ELLab

SUNG KYUN KWAN UNIVERSITY

DenseNet

2025.11.12

Introduction & Related Work

Idea

- CNN에서 layer 깊어짐에 따라 vanishing 문제 발생함
  - ResNet & Highway Networks: identity connection으로 극복 시도함
  - FractalNet: parallel layer sequences

✓ Key concept: create short paths from early layers to later layers

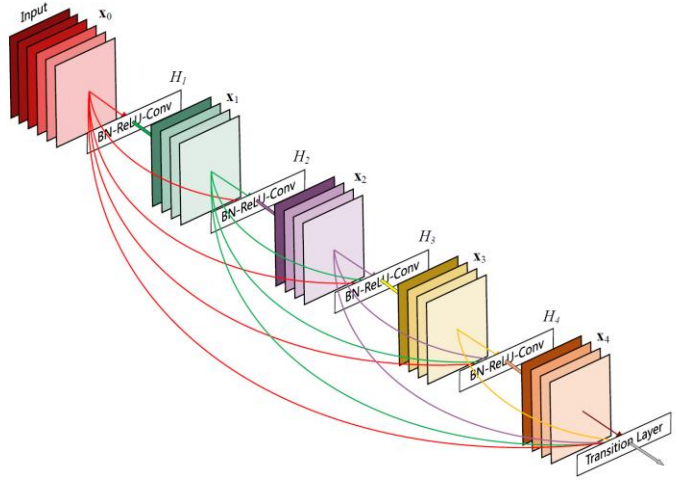


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- DenseNet Idea: 각 층에 대해 모든 이전 층의 특징맵을 해당 layer의 input으로 사용하고, 해당 layer의 모든 feature map들도 이후 모든 층의 입력으로 사용함
  - Benefit: 기울기 소실 완화, 특징 전파 강화, 특징 재사용 촉진, 파라미터 수 줄이기

