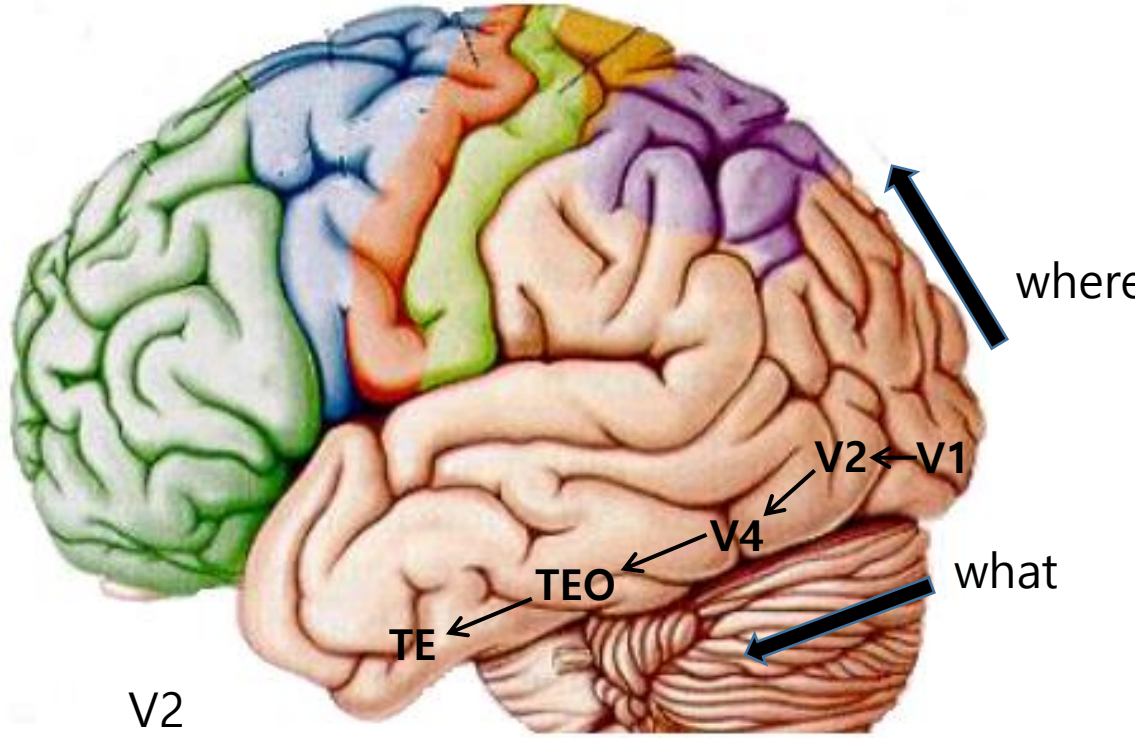
Machine Learning

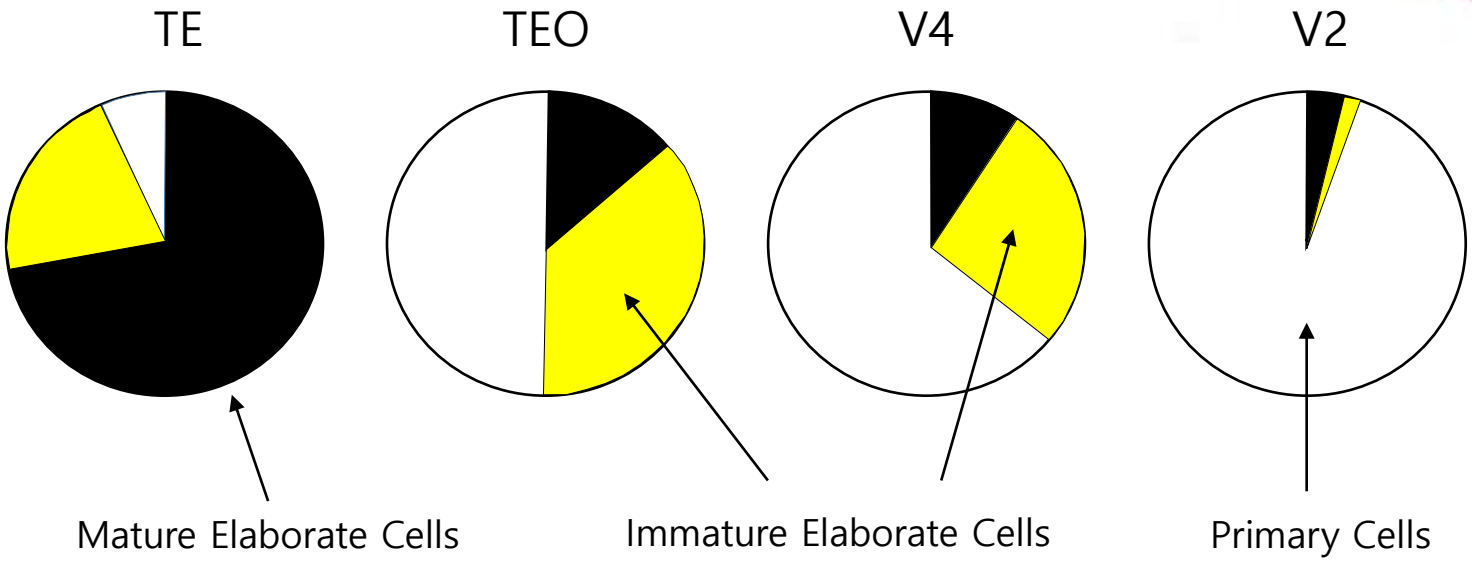
Contents

9.1. Historical Background of CNN

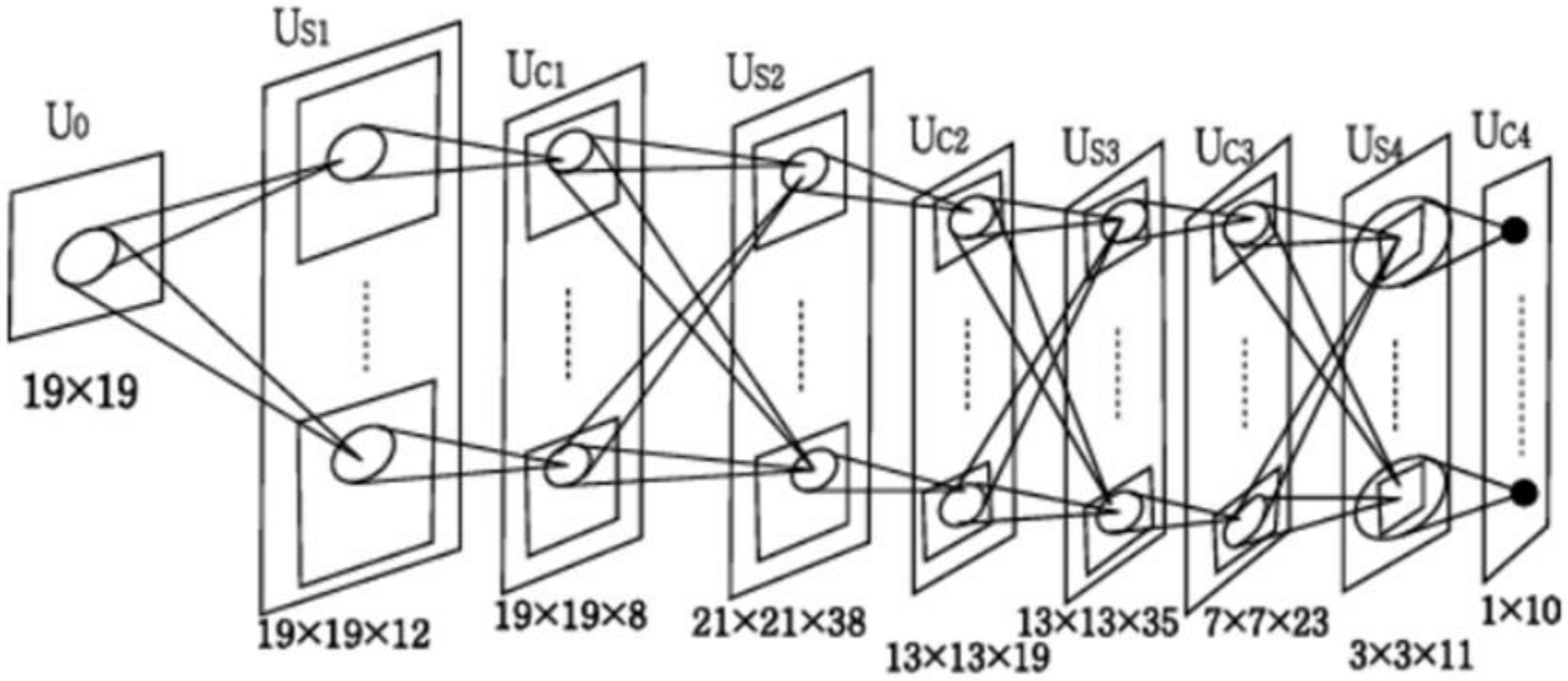
- Feed-forward Neural Networks (Chap. 5)
  - Fully-Connected
  - Too many weights for training
  - Slow learning convergence and High computational complexity
  - Overspecialization to training samples (poor generalization performance)
  - ➢ Locally-Connected Networks (Visual pathway of mammal)
- Visual Pathway
  - Hubel and Wiesel (1950s and 1960s)
    - Cat and monkey visual cortexes contain neurons that individually respond to small regions of the visual field. → Receptive Field
    - Simple Cells in V1 layer, whose output is maximized by straight edges having particular orientations within their receptive field
    - Complex Cells in V1 and V2 layer, which have larger receptive fields, whose output is insensitive to the exact position of the edges in the field.

