

CONVOLUTION NEURAL NETWORK

Add a brief explanation of the presentation's purpose



목차

01 이미지 분류

02 CNN 구성요소

03 이미지 학습과정

04 CNN 아키텍처

05 실습

이미지 분류

- 이미지 분류는 컴퓨터가 사람처럼 시각적으로 사물을 인식하고 분류하는 작업이며, Computer Vision 분야에 속함.
- 기존 신경망 모델보다 더 정교한 분석이 가능한 것이 CNN 모델(Convolutional Neural Network).
- 기본 구성요소: 합성곱(convolution) 필터
- 깊은 층을 통해 이미지의 특징을 고도화된 필터로 추출함.
- 초기의 CNN 모델인 AlexNet은 8개의 층으로 구성되어 2012년 이미지넷 대회에서 우승.
- 이후에는 ResNet, SENet 등의 더 깊은 모델이 등장.
- 전이 학습(Transfer Learning): 사전 학습된 모델의 가중치를 재사용하여 적은 데이터로도 학습 가능.

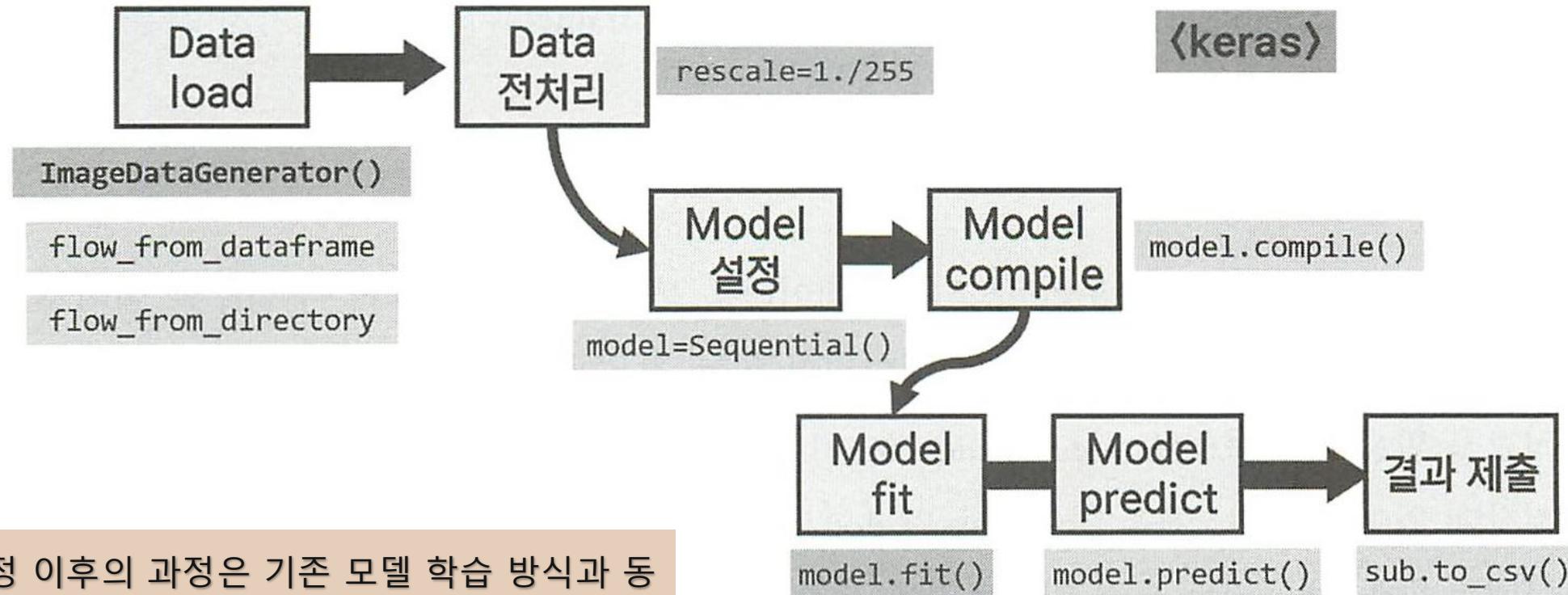
CNN 모델의 구성요소

- 합성곱 층(Convolution Layer): 특징 추출
- 풀링 층(Pooling Layer): 공간 정보를 요약하여 특징 강화
- 완전 연결 층(FC Layer): 최종 결과값 계산

ILSVRC 대회

- CNN은 ILSVRC 대회에서 우수한 성능을 보임
- 대표적인 모델: AlexNet(2012), VGG(2014), GoogLeNet(2014), ResNet(2015)
- 이 구조를 전이학습(transfer learning) 방식으로 활용하여 적은 데이터에도 효과적으로 적용 가능.

이미지 데이터 학습 과정



이미지 데이터 불러오기

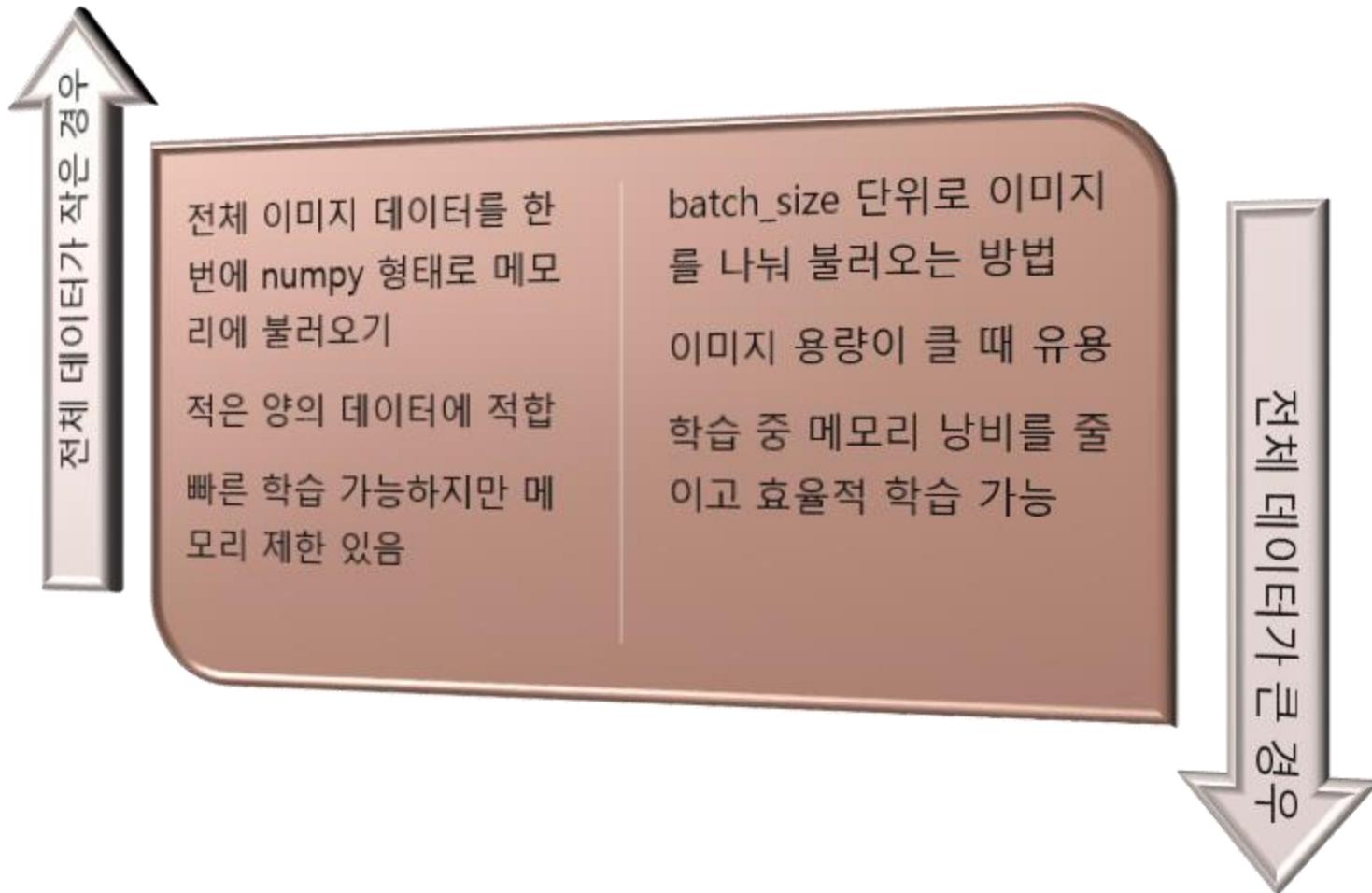


IMAGE DATA GENERATOR 클래스

✓ 기능:

