



# Densely Connected Convolutional Networks(DenseNet)

## Abstract

### DenseNet 원리

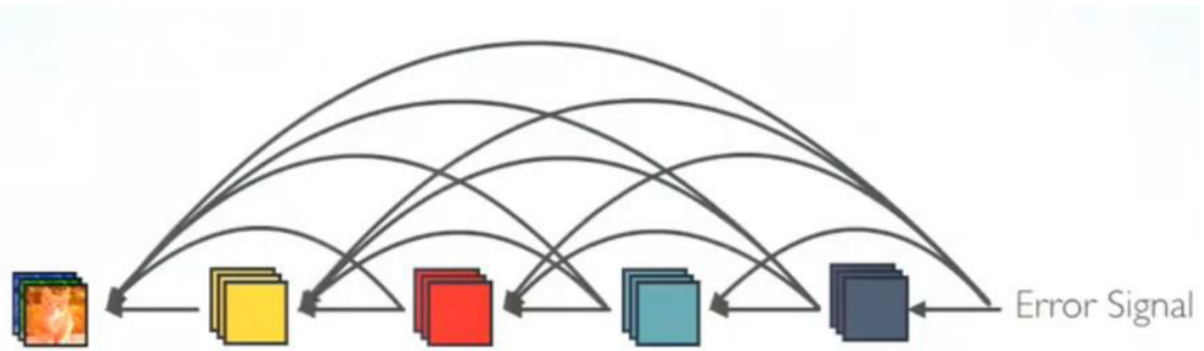
- **shortcut connections**을 통해 네트워크를 깊게 쌓고 정확도를 높임
- 위의 원리를 토대로 **layer간 최대한 많은 연결**을 시켜서 각 layer 간의 정보 흐름을 최대한 이용하자고 제안  $\Rightarrow$  DenseNet에서는 feed-forward 시, 각 layer를 **모든 다른 layer**와 연결시킴
- 기존의 convolution layer들이 L개의 layer들에 대해서 L 번의 connection  $\Rightarrow 1+2+\dots+L = L(L+1)/2$  번의 **direct connections** (Figure 1).
- 코드 : <https://github.com/liuzhuang13/DenseNet>

### ▼ Shortcut connection (ResNet)

- 레이어간의 연결이 순서대로 연속적인 것만 있는 것이 아니라, **중간을 뛰어넘어 전달하는(더하는) shortcut**이 추가된 것
- 연산은 매우 간단하고, 개념도 매우 간단하지만 이것이 gradient를 직접적으로 잘 전달하여 **gradient vanishing/exploding 문제를 해결**하는 큰 효과를 냄

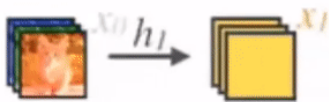
### DenseNet 장점

| 기울기 소실 문제(**gradient vanishing**) 완화



- 위 그림과 같이 DenseNet 또한 ResNet 처럼 **gradient**를 다양한 경로를 통해서 받을 수 있기 때문에 학습하는 데 도움이 됩니다.

## feature propagation 강화



- 위 그림을 보면 앞단에서 만들어진 feature를 그대로 뒤로 전달을 해서 **concatenation** 하는 방법을 사용을 합니다. 따라서 feature를 계속해서 끝단 까지 전달하는 데 장점이 있습니다.

## feature reuse → DenseNet에서 설명

## parameter 개수 줄임 → DenseNet에서 설명

## 실험 방법

- CIFAR-10/CIFAR-100/SVHN/ImageNet dataset으로 **benchmark tasks**

# Introduction

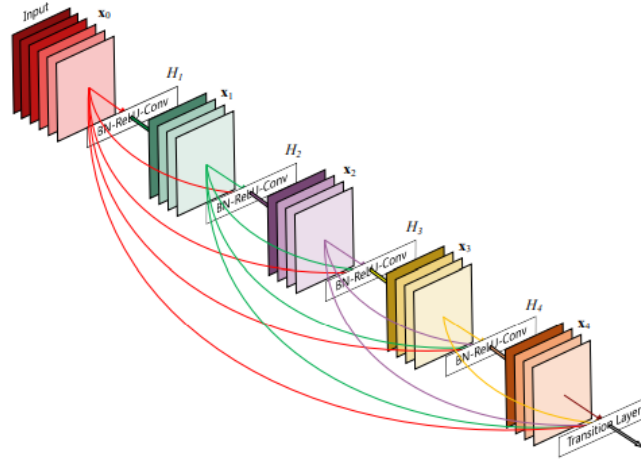
## 배경

- CNN (Convolutional Neural Networks)는 visual object recognition에 자주 사용되나, CNN의 네트워크가 깊어질수록(= input이나 gradient가 많은 layer를 거칠수록) 네트워크 끝 부분에서는 gradient가 소실 되는(vanishing) 문제 발생
- 이 문제를 해결하기 위해 **ResNet/Highway network/Stochastic depth/FractalNets** 등장
- **1 ResNet과 2 Highway Network**는 **identity connection**(자기 자신을 다시 feed시켜주는 방식)을 사용
- **3 Stochastic depth**는 Resnet의 **layer**를 **random**하게 **없애주어**(dropping layer) 크기를 줄임
- **4 Fractal Net**은 각기 다른 숫자의 convolutional block들로 이루어진 parallel layer들의 sequence를 여러 번 반복시켜 **short path**를 **유지**한 채 nominal depth(공칭두께)를 크게 하였다. (?)
- 문제를 해결하는 핵심 : **앞 쪽의 layer와 뒤 쪽의 layer를 short path로 연결**

## DenseNet

- 최대한의 정보 흐름을 보장하기 위해서, **모든 layer**를 각각 **직접 연결**
- **$L(L+1)/2$ 번의 direct connections**이 이루어진다. (Figure 1)

$$L + (L - 1) + (L - 2) + \dots + 1 = L(L + 1)/2$$



**Figure 1:** A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

## information preservation

- **ResNet**은 identity transformation을 더해서(summation) **later layer**로부터 **early layer**로의 **gradient flow**가 직접 연결된다는 장점이 있지만, **identity transformation**과 출력  $H(x-1)$ 이 summation됨에 따라 **information flow**를 방해할 수 있다.
  - gradient가 흐르게 된다는 점은 도움이 되지만, forward pass에서 보존되어야 하는 정보들이 **summation**을 통해 변경되어 보존되지 못할 수 있다는 의미이다.  
(DenseNet은 concatenation을 통해 그대로 보존)
- **DenseNet**은 feature map을 그대로 보존하면서, feature map의 일부를 layer에 **concatenation** → 네트워크에 더해질 **information**과 보존되어야 할 **information**을 분리해서 처리 → information 보존

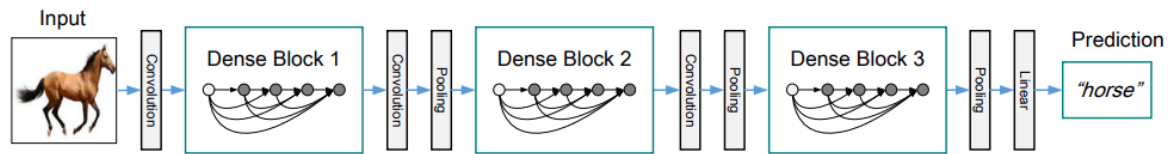
## improved flow of information and gradient