DenseNet

Representation

$x_0, x_1, x_2, \ldots, x_l$

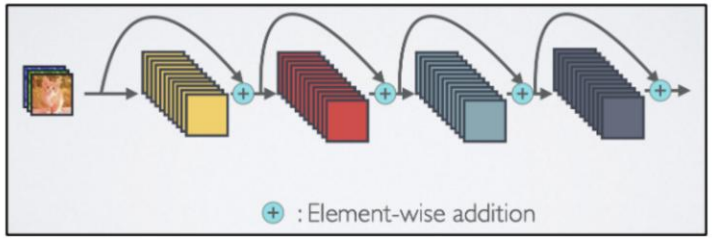
l: index of layer

ResNet: $x_l = H_l(x_{l-1}) + x_{l-1}$

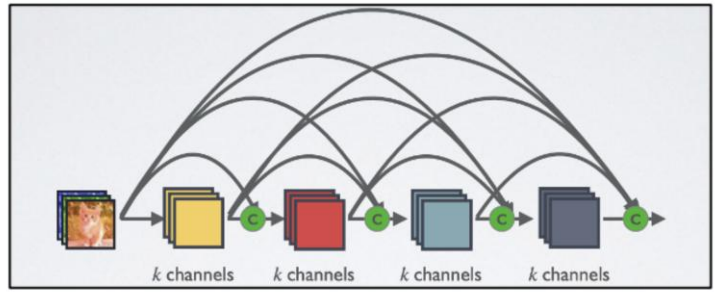
H: include BN, ReLU, Conv

DenseNet: $\mathrm{x_l} = H_l([x_0, x_1, \ldots, x_{l-1}])$

[ ]: concatenate



ResNet

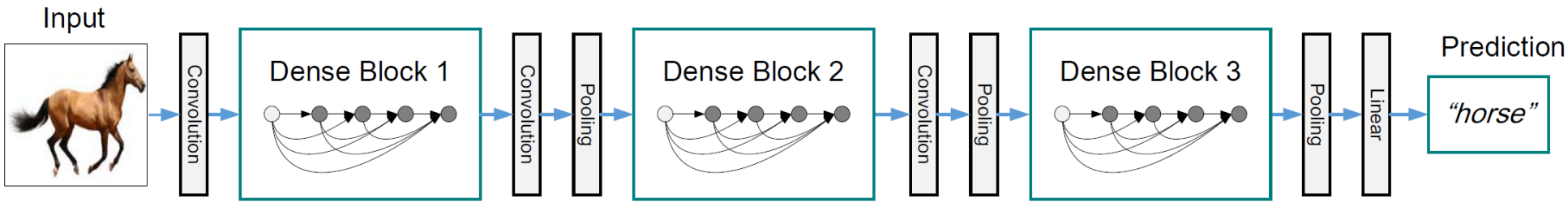


DenseNet

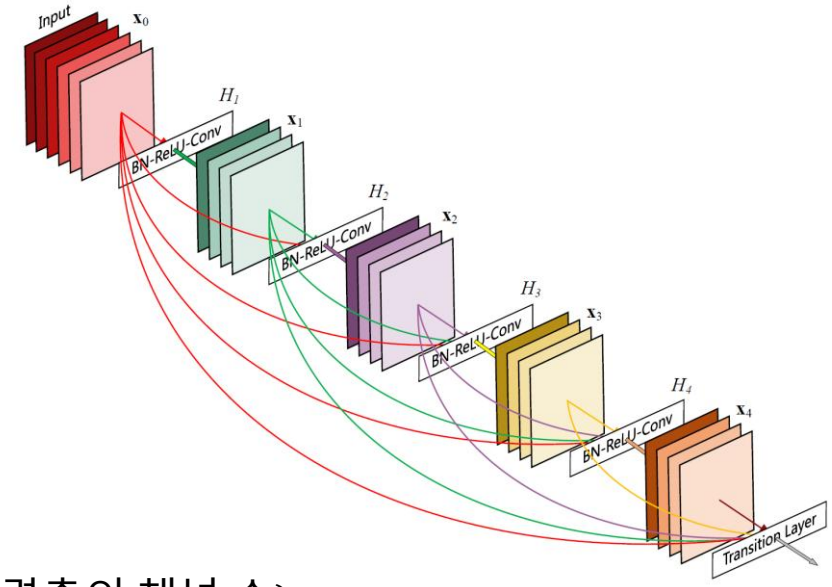
- ResNet
  - identity path를 통해 gradient가 직접 전달 될 수 있음
  - 하지만 항등 연결과 conv 출력이 덧셈으로 결합되기 때문에 네트워크 내 흐름이 일부 제한 될 수 있음
- DenseNet
  - 여러 입력들은 하나의 단일 텐서로 concatenate함
  - 따라서 각 층은 이전 층의 모든 feature를 직접 이용할 수 있음

Architecture

Basic version



- Composite function
  - $H_l(\cdot)$ is BN → ReLU → 3×3 Convolution
- Transition layers
  - Feature map 크기가 달라지는 구간에 사용
  - BN → 1×1 Conv → 2×2 Average Pooling
- Growth rate
  - 각 층이 새로 만들어내는 feature map 수
  - l번째 층의 입력 feature map 수: $k_0 + k \times (l-1)$ (k_0는 입력층의 채널 수)
  - 각 층이 만든 k개의 feature map을 네트워크의 전체 feature map인 global state에 더함
  - 따라서, growth rate는 각 층이 global state에 얼마나 새로운 정보를 추가하는지를 조절하는 역할을 함



Architecture

Upgrade version

- Compression
  - 모델을 더 압축하기 위해 transition layer에서 feature map의 수를 줄임
  - 어떤 dense block이 m개의 feature map을 가진다면 그 다음 transition layer에서 θm개의 feature map 출력하도록 함
  - $0 < \theta \leq 1$

