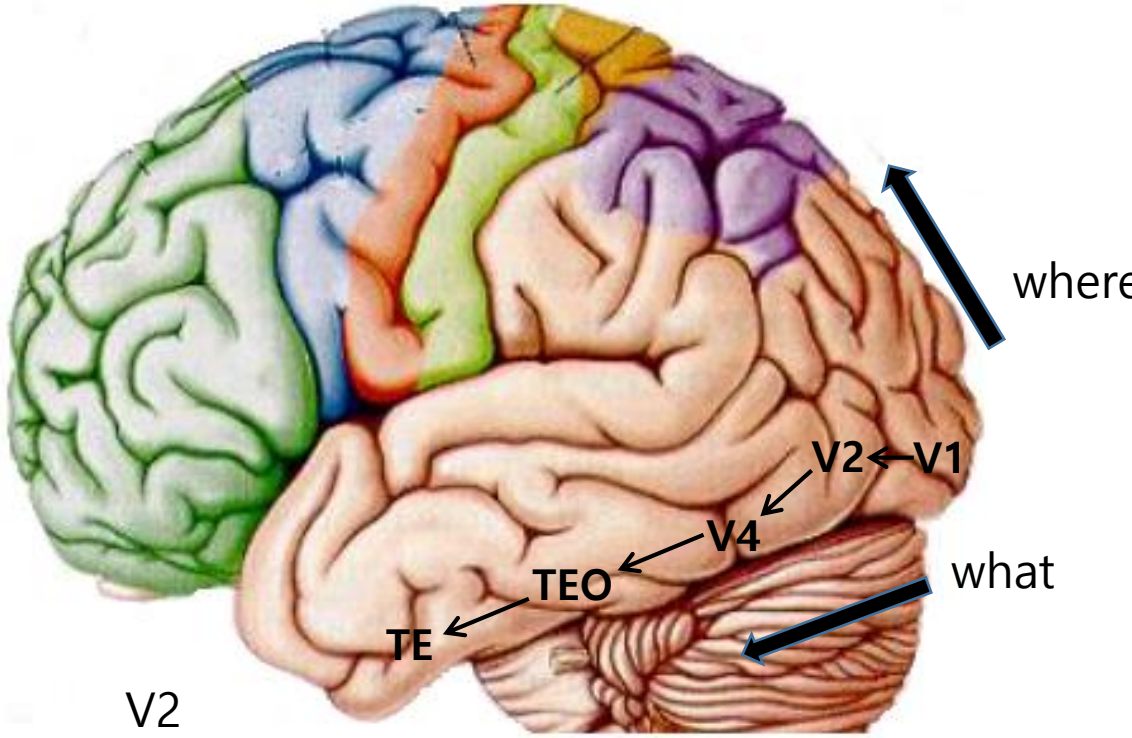
AI 실습

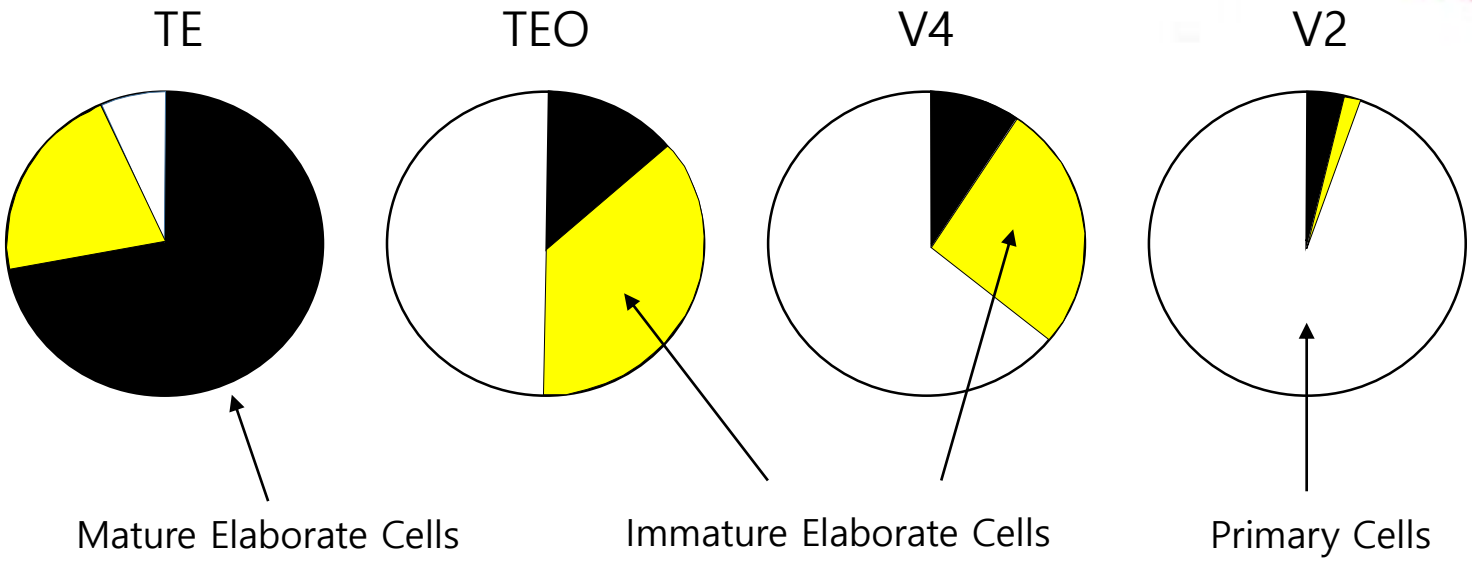
Contents

9.1. CNN (Convolutional Neural Network)의 역사적 배경과 특징

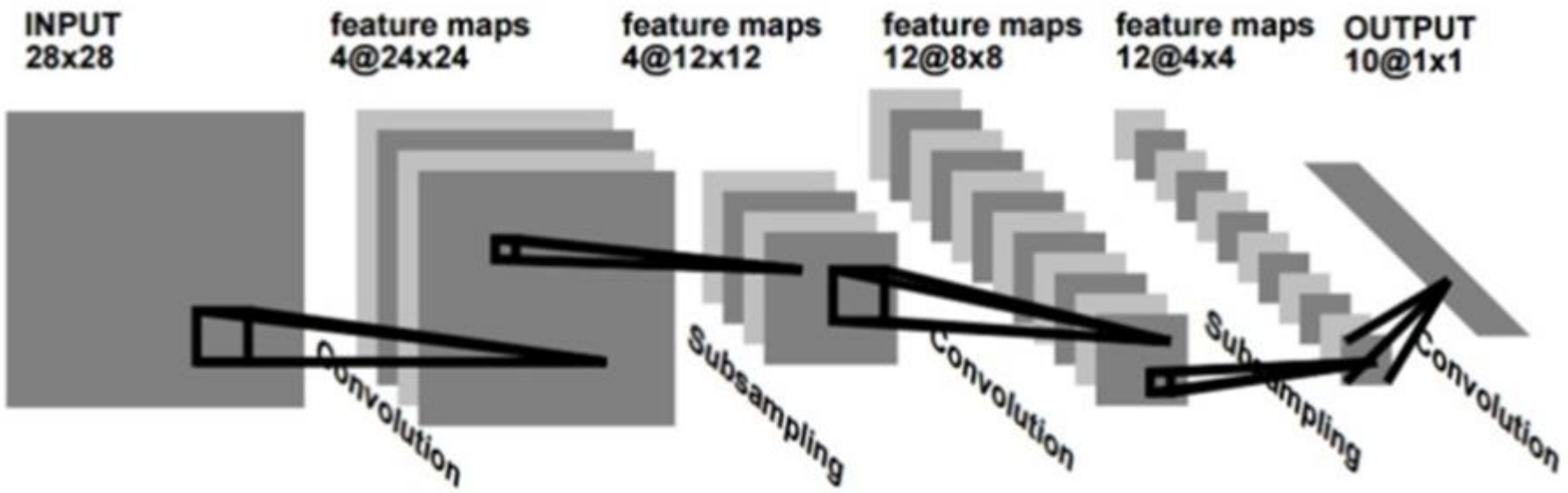
- 전방향 신경회로망/다층퍼셉트론 (Chap. 5)
  - 완전 연결(Fully-Connected)
  - 학습시킬 가중치가 아주 많음(Too many weights for training)
  - 학습속도가 느리고 계산량도 큼
  - 학습데이터에 대한 과도한 학습으로 일반화 성능 저하
  - ➢ 국소연결 망(Locally-Connected Networks) (포유류의 시각경로)
- 시각경로(Visual Pathway)
  - Hubel and Wiesel (1950s and 1960s)
    - Cat and monkey visual cortexes contain neurons that individually respond to small regions of the visual field. → Receptive Field(감수영역)
    - 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.

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





- Neural Networks
  - LeCun (1990s)
    - LeNet1



9.3. CNN(Convolutional Neural Network)

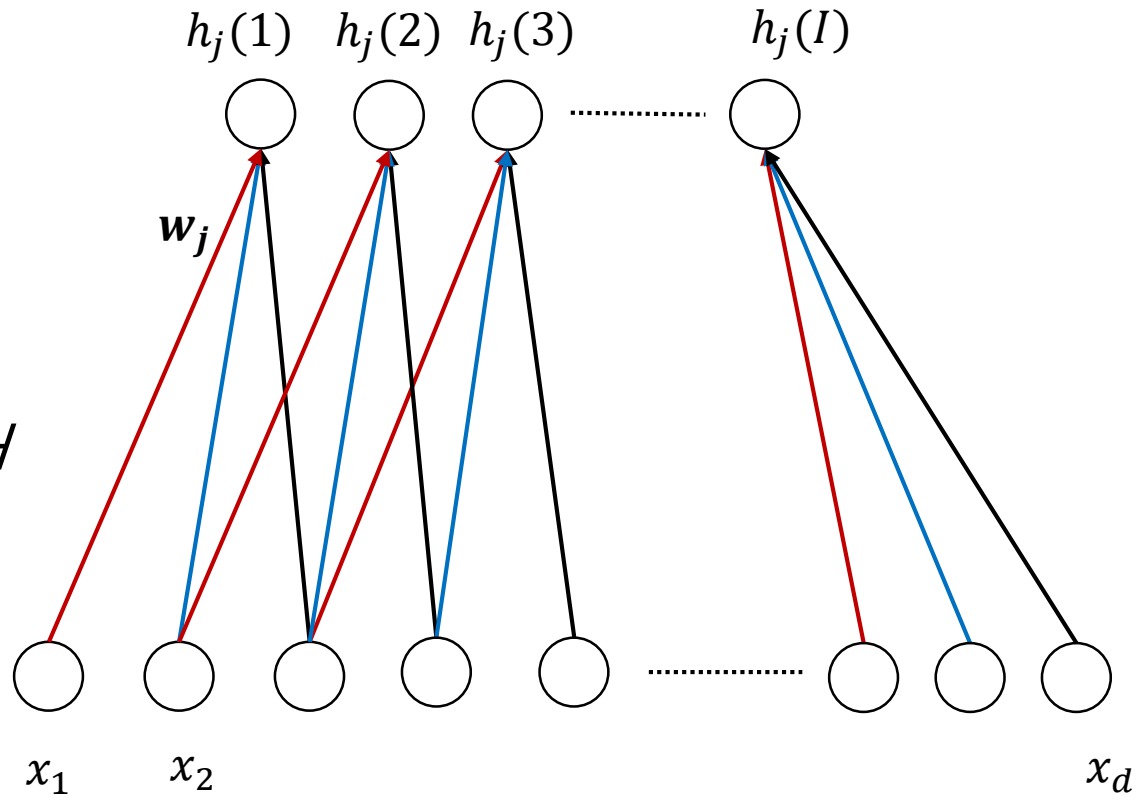
- CNN
  - 구성: Convolution layer, Pooling Layer, and Fully-Connected Layer

