CNN 모델
Convolutional Neural Network Models

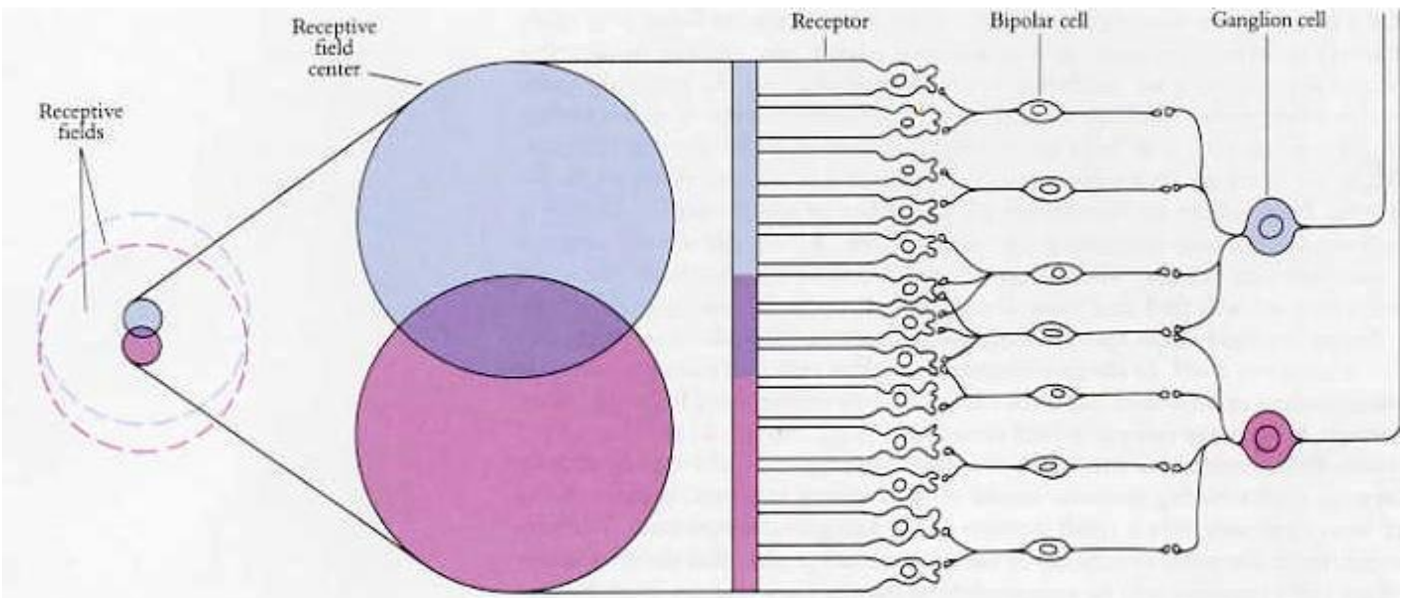
이건명
충북대학교 소프트웨어학과

학습 내용

- 컨볼루션 신경망의 구조에 대해서 알아본다.
- 컨볼루션 신경망에서 사용되는 컨볼루션, 풀링 등의 연산에 대해 알아본다.

1. 컨볼루션 신경망

- ❖ **컨볼루션 신경망**(convolutional neural network, CNN)
  - ▪ 동물의 **시각피질**(visual cortex, 視覺皮質)의 구조에서 영감을 받아 만들어진 딥러닝 신경망 모델
    - **시각피질**의 신경세포
      - – 시야 내의 특정 영역에 대한 자극만 수용
        - » 수용장(receptive field, 受容場)
      - – 해당 영역의 특정 특징에 대해서만 반응