- 모든 **layer**가 이전의 다른 **layer**들과 직접적으로 연결되어 있기 때문에, loss function 이나 input signal의 gradient에 직접적으로 접근 가능 + **gradient vanishing**이 없어짐 → 네트워크가 깊은 구조를 만드는 것이 가능

## regularizing effect

- 많은 connection으로 **depth**가 짧아지는 효과 → **regularization** 효과 (overfitting 방지)
- 상대적으로 작은 train set을 이용하여도 **overfitting** 문제에서 자유로움

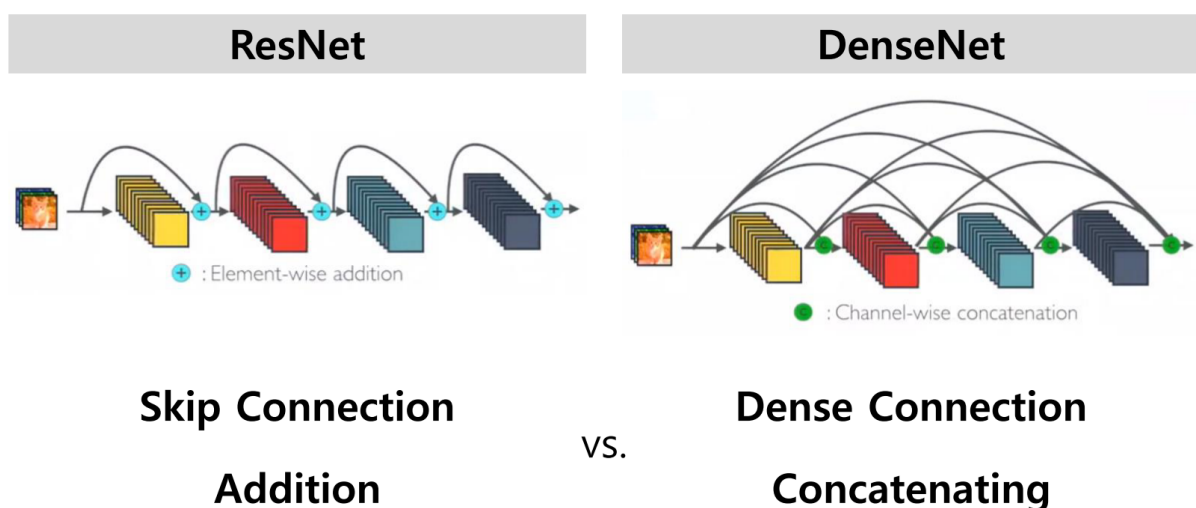
# DenseNets



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

## Dense Connectivity

### ResNet vs. DenseNet



$x_0$	네트워크로 들어오는 최초 입력 데이터
$L$	전체 레이어 수
$H_l(x)$	$x$ 가 입력으로 들어온 $l$ 번째 non-linear transformation 레이어 (일반적으로 Conv + BN + ReLU 조합으로 구성됨)
$x_l$	$l$ 번째 레이어의 출력 feature map


- **ResNet**은 gradient가 identity function을 통해 직접 earlier layer에서 later layer로 흐를 수 있으나, identity function과 output을 더하는(summation) 과정에서 information flow를 방해할 수 있음 → **L**번의 **connections**

$$\mathbf{x}_\ell = H_\ell(\mathbf{x}_{\ell-1}) + \mathbf{x}_{\ell-1}. \quad (1)$$

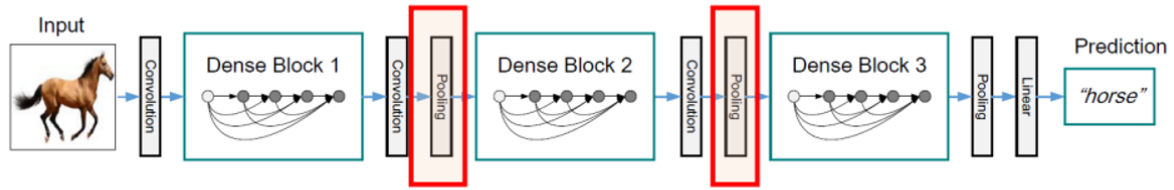
- **DenseNet**은 summation으로 layer 사이를 연결하는 대신에, concatenation으로 layer 사이를 직접 연결 →  $L(L+1)/2$ 번의 **connections** ⇒ **dense connectivity**라서 DenseNet(Dense Convolutional Network)으로 명명

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{\ell-1}]), \quad (2)$$

## Composite function

- $H_l()$ 은 합성함수로, 아래의 3개 연산이 결합된 구조
  - batch normalization (BN)
  - rectified linear unit (ReLU)
  - 3 x 3 convolution (Conv)
- ▼  Batch Normalization
  - **Batch** 단위로 학습을 하게 되면 발생하는 문제점 → **학습 과정에서 계층 별로 입력의 데이터 분포가 달라지는 현상**
  - ( $\because$  각 계층에서 입력으로 feature를 받게 되고 그 feature는 convolution이나 위와 같이 fully connected 연산을 거친 뒤 activation function을 적용 → 그러면 **연산 전/후에 데이터 간 분포가 달라질 수 있음**) → 이와 유사하게 Batch 단위로 학습을 하게 되면 **Batch 단위간에 데이터 분포의 차이**가 발생할 수 있음
  - 이 문제를 개선하기 위해 **Batch Normalization** 개념이 적용
  - **batch normalization** → 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 **각 배치별로 평균과 분산을 이용해 정규화**하는 것

## Pooling layers



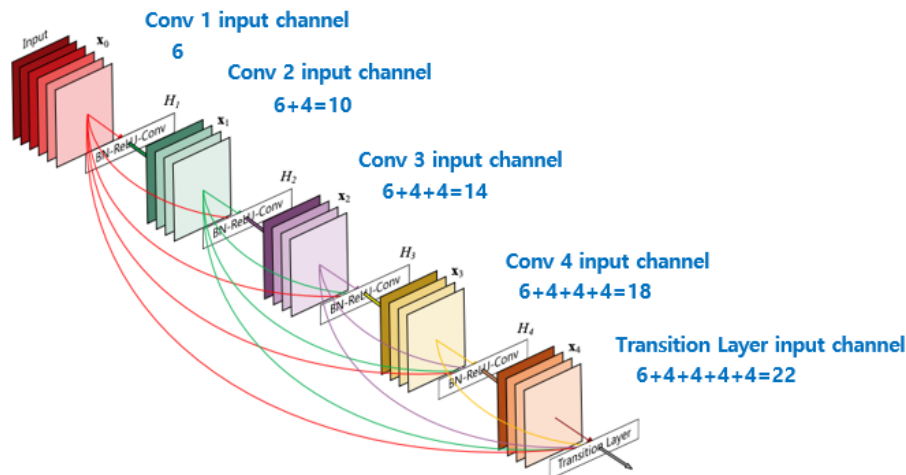
Huang, Gao, et al. "Densely connected convolutional networks." *arXiv preprint arXiv:1608.06993* (2016).

