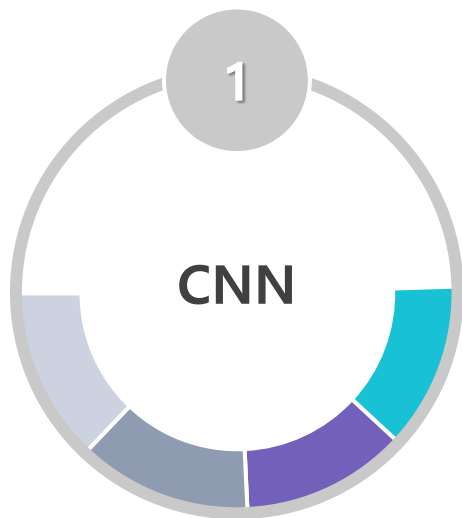


2학기 4주차 진도세션

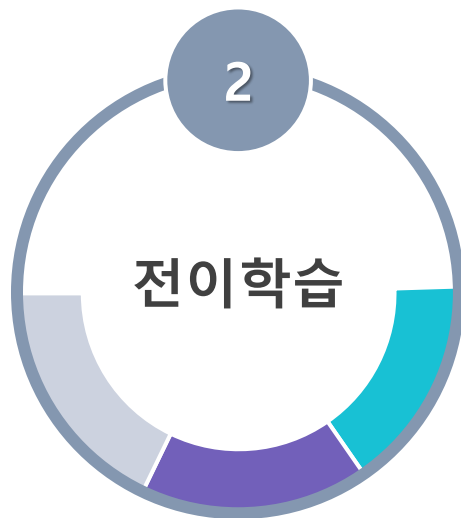
4조 노지예 임청수 한세림

# CNN & 전이학습

Convolution Neural Network & Transfer Learning



- CNN 소개
- CNN 필요성
- CNN 구조
- Baseline Code



- 전이학습 소개
- 특성추출
- 미세조정



- LeNet
- AlexNet
- VGGNet
- ResNet

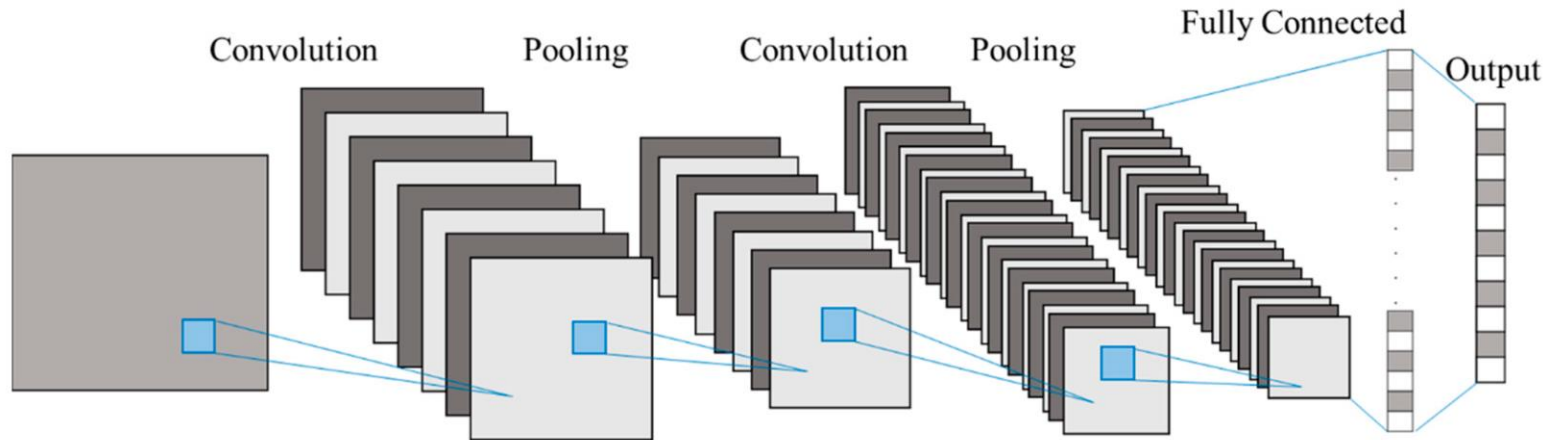


1

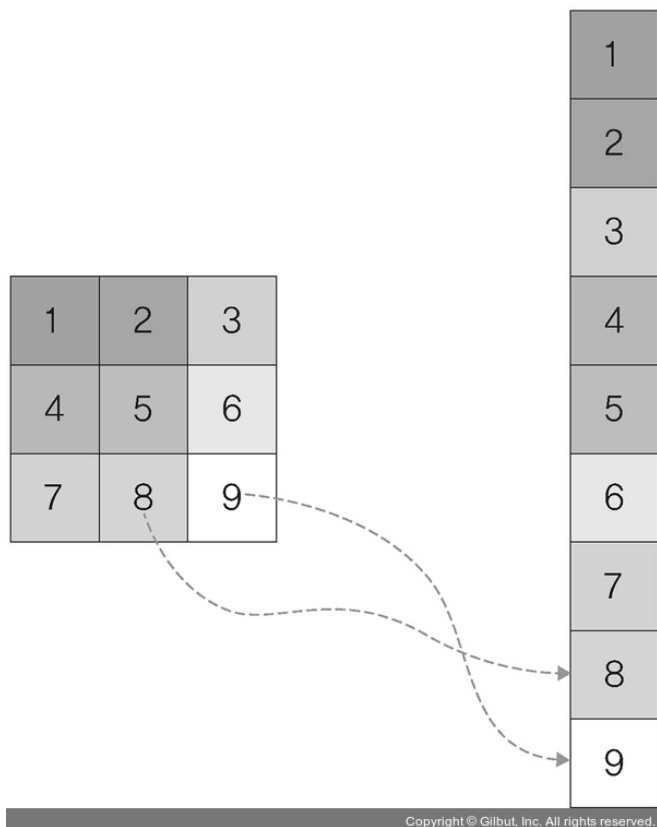
CNN

## CNN (Convolutional Neural Network; 합성곱 신경망)

- **합성곱층**(Convolution layer)과 **풀링층**(Pooling layer)을 포함하는 **이미지 처리에 특화된** 인공 신경망 알고리즘
- 영상 및 사진이 포함된 이미지 데이터에서 객체를 탐색하거나 객체 위치를 찾아내는 데 유용한 신경망



