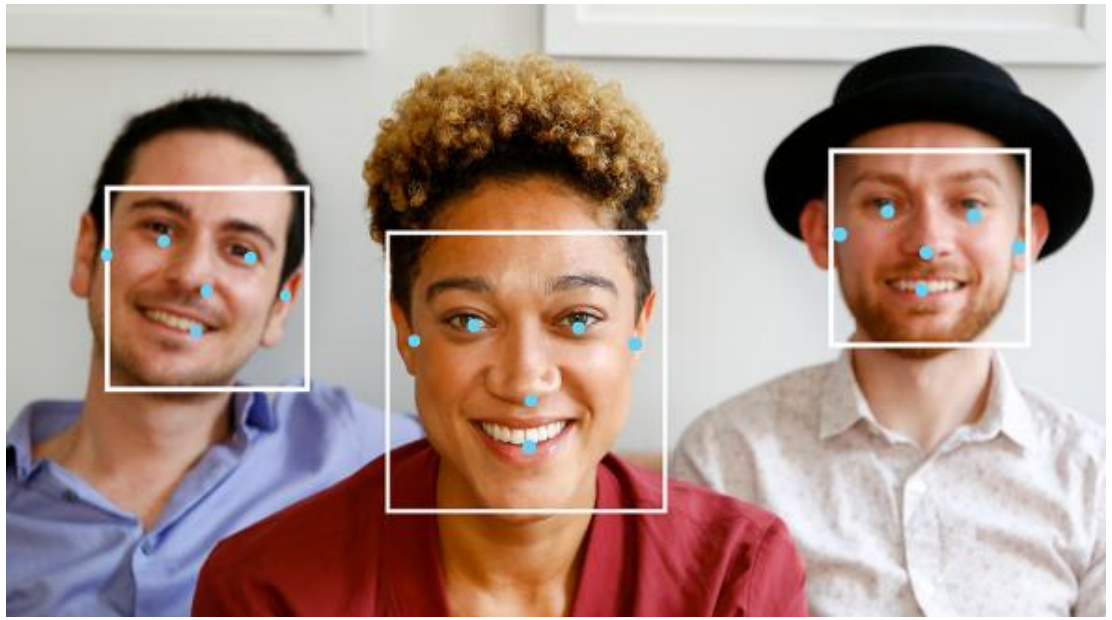
# CNN
# (Convolutional Neural Network)

# 얼굴인식



기존의 MLP

얼굴 영역에 해당하기 위한 조합 학습

사진

픽셀 값, 에지 등 등…

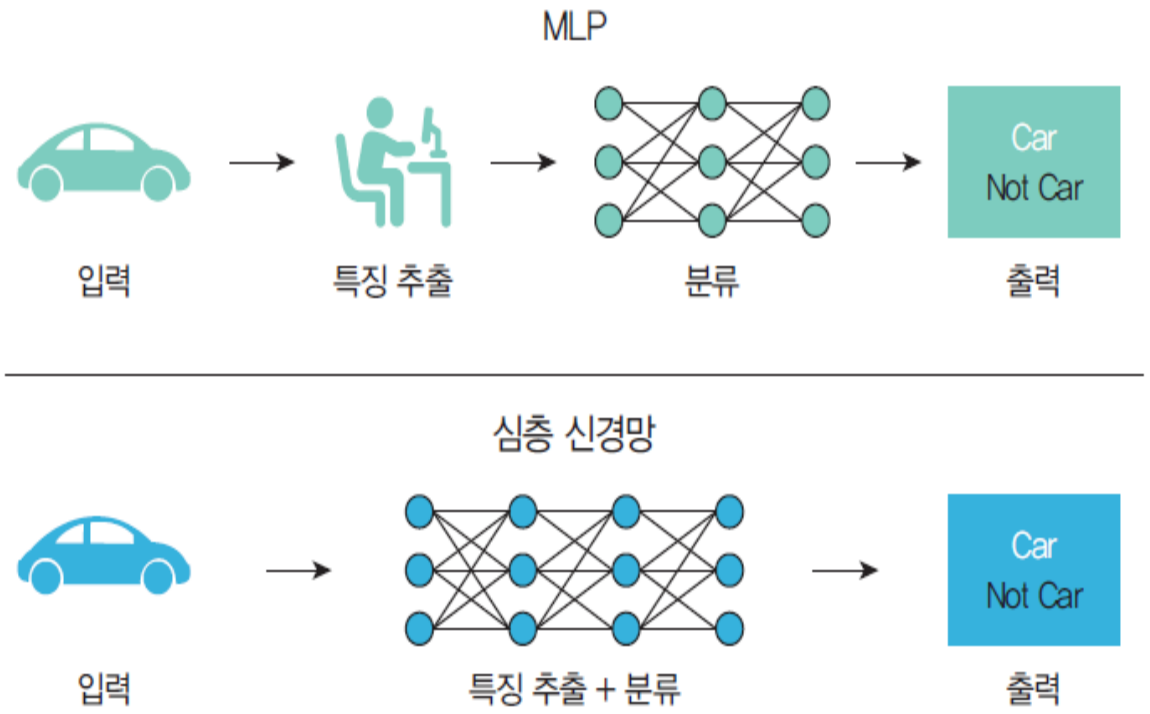
얼굴의 픽셀 값,
(피부 색, 치아, 입술…)
에지 사이의 거리

심층 신경망

사진

사진에서 얼굴영역으로 선택하기 위한
모든 결정 조건 학습

# 필요성




경험에 의해 판단에 중요하다고 생각하는 특징을 추출하여 입력
➔ 판단만 학습

특징도 알아서 뽑아 낼 수 있지 않을까?

선입견없이 원하는 목적함수에 더 좋은 결과가 낼 수 있는 특징 추출

# 필요성

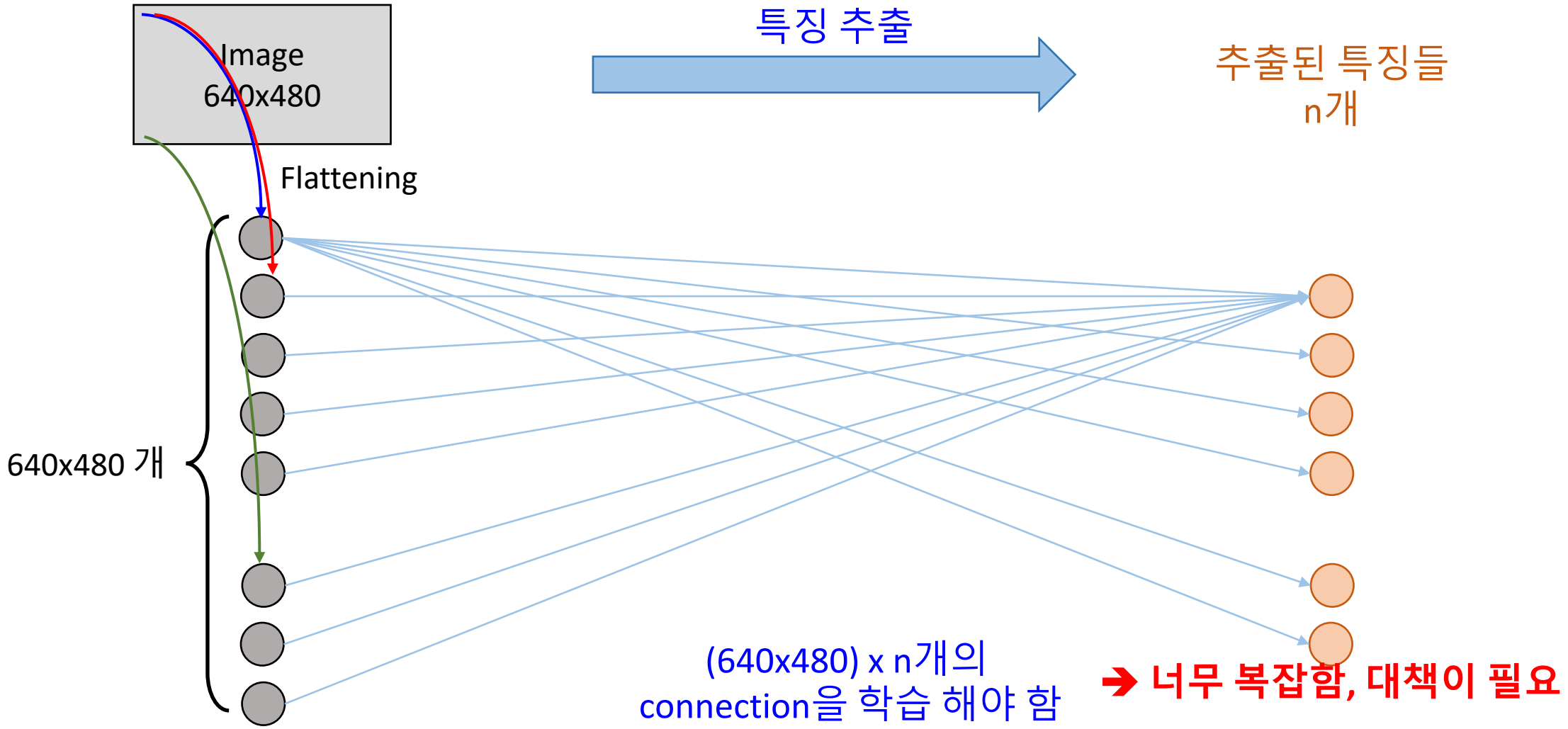


Image
640x480
Flattening
특징 추출
추출된 특징들
n개
640x480 개
(640x480) x n개의
connection을 학습 해야 함
➔ 너무 복잡함, 대책이 필요

