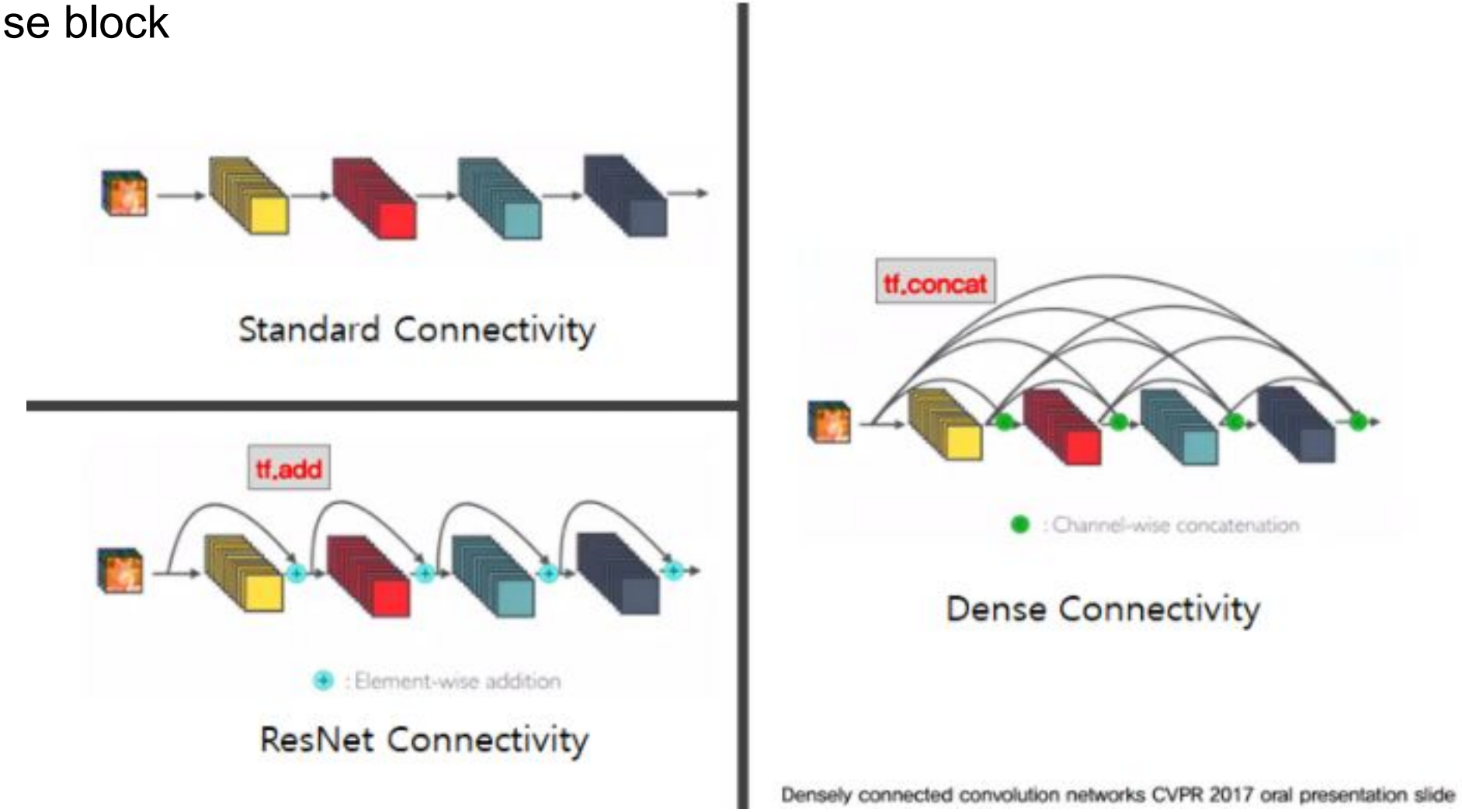
DenseNet
Densely Connected Convolutional Network

