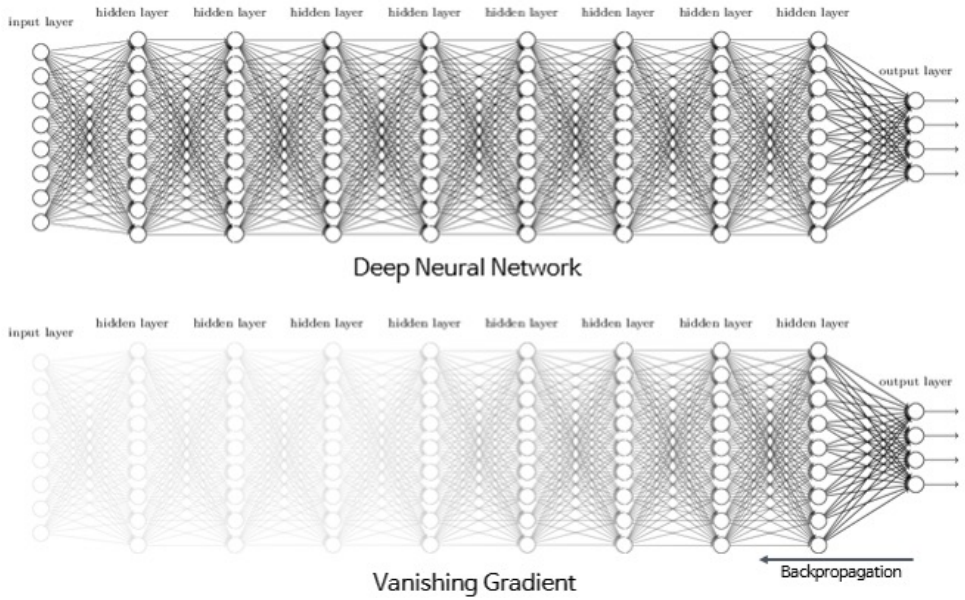
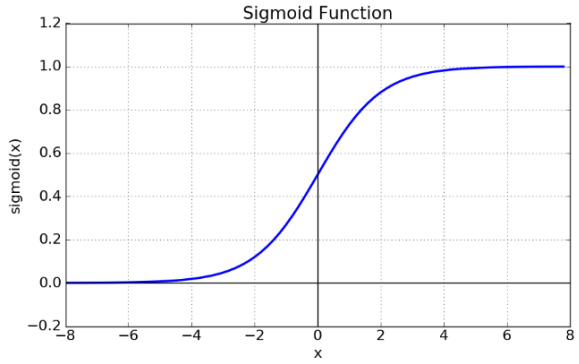
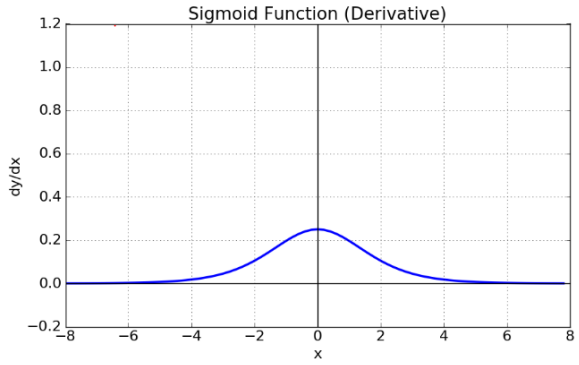
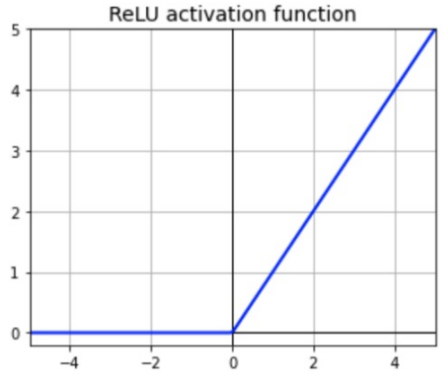
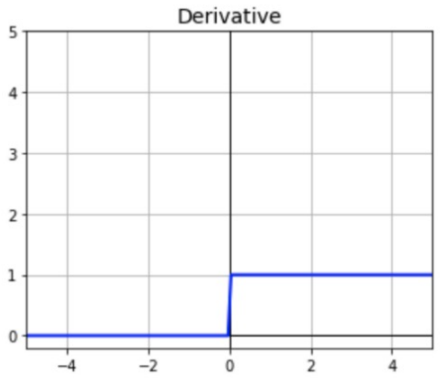
Background


input layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
output layer
Deep Neural Network
input layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
hidden layer
output layer
Backpropagation
Vanishing Gradient