# Image Filtering(Convolution)

원본 이미지

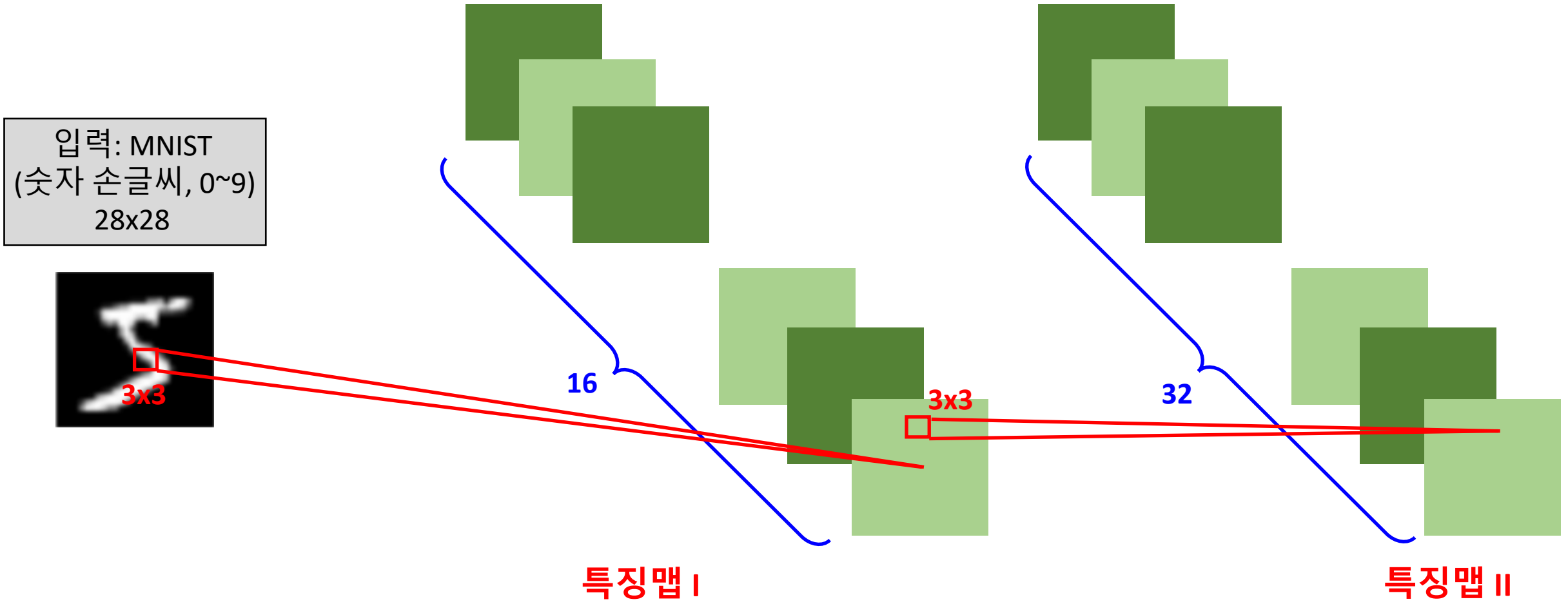
| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | −4 | 1 |
| 0 | 1 | 0 |

라플라시안 필터

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 1 | -1 | 0 | 1 |
| 0 | 0 | 0 | -2 | 0 | 2 |
| -1 | -2 | -1 | -1 | 0 | 1 |

소벨 필터

# 실습 – Convolution layer



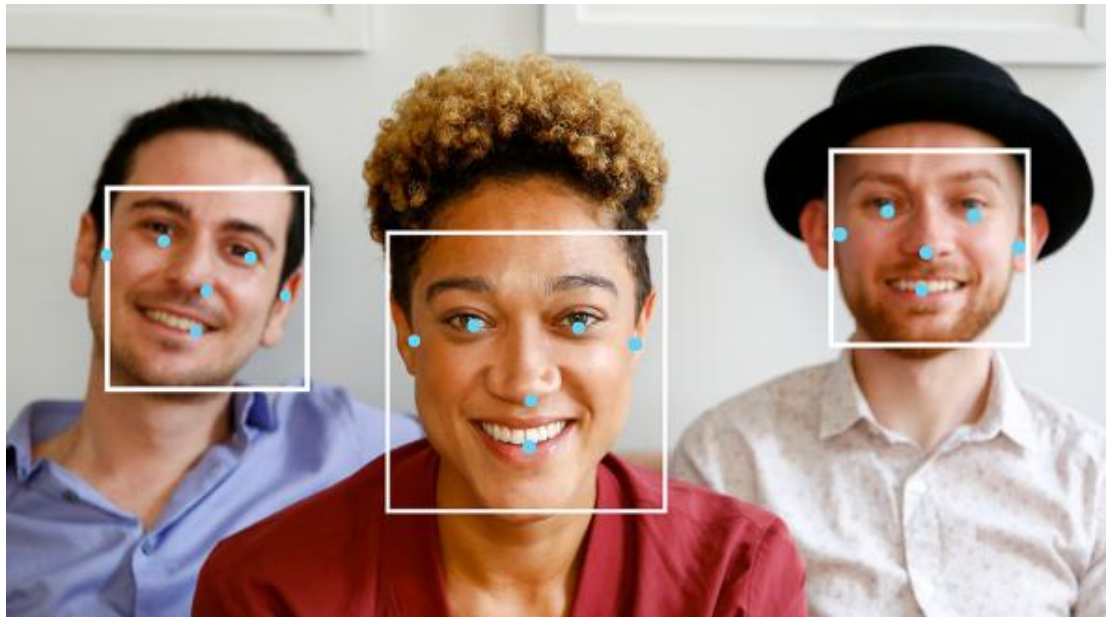
# 실습 – Convolution layer

```
#0: 필요한 라이브러리 가져오기
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms


#1: 데이터셋 로드 및 전처
transform = transforms.Compose([
    transforms.ToTensor(), # 이미지를 텐서로 변환
    transforms.Normalize((0.5,), (0.5,)) # 평균 0.5, 표준편차 0.5로 정규화
])

# 학습 및 테스트 데이터셋 다운로드 및 로드
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, transform=transform, download=True)

# 데이터로더 생성 (데이터를 배치 단위로 묶어 제공)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=1000, shuffle=False)
```

# 실습 – Convolution layer

```
#2: 모델정의
class TestCNN(nn.Module):
    def __init__(self):
        super(TestCNN, self).__init__()
        self.relu = nn.ReLU()
        # 첫 번째 합성곱 층: 입력 채널 1(흑백), 출력 채널 16
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        # 두 번째 합성곱 층: 입력 채널 16, 출력 채널 32
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)



    def forward(self, x):
        # 합성곱 -> ReLU -> 풀링
        x = self.relu(self.conv1(x)) # 입력: [B, 1, 28, 28] -> [B, 16, 28, 28]
        x = self.relu(self.conv2(x)) # 입력: [B, 16, 28, 28] -> [B, 32, 28, 28]

        return x

model = TestCNN()
```

# 얼굴인식

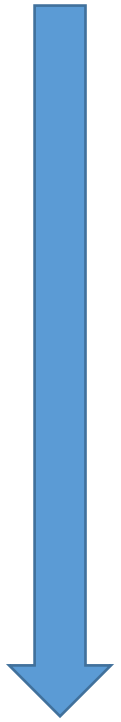


픽셀 값, 에지 등 등...

눈, 코, 입 ...

얼굴

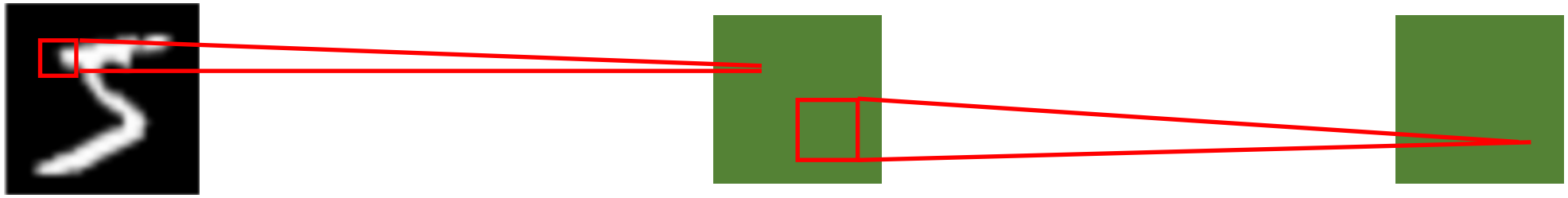
좁은 영역에서 얻을 수 있는 정보



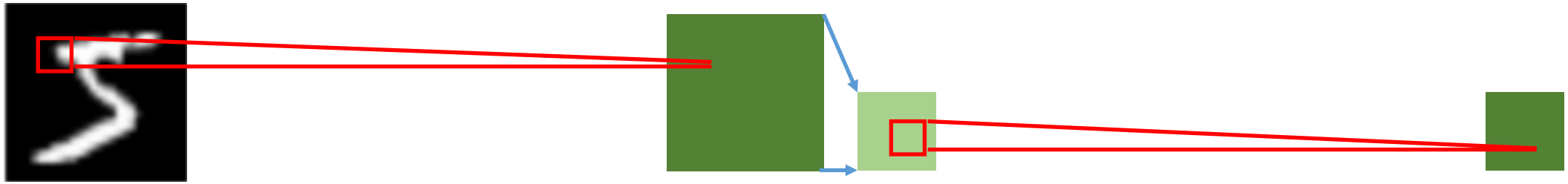
좁은 영역에서 얻은 정보를 바탕으로,
넓은 영역을 다 보아야 얻을 수 있는 정보

# 어떻게 할까?

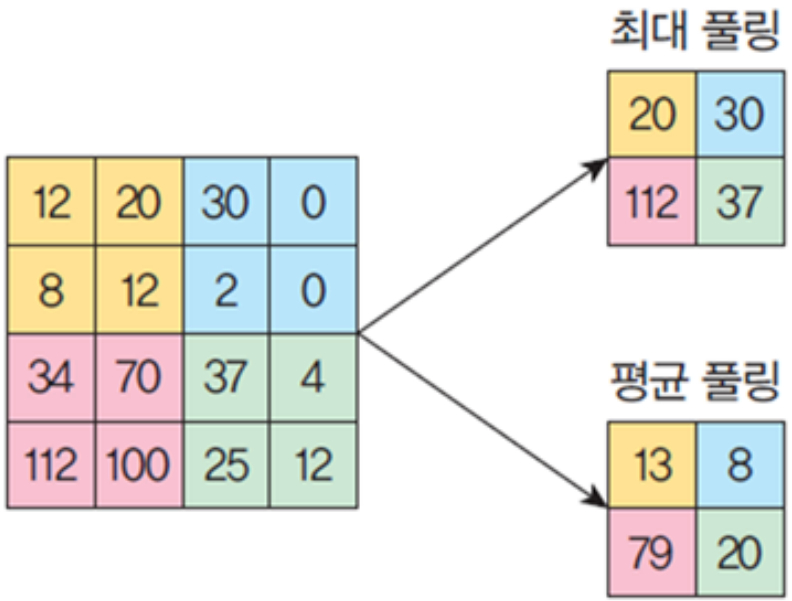
방법 1. 더 큰 영역에서 뽑아온다.