- feature map의 크기가 변경될 경우, **concatenation 연산**을 수행할 수 없음 ( $\because$  평행하게 합치는 것이 불가능)  $\leftrightarrow$  CNN은 **down-sampling**은 필수이므로, layer마다 feature map의 크기가 달라질 수 밖에 없음
- DenseNet은 네트워크 전체를 몇 개의 dense block으로 나눠서 **같은 feature map size를 가지는 레이어들은 같은 dense block내로 묶음**
- 위 그림에서는 총 3개의 dense block으로 나눔
  - **같은 블럭 내의 레이어들은 전부 같은 feature map size를 가짐**  $\Rightarrow$  concatenation 연산 가능
  - **transition layer**(빨간 네모를 친 pooling과 convolution 부분)  $\Rightarrow$  down-sampling 가능
    - Batch Normalization(BN)
    - $1 \times 1$  convolution  $\rightarrow$  feature map의 개수(= channel 개수)를 줄임
    - $2 \times 2$  average pooling  $\rightarrow$  feature map의 가로/세로 크기를 줄임
  - ex. dense block1에서  $100 \times 100$  size의 feature map을 가지고 있었다면 dense block2에서는  $50 \times 50$  size의 feature map
- 위 그림에서 **가장 처음에 사용되는 convolution 연산**  $\rightarrow$  input 이미지의 사이즈를 dense block에 맞게 조절하기 위한 용도로 사용됨  $\rightarrow$  이미지의 사이즈에 따라서 사용해도 되고 사용하지 않아도 됨

#### ▼ 📎 pooling layer

- **Pooling의 필요성 ?**
- CNN에는 많은 convolution layer를 쌓기 때문에 필터의 수가 많음  $\rightarrow$  필터가 많다면 그만큼 feature map들이 쌓이게 된다  $\Rightarrow$  CNN의 차원이 매우 크다
- 높은 차원을 다루려면 그 차원을 다룰 수 있는 많은 수의 parameter가 필요  $\rightarrow$  but, parameter가 너무 많아지면 학습 시 overfitting이 발생  $\rightarrow$  필터에 사용된 **parameter 수를 줄여서 차원을 감소시킬 방법**이 필요  $\Rightarrow$  **pooling** layer로 해결

## Growth rate



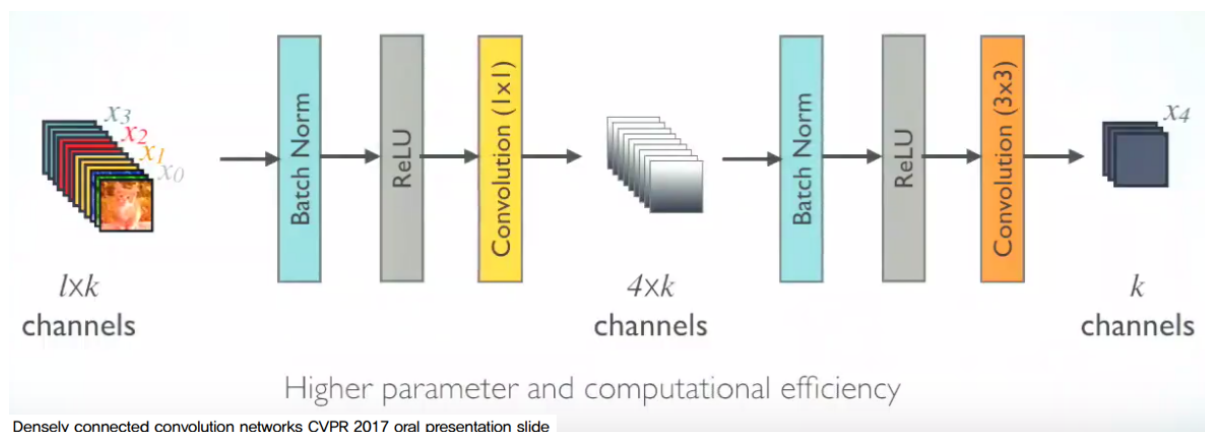
**Figure 1:** A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

- input의 채널 개수  $k_0$ 와 이전  $(l-1)$ 개의 layer  $\rightarrow H_l() \rightarrow$  output으로,  $k$  feature maps (단,  $k_0$  : input layer의 channel 개수)
  - input :  $k_0 + k * (l - 1)$
  - output :  $k$
- **Growth rate(= hyperparameter  $k$ )**  $\rightarrow$  각 layer의 feature map의 channel 개수
- 각 feature map끼리 densely connection 되는 구조이므로 자칫 feature map의 channel 개수가 많을 경우, 계속해서 channel-wise로 concatenate 되면서 channel 이 많아질 수 있음  $\Rightarrow$  DenseNet에서는 각 layer의 feature map의 channel 개수로 작은 값을 사용
- **concatenation 연산**을 하기 위해서 각 layer 에서의 output 이 똑같은 channel 개수가 되는 것이 좋음  $\rightarrow$  1x1 convolution으로 growth rate 조절
- 위의 그림 1은  **$k(\text{growth rate}) = 4$  인 경우**를 의미
  - 6 channel feature map인 input이 dense block의 4번의 **convolution block**을 통해  $(6 + 4 + 4 + 4 + 4 = 22)$  개의 channel을 갖는 feature map output으로 계산이 되는 과정
  - DenseNet의 각 dense block의 각 layer마다 feature map의 channel 개수 또한 간단한 등차수열로 나타낼 수 있음
- DenseNet은 작은  $k$ 를 사용  $\rightarrow$  (다른 모델에 비해) 좁은 layer로 구성  $\Rightarrow$  좁은 layer로 구성해도 DenseNet이 좋은 성능을 보이는 이유?

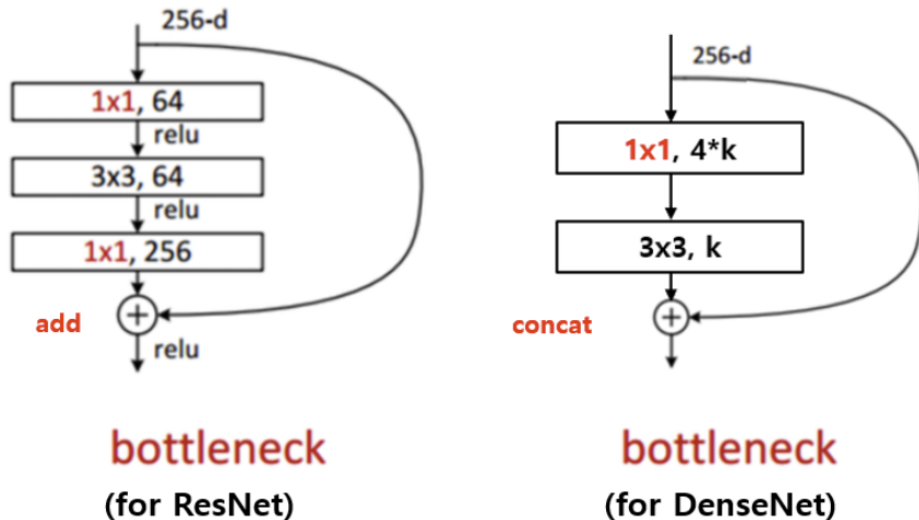


- Dense block내에서 각 layer들은 모든 **preceding feature map**에 접근 가능 (= 네트워크의 “collective knowledge”에 접근) ⇒ (생각) **preceding feature map** = 네트워크의 **global state**
- **growth rate k** → 각 layer가 **global state**에 얼마나 많은 새로운 정보를 contribute할 것인지를 조절
- ⇒ 모든 layer가 접근할 수 있는 **global state**로 인해 DenseNet은 기존의 네트워크들과 같이 layer의 feature map을 복사해서 다른 layer로 넘겨주는 등의 작업을 할 필요가 없음 (= feature reuse)

