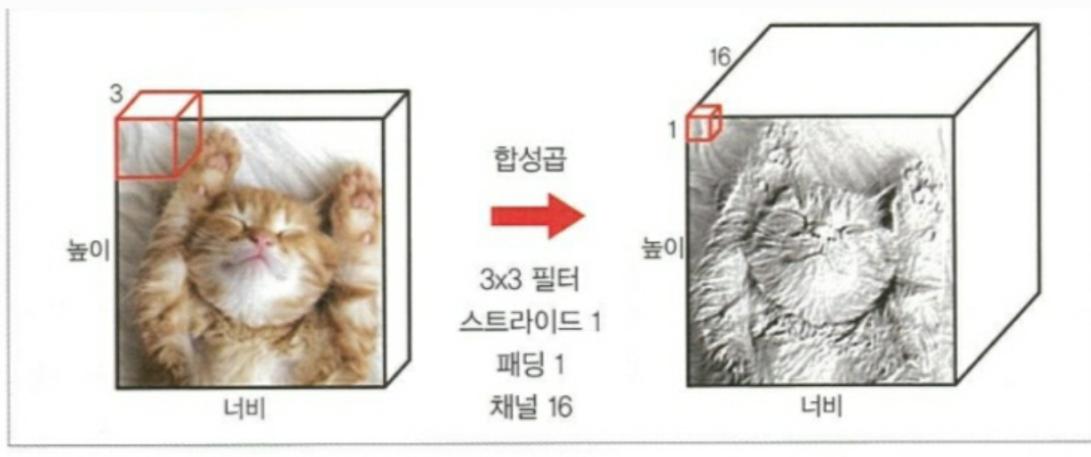
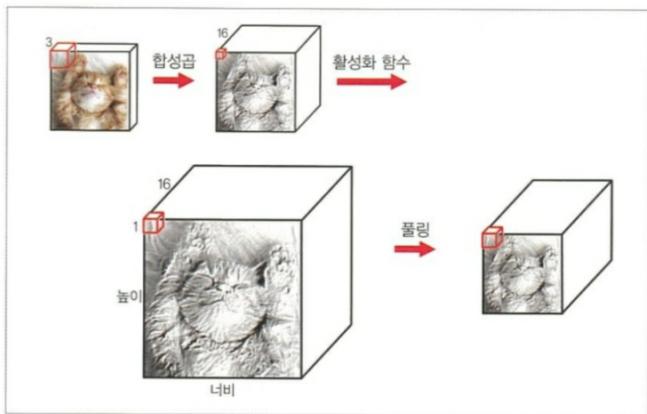


# 모델의 3차원적 이해



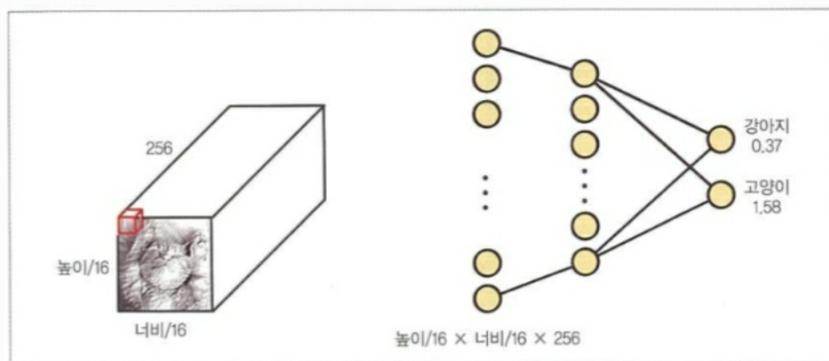
합성곱 연산의 3차원적 이해

$\text{filter: } 16 \Rightarrow \text{가로} \times \text{세로} \times 16$



합성곱, 활성화 함수, 풀링의 적용

스트라이드 2를 적용하면서 Pooling  $\Rightarrow$  크기(가로, 세로)  $\times \frac{1}{2}$ , 채널  $\times 2$