2021210088 수DA쟁이 허지혜

Network Architecture

- Dense block



Densely connected convolution networks CVPR 2017 oral presentation slide

Advantage

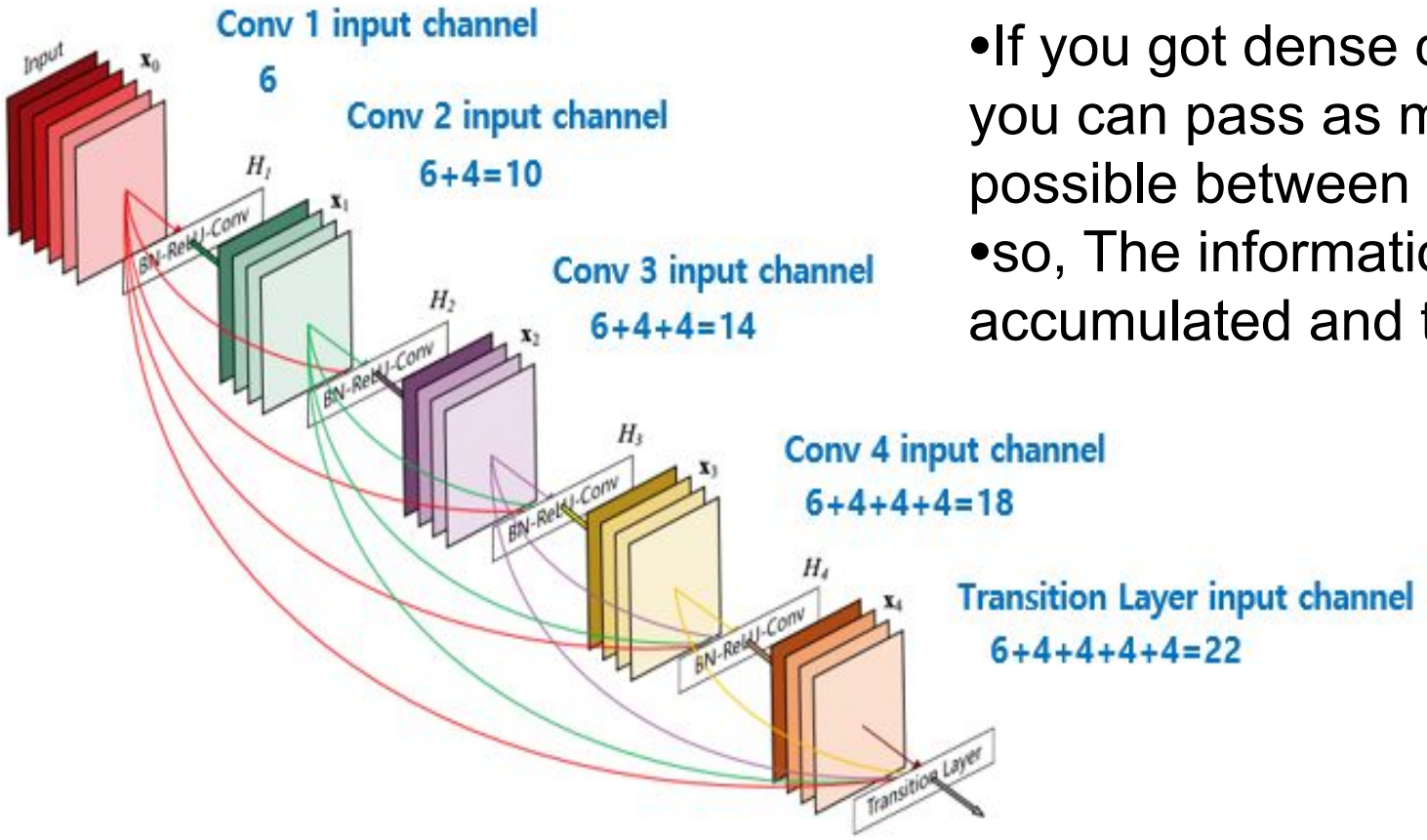


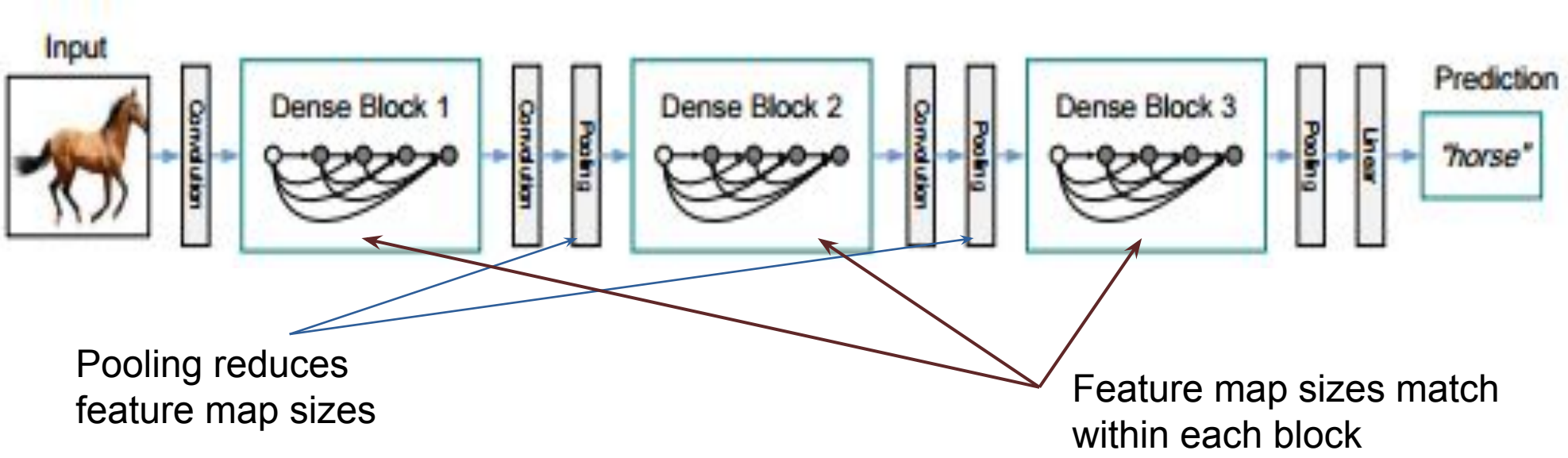
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- If you got dense connectivity pattern, you can pass as much valuable imformation as possible between layers.
- so, The information of the first layer is accumulated and transmitted to the last layers.

