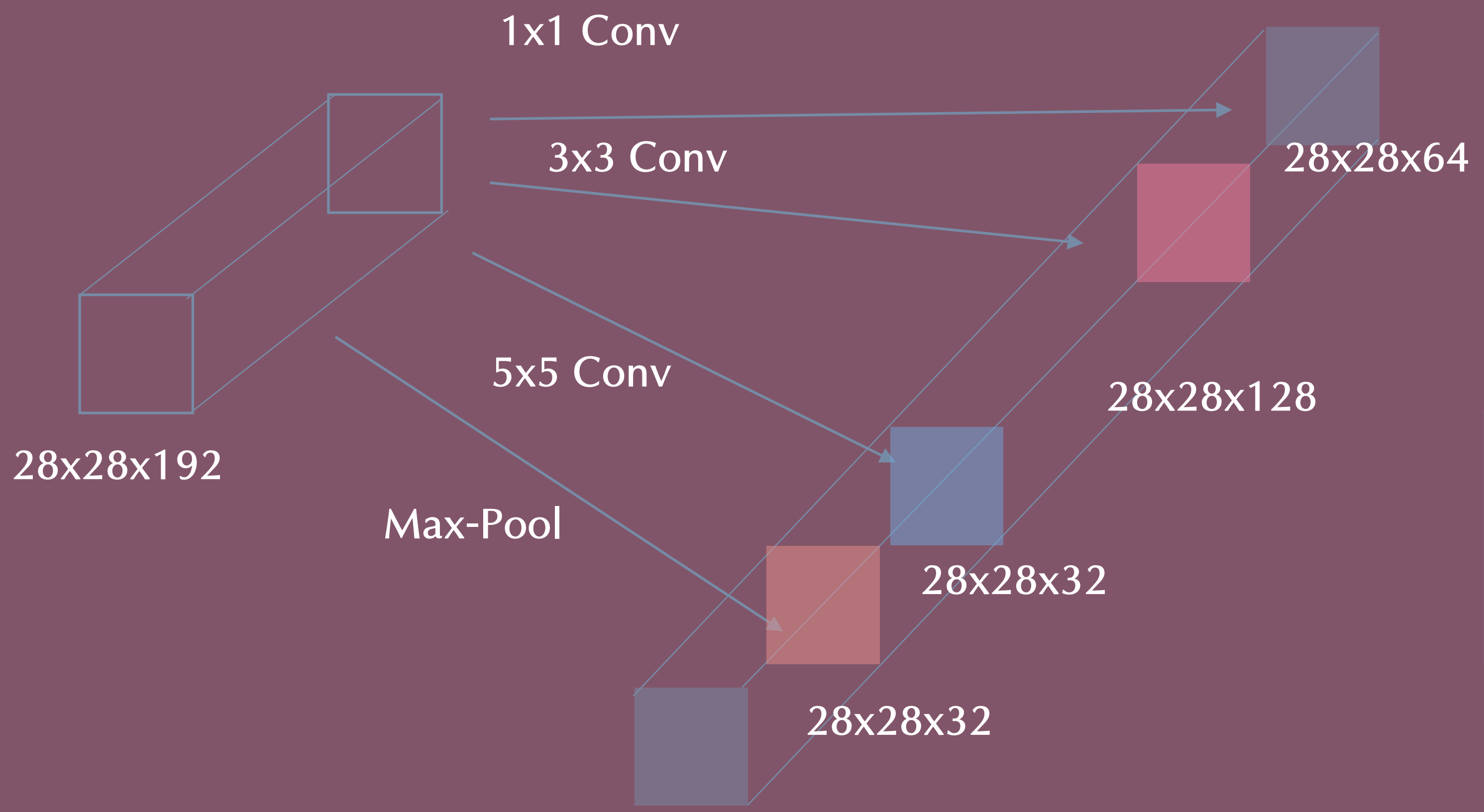
1. "Decision based more on convenience than necessity"

- Can be repeated spatially for scaling
- Avoids patch-alignment issues
- Pooling layer used to control overfitting

