

Convolution

상태	CNN
담당자	

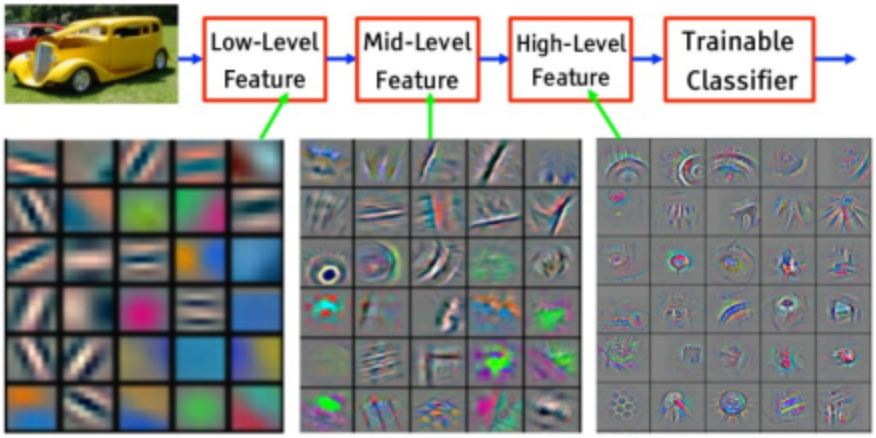
왜 CNN인가?

개와 고양이를 분류하는 태스크를 예로 들어 생각해 보자. 기존 DNN은 이미지의 픽셀값(0-255)을 input으로 받아 input 이미지가 개인지, 고양이인지 판단했다.

그런데 같은 개, 고양이더라도 이미지가 틀어져 있거나 중심이 맞지 않는 등 straightforward하지 않으면 학습에 어려움이 있다는 단점이 있었다.

마치 경주마가 주변 환경을 보지 않고 앞만 보고 달리는 것처럼, DNN은 이미지의 전반적인 특성을 보지 못한다.

이를 극복하기 위해 이미지를 대표하는 특징을 도출해서 학습할 수 있는 CNN이 등장하게 되었다.

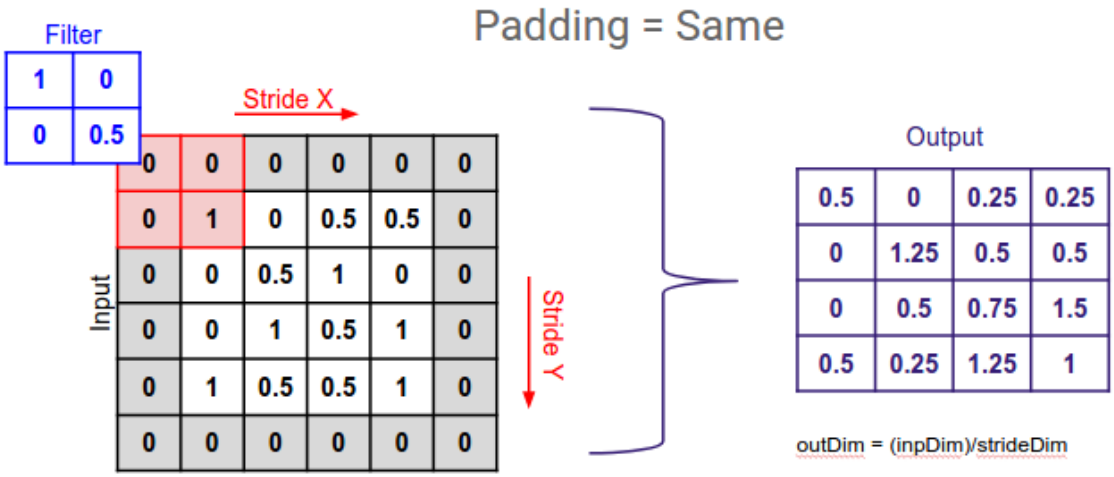


사실 내부적으로는 convolution을 통한 특성 추출 → activation function을 통한 출력 → pooling을 통한 이미지 크기 축소 → subsampling... 의 반복으로 local한 특징으로부터 global한 특징을 출력하게 된다.