## 기존 신경망의 문제점



### 3x3 흑백 이미지

- 3x3 이미지 배열을 flattening → 1차원 벡터(9개의 feature)
- 각 픽셀에 가중치를 곱하는 연산

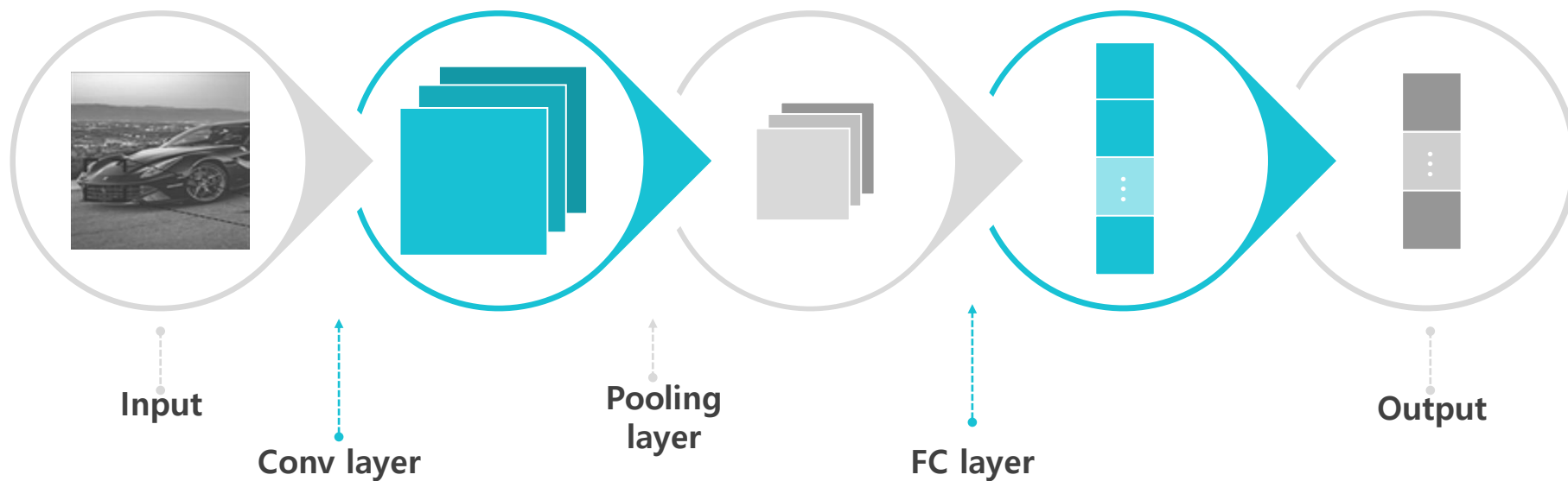
→ 이미지의 **공간적 구조**를 고려하지 않음

*ex. 좌상단은 어둡고 우하단은 밝음*



### 합성곱층 (Convolution Layer)의 등장

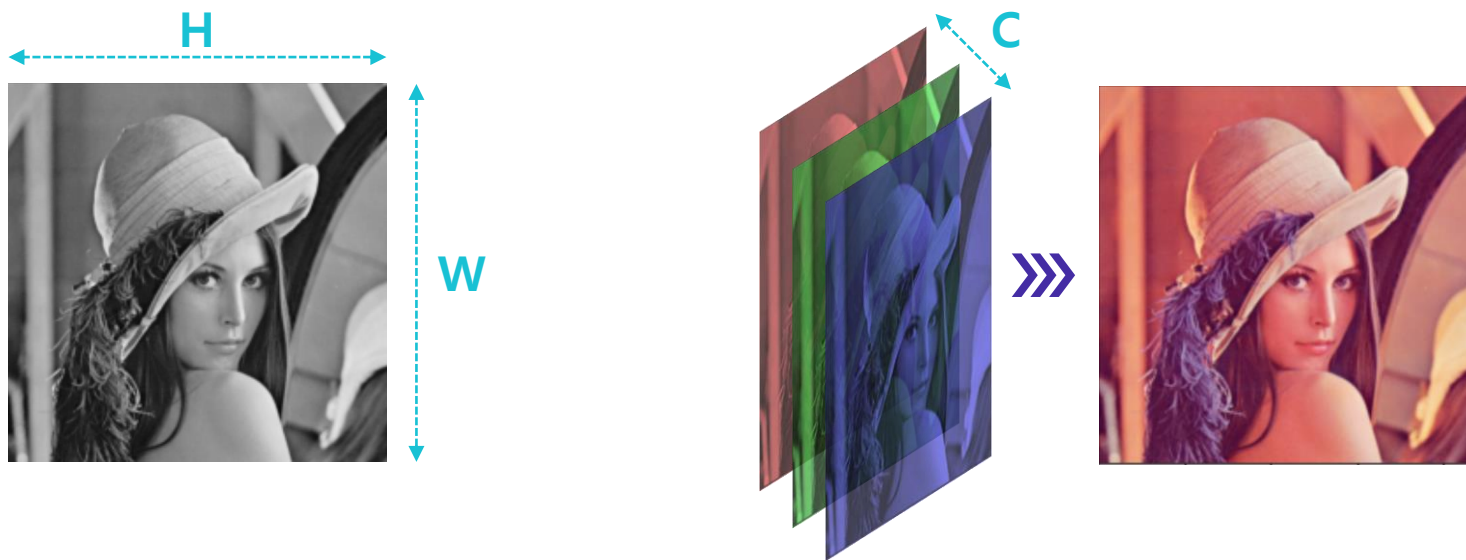
## 프로세스




- ✓ 합성곱층(Conv layer)과 풀링층(Pooling layer)을 거치면서 **이미지의 특성 벡터(feature vector)**를 추출
- ✓ 주요 특성 완전연결층(FC layer)를 거치면서 1차원 벡터로 변환
- ✓ 출력층에서 Softmax 함수를 거쳐 최종 결과 출력

## Input layer 입력층

- 높이(Height)\*너비(Weight)\*채널(Channel) 값을 갖는 3차원 데이터



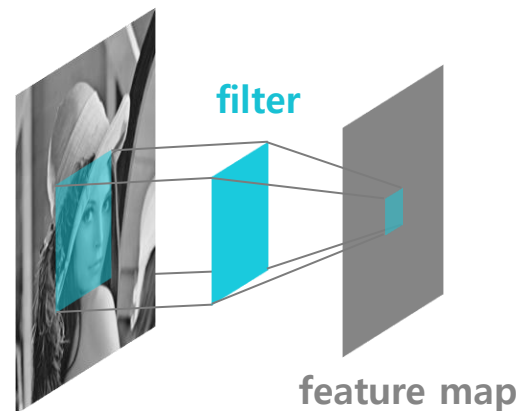
✓ 흑백 이미지는  $C=1$ , 칼라 이미지는  $C=3$  (RGB)

 'Lenna' 사진은 컴퓨터 비전 분야에서 이미지 압축과 같은 기술을 연구하는 데 사용되었으며, 이미지 처리 및 컴퓨터 비전 분야에서 표준 벤치마크로 자리 잡았습니다. Lenna 사진은 색상과 톤, 경계 및 세부 정보의 다양한 특성을 가지고 있기 때문에 이미지 처리 분야에서 공통적으로 사용되는 데이터입니다.

## Conv<sub>(olution)</sub> layer 합성곱층

- 입력 데이터에서 **특성(feature)**를 추출하는 층
  - ✓ 추출된 층은 **특성맵(feature map)**
- **커널(kernel) or 필터(filter)**를 사용
  - ✓ 합성곱층에서의 **가중치 파라미터**
  - ✓ 커널 값을 조정하면서 이미지의 다양한 특성맵을 추출할 수 있음 (ex. edge detection)
  - ✓ 커널의 크기는 하이퍼파라미터이지만 3x3, 5x5 크기가 일반적
- **스트라이드(Stride; 보폭)** : 필터를 적용하는 위치의 간격