2. However

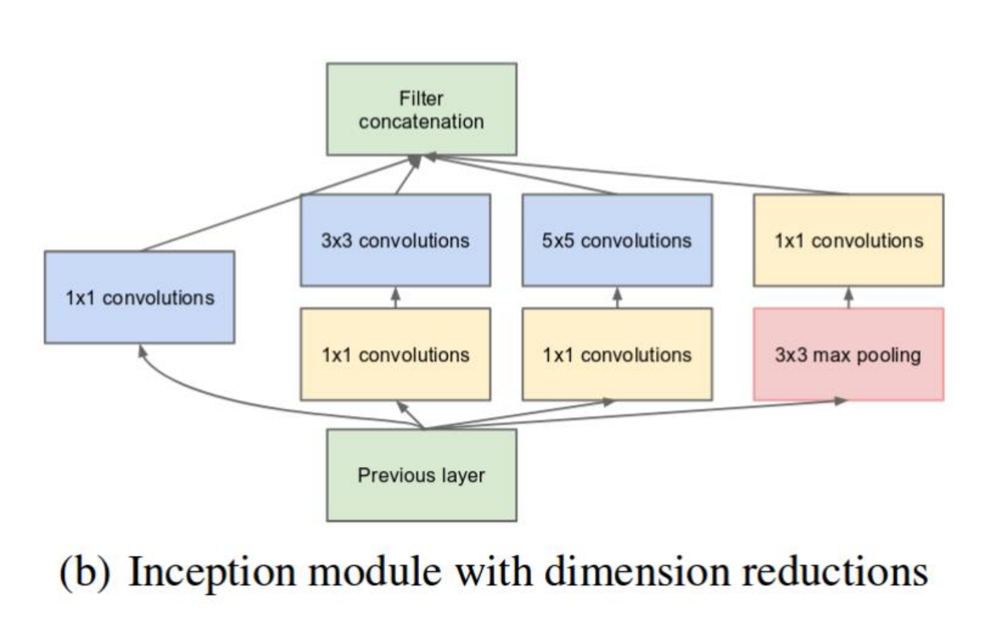
- 5x5 modules are expensive on Conv layers with many filters

3-2. Why Computational Problems?



TOTAL
=28x28x192x(32+32+128+64)
= 28x28x192x256
= 120M

3-3. Inception Architecture : Dimensionality Reduction

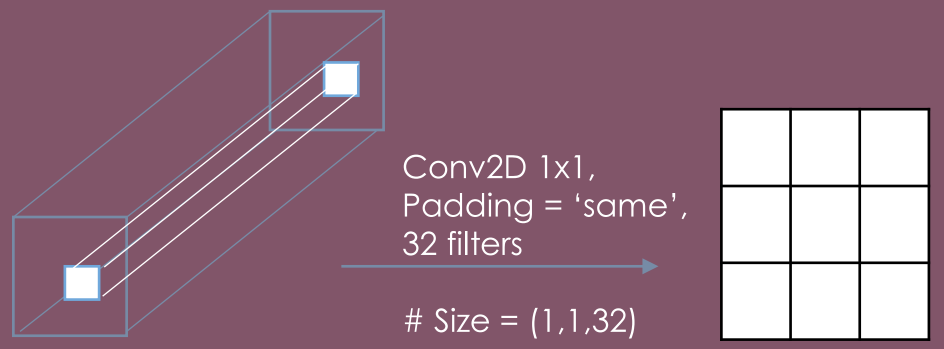


(b) Inception module with dimension reductions

Compute reductions with 1x1 Conv Layers

- Necessary processing power reduced
- Increase on number of units at each stage
- No sharp increase in computational resources in 3x3, 5x5 Conv layers

3-4. Why 1x1 Conv Layer?