분류 문제에 적용

Pooling  $\times 4 \Rightarrow$  크기  $\times \frac{1}{16}$ , 채널  $\times 16$

## 소프트맥스 함수

- OneHotEncoding : 본래의 정답을 확률분포로 변환해주는 것  
정답에 해당하는 확률은 1, 나머지는 다 0
- SoftMax : 신경망의 결과값을 확률로 바꿔주는 함수

$$\text{softmax}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

$\hookrightarrow \text{output} \Rightarrow \text{확률값}$

## 교차 엔트로피

- 엔트로피 : 정보량의 기댓값

$$H(p) = -\sum_x p(x) \log p(x) \quad \text{if } p(x) \Downarrow \rightarrow -\log p(x) \uparrow$$

-> 일어날 확률이 작을수록 가지고 있는 정보가 크고 / 일어날 확률이 클수록 가지고 있는 정보가 작은 것

Ex) 동전 던지기     $H(x) = 0.5 \times (-\log_2 0.5) + 0.5 \times (-\log_2 0.5) = 1$

- 교차 엔트로피 : 목표로 하는 최적의 확률분포  $p$ 와 이를 근사하려는 확률분포  $q$ 가 얼마나 다른지 측정하는 방법

$$H(p, q) = -\sum_x p(x) \log q(x)$$

즉, 원래  $p$ 였던 분포를  $q$ 로 표현했을 때 얼마만큼의 비용이 드는지를 측정

ex) 동전 던지기 -> 누군가  $1/4, 3/4$ 로 앞뒤가 나올 확률 예측 이때 손실은?

$$H(p, q) = - \sum_x p(x) \log q(x) \Rightarrow p = \frac{1}{2}, q = \frac{1}{4}, \frac{3}{4}$$

$$H(p, q) = 0.5 \times (-\log_2 0.25) + 0.5 \times (-\log_2 0.75) = 1 + 0.2075 = 1.2075$$

기준의  $D_{KL}$  엔트로피 | 보다 큰 1.2075  $\Rightarrow$  손해발생!!

$$\begin{aligned} H(p, q) &= H(p) + \sum_x p(x) \log \frac{p(x)}{q(x)} \\ &= H(p) + D_{KL}(p \parallel q) \end{aligned}$$

=> 최적의 분포  $p$ 의 엔트로피에 KLD항을 더한 것

(KLD: Kullback-Leibler divergence ->  $p$ 를 기준으로  $q$ 가 얼마나 다른지)

$\therefore$  교차 엔트로피 최소화  $\Rightarrow p$ 의 엔트로피 고정된 값, KLD 최소화  $\Rightarrow p$ 와  $q$ 가 최대한 같게

L1Loss 와 CrossEntropy 비교 :

$$L1Loss \Rightarrow L = \sum_{i=1}^n |y_i - f(x_i)|$$

$$\text{Ex) } \begin{bmatrix} 0.1 & 0.9 \\ p & c \end{bmatrix} \text{ 예측 } \Rightarrow \begin{bmatrix} \text{Entropy} \Rightarrow 0.1053 \\ L_1 \Rightarrow 0.1 \end{bmatrix}$$

$$\begin{bmatrix} 0.7 & 0.3 \\ p & c \end{bmatrix} \text{ 예측 } \Rightarrow \begin{bmatrix} \text{Entropy} \Rightarrow 1.2039 \\ L_1 \Rightarrow 0.7 \end{bmatrix}$$

$$\begin{bmatrix} 0.9 & 0.1 \\ p & c \end{bmatrix} \text{ 예측 } \Rightarrow \begin{bmatrix} \text{Entropy} \Rightarrow 2.3025 \\ L_1 \Rightarrow 0.9 \end{bmatrix}$$

예측이 잘못될 수록  
 $L_1$  보다 더 크게 증가

## 모델 구현, 학습 및 결과 확인

- 하이퍼파라미터 : 학습의 대상이 아닌 학습 이전에 정해놓는 변수  
ex. filter 개수, hidden\_layer 개수 등등..
- torch.utils.data.DataLoader : 효율적인 학습을 위해 배치 사이즈대로 묶어서 전달하거나 어떤 규칙에 따라 정렬하거나 섞는 모듈
- torchvision.dataset : 데이터를 읽어오는 모듈
- torchvision.transforms : 이미지를 필요에 따라 변환해주는 역할

```
mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(),
                         target_transform=None, download=True)
mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(),
                        target_transform=None, download=True)

train_loader = torch.utils.data.DataLoader(mnist_train, batch_size=batch_size,
                                           shuffle=True, num_workers=2, drop_last=True)
test_loader = torch.utils.data.DataLoader(mnist_test, batch_size=batch_size,
                                         shuffle=False, num_workers=2, drop_last=True)
```