- 원본 이미지를 불러오고,
- 크기 조정, 색상 변화, 회전, 확대/축소 등 데이터 증강도 함께 처리 가능
- 단순 불러오기만 할 경우 rescale=1./255만 추가

✓ 사용 예시 코드:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
datagen = ImageDataGenerator(rescale=1./255)
```

IMAGE DATA GENERATOR 클래스

✓ flow_from_directory

- 이름에서 알 수 있듯 디렉토리 구조에서 데이터를 불러오는 함수이다. 이 함수는 이미지 데이터가 이미 class별로 디렉토리 형태로 정리되어 있는 경우에 사용이 가능하다. 레이블은 디렉토리 정보를 활용해 자동으로 one-hot encoding된다. 각각의 이미지들이 전체 데이터 디렉토리 아래에 저장돼 있어야 한다. 예를 들어 class가 dog/cat/frog라면 강아지는 dog 디렉토리에, 고양이는 cat 디렉토리에, 개구리는 frog 디렉토리에 저장돼 있어야 한다. `flow_from_directory()`를 실행하면 각각의 디렉토리가 하나의 레이블로 one-hot encoding된다. 즉, dog 디렉토리에 있는 이미지의 경우 레이블은 [1,0,0]이 된다.

✓ flow_from_dataframe:

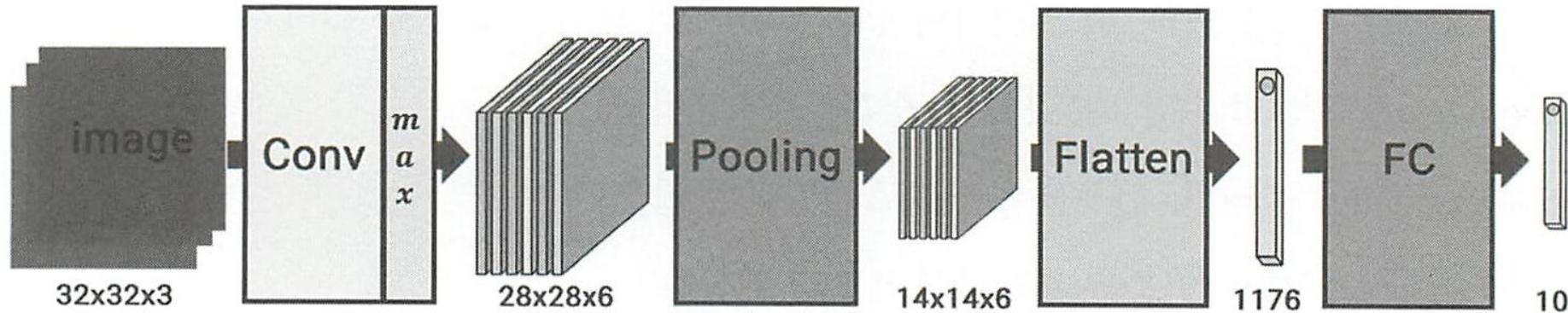
- 이 함수는 이미지 파일 경로와 이미지 레이블이 DataFrame 파일에 정리되어 있는 경우에 사용이 가능하다. DataFrame 파일에서 이미지 파일의 경로와 레이블을 불러와 `batch_size`별로 데이터를 출력한다.

데이터 증강

✓ 데이터 증강 관련 인수

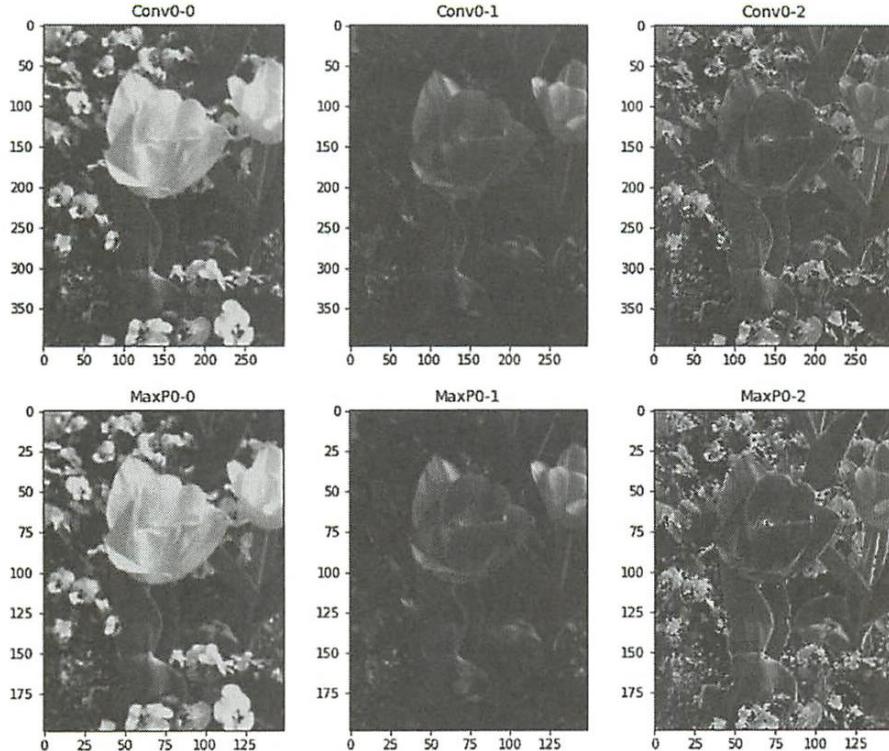
인수	설명
rotation_range	정수, 무작위 회전 각도 범위
width_shift_range	부동소수점, ID 형태의 유사배열
height_shift_range	부동소수점, ID 형태의 유사배열 혹은 정수
brightness_range	두 부동소주점 값으로 이루어진 리스트 혹은 튜플, 밝기 조절할 값의 범위
shear_range	부동소수점, 층 밀리기의 강도
zoom_range	부동소수점 혹은 [하한, 상한]. 무작위 줌의 범위
horizontal_flip	Boolean. Input을 무작위로 가로로 뒤집음.
vertical_flip	Boolean. Input을 무작위로 상하 반전

CNN 모델의 구성



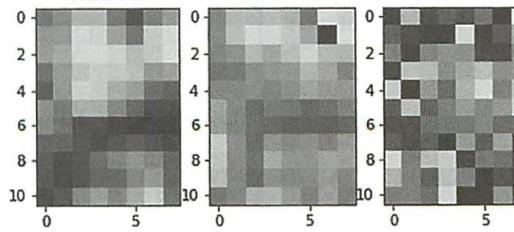