Advantage

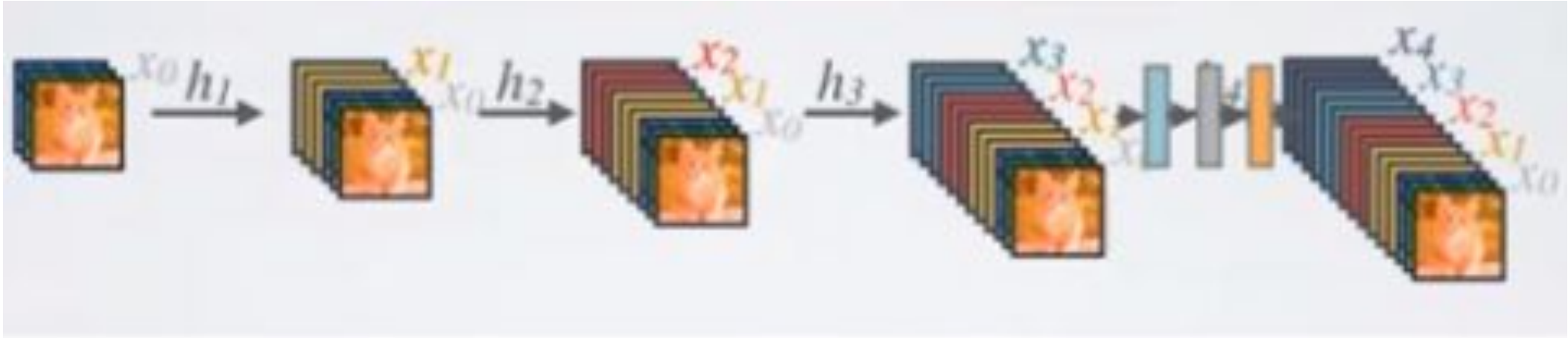
- Vanishing Gradient Improvement
- Enhanced Feature Propagation
- Feature Reuse
- Save number of parameters
- Reducing overfitting

DenseNet model architecture



[1] Dense connectivity
[2] composite function
[3] pooling layers
[4] growth rate
[5] bottleneck layers

[1] Equation



DenseNet : $x_l = H_l([x_0, x_1, \ldots, x_l])$

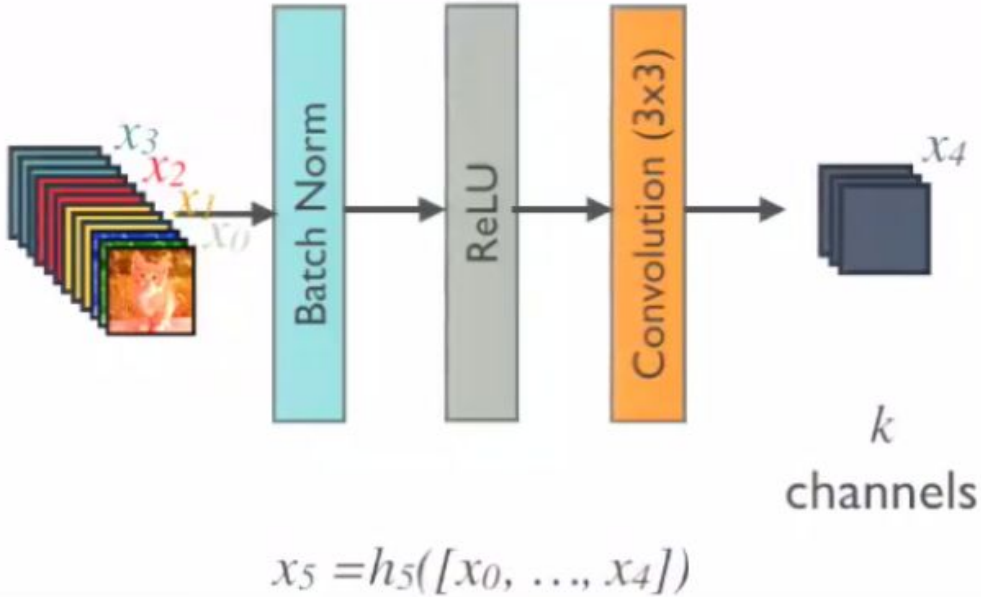
[2] Composite function

Composite function. Motivated by [12], we define $H_\ell(\cdot)$ as a composite function of three consecutive operations: batch normalization (BN) [14], followed by a rectified linear unit (ReLU) [6] and a 3×3 convolution (Conv).