Conv2D 1x1,
Padding = ‘same’,
32 filters

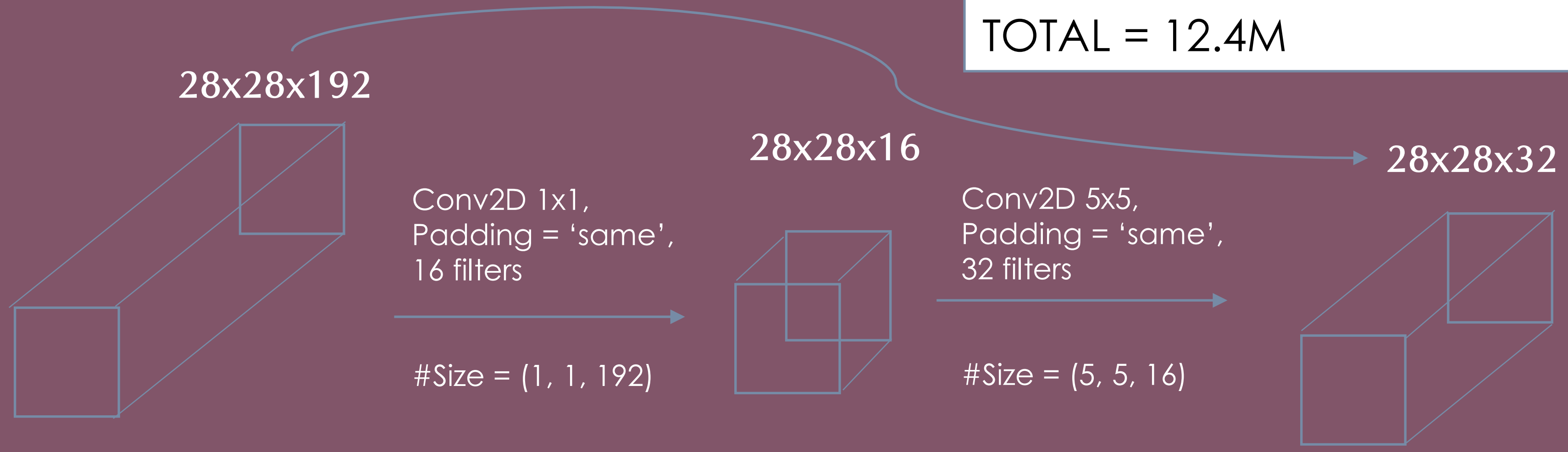
# Size = (1,1,32)

3x3x32

3x3x#filters

Using 1x1 Conv Layers

28x28x16x192 = 2.4M
28x28x16x32 = 10.0M
TOTAL = 12.4M



28x28x192

Conv2D 1x1,
Padding = ‘same’,
16 filters

#Size = (1, 1, 192)