## Bottleneck layers

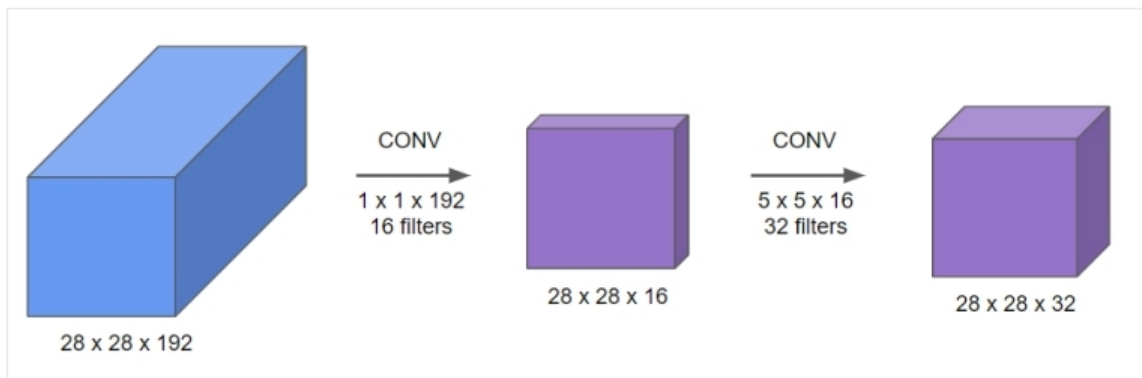


- output의 feature map 수(= channel 개수)를 조절하는 **bottleneck layer**를 사용
- 본 논문에서 H()에 **bottleneck layer**를 사용한 모델을 **DenseNet-B**로 표기
  - Batch Norm → ReLU → Conv ( $1 \times 1$ ) → Batch Norm → ReLU → Conv ( $3 \times 3$ )
  - 본 논문에서, 각  $1 \times 1$  Conv는  $4k$ 개의 feature map을 출력 (단,  $4 * \text{growth rate}$ 의 **4배**라는 수치는 hyper-parameter이고 이에 대한 자세한 설명은 하고 있지 않음)
- $1 \times 1$  convolution → channel 개수 줄임 ⇒ 학습에 사용되는  $3 \times 3$  convolution의 parameter 개수 줄임



- ResNet은 Bottleneck 구조를 만들기 위해서
  - 1x1 convolution으로 dimension reduction을 한 다음 + 다시 1x1 convolution을 이용하여 expansion
- DenseNet은 Bottleneck 구조를 만들기 위해서
  - 1x1 convolution으로 dimension reduction + but, expansion은 하지 않음
  - 대신에 feature들의 concatenation을 이용하여 expansion 연산과 같은 효과를 만듦
    - (생각) feature들의 concatenation으로 채널 개수 expansion → ex. 6 + 4 + ... + 4
- (공통점) 3x3 convolution 전에 1x1 convolution을 거쳐서 input feature map의 channel 개수를 줄임
- (차이점) 다시 input feature map의 channel 개수 만큼 생성(ResNet)하는 대신 growth rate 만큼의 feature map을 생성(DenseNet) ⇒ 이를 통해 computational cost를 줄일 수 있음
- ▼ bottleneck layer
  - Channel 개수가 많아지는 경우, 연산에 걸리는 속도도 그만큼 증가할 수 밖에 없는 데, 이때 Channel의 차원을 축소하는 개념이 Bottleneck layer
  - Convolution Parameters = Kernel Size x Kernel Size x Input Channel x Output Channel

- 이때, **1x1 Convolution** 을 input 값에 Convolution 해주면 해당 input 의 Channel 은 1x1 Convolution 의 Filter 수만큼 축소



- 위의 그림에서와 같이 **1x1 Convolution** 의 Filter 수 만큼 feature 의 Channel 수가 감소
- 이렇게 줄어든 feature 를 통해 3X3, 5X5 등의 Convolution 을 위해 연산에 사용되는 parameter를 줄여 연산의 효율성을 높임

## Compression

- **Compression**은 pooling layer(Transition layer)의 **1x1 Convolution layer** 에서 channel 개수(= feature map의 개수)를 줄여주는 비율 (hyperparameter  $\theta$ )
  - 본 논문에서는  $\theta=0.5$ 로 설정 → transition layer를 통과하면 feature map의 개수 (channel)이 절반으로 줄어들고, 2x2 average pooling layer를 통해 feature map의 가로 세로 크기 또한 절반으로 줄어듦
  - $\theta=1$ 로 설정 시 → feature map의 개수를 그대로 사용

## Implementation Details

### CIFAR, SVHN

Layers	Output Size	DenseNet (k=12, L=40)		DenseNet (k=12, L=100)		DenseNet (k=24, L=100)		DenseNet-BC (k=12, L=100)		DenseNet-BC (k=24, L=250)		DenseNet-BC (k=40, L=190)	
Convolution	32x32	3x3 conv											
Dense Block (1)	32x32	3x3 conv	x12	3x3 conv	x32	3x3 conv	x32	1x1 conv 3x3 conv	x 16	1x1 conv 3x3 conv	x 41	1x1 conv 3x3 conv	x 31
Transition Layer (1)	32x32	1x1 conv											
	16x16	2x2 average pool, stride=2											
Dense Block (2)	16x16	3x3 conv	x12	3x3 conv	x32	3x3 conv	x32	1x1 conv 3x3 conv	x 16	1x1 conv 3x3 conv	x 41	1x1 conv 3x3 conv	x 31
Transition Layer (2)	16x16	1x1 conv											
	8x8	2x2 average pool, stride=2											
Dense Block (3)	8x8	3x3 conv	x12	3x3 conv	x32	3x3 conv	x32	1x1 conv 3x3 conv	x 16	1x1 conv 3x3 conv	x 41	1x1 conv 3x3 conv	x 31
Classification Layer	1x1	8x8 global average pool											
		10D fully-connected, softmax											

- 3 dense blocks (각 block마다 layer 개수는 동일)
- 첫 번째 dense block 이전의 convolution은 16 output channel
- kernel size 3 x 3 + input : zero-padded by one pixel ⇒ feature map size를 고정
- feature map size - 1st : 32 x 32 / 2nd : 16 x 16 / 3rd : 8 x 8

