

# Designing More Efficient Convolution Neural Network

---

다양한 형태의 컨볼루션 레이어 경량화 설계 기법

Dongyi Kim

Ajou University  
Dept. of Computer Engineering  
[waps12b@ajou.ac.kr](mailto:waps12b@ajou.ac.kr)

# Dongyi Kim

## Profile

- 아주대학교 석사과정 재학 중
- 아주대학교 소프트웨어학과 졸업 (2017.02)

## Contact

- waps12b@ajou.ac.kr

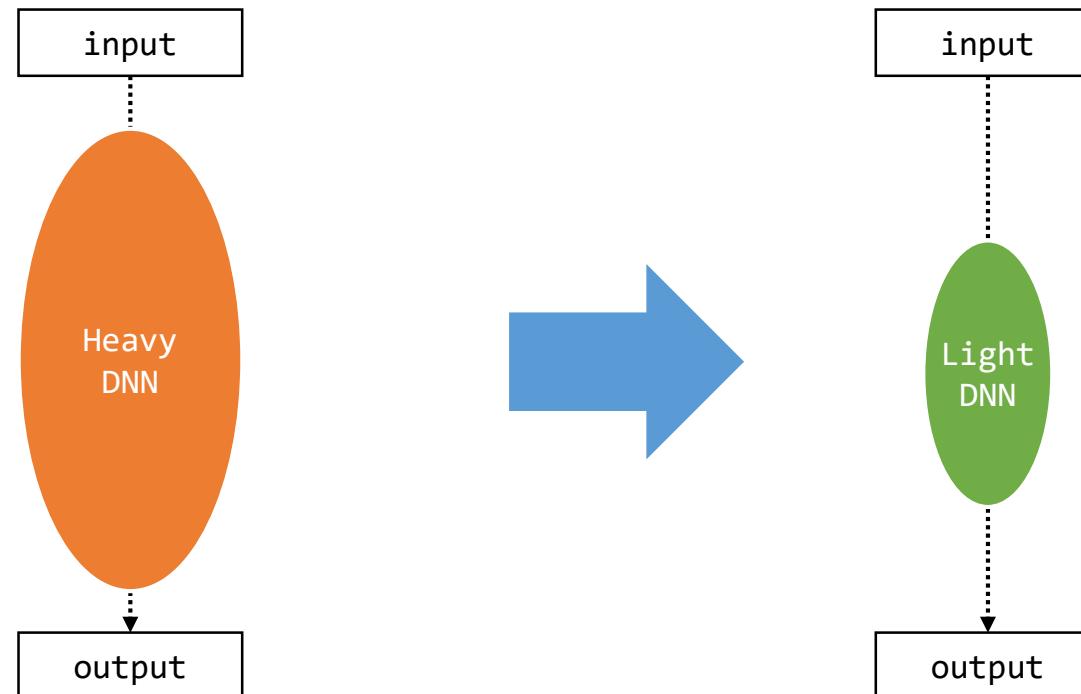
## Link

- <https://www.linkedin.com/in/dongyikim/>
- <https://github.com/waps12b>

# 모델 경량화?

비슷한 수준의 성능을 유지한채 더 적은 파라미터 수와 연산량을 가지는 모델 만들기

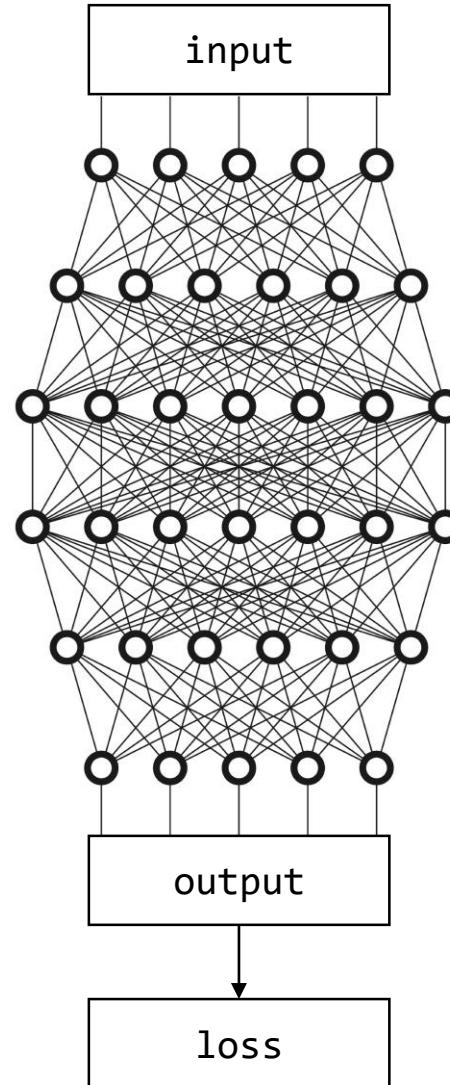
- 연산량 감소
  - Low-Power Device 지원 및 실시간성 확보
- 파라미터 수 감소
  - 모델의 저장 및 전송에 필요한 리소스 절약



# Deep Learning is not perfect

여전히 많은 부분은 사람에 의해 결정된다.

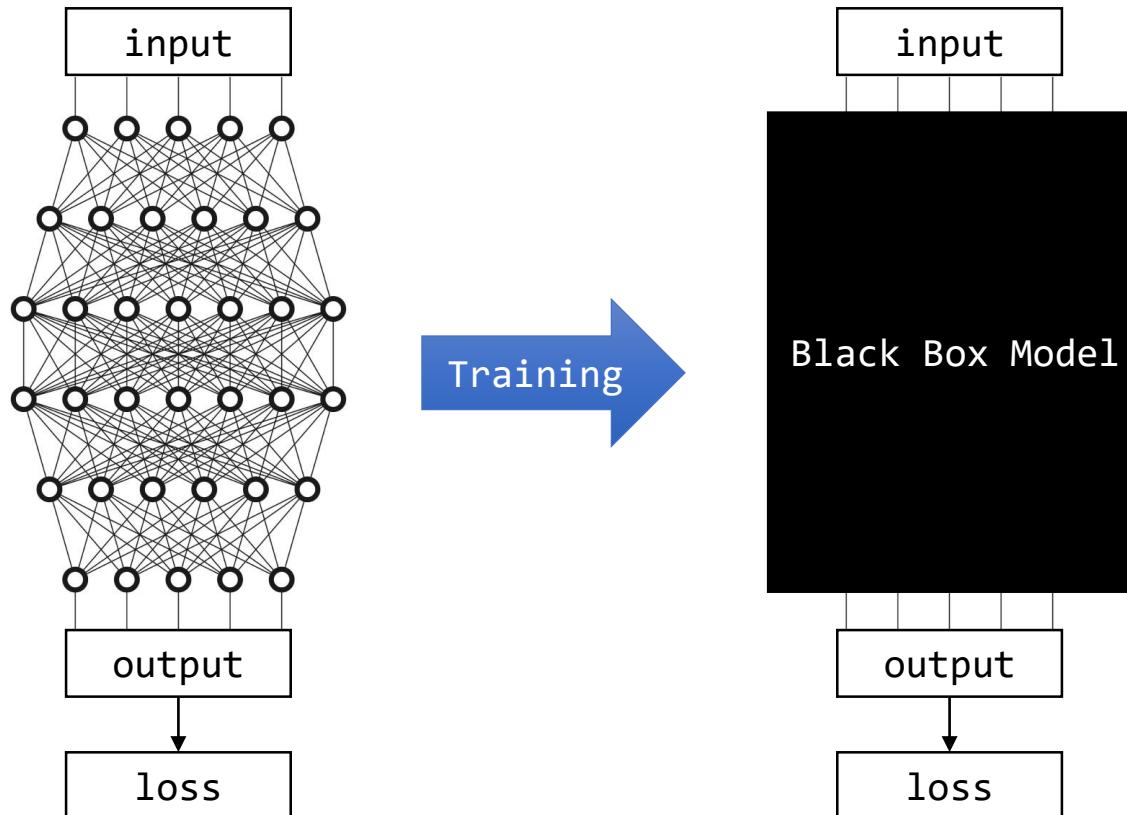
- Layers
- Network Architecture
- Loss Function
- Optimizer
- Hyper Parameters
- Data



# Deep Learning is not perfect

이로 인해 발생하는 문제점들

- 이런 Hyper Parameter들에 의해 학습 결과가 달라질 수 있다.
- Greedy한 방식의 학습 과정이 항상 최적의 네트워크를 보장하지 않는다.



# 그래서 우리는 ...

- i) 분명 어딘가에 더 효율적인 네트워크 구조가 존재한다
- ii) 기존의 모델을 더 효율적으로 개선할 수 있다

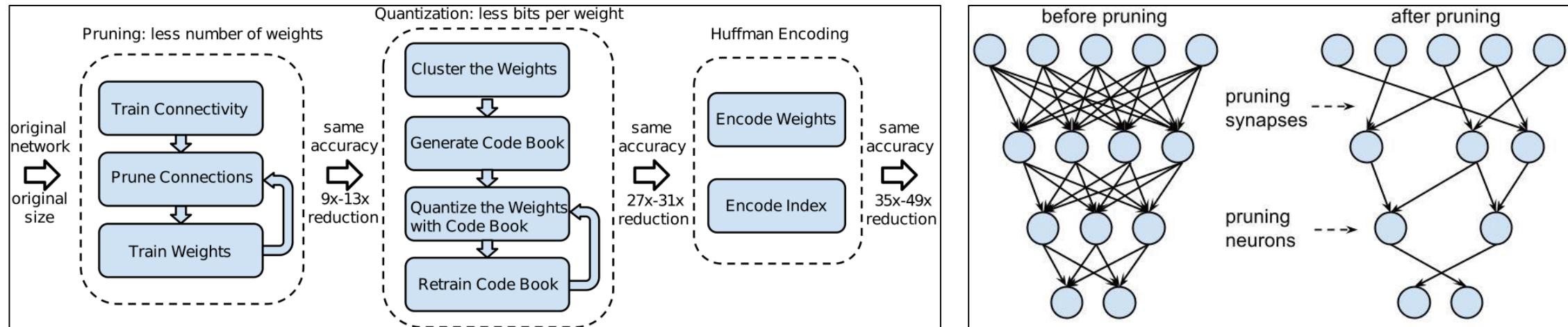
∴ 엔지니어는 찾아야 한다



# Related Works – Deep Compression

아래의 과정으로 Trained Model을 더욱 경량화 할 수 있다.

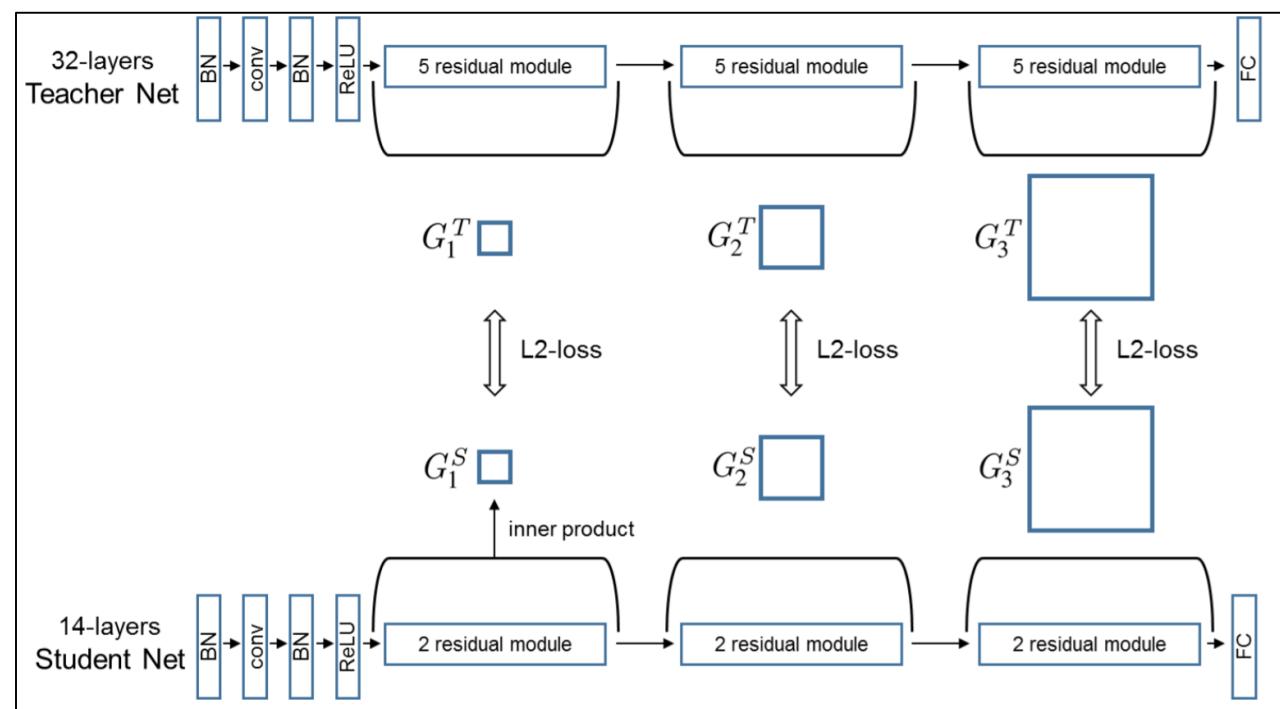
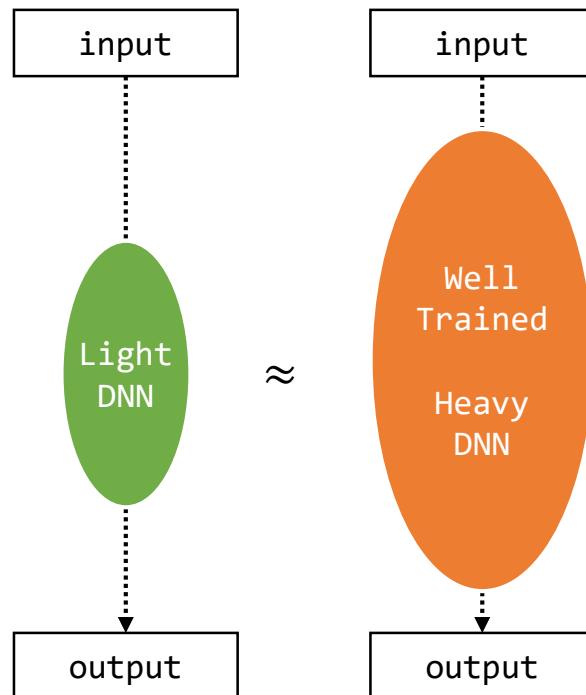
1. 불필요한 Connection들에 대한 Pruning이후 재학습
2. 가중치 Quantization
3. Encoding



[Ref] Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (S. Han, et al.)

# Related Works – Knowledge Distillation

상대적으로 작은 모델을 이미 학습된 모델과 유사한 출력을 가지도록 학습하는 방법



[Ref] A Gift From Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning (J. Yun, et al.)

## Related Works – Training Strategies

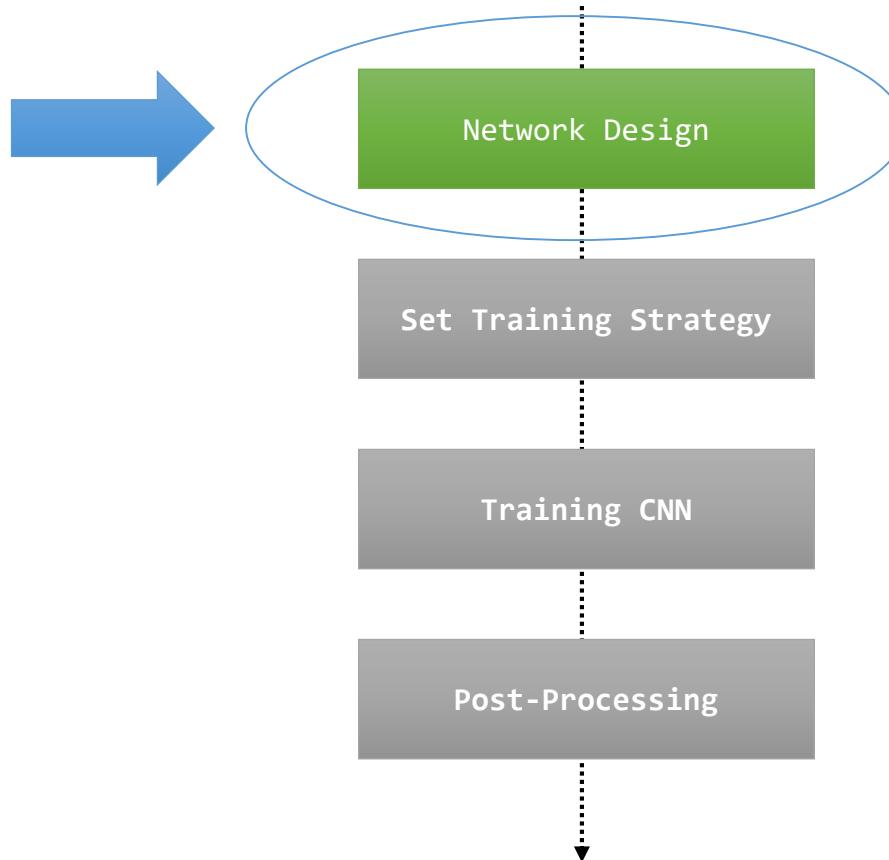
각 문제에 맞는 다양한 학습 전략들을 적용한다.

- Loss Function
- Fine Tuning
- Optimizer
- Training Techniques

...

# Main Topic

‘더욱 효율적인 구조의 CNN을 설계하기 위한 방법들’



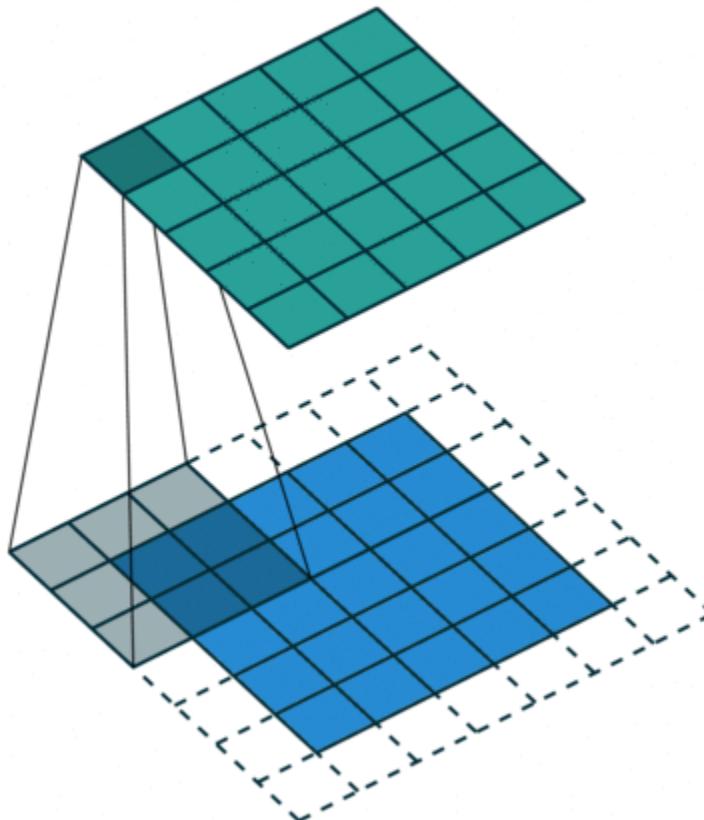
# Main Topic

다양한 개선 기법들이 존재하는데도 새로운 모델을 개발해야 하는 이유?

1. 최초 학습 과정에서의 리소스를 절약할 수 있다.
2. 더 효율적인 모델을 설계했다면? 앞서 언급한 기법들을 추가로 적용할 수 있다!

# Convolution Layer

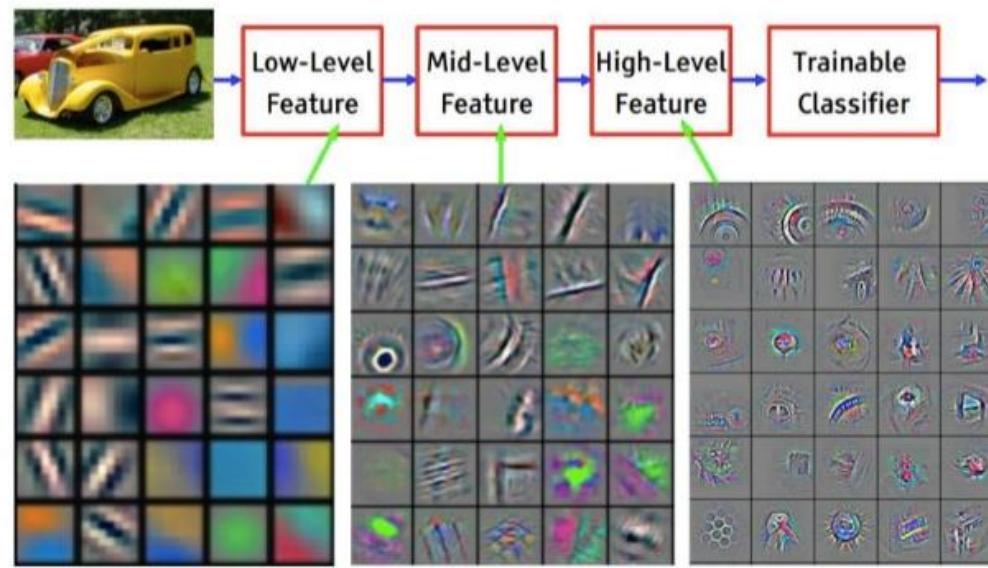
---



# Convolution (2D) Layer

영상에 대한 “2D Spatial Feature”를 추출하는 필터들로 구성되어 있다.

- x, y 방향으로 인접한 픽셀들에 대한 필터를 학습한다.
- 같은 위치에 해당하는 모든 채널에 대해 가중치를 가진다.

