

딤러닝 파이토치 교과서 5장

합성곱 신경망!

20기 분석
송여진



CONTENTS

01 합성곱 신경망

- 합성곱층의 필요성
- 합성곱 신경망 구조
- 1D, 2D, 3D 합성곱

02 합성곱 신경망 맛보기

03 전이 학습

- 특성 추출 기법
- 미세 조정 기법

04 설명 가능한 CNN

- 특성 맵 시각화

05 그래프 합성곱 네트워크

- 그래프란?
- 그래프 신경망
- 그래프 합성곱 네트워크

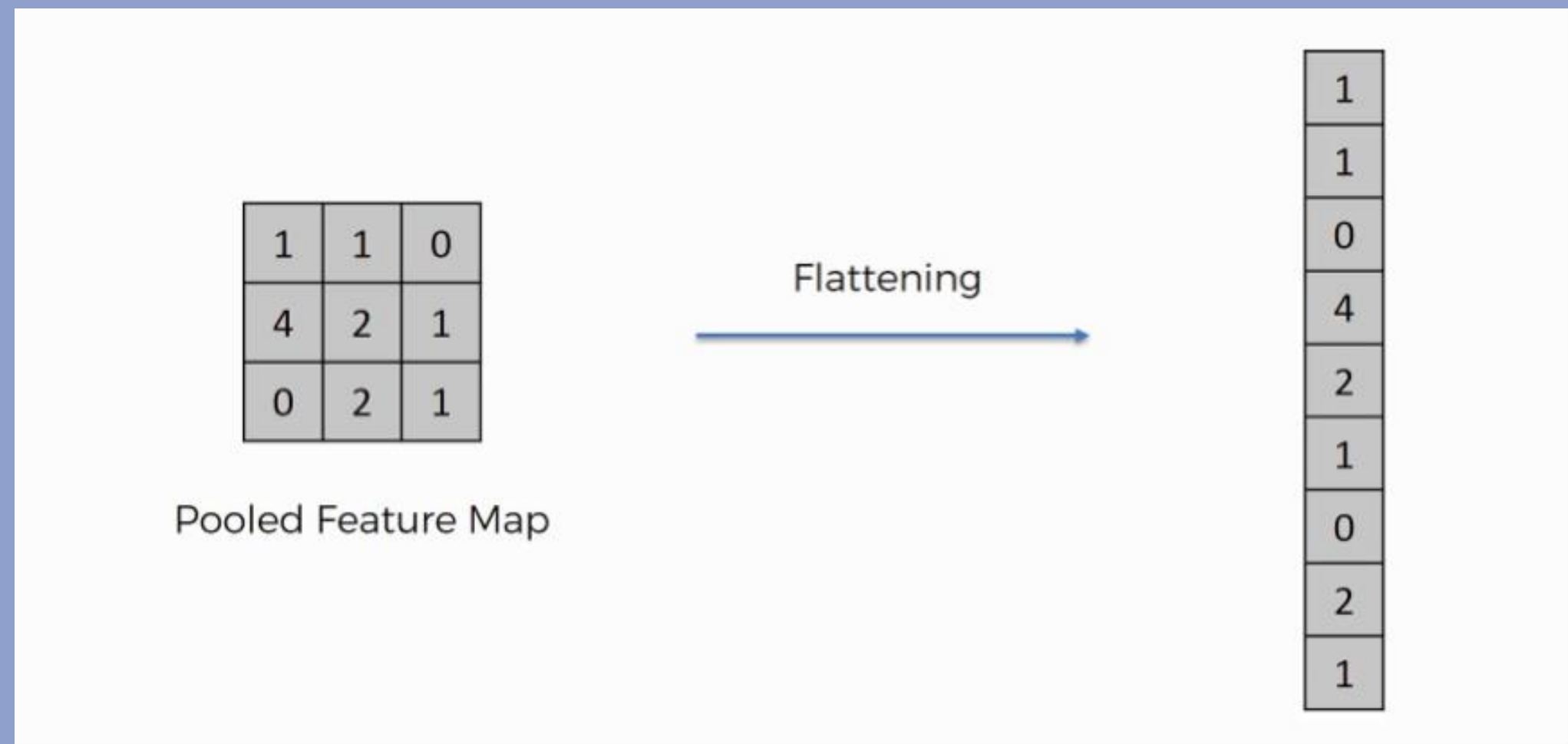
합성곱층의 필요성.



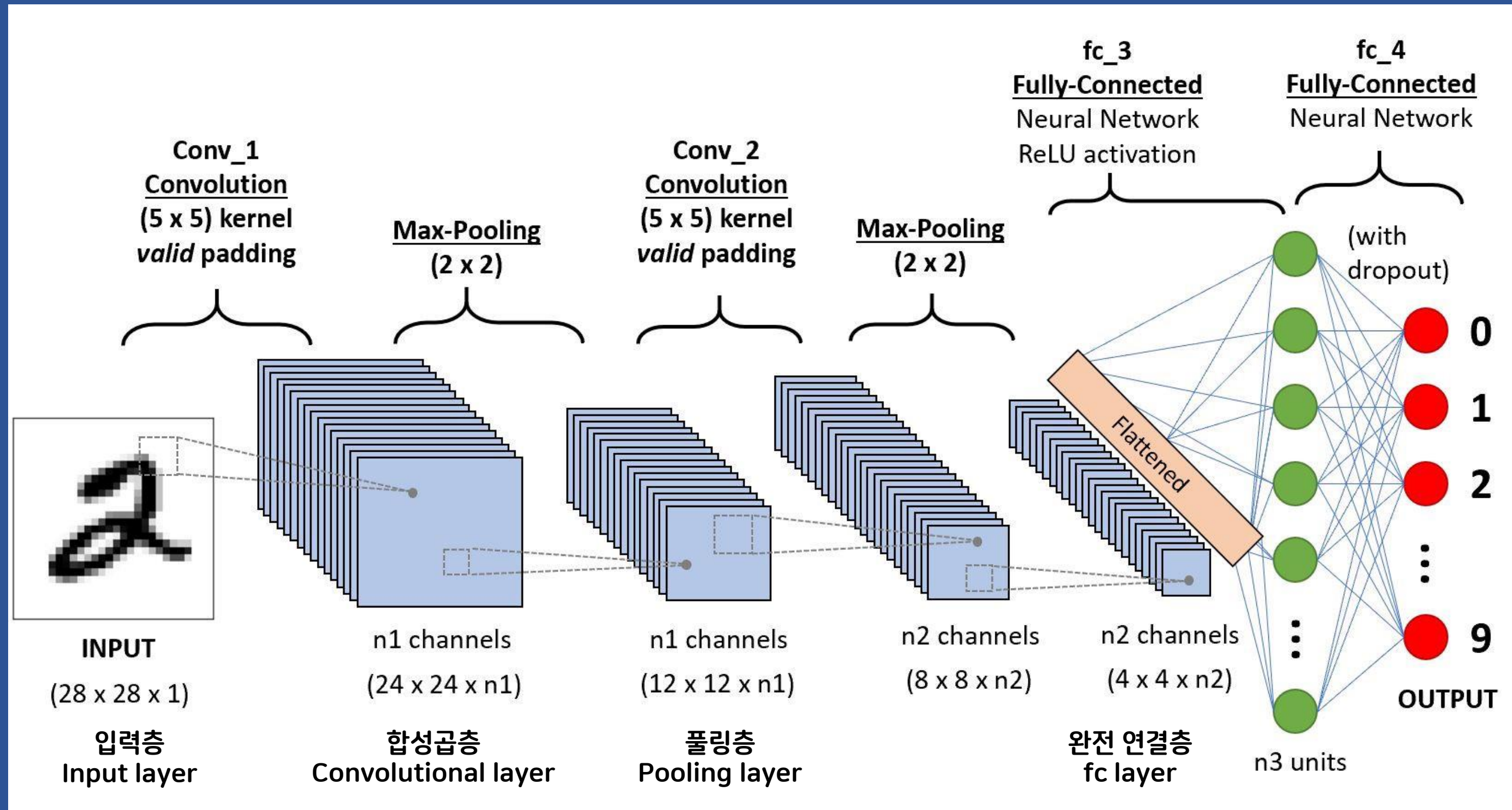
> 데이터의 공간적 구조 유지

이미지 분석은 그림과 같이 배열을 펼쳐서(flattening) 은닉층에 전달

But, 이미지를 펼쳐서 분석하면 데이터의 공간적 구조를 무시하게 됨. 이를 방지하기 위해 도입

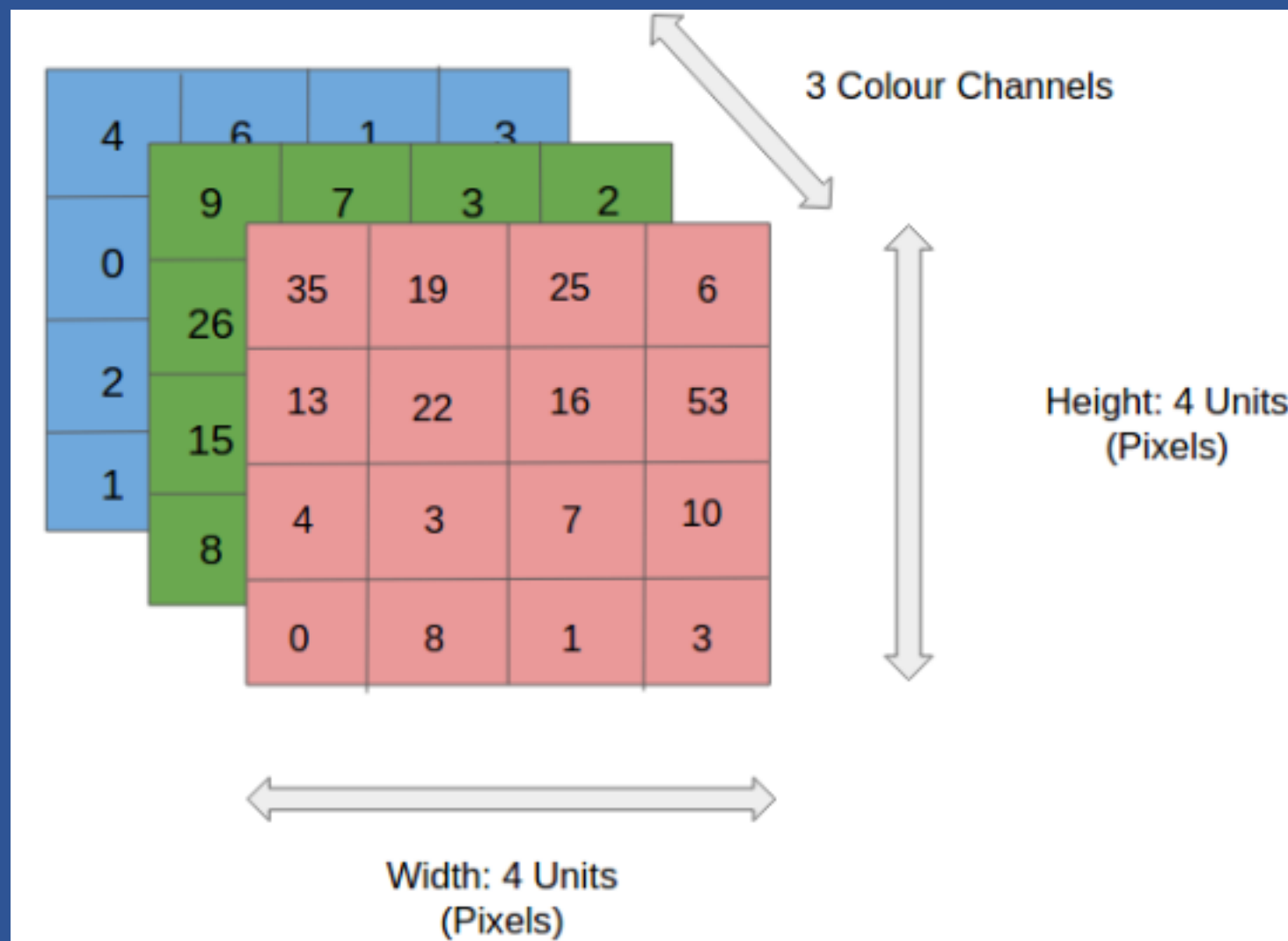


합성곱 신경망 구조.



입력층.

Input layer



> 입력 이미지 데이터가 최초로 거치게 되는 계층

이미지는 1차원 데이터가 아닌 높이(height), 너비(width), 채널(channel)의 값을 갖는 3차원 데이터
채널은 이미지가 그레이스케일(gray scale)이면 1의 값을 가지고 컬러(RGB)이면 3 값을 가짐

> 사진의 이미지 형태는 어떻게 표현할 수 있을까?

높이 4, 너비 4, 채널은 RGB이므로 3

-> shape은 (4,4,3)

합성곱층.