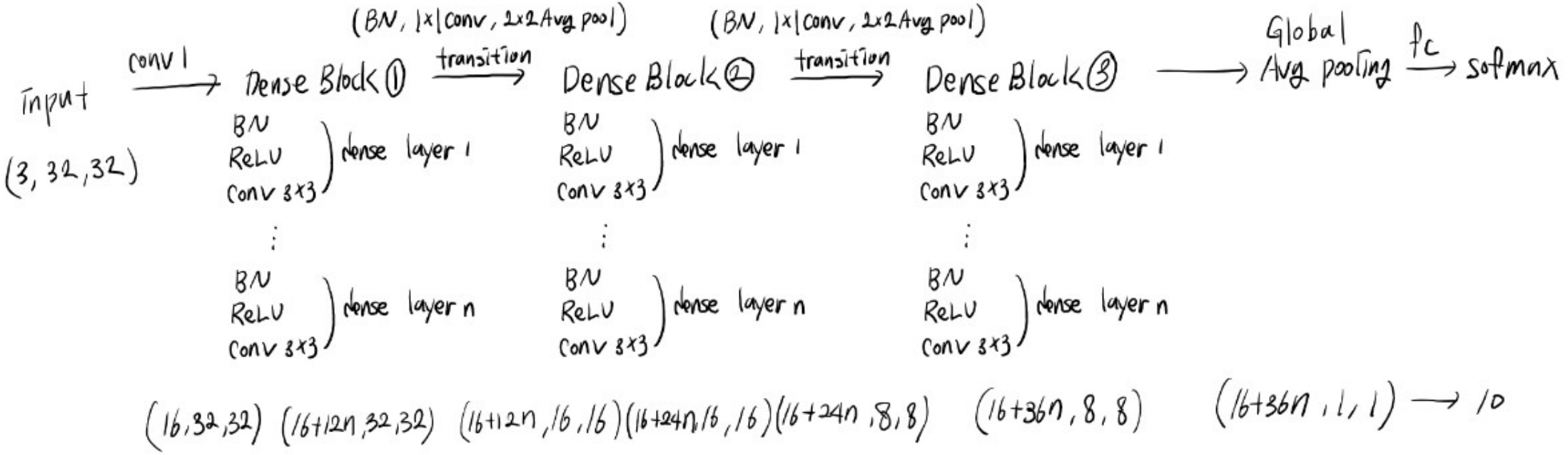
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

- Bottleneck layers
  - 각 층은 출력 feature map이 k개뿐이지만, 입력 feature map은 훨씬 많을 수 있음
  - DenseNet-B의 한 층 구조: BN → ReLU → 1×1 Conv(4k) → BN → ReLU → 3×3 Conv(k)

Architecture

Implementation Details

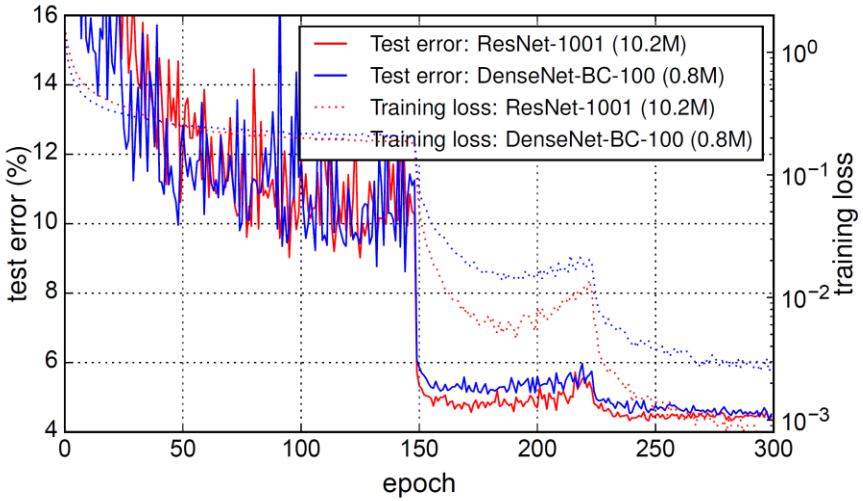
- 3개의 Dense Block으로 구성, 각 블록은 n개의 층을 가짐
- 초기 conv 출력 채널: 16 (DenseNet-BC에서는 growth rate의 두 배 2k)
- Dense Block 안에는 feature map 크기 유지 (3x3, zero padding)
- 차원 바뀌는 곳에서 transition layer