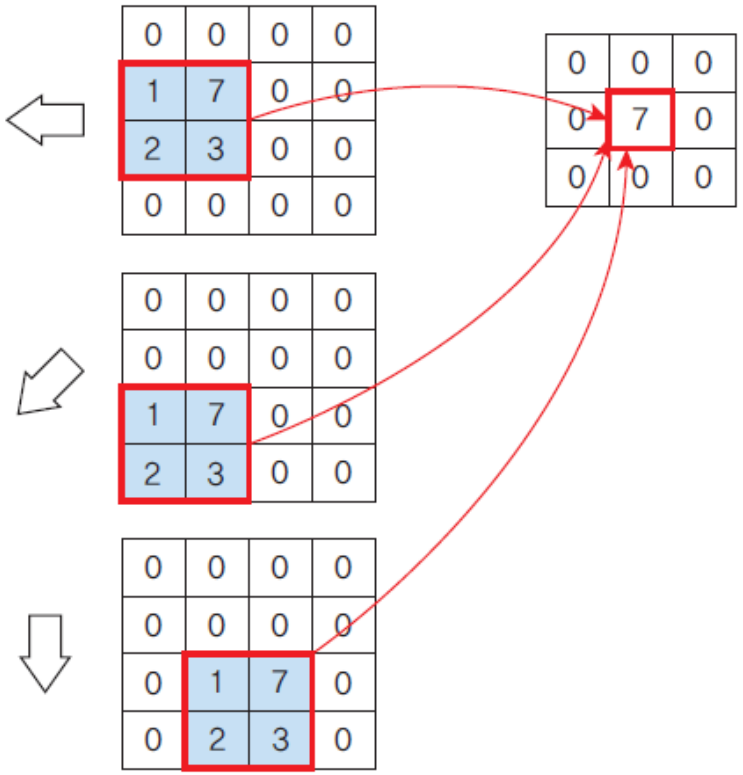
방법 2. 필요한 정보만 뽑아서 줄여서 진행한다.



# Pooling (Sub-sampling)



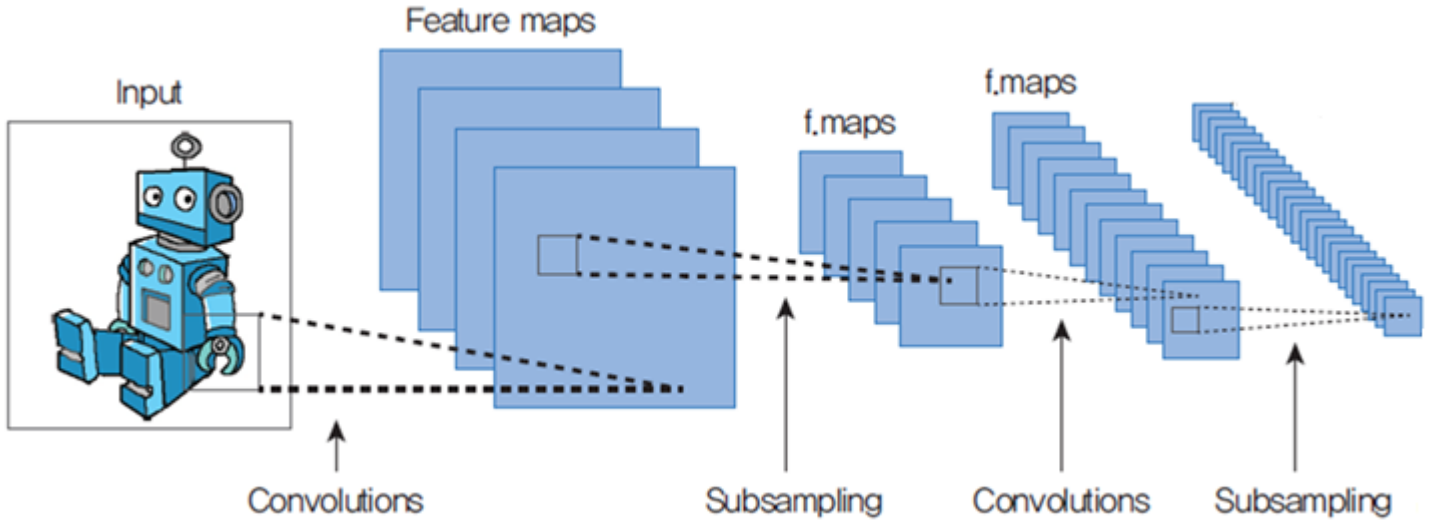
# Pooling (Sub-sampling)