#### • DenseNet structure

- {L=40, k=12}
- {L=100, k=12}
- {L=100, k=23}

#### • DenseNet-BC structure

- {L=100, k=12}
- {L=250, k=24}
- {L=190, k=40}

#### ▼ zero-padded by one pixel

- ex. 6x6 크기의 이미지가 있고, 3x3 필터로 convolution을 하면, output의 크기는 6-3+1로 4x4
- 1 만큼의 padding을 사용하면, 이미지의 외곽에 1 픽셀씩 더함 + zero-padding : 추가된 1 pixel에 0을 부여함
- ⇒ padding 후의 이미지 크기는 8x8이고, output의 크기는 8-3+1=6으로 6x6인 원본 이미지 크기와 같게 유지됨

**ImageNet**

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	$112 \times 112$	$7 \times 7$ conv, stride 2			
Pooling	$56 \times 56$	$3 \times 3$ max pool, stride 2			
Dense Block (1)	$56 \times 56$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	$56 \times 56$	$1 \times 1$ conv			
	$28 \times 28$	$2 \times 2$ average pool, stride 2			
Dense Block (2)	$28 \times 28$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	$28 \times 28$	$1 \times 1$ conv			
	$14 \times 14$	$2 \times 2$ average pool, stride 2			
Dense Block (3)	$14 \times 14$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	$14 \times 14$	$1 \times 1$ conv			
	$7 \times 7$	$2 \times 2$ average pool, stride 2			
Dense Block (4)	$7 \times 7$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	$1 \times 1$	$7 \times 7$ global average pool			
		1000D fully-connected, softmax			

**Table 1:** DenseNet architectures for ImageNet. The growth rate for all the networks is  $k = 32$ . Note that each “conv” layer shown in the table corresponds the sequence BN-ReLU-Conv.

- 4 dense blocks
- $224 \times 224$  input images
- DenseNet-BC structure

## CIFAR, SVHN VS. ImageNet

ImageNet은 다른 두가지 데이터셋에 비해 이미지 사이즈가 크기 때문에 ImageNet과 나머지 두 데이터셋이 다른 architecture를 가짐

- DenseBlock 이전 Convolution 연산의 차이
- DenseBlock, Transition Layer 개수 차이
- 각 Dense Block의 layer 개수 차이
- Fully-connected layer의 output 개수(class 개수) 차이

# Experiments

## Datasets

### ▼ CIFAR

- $32 \times 32$  pixels
- CIFAR-10 : 10 classes / CIFAR-100 : 100 classes

- training set : 50,000 images / test set : 10,000 images / validations set : 5,000 training images
- data augmentation : mirroring / shifting
- preprocessing : normalize the data using channel means + standard deviations

#### ▼ **SVHN**

- 32 x 32 digit images
- training set : 73,257 images / test set : 26,032 images / validation set : 6,000 images
- additional training set : 531,131 images

#### ▼ **ImageNet**

- training set : 1,2 million images / validation set : 50,000 images
- 1000 classes
- data augmentation + 10-crop/single-crop
- 224 x 224 images

## Training

- stochastic gradient descent (SGD)로 train
- weight decay :  $10^{-4}$
- Nesterov momentum : 0.9 without dampening

#### ▼ **CIFAR, SVHN**

- batch size : 64
- 300 or 40 epochs
- learning rate : 0.1 → training epoch가 50%, 75%일 때 0.1배

#### ▼ **ImageNet**

- batch size : 256
- 90 epochs
- learning rate : 0.1 → 30 epochs, 60 epochs마다 0.1배

#### ▼ momentum

- parameter를 update할 때, 현재 gradient에 과거에 누적했던 gradient를 어느정도 보정해서 과거의 방향을 반영하는 것

## Classification Results on CIFAR and SVHN

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ( $k = 12$ )	40	1.0M	<b>7.00</b>	5.24	<b>27.55</b>	24.42	1.79
DenseNet ( $k = 12$ )	100	7.0M	<b>5.77</b>	<b>4.10</b>	<b>23.79</b>	<b>20.20</b>	1.67
DenseNet ( $k = 24$ )	100	27.2M	<b>5.83</b>	<b>3.74</b>	<b>23.42</b>	<b>19.25</b>	<b>1.59</b>
DenseNet-BC ( $k = 12$ )	100	0.8M	<b>5.92</b>	4.51	<b>24.15</b>	22.27	1.76
DenseNet-BC ( $k = 24$ )	250	15.3M	<b>5.19</b>	<b>3.62</b>	<b>19.64</b>	<b>17.60</b>	1.74
DenseNet-BC ( $k = 40$ )	190	25.6M	-	<b>3.46</b>	-	<b>17.18</b>	-

**Table 2:** Error rates (%) on CIFAR and SVHN datasets.  $k$  denotes network's growth rate. Results that surpass all competing methods are **bold** and the overall best results are **blue**. "+" indicates standard data augmentation (translation and/or mirroring). \* indicates results run by ourselves. All the results of DenseNets without data augmentation (C10, C100, SVHN) are obtained using Dropout. DenseNets achieve lower error rates while using fewer parameters than ResNet. Without data augmentation, DenseNet performs better by a large margin.

### Accuracy

- DenseNet-BC with {L=190, k=40} → C10+, C100+에 대해 성능 좋음
- C10/C100에 대해, FractalNet with drop path-regularization 과 비교해서 error가 30% 적음
- DenseNet-BC with {L=100, k=24} → C10, C100, SVHN에 대해 성능 좋음
- SVHN이 비교적 쉬운 task이기 때문에, 깊은 모델은 overfitting할 수 있어서, DenseNet-BC with {L=250, k=24} 는 더 이상 성능이 개선되지 않음

### Capacity

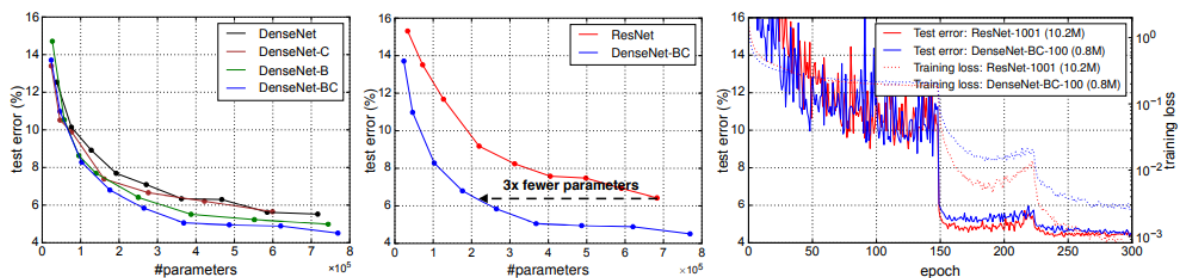
- compression과 bottleneck layer가 없을 때, L과 k가 커질수록 → DenseNet의 성능이 좋아짐
  - 모델이 더 크고(k) 더 깊어질수록(L) 더 많고 풍부한 representation을 학습 가능
- paramter 개수가 늘어날수록 → error 줄어듦

- Error : 5.24% → 4.10% → 3.74%
- Number of parameters : 1.0M → 7.0M → 27.2M
- Overfitting이나 optimization(= parameter update) difficulty가 나타나지 않음

## Parameter Efficiency

- DenseNet-BC with bottleneck structure + transition layer에서의 차원 축소 (dimension reduction)는 parameter의 효율성을 높임
- FractalNet과 Wide ResNets는 30M parameter이고, 250-layer DenseNet은 15.3M parameter 인데, DenseNet의 성능이 더 좋음

## Overfitting



**Figure 4:** Left: Comparison of the parameter efficiency on C10+ between DenseNet variations. Middle: Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. Right: Training and testing curves of the 1001-layer pre-activation ResNet [12] with more than 10M parameters and a 100-layer DenseNet with only 0.8M parameters.