Convolutional layer

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

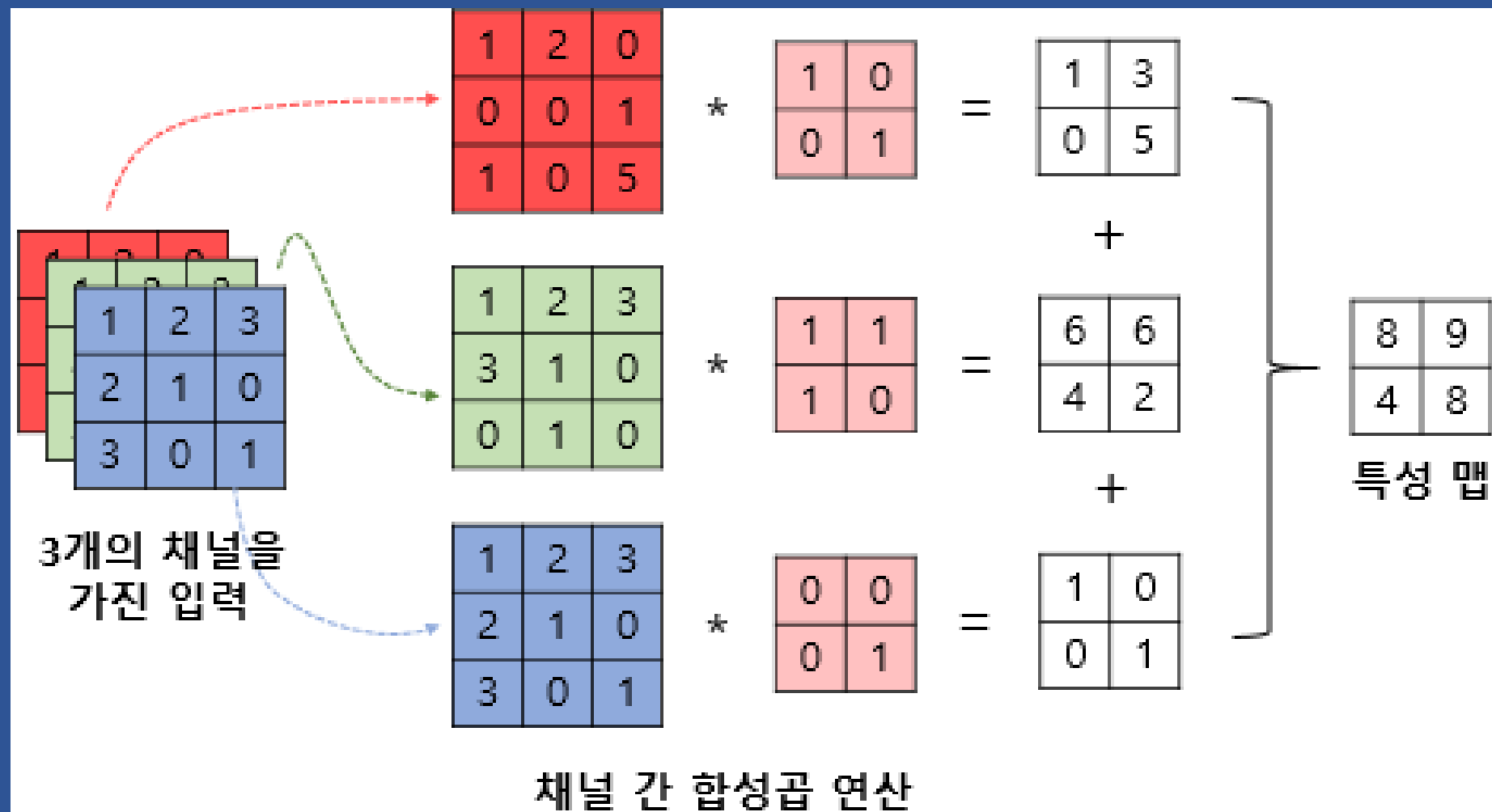
> 입력 데이터에서 특성을 추출하는 역할

입력 이미지가 들어왔을 때 이미지에 대한 특성을 감지하기 위해 커널(kernel)이나 필터를 사용
커널은 정해진 스트라이드(stride)에 따라 순차적으로 이동함

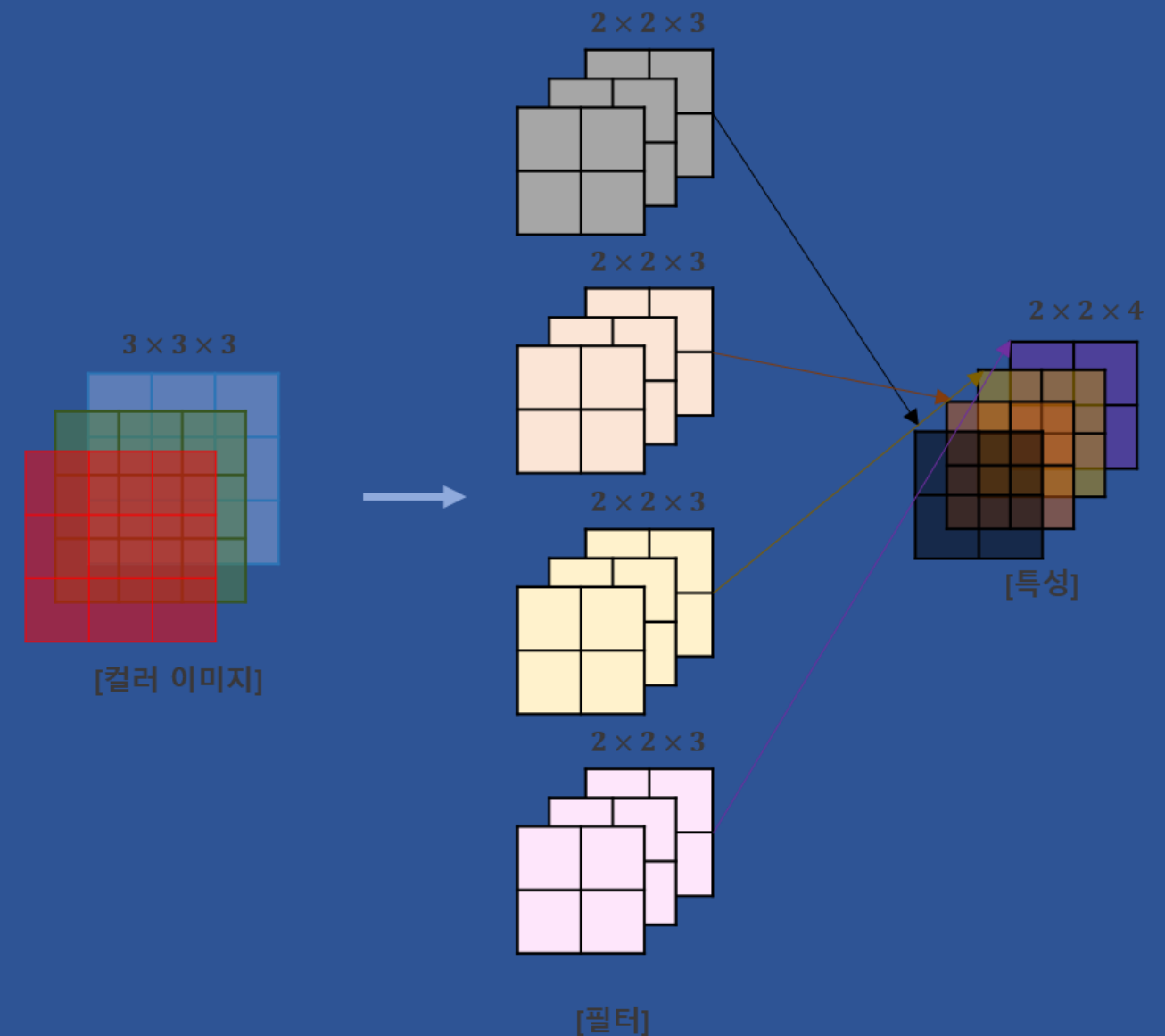
> 원본 크기가 줄어드는 특징을 가지고 있음

합성곱층.

Convolutional layer



컬러 이미지 합성곱



필터가 2 이상인 합성곱

합성곱층.

Convolutional layer

If,

Image Size = $n \times n$

Filter Size = $f \times f$

Stride = s

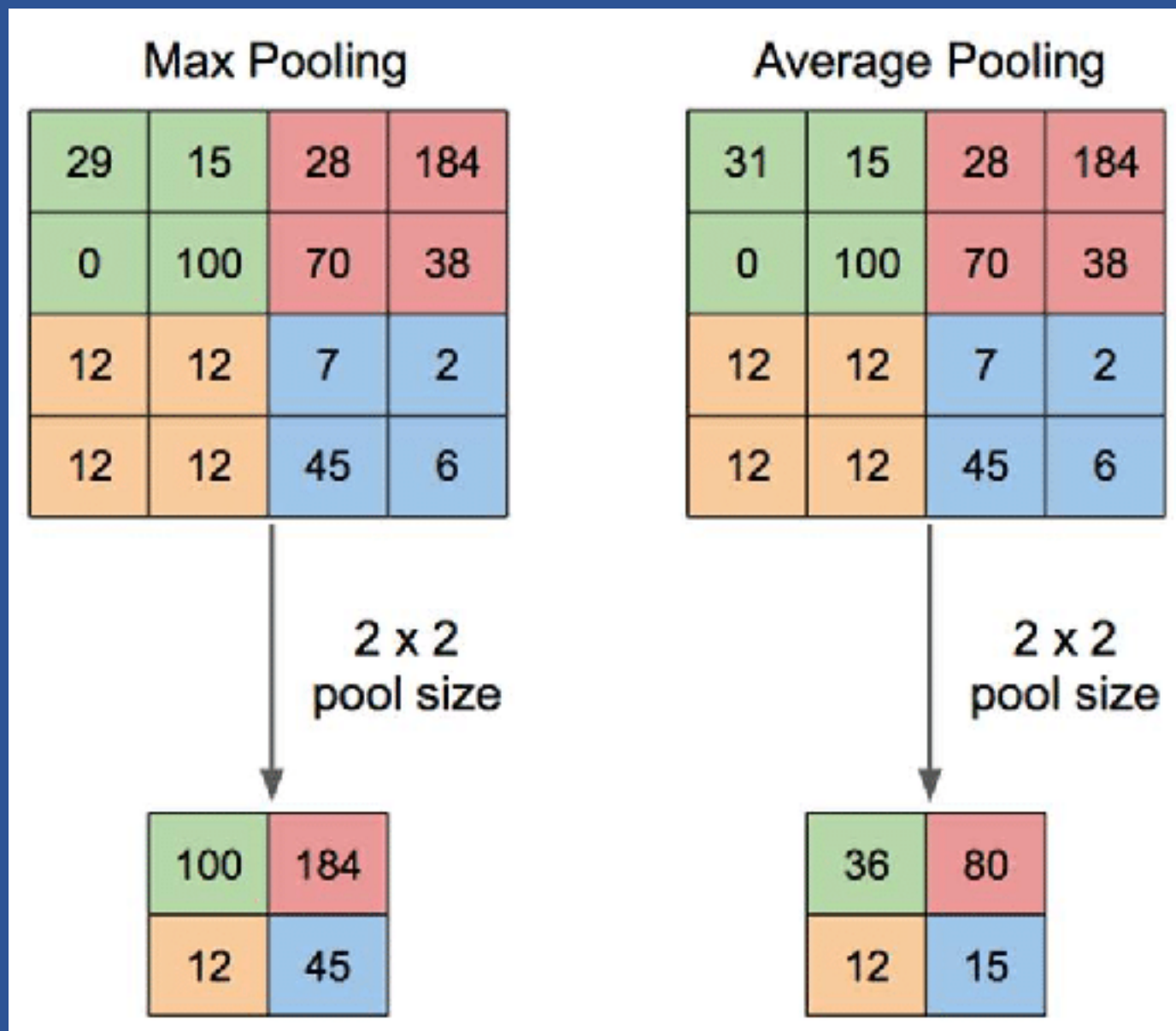
Padding = p

$$\text{Size of the output} = \left[\frac{n + 2p - f}{s} + 1 \right] \times \left[\frac{n + 2p - f}{s} + 1 \right]$$

합성곱 output size 계산 공식

풀링층.

Pooling layer



> 특징 맵의 차원을 다운샘플링하여 연산량을 감소

> Max pooling

대상 영역에서 최댓값을 추출하는 풀링 방법
대부분의 합성곱 신경망에서 사용되고 있다.

> Average pooling

대상 영역에서 평균을 반환하는 풀링 방법
값들을 평균화시키기 때문에 특성이 희미해질 수 있어 잘 사용하지 않음

풀링층.

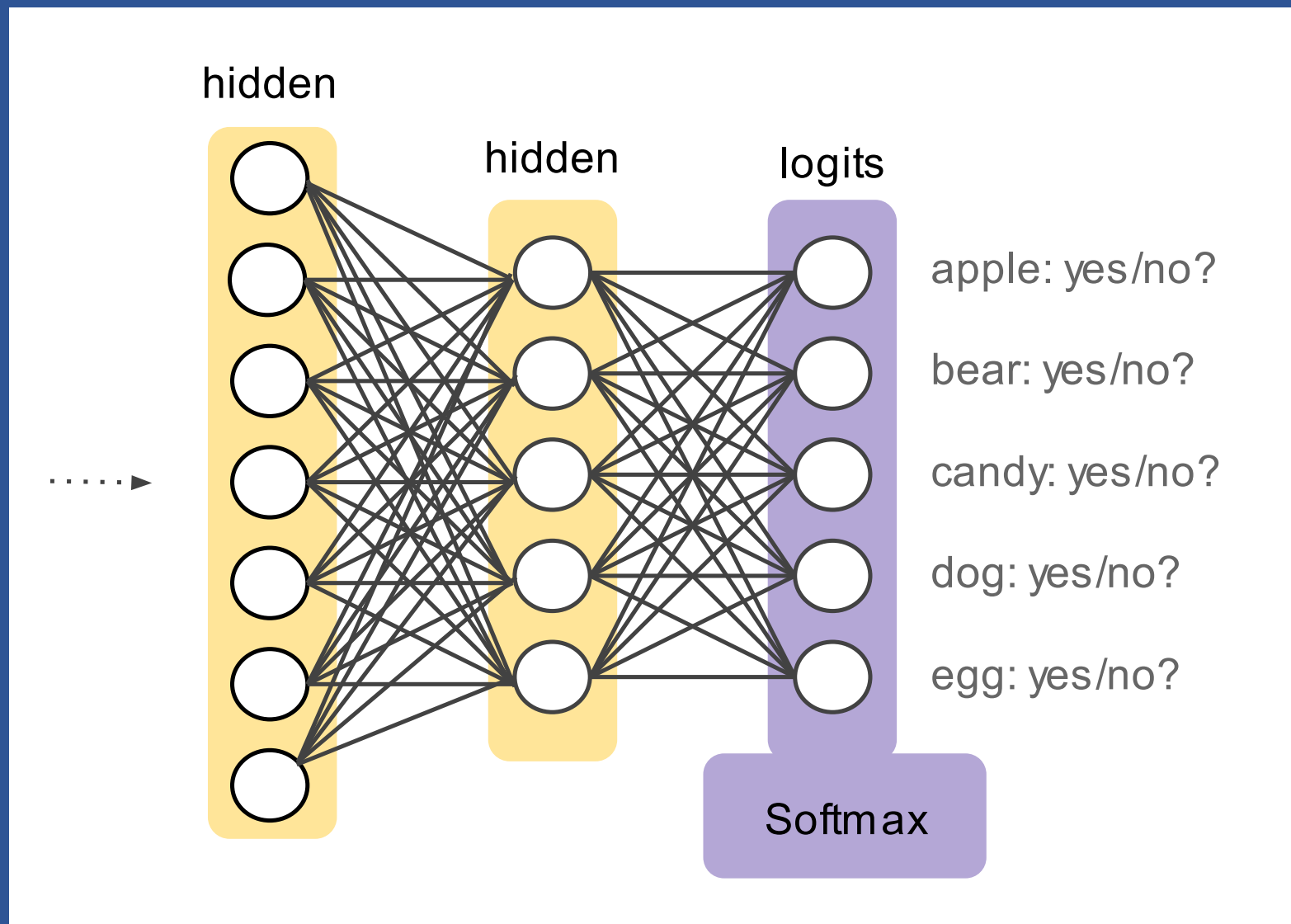
Pooling layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$

풀링층 output size 계산 공식

완전연결층 & 출력층.