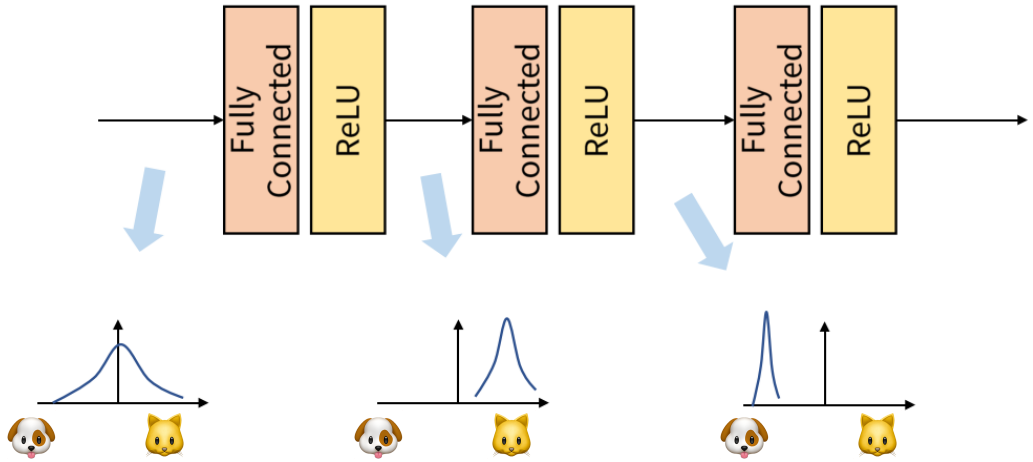

Sigmoid Function
sigmoid(x)
x

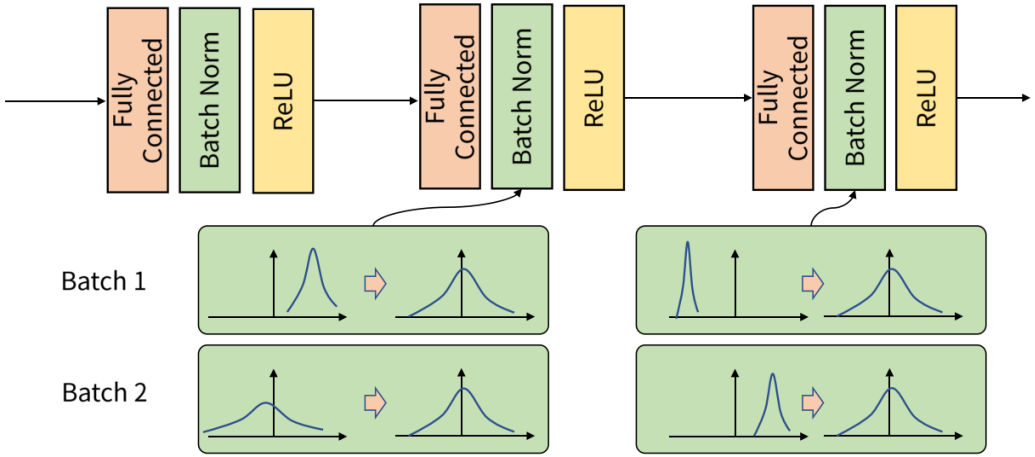

Sigmoid Function (Derivative)
dy/dx
x


ReLU activation function


Derivative

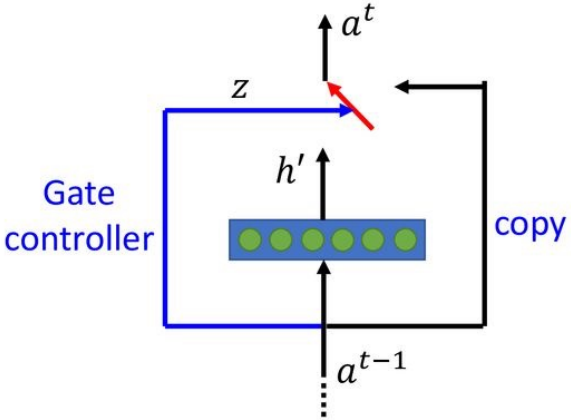
Background


Fully Connected
ReLU
Fully Connected
ReLU
Fully Connected
ReLU

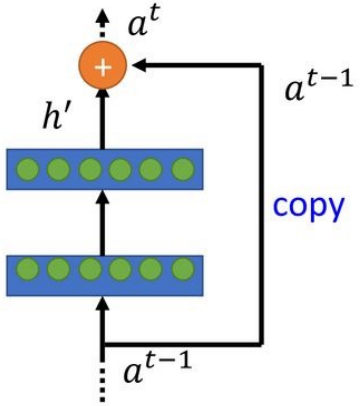

Fully Connected
Batch Norm
ReLU
Fully Connected
Batch Norm
ReLU
Fully Connected
Batch Norm
ReLU
Batch 1
Batch 2