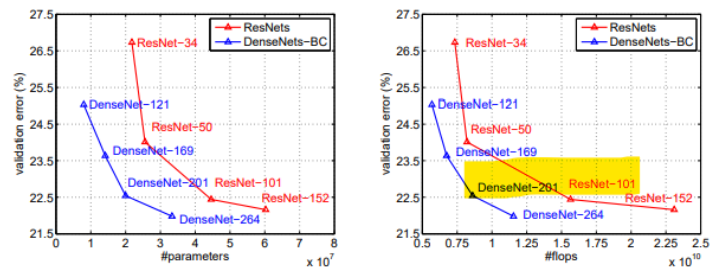
- DenseNet은 overfitting 될 가능성이 적음
- DenseNet-BC with bottleneck structure와 compression layer가 overfitting을 방지하는데 도움
- ResNet-1001과 DenseNet-BC(L=100,k=12)의 error를 비교 (맨 오른쪽 그래프)
  - ResNet-1001은 DenseNet-BC에 비해 training loss는 더 낮지만, test error는 비슷한 것을 알 수 있는데, 이는 DenseNet이 ResNet보다 overfitting이 일어나는 경향이 더 적다는 것을 보여줌

## Classification Results on ImageNet



Model	top-1	top-5
DenseNet-121	25.02 / 23.61	7.71 / 6.66
DenseNet-169	23.80 / 22.08	6.85 / 5.92
DenseNet-201	22.58 / 21.46	6.34 / 5.54
DenseNet-264	22.15 / 20.80	6.12 / 5.29

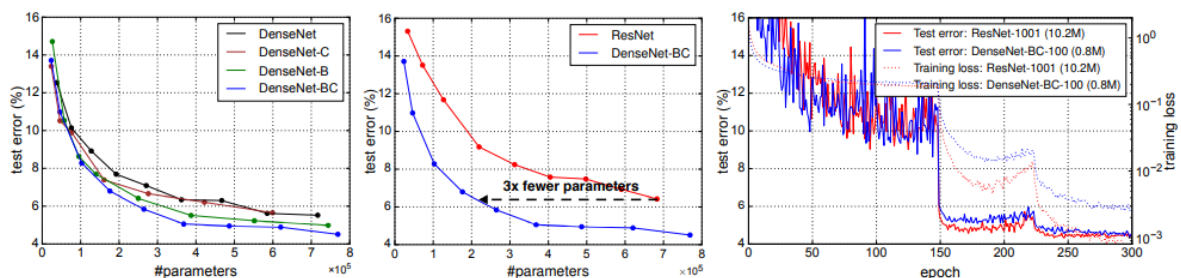
**Table 3:** The top-1 and top-5 error rates on the ImageNet validation set, with single-crop / 10-crop testing.



**Figure 3:** Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

- Table 3(왼쪽 표)은 DenseNet의 ImageNet에서의 single crop, 10-crop validation error
- Figure 3(오른쪽 그림)는 DenseNet과 ResNet의 single crop top-1 validation error를 parameter 개수와 flops를 기준으로 나타냄
  - DenseNet-201 with **20M parameters**와 101-layer ResNet with more than **40 parameter**가 비슷한 성능

## Discussion




**Figure 4:** *Left:* Comparison of the parameter efficiency on C10+ between DenseNet variations. *Middle:* Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. *Right:* Training and testing curves of the 1001-layer pre-activation ResNet [12] with more than 10M parameters and a 100-layer DenseNet with only 0.8M parameters.



### Model compactness

- feature map은 모든 다음 layer에 의해 접근 가능 → feature reuse → **모델이 compact 해짐**
- Figure 4의 가운데 그림에서, DenseNet-BC는 ResNet **parameter 개수의 1/3만**으로 도 비슷한 성과

### Implicit Deep Supervision

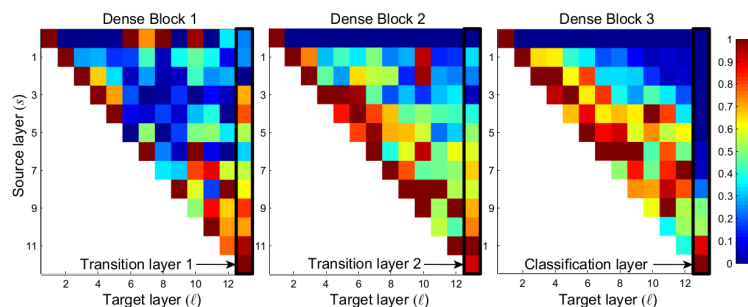
- 각 layer마다 shorter connection을 이용하여 supervision을 손실(loss) 함수에서 얻을 수 있음 ⇒ DenseNet은 deep supervision처럼 볼 수 있음 (?)
- deeply-supervised nets (DSN)
  - 모든 hidden layer마다 classifier가 존재 → 중간 layer마다 discriminative feature를 학습하도록 만들(= feature를 분류하는 능력 학습)
- DenseNet
  - 하나의 classifier가 네트워크의 맨 위에 존재 → 2~3개의 transition layer를 통해 direct supervision을 모든 layer에 전달 ⇒ DSN과 유사
- ▼  deep supervision
  - Deep Neural Network에서 classifier를 여러 개 두어 성능을 올리는 것

## Stochastic VS. deterministic connection

- **Stochastic depth**는 ResNet layer를 랜덤하게 drop하여 layer간의 direct connection을 만드는데, 이때 pooling layer는 drop되지 않아서 DenseNet connectivity pattern와 비슷함
- ⇒ DenseNet와 ResNet의 Stochastic depth은 전혀 다름에도 불구하고 **stochastic regularizer**의 효과를 낸다는 공통점
- **stochastic depth를 DenseNet의 관점에서 해석**
  - (ResNet의) **Stochastic depth**에서는, 무작위로 일부 layer를 drop하고 이들을 둘러싸고 있던 layer끼리 직접적으로 연결시킴 ⇒ 궁극적으로 차이가 있지만 랜덤하게 끊어진 레이어가 다른 레이어들에 연결되는 패턴이 DenseNet의 **dense connectivity 패턴과 비슷**
- ▼  stochastic VS. deterministic
  - stochastic → random과 비슷하고, deterministic과는 반대이고, non-deterministic과는 종종 비슷한 개념으로 쓰임 ⇒ 인풋이 같아도 아웃풋은 다를 수 있음
  - deterministic → 같은 시퀀스 안의 다음 사건이 지금 현재 사건으로부터 결정된다는 것 ⇒ 같은 인풋을 넣으면 항상 같은 결과를 냄
  - (생각) stochastic depth → 랜덤하게 일부 layer를 drop하고 그 layer끼리 직접 연결시킴 (stochastic) ↔ DenseNet → DenseNet에서는 이미 2개의 layer 사이가 직접적으로 연결 되어 있음 (deterministic)
- ▼  stochastic depth regularization

