CNN

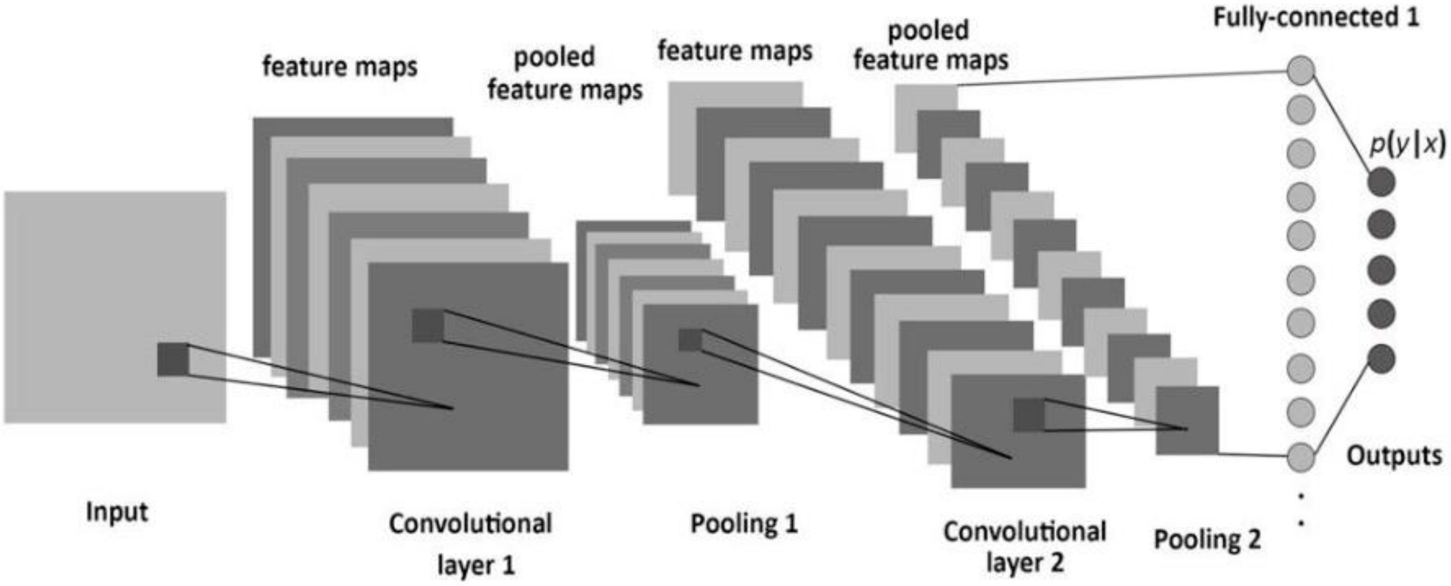
AILAB
Hanyang Univ.

오늘 실습 내용

1. CNN 구현 MNIST Dataset
2. CNN 구현 CIFAR-10

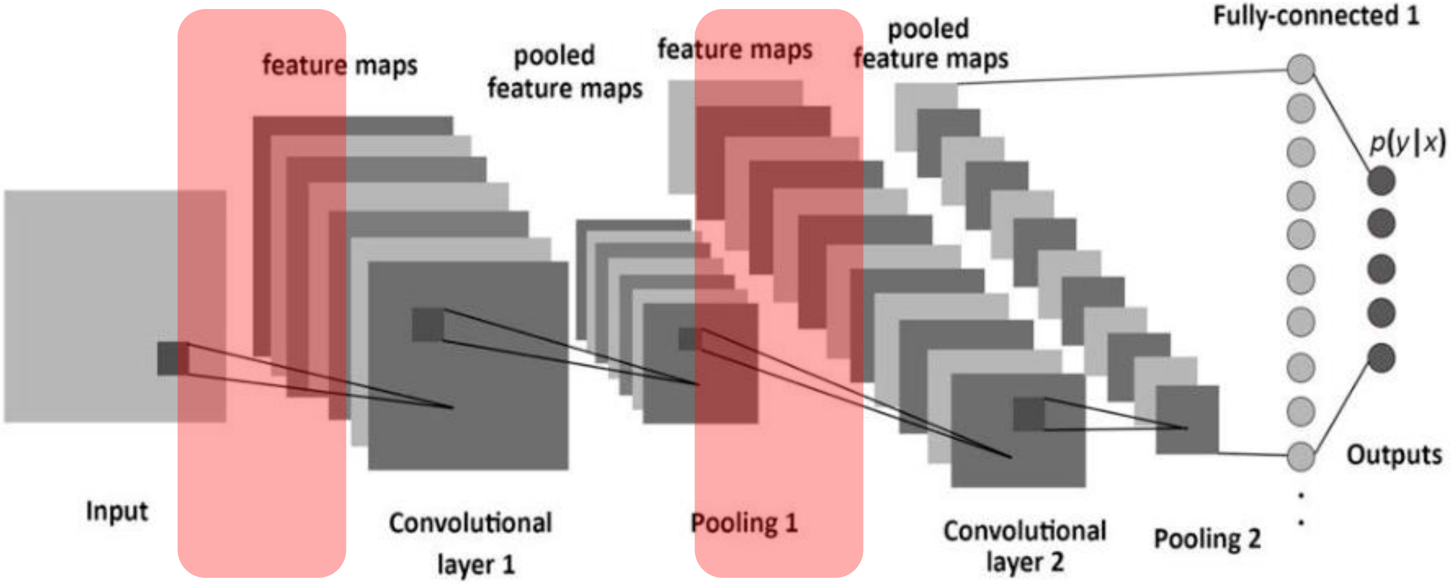
CNN

CNN 구성요소 = (1) Convolution layer (2) Sub-sampling(pooling) layer (3) FC layer(Affine)