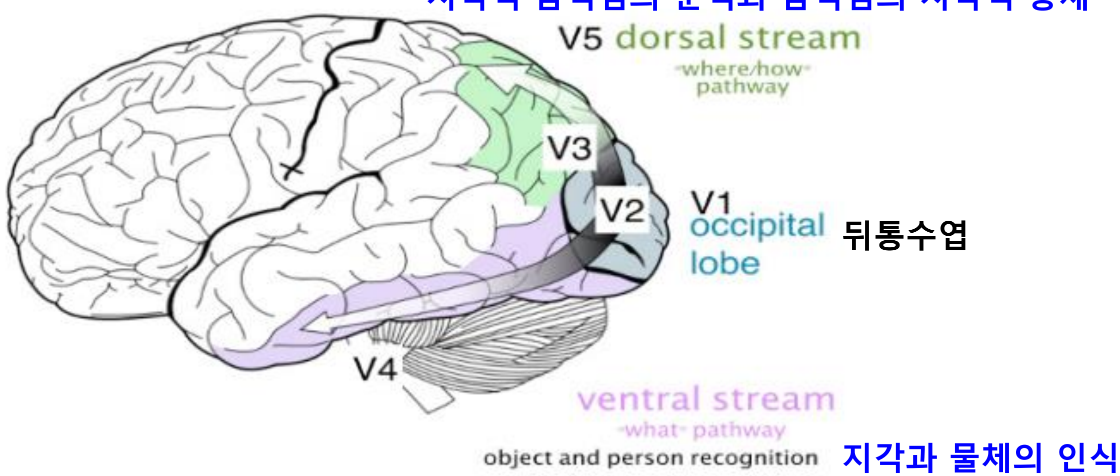


D.H. Hubel se al., 1995

컨볼루션 신경망

❖ **컨볼루션 신경망**(convolutional neural network, CNN) – cont.

- **시각 자극**이 **1차 시각피질**을 통해서 처리된 다음, **2차 시각피질**을 경유하여, **3차 시각피질** 등 여러 영역을 통과하여 계층적인 정보처리
  - 정보가 계층적으로 처리되어 가면서 점차 추상적인 특징이 추출되어 시각 인식

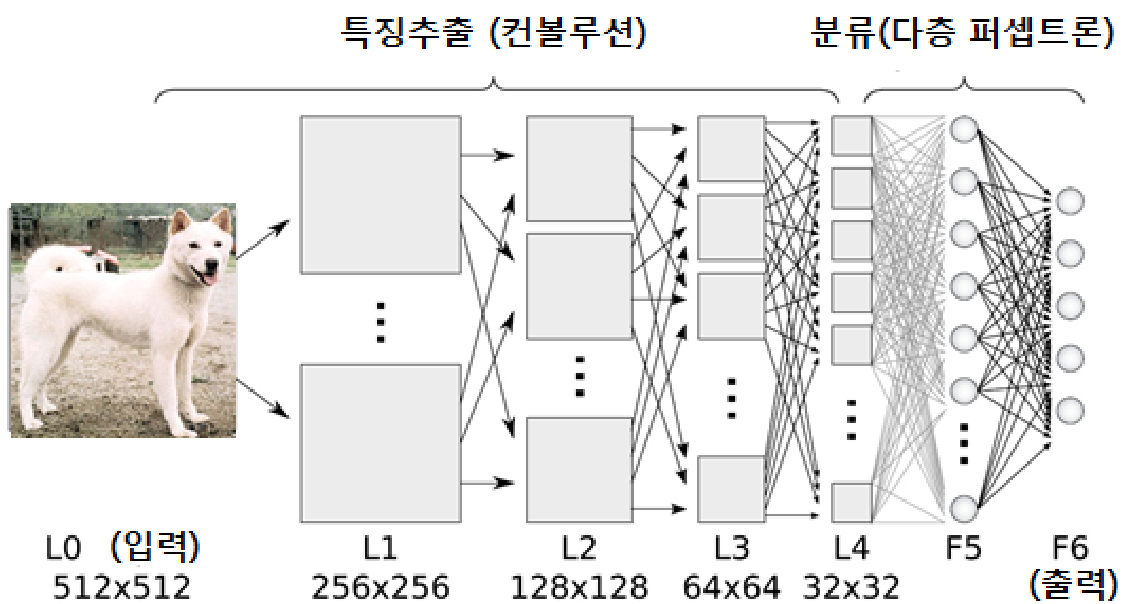


▪ 동물의 **계층적 특징 추출**과 **시각인식 체계**를 참조하여 만들어진 모델

http://s3-eu-west-1.amazonaws.com/assets.aims.fao.org/public/GODANAction_Lesson2.3.pdf