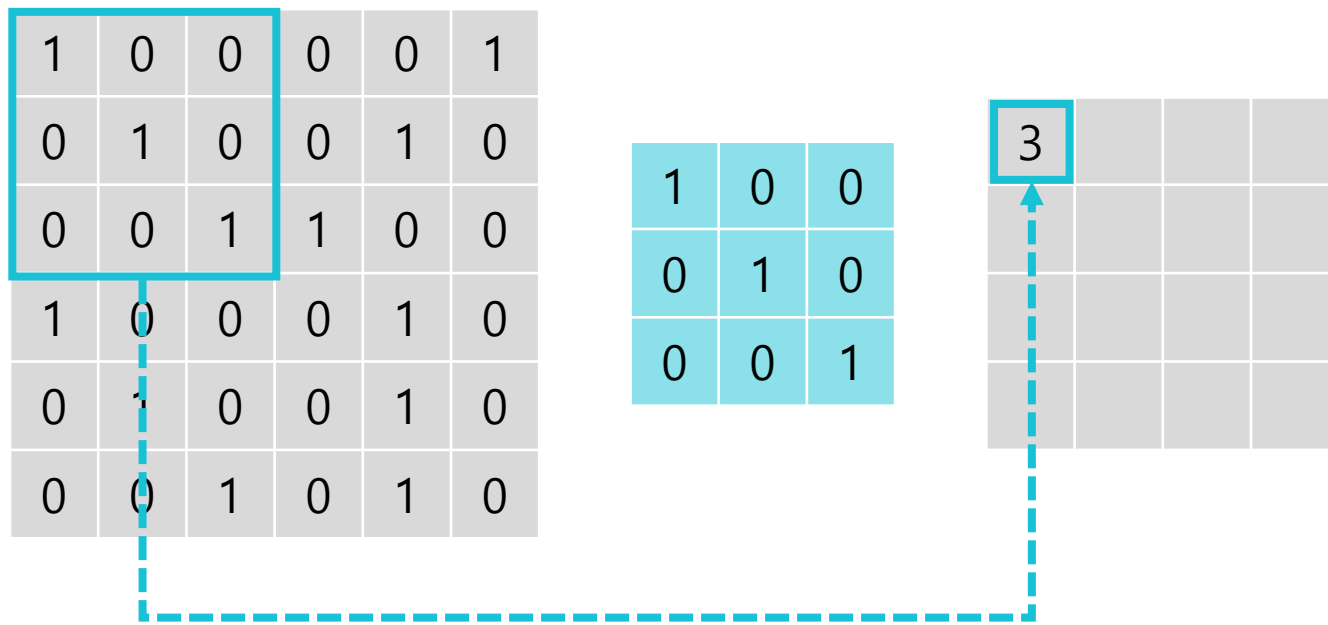
커널은 입력에 곱하는 가중치 이고, 필터는 뉴런 개수를 표현할 때 사용합니다.





## Conv<sub>(olution)</sub> layer 합성곱층

- channel = 1일때 연산과정



가중합

- ✓ 6x6 input 이미지, 3x3 filter
- ✓ 입력 이미지와 필터를 포개 놓고 대응되는 숫자끼리 곱한 후 모두 더하는 연산 수행
- ✓  $(1 \times 1) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (1 \times 1) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (1 \times 1) = 3$

## Conv<sub>(olution)</sub> layer 합성곱층

- channel = 1일때 연산과정

Stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	0	0
0	1	0
0	0	1

3	1		

가중합

- ✓ s=1이므로 filter가 1만큼 이동
- ✓ filter가 이미지 전체를 순회할 때 까지 연산과 이동을 반복

## Conv<sub>(olution)</sub> layer 합성곱층

- channel = 1일때 연산과정

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

1	0	0
0	1	0
0	0	1

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

- ✓ (6,6,1) 이미지  
→ (4,4,1) 특성맵 추출
- ✓ input 이미지의 **진한 부분**이  
filter와의 연산 횟수가 많음

가중합

## Conv(olution) layer 합성곱층

- 합성곱 연산을 이용한 흑백 Lena 이미지 edge detection

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

### 합성곱 함수 정의

```
def convolve(image, kernel):
    """
    입력 이미지(image)와 커널(kernel)을 입력으로 받아서 합성곱을 수행합니다.
    """
    # 이미지와 커널의 shape 확인
    h, w = image.shape
    k_h, k_w = kernel.shape

    # 출력 이미지 shape 계산
    out_h = h - k_h + 1
    out_w = w - k_w + 1

    # 출력 이미지 초기화
    output = np.zeros((out_h, out_w))

    # 합성곱 수행
    for i in range(out_h):
        for j in range(out_w):
            output[i, j] = np.sum(image[i:i+k_h, j:j+k_w] * kernel)

    return output
```

```
# Lena 이미지 불러오기
lena = Image.open('Lena.jpg').convert('L')
lena_arr = np.array(lena)
```

```
# edge detection 필터 정의
filter = np.array([[ -1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
# 필터 적용
edges = convolve(lena_arr, filter)
```

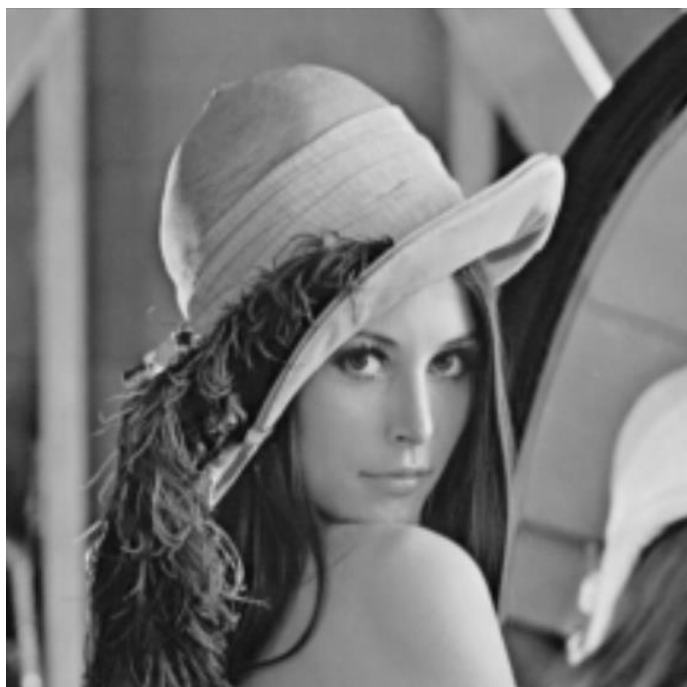
### Edge detection 필터

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

```
# 결과 출력
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection'), plt.xticks([]), plt.yticks([])
plt.show()
```