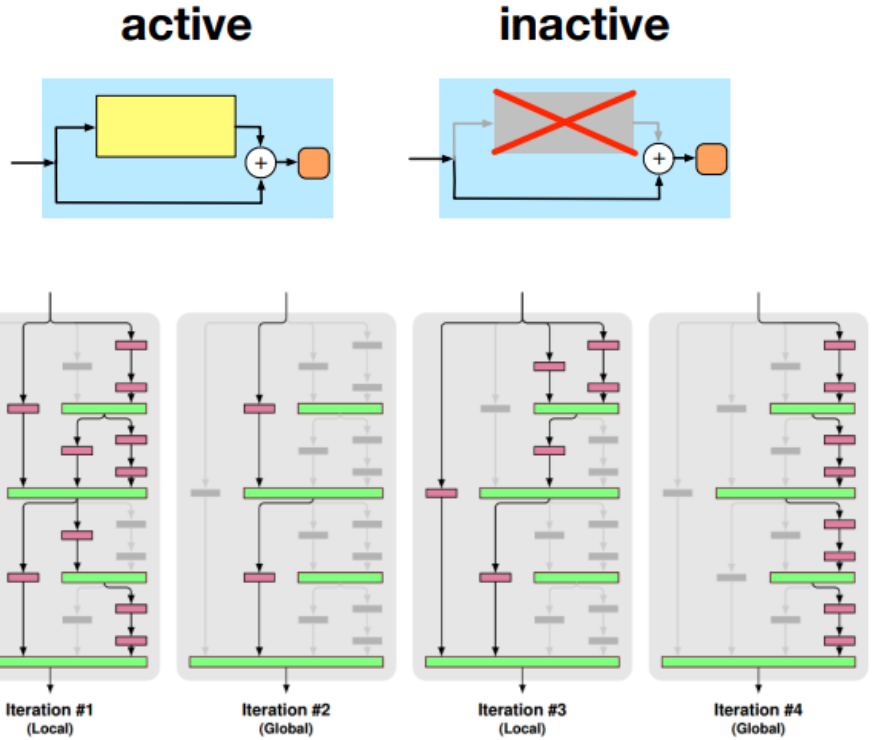
Background

• Highway Network



• Residual Network





Background

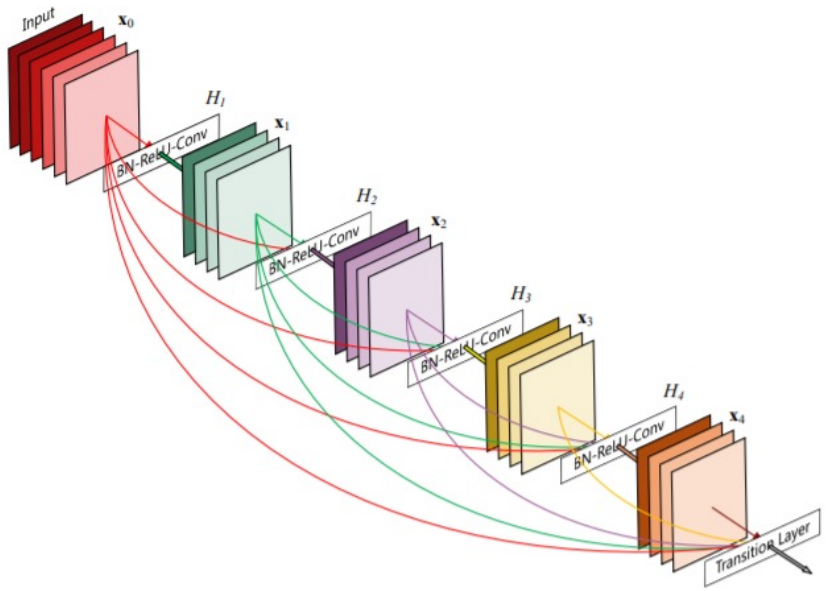


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Background

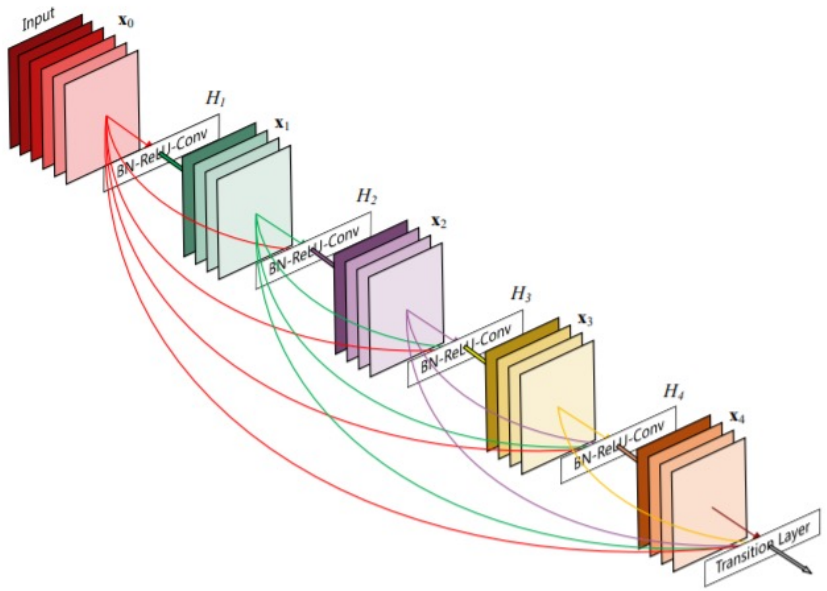
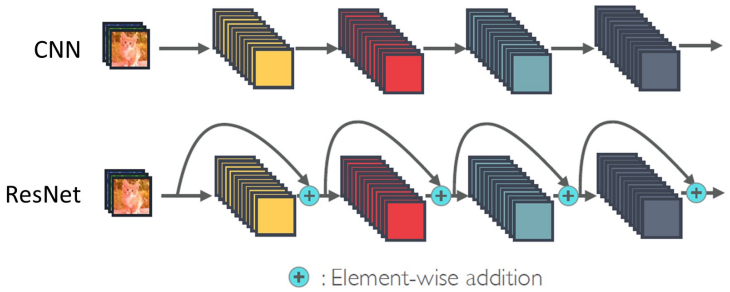
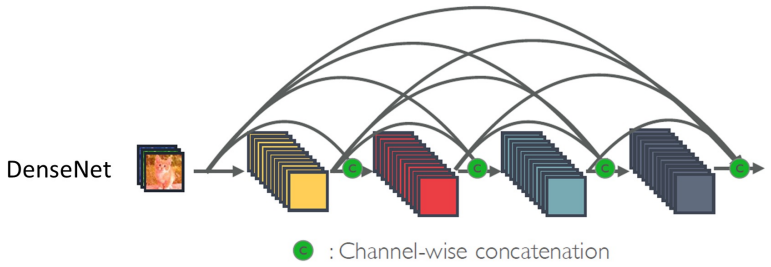


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

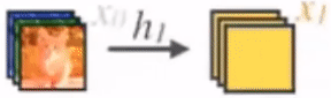
Dense Connectivity



$$X_l = H_l(X_{l-1}) + X_{l-1}$$



$$X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$$




```
import keras
import tensorflow as tf
import keras.backend as K

a = tf.constant([1,2,3])
b = tf.constant([4,5,6])

add = keras.layers.Add()
print(K.eval(add([a,b])))
#output: [5 7 9]

concat = keras.layers.Concatenate()
print(K.eval(concat([a,b])))
#output: array([1, 2, 3, 4, 5, 6], dtype=int32)
```

Growth Rate

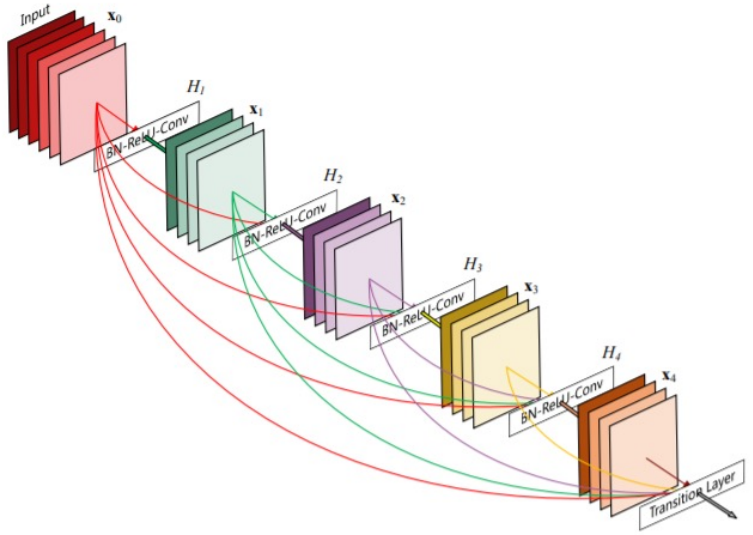


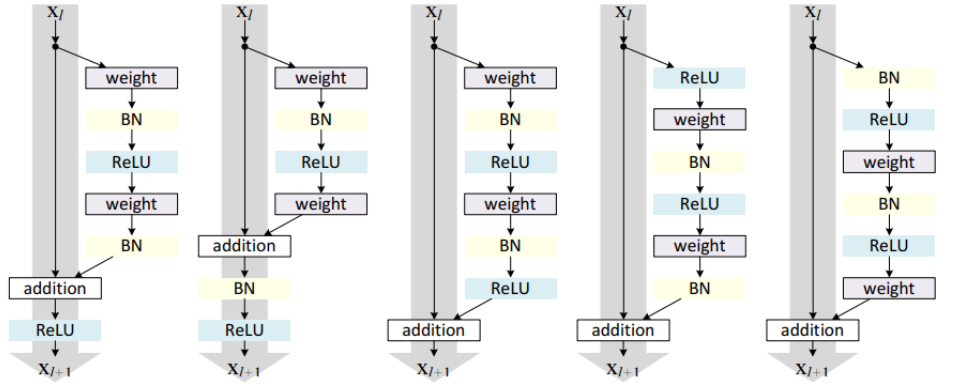
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Composite Function

$$X_l = H_l([X_0, X_1, \dots, X_{l-1}])$$

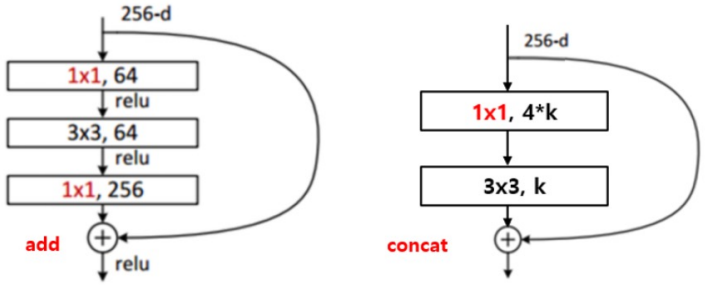
H : Batch Normalization + ReLu + 3*3 Conv