- Visual Pathway
  - Keiji Tanaka (1990s)
    - Monkey visual cortexes
    - Input End: Visual cells respond to simple features
    - Output End: Visual cells respond to complex features

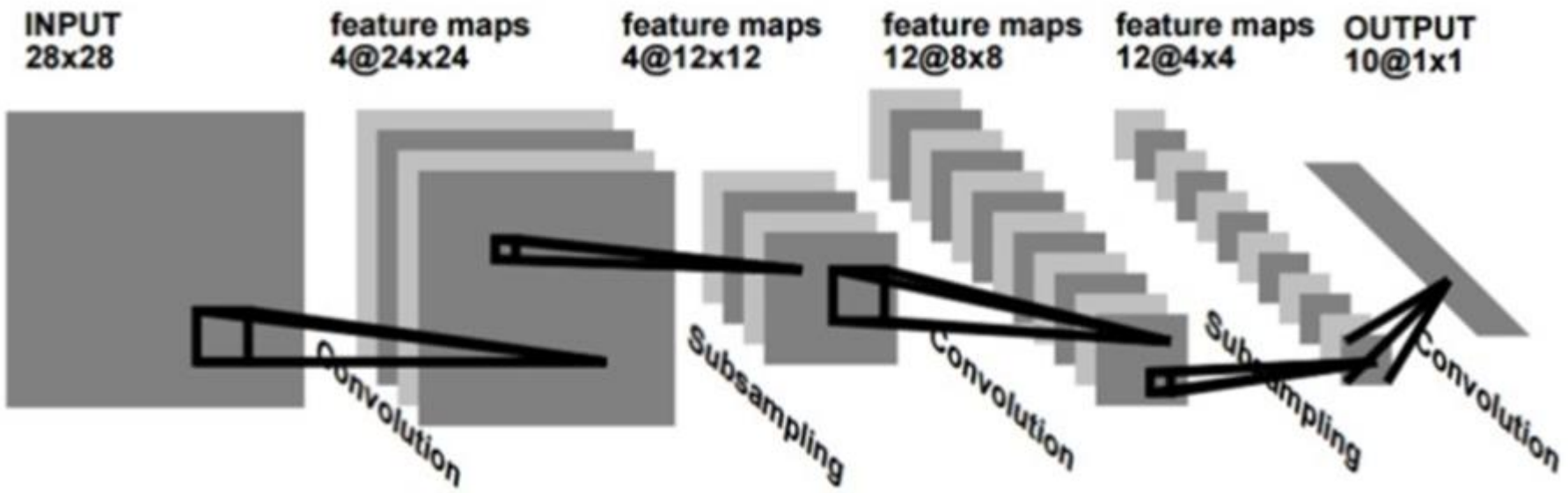




- Neural Networks
  - Fukushima (1980s)
    - Neocognitrion
    - S-cell : extracting local features
    - C-cell : tolerance to features' deformation such as local shifts



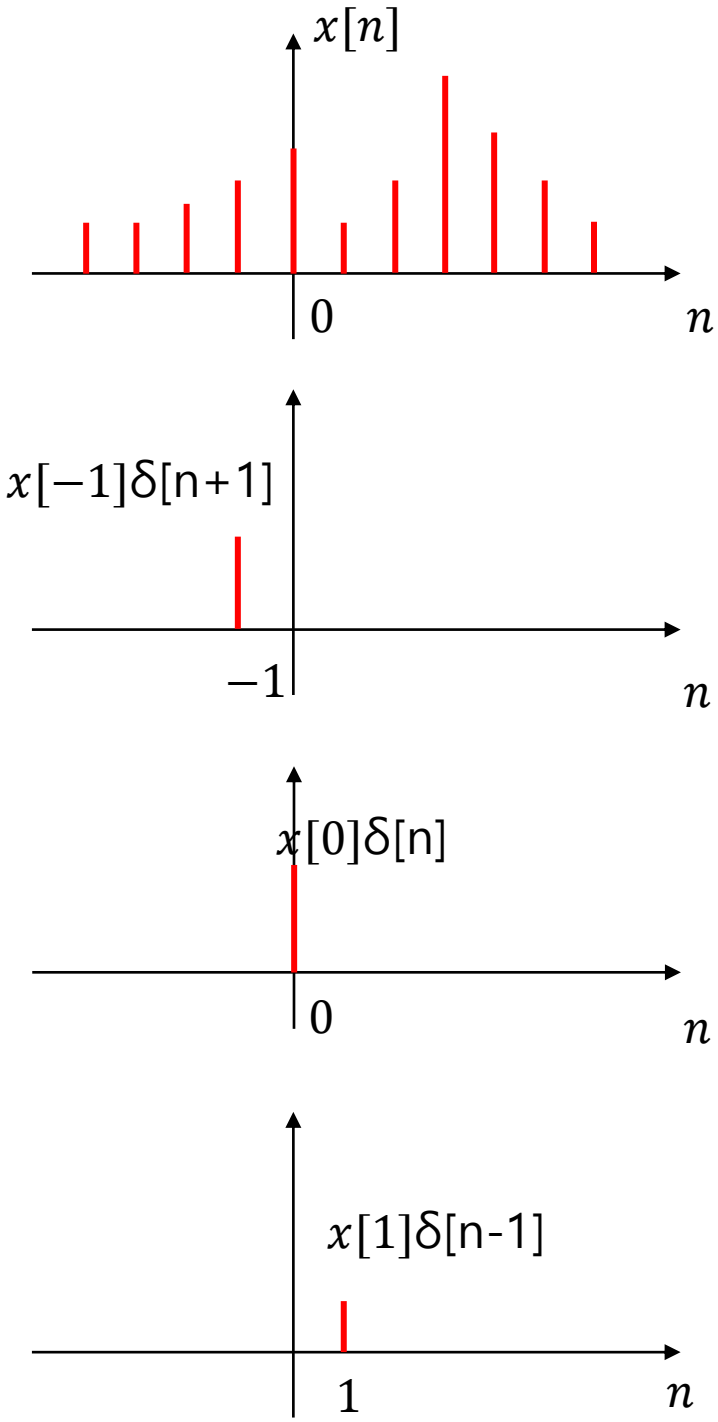
- Neural Networks
  - LeCun (1990s)
    - LeNet1



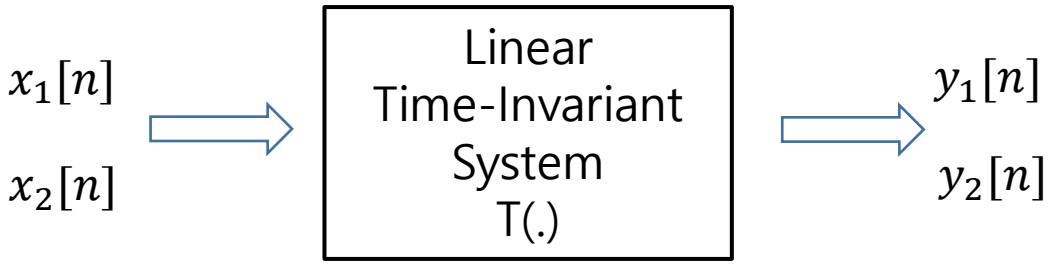
9.2. Convolution in (DSP)

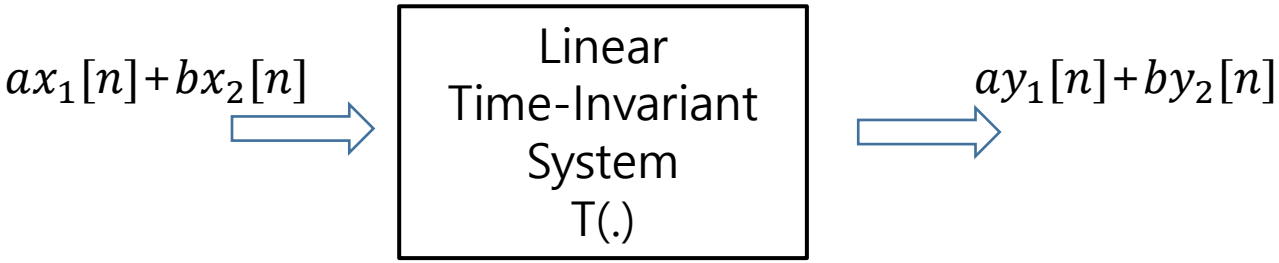
$$\delta[n] = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases}$$

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$



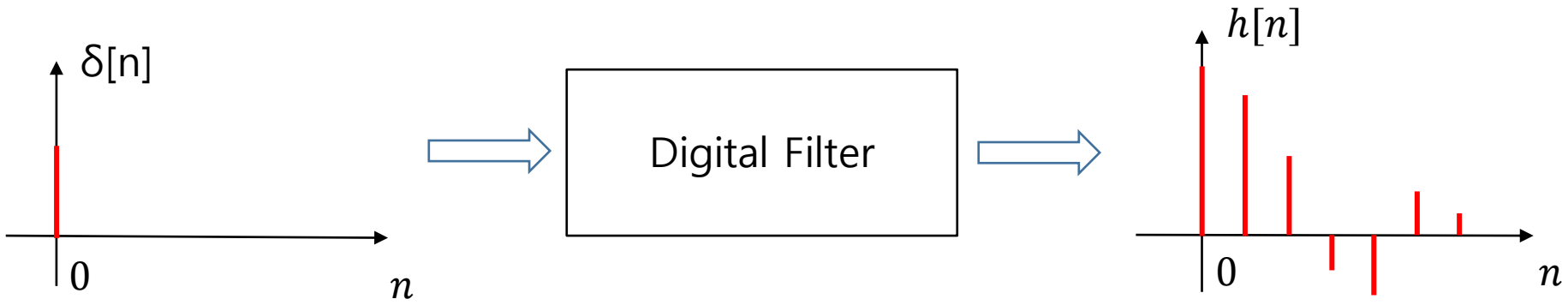
LTI(Linear Time Invariant) System





$$y[n] = T(ax_1[n] + bx_2[n]) = aT(x_1[n]) + bT(x_2[n]) = ay_1[n] + by_2[n]$$

LTI(Linear Time Invariant) System



Convolution

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = x[n] * h[n]$$

① 입력 $x[k]$에 디지털 필터 $h[k]$를 좌우대칭으로 변형하여 $h[-k]$를 만든 후,
② n만큼 이동한 $h[-(k-n)] = h[n-k]$를 곱하고,
③ 그 결과들을 모든 k에 대하여 더한 결과를 출력 함수 $y[n]$으로 내어 놓는다.
즉, 입력 $x[k]$에 필터함수 $h[k]$를 역방향으로 씌워서 필터 출력을 얻어내며,
④ 이를 모든 n에 대하여 이동하면서 출력 함수 $y[n]$을 구한다.

The Communitive Law

$$x[n] * h[n] = h[n] * x[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] = h[n] * x[n]$$

9.3. Convolutional Neural Network(CNN)

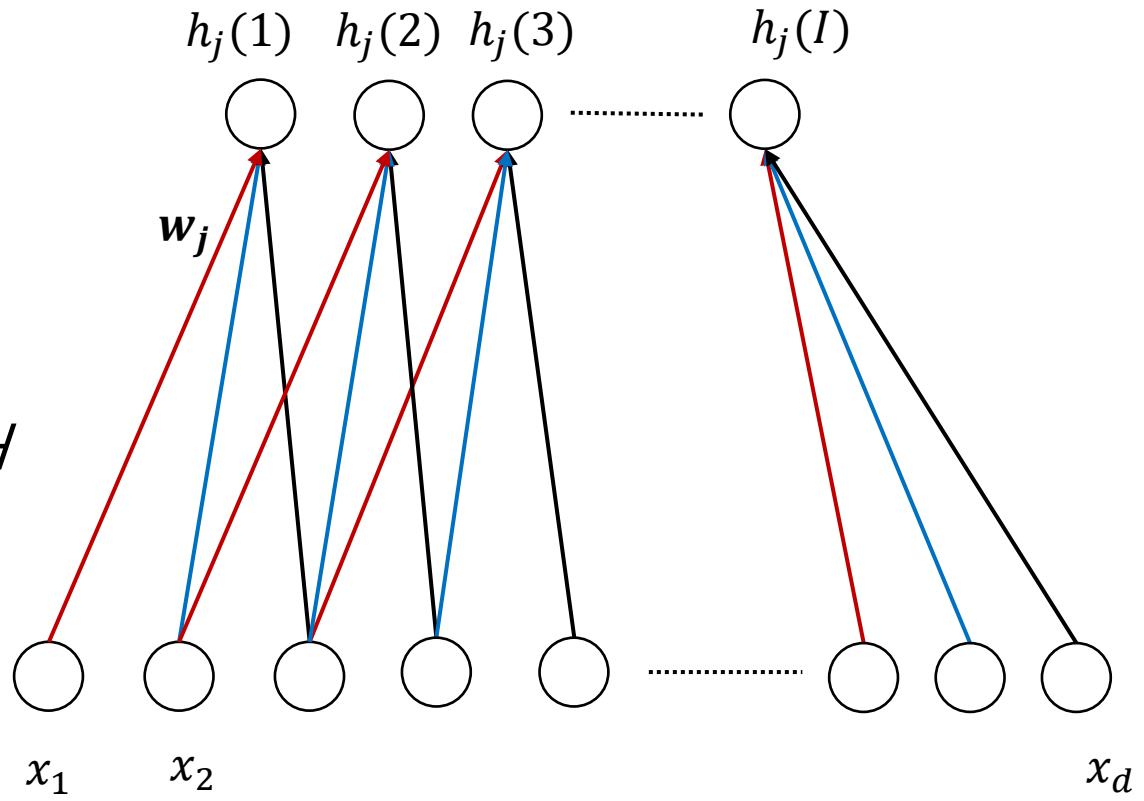
- CNN
  - consists of Convolution layer, Pooling Layer, and Fully-Connected Layer