- 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*

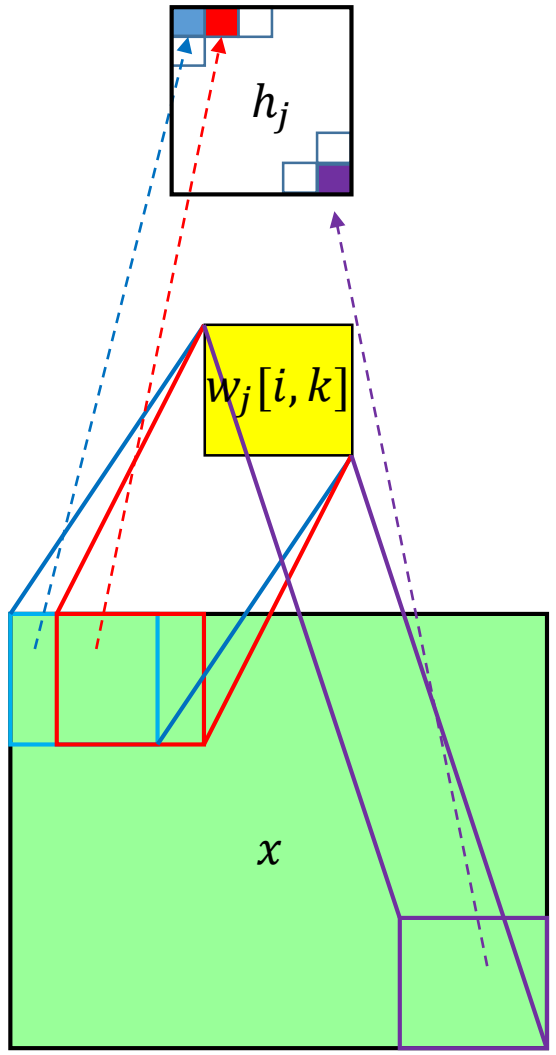
  - Feature Map (2-D)
    - *H* vectors
    - *I* elements



- Convolution Layer
  - 2-D convolution

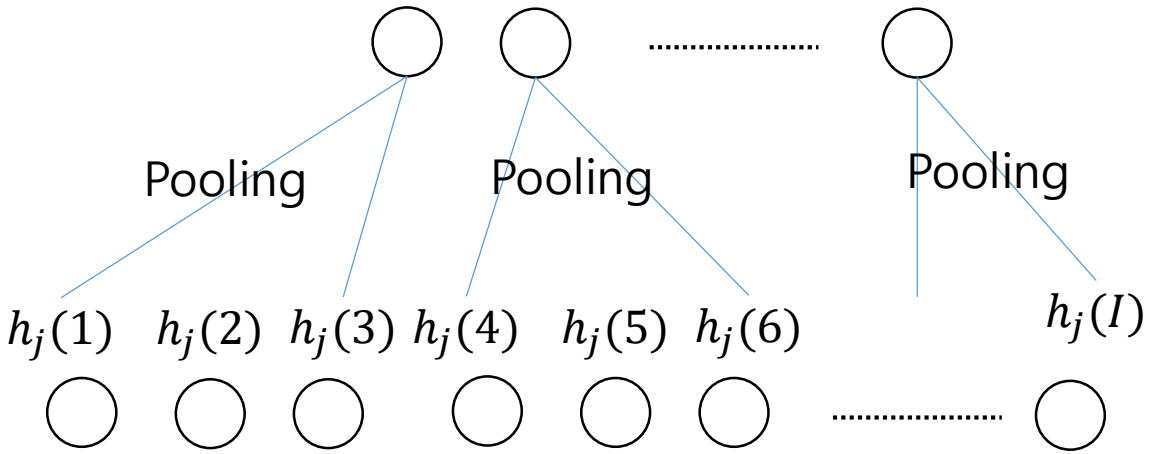
$$h_j[\mu,\nu] = \boldsymbol{w}_j * \boldsymbol{x} = \sum_{i=-(z-1)/2}^{(z-1)/2} \sum_{k=-(z-1)/2}^{(z-1)/2} w_j[i,k]\, x[\mu+i,\nu+k]$$

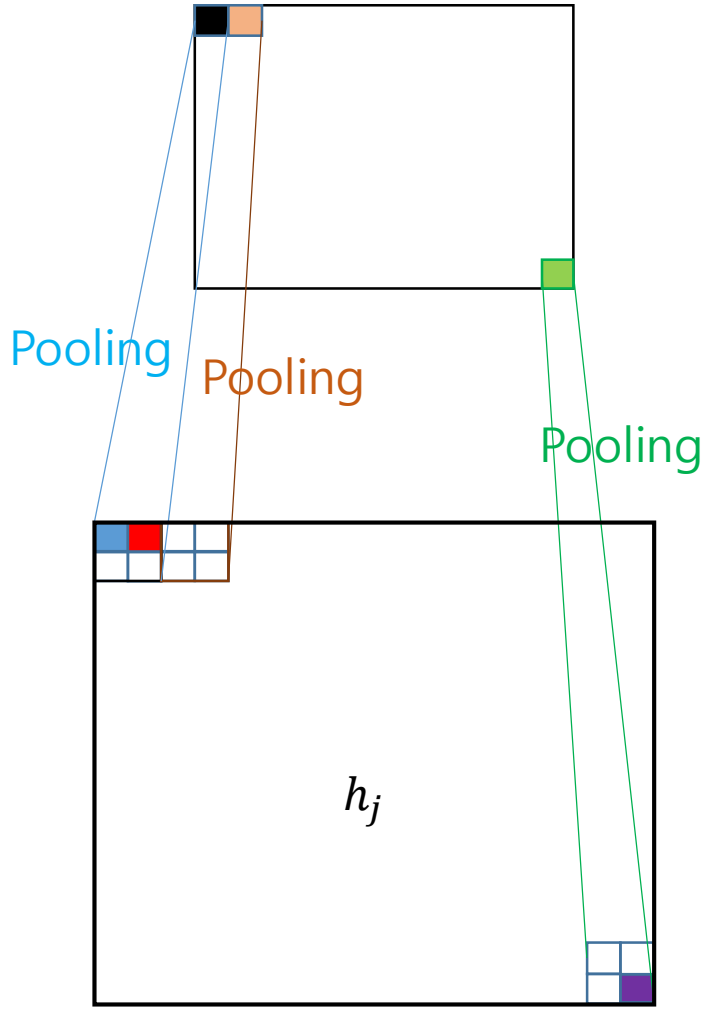
  - Feature Map (3-D)
  - Padding
  - Stride
  - Local Connectivity (receptive field of neuron)
  - Parameter Sharing (filter or kernel)



- 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





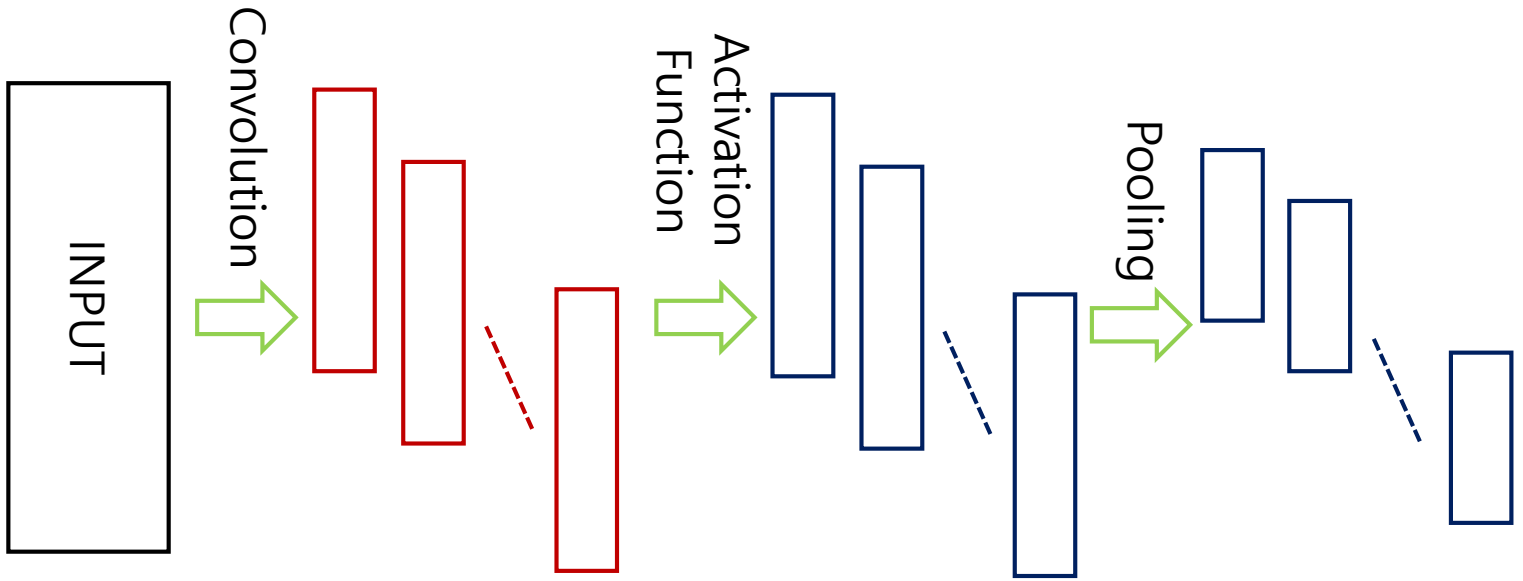
- ReLU Layer
  - ReLU(Rectified Linear Unit) for feature maps (before pooling)

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

  - Other functions are also used to increase nonlinearity

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

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



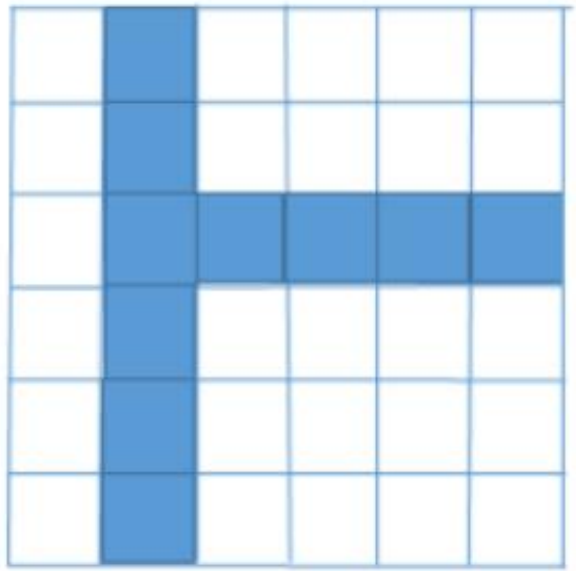
- Fully Connected Layer
  - After several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial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

  - SoftMax Output Node $$y_k = \frac{e^{\hat{y}_k}}{\sum_j e^{\hat{y}_j}}$$

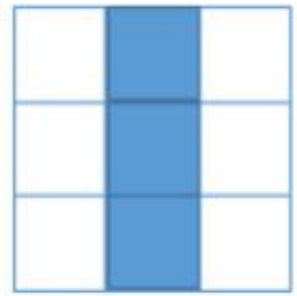
  - CE(Cross-Entropy) Loss Function $$E_{CE} = -\sum_k t_k \log(y_k)$$

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]

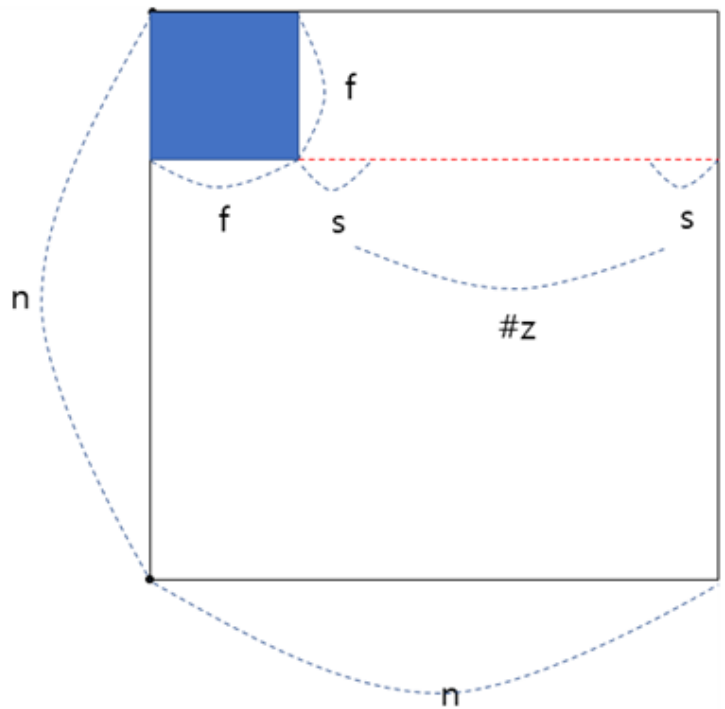


[3×3필터]

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. 파라미터 선정과 학습기법

- **파라미터 선정**
  - **필터의 개수**

✓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.

