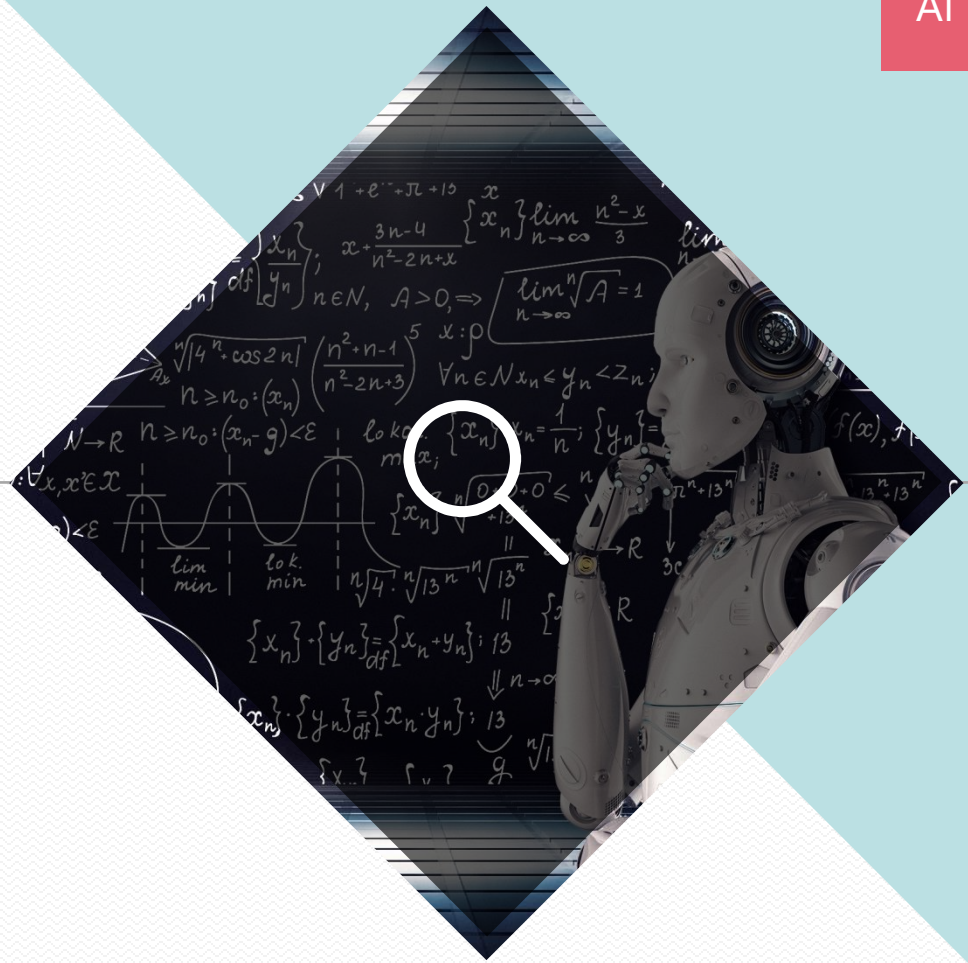
AI TRAINING SERIES

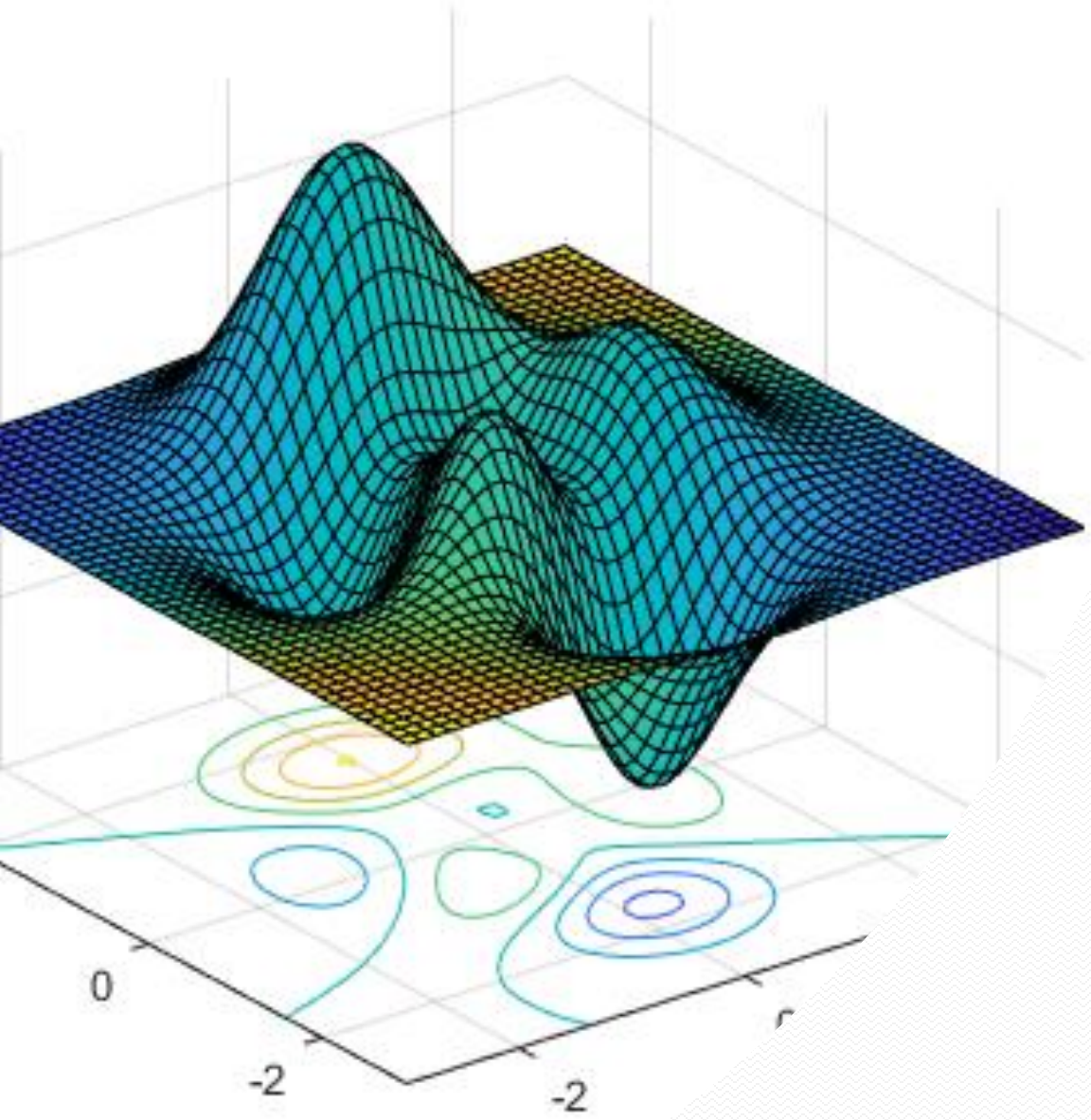


CNN

CNN & Convolution

Agenda

- *CNN*
  - *CNN*
  - *Use cases*
  - *Classification*
  - *Classifiaction Activation Map*
  - *Segmentation*
  - *Object detection*



CNN

PROBLEMS ON ANN

REQUIRES MANY PARAMETERS

28x28 image → (28×28+1)×10=7,850 ≈ 7.8K

640x480 image → (640×480+1)×10=3,072,010 ≈ 3M

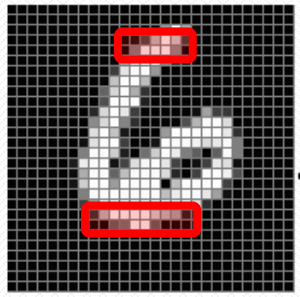
REQUIRES LOTS OF MEMORY AND COMPUTATION COST

WEIGHTS BIAS

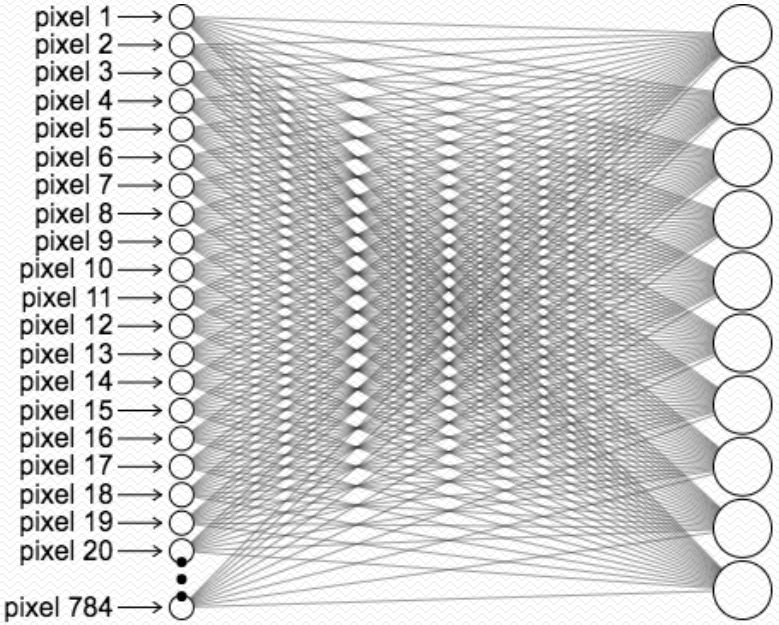
$$(N_{in} + 1) \cdot N_{out}$$

TAKES 1D FORM OF INPUT ONLY

Lose spatial information!



MISSING SPATIAL RELATIONSHIP



CNN Classification기본구조

✓ 특정 속성을 추출하는 과정후 **Flatten** 과정을 거쳐 **Fully Connected**됨
✓ 일반적인 CNN은 Convolution 연산, Activation 연산(대부분 ReLU), Pooling 연산의 반복으로구성됨 (Feature Learning)
✓ 일정 횟수 이상의 Feature Learning 과정 이후에는 Flatten 과정을 통해 이미지가 1차원의 벡터로 변환됨

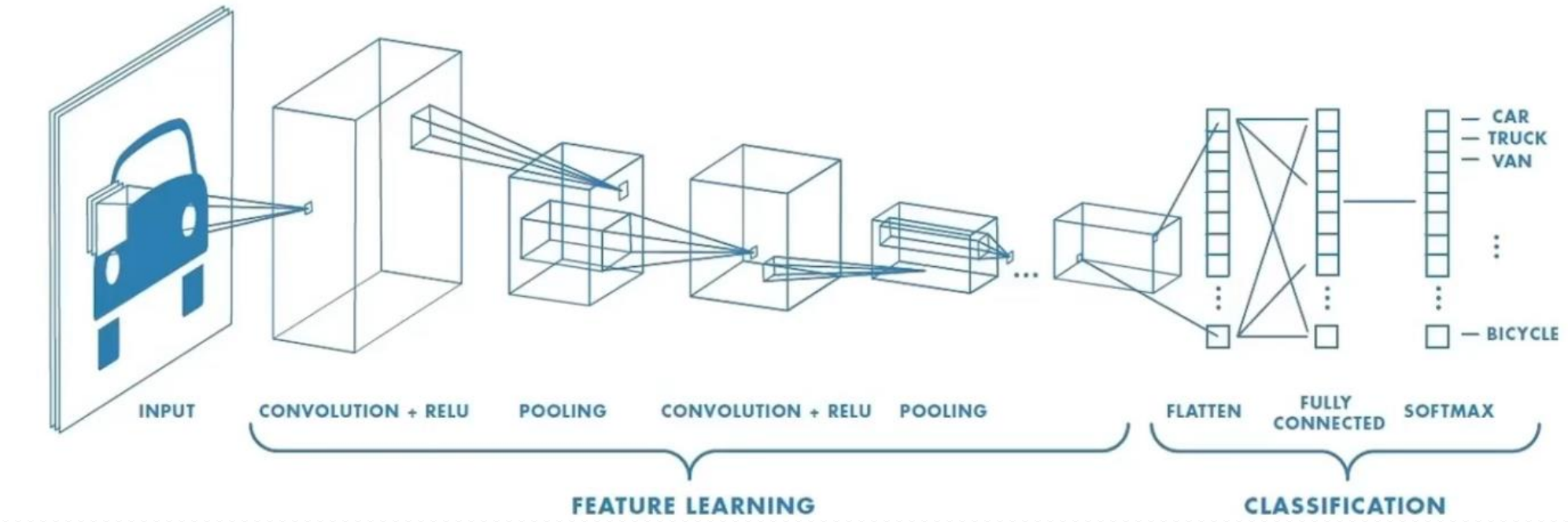


Image presentation

√ 컬러 이미지는 3차원의 Tensor로 표현됨 : Width X Height X 3 (RGB)

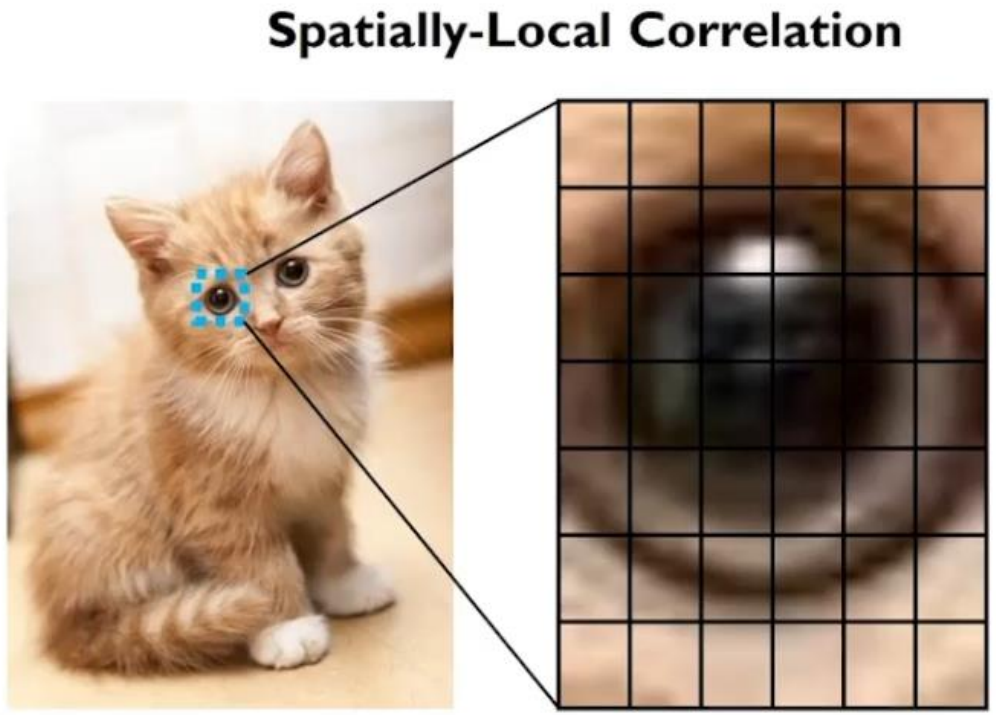

```
> pskang[1:10, 1:10, 1]
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.2784314 0.2784314 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2705882 0.2862745 0.2901961
 [2,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2666667 0.2705882
 [3,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2705882 0.2745098
 [4,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529 0.2823529
 [5,] 0.2705882 0.2705882 0.2705882 0.2745098 0.2745098 0.2784314 0.2784314 0.2784314 0.2784314 0.2784314
 [6,] 0.2705882 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2784314 0.2823529 0.2784314
 [7,] 0.2705882 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2784314 0.2941176 0.2862745
 [8,] 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2745098 0.2901961 0.2823529
 [9,] 0.2745098 0.2705882 0.2745098 0.2784314 0.2823529 0.2823529 0.2745098 0.2666667 0.2784314 0.2784314
[10,] 0.2784314 0.2784314 0.2784314 0.2823529 0.2862745 0.2862745 0.2784314 0.2745098 0.2745098 0.2745098
```

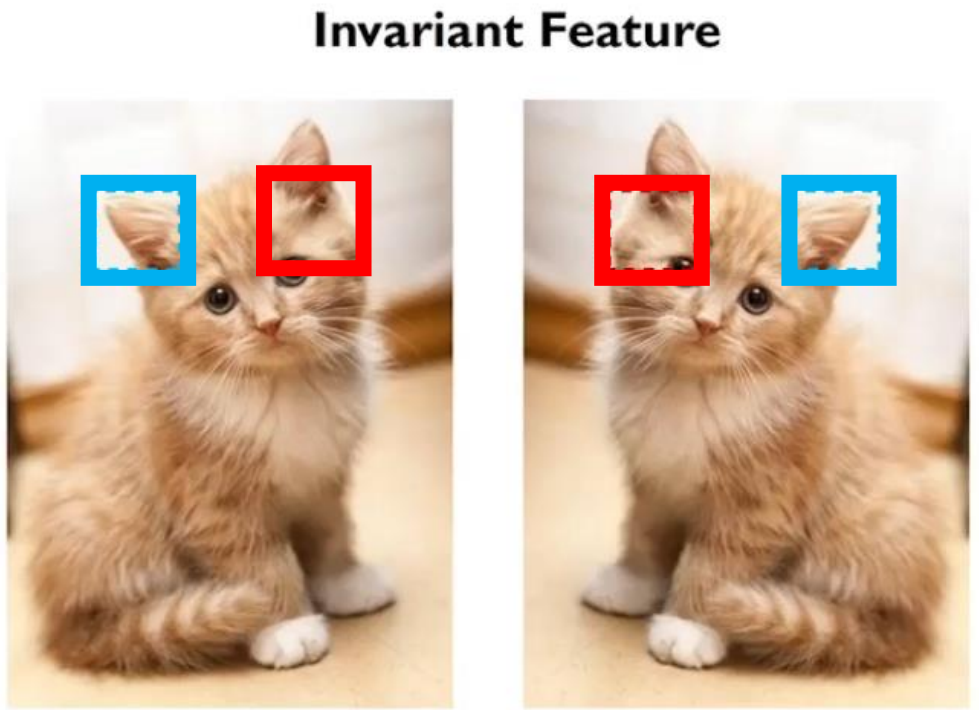
```
> pskang[1:10, 1:10, 2]
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.2980392 0.2980392 0.2941176 0.2941176 0.2941176 0.2941176 0.2901961 0.2901961 0.2980392 0.3019608
 [2,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2784314 0.2823529
 [3,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2823529 0.2862745
 [4,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2941176
 [5,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2941176 0.2980392 0.2980392 0.2980392 0.2901961 0.2901961
 [6,] 0.2901961 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.2980392 0.2941176 0.2901961
 [7,] 0.2901961 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2980392 0.3058824 0.2980392
 [8,] 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.2941176 0.3019608 0.2941176
 [9,] 0.2862745 0.2823529 0.2862745 0.2901961 0.2941176 0.2941176 0.2862745 0.2784314 0.2901961 0.2901961
[10,] 0.2901961 0.2901961 0.2901961 0.2941176 0.2980392 0.2980392 0.2901961 0.2862745 0.2862745 0.2862745
```

```
> pskang[1:10, 1:10, 3]
           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]     [,10]
 [1,] 0.3215686 0.3215686 0.3176471 0.3176471 0.3176471 0.3176471 0.3137255 0.3137255 0.3254902 0.3294118
 [2,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3058824 0.3098039
 [3,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3098039 0.3137255
 [4,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3215686
 [5,] 0.3137255 0.3137255 0.3137255 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3176471 0.3176471
 [6,] 0.3137255 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3215686 0.3215686 0.3176471
 [7,] 0.3137255 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3215686 0.3333333 0.3254902
 [8,] 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3176471 0.3294118 0.3215686
 [9,] 0.3058824 0.3019608 0.3058824 0.3098039 0.3137255 0.3137255 0.3058824 0.2980392 0.3098039 0.3098039
[10,] 0.3098039 0.3098039 0.3098039 0.3137255 0.3176471 0.3176471 0.3098039 0.3058824 0.3058824 0.3058824
```

이미지 데이터의 특징

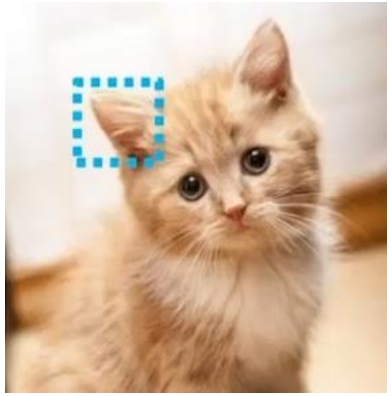
√ 인접 픽셀간 높은 상관관계(spatially-local correlation)
✓ 이미지의 부분적 특성(e.g, 눈, 귀)은 고정된 위치에 등장하지 않음(feature invariance)





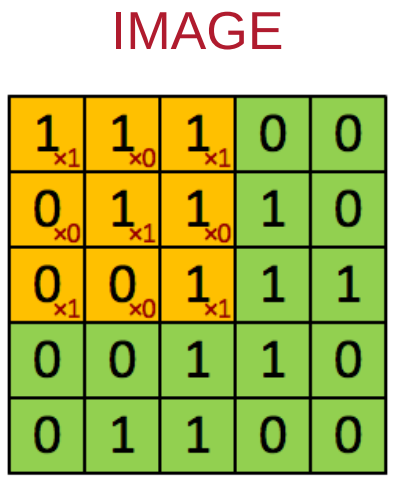
합성곱연산(Convolution)

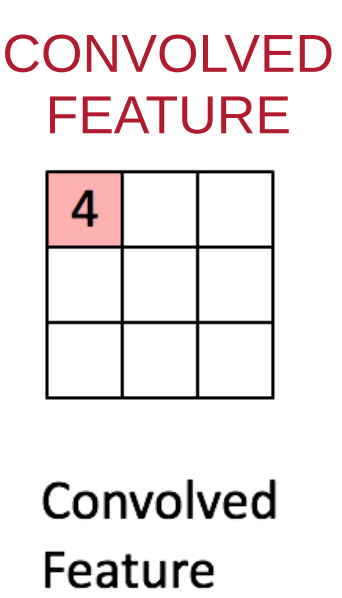
✓ **Spatially-local correlation을 고려하기 위한 Sparse connection 구성**
✓ **인접한 변수만을 이용하여 새로운 Feature 생성**
✓ **Invariant feature를 추출하기 위해 Shared weight 개념 이용·**
✓ **특정 특질을 추출하기 위한 도구로서 같은 대상 크기에는 위치가 다르더라도 동일한 Weight 적용**



Sparse connection
전체 픽셀이 아닌 일부 픽셀만
가중치로 연결하여 연산수행

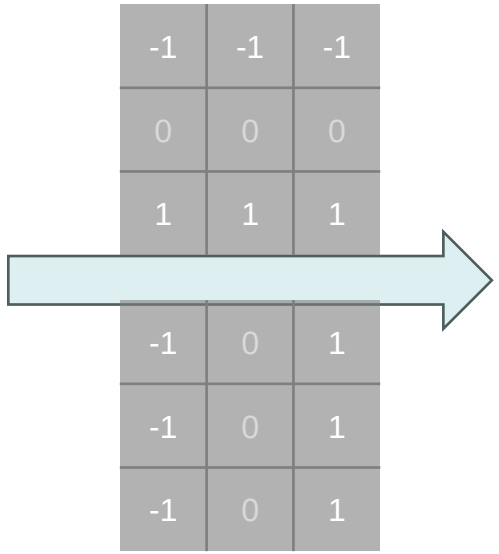
Shared weight
동일한 Filter는 대상
영역에상관없이 같은
가중치를사용하여 연산 수행

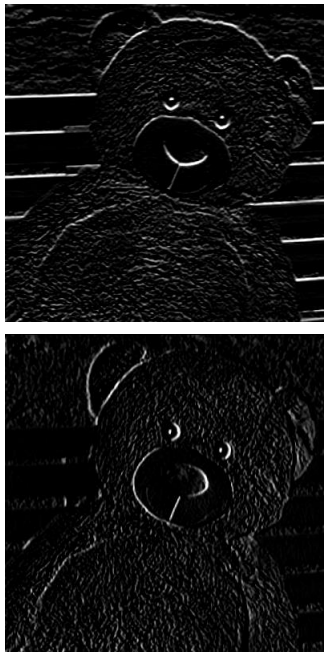




Feature Engineering

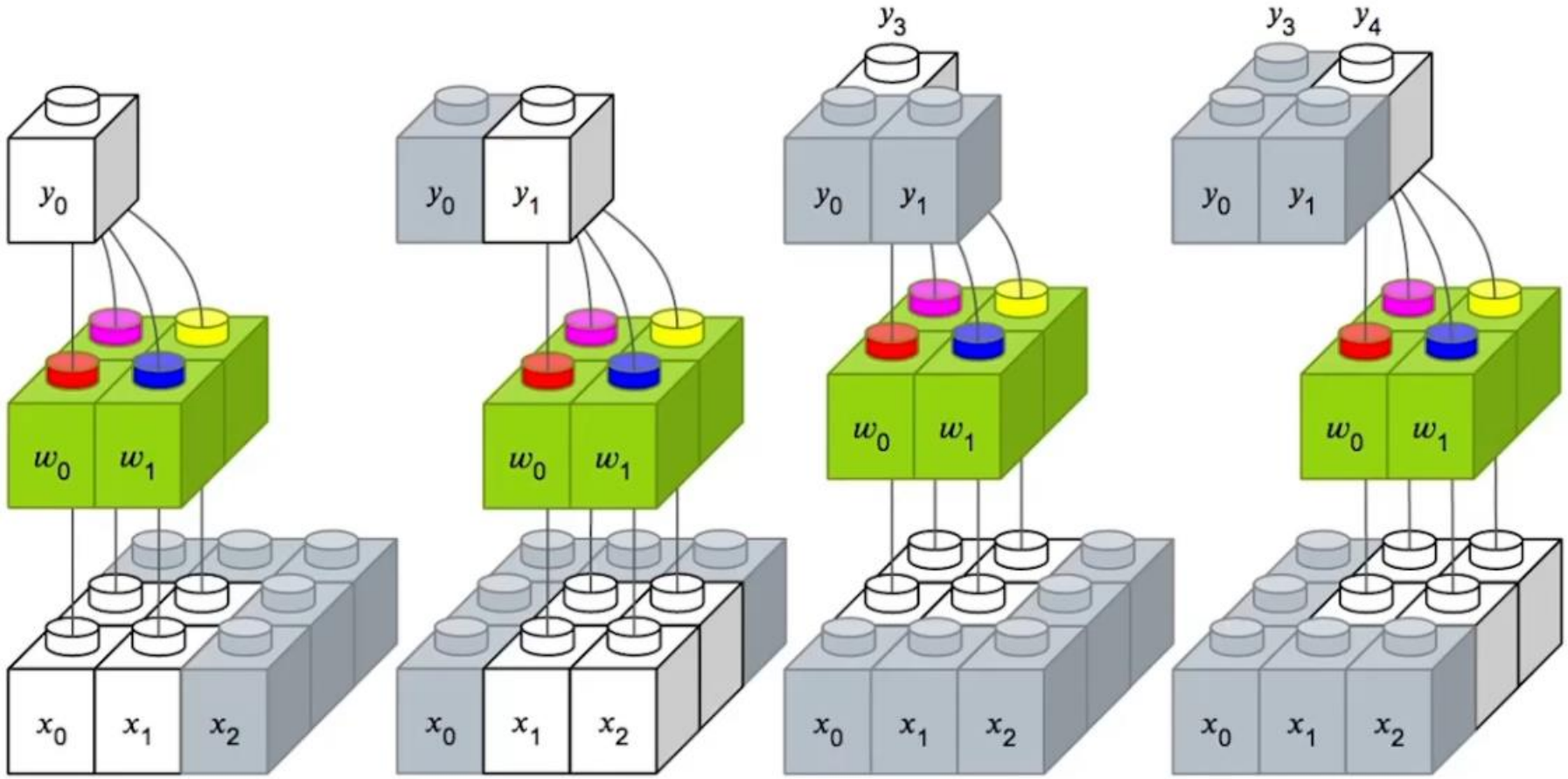
✓ **Image Convolution(Filter, Kernel)**
✓ **특정 속성을 탐지하는데 사용하는 matrix (예 : edge detection)**
✓ **CNN 등장 전 Computer Vision을 연구하는 연구자들의 핵심 주제는 "특정 특질을 잘 추출 Feature를 어떻게 설계해야 하는가?"였음**





https://www.software.ac.uk/blog/2016-10-03-how-photo-playboy-became-part-scientific-culture

합성곱연산(Convolution)



https://tykimos.github.io/2017/01/27/CNN_Layer_Talk/

CONVOLUTION

MATHEMATICAL OPERATION

Slides one function over another
e.g., Photo filters

IMAGE

$1_{\times 1}$	$1_{\times 0}$	$1_{\times 1}$	0	0
$0_{\times 0}$	$1_{\times 1}$	$1_{\times 0}$	1	0
$0_{\times 1}$	$0_{\times 0}$	$1_{\times 1}$	1	1
0	0	1	1	0
0	1	1	0	0

CONVOLVED FEATURE

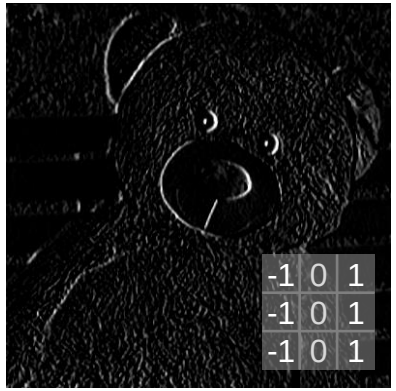
4		

ORIGINAL

SHARPEN

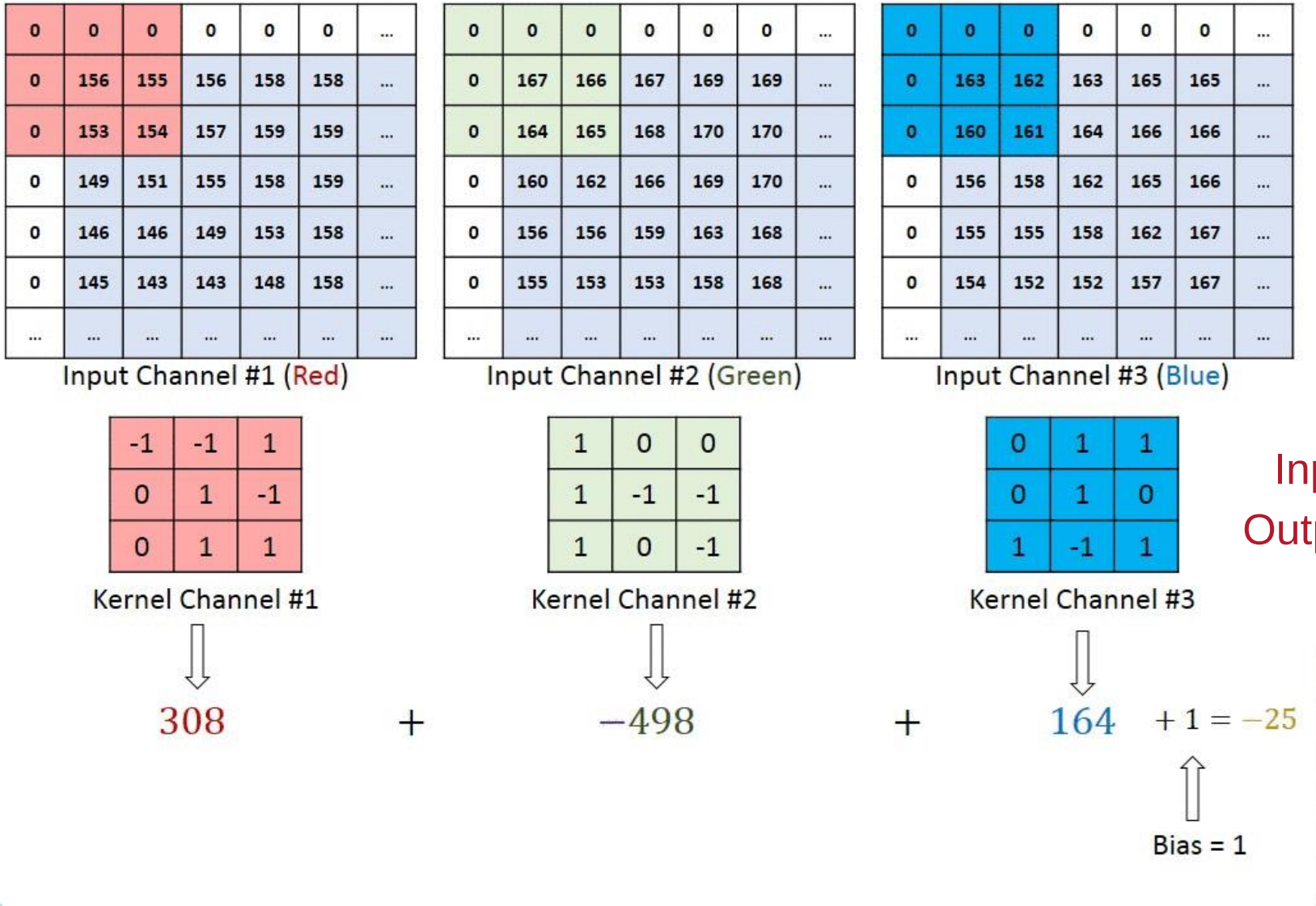
EMBOSS

EXTRACT HORIZONTAL LINE



EXTRACT VERTICAL LINE

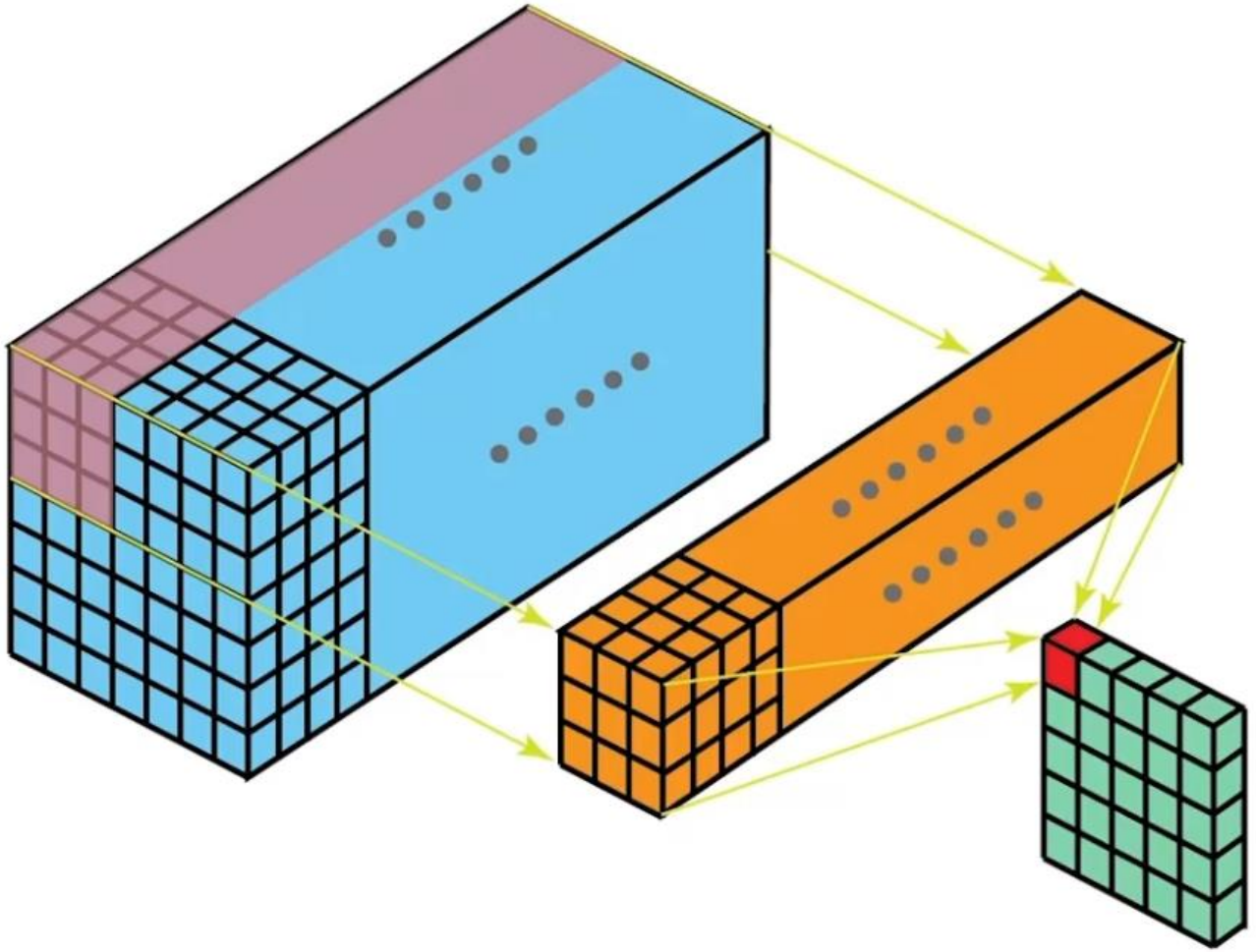
CONVOLUTION : MULTI-CHANNEL



Input CH = Filter CH

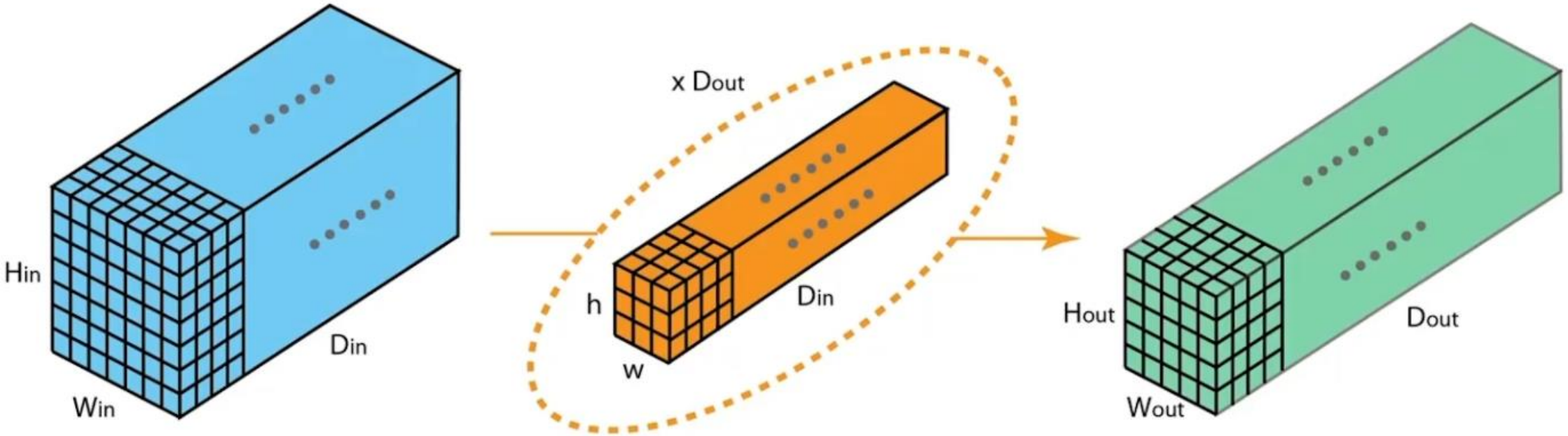
Output CH = # of filters

합성곱연산(Convolution)



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

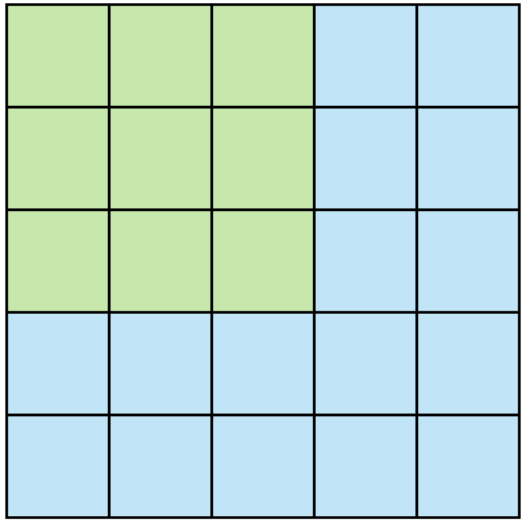
합성곱연산(Convolution)



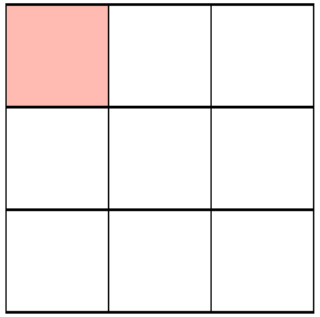
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

합성곱연산(Convolution: Stride)

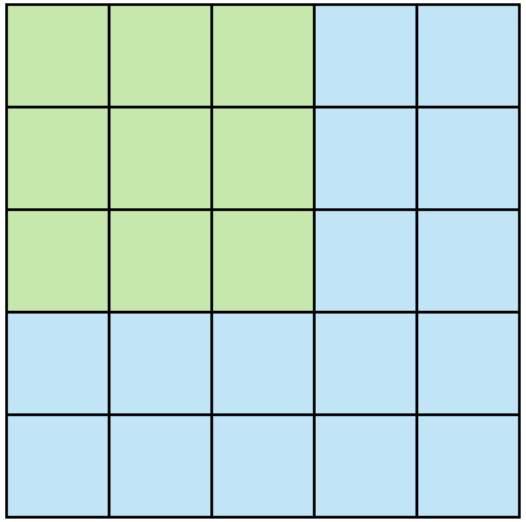
✓ 이슈 : 한번에 한 칸씩 이동하면 너무 오래 걸리지 않을까?
✓ 해결책 : Filter가 한번에 여러 칸 이동하도록 허용하자(Stride)



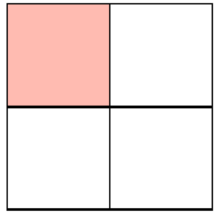
Stride 1



Feature Map



Stride 2



Feature Map

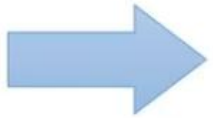
https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

합성곱연산(Convolution: Padding)

✓ 문제점 : 가장자리에 있는 픽셀들은 중앙에 위치한 픽셀들에 비해 **Convolution** 연산이 적게 수행됨
✓ 해결책 : **Padding (원래 이미지 테두리에 0의 값을 갖는 pad를 추가)**

3	1	1	2	8	4
1	0	7	3	2	6
2	3	5	1	1	3
1	4	1	2	6	5
3	2	1	3	7	2
9	2	6	2	5	1

Padding with size 1

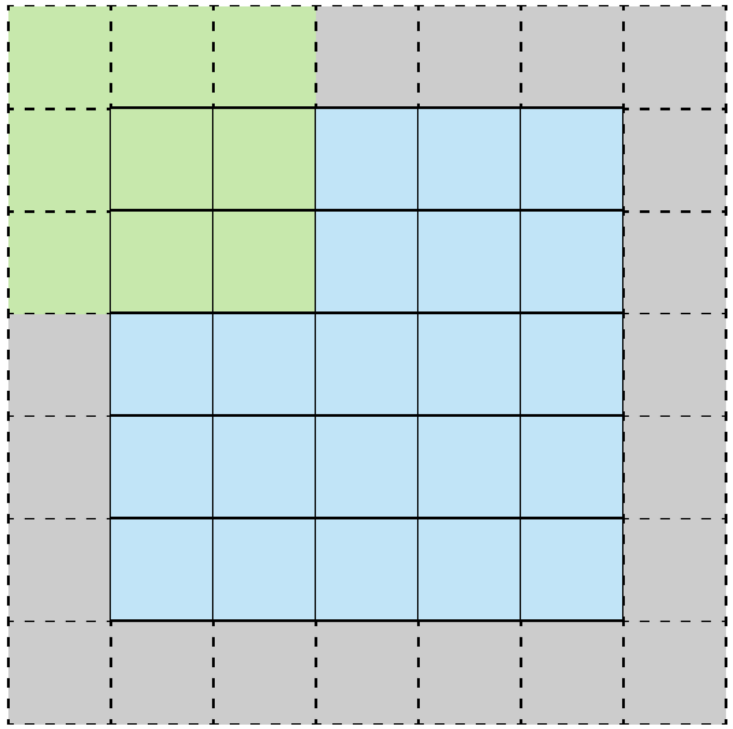


0	0	0	0	0	0	0	0
0	3	1	1	2	8	4	0
0	1	0	7	3	2	6	0
0	2	3	5	1	1	3	0
0	1	4	1	2	6	5	0
0	3	2	1	3	7	2	0
0	9	2	6	2	5	1	0
0	0	0	0	0	0	0	0

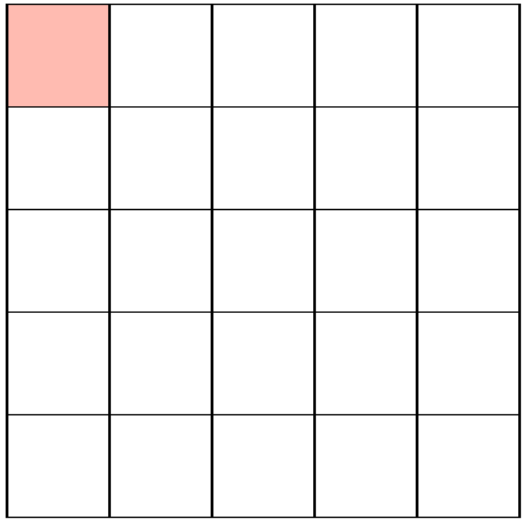
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

합성곱연산(Convolution)

Stride & padding



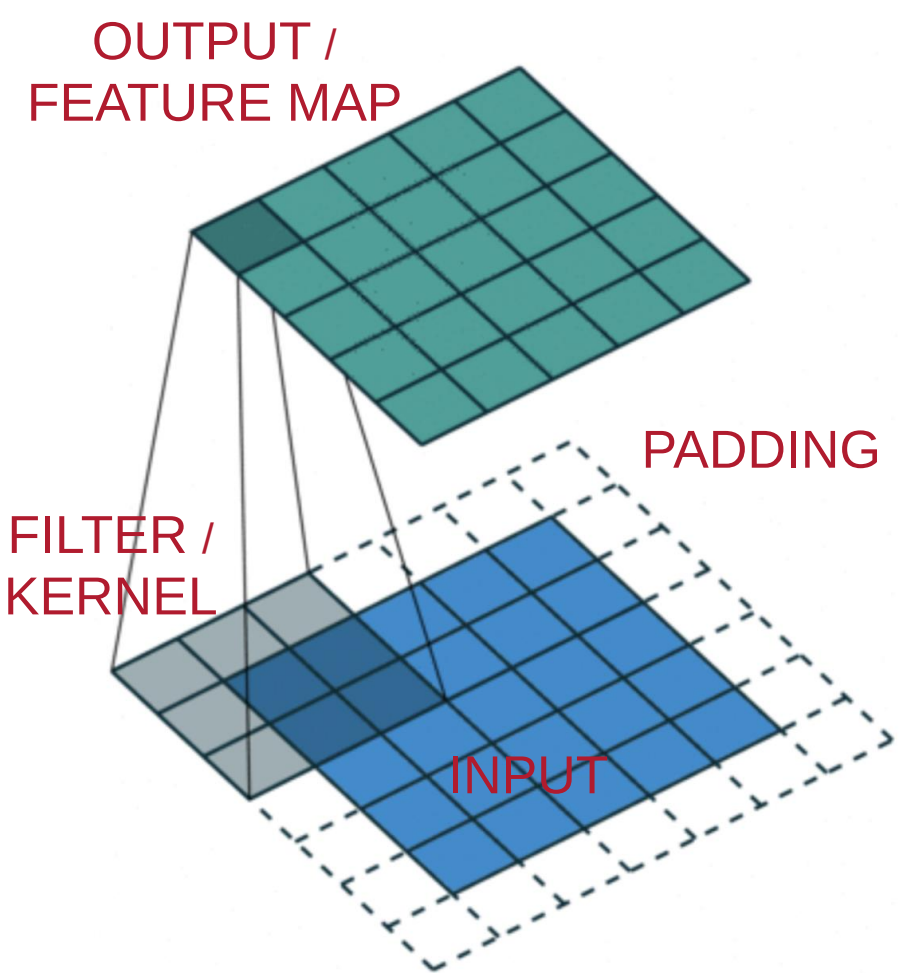
Stride 1 with Padding

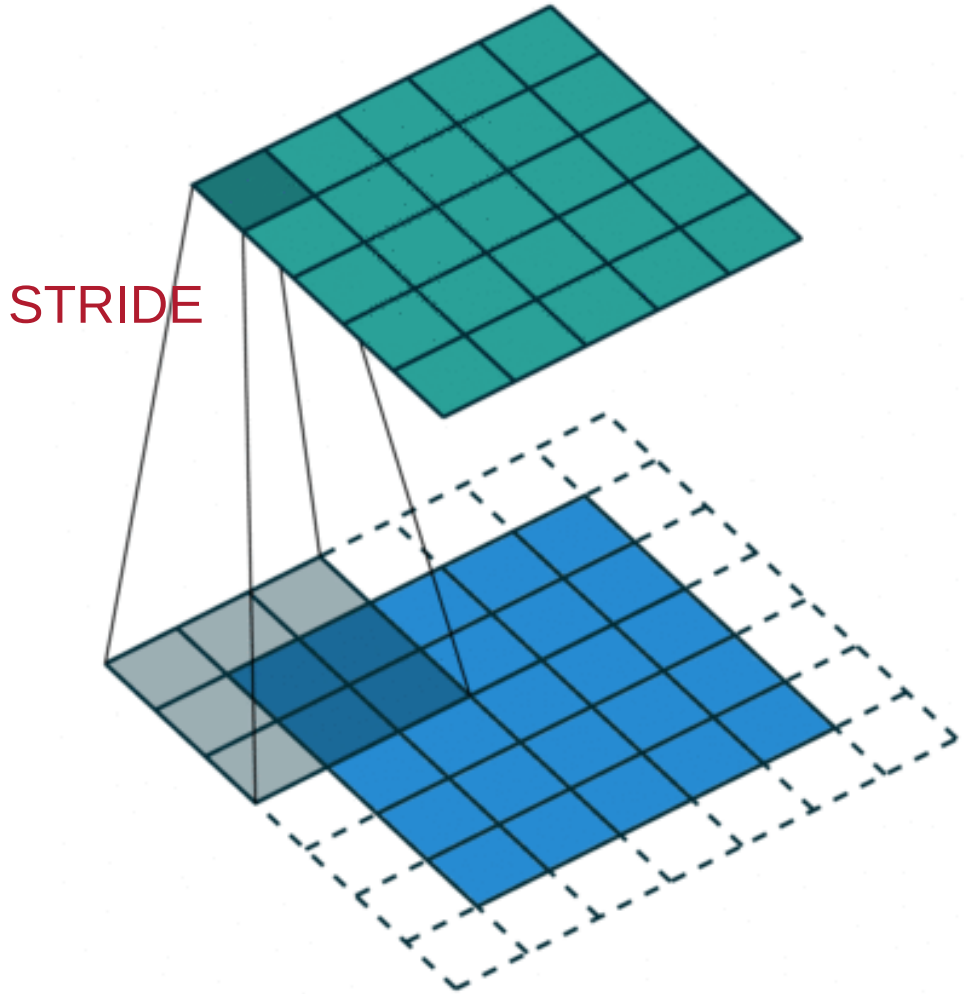


Feature Map

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

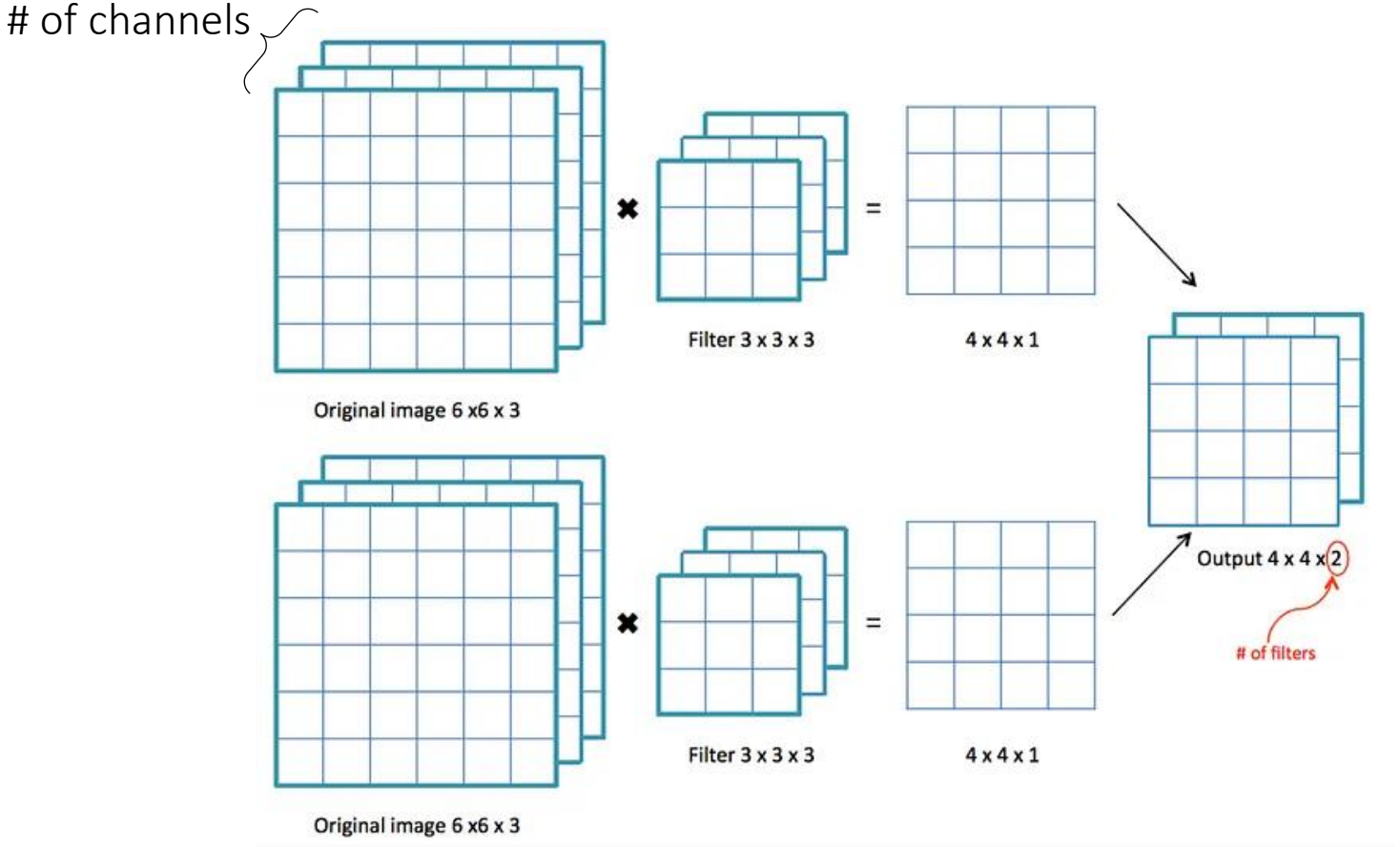
CONVOLUTIONAL LAYER / CONV





합성곱연산(Convolution)

