5장
합성곱 신경망 I

5장 합성곱 신경망 I

5.1 합성곱 신경망

5.2 합성곱 신경망 맛보기

5.3 전이 학습

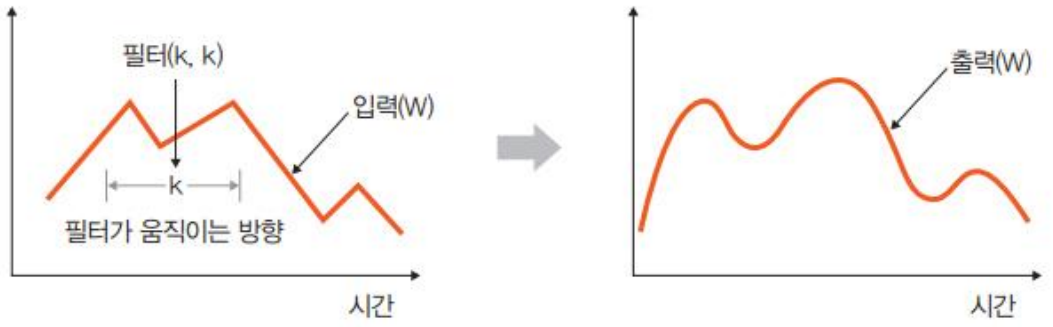
5.4 설명 가능한 CNN

5.1 합성곱 신경망

- **1D, 2D, 3D 합성곱**
  - 합성곱은 이동하는 방향의 수와 출력 형태에 따라 1D, 2D, 3D로 분류할 수 있음

1D 합성곱

- 1D 합성곱은 필터가 시간을 축으로 좌우로만 이동할 수 있는 합성곱
- 입력(W)과 필터(k)에 대한 출력은 W가 됨
- 예를 들어 입력이 [1, 1, 1, 1, 1]이고 필터가 [0.25, 0.5, 0.25]라면, 출력은 [1, 1, 1]이 됨
- 즉, 다음 그림과 같이 출력 형태는 1D의 배열이 되며, 그래프 곡선을 완화할 때 많이 사용



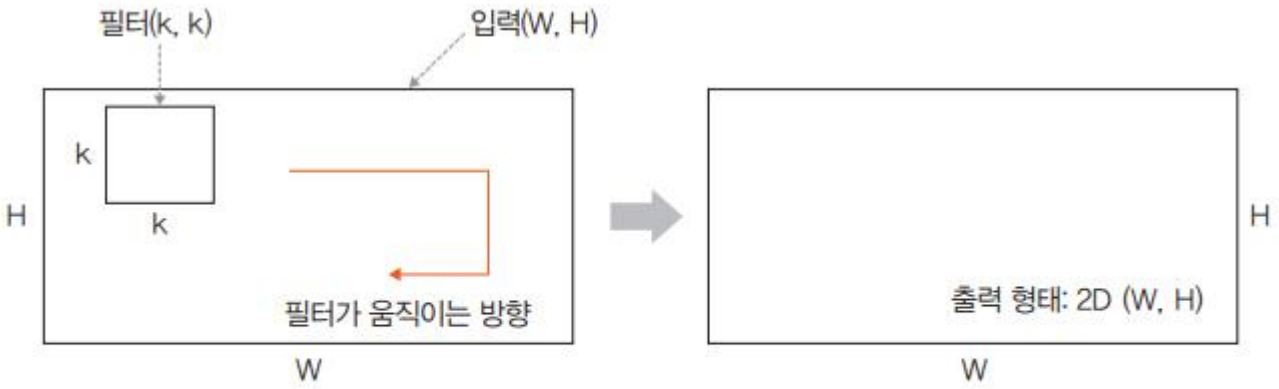
- 입력: W 너비(Width)
- 필터: k×k(높이×너비)
- 출력: W 너비(Width)

5.1 합성곱 신경망

- **1D, 2D, 3D 합성곱**

  2D 합성곱

  - 2D 합성곱은 필터가 다음 그림과 같이 방향 두 개로 움직이는 형태
  - 즉, 입력(W, H)과 필터(k, k)에 대한 출력은 (W, H)가 되며, 출력 형태는 2D 행렬이 됨