  - **필터 형태**

✓Common filter shapes found in the literature vary greatly, and are usually chosen based on the dataset.

  - **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.

• 과도한 학습 방지 기법

- **낙오(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.

- **파라미터 수 제한**

✓ 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.

- **가중치 감쇄(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

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

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

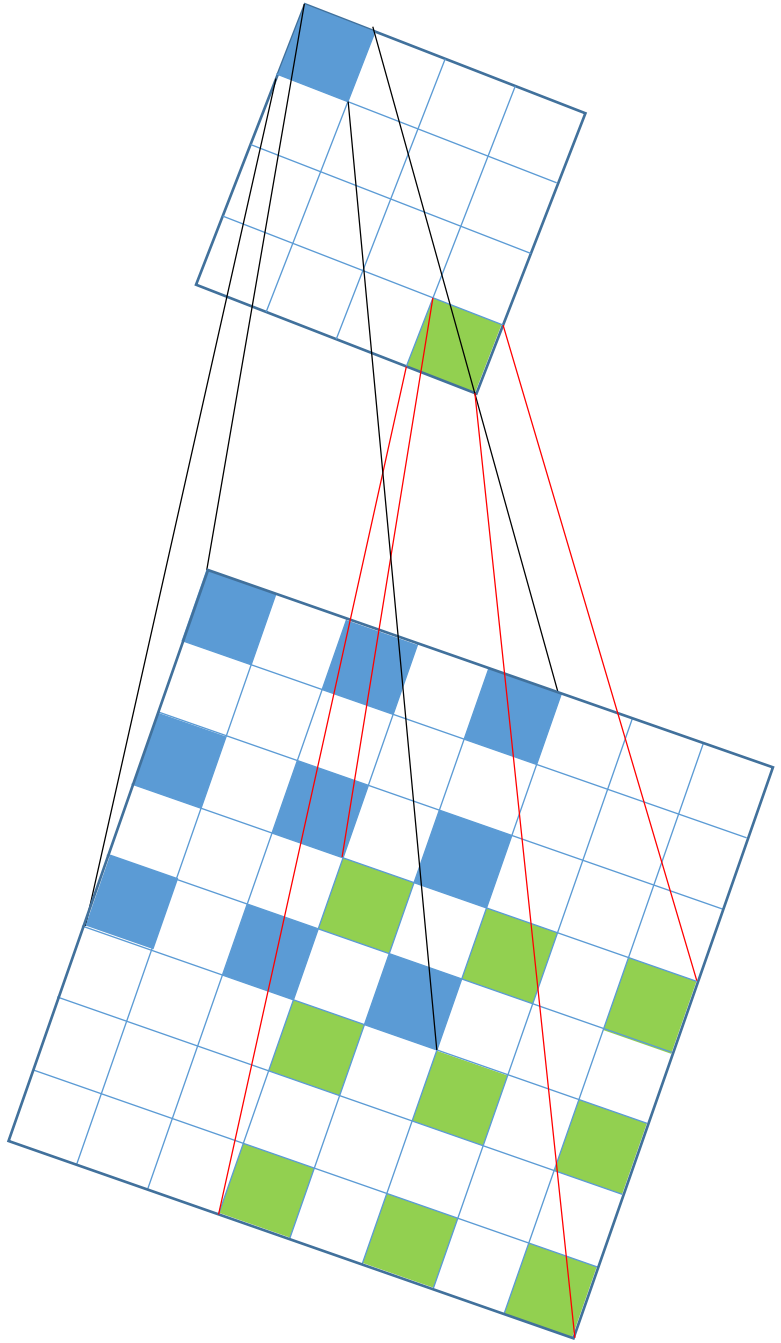
$$\sigma^{(l)} = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^{(l)} - \mu^{(l)}\right)^2} \tag{9.4.19}$$

여기서, H는 l 층의 은닉노드 수

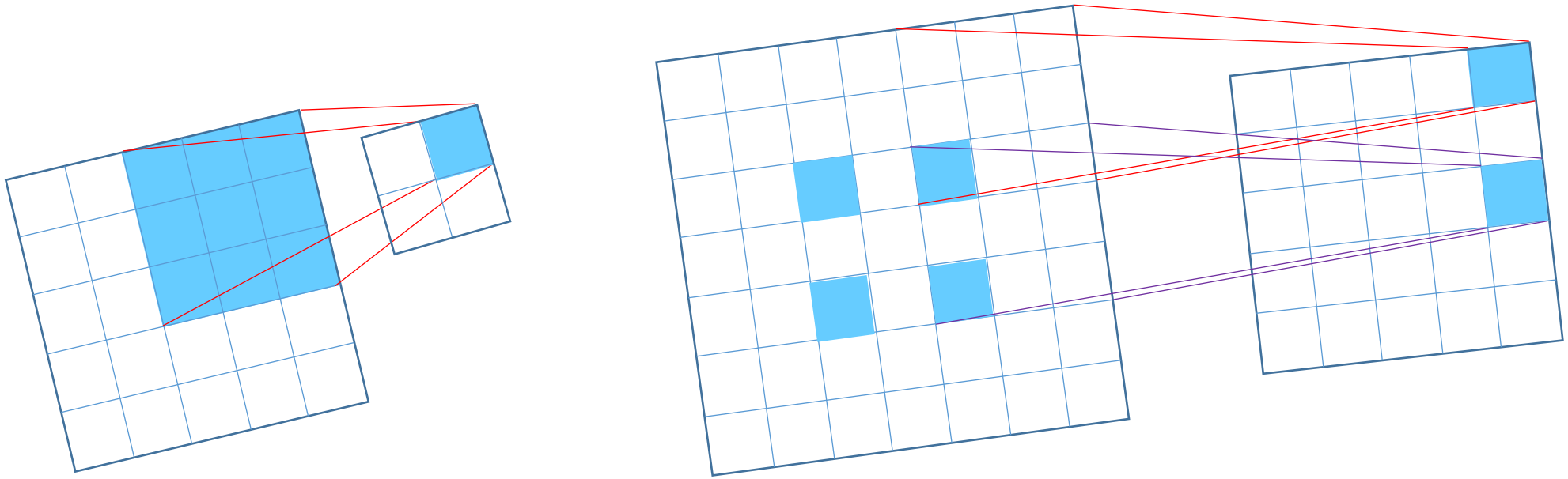
- 계층 정규화에서 l 층의 모든 노드는 같은 평균 $\mu^{(l)}$과 표준편차 $\sigma^{(l)}$를 사용하여 정규화가 이루어짐
- 계층 정규화는 미니배치 크기에 영향을 받지 않아 온라인 학습에 적용가능

9.5. 다양한 컨볼루션

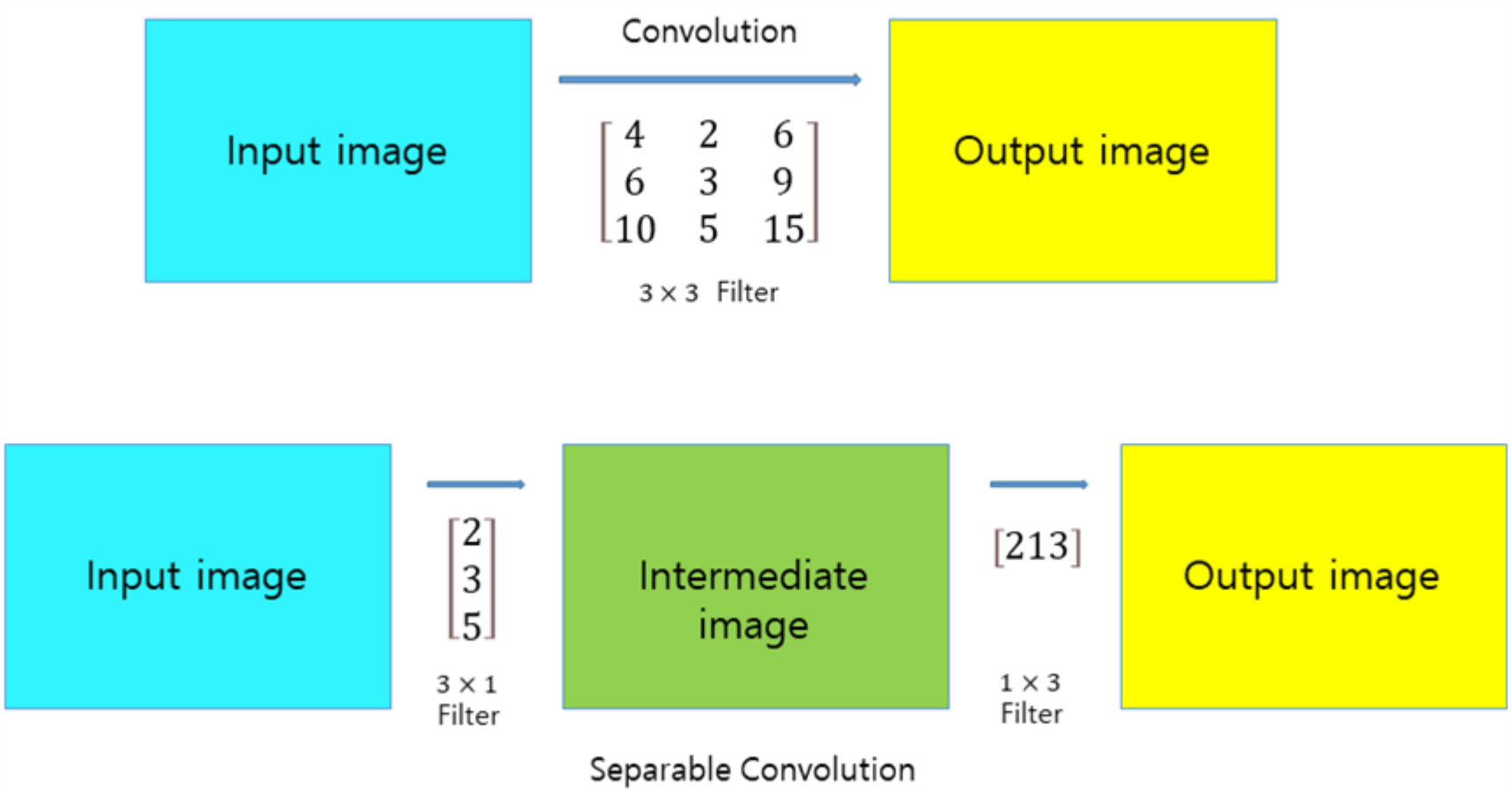
- **Dilated Convolutions**
  - Dilated convolutions introduce another parameter to convolutional layers called the **dilation rate**. This defines a spacing between the values in a kernel. A 3x3 kernel with a dilation rate of 2 will have the same field of view as a 5x5 kernel, while only using 9 parameters.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