## Conv<sub>(olution)</sub> layer 합성곱층

- 합성곱 연산을 이용한 흑백 Lena 이미지 edge detection

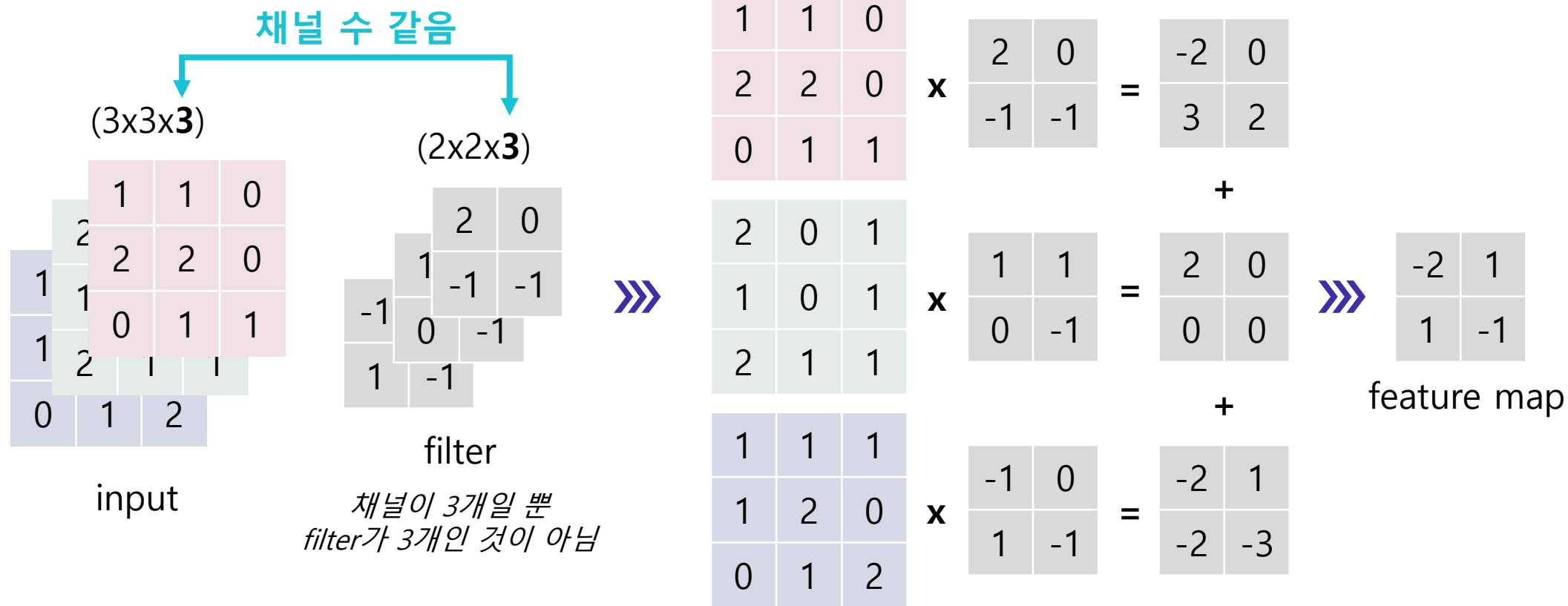


-1	-1	-1
-1	8	-1
-1	-1	-1



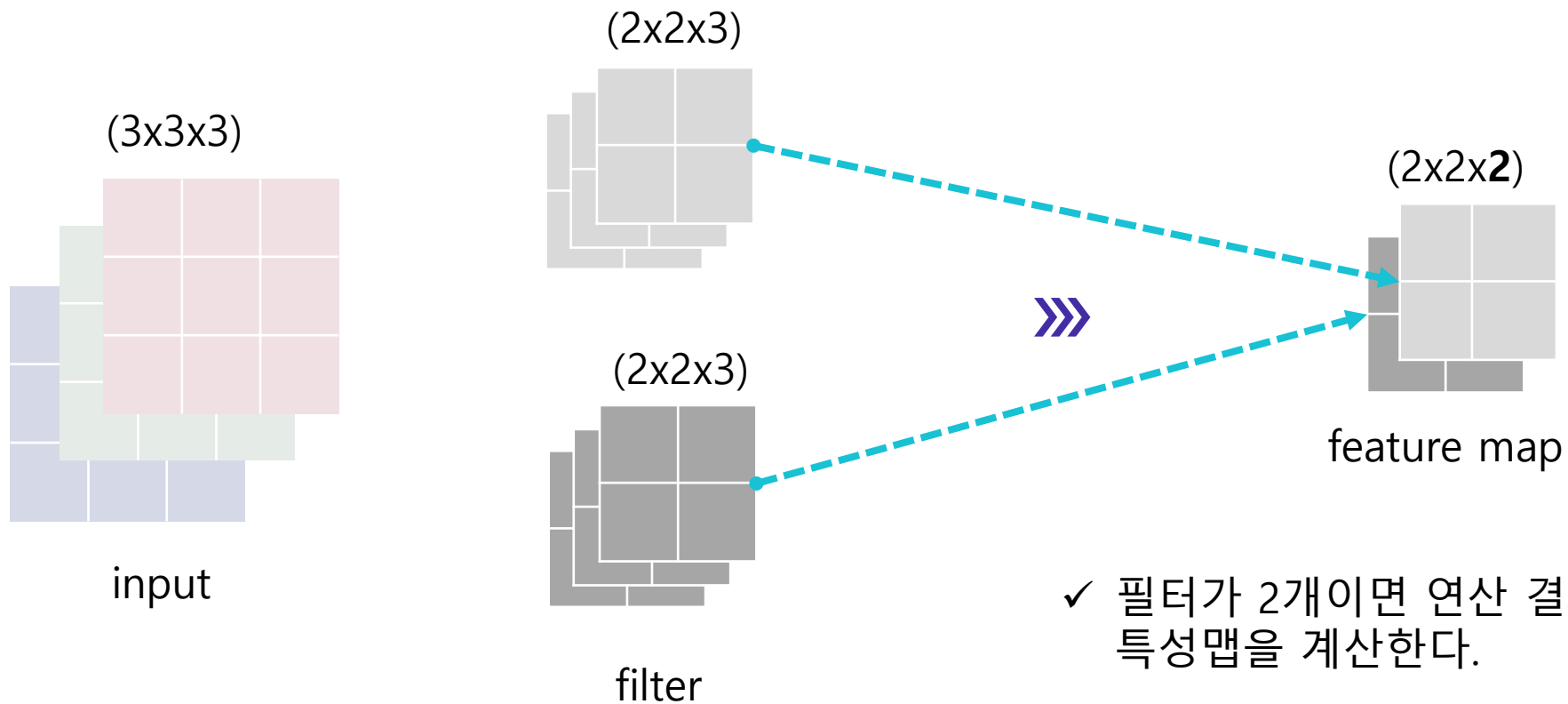
# Conv<sub>(olution)</sub> layer 합성곱층

- channel = 3 일때 합성곱 연산 과정



## Conv<sub>(olution)</sub> layer 합성곱층

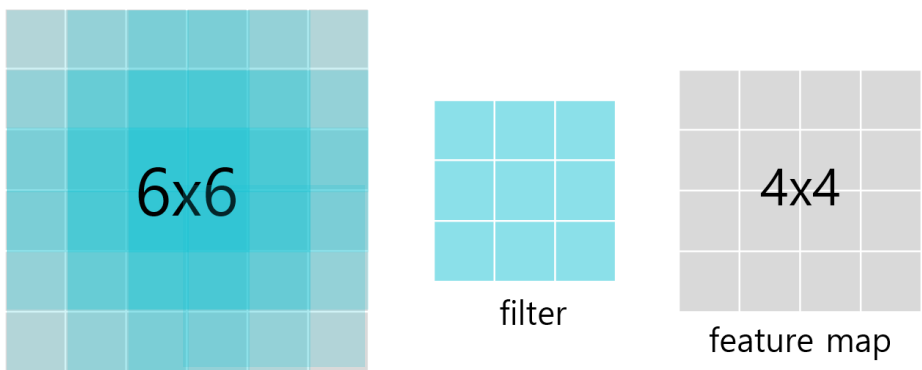
- filter가 여러 개일 경우



✓ 필터가 2개이면 연산 결과로 **채널이 2인** 특성맵을 계산한다.