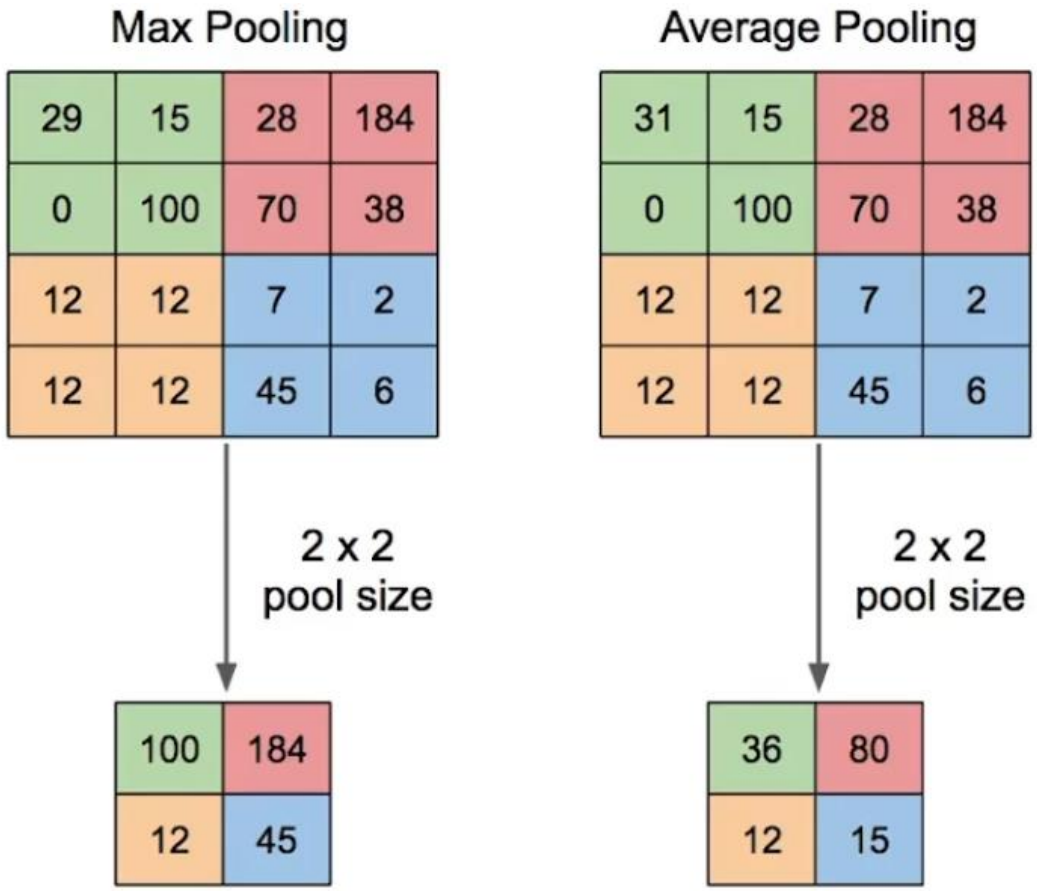
$$Output\ size = \left(\frac{\mathbf{H} + 2p - f}{s} + 1\right) \times \left(\frac{\mathbf{W} + 2p - f}{s} + 1\right) \times \text{(\# of filters)}$$



https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Pooling

✓ 문제점 : 고차원의 Tensor를 보다 Compact하게 축약해야 하지 않을까?
✓ Pooling : 일정 영역의 정보를 축약하는 역할



https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/

POOLING LAYER / POOL

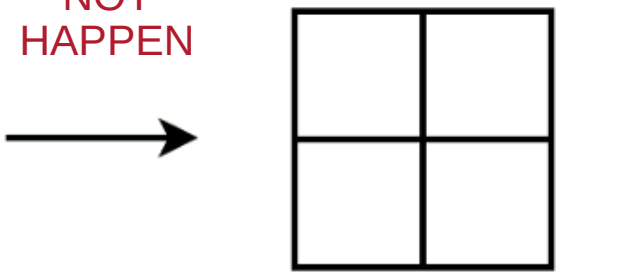
REDUCE THE SPATIAL SIZE

CONTROL OVERFITTING

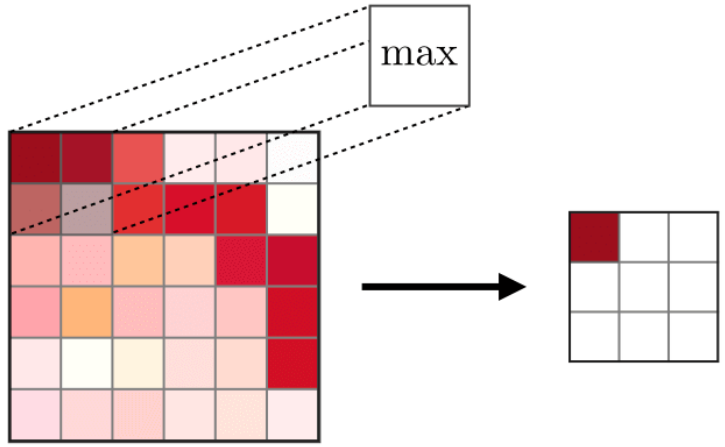
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

FEATURE MAP

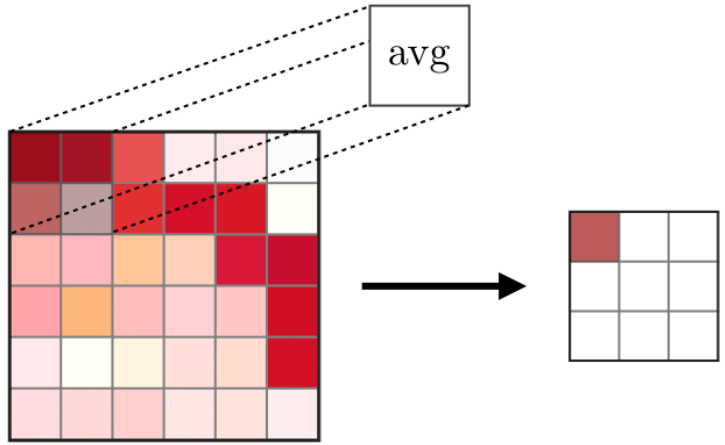
TRAINING NOT HAPPEN



POOLED FEATURE MAP

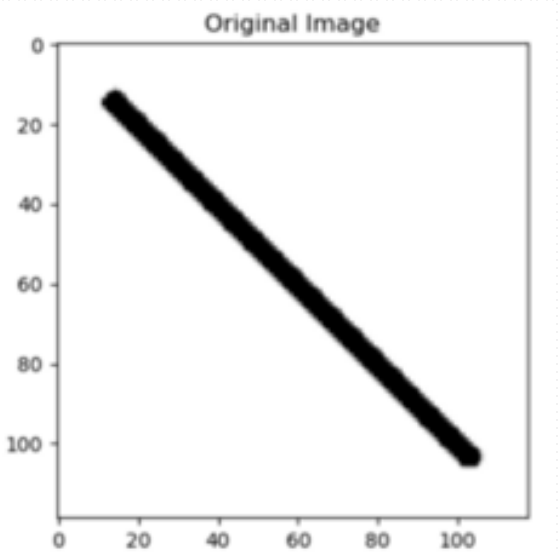


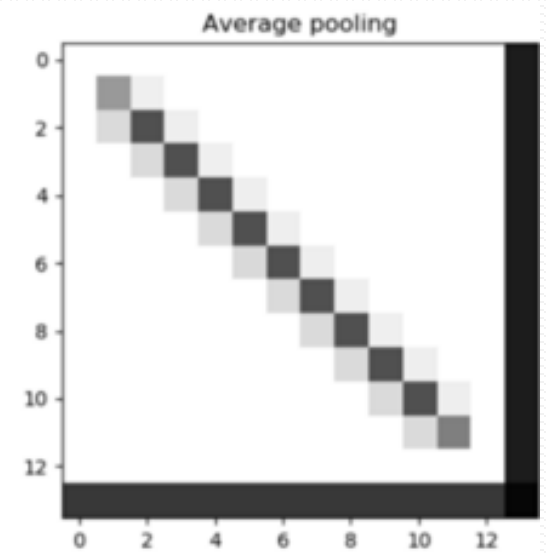
MAX POOLING

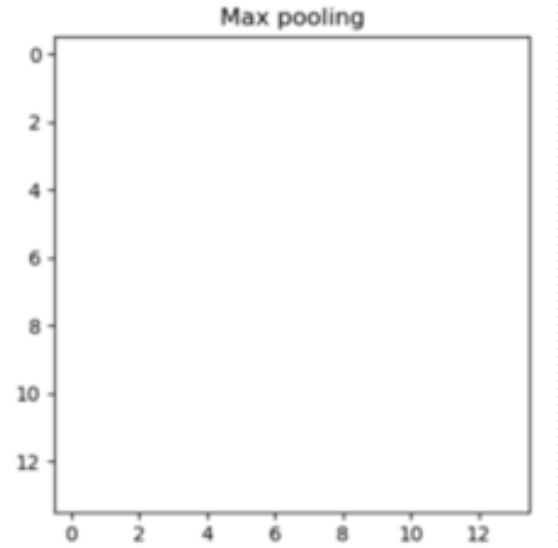


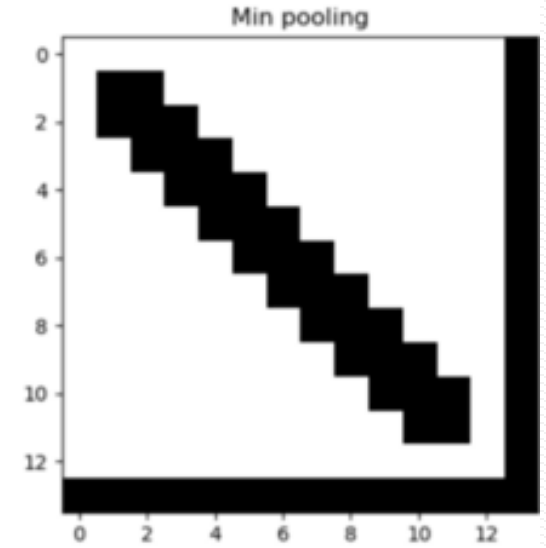
AVERAGE POOLING

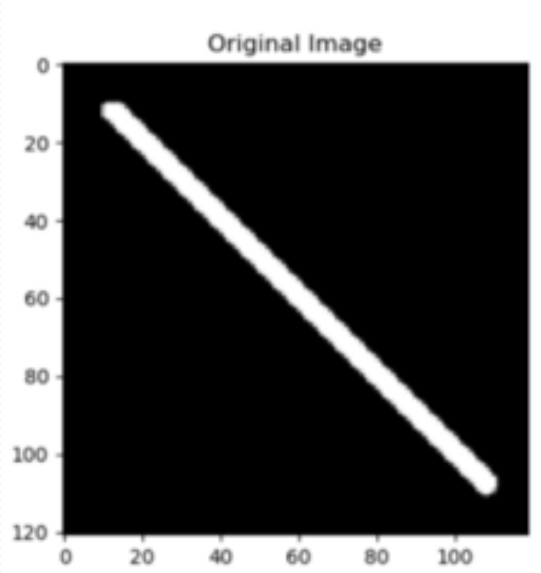
Compare Pooling algorithm (cont.)

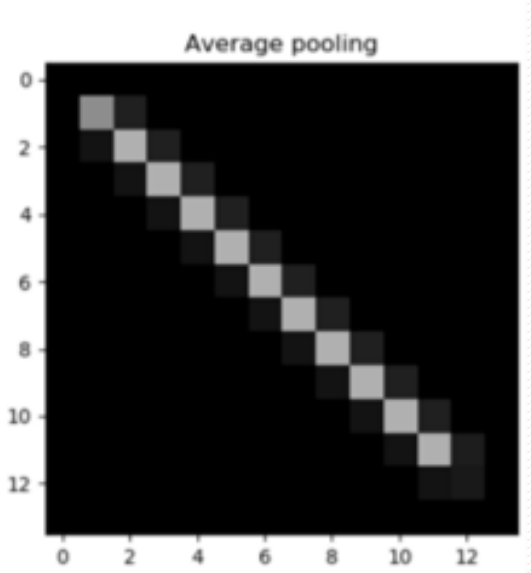


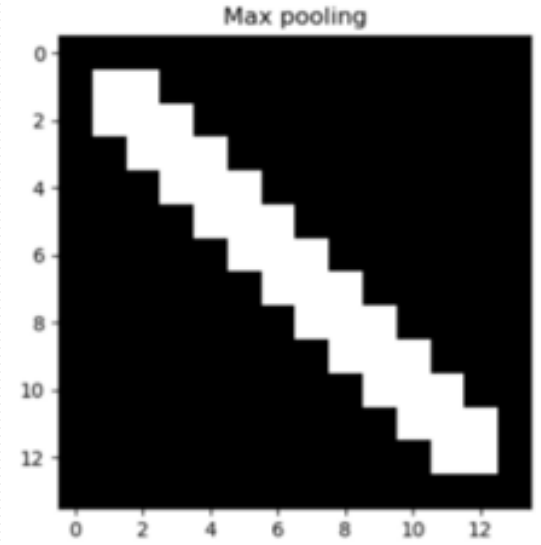


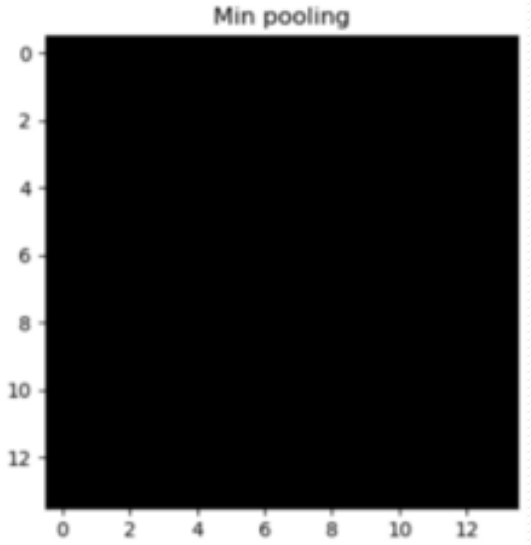




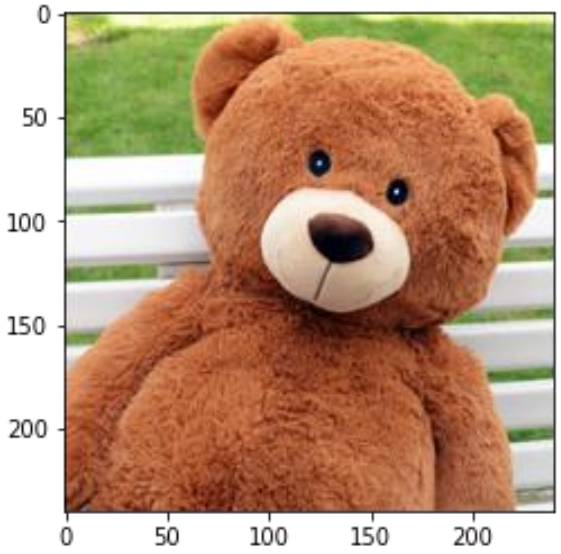




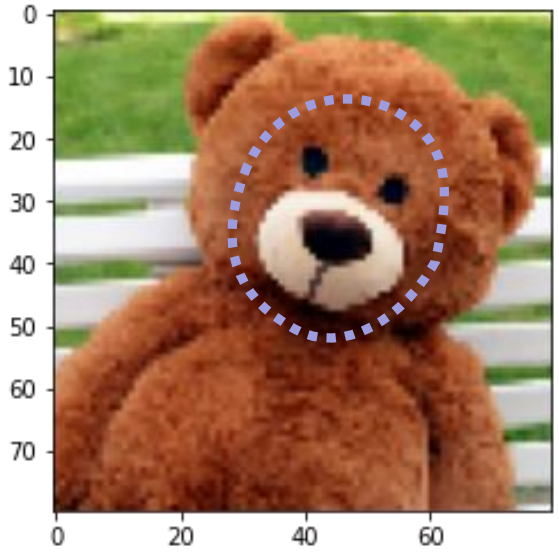




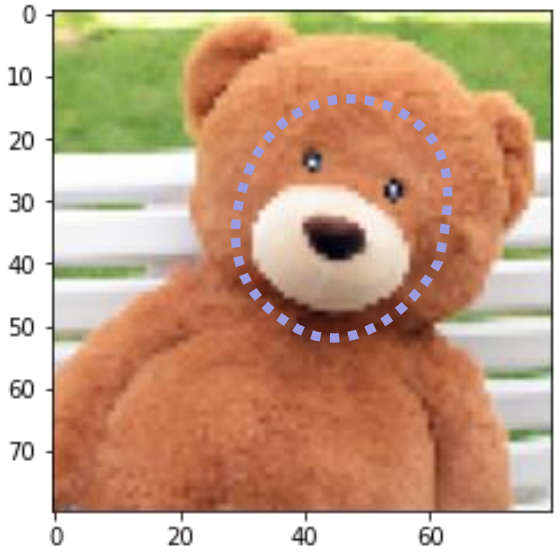
COMPARE POOLING ALGORITHM



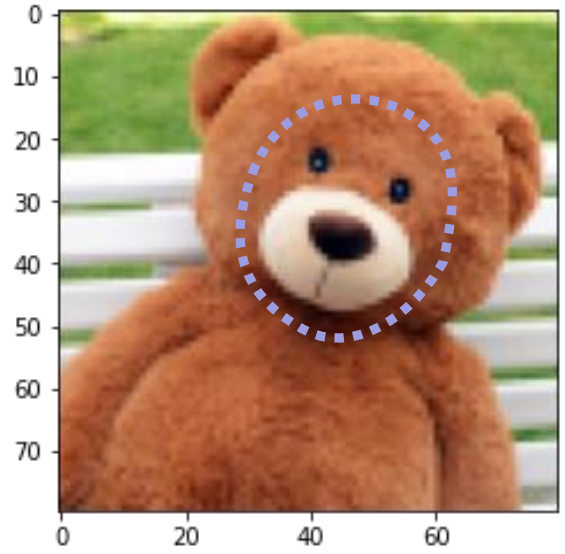
ORIGINAL



MIN



MAX



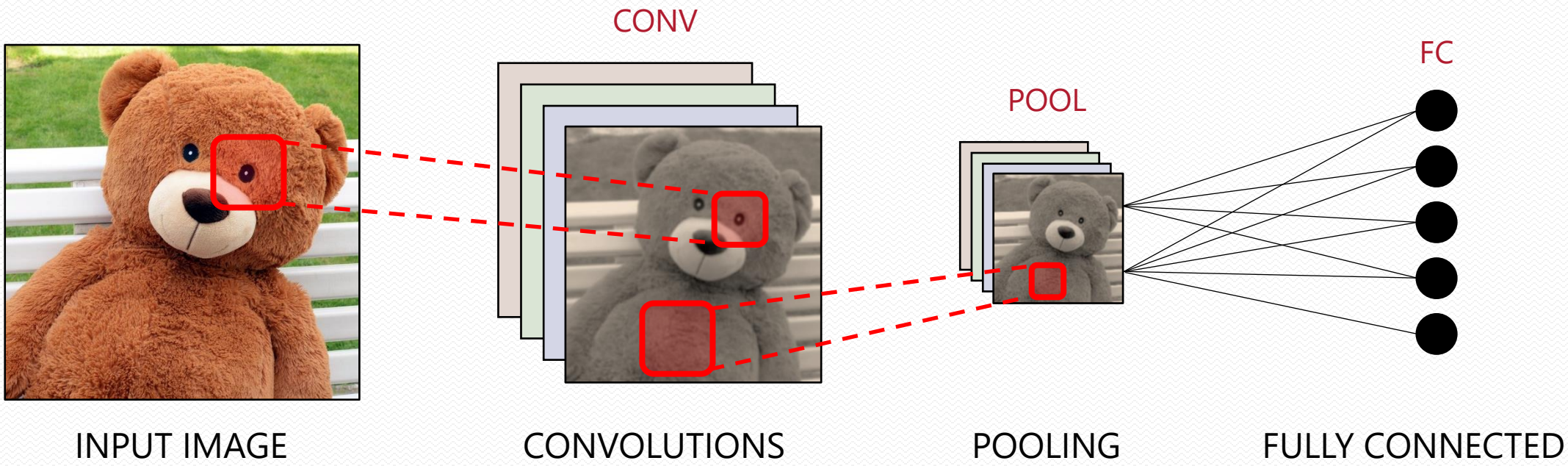
AVERAGE

Compare Pooling algorithm

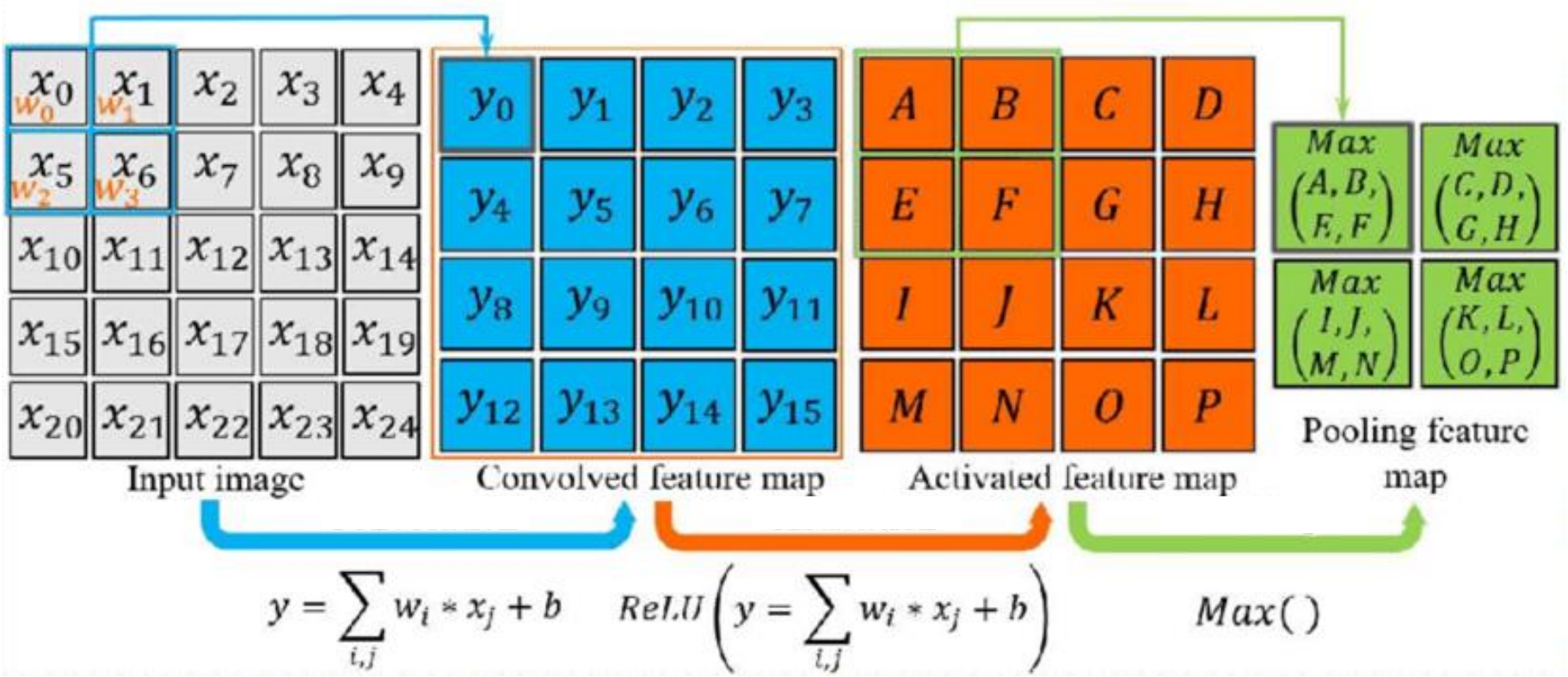
CONVOLUTIONAL NEURAL NETWORK

SPECIFIC TYPE OF NEURAL NETWORKS

MAINLY USED FOR IMAGE CLASSIFICATION
GENERALLY COMPOSED OF THE 3 LAYERS



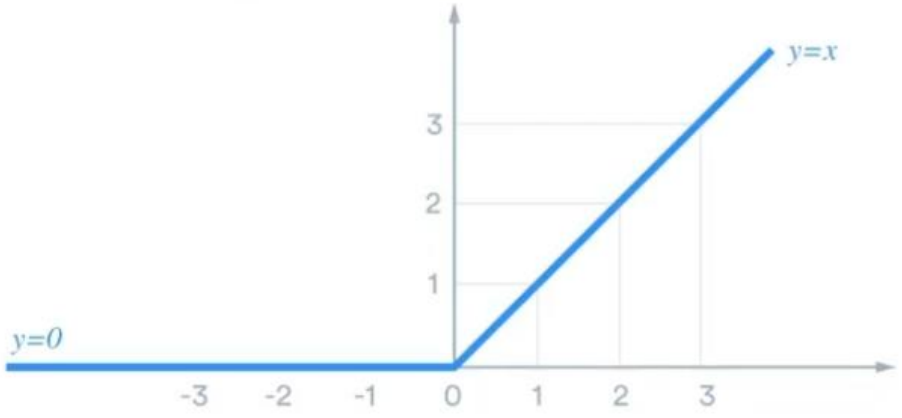
PUT ALL TOGETHER

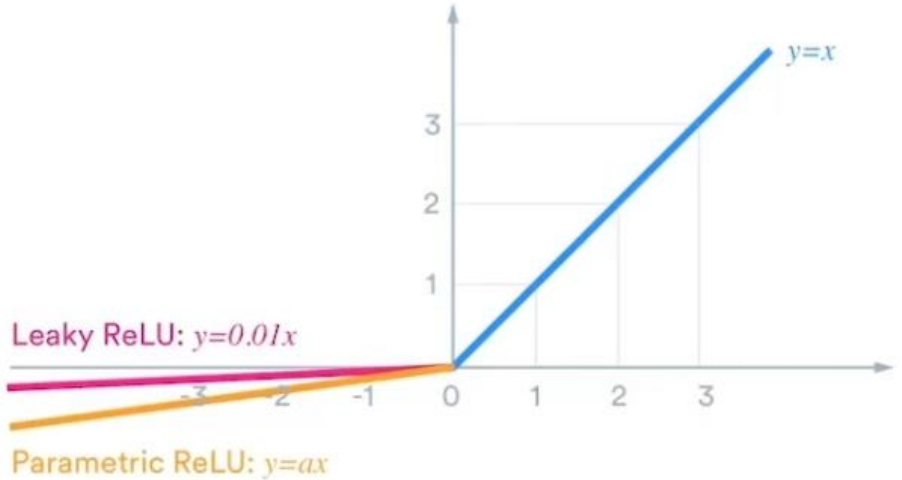


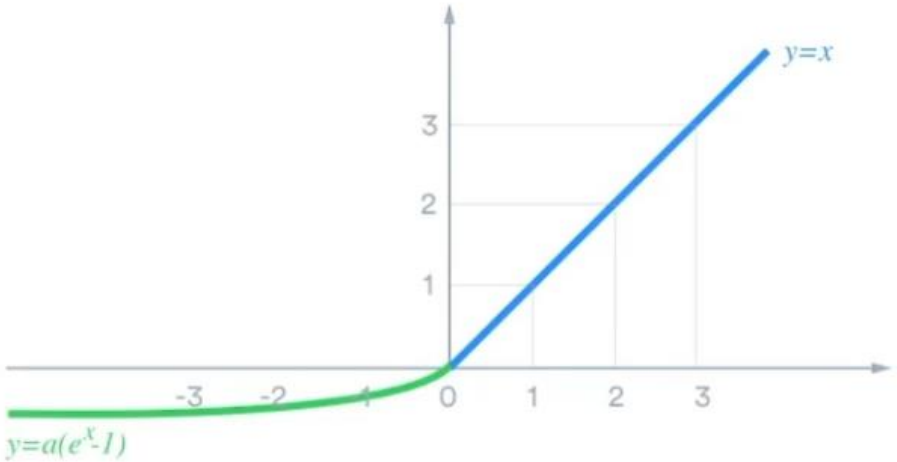
CONVOLUTION ACTIVATION POOLING

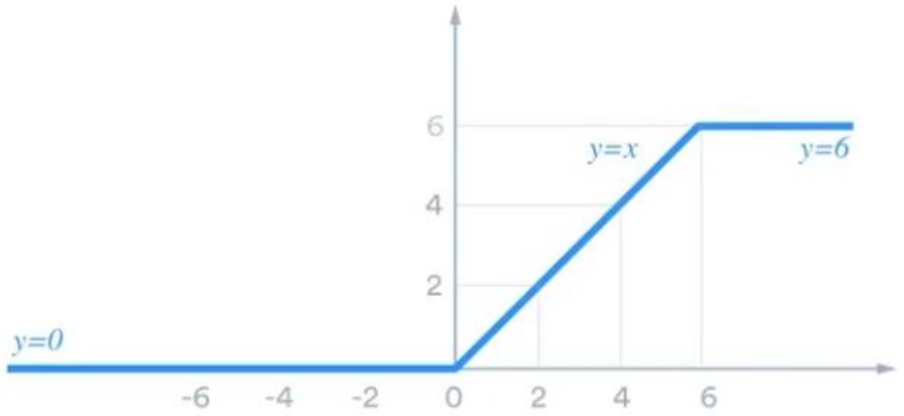
Activation

ReLU(Rectified Linear Unit) 사용 → 최근에는 변형 사용





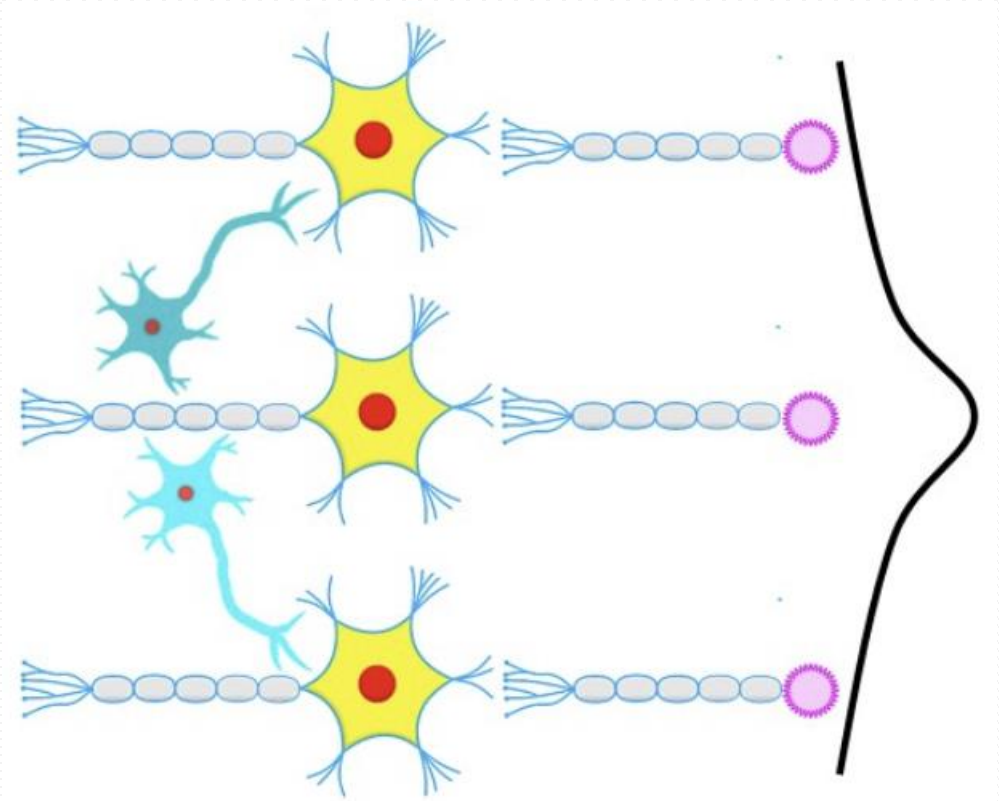


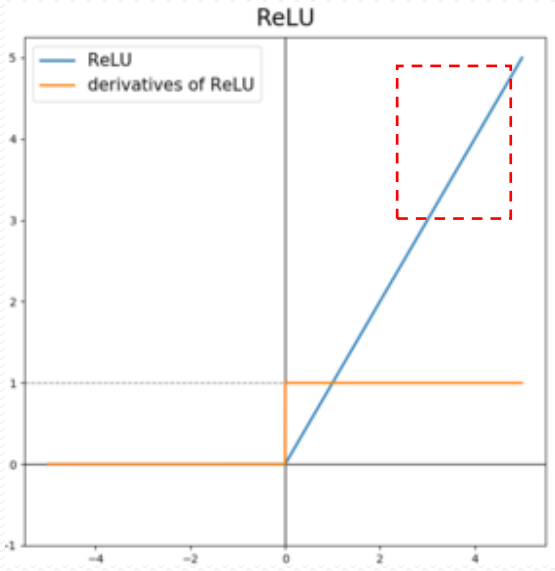


https://www.tinymind.com/learn/terms/relu

Activation

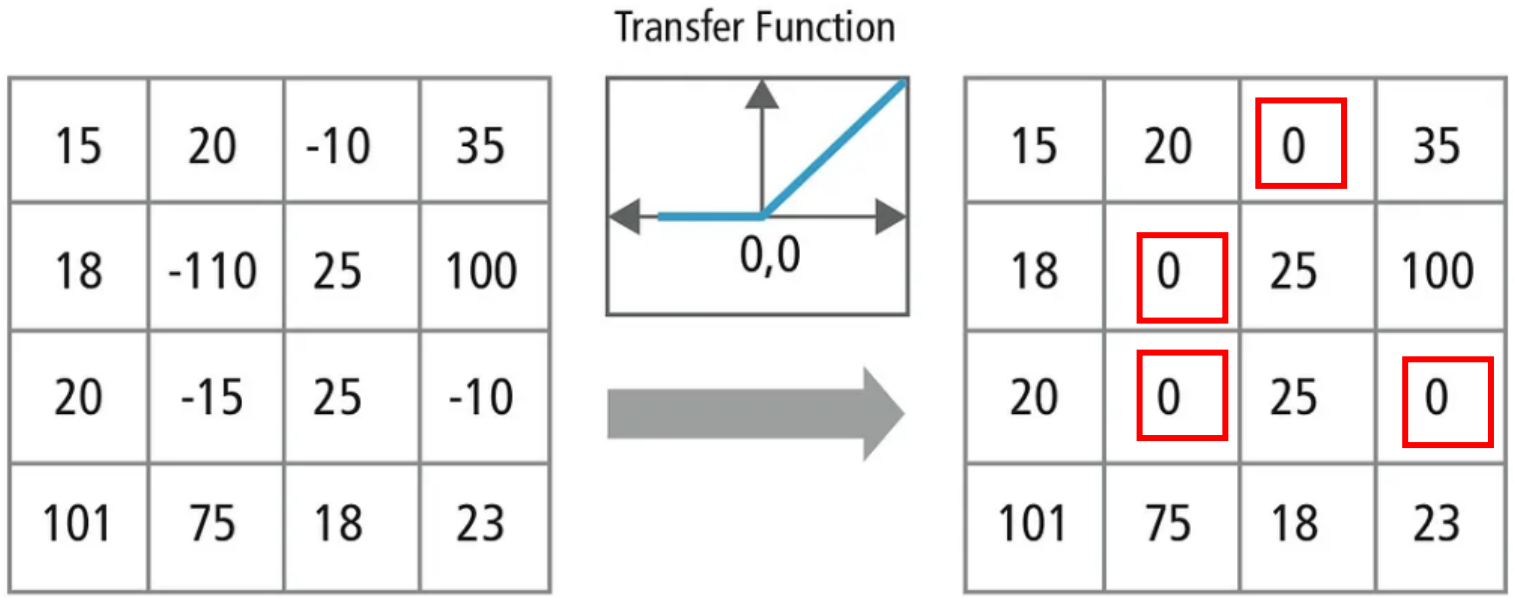
입출력단을 갖는 2개의 신경세포에서 각각의 신경세포의 입력 또는 출력이 서로 다른 신경세포의 입력 또는 출력에 의해 억제되는 현상.





Activation

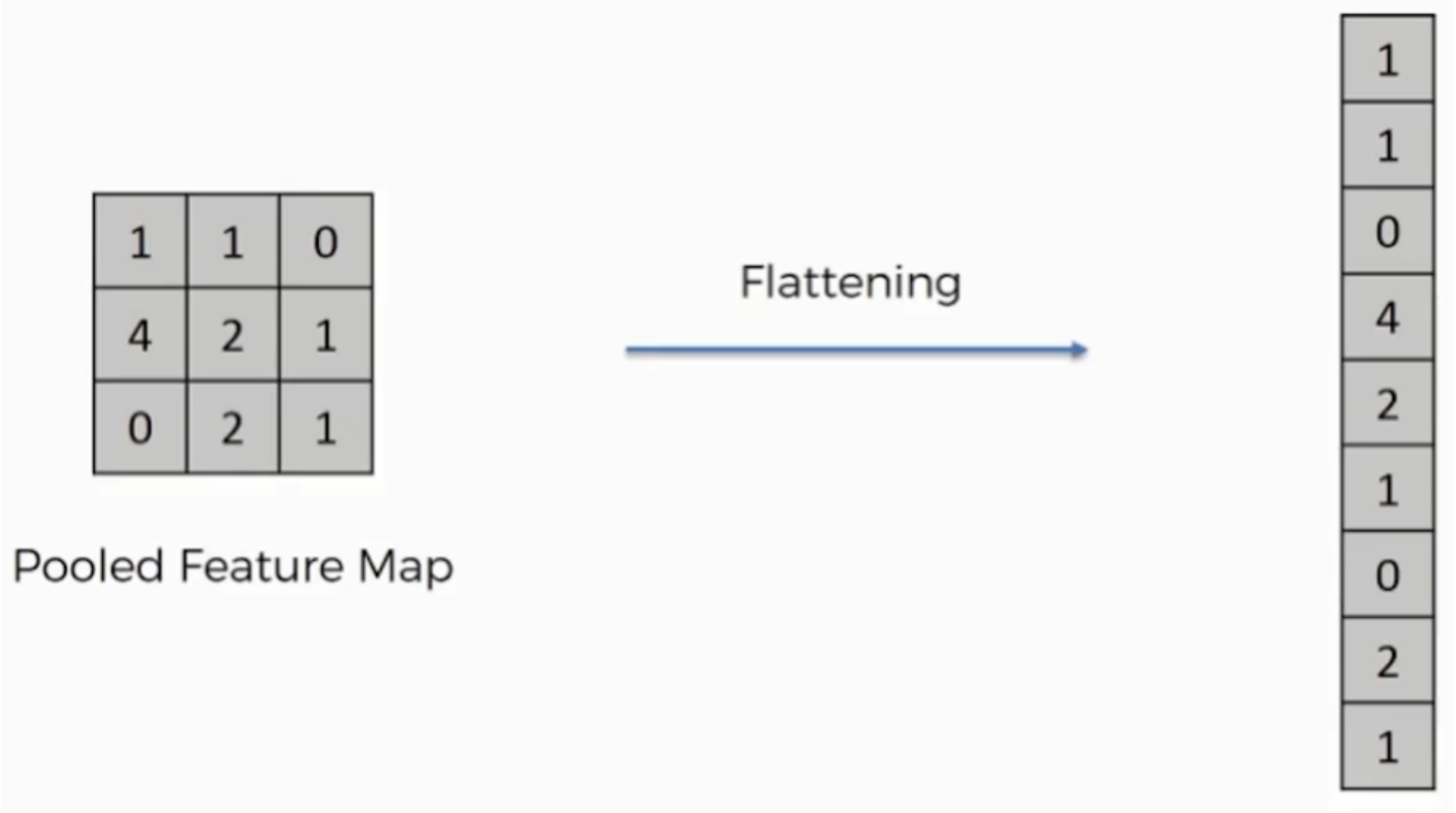
✓ **Convolution을 통해 학습된 값들의 비선형 변환을 수행**
✓ **대부분 Rectified Linear Unit (ReLU)을 사용**



https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

평탄화 : Flatten

✓ **2차원/3차원의 Matrix/Tensor 구조를 1차원의 Vector로 변환하는 과정**

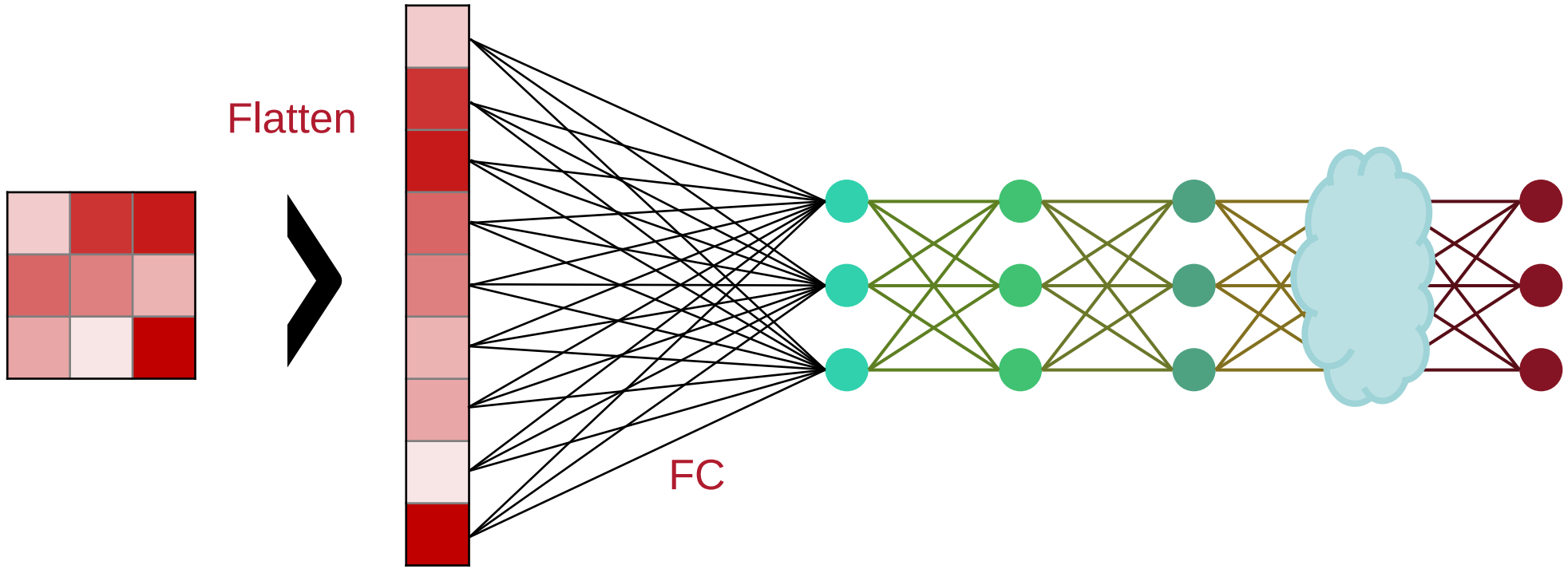


https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/

FULLY CONNECTED LAYER / FC

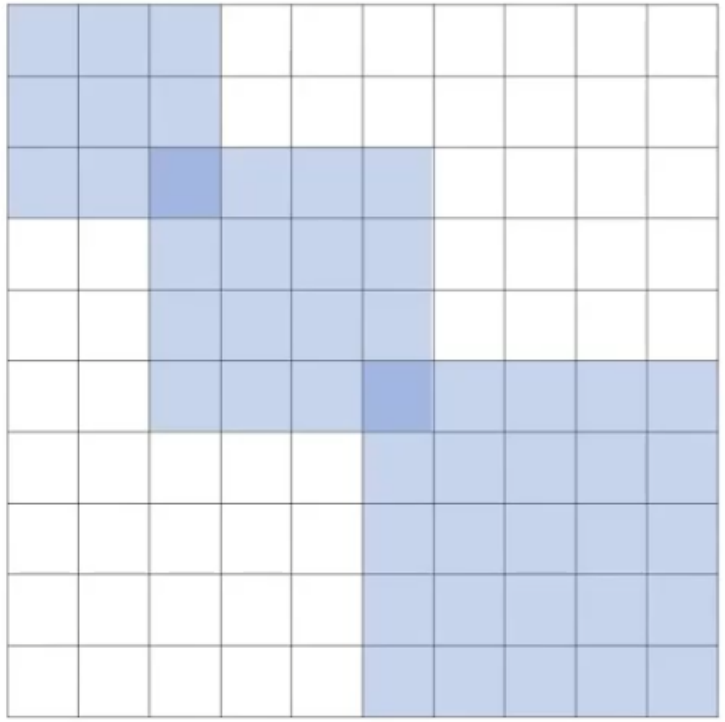
WORKS ON FLATTENED INPUTS WHERE EACH INPUT CONNECTS TO ALL NEURONS

FC LAYERS ARE USUALLY FOUND TOWARDS THE END OF CNN ARCHITECTURES



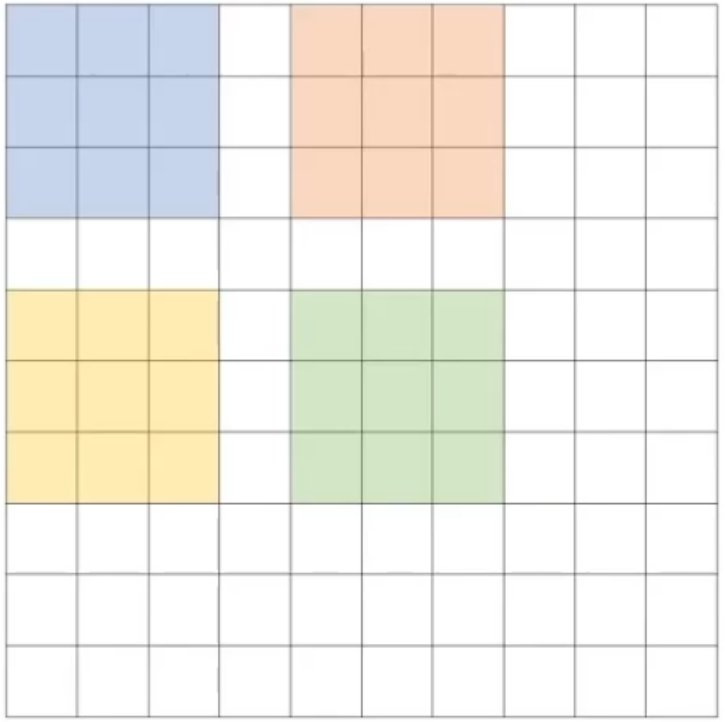
CNN의 하이퍼 파라미터

[Convolution Filter의 크기]



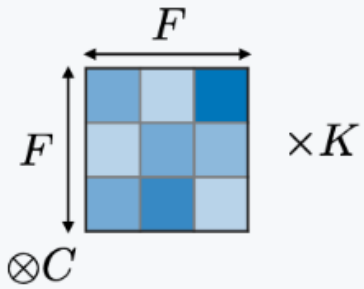
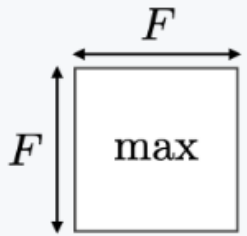
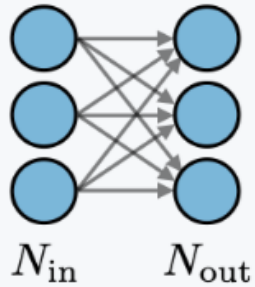
- Convolution Filter의 가로/세로 크기
- 이미지 데이터에는 주로 2-d Conv를 사용 (가로와 세로의 크기가 같은 Conv 2*2, 3*3 등)

[Convolution Filter의 수]



- Convolution Filter의 개수
- 이 수가 많을 수록 동일 영역에서 다양한 형태의 특질을 추출할 수 있음

Determine the *N* of parameters

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks *S*: Stride	• One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$	• Pooling operation done channel-wise • In most cases, $S = F$	• Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints

SIMPLE CNN EXAMPLE WITH KERAS

```
model = Sequential(name="Simple-CNN")
model.add(
    Conv2D(filters=8,
           kernel_size=(3, 3),
           padding="same",
           activation='relu',
           input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))

model.add(Flatten())
model.add(Dense(64, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))
```

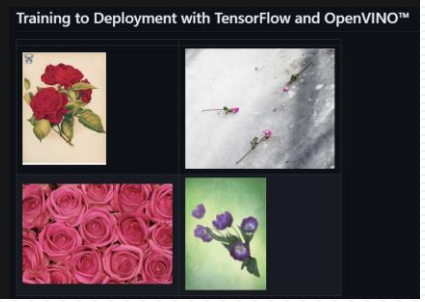
SIMPLE CNN EXAMPLE WITH KERAS / CONT.

```
Model: "Simple-CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 8)         80
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 12, 12, 16)        1168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 16)          2320
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 64)                16448
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
```