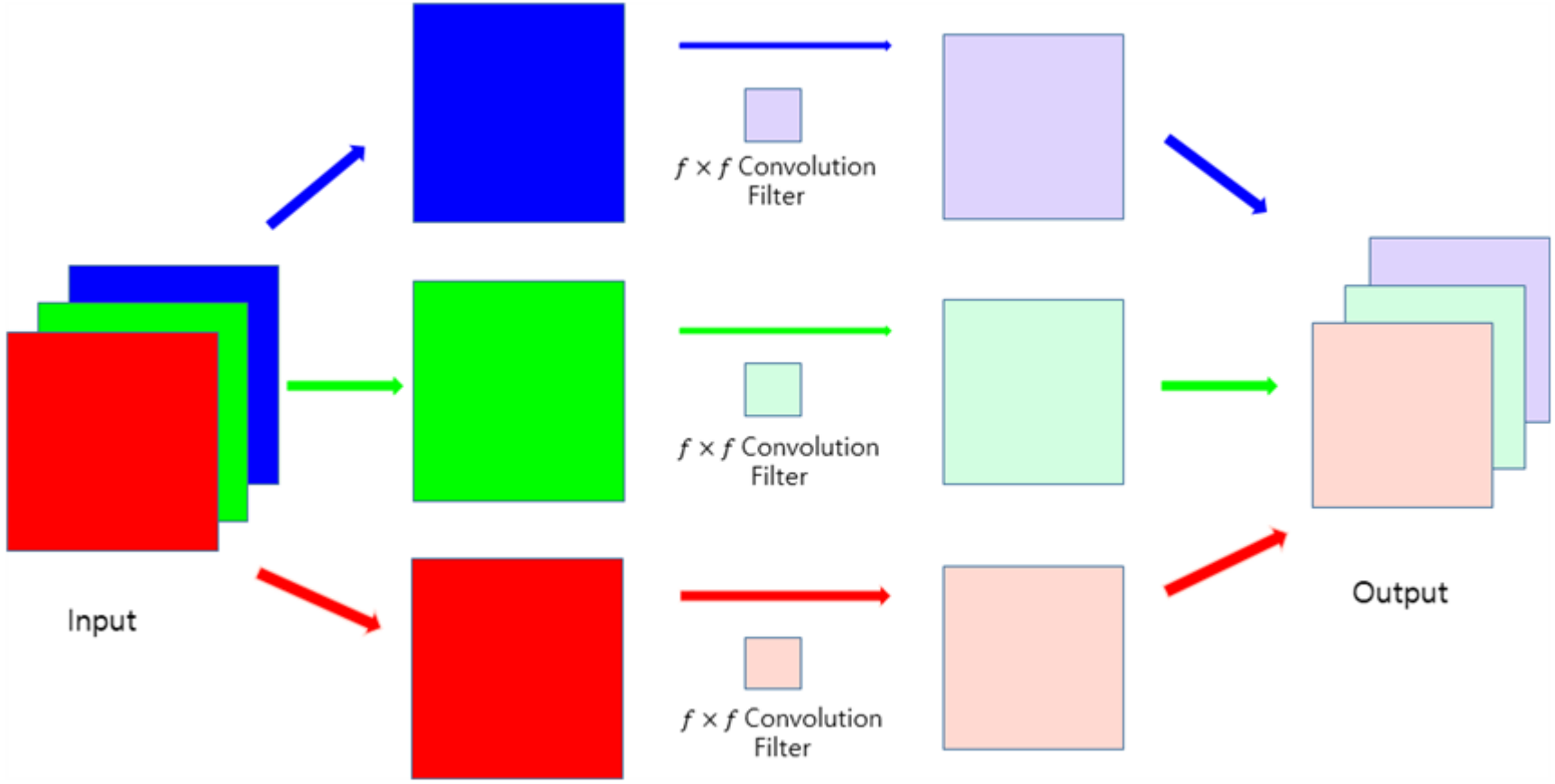
- **Transposed Convolutions**
  - An actual deconvolution reverts the process of a convolution.
  - A transposed convolution is somewhat similar because it produces the same spatial resolution a hypothetical deconvolutional layer would. However, the actual mathematical operation that's being performed 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.



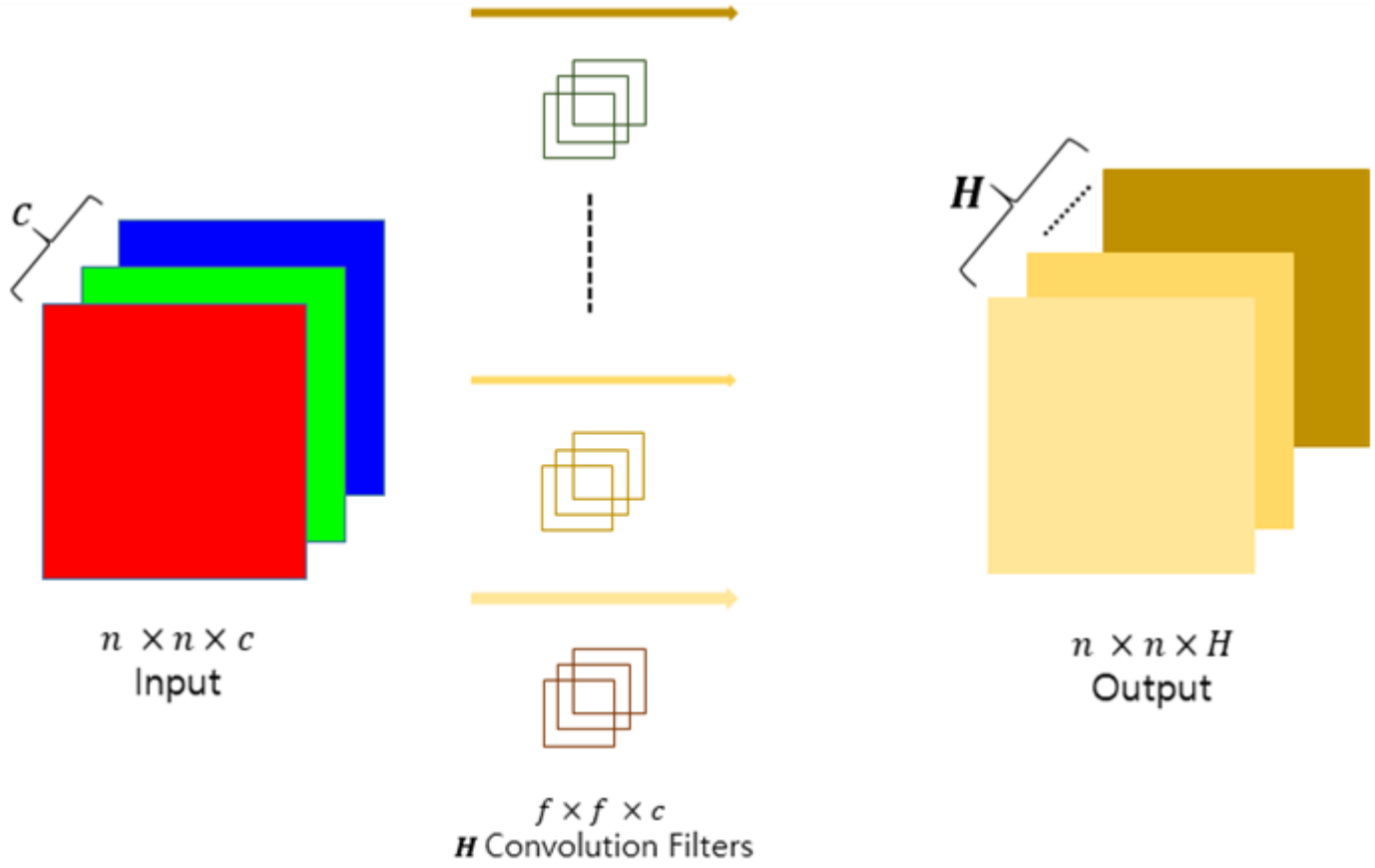
- **Separable Convolutions**
  - To save computational cost by substituting $f \times f$ convolution filters with the combination $f \times 1$ and $1 \times f$ convolution filters.



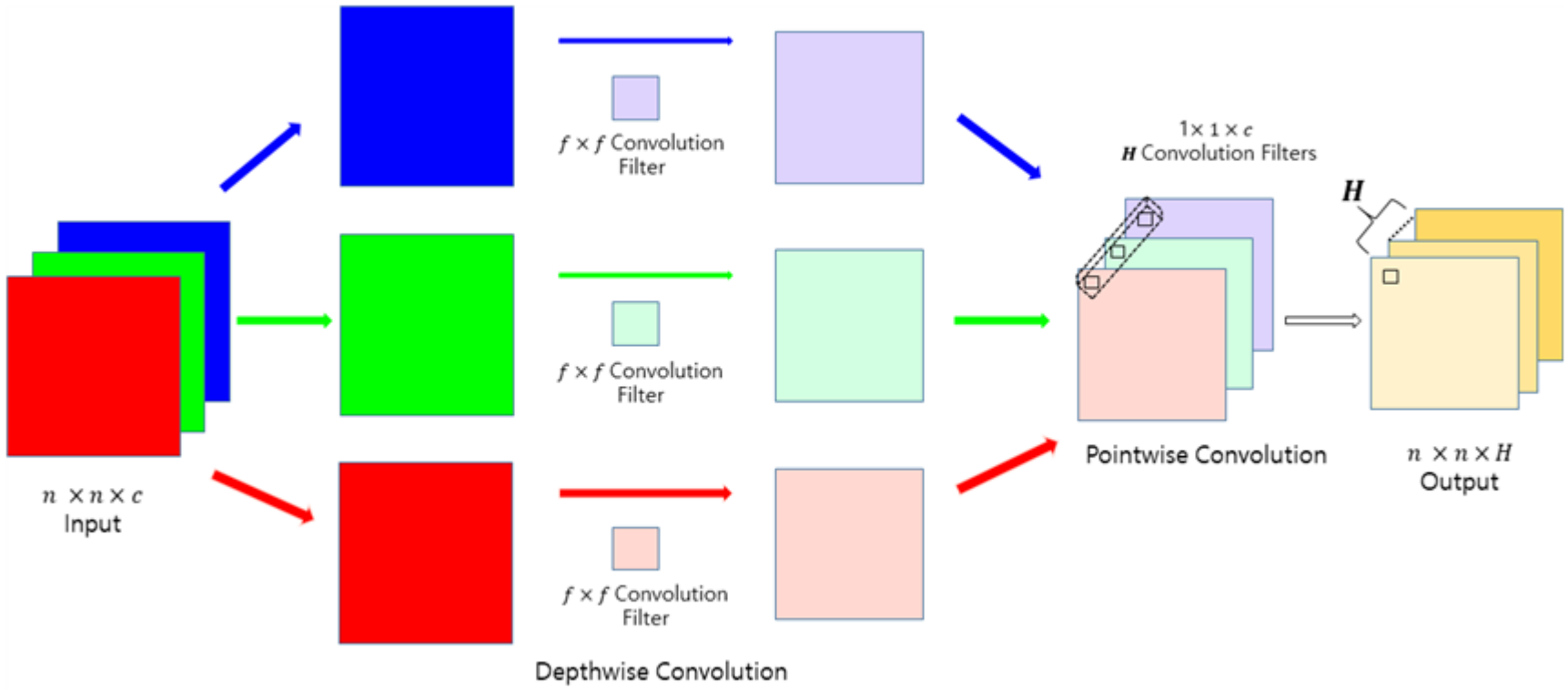
- **Depthwise Convolutions**
  - Assuming $n \times n \times c$ sized input with c channels and $f \times f \times c$ convolution filters.
  - Step 1:Convolution of each $n \times n$ input with $f \times f$ filter
  - Step 2: Combining the convolution results ..Output with c channels



- **Depthwise Separable Convolutions**
  - Assuming convolution of $n \times n \times c$ input with H $f \times f \times c$ filters (stride 1) attains $n \times n \times H$ output
  - To save computational costs but attain $n \times n \times H$ output.
  - How? – depthwise separable convolution!