## Conv<sub>(olution)</sub> layer 합성곱층

- 패딩(Padding)



input    input 이미지의 진한 부분이 filter와의 연산 횟수가 많음

- ✓ 6x6 이미지에 3x3 filter로 합성곱 연산 결과 4x4의 특성맵을 얻을 수 있음  
→ 특성맵의 크기가 input의 크기보다 작아짐

🚨 문제 1 : Conv Layer를 쌓다보면 마지막 특성맵의 사이즈는 매우 작아질 것임

🚨 문제 2 : 이미지의 가장자리에 있는 픽셀은 필터와의 합성곱 연산 횟수가 이미지 중앙의 픽셀에 비해 매우 적어 정보가 이미지 가장자리의 정보가 손실될 수 있음

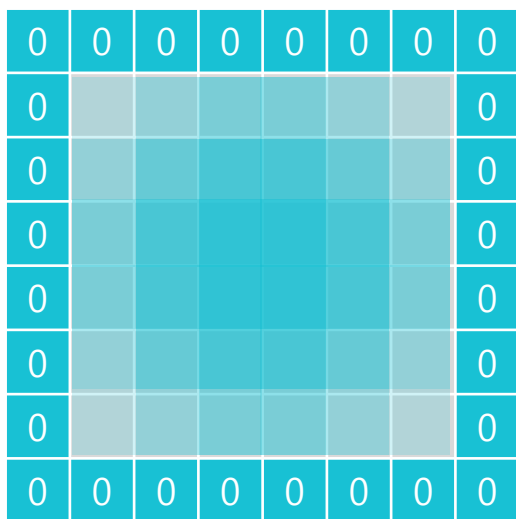


## Conv<sub>(olution)</sub> layer 합성곱층

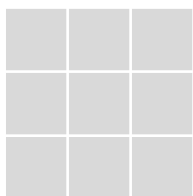
### • 패딩(Padding)

input 가장자리에 0 값을 가진 픽셀을 추가함

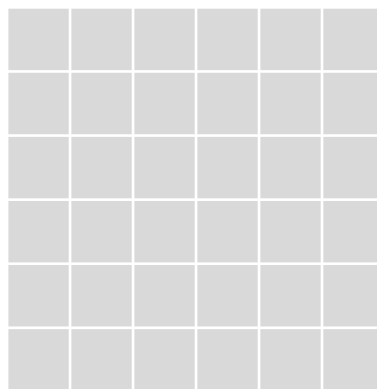
➡ 패딩 추가



Input  
(6x6x1)



filter



feature map  
(6x6x1)

- ✓ input =  $W \times H \times C$
- ✓ hyper parameter
  - filter 크기 :  $F$
  - filter 개수 :  $FN$
  - stride :  $S$
  - 패딩 :  $P$
- ✓ Output
  - $W_2 = H_2 = \frac{W-F+2P}{S} + 1$
  - $C_2 = FN$

## Pooling layer 풀링층

- 풀링층은 특성맵의 크기를 다운 샘플링하여 연산량을 감소시키고, 주요한 특성 벡터를 추출하여 학습을 효과적으로 할 수 있게 해줌
- 합성곱층 (합성곱 연산 + 활성화 함수) 다음에 풀링층을 추가하는 것이 일반적
- 특성맵(Conv layer의 output)에 적용하는 가중치 없는 필터
- 2가지 풀링 연산
  - ✓ **평균풀링 (average pooling)** : 대상 영역에서 평균을 반환
  - ✓ **최대풀링 (max pooling)** : 대상 영역에서 최댓값을 추출
    - 일반적으로 최대풀링이 사용됨
    - 평균풀링은 각 필터 값을 평균화 시켜 중요한 가중치를 갖는 값의 특성이 희미해질 수 있음

## Pooling layer 풀링층

Stride=2

3	-1	12	-1
-3	1	0	1
2	-3	0	1
3	-2	4	-1

input

3	12
3	4

max  
pooling

0	3
0	1

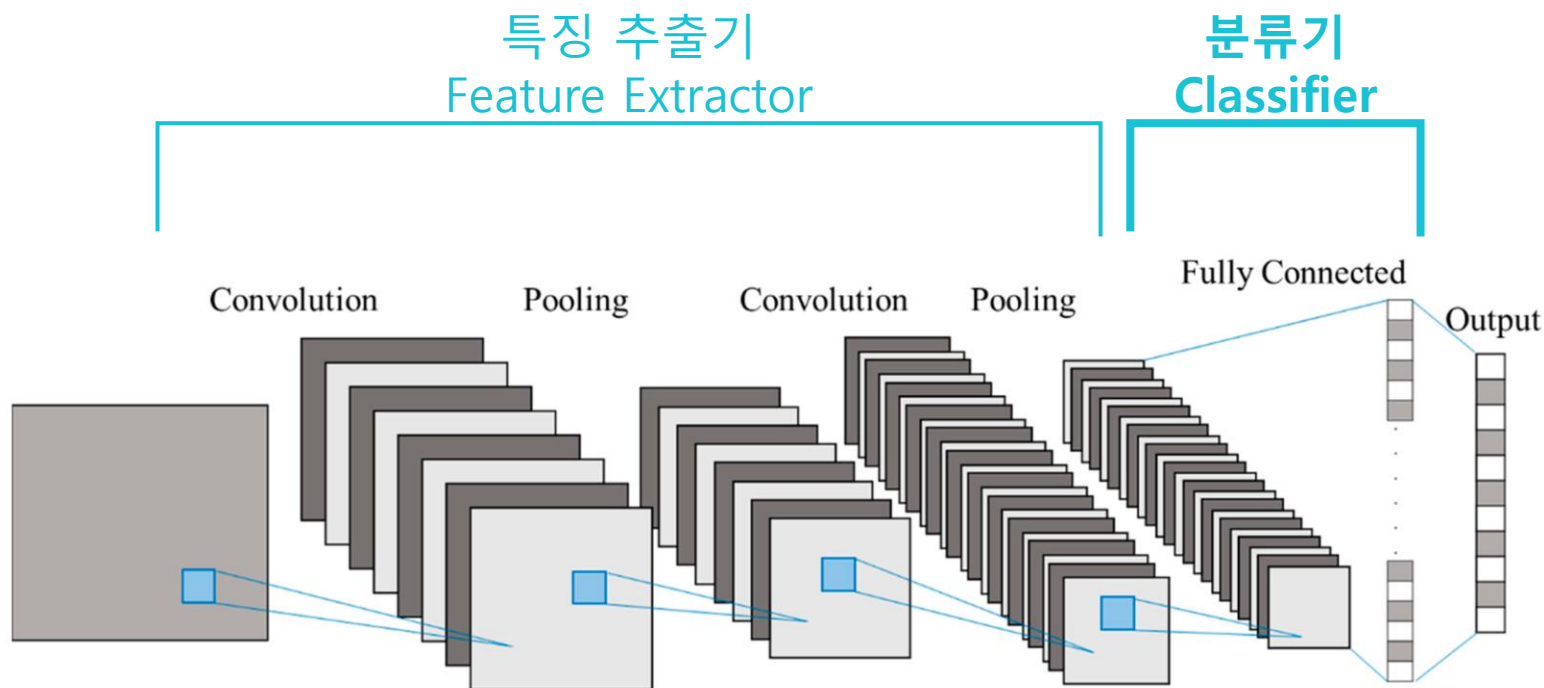
average  
pooling

output

- ✓ input =  $W \times H \times C$
- ✓ hyper parameter
  - filter 크기 :  $F$
  - stride :  $S$
- ✓ Output
  - $W_2 = H_2 = \frac{W-F}{S} + 1$
  - $C_2 = C$