Experiment

Parameter Efficiency

- 특히 bottleneck 구조와 transition layer에서의 차원 축소를 적용한 DenseNet-BC는 매우 파라미터 효율적임
- 1001층 ResNet은 훈련 손실은 더 낮지만, 테스트 오류율은 거의 동일함
- L=100,k=12 DenseNet-BC는 1001층 pre-activation ResNet과 거의 동일한 성능(C10+: 4.51% vs 4.62%, C100+: 22.27% vs 22.71%)을 달성하면서,파라미터는 90%나 적게 사용함
- DenseNet은 일반화 성능이 훨씬 효율적임을 보여줌



"Training and testing curves of the 1001*−layer preactivation ResNet* [12] *with more than* 10*M parameters and a* 100*−layer DenseNet with only* 0.8*M parameters."*

Experiment

CIFAR 10, 100, SVHN

- Capacity
  - Compression이나 bottleneck을 사용하지 않는 경우 L과 k가 커질수록 DenseNet의 성능이 향상되는 일반적 경향을 보임
  - DenseNet이 더 크고 깊은 모델의 표현 능력을 효율적으로 활용할 수 있음

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	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

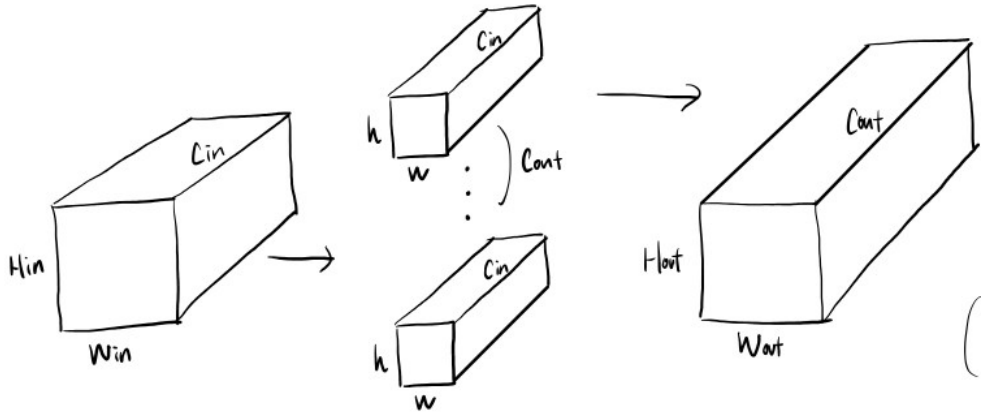
- Overfitting
  - C10에서 k=12를 k=24로 늘려 파라미터가 약 4배 증가했지만, 오히려 오류율이 5.77% → 5.83%로 소폭 증가함
  - Feature 재사용을 너무 많이 했거나 k가 너무 커서 파라미터가 task에 비해 너무 많아졌을 것으로 추정됨

✓ DenseNet-BC의 bottleneck 및 compression 구조가 이러한 과적합 경향을 효과적으로 완화함

Experiment

CIFAR 10, 100, SVHN

"on C10, a 4 growth of parameters produced by increasing k=12 to k=24"



파라미터 수 $\propto \mathrm{h} \times \mathrm{w} \times \mathrm{C_{in}} \times \mathrm{C_{out}}$