- **Depthwise Separable Convolutions**
  - ① Depthwise convolution : $n \times n \times c$ output
  - ② Convolution of $n \times n \times c$ data with H $1 \times 1 \times c$ filters
  - Pointwise convolution : convolution with $1 \times 1 \times c$ filters
    - Combining c channel data into 1 channel $1 \times 1 \times c$ filters



- **Depthwise Separable Convolutions**
  - Comparison of computational costs

A. Convolution of $n \times n \times c$ input with H $f \times f \times c$ filters (stride 1)

$\Rightarrow f \times f \times c \times H \times n \times n$ multiplications

B. Depthwise separable convolution

$\Rightarrow$ ① depthwise convolution: $f \times f \times c \times n \times n$ multiplications

$\Rightarrow$ ② pointwise convolution: $c \times H \times n \times n$ multiplications

$f \times f \times c \times n \times n + c \times H \times n \times n$ multiplications

Ratio of computational Costs: $\frac{f \times f \times c \times n \times n + c \times H \times n \times n}{f \times f \times c \times H \times n \times n} = \frac{1}{H} + \frac{1}{f^2}$!!

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)

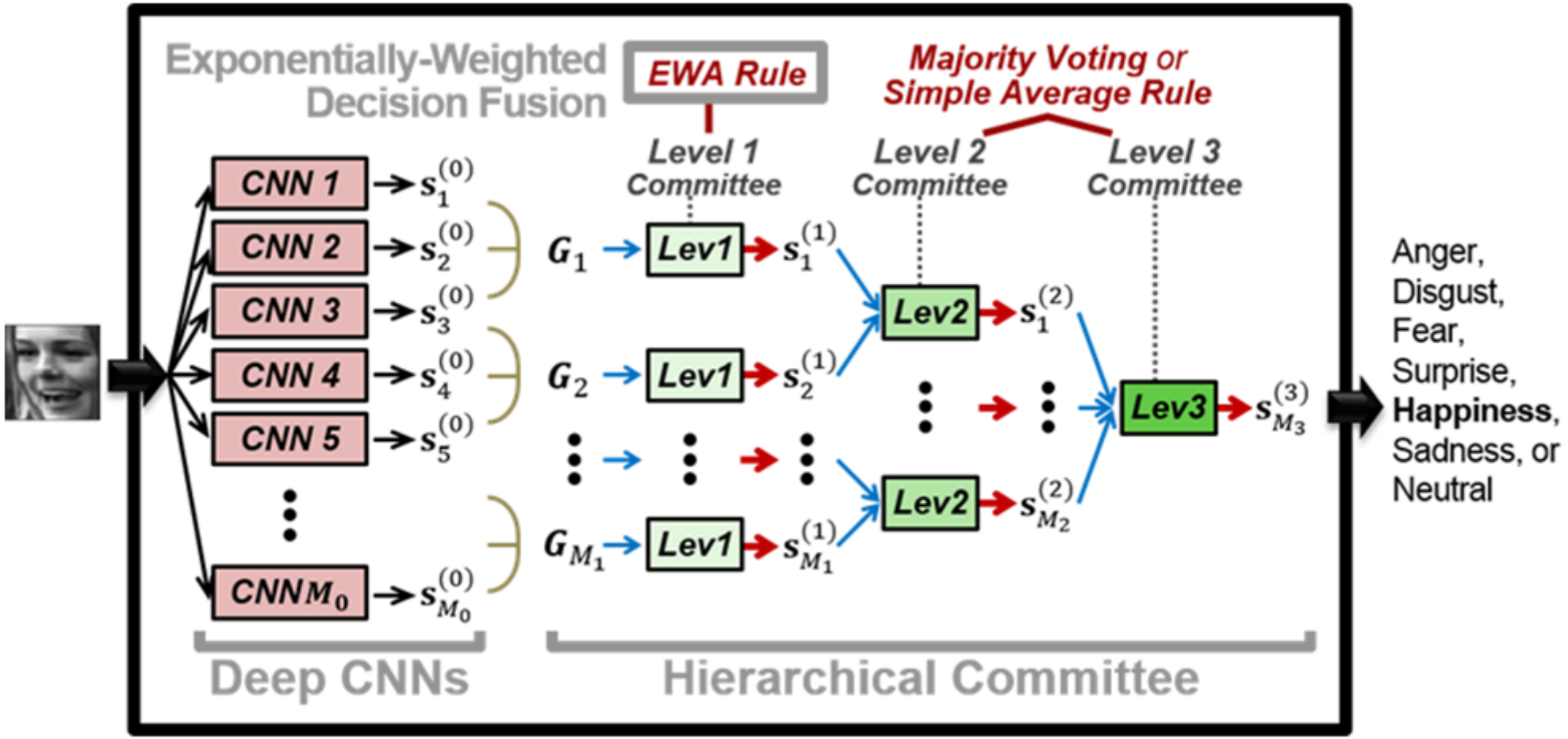


그림 9.17. CNSL팀이 제안한 얼굴 표정 인식을 위한 심층 CNN 위원회 구조

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

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

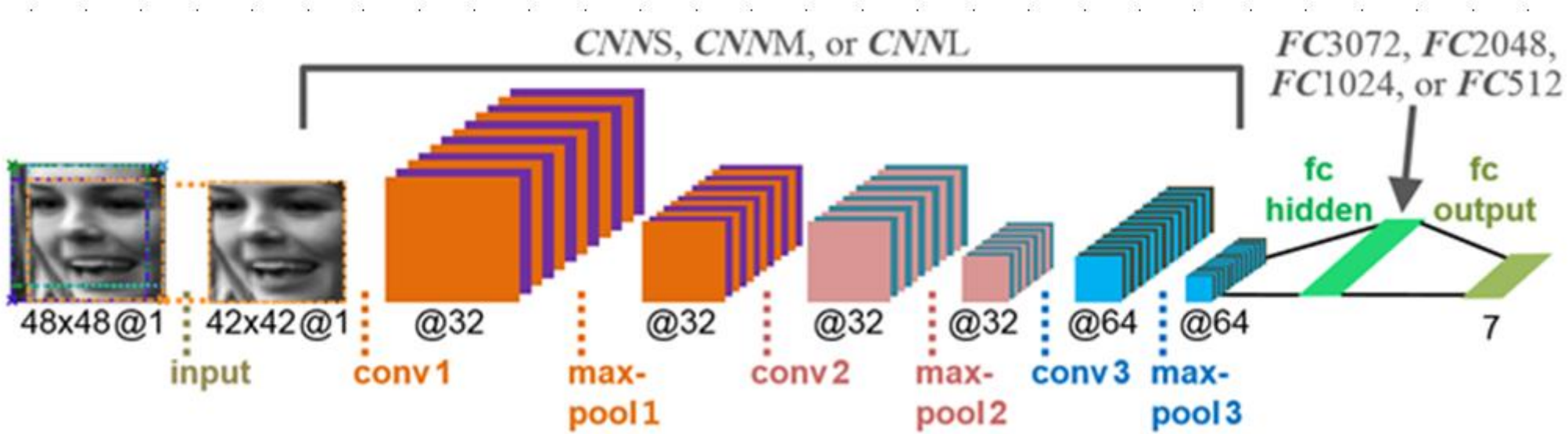


그림 9.18. 얼굴 표정 인식을 위한 심층 CNN 구조 [16. Bo-Kyung Kim et al. 2016]

- 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)