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**



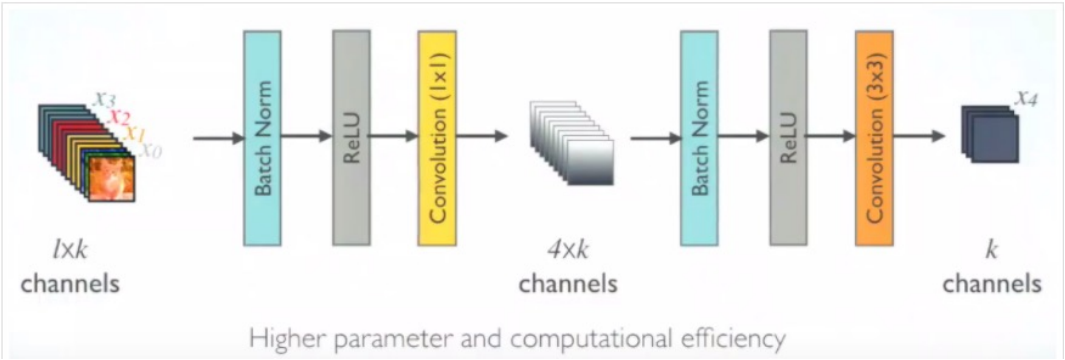
(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

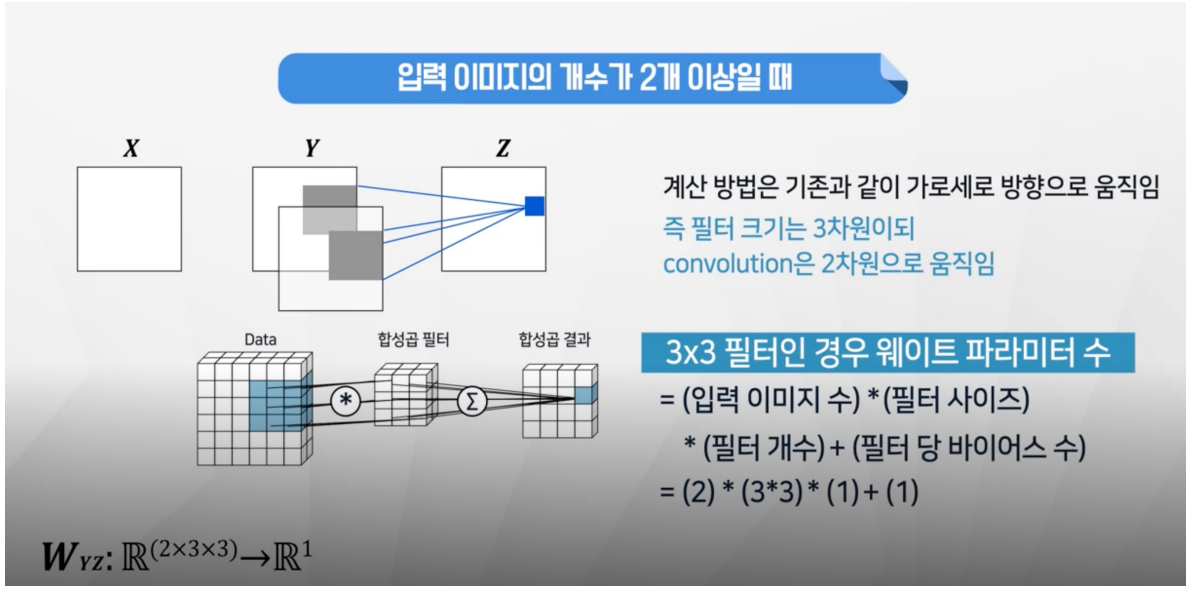
Bottleneck Layer



bottleneck (for ResNet)

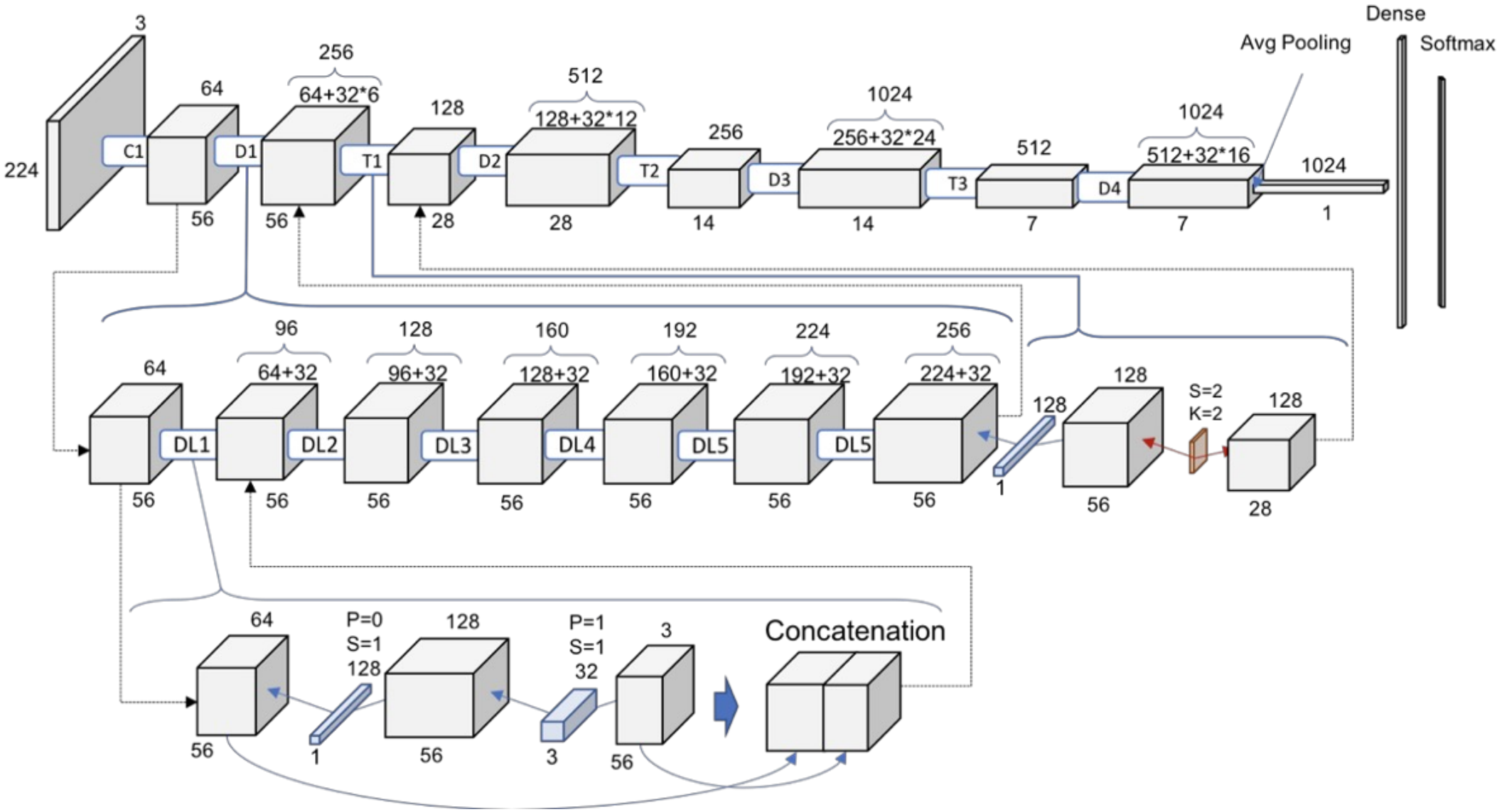
bottleneck (for DenseNet)