컨볼루션 신경망

- ❖ **컨볼루션 신경망**(Convolutional Neural Network, **CNN**)
  - 전반부 : 컨볼루션 연산을 수행하여 **특징 추출**
  - 후반부 : 특징을 이용하여 **분류**
  - 영상분류, 문자 인식 등 인식문제에 높은 성능



2. 컨볼루션

❖ **컨볼루션**(covolution)

- 일정 영역의 값들에 대해 가중치를 적용하여 하나의 값을 만드는 연산

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}

입력

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

컨볼루션 필터
커널
마스크

y_{11}	y_{12}	y_{13}
y_{21}	y_{22}	y_{23}
y_{31}	y_{32}	y_{33}

컨볼루션 결과

$$
\begin{aligned}
y_{11} = \; & w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} \\
& + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} \\
& + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33} \\
& + w_0
\end{aligned}
$$

컨볼루션

❖ 컨볼루션

11	10	10	00	01
00	10	10	10	00
00	00	10	10	10
00	00	10	10	00
01	10	10	00	01

입력

1	0	1
0	1	0
1	0	1

컨볼루션 필터
커널
마스크

4	3	4
2	4	3
2	3	4

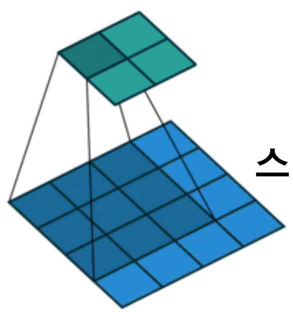
컨볼루션 결과

$$
\begin{aligned}
y_{11} = {} & w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} \\
& + w_{21}x_{21} + w_{22}x_{22} + w_{23}x_{23} \\
& + w_{31}x_{31} + w_{32}x_{32} + w_{33}x_{33} \\
& + w_0
\end{aligned}
$$

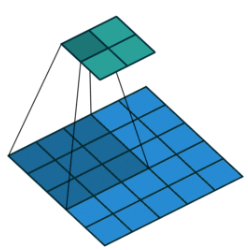
컨볼루션

- ❖ **스트라이드**(stride, 보폭)
  - 커널을 다음 컨볼루션 연산을 위해 이동시키는 칸 수

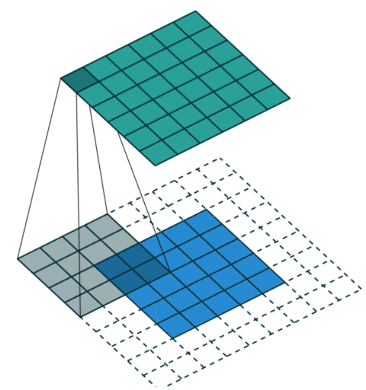
- ❖ **패딩**(padding)
  - 컨볼루션 결과의 크기를 조정하기 위해 입력 배열의 둘레를 확장하고 0으로 채우는 연산