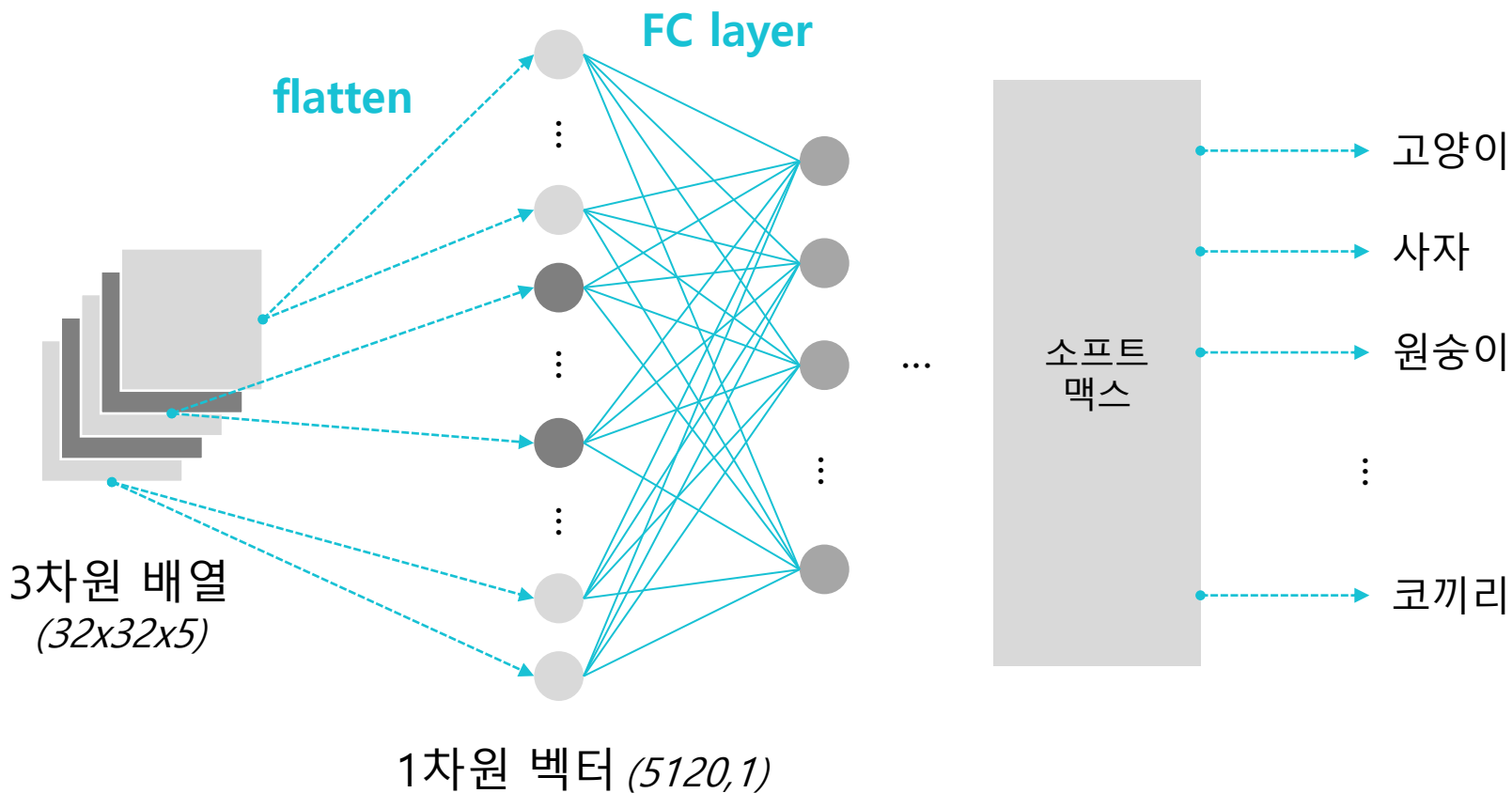
## FC layer 완전연결층 (Fully Connected)

- Conv layer와 Pooling layer로 추출된 특징을 이용하여 **이미지를 분류하는 층**
- 일반적인 신경망 구조인 DNN을 사용



## FC layer 완전연결층 (Fully Connected)

- Flatten 과정을 거친 후 FC층에서 분류가 일어남



## Sample Code

- Input Image : 3x32x32 (CxHxW)
- layer1

Conv2d	Input	Output
Channel	3	7
Height & Width	32	$\frac{32 - 5 + 2 \times 0}{1} + 1 = 28$

$$\checkmark W_2 = H_2 = \frac{W-F+2P}{S} + 1$$

**BatchNorm2d** : num\_features (int) – C from (C,H,W)

MaxPool2d	Input	Output
Channel	7	7
Height & Width	28	$\frac{28-2}{2} + 1 = 14$

$$\checkmark W_2 = H_2 = \frac{W-F}{S} + 1$$

```
class CNN(nn.Module):
    def __init__(self, num_classes=47):
        super(CNN, self).__init__()

        ...
        사용할 data 원본: 3(channel) x 32(height) x 32(width)
        분류할 class 수: 10
        layer1, layer2 - Convolution layer
        fc1, fc2 - Fully-Connected layer
        ...

    #input size: 3x32x32
    self.layer1 = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=7, kernel_size=5, stride=1, padding=0),
        nn.BatchNorm2d(7),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2), # stride : Default value is kernel_size
    )

    self.layer2 = nn.Sequential(
        nn.Conv2d(in_channels=7, out_channels=15, kernel_size=5, stride=1, padding=2),
        nn.BatchNorm2d(15),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2),
    )

    self.fc1 = nn.Linear(735, 300)
    self.fc2 = nn.Linear(300, 10)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)

    return x
```

## Sample Code

- Input Image : 3x32x32 (CxHxW)
- layer1
- layer2

Conv2d	Input	»»»	Output
Channel	7	»»»	15
Height & Width	14	»»»	$\frac{14 - 5 + 2 \times 2}{1} + 1 = 14$

✓  $W_2 = H_2 = \frac{W-F+2P}{S} + 1$

**BatchNorm2d** : num\_features (int) – C from (C,H,W)

MaxPool2d	Input	»»»	Output
Channel	15	»»»	15
Height & Width	14	»»»	$\frac{14 - 2}{2} + 1 = 7$

✓  $W_2 = H_2 = \frac{W-F}{S} + 1$

```
class CNN(nn.Module):
    def __init__(self, num_classes=47):
        super(CNN, self).__init__()

        '''
        사용할 data 원본: 3(channel) x 32(height) x 32(width)
        분류할 class 수: 10
        layer1, layer2 - Convolution layer
        fc1, fc2 - Fully-Connected layer
        '''

        #input size: 3x32x32
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=7, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(7),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2), # stride : Default value is kernel_size
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=7, out_channels=15, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(15),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.fc1 = nn.Linear(735, 300)
        self.fc2 = nn.Linear(300, 10)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)

        return x
```

## Sample Code

- Input Image : 3x32x32 (CxHxW)
- layer1
- layer2
- FC layers (Classifier)

FC layer	Input	»»	Output
Fc1	15*7*7=735	»»	300
Fc2	300	»»	10

3차원 → 1차원 Flatten ←

```
class CNN(nn.Module):
    def __init__(self, num_classes=47):
        super(CNN, self).__init__()

        """
        사용할 data 원본: 3(channel) x 32(height) x 32(width)
        분류할 class 수: 10
        layer1, layer2 - Convolution layer
        fc1, fc2 - Fully-Connected layer
        """

        #input size: 3x32x32
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=7, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(7),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2), # stride : Default value is kernel_size
        )

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=7, out_channels=15, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(15),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )

        self.fc1 = nn.Linear(735, 300)
        self.fc2 = nn.Linear(300, 10)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = x.reshape(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)

        return x
```