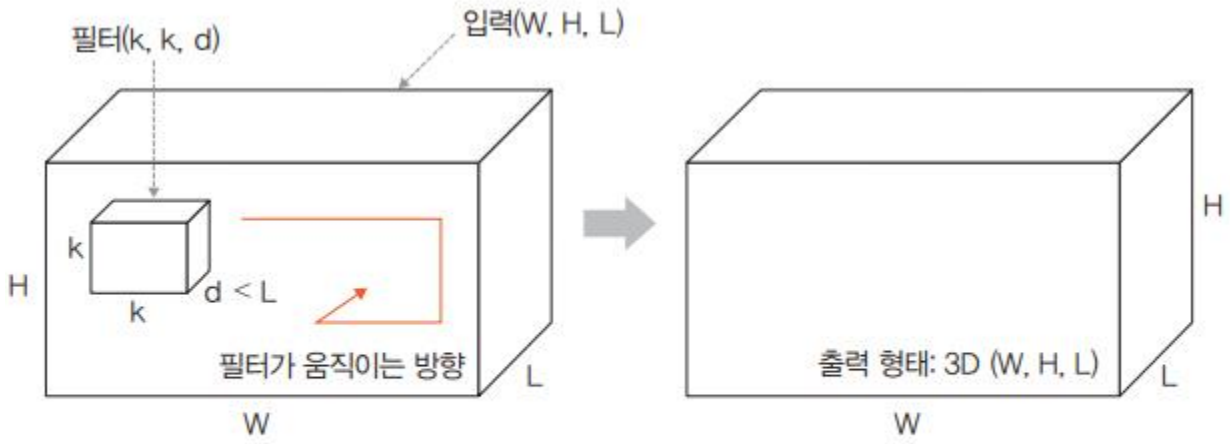
- 입력: W 너비(Width), H 높이(Height)
- 필터: k×k(높이×너비)
- 출력: W 너비(Width), H 높이(Height)

5.1 합성곱 신경망

- **1D, 2D, 3D 합성곱**

  3D 합성곱

  - 3D 합성곱은 필터가 움직이는 방향이 그림 5-21과 같이 세 개 있음
  - 입력(W, H, L)에 대해 필터(k, k, d)를 적용하면 출력으로 (W, H, L)을 갖는 형태가 3D 합성곱
  - 출력은 3D 형태이며, 이때 d < L을 유지하는 것이 중요



- 입력: W 너비(Width), H 높이(Height), L 길이(Length)
- 필터: k×k(높이×너비), d: 깊이(depth)
- 출력: W 너비(Width), H 높이(Height), L 길이(Length)

5.2 합성곱 신경망 맛보기

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**

  fashion_mnist 데이터셋

  - fashion_mnist 데이터셋은 토치비전(torchvision)에 내장된 예제 데이터로 운동화, 셔츠, 샌들 같은 작은 이미지의 모음
  - 기본 MNIST 데이터셋처럼 열 가지로 분류될 수 있는 28×28 픽셀의 이미지 7만 개로 구성되어 있음
  - 데이터셋을 자세히 살펴보면 훈련 데이터(train_images)는 0에서 255 사이의 값을 갖는 28×28 크기의 넘파이(NumPy) 배열
  - 레이블(정답) 데이터(train_labels)는 0에서 9까지 정수 값을 갖는 배열

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 0에서 9까지 정수 값은 이미지(운동화, 셔츠 등)의 클래스를 나타내는 레이블
  - 각 레이블과 클래스는 다음과 같음

```
0 : T-Shirt
1 : Trouser
2 : Pullover
3 : Dress
4 : Coat
5 : Sandal
6 : Shirt
7 : Sneaker
8 : Bag
9 : Ankle Boot
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 예제 진행을 위해 먼저 필요한 라이브러리를 호출

코드 5-1 라이브러리 호출

```
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F

import torchvision
import torchvision.transforms as transforms ------ 데이터 전처리를 위해 사용하는 라이브러리
from torch.utils.data import Dataset, DataLoader
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 파이토치는 기본적으로 GPU 사용을 권장
  - GPU가 장착되지 않은 환경에서도 파이토치를 정상적으로 실행하고 사용할 수 있음
  - GPU가 장착되어 있고, GPU를 사용하기 위한 설정이 되어 있다면 파이토치에서 자동으로 인식