Fully connected layer & output layer



> 이미지가 3차원 벡터에서 1차원 벡터로 펼쳐지게 됨

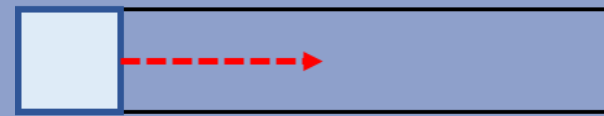
> Output layer에서는 Softmax 활성화 함수 사용

입력받은 값을 0과 1 사이의 값으로 출력하는 함수

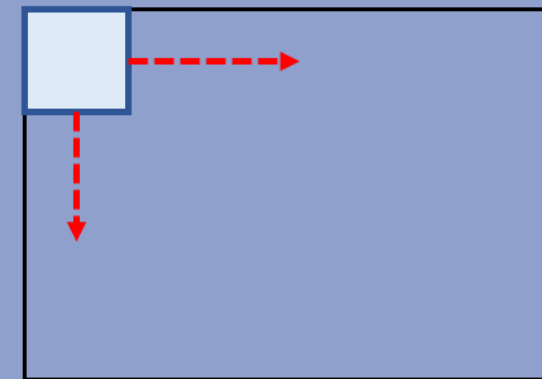
따라서 이미지가 해당 레이블(label)에 속할 확률 값이 출력되고 가장 높은 확률 값을 갖는 레이블이 선정

합성곱층의 분류.

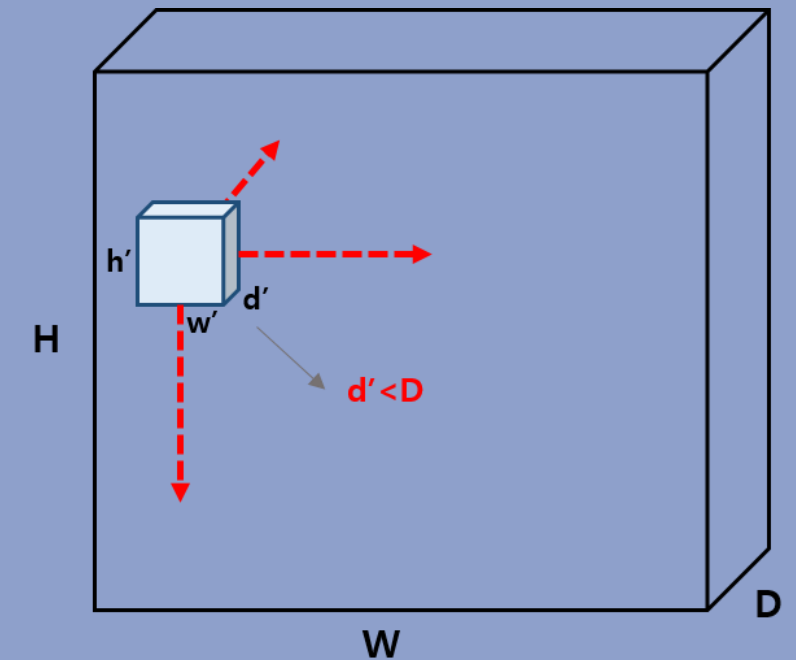
> 이동하는 방향의 수와 출력 형태에 따라 합성곱을 분류



[1D Convolution]



[2D Convolution]

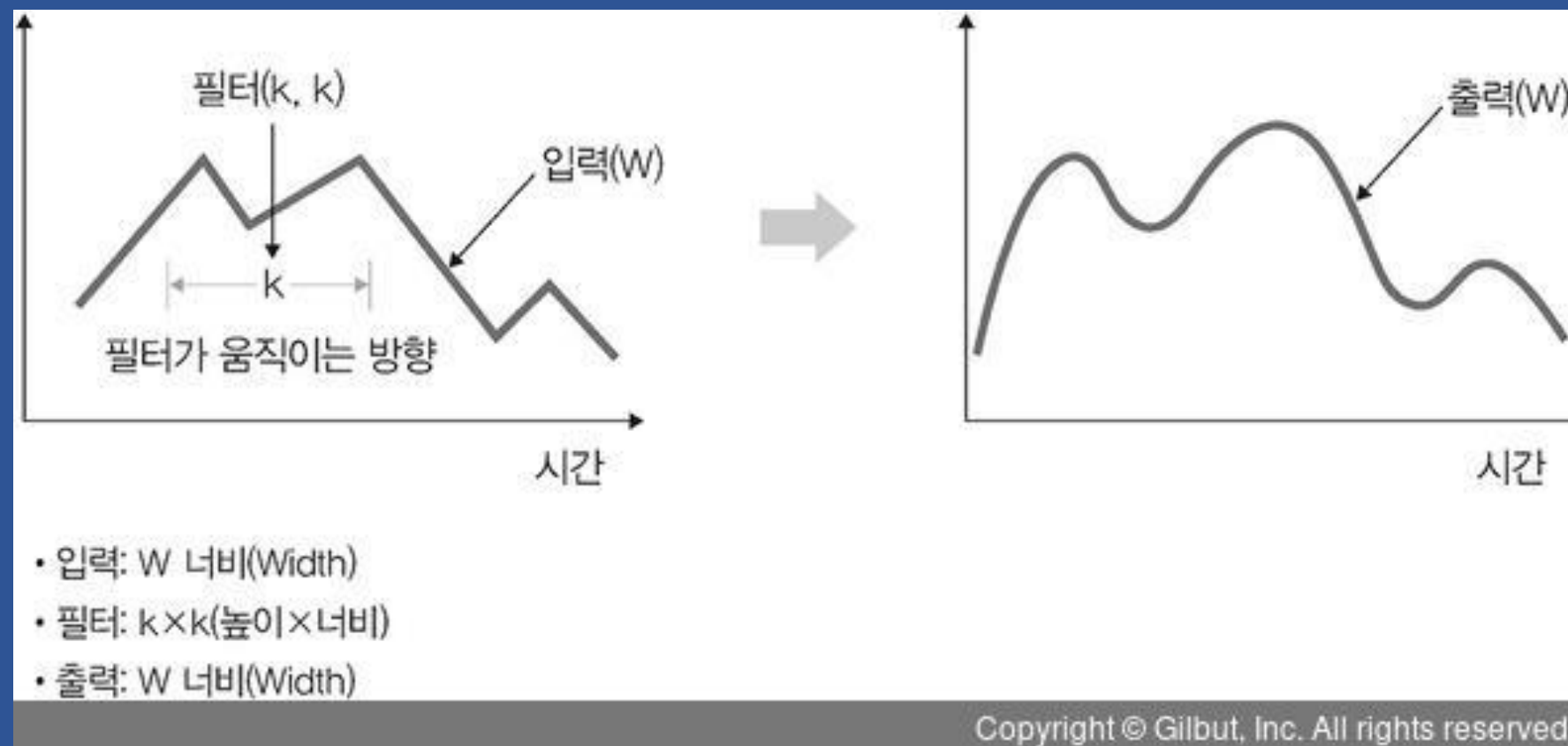


[3D Convolution]



1D 합성곱.

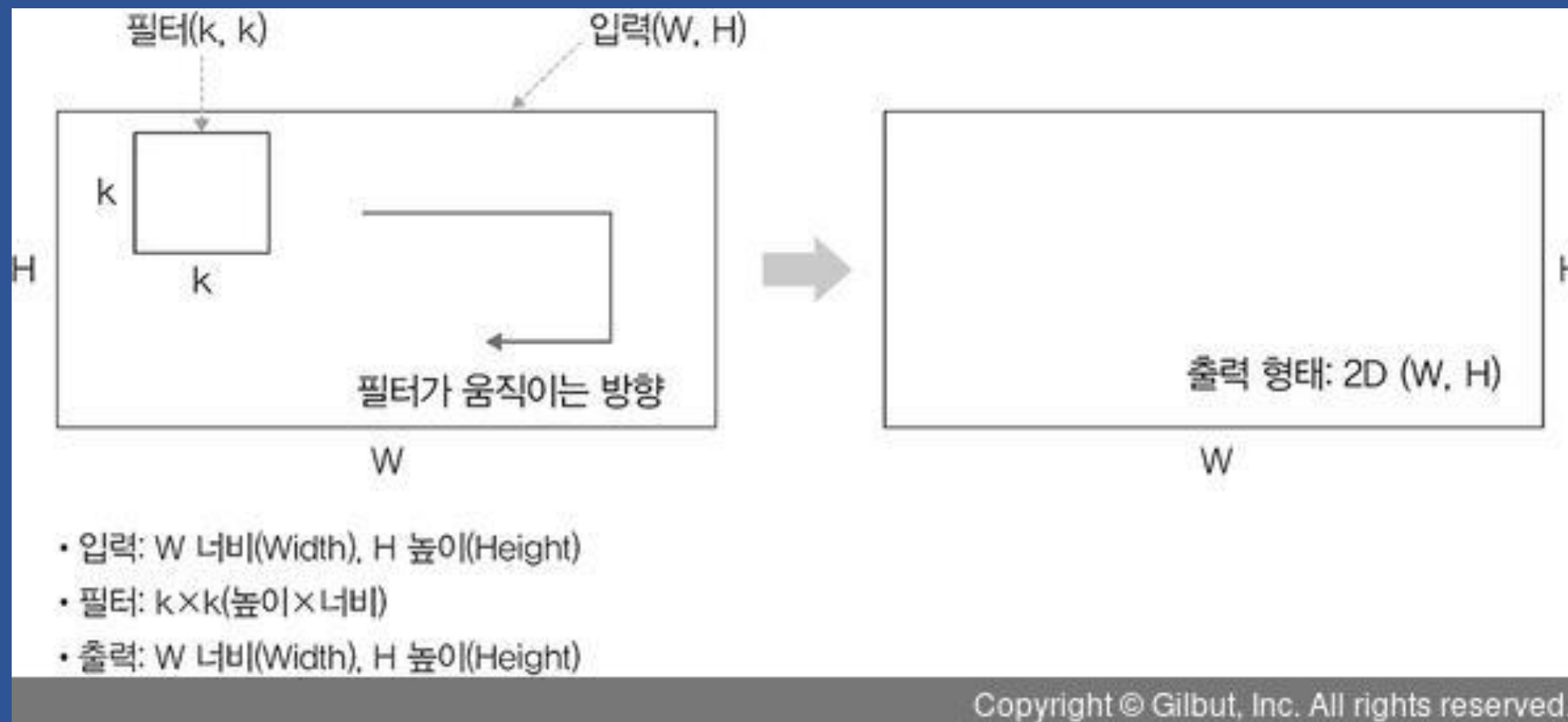
1D Convolution



- > 필터가 시간을 축으로 좌우로만 이동할 수 있는 합성곱
- > 출력 형태는 1D의 배열이 됨
- > 그래프 곡선을 완화할 때 많이 사용

2D 합성곱.