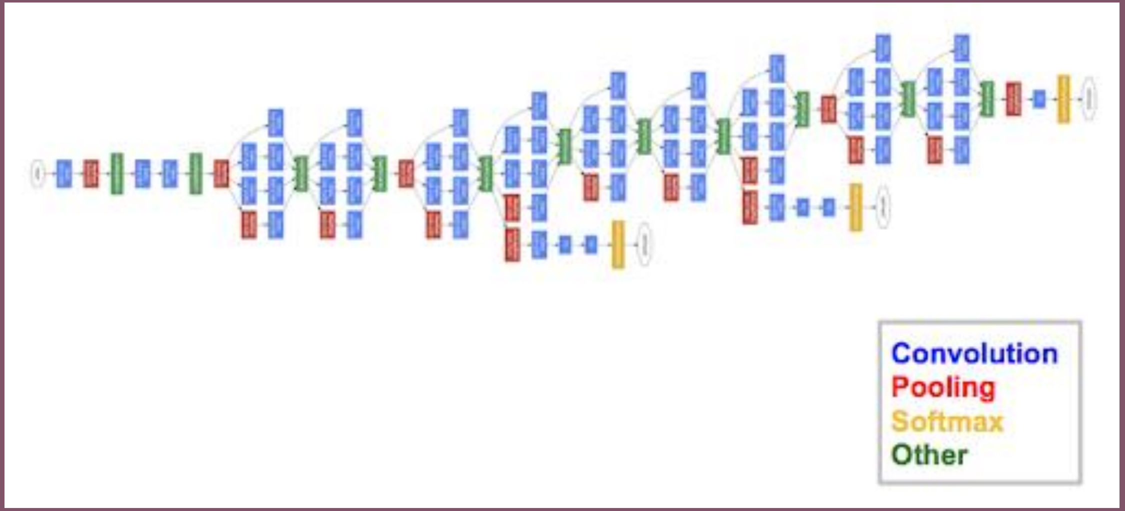
28x28x16

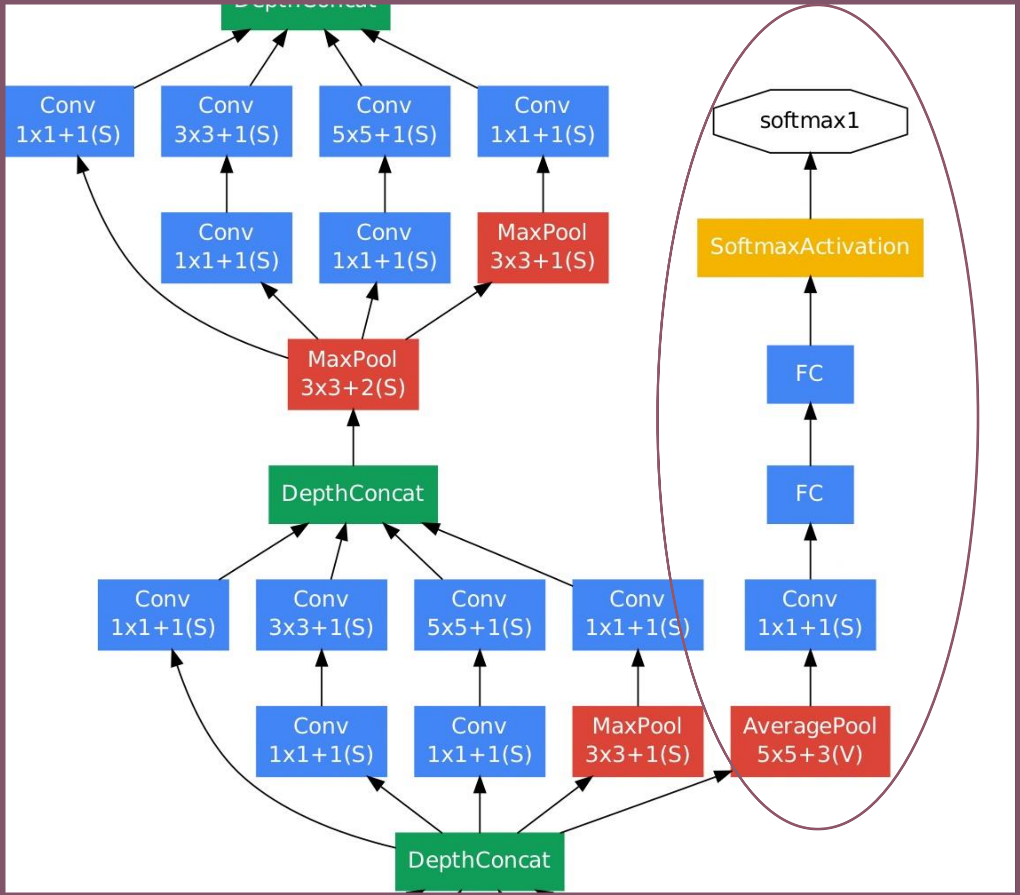
Conv2D 5x5,
Padding = ‘same’,
32 filters

#Size = (5, 5, 16)