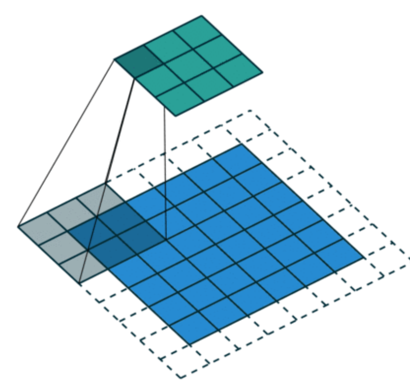
스트라이드: 1



스트라이드: 2



패딩: 2
스트라이드: 1



패딩: 1
스트라이드: 2

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

[실습] 컨볼루션

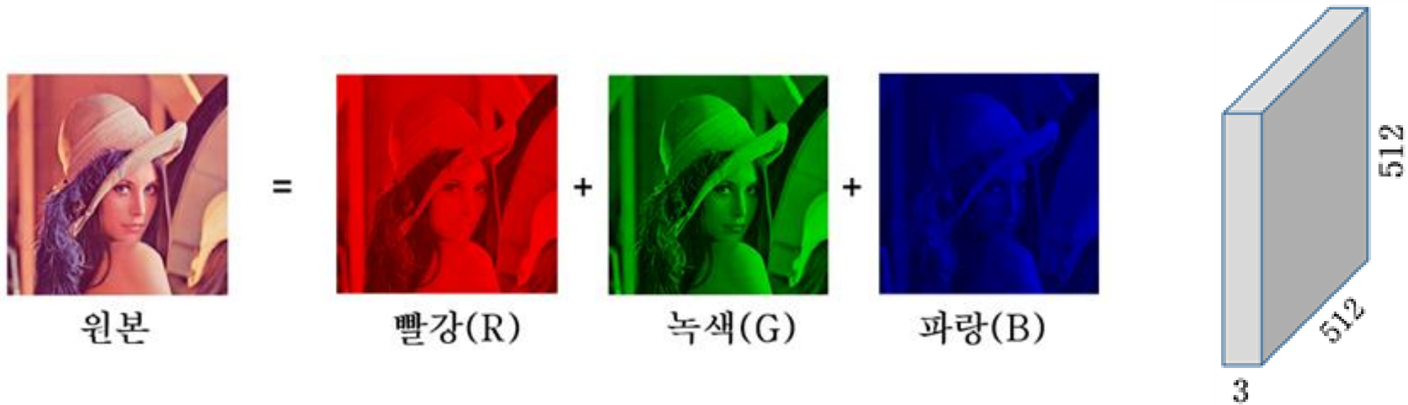
```
import numpy as np

def Conv2D(X, W, w0, p=(0,0), s=(1,1)):
    n1 = X.shape[0] + 2*p[0]   # 패딩 반영
    n2 = X.shape[1] + 2*p[1]
    X_p = np.zeros(shape=(n1,n2))
    X_p[p[0]:p[0]+X.shape[0], p[1]:p[1]+X.shape[1]]= X  # 입력 X 복사
    res = [ ]
    for i in range(0, int((X_p.shape[0] - W.shape[0])/s[0])+1, s[0]):
        res.append([ ])
        for j in range(0, int((X_p.shape[1] - W.shape[1])/s[1])+1, s[1]):
            X_s = X_p[i:i+W.shape[0], j:j+W.shape[1]]  # 컨볼루션 영역
            res[-1].append(np.sum(X_s * W) + w0)   # 컨볼루션
    return (np.array(res))

X = np.array([[1,1,1,0,0], [0,1,1,1,0], [0,0,1,1,1], [0,0,1,1,0],[0,1,1,0,0]])
W = np.array([[1,0,1], [0,1,0], [1,0,1]])
w0 = 1

conv = Conv2D(X, W, w0, p=(0,0), s=(1,1))
print('X = ', X)
print('₩nW = ', W)
print('₩n컨볼루션 결과 p=(0,0), s=(1,1) ₩n', conv)
conv = Conv2D(X, W, w0, p=(1,1), s=(1,1))
print('₩n컨볼루션 결과 p=(1,1), s=(1,1) ₩n', conv)
conv = Conv2D(X, W, w0, p=(1,1), s=(2,2))
print('₩n컨볼루션 결과 p=(1,1), s=(2,2) ₩n', conv)
```

```
X =
[[1 1 1 0 0]
 [0 1 1 1 0]
 [0 0 1 1 1]
 [0 0 1 1 0]
 [0 1 1 0 0]]

W =
[[1 0 1]
 [0 1 0]
 [1 0 1]]

컨볼루션 결과 p=(0,0), s=(1,1)
 [[5. 4. 5.]
 [3. 5. 4.]
 [3. 4. 5.]]

컨볼루션 결과 p=(1,1), s=(1,1)
 [[3. 3. 4. 2. 2.]
 [2. 5. 4. 5. 2.]
 [2. 3. 5. 4. 4.]
 [2. 3. 4. 5. 2.]
 [1. 3. 3. 2. 2.]]

컨볼루션 결과 p=(1,1), s=(2,2)
 [[3. 4.]
 [2. 5.]]
```