코드 5-2 CPU 혹은 GPU 장치 확인

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**

  GPU 사용

  - 일반적으로 하나의 GPU를 사용할 때는 다음과 같은 코드를 이용

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = Net()
model.to(device)
```

  - 사용하는 PC에서 다수의 GPU를 사용한다면 다음 코드와 같이 nn.DataParallel을 이용

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Net()
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(net)
model.to(device)
```

  - nn.DataParallel을 사용할 경우 배치 크기(batch size)가 알아서 각 GPU로 분배되는 방식으로 작동. 파이토치에서 제공하는 모델 병렬 처리 기능 중 하나.
  - GPU 수만큼 배치 크기도 늘려 주어야 함

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ① torchvision.datasets는 torch.utils.data.Dataset의 하위 클래스로 다양한 데이터셋(CIFAR, COCO, MNIST, ImageNet 등)을 포함
  - torchvision.datasets에서 사용하는 주요한 파라미터는 다음과 같음

```
torchvision.datasets.FashionMNIST("../chap05/data", download=True,
                                  transform=transforms.Compose([transforms.ToTensor()]))
```

ⓐ ⓑ ⓒ

ⓐ 첫 번째 파라미터: FashionMNIST를 내려받을 위치를 지정

ⓑ download: download를 True로 변경해 주면 첫 번째 파라미터의 위치에 해당 데이터셋이 있는지 확인한 후 내려받음

ⓒ transform: 이미지를 텐서(0~1)로 변경

사용할 fashion_mnist 데이터셋은 토치비전으로 내려받을 수 있음

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ① torch.utils.data.DataLoader()를 사용하여 원하는 크기의 배치 단위로 데이터를 불러오거나, 순서가 무작위로 섞이도록(shuffle) 할 수 있음
  - 데이터로더에서 사용하는 파라미터는 다음과 같음

```
torch.utils.data.DataLoader(train_dataset, batch_size=100)
                            ⓐ              ⓑ
```

ⓐ 첫 번째 파라미터: 데이터를 불러올 데이터셋을 지정

ⓑ batch_size: 데이터를 배치로 묶어 줌

여기에서는 batch_size=100으로 지정했기 때문에 100개 단위로 데이터를 묶어서 불러옴

코드 5-4 fashion_mnist 데이터를 데이터로더에 전달

```
train_loader = torch.utils.data.DataLoader(train_dataset,
                                           batch_size=100) ------①
test_loader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=100)
```

내려받은 fashion_mnist 데이터를 메모리로 불러오기 위해 데이터로더(DataLoader)에 전달

5.2 합성곱 신경망 맛보기

▼ 그림 5-24 20개의 이미지 데이터를 시각적으로 표현

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ① 클래스(class) 형태의 모델은 항상 torch.nn.Module을 상속받음
  - _ _init_ _()은 객체가 갖는 속성 값을 초기화하는 역할을 하며, 객체가 생성될 때 자동으로 호출
  - super(FashionDNN, self)._ _init_ _()은 FashionDNN이라는 부모(super) 클래스를 상속받겠다는 의미로 이해하면 됨

  - 이때 객체(object)란 메모리를 할당받아 프로그램에서 사용되는 모든 데이터를 의미하기 때문에 변수, 함수 등은 모두 객체라고 할 수 있음
  - 객체는 다음과 같은 방식으로 사용

```
객체명 = 클래스명( )
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**

  클래스와 함수

  - 함수(function)란 하나의 특정 작업을 수행하기 위해 독립적으로 설계된 프로그램 코드
  - 함수의 호출은 특정 작업만 수행할 뿐 그 결괏값을 계속 사용하기 위해서는 반드시 어딘가에 따로 그 값을 저장해야만 함
  - 즉, 함수를 포함한 프로그램 코드의 일부를 재사용하기 위해서는 해당 함수뿐만 아니라 데이터가 저장되는 변수까지도 한꺼번에 관리
  - 이처럼 함수뿐만 아니라 관련된 변수까지도 한꺼번에 묶어서 관리하고 재사용할 수 있게 해 주는 것이 클래스(class)

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 클래스와 함수의 차이를 코드로 살펴보자
  - 먼저 함수에 대한 예시