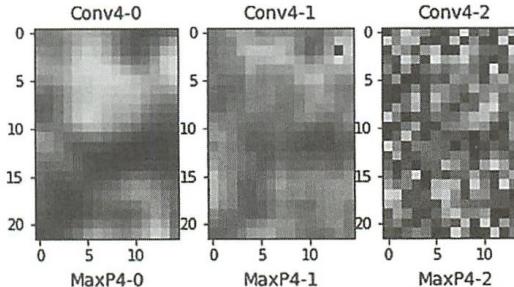
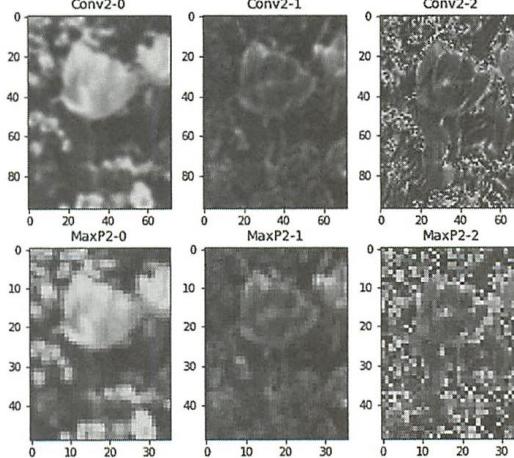
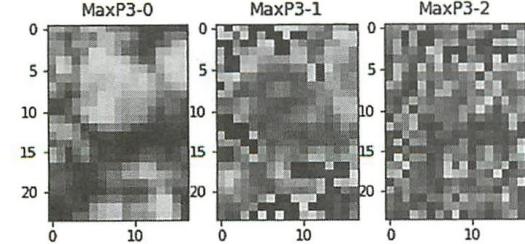
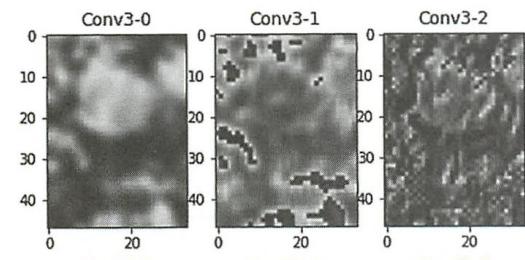
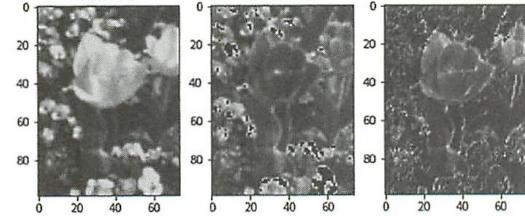
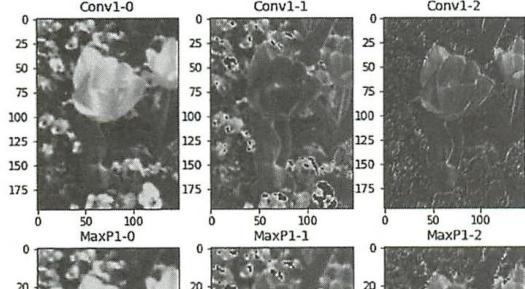
- 처음 입력은 $32 \times 32 \times 3$ 의 3차원 데이터이다. 한 번의 합성곱 층을 거쳐 $28 \times 28 \times 6$ 인 크기의 3차원 데이터가 되었다. 이 CNN Layer에는 합성곱이 6번(6개의 필터) 사용된 것을 알 수 있다. 다음으로 Pooling layer를 거쳐 $14 \times 14 \times 6$ 인 크기의 데이터가 되었다. 이것을 벡터화(Flatten)함으로써 1,176 크기의 1차원 벡터가 된다. 마지막으로 FC layer를 거쳐 최종적으로 class 개수에 맞도록 10개의 값을 갖는 출력층이 만들어진다.

CNN 특징



- 입력 데이터의 형태는 $300 \times 400 \times 3$ 인 컬러 이미지이고 각 층은 Conv2D와 MaxPooling을 한번씩 진행.
- 진행 과정은 데이터가 입력되면 각 3개의 Convolution filter로 convolution 연산해 feature map을 만들고 activation function(ReLU)을 적용해 activation map이 만들어진다. 이 결과가 그림에 Conv0-0 등으로 이름 붙여진 부분이다.
- 다음은 activation map을 MaxPooling 층에 입력해 pooling한다. 그러면 결과적으로 3개의 출력 map이 만들어진다. 이 결과가 그림의 MaxP0-0 등으로 이름 붙여진 부분이다.

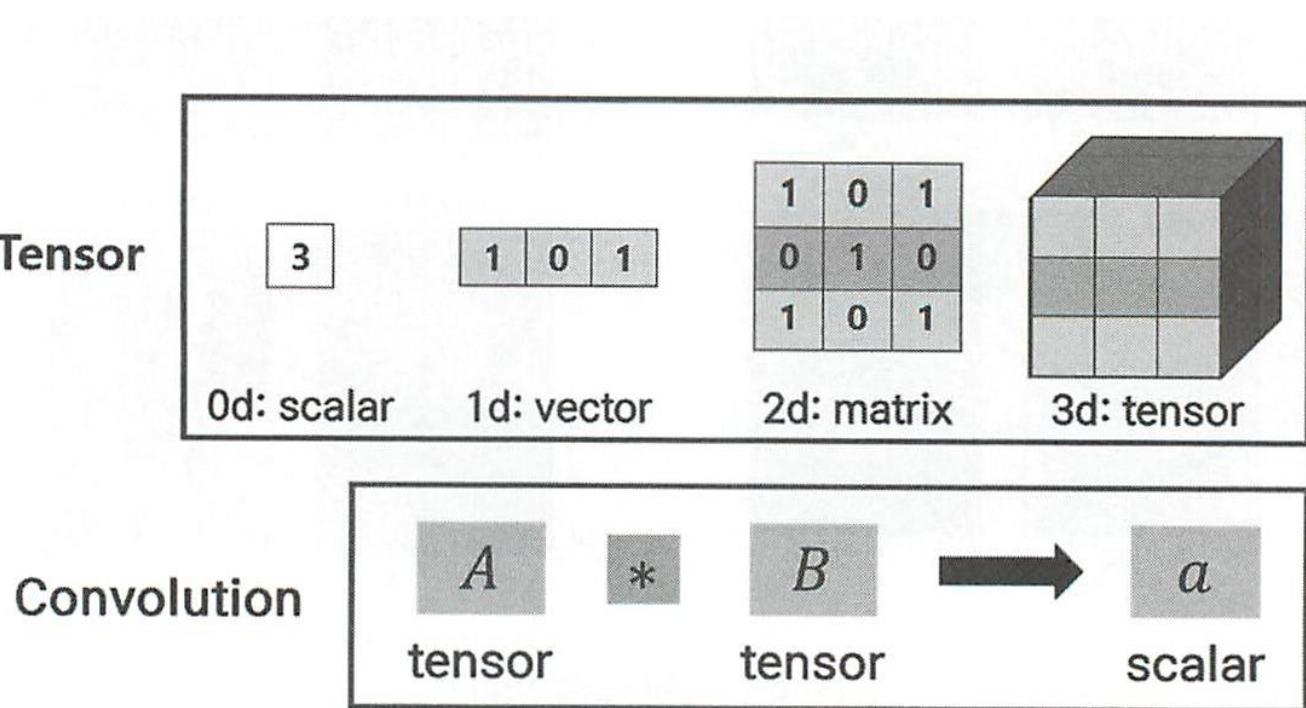
CNN 특징



- Convolution filter는 1, 2 번은 값을 지정 해서 사용했고, 3번은 랜덤값으로 생성해 진행.

CNN 구성 요소

- ✓ Convolution Layer 연산
- 합성곱은 텐서(tensor)와 텐서 사이에서 정의되는 연산이다.
- 텐서는 차원에 따라 0차원은 scalar, 1차원은 벡터, 2차원은 행렬, 3차원은 3차원 행렬(텐서)라고 부른다.
- 4차원 텐서의 경우 4차원 벡터처럼 수식으로만 표현되며, 보통 3차원 텐서(또는 이미지)가 여러 개 모여 있다는 의미가 된다.



CNN 구성 요소

✓ 합성곱 계산

- 차원의 크기가 같은 두 텐서를 계산해 scalar 값이 되는 연산.