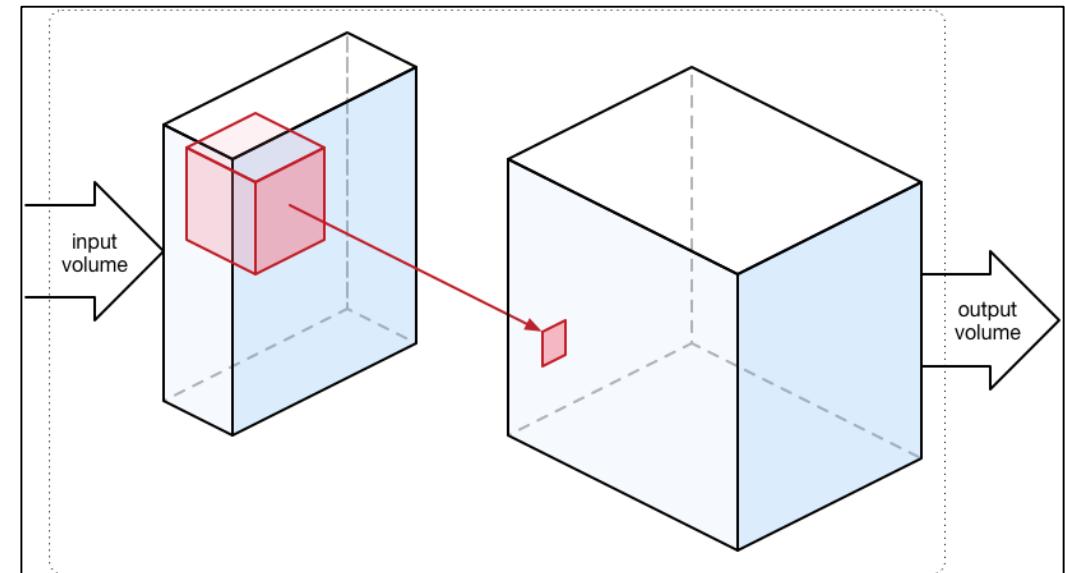
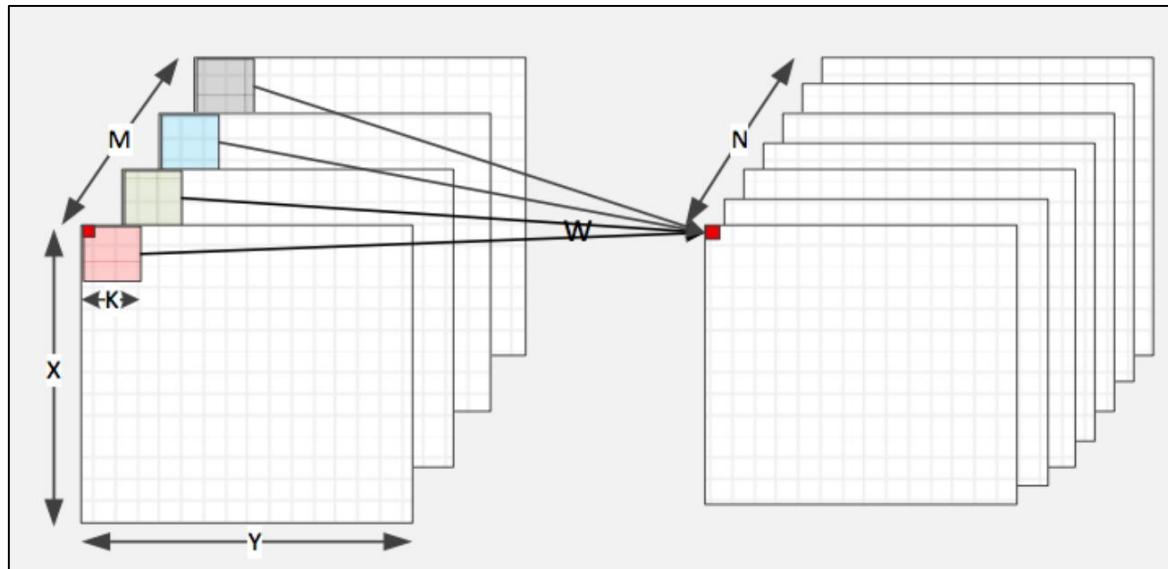


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Convolution (2D) Layer

하나의 커널은 입력 영상의 각 위치 별 반응치를 계산하여 2D Response Map을 출력한다.

- 각 위치별로 모든 채널의 픽셀들에 대한 Linear Combination을 수행한다.



# Convolution Layer vs. Fully Connected Layer

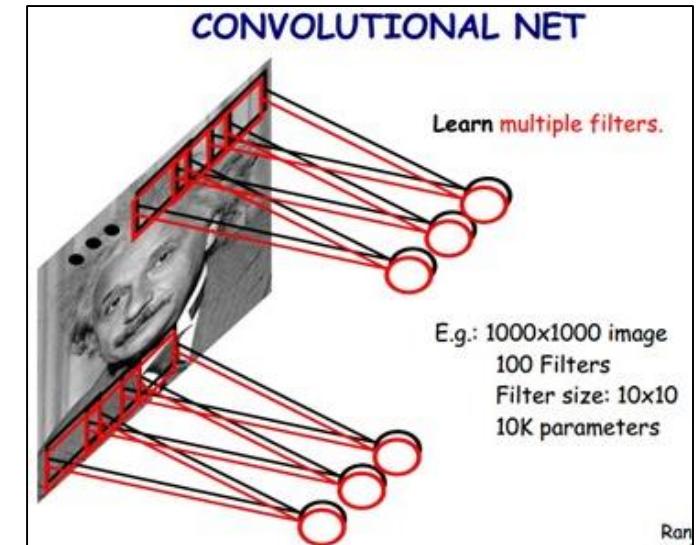
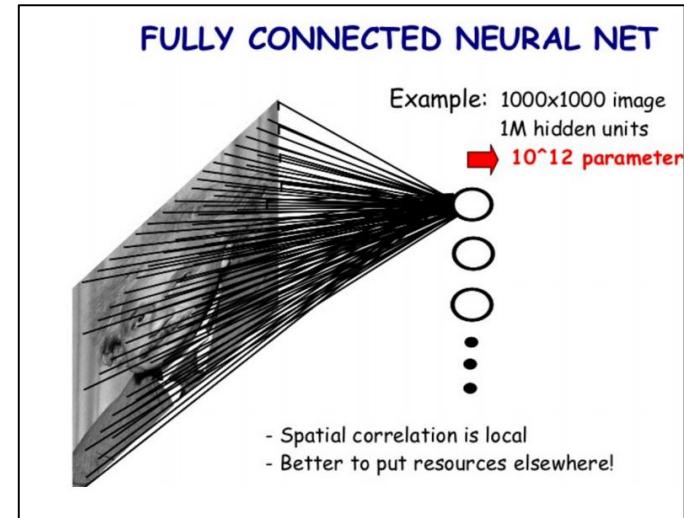
## Fully Connected Layer

- 영상의 다양한 변화에 대응하는 패턴을 학습하기가 어렵다.
- 입력 영상의 해상도에 비례하는 파라미터 수를 가진다.

## Convolution Layer

- 입력 영상내에서 인접한 픽셀사이에 존재하는 패턴을 학습한다.
- 영상의 다양한 변화에 상대적으로 자유롭다.

=> 과도하게 복잡한 패턴을 찾는 건 오히려 불리할 수 있다.



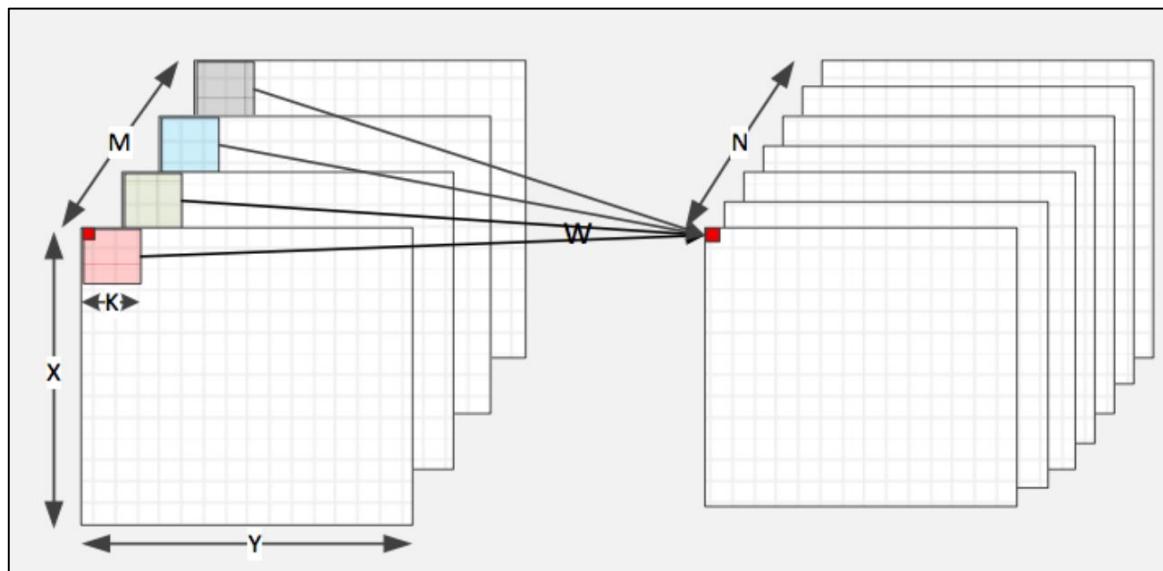
# Memory & Time Cost

연산량은  $MXYK^2N$ 가 되고 파라미터의 수는  $N(MK^2 + 1)$ 개가 된다.

- M는 입력 채널 수, (X, Y)는 입력 영상의 크기, (K, K)는 필터 커널의 크기, N은 필터의 수(출력 채널)

영상의 크기 뿐만 아니라 많은 요소들에 의해 연산량과 파라미터 수가 증가한다.

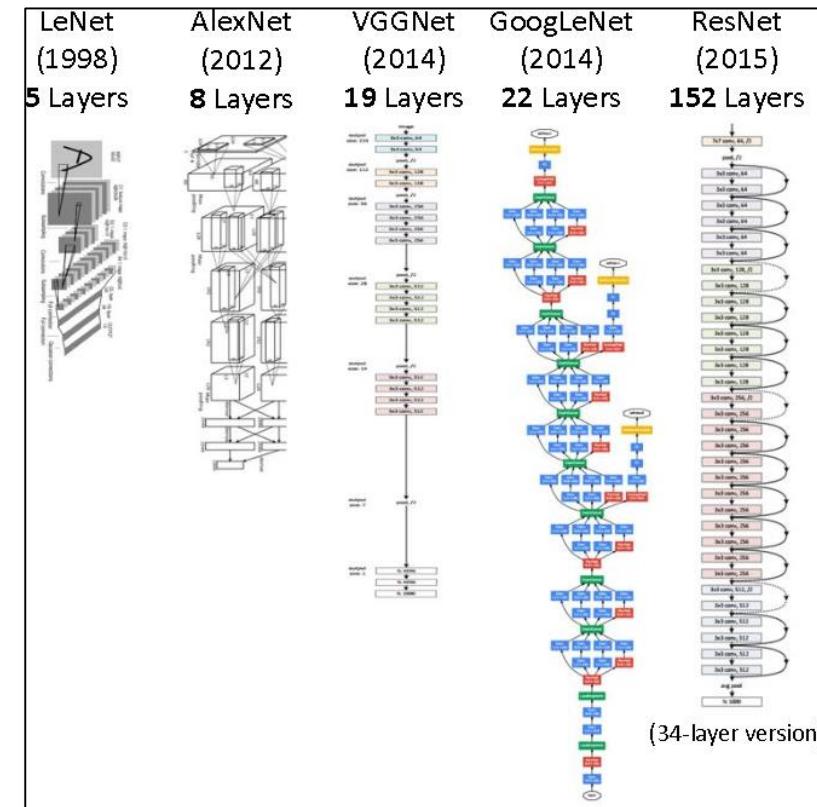
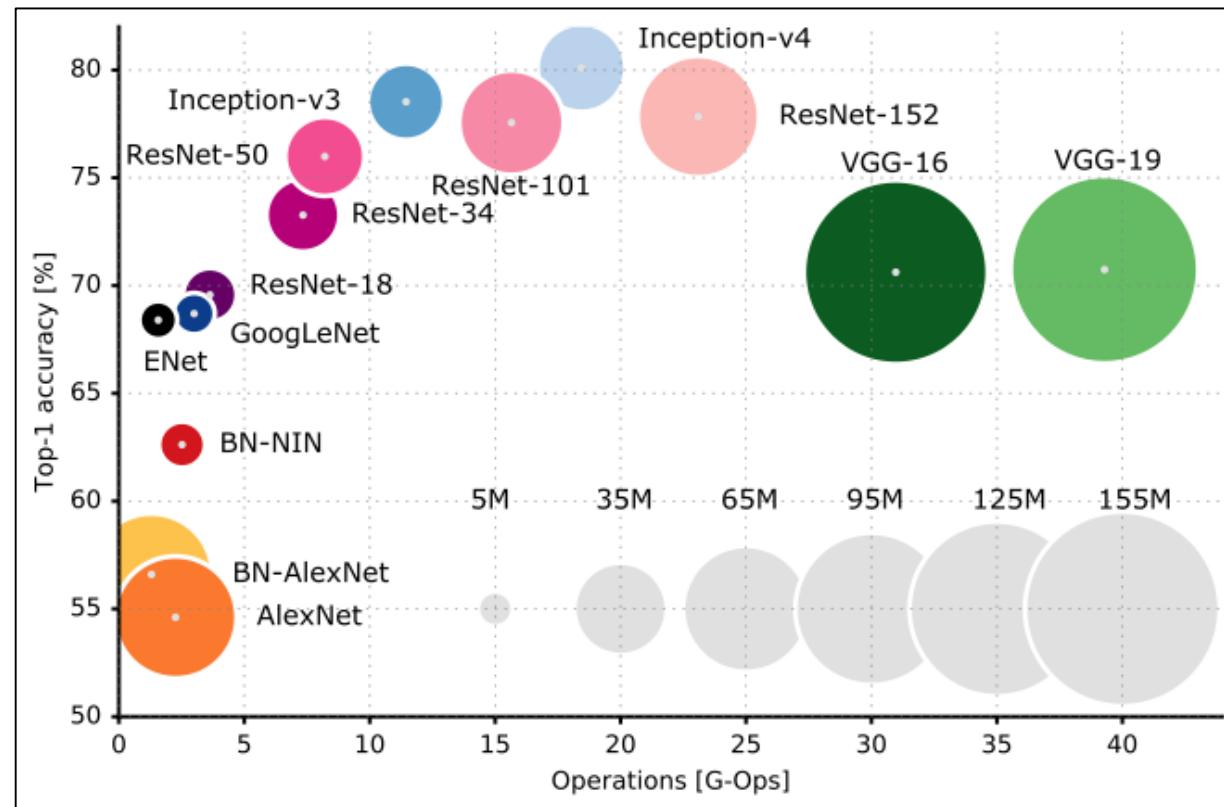
- 입력 채널의 수
- 필터의 수
- 커널의 크기



# Problem #1 – Expensive Cost

네트워크는 점점 깊어지고, 채널의 수는 증가한다.

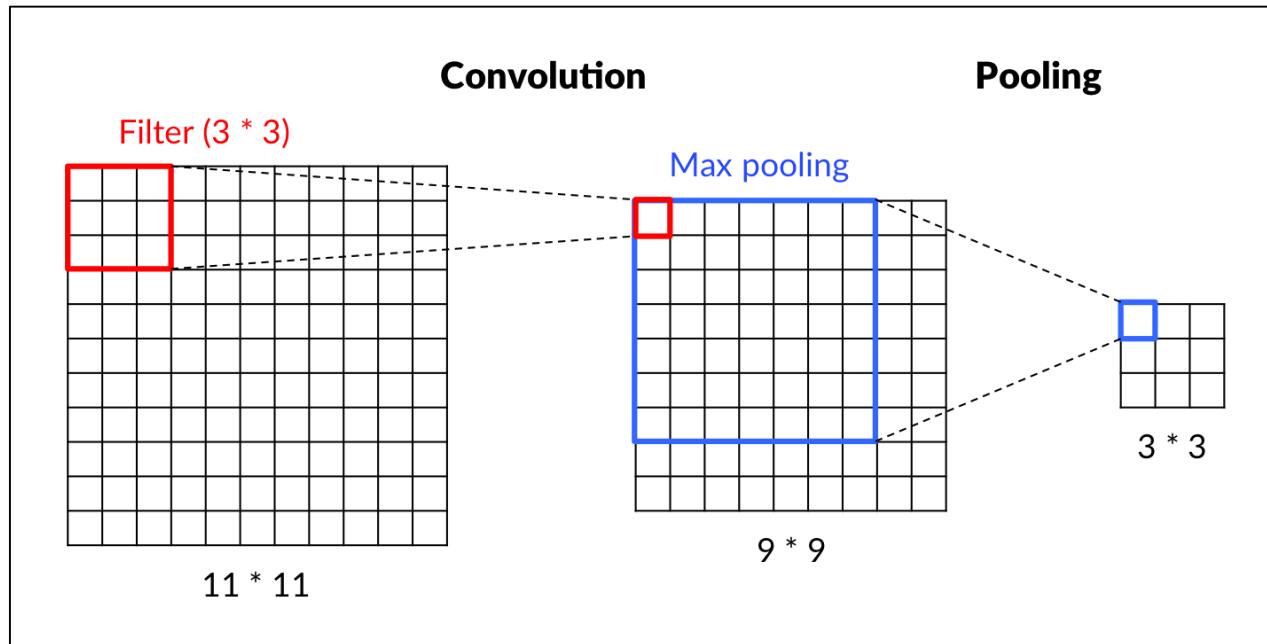
↔ 연산량도 가중치도 비례해서 증가한다.



# Problem #1 – Expensive Cost

일반적으로 큰 Feature Map을 컨볼루션의 입력으로 가지는 것이 유리하지만, 대부분의 CNN에서는 코스트를 고려해 Sub-sampling을 수행한다.

큰 Feature Map == 더 많은 Information ?



<https://saitoxu.io/2017/01/01/convolution-and-pooling.html>

# Problem #1 – Expensive Cost

채널의 수를 늘릴수록 더 다양한 Filter를 학습할 수 있다.

Group name	Output size	ResNet	Wide ResNet	PyramidNet
conv1	32x32	3x3, 16	3x3, 16	3x3, 16
conv2	32x32	3x3, 32 3x3, 32 3x3, 32	3x3, 64 3x3, 64 3x3, 64	3x3, 42 3x3, 69 3x3, 96
conv3	16x16	3x3, 64 3x3, 64 3x3, 64	3x3, 128 3x3, 128 3x3, 128	3x3, 122 3x3, 149 3x3, 176
conv4	8x8	3x3, 128 3x3, 128 3x3, 128	3x3, 256 3x3, 256 3x3, 256	3x3, 202 3x3, 299 3x3, 256

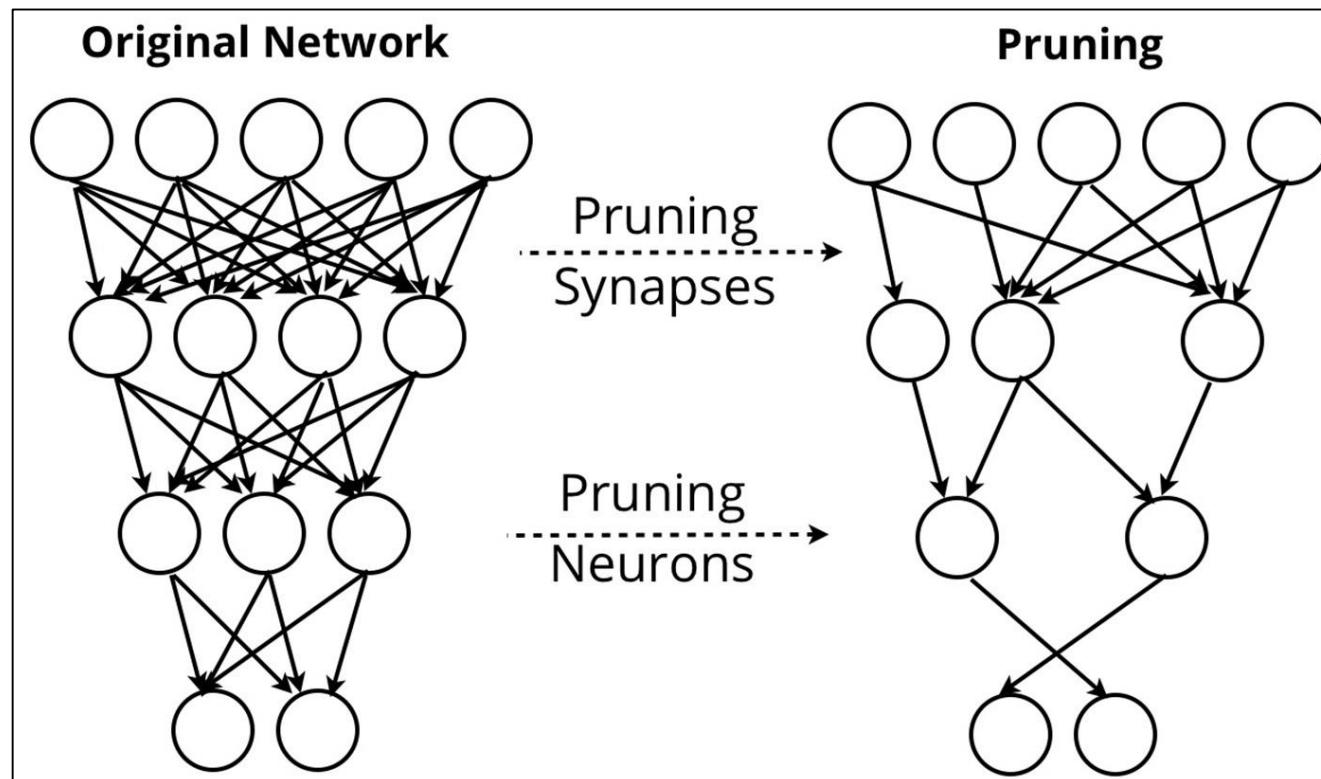
하지만 너무 많은 채널은 오히려 비용을 증가시킨다

- 파라미터의 수 증가
- 학습 속도 하락
- Overfitting
- ...

## Problem #2 – Dead Channels

신경망의 학습 과정에서, 출력 결과에 영향을 거의 주지 않는 간선(혹은 정점)이 나타난다.

- 일반적으로 Pruning을 통해 신경망의 연산량과 파라미터 수를 경량화 할 수 있다.
- Edge Pruning이 Neuron Pruning에 비해 상대적으로 간단하다.



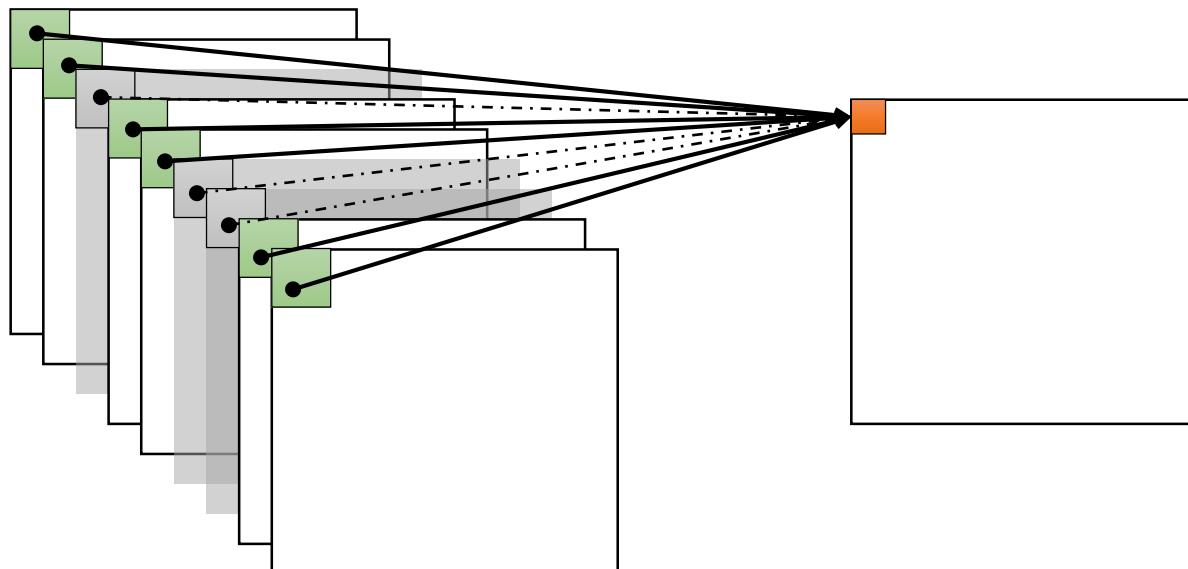
## Problem #2 – Dead Channels

마찬가지로, 이런 문제들이 채널 단위에서 발생할 수 도 있다.