2D Convolution

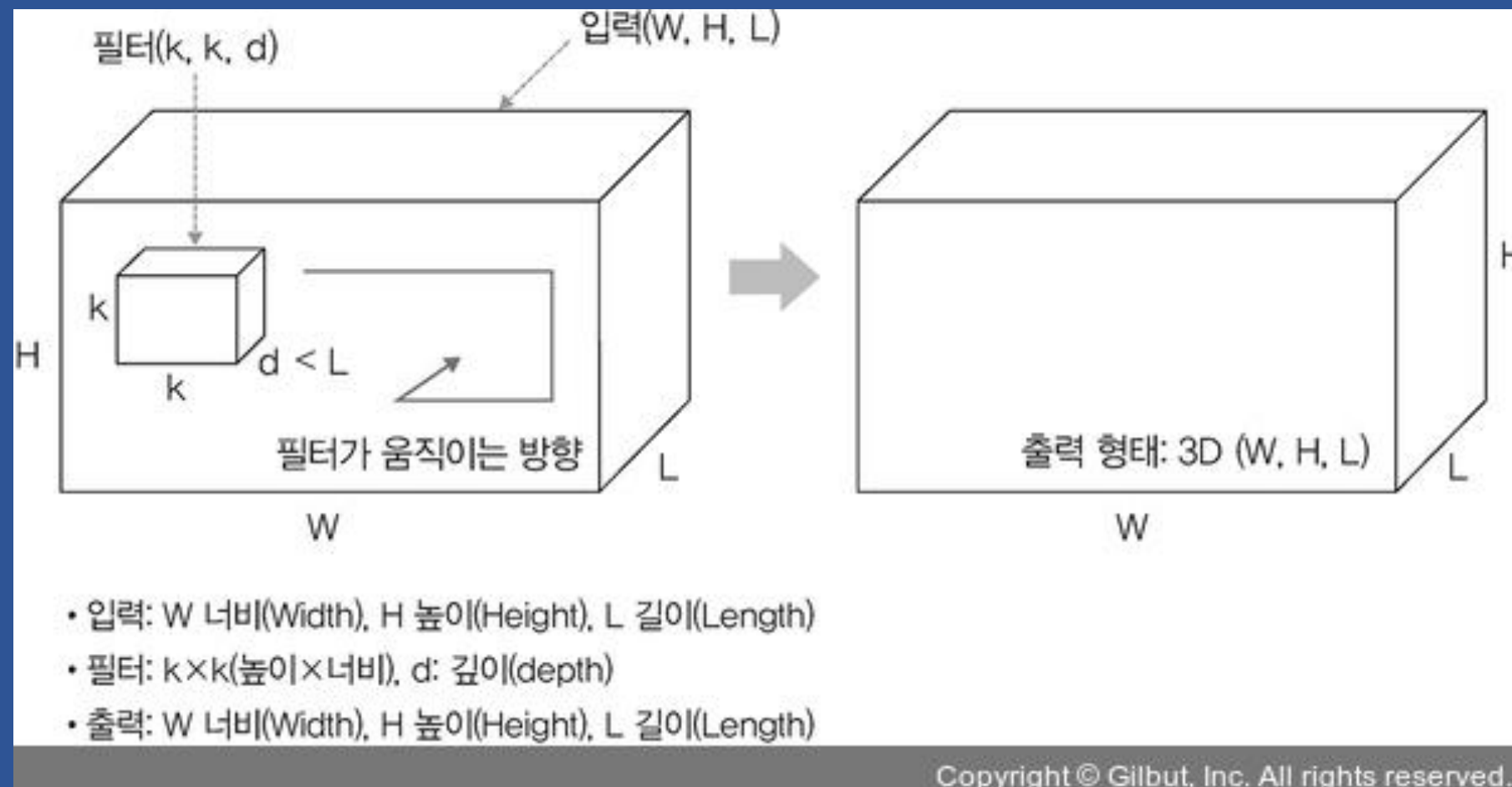


> 필터가 방향 두 개로 움직이는 형태

> 출력 형태는 2D의 행렬이 됨

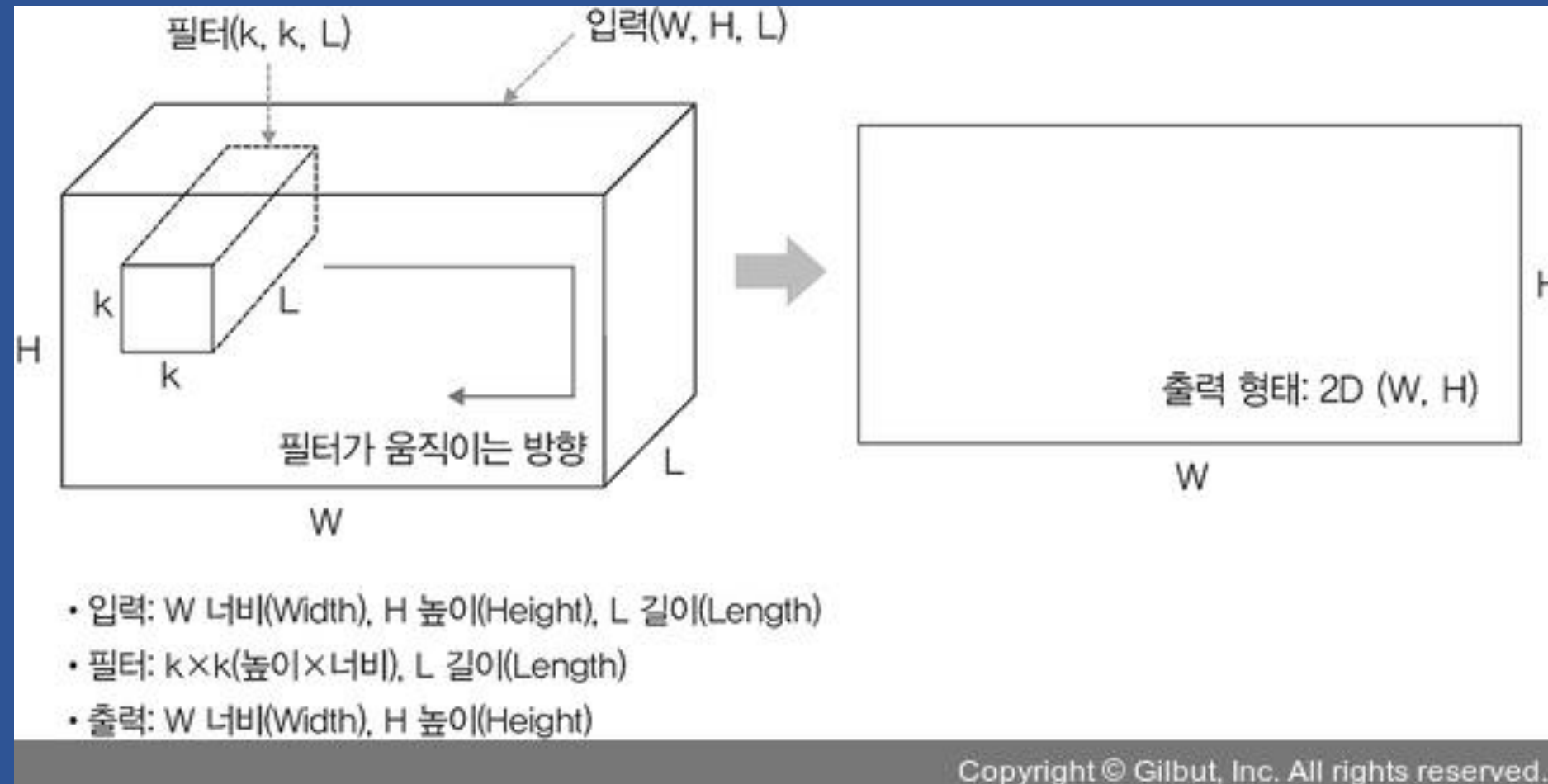
3D 합성곱.

3D Convolution



- > 필터가 움직이는 방향이 세 개인 합성곱
- > 필터의 깊이(d) < 입출력의 길이(L)를 유지하는 것이 중요

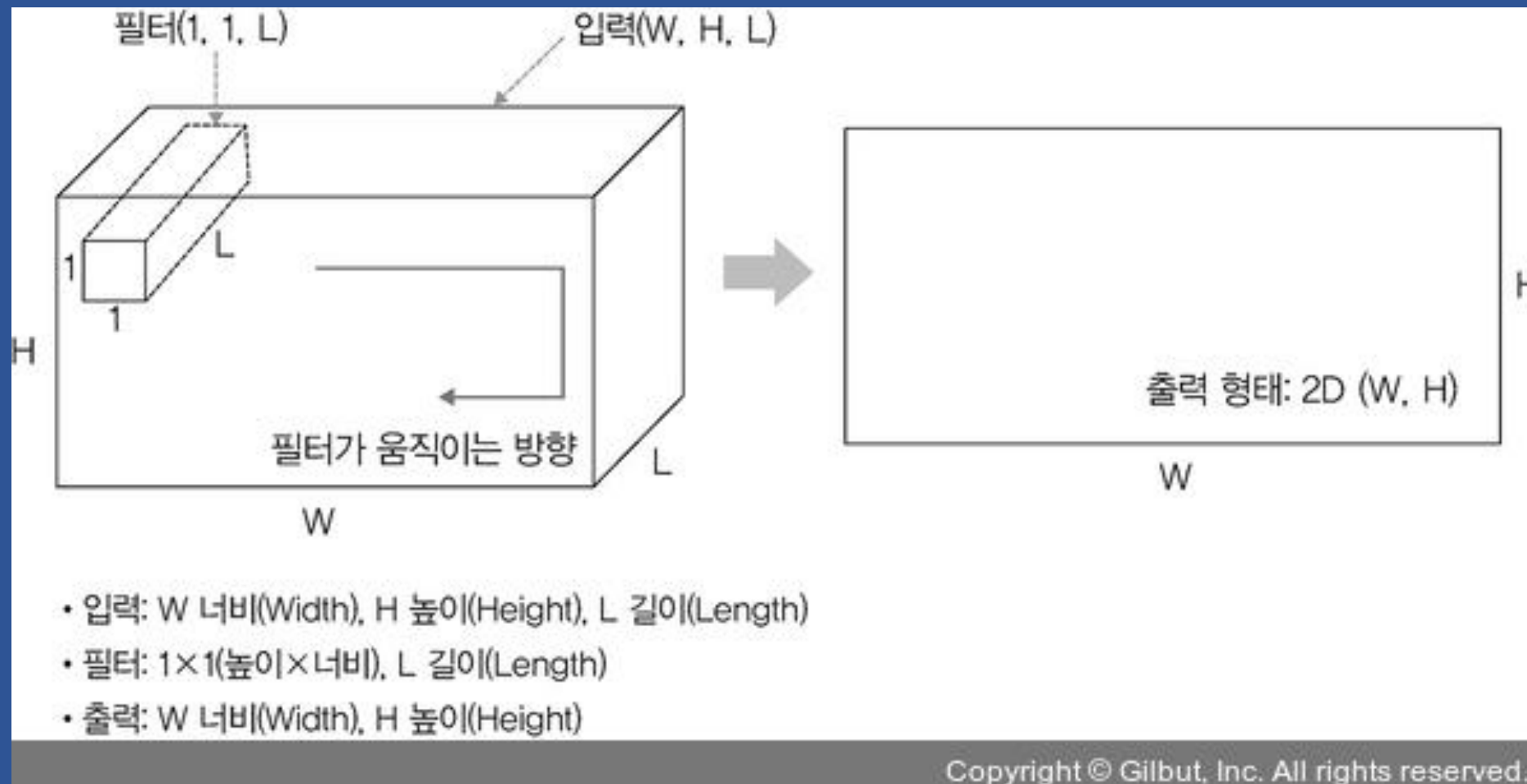
3D 입력을 갖는 2D 합성곱.



- > 입력이 3D 형태임에도 출력이 2D 형태로 나오는 합성곱
- > 필터의 길이(L) = 입출력의 길이(L)여야 하기 때문에 발생
- > LeNet-5, VGG

1 x 1 합성곱.

1 x 1 Convolution



- > 3D 형태로 입력되는 합성곱
- > 채널(=깊이) 수를 조정해 연산량이 감소되는 효과를 가짐
- > GoogLeNet

합성곱 신경망 맛보기'

➤ 데이터셋 불러오기

클래스별 label map을 만들고 랜덤으로 20개의 이미지를 불러옴

```
[5] labels_map = {0 : 'T-Shirt', 1 : 'Trouser', 2 : 'Pullover', 3 : 'Dress', 4 : 'Coat', 5 : 'Sandal', 6 : 'Shirt',
                  7 : 'Sneaker', 8 : 'Bag', 9 : 'Ankle Boot'} #10개의 클래스

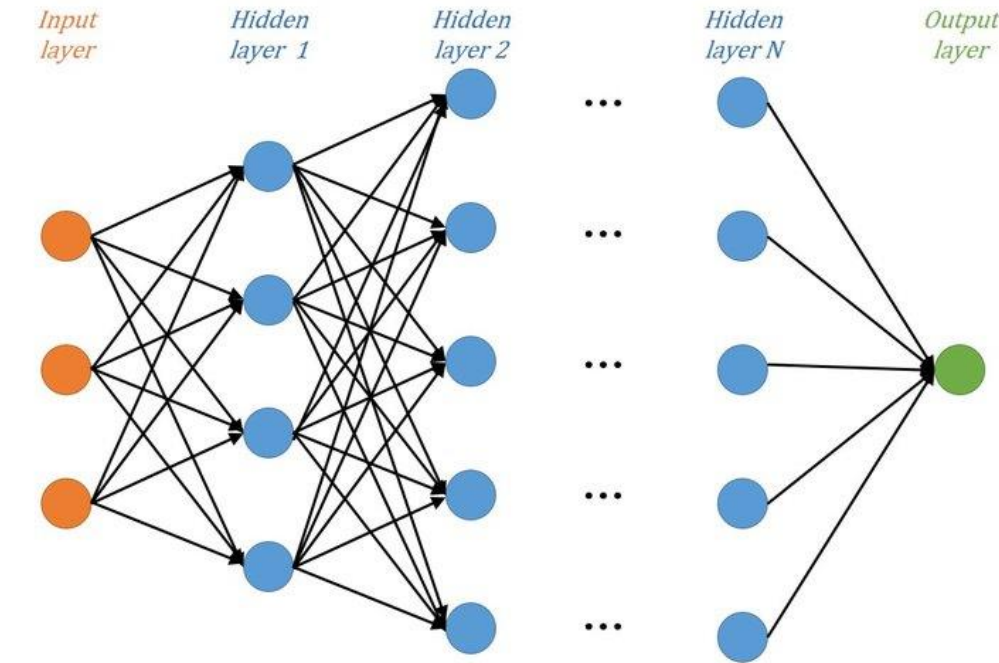
fig = plt.figure(figsize=(8,8)); #출력할 이미지의 가로세로 길이
columns = 4;
rows = 5;
for i in range(1, columns*rows +1):
    img_xy = np.random.randint(len(train_dataset)); #무작위 표본을 추출할 때 사용
    img = train_dataset[img_xy][0][0,:,:] #3차원 배열 생성
    fig.add_subplot(rows, columns, i)
    plt.title(labels_map[train_dataset[img_xy][1]])
    plt.axis('off')
    plt.imshow(img, cmap='gray')
plt.show() #20개의 이미지 표현
```



합성곱 신경망 맛보기

➤ 심층 신경망 네트워크 생성

ConvNet이 적용되지 않은 네트워크, DNN이라고도 함



```
class FashionDNN(nn.Module):
    def __init__(self):
        super(FashionDNN, self).__init__()
        self.fc1 = nn.Linear(in_features=784, out_features=256) #input size 784, output size 256
        self.drop = nn.Dropout2d(0.25) # dropout 진행, 0.25만큼의 비율
        self.fc2 = nn.Linear(in_features=256, out_features=128)
        self.fc3 = nn.Linear(in_features=128, out_features=10)

    def forward(self, input_data): #모델이 forward propagation 진행
        out = input_data.view(-1, 784) #numpy의 reshape과 같은 기능, input data를 (?, 784)의 2차원 텐서로 변환
        out = F.relu(self.fc1(out)) #활성화 함수(relu) 지정
        out = self.drop(out)
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        return out
```