1D Convolution

$$\begin{bmatrix} 1 & 0 & 2 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot 2 & 0 \cdot 3 & 2 \cdot 1 \end{bmatrix} \rightarrow 2 + 0 + 2 = 4$$

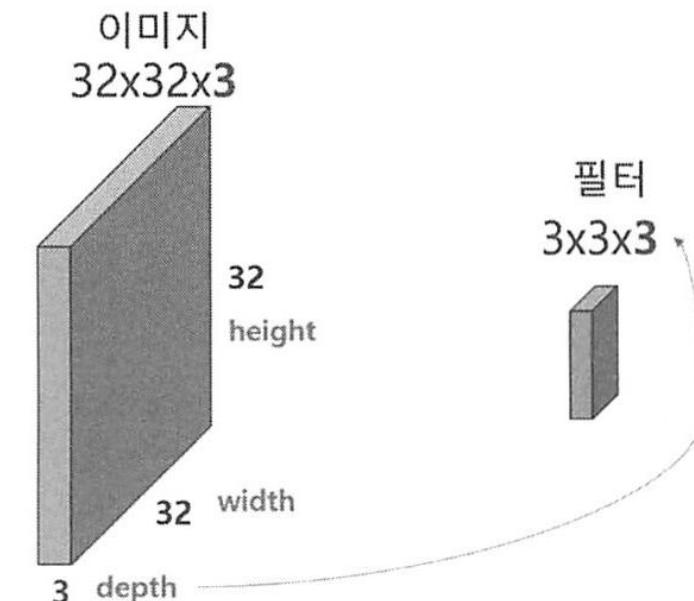
2D Convolution

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \cdot 0 & 0 \cdot 1 & 1 \cdot 1 \\ 0 \cdot 0 & 1 \cdot 1 & 0 \cdot 1 \\ 1 \cdot 0 & 0 \cdot 0 & 1 \cdot 1 \end{bmatrix} \rightarrow 0 + 0 + 1 + 0 + 1 + 0 + 0 + 0 + 1 = 3$$

CNN 구성 요소

✓ Filter

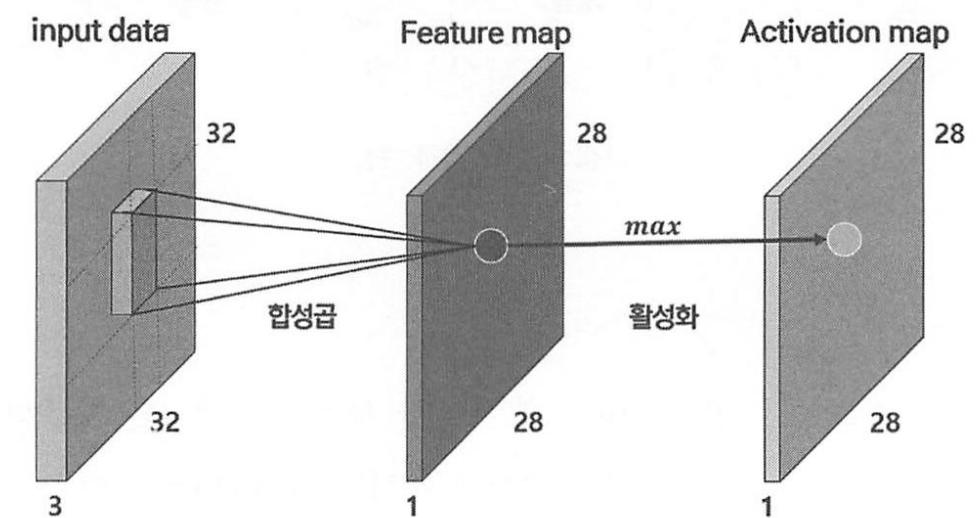
- CNN 모델에서 합성곱 연산 과정은 입력 이미지에 특정한 사이즈의 텐서를 사용해 전체 이미지의 일부를 스캔하듯이 이동하면서 연산을 한다.
- 이때 스캔하는 텐서를 필터(filter)라 하며 보통 3×3 크기의 필터를 많이 사용한다.
- 필터의 두께는 자동으로 입력 데이터(텐서)의 두께로 설정된다.
- 입력 데이터가 컬러 이미지인 경우 그 두께는 3차원(예: $32 \times 32 \times 3$)이고 필터의 두께도 자동적으로 3이 된다. 만약 입력 텐서가 $28 \times 28 \times 6$ 이면 필터의 두께는 6이 된다.



CNN 구성 요소

✓ Feature Map

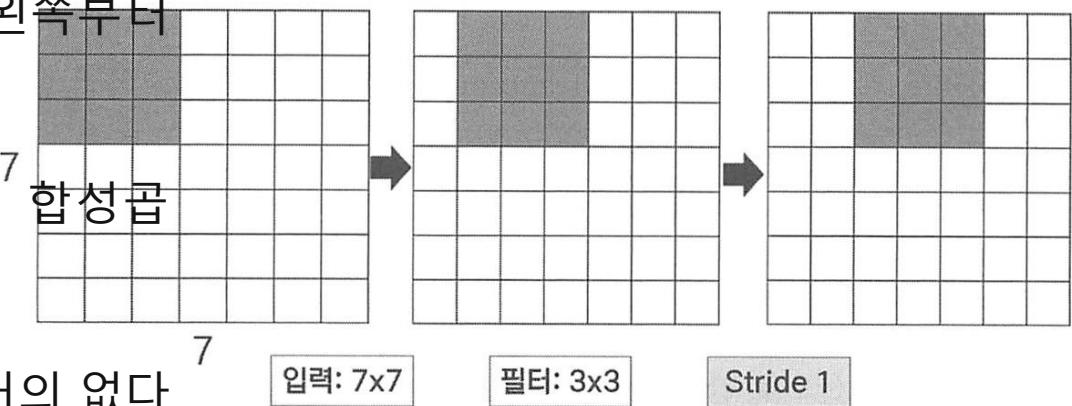
- 입력 데이터에 필터로 스캔한 결과로 만들어지는 출력 텐서를 feature map(특성 맵)이라 한다.
- 보통 하나의 합성곱 층에서 여러 개의 필터가 사용되며, 그 결과로 필터 수만큼의 특성 맵이 만들어진다.
- 합성곱 연산 후에 활성화 함수를 적용하는데 이것은 특성 맵이 만들어진 후에 작용한다. 특성 맵에 활성화 함수를 작용시켜 만들어진 결과를 activation map이라고도 한다.
- CNN의 다음 층에서는 계산된 activation map들을 모아 하나의 텐서로 만들어서 새로운 입력으로 사용한다. 예를 들어 activation map의 크기가 28×28 이고 작용한 필터의 개수가 5개였다면, 다음 입력 데이터는 $28 \times 28 \times 5$ 의 크기가 된다. 여기서 activation map과 특성 맵은 같은 크기이다.