CNN

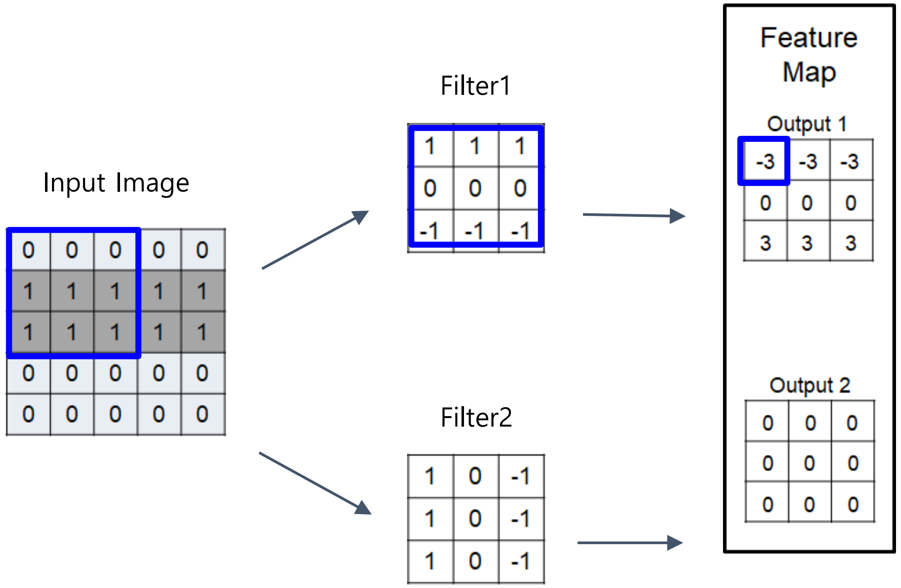
CNN 구성요소 = (1) Convolution layer (2) Sub-sampling(pooling) layer (3) FC layer(Affine)



Convolution

Convolution?

: Convolution(합성곱)은 하나의 함수와 또 다른 함수를 반전 이동한 값을 곱한 다음, 구간에 대해 적분하여 새로운 함수를 구하는 수학 연산자이다.



Convolution

torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

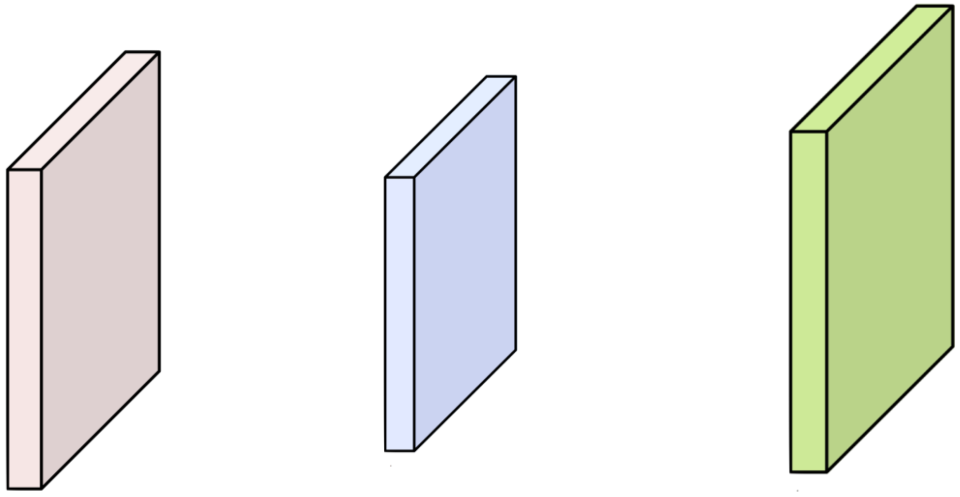
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolution

Channel

- 입력 데이터의 Channel 수와 필터의 Channel 수가 일치 해야 함
- 필터의 개수가 아웃풋의 Channel을 결정



Input image
(3, 3, 1)
(width, height, channel)

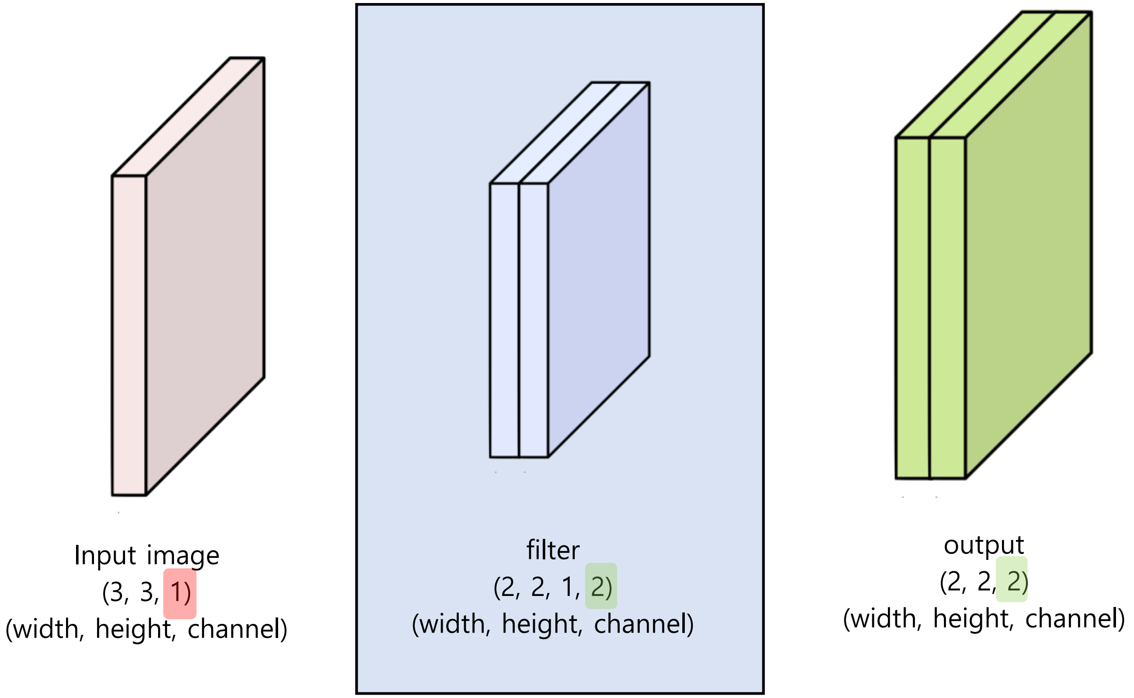
filter
(2, 2, 1)
(width, height, channel)