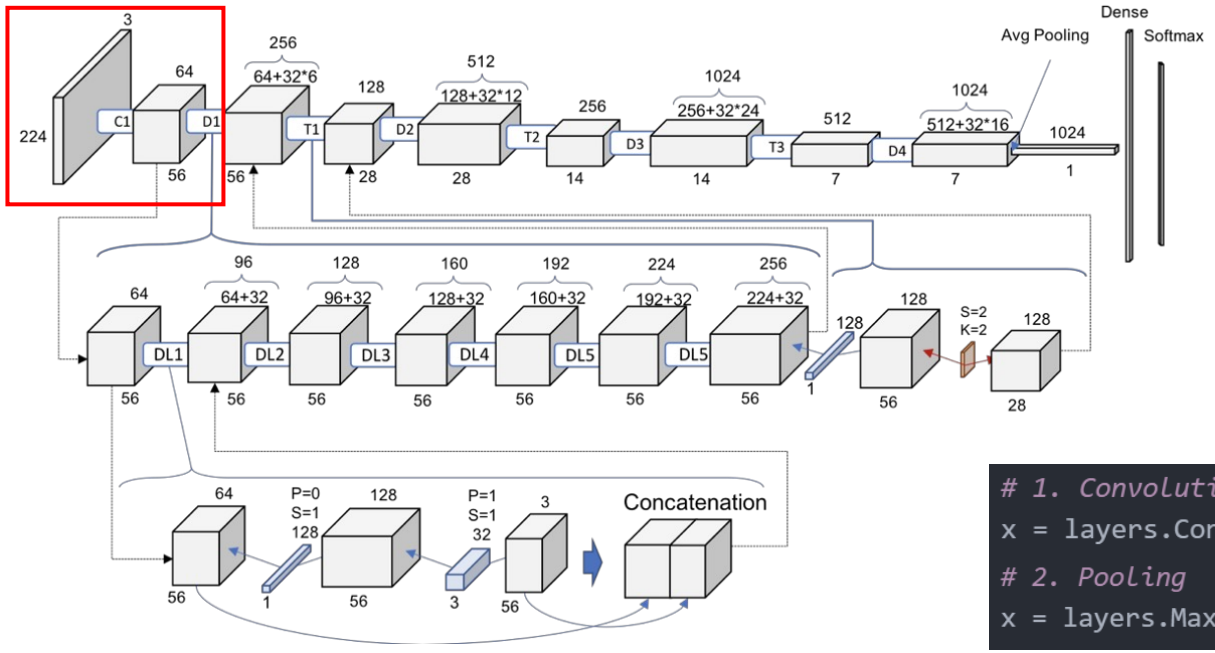


Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times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

Dense-121


3
224
C1
64
56
D1
256
64+32*6
56
T1
128
28
D2
512
128+32*12
28
T2
256
14
D3
1024
256+32*24
14
T3
512
7
D4
1024
512+32*16
7
Avg Pooling
1024
1
Dense
Softmax
DL1
64
56
DL2
96
64+32
56
DL3
128
96+32
56
DL4
160
128+32
56
DL5
192
160+32
56
DL5
224
192+32
56
224+32
256
56
128
1
128
56
S=2
K=2
128
28
64
56
P=0
S=1
128
1
128
56
P=1
S=1
32
3
3
56
Concatenation

Convolution Layer



Layers	Output Size	DenseNet-121
Convolution	112 × 112	7 × 7 conv, stride 2
Pooling	56 × 56	3 × 3 max pool, stride 2

```
# 1. Convolution
x = layers.Conv2D(k * 2, (7, 7), strides=2, padding='same', input_shape=(224, 224, 3))(x)
# 2. Pooling
x = layers.MaxPool2D((3, 3), 2, padding='same')(x)  # 56x56x64
```

Dense Block

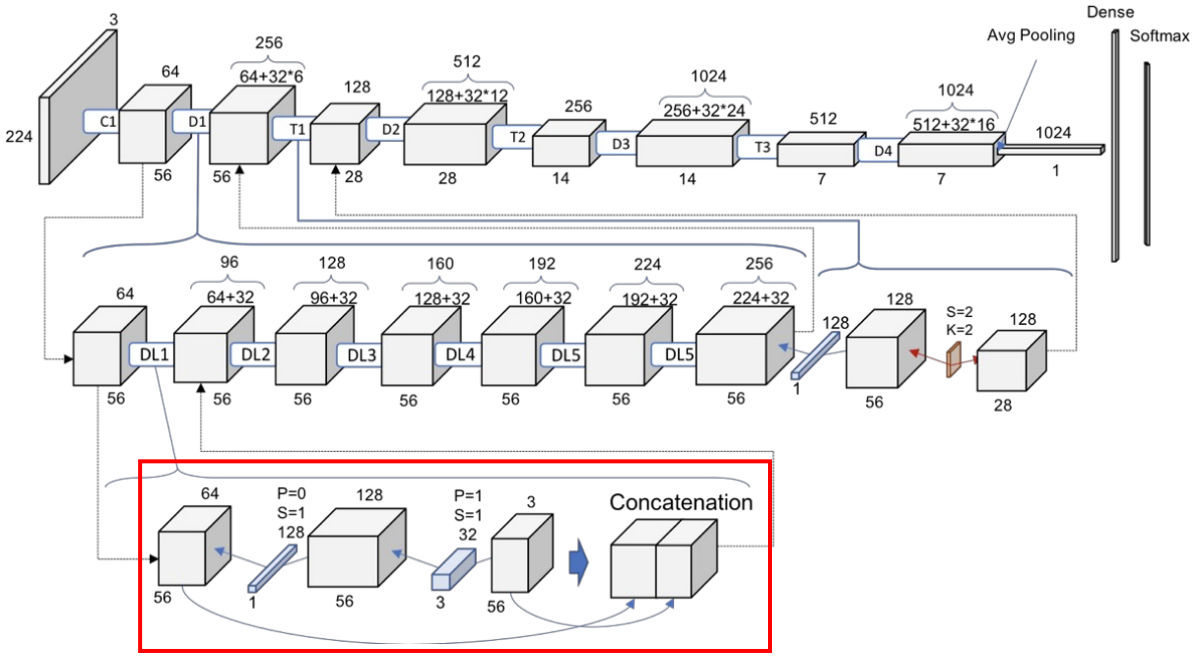
$$X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$$

```python
# 3. Dense Block (1)
for i in range(6) :
    x_l = layers.Conv2D(k * 4, (1, 1), strides=1, padding='same')(x)    # 56x56x128
    x_l = layers.BatchNormalization()(x_l)
    x_l = layers.Activation('relu')(x_l)

    x_l = layers.Conv2D(k, (3, 3), strides=1, padding='same')(x_l)  # 56x56x32
    x_l = layers.BatchNormalization()(x_l)
    x_l = layers.Activation('relu')(x_l)

    x = layers.Concatenate()([x, x_l])  # 96 -> 128 -> 160 -> 192 -> 224 -> 256
```