- Convolution Layer
  - 1-D convolution

$$h_j[i] = \boldsymbol{w}_j * \boldsymbol{x} = \sum_{k=-(z-1)/2}^{(z-1)/2} w_j[k]x[i+k]$$

  - *H* convolution nodes : depth *H*

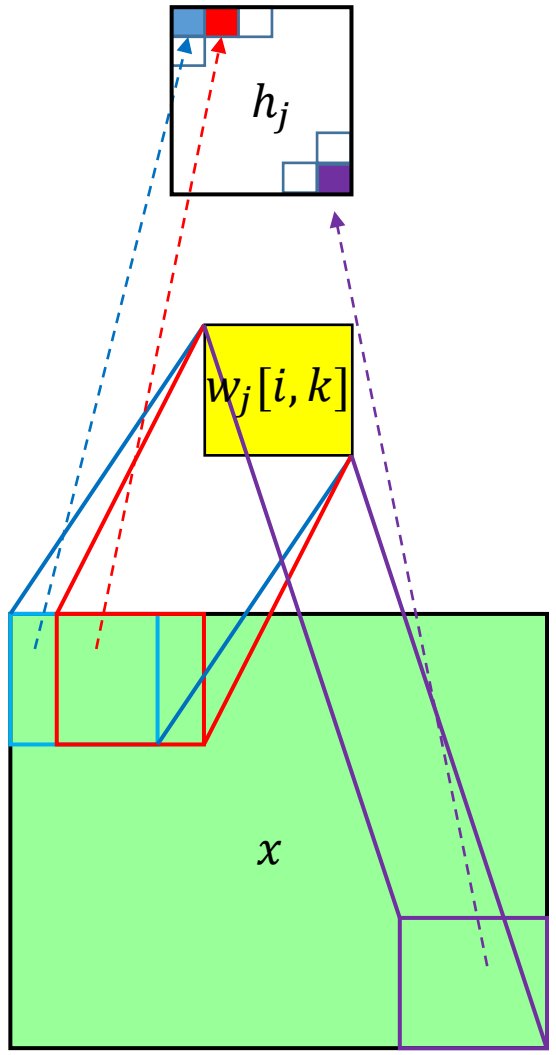
  - Feature Map (2-D)
    - *H* vectors
    - *I* elements



- Convolution Layer
  - 2-D convolution

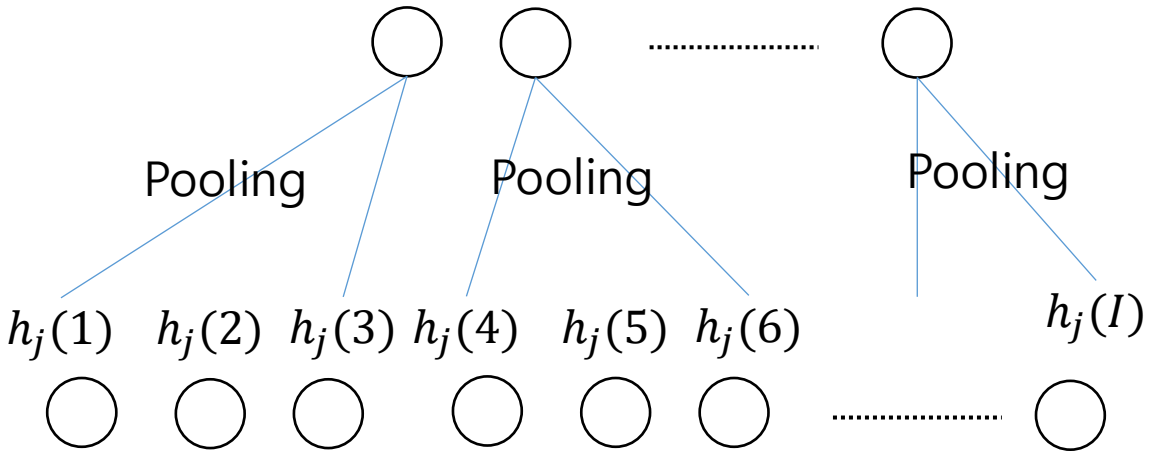
$$h_j[\mu,\nu] = \boldsymbol{w}_j * \boldsymbol{x} = \sum_{i=-(z-1)/2}^{(z-1)/2} \sum_{k=-(z-1)/2}^{(z-1)/2} w_j[i,k]x[\mu+i,\nu+k]$$

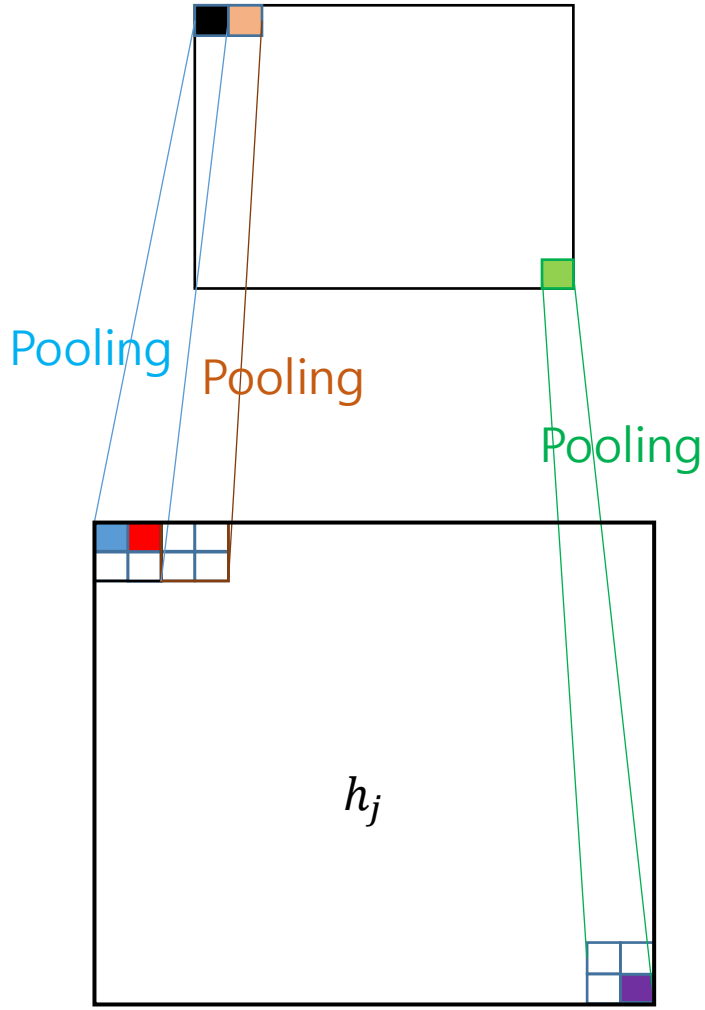
  - Feature Map (3-D)
  - Padding
  - Stride
  - Local Connectivity (receptive field of neuron)
  - Parameter Sharing (filter or kernel)



- Pooling Layer
  - A form of nonlinear down sampling
    - The exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.
  - Max pooling is the most common
    - It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.
  - Average Pooling, L2-norm Pooling

- Pooling Layer





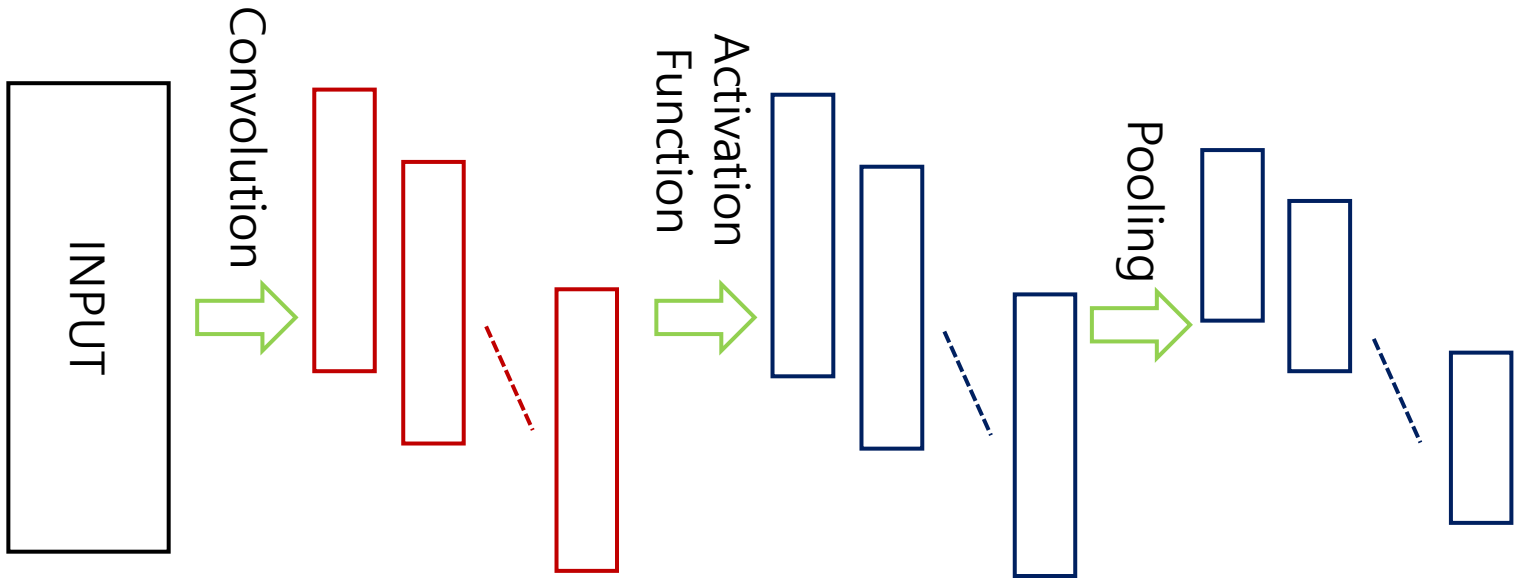
- ReLU Layer
  - ReLU(Rectified Linear Unit) for feature maps (before pooling)

$$f(x) = \max(0, x)$$

  - Other functions are also used to increase nonlinearity

$$f(x) = \tanh(x)$$

$$f(x) = (1 + e^{-x})^{-1}$$



- Fully Connected Layer
  - After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks.