output
(2, 2, 1)
(width, height, channel)

Convolution

Channel

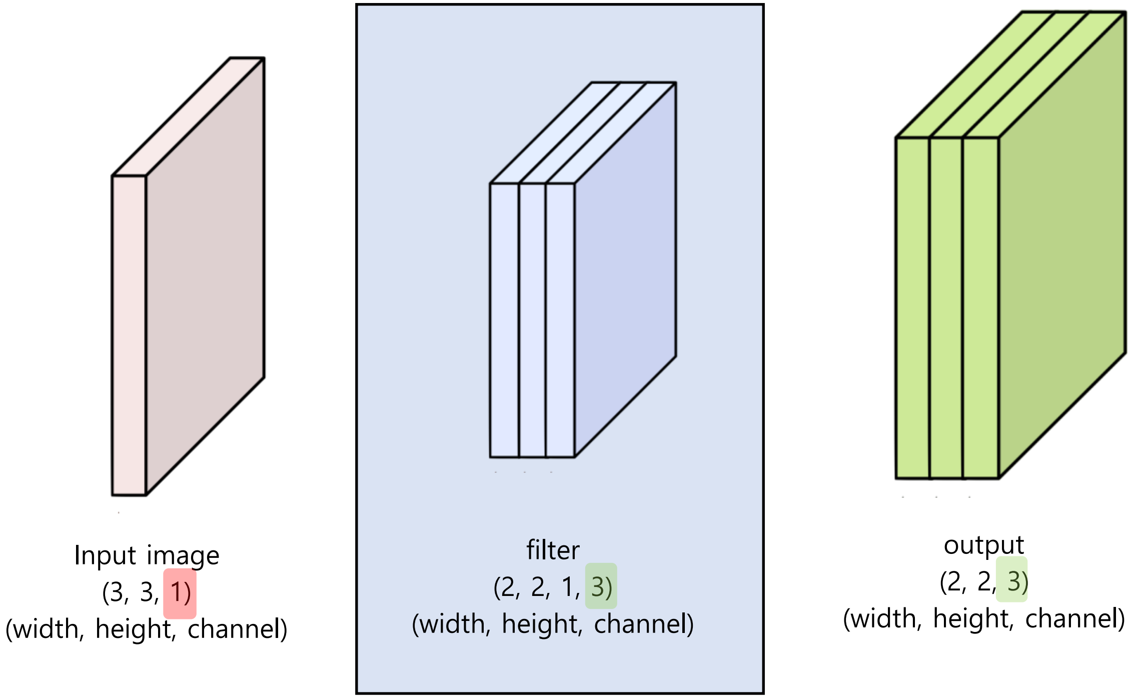
- 입력 데이터의 Channel 수와 필터의 Channel 수가 일치 해야 함
- 필터의 개수가 아웃풋의 Channel을 결정



Convolution

Channel

- 입력 데이터의 Channel 수와 필터의 Channel 수가 일치 해야 함
- 필터의 개수가 아웃풋의 Channel을 결정



Convolution

torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

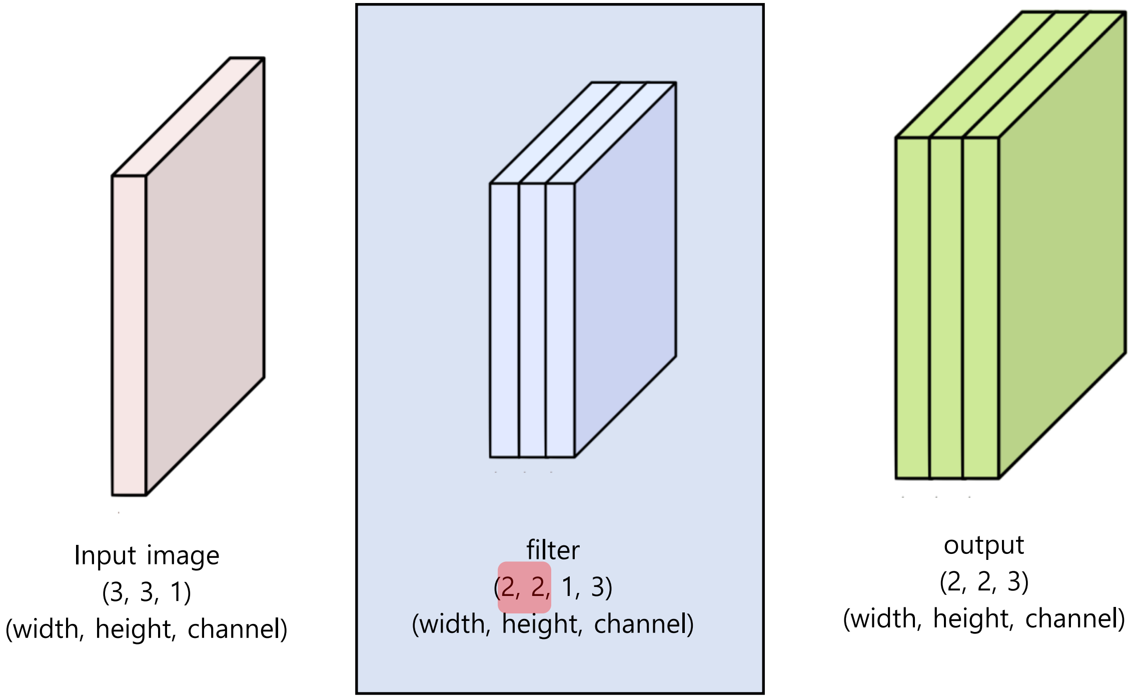
where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