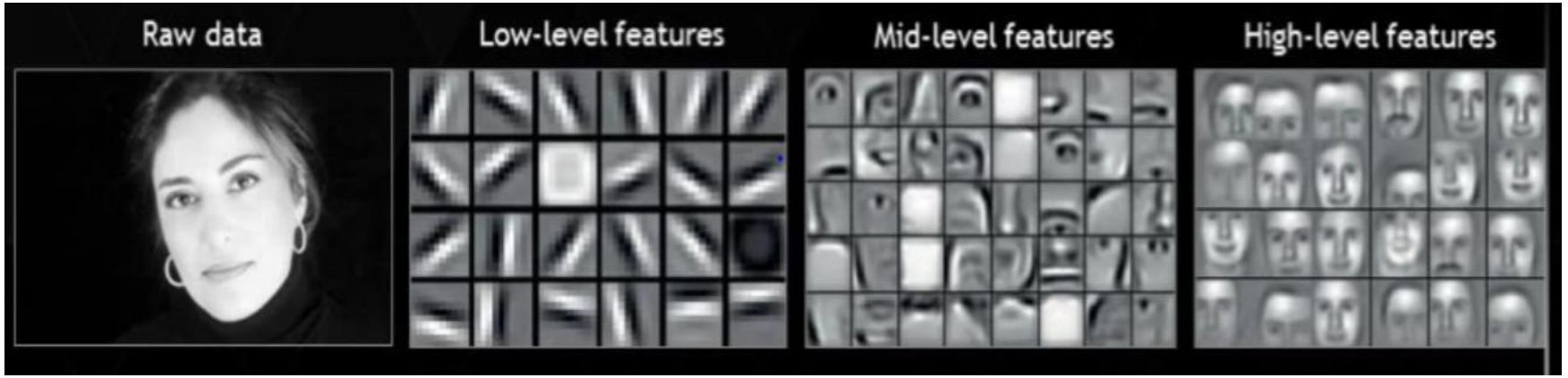



박스 범위안에서 최대 값 추출

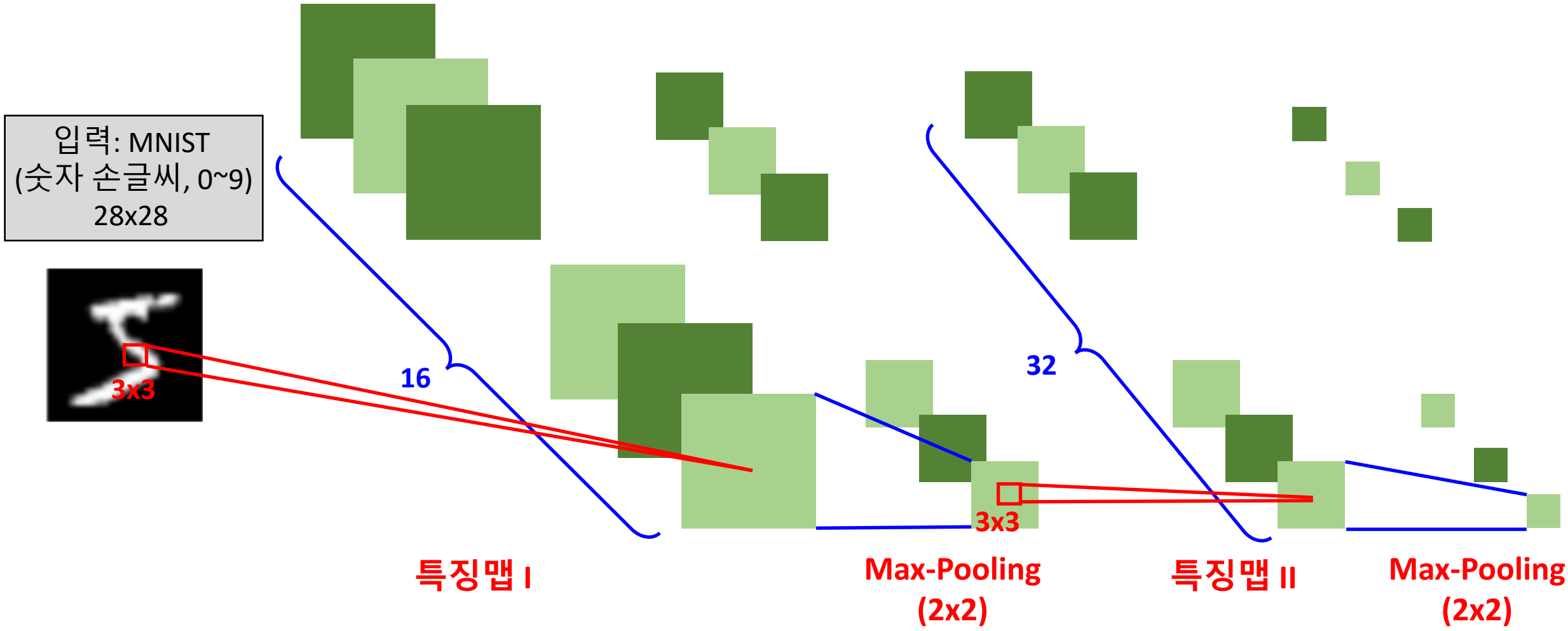
물체의 이동에 강인(둔감)해 짐

# 실습 – Pooling Layer



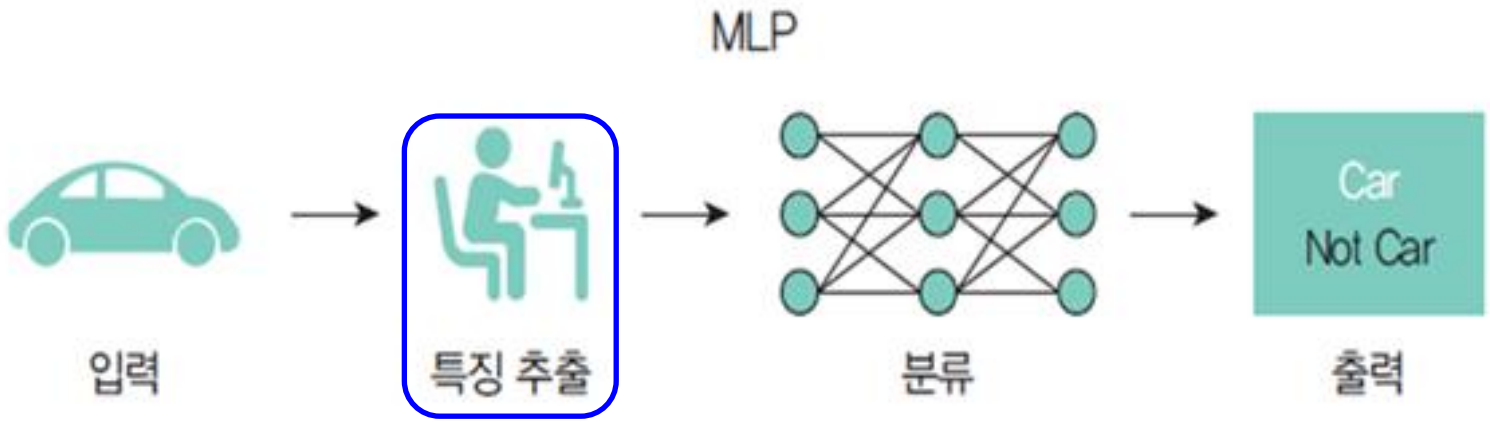
# 실습 – Convolution layer

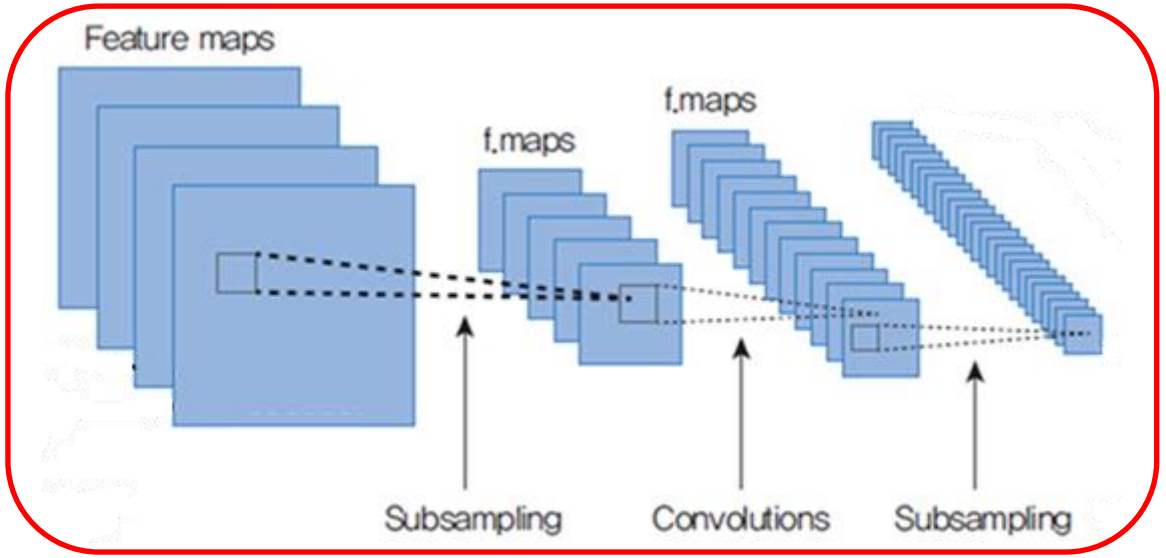


# 실습 – Convolution layer

```
#2: 모델정의
class TestCNN(nn.Module):
    def __init__(self):
        super(TestCNN, self).__init__()
        self.relu = nn.ReLU()
        # 첫 번째 합성곱 층: 입력 채널 1(흑백), 출력 채널 16
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        # 두 번째 합성곱 층: 입력 채널 16, 출력 채널 32
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)

       # 풀링 층: 2x2 윈도우로 크기 절반으로 축소
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)



    def forward(self, x):
        # 합성곱 -> ReLU -> 풀링
        x = self.pool(self.relu(self.conv1(x))) # 입력: [B, 1, 28, 28] -> [B, 16, 14, 14]
        x = self.pool(self.relu(self.conv2(x))) # 입력: [B, 16, 14, 14] -> [B, 32, 7, 7]

        return x

model = TestCNN()
```

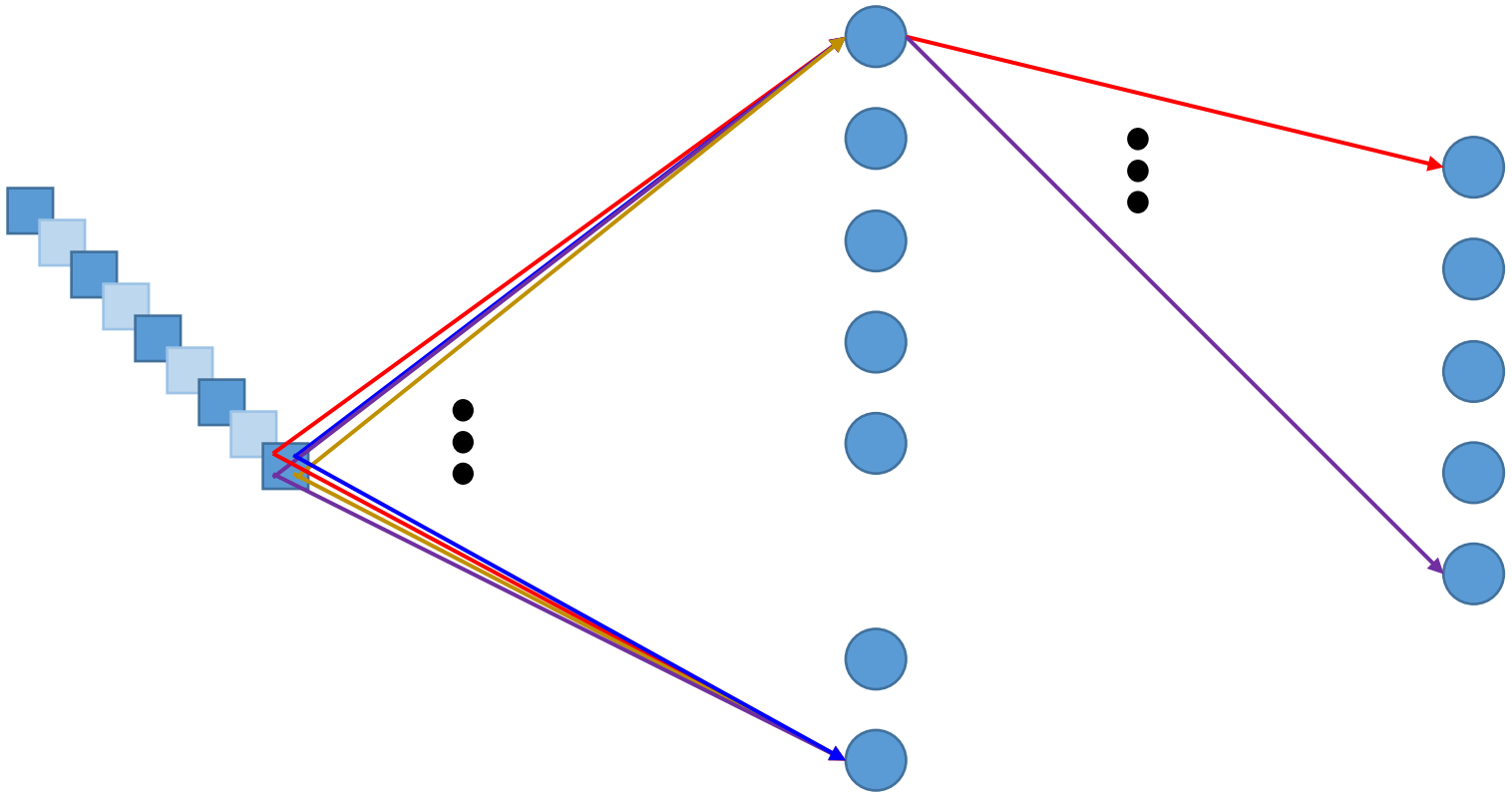
# 충분히 feature를 뽑고 나면?



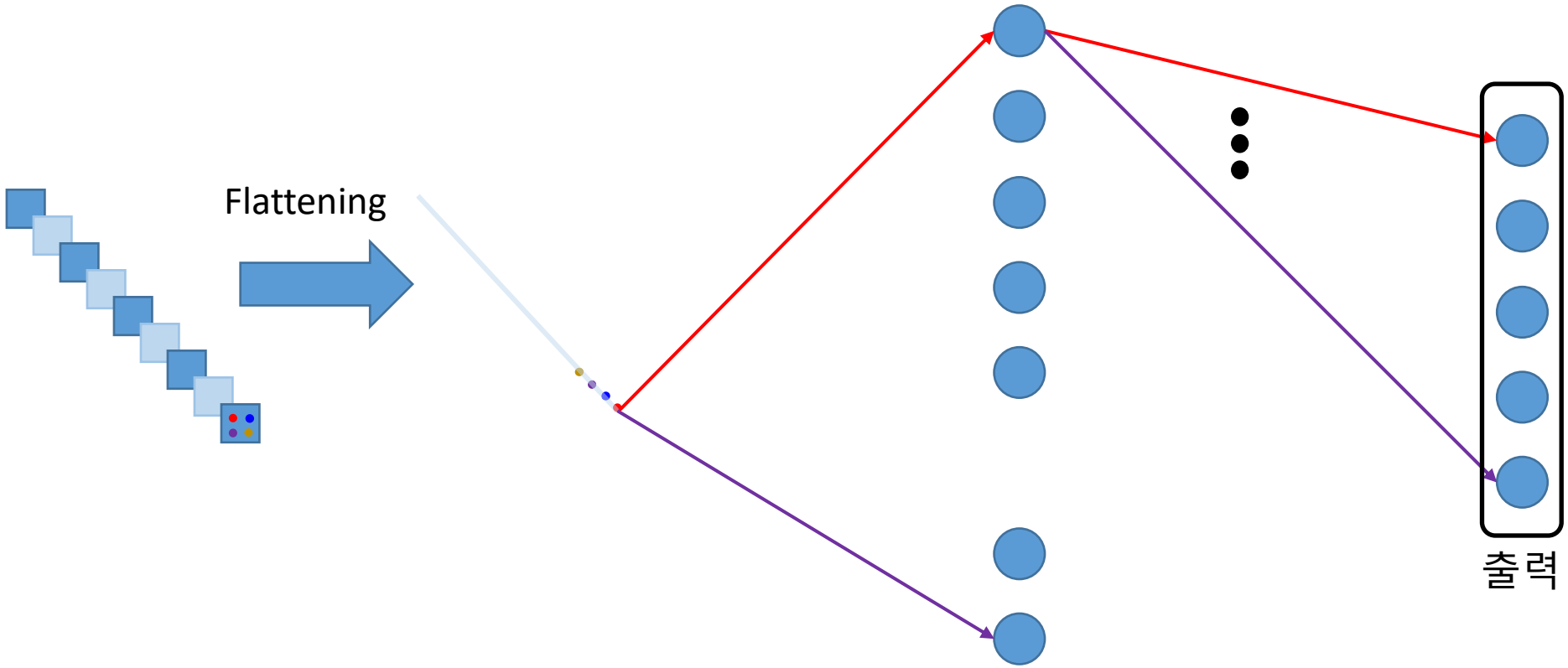

충분히 정제된 feature들
➔ MLP(Fully-connected Layer)