  - SoftMax Output Node

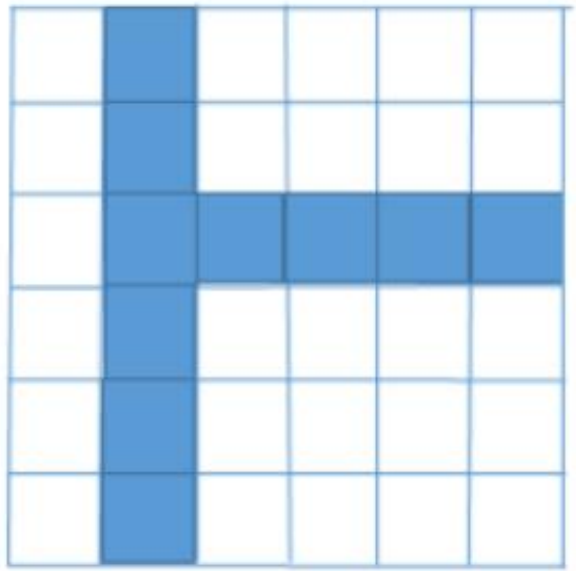
$$y_k = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

  - CE(Cross-Entropy) Loss Function

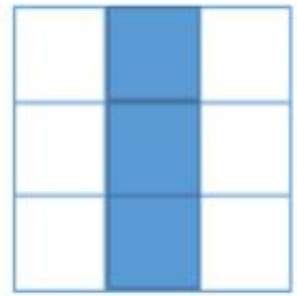
$$E_{CE} = -\sum_k t_k \log(y_k)$$

예제 9.3-1

아래 그림으로 2차원 공간 상에 6×6 크기의 입력과 3×3 크기의 수직성분 추출 필터가 주어졌다. 패딩은 영의 값을 사용하여, 이동 폭은 1로 한 경우 컨볼루션 층의 결과를 구하라. 그 다음에 ReLU 함수를 거친 후 2×2 맥스풀링을 한 결과를 단계적으로 보여라. 여기서, 입력의 픽셀 값과 필터 값은 0(밝은 부분)과 1(어두운 부분)으로 정하였다.



[6×6 입력]



[3×3필터]

풀이

6×6 크기의 입력에 패딩을 적용 후 3×3 크기의 필터를 적용한 컨볼루션 층 출력은 6×6 으로

0	2	0	0	0	0
0	3	1	1	1	1
0	3	1	1	1	1
0	3	1	1	1	1
0	3	0	0	0	0
0	2	0	0	0	0

와 같이 된다. 컨볼루션 결과에 음의 값이 없으므로 ReLU층을 거쳐도 동일한 결과를 얻게 된다. 여기에 2×2 맥스풀링을 거치면 그 결과는

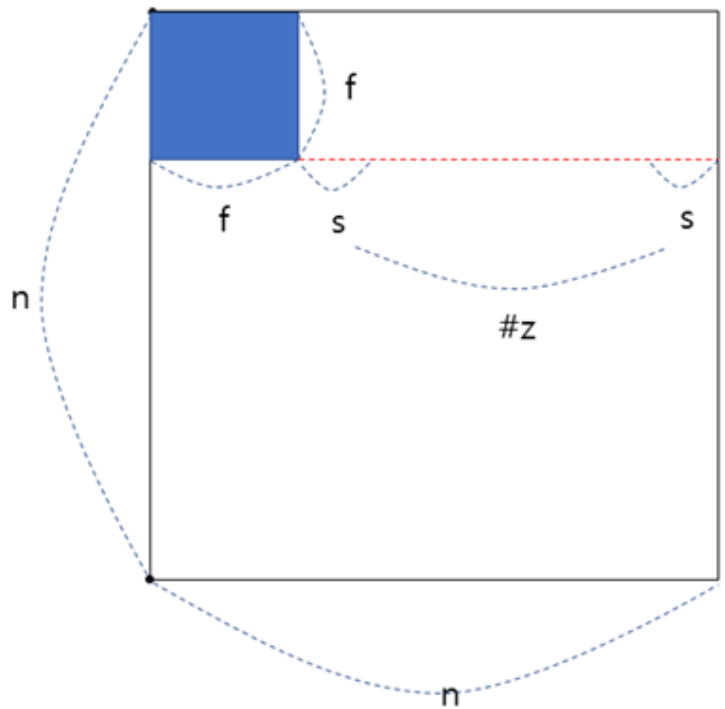
3	1	1
3	1	1
3	0	0

이 된다.

예제 9-3-2. $n \times n$ 크기의 입력에 패딩 p와 이동폭(stride) s를 사용하여 $f \times f$ 크기의 필터를 적용하였다. 필터 적용결과 얻게 되는 특징 맵의 크기를 구하라.

풀이) 먼저 패딩을 적용하지 않고 $n \times n$ 입력에 이동폭 s를 사용하여 $f \times f$ 필터를 $z+1$ 번 적용한 경우 다음 그림과 같이 $n = f + s \times z$이 된다. 즉, 특징 맵 크기는 $(\frac{n-f}{s}+1) \times (\frac{n-f}{s}+1)$가 된다.

이제, 패딩 p를 사용하면, 입력의 크기가 $(n+2p) \times (n+2p)$이 되고, 특징 맵 크기는 $(\frac{n+2p-f}{s}+1) \times (\frac{n+2p-f}{s}+1)$ 이 된다.



9.4. Hyperparameters and Regularization

- **Choosing Hyperparameters**

  CNNs use more hyperparameters than a standard multilayer perceptron (MLP). While the usual rules for learning rates and regularization constants still apply, the following should be kept in mind when optimizing.

  - **Number of filters**

  ✓ Since feature map size decreases with depth, layers near the input layer will tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the product of feature values v_a with pixel position is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next.

  - **Filter shape**

  ✓ Common filter shapes found in the literature vary greatly, and are usually chosen based on the dataset. The challenge is, thus, to find the right level of granularity so as to create abstractions at the proper scale, given a particular dataset, and without overfitting.

  - **Max pooling shape**

  ✓ Typical values are 2×2. Very large input volumes may warrant 4×4 pooling in the lower layers. However, choosing larger shapes will dramatically reduce the dimension of the signal, and may result in excess information loss. Often, non-overlapping pooling windows perform best.

• Regularization Methods

• Dropout

✓ Because a fully connected layer occupies most of the parameters, it is prone to overfitting.

✓ One method to reduce overfitting is dropout. At each training stage, individual nodes are either "dropped out" of the net with probability *1-p* so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

• Stochastic pooling

✓ In stochastic pooling,[52] the conventional deterministic pooling operations are replaced with a stochastic procedure, where the activation within each pooling region is picked randomly according to a multinomial distribution, given by the activities within the pooling region. This approach is free of hyperparameters and can be combined with other regularization approaches, such as dropout and data augmentation.

• Number of parameters

✓ Another simple way to prevent overfitting is to limit the number of parameters, typically by limiting the number of hidden units in each layer or limiting network depth. For convolutional networks, the filter size also affects the number of parameters. Limiting the number of parameters restricts the predictive power of the network directly, reducing the complexity of the function that it can perform on the data, and thus limits the amount of overfitting.

• Weight decay

✓ A simple form of added regularizer is weight decay, which simply adds an additional error, proportional to the sum of weights (L1 norm) or squared magnitude (L2 norm) of the weight vector, to the error at each node. The level of acceptable model complexity can be reduced by increasing the proportionality constant, thus increasing the penalty for large weight vectors.

참조: 낙오(Dropout)의 수학적 분석

하나의 선형 노드를 가정--선형 노드에 입력되는 가중치 합에 의한 출력

$$o = \sum_{i=1}^{n} w_i I_i \tag{9.4.1}$$

I_i:노드의 i번째 입력, w_i: 노드 연결 가중치

각 가중치 w_i가 p_i의 확률로 선택되는 하위 회로망(sub-network)을 구성

모든 가능한 하위 회로망의 앙상블 출력

$$o_{ENS} = \sum_{i=1}^{n} p_i w_i I_i \tag{9.4.2}$$

입력노드 I_i가 $q_i = 1 - p_i$의 확률로 낙오: 낙오가 적용된 선형노드의 출력

$$o_D = \sum_{i=1}^{n} \delta_i w_i I_i \tag{9.4.3}$$

여기서 δ_i는 Bernoulli 확률변수로써

$$\delta_i = \begin{cases} 1 & \text{with probility } p_i \\ 0 & \text{with probility } 1-p_i \end{cases} \tag{9.4.4}$$

선형노드의 목표 값을 t: 오차함수는 각각 MSE로

$$E_{ENS} = \frac{1}{2}(t - o_{ENS})^2 = \frac{1}{2}(t - \sum_{i=1}^{n} p_i w_i I_i)^2 \tag{9.4.5}$$

와

$$E_D = \frac{1}{2}(t - o_D)^2 = \frac{1}{2}(t - \sum_{i=1}^{n} \delta_i w_i I_i)^2 \tag{9.4.6}$$

δ_i는 확률변수이므로 o_D와 E_D 역시 확률변수

E_{ENS}는 확률변수가 아님

학습은 오차함수의 기울기에 의해 결정

$$\frac{\partial E_{ENS}}{\partial w_i} = \frac{\partial E_{ENS}}{\partial o_{ENS}} \frac{\partial o_{ENS}}{\partial w_i} = -(t - o_{ENS}) p_i I_i \tag{9.4.7}$$

$$\frac{\partial E_D}{\partial w_i} = \frac{\partial E_D}{\partial o_D} \frac{\partial o_D}{\partial w_i} = -(t - o_D)\delta_i I_i = -t\delta_i I_i + w_i \delta_i^2 I_i^2 + \sum_{j \neq i} w_j \delta_i \delta_j I_i I_j \tag{9.4.8}$$

$$\frac{\partial E_D}{\partial w_i} = \frac{\partial E_D}{\partial o_D}\frac{\partial o_D}{\partial w_i} = -(t - o_D)\delta_i I_i = -t\delta_i I_i + w_i \delta_i^2 I_i^2 + \sum_{j \neq i} w_j \delta_i \delta_j I_i I_j \quad (9.4.8)$$

Bernoulli 확률변수 δ_i의 평균과 분산: 식 (9.4.4)에 의해 $E[\delta_i] = p_i$

$$Var[\delta_i] = E[(\delta_i - p_i)^2] = E[\delta_i^2] - p_i^2 = p_i - p_i^2 = p_i(1 - p_i) \quad (9.4.9)$$

식 (9.4.8)의 기대치는

$$E\left[\frac{\partial E_D}{\partial w_i}\right] = -tp_i I_i + w_i p_i I_i^2 + \sum_{j \neq i} w_j p_i p_j I_i I_j \quad (9.4.10)$$

여기에 식 (9.4.2)와 (9.4.9)를 적용하면

$$E\left[\frac{\partial E_D}{\partial w_i}\right] = -(t - o_{ENS})p_i I_i + w_i I_i^2 Var[\delta_i] \quad (9.4.11)$$

결국 식 (9.4.7)에 의해

$$E\left[\frac{\partial E_D}{\partial w_i}\right] = \frac{\partial E_{ENS}}{\partial w_i} + w_i I_i^2 Var[\delta_i] \quad (9.4.12)$$