합성곱 신경망 맛보기

➤ 모델 학습

최종 Accuracy는 약 86.65%

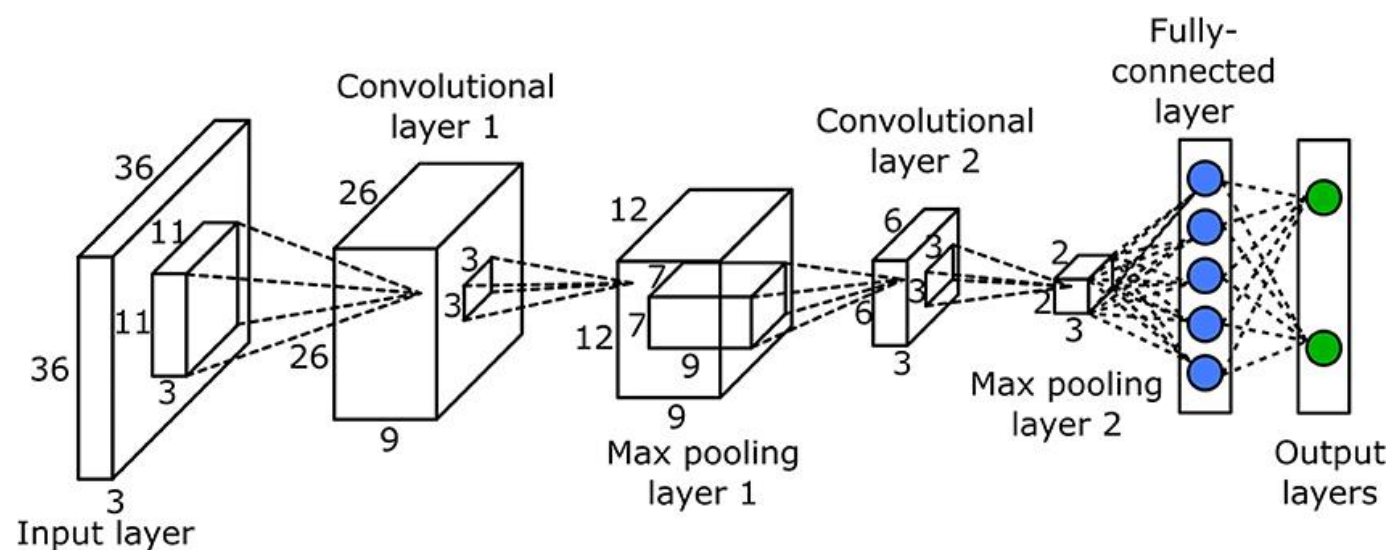
```
Iteration: 500, Loss: 0.5497425198554993, Accuracy: 83.3499984741211%  
Iteration: 1000, Loss: 0.48211437463760376, Accuracy: 84.30999755859375%  
Iteration: 1500, Loss: 0.3537752628326416, Accuracy: 84.63999938964844%  
Iteration: 2000, Loss: 0.35956668853759766, Accuracy: 85.33999633789062%  
Iteration: 2500, Loss: 0.25863322615623474, Accuracy: 86.63999938964844%  
Iteration: 3000, Loss: 0.28179723024368286, Accuracy: 86.6500015258789%
```

합성곱 신경망 맛보기

➤ 합성곱 네트워크 생성

```
class FashionCNN(nn.Module):
    def __init__(self):
        super(FashionCNN, self).__init__()
        self.layer1 = nn.Sequential( #첫번째 레이어
            nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, padding=1), #합성곱층(conv layer)
            nn.BatchNorm2d(32), #batch normalization
            nn.ReLU(), #ReLU 활성화함수
            nn.MaxPool2d(kernel_size=2, stride=2) #max pooling layer
        )
        self.layer2 = nn.Sequential( #두번째 레이어
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.fc1 = nn.Linear(in_features=64*6*6, out_features=600) #공식을 이용해 1차원으로 변경
        self.drop = nn.Dropout2d(0.25) #dropout(위랑 똑같은)
        self.fc2 = nn.Linear(in_features=600, out_features=120) #fc layer
        self.fc3 = nn.Linear(in_features=120, out_features=10) #마지막 out은 클래스의 개수를 의미함

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1) #합성곱층에서 fc layer로 연결되는 부분이기 때문에 1차원으로 바꿔줌
        out = self.fc1(out)
        out = self.drop(out)
        out = self.fc2(out)
        out = self.fc3(out)
        return out
```



합성곱 신경망 맛보기

➤ 모델 학습

최종 Accuracy는 약 89.43%으로 DNN보다 높아진 것을 확인할 수 있음

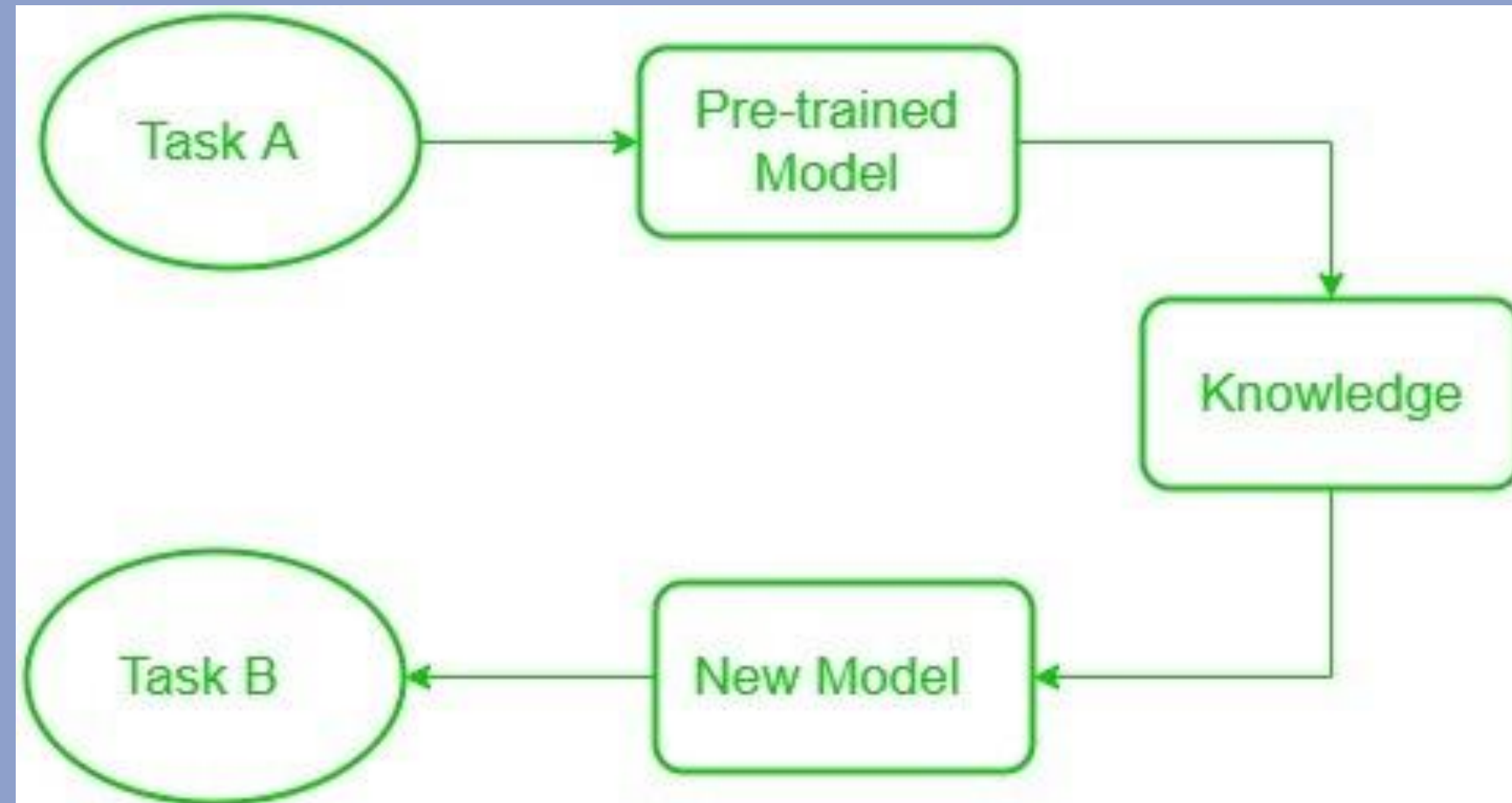
```
Iteration: 500, Loss: 0.4876278340816498, Accuracy: 87.18000030517578%  
Iteration: 1000, Loss: 0.35017630457878113, Accuracy: 87.12999725341797%  
Iteration: 1500, Loss: 0.28021568059921265, Accuracy: 88.69999694824219%  
Iteration: 2000, Loss: 0.19052854180335999, Accuracy: 89.6199951171875%  
Iteration: 2500, Loss: 0.14415787160396576, Accuracy: 89.94999694824219%  
Iteration: 3000, Loss: 0.20658329129219055, Accuracy: 89.43000030517578%
```

전이학습.

Transfer Learning

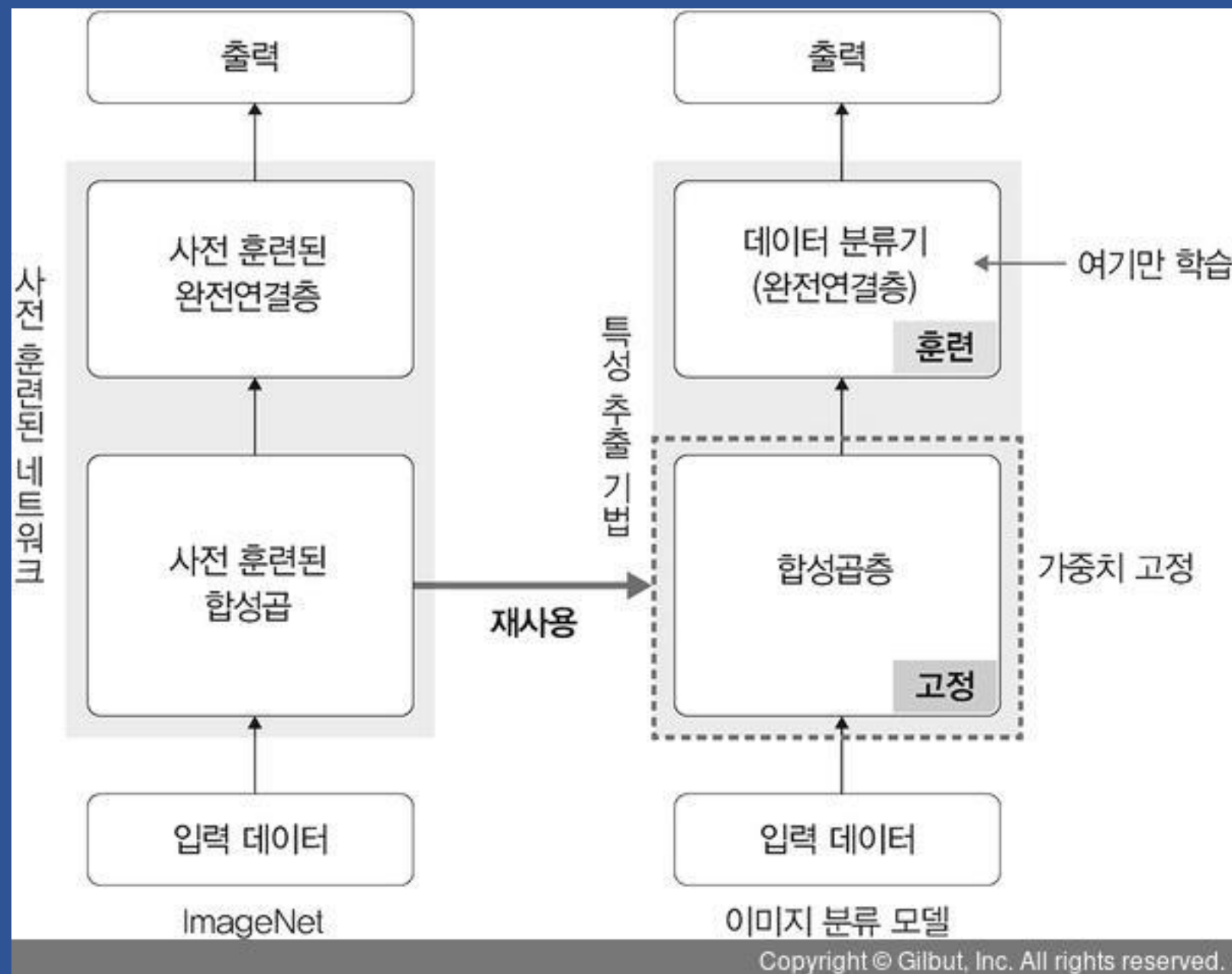


- 사전 훈련된 모델의 가중치를 가져와서 사용하는 방법
적은 수의 이미지로도 좋은 성능을 얻을 수 있다



특성 추출 기법.

Feature extractor



➤ 사전 훈련된 모델을 가져온 후 FC layer 부분만 새로 만듦

➤ 이미지 분류를 위해 두 부분으로 나뉨

합성곱층: 합성곱층과 풀링층으로 구성

데이터 분류기(Fc layer): 추출된 특성을 입력받아 최종적으로 이미지 클래스를 분류

➤ 사용 가능한 모델

Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet

특성 추출 기법.