$Number\ of\ channel\ in\ l\colon\ C_l^{in} = C_0 + (l-1)k$

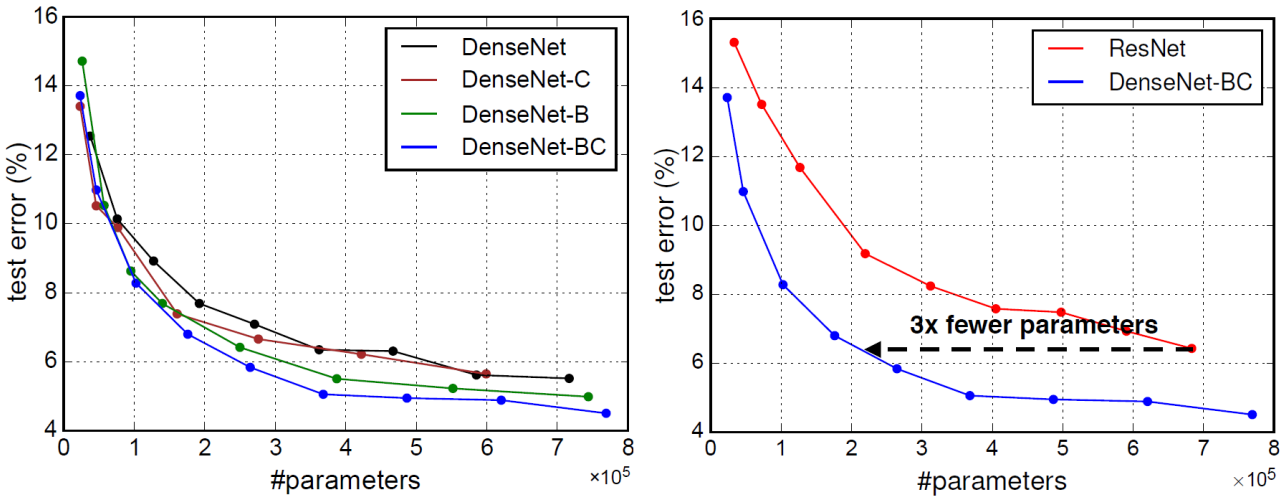
✓ $k \rightarrow 2k$, 파라미터는 약 k^2만큼 증가함

✓ 12 -> 24로 두배 증가했으므로 파라미터는 약 4배 증가함

$$P_{block} = \sum_{\mathrm{l}=1}^{\mathrm{n}} (3 \times 3(C_0 + (l-1)k) \times k) = (C_0 n + \frac{\mathrm{k}n(n-1)}{2}) \times 9\mathrm{k} = \frac{9}{2}k^2 n(n-1) + 9C_0 n$$