Three consecutive operations

$H_\ell(\cdot)$

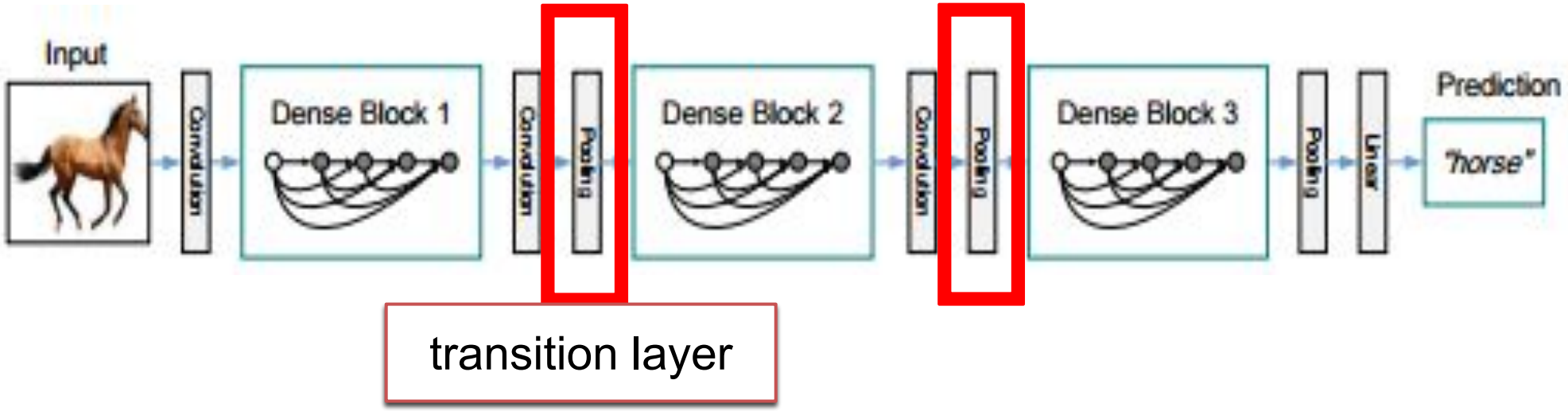
1. **Batch normalization (BN)**
2. **Rectified linear unit (ReLU)**
3. **3*3 convolution**