Feature extractor


```
data_path = "../drive/My Drive/sorce/chap05/data/catanddog/train/"  
#local에 했을 경우  
#data_path = '/catanddog/train/' 이나 자기가 파일 위치한 곳으로 바꾸기  
transform = transforms.Compose(  
    [  
        transforms.Resize([256, 256]), #이미지 크기 조정  
        transforms.RandomResizedCrop(224), #이미지 랜덤한 비율로 자른 후 크기 조정,데이터 확장 용도  
        transforms.RandomHorizontalFlip(), #이미지를 랜덤하게 수평으로 뒤집음  
        transforms.ToTensor(), #텐서로 변환  
    ]  
)  
train_dataset = torchvision.datasets.ImageFolder(  
    data_path,  
    transform=transform  
)  
train_loader = torch.utils.data.DataLoader(  
    train_dataset,  
    batch_size=32, #한 번에 불러올 데이터양  
    num_workers=8, #하위 프로세스 개수  
    shuffle=True #무작위로 섞을 것인지 결정  
)  
  
print(len(train_dataset)) #데이터의 개수 출력
```

385

특성 추출 기법.

Feature extractor

```
import numpy as np
#samples, labels = iter(train_loader).next() 오류나시는 분은 바로 밑에 걸로 실행~ pytorch 버전 문제인듯해용
samples, labels = next(iter(train_loader))
classes = {0:'cat', 1:'dog'}
fig = plt.figure(figsize=(16,24))
for i in range(24):
    a = fig.add_subplot(4,6,i+1)
    a.set_title(classes[labels[i].item()])
    a.axis('off')
    a.imshow(np.transpose(samples[i].numpy(), (1,2,0)))
plt.subplots_adjust(bottom=0.2, top=0.6, hspace=0)
```



> 학습에 사용될 이미지 미리보기

특성 추출 기법.

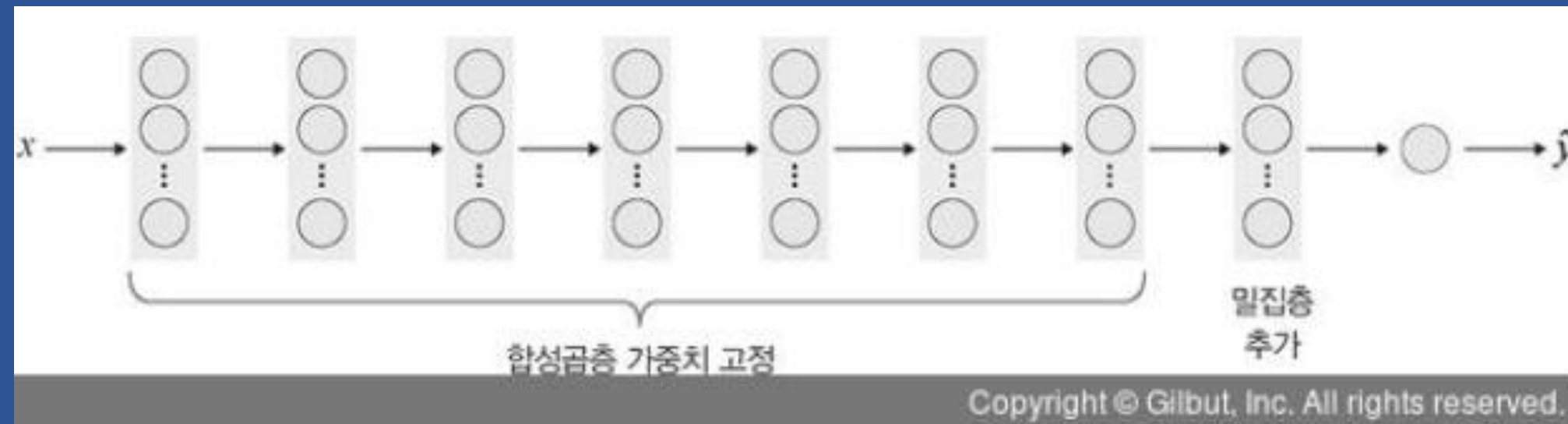
Feature extractor

```
resnet18 = models.resnet18(pretrained=True) #ResNet18 모델
```

/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:208: UserWarning: warnings.warn(
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints
100% 44.7M/44.7M [00:00<00:00, 86.4MB/s]

ResNet18에 FC layer 추가

```
[ ] resnet18.fc = nn.Linear(512, 2) #2는 클래스가 2개라는 뜻(cat 아니면 dog)
```



➤ Pretrained된 모델을 불러오고 FC layer 추가

특성 추출 기법.

Feature extractor

모델의 파라미터 값 확인

```
▶ for name, param in resnet18.named_parameters():
    if param.requires_grad:
        print(name, param.data)

fc.weight tensor([[ 0.0375,  0.0057,  0.0145, ..., -0.0073, -0.0414, -0.0175],
                  [ 0.0427,  0.0148,  0.0182, ..., -0.0146, -0.0393, -0.0294]])
fc.bias tensor([-0.0392, -0.0001])
```

```
▶ model = models.resnet18(pretrained = True) #모델 객체 생성

for param in model.parameters(): #conv layer 가중치 고정
    param.requires_grad = False

model.fc = torch.nn.Linear(512, 2)
for param in model.fc.parameters(): #fc layer는 학습
    param.requires_grad = True

optimizer = torch.optim.Adam(model.fc.parameters())
cost = torch.nn.CrossEntropyLoss() #손실함수 정의
print(model)

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
)
```

특성 추출 기법.

Feature extractor

```
[ ] device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    criterion = nn.CrossEntropyLoss()
    train_acc_hist, train_loss_hist = train_model(resnet18, train_loader, criterion, optimizer, device)
```

Epoch 0/12

Loss: 0.4269 Acc: 0.8208

Epoch 1/12

Loss: 0.3737 Acc: 0.8286

Epoch 2/12

Loss: 0.2843 Acc: 0.9143

Epoch 3/12

Loss: 0.2748 Acc: 0.8961

Epoch 4/12

Loss: 0.2238 Acc: 0.9091

Epoch 5/12

Loss: 0.2108 Acc: 0.9273

Epoch 12/12

Loss: 0.2384 Acc: 0.8987

Training complete in 0m 45s

Best Acc: 0.935065

특성 추출 기법.

Feature extractor

```
test_path = "../drive/My Drive/sorce/chap05/data/catanddog/test"

transform = transforms.Compose(
    [
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])
test_dataset = torchvision.datasets.ImageFolder(
    root=test_path,
    transform=transform
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=32,
    num_workers=1,
    shuffle=True
)

print(len(test_dataset))
```

98

테스트 데이터를 평가 함수에 적용

```
[ ] val_acc_hist = eval_model(resnet18, test_loader, device)
```

```
saved_model ['./drive/My Drive/sorce/chap05/data/catanddog/00.pth', '..
Loading model ./drive/My Drive/sorce/chap05/data/catanddog/00.pth
Acc: 0.8878
```

```
Loading model ./drive/My Drive/sorce/chap05/data/catanddog/01.pth
Acc: 0.9082
```

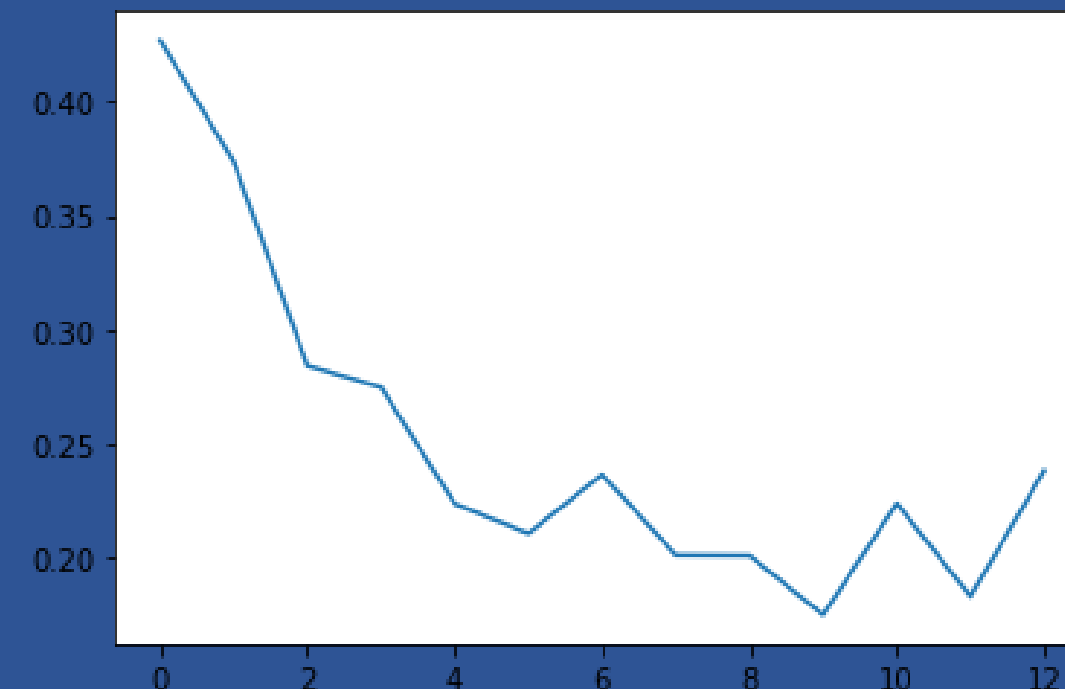
```
Loading model ./drive/My Drive/sorce/chap05/data/catanddog/02.pth
Acc: 0.9184
```

특성 추출 기법.

Feature extractor



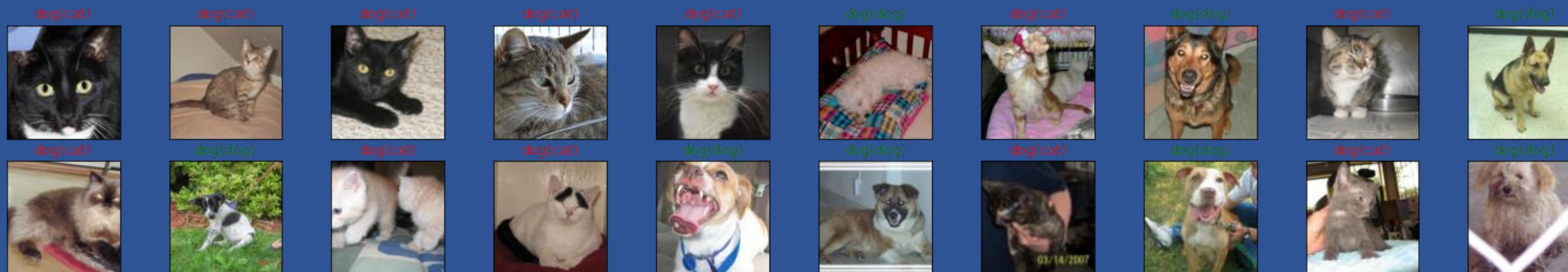
➤ Train/test data 정확도 그래프



➤ Train data 오차 그래프

특성 추출 기법.

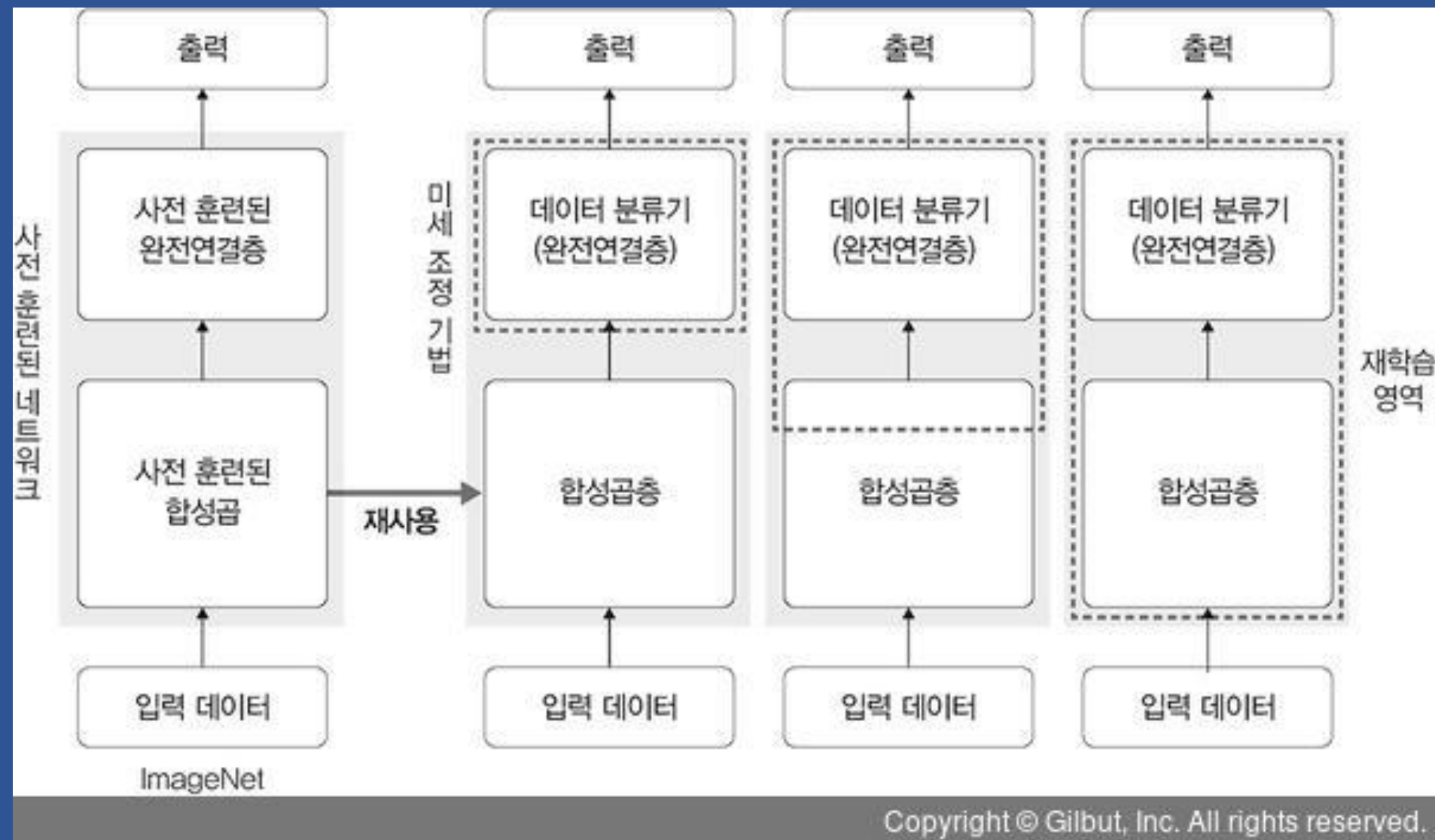
Feature extractor



> 빨간색은 잘못 예측한 것, 초록색은 잘 예측한 것

미세 조정 기법.