### - dset.MNIST

**transform** 옵션 : 이미지에 대한 변형(tensor형으로 변환함)

**target\_transform** 옵션 : 라벨에 대한 변형(ex. OneHotEncoding)

### - torch.utils.data.DataLoader

**batch\_size** = 배치 사이즈만큼 데이터를 묶는다

**num\_workers** = 데이터를 묶을 때 사용할 프로세스 개수

**drop\_last** = 묶고 남는 데이터는 버릴지 여부

```

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(1, 16, 5), out_channels → kernel_size
            nn.ReLU(),
            nn.Conv2d(16, 32, 5), [batch, 16, 24, 24]
        )

```

MNIST Dataset  
[batch-size, 1, 가로, 세로]

5장 합성곱 신경망

```

        nn.ReLU(), → [batch, 32, 20, 20]
        nn.MaxPool2d(2,2), → [batch, 32, 10, 10]
        nn.Conv2d(32,64,5) → [batch, 64, 6, 6]
        nn.ReLU(),
        nn.MaxPool2d(2,2) → [batch, 64, 3, 3]
    )
    self.fc_layer = nn.Sequential(
        nn.Linear(64*3*3, 100),
        nn.ReLU(),
        nn.Linear(100, 10)
    ) → output 클래스 개수

```

def forward(self, x):
 out = self.layer(x)
 out = out.view(batch\_size, -1)
 out = self.fc\_layer(out)
 return out

\_\_init\_\_ : super - CNN class의 부모 클래스인 nn.Module 초기화  
합성곱 연산 부분 :

- in\_channels,out\_channels : 입력 채널, 필터 개수

• 예시에서 in\_channels = RGB 채널=3 , out\_channels = 필터 개수 16  
(고양이)

배치까지 포함한 tensor의 형 태에서 입력 [batch\_size, in\_channels, 가로, 세로]

↳ 합성곱 연산의 결과 [batch\_size, out\_channels, 가로, 세로]

[batch, 1, 28, 28]  $\xrightarrow{\text{nn.Conv2d}(1, 16, 5)}$  [batch, 16, 24, 24] ( $\because O = \text{floor} \left( \frac{I - k + 2P}{s} + 1 \right) \Rightarrow \text{floor} \left( \frac{28 - 5 + 0}{4} + 1 \right) = 24$ )  
I : 가로 or 세로 길이

P : 따로 지정 안하면 0

S : default = 1

K : kernel size

nn.MaxPool2d(2,2) : kernel\_size,stride,padding 등등..을 인수로 받음

- kernel\_size : 풀링 연산할 때 한 번에 훑는 영역

- stride : 이동하는 수

따라서 nn.MaxPool2d(2,2) : 2x2 영역에서 풀링을 하고 2만큼 이동

-> 텐서가 반으로 줄어든다!

fc\_layer : 계산된 tensor의 채널x가로x세로를 input으로 넣어주고

hidden layer로 보냄

forward 함수 부분: view 함수로 tensor를 reshape 해준다.

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = CNN().to(device) ~ GPU에 옮김
loss_func = nn.CrossEntropyLoss() ~ 손실함수 지정
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
    최적화 함수
```

```
loss_arr = []
for i in range(num_epoch):
```

76 파이토치 첫걸음

```
for j,[image,label] in enumerate(train_loader):
    x = image.to(device)
    y_ = label.to(device)

    optimizer.zero_grad() ~ 초기화 함수 초기화
    output = model.forward(x)
    loss = loss_func(output,y_)
    loss.backward()
    optimizer.step()

    if j % 1000 == 0:
        print(loss)
        loss_arr.append(loss.cpu().detach().numpy())
```

```
correct = 0
total = 0
with torch.no_grad():
    for image,label in test_loader:
        x = image.to(device)
        y_= label.to(device)

        output = model.forward(x)
        _,output_index = torch.max(output,1)

        total += label.size(0)
        correct += (output_index == y_).sum().float()

print("Accuracy of Test Data: {}".format(100*correct/total))
```

기울기 계산 X