Densely connected convolution networks CVPR 2017 oral presentation slide

BN(Batch Normalization) + ReLU + Convolution

[3] Pooling layers

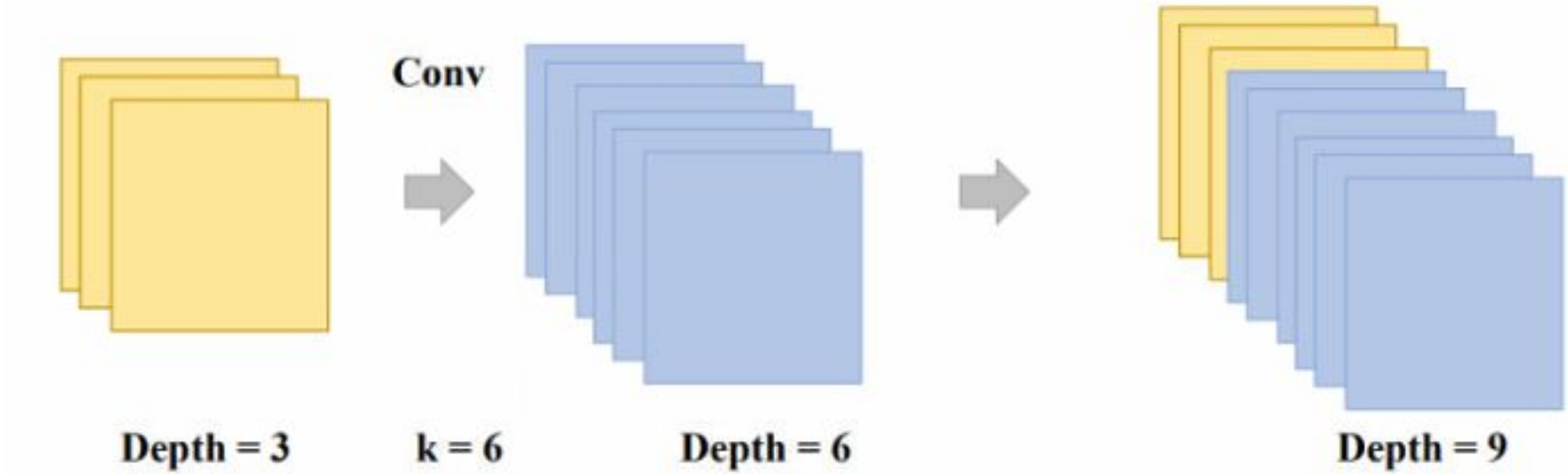


•pooling -> reduced feature map size

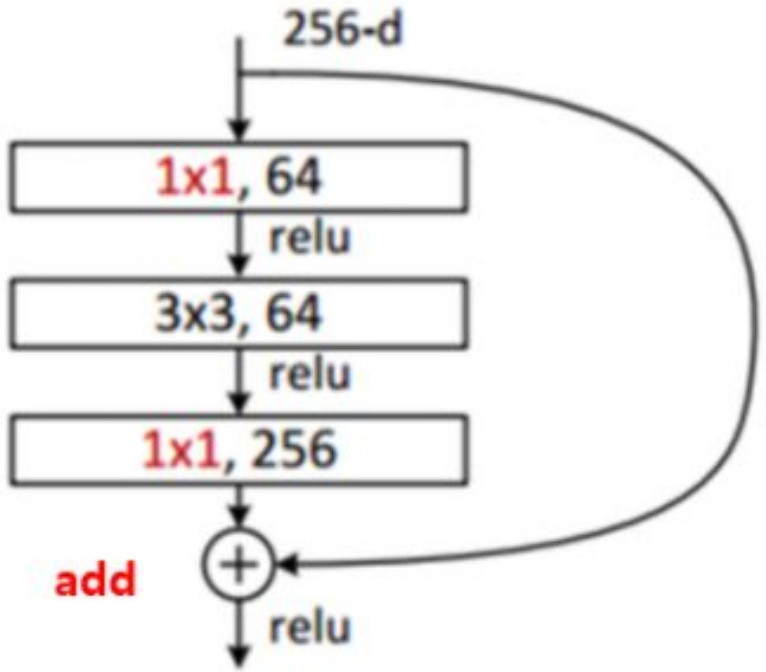
•By dividing into several dense blocks,
layers with the same feature map size are grouped into the same dense block.

[4] Growth rate

Growth rate : Dense block Hyperparameter k

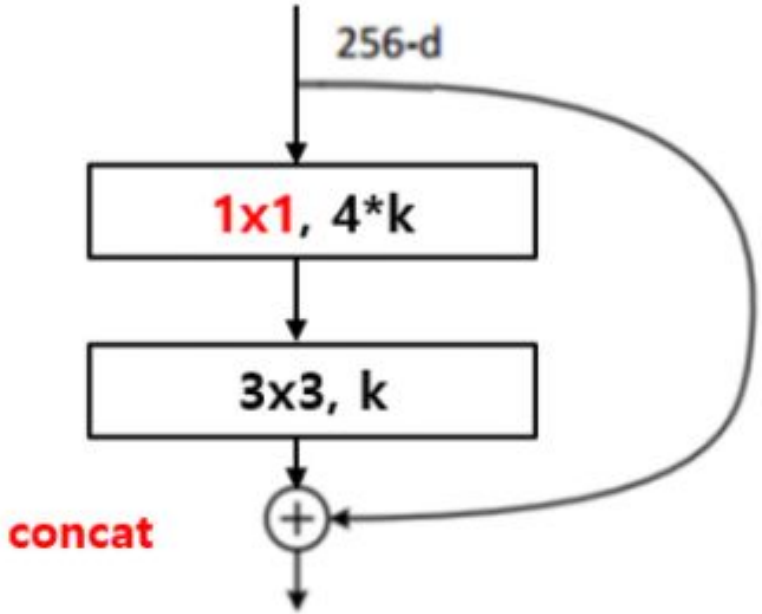


[5] Bottleneck layers



bottleneck

(for ResNet)