Fine-tuning



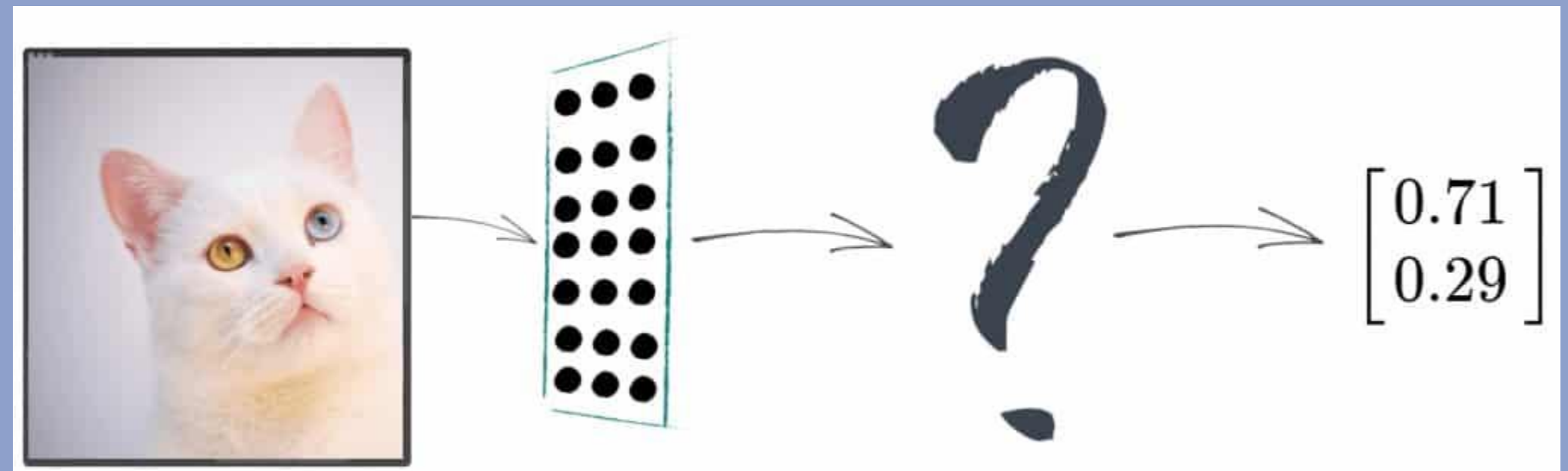
- ▶ 사전 훈련된 모델과 합성곱층, 데이터 분류기의 가중치 업데이트
- ▶ 모델의 파라미터를 조정하는 과정
- ▶ 훈련시키려는 데이터셋의 크기와 사전 훈련 모델에 따라 전략을 세움

설명 가능한 CNN.

Explainable CNN

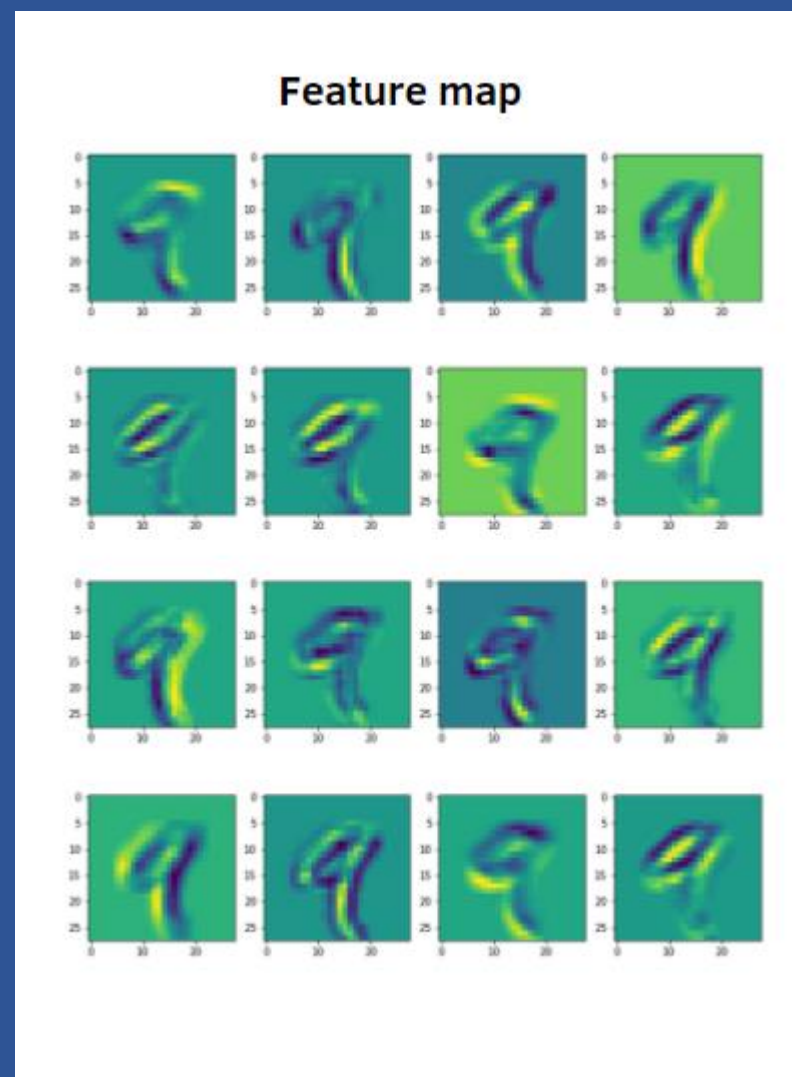


- 딥러닝 처리 결과를 사람이 이해할 수 있는 방식으로 제시하는 기술
CNN은 내부에서 어떻게 동작하는지 알기 어려워 처리 과정을 시각화할 필요성이 있음



특성 맵 시각화.

Feature map visualization



> Feature map?

활성화 맵이라고도 하며, 입력 이미지 또는 필터를 입력에 적용한 결과

즉 특성 맵을 시각화한다 -> 특성 맵에서 입력되는 특성을 감지하는 방법을 이해할 수 있도록 돕는다

특성 맵 시각화.

Feature map visualization

```
class XAI(torch.nn.Module):
    def __init__(self, num_classes=2):
        super(XAI, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Dropout(0.3),
            nn.Conv2d(64, 64, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.4),
            nn.Conv2d(128, 128, kernel_size=3, padding = 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
```