Convolution

> 💡 이미지 위에서 stride만큼 filter(kernel)을 이동시키면서, 겹쳐지는 부분의 값 원소의 값을 곱해 모두 더한(element-wise) 값을 출력으로 하는 연산

- stride : filter를 한 번에 얼마나 이동할 것인가
- padding : 이미지 상하좌우에 둘러지는 띠



PyTorch

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, bias=True)
# dilation이나 groups 옵션은 잘 사용하지 않음
# kernel_size 자리에 (n, m)처럼 작성하면 nxm 크기의 커널을 만들 수 있음
```

예를 들어, 입력 채널 `1`, 출력 채널 `1`, 커널 크기 `3x3` 인 네트워크는 `nn.Conv2d(1, 1, 3)`으로 사용할 수 있다.

입력 형태

> `conv = nn.Conv(1, 1, 3)`을 어떤 input이 통과해 output 형태로 (`output = conv(input)`) 나오게 하고 싶다.

- input type : `torch.Tensor`
- input shape : (N x C x H x W) 즉 (batch_size, input channel, height, width)

예제

Convolution의 output 크기

$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

예제 1)
input image size : 227 x 227
filter size = 11x11
stride = 4
padding = 0
output image size = ?

예제 2)
input image size : 64 x 64
filter size = 7x7
stride = 2
padding = 0
output image size = ?

예제 3)
input image size : 32 x 32
filter size = 5x5
stride = 1
padding = 2
output image size = ?

예제 4)
input image size : 32 x 64
filter size = 5x5
stride = 1
padding = 0
output image size = ?

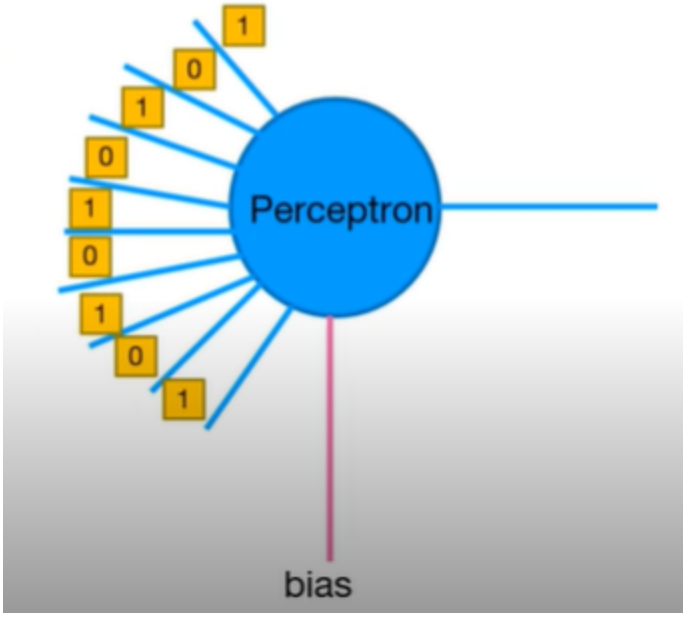
예제 5)
input image size : 64 x 32
filter size = 3x3
stride = 1
padding = 1
output image size = ?

- 예제 2처럼 output size 계산 결과가 소숫점이 나오면, 소수점은 버린다.
- 예제 4, 5처럼 input size가 정방형이 아닌 경우 (n, m)으로 생각해 계산한다.

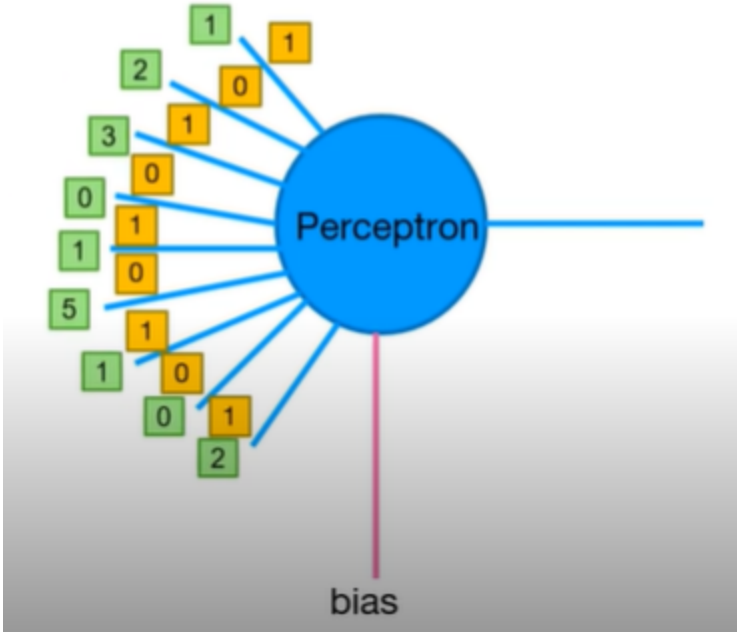
$\frac{(32,64)-5+2\times0}{1} + 1 = \frac{(27,59)}{1} + 1 = (28, 60)$