SIMPLE CNN EXAMPLE WITH KERAS

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2D  (None, 90, 90, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0
_________________________________________________________________
```

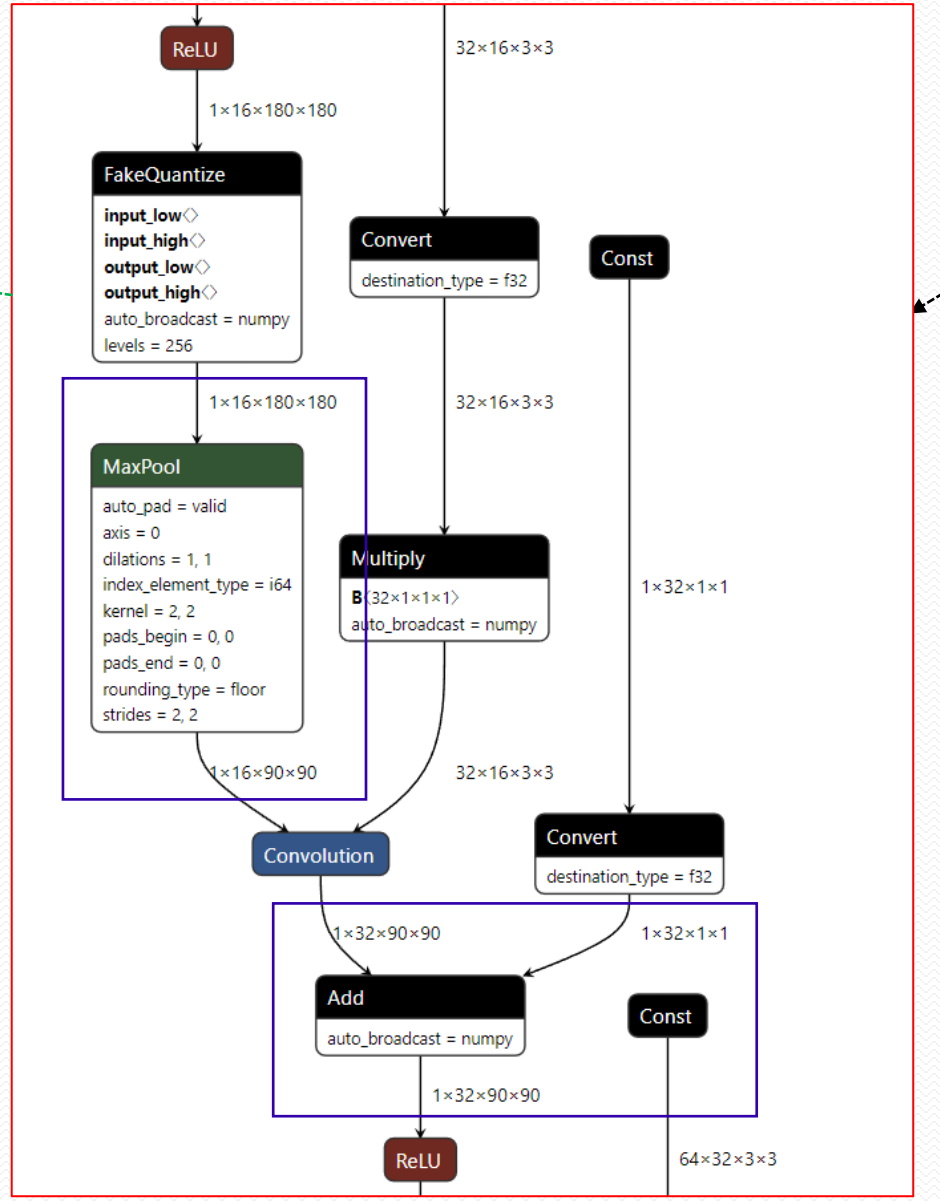
```python
# 모델 만들기
model = tf.keras.Sequential()
model.add(
    tf.keras.layers.Rescaling(
        1./255, input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT, 3)))
model.add(
    tf.keras.layers.Conv2D(16, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Conv2D(32, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Conv2D(64, 3, padding="same", activation="relu"))
model.add(
    tf.keras.layers.MaxPool2D())
model.add(
    tf.keras.layers.Flatten())
model.add(
    tf.keras.layers.Dropout(0.2)
)
model.add(
    tf.keras.layers.Dense(128, activation="relu")
)
model.add(
    tf.keras.layers.Dense(5))
model.summary()
```

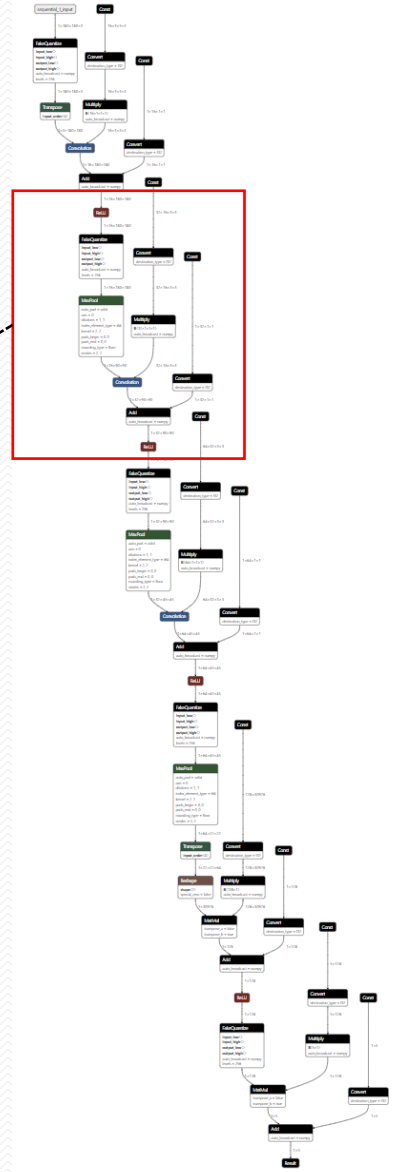


https://netron.app/

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3965056
dense_1 (Dense)	(None, 5)	645

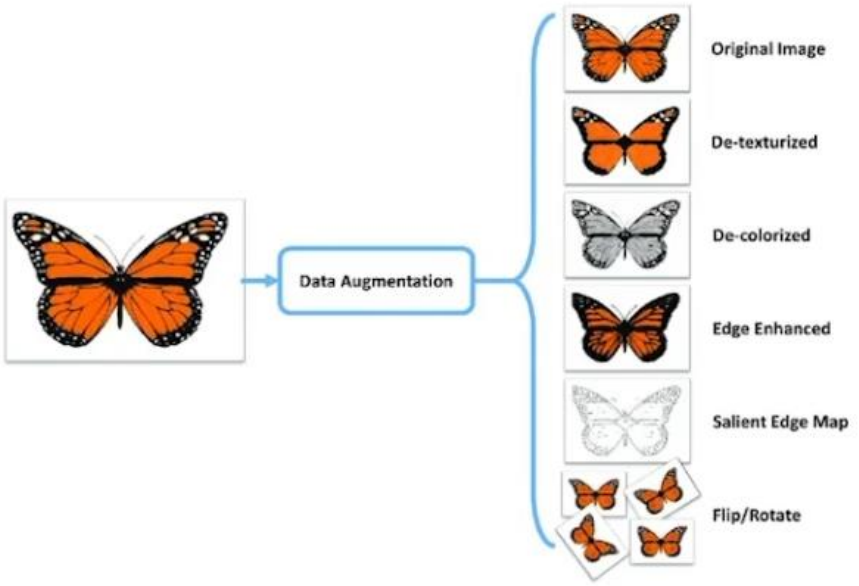
Total params: 3,989,285
Trainable params: 3,989,285
Non-trainable params: 0





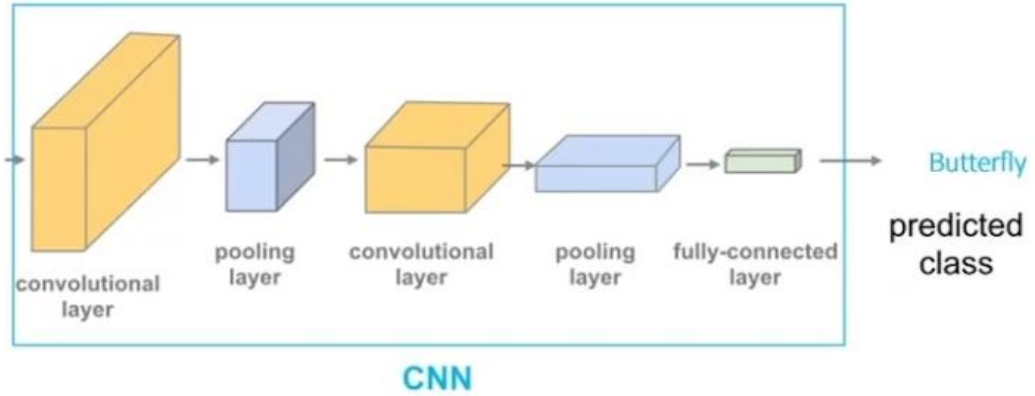
Data Augmentation

✓ 현실에서 충분히 확보하지 못한 데이터에 대해 적절한 변형을 가하여 인위적으로 데이터를 증강시키자!
✓ 그럴듯하게 증강시키면(이미지 회전, 상하 반전, 약간의 노이즈 추가 등) 오히려 보다 다양한 형태를 학습하게 됨으로써 더 높은 정확도를 얻을 수 있지 않을까?



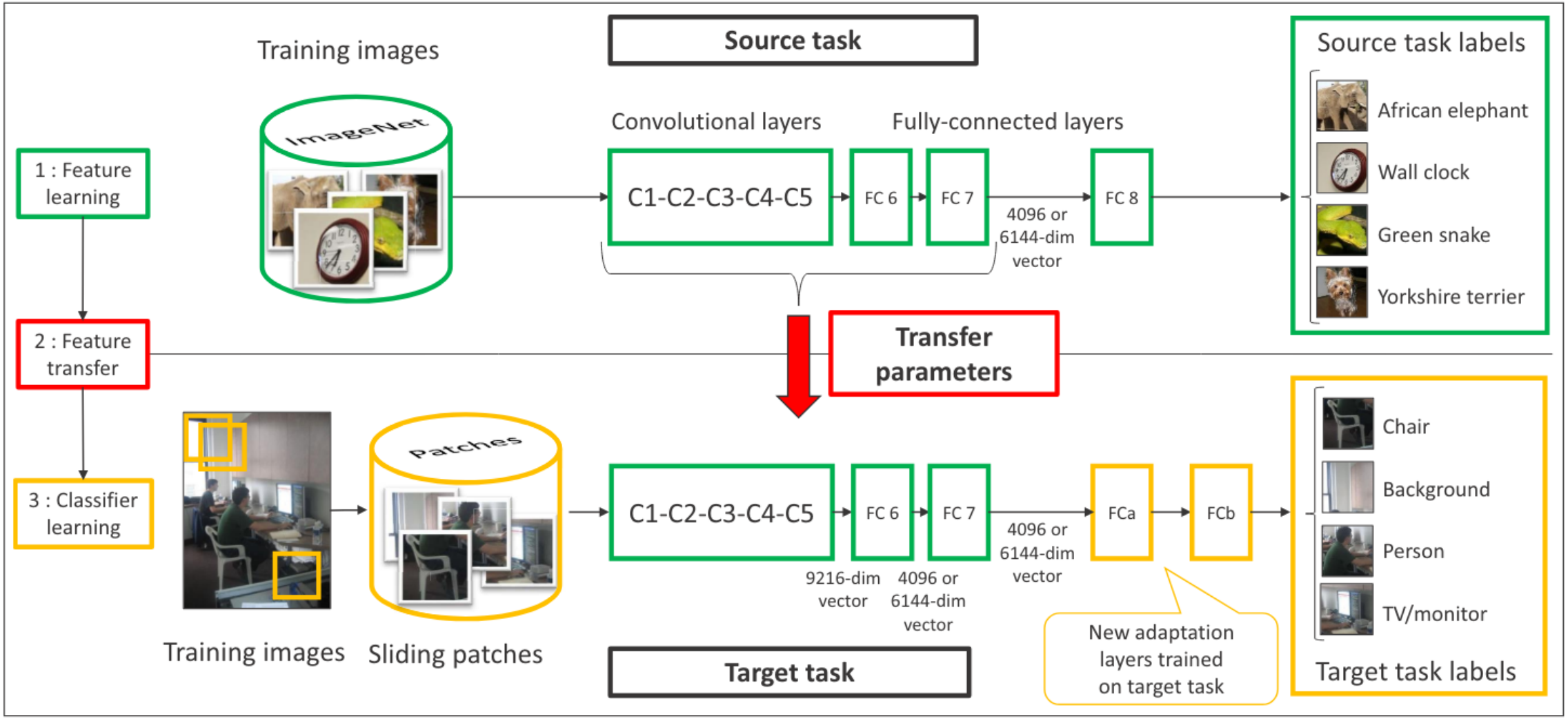
내 비록 한장의 이미지만
가지고 있지만

이를 적절히
뻥튀기 하여



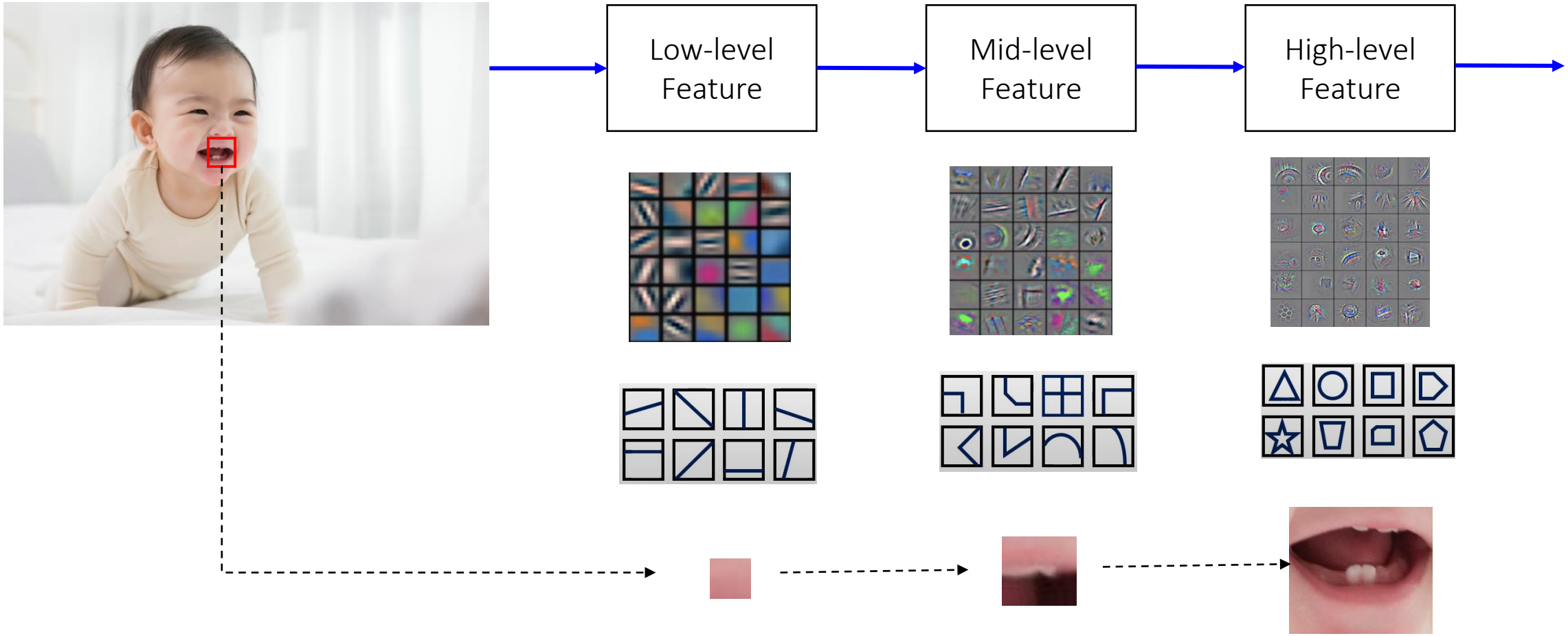
모델 학습에 도움이 되도록 하리라…

Transfer learning



Oquab, M., Bottou, L, Laptev,I,&Sivic,J.(2014). Learning and transferring mid-level image representations using convolutional neuralnetworks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1717-1724).

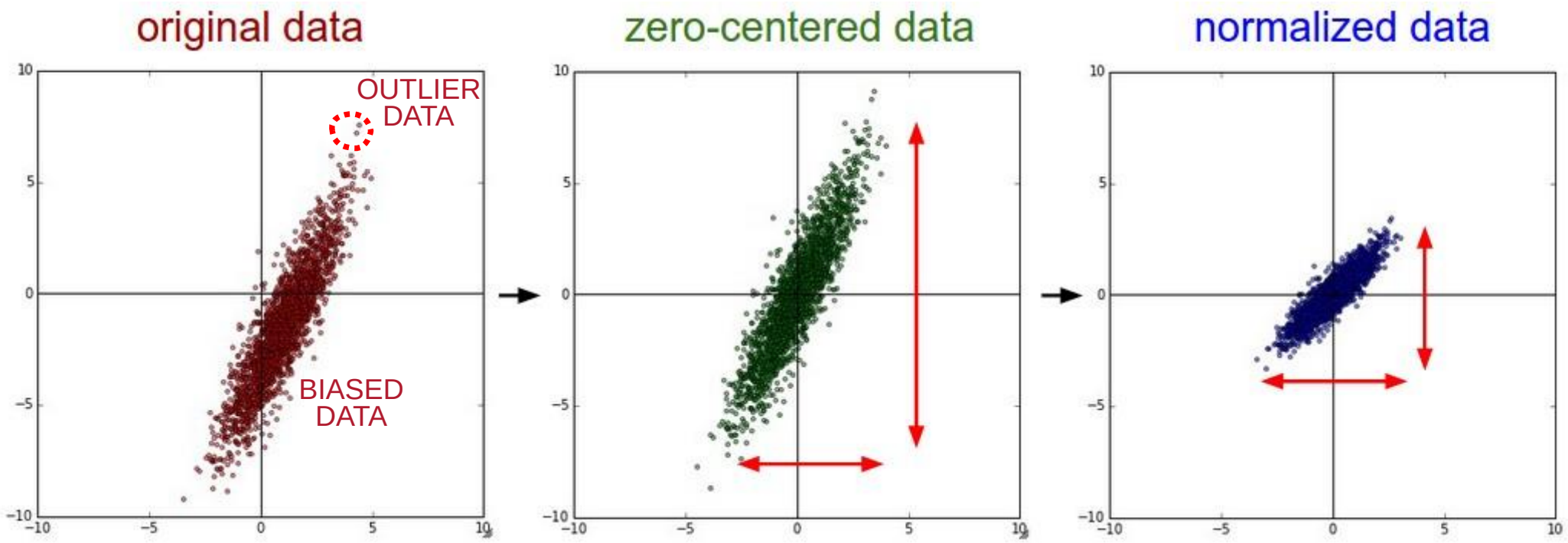
Transfer learning



BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

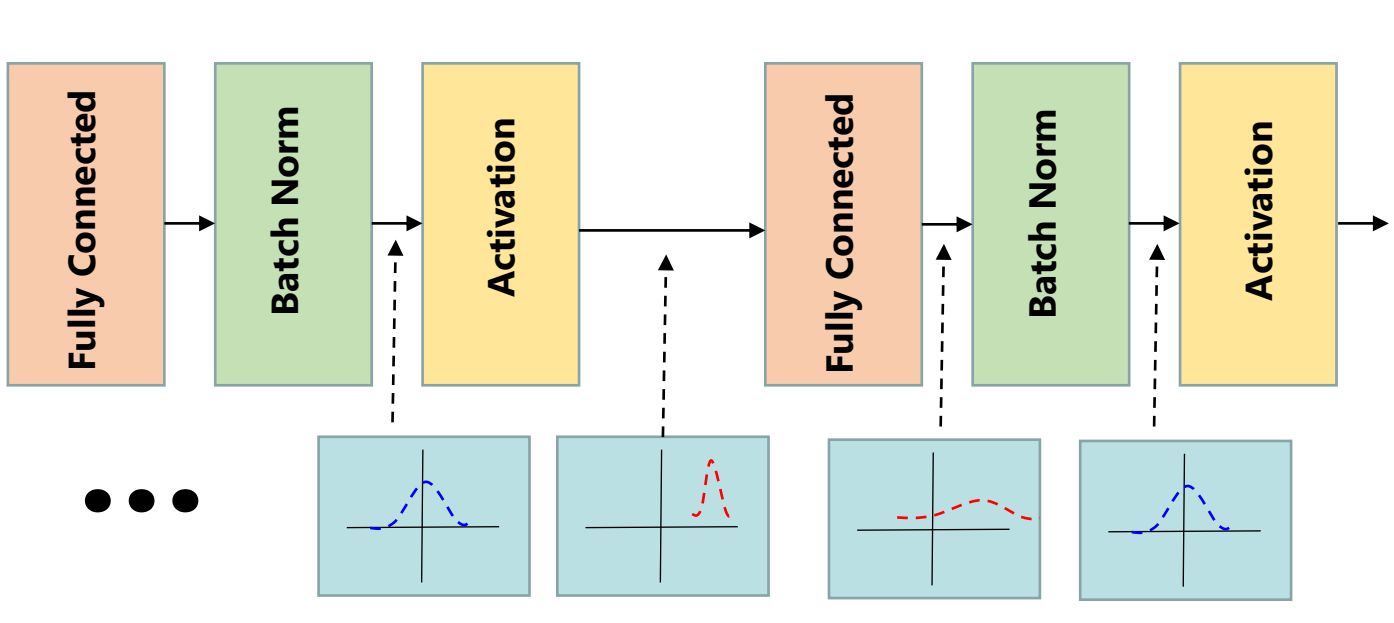
Method to help the network learn better

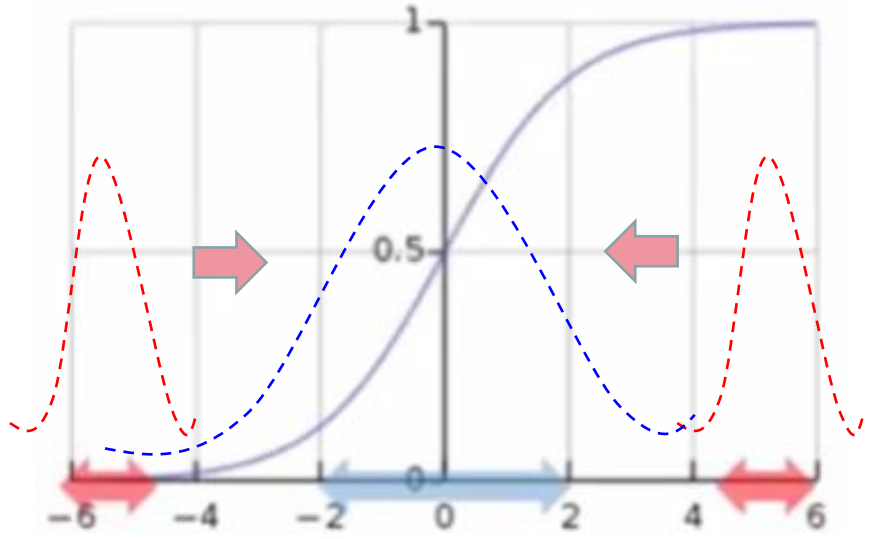


BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better



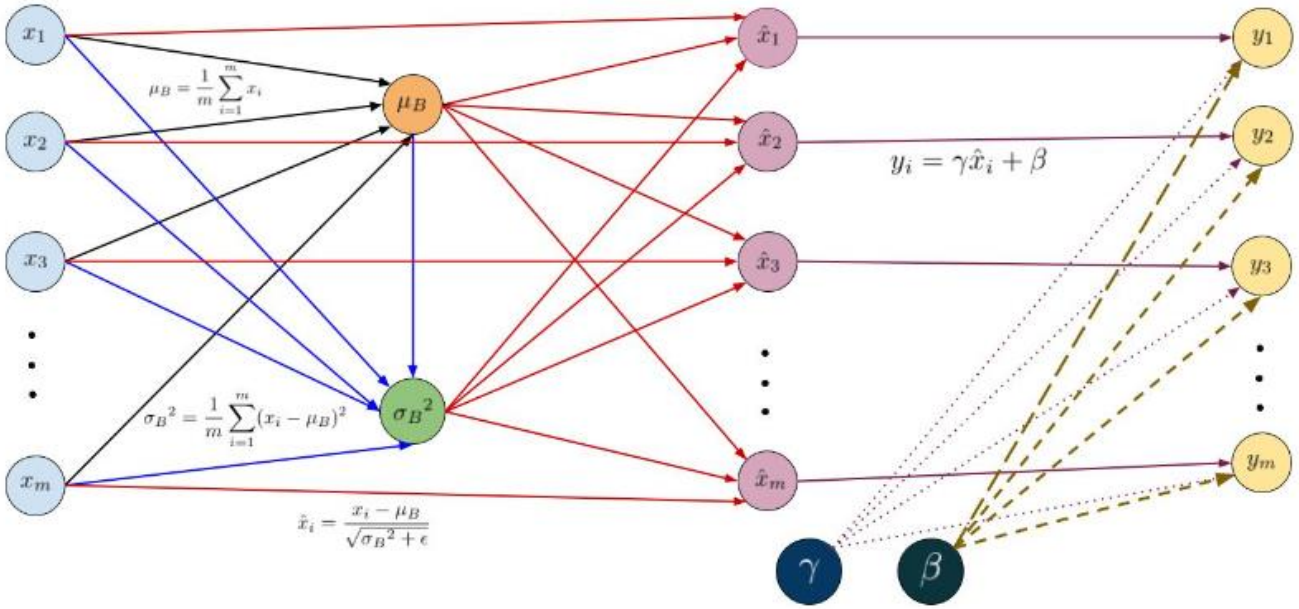


γ, β

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

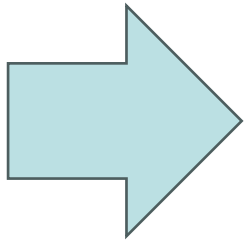
```python
def batchnorm_forward(x, gamma, beta, eps=1e-5):
    N, D = x.shape

    sample_mean = x.mean(axis=0)
    sample_var = x.var(axis=0)

    std = np.sqrt(sample_var + eps)
    x_centered = x - sample_mean
    x_norm = x_centered / std
    out = gamma * x_norm + beta

    cache = (x_norm, x_centered, std, gamma)

    return out, cache
```

BATCH NORMALIZATION

NORMALIZING THE DATA DIMENSIONS

Method to help the network learn better

$$\gamma\left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) + \beta$$



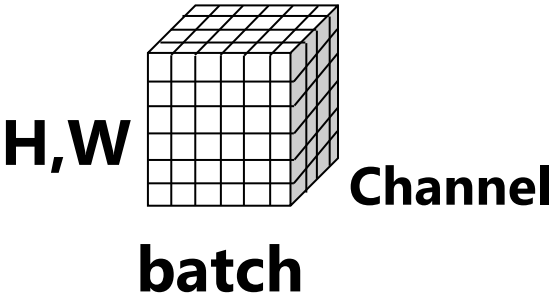
scale and shift

→ 분포재조정됨

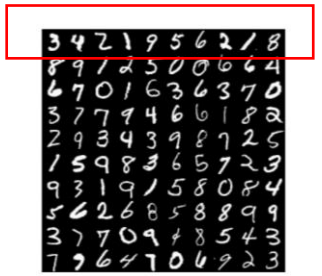
- 평균: β

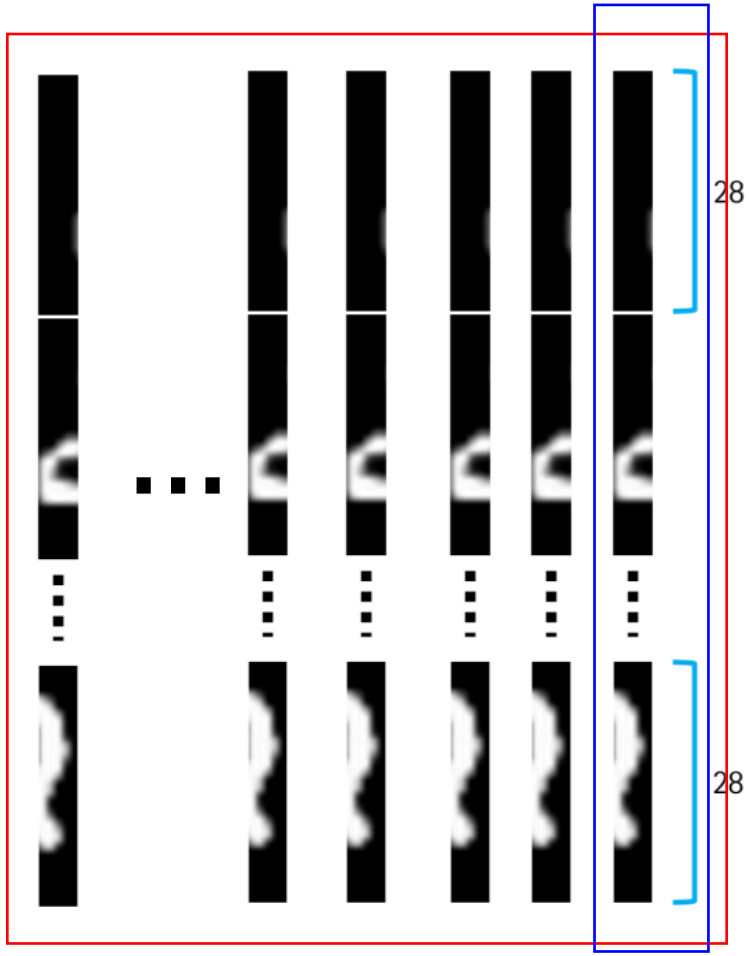
- 분산: γ^2

BATCH & LAYER NORMALIZATION

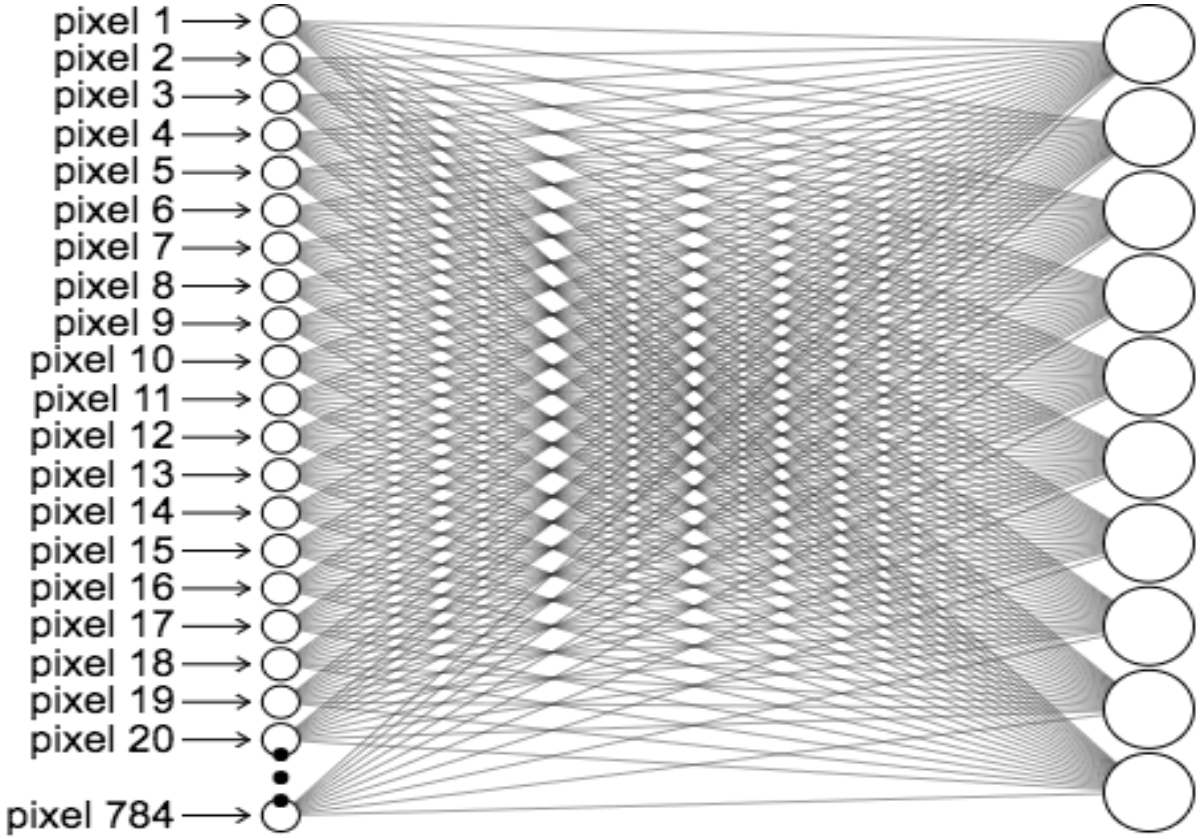


Batch Normalization





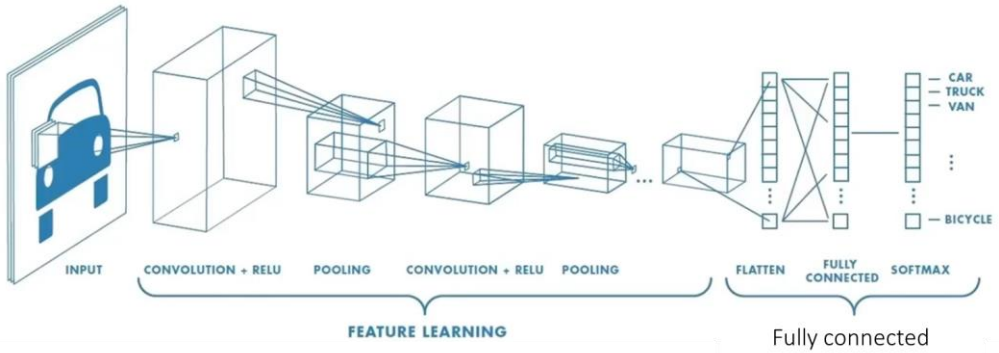
총 784차원 벡터

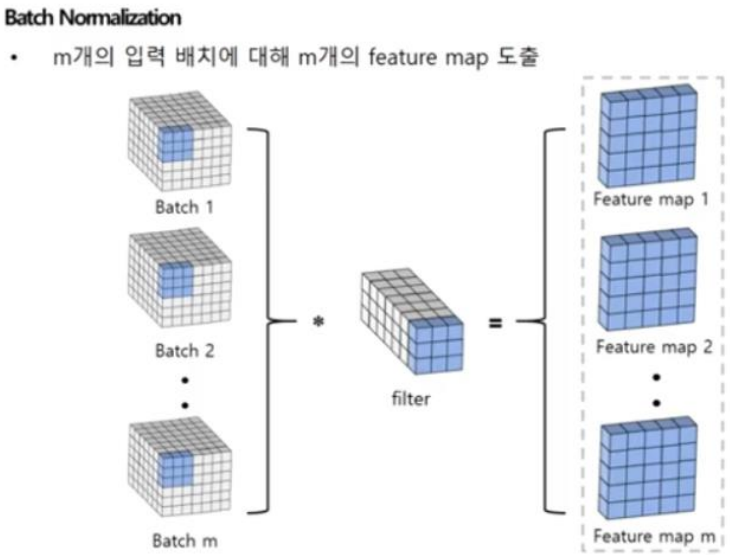


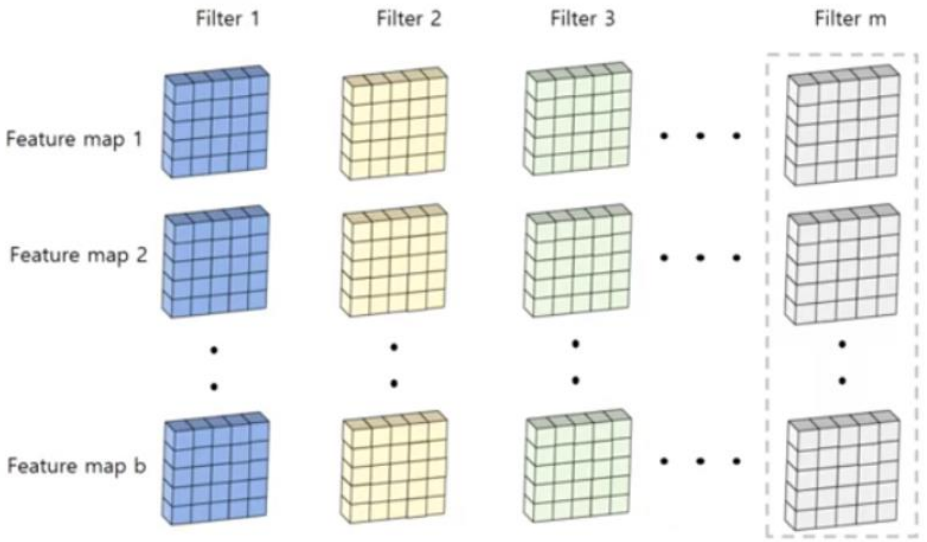
Layer Normalization

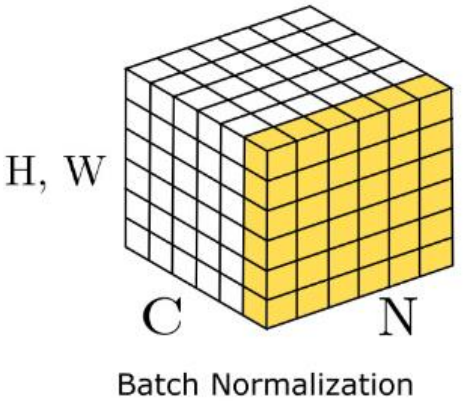
BATCH & LAYER NORMALIZATION

- 각 feature별로,한 mini-batch에 대한summed input의 평균과 분산을 구한다.
- 일반화를 개선하고 overfitting을 감소시킨다.
- 학습의 수렴속도를 높이고 가중치 초기화에 대해 강건해진다.
- 학습의 안정성을 향상시키고 기울기 소실 문제를 완화한다.
- 피드포워드 네트워크 모델의 성능과 안정성 개선에 뛰어나다
- 미니배치 크기에 의존한다.
- RNN Model에 적용하기 힘들다



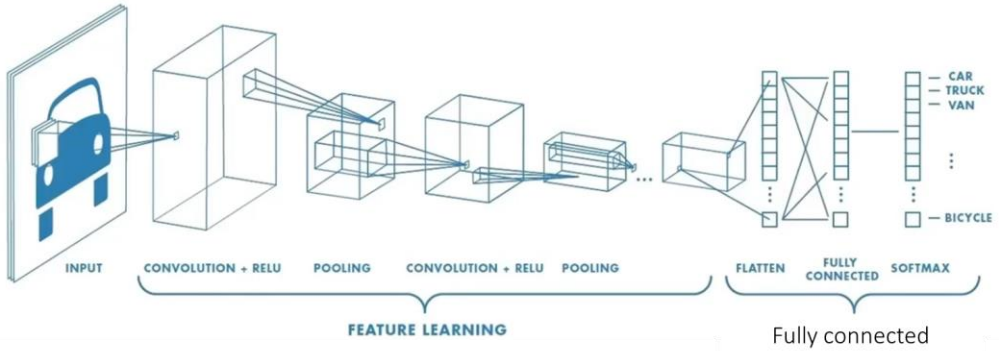


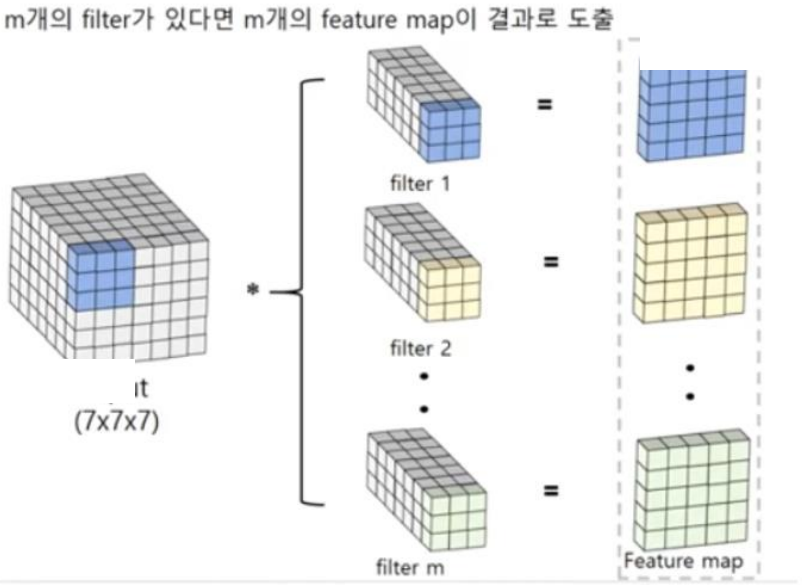


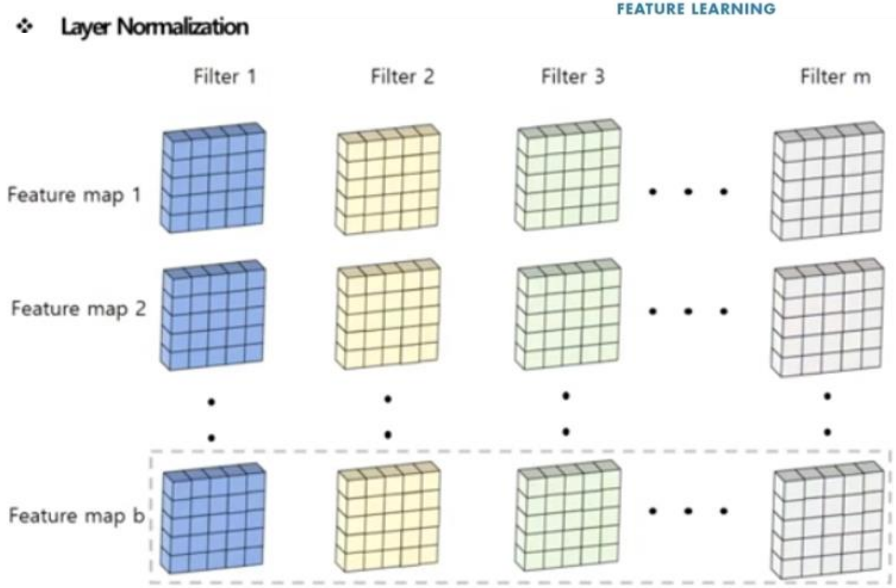


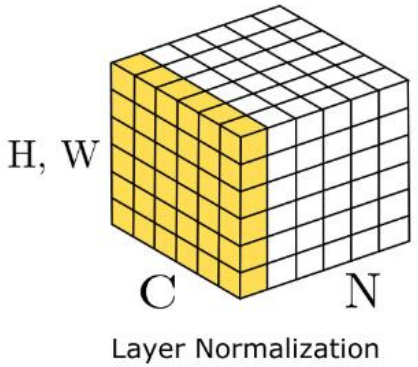
BATCH & LAYER NORMALIZATION

- Batch Normalization은 작은 배치 크기에서 극단적 결과를 내는데 반해, 작은 batch size에서도 효과적인 이용이 가능
- sequence에 따른 고정길이 정규화로 batch normalization에 비해 RNN 모델에 더 효과적인 방법이다.
- 일반화 성능 향상이 가능하다
- 피드포워드 네트워크 모델에서는 Batch Normalization 만큼 잘 동작하지 않을 수 있다.
- learning_rate, 가중치 초기화 같은 하이퍼파라미터 조정에 민감할 수 있다.
- 추가 계산 및 메모리 오버헤드가 발생할 수 있다

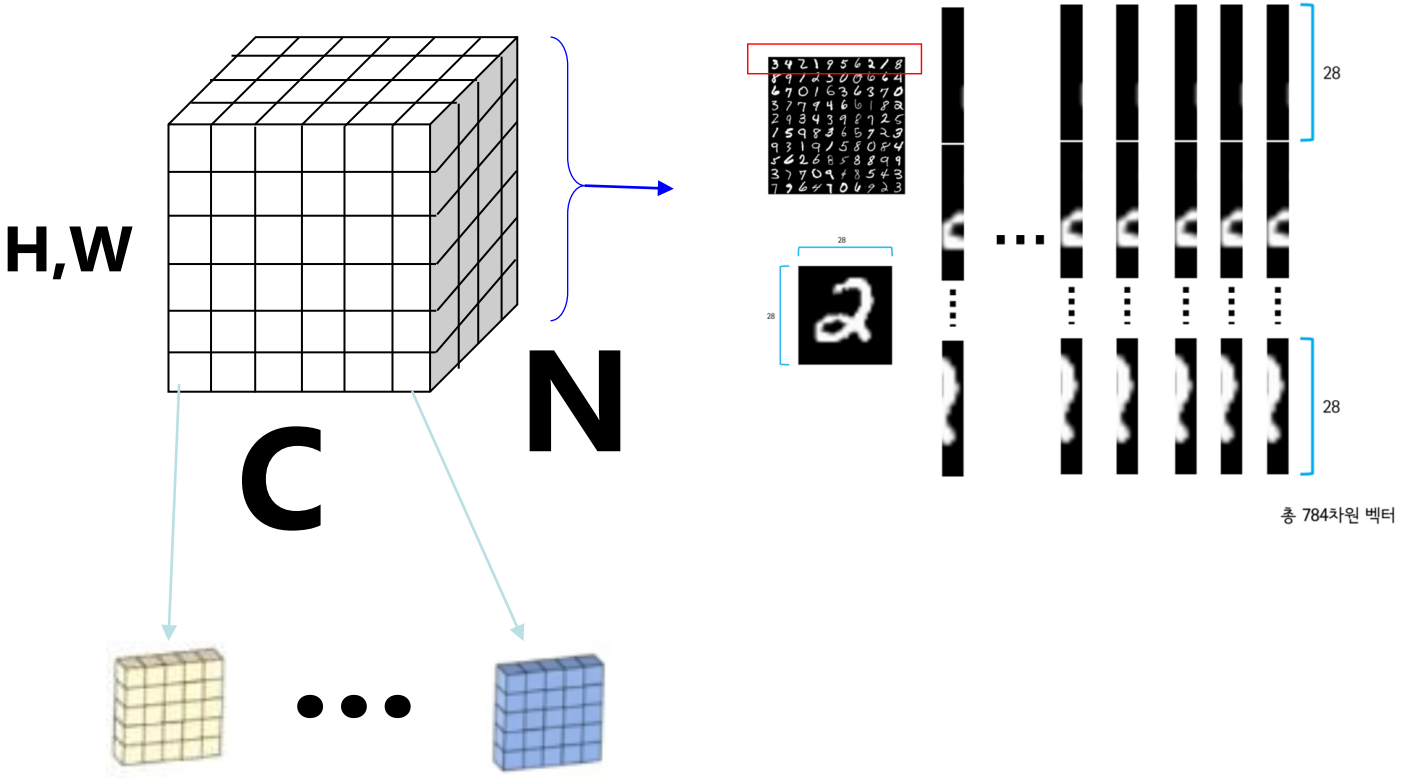


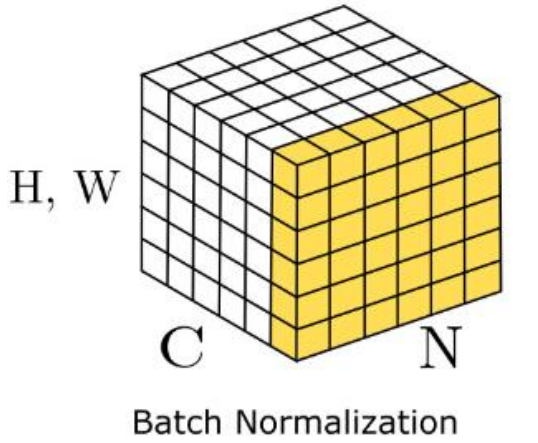




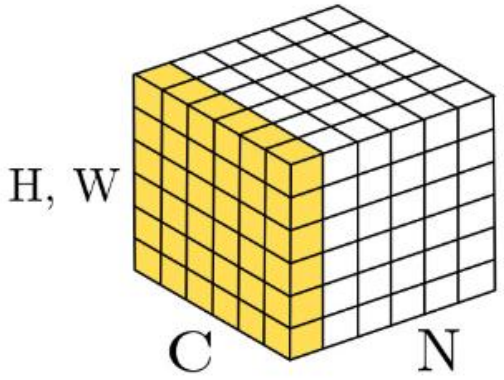


BATCH & LAYER NORMALIZATION



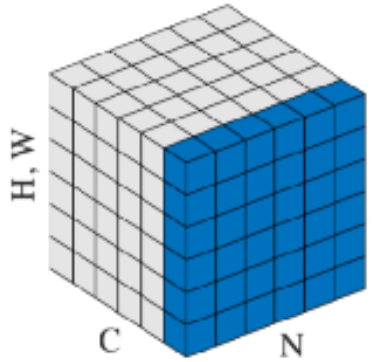
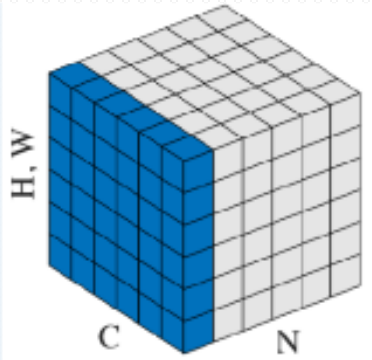


Batch Normalization



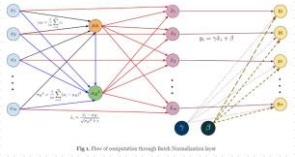
Layer Normalization

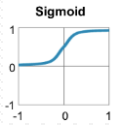
BATCH & LAYER NORMALIZATION

정규화	설명	그림	Shape 비교
Batch Norm	각 feature별로, 한 mini-batch에 대한 summed input의 평균과 분산을 구한다.		• fully-connected networks • x : N x D • $\mu, \sigma, \gamma, \beta$: 1 x D • convolutional networks • x : N x C x H x W • $\mu, \sigma, \gamma, \beta$: 1 x C x 1 x 1
Layer Norm	각 training case별로, 한 레이어의 모든 뉴런에 대한 summed input의 평균과 분산을 구한다.		• fully-connected networks • x : N x D • $\mu, \sigma, \gamma, \beta$: N x 1 • convolutional networks • x : N x C x H x W • $\mu, \sigma, \gamma, \beta$: N x 1 x 1 x 1

BATCH & LAYER NORMALIZATION

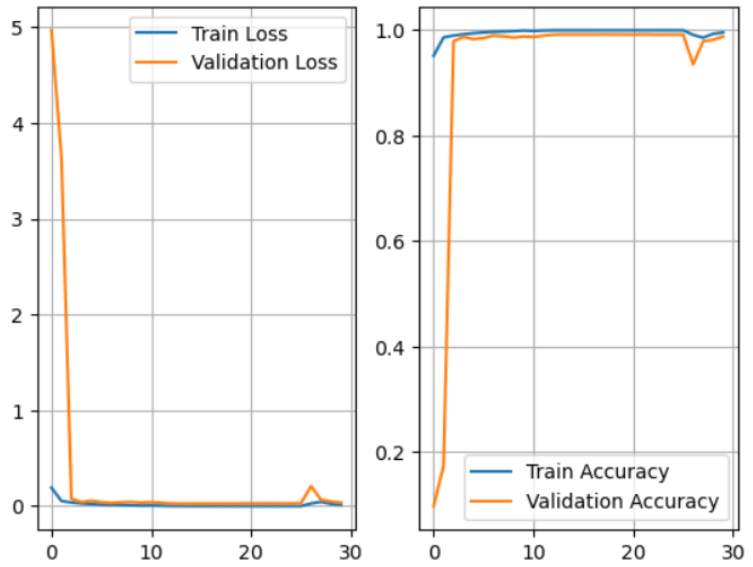
$$\Sigma(weight \cdot input) + bias$$



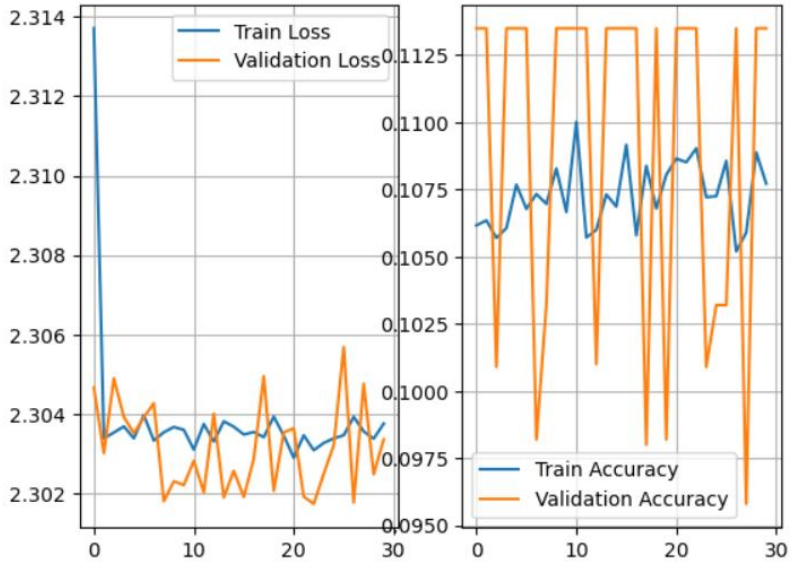


배치정규화 알고리즘