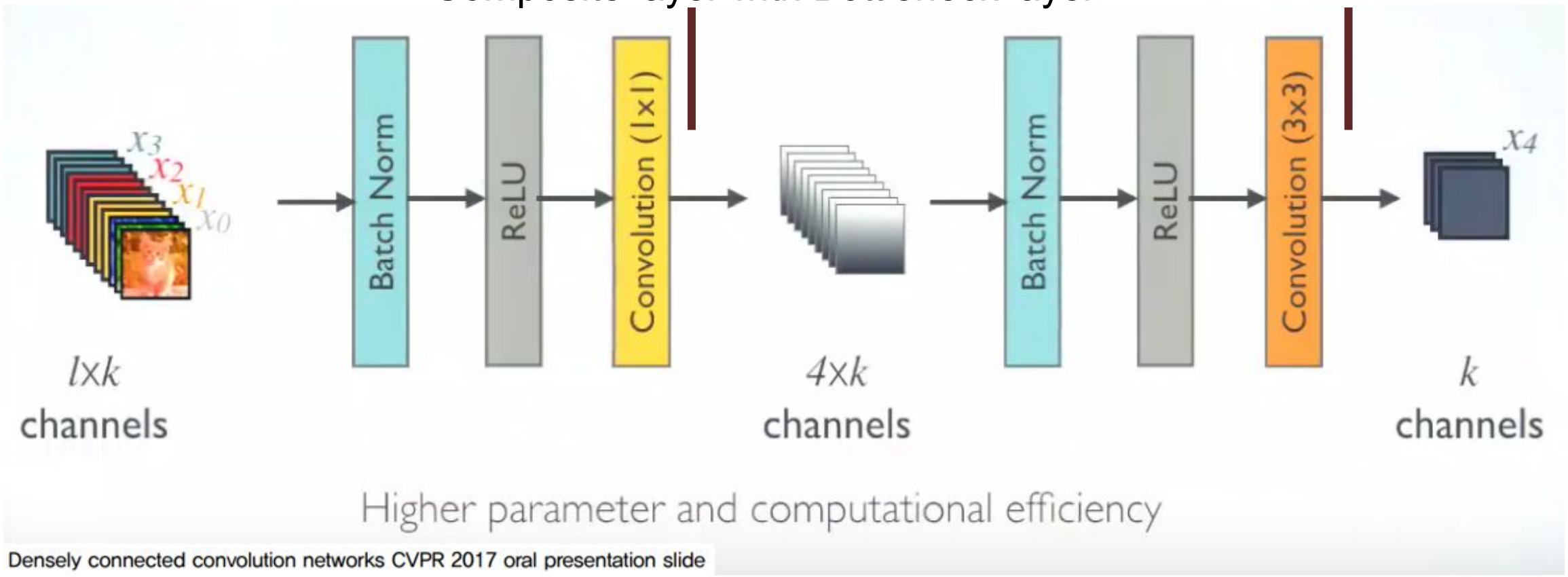


bottleneck

(for DenseNet)

[5] Bottleneck layers

Composite layer with Bottleneck layer



effect : Reduced computational complexity

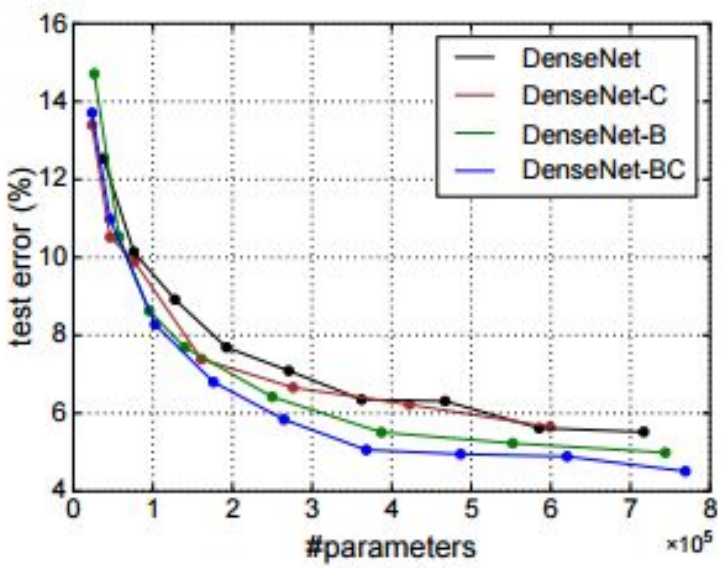
DenseNet Kinds

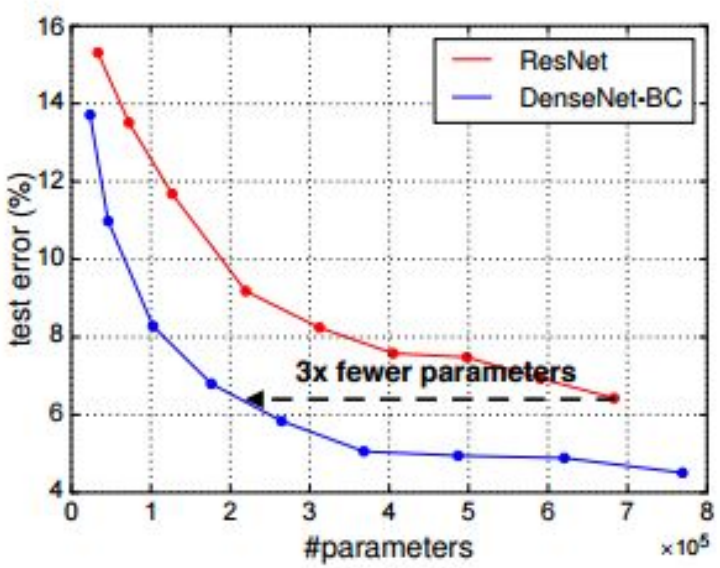
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
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			