# 충분히 feature를 뽑고 나면?



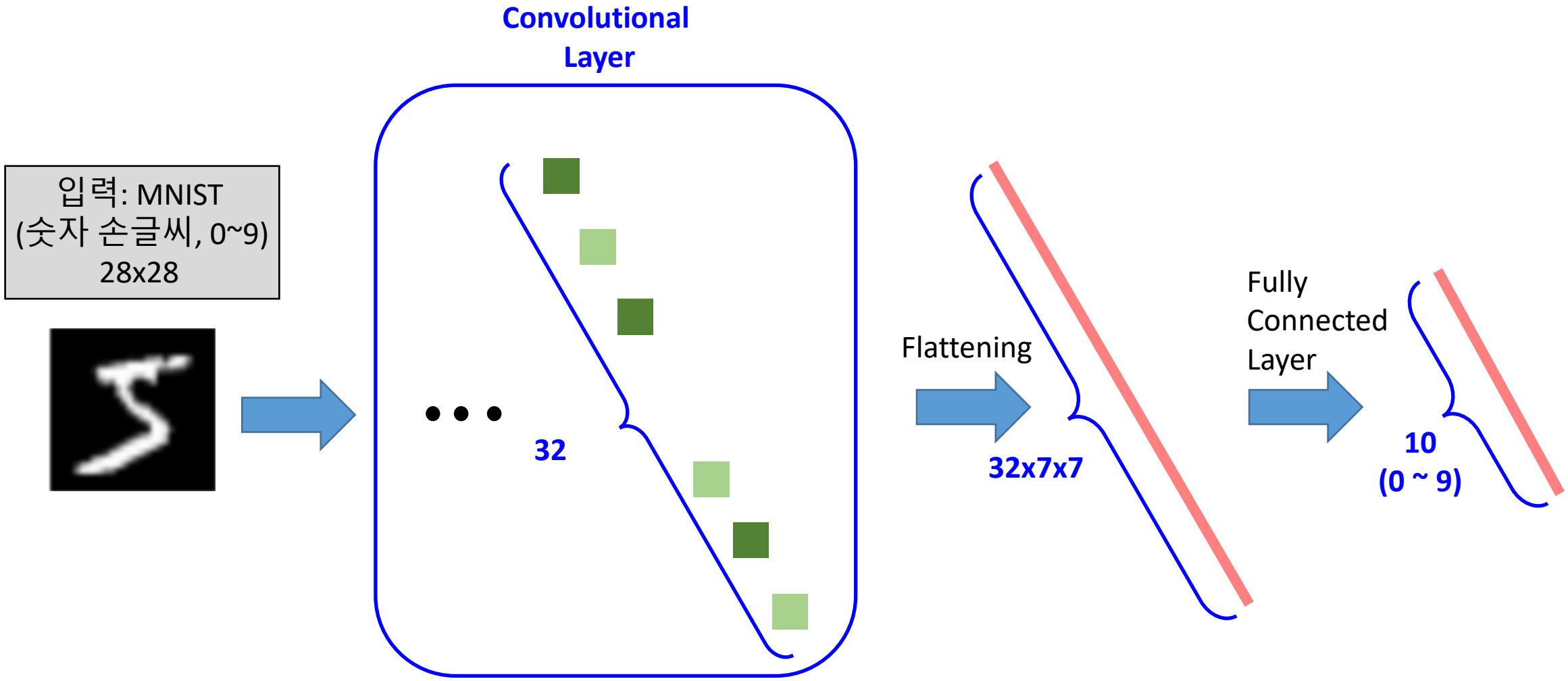
Fully-connected Layer: 모든 feature 값들을 다음 layer 뉴런과 연결

# 충분히 feature를 뽑고 나면?




Flattening: 모든 feature 값들을 연결하여 1차원 배열로 변경

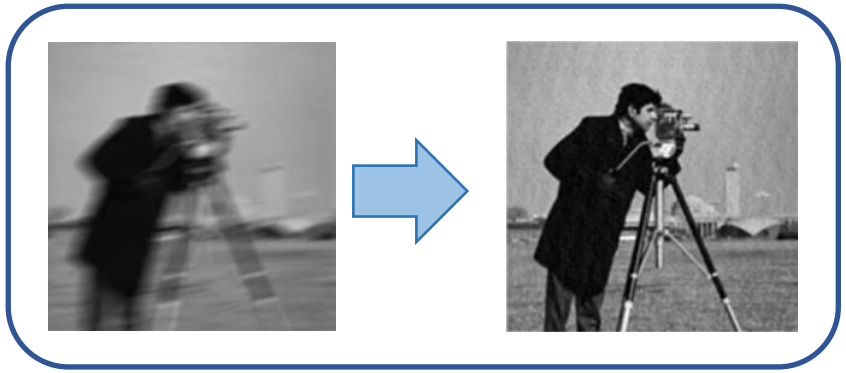
# 실습 – Convolution layer



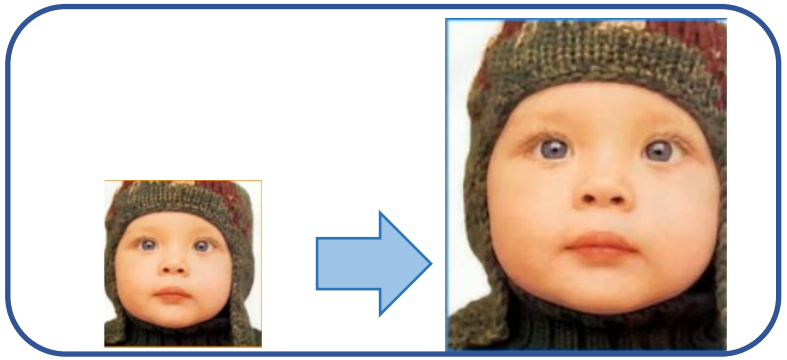
# 실습 – Convolution layer

```
#2: 모델정의
class TestCNN(nn.Module):
    def __init__(self):
        super(TestCNN, self).__init__()
        self.relu = nn.ReLU()
        # 첫 번째 합성곱 층: 입력 채널 1(흑백), 출력 채널 16
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3, stride=1, padding=1)
        # 두 번째 합성곱 층: 입력 채널 16, 출력 채널 32
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1, padding=1)

        # 풀링 층: 2x2 윈도우로 크기 절반으로 축소
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # 완전 연결 층: 7x7 크기의 32개 채널을 10개 클래스로
        self.fc = nn.Linear(32 * 7 * 7, 10)

    def forward(self, x):
        # 합성곱 -> ReLU -> 풀링
        x = self.pool(self.relu(self.conv1(x))) # 입력: [B, 1, 28, 28] -> [B, 16, 14, 14]
        x = self.pool(self.relu(self.conv2(x))) # 입력: [B, 16, 14, 14] -> [B, 32, 7, 7]

        # 텐서를 1차원으로 평탄화
        x = x.view(-1, 32 * 7 * 7)

        # 완전 연결 층
        x = self.fc(x)
        return x

model = TestCNN()
```