- 좁게는 하나의 필터에서, 넓게는 전체 신경망에서 불필요한 채널이 발생할 수 있다.
- 일종의 Neuron Pruning으로 볼 수 있으며, 제거 여부를 판단하기도 어렵다.

채널의 수는 곧 파라미터 수와 연산량에 직결되기 때문에, 상당한 자원 낭비가 될 수 있다.

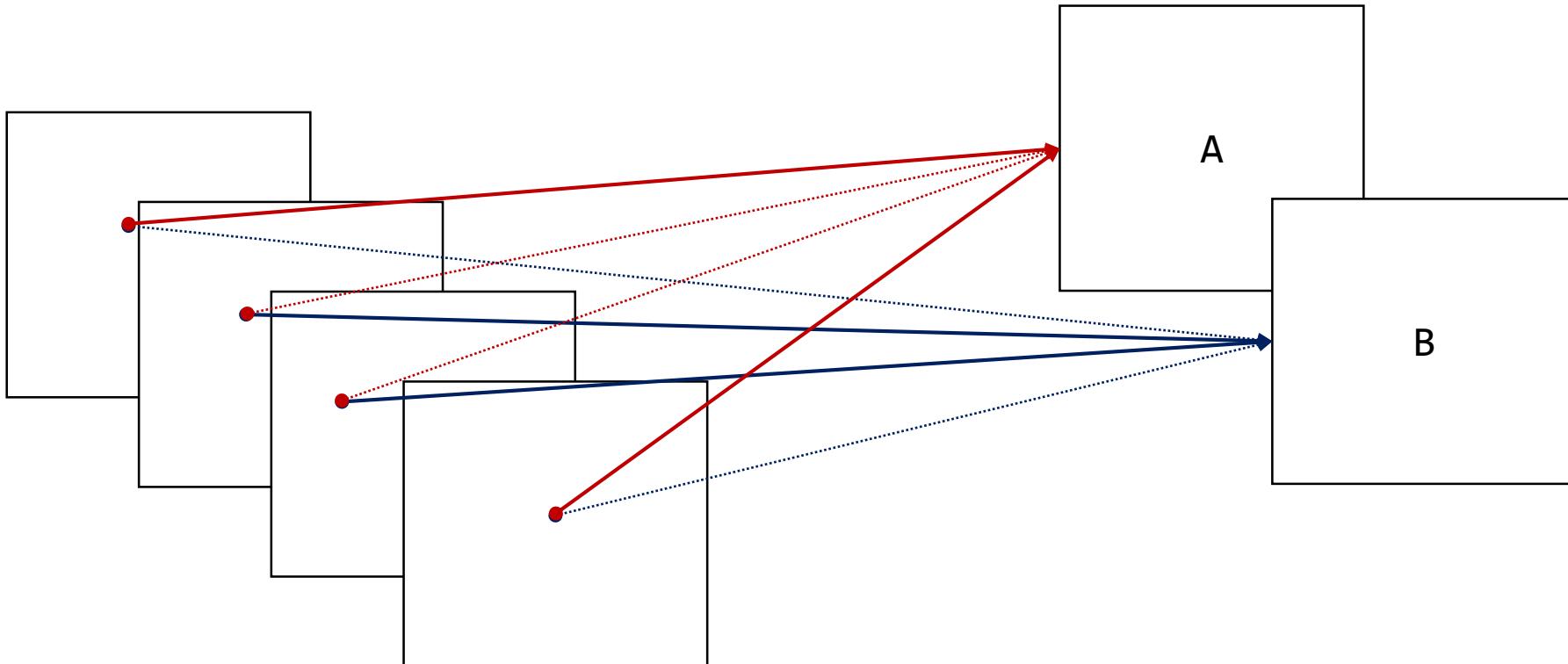
- 당연히 채널의 수는 Hyper Parameter이기 때문에 최적의 수를 찾는 것도 한계가 있다.



불필요한 채널일지라도 가중치를 가지고, 계산 시간을 소요한다

## Problem #3 – Low Correlation between channels

각 필터는 입력 영상의 모든 채널을 사용하지만,  
모든 채널-필터 쌍이 높은 Correlation을 가질 수 만은 없다.

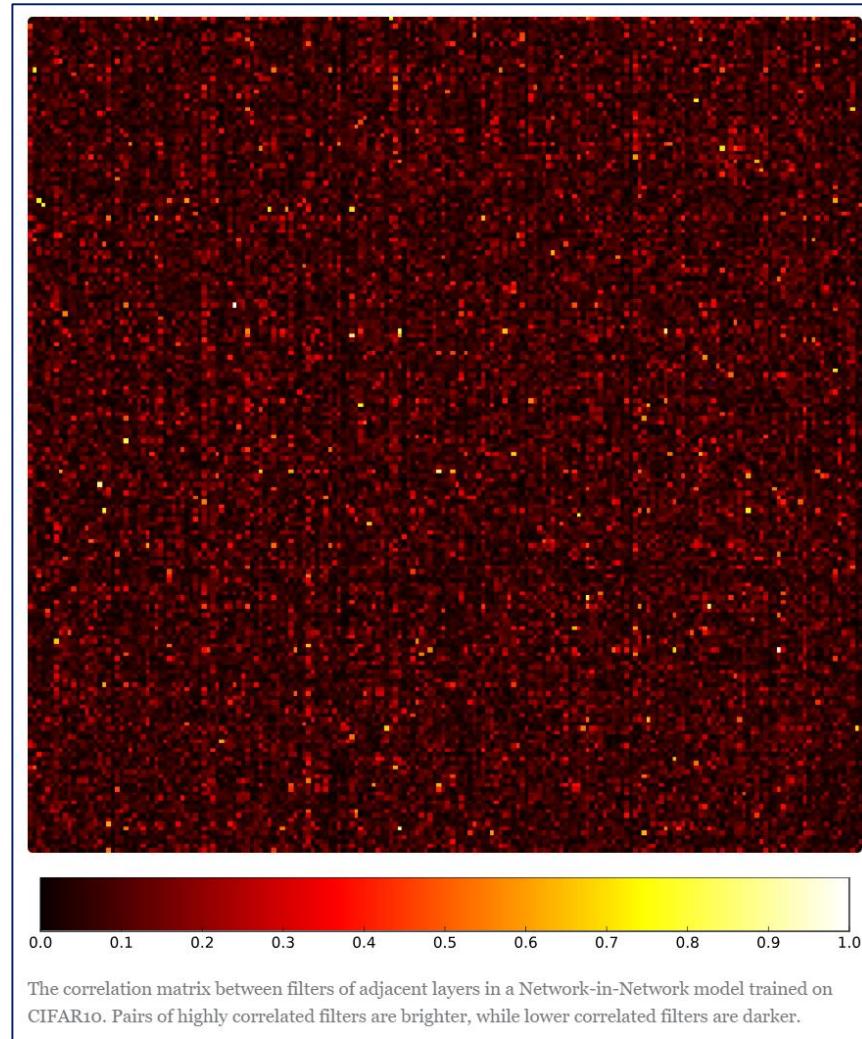
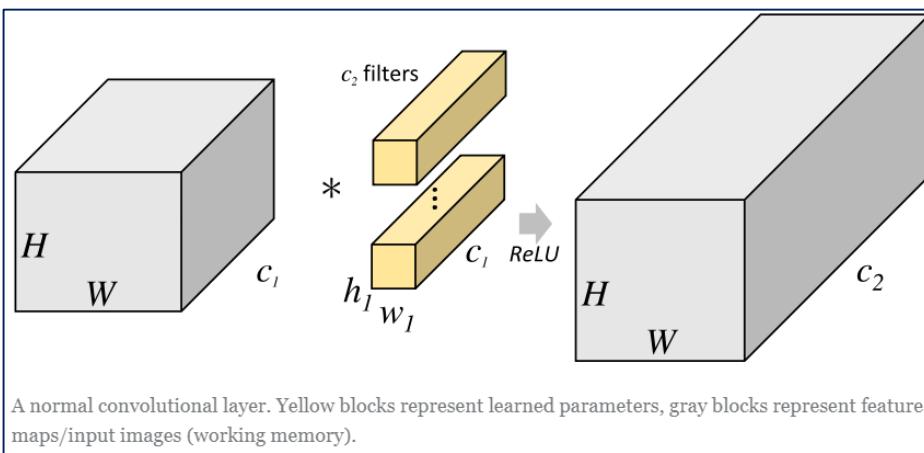


# Problem #3 – Low Correlation between channels

결과적으로 Sparse한 Correlation Matrix를 보일 수 있다.

이로 인해 다음과 같은 문제점이 생긴다

- 학습 수렴 속도 저하
- 일종의 Noise처럼 작용
- 불필요한 가중치와 연산의 존재



# Abstract

요약하자면, Convolution Layer에는 크게 다음과 같은 세 가지 문제점이 존재한다.

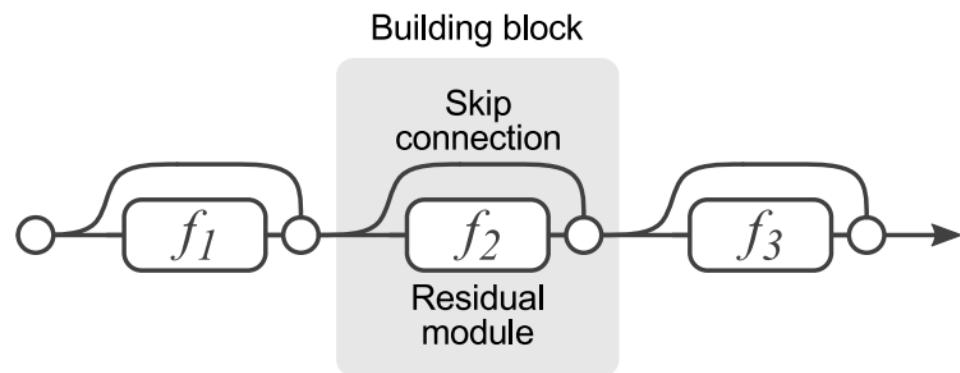
- Expensive Cost
- Dead Channels
- Low Correlation between channels

본 발표에서는 이런 문제점들을 해결하기 위한 다양한 관점의 접근 방법들과 관련 논문들의 아이디어에 대해서 알아본다.

1. Residual Connection
2. Dilated Convolution
3. Point-wise Convolution
4. Grouped Convolution
5. Depth-wise (Separable) Convolution

# Residual Connection

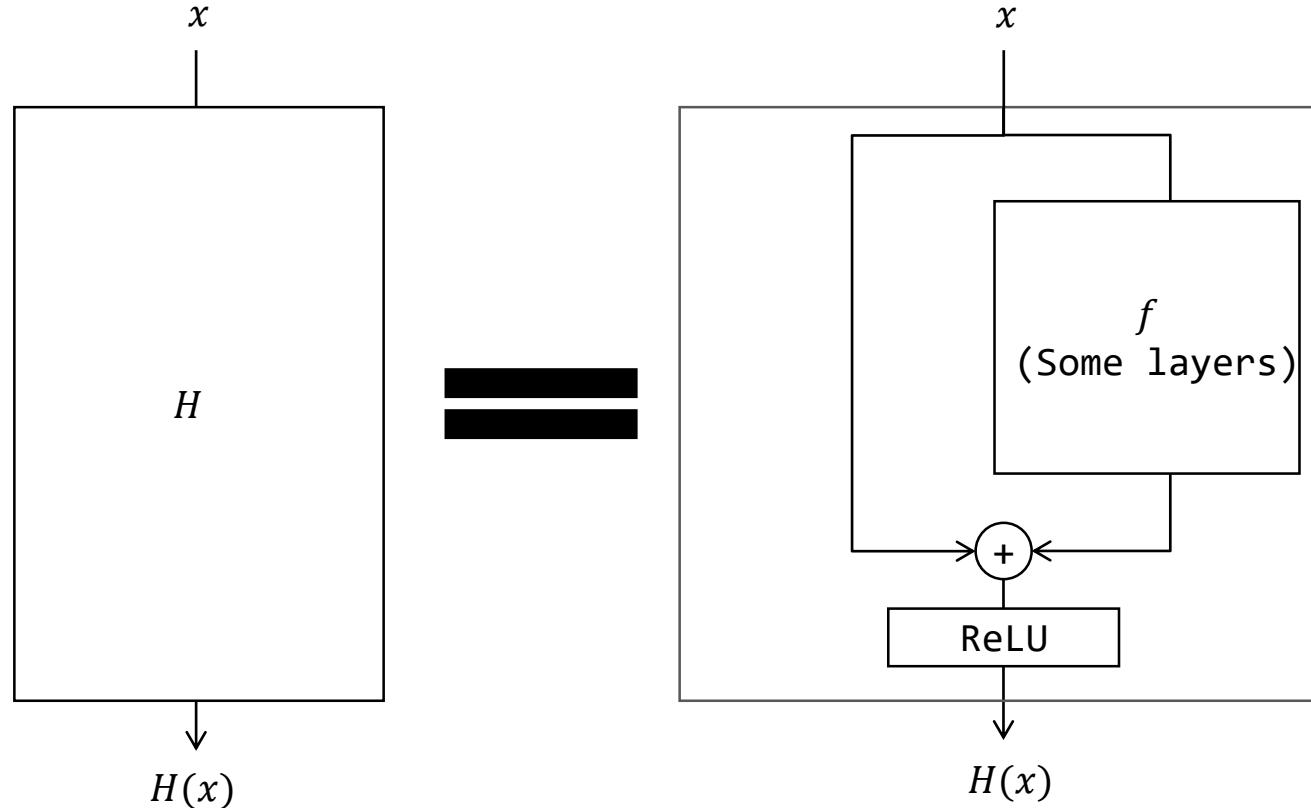
---



# Residual Connection

입력 영상과 레이어의 출력 영상의 Element-wise Addition으로 연결한다.  
입력과 출력 두 영상에 대한 Representation이 가능하다.

$$H(x) = x + f(x).$$



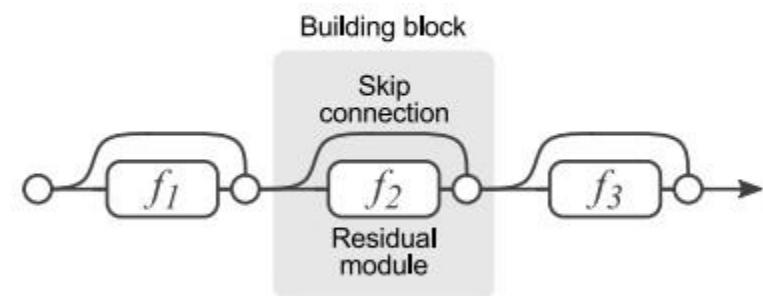
## Residual Networks Behave Like Ensembles of Relatively Shallow Networks (A. Veit et al.)

이러한 구조는 적층을 하면 할수록 더 Complex한 특징의 학습이 가능하다.

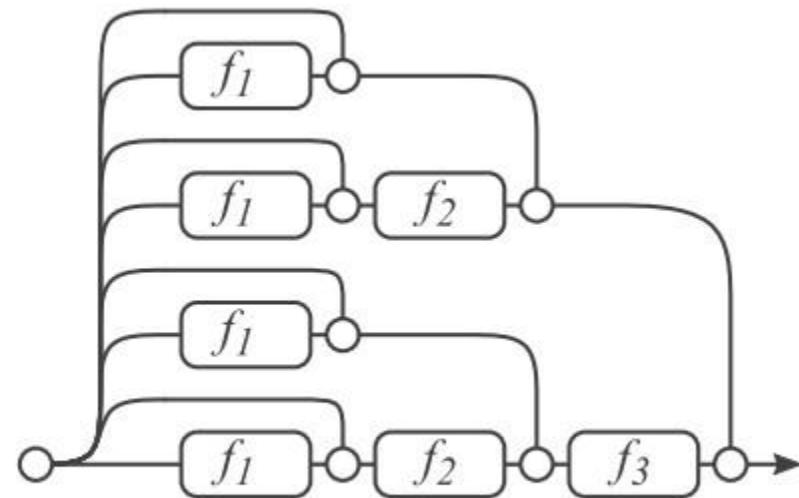
$$H(x) = x + f(x).$$

$$H(H(x)) = H(x) + f(H(x)) = x + f(x) + f(x + f(x))$$

...



(a) Conventional 3-block residual network



(b) Unraveled view of (a)

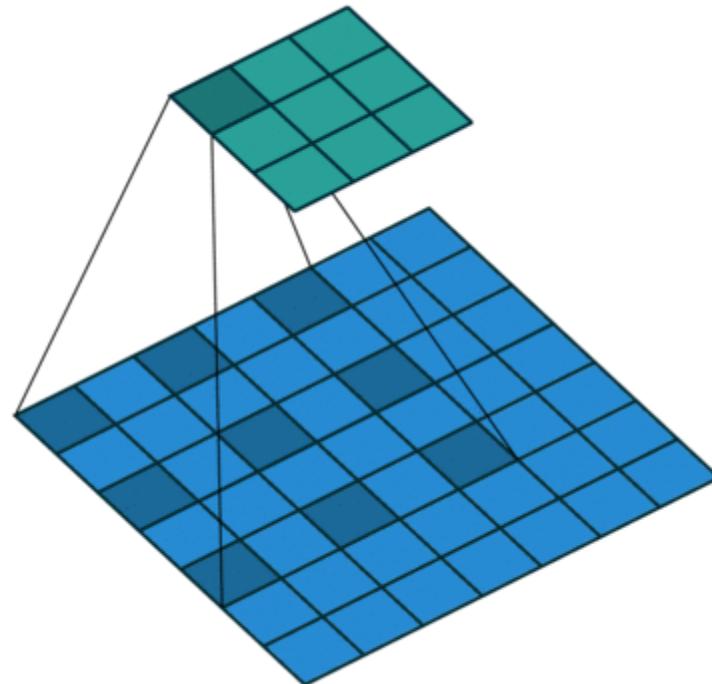
## Residual Networks Behave Like Ensembles of Relatively Shallow Networks (A. Veit et al.)

결과적으로, 동일한 파라미터 수를 통해 더욱 다양한 형태의 Feature Representation을 가진다.

다양한 실험에서 학습 속도 개선 등이 나타나는 것은 덤

# Dilated Convolution

---



# Contextual Information & Receptive Fields

영상 내의 Objects에 대한 정확한 판단을 위해서는 Contextual Information이 중요하다

- 객체 주변의 배경은 어떤 환경인가?
- 객체 주변의 다른 객체들은 어떤 종류인가?

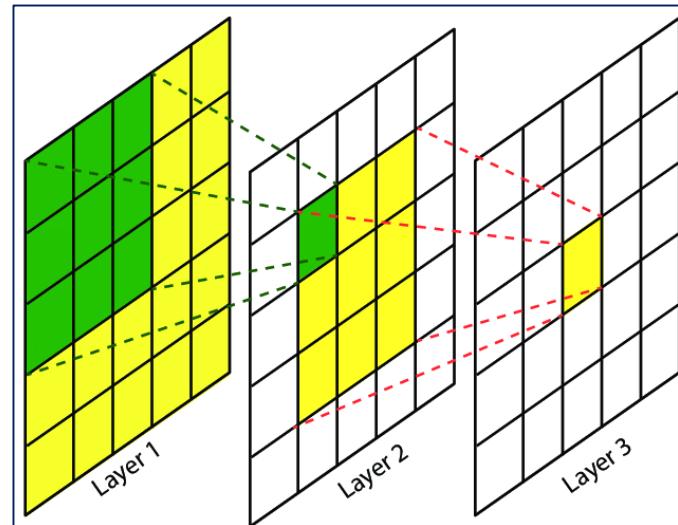


# Contextual Information & Receptive Fields

충분히 Contextual Information을 반영하기 위해서는?

- 컨볼루션 필터의 Receptive Field를 확장할 필요가 있다.

Receptive Field는 해당 Feature를 추출하기 위해 고려된 원본 영상에서의 영역을 나타낸다.

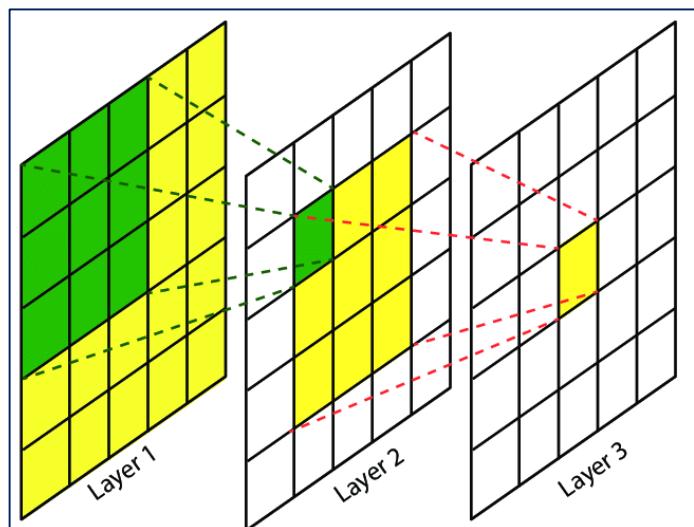


# Contextual Information & Receptive Fields

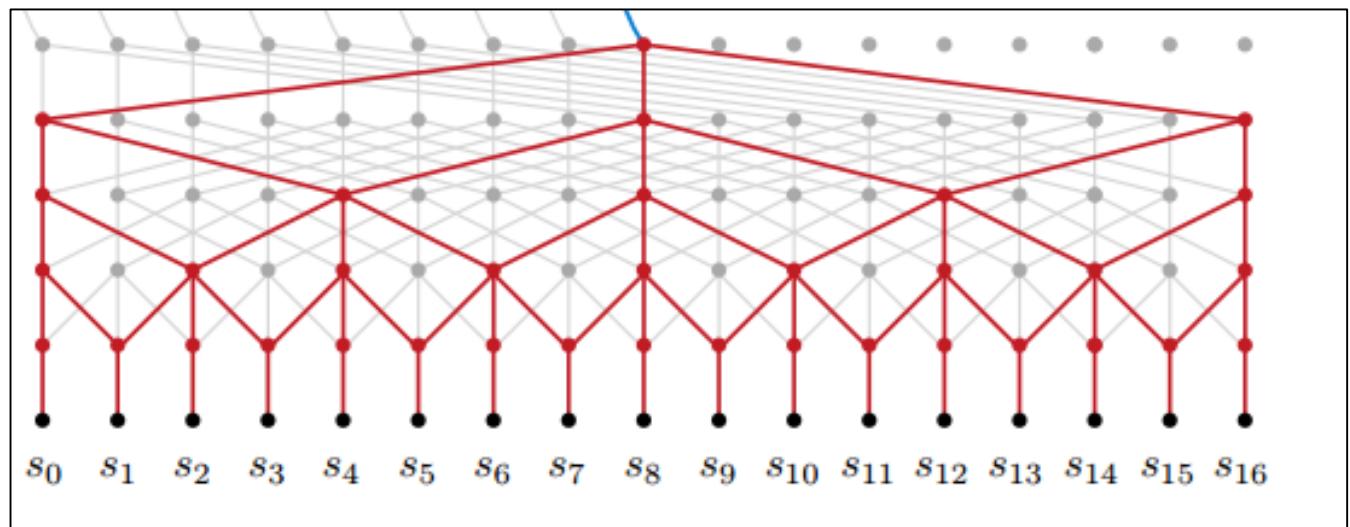
일반적으로 CNN에서 Receptive Field를 확장하기 위해서는 다음을 고려해볼 수 있다.