낙오에 의한 오차함수의 기울기는 앙상블에 의한 오차함수의 기울기와 방향 동일

$$E\left[\frac{\partial E_D}{\partial w_i}\right] = \frac{\partial E_{ENS}}{\partial w_i} + w_i I_i^2 \, Var[\delta_i] \tag{9.4.12}$$

낙오에 의한 오차함수의 기울기는 앙상블에 의한 오차함수의 기울기와 방향 동일

식(9.4.12)는 앙상블 오차함수가 조정된(regularized) 형태

$$E_R = E_{ENS} + \frac{1}{2}\sum_{1=1}^{n} w_i^2 I_i^2 \, Var[\delta_i] \tag{9.4.13}$$

식 (9.4.13)의 마지막 항: 오차함수의 조정 항(regularization term)

- 가중치의 크기를 제한: 과도한 학습 방지
- 입력 크기의 제곱과 낙오 변수의 분산에 의해 조정 항의 크기 설정
- $p_i = 0.5$: $Var[\delta_i]$가 최대가 되어 낙오에 의한 조정항의 효과가 가장 큼
- $1 - p_i > 0.5$: 조정의 효과가 크지 않으면서 연결을 많이 끊게 됨
- 시험 샘플 입력: $E\left[\sum_{i=1}^{n} \delta_i w_i I_i\right] = \sum_{i=1}^{n} p_i w_i I_i$, 전방향 계산 시 w_i는 p_i를 곱함

예제 9-4-1. 선형노드의 출력이 $o = \sum_{i=1}^{n} w_i I_i$와 같이 주어졌다. 여기에 Gaussian 낙오를 적용한 노드의 출력은 $o_D = \sum_{i=1}^{n} \delta_i w_i I_i$로 나타내었으며, 여기서 δ_i는 평균이 1이고, 표준편차가 σ_i이다. 이 경우, 시험 패턴의 입력이 선형노드의 출력은 어떻게 계산되어야 하는지 구하여라.

풀이) $\delta_i \sim N(1, \sigma_i)$ 이므로 $E\left[\sum_{i=1}^{n} \delta_i w_i I_i\right] = \sum_{i=1}^{n} w_i I_i$ 이다. 따라서, Gaussian 낙오에서 시험패턴이 입력 시 선형노드의 출력은 가중치의 크기 조정 없이 $\sum_{i=1}^{n} w_i I_i$ 로 계산한다.

참조: Batch Normalization과 Layer Normalization

- l 층의 i번째 노드에 입력되는 가중치 합 $a_i^{(l)}$의 분포가 학습에 영향을 미치므로, 이 값이 적절한 분포를 지니도록 하는 과정이 배치 정규화[21].
- 초기 가중치 값의 분포에 학습이 받는 영향을 제거(가중치 초기화의 문제 해결), 학습속도 개선과 과도한 학습 방지 효과
- 배치 정규화는 각 계층의 노드별로 진행이 되는데, 미니 배치를 대상으로

$$\mu_i^{(l)} = E_{\boldsymbol{x}\sim P(\boldsymbol{x})}[a_i^{(l)}] \tag{9.4.14}$$

$$\sigma_i^{(l)} = \sqrt{E_{\boldsymbol{x}\sim P(\boldsymbol{x})}\left[a_i^{(l)} - \mu_i^{(l)}\right]^2} \tag{9.4.15}$$

$$\widehat{a_i^{(l)}} \leftarrow \frac{a_i^{(l)} - \mu_i^{(l)}}{\sqrt{\sigma_i^{(l)^2} + \epsilon}} \tag{9.4.16}$$

- 그 다음에 크기변화와 이동 실시

$$y_i^{(l)} \leftarrow \gamma \widehat{a_i^{(l)}} + \beta \tag{9.4.17}$$

여기서, γ와 β는 EBP 알고리즘으로 학습된다.

- 각 계층에서 모든 노드를 대상으로 평균과 분산을 구하여 정규화를 하는 것이 계층 정규화
- 평균과 표준편차: l 층의 가중치 합을 대상

$$\mu^{(l)} = \frac{1}{H}\sum_{i=1}^{H} a_i^{(l)} \tag{9.4.18}$$

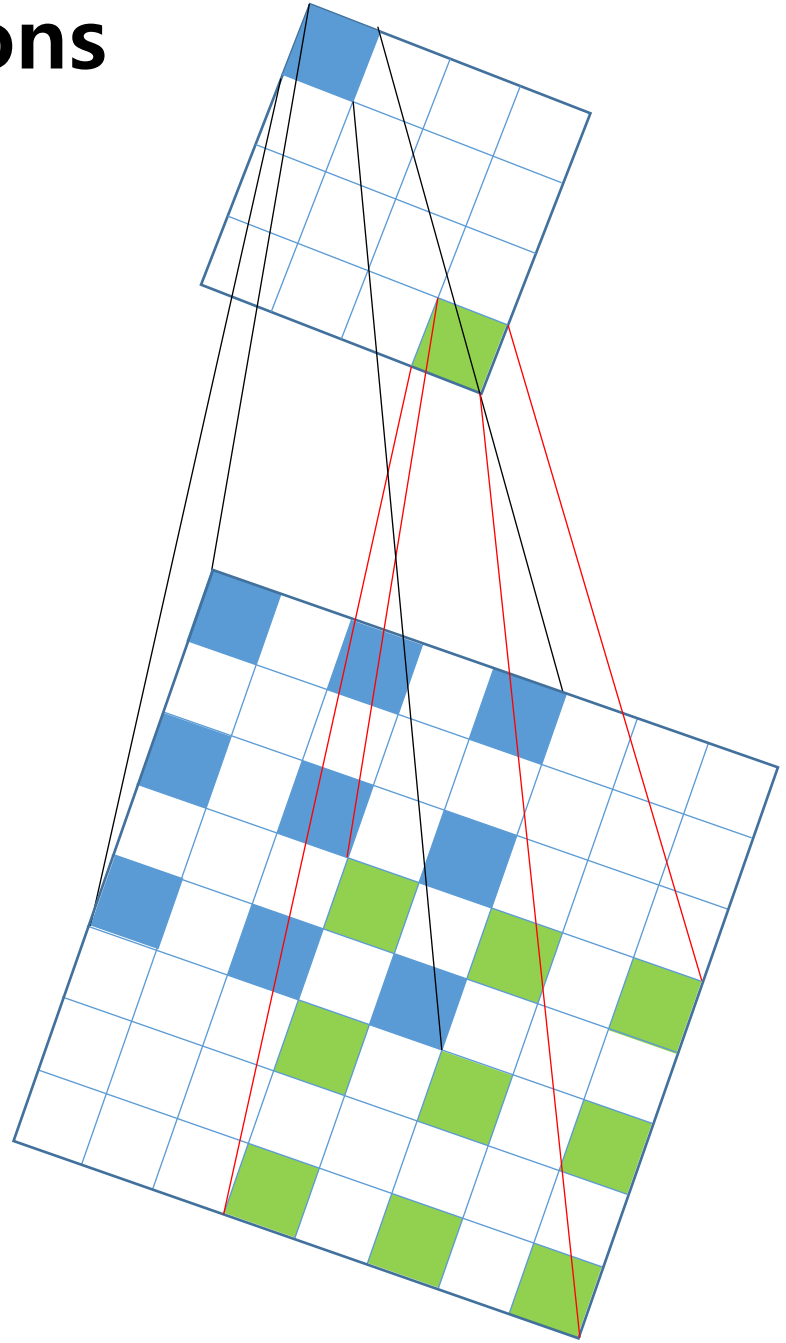
$$\sigma^{(l)} = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^{(l)} - \mu^{(l)}\right)^2} \tag{9.4.19}$$

여기서, H는 l 층의 은닉노드 수

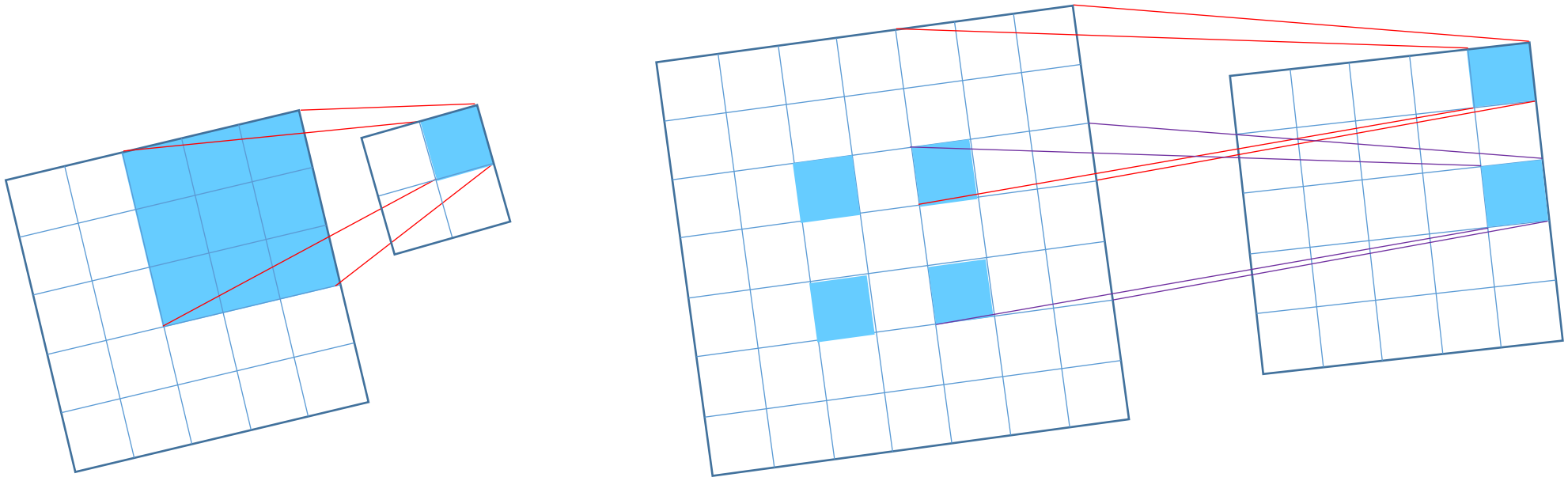
- 계층 정규화에서 l 층의 모든 노드는 같은 평균 $\mu^{(l)}$과 표준편차 $\sigma^{(l)}$를 사용하여 정규화가 이루어짐
- 계층 정규화는 미니배치 크기에 영향을 받지 않아 온라인 학습에 적용가능

9.5. Different Type of Convolutions

- **Dilated Convolutions**
  - Dilated convolutions introduce another parameter to convolutional layers called the **dilation rate**. This defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters. Imagine taking a 5x5 kernel and deleting every second column and row.



- **Transposed Convolutions**
  - An actual deconvolution reverts the process of a convolution.
  - A transposed convolution is somewhat similar because it produces the same spatial resolution a hypothetical deconvolutional layer would. However, the actual mathematical operation that's being performed on the values is different. A transposed convolutional layer carries out a regular convolution but reverts its spatial transformation.
  - To achieve this, we need to perform some fancy padding on the input.