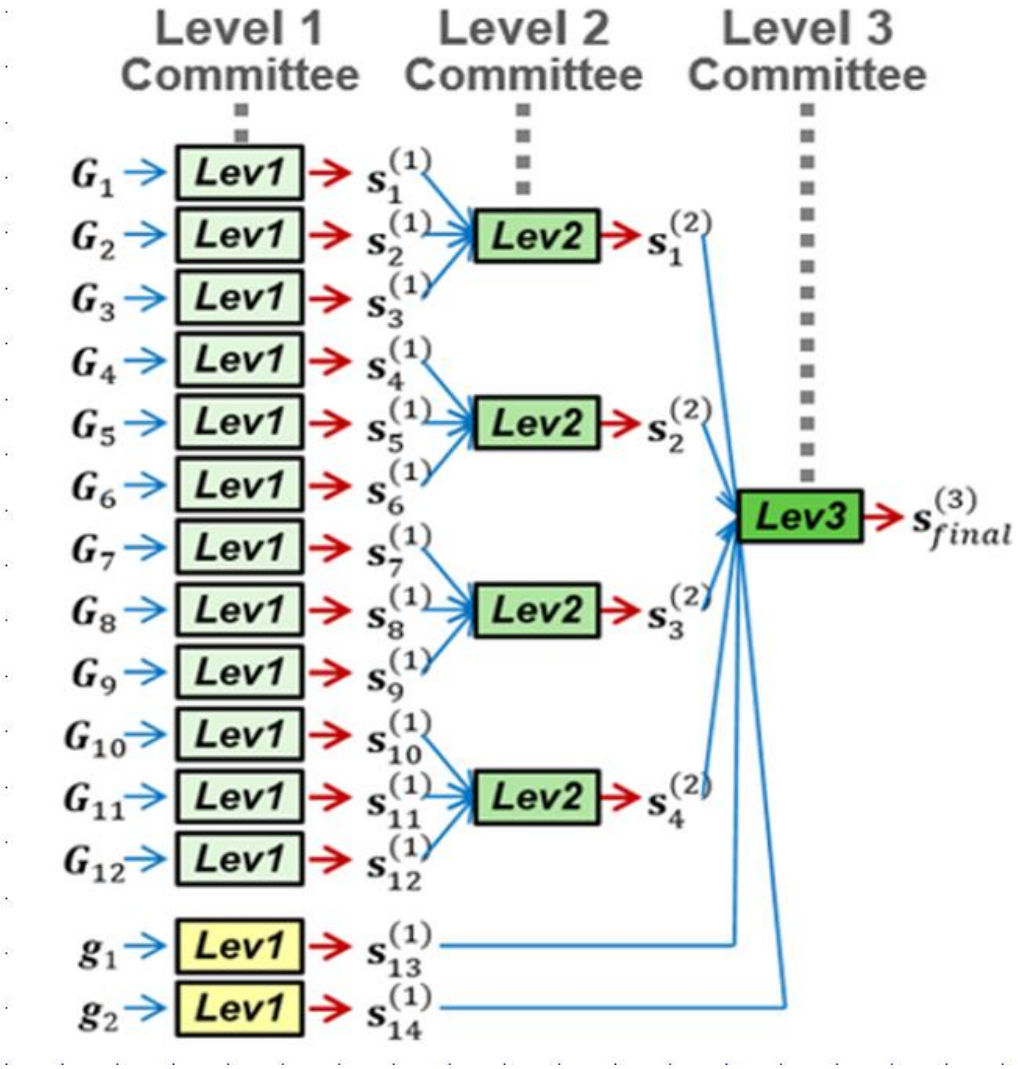


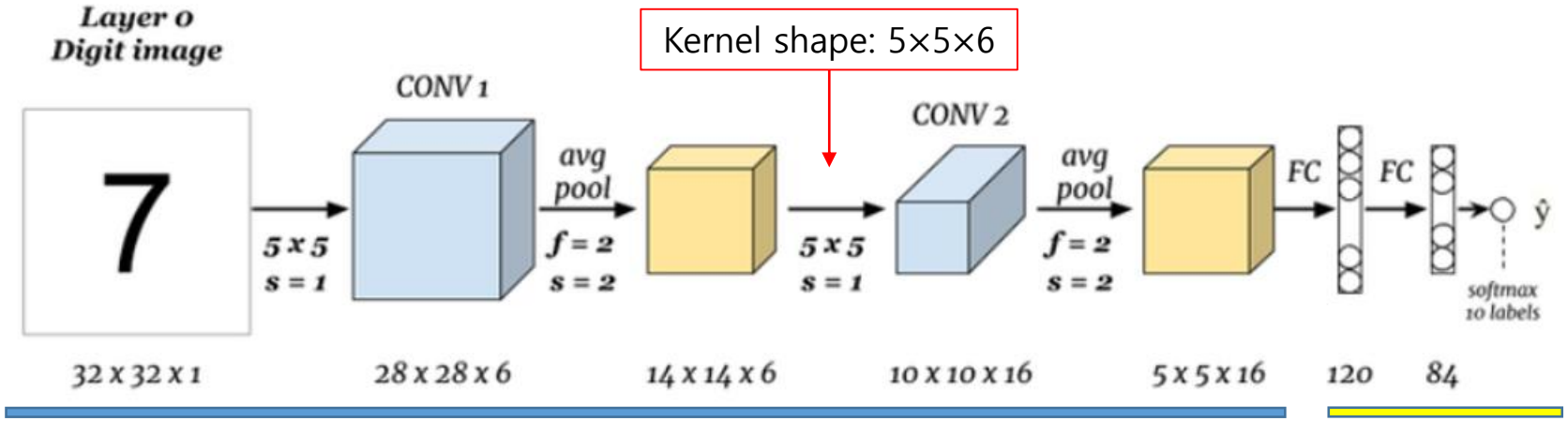
그림 9.20. 3개의 레벨로 구성된 위원회 구조도

[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

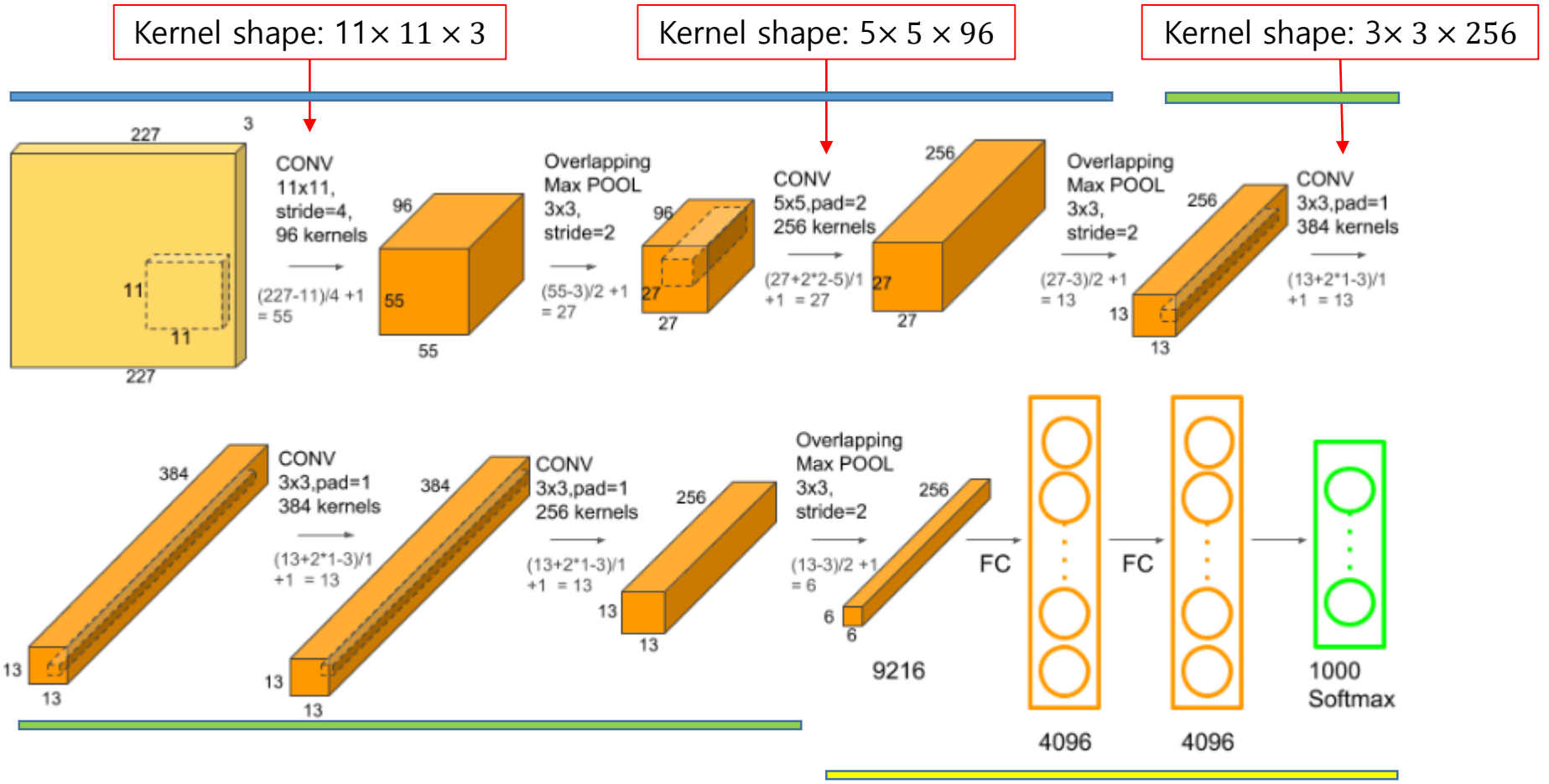
- AlextNet: ILSVRC2012(120만장의 영상을 1000개의 클래스로 구분)

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

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

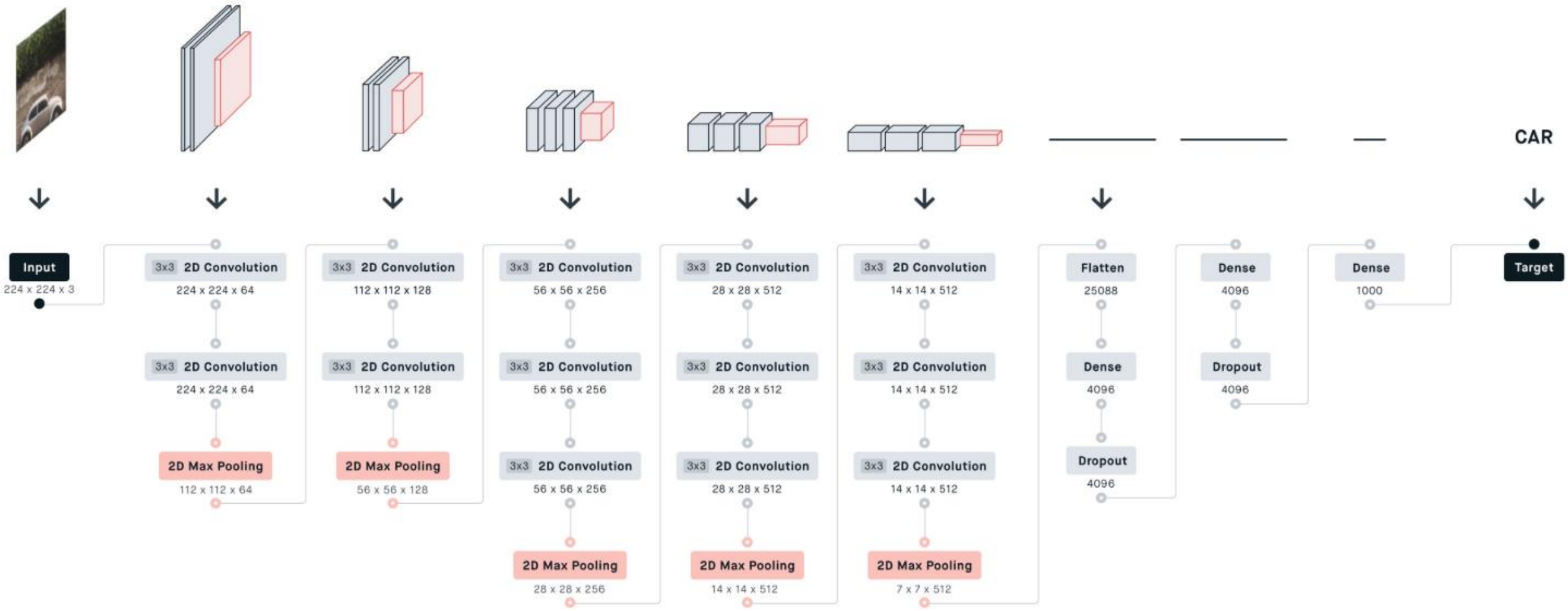
1. 227x227 RGB영상에 컨볼루션층과 풀링층 통과
2. 컨볼루션층 통과
3. 풀링층 거친 후 완전 연결층으로 결과 출력

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

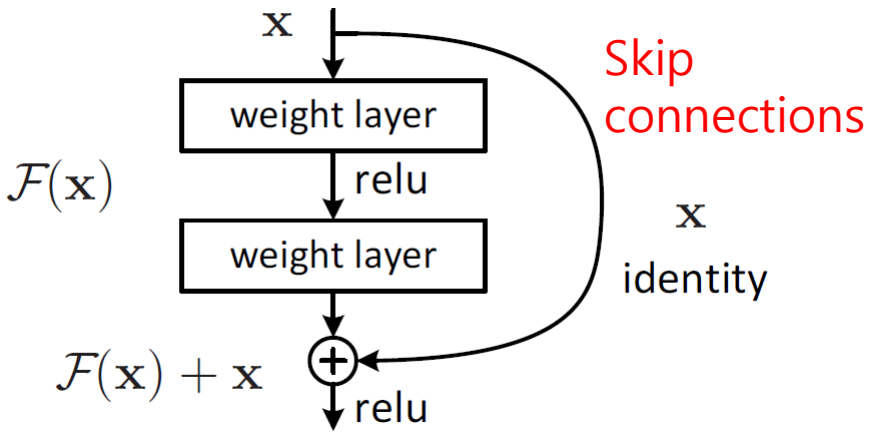
- ResNet: ILSVRC2015

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

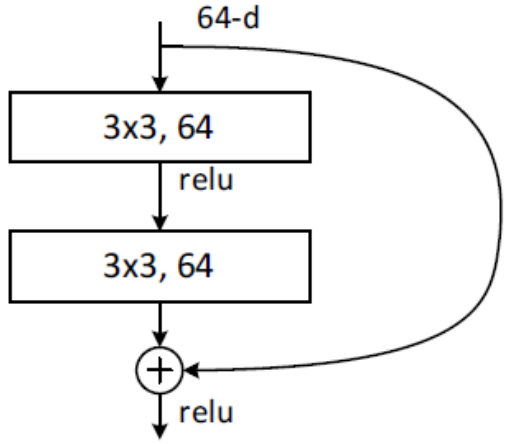
  - 가정: 원하는 사상 H(x)를 직접 학습하는 것 보다 F(x)=H(x)-x 를 학습하는 것이 쉽다.

⇒ 잔류학습 블록을 제시함.

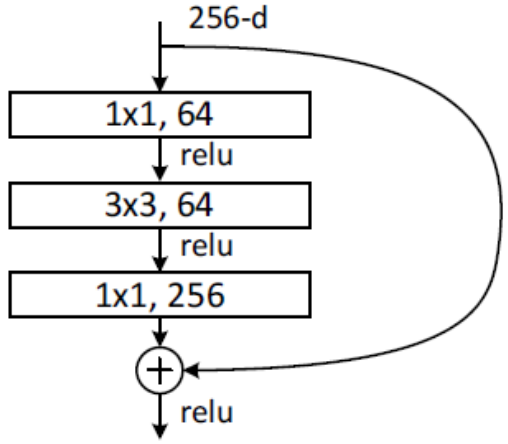
⇒ 신경회로망이 F(x)를 학습 후, 잔류학습 블록에서 F(x)+x를 출력함(ReLU 활성화 함수).