```
import torch
import torch.nn as nn

inputs = torch.Tensor(1, 1, 227, 227)
inputs.shape # torch.Size([1, 1, 227, 227])

conv = nn.Conv2d(1, 1, kernel_size=(11, 11), stride=(4, 4))
out = conv(inputs)
out.shape
# torch.Size([1, 1, 55, 55])
```

Neuron과 Convolution



💡 Perceptron의 weight 값으로 convolution filter의 값들이 들어간다.



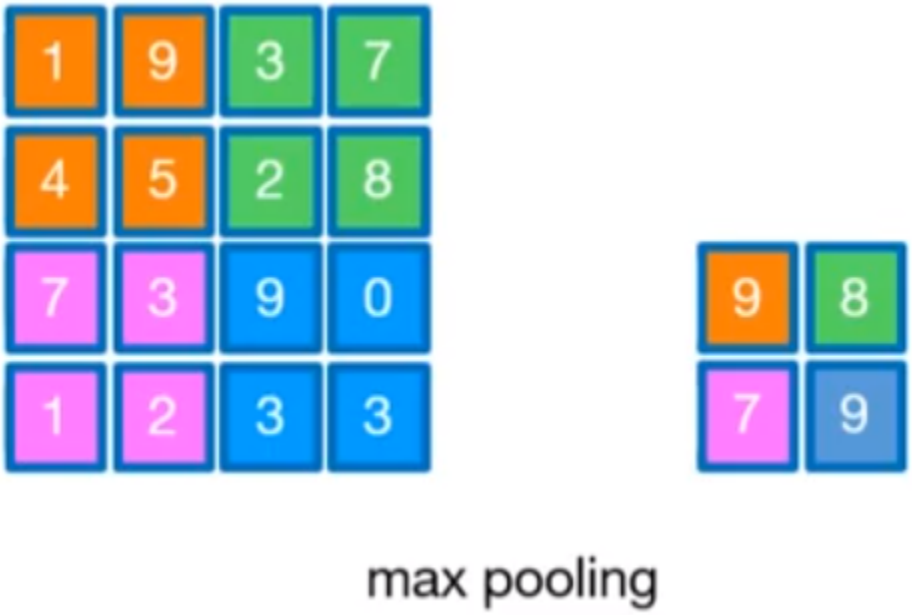
💡 input에 대해 매 filter마다 element-wise 연산이 들어가는 element들이 연결되어 연산이 진행된다.

연산을 수행하면 8이 되지만, bias의 존재로 실제 output의 값은 8+ bias가 될 것이다.

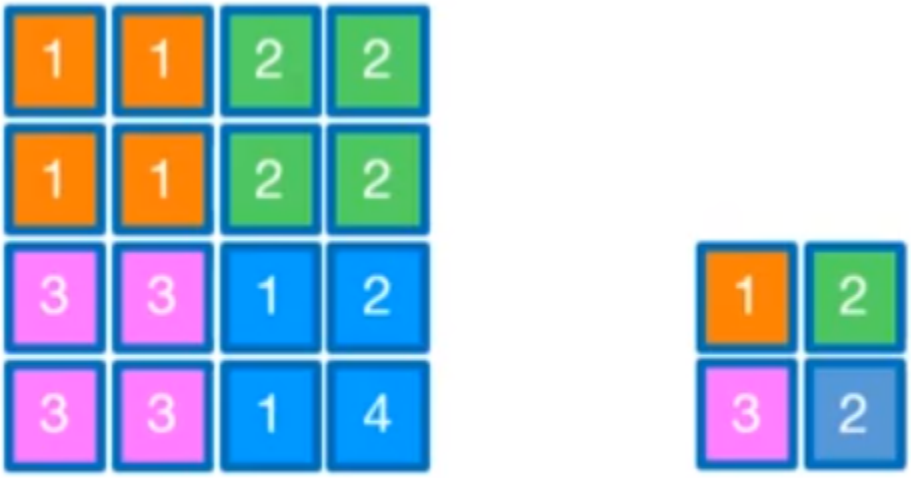
Pooling

> 💡 이미지 사이즈를 줄이거나 fully-connected 연산을 대체하기 위해 사용한다.