N : batch size
C : # channel
H : height
W : width

Convolution

Channel

- 입력 데이터의 Channel 수와 필터의 Channel 수가 일치 해야 함
- 필터의 개수가 아웃풋의 Channel을 결정



Convolution

torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

N : batch size
C : # channel
H : height
W : width

Convolution

torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolution

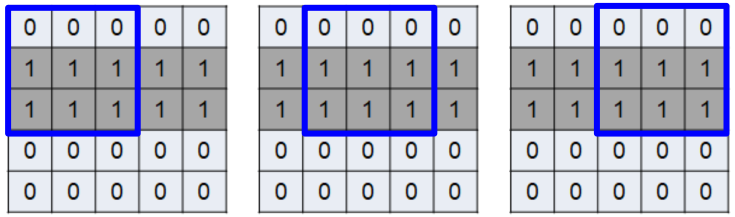
torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

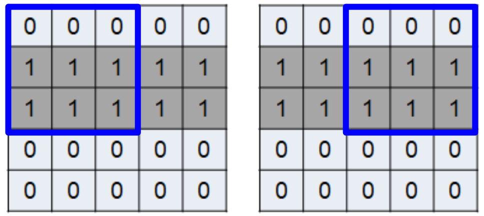
```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Stride

: 필터를 적용하는 위치의 간격



Stride = 1



Stride = 2

Convolution

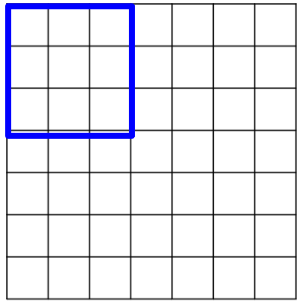
torch.nn.Conv2d

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

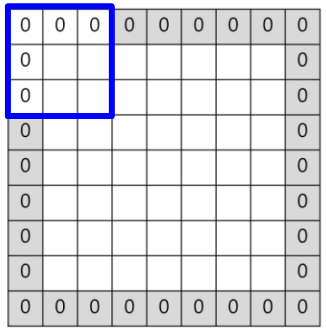
```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

Padding

: Convolution으로 인한 image 모서리 부분 정보 손실 방지를 위해 입력데
이터 주변을 특정값으로 채우는 것 (일반적으로 Zero Padding사용)



zero padding 사용X
7*7 → 5*5

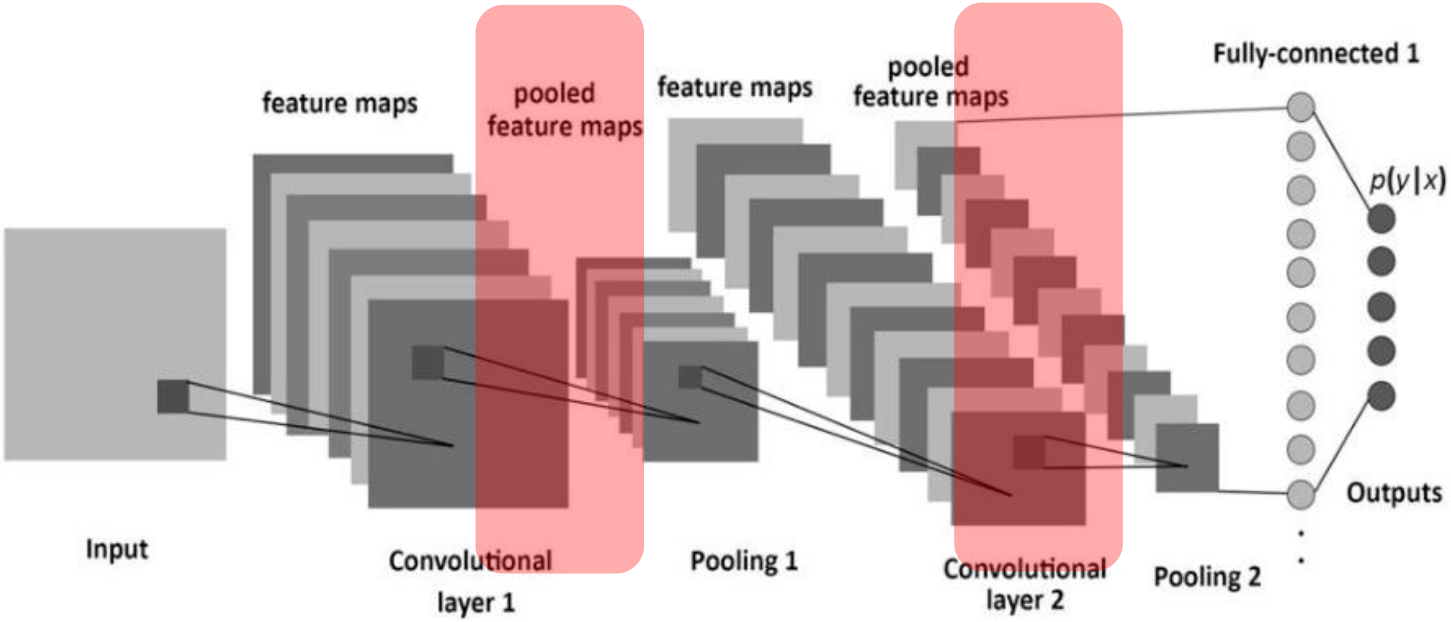


zero padding 사
용
7*7 → 7*7

`padding='valid'` is the same as no padding. `padding='same'` pads the input so the output has the shape as the input. However, this mode doesn't support any stride values other than 1.