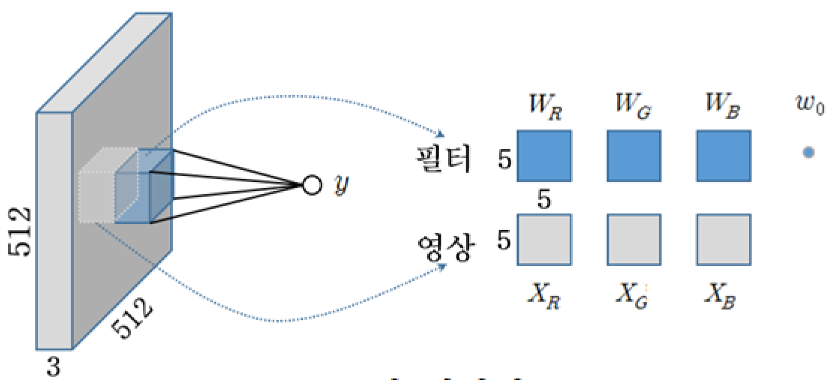
컨볼루션

❖ 칼러 영상의 컨볼루션

- 칼러 영상의 다차원 행렬 표현



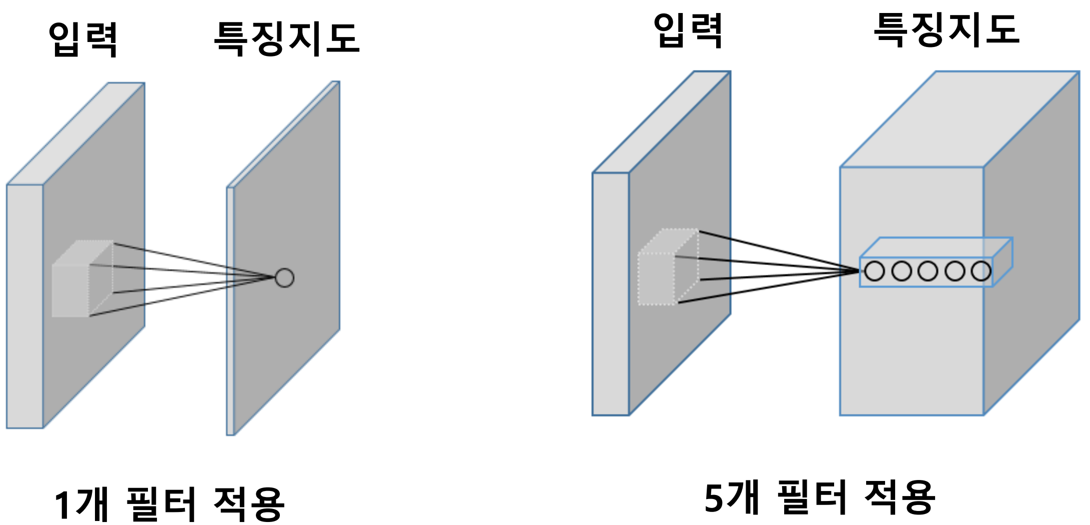
- 칼러영상의 컨볼루션



$$y = X_R * W_R + X_G * W_G + X_B * W_B + w_0$$

컨볼루션

❖ **특징지도**(feature map)
- 컨볼루션 필터의 적용 결과로 만들어지는 2차원 행렬
- 특징지도의 원소값
  - 컨볼루션 필터에 표현된 특징을 대응되는 위치에 포함하고 있는 정도
- **k개의 컨볼루션 필터**를 적용하면 **k의 2차원 특징지도** 생성



3. 풀링

❖ **풀링**(pooling)

- 일정 크기의 블록을 통합하여 하나의 대푯값으로 대체하는 연산

- **최대값 풀링**(max pooling)
  - 지정된 블록 내의 원소들 중에서 최대값을 대푯값으로 선택

1	1	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

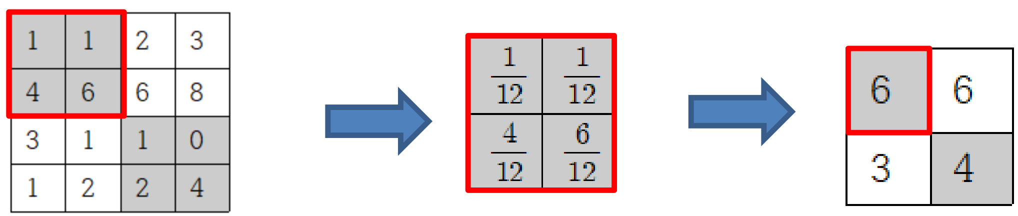
- **평균값 풀링**(average pooling)
  - 블록 내의 원소들의 평균값을 대푯값으로 사용