- 네트워크에서 layer를 일정하게 drop하는 기법
- dropout과 비슷하게 일정 확률로 특정 layer를 drop시켜 주위에 있는 다른 layer와 연결되게 만드는 것
- Dropout을 사용하는 이유?
- 어떤 특정한 설명변수 feature만을 과도하게 집중하여 학습하는 과적합 (Overfitting)을 방지하기 위해 사용

## Feature reuse

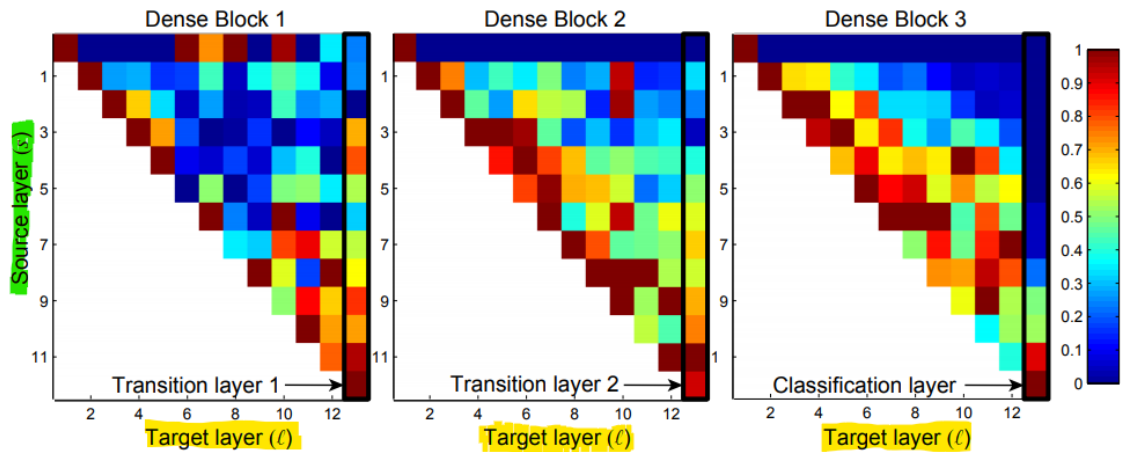


**Figure 5:** The average absolute filter weights of convolutional layers in a trained DenseNet. The color of pixel  $(s, \ell)$  encodes the average  $L1$  norm (normalized by number of input feature-maps) of the weights connecting convolutional layer  $s$  to  $\ell$  within a dense block. Three columns highlighted by black rectangles correspond to two transition layers and the classification layer. The first row encodes weights connected to the input layer of the dense block.

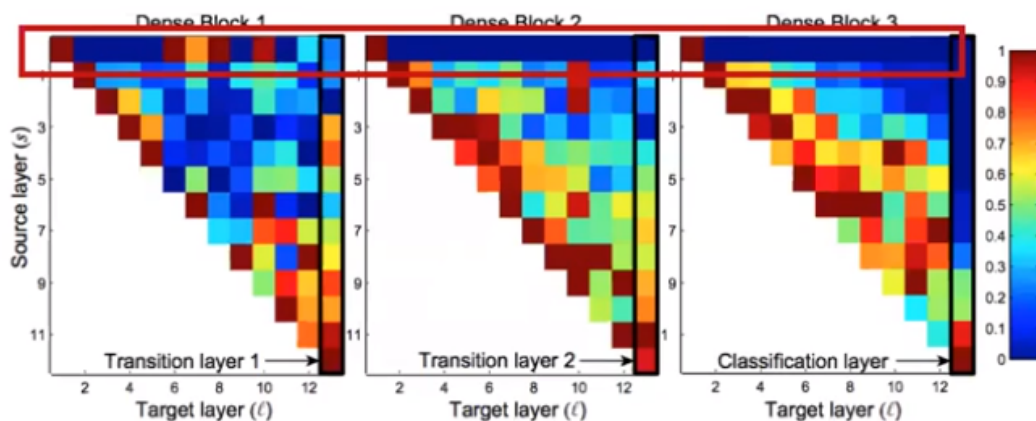
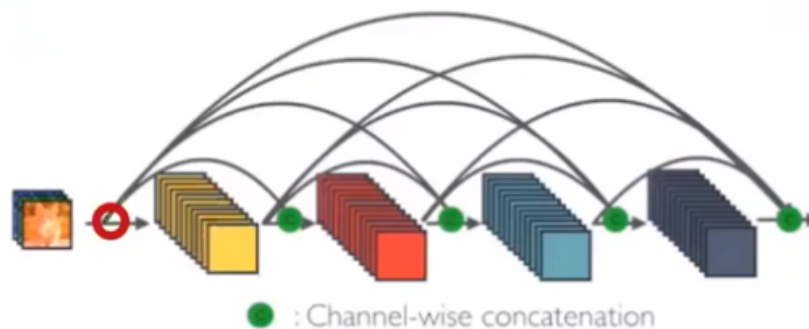
- 학습된 DenseNet의 각 layer가 실제로 preceding layer들의 feature map을 활용하는지를 실험
  - 학습한 네트워크의 각 dense block에서,  $\ell$ 번째 convolution layer에서  $s$ 번째 layer로의 할당된 average absolute weight를 계산 (absolute는 음의 값을 갖는 weight를 고려한 것으로 보임)
- 위 그림은 dense block 내부에서 convolution layer들의 weight의 평균이 어떻게 분포되어있는지 보여줌
- Pixel  $(s, \ell)$ 의 색깔은 dense block 내의 conv layer  $s$ 와  $\ell$ 을 연결하는 weight의 average L1 norm으로 인코딩 한 것  $\Rightarrow$  각 dense block의 weight들이 가지는 크기 값을 0 ~ 1 사이 범위로 normalization 한 결과
  - 빨간색인 1에 가까울 수록 큰 값  $\leftrightarrow$  파란색인 0에 가까울수록 작은 값
- 실험 결과