- 커널 크기의 확장
- 더 많은 컨볼루션 레이어의 적층

하지만 두 방법 모두 연산량을 증가시킨다.



<https://blog.yani.io/filter-group-tutorial/>



<https://medium.com/@TalPerry/convolutional-methods-for-text-d5260fd5675f>

# Dilated Convolution

Convolution filter의 수용 픽셀사이에 간격을 둔 형태

입력 픽셀의 수는 동일하지만, 더 넓은 범위에 대한 입력을 수용할 수 있게 된다.

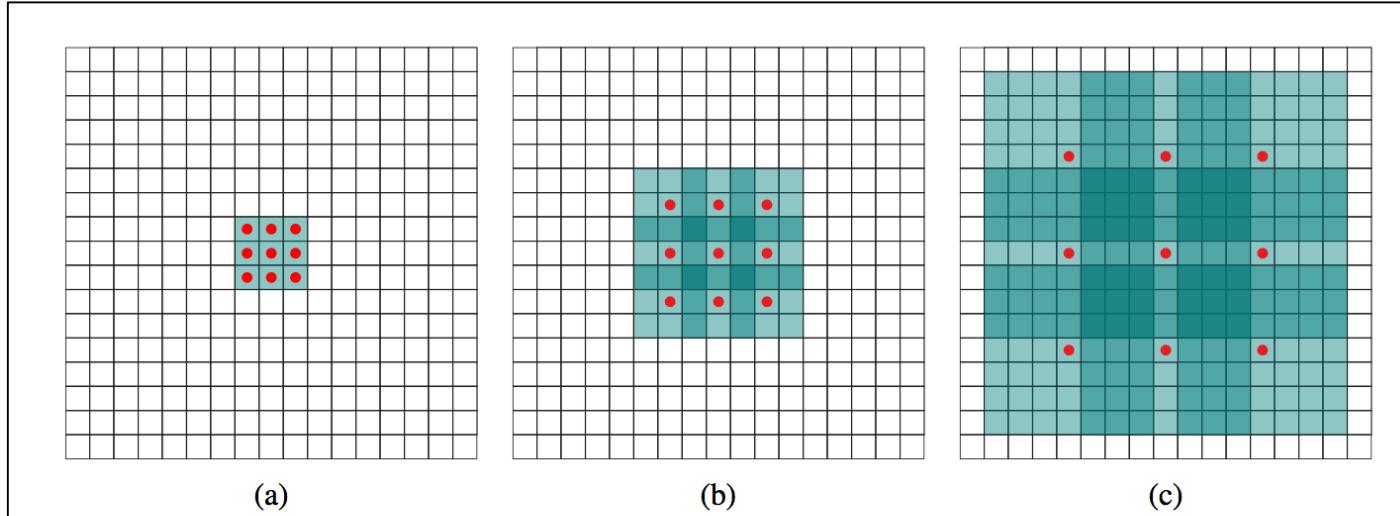
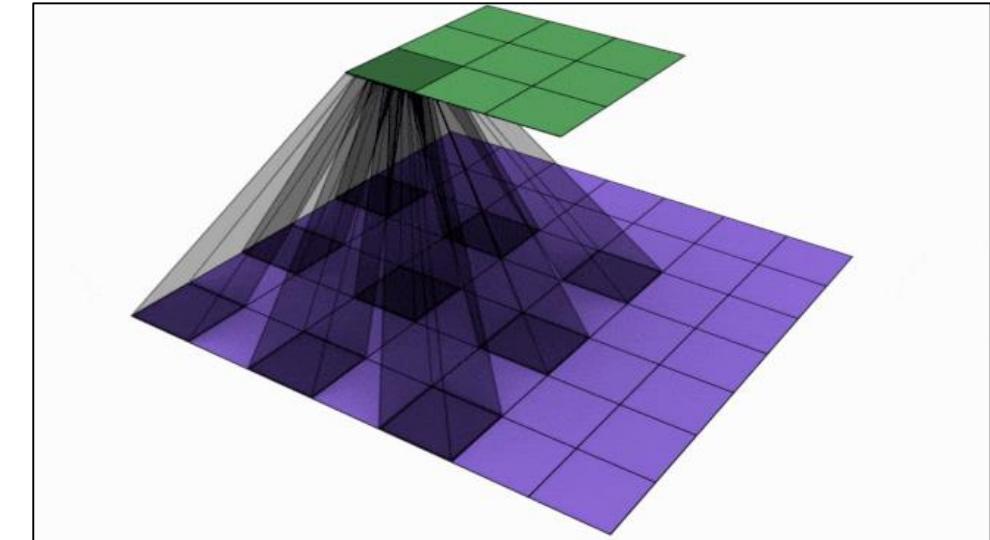


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a)  $F_1$  is produced from  $F_0$  by a 1-dilated convolution; each element in  $F_1$  has a receptive field of  $3 \times 3$ . (b)  $F_2$  is produced from  $F_1$  by a 2-dilated convolution; each element in  $F_2$  has a receptive field of  $7 \times 7$ . (c)  $F_3$  is produced from  $F_2$  by a 4-dilated convolution; each element in  $F_3$  has a receptive field of  $15 \times 15$ . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.



<https://mlblr.com/includes/mlai/index.html>

# Dilated Convolution

기존 컨볼루션 레이어의 Stride, Pooling으로 인해 발생하는 정보손실을 최소화

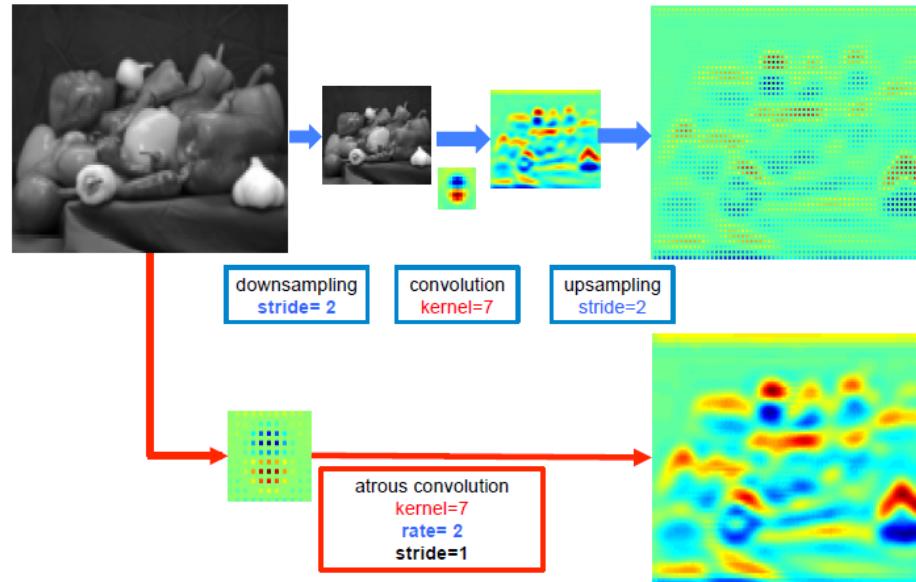


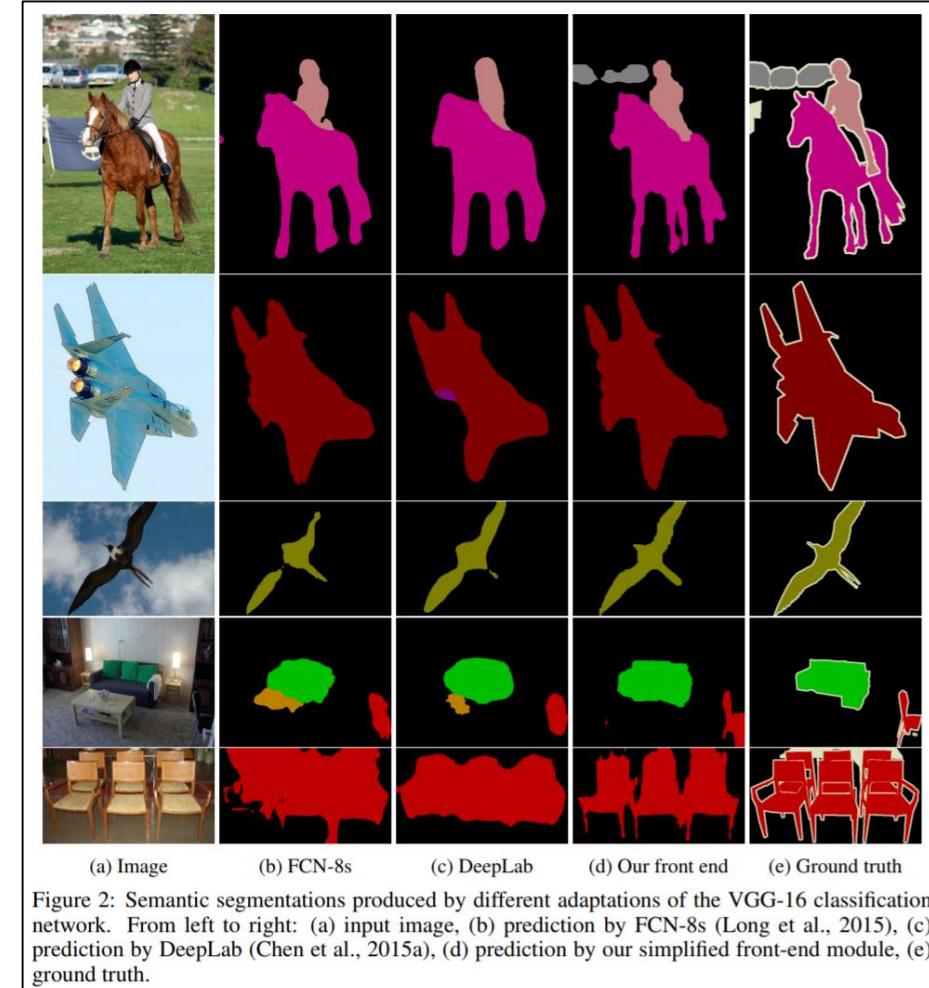
Fig. 3: Illustration of atrous convolution in 2-D. Top row: sparse feature extraction with standard convolution on a low resolution input feature map. Bottom row: Dense feature extraction with atrous convolution with rate  $r = 2$ , applied on a high resolution input feature map.

# Dilated Convolution

실제 Receptive Field 상의 모든 정보를 입력 받지는 않는다.

하지만, 같은 수의 파라미터를 통해 상대적으로 넓은 영역의 정보들을 수용할 수 있다.

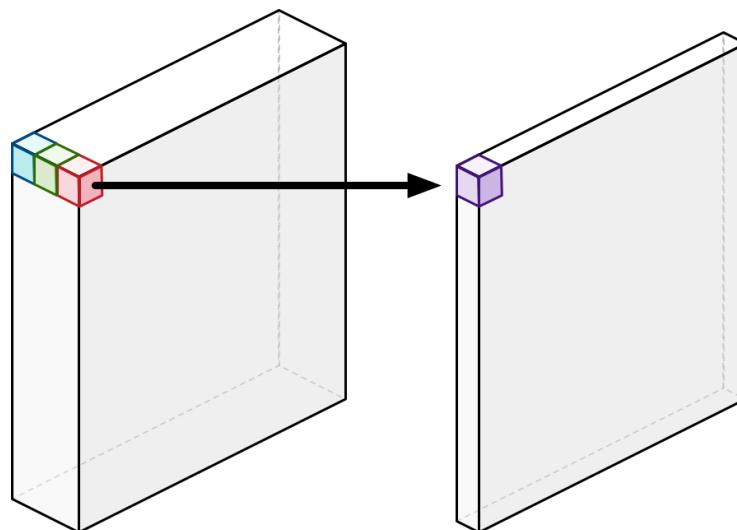
- Segmentation, Object Detection과 같이 상대적으로 Contextual Information이 중요한 문제에 적용하기 유리하다.
- 간격을 조절하여 다양한 Scale에 대한 대응이 가능하다.



# Point-wise Convolution

---

Convolution Layer with a  $1 \times 1$  kernels

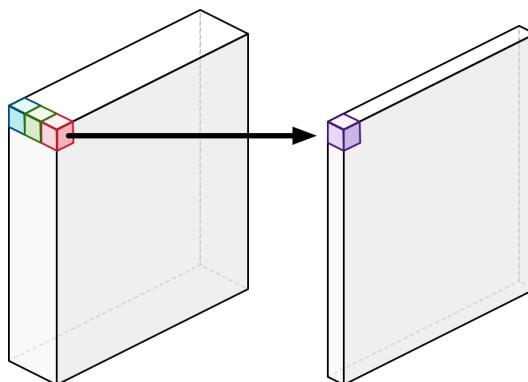


# Point-wise Convolution (1 by 1 Convolution)

Point-wise Convolution은 커널 크기가  $1 \times 1$ 로 고정된 Convolution Layer를 말한다.

입력 영상에 대한 Spatial Feature는 추출하지 않은 상태로,  
각 채널들에 대한 연산만을 수행한다.

- 출력 영상의 크기는 변하지 않으며, 출력 채널의 수(필터 수)를 자유롭게 조정할 수 있다.

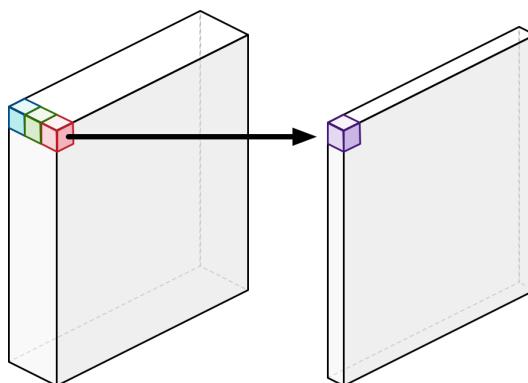


# Point-wise Convolution (1 by 1 Convolution)

커널 크기가  $1 \times 1$ 로 고정된 컨볼루션 레이어이므로

연산량은  $MXYN$ 가 되고 파라미터의 수는  $N(M + 1)$ 개가 된다.

- M는 입력 채널 수, (X, Y)는 입력 영상의 크기, (K, K)는 필터 커널의 크기, N은 필터의 수(출력 채널)



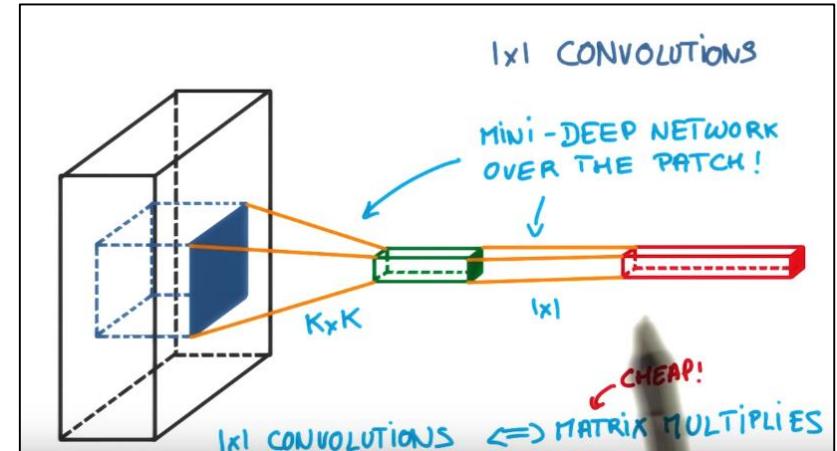
# Point-wise Convolution (1 by 1 Convolution)

하나의 필터는 각 입력 채널별로 하나의 가중치만을 가진다.

- 이 가중치는 해당 채널의 모든 영역에 동일하게 적용된다.

즉, 입력 채널들에 대한 Linear Combination과 같다.

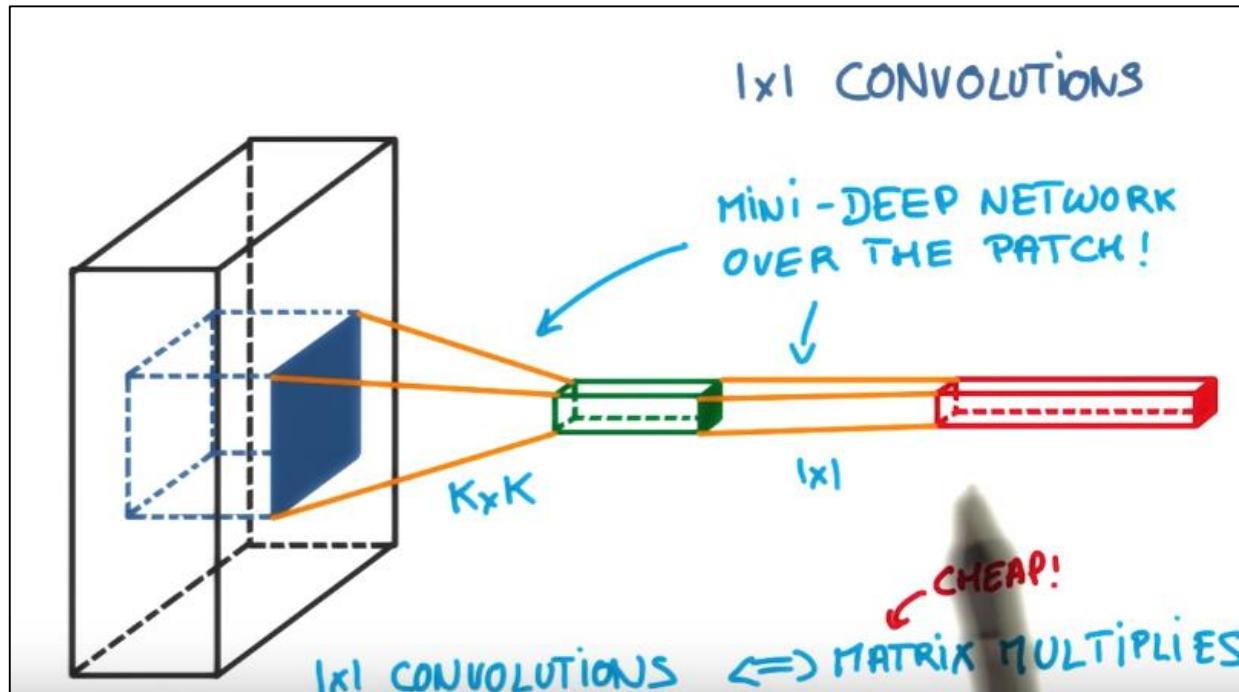
- 각 채널의 가중치는 Coefficient가 된다.



$$w_1 \times \text{[Blue Cube]} + w_2 \times \text{[Green Cube]} + w_3 \times \text{[Orange Cube]} + \dots + w_c \times \text{[Yellow Cube]} = \text{[Grey Cube]}$$

# Point-wise Convolution (1 by 1 Convolution)

즉, 채널 단위의 Linear Combination을 통한 채널 수의 변화가 가능하다.

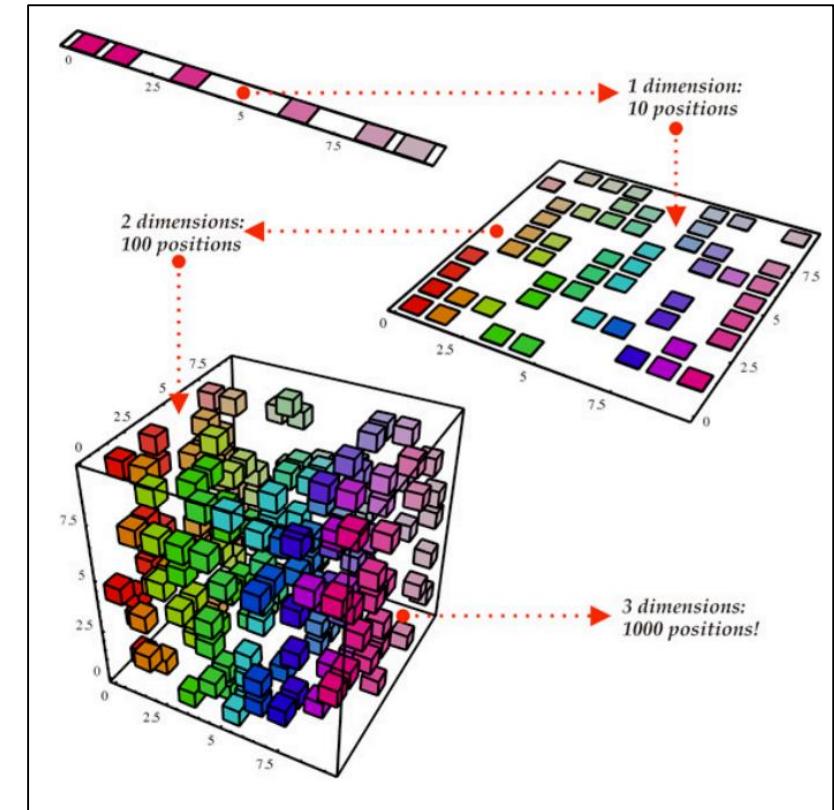


# Point-wise Convolution

일반적으로 Point-wise Convolution의 경우 출력 영상의 채널 수를 줄여 사용한다.  
이를 일종의 Dimensionality Reduction으로 이해할 수 있다.

즉, 다채널 입력 영상을 더 적은 채널의 영상(Low dimension)으로의 Embedding하는  
것으로 이해할 수 있다.

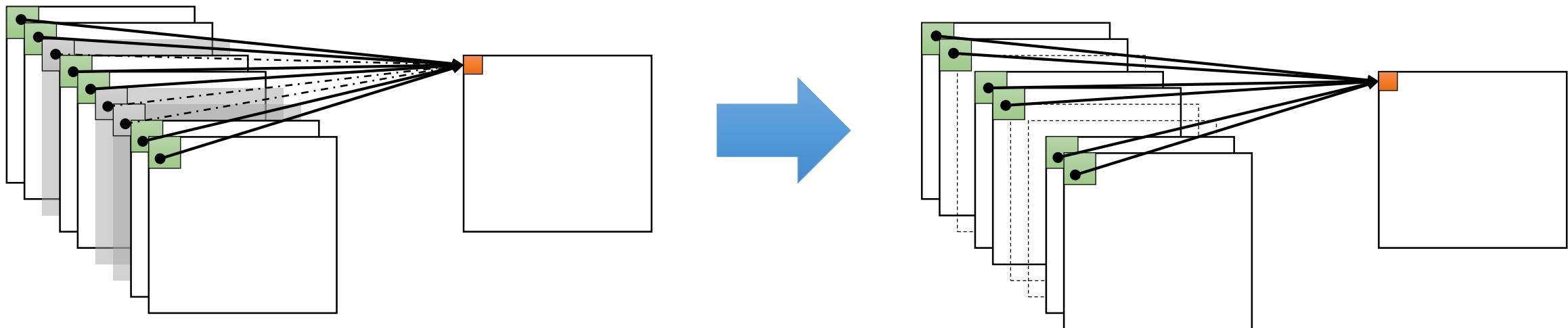
- 출력 채널 수를 줄임으로써, 다음 레이어의  
계산 코스트와 파라미터 수를 줄일 수 있다.