1	1	2	3
4	6	6	8
3	1	1	0
1	2	2	4

3	4.75
1.75	1.75

풀링

- **확률적 풀링**(stochastic pooling)
  - 블록 내의 각 원소가 원소값의 크기에 비례하는 선택 확률을 갖도록 하고, 이 확률에 따라 원소 하나를 선택



  - 학습시: 확률적 풀링

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

p_i : 블록 R_j에서 원소 a_i가 선택될 확률

  - 추론시 : 확률적 가중합 사용

$$s_j = \sum_{i \in R_j} p_i a_i$$

풀링

❖ 풀링 연산의 역할

- 중간 연산 과정에서 만들어지는 **특징지도**들의 **크기 축소**
  - 다음 단계에서 사용될 메모리 크기와 계산량 감소
- 일정 영역 내에 나타나는 **특징**들을 **결합**하거나,
  위치 변화에 **강건**한 특징 선택

1	1	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

[실습] 풀링

```
import numpy as np

def maxPooling(mat, K, L):
    M, N = mat.shape
    MK = M // K
    NL = N // L
    pmat = mat[:MK*K, :NL*L].reshape(MK, K, NL, L).max(axis=(1, 3))
    return pmat

mat = np.array([[  20,  200,   -5,   23],
       [ -13,  134,  119,  100],
       [ 120,   32,   49,   25],
       [-120,   12,    9,   23]])

print(maxPooling(mat, 2,2))
```

```
[[200 119]
 [120  49]]
```

4. 컨볼루션 신경망의 구조

❖ 컨볼루션 신경망의 구조

- **특징 추출**을 위한 **컨볼루션 부분**
  - 컨볼루션 연산을 하는 **Conv층**
  - ReLU 연산을 하는 **ReLU**
  - 풀링 연산 **Pool**(선택)]