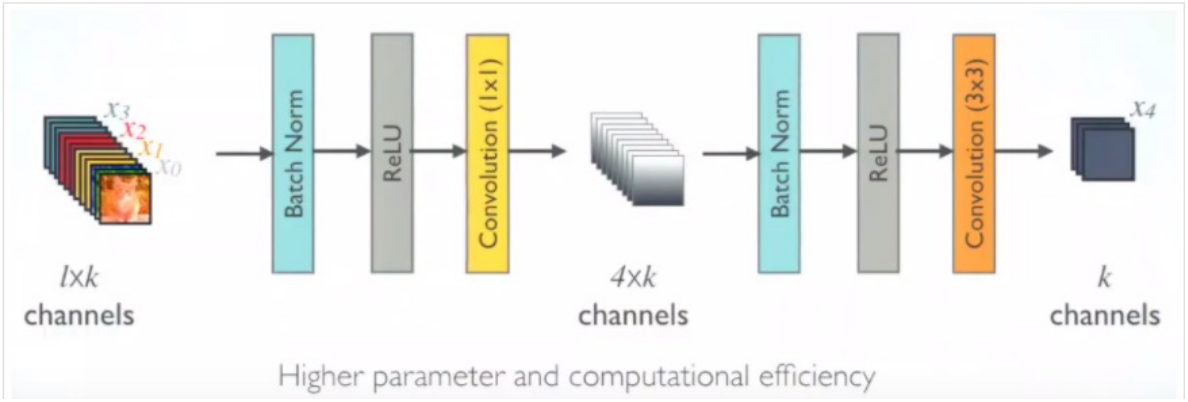
Layers	Output Size	DenseNet-121
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$

$X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$

Bottleneck Layer

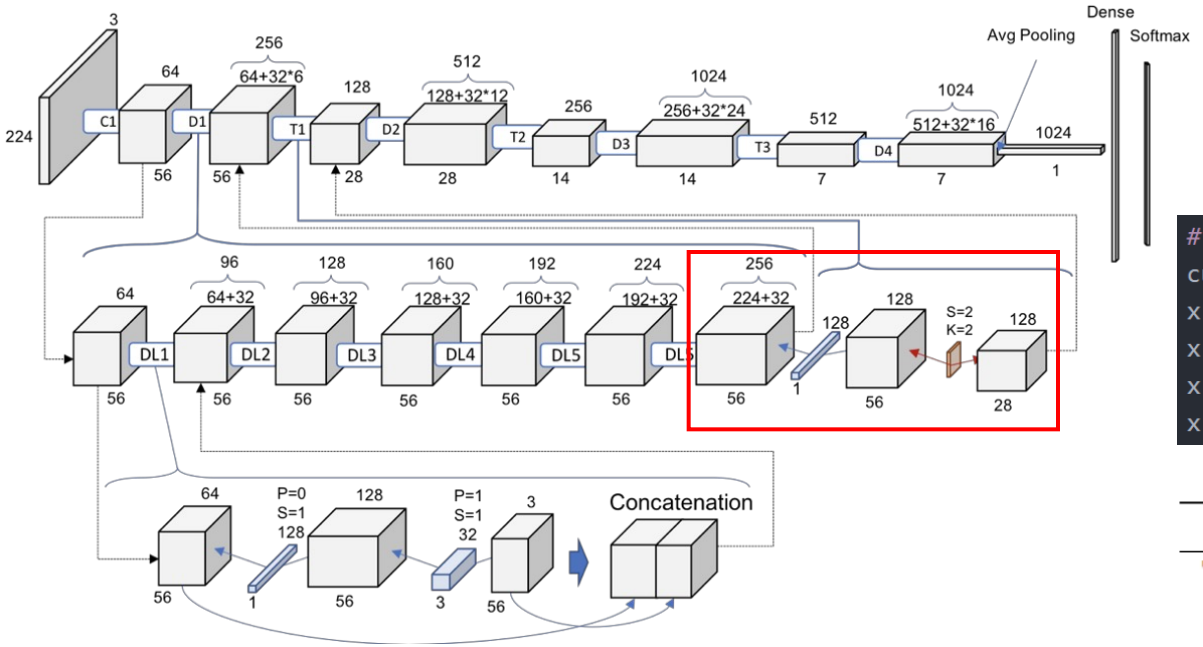



```python
# 3. Dense Block (1)
for i in range(6) :
    x_l = layers.Conv2D(k * 4, (1, 1), strides=1, padding='same')(x)    # 56x56x128
    x_l = layers.BatchNormalization()(x_l)
    x_l = layers.Activation('relu')(x_l)

    x_l = layers.Conv2D(k, (3, 3), strides=1, padding='same')(x_l)  # 56x56x32
    x_l = layers.BatchNormalization()(x_l)
    x_l = layers.Activation('relu')(x_l)

    x = layers.Concatenate()([x, x_l])  # 96 -> 128 -> 160 -> 192 -> 224 -> 256
```



Layers	Output Size	DenseNet-121
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$