- 그림을 살펴보면, 입력 이미지가 $32 \times 32 \times 3$ 이고 하나의 필터를 통해 $28 \times 28 \times 1$ 크기를 갖는 하나의 특성 맵이 만들어지며, 여기에 activation function(예: ReLU)이 작용해 같은 크기의 activation map이 만들어진다.



CNN 구성 요소

✓ Stride

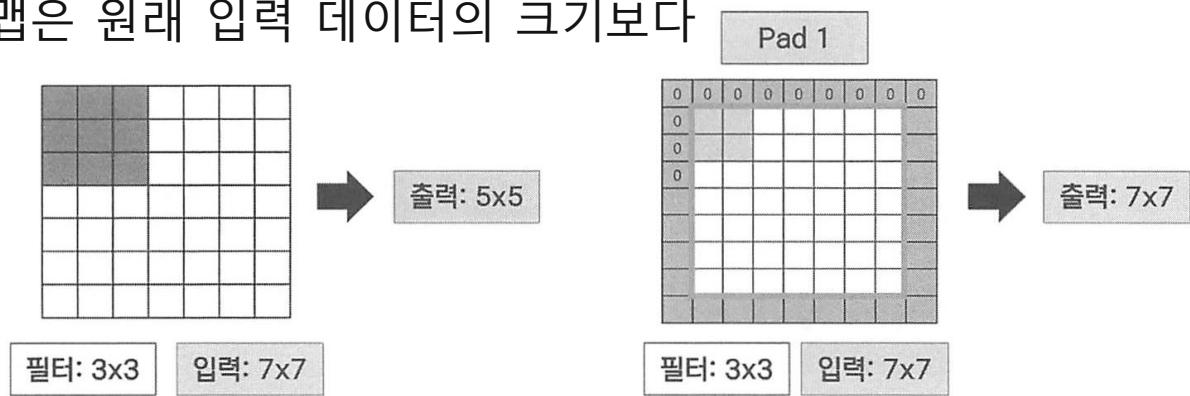
- CNN 과정에서 합성곱 연산을 할 때 필터가 움직이는 간격을 stride라고 한다.
- 합성곱이 2차원(3×3)으로 정의되므로 stride도 2차원 (m, n)으로 정의된다.
- 즉 한 번 합성곱 연산을 한 후에 우측으로 m 만큼씩 이동해 입력 텐서의 끝까지 이동한 후, 아래로 n 만큼씩 움직여서 맨 왼쪽부터 다시 스캔하는 방식이다.
- 보통 stride는 $(1, 1)$ 로 설정하는데, 한 칸씩 움직이면서 합성곱 연산을 한다.
- 때때로 $(2, 2)$ 를 사용하기도 하며, $(3, 3)$ 을 사용하는 경우는 거의 없다.



CNN 구성 요소

✓ padding

- 입력 데이터에 합성곱 연산을 통해 만들어진 특성 맵은 원래 입력 데이터의 크기보다 작아지게 된다.



- 만약 입력 이미지의 크기가 작은 경우라면 몇 번의 합성곱을 실행하면 데이터의 크기가 작아져 더 이상 합성곱 층을 추가하는 것이 불가능하다. 그래서 합성곱 계산을 할 때 padding을 덧대어 출력 크기를 입력 데이터의 크기와 같게 만들 수 있다.
- 합성곱 층에서 인수 설정을 `padding=same`으로 사용하면 된다.
- 그림처럼 `pad`를 덧댐으로써 입력 텐서와 출력 텐서의 크기를 같게 할 수 있다.

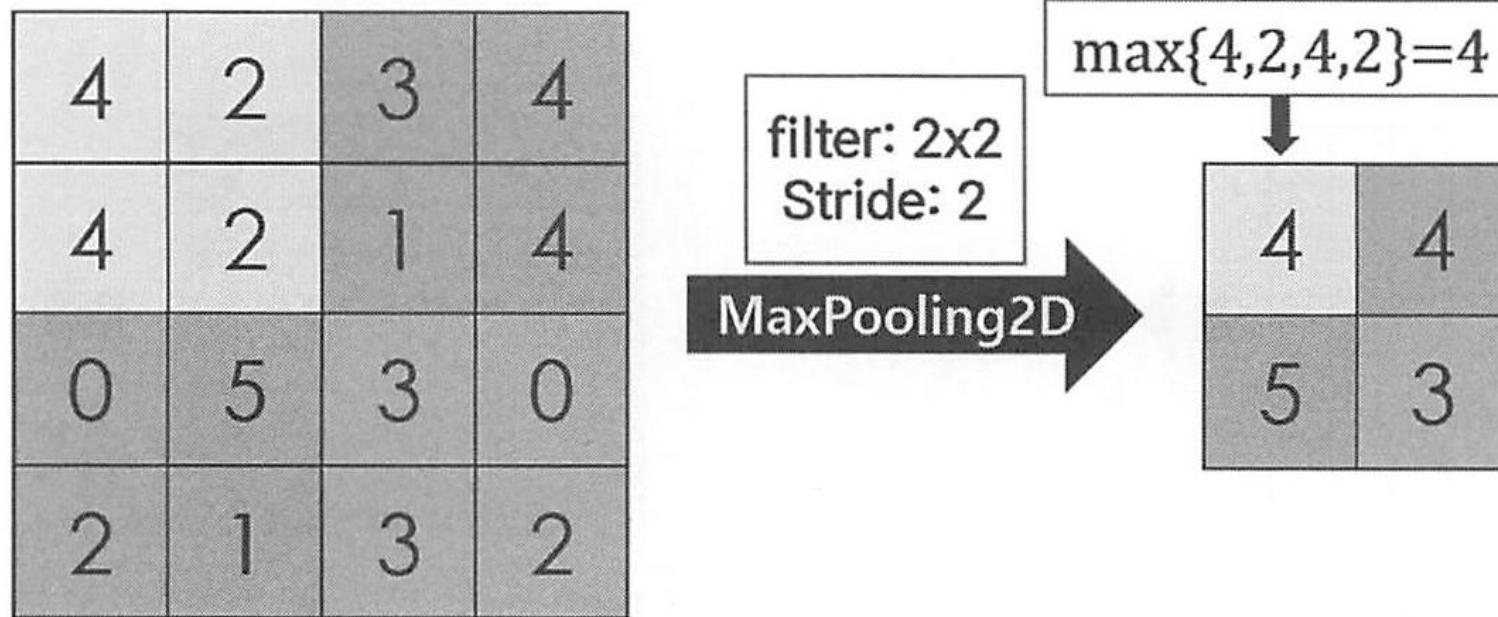
CNN 구성 요소

✓ 활성화 (Activation)

- 합성곱 연산 후, 신경망처럼 활성화 함수(ReLU 등)를 사용하여 비선형성을 부여함.
- 일반적으로 ReLU 또는 ReLU 변형 함수를 사용.
- 이러한 처리 과정은 기존 신경망과 동일함.
- 합성곱 층 개수는 설정에 따라 달라짐 (1개만 사용하거나 2~3개 사용할 수도 있음).
- 각 합성곱 층 뒤에 Pooling Layer를 붙이기도 함.
- 여러 유명한 신경망 구조(Architecture)를 참고하면 더 좋은 모델을 설계하는 데 도움됨. (예: VGG, ResNet 등)

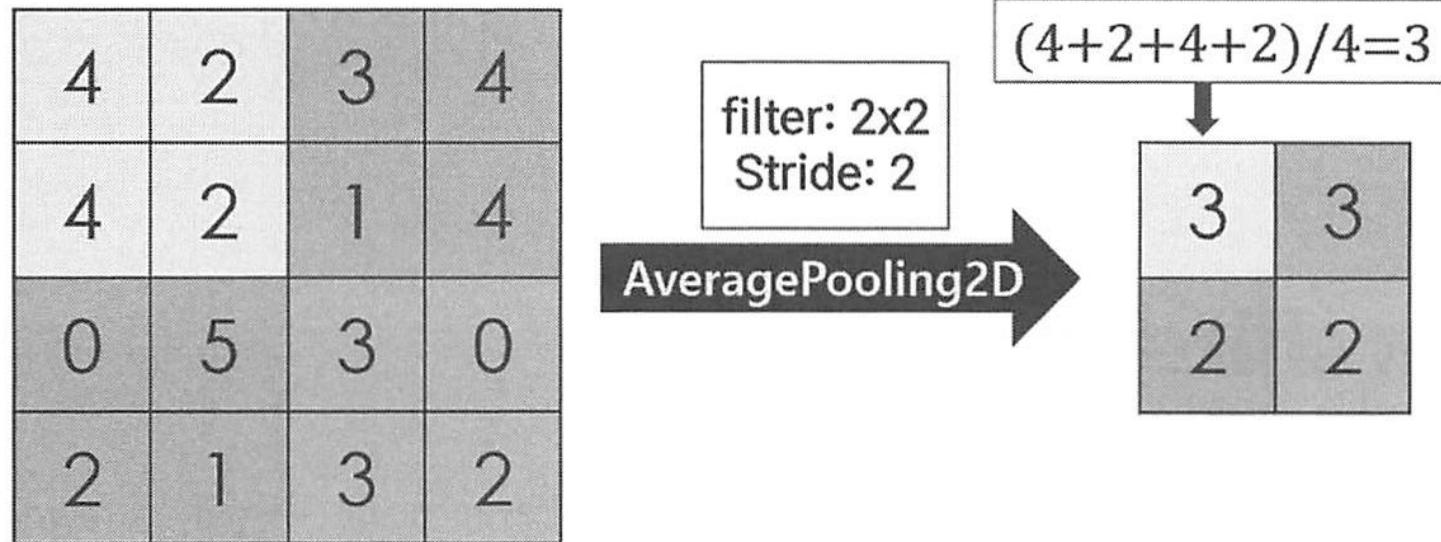
CNN 구성 요소 (POOLING LAYER)

- ✓ MaxPooling2D