# Sample Code

```
model = CNN(10).to(device)
torchsummary.summary(model, input_size=(3, 32, 32), device='cuda')
```



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 7, 28, 28]	532
BatchNorm2d-2	[-1, 7, 28, 28]	14
ReLU-3	[-1, 7, 28, 28]	0
MaxPool2d-4	[-1, 7, 14, 14]	0
Conv2d-5	[-1, 15, 14, 14]	2,640
BatchNorm2d-6	[-1, 15, 14, 14]	30
ReLU-7	[-1, 15, 14, 14]	0
MaxPool2d-8	[-1, 15, 7, 7]	0
Linear-9	[-1, 300]	220,800
Linear-10	[-1, 10]	3,010
Total params: 227,026		
Trainable params: 227,026		
Non-trainable params: 0		

(Channel, Height, Width)

layer 1 : (3,32,32) → (7,28,28) → (7,14,14)

layer 2 : (7,14,14) → (15,14,14) → (15,7,7)

fc layers : 15\*7\*7 → 300 → 10

2

전이학습

## 전이학습 (Transfer Learning)

- CNN 기반 딥러닝 모델을 제대로 훈련시키려면 **많은 양의 데이터가 필요**
- 개인이 충분한 데이터셋을 확보하는 것은 어려움



- **전이학습 :**

이미지넷\* IMAGENET 처럼 아주 큰 데이터셋을 써서 훈련된 모델의 가중치를 가져와서 우리가 해결하려는 과제에 맞게 보정해서 사용하는 것을 의미함

- ✓ **특징 추출** (Feature extraction)

- ✓ **미세 조정** (Fine-tuning)

- 새로운 **데이터의 특성**을 고려하여 미세조정 범위를 설정해야 함

- ① **Dataset size**

- ② **Dataset similarity**

## 이미지넷 ImageNet

- **ILSVR** (ImageNet Large Scale Visual Recognition Challenge)

- ✓ 이미지넷 이미지 인식 대회
- ✓ 컴퓨터 비전 분야의 '올림픽'

[paperswithcode 리더보드 확인](https://paperswithcode.com/sota/image-classification-on-imagenet)

- 이미지넷 데이터

1000개의 클래스, 100만개가 넘는 데이터

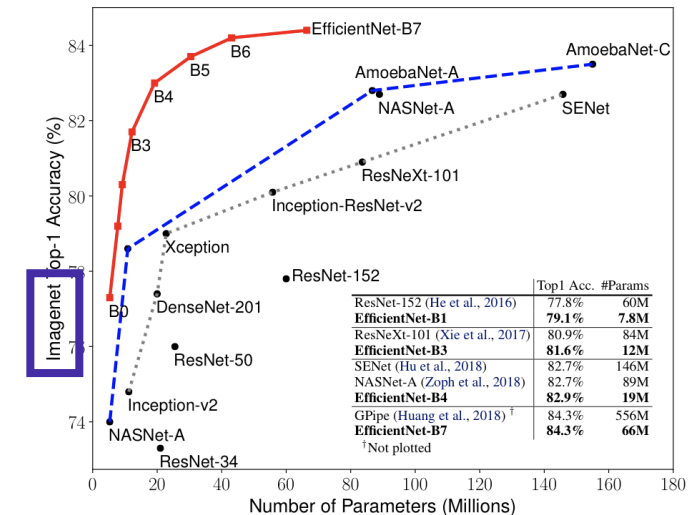
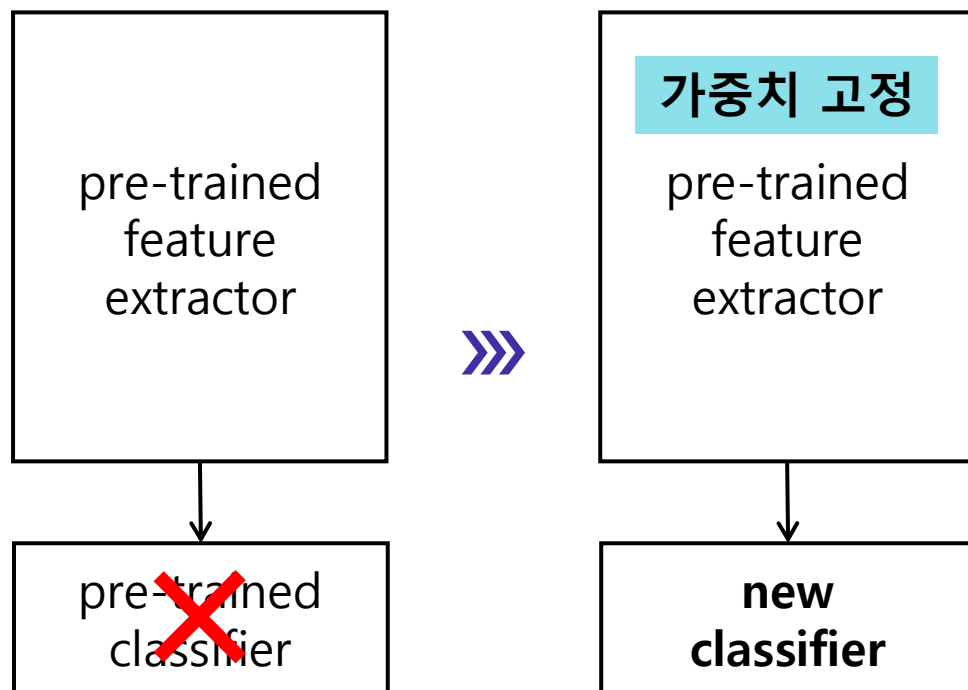


Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

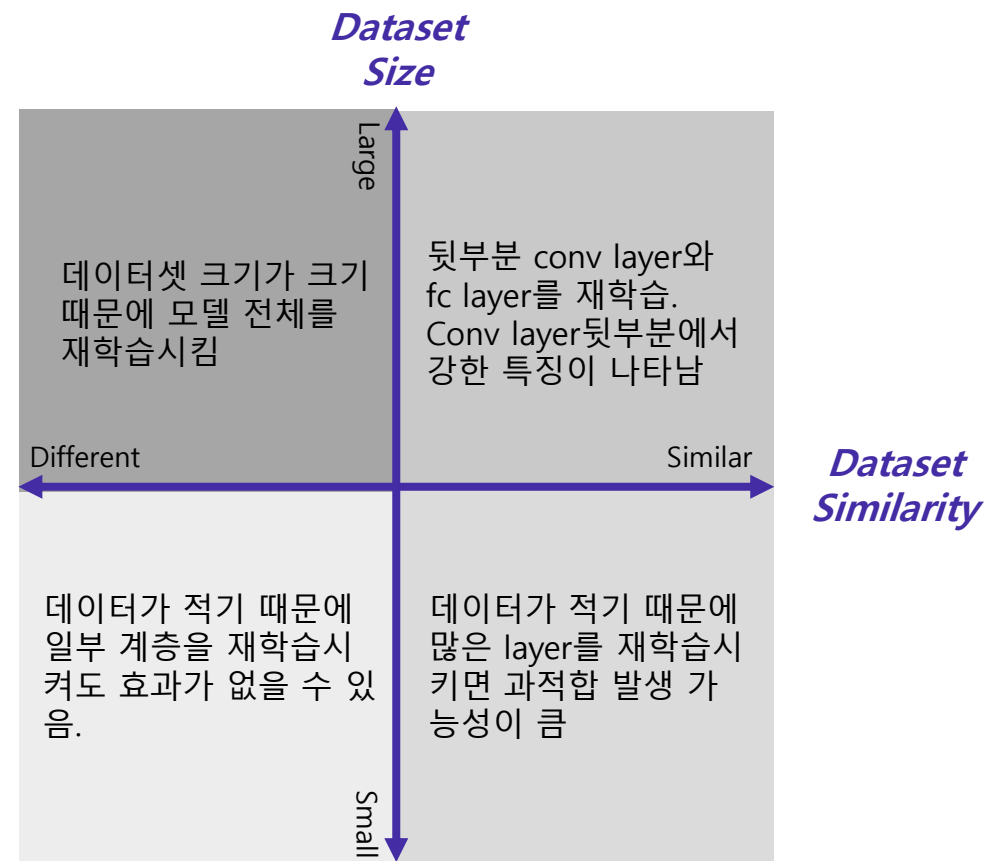
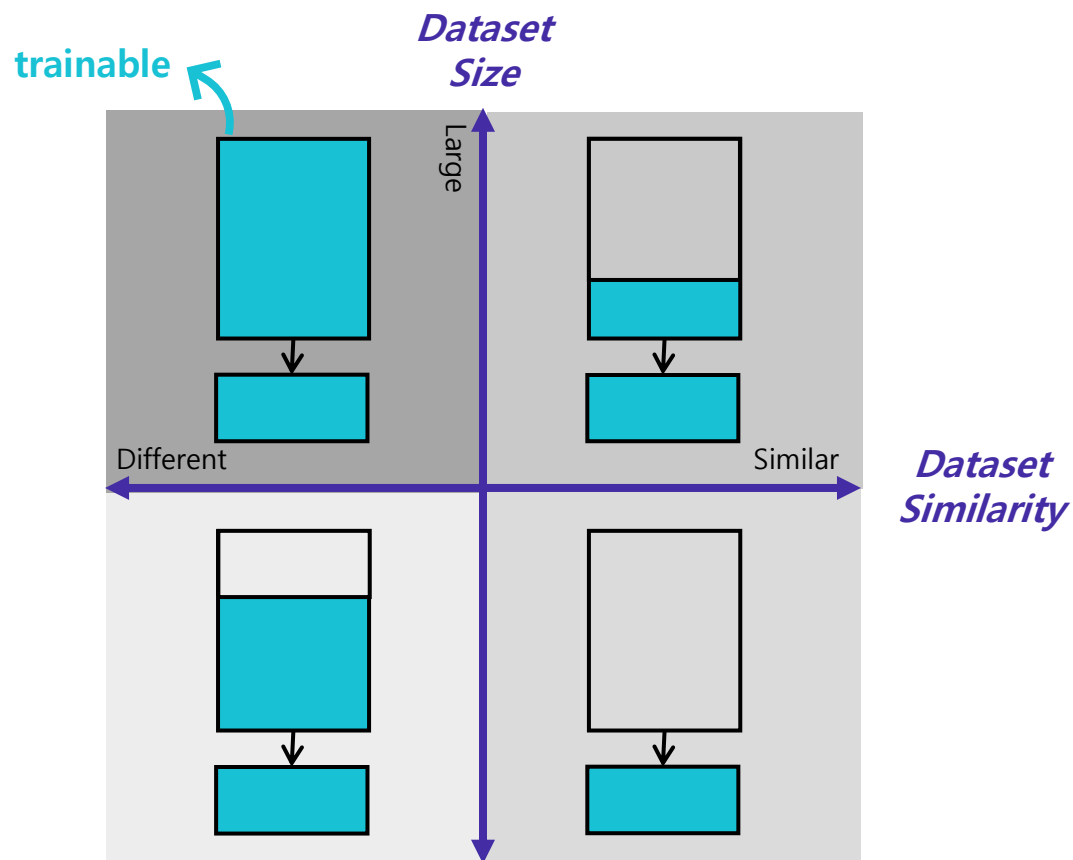
## 사전 학습된 신경망을 특징 추출기로 이용

- 이미지넷 데이터셋을 학습한 CNN의 특징 추출기 부분의 가중치를 고정하고 분류기를 제거한 다음 새로운 분류기를 추가함



- ✓ Ex. 개와 고양이를 분류하는 문제
  - 이미지넷은 1,000개 이상의 클래스로 구성되어 있으므로 기존 신경망의 classifier도 1,000가지로 분류하도록 설계 및 학습됨
  - 새로운 classifier를 만들어 추가하는 것이 효율적임

## 미세 조정 범위 설정



## Sample Code

```
import torch.nn as nn
import torch.nn.functional as F
import torchvision
import torchsummary
import torchvision.models as models
```

```
model=models.resnet18(pretrained=True)
torchsummary.summary(model, input_size=(3, 28, 28))
```

⋮

Conv2d-63	[-1, 512, 1, 1]	2,359,296
BatchNorm2d-64	[-1, 512, 1, 1]	1,024
ReLU-65	[-1, 512, 1, 1]	0
BasicBlock-66	[-1, 512, 1, 1]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 1000]	513,000

```
=====
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
```

```
-----
Input size (MB): 0.01
Forward/backward pass size (MB): 1.10
Params size (MB): 44.59
Estimated Total Size (MB): 45.70
-----
```

```
# fc layer의 input features
print(f'input : {model.fc.in_features}')
print(f'output : {model.fc.out_features}') #1000개의 class로 분류
```

```
input : 512
output : 1000
```

```
# classifier out_features 변경
model.fc=nn.Linear(in_features=512,out_features=2) # 이진분류일때
torchsummary.summary(model,input_size=(3, 28, 28))
```

⋮

Conv2d-63	[-1, 512, 1, 1]	2,359,296
BatchNorm2d-64	[-1, 512, 1, 1]	1,024
ReLU-65	[-1, 512, 1, 1]	0
BasicBlock-66	[-1, 512, 1, 1]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 2]	1,026

```
=====
Total params: 11,177,538
Trainable params: 11,177,538
Non-trainable params: 0
```

```
-----
Input size (MB): 0.01
Forward/backward pass size (MB): 1.09
Params size (MB): 42.64
Estimated Total Size (MB): 43.74
-----
```

# Sample Code

```
for param in model.parameters(): #----- 모델의 모든 가중치 고정
    param.requires_grad=False
```

```
for param in model.fc.parameters(): #----- 모델의 fc층은 학습
    param.requires_grad = True
torchsummary.summary(model,input_size=(3,28,28))
```

⋮

Conv2d-63	[-1, 512, 1, 1]	2,359,296
BatchNorm2d-64	[-1, 512, 1, 1]	1,024
ReLU-65	[-1, 512, 1, 1]	0
BasicBlock-66	[-1, 512, 1, 1]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 2]	1,026

```
=====  
Total params: 11,177,538  
Trainable params: 1,026  
Non-trainable params: 11,176,512  
=====
```

```
-----  
Input size (MB): 0.01  
Forward/backward pass size (MB): 1.09  
Params size (MB): 42.64  
Estimated Total Size (MB): 43.74  
-----
```

For 문 범위 조절을 통해 fine tuning

Trainable params  
=  $512 * 2 + 2 = 1,026$



2학기 4주차 진도세션

4조 노지예 임청수 한세림

**감사합니다**

*Convolution Neural Network & Transfer Learning*