```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```




```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
# model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
# model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```



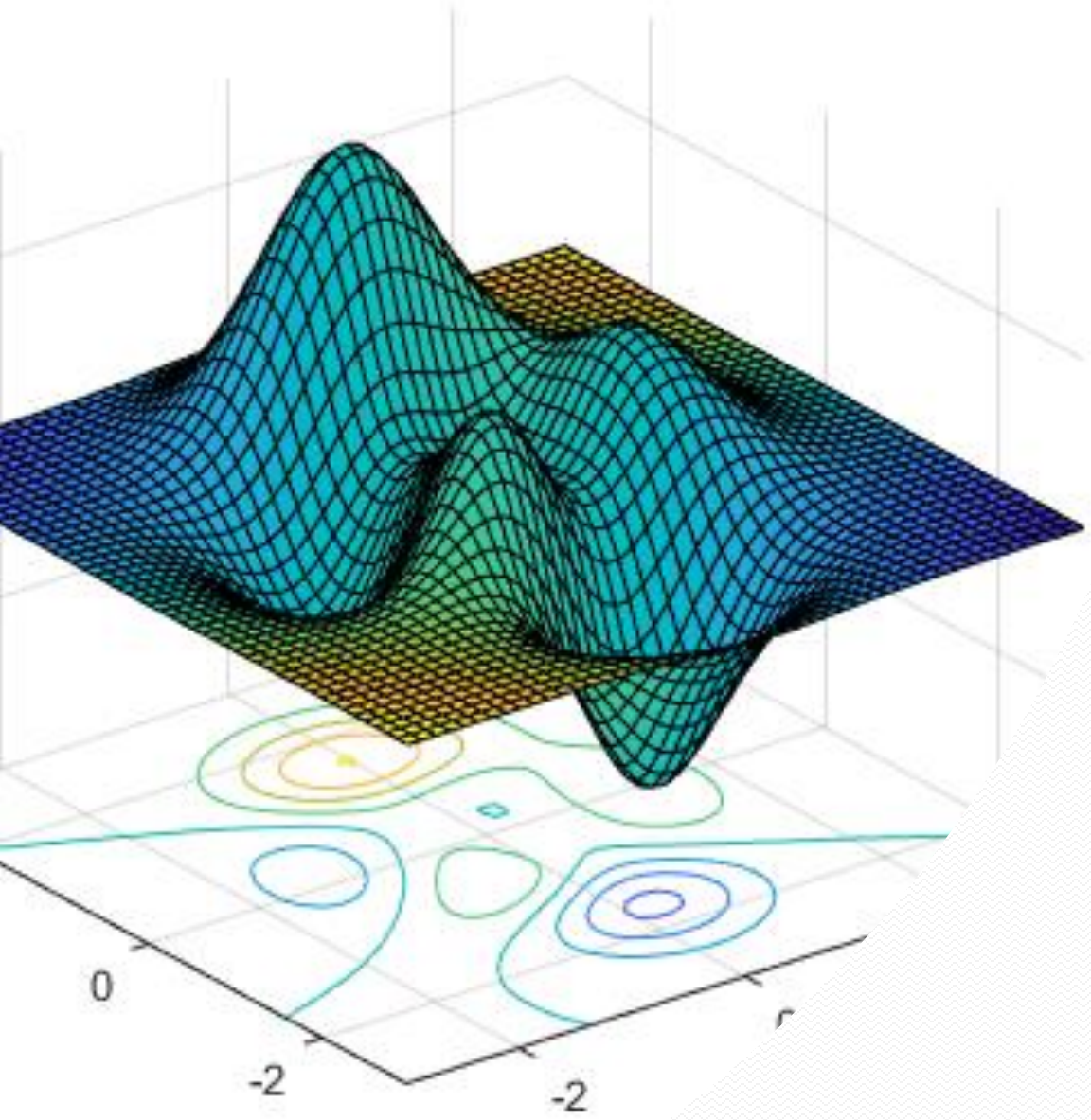
배치정규화 알고리즘

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 27, 27, 32)	160
batch_normalization (Batch Normalization)	(None, 27, 27, 32)	128
conv2d_1 (Conv2D)	(None, 26, 26, 64)	8256
batch_normalization_1 (BatchNormalization)	(None, 26, 26, 64)	256
conv2d_2 (Conv2D)	(None, 13, 13, 128)	32896
batch_normalization_2 (BatchNormalization)	(None, 13, 13, 128)	512
conv2d_3 (Conv2D)	(None, 12, 12, 32)	16416
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 32)	128
conv2d_4 (Conv2D)	(None, 11, 11, 64)	8256
batch_normalization_4 (BatchNormalization)	(None, 11, 11, 64)	256
conv2d_5 (Conv2D)	(None, 5, 5, 128)	32896
batch_normalization_5 (BatchNormalization)	(None, 5, 5, 128)	512
flatten (Flatten)	(None, 3200)	0
dense (Dense)	(None, 128)	409728
dense_1 (Dense)	(None, 10)	1290

Total params: 511690 (1.95 MB)
Trainable params: 510794 (1.95 MB)
Non-trainable params: 896 (3.50 KB)

None

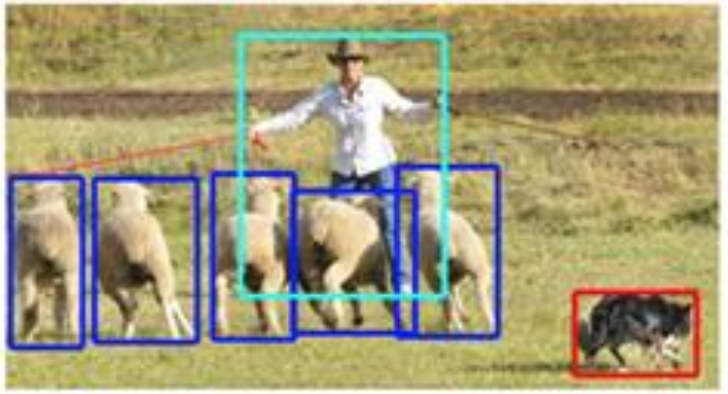
```
model = Sequential()
model.add(Conv2D(32,(2,2),activation="sigmoid",input_shape=(28,28,1)))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(32,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(64,(2,2),activation="sigmoid"))
model.add(BatchNormalization())
model.add(Conv2D(128,(2,2),2,activation="sigmoid"))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(128,activation="sigmoid"))
model.add(Dense(10,activation="softmax"))
```



USE CASES

CNN USE CASES

CLASSIFICATION



DETECTION

SEGMENTATION

NEURAL STYLE TRANSFER

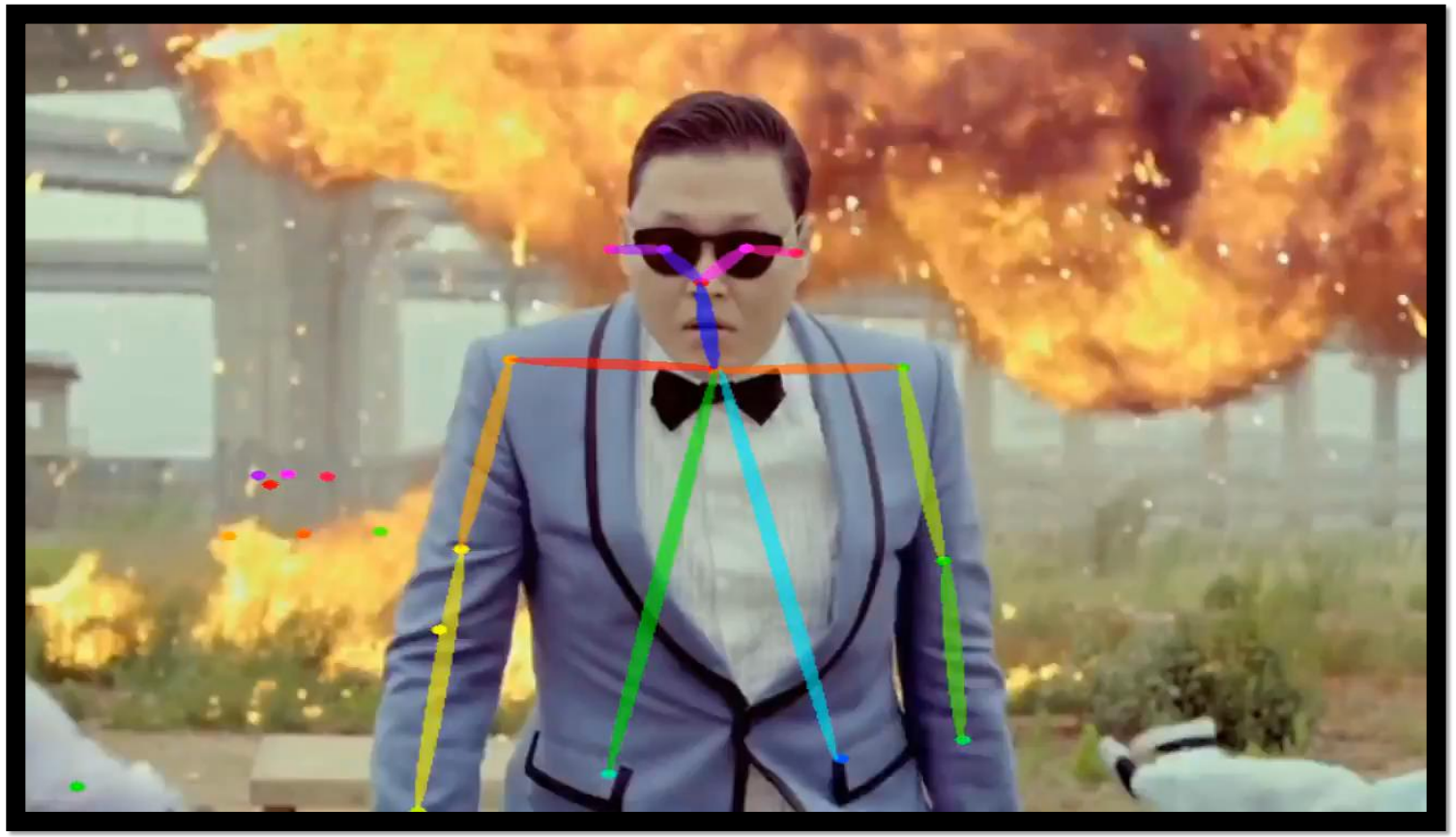
INPAINTING

COLORIZATION

CNN USE CASES / CONT.

SUPER RESOLUTION

MONO DEPTH



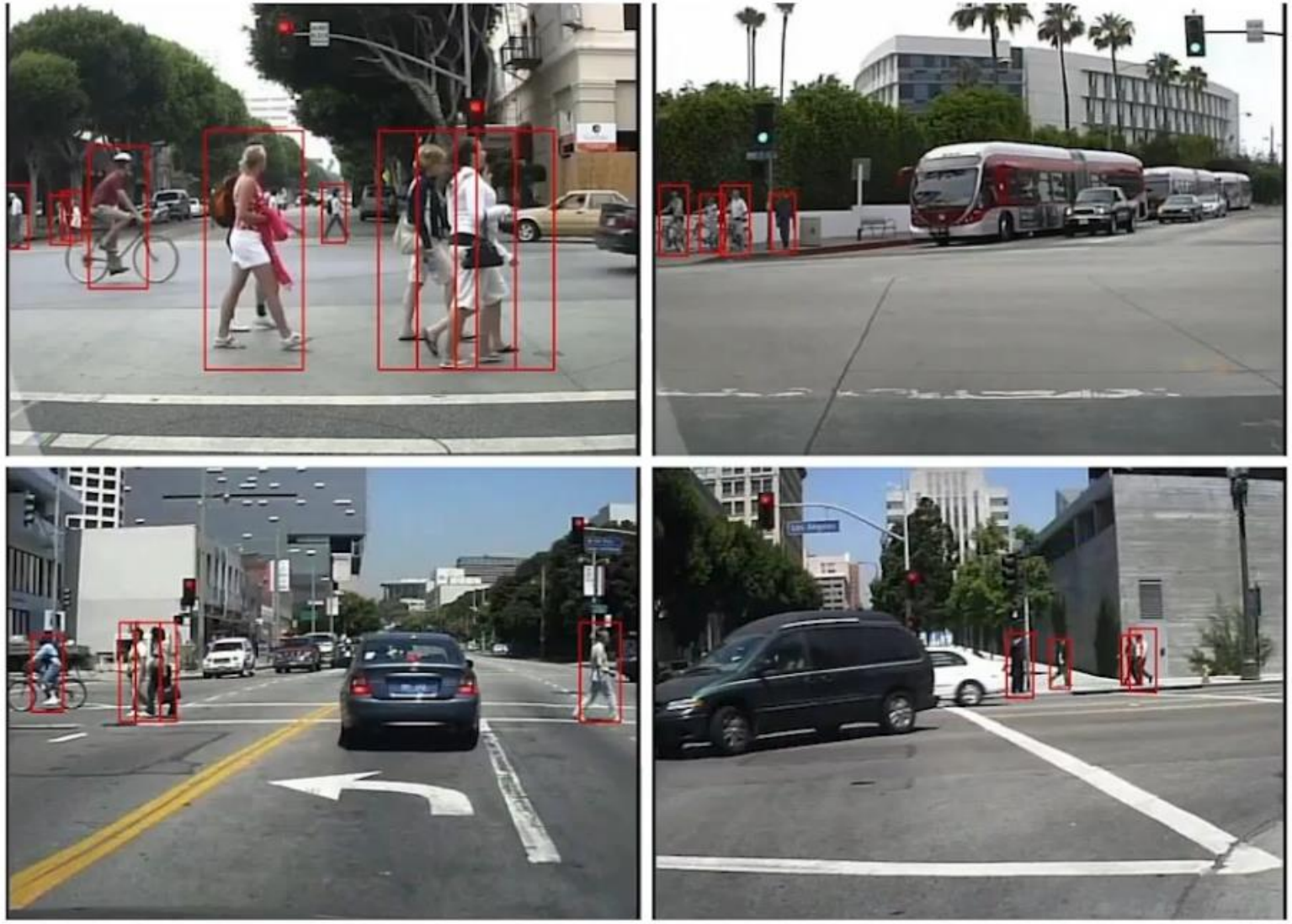
HUMAN POSE ESTIMATION

CNN USE CASES / CONT.

A young boy is playing basketball. 	Two dogs play in the grass. 	A dog swims in the water. 	A little girl in a pink shirt is swinging. 
A group of people walking down a street. 	A group of women dressed in formal attire. 	Two children play in the water. 	A dog jumps over a hurdle. 

IMAGE CAPTIONING

CNN USE CASES / CONT.



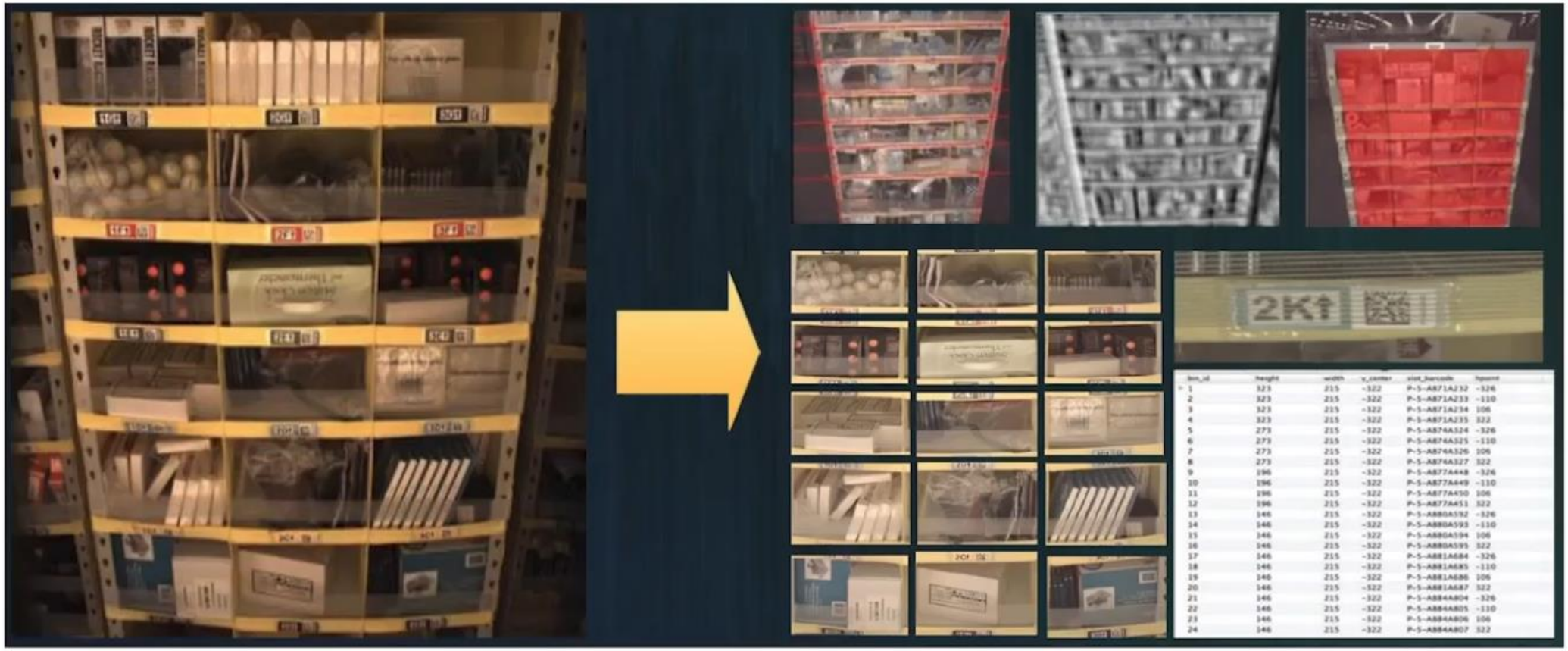
CNN USE CASES / CONT.

누가 지각/결석을 했지?

수업이 얼마나 따분한가?

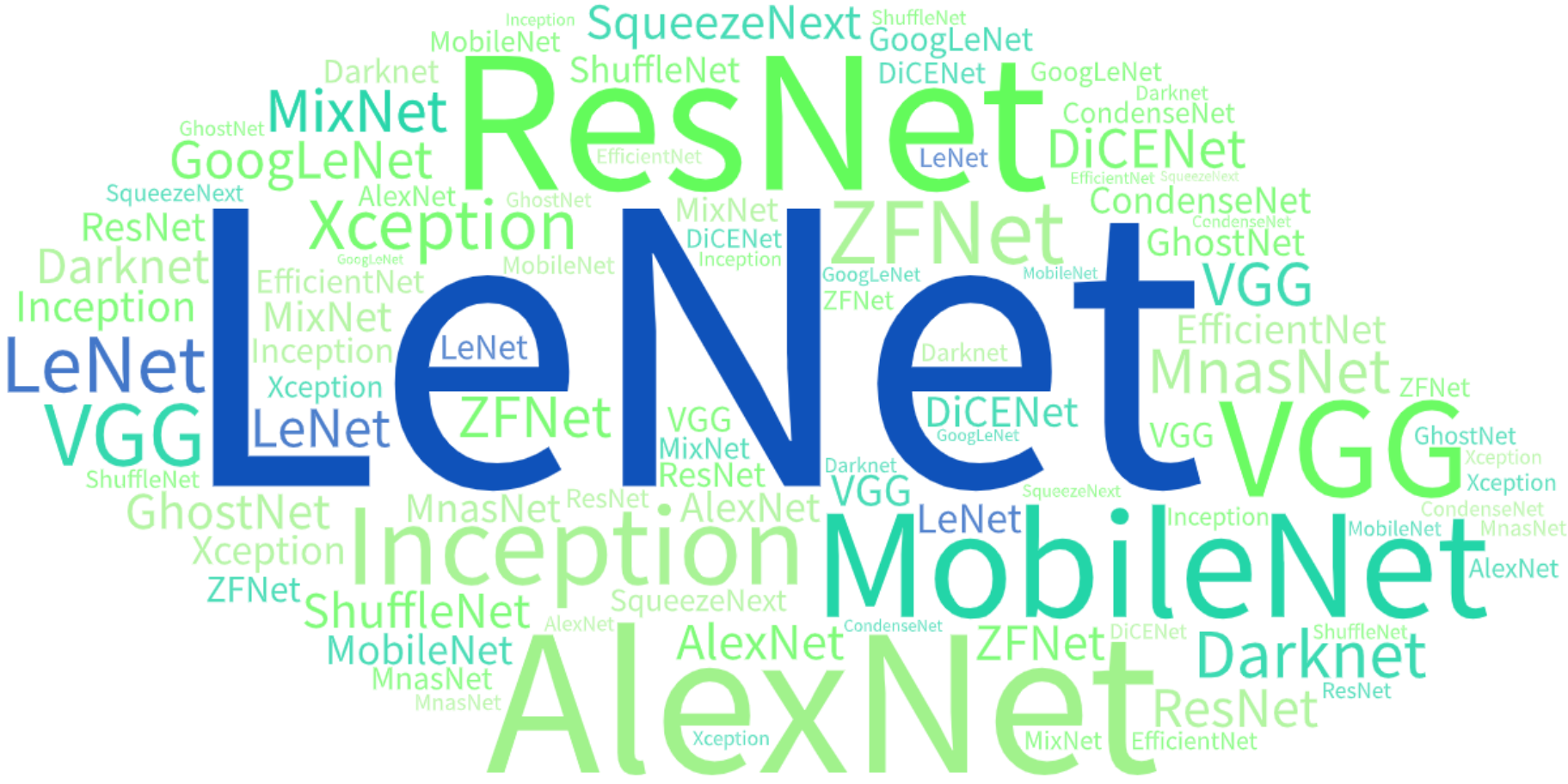
CNN USE CASES / CONT.

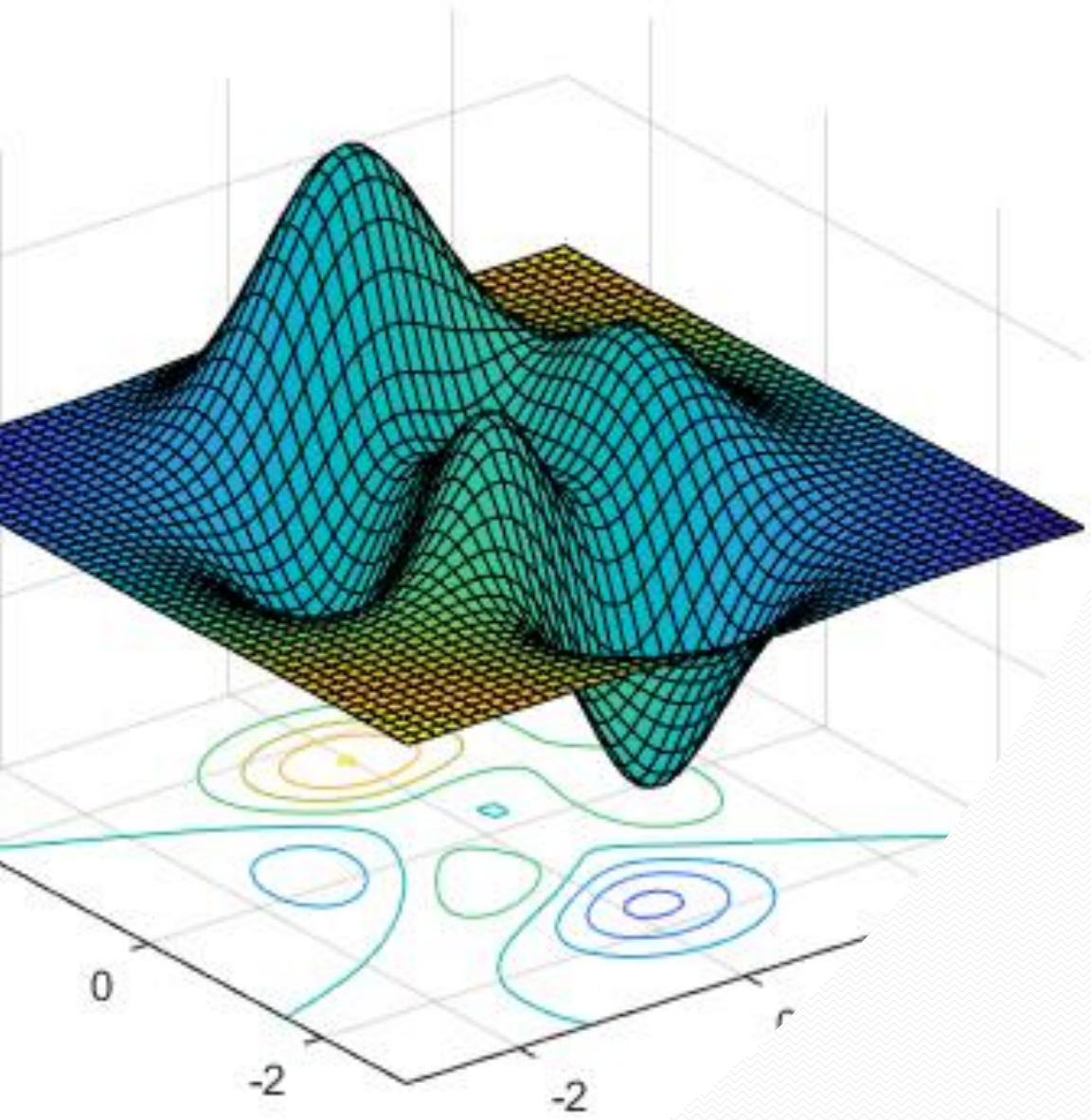
비즈니스 활용 사례 : 재고 파악



https://www.slideshare.net/awskorea/amazon-deeplearningcasesandmlonaws

NETWORKS WITH CNN





Classification

IMAGENET LARGE SCALE VISUAL RECOGNITION CHALLENGE ILSVRC

✓ **Layer의 수가 많아질수록 이미지 분류 성능이 증가하는 추세를 보임**

2010 - NEC-UIUC Lin et al.

2011 - XRCE Florent Perronnin, Jorge Sanchez

2012 - AlexNet

2013 - ZFNet

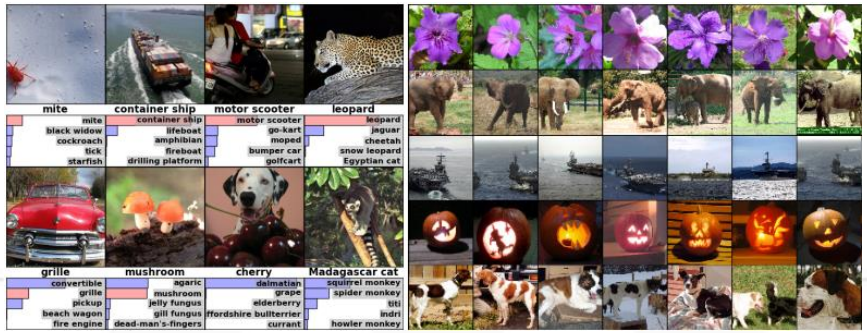
2014 – GoogLeNet

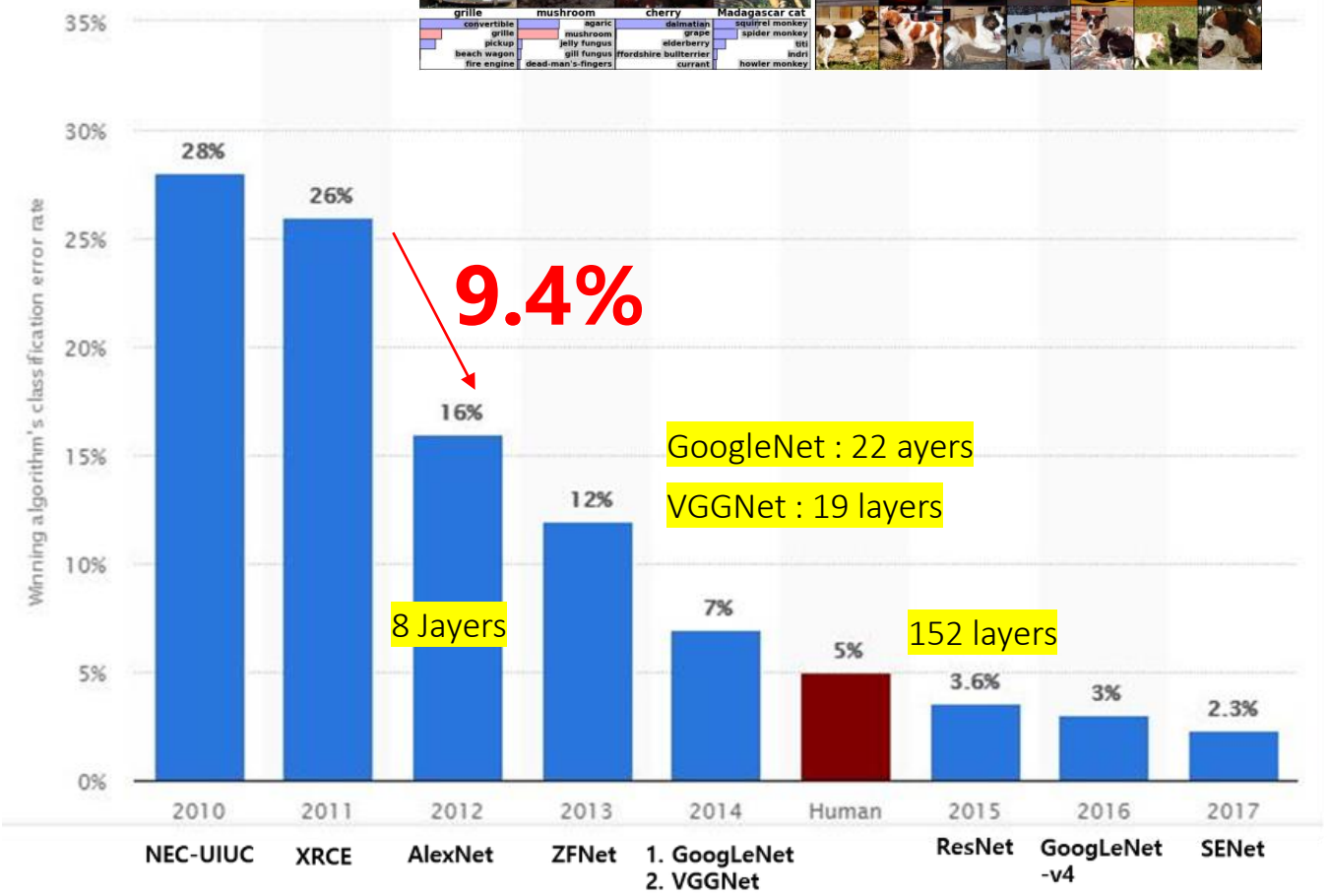
VGGNet Second Winner

2015 - ResNet

2016 - GoogLeNet-v4

2017 - SENet





LeNET-5 ARCHITECTURE

Yann LeCun은 CNN 개념을최초로 개발했으며 LeNet은 Yann LeCun이 개발한 CNN architecture 이다.

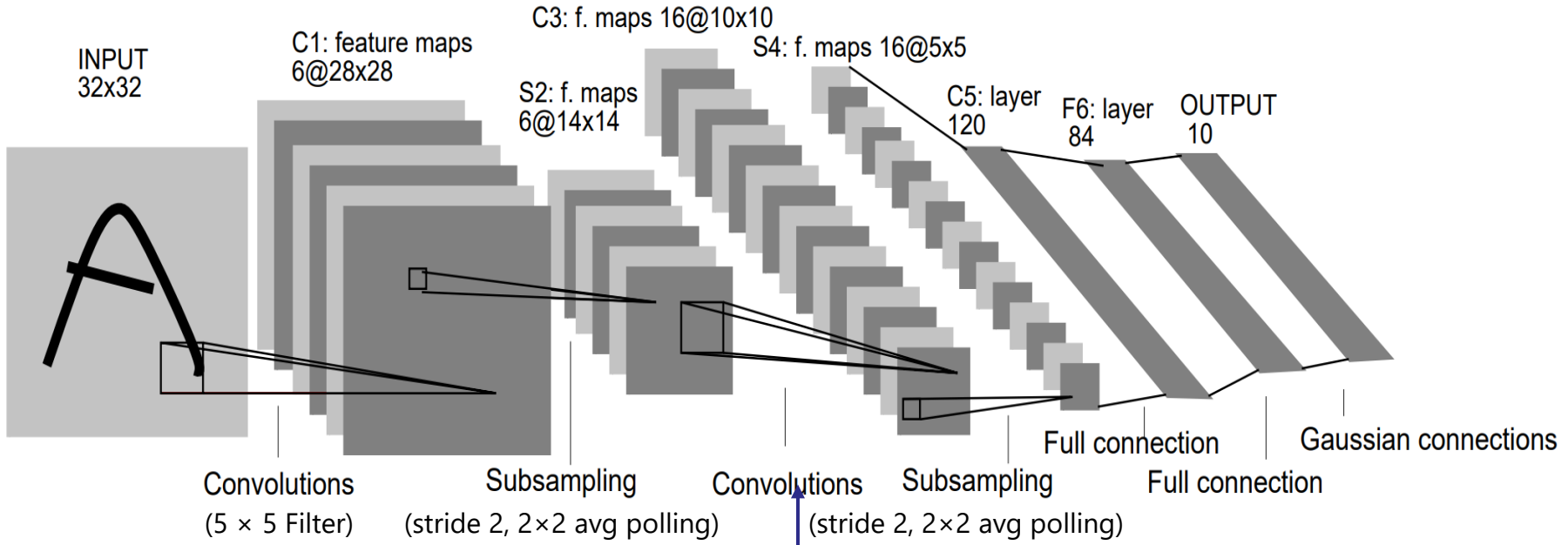


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Channel

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

<-논문의 작성자가
임의로 선택한 값

http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

AlexNet ARCHITECTURE

ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation

LeNet

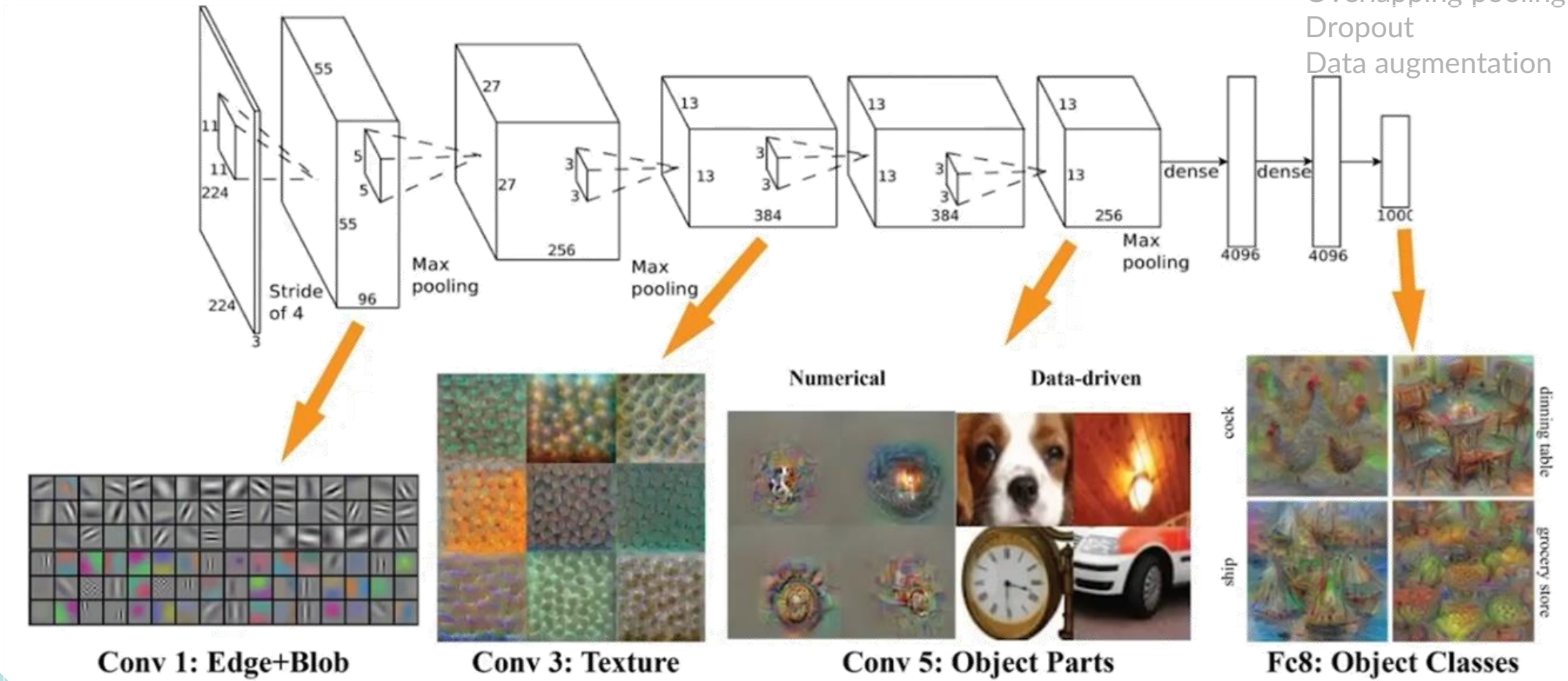
AlexNet

https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

AlexNet ARCHITECTURE

AlexNet은 분류문제의 오차율을 25% 에서 16%로 급격한 성능 향상을 보인 모델이다.

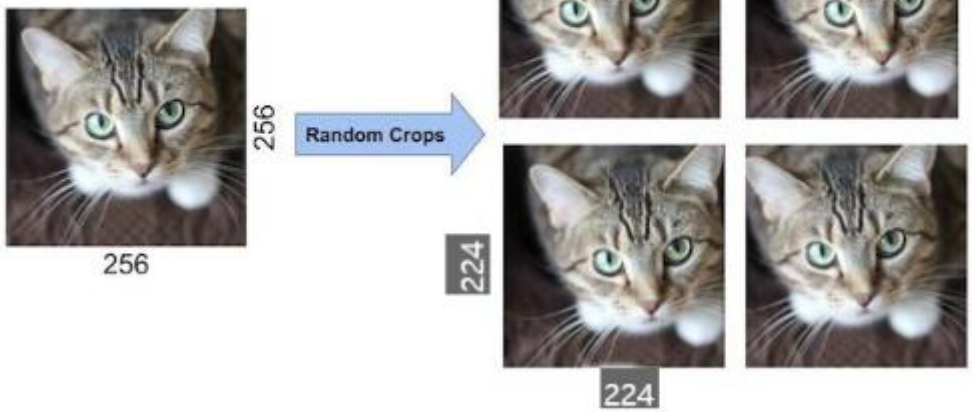
ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation

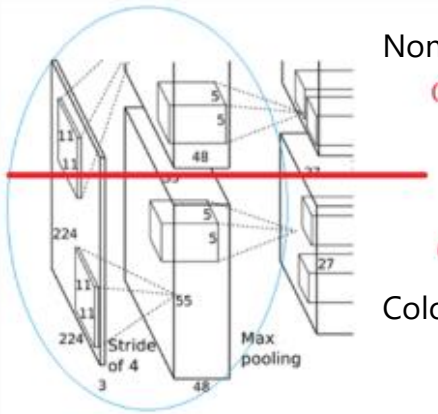


https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

AlexNet ARCHITECTURE

ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation



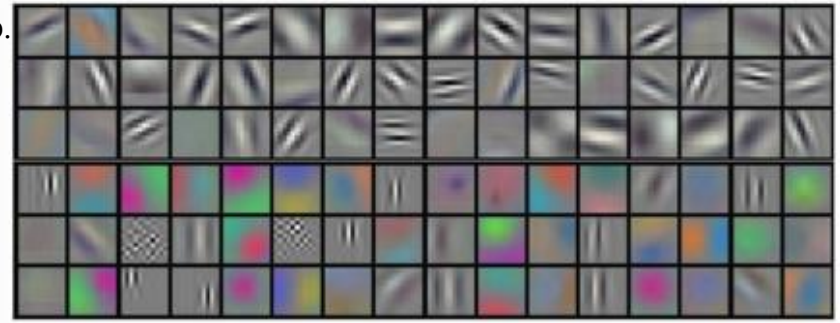


Non-Colored info.

GPU-1

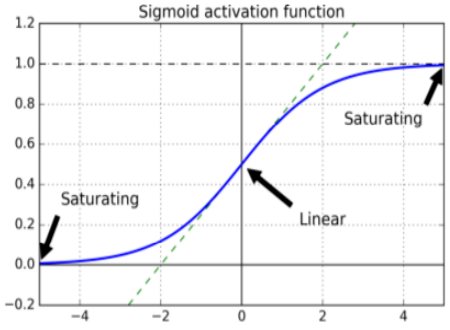
GPU-2

Color info.

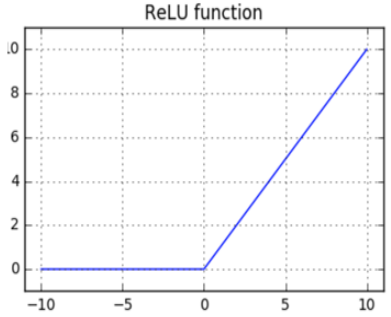


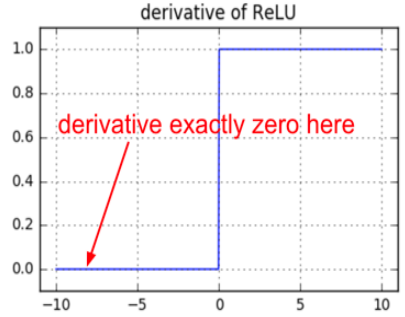
Filters in a first convolution layer

Training speed slow down in the saturating point due to a small gradient.

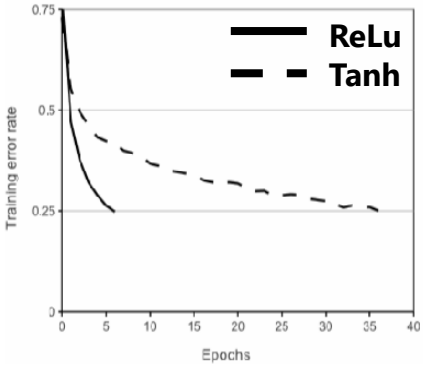


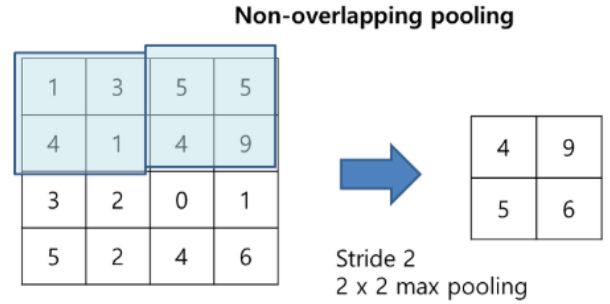
Specific values can increase infinitely.
Therefore, neighboring value can be ignored.
(this issue was solved by normalization (LRN)).

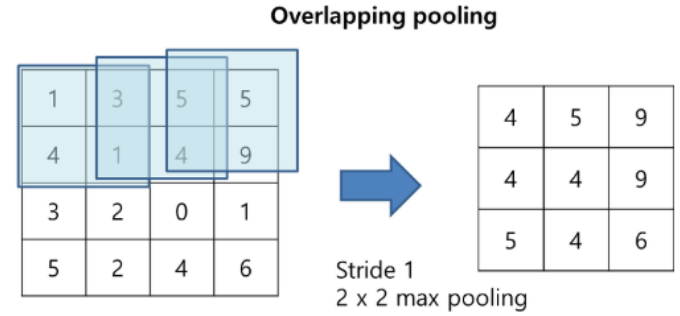




ReLu shows faster training speed compared to Tanh.



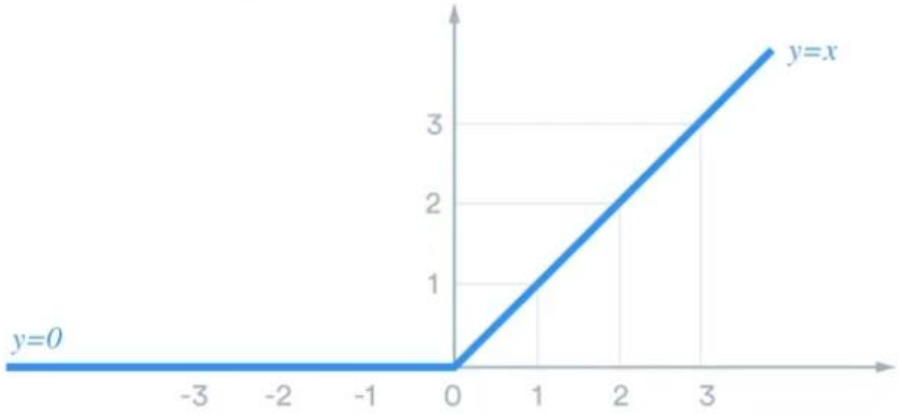


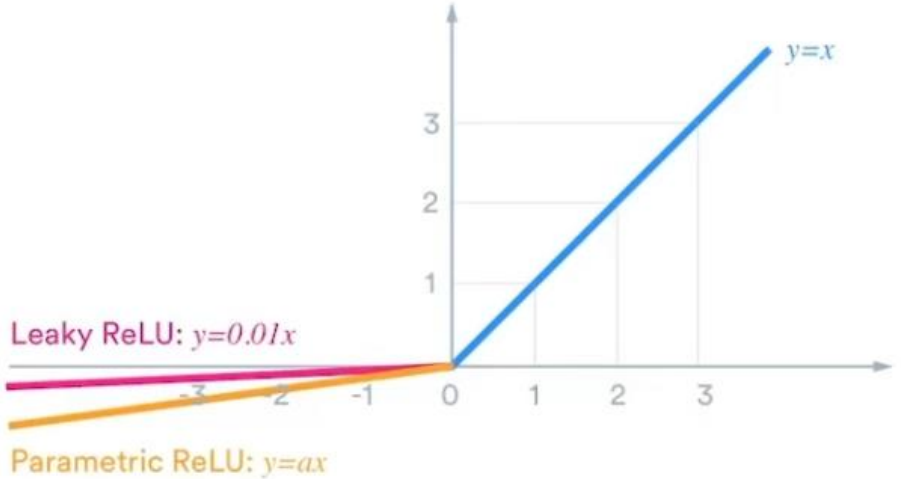


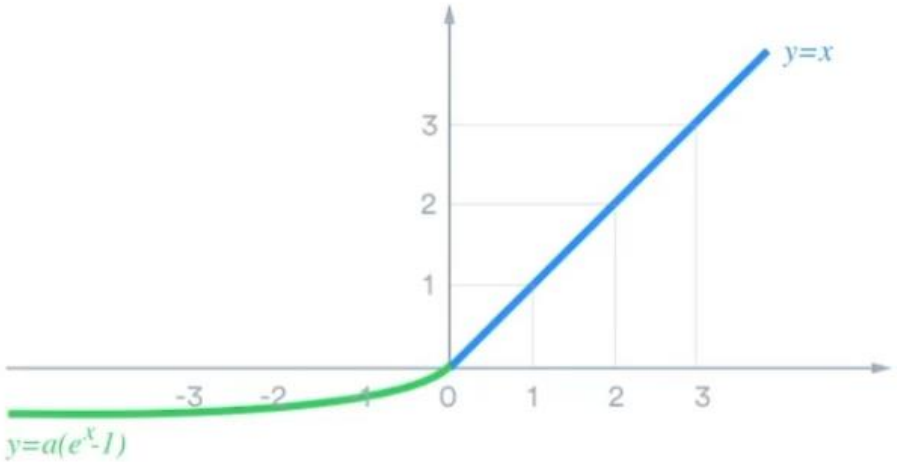
https://oi.readthedocs.io/en/latest/computer_vision/cnn/alexnet.html