CNN

CNN 구성요소 = (1) Convolution layer (2) Sub-sampling(pooling) layer (3) FC layer(Affine)

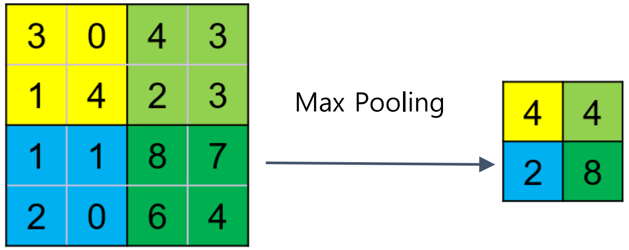


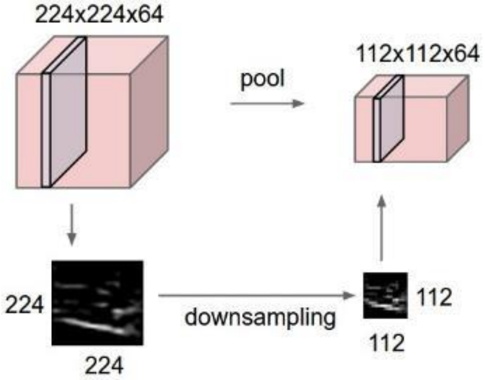
Pooling

Pooling?

: 가로 세로 방향의 공간을 줄이는 연산

- 출력의 해상도를 낮춰 변형이나 이동에 대한 민감도를 감소
- 이미지의 크기를 줄이기 때문에 학습할 노드의 수가 줄어들어 학습속도를 높이는 효과
- 하지만, 정보의 손실이 일어남
- CNN에서는 일반적으로 Max Pooling 사용