28x28x32

4. GoogLeNet





1. Deep but Efficient

- enough to run on individual devices with low computational resources

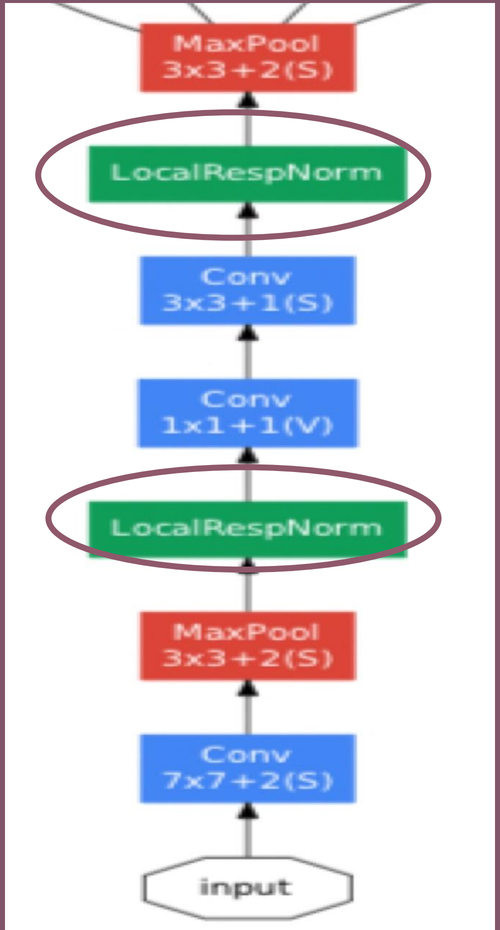
2. Auxiliary Layer used

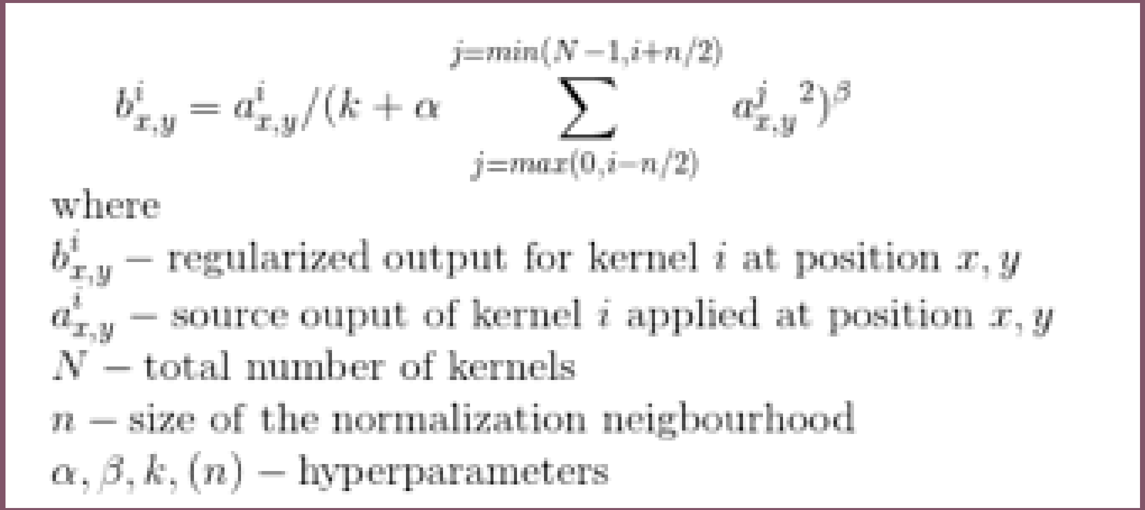
- REGULARIZATION (정규화 효과) -> avoid overfitting
- Use 0.3 of the loss
- Expect discrimination in the lower stages in the classifier

4. GoogLeNet

Local Response Normalization

- 측면 억제(lateral inhibition) 의 역할





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

where

$b^i_{x,y}$ – regularized output for kernel i at position x, y

$a^i_{x,y}$ – source ouput of kernel i applied at position x, y

N – total number of kernels

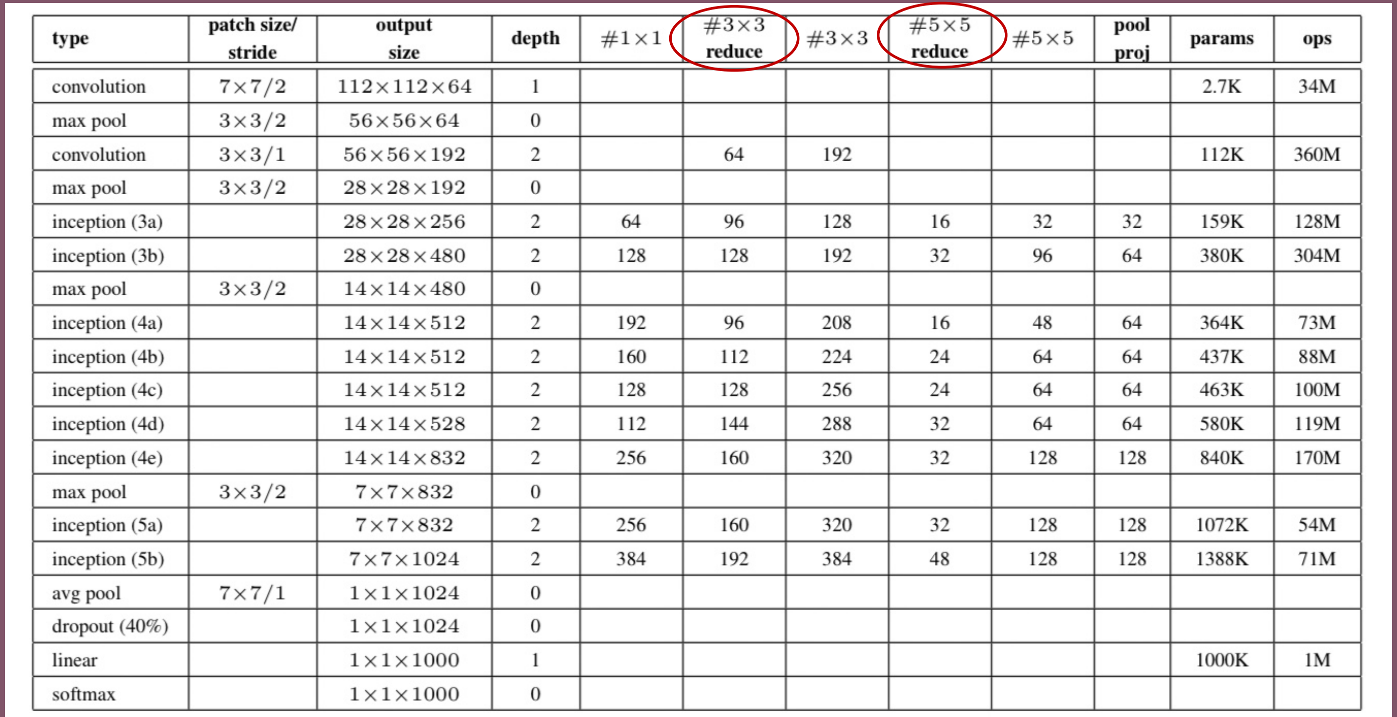
n – size of the normalization neigbourhood