> Pooling을 사용하지 않으면 불필요한 연산이 많아져 overfitting의 위험이 커진다.

- Max Pooling : 각 영역의 최댓값을 가져온다.



- Average Pooling : 각 영역의 평균값을 가져온다.



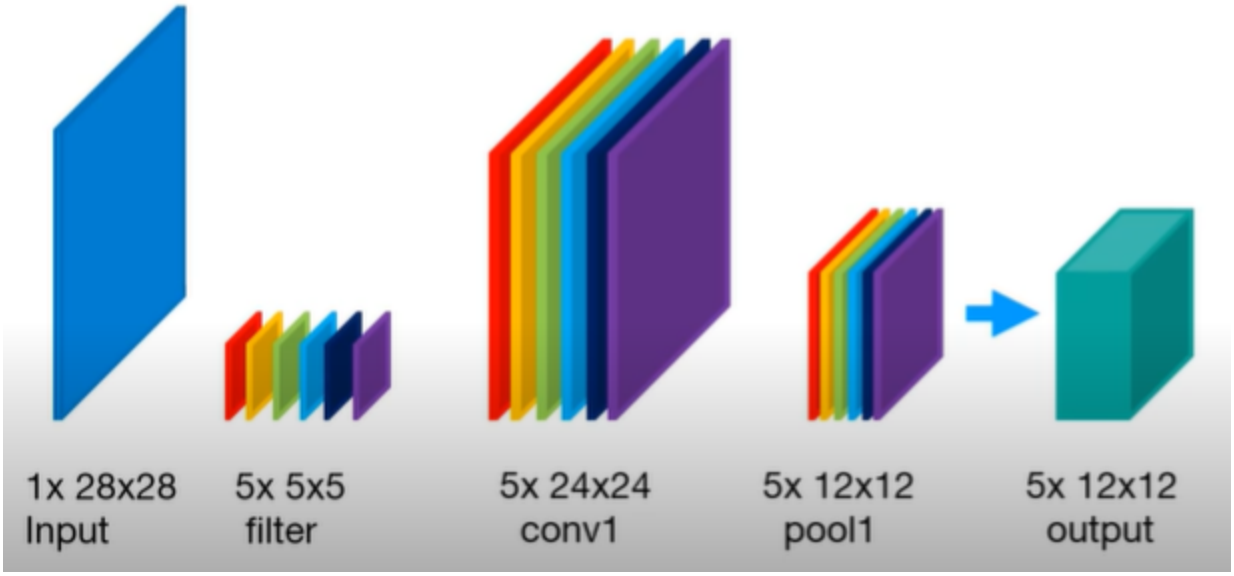
Average pooling

PyTorch

- MaxPool2d

```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0)
# 다른 옵션들은 일단 나중에 생각하자.
```

CNN Implementation



1채널의 28x28 input을 받아 최종적으로 5채널의 12x12 output을 출력하는 CNN 네트워크를 구성해 보자.

```
input = torch.Tensor(1, 1, 28, 28)
conv1 = nn.Conv2d(1, 5, 5)
pool = nn.MaxPool2d(2)
out = conv1(input)
out2 = pool(out)
print('out1 : ', out.size())
print('out2 : ', out2.size())

>>
out1 :  torch.Size([1, 5, 24, 24])
out2 :  torch.Size([1, 5, 12, 12])
```