Pooling

torch.nn.MaxPool2d

https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)  [SOURCE]
```

Applies a 2D max pooling over an input signal composed of several input planes.

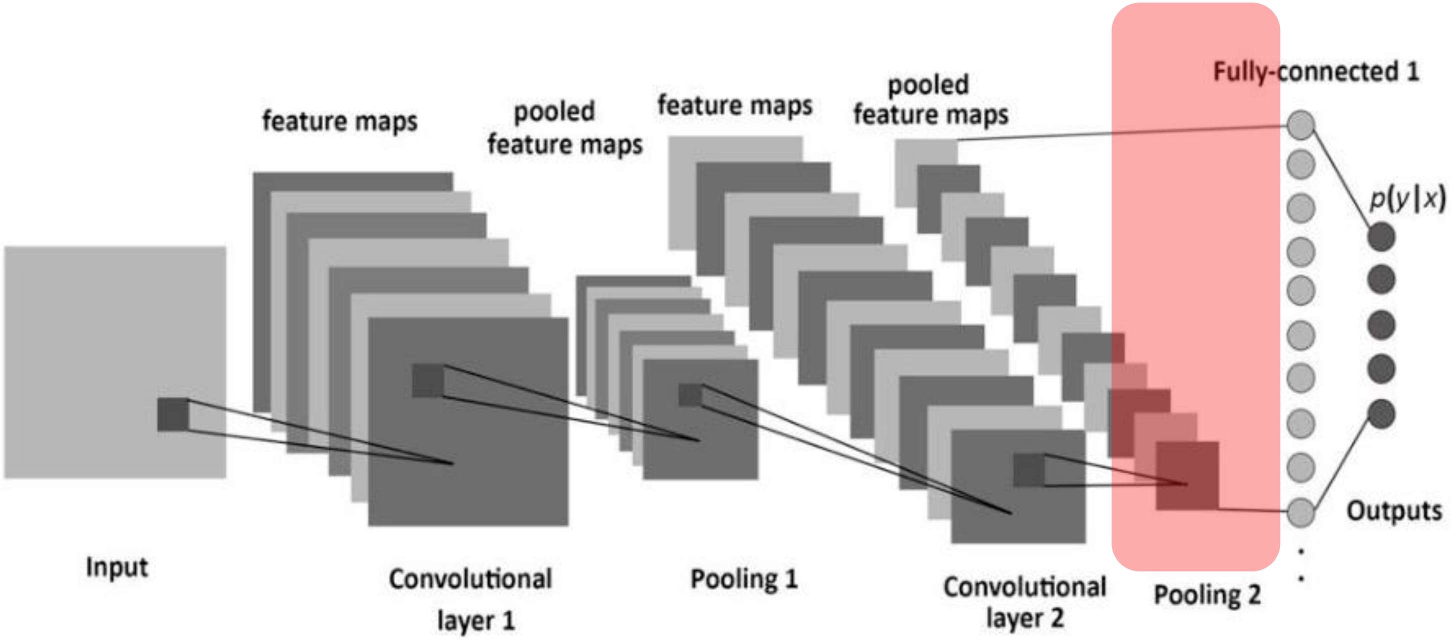
In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly padded with negative infinity on both sides for `padding` number

CNN

CNN 구성요소 = (1) Convolution layer (2) Sub-sampling(pooling) layer (3) FC layer(Affine)



Pooling

torch.flatten

https://pytorch.org/docs/stable/generated/torch.flatten.html#torch.flatten

nn.Linear를 사용하기 위해서 인풋을 반드시 1d로 만들어야 한다.

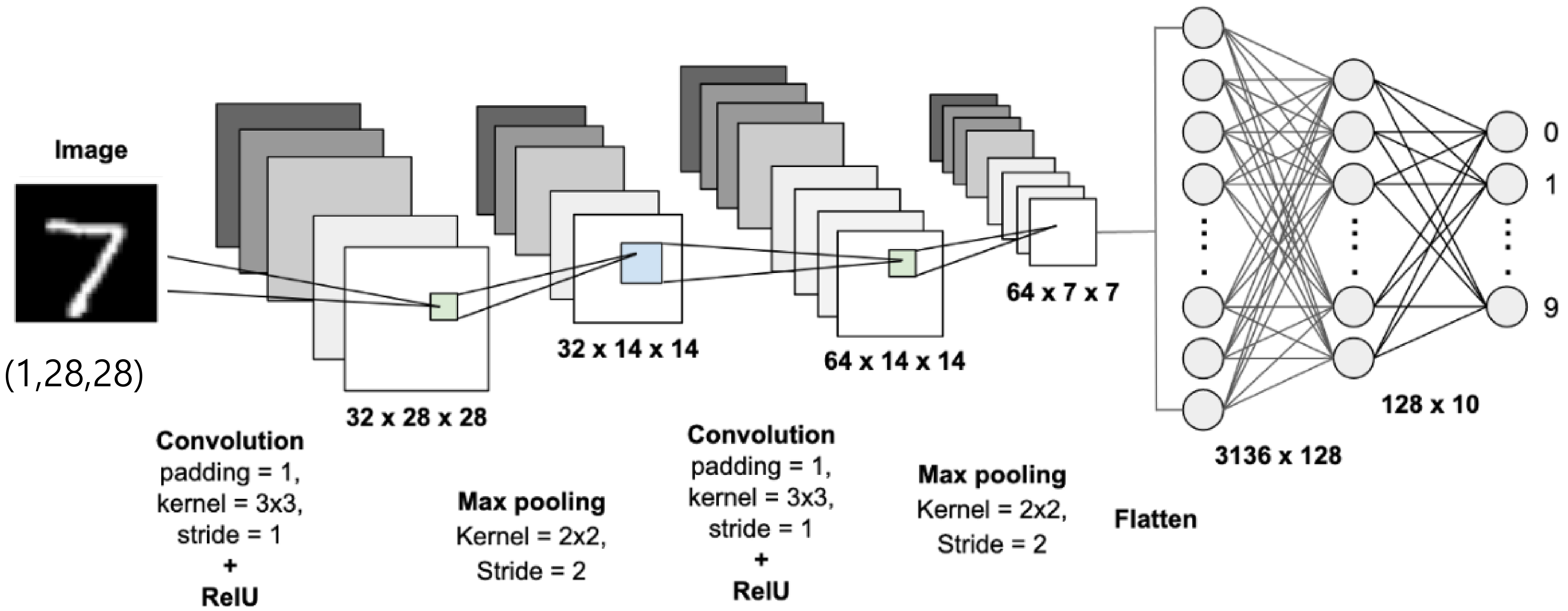
```
torch.flatten(input, start_dim=0, end_dim=- 1)
```

Flattens `input` by reshaping it into a one-dimensional tensor. If `start_dim` or `end_dim` are passed, only dimensions starting with `start_dim` and ending with `end_dim` are flattened. The order of elements in `input` is unchanged.

MNIST dataset

MNIST dataset

구조 출처: https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9



MNIST dataset

```python
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
```

```python
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.cuda.manual_seed_all(0)
```

```python
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')
```

```python
import torchvision
import torchvision.transforms as transforms

train_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                    train=True,
                    transform=transforms.ToTensor(),
                    download=True)
test_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                    train=False,
                    transform=transforms.ToTensor(),
                    download=True)
```

```python
batch_size = 128

train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
```

```python
x,y=train_dataset[0]
x.shape
```

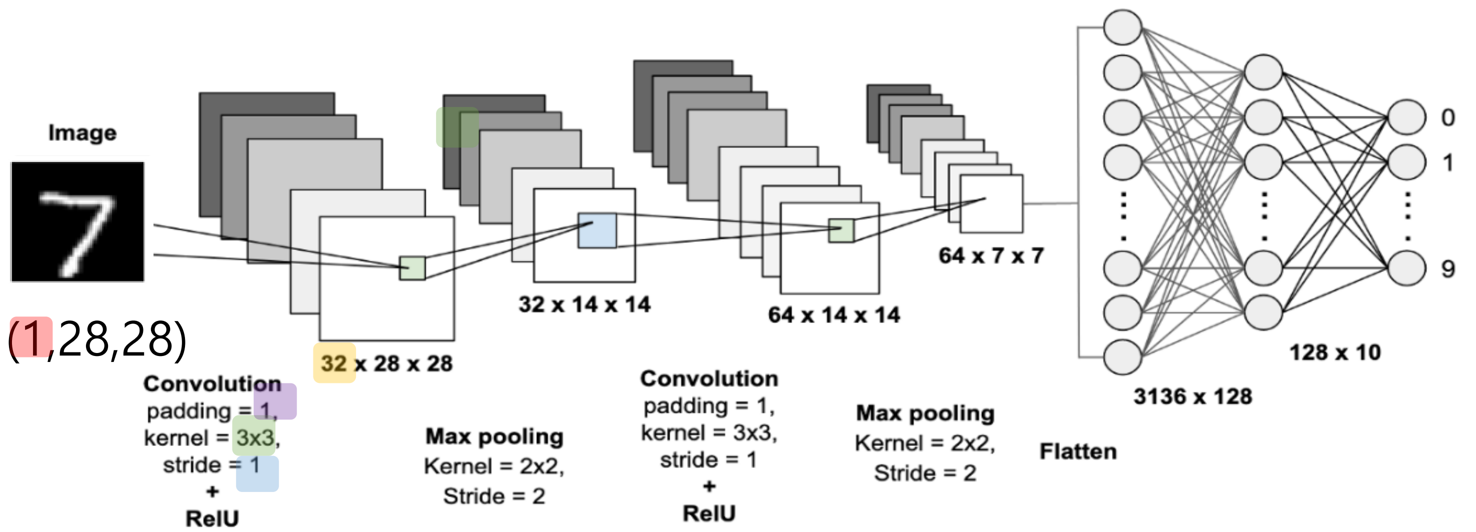
```
torch.Size([1, 28, 28])
```

MNIST dataset

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.fc1 = nn.Linear(64*7*7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.activation(self.conv1(x)))
        x = self.pool(self.activation(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.activation(self.fc1(x))
        x = self.fc2(x)
        return x
```