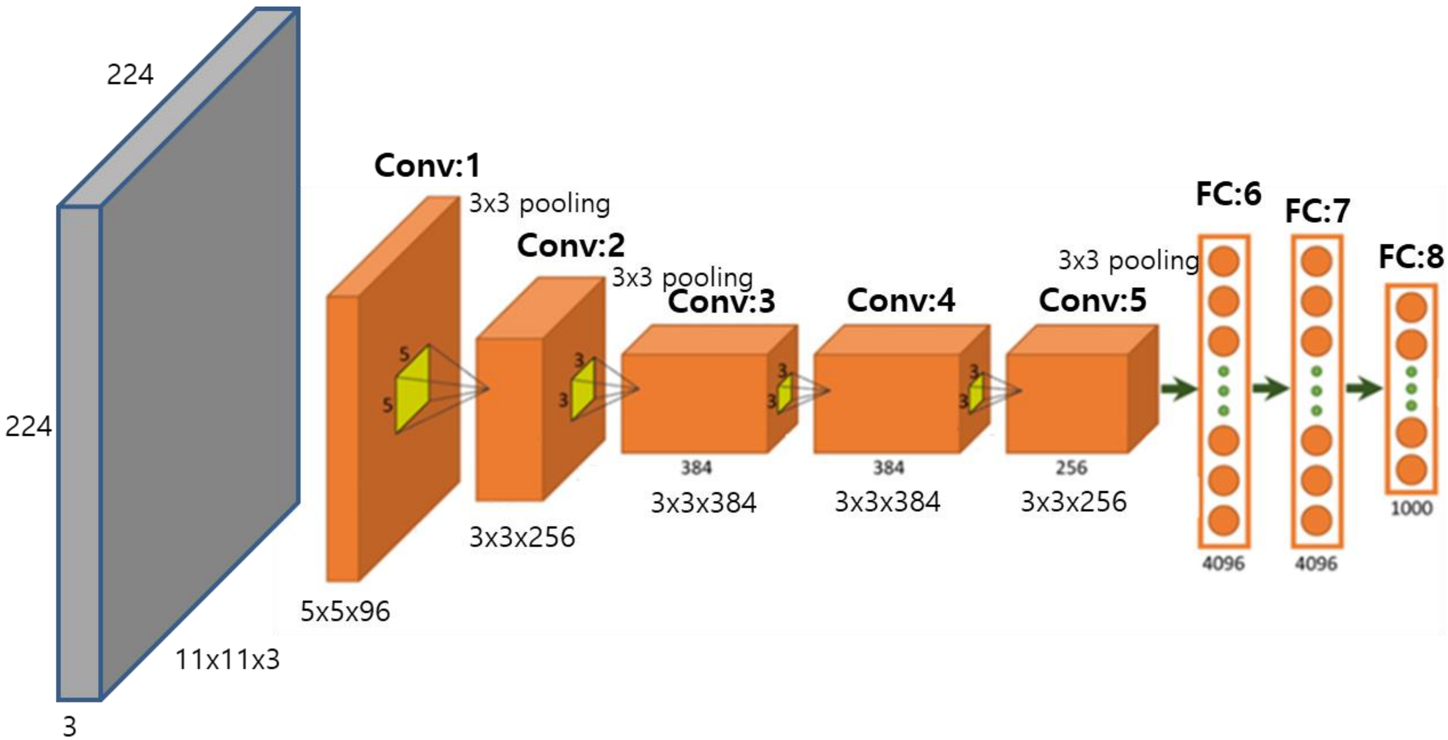
  반복

- 추출된 특징을 사용하여 **분류** 또는 **회귀**를 수행하는 **다층 퍼셉트론 부분**
  - 전방향으로 **전체 연결**된(fully connected) **FC층** 반복
  - 분류의 경우 **마지막 층**에 **소프트맥스**(softmax)을 하는 SM 연산 추가
    - 소프트맥스 연산 : 출력의 값이 0이상이면서 합은 1로 만듦

- 컨볼루션 신경망 구조의 예
  - **Conv-ReLU-Pool-Conv-ReLU-Pool-Conv-ReLU-Pool-FC-SM**
  - **Conv-Pool-Conv-Pool-Conv-FC-FC-SM**
  - **Conv-Pool-Conv-Pool-Conv-Conv-Conv-Pool-FC-FC-SM**
  - **Conv-ReLU-Pool-Conv-ReLU-Pool-Conv-ReLU-Pool-FC-FC-SM**

컨볼루션 신경망의 구조

- ❖ **컨볼루션 신경망의 구조 예**
  - **Conv:1-Pool:1-Conv:2-Pool:2-Conv:3-Conv:4-Conv:5-Pool:4-FC:6-FC:7-FC:8**



컨볼루션 신경망의 구조

❖ 컨볼루션 신경망의 학습대상 가중치 개수와 메모리 요구량

층	필터/블록 크기	필터 개수	스트라 이드	패딩	노드개수 (출력 크기)	학습대상 가중치 개수
입력					224×224×3 (=150,528)	
Conv:1	11×11x3	96	4	3	55x55x96 (=290,400)	(11×11×3+1)x96 (=34,944)
Pool:1	3×3		2		27×27×96 (=69,984)	
Conv:2	5×5×96	256	1	2	27×27×256 (=186,624)	(5×5×96+1)×256 (=614,656)
Pool:2	3×3		2		13×13×256 (=43,264)	
Conv:3	3×3×256	384	1	1	13×13×384 (=64,896)	(3×3×256+1)×384 (=885,120)
Conv:4	3×3×384	384	1	1	13×13×384 (=64,896)	(3×3×384+1)×384 (=1,327,488)
Conv:5	3×3×384	256	1	1	13×13×256 (=43,264)	(3×3×384+1)×256 (=884,992)
Pool:5	3×3	256	2		6×6×256 (=9,216)	
FC:6					4096	6×6×256×4096 (=37,748,736)
FC:7					4096	4096×4096 (=16,777,216)
FC:8					1000	4096×1000 (=4,096,000)

- **가중치**
  개수 : **58,621,952**
  메모리 요구량 :
  4바이트 float 사용시
  249,476,608 바이트
  (≈ 237MB)