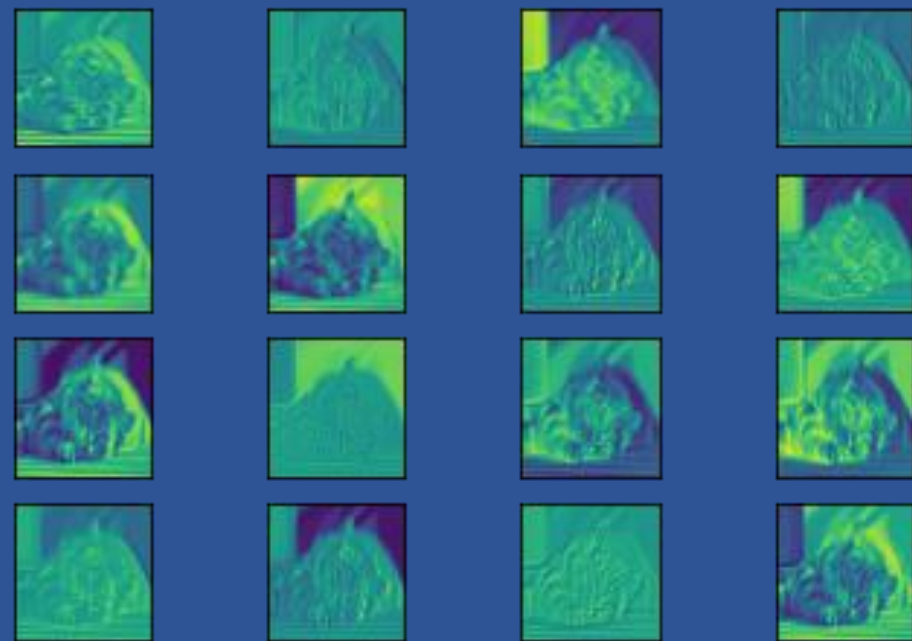
```
[ ] model=XAI()
model.cpu() #모델에 입력되는 이미지를 넘파이로 받아오는 부분때문에 CPU를 사용하도록 지정하였습니다
model.eval()

XAI(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.3, inplace=False)
    (4): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

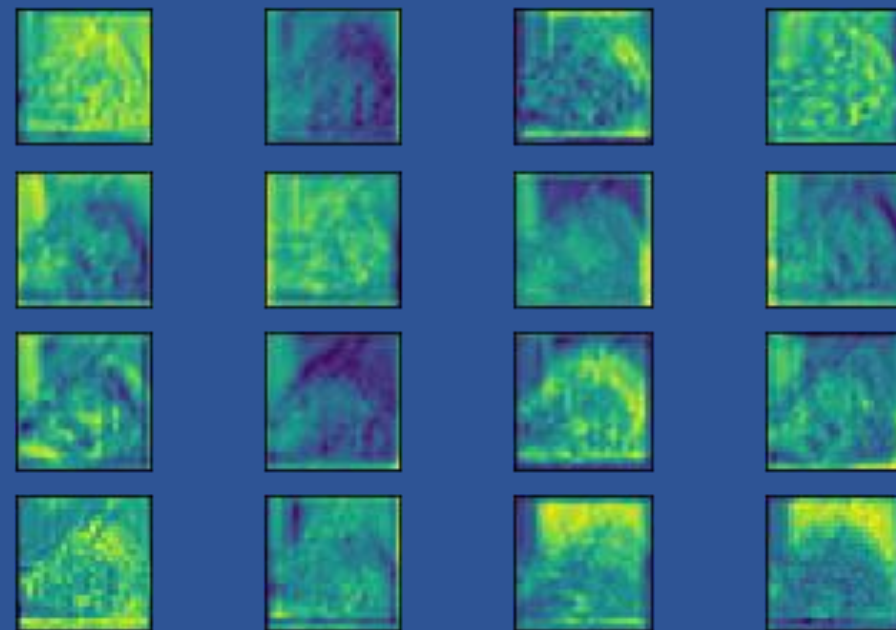
➤ 설명 가능한 모델을 위해 XAI 네트워크 생성 & 객체화

특성 맵 시각화.

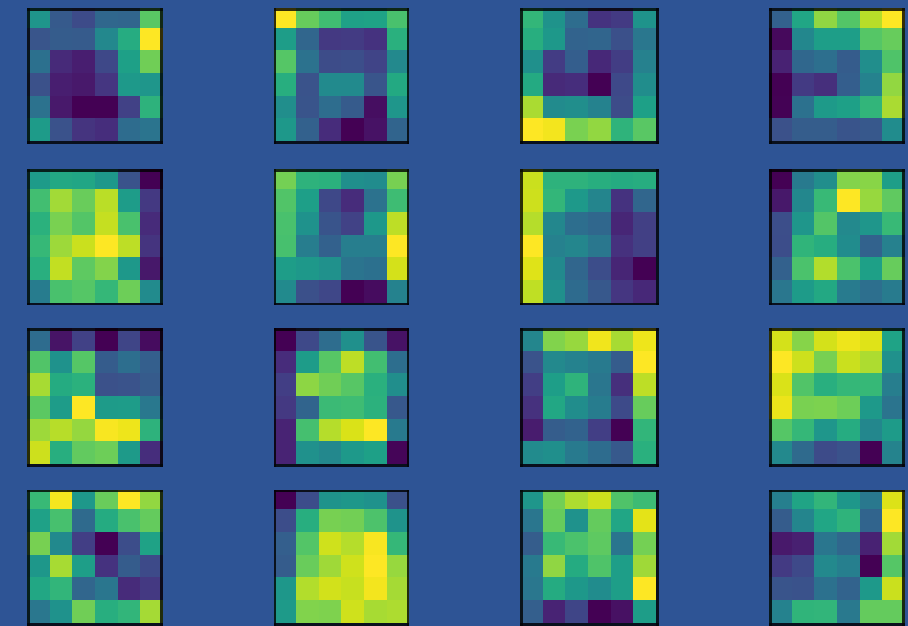
Feature map visualization



➤ 첫번째 계층에 대한 특성 맵



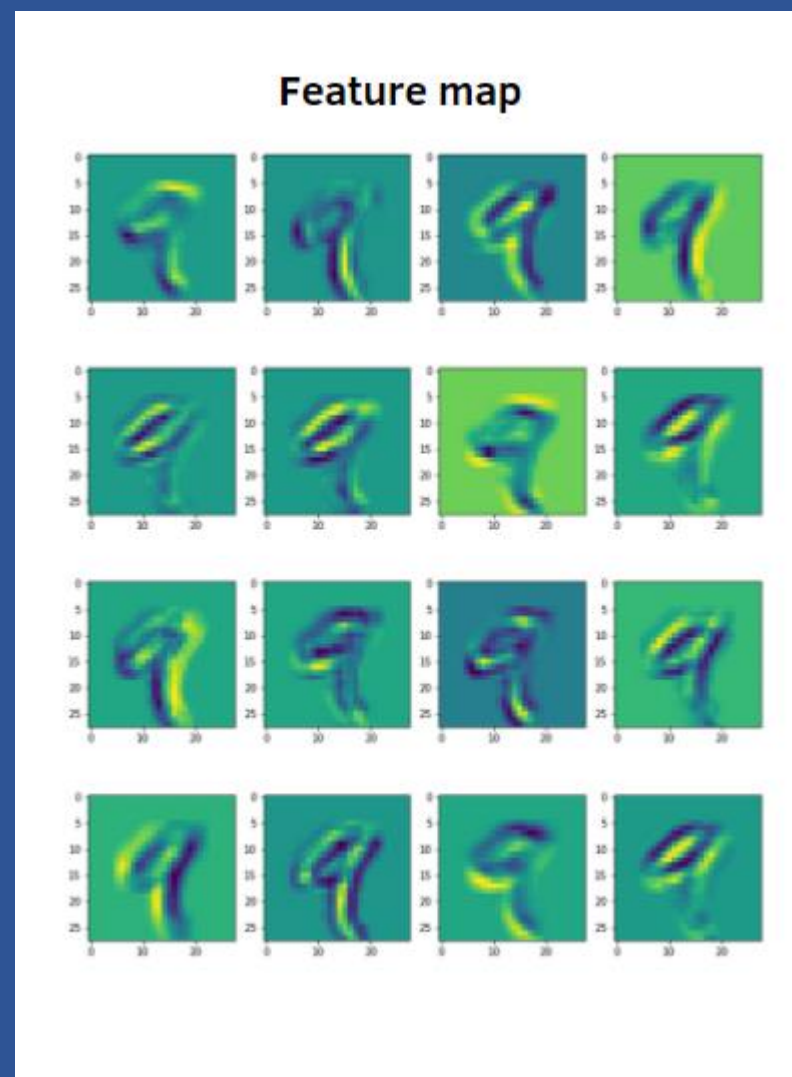
➤ 20번째 계층에 대한 특성 맵



➤ 40번째 계층에 대한 특성 맵

특성 맵 시각화.

Feature map visualization



> Feature map?

활성화 맵이라고도 하며, 입력 이미지 또는 필터를 입력에 적용한 결과

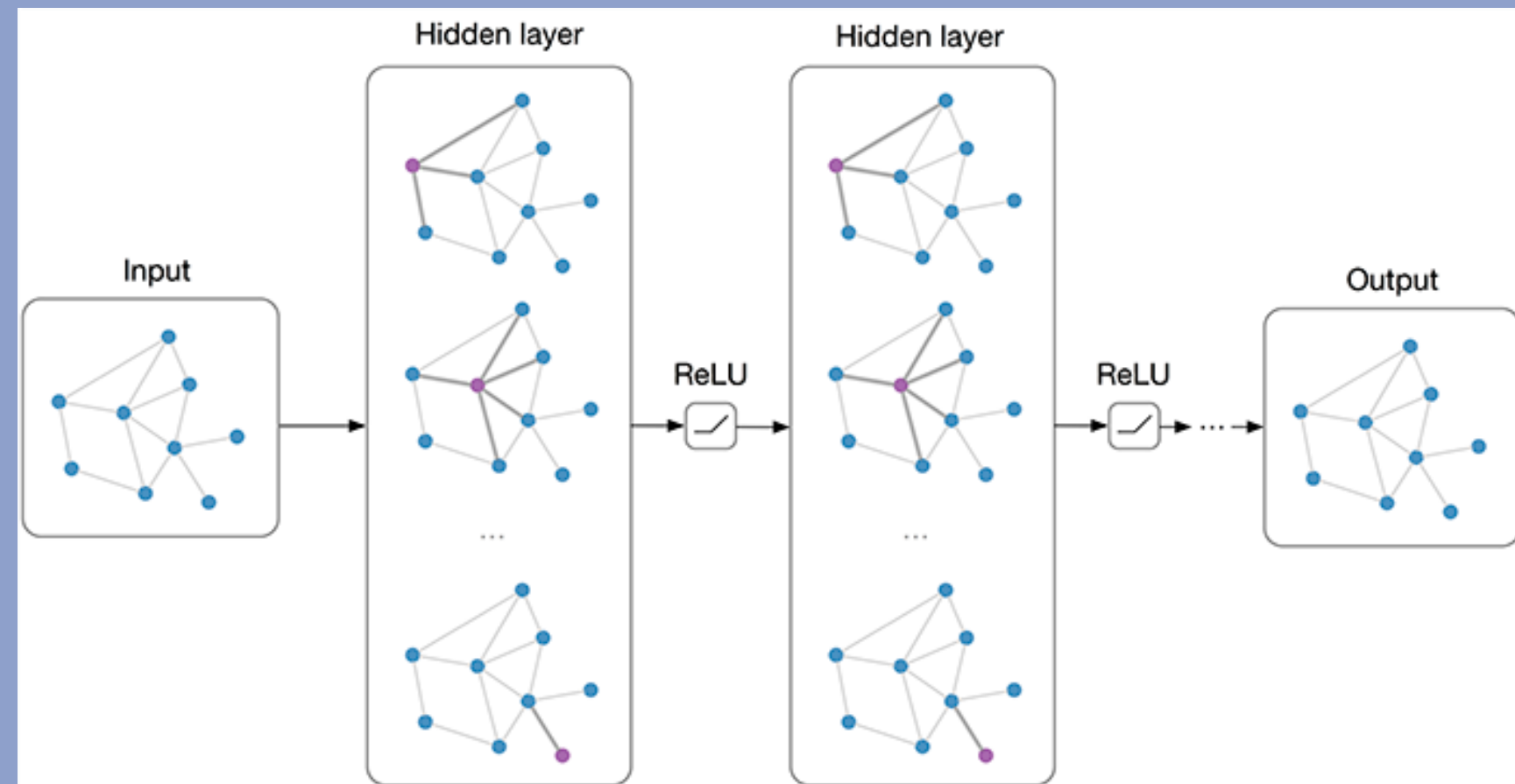
즉 특성 맵을 시각화한다 -> 특성 맵에서 입력되는 특성을 감지하는 방법을 이해할 수 있도록 돕는다

그래프 합성곱 네트워크.

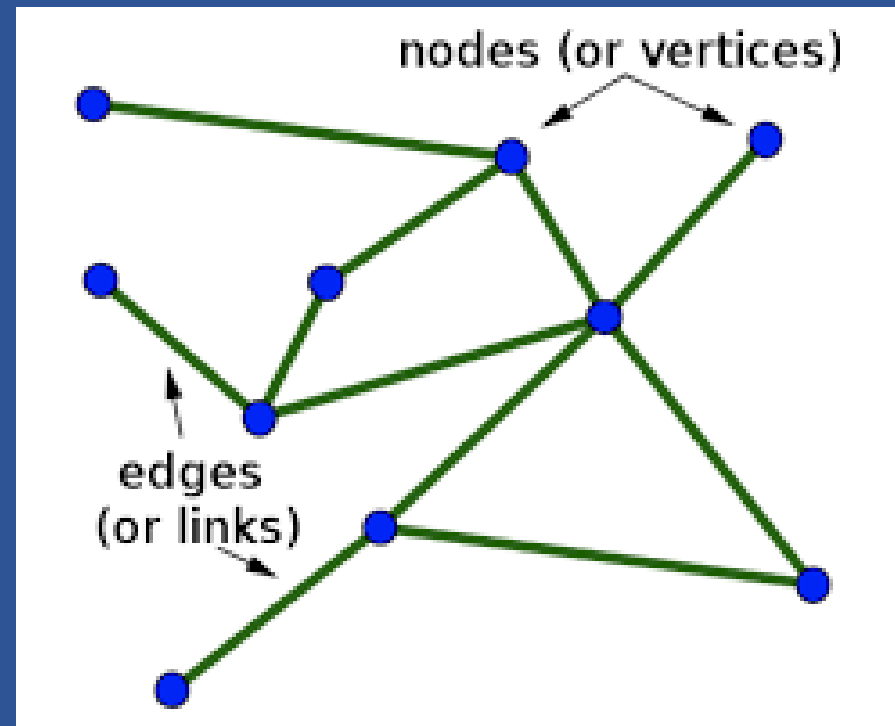
Graph convolutional network



> 그래프 데이터를 위한 신경망



그래프란?



➤ 방향성이 있거나 없는 에지로 연결된 노드의 집합

➤ 노드(node, vertex)

그림의 파란색 원. 즉 요소들을 의미한다

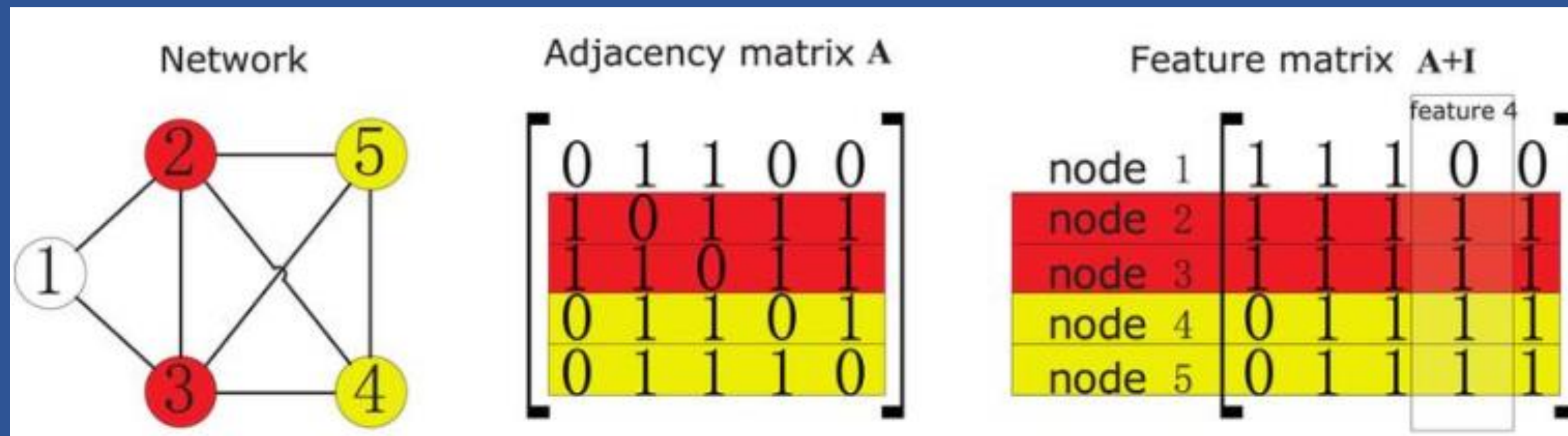
➤ 에지(edge, link)

그림의 두 노드를 연결한 초록색 선. 즉 결합 방법(single, double, automatic 등)을 의미한다

그래프 신경망.

Graph Neural Network

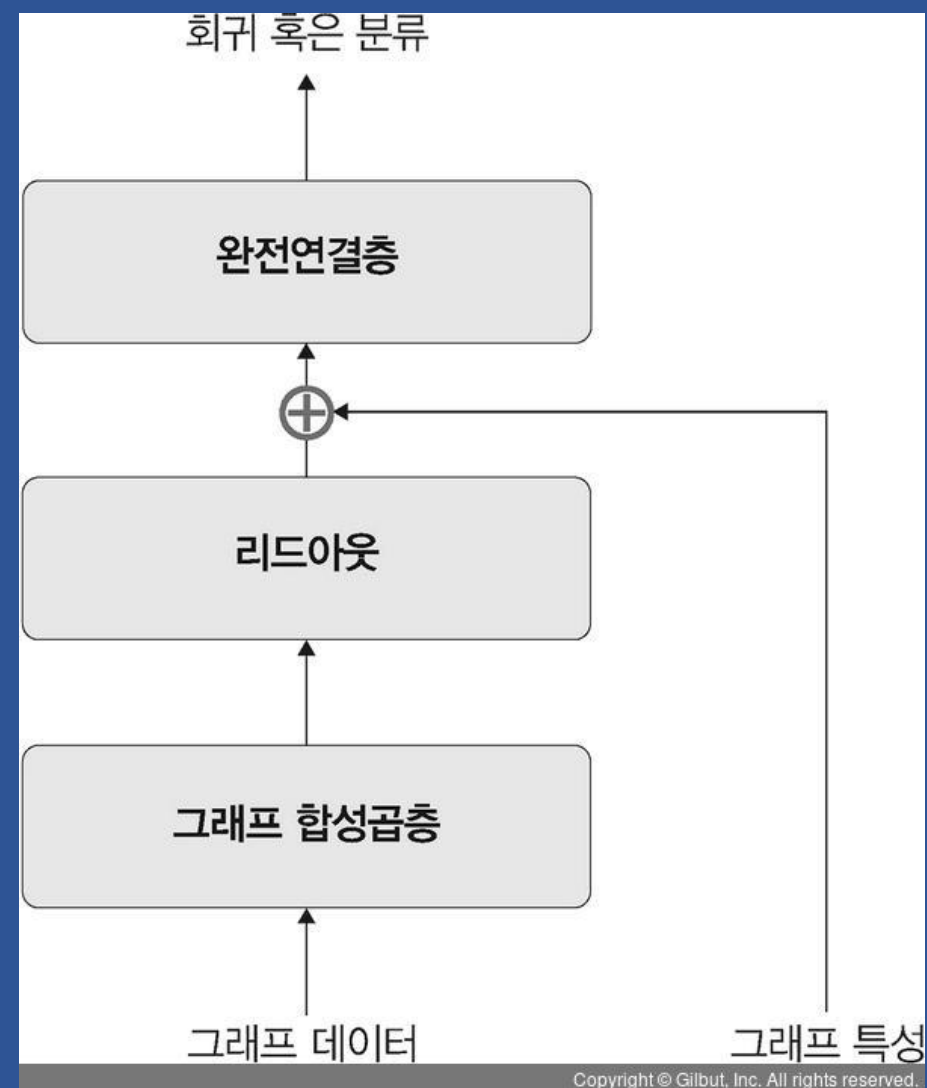
▶ 그래프 구조에서 사용하는 신경망



그래프 합성곱 네트워크.

Graph Convolutional Network

> 이미지에 대한 합성곱을 그래프 데이터로 확장한 알고리즘



> 리드아웃(readout)

특성 행렬을 하나의 벡터로 변환하는 함수

> 그래프 합성곱층(graph convolutional layer)

GCN에서 가장 중요한 부분

그래프 형태의 데이터를 행렬 데이터로 변환해 딥러닝 알고리즘을 적용 가능하게 만듦

감사합니다 •

20기 분석 송여진