# Point-wise Convolution (1 by 1 Convolution)

채널에 대한 Linear Combination을 수행할 시,

- 불필요한 채널들이 낮은 Coefficient를 가지며 연산 결과에서 희석될 수 있다.
- 연관된 채널들이 조합된 새로운 Feature Map을 기대할 수 있다.



# Point-wise Convolution (1 by 1 Convolution)

이미 많은 구조에서 활용되고 있다.

- ResNet
- Inception
- Squeeze Net
- ...

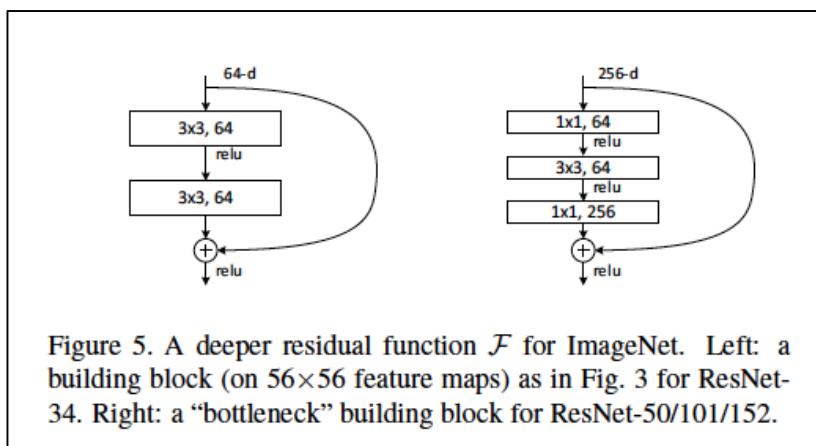
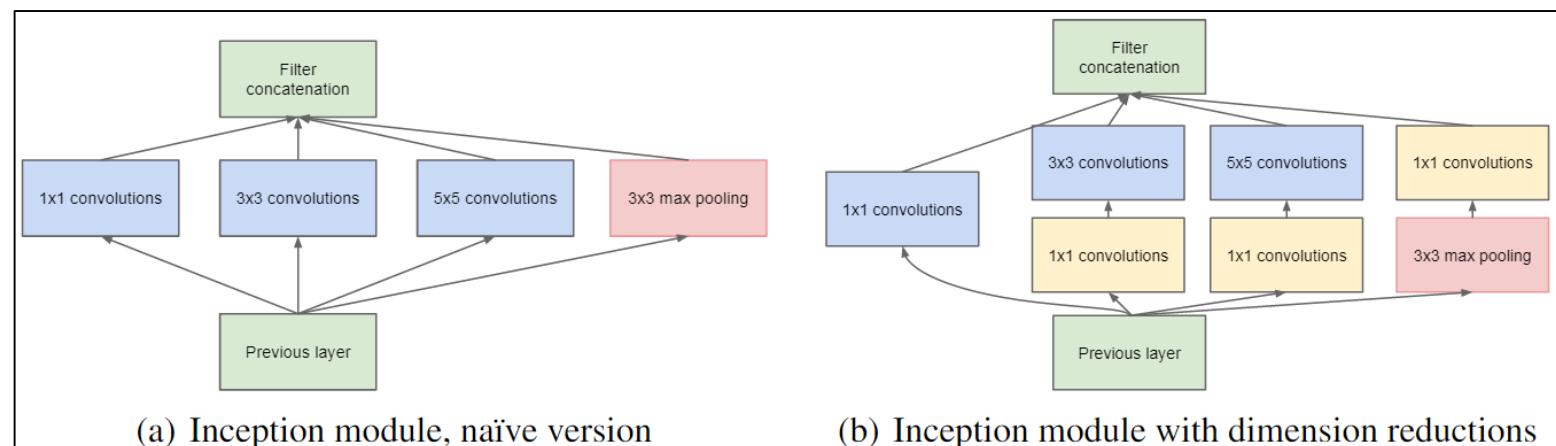


Figure 5. A deeper residual function  $\mathcal{F}$  for ImageNet. Left: a building block (on  $56 \times 56$  feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.



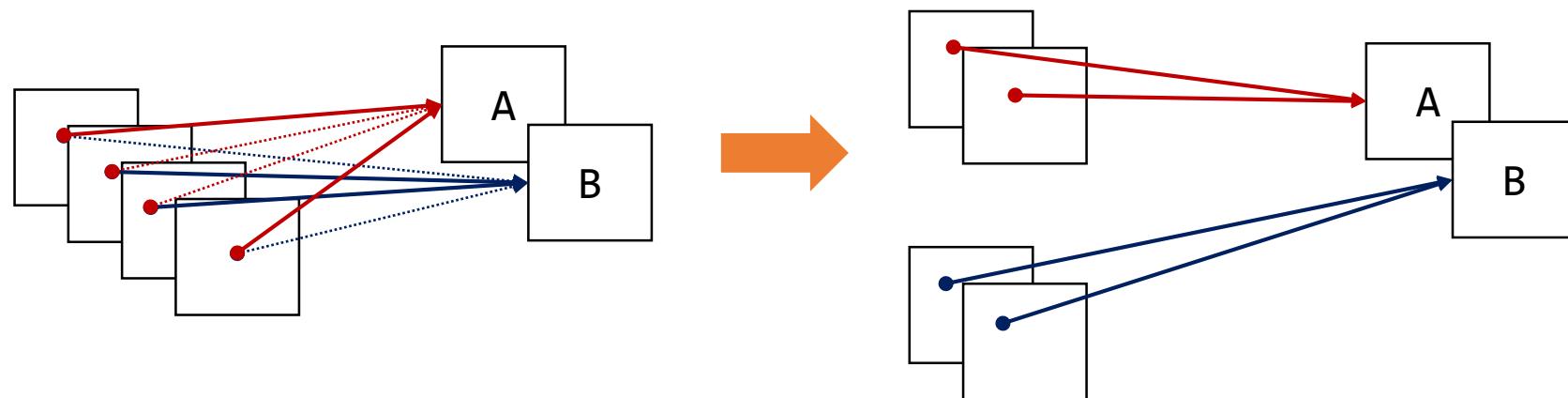
(a) Inception module, naïve version

(b) Inception module with dimension reductions

# Grouped Convolution

---

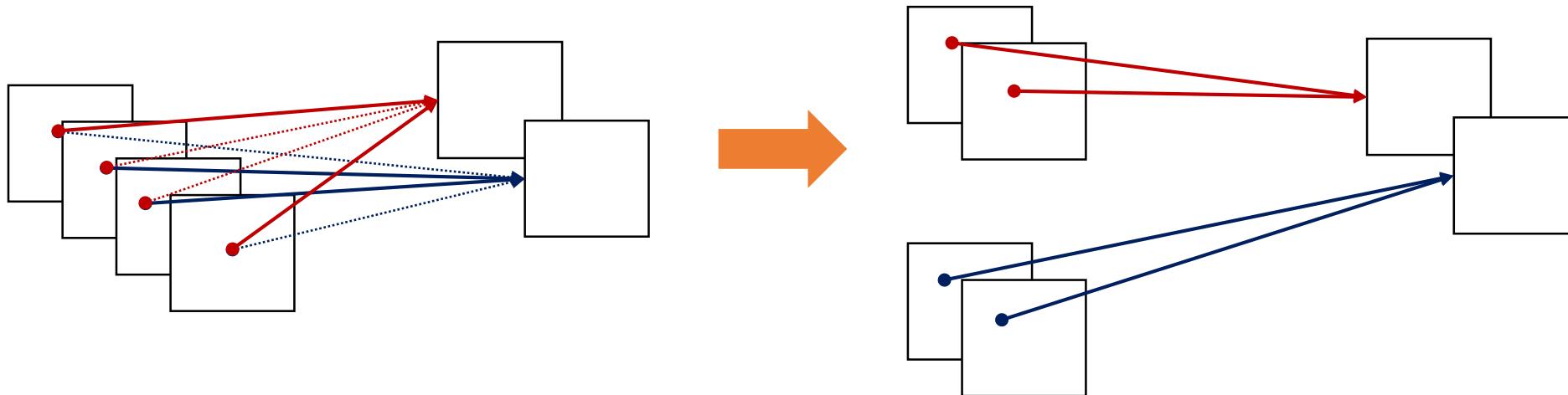
Dividing channels into independent groups



# Grouped Convolution

입력 영상의 채널들을 여러 개의 그룹으로 나누어 독립적인 컨볼루션을 수행하는 방식

- 아이디어와 구현 방법이 간단하다
- 병렬처리에 유리하다



```
def grouped_convolution(input, n_groups):
    input_groups = tf.split(input, n_groups, axis = -1)

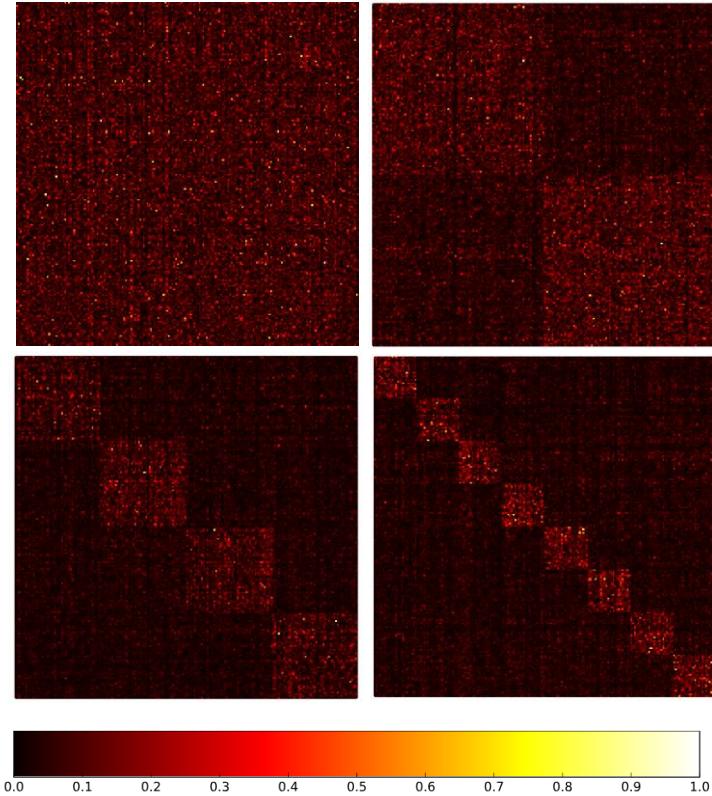
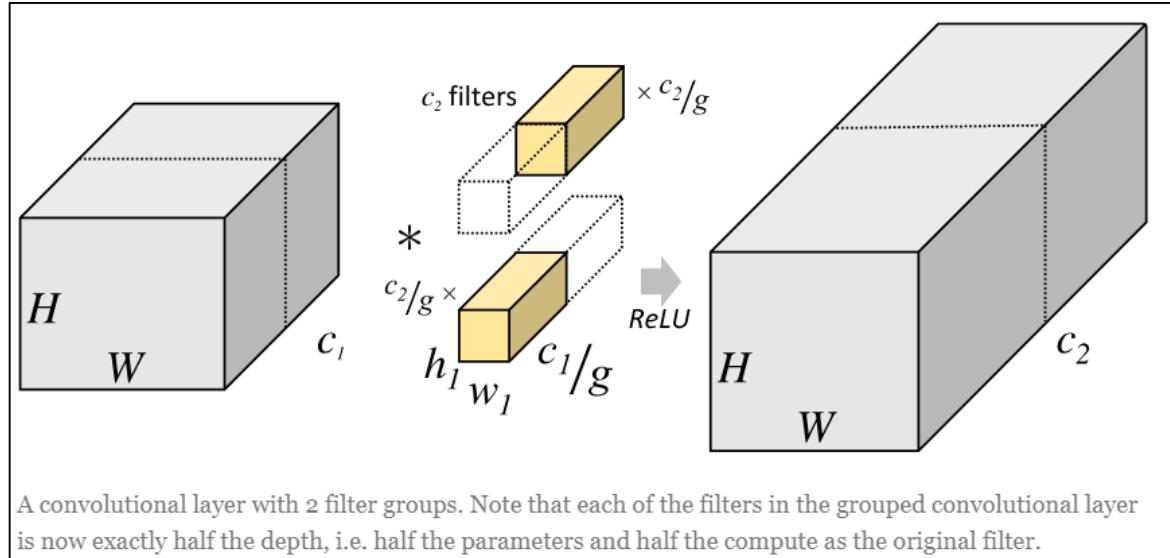
    output_groups = [
        tf.layers.conv2d(input_groups[i], """ some params """) for i in range(n_groups)
    ]

    output = tf.concat(output_groups, axis = -1)
    return output
```

# Grouped Convolution

각 그룹의 입력 채널들 사이에 독립적인 필터를 학습한다.

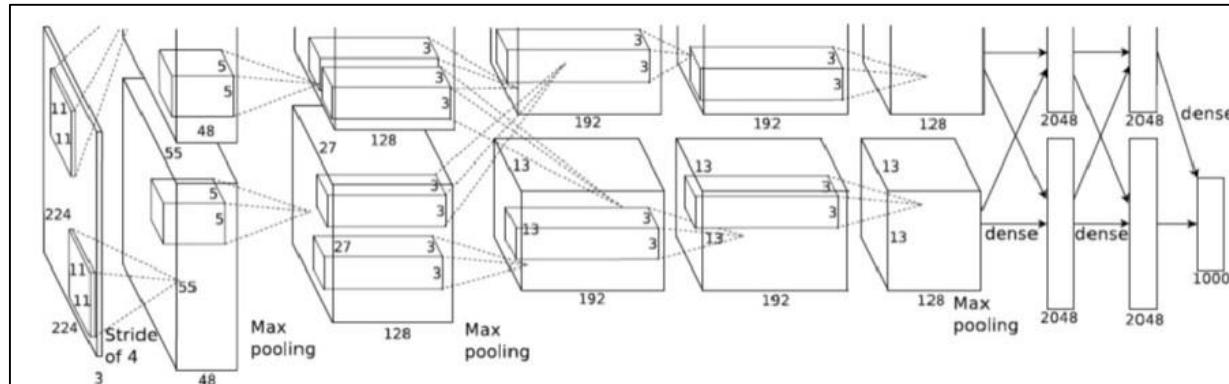
- 더 낮은 파라미터 수와 연산량을 가진다
- 각 그룹에 높은 Correlation을 가지는 채널들이 학습될 수 있다



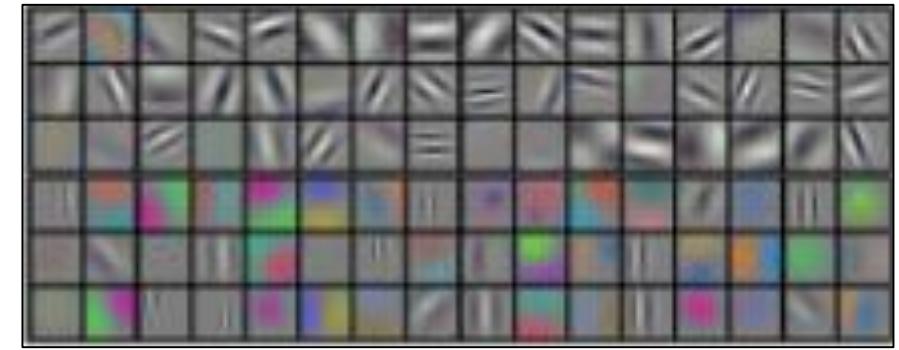
# Grouped Convolution

결과적으로 각 그룹마다 독립적인 필터의 학습을 기대할 수 있다.

- AlexNet에 사용된 Grouped Convolution의 예시



Example : AlexNet



음영과 외곽선 위주의 커널을 학습한 그룹(위)  
색상과 패턴 위주의 커널을 학습한 그룹(아래)

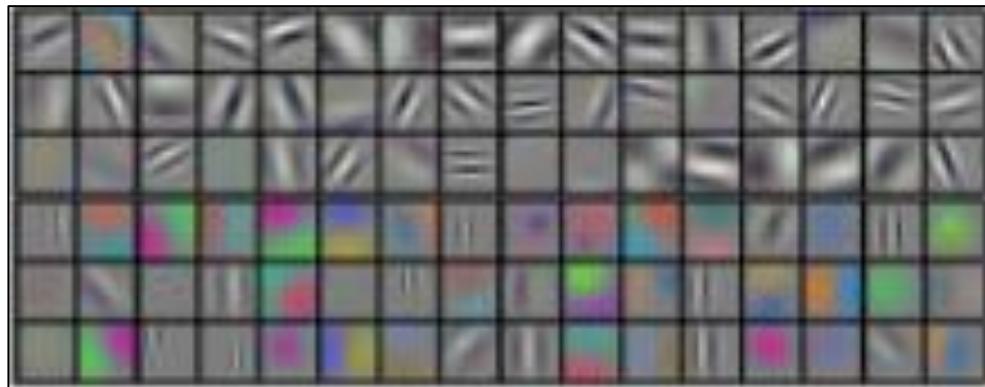
# Grouped Convolution

그룹 수를 조정해 독립적인 필터 그룹을 조정할 수 있다.

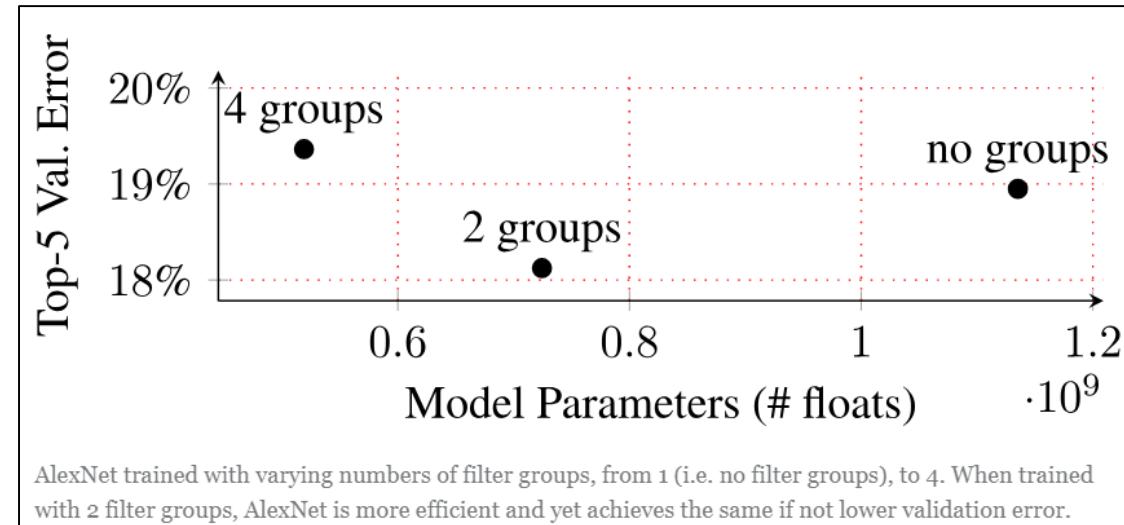
- 그룹의 수를 늘리면 파라미터의 수는 줄면서도, 성능 향상이 일어나는 경우가 있다.

그룹의 수는 Hyper Parameter라는 단점이 있다.

- 너무 많은 그룹으로의 분할은 오히려 성능이 하락할 수 있다.



음영과 외곽선 위주의 커널을 학습한 그룹(위)과  
색상과 패턴 위주의 커널을 학습한 그룹(아래)

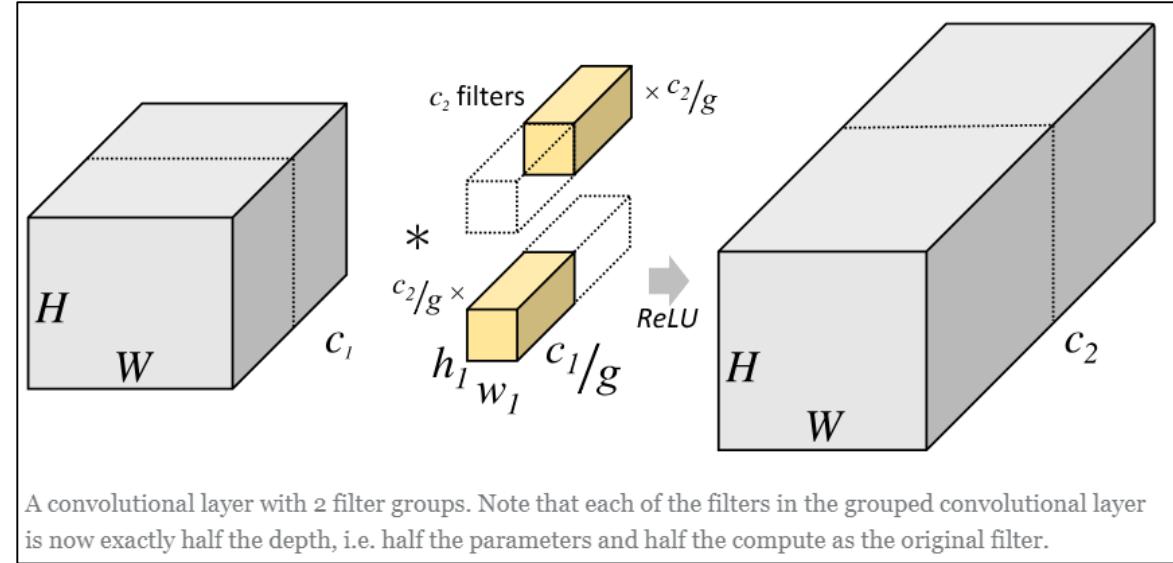
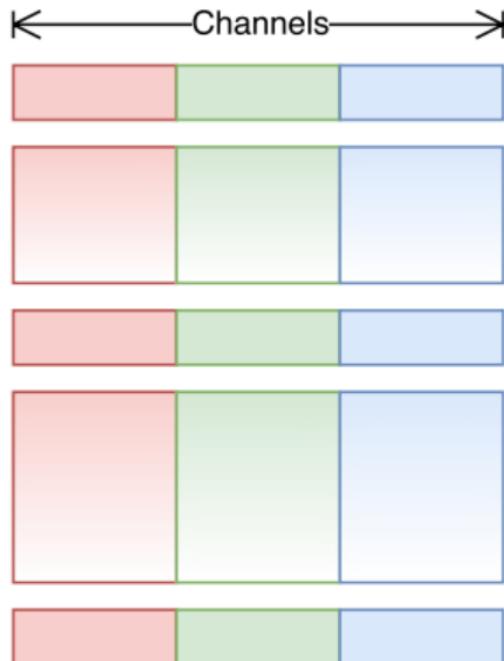


AlexNet 그룹 수에 따른 성능 변화

# Grouped Convolution

그룹 수를 늘리면 각 컨볼루션 레이어의 입력 채널 수가 줄어든다.

∴ 과도하게 그룹을 나누면 각 레이어가 충분한 수의 채널을 입력으로 가질 수 없게 된다.