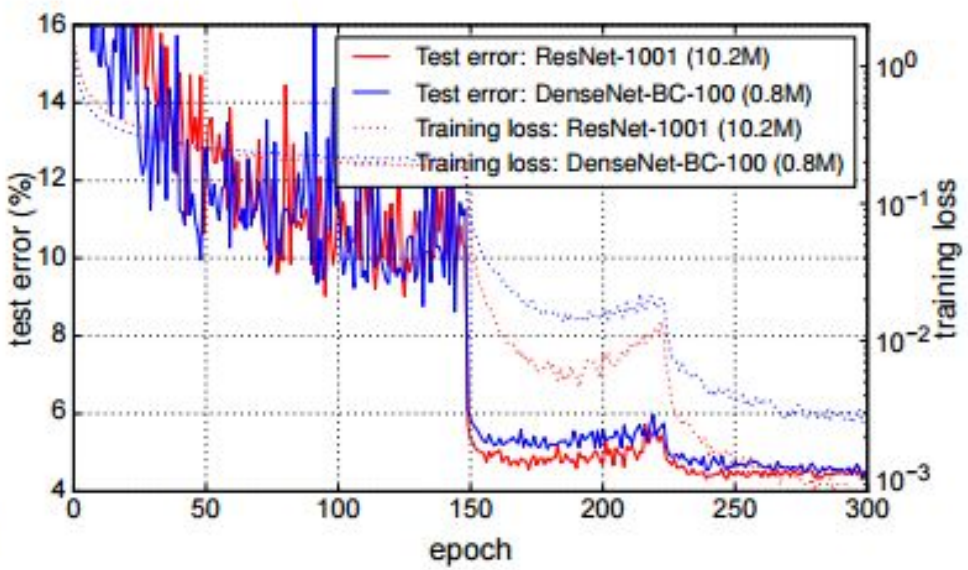
Results on CIFAR/SVHN

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**	-

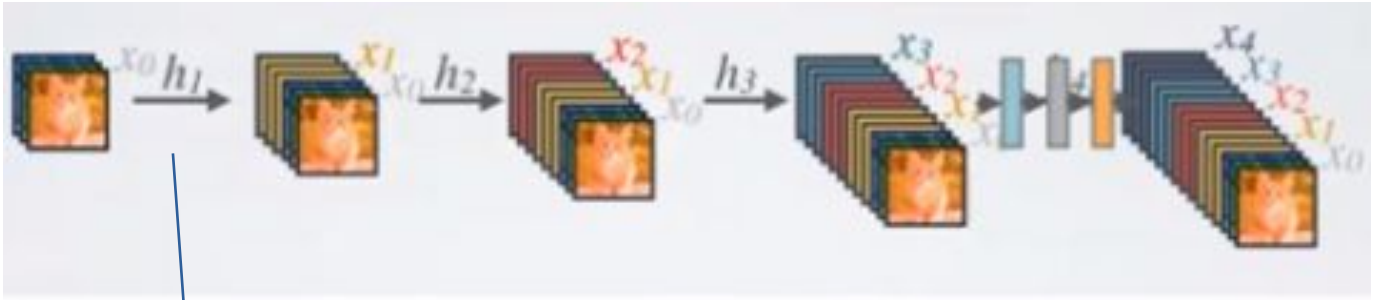
Results on ImageNet

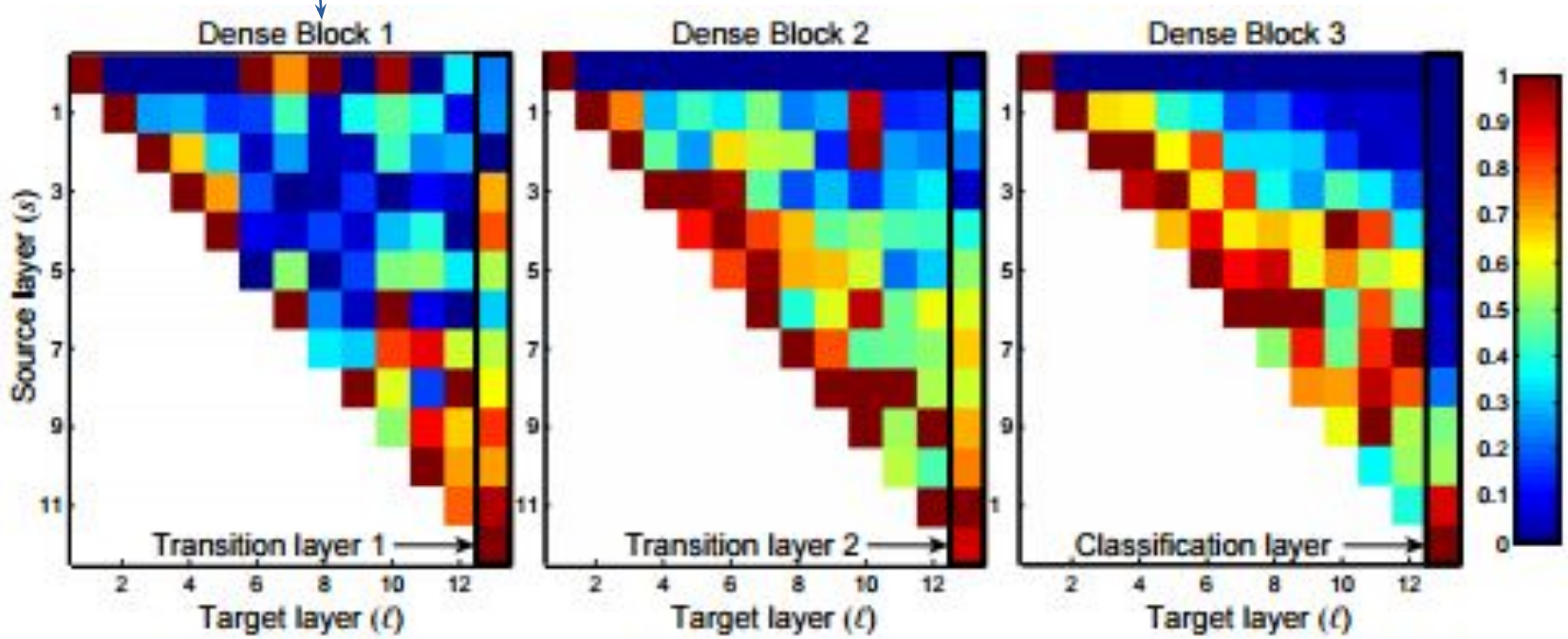






Discussion


x_0
h_1
x_1
x_0
h_2
x_2
x_1
x_0
h_3
x_3
x_2
x_1
x_0
x_4
x_3
x_2
x_1
x_0


Dense Block 1
Dense Block 2
Dense Block 3
Source layer (s)
Target layer (ℓ)
Transition layer 1
Transition layer 2
Classification layer
1
3
5
7
9
11
2
4
6
8
10
12
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Code implementation

```
def conv(name, l, channel, stride):
    return Conv2D(name, l, channel, 3, stride=stride,
                  nl=tf.identity, use_bias=False,
                  W_init=tf.random_normal_initializer(stddev=np.sqrt(2.0/9/channel)))
def add_layer(name, l):
    shape = l.get_shape().as_list()
    in_channel = shape[3]
    with tf.variable_scope(name) as scope:
        c = BatchNorm('bn1', l)
        c = tf.nn.relu(c)
        c = conv('conv1', c, self.growthRate, 1)
        l = tf.concat([c, l], 3)
    return l

def add_transition(name, l):
    shape = l.get_shape().as_list()
    in_channel = shape[3]
    with tf.variable_scope(name) as scope:
        l = BatchNorm('bn1', l)
        l = tf.nn.relu(l)
        l = Conv2D('conv1', l, in_channel, 1, stride=1, use_bias=False, nl=tf.nn.relu)