Residual Learning Block



Residual Learning Block for ReseNet-34

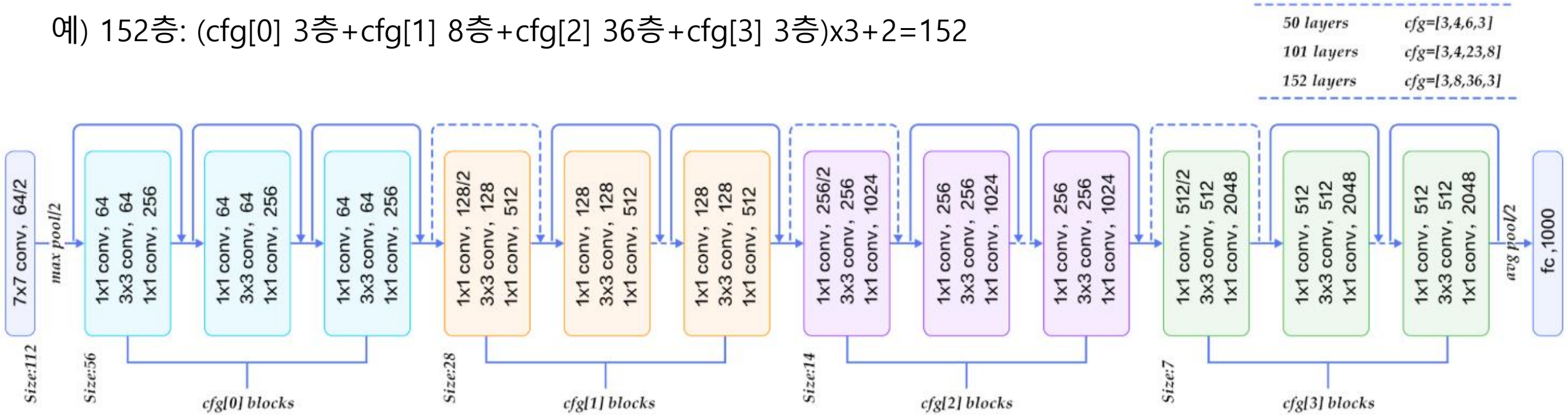


Residual Learning Block for ReseNet-50/101/152

• 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%

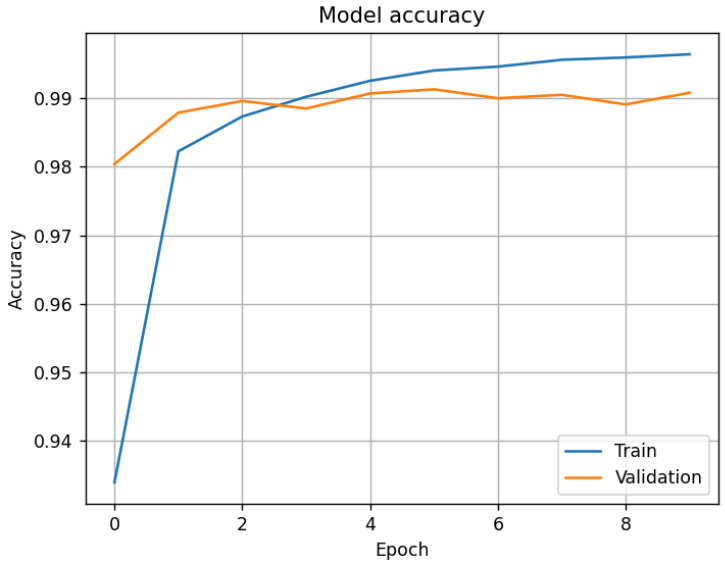
실습

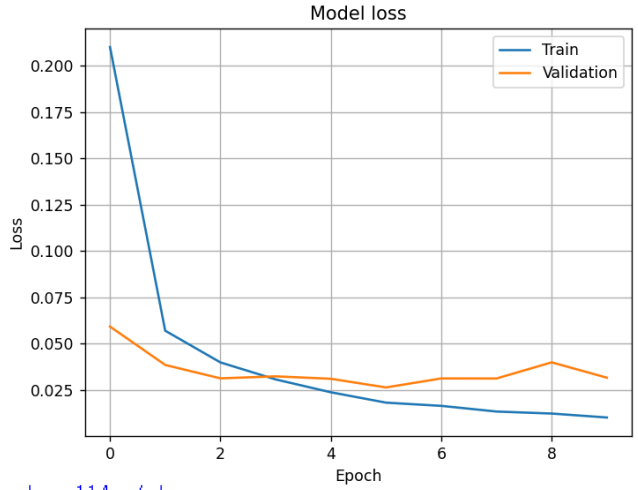
6-1b.py

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense
from tensorflow.keras.optimizers import Adam

import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

# MNIST 읽어 와서 신경망에 입력할 형태로 변환
(x_train, y_train), (x_test, y_test) = load_mnist()
img = x_train[0]*1000
label = y_train[0]
print(label)  # 5
print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)
img_show(img)

x_train=x_train.reshape(60000,28,28,1)
x_test=x_test.reshape(10000,28,28,1)
#x_train=x_train.astype(np.float32)/255.0
#x_test=x_test.astype(np.float32)/255.0
x_train=x_train.astype(np.float32)
x_test=x_test.astype(np.float32)
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)
```

실습

```
# LeNet-5 신경망 모델 설계
cnn=Sequential()
cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_shape=(28,28,1)))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Conv2D(16,(5,5),padding='same',activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Conv2D(120,(5,5),padding='same',activation='relu'))
cnn.add(Flatten())
cnn.add(Dense(84,activation='relu'))
cnn.add(Dense(10,activation='softmax'))

# 신경망 모델 학습
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=10,validation_data=(x_test,y_test),verbose=2)

# 신경망 모델 정확률 평가
res=cnn.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)

import matplotlib.pyplot as plt

# 정확률 그래프
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'],loc='best')
plt.grid()
plt.show()

# 손실 함수 그래프
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'],loc='best')
plt.grid()
plt.show()
```






```
Epoch 10/10
469/469 - 54s - loss: 0.0102 - accuracy: 0.9964 - val_loss: 0.0317 - val_accuracy: 0.9908 - 54s/epoch - 114ms/step
정확률은 99.08000230789185
```

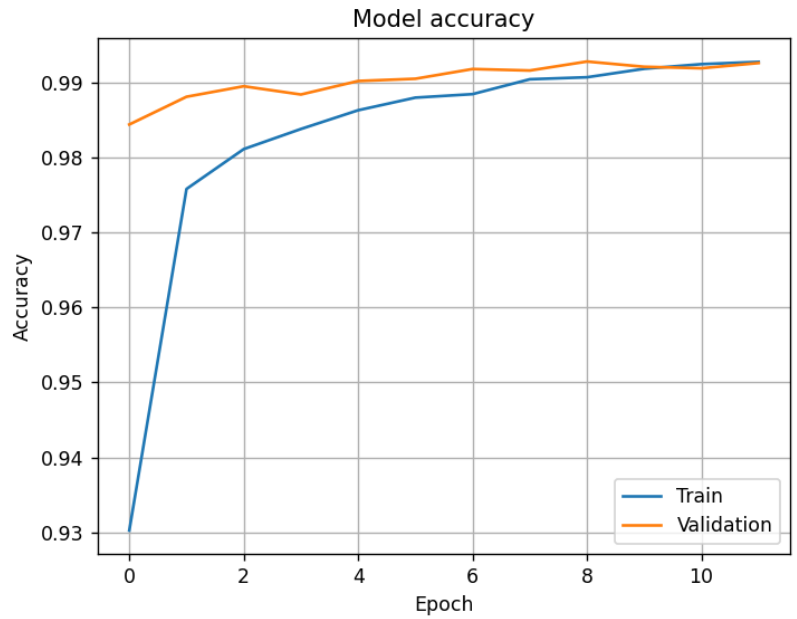
실습

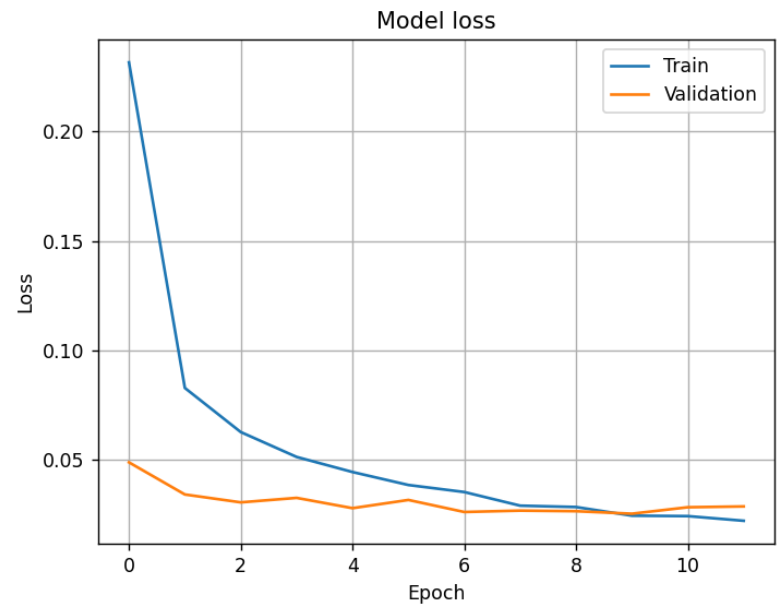
6-2b.py

구조:C-C-P-dropout-FC-dropout-FC

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam

import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

# MNIST 읽어 와서 신경망에 입력할 형태로 변환
(x_train, y_train), (x_test, y_test) = load_mnist()
img = x_train[0]*1000
label = y_train[0]
print(label)  # 5
print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)
img_show(img)

x_train=x_train.reshape(60000,28,28,1)
x_test=x_test.reshape(10000,28,28,1)
#x_train=x_train.astype(np.float32)/255.0
#x_test=x_test.astype(np.float32)/255.0
x_train=x_train.astype(np.float32)
x_test=x_test.astype(np.float32)
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

# 신경망 모델 설계
cnn=Sequential()
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(128,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10,activation='softmax'))
```

```python
# 신경망 모델 학습
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=12,validation_data=(x_test,y_test),verbose=2)

# 신경망 모델 정확률 평가
res=cnn.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)

import matplotlib.pyplot as plt

# 정확률 그래프
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()

# 손실 함수 그래프
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()
```






```
Epoch 12/12
469/469 - 134s - loss: 0.0222 - accuracy: 0.9927 - val_loss: 0.0287 - val_accuracy: 0.9926 - 134s/epoch - 286ms/step
정확률은 99.26000237464905
```