$\alpha, \beta, k, (n)$ – hyperparameters

- 상호 연결된 신경 세포가 interneuron을 통해 이웃 신경 세포 억제
- ReLU를 사용하면 양수의 방향으로는 입력 값 그대로 사용
- Conv / Pooling에서 매우 높은 하나의 픽셀 값이 주변에 영향
- 이것을 방지하기 위해 다른 Activation Map의 같은 위치의 픽셀끼리 정규화

- 현재는 Batch Normalization이 주로 쓰임

4. 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								

1. #3x3 / #5x5 reduce

- Number of 1x1 filters in reduction layer

2. Softmax

3. Image Sampling

- [3/4, 4/3]

3. Activation

- ReLU
- Same for every layer

4. Optimizer

- Asynchronous SGD (momentum = 0.9)

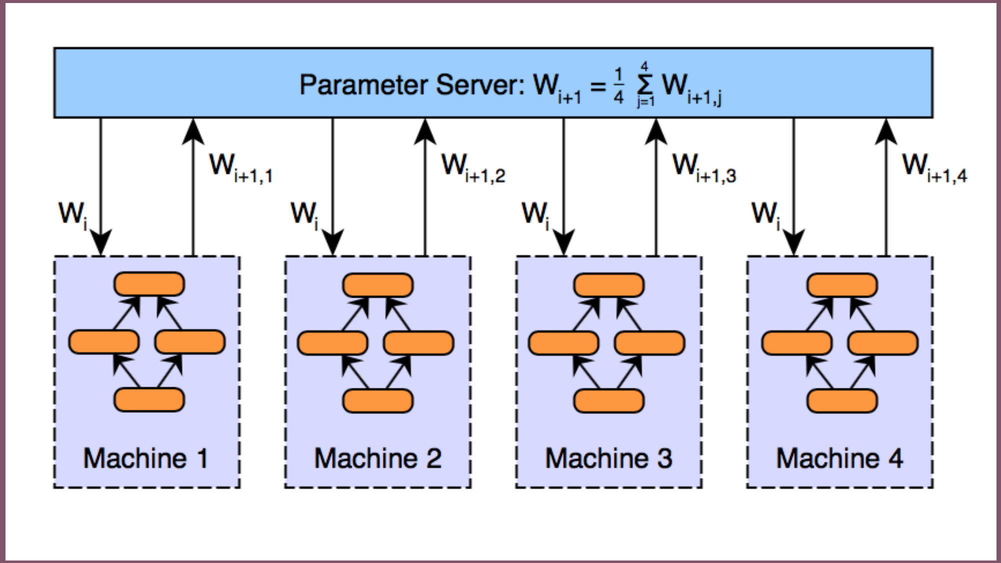
5. Polyack Averaging

- Keep a moving average of the parameter vector

6. Fixed Learning Rate Schedule

- Learning rate x 0.96 for every 8 epoch

5-1. Polyack Averaging

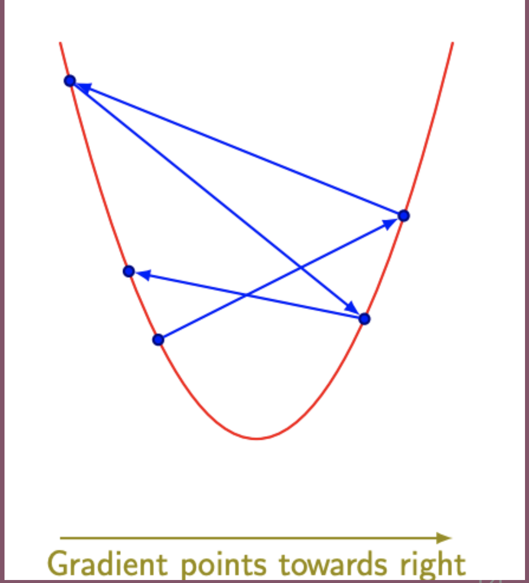


Parameter Server: $W_{i+1} = \frac{1}{4}\sum_{j=1}^{4} W_{i+1,j}$

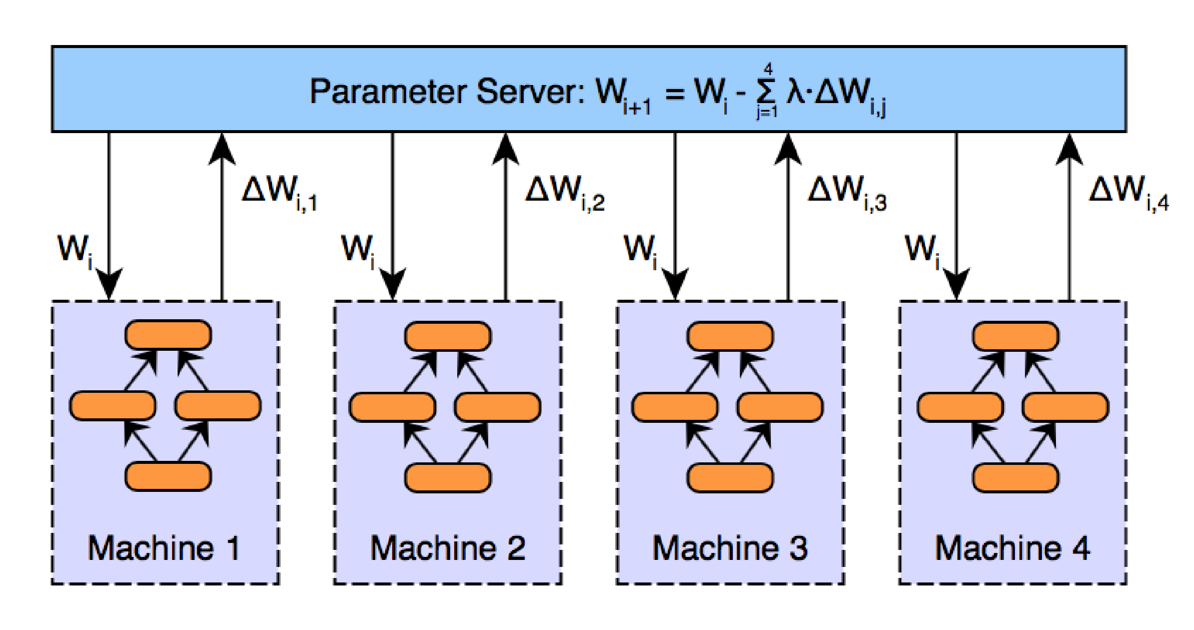
- Sets final parameters to an average of (recent) parameters visited in the optimization trajectory

- Specifically if in t iterations we have parameters $\theta_1, \theta_2, \theta_3 \ldots \theta_t$

- $\theta_t = \frac{1}{t}\sum_i \theta_i$



5-2. Asynchronous(=비동기적) SGD



- Accelerated by leveraging variance reduction
- Coordinate sampling
- Nesterov Momentum

$$v_{temp} = \mu v_{t-1} - \epsilon g(\theta_t)$$
$$v_{t+1} = \mu(v_{temp}) - \epsilon g(\theta_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

- 각 worker device 가 분할된 data set 으로 학습된 것을 비동기적인 방법으로 weight parameter 저장소를 update

- 많은 data set 환경에서 하나의 GPU로 계산을 할 때, SGD 가 전체적으로 효과적 학습 가능

- 분산된 환경에서 효율적인 학습 가능

6. Conclusion

- CNN is still top performers in the Neural Network
- Inception architecture allowing large scaling + minimizing bottlenecks
- Proves using 1x1 Conv to be effective in reducing dimension

감사합니다.