9.6. Applications

- Facial Expression Recognition
- EmotiW(Emotion Recognition in the Wild)
  - Sub-competition: SFEW(Static Facial Recognition in the Wild)

그림 9.16. SFWE 2.0 얼굴 표정 인식 용 이미지 예시

- EmotiW(Emotion Recognition in the Wild)
  - Sub-competition: SFEW(Static Facial Recognition in the Wild)

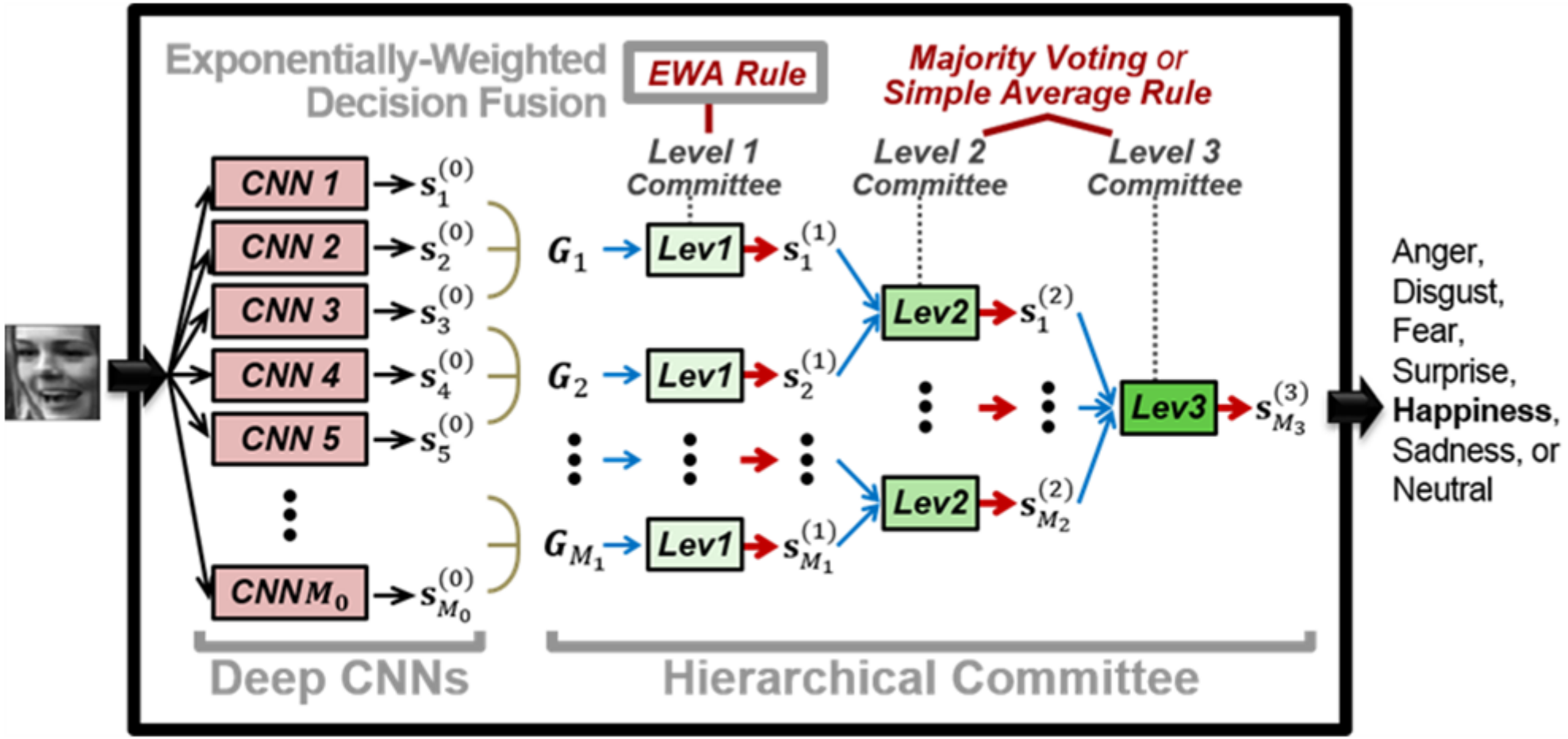


그림 9.17. CNSL팀이 제안한 얼굴 표정 인식을 위한 심층 CNN 위원회 구조

[16. Bo-Kyung Kim et al. 2016]

- EmotiW(Emotion Recognition in the Wild)
  - Sub-competition: SFEW(Static Facial Recognition in the Wild)

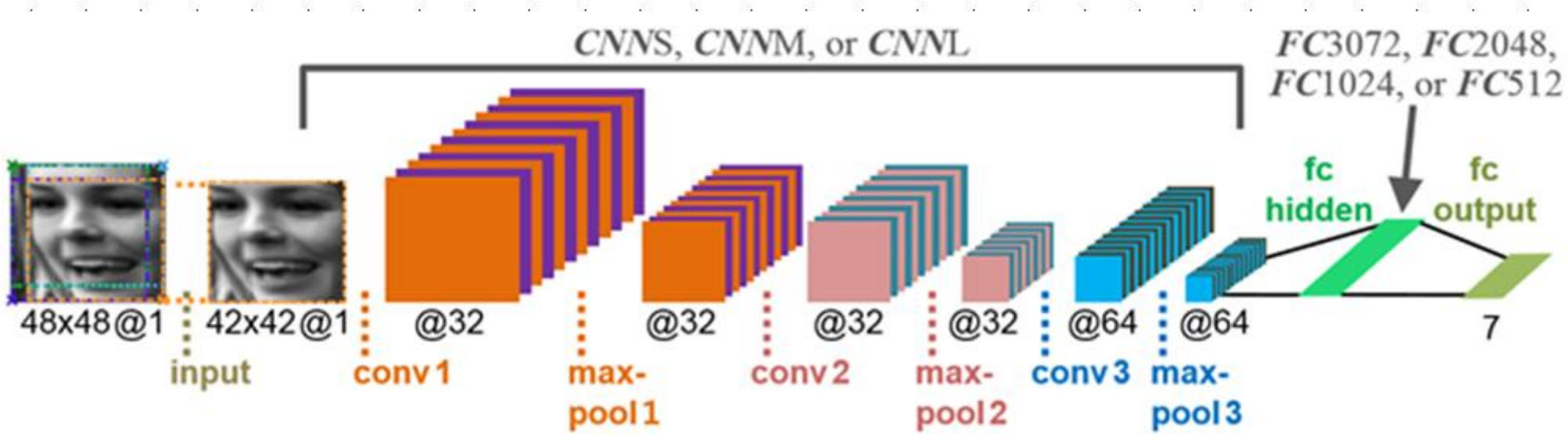


그림 9.18. 얼굴 표정 인식을 위한 심층 CNN 구조 [16. Bo-Kyung Kim et al. 2016]

- EmotiW(Emotion Recognition in the Wild)
  - Sub-competition: SFEW(Static Facial Recognition in the Wild)

그림 9.19. 전처리 방법에 따른 이미지의 변화

- EmotiW(Emotion Recognition in the Wild)
  - Sub-competition: SFEW(Static Facial Recognition in the Wild)

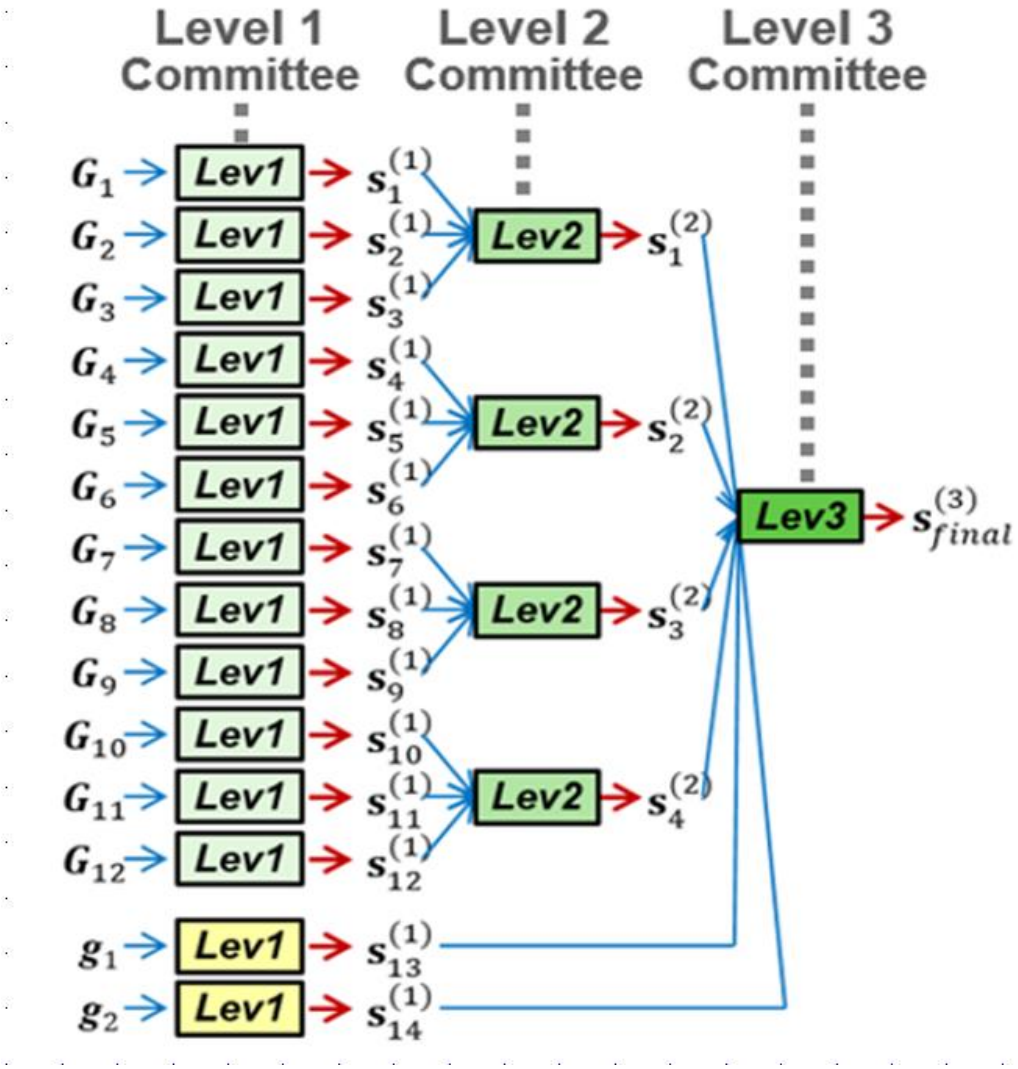


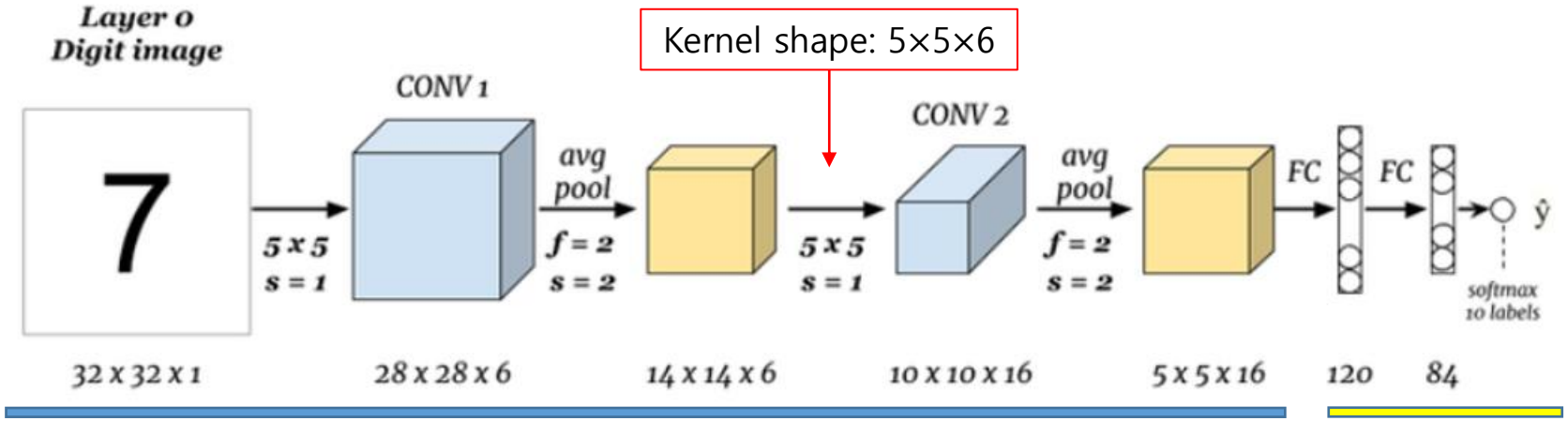
그림 9.20. 3개의 레벨로 구성된 위원회 구조도

[16. Bo-Kyung Kim et al. 2016]