# Low-Level Vision Task



Deblurring

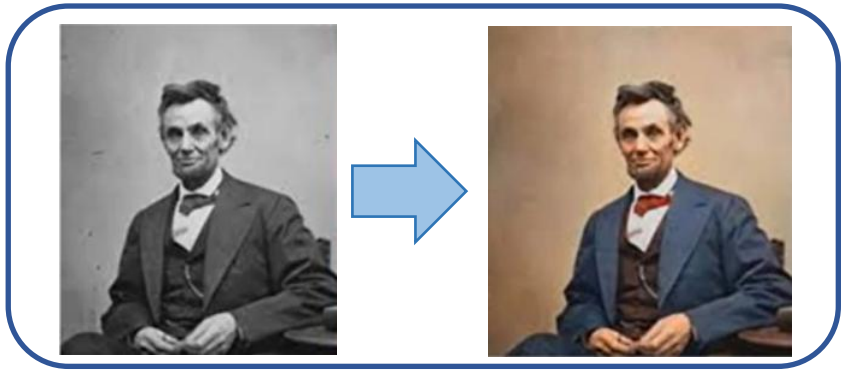


Super Resolution

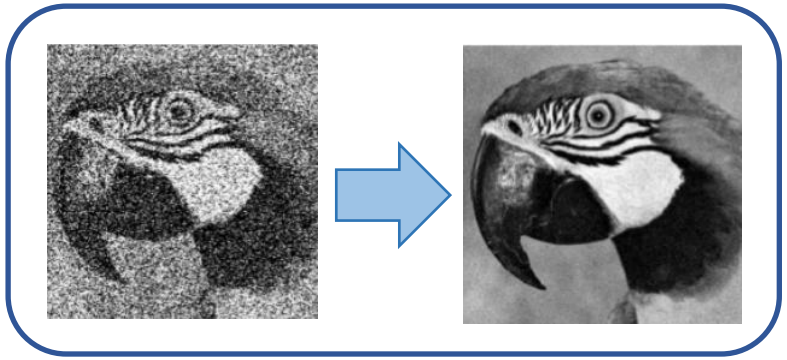
조작, 변경, 개선 …

입력: 이미지 ⟶ 출력: 이미지

Colorization

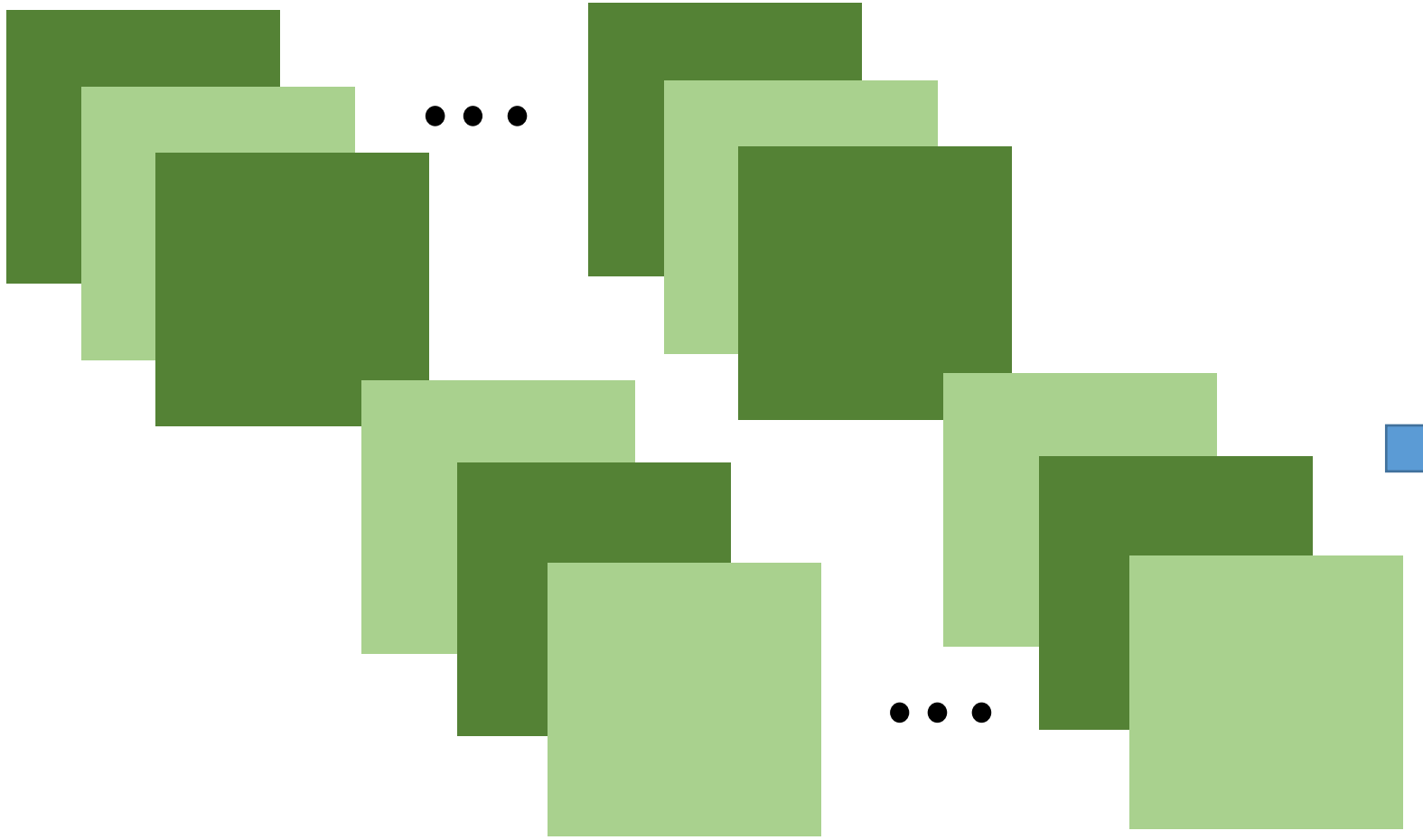


Denoising



# Super-Resolution

낮은 품질의
image

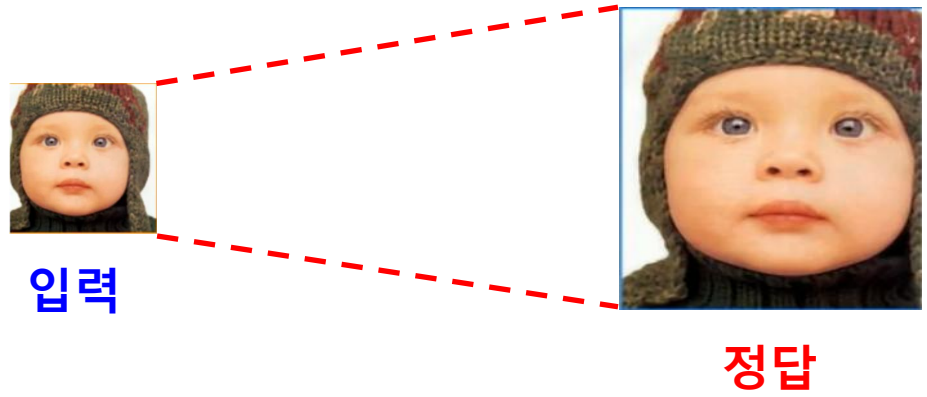


N번이 convolutional layer

높은 품질의
image

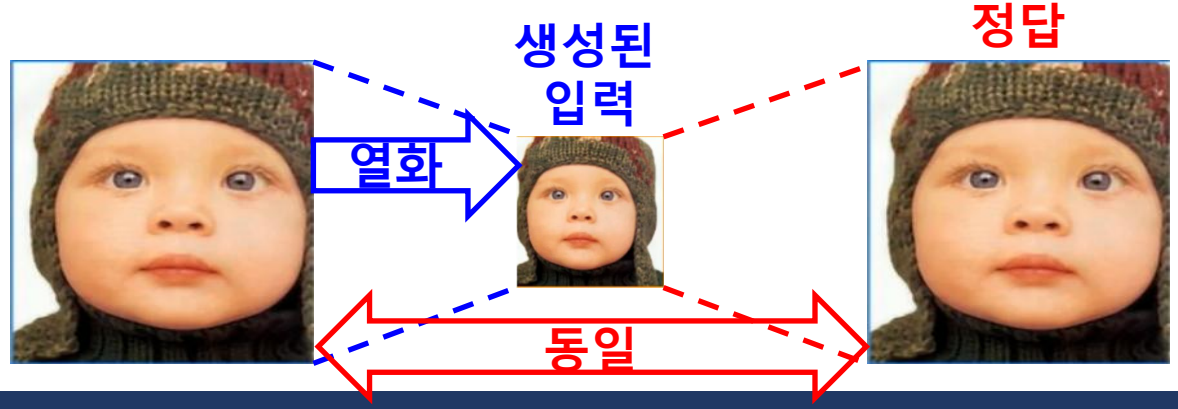
# Dataset

학습할 만 한 충분한 dataset 이 주어진 경우
➔ 다운 받아 사용하기만 하면 됨

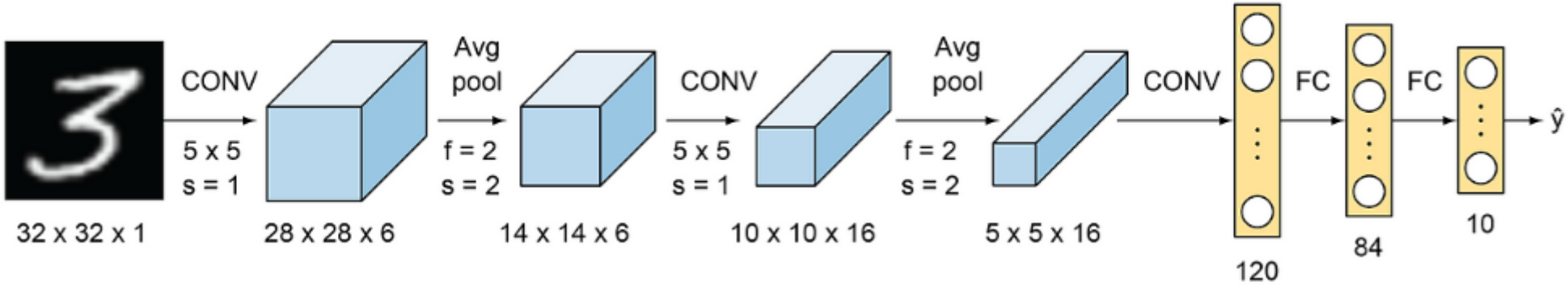



부족하거나 없다면?

가상의 데이터를 만든다.



# LeNet




최초의 CNN 모델 제안

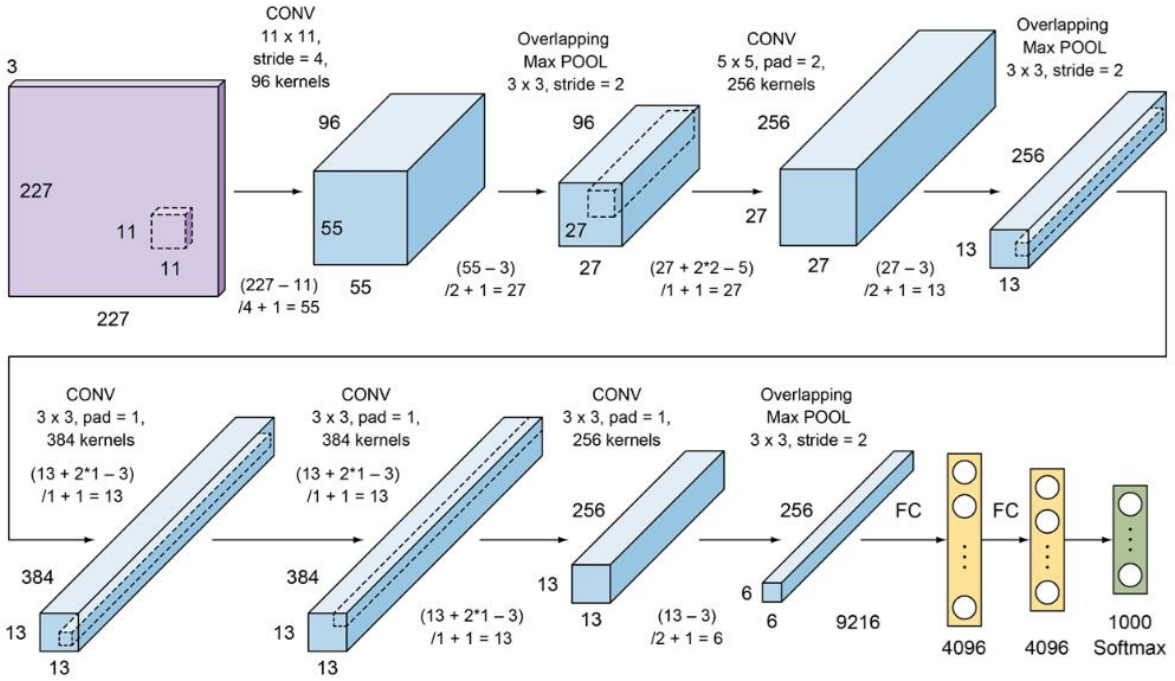
**"Gradient-based learning applied to document recognition,"**

vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

# 실습 – LeNet

```
import torch.nn.functional as F

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)   # (input=1, output=6, kernel=5x5)
        self.pool = nn.AvgPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*4*4, 120) # MNIST 기준
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

# AlexNet




**ImageNet 경진대회 우승**
**제안자의 이름을 땀**
**Max pooling**
**ReLU 사용**

**"Imagenet classification with deep convolutional neural networks,"**

proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

# 실습 – AlexNet

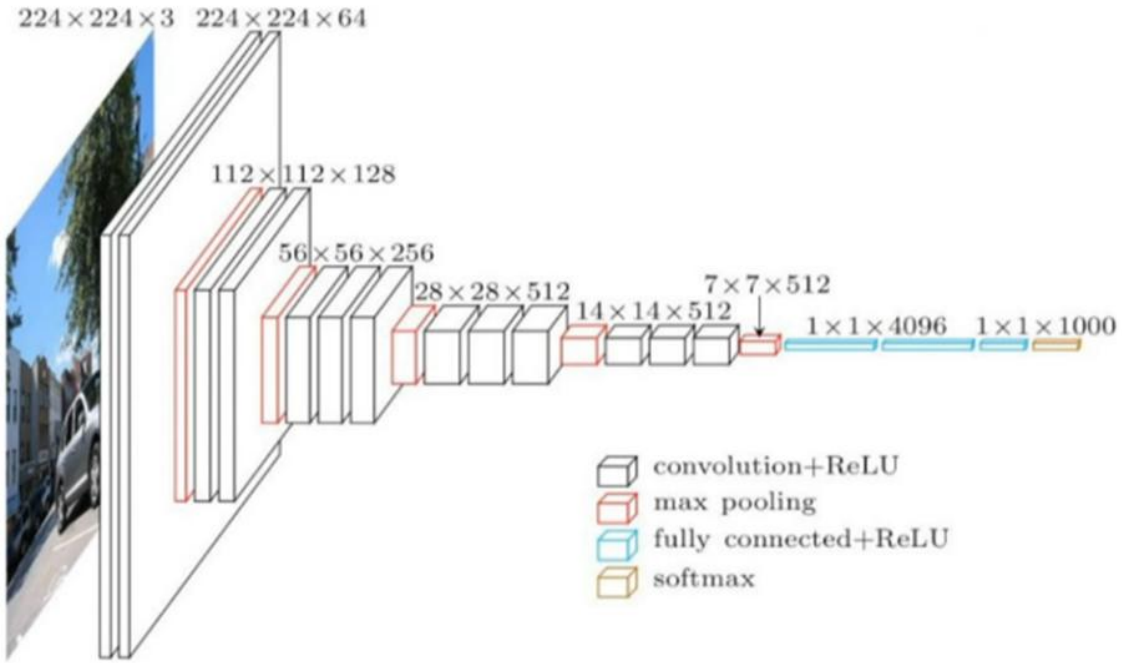
```
import torchvision
torchvision.models.list_models()
```

```
'alexnet',
'convnext_base',
'convnext_large',
'convnext_small',
'convnext_tiny',
'deeplabv3_mobilenet_v3_large',
'deeplabv3_resnet101',
'deeplabv3_resnet50',
'densenet121',
'densenet161',
'densenet169',
'densenet201',
'efficientnet_b0',
```

# 실습 – AlexNet

```
import torchvision.models as models

# 사전 학습된(pretrained) AlexNet 불러오기
alexnet = models.alexnet(pretrained=True)

# 또는 랜덤 초기화된 AlexNet 불러오기
alexnet = models.alexnet(pretrained=False)
```

# VGGNet




**큰 filter를 사용할 경우 과적합(overfitting)이 발생할 수 있음**
**➔ 작은 필터를(3x3) 여러 번 사용 (Receptive fieid)**

**"Very deep convolutional networks for large-scale image recognition,"**

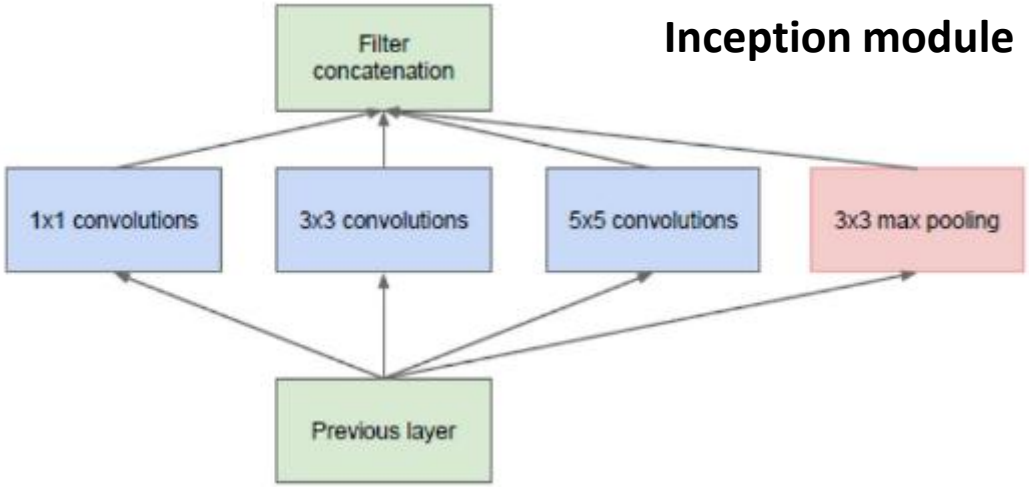
arxiv.org/pdf/1409.1556

# 실습 – VGGNet

```
import torchvision
torchvision.models.list_models()
```

```
...
'swin_b',
'swin_s',
'swin_t',
'swin_v2_b',
'swin_v2_s',
'swin_v2_t',
'vgg11',
'vgg11_bn',
'vgg13',
'vgg13_bn',
'vgg16',
'vgg16_bn',
'vgg19',
'vgg19_bn',
'vit_b_16',
'vit_b_32',
'vit_h_14',
'vit_l_16',
'vit_l_32',
...
```

# GoogLeNet




**Inception module**

**다양한 크기의 filter 사용 ➔ 다양한 receptive field**
**➔다양한 관점에서 정보 처리**

**“Going deeper with convolutions,”**

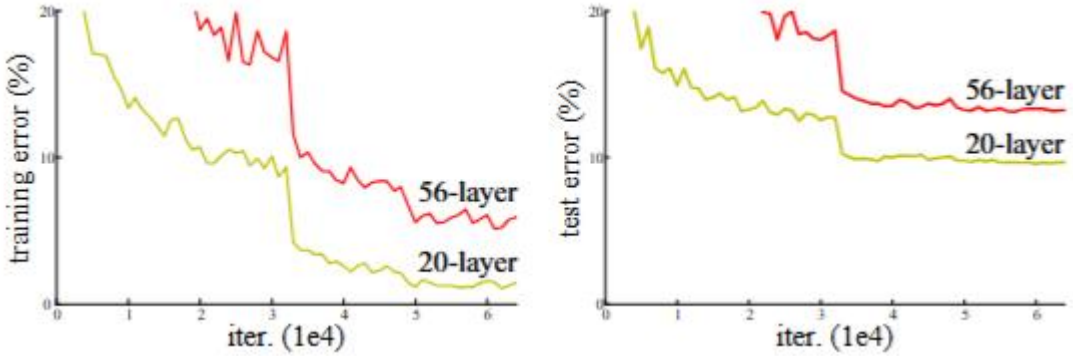
arxiv.org/pdf/1409.4842

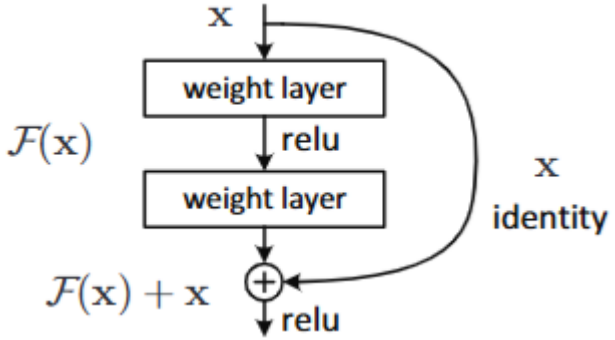
# 실습 – GoogLeNet

```
import torchvision
torchvision.models.list_models()
```

```
fasterrcnn_resnet50_fpn_v2',
fcn_resnet101',
fcn_resnet50',
fcos_resnet50_fpn',
googlenet',
inception_v3',
keypointrcnn_resnet50_fpn',
lraspp_mobilenet_v3_large',
maskrcnn_resnet50_fpn',
maskrcnn_resnet50_fpn_v2',
```

# ResNet




Layer가 깊어질 때, 정확도가 오히려 하락

➔ 향상시키지 못할 거면, 유지라도 하자!

➔ Gradient vanishing 문제가 해결되면서, 성능도 향상

"Deep residual learning for image recognition,"

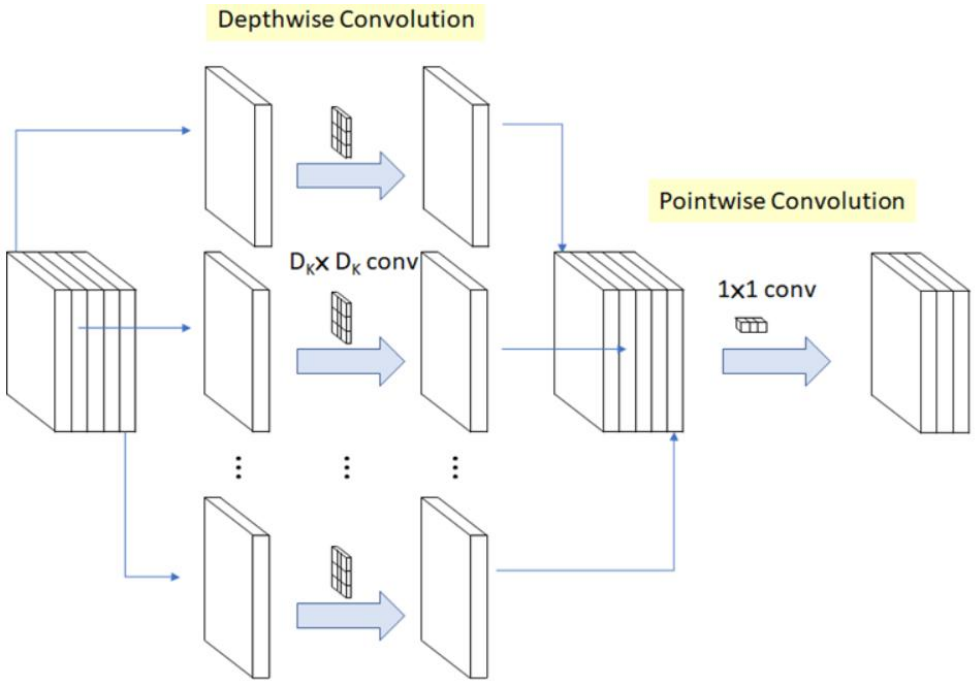
arxiv.org/pdf/1512.03385

# 실습 – ResNet