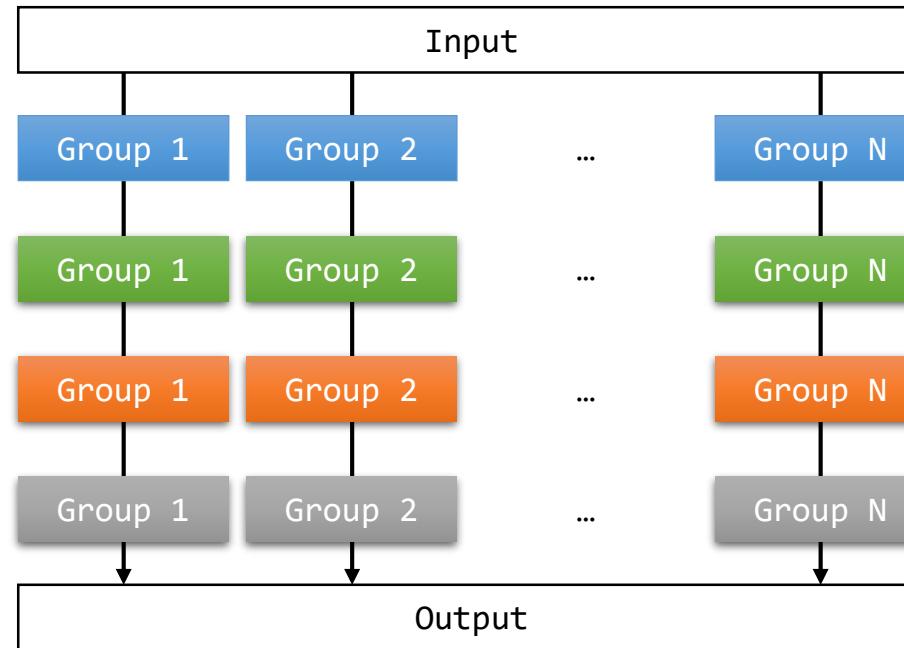


# Grouped Convolution

그룹 컨볼루션의 적층되어 반복되는 구조를 생각해보자.

각 Path가 ‘같은 구조를 가진’ 독립적인 Network들 처럼 학습된다.

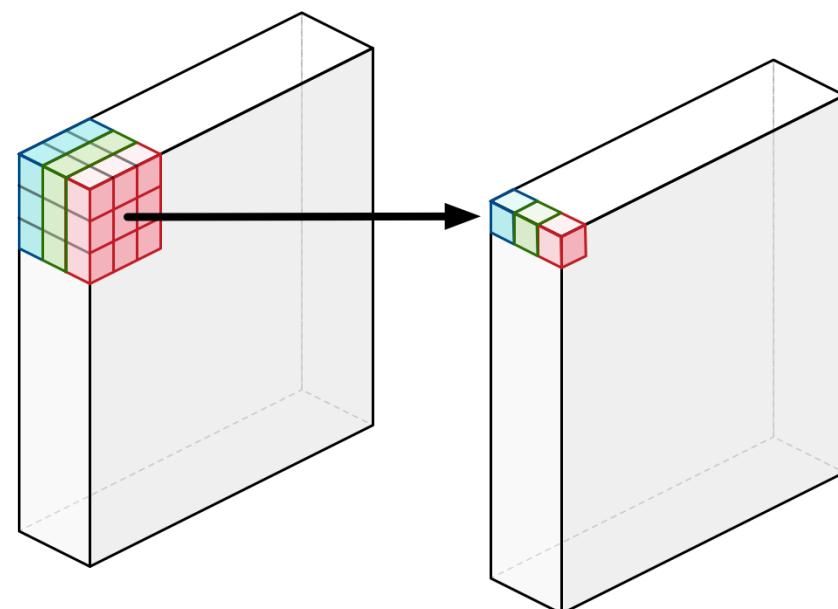
- 한 번에 여러 네트워크를 학습하는 꼴이 되며
- 서로 완전히 다른 특성을 가지는 망이 될 것이라는 보장도 없다.



# Depth-wise Convolution

---

Channel-independent Convolution



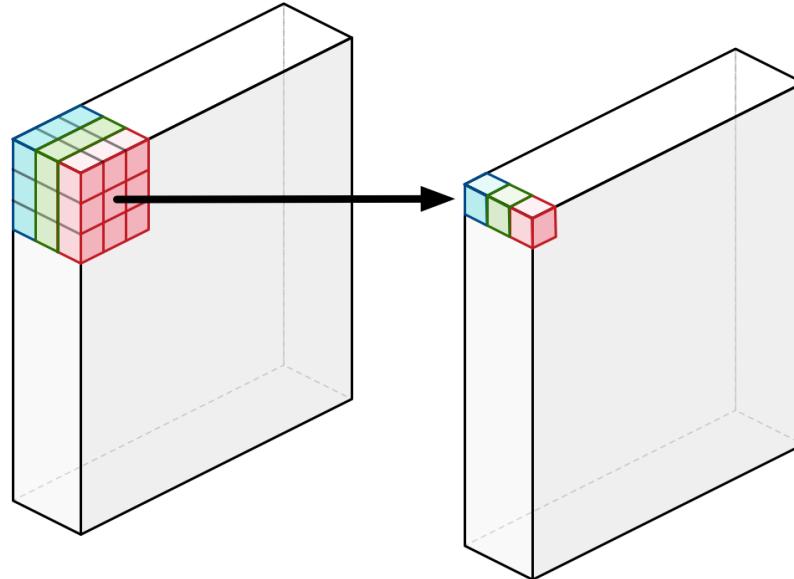
# Depth-wise Convolution

일반적인 Convolution Filter는 입력 영상의 모든 채널의 영향을 받게 되므로, 완벽히 특정 채널만의 Spatial Feature를 추출하는 것이 불가능하다.

- 다른 채널의 정보가 관여하는 것이 불가피하므로

Depth-wise Convolution은 각 단일 채널에 대해서만 수행되는 필터들을 사용한다.

- 그렇기에 필터 수는 입력 채널의 수와 같다.

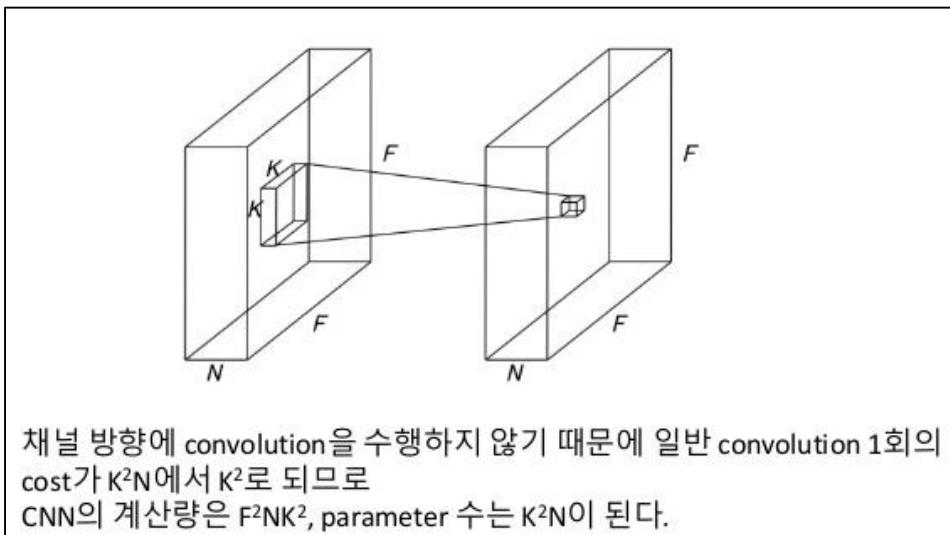


# Depth-wise Convolution

각 커널들은 하나의 채널에 대해서만 파라미터를 가진다.

- 입력-출력 채널의 수가 동일하다.
- 각 채널 고유의 Spatial 정보만을 사용하여 필터를 학습한다.

결과적으로 입력 채널 수 만큼 그룹을 나눈 Grouped Convolution과 같다.

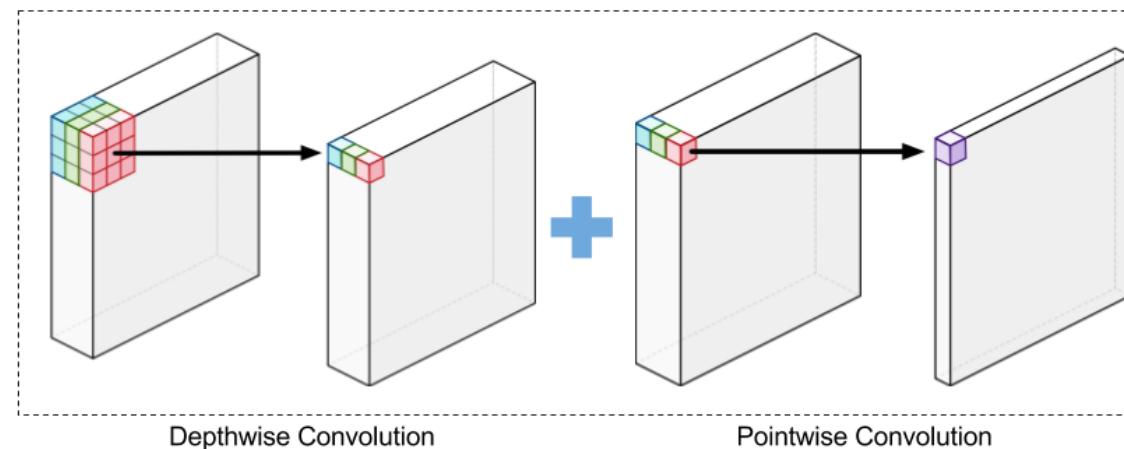


채널 방향에 convolution을 수행하지 않기 때문에 일반 convolution 1회의 cost가  $K^2N$ 에서  $K^2$ 로 되므로  
CNN의 계산량은  $F^2NK^2$ , parameter 수는  $K^2N$ 이 된다.

# Depth-wise Separable Convolution

---

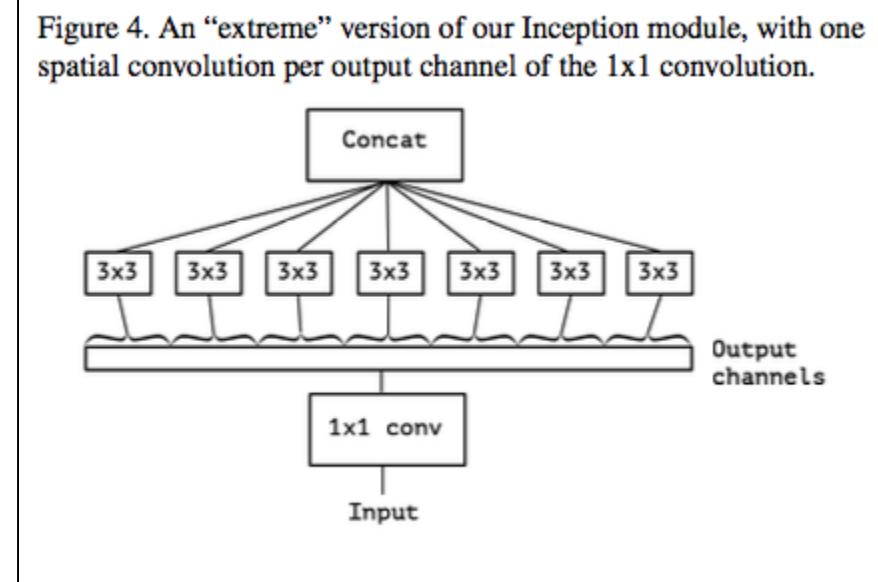
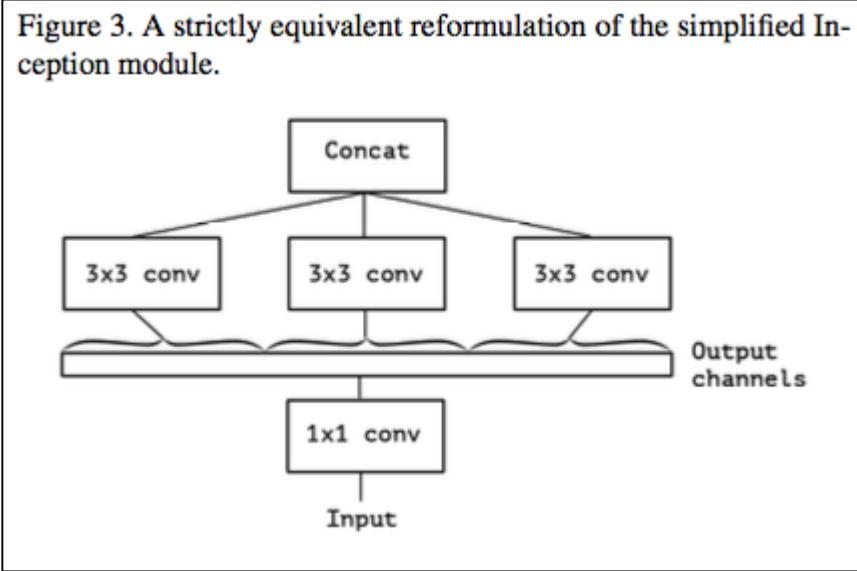
Separate 'Spatial Feature' and 'Cross channel Correlation'



# Idea in Xception(F. chollet et al)

기존의 Convolution에서

Cross-channel Correlation을 완벽히 분리하기 위한 구조 제안

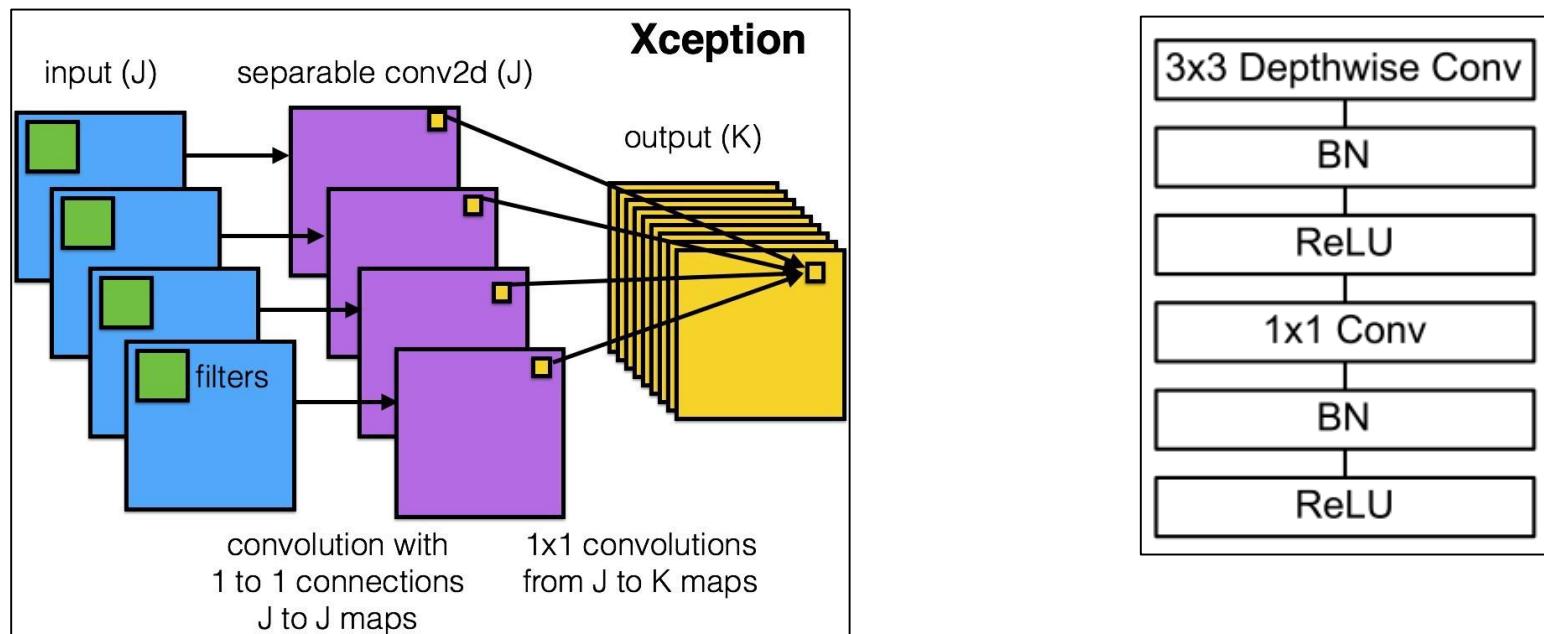


# Depth-wise '*Separable*' Convolution

Depth-wise Convolution과 Point-wise Convolution를 조합해 사용하는 방식

- 기존의 Convolution과 거의 유사하게 동작하지만 파라미터 수와 연산량은 훨씬 적다.

채널 별 Spatial Feature와 Channel Combination을 완벽히 분리하기 위한 구조



<https://www.kdnuggets.com/2017/08/intuitive-guide-deep-network-architectures.html/2>

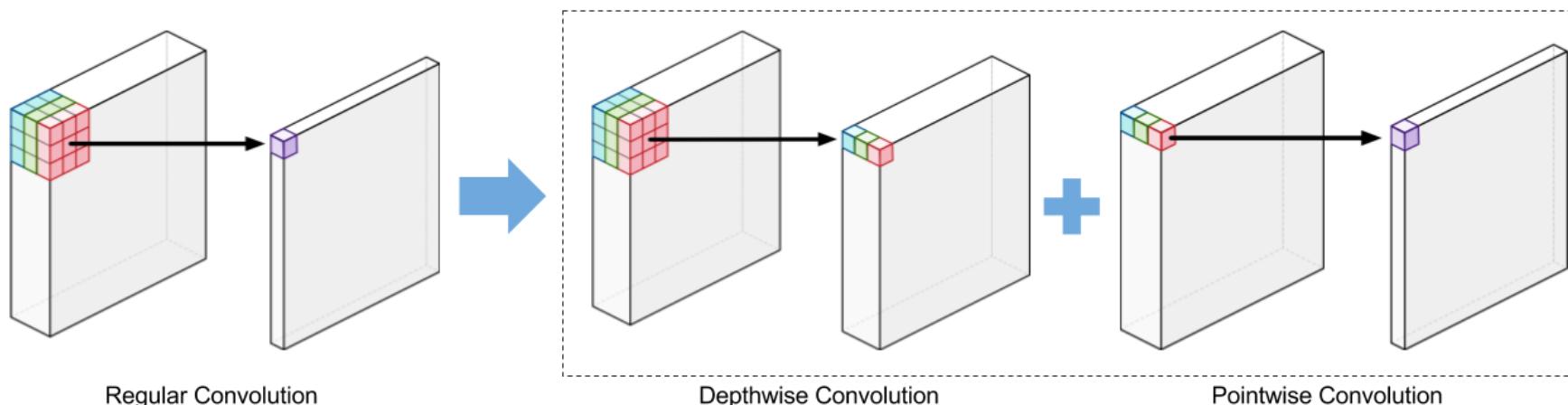
# Depth-wise '*Separable*' Convolution

## Original Convolution

- 전체 채널에 대한 Spatial Convolution

## Depth-wise Separable Convolution

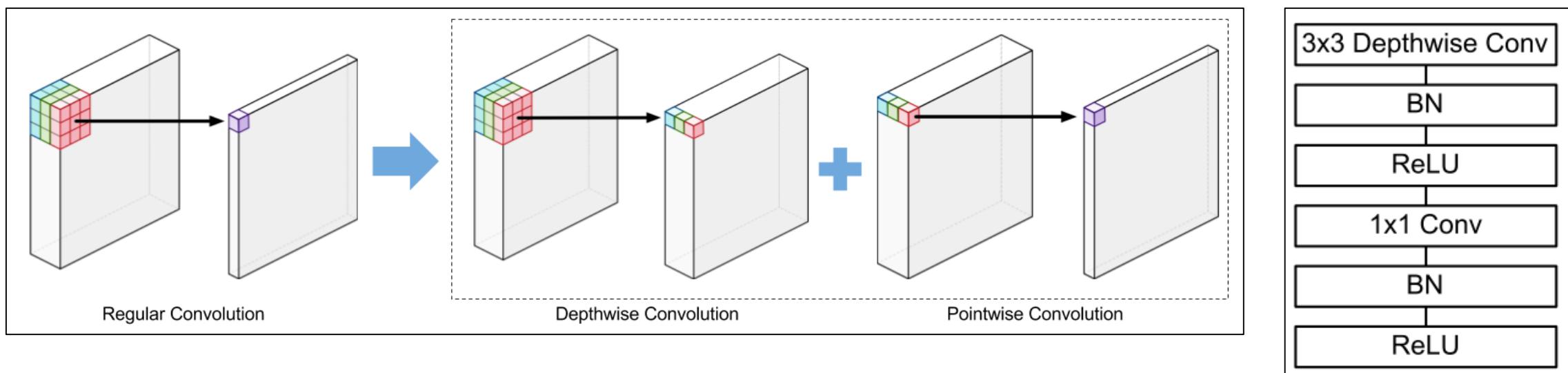
- 각 채널 별 Spatial Convolution 이후 Feature별 Linear Combination



# Depth-wise '*Separable*' Convolution

Spatial Feature과 Channel-wise Feature를 모두 고려하는 경량화된 모델 제안

	Baseline (Original Convolution)	Proposed (Depth-wise Separable Convolution)
# of params.	$N(CK^2 + 1)$ $= NCK^2 + N$	$C(K^2 + 1) + N(C + 1)$ $= CK^2 + NC + C + N$
# of operations.	$CHWK^2N$	$CHWK^2 + CHWN$ $= CHW(K^2 + N)$



# Experiment Results of Xception (F. chollet et al)

Spatial Feature과 Channel-wise Feature를 모두 고려하는 경량화된 모델 제안

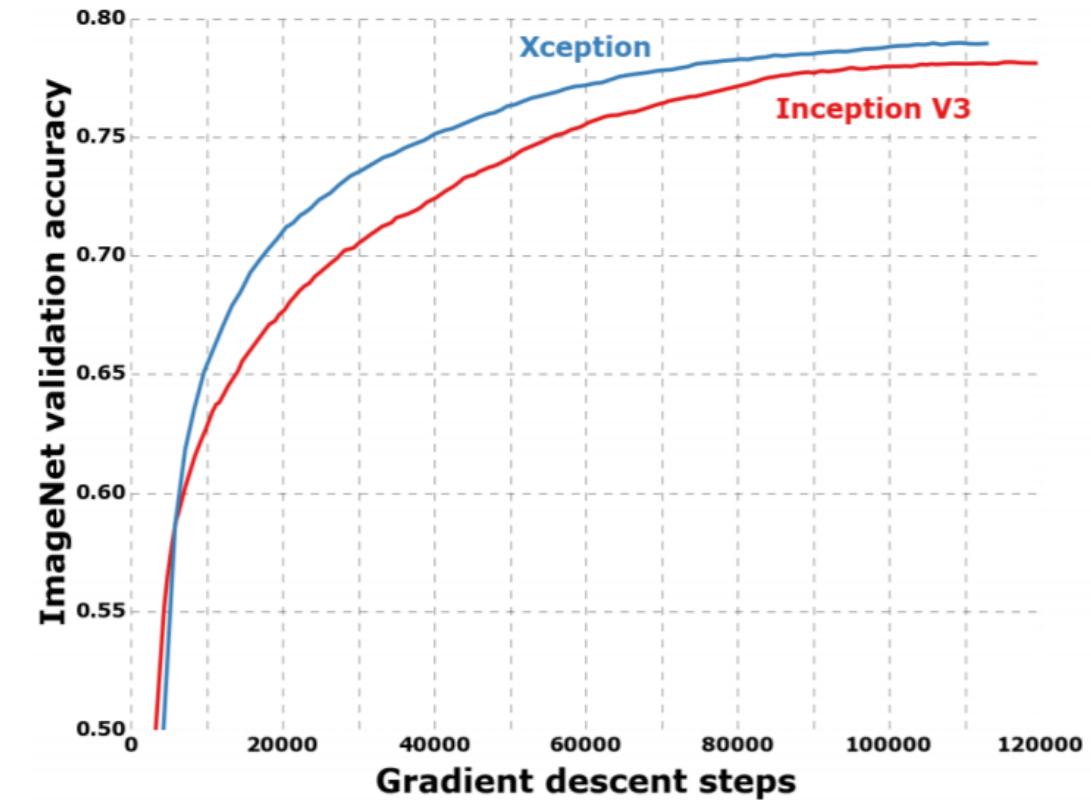
Table 1. Classification performance comparison on ImageNet (single crop, single model). VGG-16 and ResNet-152 numbers are only included as a reminder. The version of Inception V3 being benchmarked does not include the auxiliary tower.