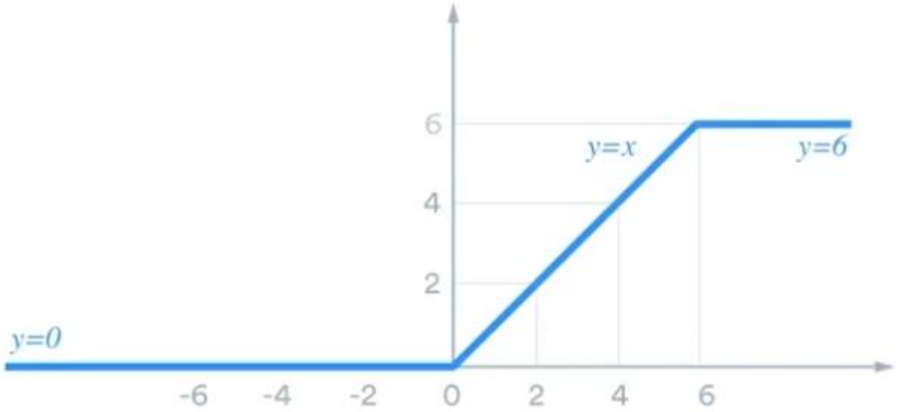
AlexNet ARCHITECTURE:ReLu

ReLU(Rectified Linear Unit) 사용 → 최근에는 변형 사용





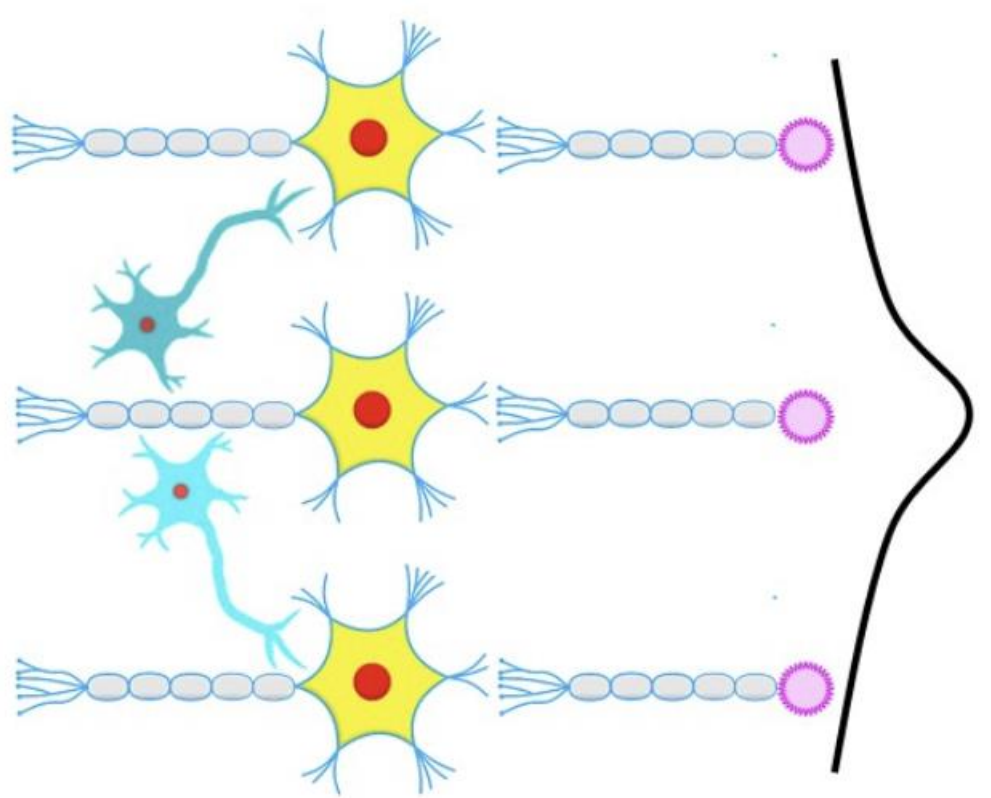


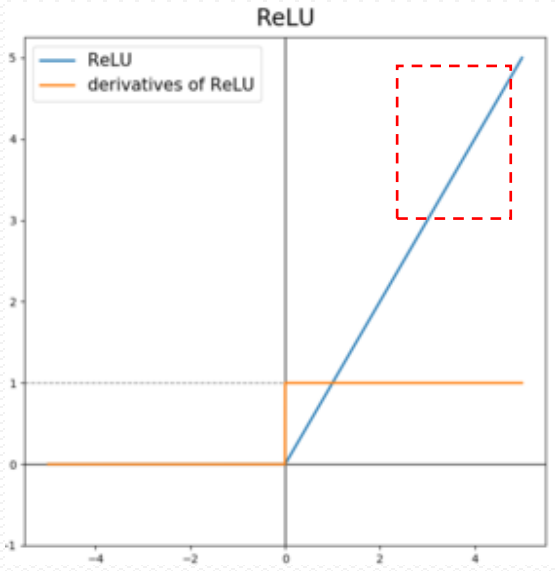


https://www.tinymind.com/learn/terms/relu

AlexNet ARCHITECTURE:ReLu vs 측면억제

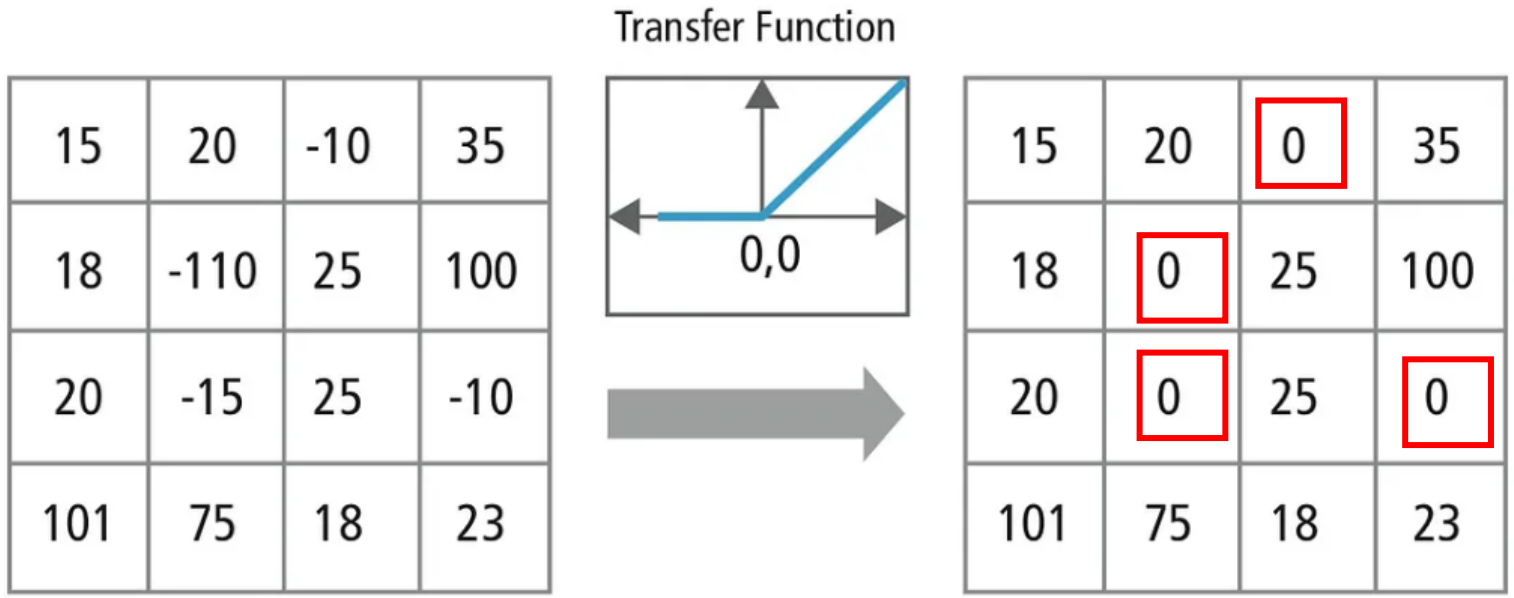
입출력단을 갖는 2개의 신경세포에서 각각의 신경세포의 입력 또는 출력이 서로 다른 신경세포의 입력 또는 출력에 의해 억제되는 현상.





Activation

✓ **Convolution을 통해 학습된 값들의 비선형 변환을 수행**
✓ **대부분 Rectified Linear Unit (ReLU)을 사용**

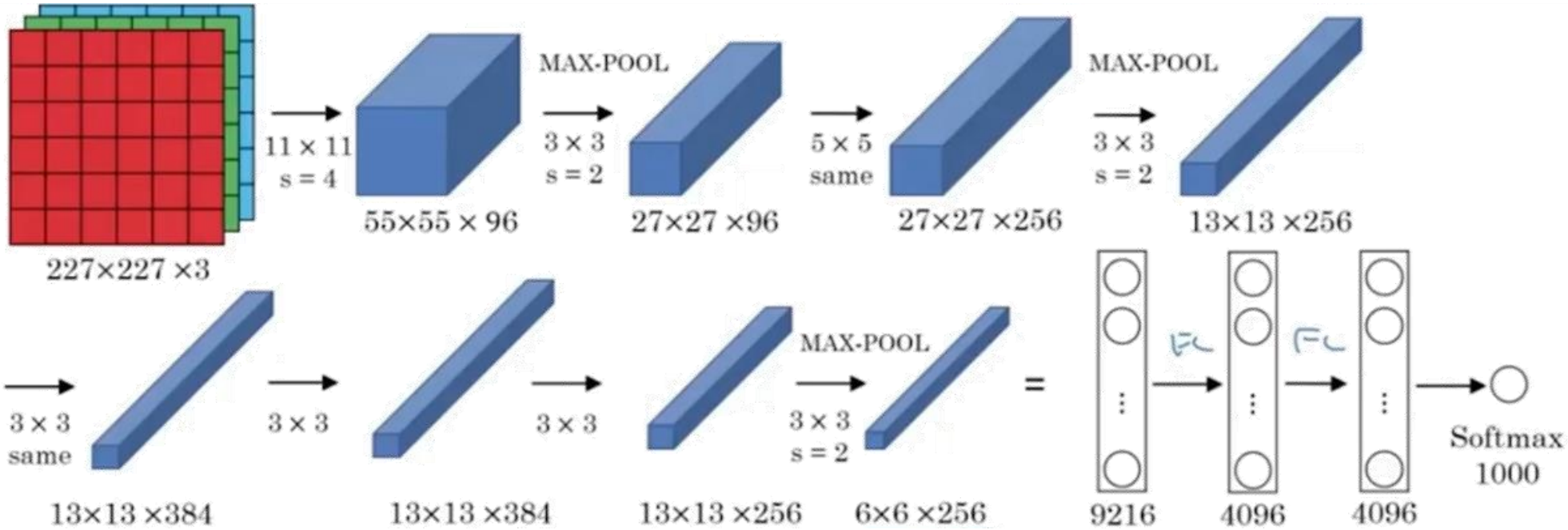


https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

AlexNet ARCHITECTURE

- ✓ 224*224 크기의 이미지를 Input으로 사용
- ✓ 초기 단계에서는 큰 필터 사이즈와 Stride를 사용
- ✓ 상위 Layer로 갈수록 작은 필터 사이즈와 Stride를 사용
- ✓ 2개의 Fully connected layer 존재
- ✓ 파라미터 총 수 : 60 million

ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation

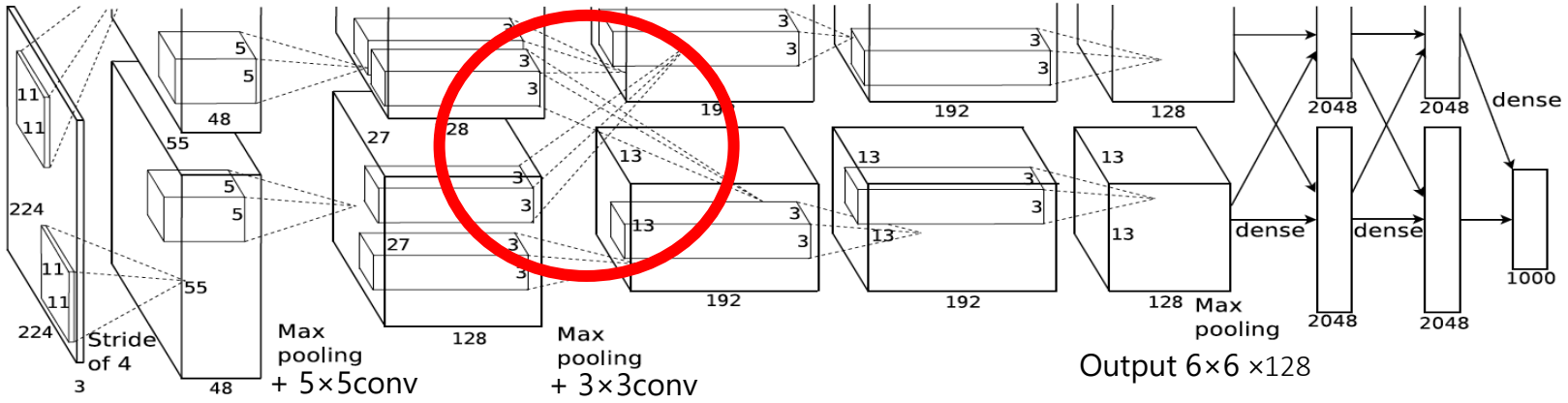


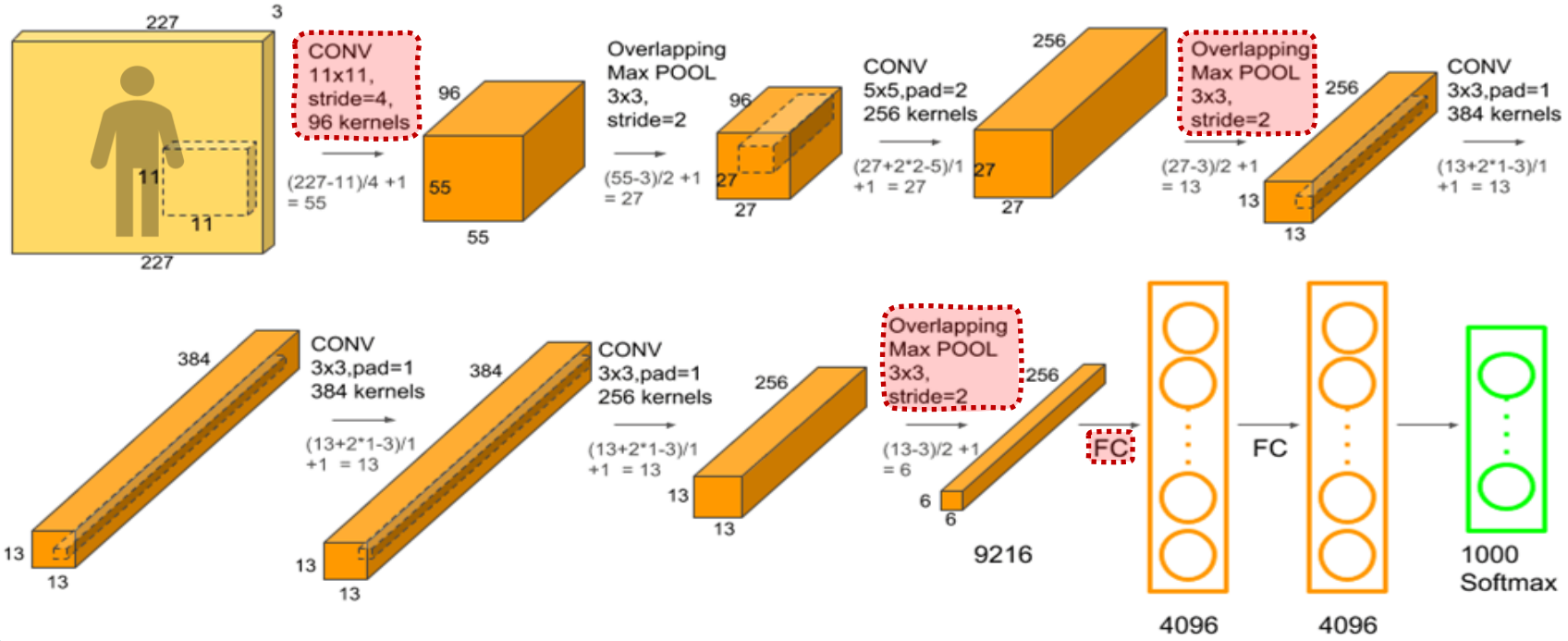
[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]

AlexNet ARCHITECTURE

ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation

AlexNet은 분류문제의 오차율을 25% 에서 16%로 급격한 성능 향상을 보인 모델이다.



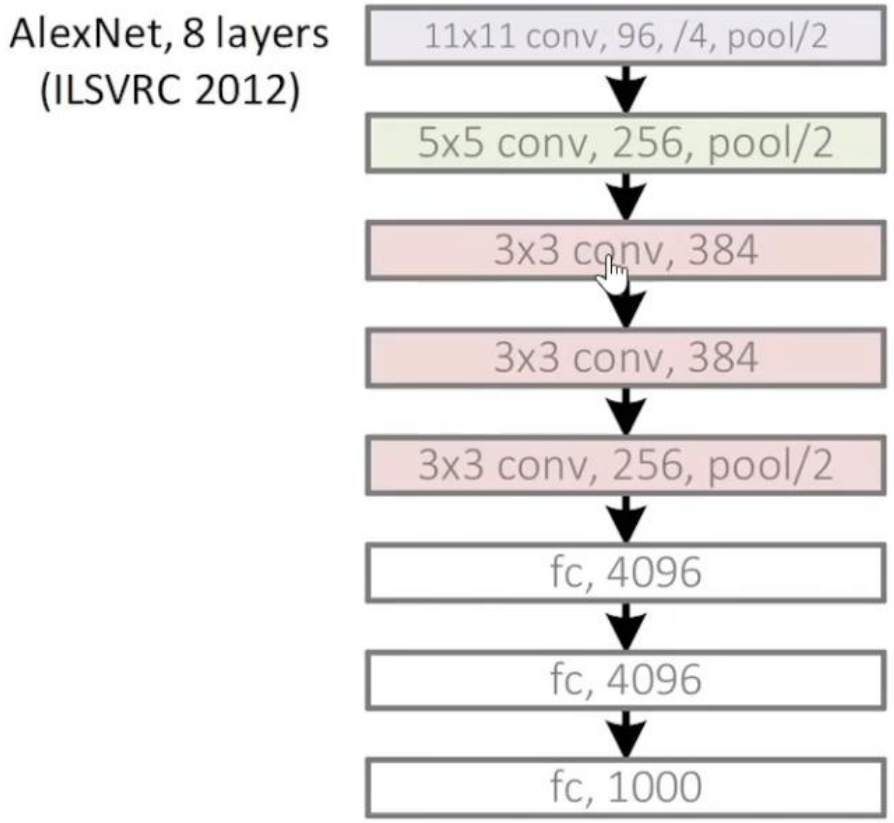


https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

AlexNet ARCHITECTURE

✓ 최초의 ReLu 함수 제시
✓ 최초의 GPU 사용

ReLU + Local response normalization
Multiple GPU
Overlapping pooling
Dropout
Data augmentation



https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf

VGG, 2014

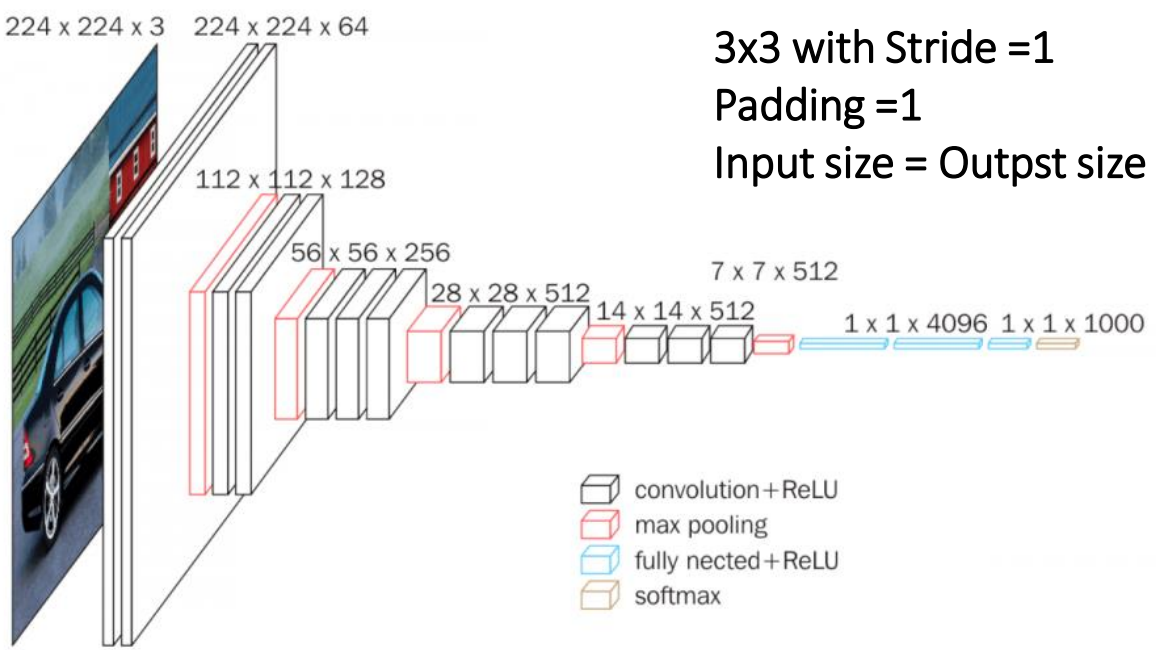
✓ The key is to see how deeper the depth of the network affects performance

✓ AlexNet에 비해 단순하지만 깊은 구조로 단순하면서 성능이 좋은 네트워크

✓ 3 by 3 convolution with stride 1을 기본 연산으로 하며 중간 중간에 2 by 2 max polling을 수행

✓ 파라미터 총 수 : 138 million

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



3x3 with Stride =1
Padding =1
Input size = Outpst size

- conv[필터 사이즈]-[필터 개수]의 방식으로 표기.
- 이 중 16레이어와 19레이어로 구성된 VGG16과 VGG19가 오류율이 낮아서 많이 사용.

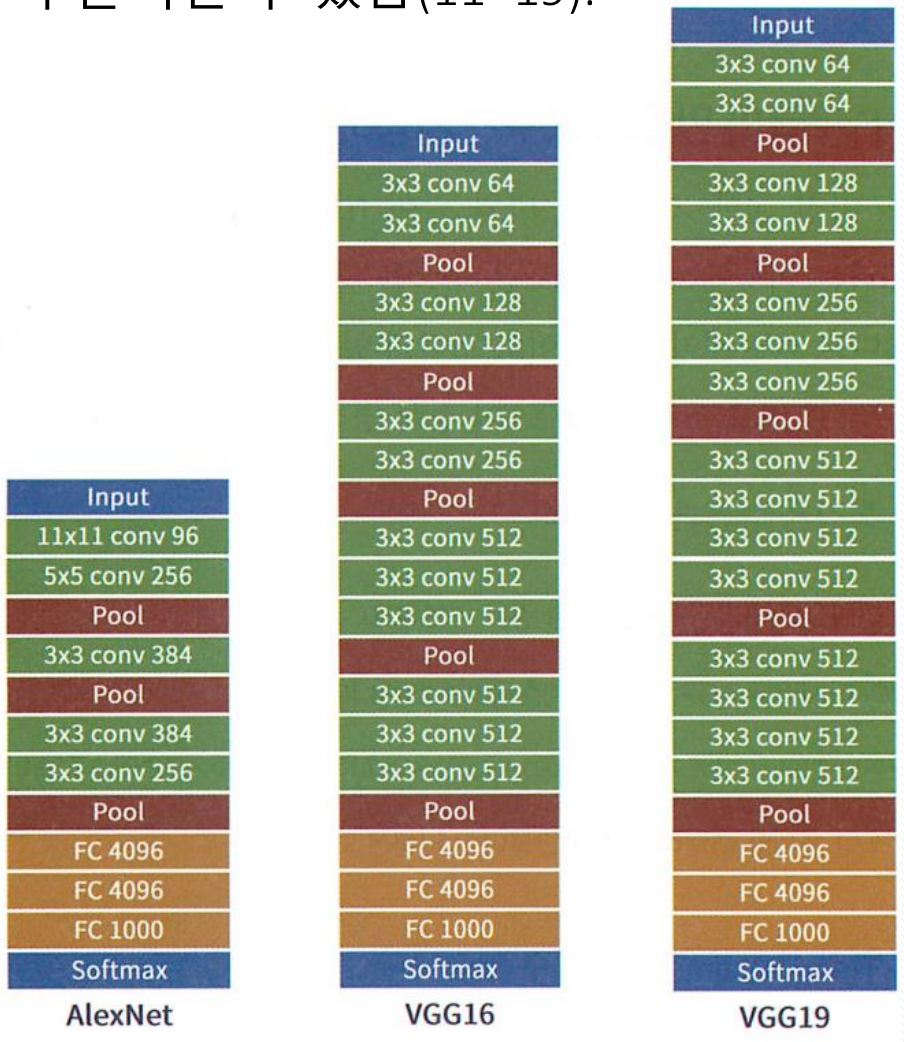
VGG, 2014

Factorizing Conv.
Depth increased

✓ VGGNet에서는 AlexNet에서 사용한 큰 사이즈의 필터가 없고 3×3 필터와 1×1 필터만 사용.

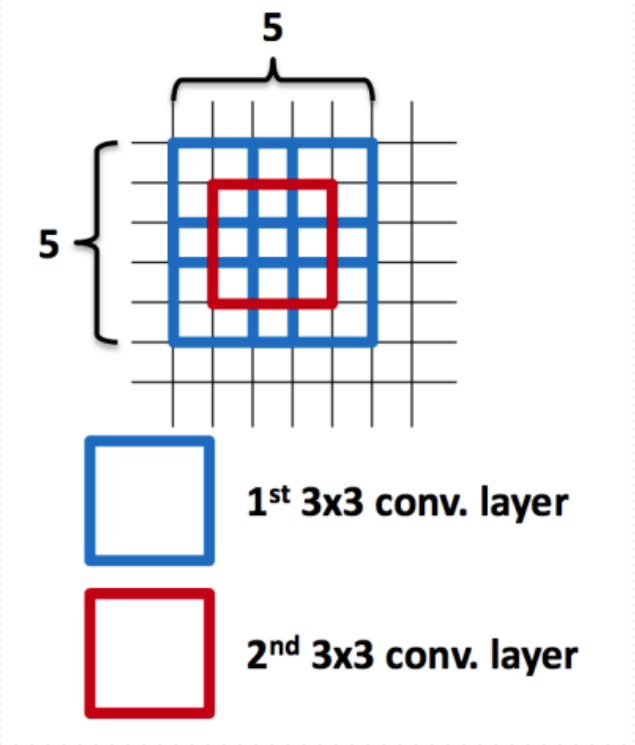
✓ AlexNet보다 레이어의 수가 늘어날 수 있음(11~19).

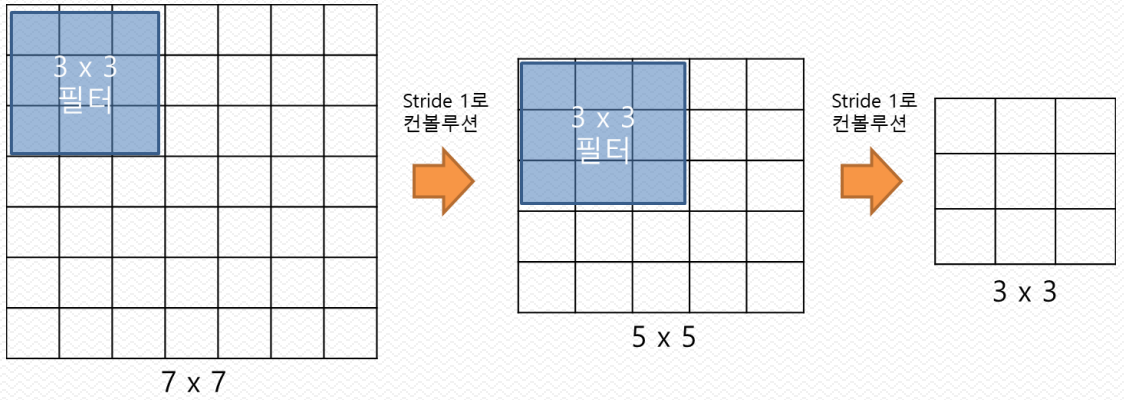
✓ 1×1 콘볼루션을 사용.

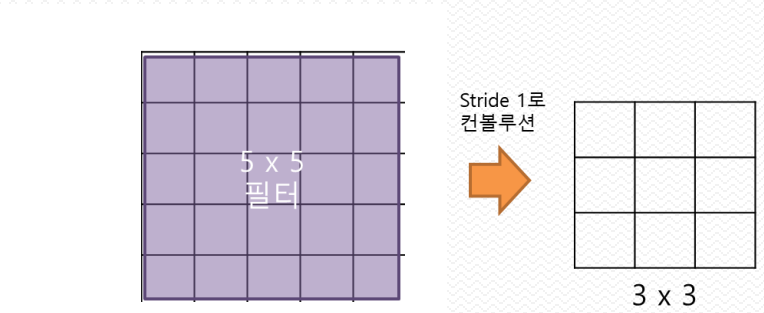


VGGNet Architecture

- 3x3 conv. Layer
  - Stacked conv. layers have a large receptive field
    - 3x3 (x2) – 5x5 receptive field
    - 3x3 (x3) – 7x7 receptive field
  - Less parameters to learn





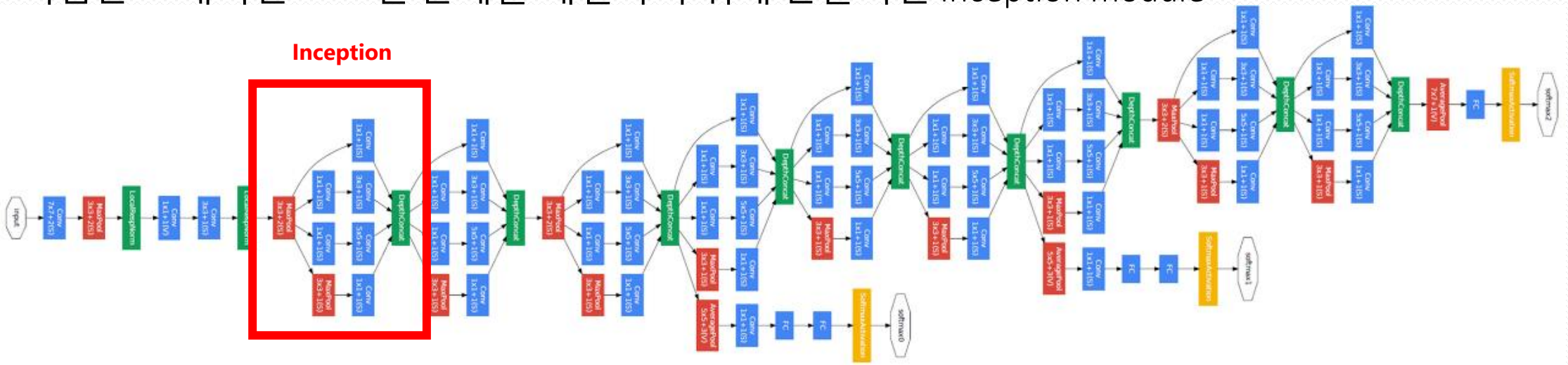


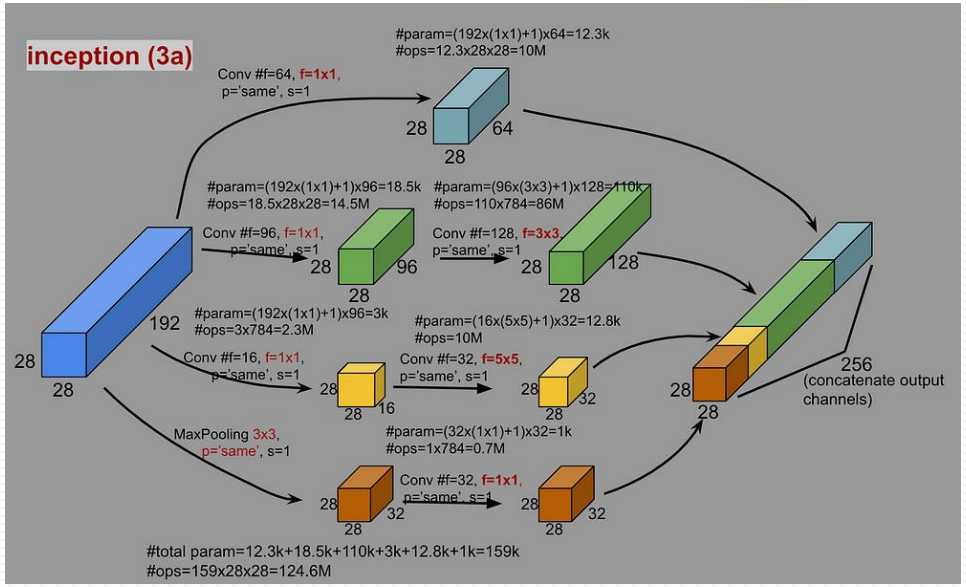
https://bskyvision.com/504

GoogLeNet (Inception), 2014

Inception Module
Depth increased

- ✓ 구글이 만들고 모두가 사용하는 22계층으로 이뤄진 네트워크
- ✓ 핵심은 그래디언트 소실 문제를 해결하기 위해 만들어진 Inception module

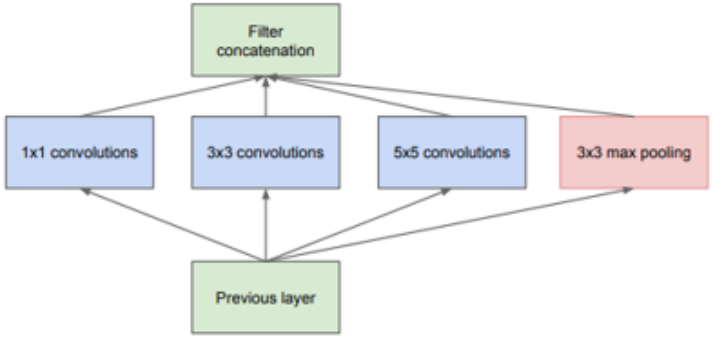




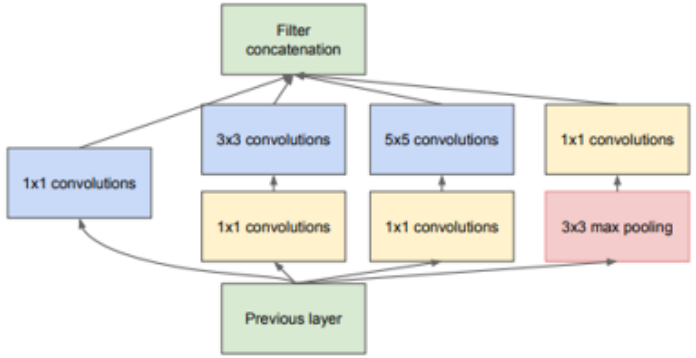
https://arxiv.org/pdf/1409.4842.pdf

GoogLeNet Architecture

1. Inception Module



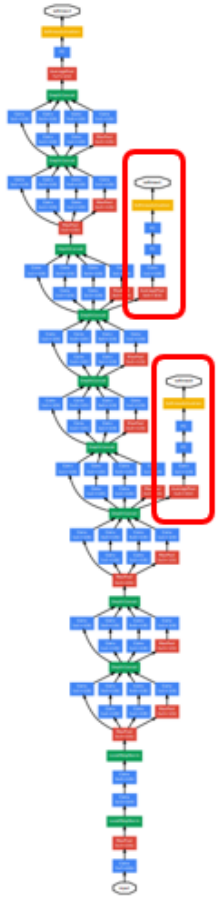
(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

Figure 2: Inception module

2. Auxiliary Classifier



AvgPool

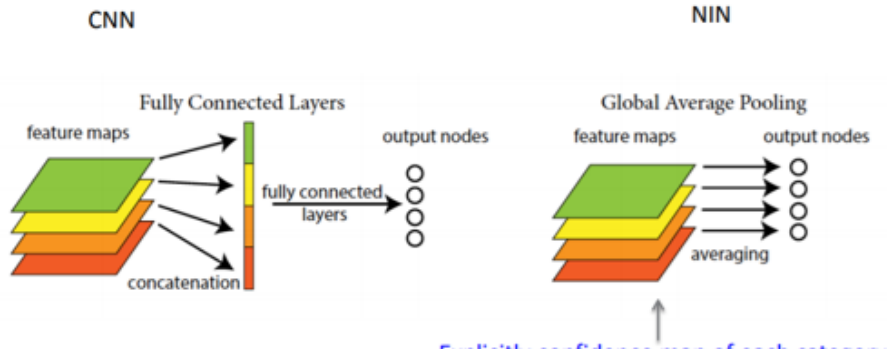
1x1Conv

FC

FC

Softmax

3. FC layer → Global Average Pooling



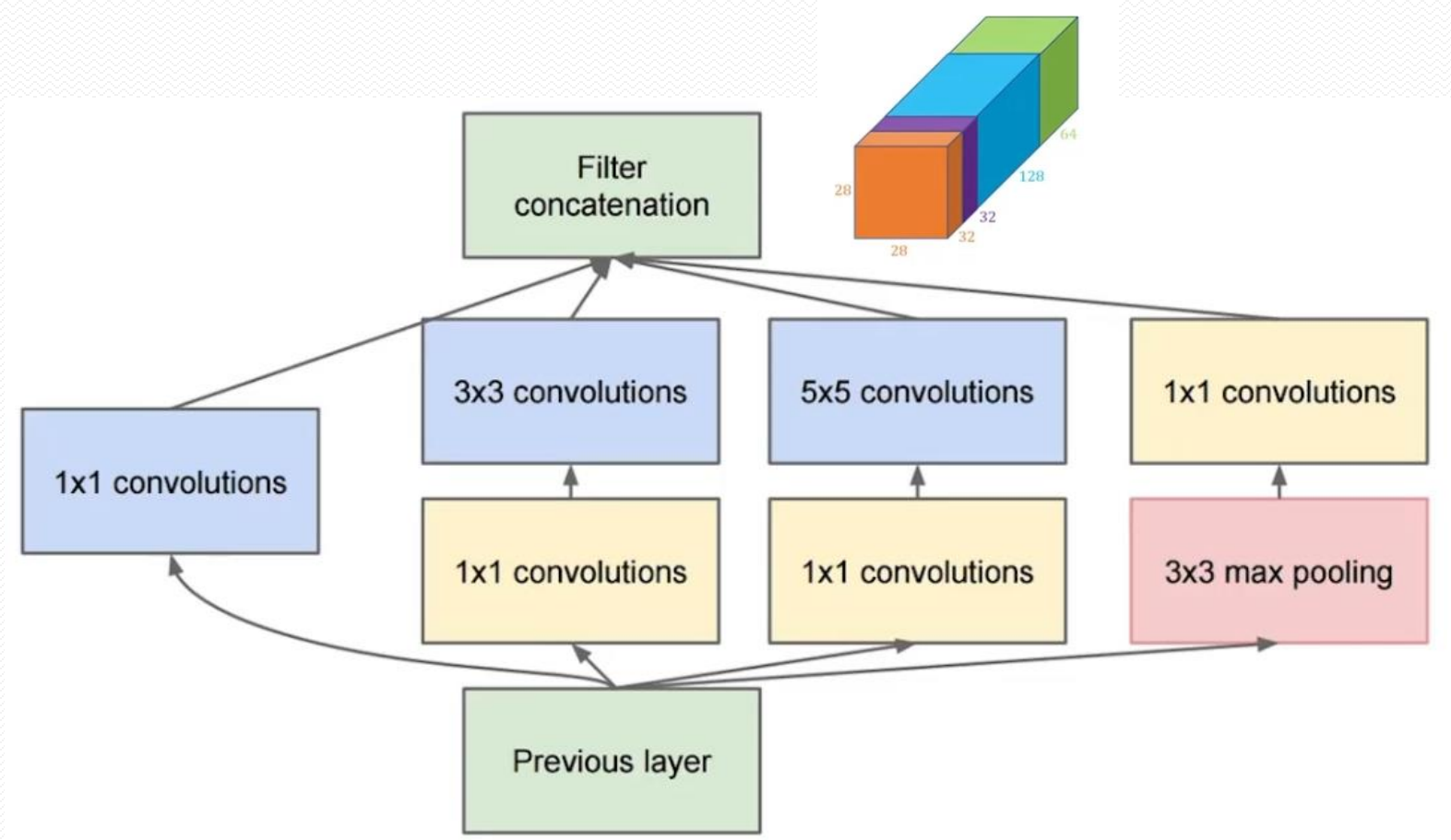
Save a large portion of parameters

inception (5b)		7×7×1024
avg pool	7×7/1	1×1×1024
dropout (40%)		1×1×1024
linear		1×1×1000
softmax		1×1×1000

Inception module

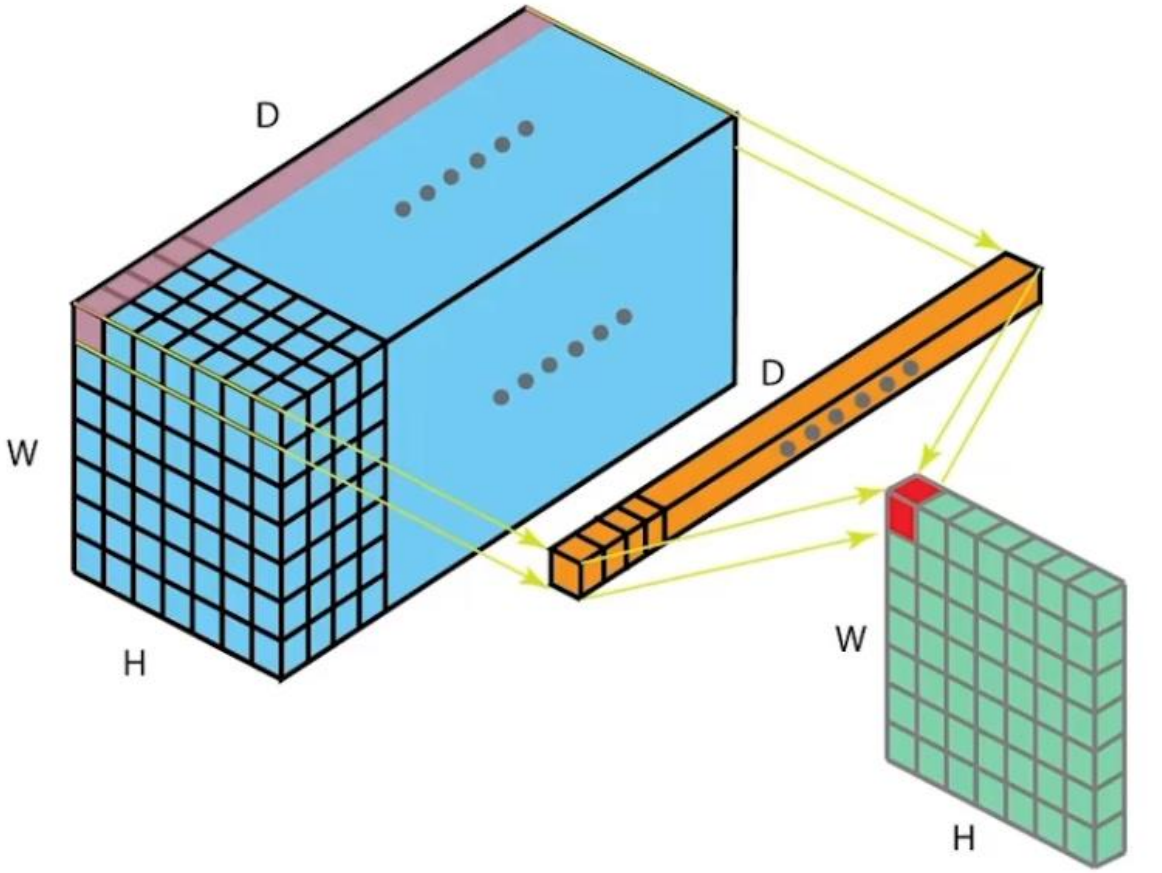
✓ 왜 한 입력 image에 한 종류의 filter size만 적용해야 하는가?

✓ 뭐가 잘 될지 모르니 1 by 1, 3 by 3, 5 by 5, 3 by 3 max pooling을 다 해보자!



Inception module : 1x1 conv.

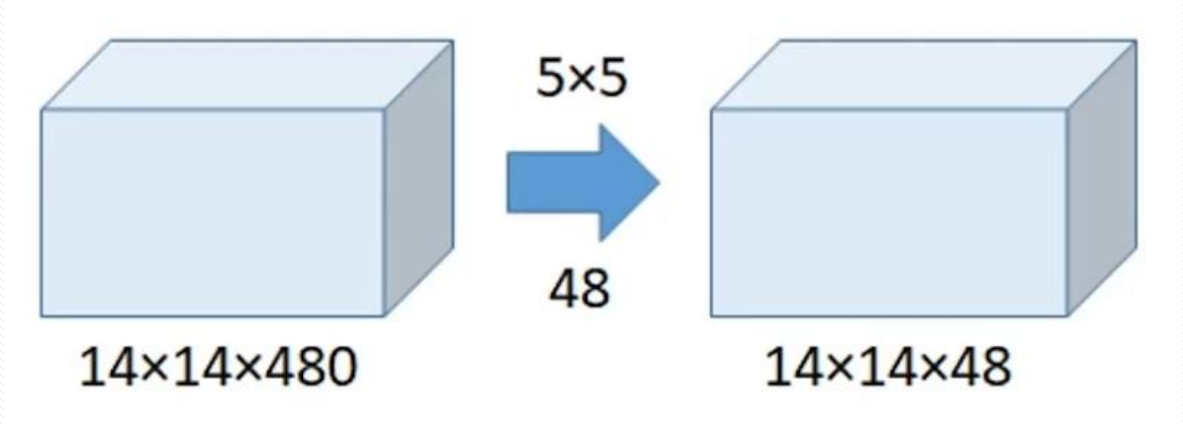
✓ 1 by 1 convolution의 역할 : feature map의 depth를 감소시켜 연산량을 감소시킴



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

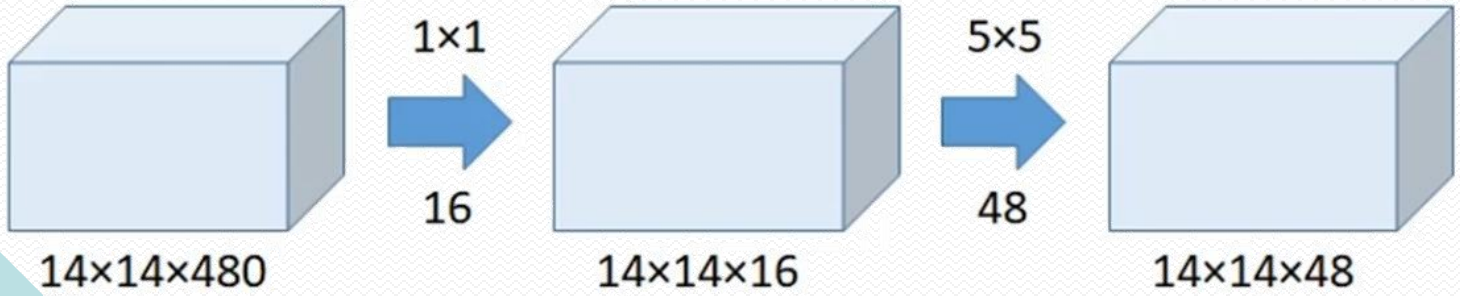
Inception module : 1x1 conv.

[1 by 1 convolution을 수행하지 않고 5 by 5 convolution을 수행]



✓ 연산량 : (14× 14 × 48) X (5× 5× 480) = 112.9M

[1 by 1 convolution을 수행후 5 by 5 convolution을 수행]



*연산량

✓ 1 by 1 Conv. : (14 × 14 × 16) X (1 × 1 × 480) = 1.5M

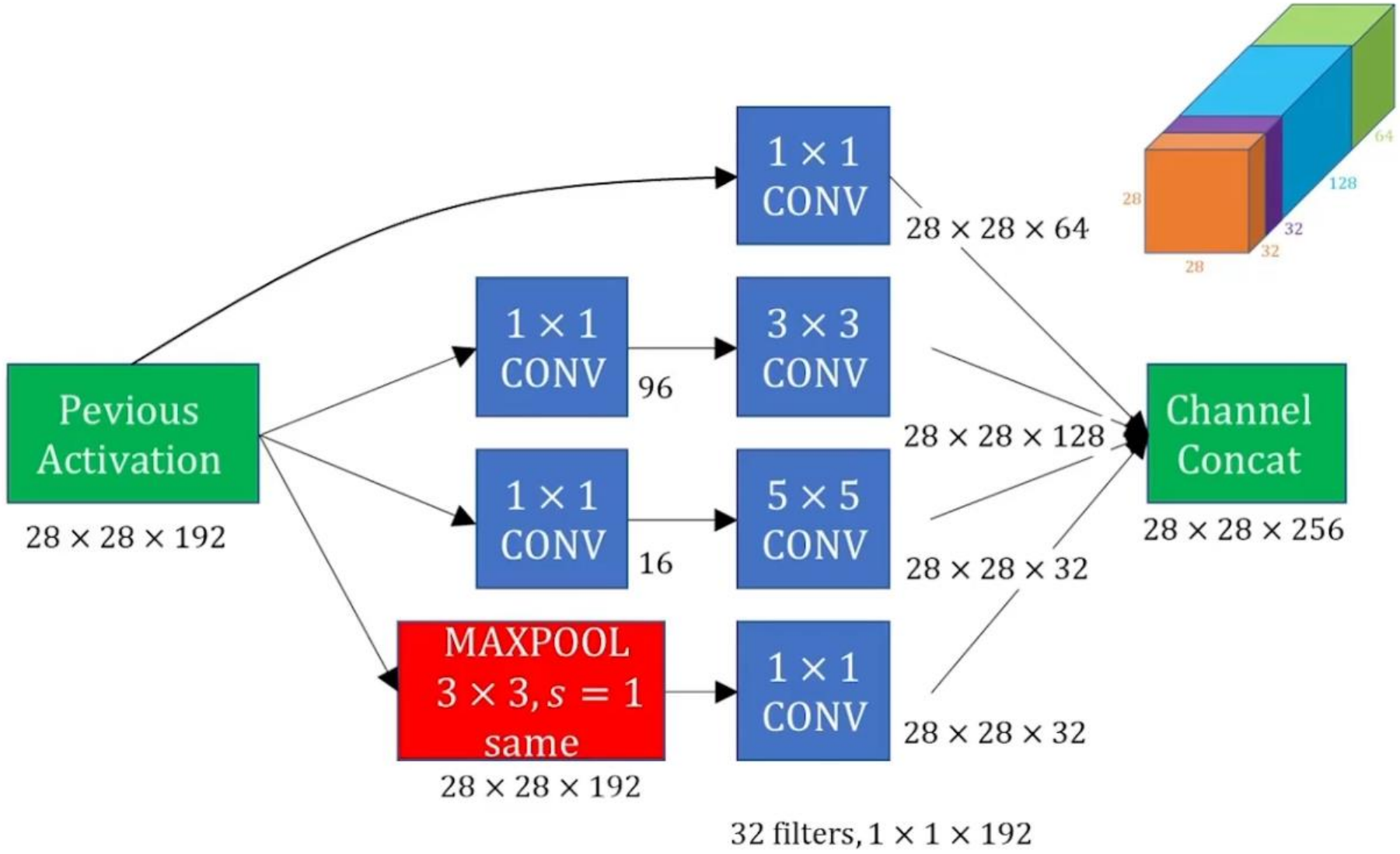
✓ 5 by 5 Conv. :(14× 14× 48) X (5 × 5 × 16) =3.8M

✓ 총 연산량 : 1.5M + 3.8M = 5.3M << 112.9M

✓ 1 by 1 convolution의 역할 : feature map의 depth를 감소시켜 연산량을 감소시킴

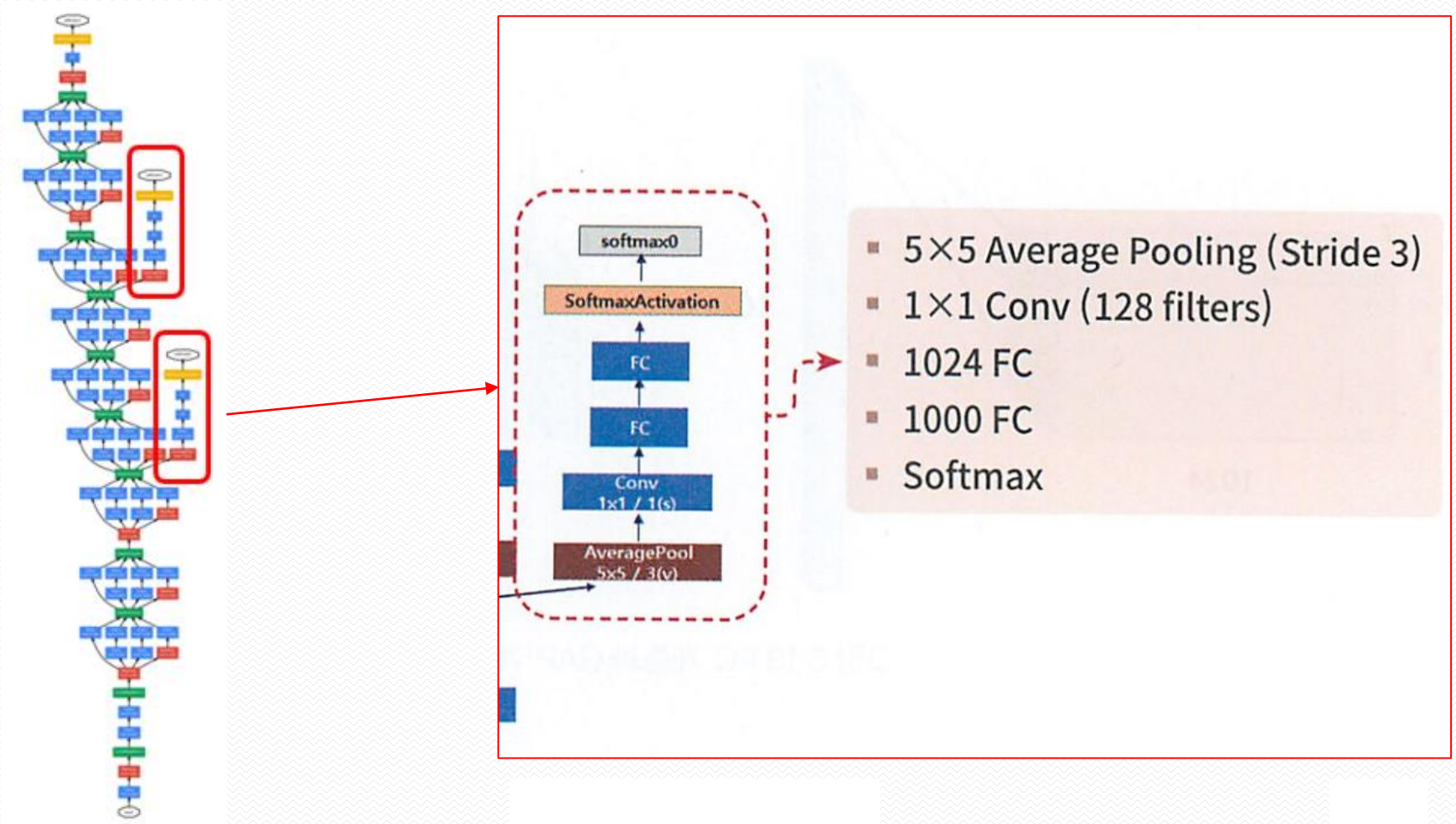
Inception module : concatnation

✓ Channel-wise concatnation



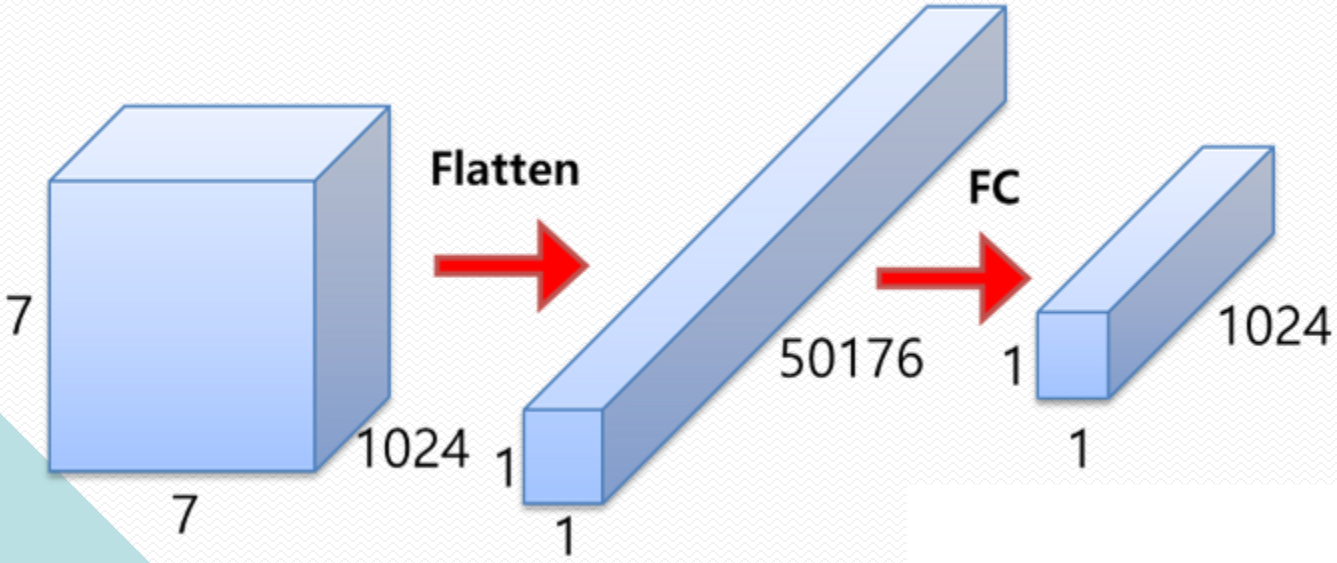
Auxiliary Classifiers

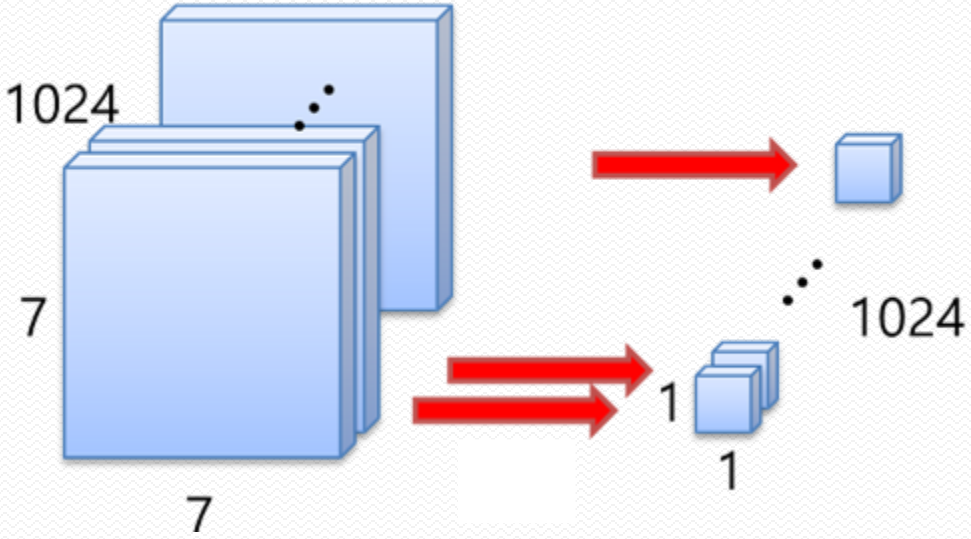
- 보조 분류기는 그래디언트 소실 문제를 해결하고 정규화를 제공



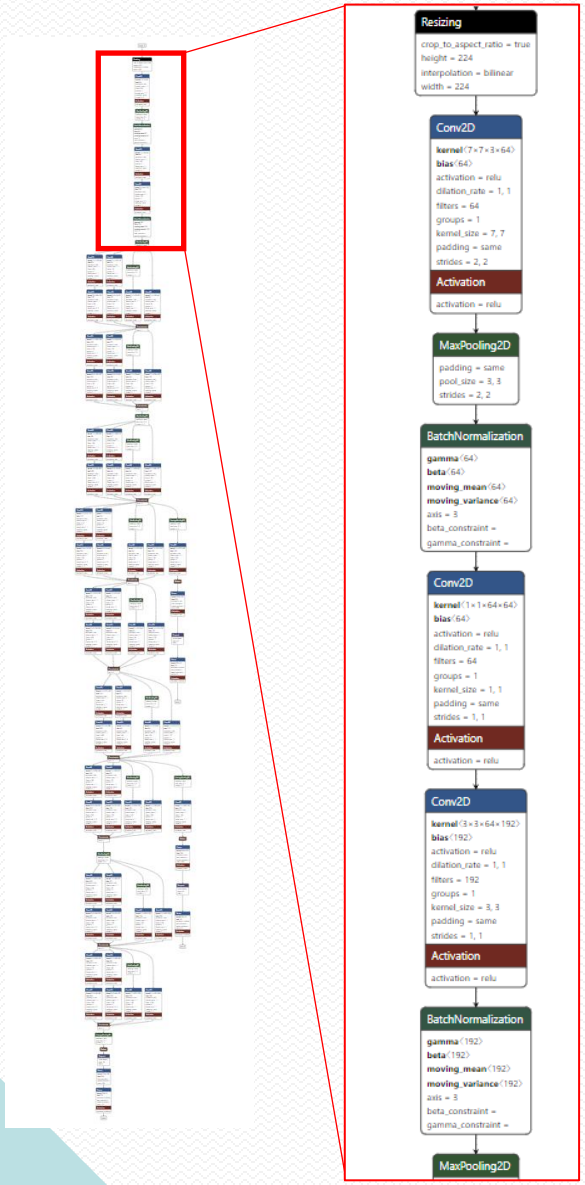
Global Average Pooling

- FC (Fully Connected)
  - Weights: 7 x 7 x 1024 x 1024 = 51.3M
- GAP (Global Average Pooling)
  - Weights: 0



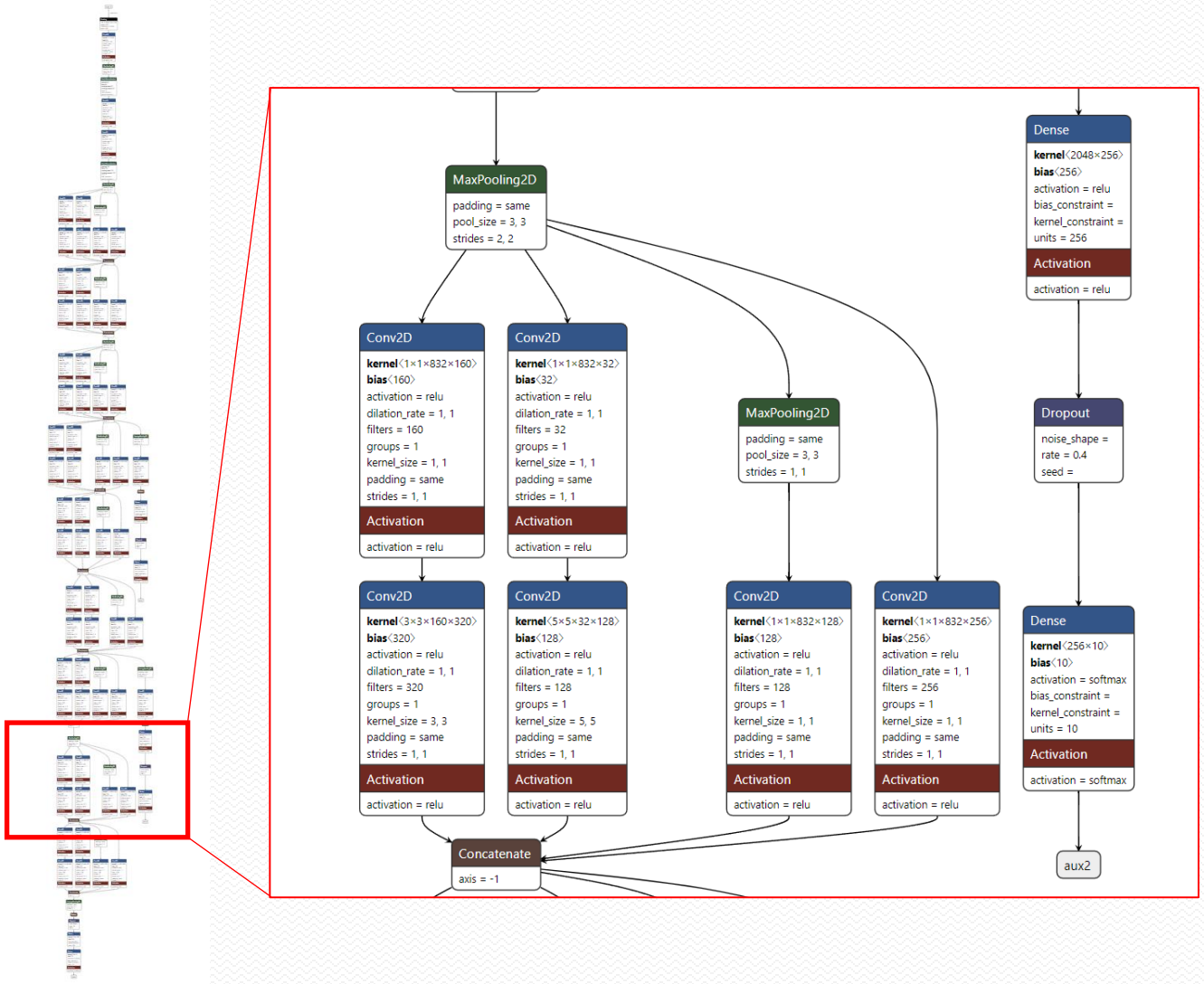


GoogLeNet Architecture:https://netron.app/



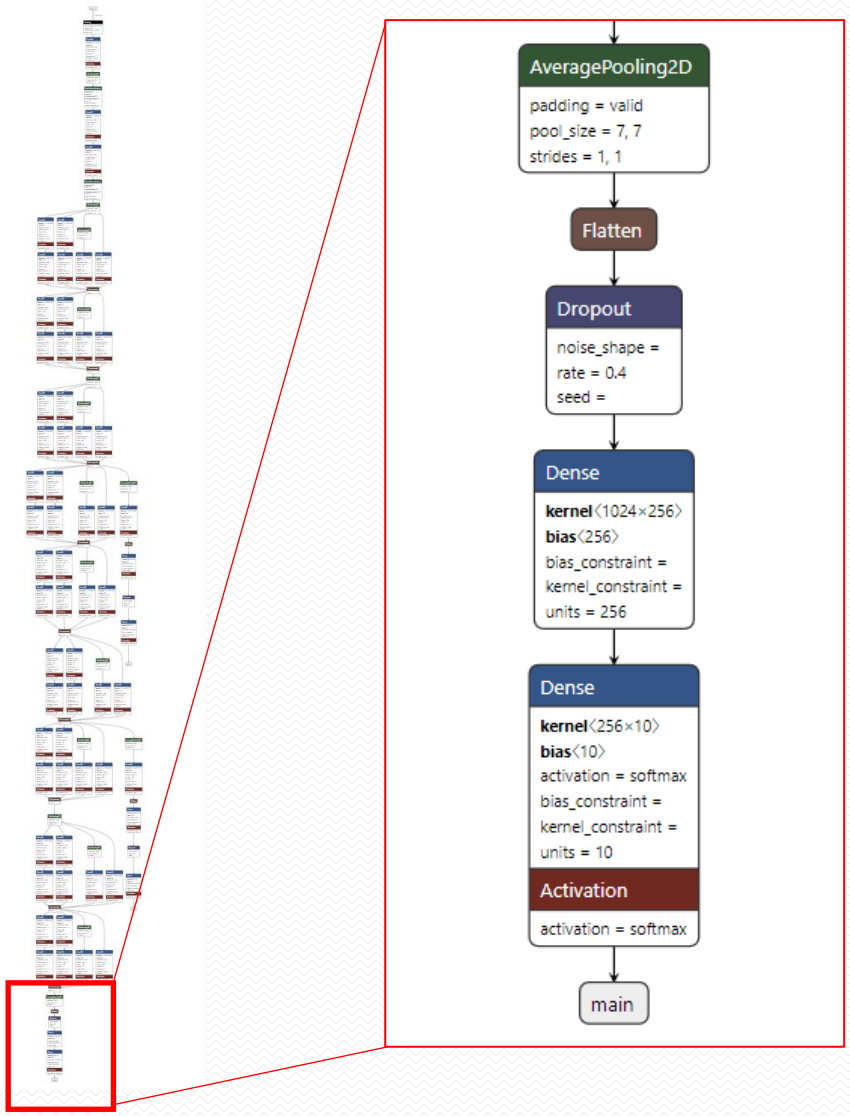