• LeNet-5: MNIST 필기체 인식

n: input size
f: filter size
p: padding
s: stride
N_c: number of filters (channels)

Convolution Results: $\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right] \times N_c$



1. 32x32 입력에 컨볼루션층과 풀링층을 적용
2. 완전연결층을 거친 후 10개의 SoftMax 출력노드로 필기체 숫자 인식

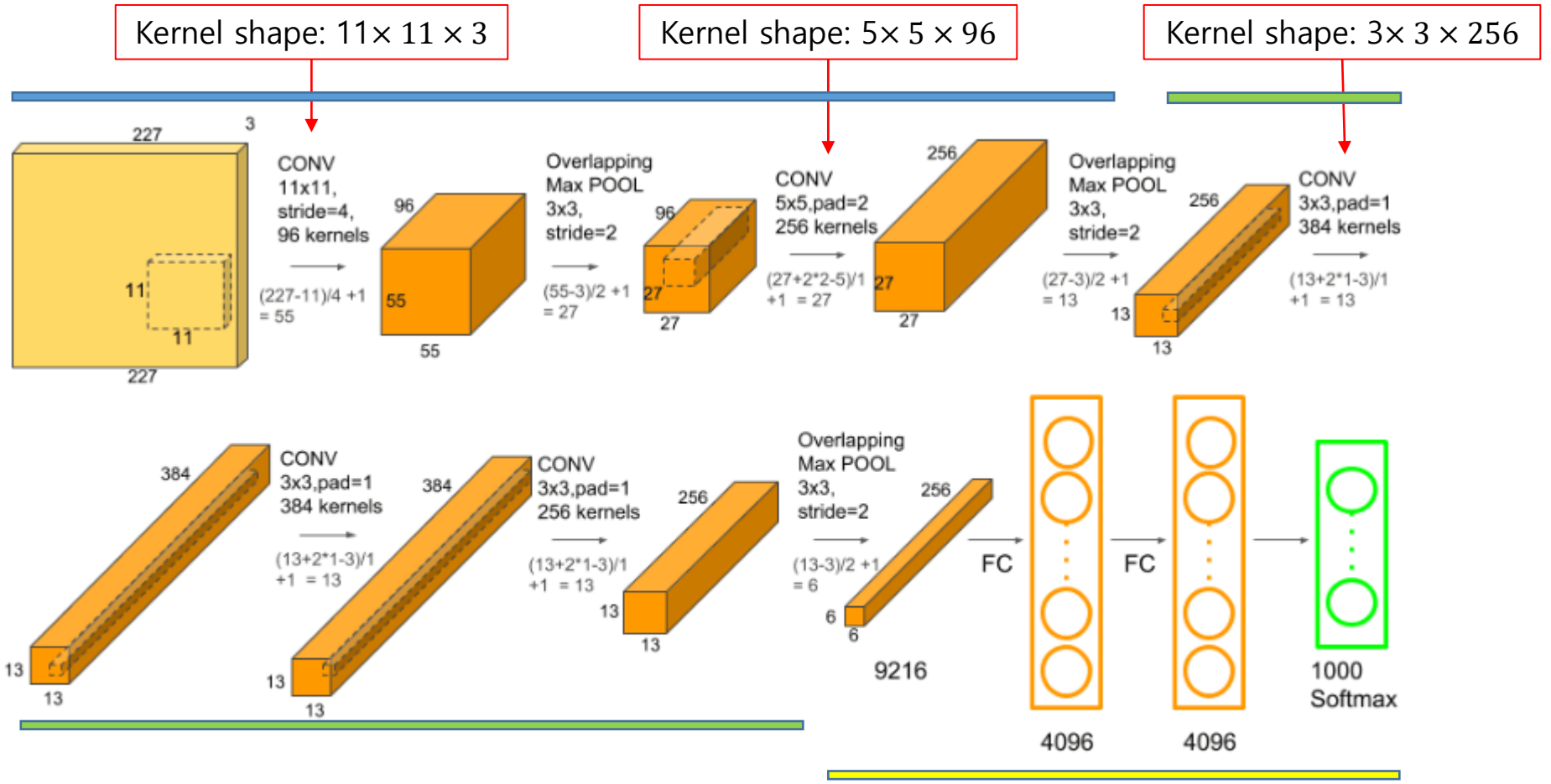
- AlextNet: ILSVRC2012(120만장의 영상을 1000개의 클래스로 구분)

n: input size
f: filter size
p: padding
s: stride
N_c: number of filters

$$\left[\frac{n+2p-f}{s}+1\right]\times\left[\frac{n+2p-f}{s}+1\right]\times N_c$$

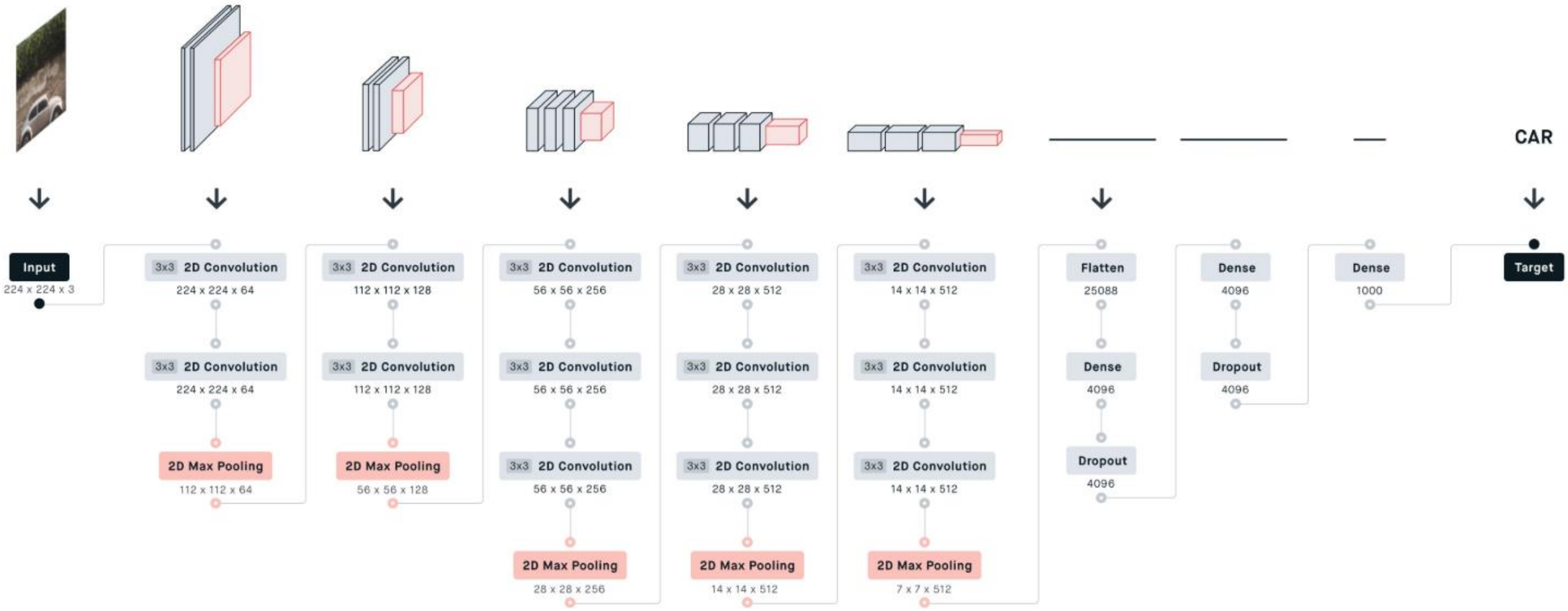
1. 227x227 RGB영상에 컨볼루션층과 풀링층 통과
2. 컨볼루션층 통과
3. 풀링층 거친 후 완전 연결층으로 결과 출력

ILSVRC2012
Top-5 error 17%



AlexNet network architecture

- VGG-16: ILSVRC2014



층수를 늘리고 3x3 컨볼루션 필터 사용(s=1,p=1),

완전연결층: 25088-4096-4096-1000

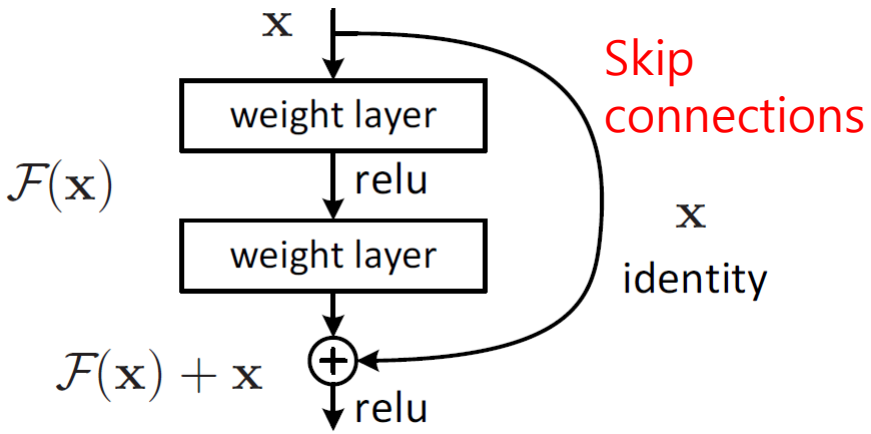
- ResNet: ILSVRC2015

  - 층수가 아주 많아지면 성능의 저하 문제 발생: 해결책으로 잔류학습 블록을 제시

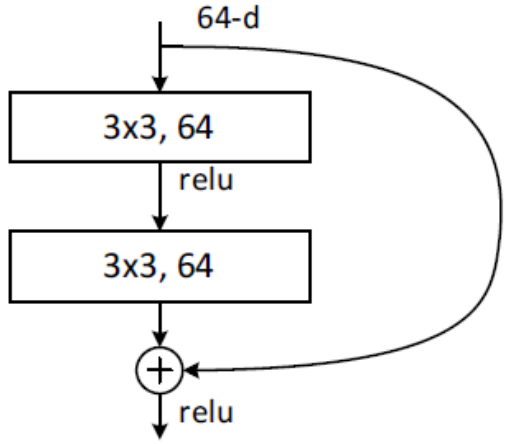
  - 가정: 원하는 사상 H(x)를 직접 학습하는 것 보다 F(x)=H(x)-x 를 학습하는 것이 쉽다.