Experiment

Model compactness

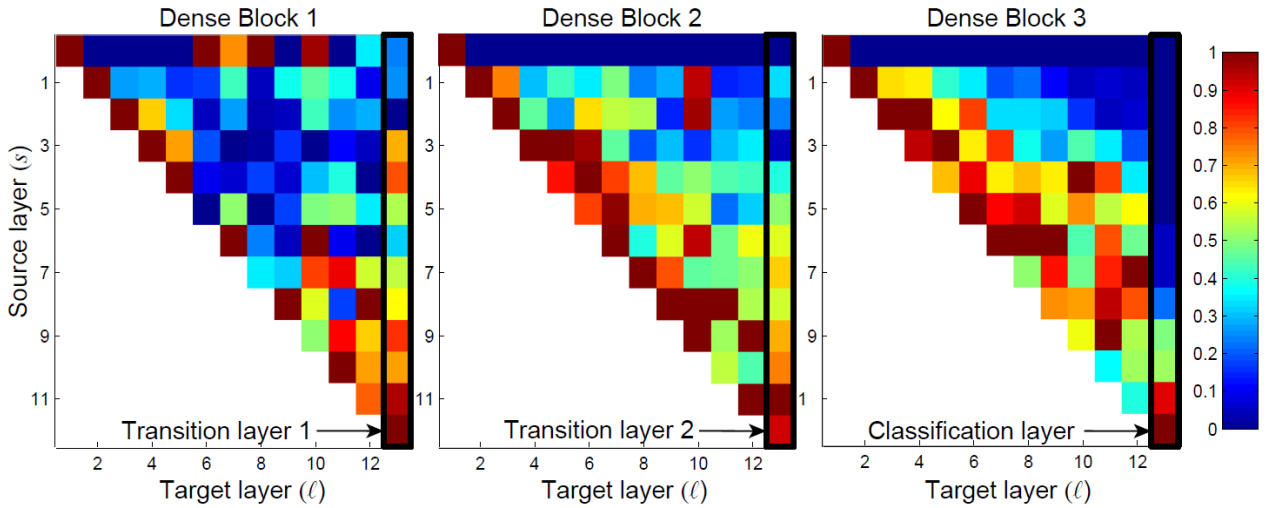


- DenseNet-BC가 항상 DenseNet 계열 중 파라미터 효율이 가장 높음
- 같은 수준의 정확도를 내기 위해 DenseNet-BC는 (PreAct)ResNet이 쓰는 파라미터의 약 1/3만 필요함
- 입력들을 concat 하면서 이전에 어떤 layer가 학습한 feature map에 이후의 모든 레이어가 직접적으로 접근할 수 있음
- 네트워크 전반에서 feature 재사용 촉진하고 bottlenec과 compression를 사용했을때 더 compact한 모델이 됨

Experiment

Feature Reuse

- Dense Block 1
  - 초반에서 후반으로 갈수록 빨간색이 많음 -> 후반 layer에서 앞쪽 feature들을 조합해서 사용함
  - Feature를 점점 누적해서 풍부하게 만듦
- Dense Block 2
  - 내부 색이 다채로워짐- >layer간 연결이 강해지면서 내부에서 새로운 feature를 더 만들어냄



- Dense Block 3
  - 중후반부에 진한 빨강 많음 -> 후반 layer일수록 더 복잡한 feature를 많이 사용함
  - 마지막 쪽 feature에 집중해서 classification 수행함

✓ 모든 레이어가 앞선 레이어의 feature를 재사용하며 Transition Layer는 전체 feature를 통합하고 후반부에서는 고수준 feature에 집중하여 효율적으로 분류를 수행함