CNN 구성 요소 (POOLING LAYER)

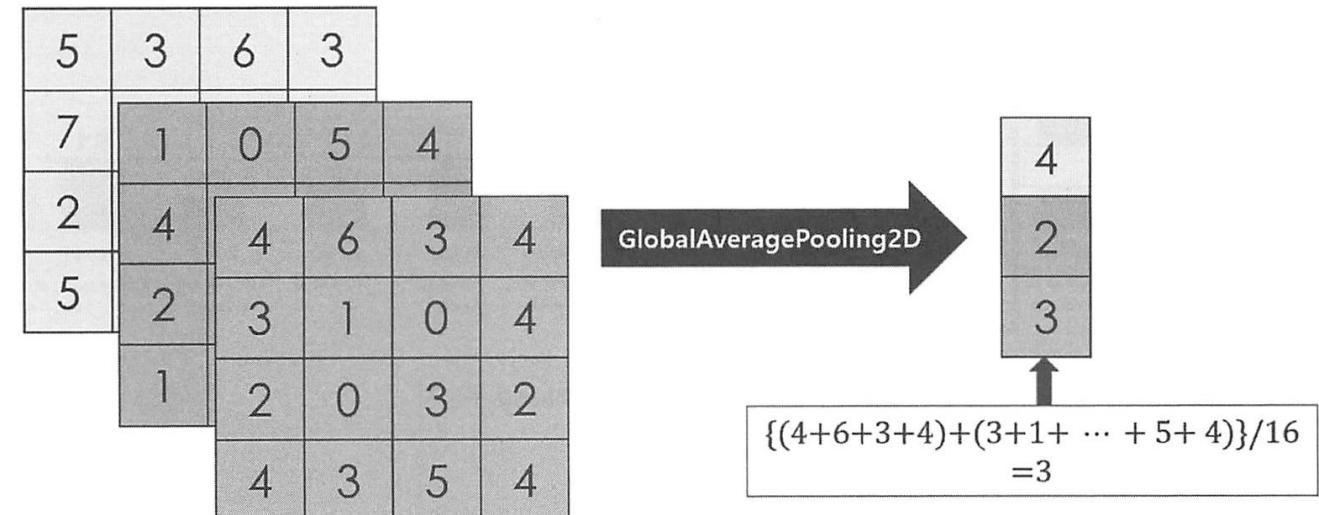
- ✓ AveragePooling2D



CNN 구성 요소 (POOLING LAYER)

✓ GlobalAveragePooling2D

- 합성곱 연산으로 만들어진 하나의 특성 맵에서 평균값을 출력하는 Pooling.
- 이전의 Pooling 계산보다 크기를 많이 줄이게 된다.
- GoogLeNet에서 FC Layer 직전에 Flatten 대신 사용함.

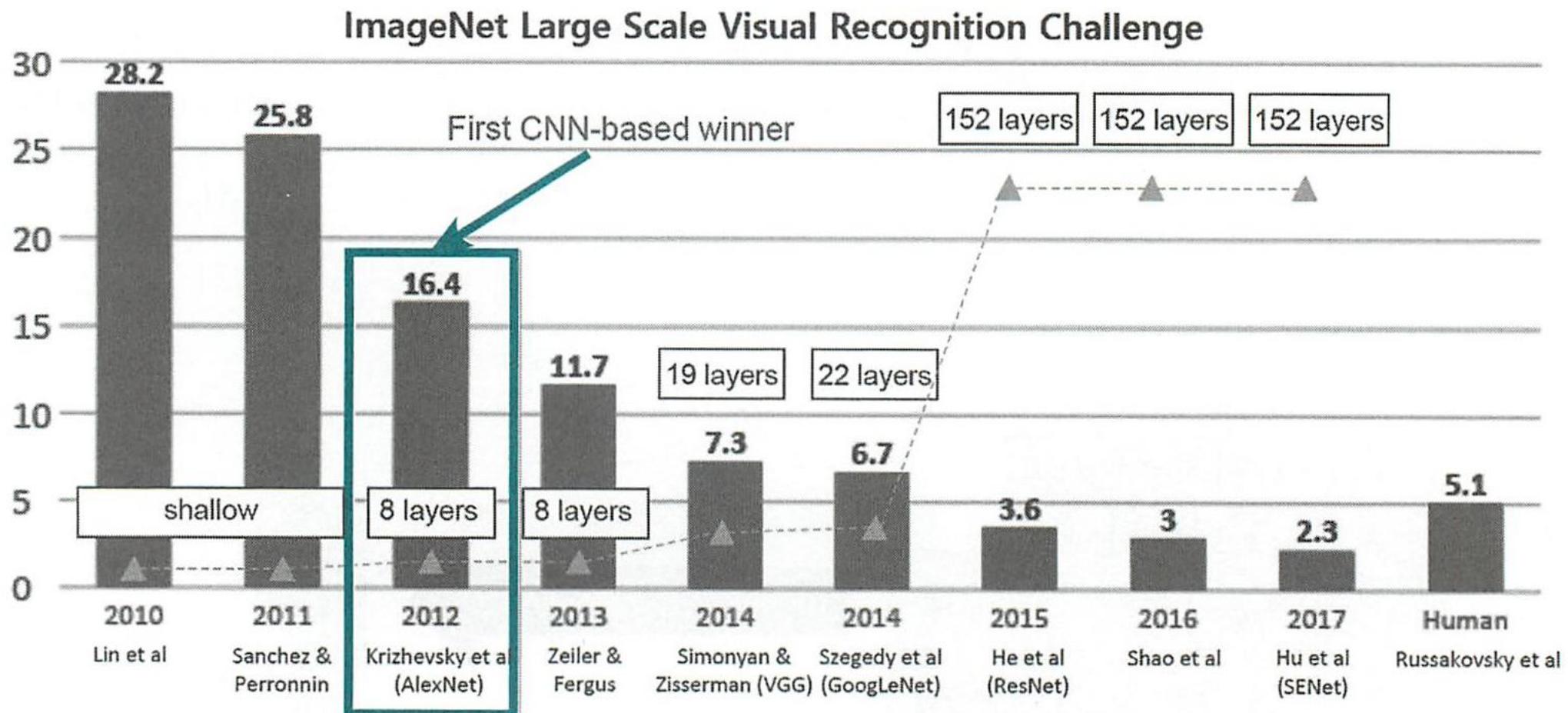


CNN 구성 요소 (FULLY CONNECTED LAYER)

✓ Fully Connected Layer

- Fully Connected Networks는 신경망의 다른 이름
- FC Layer는 합성곱 층과 Pooling Layer로 처리된 결과를 Flatten으로 벡터화한 후 최종적으로 딥러닝의 목적에 맞도록 출력 크기를 맞춰주는 과정이다.
- 이전까지 처리된 데이터는 텐서로 출력되므로 Flatten을 통해 벡터로 만들어줘야 한다. 그런 다음 최종 출력 크기(class 종류)와 맞도록 적절하게 신경망을 연결해준다.
- 이 과정은 신경망 모델과 동일한데, CNN 과정으로 전달되어 온 데이터(텐서)를 Flatten으로 벡터화해 다시 신경망 모델로 학습시키는 것과 같다.

CNN ARCHITECTURES



CNN ARCHITECTURES

✓ AlexNet

- GPU 2개를 이용한 병렬 구조 / TOP-5 인식 오류율을 약 20%에서 16.7%로 획기적으로 낮춤

Convolution

1층	11x11x3	96
2층	5x5x48	256
3층	3x3x256	384
4층	3x3x192	384
5층	3x3x192	256

FC

$6 \times 6 \times 256$ 특성맵 flatten
 $6 \times 6 \times 256 = 9216$ 차원의 벡터

6층	9216
7층	4096
8층	1000 Output

결과 : 1000개 클래스 분류

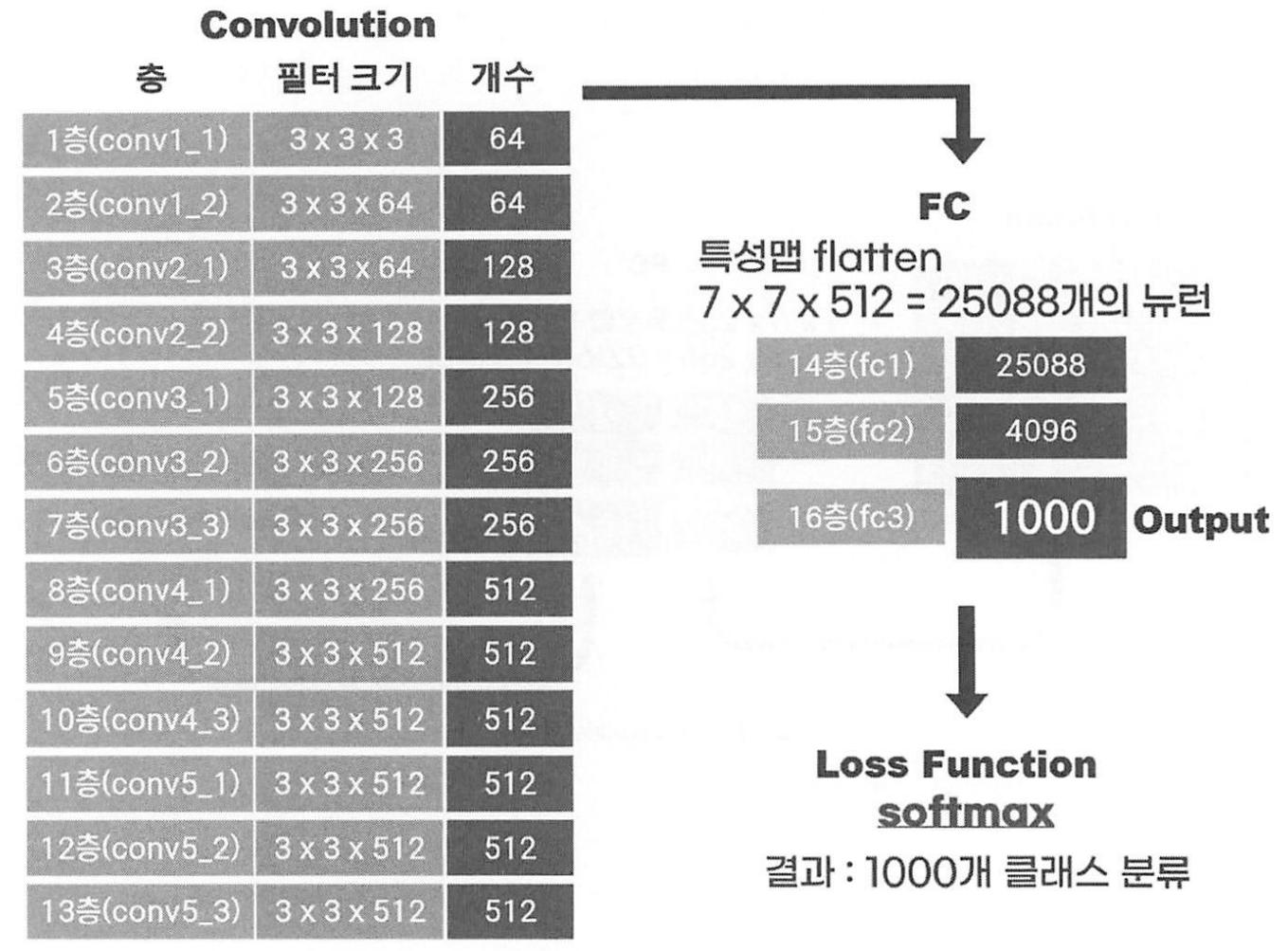
Loss Function
softmax



CNN ARCHITECTURES

✓ VGGNet

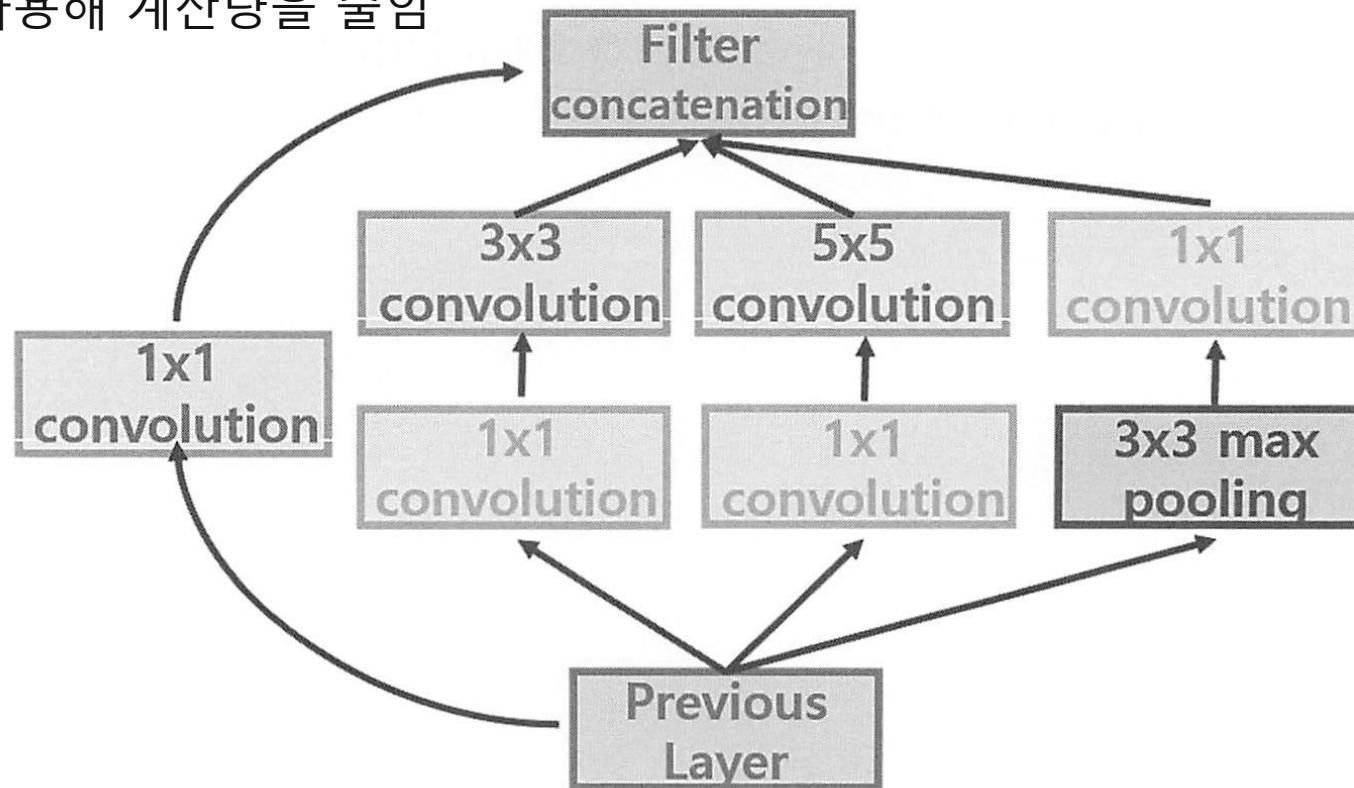
- VGG16 (16층)
- VGG19 (19층)
- CNN Layer에서 합성곱 필터 크기는 3×3 을 사용했으며, MaxPooling은 2×2 크기로 stride (2, 2)를 사용했다.
- 3×3 필터를 사용하는 것은 매우 효율적인 선택이다.