- **계산 결과저장**
  노드 개수: **781,736**
  메모리 요구량: ≈ 3MB

가중치의 학습

❖ **미분 구현의 방법**

- **수치적 미분(numerical differentiation)**
  - 작은 h을 사용하여 $f(x+h)$와 $f(x)$의 값을 계산하여 미분

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

- **기호적 미분(symbolic differentiation)**
  - 미분의 수식 이용

$$\frac{d}{dx}\left(\frac{x^2 \cos(x-7)}{\sin(x)}\right) = x^2 \sin(7-x) \csc(x) + x^2 (-\cos(7-x)) \cot(x) \csc(x) + 2\, x \cos(7-x) \csc(x)$$

가중치의 학습

❖ **미분 구현의 방법 – cont.**

- **자동 미분**(automatic differentiation)
  - 함수를 **기본 연산**(primitive)의 **제어 흐름**(control flow)로 나타내서, 각 **기본 연산**의 **도함수**를 사용하여 **연쇄법칙**에 의해 미분 값 계산

$$f(x_1, x_2) = x_1 x_2 + \sin x_1$$

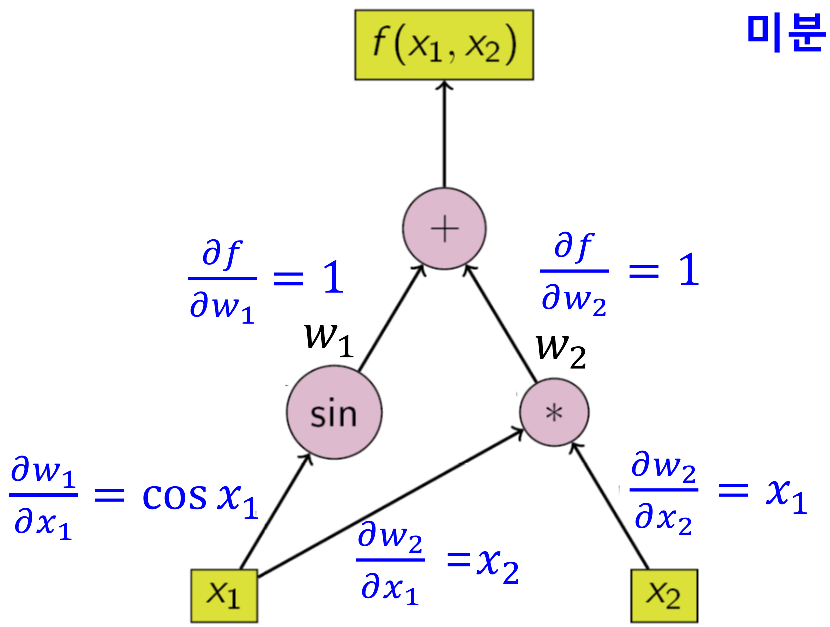
함수 값의 계산

$$f(x_1, x_2) = w_1 + w_2$$

$$w_1 = \sin x_1$$

$$w_2 = x_1 x_2$$

미분 값의 계산



$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial w_1}\frac{\partial w_1}{\partial x_1} + \frac{\partial f}{\partial w_2}\frac{\partial w_2}{\partial x_1} = \cos x_1 + x_2$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial w_2}\frac{\partial w_2}{\partial x_2} = x_1$$

Quiz

1. 컨볼루션 연산은 일반적으로 한번에 입력 전체에 대해서 가중치를 적용하여 하나의 값을 만든다. (O,X)
2. k개의 컨볼루션 필터를 적용하면 k개의 2차원 특징지도가 생성된다. (O,X)
3. 최대값 풀링은 지정된 블록 내의 원소들 중에서 최대값을 대푯값으로 선택한다. (O,X)
4. 풀링 연산을 하면 일반적으로 입력보다 더 큰 크기의 특징지도가 만들어진다. (O,X)
5. 딥러닝 신경망에는 학습 가능한 가중치가 많기 때문에, 학습 데이터의 개수가 적어도 충분히 좋은 성능을 갖도록 학습시킬 수 있다. (O,X)