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	<b>0.790</b>	<b>0.945</b>

Table 3. Size and training speed comparison.

	Parameter count	Steps/second
Inception V3	23,626,728	31
Xception	22,855,952	28

Figure 6. Training profile on ImageNet



# Related Papers

---

Squeeze Net

Mobile Net

Shuffle Net

ClcNet

# Squeeze Net (F. N. Iandola et al)

다음 세 가지 아이디어를 사용한 네트워크 설계법 제안

1. 더 큰 Feature Map을 사용하는 것이 성능 향상에 유리하다
  1. Sub-sampling을 최대한 후반부에 배치하여 Feature Map의 크기를 유지
2. 너무 많은 채널은 컨볼루션 연산의 비용을 증가시킨다
  1. Point-wise Convolution을 사용해 Channel Reduction
3. 커널 크기가 작은 컨볼루션 레이어가 더 경제적이다
  1.  $3 \times 3$  컨볼루션 레이어를 가능한  $1 \times 1$  컨볼루션으로 대체하자

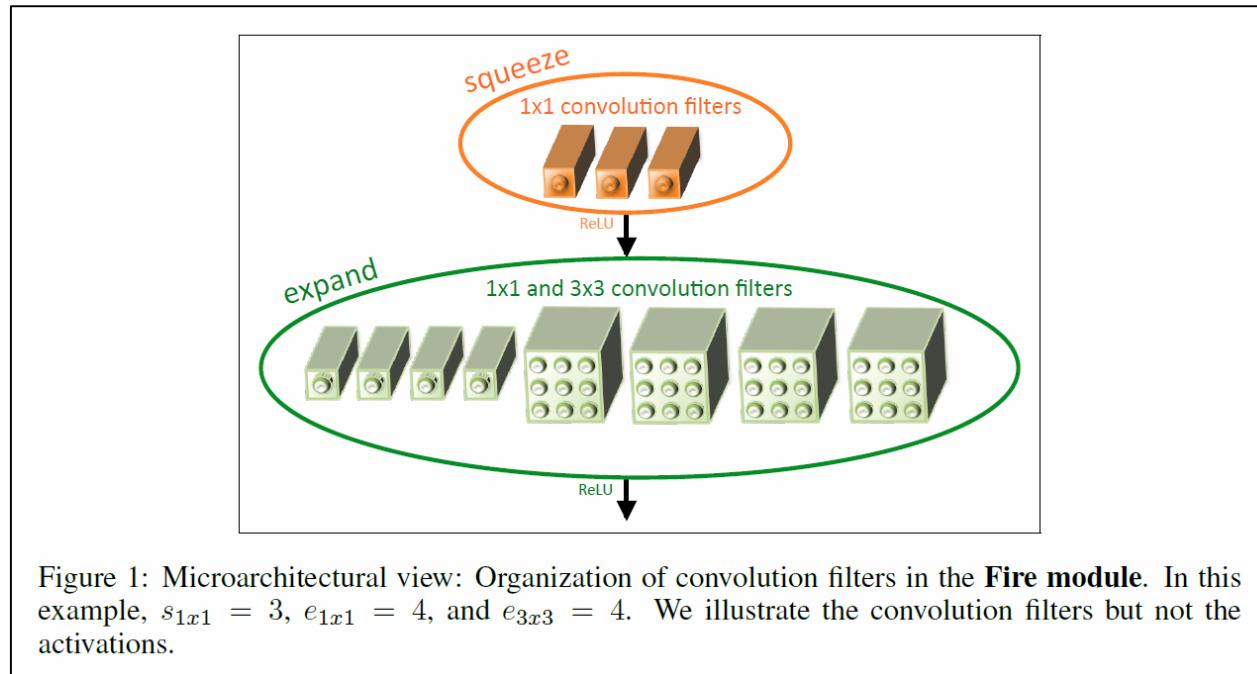
# Fire Module

## 1. Squeeze Layer

Point-wise Convolution을 사용한 Channel Reduction

## 2. Expand Layer

일부 3x3 Convolution Layer를 Point-wise Convolution으로 대체하여 계산 비용 감소



# Network Architecture

Fire Module의 단순 적층 구조 네트워크로 실험 진행

- Residual Connection을 사용한 성능 개선 실험
- 추가 Model Compression 기법 적용 실험

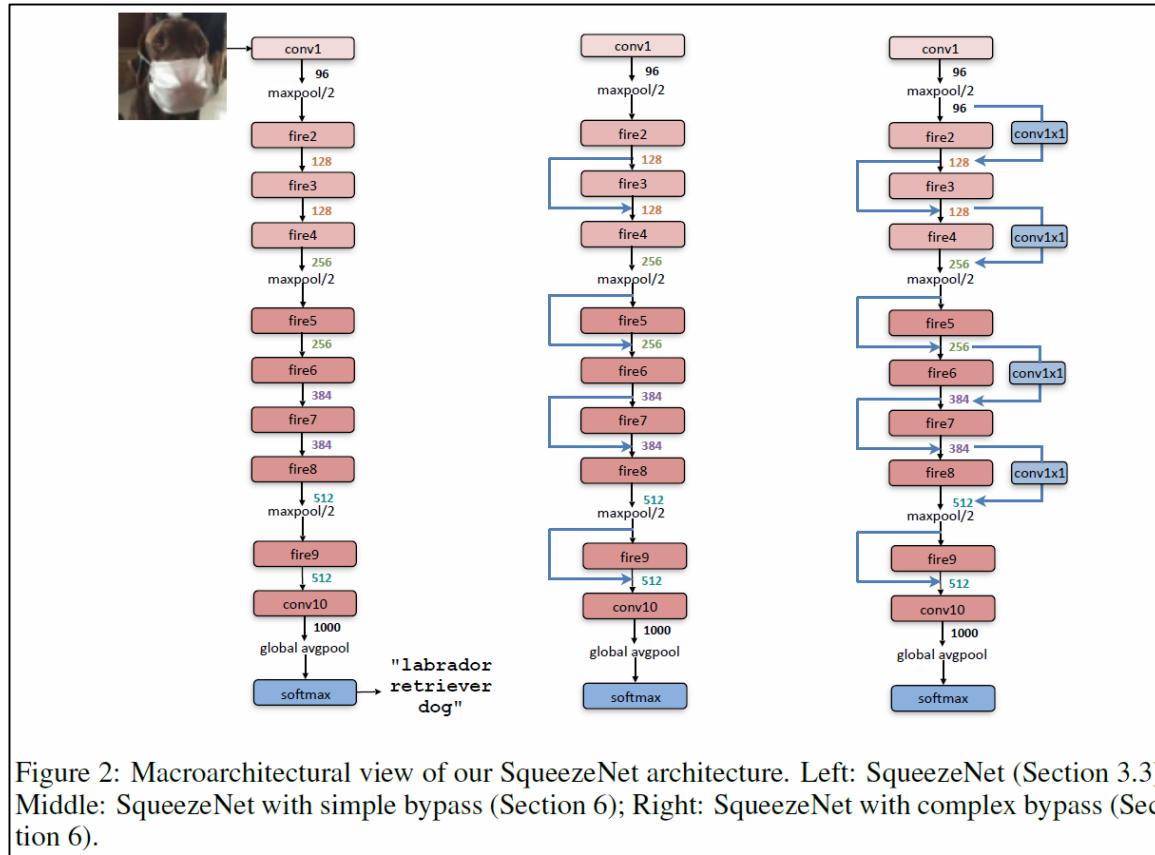


Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

# Experiment results of Squeeze Net

베이스라인(AlexNet)과 비교한 실험 결과

- 같은 수준의 Accuracy를 유지하면서 더욱 경량화 된 네트워크 설계 가능
- 추가적인 Compression기법으로 모델을 더욱 경량화 할 수 있다.

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	<b>50x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	<b>363x</b>	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	<b>510x</b>	57.5%	80.3%

# Experiment results of Squeeze Net

Squeeze Net 구조에서도 Residual Connection을 사용한 Bypass가 성능을 향상시킨다

Table 3: SqueezeNet accuracy and model size using different macroarchitecture configurations

Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	60.4%	82.5%	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

# Mobile Net (A. G. Howard et al)

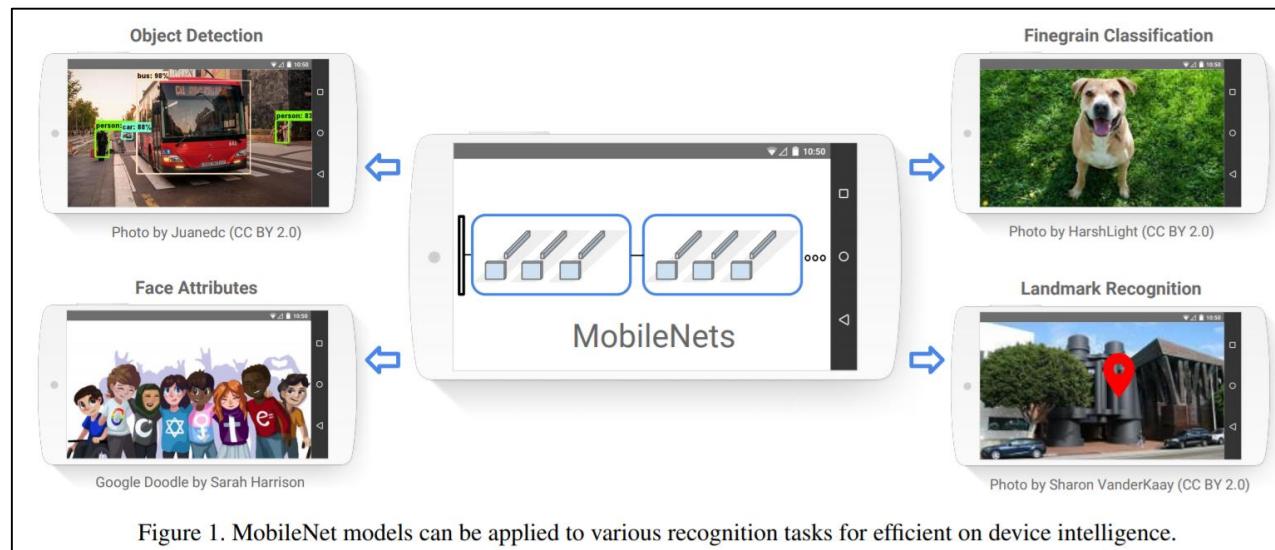
## 1. Point-wise Convolution의 적극 활용

Scalar-Matrix Multiplication으로 계산되기에 GEMM을 사용한 모델 가속화 용이

## 2. Depth-wise Separable Convolution의 사용

일반적인 Convolution Layer에 비해 좋은 계산/메모리 효율

## 3. Resolution과 Channel Size 보정을 통한 모델 경량화



# Mobile Net (A. G. Howard et al)

Inference 시점에서 Point-wise Convolution의 계산은 Matrix-Scalar Multiplication과 동일하다.

- 행렬 연산 최적화 기법을 적용하여 실제 연산성능 개선이 가능하다.

$$w_1 \times \begin{matrix} \text{blue cube} \\ + w_2 \times \end{matrix} \begin{matrix} \text{green cube} \\ + w_3 \times \end{matrix} \begin{matrix} \text{orange cube} \\ \dots + w_c \times \end{matrix} \begin{matrix} \text{yellow cube} \\ = \end{matrix} \begin{matrix} \text{gray cube} \end{matrix}$$

# Mobile Net: Network Architecture

첫 번째 레이어를 제외하고는 모두 Depth-wise Separable Convolution을 적용해 설계

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

# Mobile Net: Network Architecture

## Depthwise Separable Convolution 사용

- 일반적 Convolution에 비해 연산량과 파라미터 수가 훨씬 적다.

대부분의 리소스는 Point-wise Convolution에서 사용된다

- 다른 부분에서 사용되는 것 보다 유리하다.

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv $1 \times 1$	94.86%	74.59%
Conv DW $3 \times 3$	3.06%	1.06%
Conv $3 \times 3$	1.19%	0.02%
Fully Connected	0.18%	24.33%

# Mobile Net: Width Multiplier / Resolution Multiplier

추가적인 모델 경량화를 위한 두 가지 Hyper Parameter 설정

## 1. Width Multiplier $\alpha$

각 레이어의 입출력 채널의 수를 감소시키는 파라미터

## 2. Resolution Multiplier $\rho$

각 레이어의 입력 해상도를 감소시키는 파라미터

이런 강제적인 축소에도 어느정도 강인함을 보인다.

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

# Mobile Net: Experiment Result

유명 모델들과 비교해 다양한 데이터셋에서 좋은 경량화 효과를 보여준다.

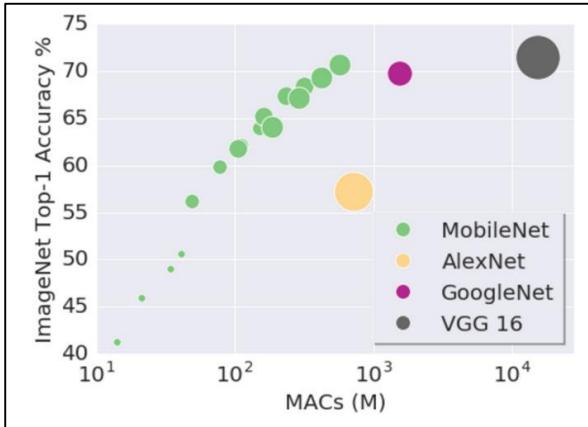


Table 8. MobileNet Comparison to Popular Models			
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models			
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
SqueezeNet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

Scale	Im2GPS [7]	PlaNet [35]	PlaNet MobileNet
Continent (2500 km)	51.9%	77.6%	79.3%
Country (750 km)	35.4%	64.0%	60.3%
Region (200 km)	32.1%	51.1%	45.2%
City (25 km)	21.9%	31.7%	31.7%
Street (1 km)	2.5%	11.0%	11.4%

# Shuffle Net (X. Zhang et al)

Mobile Net이 여전히 가지는 문제점을 해결하기 위한 아이디어 제안

## 1. Dense Connections between channels

- 채널 수가 늘어갈 수록 Cost가 증가
- Sparse Correlation Matrix Problem

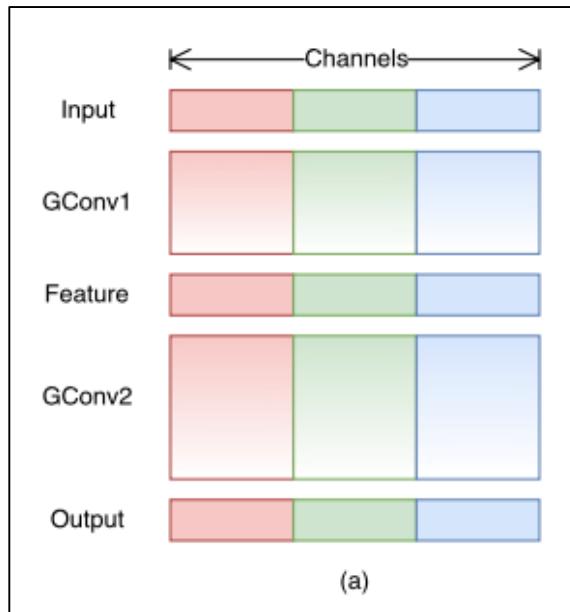
Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv $1 \times 1$	94.86%	74.59%
Conv DW $3 \times 3$	3.06%	1.06%
Conv $3 \times 3$	1.19%	0.02%
Fully Connected	0.18%	24.33%

# Shuffle Net (X. Zhang et al)

많은 채널로 인한 문제?

그냥 Grouped Convolution을 적용하면 될까?

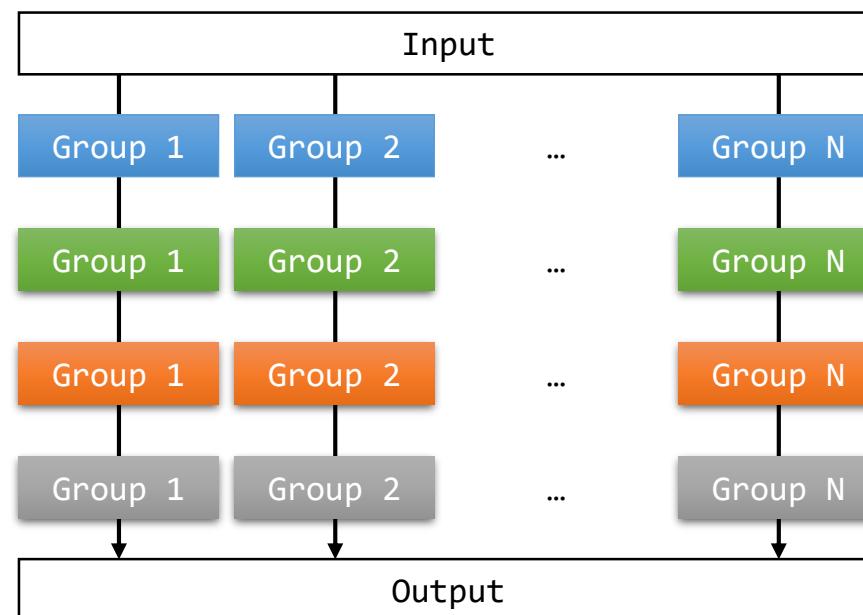
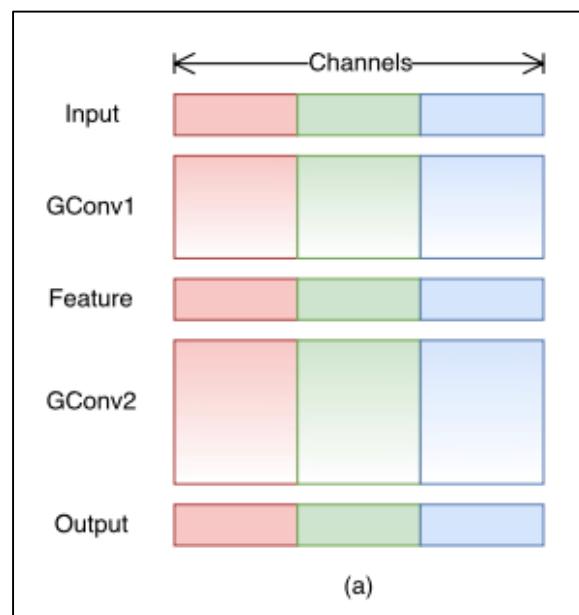


# Shuffle Net (X. Zhang et al)

## Grouped Convolution의 문제점

Grouped Convolution을 적층 할 시 여러 독립적인 네트워크처럼 경로가 분할  
하나의 네트워크가 Ensemble처럼 동작해버리는 문제 발생

즉, Channel Receptive Field가 너무 협소해지는 문제가 있다.



# Shuffle Net: Channel Shuffle

Grouped Convolution을 반복하는 사이에 그룹간 일부 채널을 Shuffle.

- 간단한 구현으로 그룹간 Independence 해결
- Grouped Convolution으로 Sparse Correlation Matrix 문제 해결

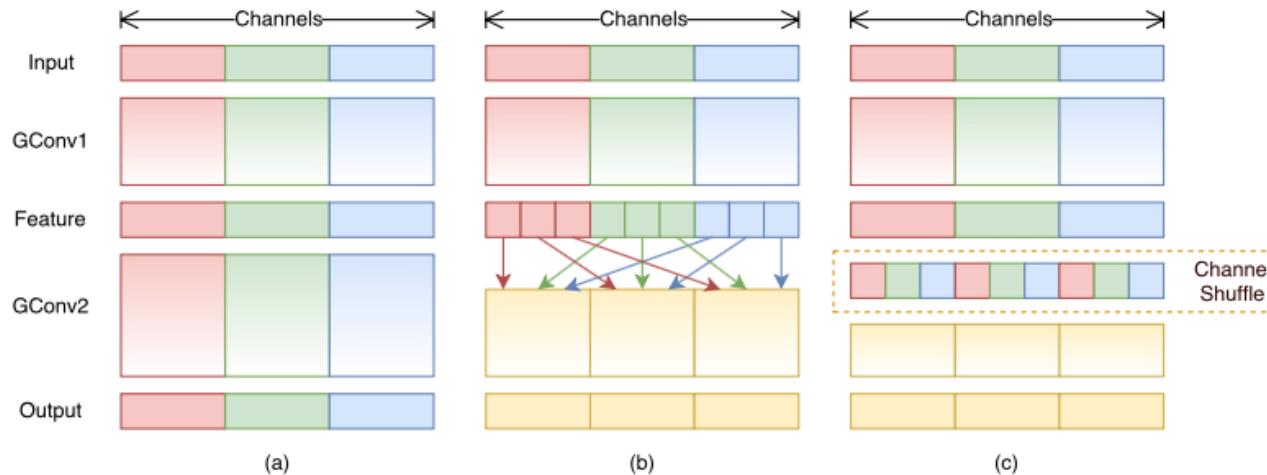


Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution.  
a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

# Shuffle Net: Shuffle-Net Unit

결과적으로 다음과 같은 Shuffle-Net Unit의 적층을 통한 네트워크 구현

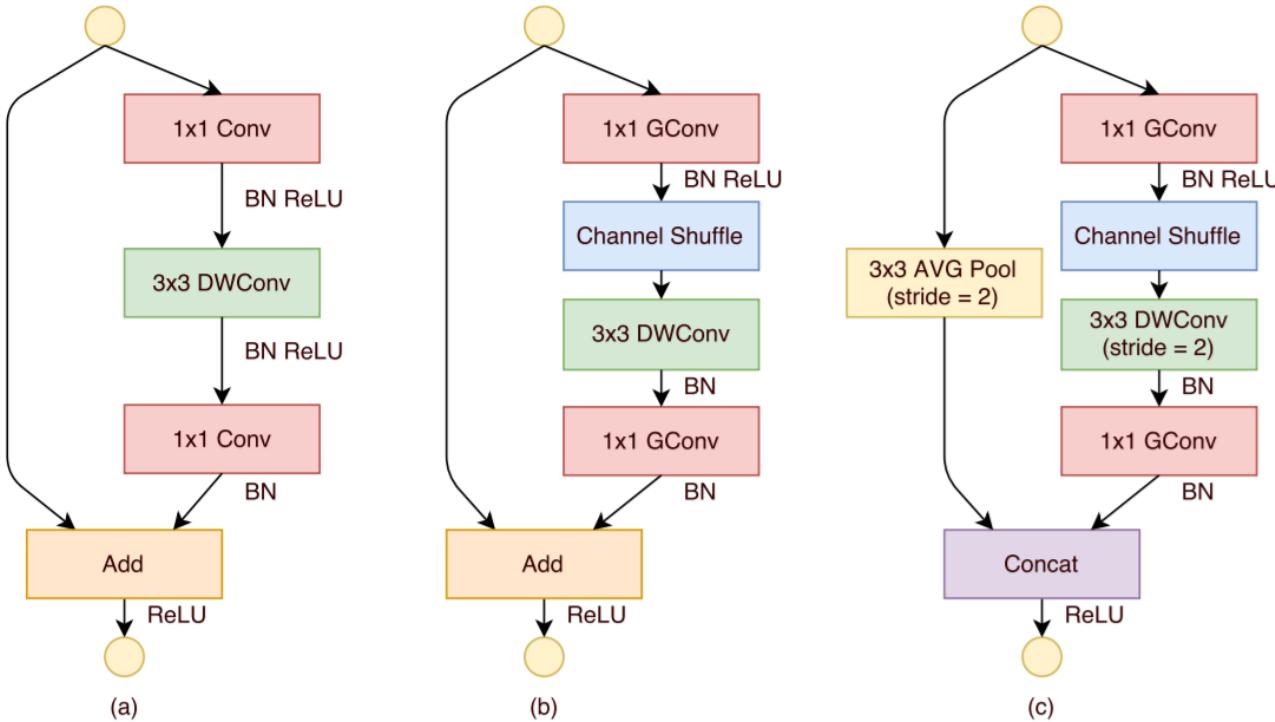


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

# Shuffle Net: Network Architecture

네트워크 상세 설계와 그룹별 파라미터 사이즈

- 그룹 수를 늘리면 더 많은 채널 사이즈를 확보할 수 있다.