- 각 layer들이 동일한 block 내에 있는 preceding layer들에 weight를 분산 시킴 ( $\because$  각 열에서 weight가 골고루 spread되어 있음)
  - $\Rightarrow$  Dense block 내에서, **실제로 later layer는 early layer의 feature map을 사용하고 있음**
- Transition layer도 preceding layer들에 weight를 분산 시킴 ( $\because$  가장 오른쪽 열에서 weight가 골고루 spread 되어 있음)
  - $\Rightarrow$  Dense block 내에서, **1번째 layer에서 가장 마지막 layer까지 information flow가 형성되어 있음**
- 2, 3번째 dense block은 transition layer의 output에 매우 적은 weight를 일관되게 할당 ( $\because$  2, 3번째 dense block의 첫번째 행에서 weight가 거의 0에 가까움)
  - $\Rightarrow$  **2, 3번째 dense block의 transition layer output은 redundant features가 많아서 매우 적은 weight를 할당(중복된 정보들이 많아 모두 사용하지 않아도 된다는 의미)**
  - $\Rightarrow$  **DenseNet-BC에서 compression  $\theta$ 로 이러한 redundant feature들을 compress하는 현상과 일치**
  - (생각) **Compression**은 pooling layer(Transition layer)의 **1x1 Convolution layer** 에서 **channel 개수(= feature map의 개수)를 줄여주는 비율** (hyperparameter  $\theta$ )이므로, 중복된 정보들이 transition layer에서 제거된다는 의미  $\rightarrow$  channel 개수 감소
- 마지막 classification layer는 전체 dense block의 weight를 사용하긴 하지만, early layer보다 later layer의 feature map을 더 많이 사용함 ( $\because$  3번째 dense block의 가장 마지막 열에서 weight가 아래쪽으로 치우쳐 있음)
  - $\Rightarrow$  **High-level feature가 later layer에 더 많이 존재함**

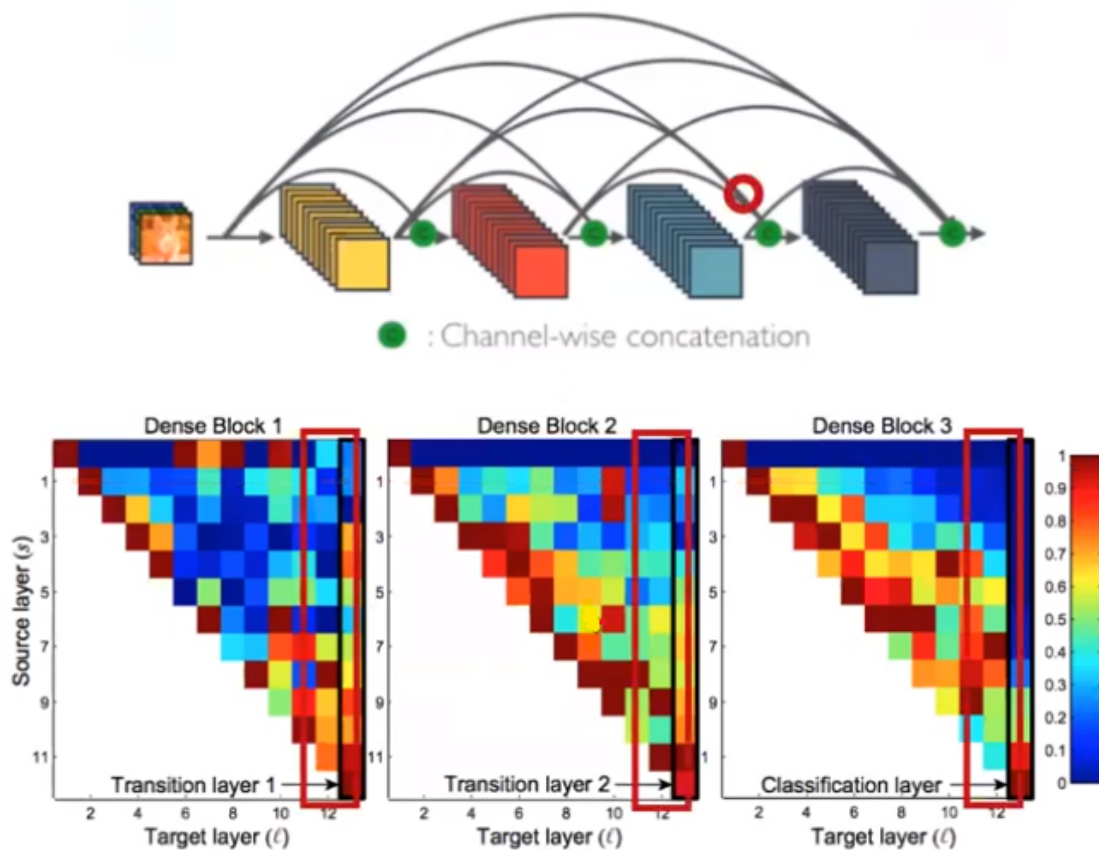
▼ 참고 : [DenseNet \(Densely connected convolution networks\) - gaussian37](#)



- 위 그림은 각 **source** → **target**으로 propagation된 **weight**의 값 분포를 나타냄
- 세로축 **Source layer** → layer가 propagation 할 때, 그 Source에 해당하는 layer가 몇번째 layer인 지 나타냄
- 가로축 **Target layer** → Source에서 부터 전파된 layer의 목적지가 어디인지 나타냄
- ex. dense block 1의 세로축(5), 가로축 (8)에 교차하는 작은 사각형이 의미하는 것은 dense block 1에서 5번째 layer에서 시작하여 8번째 layer로 propagation된 **weight**



- ex. 각 dense block의 **Source가 1인** 부분들을 살펴 보면 각 Block의 **첫 layer**에서 펼쳐진 propagation에 해당 (위 그림에서 빨간색 동그라미에 해당하는 부분)



- ex. 각 dense block의 **Target이 12인** 부분들을 살펴 보면 **다양한 Source에서 weight들이 모이게** 된 것을 볼 수 있음 (위 그림에서 빨간색 동그라미에 해당하는 부분)

## 참고

- [DenseNet \(Densely Connected Convolutional Networks\)\\_ \(tistory.com\)](https://tistory.com/1111111)
- [DenseNet \(Densely connected convolution networks\) - gaussian37](https://tistory.com/1111111)
- [DenseNet Tutorial \[1\] Paper Review & Implementation details](https://tistory.com/1111111)
- [Bottleneck layer \(tistory.com\)](https://tistory.com/1111111)
- [DenseNet논문 \(velog.io\)](https://velog.io/1111111)
- [Densely Connected Convolutional Networks - 딥러닝의 정리노트 \(younnggsuk.github.io\)](https://younnggsuk.github.io/1111111)
- [Densely Connected Convolutional Networks \(tistory.com\)](https://tistory.com/1111111)

- [hoya012.github.io](https://hoya012.github.io)
- [배치 정규화\(Batch Normalization\) - gaussian37](#)
- [Global Average Pooling 이란 - gaussian37](#)
- [\[CV Study\] Densely Connected Convolutional Networks \(tistory.com\)](#)
- [Dense Net\(2018\)논문 정리 \(tistory.com\)](#)
- [DenseNet\(Densely connected Convolutional Networks\) - 2 \(jayhey.github.io\)](#)
- [CV-Paper-Implementation/densenet.py at main · younnggsuk/CV-Paper-Implementation · GitHub](#) - DenseNet 코드 참고 (파이썬)
- [Densely connected convolutional networks \(tistory.com\)](#)
- [Padding은 왜 할까? \(brunch.co.kr\)](#)
- [Densely Connected Convolutional Networks \(tistory.com\)](#)
- [\[딥러닝\] Drop-out\(드롭아웃\)은 무엇이고 왜 사용할까? \(tistory.com\)](#)
- [ch2 용어정리-1 stochastic이란 \(velog.io\)](#)
- [Nesterov Momentum \(velog.io\)](#)