```
def add(num1, num2): ····· 함수 정의(num1, num2를 받아서 더해 주는 함수)
    result = num1 + num2
    return result

print(add(1, 2))
print(add(2, 3))
```

  - 코드를 실행하면 다음과 같이 출력

```
3
5
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 다음은 클래스에 대한 코드

```
class Calc:
    def __init__(self): ------ 객체를 생성할 때 호출하면 실행되는 초기화 함수
        self.result = 0

    def add(self, num1, num2):
        self.result = num1 + num2
        return self.result

obj1 = Calc()
obj2 = Calc()

print(obj1.add(1, 2))
print(obj1.add(2, 3))
print('--------------------')
print(obj2.add(2, 2))
print(obj2.add(2, 3))
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ② nn은 딥러닝 모델(네트워크) 구성에 필요한 모듈이 모여 있는 패키지이며, Linear는 단순 선형 회귀 모델을 만들 때 사용
  - 이때 사용되는 파라미터는 다음과 같음

```
nn.Linear(in_features=784, out_features=256)
          ⓐ                ⓑ
```

ⓐ in_features: 입력의 크기(input size)

ⓑ out_features: 출력의 크기(output size)

  - 실제로 데이터 연산이 진행되는 forward() 부분에는 첫 번째 파라미터 값만 넘겨주게 되며, 두 번째 파라미터에서 정의된 크기가 forward() 연산의 결과가 됨

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ④ forward() 함수는 모델이 학습 데이터를 입력받아서 순전파(forward propagation) 학습을 진행시키며, 반드시 forward라는 이름의 함수여야 함
  - 즉, forward()는 모델이 학습 데이터를 입력받아서 순전파 연산을 진행하는 함수이며, 객체를 데이터와 함께 호출하면 자동으로 실행
  - 이때 순전파 연산이란 H(x) 식에 입력 x로부터 예측된 y를 얻는 것

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ⑤ 파이토치에서 사용하는 뷰(view)는 넘파이의 reshape과 같은 역할로 텐서의 크기(shape)를 변경해 주는 역할을 함
  - input_data.view(-1, 784)는 input_data를 (?, 784)의 크기로 변경하라는 의미
  - 이때 첫 번째 차원(-1)은 사용자가 잘 모르겠으니 파이토치에 맡기겠다는 의미이고, 두 번째 차원의 길이는 784를 가지도록 하라는 의미
  - 다시 말해 2차원 텐서로 변경하되 (?, 784)의 크기로 변경하라는 의미

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ⑥ 활성화 함수를 지정할 때는 다음 두 가지 방법이 가능

    - F.relu(): forward() 함수에서 정의
    - nn.ReLU(): _ _init_ _() 함수에서 정의

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - 모델을 학습시키기 전에 손실 함수, 학습률(learning rate), 옵티마이저(optimizer)에 대해 정의

코드 5-7 심층 신경망에서 필요한 파라미터 정의

```
learning_rate = 0.001;
model = FashionDNN();
model.to(device)

criterion = nn.CrossEntropyLoss(); ------ 분류 문제에서 사용하는 손실 함수
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate); ------ ①
print(model)
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ① 옵티마이저를 위한 경사 하강법은 Adam을 사용하며, 학습률을 의미하는 lr은 0.001을 사용한다는 의미
  - 앞 코드를 실행하면 앞에서 다음과 같이 생성한 심층 신경망 모델을 보여 줌

```
FashionDNN(
  (fc1): Linear(in_features=784, out_features=256, bias=True)
  (drop): Dropout(p=0.25, inplace=False)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
)
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ③ 모델이 데이터를 처리하기 위해서는 모델과 데이터가 동일한 장치(CPU 또는 GPU)에 있어야 함
  - model.to(device)가 GPU를 사용했다면, images.to(device), labels.to(device)도 GPU에서 처리되어야 함
  - 참고로 CPU에서 처리된 데이터를 GPU 모델에 적용하거나 그 반대의 경우 런타임 오류가 발생

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ④ Autograd는 자동 미분을 수행하는 파이토치의 핵심 패키지로, 자동 미분에 대한 값을 저장하기 위해 테이프(tape)를 사용
  - 순전파(foward) 단계에서 테이프는 수행하는 모든 연산을 저장
  - 역전파(backward) 단계에서 저장된 값들을 꺼내서 사용
  - 즉, Autograd는 Variable을 사용해서 역전파를 위한 미분 값을 자동으로 계산해 줌
  - 자동 미분을 계산하기 위해서는 torch.autograd 패키지 안에 있는 Variable을 이용해야 동작
  - 신경망(NN; Neural Network)은 어떤 입력 데이터에 대해 실행되는 중첩(nested)된 함수들의 모음(collection)입니다. 이 함수들은 PyTorch에서 Tensor로 저장되는, (가중치(weight)와 편향(bias)로 구성된) 매개변수들로 정의됩니다.

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ⑤ 분류 문제에 대한 정확도는 전체 예측에 대한 정확한 예측의 비율로 표현할 수 있으며, 코드는 다음과 같음