Transition Layer

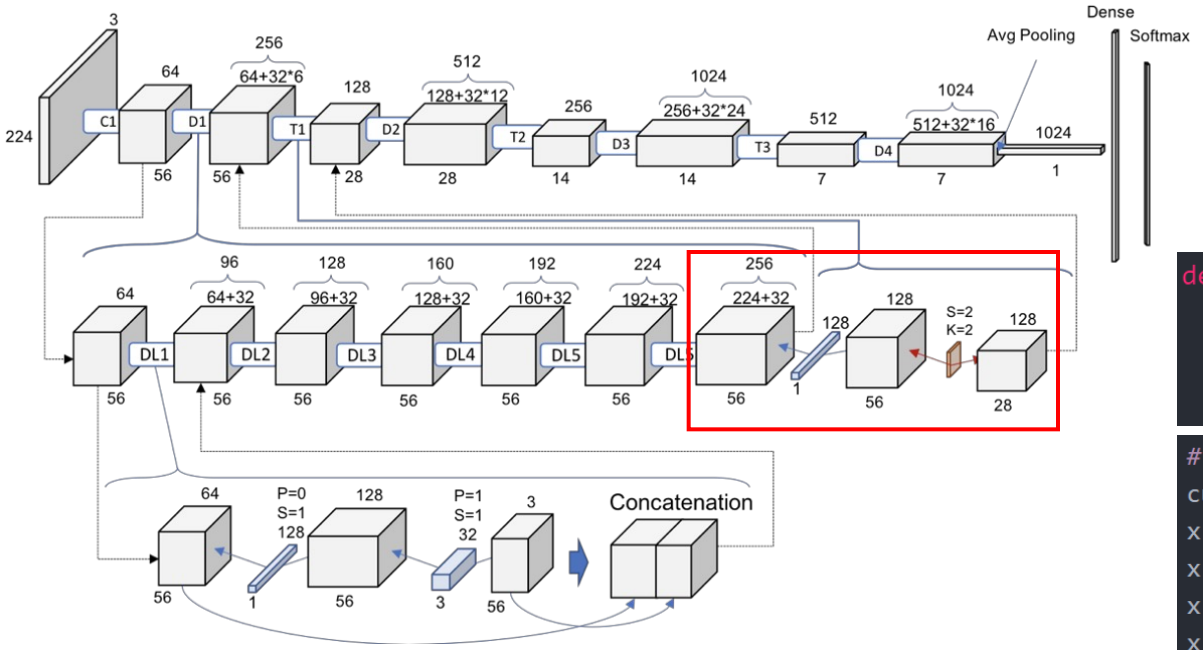


$$X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$$

```
# 4. Transition Layer (1)
current_shape = int(x.shape[-1]) # 56x56x256
x = layers.Conv2D(int(current_shape * compression), (1, 1), strides=1, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.AveragePooling2D((2, 2), strides=2, padding='same')(x)   # 28x28
```

Layers	Output Size	DenseNet-121
Transition Layer (1)	56×56	1×1 conv
	28×28	2×2 average pool, stride 2

Transition Layer - Compression

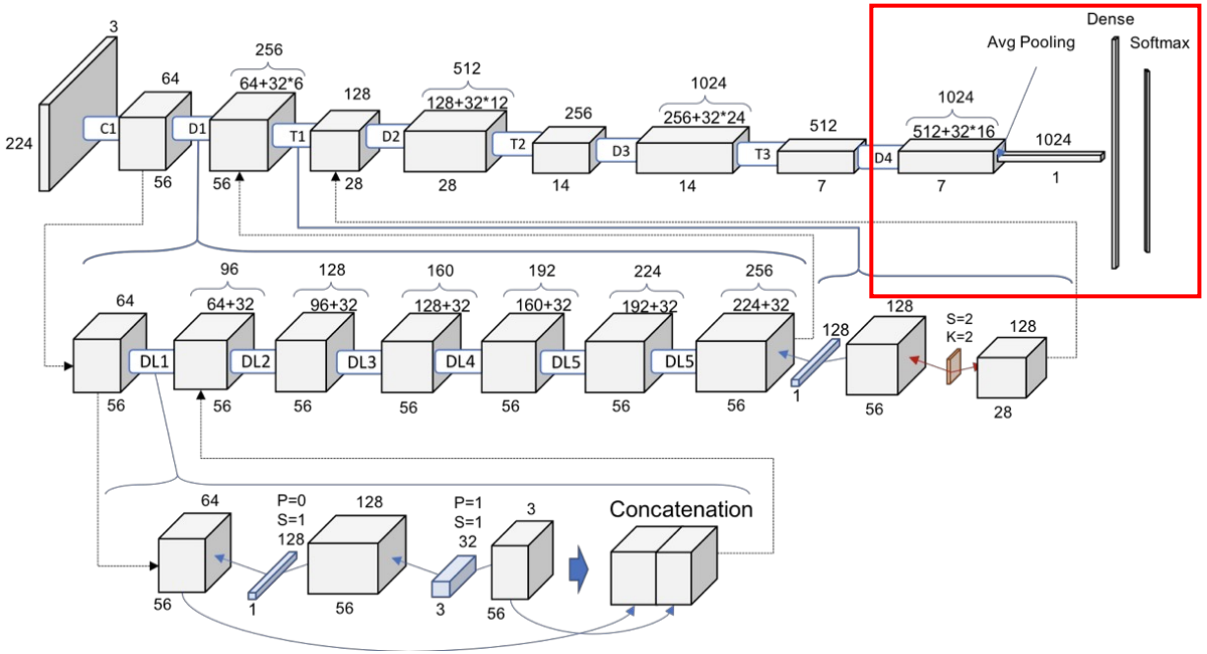


$$X_l = H_l([X_0, X_1, \ldots, X_{l-1}])$$

```
def DenseNet(x):
    # input = 224 x 224 x 3
    k = 32  # Grow Rate
    compression = 0.5  # compression factor
```

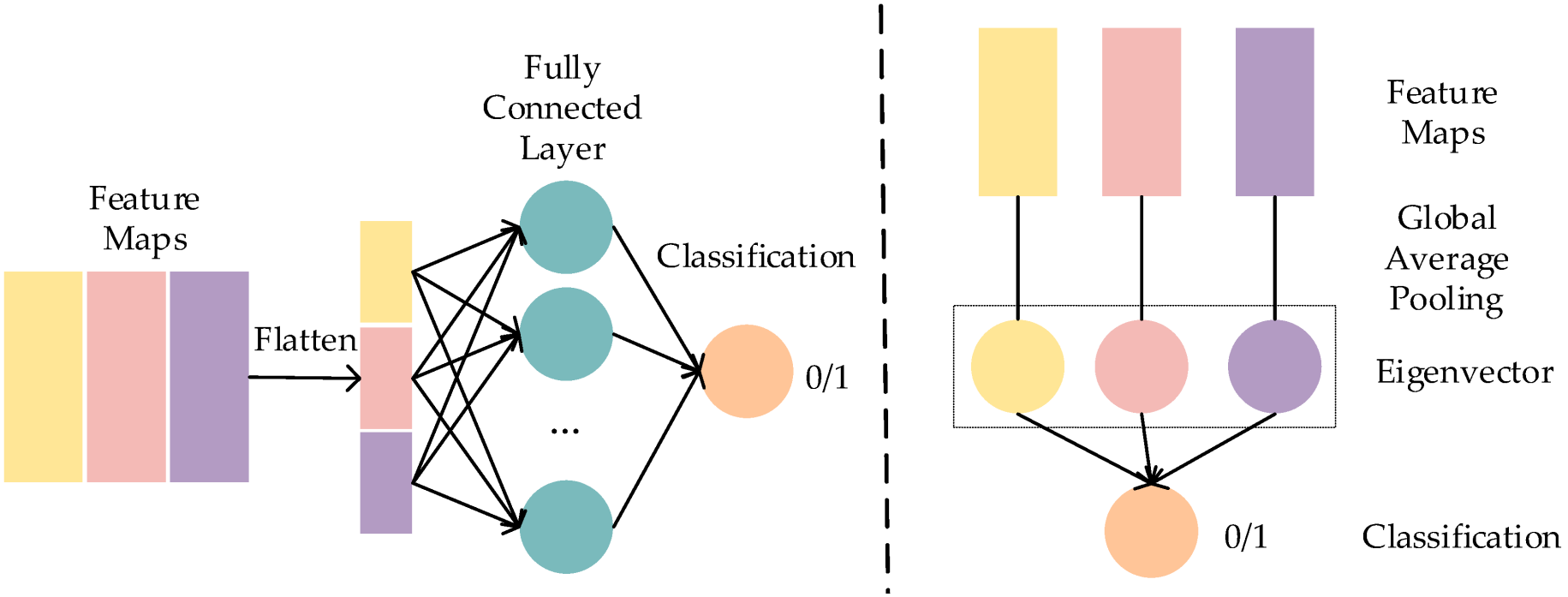
```
# 4. Transition Layer (1)
current_shape = int(x.shape[-1]) # 56x56x256
x = layers.Conv2D(int(current_shape * compression), (1, 1), strides=1, padding='same')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.AveragePooling2D((2, 2), strides=2, padding='same')(x)   # 28x28
```