MNIST dataset

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1, 1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, 1, 1)
        self.fc1 = nn.Linear(64*7*7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.activation(self.c
        x = self.pool(self.activation(self.c
        x = torch.flatten(x, 1) # flatten al
        x = self.activation(self.fc1(x))
        x = self.fc2(x)
        return x
```



MNIST dataset

```
model=CNN().to(device)
```

```
optimizer = optim.Adam(model.parameters(), lr=0.001) # set optimizer
```

```
criterion = nn.CrossEntropyLoss()
```

```
epochs = 30

model.train()
for epoch in range(epochs):
    model.train()
    avg_cost = 0
    total_batch_num = len(train_dataloader)

    for b_x, b_y in train_dataloader:
      logits = model(b_x.to(device))  # forward propagation
      loss = criterion(logits, b_y.to(device)) # get cost

      avg_cost += loss / total_batch_num
      optimizer.zero_grad()
      loss.backward() # backward propagation
      optimizer.step() # update parameters
    print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

MNIST dataset

```python
correct = 0
total = 0

model.eval()
for b_x, b_y in test_dataloader:
  with torch.no_grad():
    logits = model(b_x.to(device))

  probs = nn.Softmax(dim=1)(logits)
  predicts = torch.argmax(logits, dim=1)

  total += len(b_y)
  correct += (predicts == b_y.to(device)).sum().item()

print(f'Accuracy of the network on test images: {100 * correct // total} %')
```

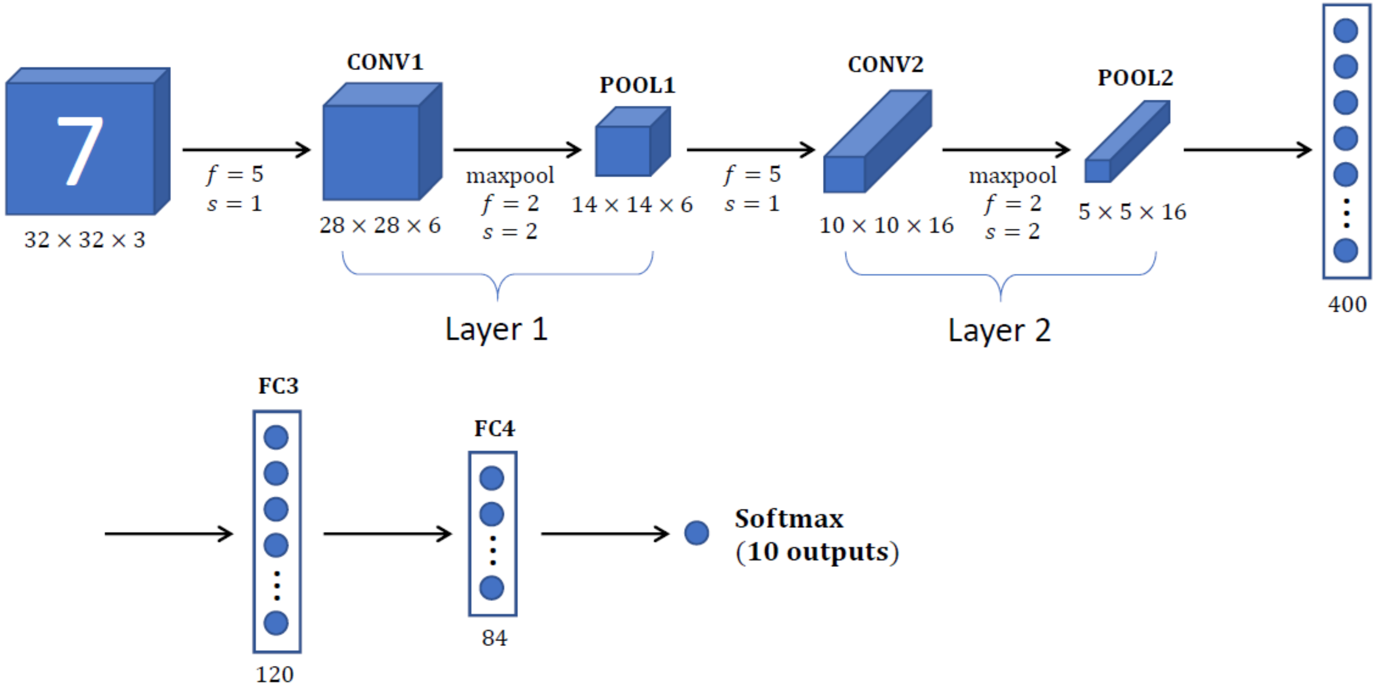
```
Accuracy of the network on test images: 99 %
```