```
classification accuracy = correct predictions / total predictions
```

  - 이때 결과에 100을 곱하여 백분율로 표시하는 코드는 다음과 같음

```
classification accuracy = correct predictions / total predictions * 100
```

  - 분류 문제에 대한 정확도는 다음과 같이 값을 반전시켜 오분류율 또는 오류율로 표현할 수 있음

```
error rate = (1 - (correct predictions / total predictions)) * 100
```

5.2 합성곱 신경망 맛보기

- **합성곱 신경망 맛보기**
  - ① nn.Sequential을 사용하면 _ _init_ _()에서 사용할 네트워크 모델들을 정의해 줄 뿐만 아니라, forward() 함수에서 구현될 순전파를 계층(layer) 형태로 좀 더 가독성이 뛰어난 코드로 작성할 수 있음
  - 즉, nn.Sequential은 계층을 차례로 쌓을 수 있도록 Wx + b와 같은 수식과 활성화 함수를 연결해 주는 역할을 함
  - 특히 데이터가 각 계층을 순차적으로 지나갈 때 사용하면 좋은 방법
  - 정리하면 nn.Sequential은 여러 개의 계층을 하나의 컨테이너에 구현하는 방법이라고 생각하면 됨

5.3 전이 학습

- **특성 추출 기법**
  - 이미지 분류 모델

    - Xception : Google에서 개발한 컨볼루션 신경망, 파라미터 수를 줄여 성능향상
    - Inception V3 : 이미지 분류, 객체 검출, 분할 등의 다양한 작업에서 사용됩니다.
    - ResNet50 : 깊이가 깊어질수록 발생하는 그래디언트 소실 문제를 해결
    - VGG16 , 19 : 모든 합성곱 층이 3x3 크기의 작은 필터(filter)를 사용하여 구성
    - MobileNet :모바일 환경에서의 컴퓨팅 리소스를 고려하여 경량화

5.3 전이 학습

- **특성 추출 기법**
  - ① torchvision.transfrom은 이미지 데이터를 변환하여 모델(네트워크)의 입력으로 사용할 수 있게 변환해 줌
  - 이때 사용되는 파라미터는 다음과 같음

```
transform = transforms.Compose([transforms.Resize([256, 256]),
                                ⓐ
                transforms.RandomResizedCrop(224), transforms.RandomHorizontalFlip(),
                                ⓑ                                ⓒ
                transforms.ToTensor()])
                                ⓓ
```

ⓐ Resize: 이미지의 크기를 조정. 즉, 256×256 크기로 이미지 데이터를 조정

ⓑ RandomResizedCrop: 이미지를 랜덤한 크기 및 비율로 자름

Resize와 RandomResizedCrop 모두 이미지를 자르는 데 사용하지만 그 용도는 다름

Resize가 합성곱층을 통과하기 위해 이미지 크기를 조정하는 전처리 과정이라면,

RandomResizedCrop은 데이터 확장 용도로 사용

RandomResizedCrop은 이미지를 랜덤한 비율로 자른 후 데이터 크기를 조정

ⓒ RandomHorizontalFlip: 이미지를 랜덤하게 수평으로 뒤집음

ⓓ ToTensor: 이미지 데이터를 텐서로 변환

5.3 전이 학습

- **특성 추출 기법**
  - ② datasets.ImageFolder는 데이터로더가 데이터를 불러올 대상(혹은 경로)과 방법(transform)(혹은 전처리)을 정의하며, 사용하는 파라미터는 다음과 같음