```
import torchvision
torchvision.models.list_models()
```

```
'regnet_y_800mf',
'regnet_y_8gf',
'resnet101',
'resnet152',
'resnet18',
'resnet34',
'resnet50',
'resnext101_32x8d',
'resnext101_64x4d',
```

# MobileNet




깊어지면 성능이 좋아짐 but, 복잡도가 증가함
Edge(Mobile) 장비에서 효율적으로 convolution 할 방법이 없을까?
Conventional Convolution ➔ Depth-wise + Point-wise

"MobileNets: Efficient convolutional neural networks for mobile vision applications,"

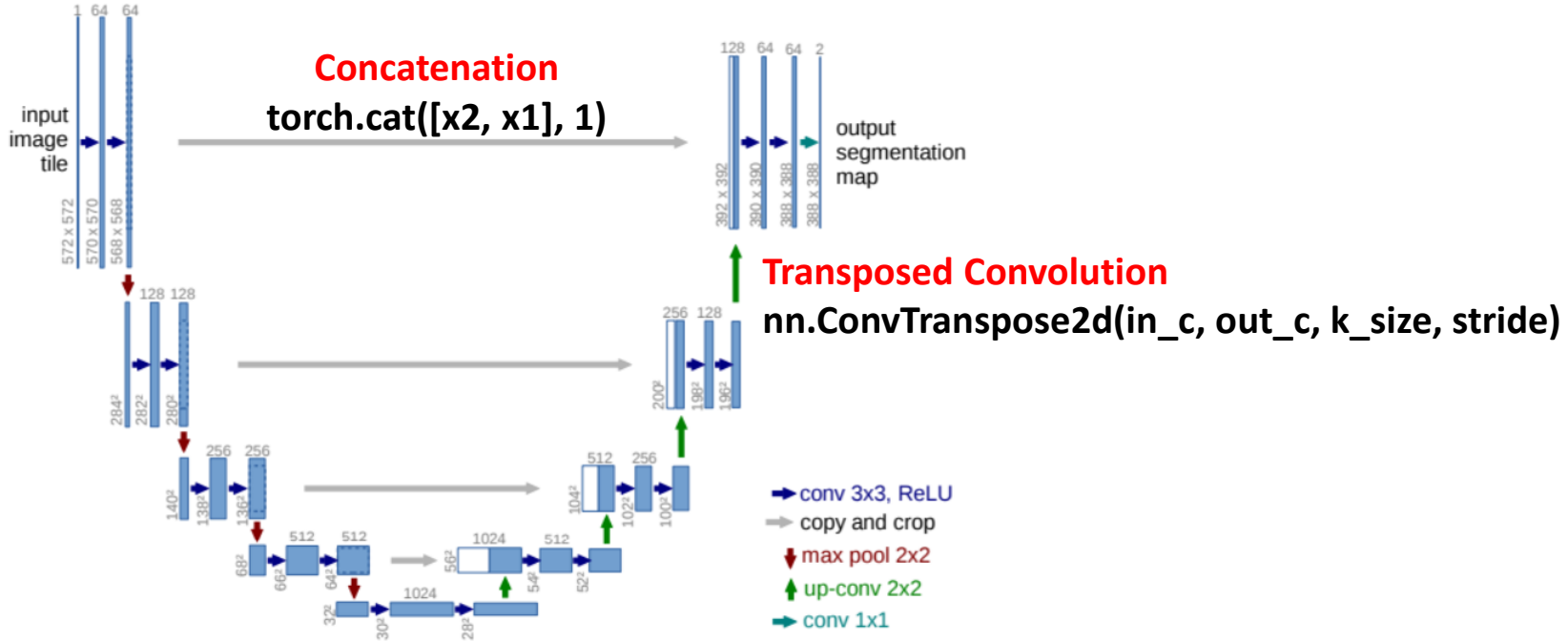
arxiv.org/pdf/1704.04861

# 실습 – MobileNet

```
import torchvision
torchvision.models.list_models()
```

```
'maxvit_t',
'mc3_18',
'mnasnet0_5',
'mnasnet0_75',
'mnasnet1_0',
'mnasnet1_3',
'mobilenet_v2',
'mobilenet_v3_large',
'mobilenet_v3_small',
'mvit_v1_b',
'mvit_v2_s',
'quantized_googlenet',
```

# U-Net




초기 영상 분할을 위해 제안

➔ 이 후, 높은 효율성으로 픽셀 수준의 이미지 생성에 많이 사용

**"U-Net: Convolutional Networks for Biomedical Image Segmentation,"**

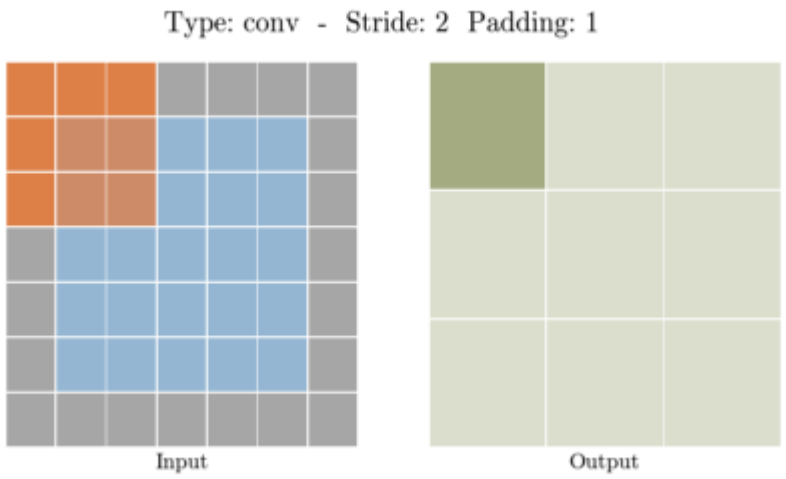
[1505.04597] U-Net: Convolutional Networks for Biomedical Image Segmentation

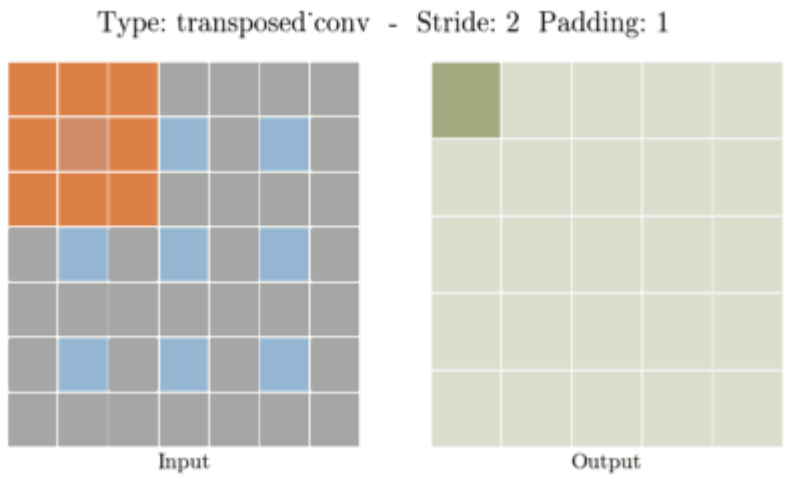
# U-Net: Transposed Convolution

**Feature map 크기 키우기 (Up-sampling)**

➔ **De-Convolution: 새로운 필터 학습 X**

➔ **Transposed Convolution: 새로운 필터 학습 O**



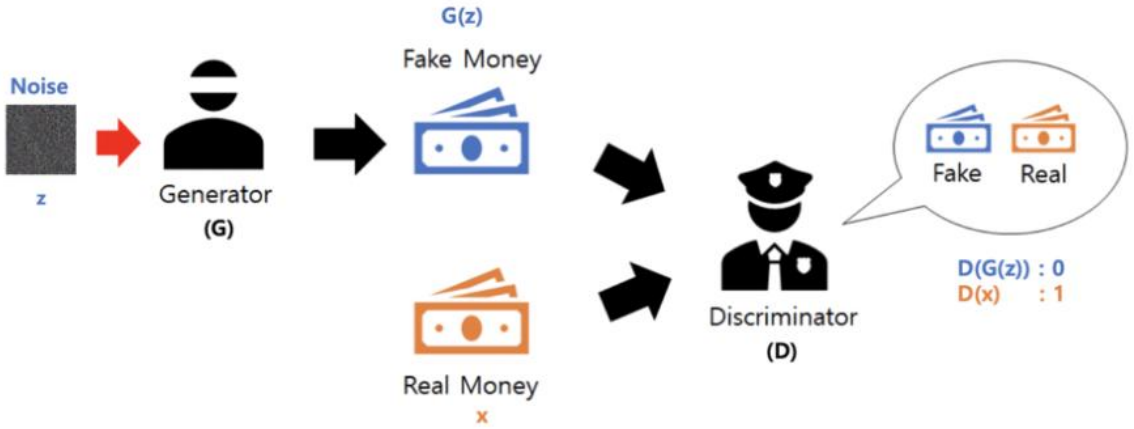



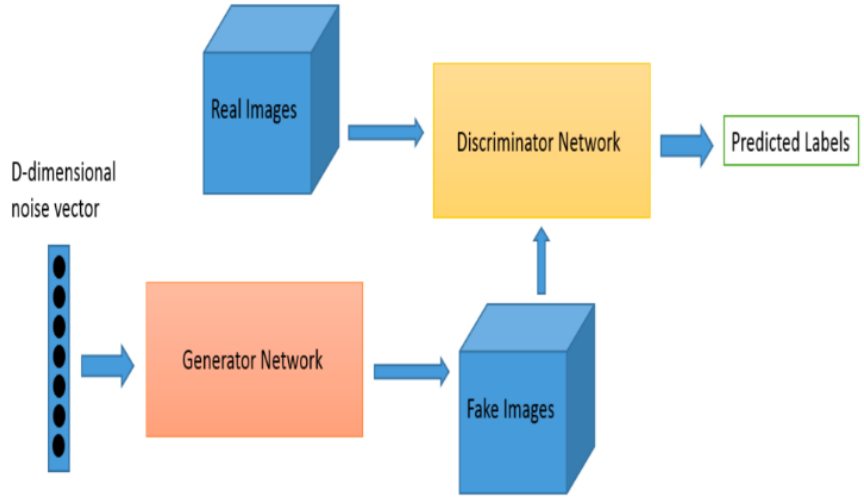
# 실습 – U-Net

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\DL_EX> pip install segmentation_models_pytorch
```

```
import segmentation_models_pytorch as smp

model = smp.Unet(
    encoder_name="resnet34",      # 백본 network
    encoder_weights="imagenet",  # 사전 학습된 가중치 사용
    in_channels=3,                # 입력 이미지 채널 (RGB)
    classes=1                     # 최종 출력 클래스 수
)
```

# GAN(Generative Adversarial Networks)




고품질 & 비지도 학습

But, 훈련이 불안정함

**"Generative Adversarial Networks,"**

[1406.2661] Generative Adversarial Networks

# 실습 – GAN

## 다양한 GAN 구현들

GitHub - eriklindernoren/PyTorch-GAN: PyTorch implementations of Generative Adversarial Networks. · GitHub