        l = AvgPooling('pool', l, 2)
    return l
```

```
def dense_net(name):
    l = conv('conv0', image, 16, 1)
    with tf.variable_scope('block1') as scope:

        for i in range(self.N):
            l = add_layer('dense_layer.{}'.format(i), l)
        l = add_transition('transition1', l)

    with tf.variable_scope('block2') as scope:

        for i in range(self.N):
            l = add_layer('dense_layer.{}'.format(i), l)
        l = add_transition('transition2', l)

    with tf.variable_scope('block3') as scope:

        for i in range(self.N):
            l = add_layer('dense_layer.{}'.format(i), l)
    l = BatchNorm('bnlast', l)
    l = tf.nn.relu(l)
    l = GlobalAvgPooling('gap', l)
    logits = FullyConnected('linear', l, out_dim=10, nl=tf.identity)

    return logits
```

https://github.com/YixuanLi/densenet-tensorflow/blob/master/cifar10-densenet.py

Reference

- https://jayhey.github.io/deep%20learning/2017/10/15/DenseNet_2/
- https://hoya012.github.io/blog/DenseNet-Tutorial-1/
- https://velog.io/@dope/%EB%85%BC%EB%AC%B8%EA%B5%AC%ED%98%84-DenseNet-By-Tensorflow-2
- https://github.com/pytorch/vision/blob/6db1569c89094cf23f3bc41f79275c45e9fcb3f3/torchvision/models/densenet.py#L126
- https://github.com/titu1994/DenseNet/blob/master/densenet.py