- 3×3 필터를 두 번 사용하는 것이 5×5 필터 하나를 사용했을 때와 같은 크기의 특성 맵을 만든다.
- 하지만 3×3 크기의 필터를 두 번 사용하는 것은 가중치가 $(9+1) \cdot 2 = 20$ 개인 반면 5×5 크기 필터는 $5 \cdot 5 + 1 = 26$ 개의 가중치를 가지고 있으므로 3×3 크기를 두 번 사용하는 것이 parameter 개수가 적어 계산량을 줄여주므로 더 효율적이다.



CNN ARCHITECTURES

✓ GooLeNet

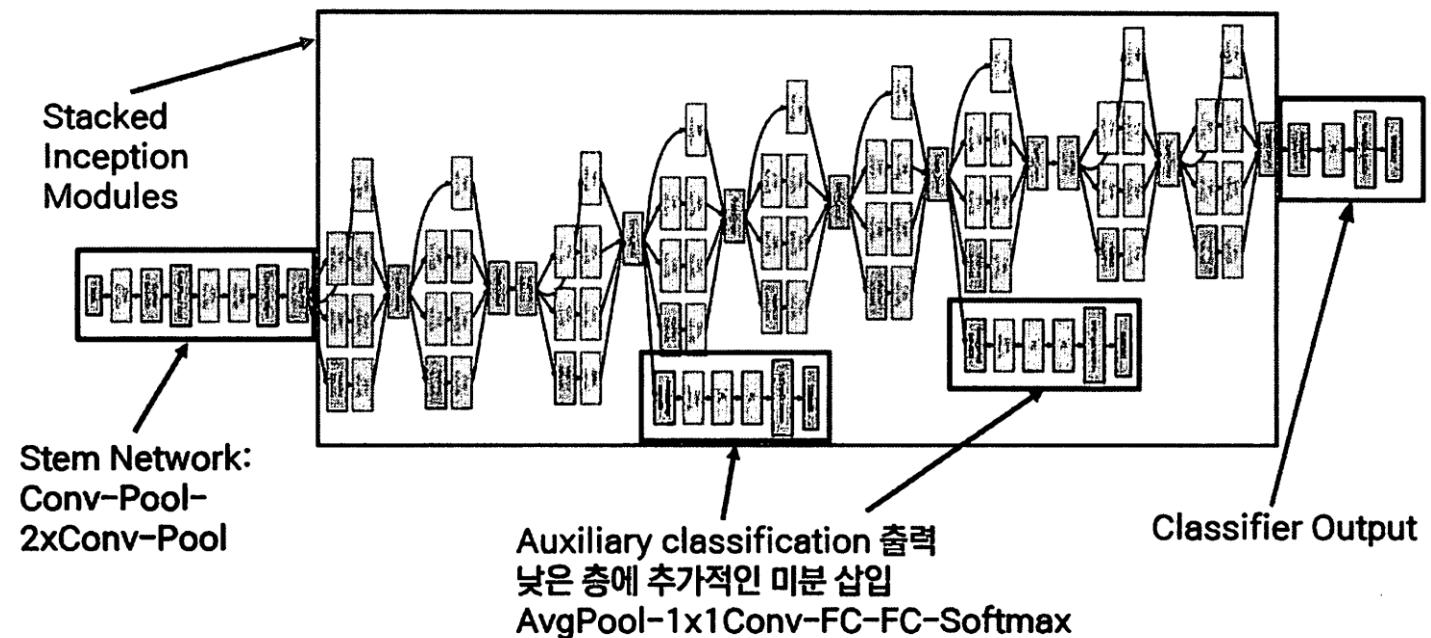
- Inception 모듈을 기본 단위로 하는 모듈을 연결해 전체 모델을 만들었음.
- 1×1 합성곱을 사용해 계산량을 줄임



CNN ARCHITECTURES

✓ GooLeNet

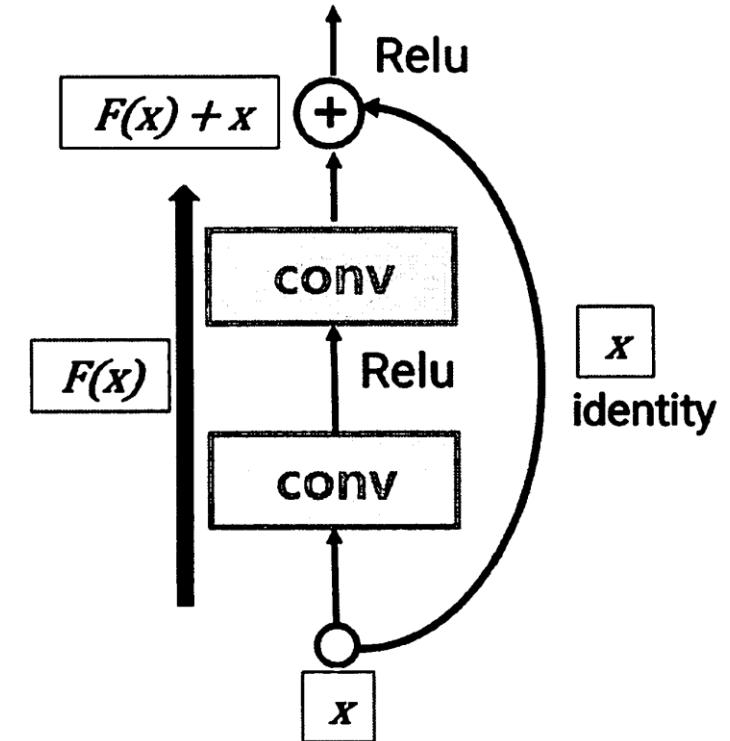
- GAP(Global average pooling)을 사용해 flatten과정을 대체
- 층이 깊어지면서 발생하기 쉬운 미분 소실 문제를 해결하기 위해 Auxiliary Classifier를 사용



CNN ARCHITECTURES

✓ ResNet

- 2015년 경진대회에서 우승한 Architecture로 152개의 매우 깊은 구조임.
- 사람의 정확도로 알려진 오류율 5%를 처음으로 추월.
- ResNet은 VGG19를 토대로 CNN 층을 추가해 층을 깊게 한 후 **Residual connection**을 만들었으므로써 전체 구조를 완성했다.
- ResNet의 가장 큰 특징은 **Residual Block**으로, 이는 층이 깊어질수록 발생하기 쉬운 미분 소실을 해결하는 데 효과적이다.
- ResNet의 이름은 이 Residual에서 따온 것이다.



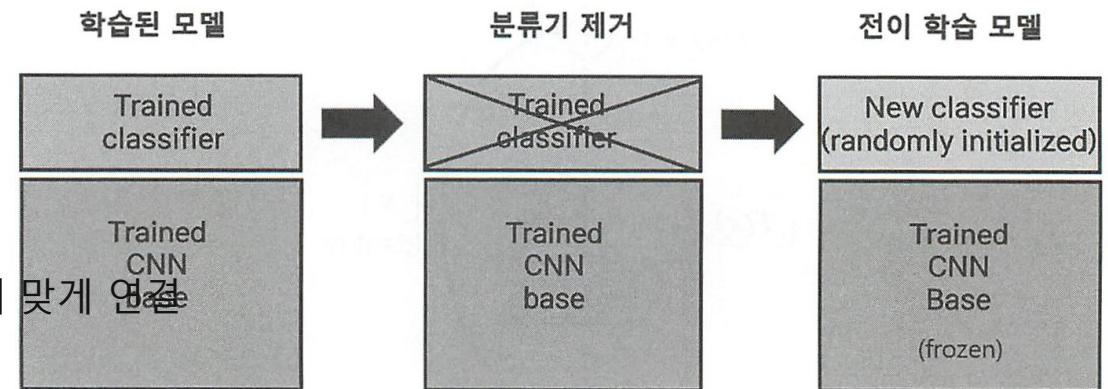
TRANSFER LEARNING

✓ 전이학습

- 전이 학습은 이미 학습된 모델을 기반으로 출력층만 변경해 새로운 데이터를 학습시키는 방식
- 처음부터 학습시키는 것보다 데이터와 시간 절약, 좋은 성능을 기대할 수 있음

✓ 전이 학습 방법

- 특성 추출 (Feature Extraction)
 - 기존 모델의 가중치를 고정하고 출력층만 새롭게 학습
 - CNN 기반의 사전학습된 모델에서 분류기만 제거하고 새 데이터에 맞게 연결
- 미세조정 (Fine-tuning)
 - 기존 모델의 일부 층까지 고정하고, 일부 층 또는 전부를 재학습
 - 새 데이터 양이 많을수록 미세조정 범위를 넓히고, 적을 경우 출력층만 조정 가능
 - 적은 데이터로도 좋은 성능 가능



실습

- ✓ CNN Layer 구현