⇒ 잔류학습 블록을 제시함.

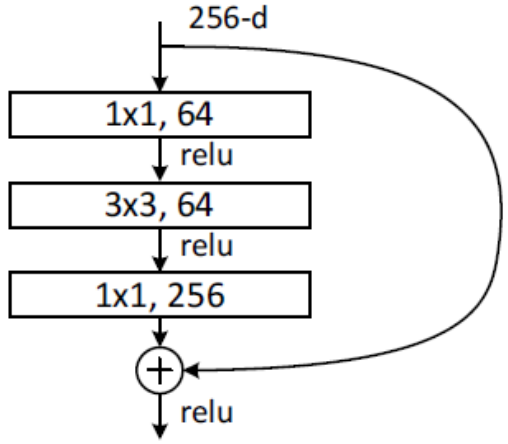
⇒ 신경회로망이 F(x)를 학습 후, 잔류학습 블록에서 F(x)+x를 출력함(ReLU 활성화 함수).



Residual Learning Block



Residual Learning Block for ReseNet-34

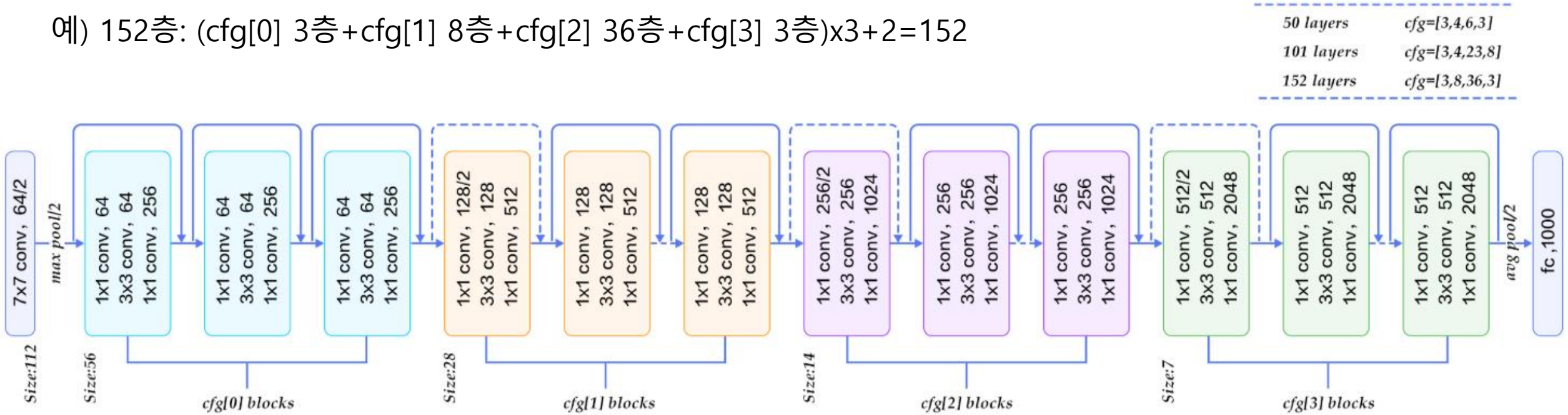


Residual Learning Block for ReseNet-50/101/152

• ResNet: ILSVRC2015

- 구조: 영상입력-7x7 컨볼루션층-cfg[0]블럭-cfg[1]블럭-cfg[2]블럭-cfg[3]블럭-완전연결 fc 1000
- 리스트 cfg는 아래 그림의 cfg[0], cfg[1], cfg[2], cfg[3] 블록을 주어진 수 만큼 지님을 나타냄
- 아래 그림은 cfg=[3,3,3,3]을 나타냄

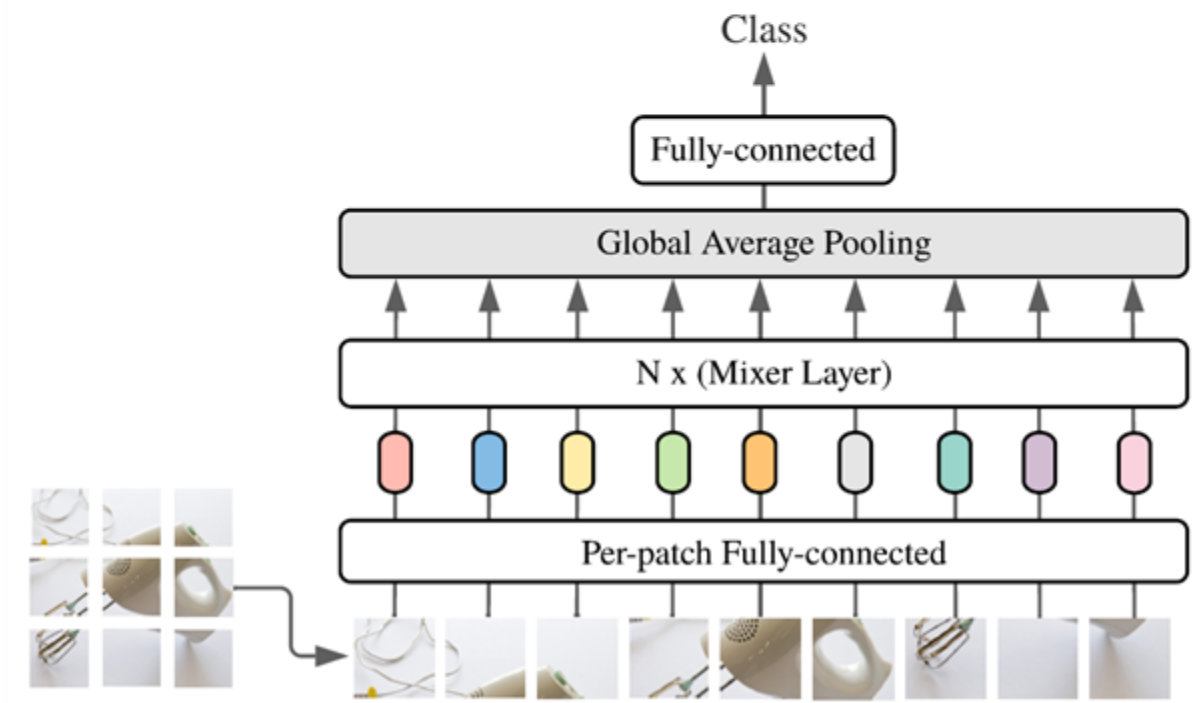
예) 152층: (cfg[0] 3층+cfg[1] 8층+cfg[2] 36층+cfg[3] 3층)x3+2=152



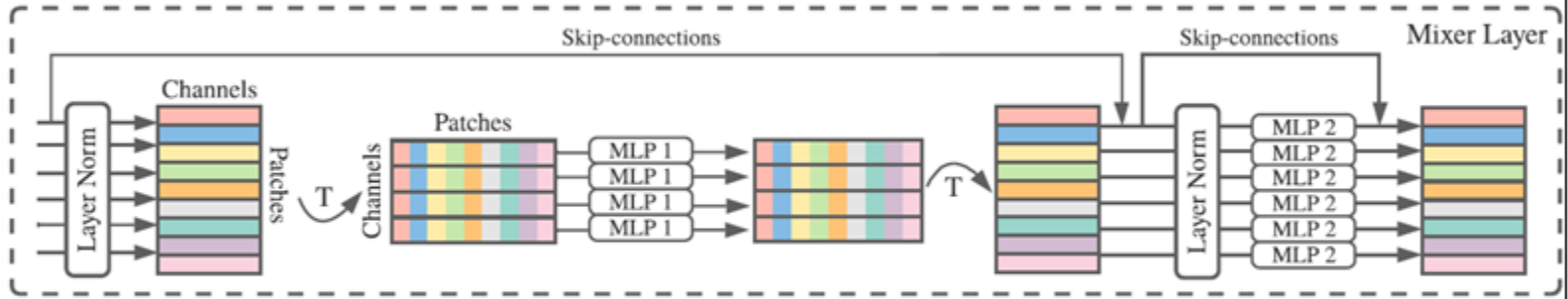
ILSVRC2015
ResNet-152: Top-5 error 4.49%

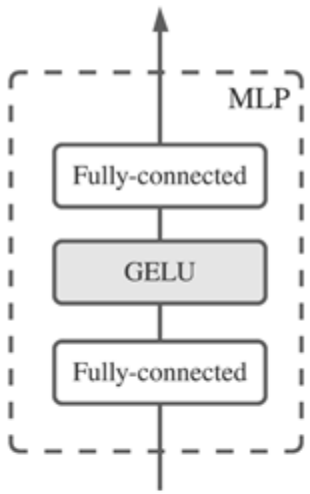
참조: MLP-Mixer[23]

- 단순한 다층퍼셉트론(MLP)를 기반으로 CNN과 같은 우수한 성능 보여줌
- 전체적인 구조와 동작: ① 입력 해상도 (H, W)인 입력 영상이 주어지면 이를 (P, P) 크기의 패치 $S(= HW/P^2)$개로 분리 ② 각 패치를 C차원 벡터-일명, 'token'-로 선형변환 하여 입력행렬 $\boldsymbol{X} \in R^{S \times C}$을 얻음(모든 패치는 같은 사상행렬로 선형변환) ③ $\boldsymbol{X}$를 MLP로 이루어진 Mixer 층에 입력하여 출력을 얻은 후(N번 통과), ④ 전역 평균 풀링 (Global Average Poolong)과 퍼셉트론 형태의 선형 분류기를 통하여 인식



- Mixer 층: MLP1 block과 MLP2 block으로 구성 (MLP1과 MLP2에 입력되기 전 계층 정규화와 행렬의 전치가 번갈아 처리되며, skip 연결도 있음)





- MLP(MLP1과 MLP2) 구성: "FC-GELU-FC"
- 입력행렬 $\boldsymbol{X} \in R^{S \times C}$가 Mixer 층에서 처리되는 과정
① 전치(입력행렬: $C \times S$ 형태), row($1 \times S$차원)이 MLP1에 입력되고 $1 \times S$차원의 출력. "token-mixing MLP"
② C 개의 MLP1이 내어놓는 출력을 모으면 $C \times S$ 행렬
③ 전치($S \times C$ 행렬), row($1 \times C$차원)이 MLP2에 입력되고 $1 \times C$차원의 출력 "channel-mixing MLP"

- token-mixing MLP: 다른 공간상에 존재하는 token 간의 관계 처리
- channel-mixing MLP: 다른 channel 간의 관계 처리
- 이 두 가지 방식의 MLP가 맞물리면서 입력 행렬의 두 가지 차원에 대한 상호작용이 가능하게 한다.