```python
# stage 1 (conv, polling, normalize, conv, conv, normalize, pooling)
model_in = Input(shape=f_image_train.shape[1:])
model = tf.keras.layers.Resizing(\
    224, 224, interpolation="bilinear", crop_to_aspect_ratio=True,\
    input_shape=f_image_train.shape[1:], name="Resizing_Layer")(model_in)


#model = tf.keras.layers.Rescaling(1./255)(model)
model = Conv2D(filters=64, kernel_size=(7,7), strides=2, padding = 'same', activation = 'relu')(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)
model = BatchNormalization()(model)

# stage 2
model = Conv2D(filters=64, kernel_size=(1,1), strides=1, padding='same', activation='relu')(model)
model = Conv2D(filters=192, kernel_size=(3,3), strides=1, padding='same', activation='relu')(model)
model = BatchNormalization()(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(model)

# stage 3
model = Inception(model, [64, (96, 128), (16, 32), 32])
model = Inception(model, [128, (128, 192), (32, 96), 64])
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)

# stage 4
model = Inception(model, [192, (96, 208), (16, 48), 64]) #a
aux1  = Auxiliary_classifier(model, name = 'aux1')
model = Inception(model, [160, (112, 224), (24, 64), 64])#b
model = Inception(model, [128, (128, 256), (24, 64), 64])#c
model = Inception(model, [112, (144, 288), (32, 64), 64])#d
aux2  = Auxiliary_classifier(model, name = 'aux2')
model = Inception(model, [256, (160, 320), (32, 128), 128])#e
model = MaxPooling2D(pool_size=(3, 3), strides = 2, padding = 'same')(model)

# stage 5
model = Inception(model, [256, (160, 320), (32, 128), 128])
model = Inception(model, [384, (192, 384), (48, 128), 128])
model = AveragePooling2D(pool_size=(7,7), strides = 1, padding = 'valid')(model)

#stage 6
model = Flatten()(model)
model = Dropout(0.4)(model)
#model = Dense(480, activation='linear')
model = Dense(units=256, activation='linear')(model)
main_branch = Dense(units=10, activation='softmax', name='main')(model)
model_fin = Model(inputs=model_in, outputs=[main_branch, aux1, aux2])

model_fin.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

model_fin.fit(f_image_train, f_label_train, epochs=100, batch_size = 5)
model_fin.save('GoogleNet_mnist.h5')
```

GoogLeNet Architecture




```python
# stage 1 (conv, polling, normalize, conv, conv, normalize, pooling)
model_in = Input(shape=f_image_train.shape[1:])
model = tf.keras.layers.Resizing(\
    224, 224, interpolation="bilinear", crop_to_aspect_ratio=True,\
    input_shape=f_image_train.shape[1:], name="Resizing_Layer")(model_in)


#model = tf.keras.layers.Rescaling(1./255)(model)
model = Conv2D(filters=64, kernel_size=(7,7), strides=2, padding = 'same', activation = 'relu')(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)
model = BatchNormalization()(model)

# stage 2
model = Conv2D(filters=64, kernel_size=(1,1), strides=1, padding='same', activation='relu')(model)
model = Conv2D(filters=192, kernel_size=(3,3), strides=1, padding='same', activation='relu')(model)
model = BatchNormalization()(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(model)

# stage 3
model = Inception(model, [64, (96, 128), (16, 32), 32])
model = Inception(model, [128, (128, 192), (32, 96), 64])
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)

# stage 4
model = Inception(model, [192, (96, 208), (16, 48), 64]) #a
aux1  = Auxiliary_classifier(model, name = 'aux1')
model = Inception(model, [160, (112, 224), (24, 64), 64])#b
model = Inception(model, [128, (128, 256), (24, 64), 64])#c
model = Inception(model, [112, (144, 288), (32, 64), 64])#d
aux2  = Auxiliary_classifier(model, name = 'aux2')
model = Inception(model, [256, (160, 320), (32, 128), 128])#e
model = MaxPooling2D(pool_size=(3, 3), strides = 2, padding = 'same')(model)

# stage 5
model = Inception(model, [256, (160, 320), (32, 128), 128])
model = Inception(model, [384, (192, 384), (48, 128), 128])
model = AveragePooling2D(pool_size=(7,7), strides = 1, padding = 'valid')(model)

#stage 6
model = Flatten()(model)
model = Dropout(0.4)(model)
#model = Dense(480, activation='linear')
model = Dense(units=256, activation='linear')(model)
main_branch = Dense(units=10, activation='softmax', name='main')(model)
model_fin = Model(inputs=model_in, outputs=[main_branch, aux1, aux2])

model_fin.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

model_fin.fit(f_image_train, f_label_train, epochs=100, batch_size = 5)
model_fin.save('GoogleNet_mnist.h5')
```

GoogLeNet Architecture




```python
# stage 1 (conv, polling, normalize, conv, conv, normalize, pooling)
model_in = Input(shape=f_image_train.shape[1:])
model = tf.keras.layers.Resizing(\
    224, 224, interpolation="bilinear", crop_to_aspect_ratio=True,\
    input_shape=f_image_train.shape[1:], name="Resizing_Layer")(model_in)


#model = tf.keras.layers.Rescaling(1./255)(model)
model = Conv2D(filters=64, kernel_size=(7,7), strides=2, padding = 'same', activation = 'relu')(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)
model = BatchNormalization()(model)

# stage 2
model = Conv2D(filters=64, kernel_size=(1,1), strides=1, padding='same', activation='relu')(model)
model = Conv2D(filters=192, kernel_size=(3,3), strides=1, padding='same', activation='relu')(model)
model = BatchNormalization()(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(model)

# stage 3
model = Inception(model, [64, (96, 128), (16, 32), 32])
model = Inception(model, [128, (128, 192), (32, 96), 64])
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)

# stage 4
model = Inception(model, [192, (96, 208), (16, 48), 64]) #a
aux1  = Auxiliary_classifier(model, name = 'aux1')
model = Inception(model, [160, (112, 224), (24, 64), 64])#b
model = Inception(model, [128, (128, 256), (24, 64), 64])#c
model = Inception(model, [112, (144, 288), (32, 64), 64])#d
aux2  = Auxiliary_classifier(model, name = 'aux2')
model = Inception(model, [256, (160, 320), (32, 128), 128])#e
model = MaxPooling2D(pool_size=(3, 3), strides = 2, padding = 'same')(model)

# stage 5
model = Inception(model, [256, (160, 320), (32, 128), 128])
model = Inception(model, [384, (192, 384), (48, 128), 128])
model = AveragePooling2D(pool_size=(7,7), strides = 1, padding = 'valid')(model)

#stage 6
model = Flatten()(model)
model = Dropout(0.4)(model)
#model = Dense(480, activation='linear')
model = Dense(units=256, activation='linear')(model)
main_branch = Dense(units=10, activation='softmax', name='main')(model)
model_fin = Model(inputs=model_in, outputs=[main_branch, aux1, aux2])

model_fin.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

model_fin.fit(f_image_train, f_label_train, epochs=100, batch_size = 5)
model_fin.save('GoogleNet_mnist.h5')
```

GoogLeNet Architecture

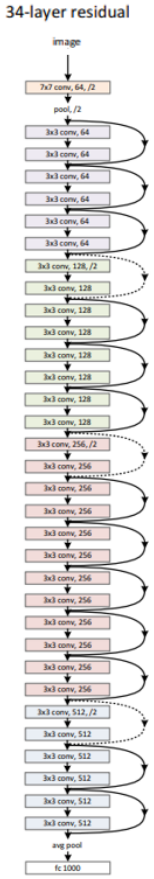
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	p
convolution	7×7/2	112×112×64	1							
max pool	3×3/2	56×56×64	0							
convolution	3×3/1	56×56×192	2		64	192				
max pool	3×3/2	28×28×192	0							
inception (3a)		28×28×256	2	64	96	128	16	32	32	
inception (3b)		28×28×480	2	128	128	192	32	96	64	
max pool	3×3/2	14×14×480	0							
inception (4a)		14×14×512	2	192	96	208	16	48	64	
inception (4b)		14×14×512	2	160	112	224	24	64	64	
inception (4c)		14×14×512	2	128	128	256	24	64	64	
inception (4d)		14×14×528	2	112	144	288	32	64	64	
inception (4e)		14×14×832	2	256	160	320	32	128	128	
max pool	3×3/2	7×7×832	0							
inception (5a)		7×7×832	2	256	160	320	32	128	128	
inception (5b)		7×7×1024	2	384	192	384	48	128	128	
avg pool	7×7/1	1×1×1024	0							
dropout (40%)		1×1×1024	0							
linear		1×1×1000	1							
softmax		1×1×1000	0							

```
# stage 1 (conv, polling, normalize, conv, conv, normalize, pooling)
model_in = Input(shape=f_image_train.shape[1:])
model = tf.keras.layers.Resizing(\
    224, 224, interpolation="bilinear", crop_to_aspect_ratio=True,\
    input_shape=f_image_train.shape[1:], name="Resizing_Layer")(model_in)


#model = tf.keras.layers.Rescaling(1./255)(model)
model = Conv2D(filters=64, kernel_size=(7,7), strides=2, padding = 'same', activation = 'relu')(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)
model = BatchNormalization()(model)

# stage 2
model = Conv2D(filters=64, kernel_size=(1,1), strides=1, padding='same', activation='relu')(model)
model = Conv2D(filters=192, kernel_size=(3,3), strides=1, padding='same', activation='relu')(model)
model = BatchNormalization()(model)
model = MaxPooling2D(pool_size=(3,3), strides=2, padding='same')(model)

# stage 3
model = Inception(model, [64, (96, 128), (16, 32), 32])
model = Inception(model, [128, (128, 192), (32, 96), 64])
model = MaxPooling2D(pool_size=(3,3), strides=2, padding = 'same')(model)

# stage 4
model = Inception(model, [192, (96, 208), (16, 48), 64]) #a
aux1  = Auxiliary_classifier(model, name = 'aux1')
model = Inception(model, [160, (112, 224), (24, 64), 64])#b
model = Inception(model, [128, (128, 256), (24, 64), 64])#c
model = Inception(model, [112, (144, 288), (32, 64), 64])#d
aux2  = Auxiliary_classifier(model, name = 'aux2')
model = Inception(model, [256, (160, 320), (32, 128), 128])#e
model = MaxPooling2D(pool_size=(3, 3), strides = 2, padding = 'same')(model)

# stage 5
model = Inception(model, [256, (160, 320), (32, 128), 128])
model = Inception(model, [384, (192, 384), (48, 128), 128])
model = AveragePooling2D(pool_size=(7,7), strides = 1, padding = 'valid')(model)

#stage 6
model = Flatten()(model)
model = Dropout(0.4)(model)
#model = Dense(480, activation='linear')
model = Dense(units=256, activation='linear')(model)
main_branch = Dense(units=10, activation='softmax', name='main')(model)
model_fin = Model(inputs=model_in, outputs=[main_branch, aux1, aux2])