CNN

CIFAR-10 dataset

(3,32,32)

Neural network example (LeNet-5)



CIFAR-10 dataset

```
[1] import torch
    import torch.nn as nn
    import torch.optim as optim
```

```
[2] torch.manual_seed(0)
    torch.cuda.manual_seed(0)
    torch.cuda.manual_seed_all(0)
```

```
[3] if torch.cuda.is_available():
        device = torch.device('cuda')
    else:
        device = torch.device('cpu')
```

```
[4] import torchvision
    import torchvision.transforms as transforms

    train_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                        train=True,
                                        transform=transforms.ToTensor(),
                                        download=True)
    test_dataset = torchvision.datasets.CIFAR10(root="CIFAR10/",
                                        train=False,
                                        transform=transforms.ToTensor(),
                                        download=True)
```

```
[5] batch_size = 128

    train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
```

```
x,y=train_dataset[0]
x.shape
```

```
torch.Size([3, 32, 32])
```

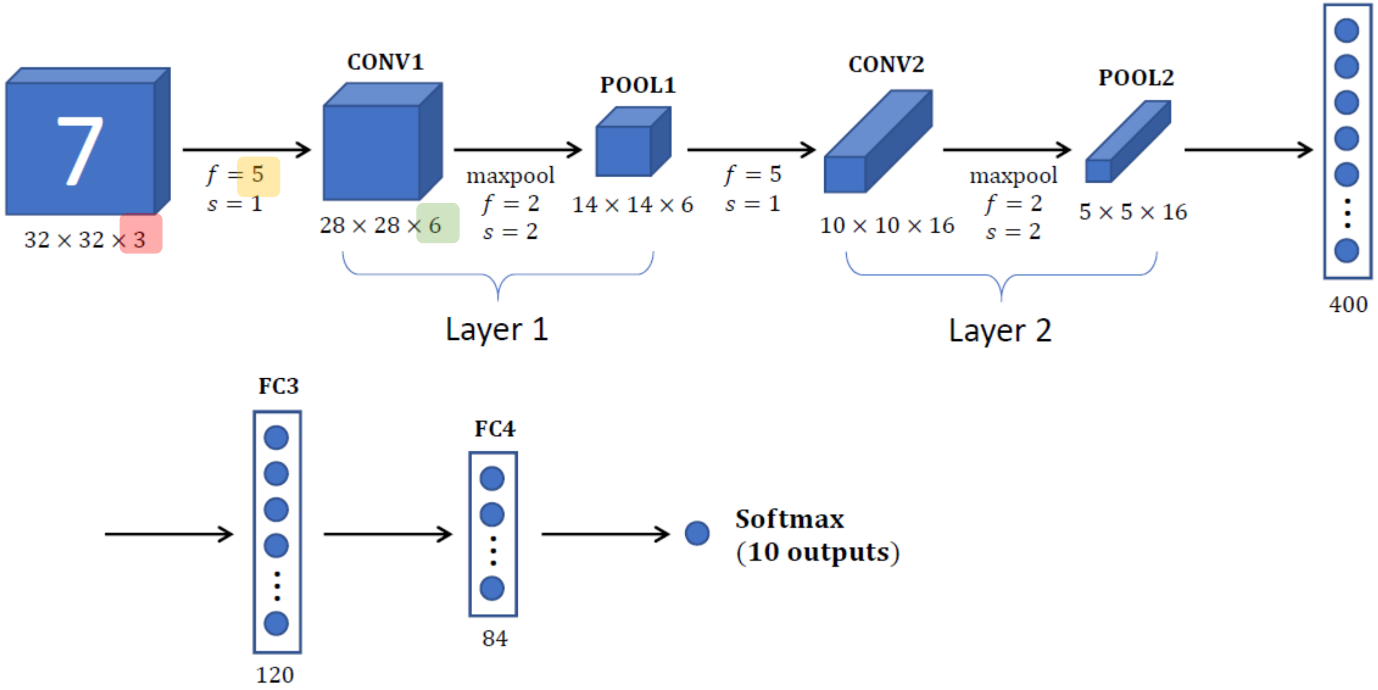
CIFAR-10 dataset

```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.activation(self.conv1(x)))
        x = self.pool(self.activation(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.fc3(x)
        return x
```

CIFAR-10 dataset

```python
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.activation(self.conv1(x)))
        x = self.pool(self.activation(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions e
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.fc3(x)
        return x
```

Neural network example (LeNet-5)



CIFAR-10 dataset

```
model=CNN().to(device)
```

```
optimizer = optim.Adam(model.parameters(), lr=0.001) # set optimizer
```

```
criterion = nn.CrossEntropyLoss()
```

```
epochs = 30

model.train()
for epoch in range(epochs):
    model.train()
    avg_cost = 0
    total_batch_num = len(train_dataloader)

    for b_x, b_y in train_dataloader:
      logits = model(b_x.to(device))  # forward propagation
      loss = criterion(logits, b_y.to(device)) # get cost

      avg_cost += loss / total_batch_num
      optimizer.zero_grad()
      loss.backward() # backward propagation
      optimizer.step() # update parameters
    print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

CIFAR-10 dataset

```python
correct = 0
total = 0

model.eval()
for b_x, b_y in test_dataloader:
  with torch.no_grad():
    logits = model(b_x.to(device))

  probs = nn.Softmax(dim=1)(logits)
  predicts = torch.argmax(logits, dim=1)

  total += len(b_y)
  correct += (predicts == b_y.to(device)).sum().item()

print(f'Accuracy of the network on test images: {100 * correct // total} %')
```

```
Accuracy of the network on test images: 63 %
```

오늘 실습 내용

1. MNIST에 대해 CNN구현
2. CIFAR-10에 대해 CNN구현