Classification Layer



Layers	Output Size	DenseNet-121
Classification Layer	1×1	7×7 global average pool
		1000D fully-connected, softmax

```
# 10. Classification Layer
x = layers.GlobalAveragePooling2D()(x)
# classes = 2 (softmax)
x = layers.Dense(2, activation='softmax')(x)
```


Feature
Maps
Flatten
Fully
Connected
Layer
Classification
0/1
...
Feature
Maps
Global
Average
Pooling
Eigenvector
0/1
Classification

Dataset & Training

Dataset

- CIFAR – CIFAR10, CIFAR100을 사용하여 각각 10개의 클래스 100개의 클래스를 사용
- SVHN – 거리 뷰 집 번호 데이터 셋
- ImageNet – ILSVRC 2012 분류 데이터셋 사용

Training

- SGD사용
- CIFAR10 – epoch 300, batch size 64, lr 0.1, epoch가 50%일 때 lr*0.1
- SVHN – epoch 40, batch size 64, lr0.1, epoch가 75%일 때 lr*0.1
- ImageNet – epoch 90, batch size 256, lr 0.1, epoch가 30, 60일 때 lr*0.1
- 10~4 weight decay와 0,9 Nesteroy momentum을 사용
- Dropout rate = 0.2

Classification Result 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	**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**	-

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.

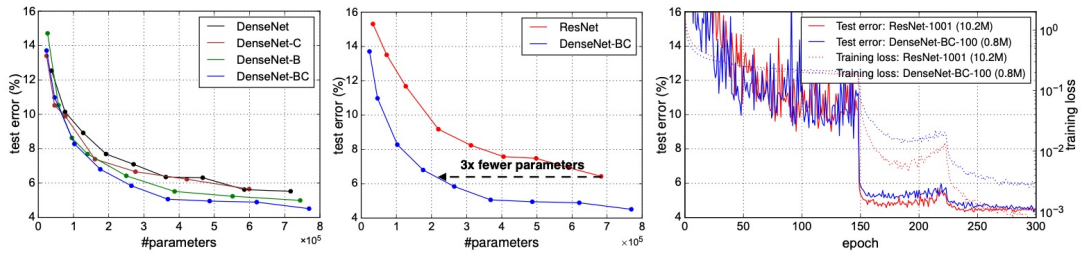


Figure 4: *Left:* Comparison of the parameter efficiency on C10+ between DenseNet variations. *Middle:* Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. *Right:* Training 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.

Classification Result on ImageNet

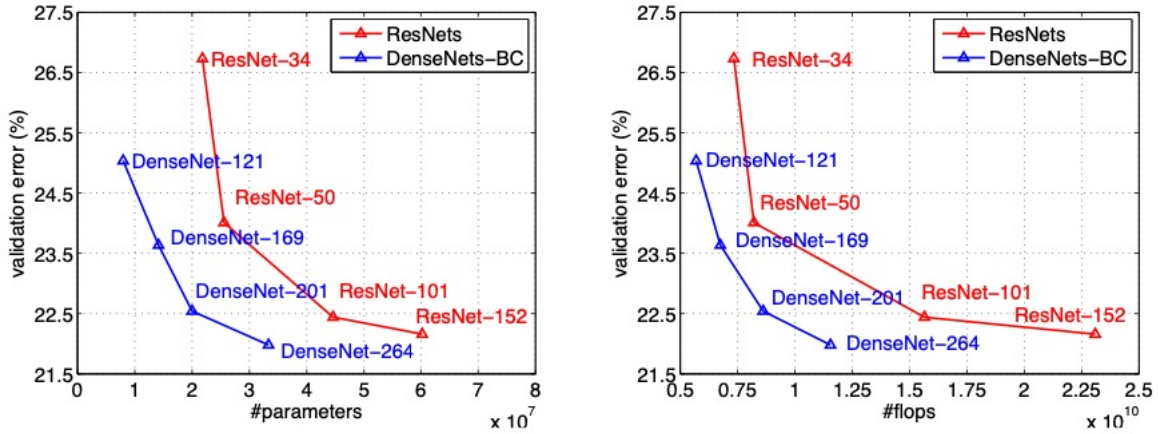


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

플롭스(**FLOPS**, **FL**oating point **O**perations **P**er **S**econd)는 컴퓨터의 성능을 수치로 나타낼 때 주로 사용되는 단위이다.

초당 부동소수점 연산이라는 의미로 컴퓨터가 1초동안 수행할 수 있는 부동소수점 연산의 횟수를 기준으로 삼는다. 상위 단위와 하위 단위로 국제단위계의 표준 접두어를 사용하며, 슈퍼 컴퓨터의 성능을 나타낼 경우에는 테라플롭스(1×10^{12} 플롭스)가 주로 쓰인다.

개인용 컴퓨터의 CPU 성능의 척도로 클럭의 속도 단위인 헤르츠를 주로 사용하는데, 이는 마이크로프로세서의 아키텍처의 구조에 따라 클럭당 연산 속도가 다르기 때문에 객관적인 성능을 비교할 때에는 플롭스를 사용한다.

Conclusion

- 실험결과 DenseNet은 수백개의 layer로 자연스럽게 확장하면서도 최적화 문제를 일으키지 않음
- 성능 저하나 overfitting의 징후 없이 매개변수 수가 증가함에 따라 정확도가 지속적으로 향상 됨
- 적은 수의 파라미터와 컴퓨팅만으로도 최신 성능에 도달할 수 있음