```
train_dataset = torchvision.datasets.ImageFolder(data_path,
                                                 ⓐ
                                                 transform=transform)
                                                 ⓑ
```

ⓐ 첫 번째 파라미터: 불러올 데이터가 위치한 경로

ⓑ transform: 이미지 데이터에 대한 전처리

5.3 전이 학습

- **특성 추출 기법**
  - ③ 데이터로더는 데이터를 불러오는 부분으로 앞에서 정의한 ImageFolder(train_dataset)을 데이터로더에 할당하는데, 이때 한 번에 불러올 데이터양을 결정하는 batch_size를 지정
  - 또한, 추가적으로 데이터를 무작위로 섞을(shuffle) 것인지도 설정
  - 데이터로더에서 사용하는 파라미터는 다음과 같음

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
                                           num_workers=8, shuffle=True)
```

ⓐ ⓑ ⓒ ⓓ

ⓐ 첫 번째 파라미터: 데이터셋을 지정

ⓑ batch_size: 한 번에 불러올 데이터양을 결정하는 배치 크기를 설정

5.4 설명 가능한 CNN

- **특성 맵 시각화**
  - 특성 맵(feature map)(혹은 활성화 맵)은 입력 이미지 또는 다른 특성 맵처럼 필터를 입력에 적용한 결과
  - 특정 입력 이미지에 대한 특성 맵을 시각화한다는 의미는 특성 맵에서 입력 특성을 감지하는 방법을 이해할 수 있도록 돕는 것
  - 먼저 이미지 분석 및 처리를 쉽게 할 수 있도록 도와주는 라이브러리인 PIL(Python Image Library)을 설치
  - PIL은 다양한 이미지 파일 형식을 지원하며, 강력한 이미지 처리와 그래픽 기능을 제공하는 이미지 프로세싱 라이브러리

```
> pip install pillow
```

5.4 설명 가능한 CNN

- **특성 맵 시각화**
  - ① 로그 소프트맥스(log_softmax())는 신경망 말단의 결괏값들을 확률 개념으로 해석하기 위해 소프트맥스(softmax) 함수의 결과에 log 값을 취한 연산
  - 소프트맥스를 사용하지 않고 로그 소프트맥스를 사용하는 이유는 소프트맥스는 기울기 소멸 문제(vanishing gradient problem)에 취약하기 때문임
  - 다음은 소프트맥스와 로그 소프트맥스에 대한 수식

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1} e^{x_j}}$$

$$\text{logsoftmax} = \log\left(\frac{e^{x_i}}{\sum_{j=1} e^{x_j}}\right)$$

$$= x_i - \log\left(\sum_{j=1} e^{x_j}\right)$$

5.4 설명 가능한 CNN

- **특성 맵 시각화**
  - 앞에서 생성된 모델을 model = XAI()로 객체화한 후 장치(CPU 혹은 GPU)에 할당

코드 5-32 모델 객체화

```
model = XAI()  ------ model이라는 이름의 객체를 생성
model.to(device)  ------ model을 장치(CPU 혹은 GPU)에 할당
model.eval()  ------ 테스트 데이터에 대한 모델 평가 용도로 사용
```

특성 맵은 합성곱층을 입력 이미지와 필터를 연산하여 얻은 결과
합성곱층에서 입력과 출력을 알 수 있다면 특성 맵에 대한 값들을 확인할 수 있다는 의미

5.4 설명 가능한 CNN

- **특성 맵 시각화**
  - ① cv2.resize는 이미지 크기를 변경할 때 사용하며, 파라미터는 다음과 같음

```
cv2.resize(img, (100,100), interpolation=cv2.INTER_LINEAR)
           ⓐ      ⓑ                    ⓒ
```

ⓐ 첫 번째 파라미터: 변경할 이미지 파일

ⓑ 두 번째 파라미터: 변경될 이미지 크기를 (너비, 높이)로 지정

ⓒ interpolation: 보간법