model_fin.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

model_fin.fit(f_image_train, f_label_train, epochs=100, batch_size = 5)
model_fin.save('GoogleNet_mnist.h5')
```

ResNet, 2015

- Revolution of depth
- 가장 깊고 성능이 좋은 네트워크

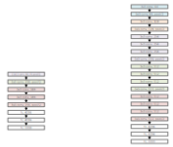
Residual Network
Great Depth
Better recognition than human



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet, 152 layers
(ILSVRC 2015)

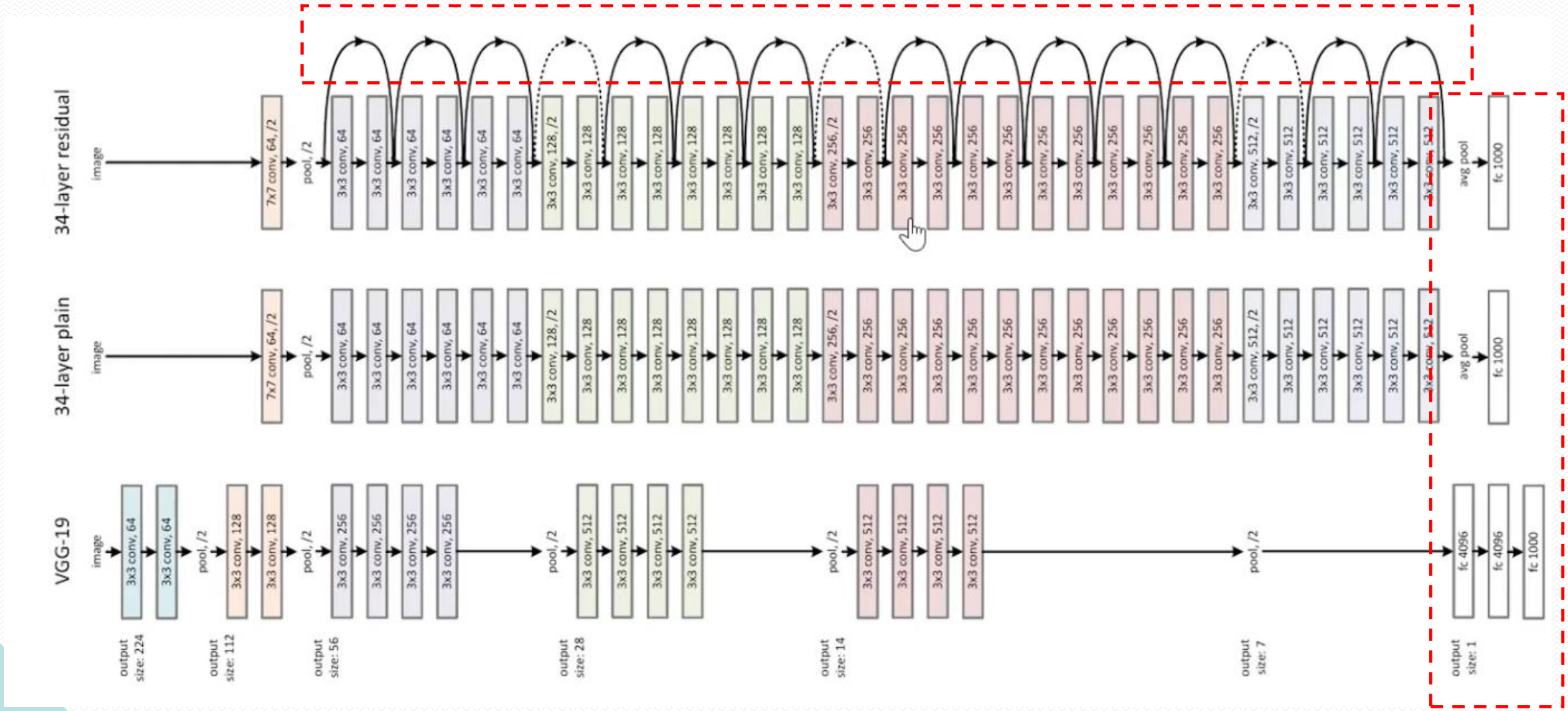
VGG, 19 layers
(ILSVRC 2014)



AlexNet, 8 layers
(ILSVRC 2012)

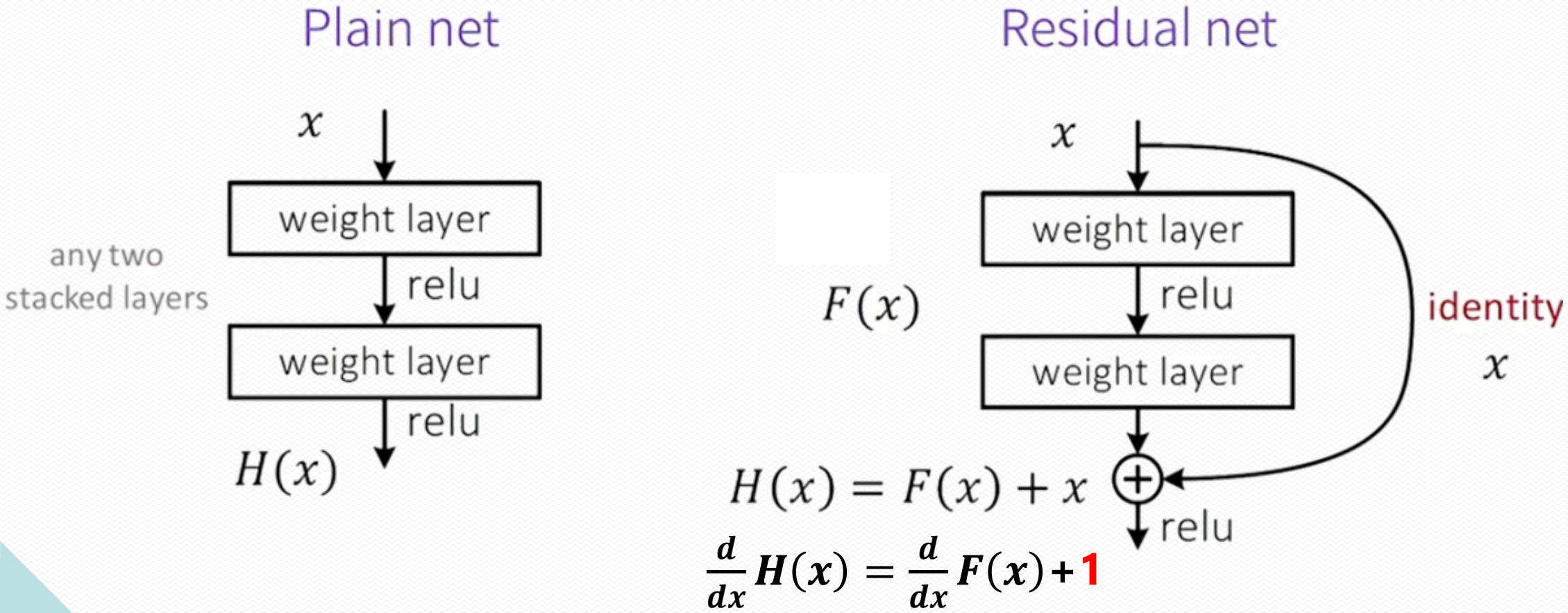
ResNet, 2015

- 핵심은 residual learning

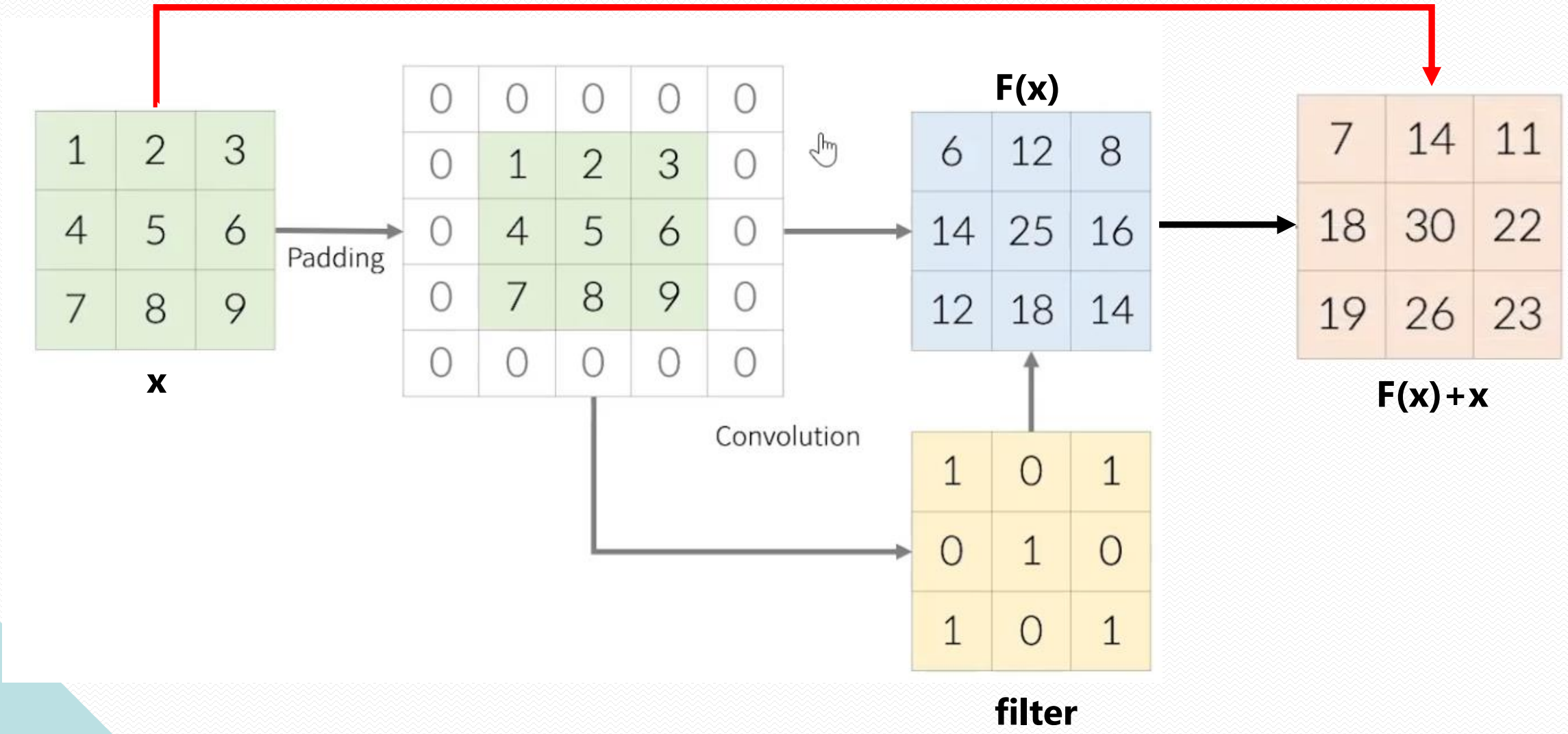


ResNet, 2015 : residual block (skip connection)

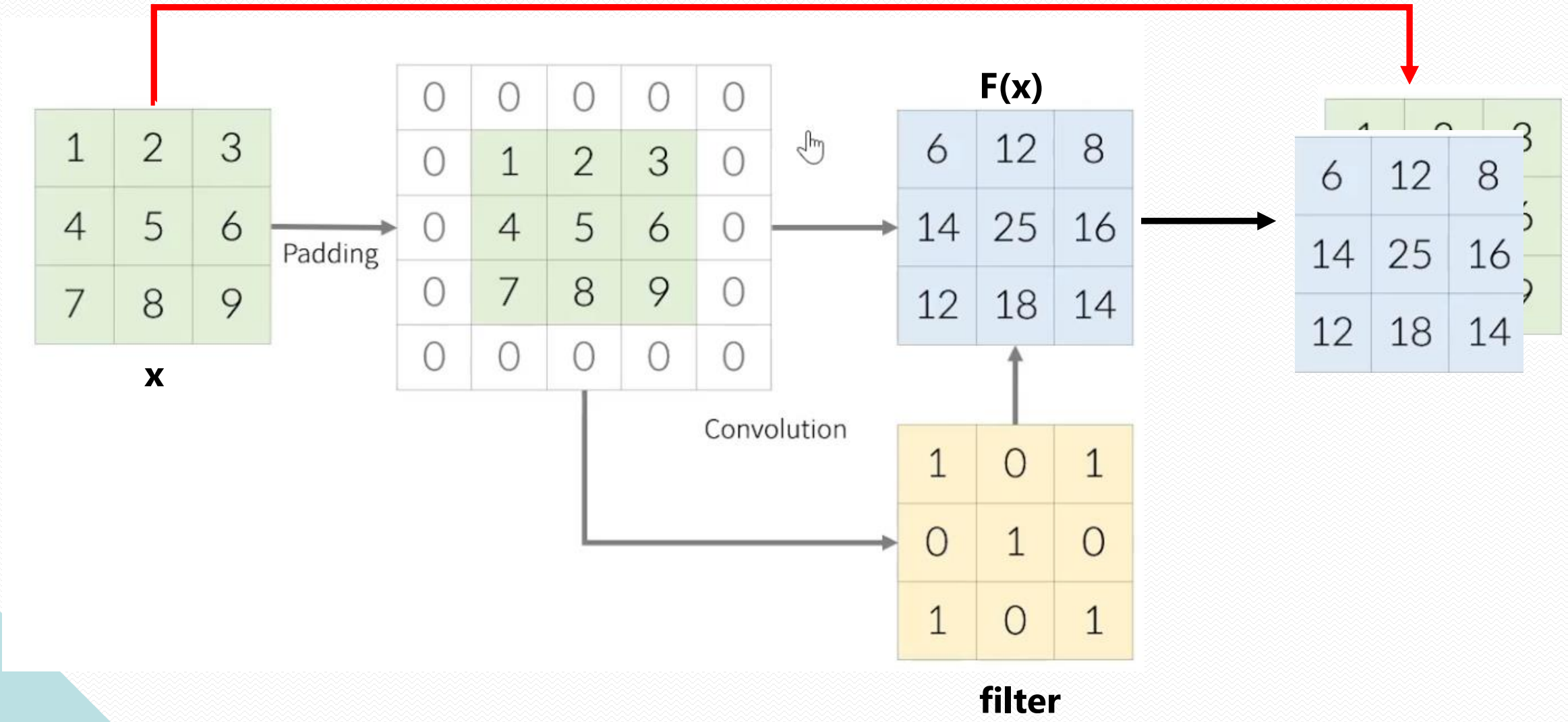
- ✓ 일정 시점마다 input x 자체를 skip connection을 통해 연결
- ✓ Gradient flow가 원활하게 이루어짐 (모델을 깊게 쌓는 것에 대한 부담이 줄어듦)
- ✓ Fully connected layer 제거 (모델의 파라미터 수를 줄임)



ResNet, 2015 : skip connection



DenseNet : channel-wise concatnation



Network Design

- Simple design
  - All 3x3 conv (almost)
  - No Hidden FC
- Spatial size /2 → # filters x2

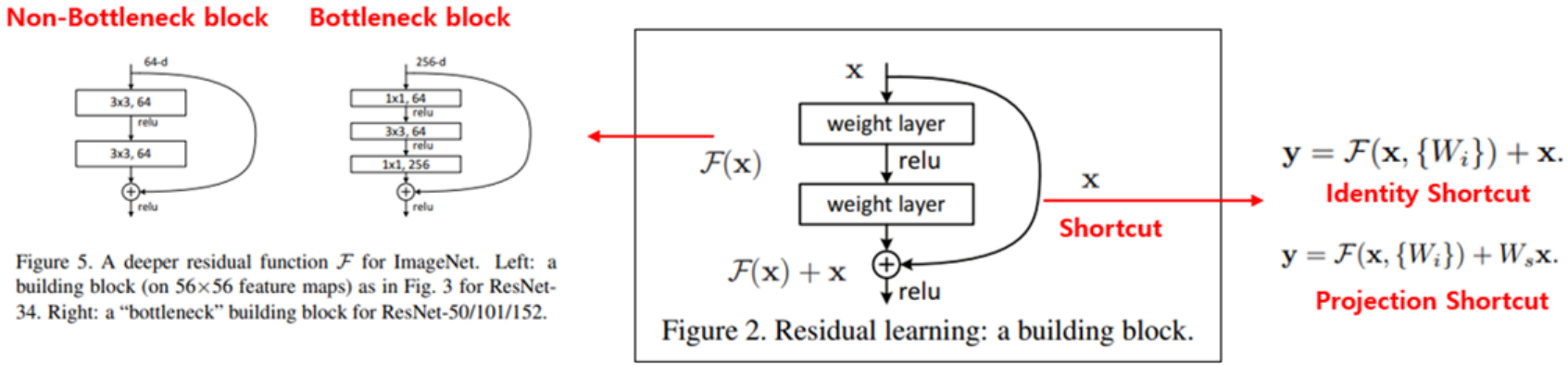


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

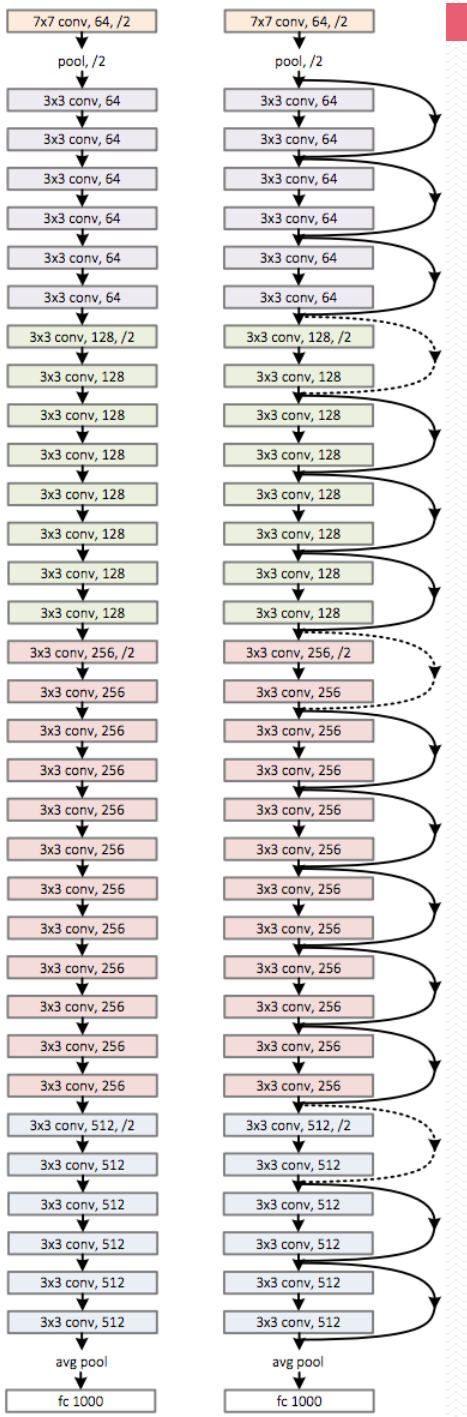
Figure 2. Residual learning: a building block.

$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$

Identity Shortcut

$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$

Projection Shortcut

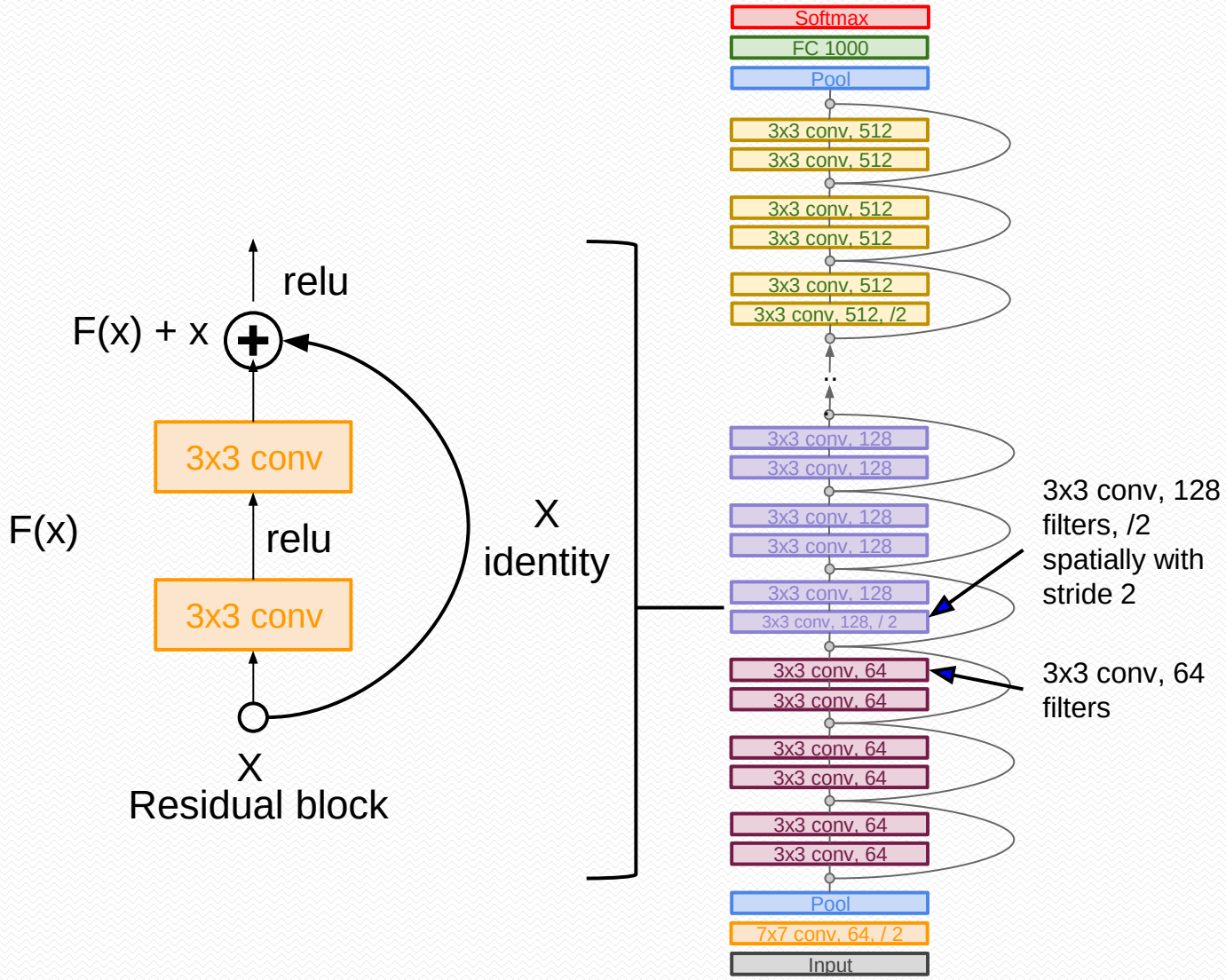


Identity shortcut

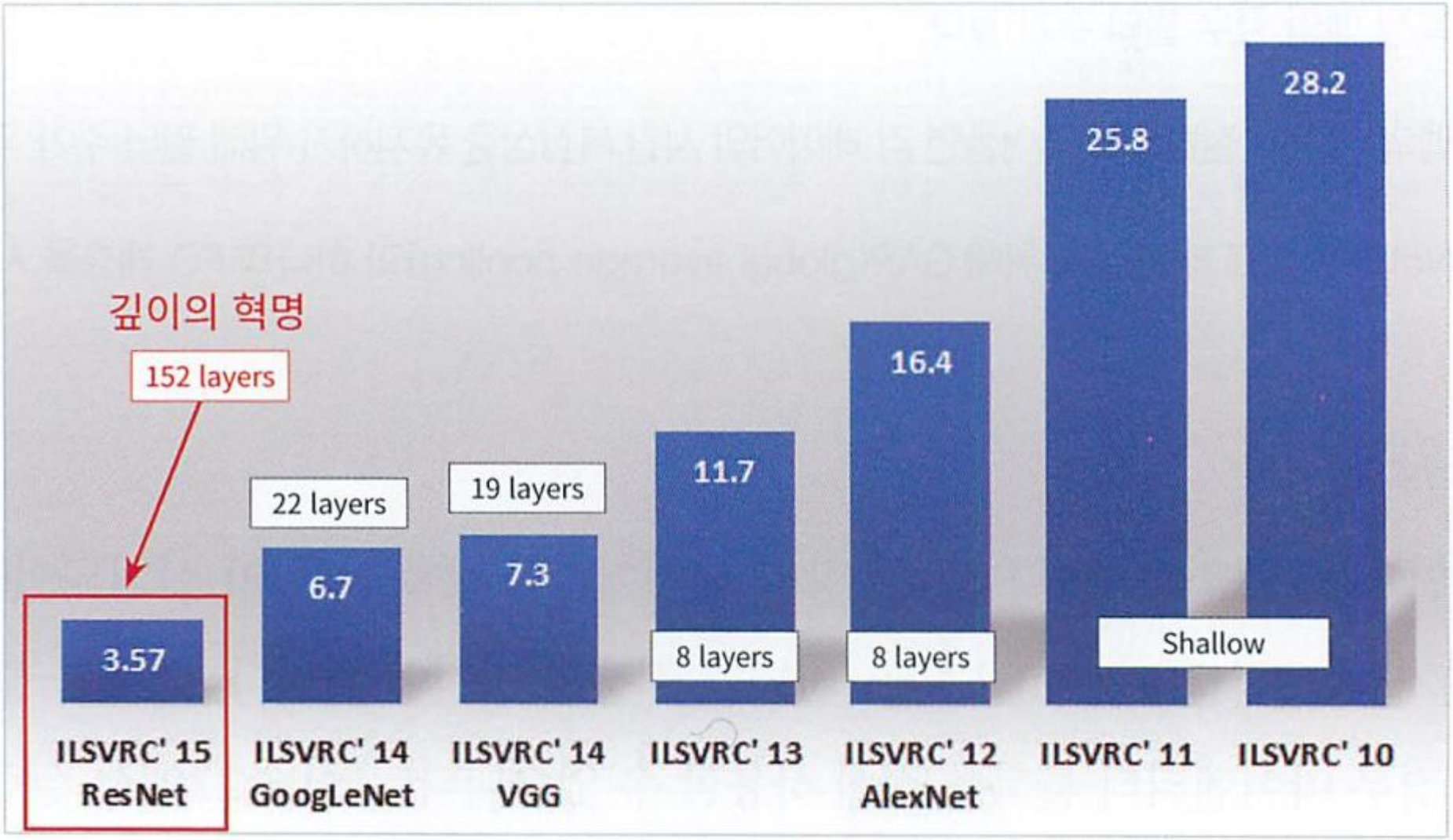
Projection shortcut

Full ResNet architecture

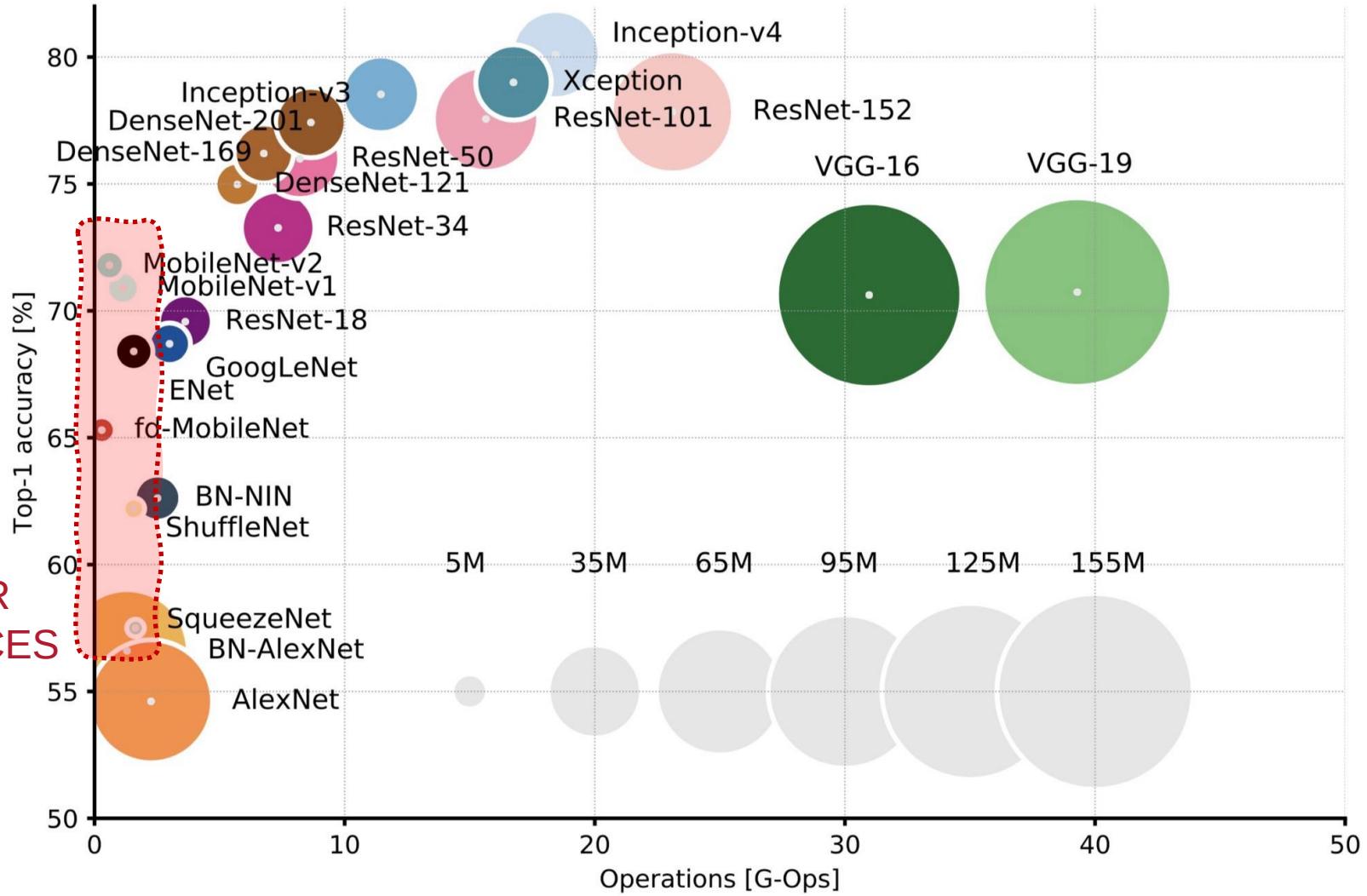
- Stack residual blocks
  - Consist of two 3x3 conv layers
- Periodically,
  - Double # of filters
  - Downsample spatially
    By using stride of 2
  - (2 in each dimension)



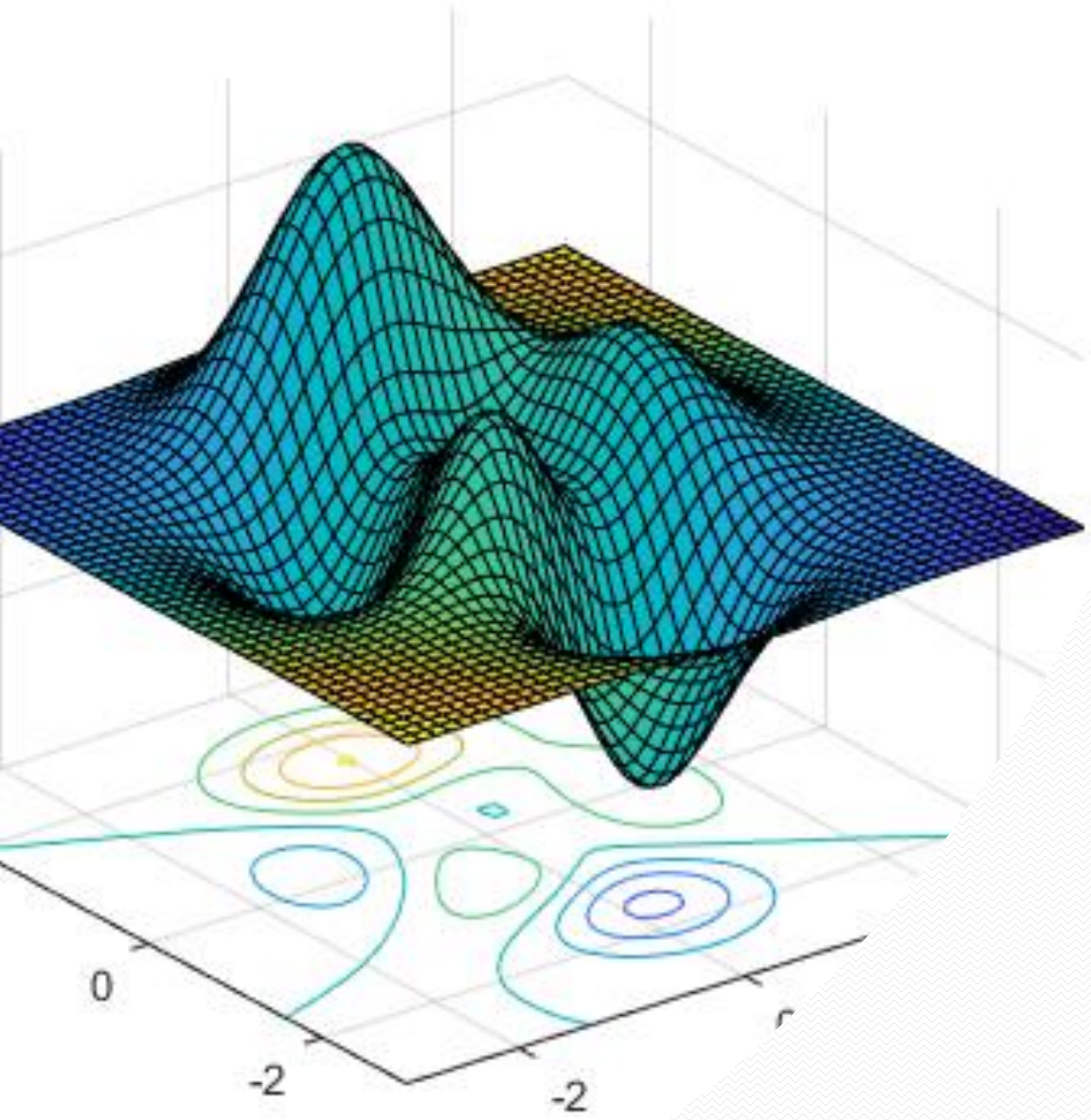
ResNet, 2015


깊이의 혁명
152 layers
3.57
ILSVRC' 15
ResNet
22 layers
6.7
ILSVRC' 14
GoogLeNet
19 layers
7.3
ILSVRC' 14
VGG
11.7
8 layers
ILSVRC' 13
16.4
8 layers
ILSVRC' 12
AlexNet
25.8
Shallow
ILSVRC' 11
28.2
ILSVRC' 10

TOP 1 VS. OPERATIONS, PARAMETERS



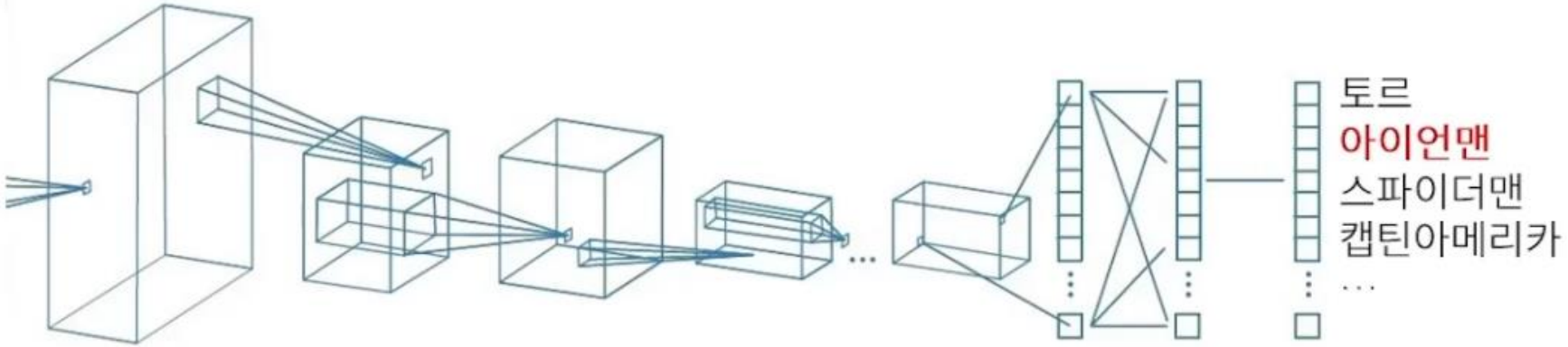
SUITED FOR MOBILE DEVICES

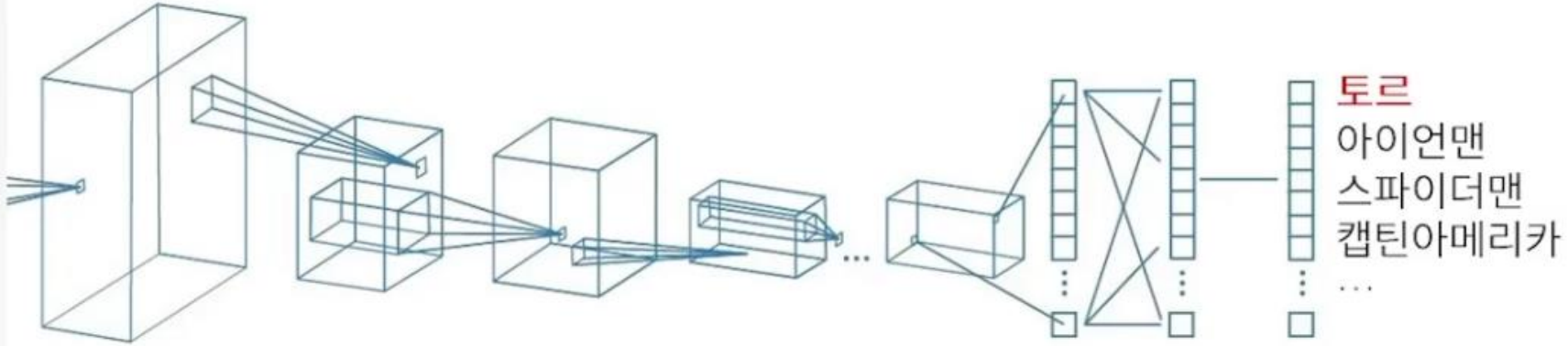


Class Localization

Class Activation Map

CNN Class Activation Map

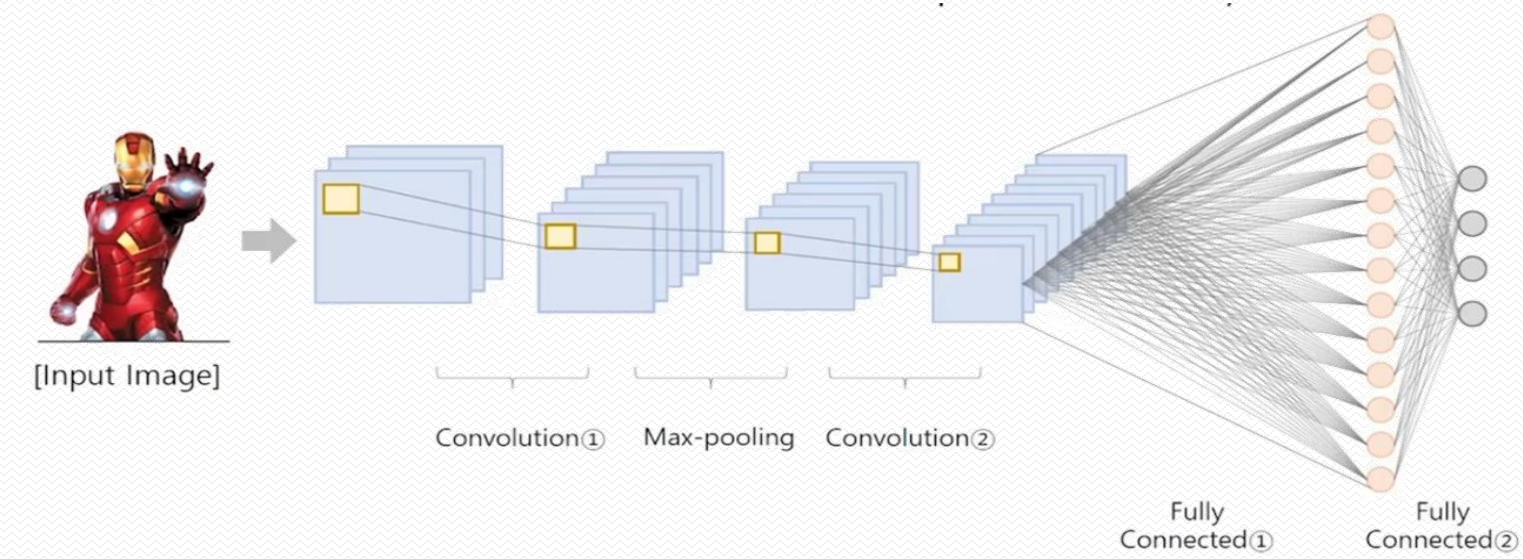


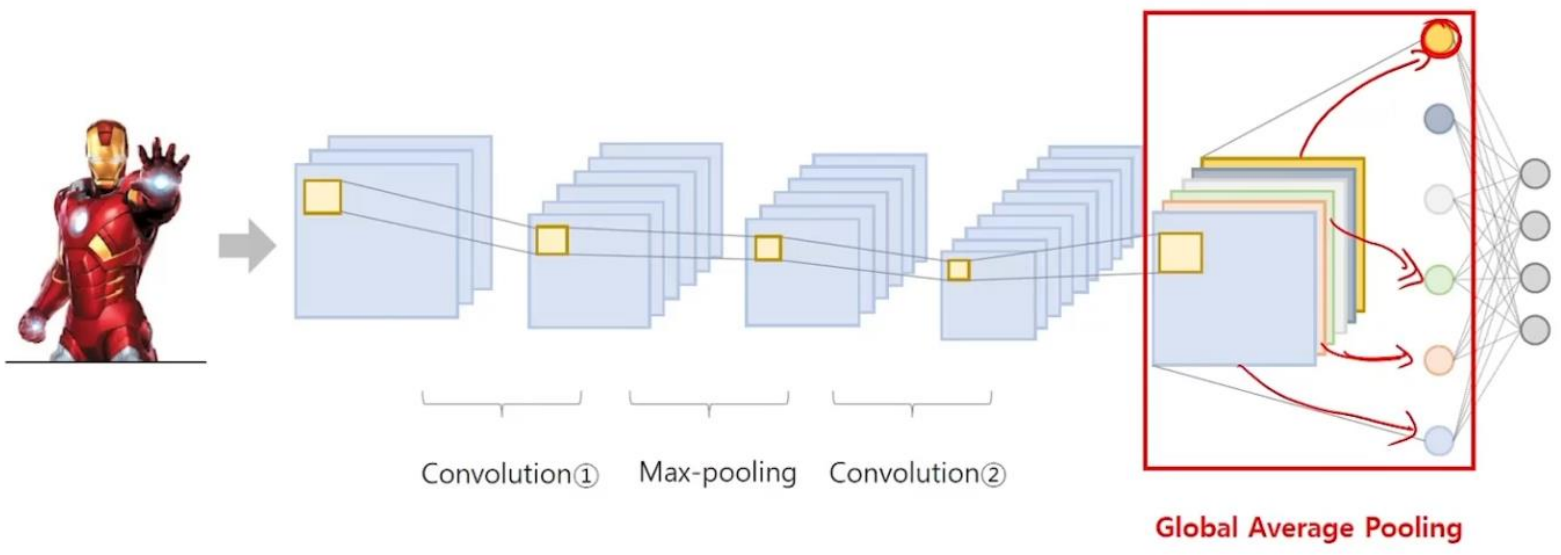


어딜 봐서
토르인거야?

CNN Class Activation Map

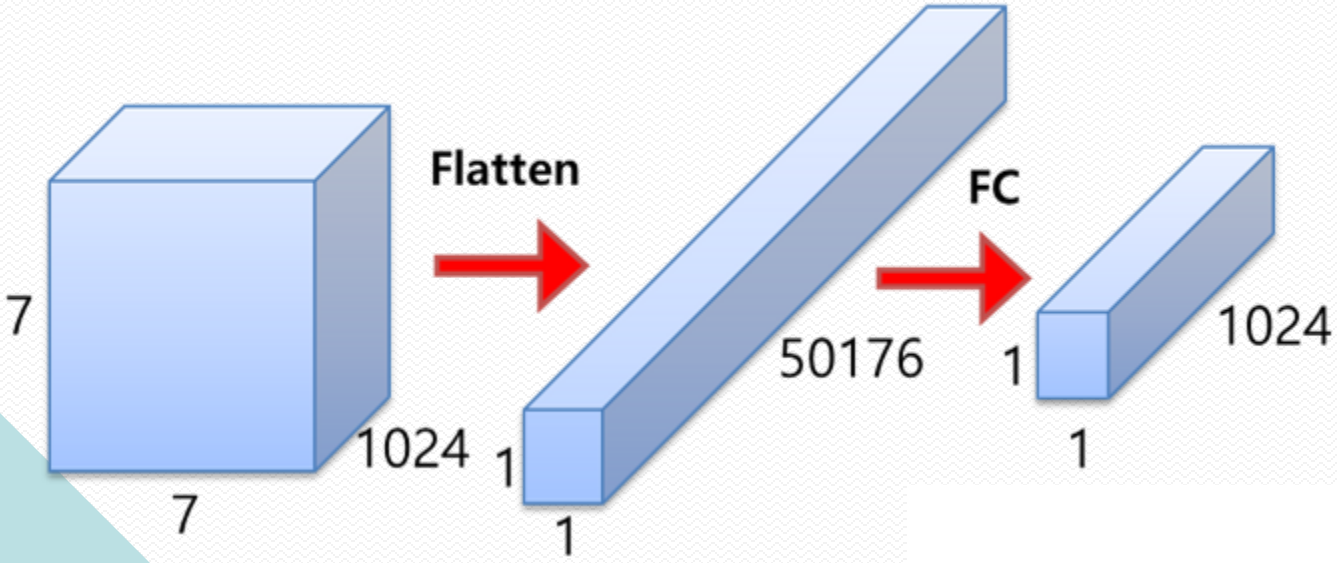
✓ Fully Connected Layer 대신 Global Average Pooling (GAP) 이라는 개념을 도입

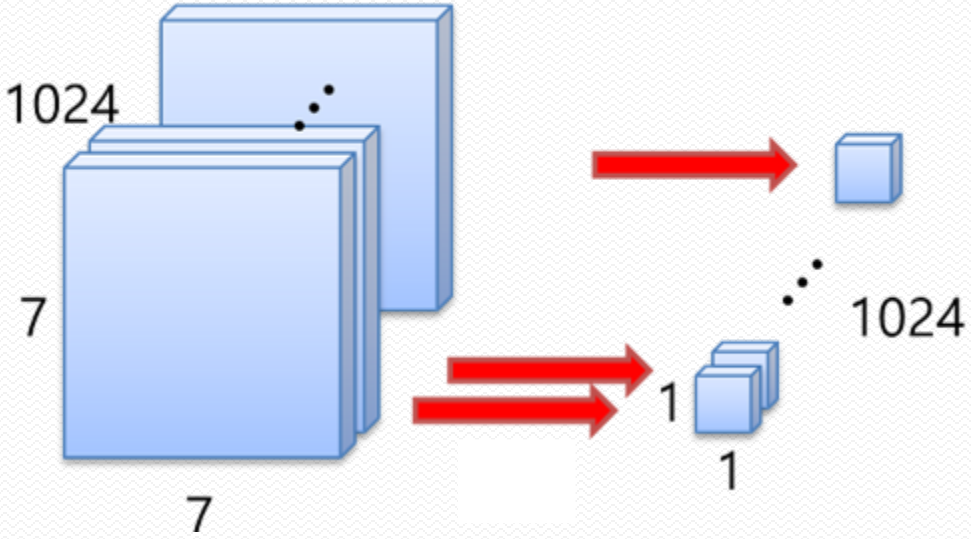




Global Average Pooling

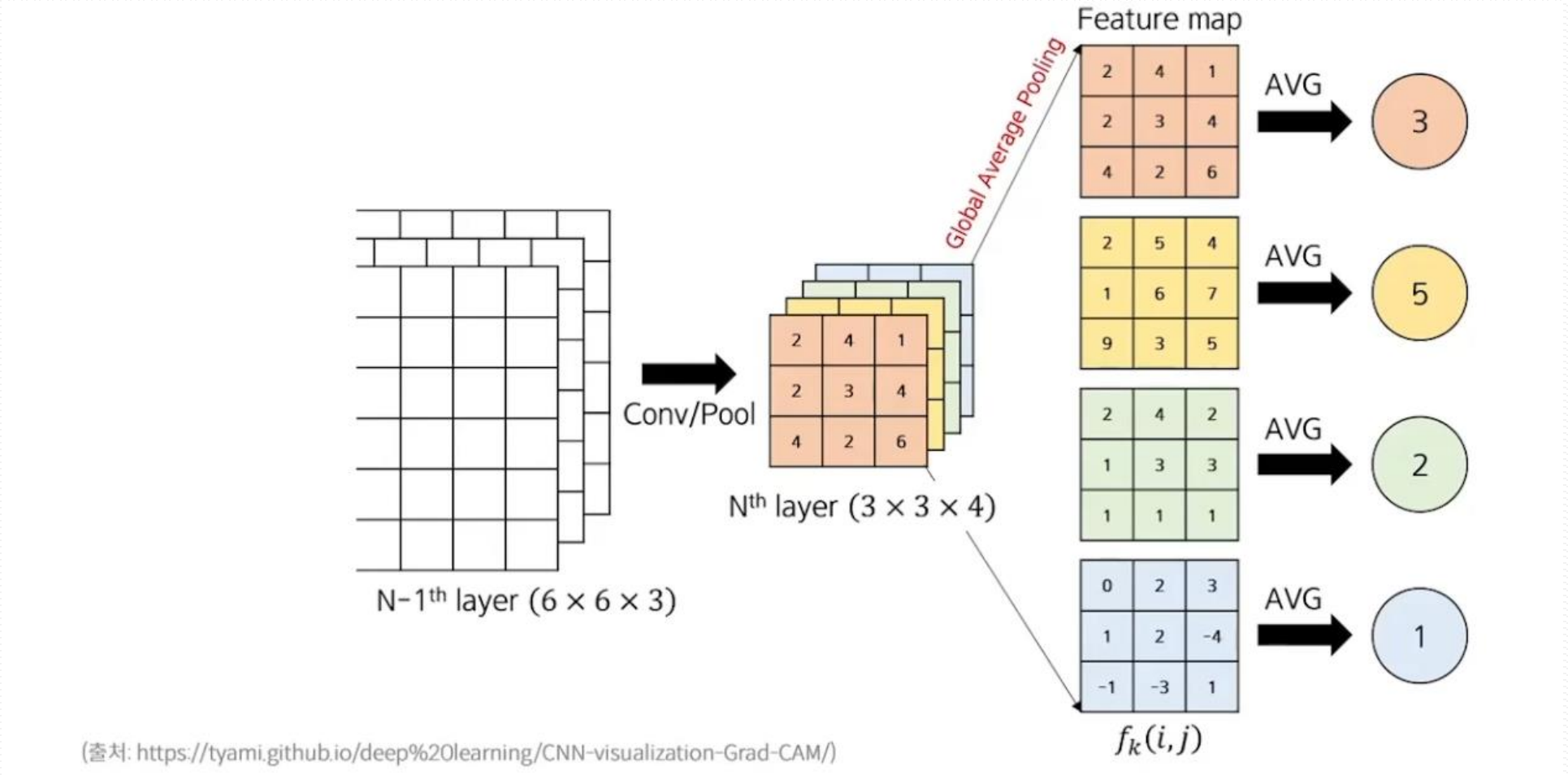
- FC (Fully Connected)
  - Weights: 7 x 7 x 1024 x 1024 = 51.3M
- GAP (Global Average Pooling)
  - Weights: 0





CNN Class Activation Map

✓ Global Average Pooling : 각 Feature Map (채널)의 가중치 값들의 평균

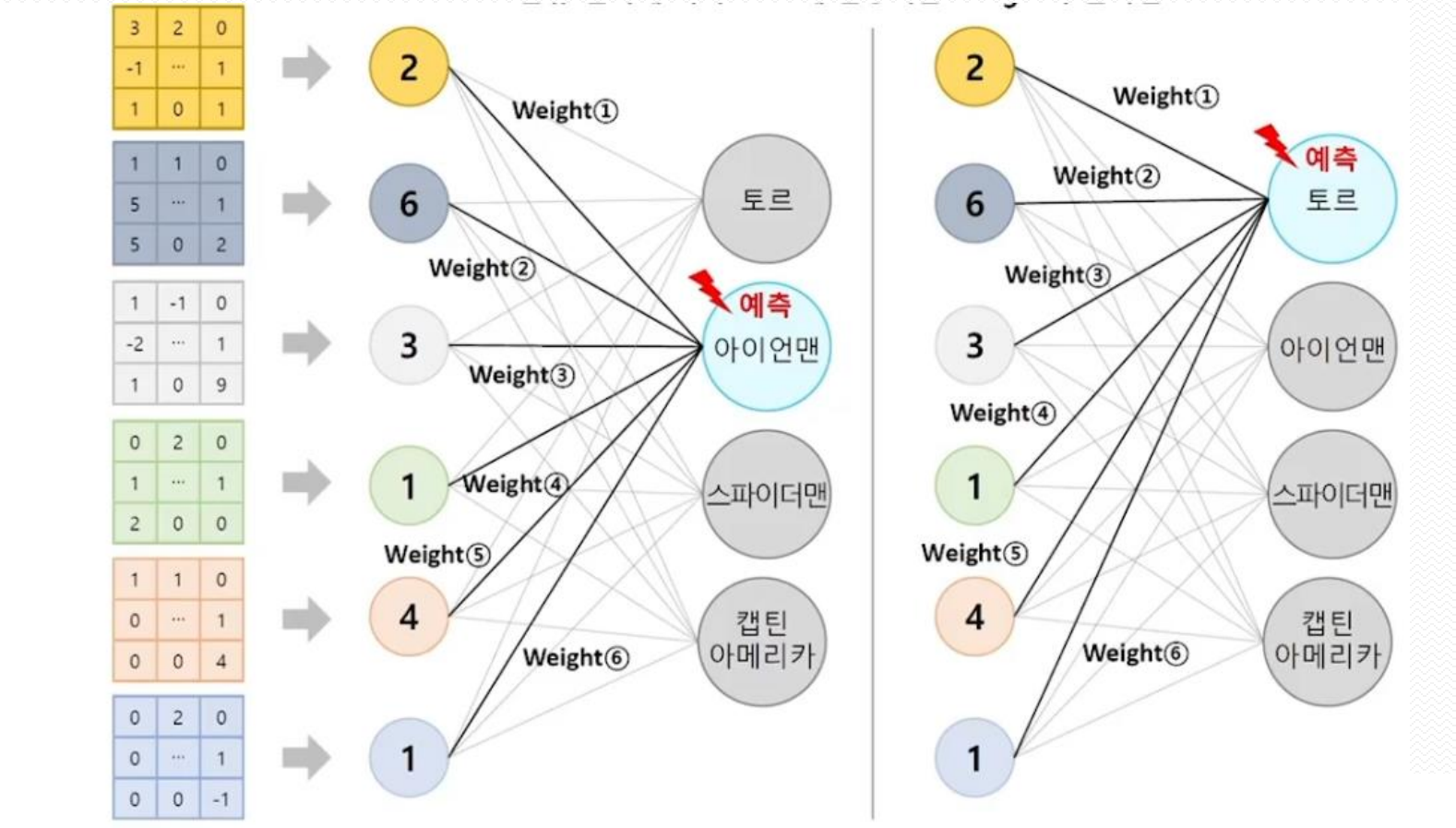


(출처: https://tyami.github.io/deep%20learning/CNN-visualization-Grad-CAM/)

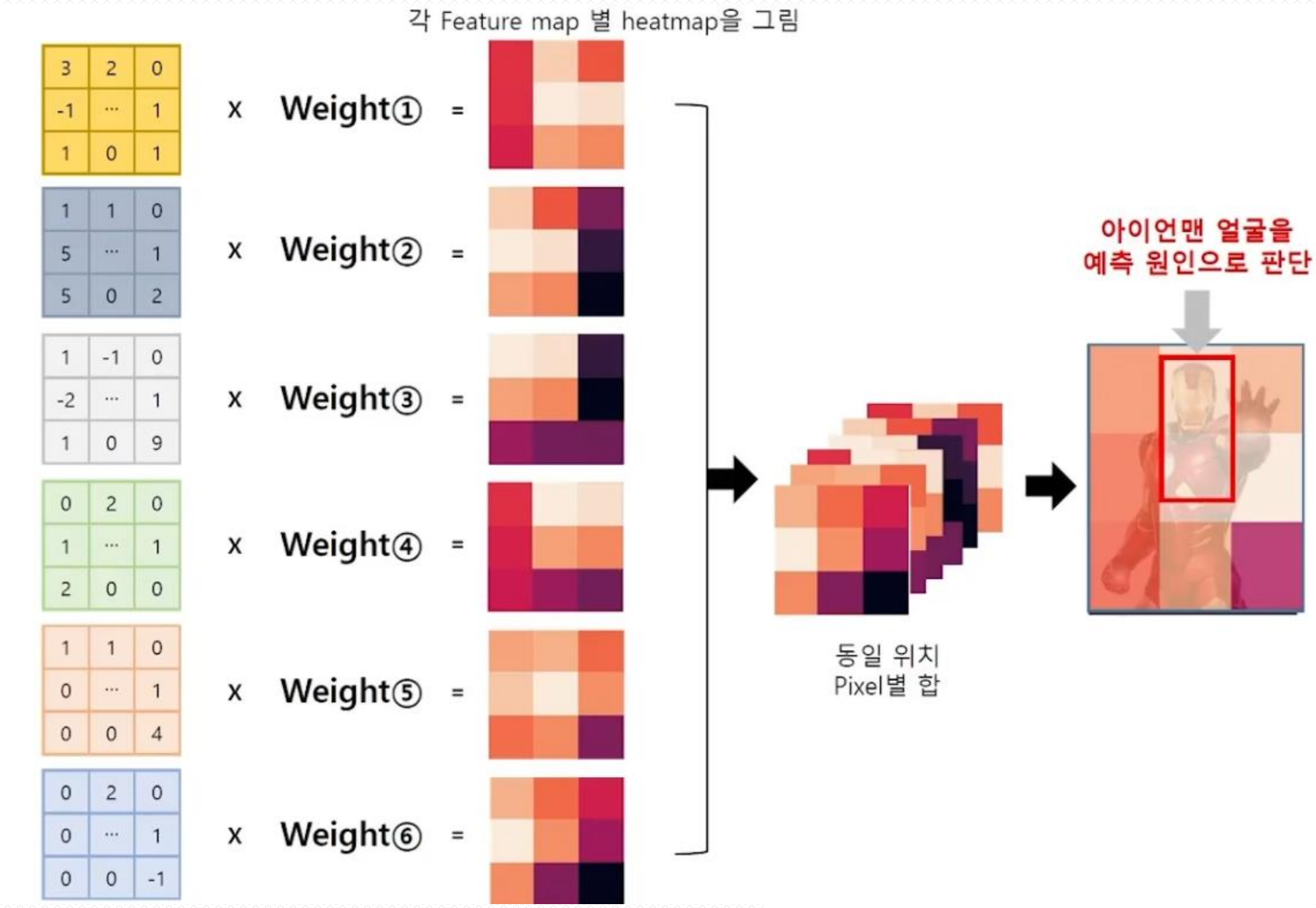
CNN Class Activation Map

✓ 각 가중치의 값이 클수록 해당 Feature map의 기여도가 크다.

✓ 분류 결과에 따라서 CAM에 활용되는 가중치가 달라짐



CNN Class Activation Map



CNN Class Activation Map

✓ 마지막 Convolution Feature Map에 가장 많은 정보가 축약되어 있음

✓ 마지막 Feature Map 내의 1개 값은 원본 이미지에서 많은 부분이 요약된 결과물임

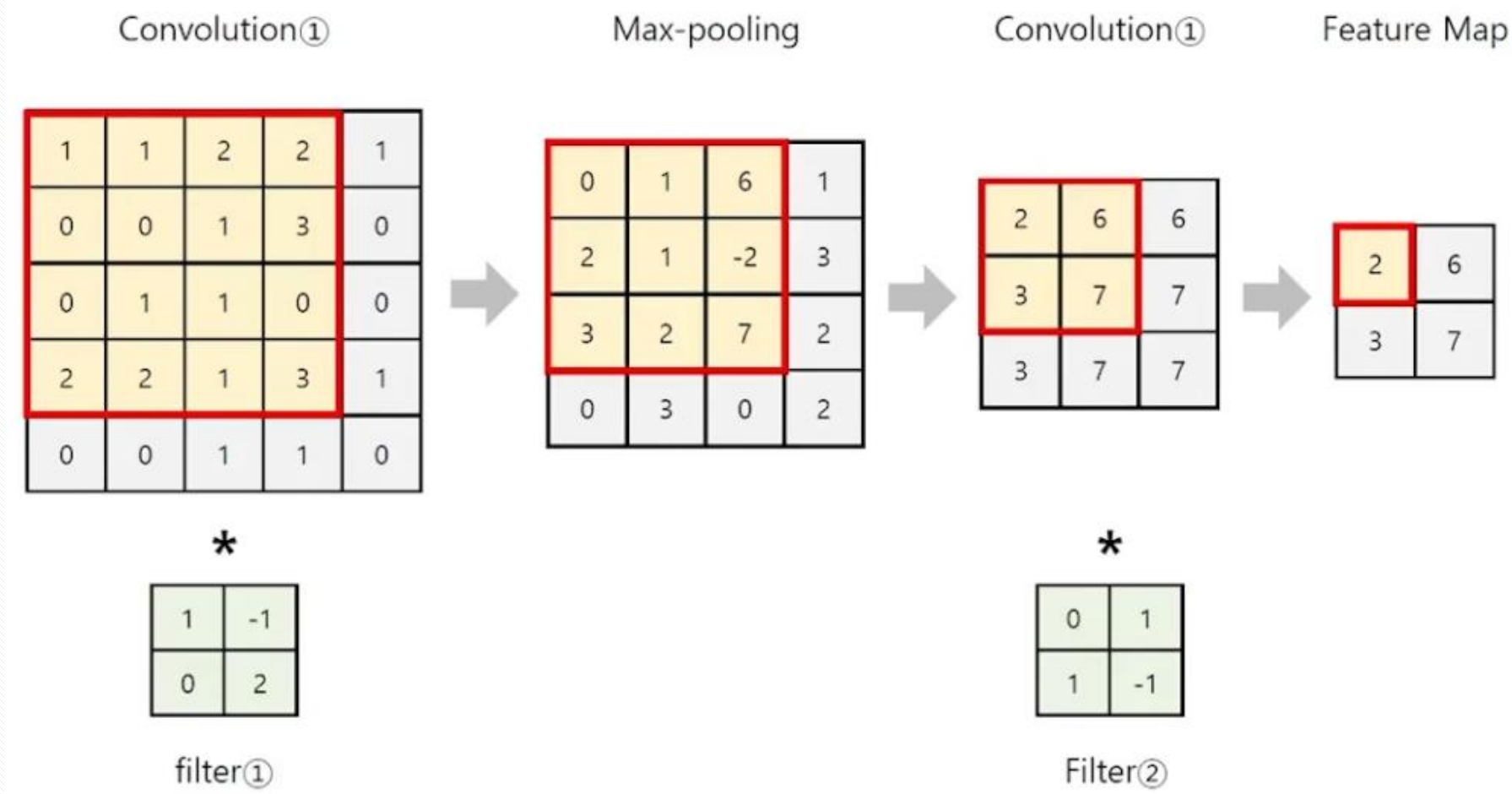


Image Localization: CNN Class Activation Map

✓ 예측된 범주 정보를 바탕으로 상대적으로 강하게 활성화가 되는 입력 이미지의 영역을 추정하는 것

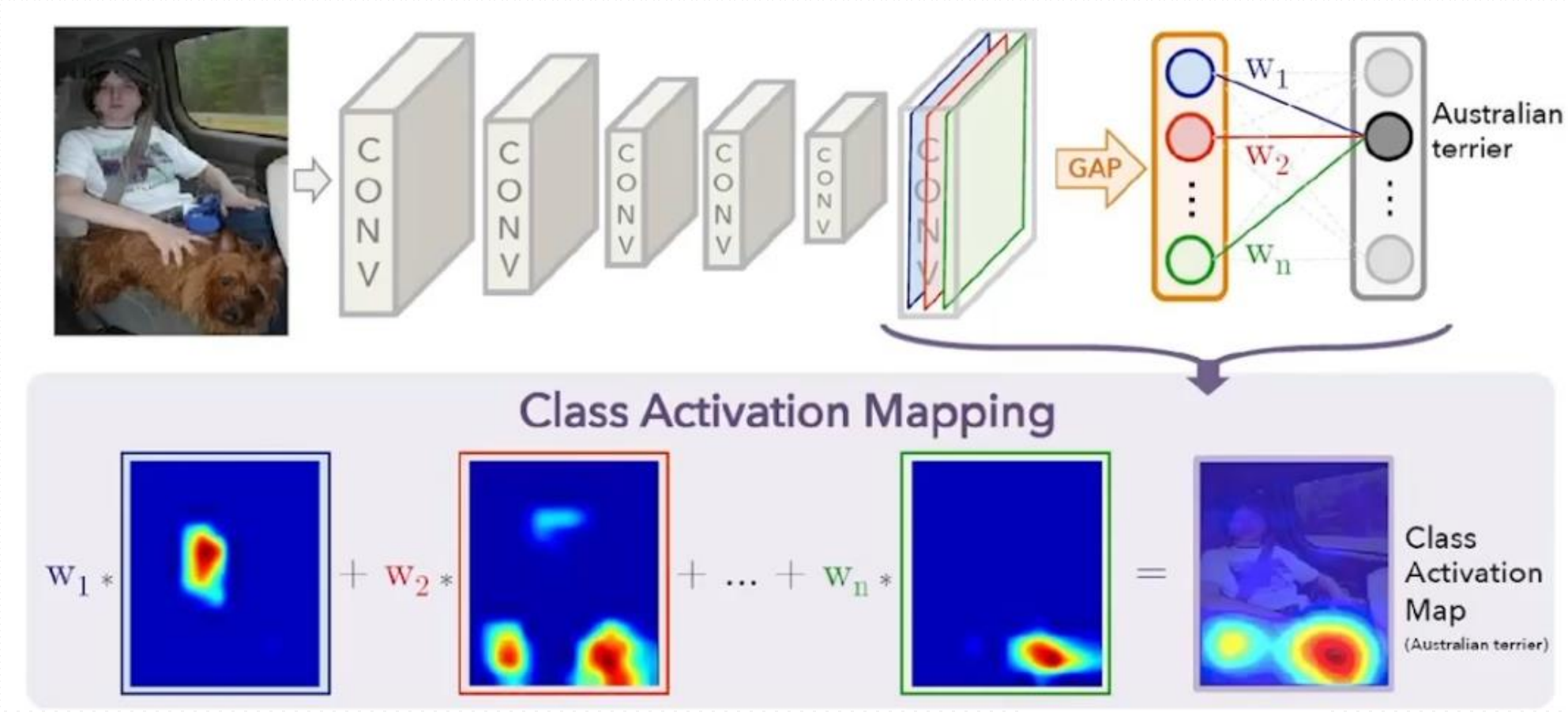
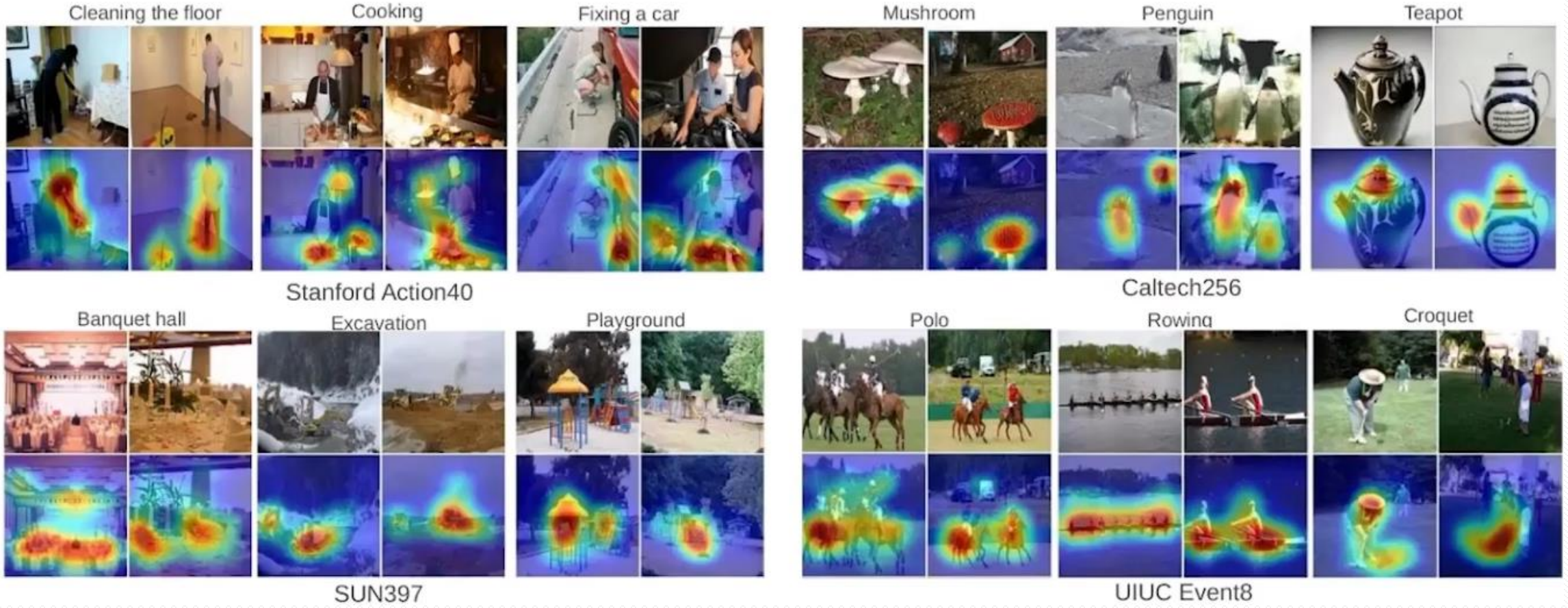
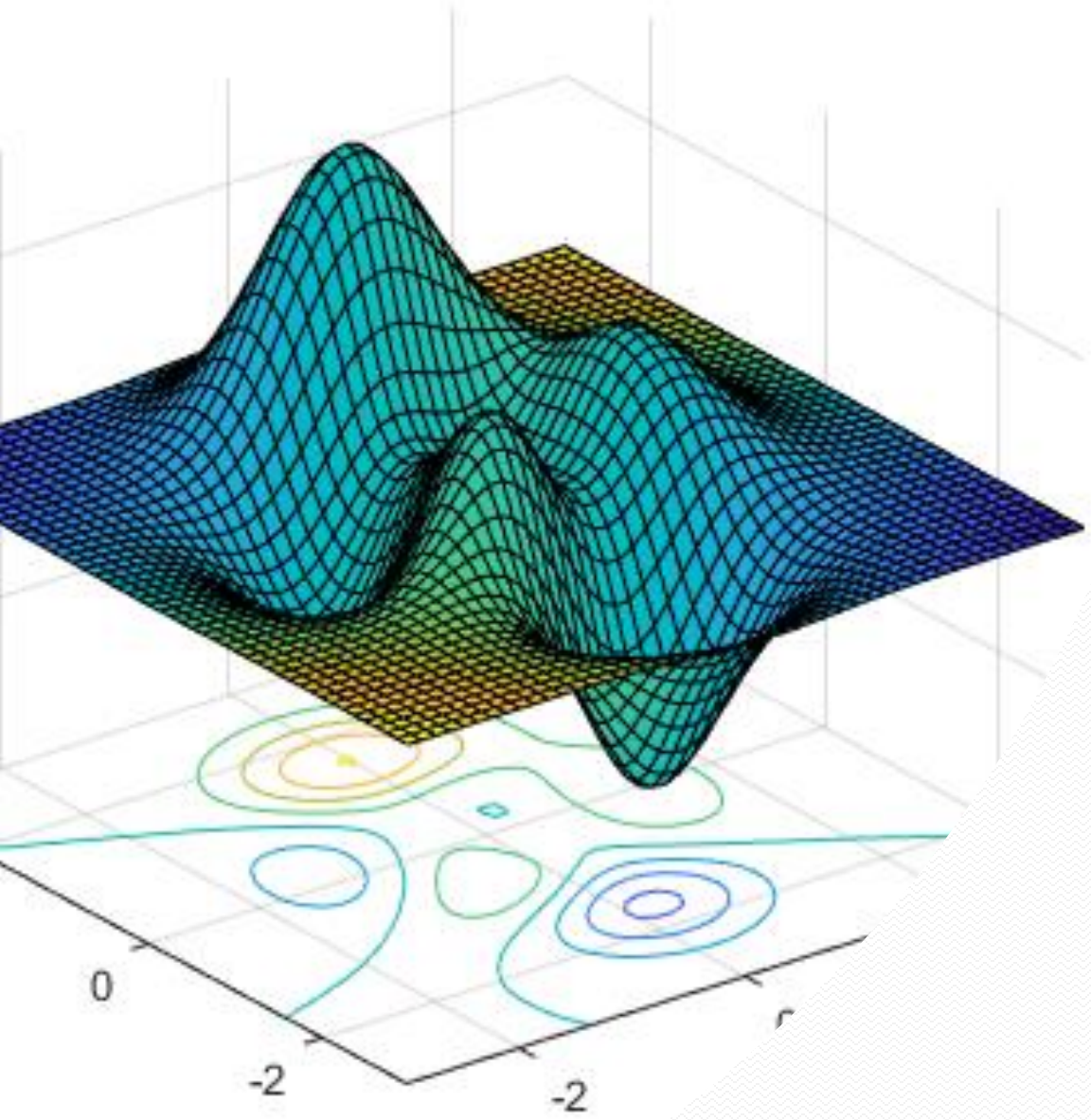


Image Localization: CNN Class Activation Map





Segmentation

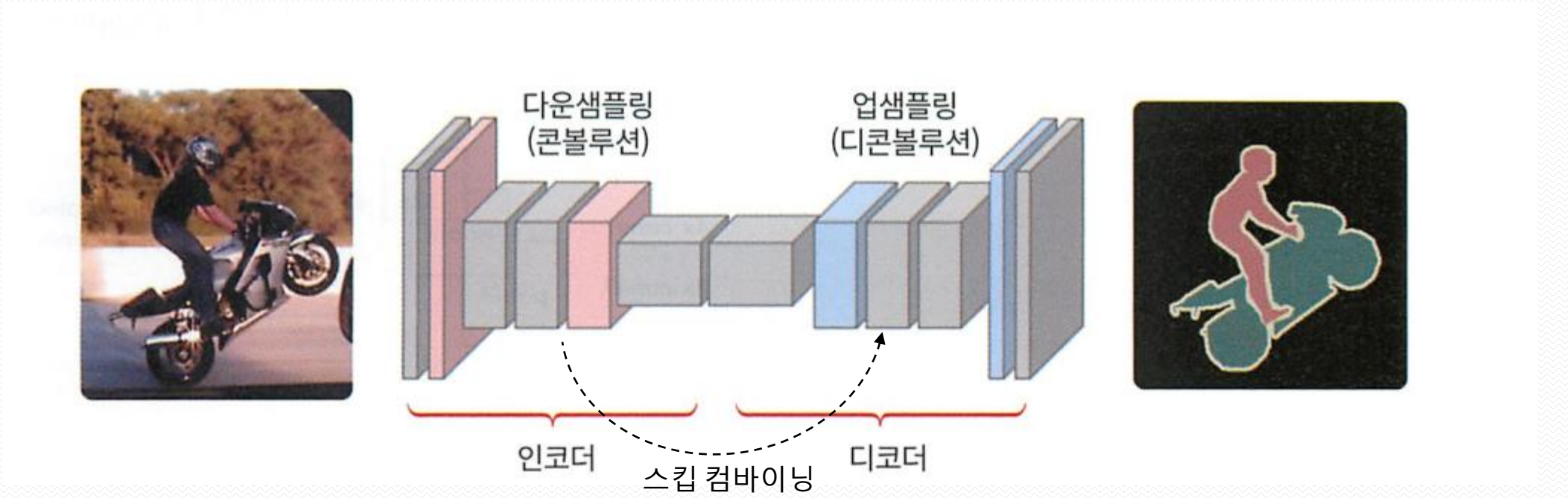
Segmentation

✓ 이미지의 각 픽셀은 범주에 따라 분류되므로 픽셀 수준 예측.

✓ 덴스 예측(dense prediction)이라고 부르기도 함.

Segmentation

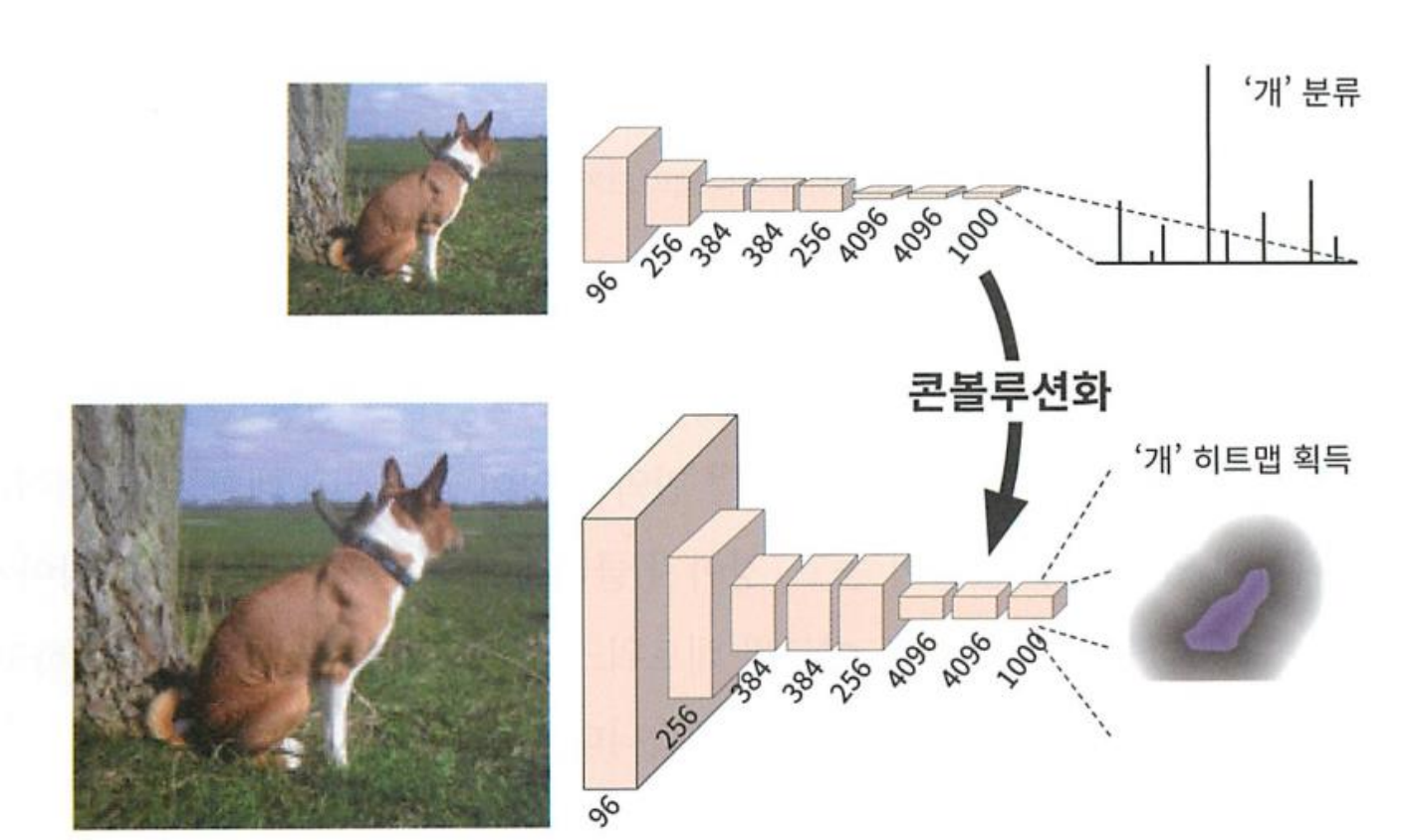
- ✓ 완전한 콘볼루션화(fully convolutionalization)
- ✓ 디콘볼루션(deconvolution)
- ✓ 스킵 아키텍처(skip architecture)



Fully convolutionalization

✓ FC 계층을 모두 Conv 계층으로 대체해 히트맵을 획득

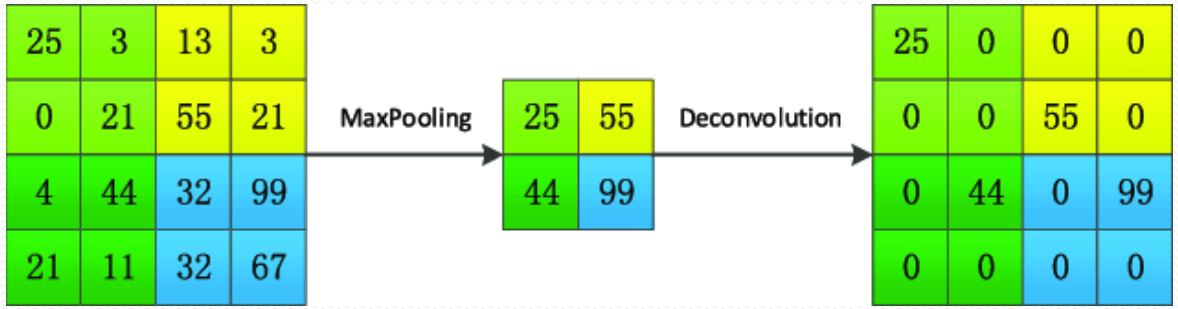
✓ 마지막 특징 맵의 개수는 훈련된 클래스의 개수와 동일

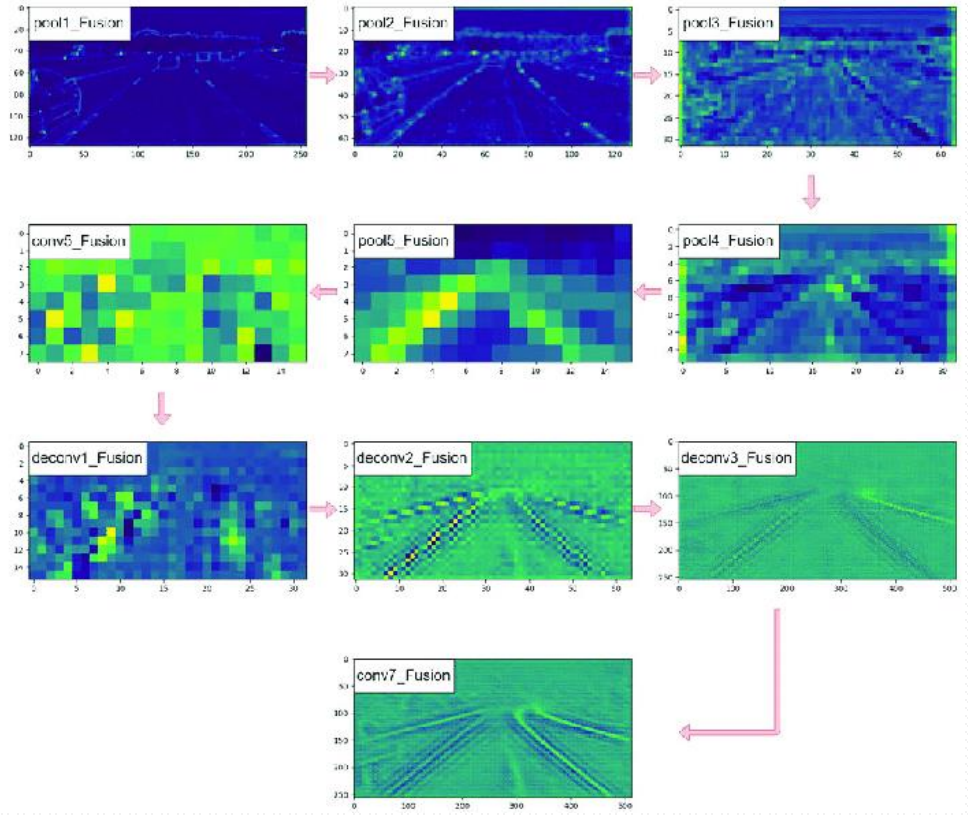


Deconvolution

✓ 특징 맵을 이용한 이미지 분할의 결과물은 이미지 원본 크기에서 픽셀 단위 예측에 사용하기에는 해상도가 너무나 부족

✓ 디콘볼루션(deconvolution)이라는 과정을 통해 덴스 결과 맵(dense output map)이라고 하는 해상도가 높은 상태로 만듦



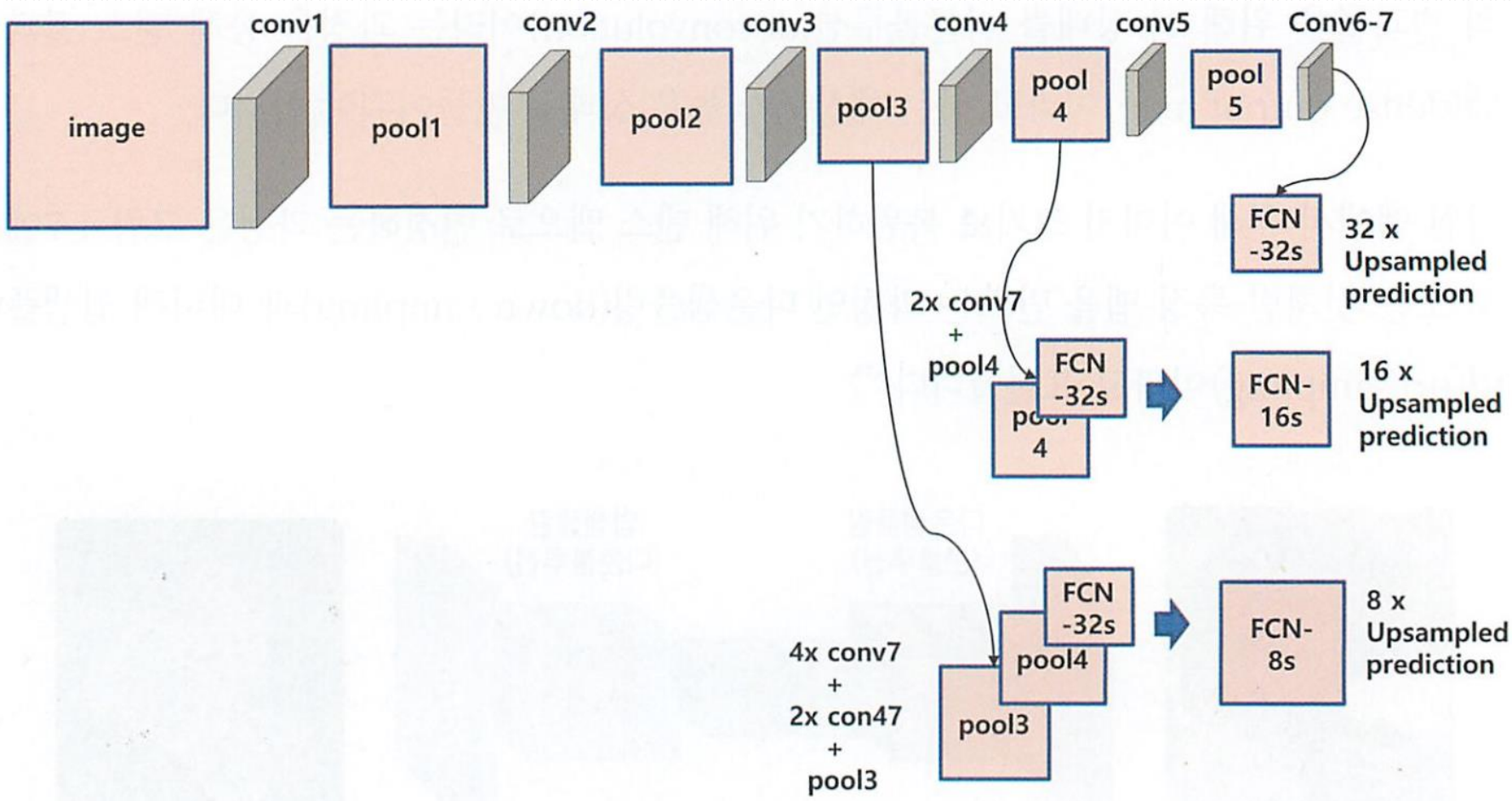


https://www.researchgate.net/publication/336571511_Multi-Lane_Detection_Based_on_Deep_Convolutional_Neural_Network

스킵 구조

✓ 이전 콘볼루션 층의 특징 맵을 참고해 업샘플링해 줄 때 정확도를 높이는 방법

✓ 정확도를 높일 수 있는 이유는 이전 콘볼루션 층의 특징 맵들이 해상도 면에서는 더 좋기때문

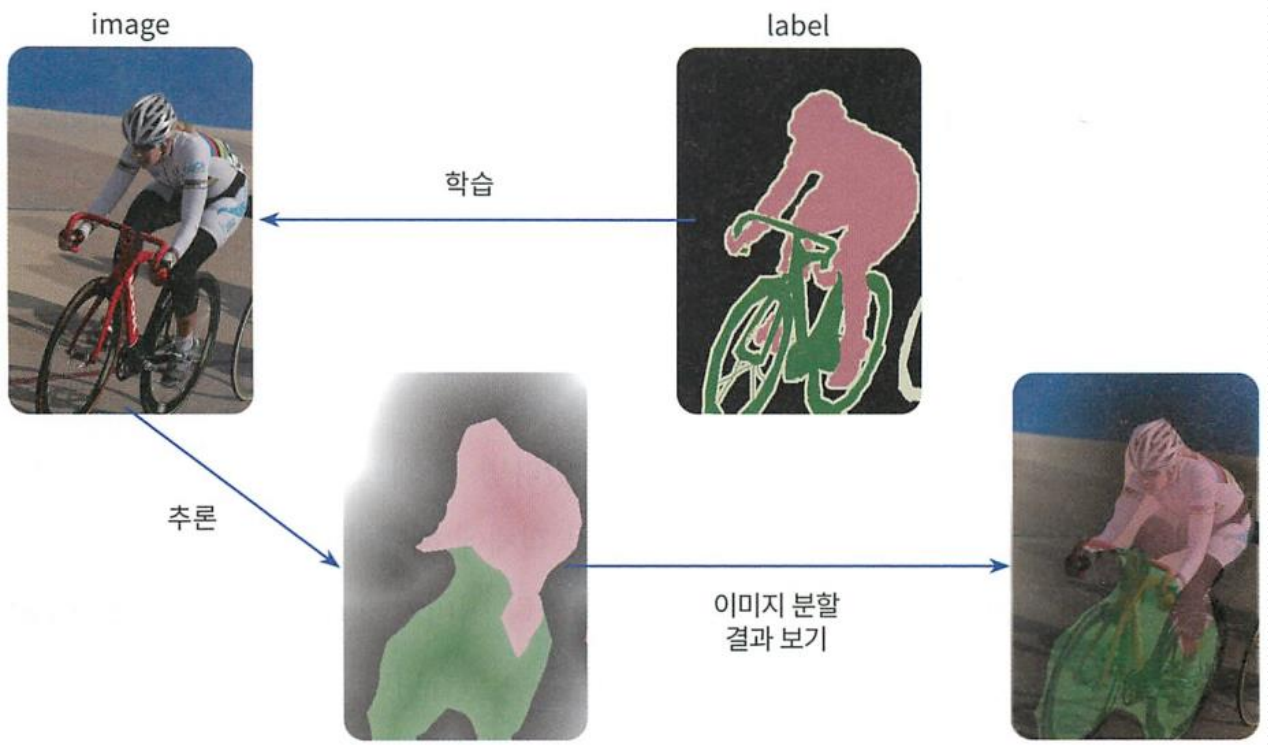


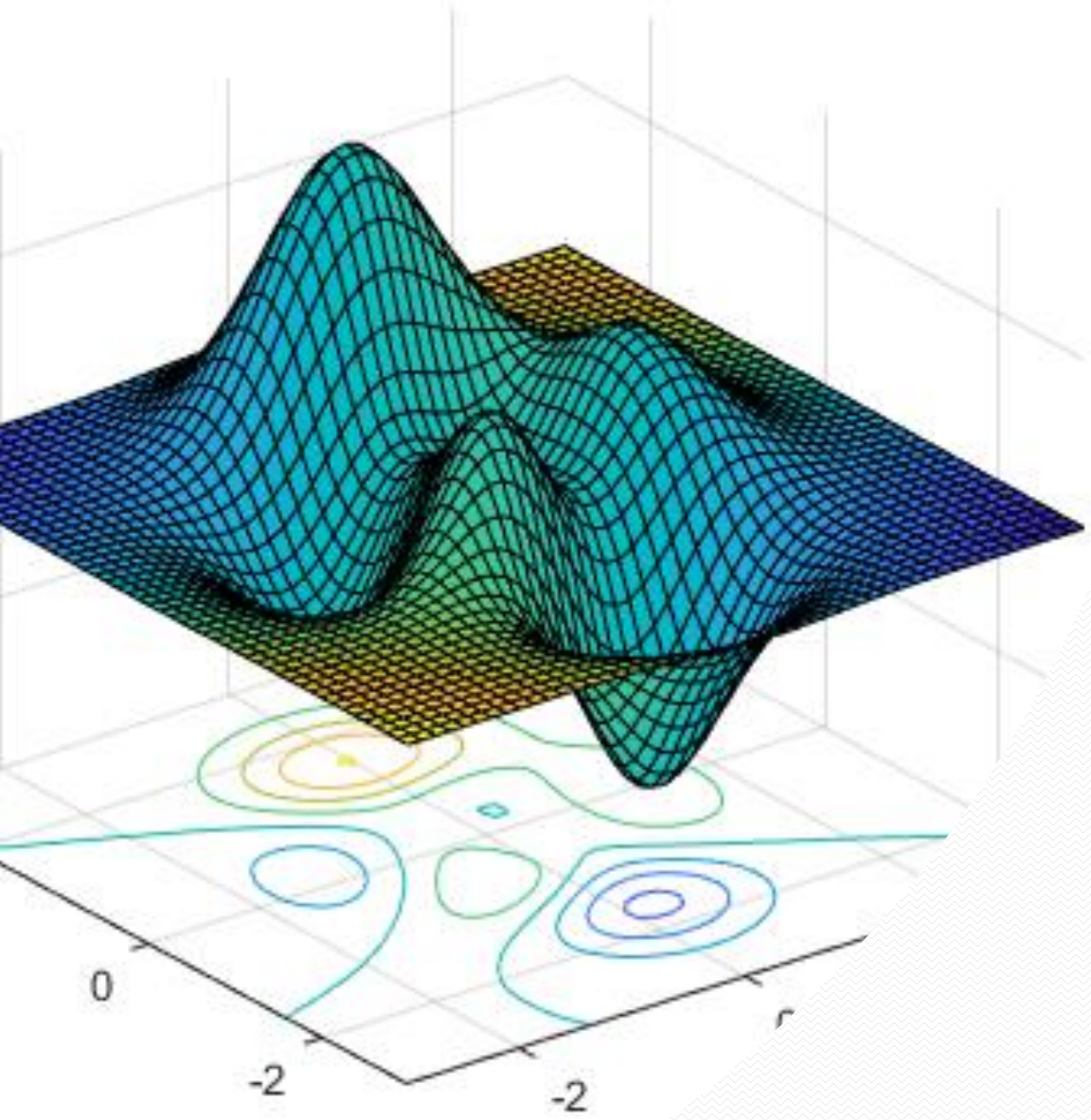
Segmentation

✓ 콘볼루션 과정의 입력은 사이즈 제약이 필요 없음.

✓ 콘볼루션의 결과인 특징 맵에는 사물의 위치 정보가 들어 있음.

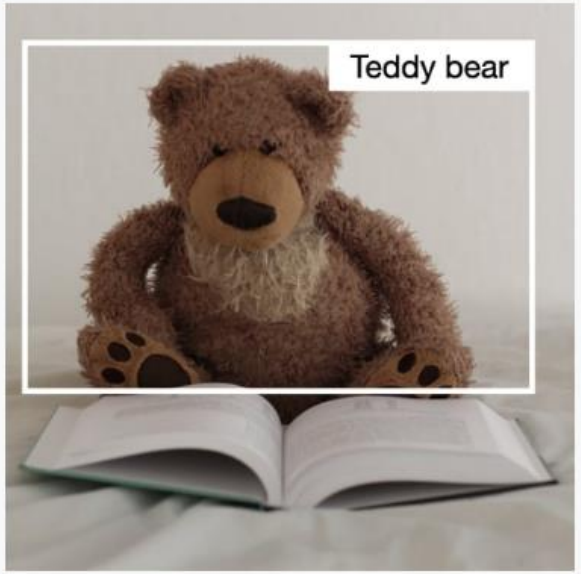
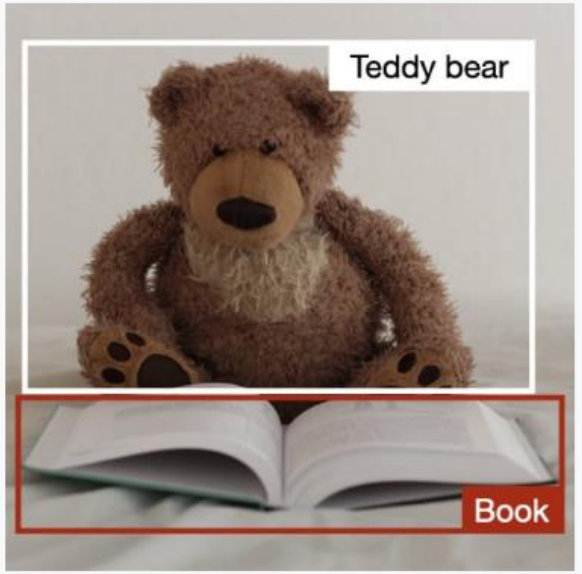
✓ 최종 특징 맵뿐 아니라 중간 레이어의 특징 맵도 유용하게 이용할 수 있음.



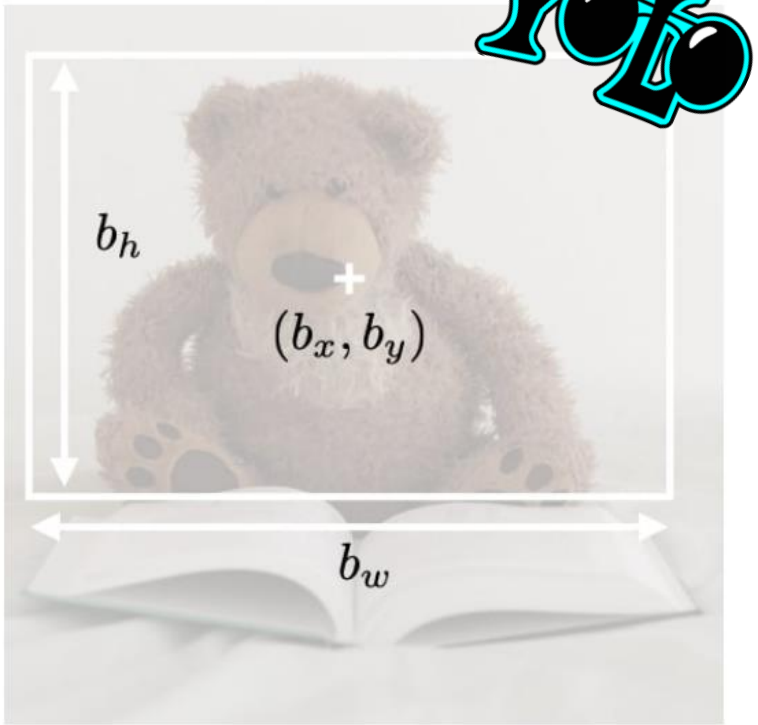
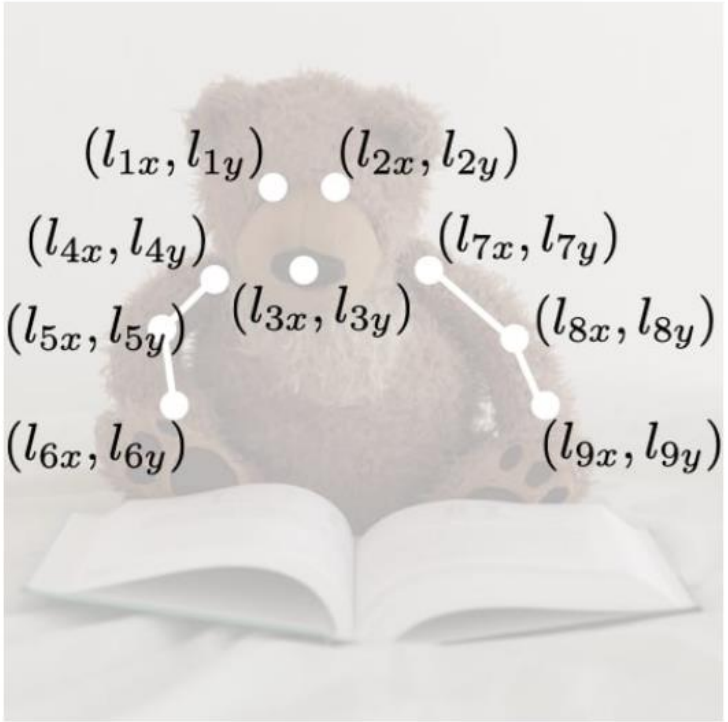