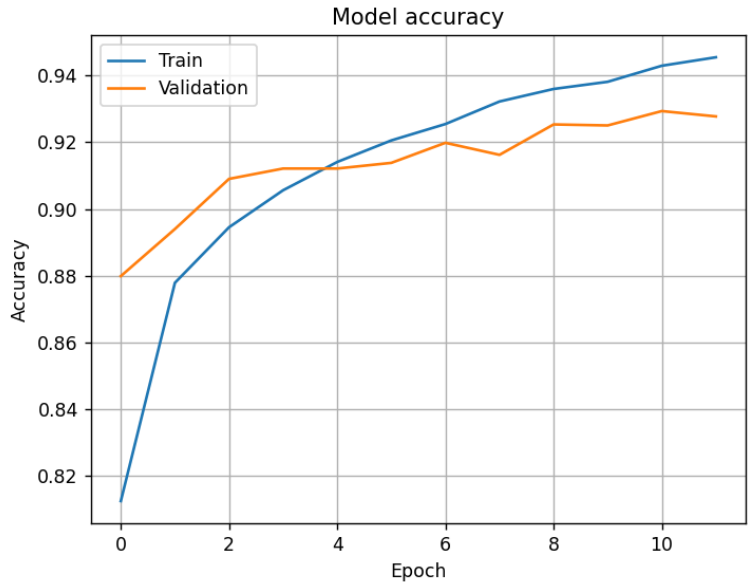
실습

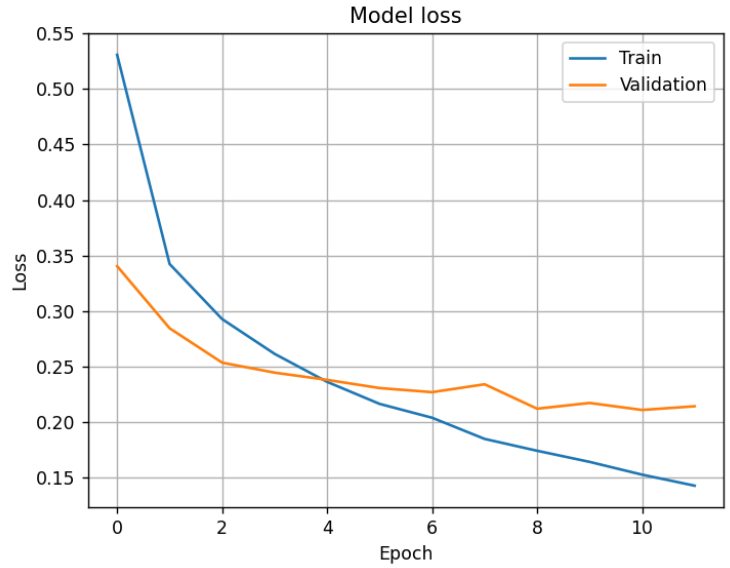
6-3b.py

구조:C-C-P-dropout-FC-dropout-FC

```
import numpy as np
import tensorflow as tf
#from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam


import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정

path='dataset₩MNISTfashion'
os.chdir(path)

# fashion MNIST 데이터셋을 읽어와 신경망에 입력할 형태로 변환
x_train = np.load('x_train.npy')
y_train = np.load('y_train.npy')
x_test = np.load('x_test.npy')
y_test = np.load('y_test.npy')


print(x_train.shape, x_test.shape)


x_train=x_train.reshape(60000,28,28,1)
x_test=x_test.reshape(10000,28,28,1)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

# 신경망 모델 설계
cnn=Sequential()
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(128,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10,activation='softmax'))
```

```python
# 신경망 모델 학습
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=12,validation_data=(x_test,y_test),verbose=2)

# 신경망 모델 정확률 평가
res=cnn.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)

import matplotlib.pyplot as plt

# 정확률 그래프
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()

plt.show()

# 손실 함수 그래프
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()
```






```
Epoch 12/12
469/469 - 175s - loss: 0.1427 - accuracy: 0.9454 - val_loss: 0.2142 - val_accuracy: 0.9277 - 175s/epoch - 373ms/step
정확률은 92.76999831199646
```

6.4.4 자연 영상 인식

- 딥러닝 프로그래밍에서 주로 사용하는 자연 영상 데이터베이스
  - ImageNet
  - MSCoCo
  - CIFAR: 작아서 MNIST 다음에 주로 사용
- CIFAR-10
  - {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}의 10부류
  - 영상은 32*32 맵으로 표현

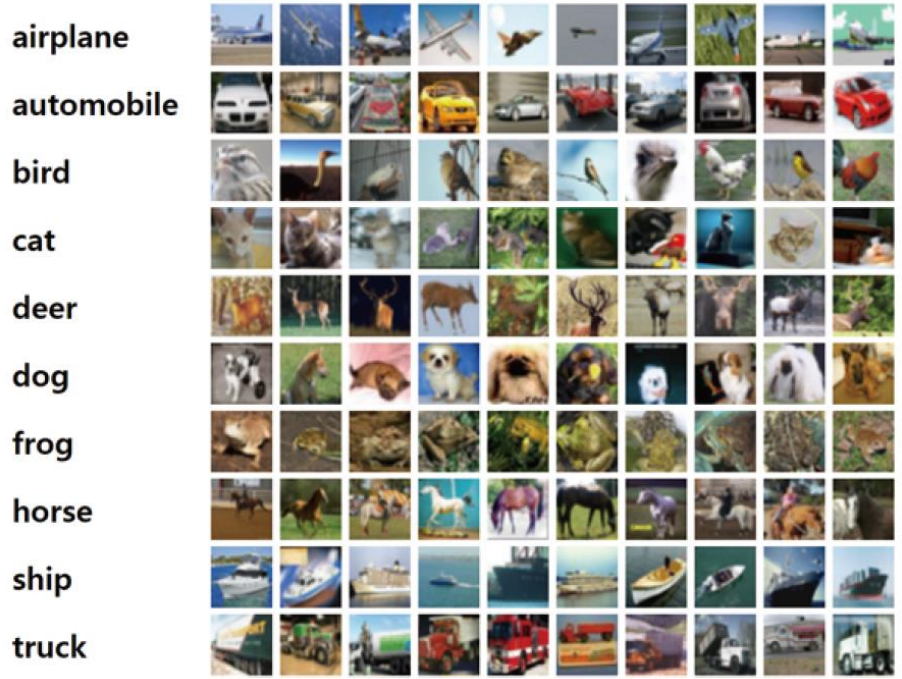


그림 6-14 CIFAR-10 데이터셋

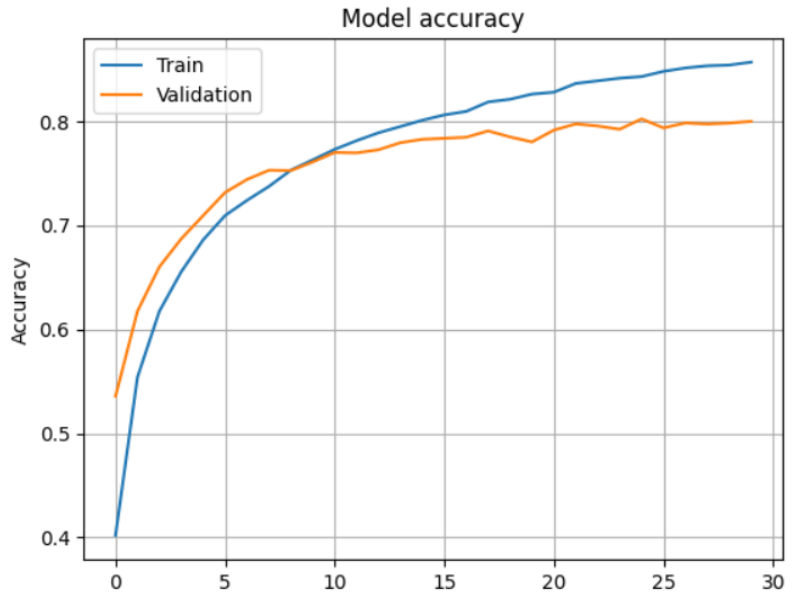
실습

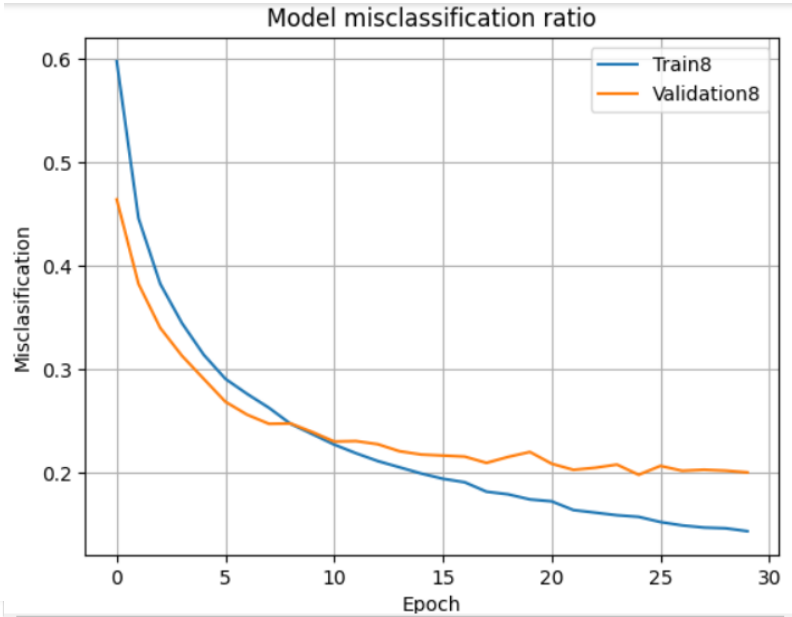
6-4.py

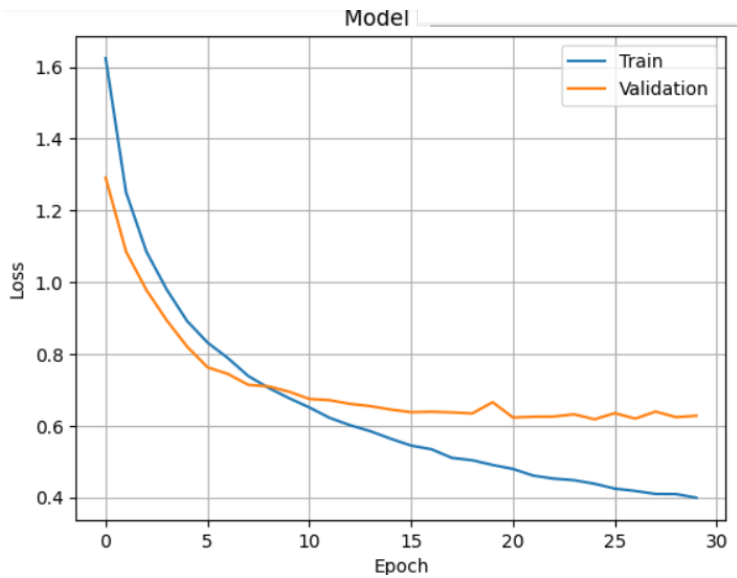
Colab에서 실행

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout
from tensorflow.keras.optimizers import Adam

# CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

# 신경망 모델 설계
cnn=Sequential()
cnn.add(Conv2D(32,(3,3),activation='relu',input_shape=(32,32,3)))
cnn.add(Conv2D(32,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(Conv2D(64,(3,3),activation='relu'))
cnn.add(MaxPooling2D(pool_size=(2,2)))
cnn.add(Dropout(0.25))
cnn.add(Flatten())
cnn.add(Dense(512,activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(10,activation='softmax'))
```

```python
# 신경망 모델 학습
cnn.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])
hist=cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_data=(x_test,y_test),verbose=2)

# 신경망 모델 정확률 평가
res=cnn.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)

import matplotlib.pyplot as plt

# 정확률 그래프
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()

#오인식률
plt.plot(1.-np.array(hist.history['accuracy']))
plt.plot(1.-np.array(hist.history['val_accuracy']))
plt.title('Model misclassification ratio')
plt.ylabel('Misclasification')
plt.xlabel('Epoch')
plt.legend(['Train8','Validation8'], loc='best')
plt.grid()
plt.show()

# 손실 함수 그래프
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'],loc='best')
plt.grid()
plt.show()
cnn.save("my_cnn.h5")
```








```
Epoch 30/30
391/391 - 4s - loss: 0.3991 - accuracy: 0.8567 - val_loss: 0.6276 - val_accuracy: 0.7998 - 4s/epoch - 10ms/step
정확률은 79.97999787330627
```

6-5.py

Colab에서 실행

저장된 모델 'my_cnn.h5' 불러와서 사용

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10

# 신경망 구조와 가중치를 저장하고 있는 파일을 읽어 옴
cnn=tf.keras.models.load_model("my_cnn.h5")
cnn.summary()

# CIFAR-10 데이터셋을 읽고 신경망에 입력할 형태로 변환
(x_train,y_train),(x_test,y_test)=cifar10.load_data()
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

res=cnn.evaluate(x_test,y_test,verbose=0)
print("정확률은",res[1]*100)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d (MaxPooling2  (None, 14, 14, 32)        0
 D)

 dropout (Dropout)           (None, 14, 14, 32)        0

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        18496

 conv2d_3 (Conv2D)           (None, 10, 10, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 890410 (3.40 MB)
Trainable params: 890410 (3.40 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

정확률은 79.97999787330627