Layer	Output size	KSize	Stride	Repeat	Output channels ( $g$ groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	$224 \times 224$				3	3	3	3	3
Conv1	$112 \times 112$	$3 \times 3$	2	1	24	24	24	24	24
MaxPool	$56 \times 56$	$3 \times 3$	2						
Stage2	$28 \times 28$		2	1	144	200	240	272	384
	$28 \times 28$		1	3	144	200	240	272	384
Stage3	$14 \times 14$		2	1	288	400	480	544	768
	$14 \times 14$		1	7	288	400	480	544	768
Stage4	$7 \times 7$		2	1	576	800	960	1088	1536
	$7 \times 7$		1	3	576	800	960	1088	1536
GlobalPool	$1 \times 1$	$7 \times 7$							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

Table 1. ShuffleNet architecture. The complexity is evaluated with FLOPs, i.e. the number of floating-point multiplication-adds. Note that for Stage 2, we do not apply group convolution on the first pointwise layer because the number of input channels is relatively small.

# Shuffle Net: Network Architecture

그룹 크기와 채널 사이즈 경량화에 따른 실험 결과

Model	Complexity (MFLOPs)	Classification error (%)				
		$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
ShuffleNet 1×	140	33.6	32.7	32.6	32.8	<b>32.4</b>
ShuffleNet 0.5×	38	45.1	44.4	43.2	<b>41.6</b>	42.3
ShuffleNet 0.25×	13	57.1	56.8	55.0	54.2	<b>52.7</b>

Table 2. Classification error vs. number of groups  $g$  (*smaller number represents better performance*)

Model	Cls err. (%), no shuffle	Cls err. (%), shuffle	$\Delta$ err. (%)
ShuffleNet 1x ( $g = 3$ )	34.5	<b>32.6</b>	1.9
ShuffleNet 1x ( $g = 8$ )	37.6	<b>32.4</b>	5.2
ShuffleNet 0.5x ( $g = 3$ )	45.7	<b>43.2</b>	2.5
ShuffleNet 0.5x ( $g = 8$ )	48.1	<b>42.3</b>	5.8
ShuffleNet 0.25x ( $g = 3$ )	56.3	<b>55.0</b>	1.3
ShuffleNet 0.25x ( $g = 8$ )	56.5	<b>52.7</b>	3.8

Table 3. ShuffleNet with/without channel shuffle (*smaller number represents better performance*)

# Shuffle Net: Network Architecture

그룹 크기와 채널 사이즈 경량화에 따른 실험 결과

Complexity (MFLOPs)	VGG-like	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	50.7	37.3	33.6	33.3	<b>32.4</b> ( $1\times, g = 8$ )
38	-	48.8	45.1	46.0	<b>41.6</b> ( $0.5\times, g = 4$ )
13	-	63.7	57.1	65.2	<b>52.7</b> ( $0.25\times, g = 8$ )

Table 4. Classification error vs. various structures (%), smaller number represents better performance). We do not report VGG-like structure on smaller networks because the accuracy is significantly worse.

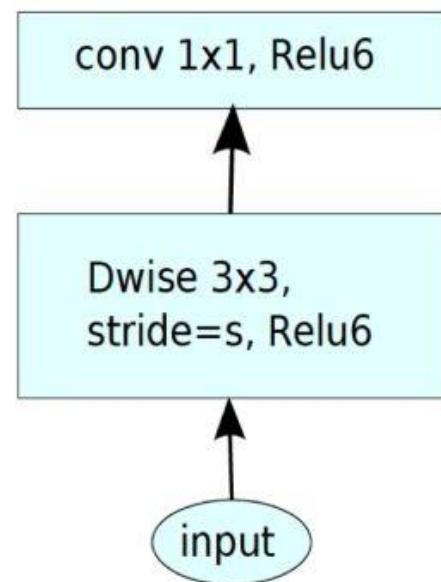
Model	Complexity (MFLOPs)	Cls err. (%)	$\Delta$ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times$ ( $g = 3$ )	524	<b>26.3</b>	3.1
ShuffleNet $2\times$ (with SE[13], $g = 3$ )	527	<b>24.7</b>	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times$ ( $g = 3$ )	292	<b>28.5</b>	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times$ ( $g = 8$ )	140	<b>32.4</b>	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times$ ( $g = 4$ )	38	<b>41.6</b>	7.8
ShuffleNet $0.5\times$ (shallow, $g = 3$ )	40	42.8	6.6

Table 5. ShuffleNet vs. MobileNet [12] on ImageNet Classification

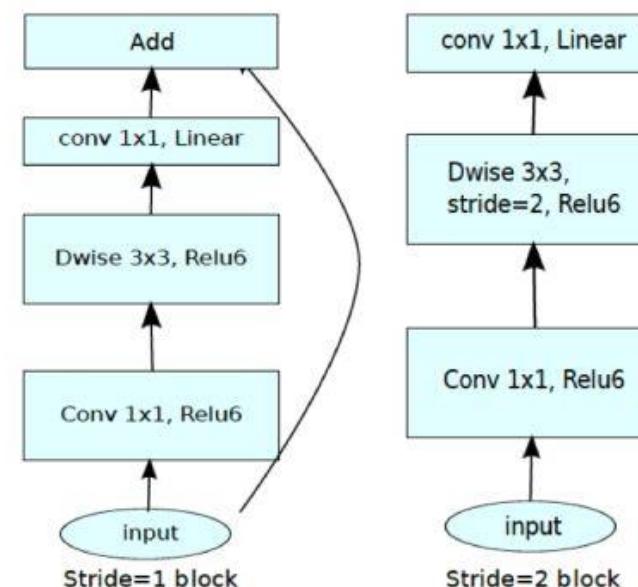
# Mobile Net V2

기존의 Mobile Net에서 두 가지 테크닉을 사용해 모델 개선

- Inverted Residual Connection
- Linear Bottleneck



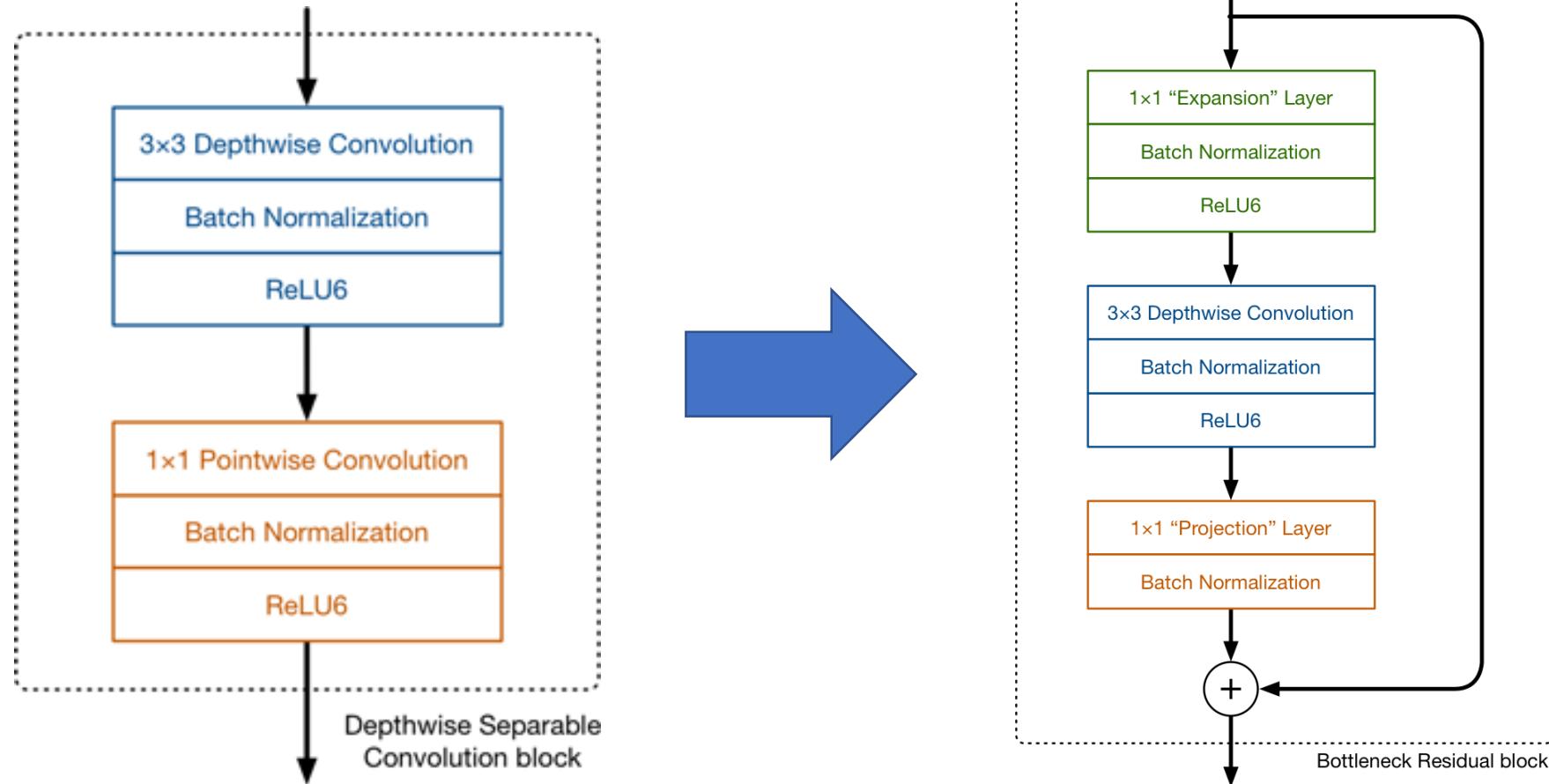
(b) MobileNet[26]



(d) Mobilenet V2

# What's different?

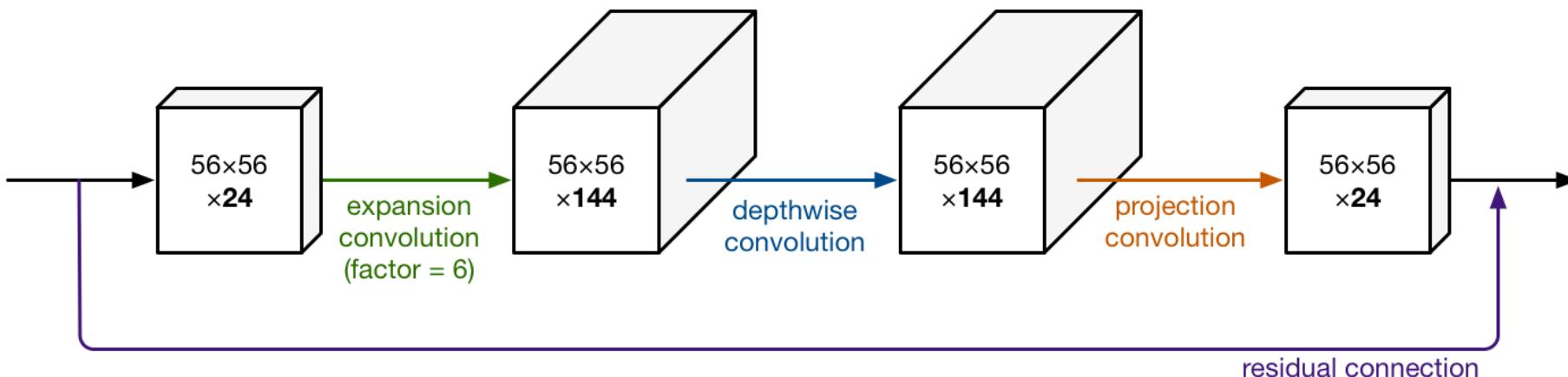
- Residual Connection
- Bottleneck With Expansion Layer



# Expansion Layer?

채널을 확장하는 Bottleneck구조?

- 일반 Convolution은 채널의 수에 비례해 코스트가 급격히 증가하기 때문에 Bottleneck구조를 사용해 채널 Reduction을 수행한다.
- 하지만 DWS Convolution은 채널의 수에 크게 영향을 받지 않는다. 그러므로 입력단에서 Channel의 수를 증가시켜준다.



# Expansion Layer?

DWS Convolution은 각 입력 채널에 대해 독립적인 Spatial Feature를 학습한다.

- 즉, Spatial Pattern의 수가 다양하지 않다

그러므로 채널의 수를 증가시켜주면 Spatial Representation을 향상시킬 수 있다.

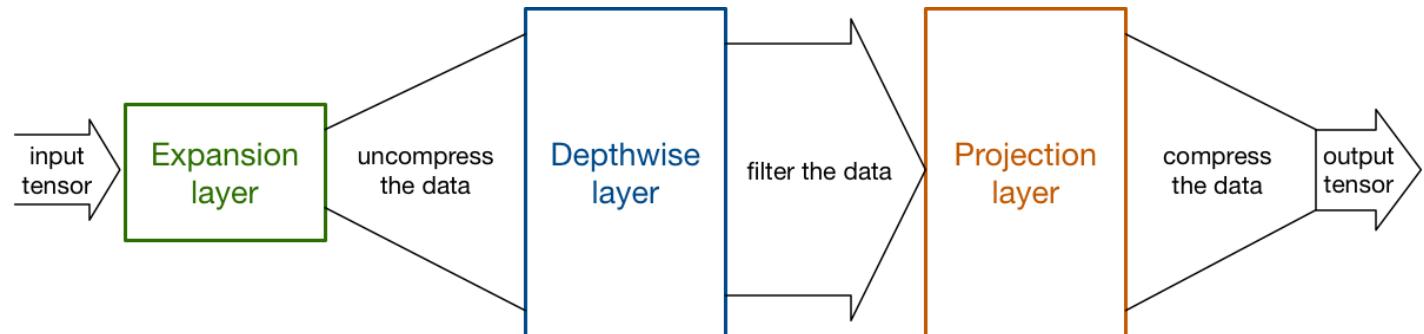
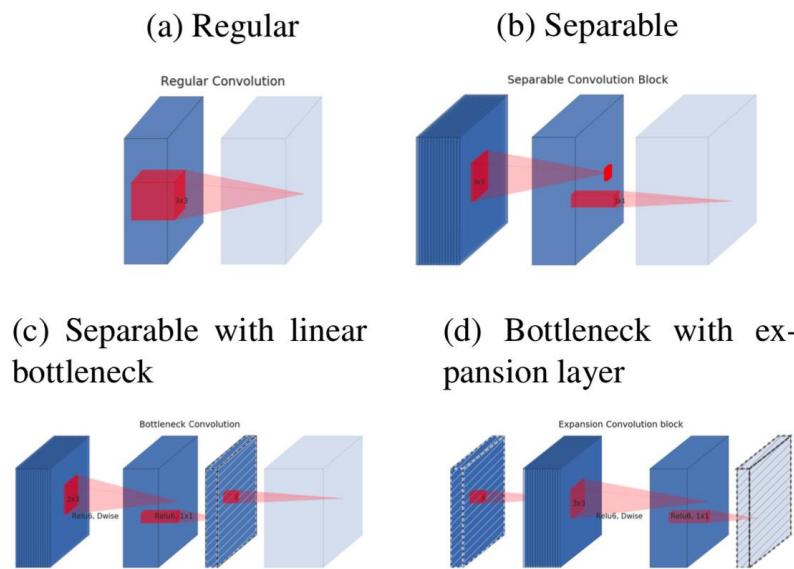
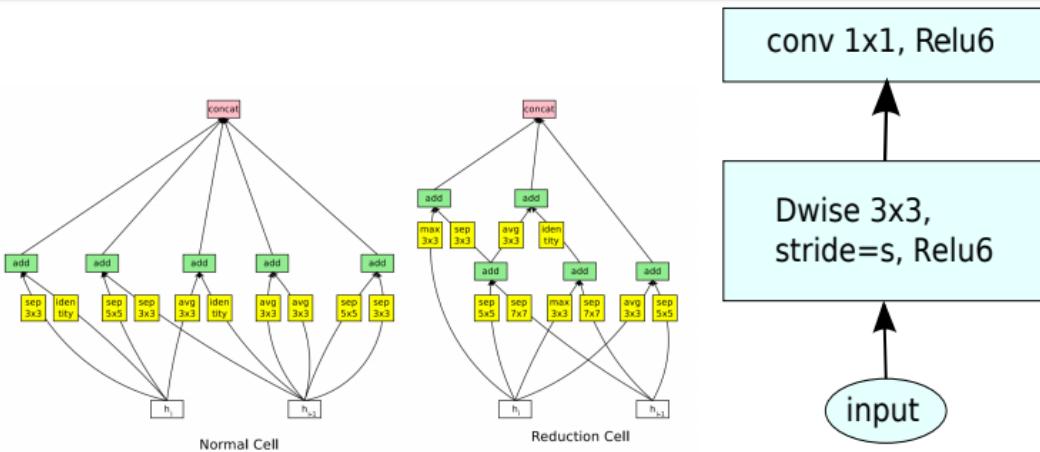


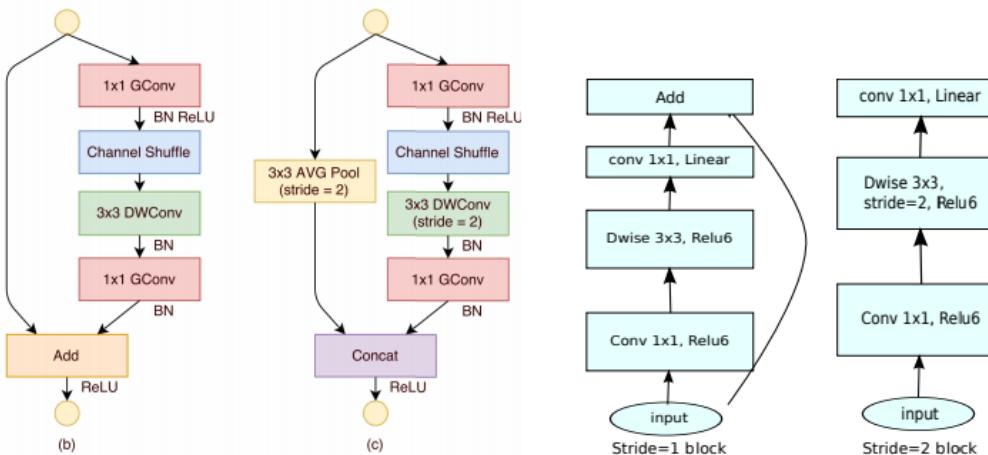
Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

# Architecture Comparison



(a) NasNet[23]

(b) MobileNet[27]



(c) ShuffleNet [20]

(d) Mobilenet V2

# Architecture Comparison (V1 - V2)

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Input	Operator	$t$	$c$	$n$	$s$
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated  $n$  times. All layers in the same sequence have the same number  $c$  of output channels. The first layer of each sequence has a stride  $s$  and all others use stride 1. All spatial convolutions use  $3 \times 3$  kernels. The expansion factor  $t$  is always applied to the input size as described in Table 1.

# Experimental Results

기존의 Mobile Net 대비 더욱 좋은 경량화 및 성능 효율을 보여준다.

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	<b>3.4M</b>	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	<b>72.0</b>	<b>3.4M</b>	<b>300M</b>	<b>75ms</b>
MobileNetV2 (1.4)	<b>74.7</b>	6.9M	585M	<b>143ms</b>

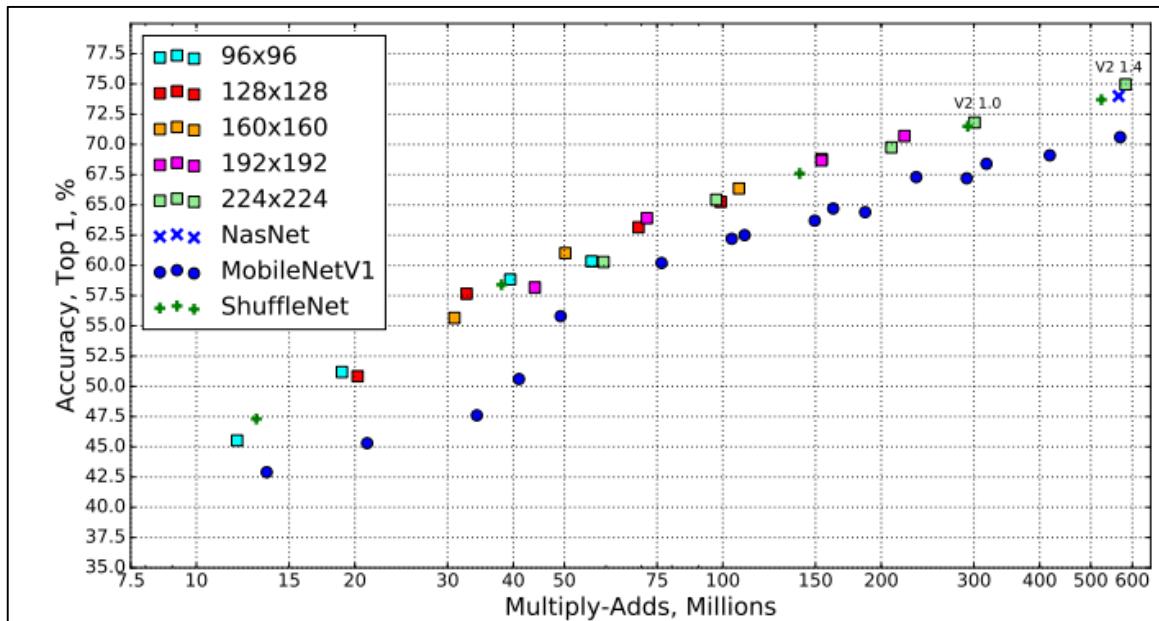


Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for 224. Best viewed in color.

Fin.

감사합니다.

\* 이 PPT는 네이버에서 제공한 나눔글꼴과 아모레퍼시픽의 아리따부리 폰트를 사용하고 있습니다.