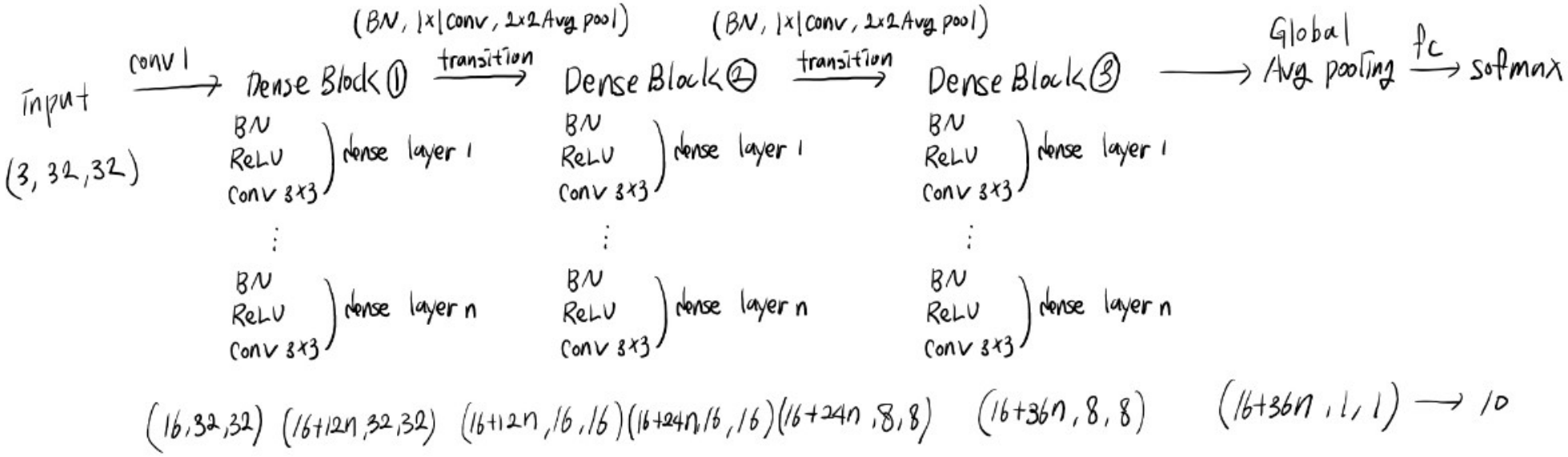
My Code

학습 세팅

항목	논문 설정값
데이터셋	CIFAR-10 (4.5k train / 0.5k test)
이미지 전처리	32×32, 4픽셀 zero-padding 후 32×32 랜덤 crop, horizontal flip
모델 깊이	n={12, 32, 31, 41} → 40, 100, 190, 250층
Optimizer	SGD (momentum=0.9)
Batch size	64
Weight decay	1e-4
Learning rate	0.1(0.05) → 1/10 at 50%, 75% epoch
Scheduler	epoch = [150, 225]
Drop out	drop_rate=0.2
Initialization	He initialize
Training epochs	300 epochs

My Code

Architecture



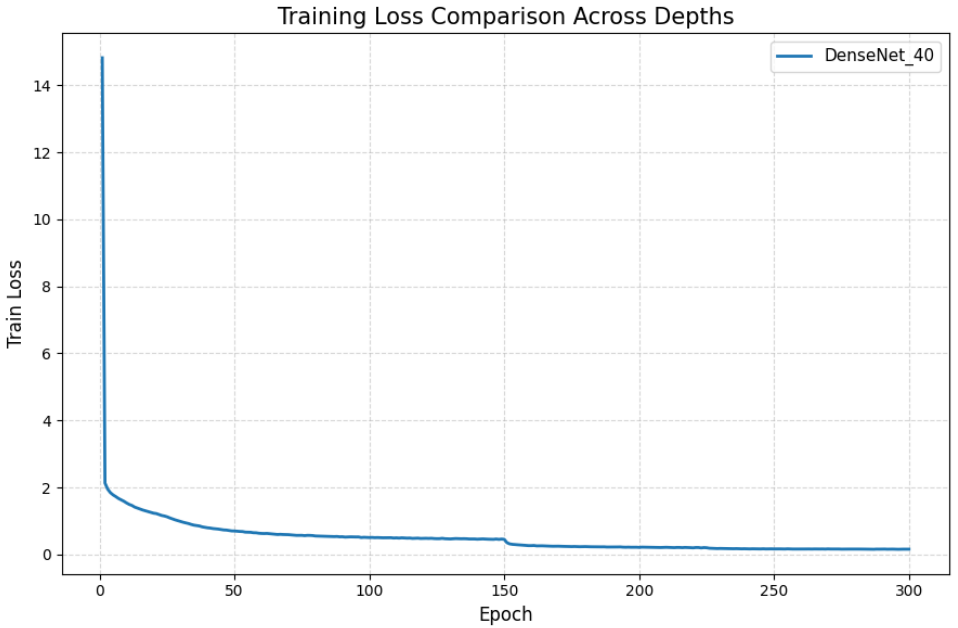
- Class 종류: DenseLayer, DenseBlock, TransitionLayer, DenseNet
- 입력변수: compensate(theta), Growth rate(k)
- Bottleneck 구조는 첫번째 conv1 출력을 2k로 함
- Layer 수: 기본 구조는 3n+4, bottleneck구조는 6n+4

My Code

Experiment

- 논문과 최대한 비슷하게 구현했지만 0.1의 learning rate로 돌렸을때 train loss가 계속 nan이 나왔음
- 0.05로 낮추어 설정하였을때 수렴 nan이 안뜨고 수렴하길래 끝까지 학습을 시킴
- 그러나 shcheduler로 인해 0.1씩 150, 225 epoch에서 lr가 또 줄어들어서 lr가 너무 작아짐
- 300 epoch동안 제대로 수렴하지 못함
- Scheduler 타이밍을 뒤로 미루고 다시 실험 해보려 했으나 Colab gpu 부족 이슈로 실험을 끝마치지 못함

https://colab.research.google.com/drive/11duM97_KsVQy-L_VCOK4YhWC4v1jKs8V?usp=sharing



Layers	Train Loss (last epoch)	Validation Accuracy
40 (n=12)	0.1579	89.08%
100 (n=32)	..	..
190 (n=31)	..	..
250 (n=41)	..	..

ELLab