```
import numpy as np  
from PIL import Image  
import matplotlib.pyplot as plt
```

합성곱 함수 구현

```
def conv(a, b):  
    c = np.array(a)*np.array(b)  
    return np.sum(c)
```

실습

✓ CNN Layer 구현: MaxPooling 함수 구현(한 개의 map 계산)

```
def MaxPooling(nimg): # 2d input
    nimg = np.array(nimg)
    i0, j0 = nimg.shape # i0 = nimg.shape[0], j0 = nimg.shape[1]
    i1 = int((i0+1)/2)
    j1 = int((j0+1)/2)
    output = np.zeros((i1, j1))

    if i0%2 ==1:
        i0+=1
        tmp = np.zeros((1, j0))
        nimg = np.concatenate([nimg, tmp], axis = 0)

    if j0%2 ==1:
        j0+=1
        tmp = np.zeros((i0, 1))
        nimg = np.concatenate([nimg, tmp], axis = 1)

    for i in range(output.shape[0]):
        for j in range(output.shape[1]):
```

```
a = np.array(nimg[2*i:2*i+2,2*j:2*j+2])
output[i, j] = a.max()
return output
```

실습

✓ CNN Layer 구현

합성곱 출력 층(feature map) 함수 구현(한 개의 filter 계산)

```
def featuring(nimg, filters):  
    feature = np.zeros((nimg.shape[0]-2, nimg.shape[1]-2))  
    for i in range(feature.shape[0]):  
        for j in range(feature.shape[1]):  
            a = nimg[i:i+3,j:j+3,:]  
            feature[i,j] = conv(a, filters)  
  
    return feature
```

실습

- ✓ CNN Layer 구현

MaxPooling 출력 층 합수 구현(여러 map 계산)

```
def Pooling(nimg):  
    nimg = np.array(nimg)  
    pool0 = []  
    for i in range(len(nimg)):  
        pool0.append(MaxPooling(nimg[i]))  
    return pool0
```

실습

✓ CNN Layer 구현

배열을 그림으로 변환

```
def to_img(nimg):
    nimg = np.array(nimg)
    nimg = np.uint8(np.round(nimg))
    fimg = []
    for i in range(len(nimg)):
        fimg.append(Image.fromarray(nimg[i]))
    return fimg
```

실습

- ✓ CNN Layer 구현

feature map 생성(여러 filter 계산)

```
def ConvD(nimg, filters):  
    return [featuring(nimg, f) for f in filters]
```

실습

✓ CNN Layer 구현

ReLU 활성화 함수

```
def ReLU(f0):
    f0 = np.array(f0)
    f0 = (f0>0)*f0
    return f0
```

CNN Layer 함수 : Conv+ReLU+MaxPooling

```
def ConvMax(nimg):
    nimg = np.array(nimg)
    f0 = ConvD(nimg)
    f0 = ReLU(f0)
    fg = Pooling(f0)
    return f0, fg
```

실습

✓ CNN Layer 구현

그림 그리기 : 합성곱 후의 상태와 MaxPooling 후의 상태를 그림으로 그리기

```
def draw(f0, fg0, size = (12, 8), k = -1):    # size와 k는 기본값 설정  
    plt.figure(figsize = size)  
  
    for i in range(len(f0)):  
        plt.subplot(2, len(f0), i+1)  
        plt.gca().set_title('Conv'+str(k)+ '-' + str(i))  
        plt.imshow(f0[i])
```

```
for i in range(len(fg0)):  
    plt.subplot(2, len(fg0), len(f0)+i+1)  
    plt.gca().set_title('MaxP'+str(k)+ '-' + str(i))  
    plt.imshow(fg0[i])  
  
if k != -1:    # k=-10 | 아니면 그림을 저장  
    plt.savefig('conv'+str(k)+'.png')
```

실습

✓ CNN Layer 구현

3개의 activation map 합치기 : MaxPooling 후의 결과 map들을 하나의 데이터로 통합

```
def join(mm):
    mm = np.array(mm)
    m1 = np.zeros((mm.shape[1], mm.shape[2], mm.shape[0]))
    for i in range(mm.shape[1]):
        for j in range(mm.shape[2]):
            for k in range(mm.shape[0]):
                m1[i][j][k] = mm[k][i][j]
    return m1
```

실습

✓ CNN Layer 구현

CNN Layer 과정을 계산하고 결과를 그림으로 출력

```
def ConvDraw(p0, filters, size=(12,8), k=-1):
    f0, fg0 = ConvMax(p0, filters)
    f0 = to_img(f0)
    fg1 = to_img(fg0)
    draw(f0, fg1, size, k)
    p1 = join(fg0)
    return p1
```

```
nimg31 = np.random.rand(10,10) # 10x10 이미지
filters = [np.ones((3,3))] * 3 # 필터 3개

# 실행

m0 = ConvDraw(nimg31, filters, (12, 10), 0)
```