Object Detection

Object Detection

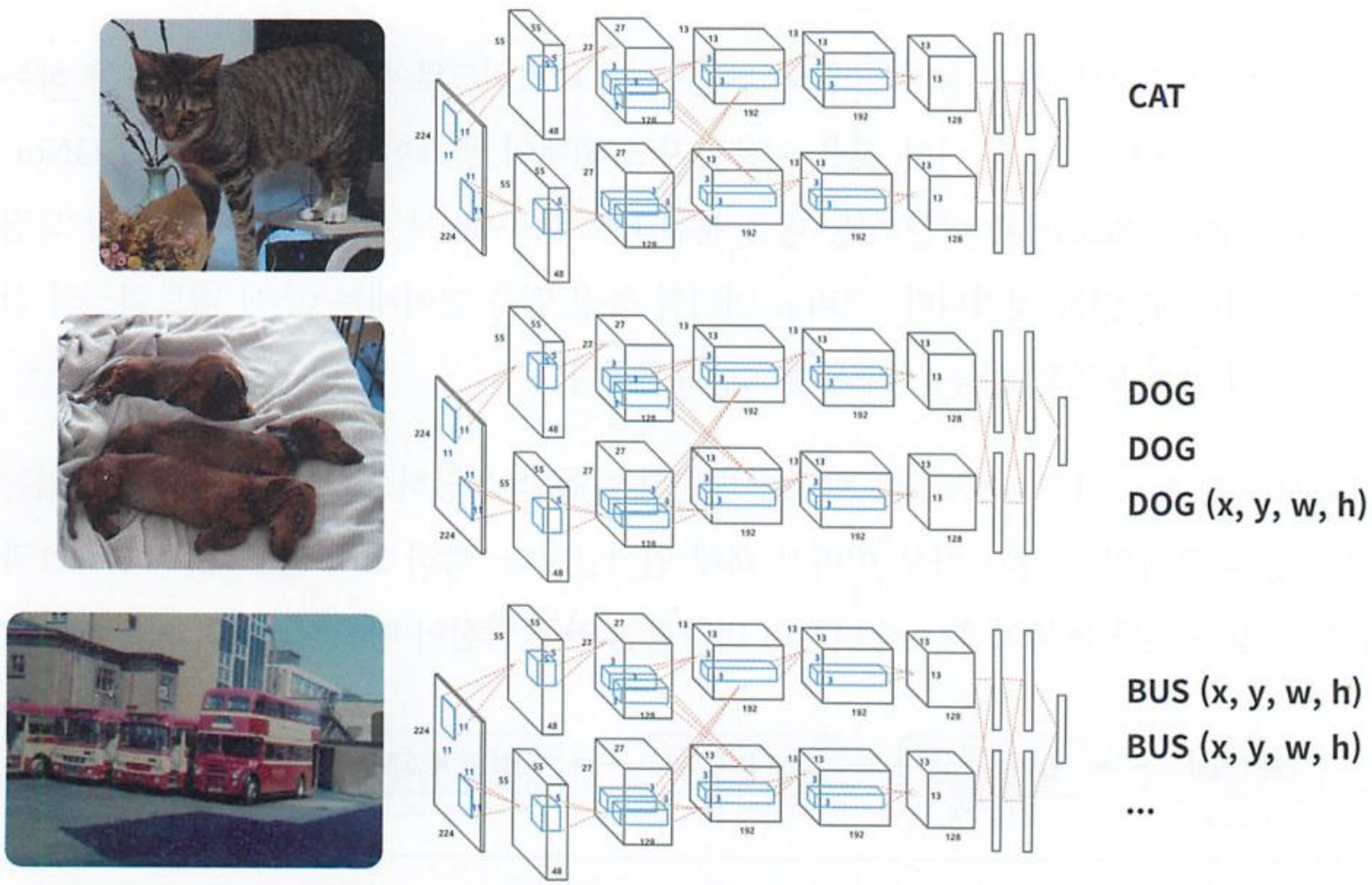
Image classification	Classification w. localization	Detection
		
• Classifies a picture • Predicts probability of object	• Detects an object in a picture • Predicts probability of object and where it is located	• Detects up to several objects in a picture • Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO R-CNN

Object Detection (cont.)

Bounding box detection	Landmark detection
• Detects the part of the image where the object is located	• Detects a shape or characteristics of an object (e.g. eyes) • More granular
	
Box of center (b_x, b_y), height b_h and width b_w	Reference points (l_{1x}, l_{1y}), ..., (l_{nx}, l_{ny})

Object Detection

✓ 이미지에서 특정 대상을 검출하고자 할 때 멀티 레이블 이미지 분류와 바운딩 박스 회귀를 위한 기본적인 영역을 정하기 어려운 문제가 있음



OBJECT DETECTION

2-STAGE

REGION PROPOSAL

SELECTIVE SEARCH
REGIONAL PROPOSAL NETWORK

PLAIN CNNS

CLASSIFICATION

FOR EACH PROPOSED REGION

MULTI-CLASS CLASSIFICATION

BOUNDING BOX REGRESSION

1-STAGE

CNN

CONV LAYERS

FEATURE EXTRACTION

...

FEATURE MAPS

FOR EACH GRID / SPATIAL LOCATION

1-Stage & 2-Stage Detector

2-Stage

Input Image

Region Proposal

- Selective Search
- Region Proposal Network

Classification

For Each Proposed Region

Multi-Class Classification

Bounding Box Regression

Output

1-Stage

Input Image

Conv Layers

Feature Extraction

Feature Maps

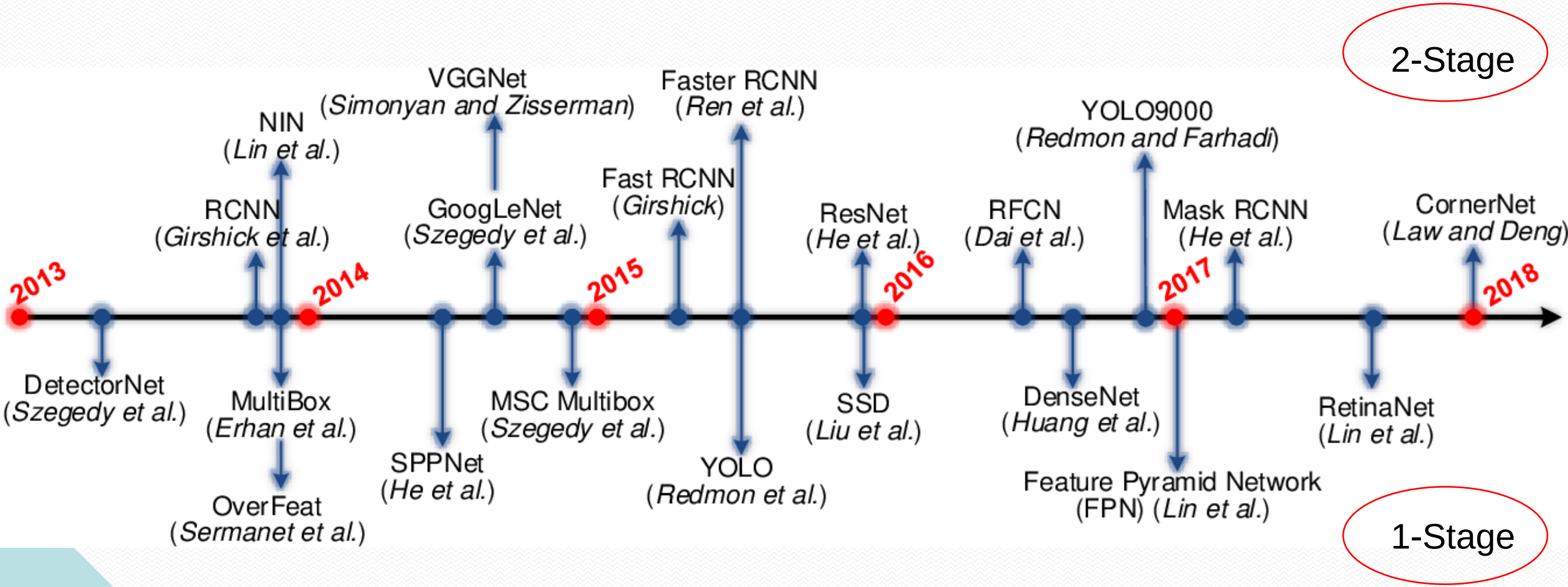
For Each Grid or Spatial Location

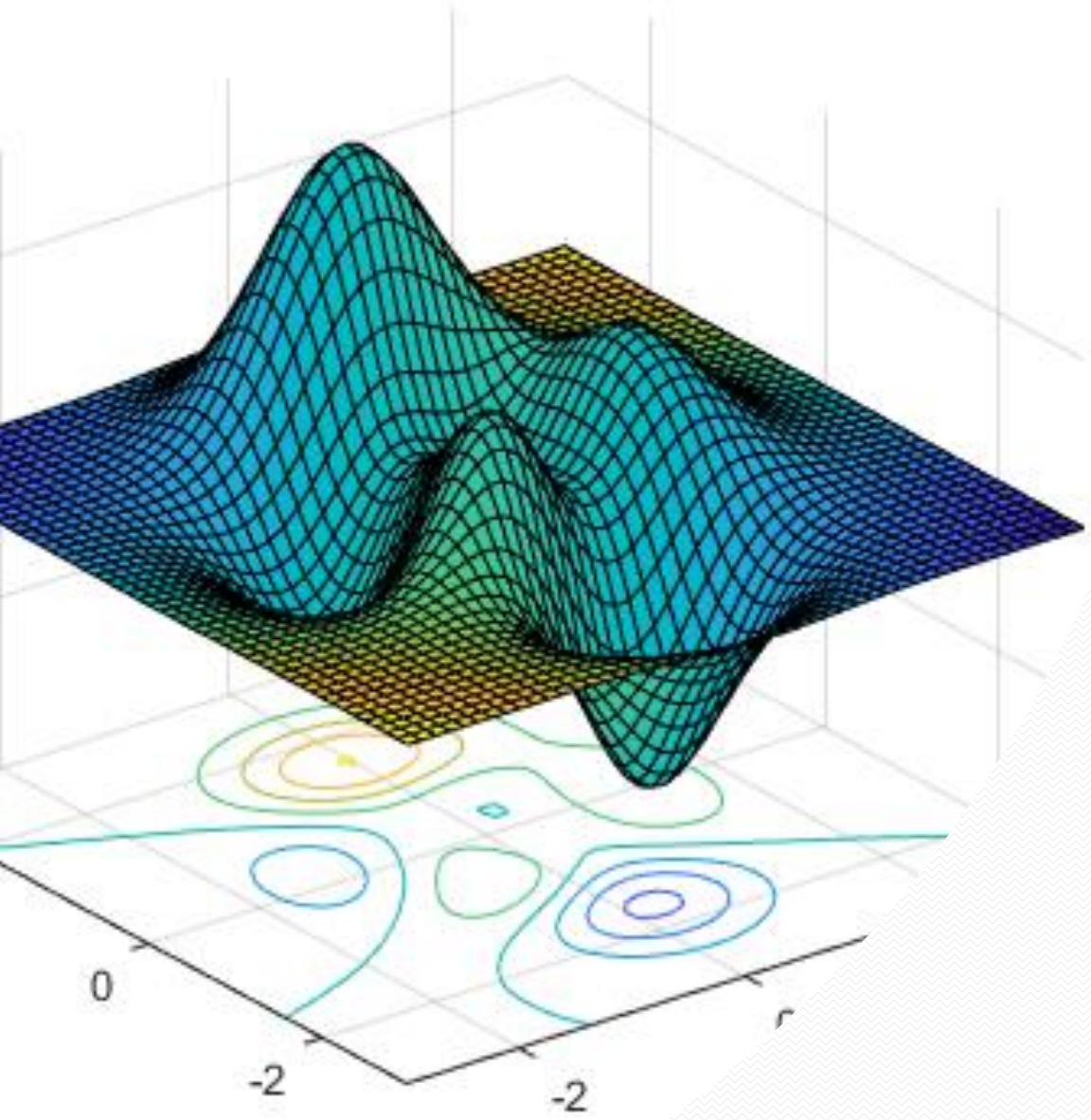
Multi-Class Classification

Bounding Box Regression

Output

1-Stage & 2-stage Detector (cont.)

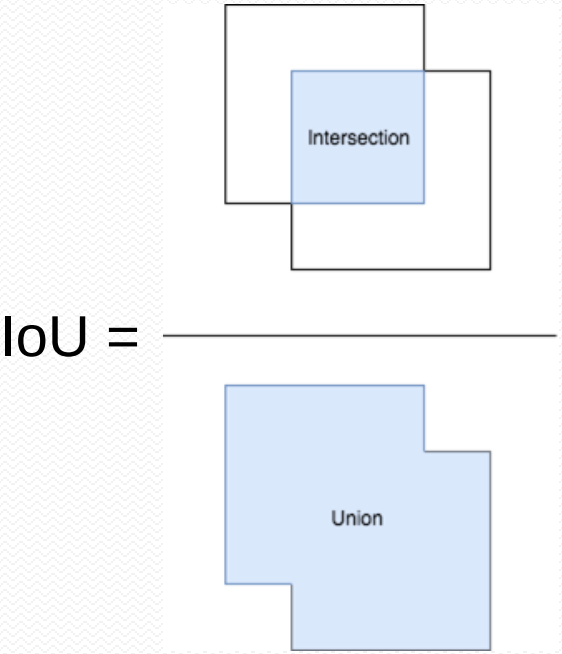




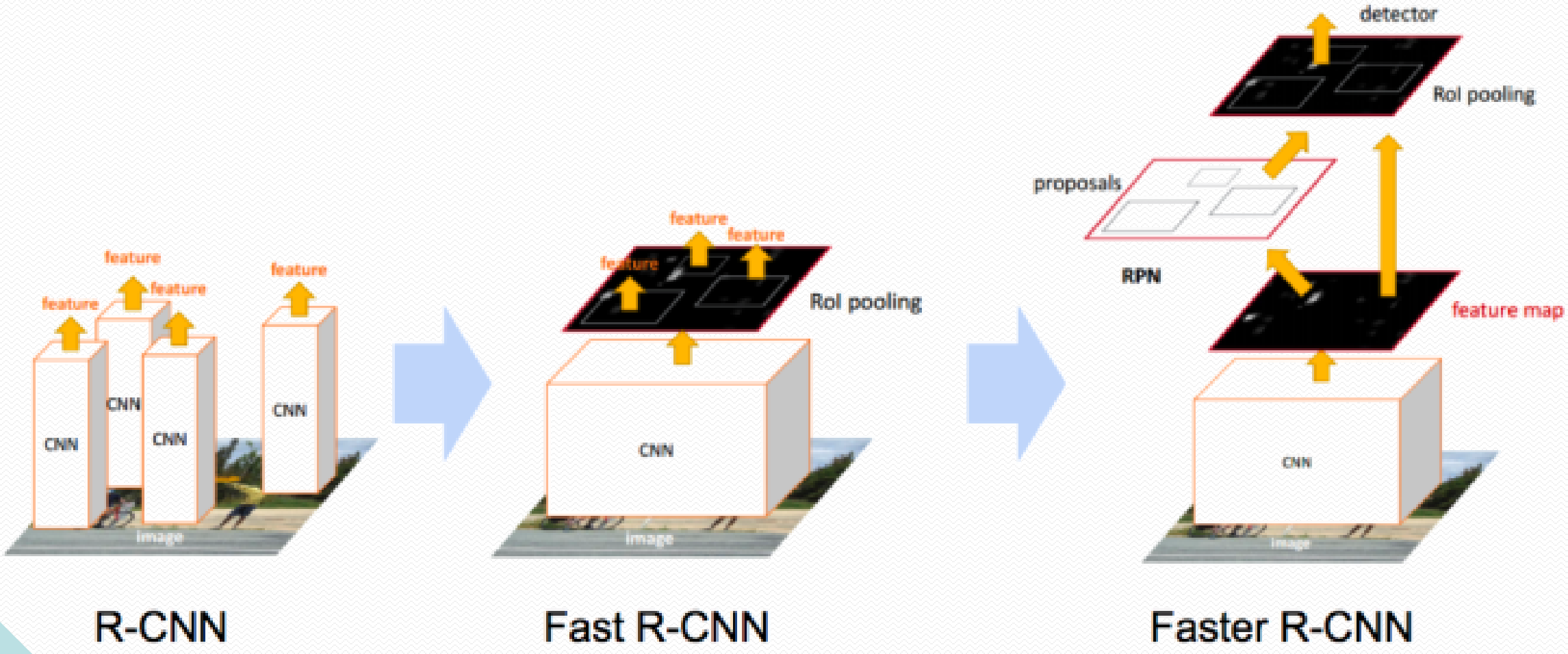
2 stage

NMS(Non-Maximum Suppression)

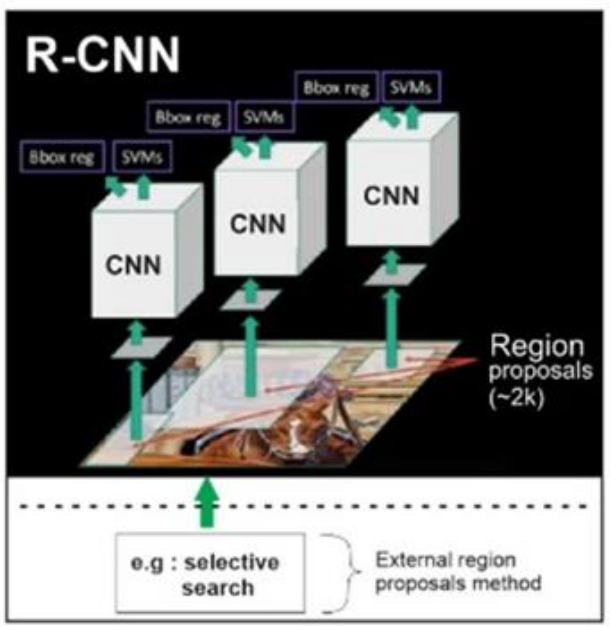
- If there are multiple boxes on the same object, only the box with the highest score is left

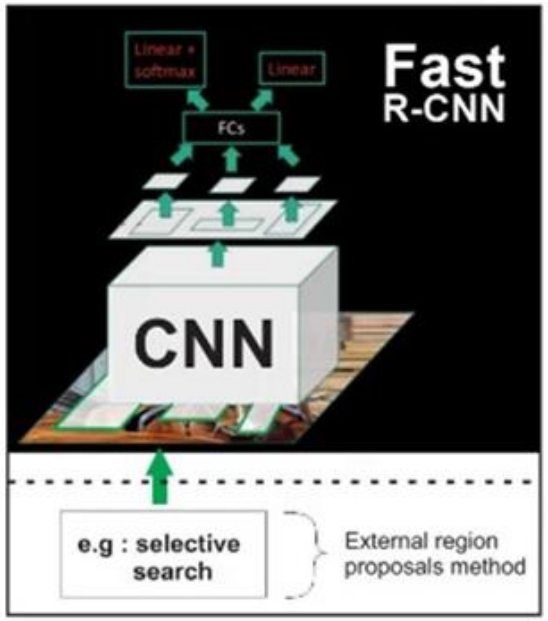


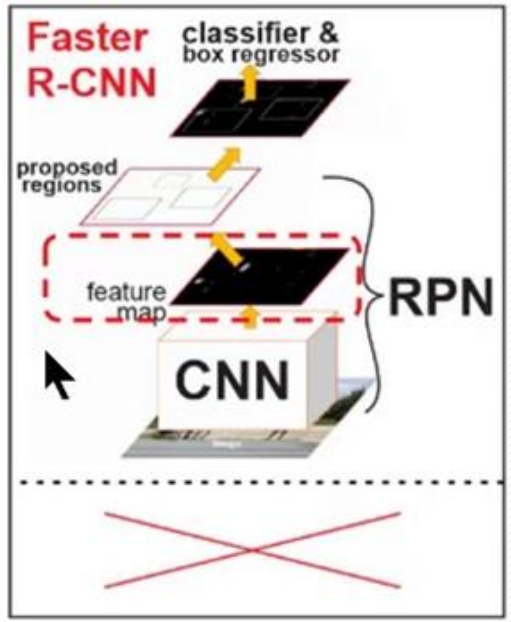
Evolution from R-CNN to Faster R-CNN



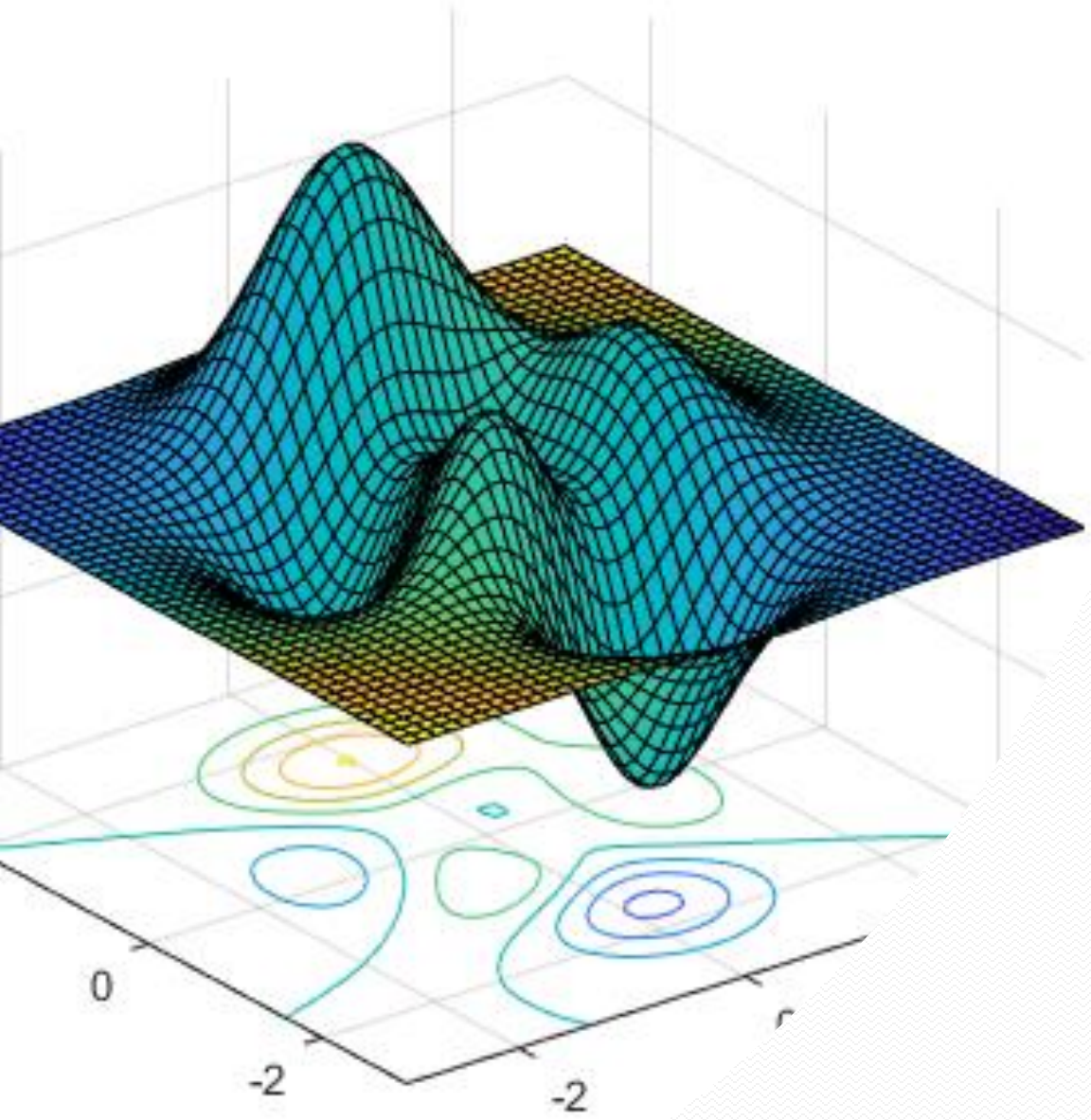
Compare performance





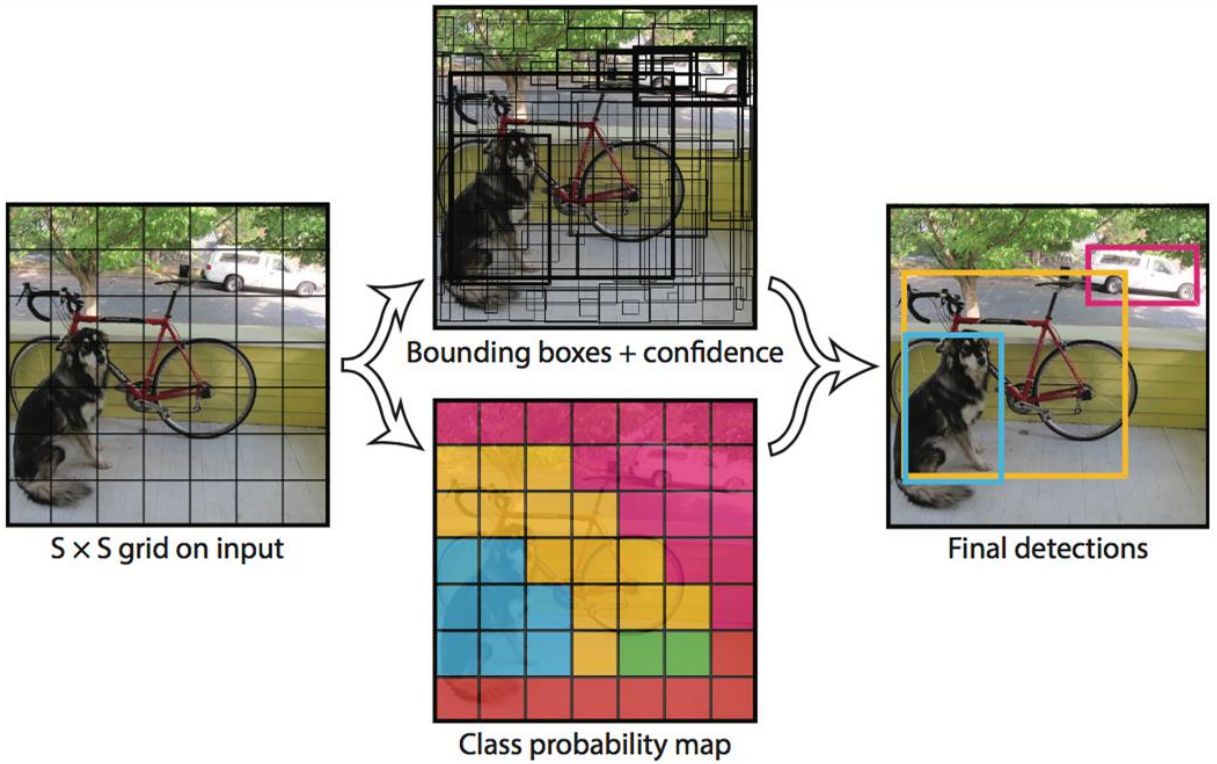


	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

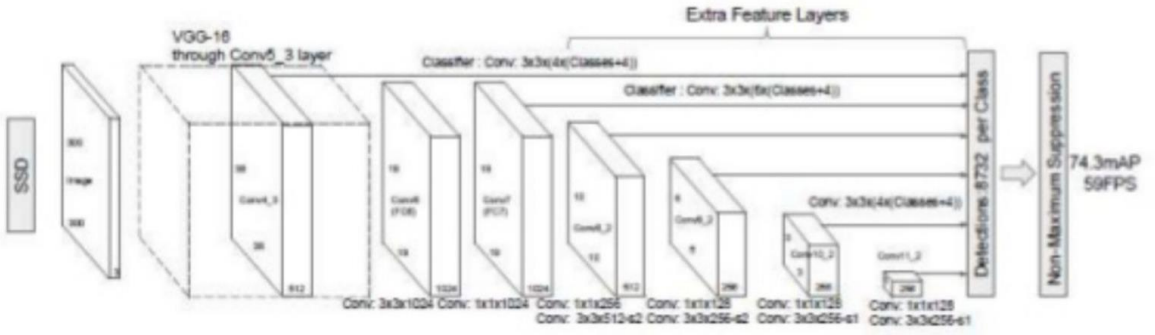


1 stage

1 STAGE OBJECT DETECTION



YOLO

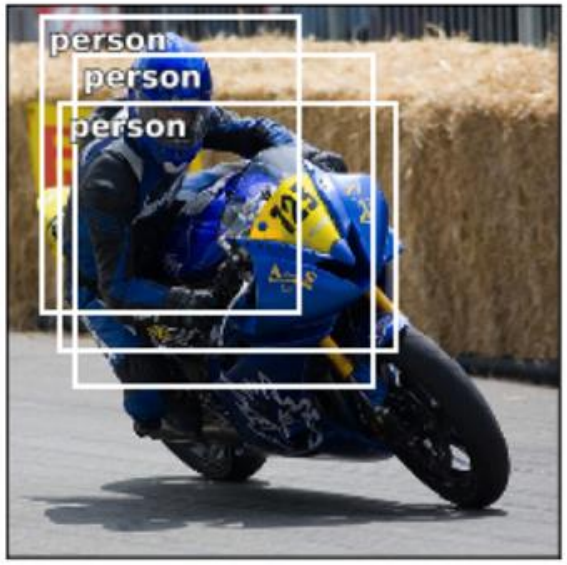


SSD

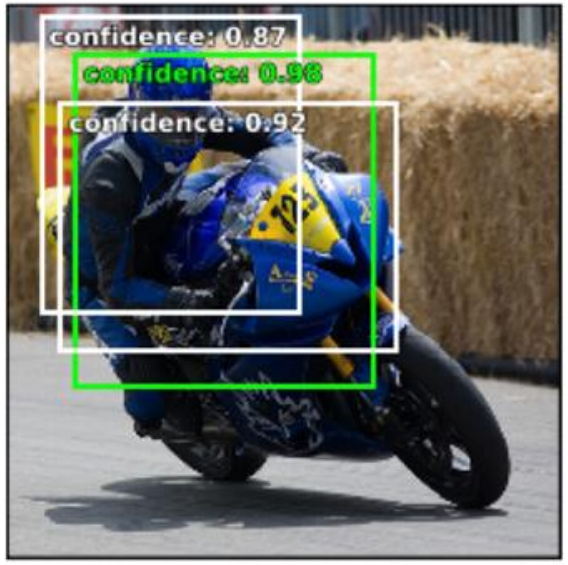
Predictions on a grid (cont.)

Repeat with next highest confidence prediction until no more boxes are being suppressed

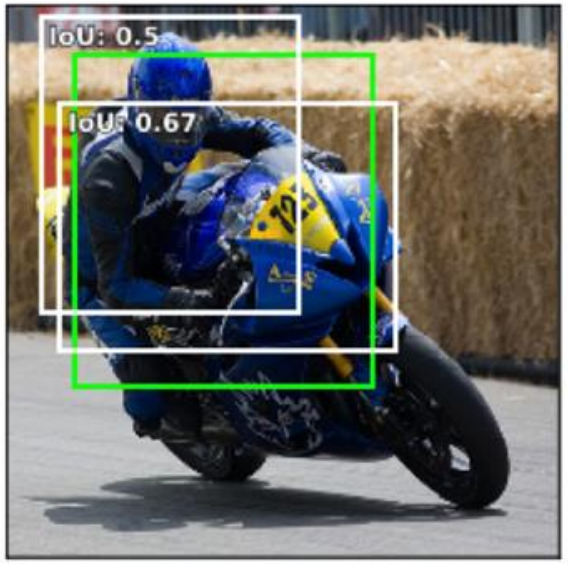
For each class...



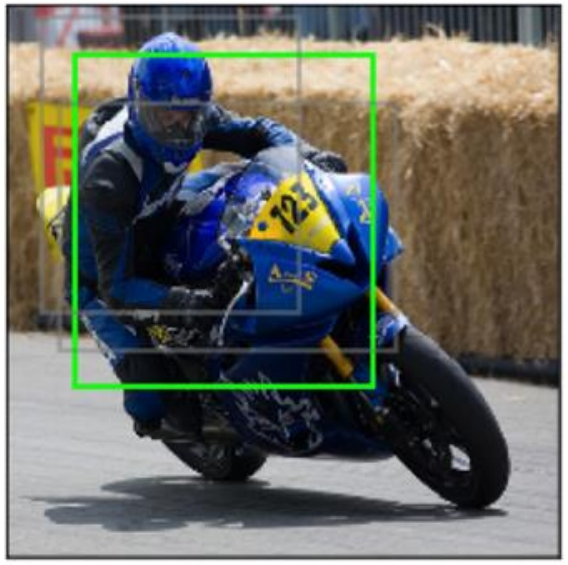
After filtering out low confidence predictions, we may still be left with ***redundant*** **detections**



Select the bounding box prediction with the **highest confidence**



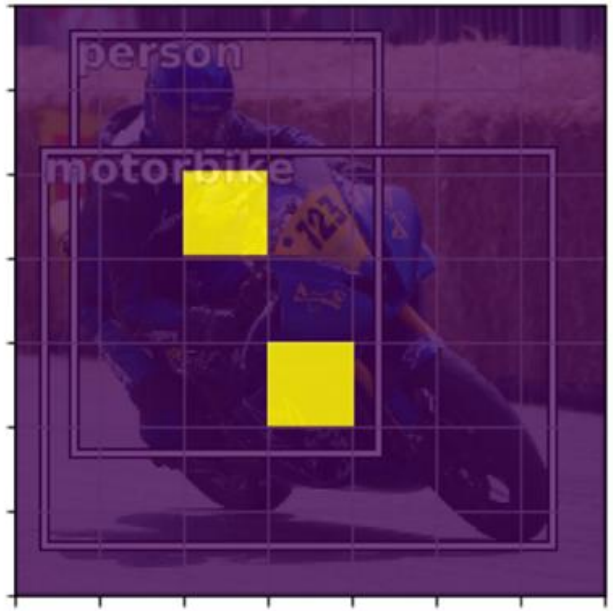
Calculate the IoU between the selected box and all remaining predictions

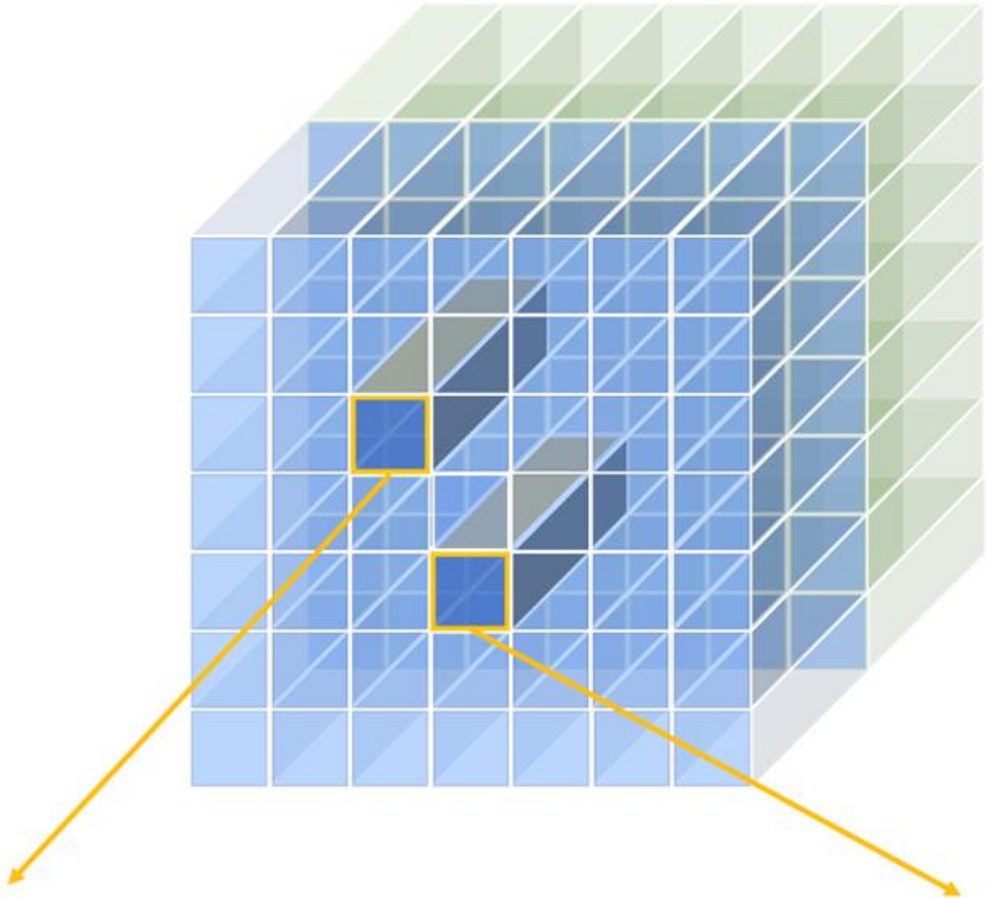


Remove any boxes which have an IoU score above some defined threshold

Predictions on a grid (cont.)

multiple objects can be detected in parallel



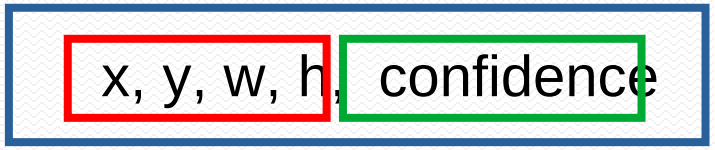


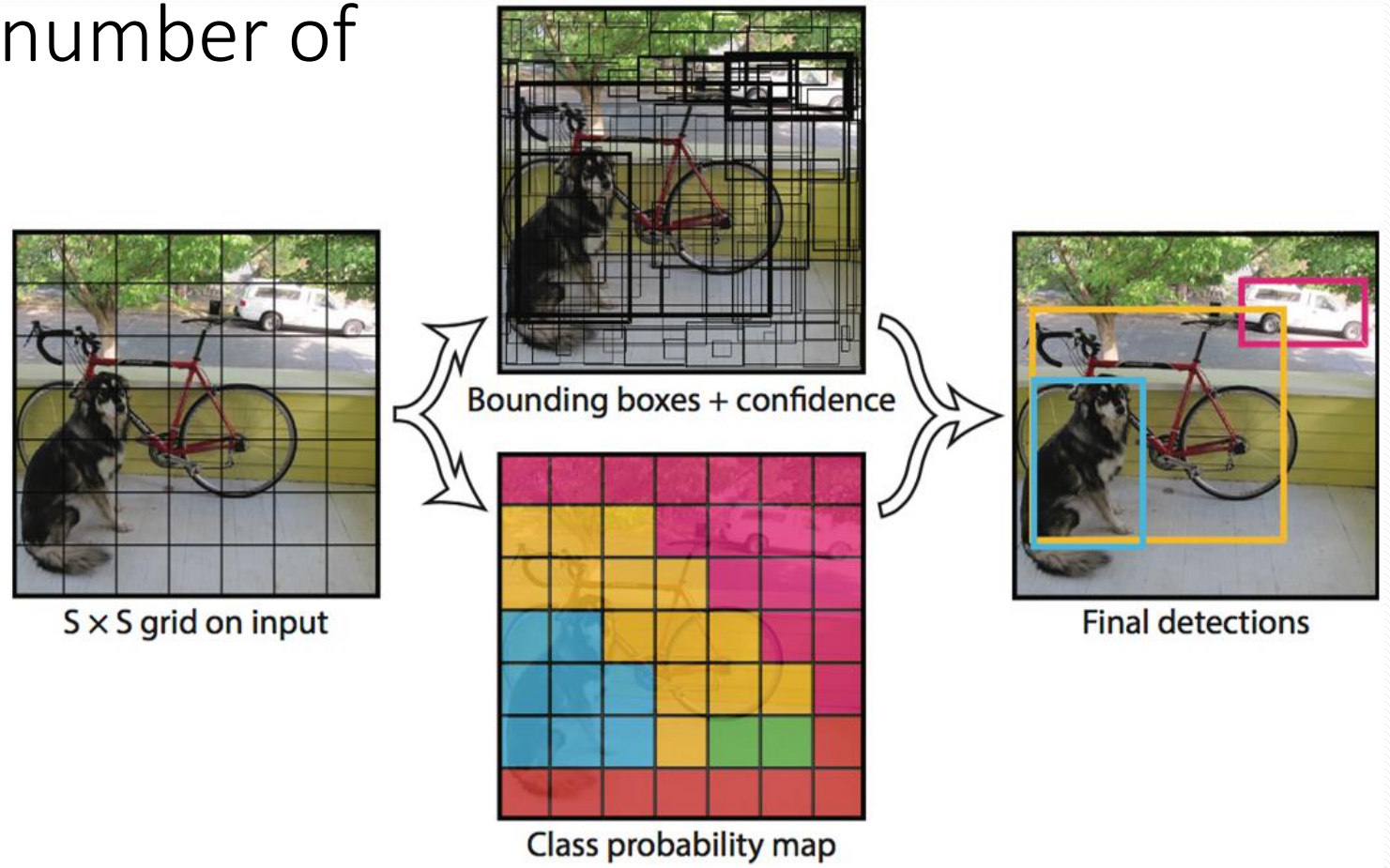
Person bounding box descriptor

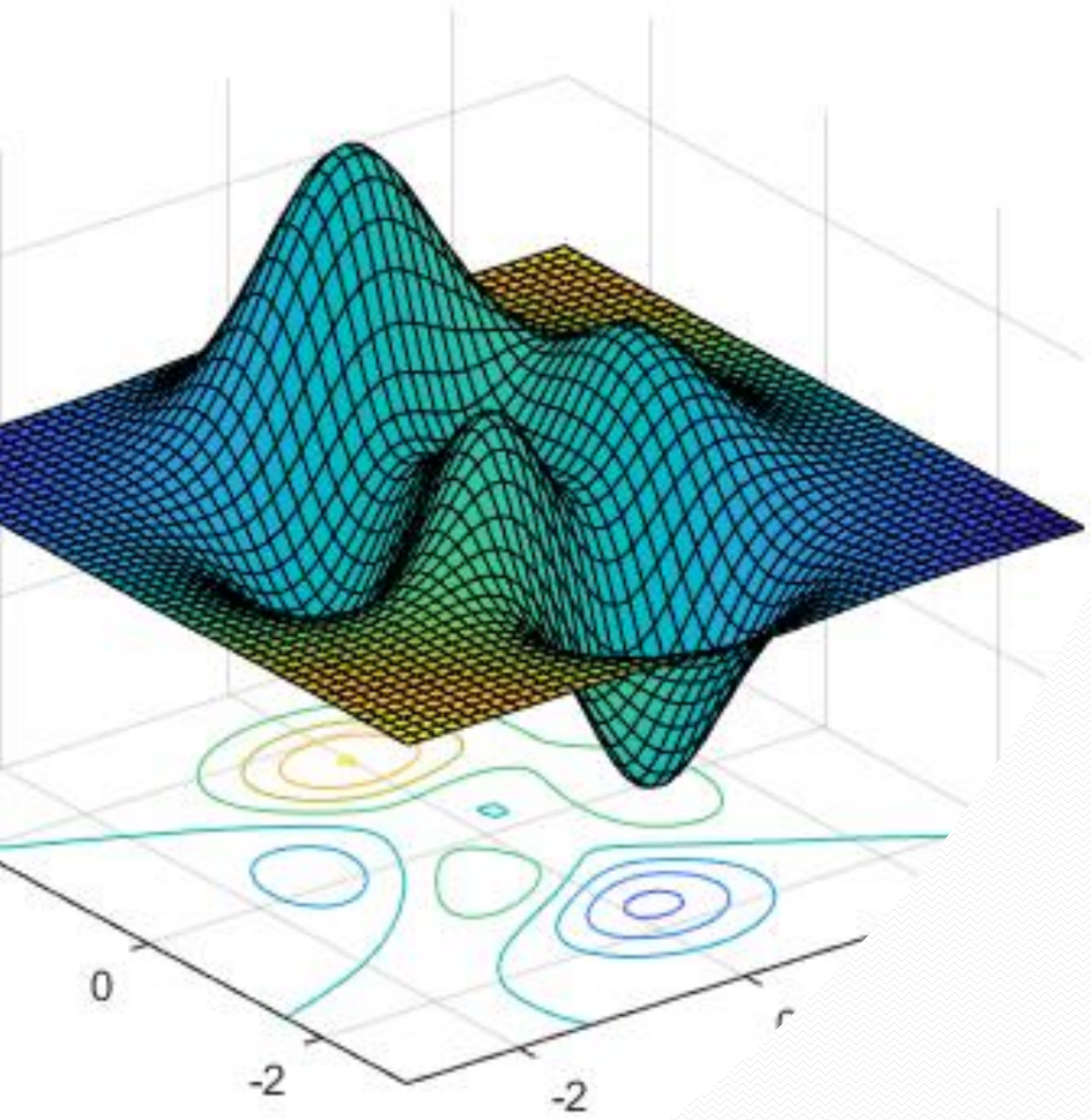
Motorbike bounding box descriptor

Yolo

- Predict one set of class probabilities per grid cell, regardless of the number of boxes B
- Image
  - **S** x **S** grids
- Grid cell
  - **B**: Bboxes & Confidence score
  - **C**: class probabilities
  - with regard to #classes







CONFUSION MATRIX

CONFUSION MATRIX

QUANTIFIABLE MEASURE THAT IS USED TO TRACK AND ASSESS THE STATUS OF A SPECIFIC PROCESS

Model classification & answer → TRUE / FALSE

- True Positive / TP
- False Positive / FP
- False Negative / FN
- True Negative / TN

		ACTUAL	
		POSITIVE	NEGATIVE
PREDICTED	POSITIVE	**TRUE** POSITIVE	FALSE POSITIVE
	NEGATIVE	FALSE NEGATIVE	**TRUE** NEGATIVE

PRECISION / RECALL

		ACTUAL	
		POSITIVE	NEGATIVEW
PREDICTED	POSITIVE	TRUE POSITIVE	FALSE POSITIVE
	NEGATIVE	FALSE NEGATIVE	TRUE NEGATIVE

Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\#\text{ predictions}}$$

Proportion of what the actual true among the model classifies as true

Accuracy rate

Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\#\text{ ground truths}}$$

Proportion of what the model predicted to be true among the actual true

Detection rate

Precision = 100%

90	0
10	

		ACTUAL	
		강아지	고양이
PREDICTED	강아지	**90**	0
	고양이	10	**90**

Recall = 90%

90	0
10	

IOU INTERSECTION OVER UNION

EVALUATION METRIC USED TO MEASURE THE ACCURACY OF AN OBJECT DETECTOR

MOSTLY USED IN OBJECT DETECTION

PASCAL VOC challenge

$$IoU = \frac{TP}{(TP + FP + FN)}$$

PREREQUISITES

The ground-truth bounding boxes = hand-labeled bounding boxes

The predicted bounding boxes from the model.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

THANK YOU