5️⃣

합성곱 신경망 II - Part 1

날짜	@2024년 10월 30일 → 2024년 11월 4일
범위	6.1.1~6.1.2장
상태	완료
주차	7주차

6장 합성곱 신경망 II

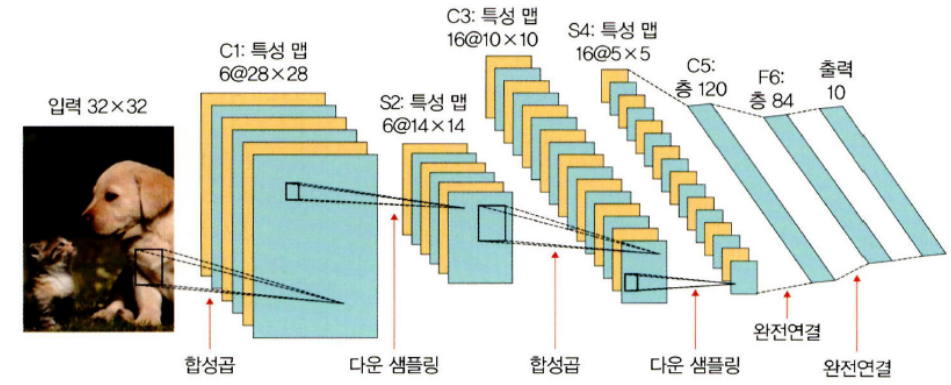
6.1 이미지 분류를 위한 신경망

- 입력 데이터로 이미지를 사용한 분류(classification)
  - 특정 대상이 영상 내에 존재하는지 여부를 판단하는 것.
- 이미지 분류(image classification)에서 주로 사용되는 합성곱 신경망의 유형 알아보기

6.1.1 LeNet-5

- **LeNet-5** : 합성곱 신경망이라는 개념을 최초로 얀 르쿤(Yann LeCun)이 개발한 구조.
  - 1995년 얀 르쿤, 레옹 보토(Leon Bottu), 요슈아 벤지오(Yosua Bengio), 패트릭 하프너(Patrick Haffner)가 수표에 쓴 손글씨 숫자를 인식하는 딥러닝 구조 LeNet-5 발표 → 그것이 현재 CNN의 초석.
  - 합성곱(convolutional)과 다운 샘플링(sub-sampling)(혹은 풀링)을 반복적으로 거치면서 마지막에 완전연결층에서 분류 수행함.

❤ 그림 6-1 LeNet-5

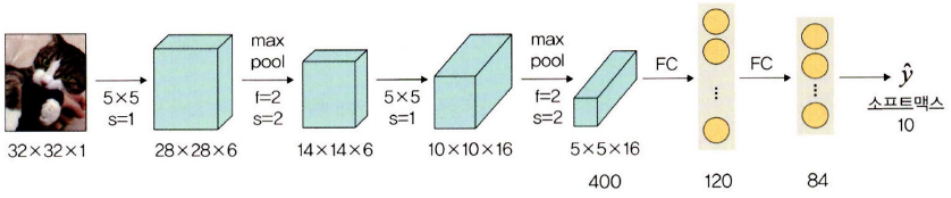


- C1 : 5×5 합성곱 연산 후 28x28 크기의 특성 맵 여섯 개를 생성함.
- S2 : 다운 샘플링하여 특성 맵 크기를 14x14로 줄임.
- C3 : 5×5 합성곱 연산하여 10x10 크기의 특성맵 16개 생성함.
- C4 : 다운 샘플링 하여 특성 맵 크기를 5x5로 줄임.
- C5 : 5×5 합성곱 연산하여 1x1 크기의 특성 맵 120개 생성함.
- F6 : 완전연결층으로 C5 결과를 유닛(unit) 64개에 연결시킴.

⇒ C로 시작하는 것은 합성곱층, S로 시작하는 것은 풀링층, F로 시작하는 것은 완전연결층 의미.

- LeNet-5 사용하는 예제 구현하기
  - 개와 고양이 데이터셋 다시 사용.

❤ 그림 6-2 LeNet-5 예제 신경망



구현할 신경망

  - 32×32 크기의 이미지에 합성곱층과 최대 풀링층이 쌍으로 두 번 적용된 후 완전연결층을 거쳐 이미지가 분류되는 신경망.

❮ 표 6-1 LeNet-5 예제 신경망 상세

계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
이미지	1	32×32	–	–	–
합성곱층	6	28×28	5×5	1	렐루(ReLU)
최대 풀링층	6	14×14	2×2	2	–
합성곱층	16	10×10	5×5	1	렐루(ReLU)
최대 풀링층	16	5×5	2×2	2	–
완전연결층	–	120	–	–	렐루(ReLU)
완전연결층	–	84	–	–	렐루(ReLU)
완전연결층	–	2	–	–	소프트맥스(softmax)

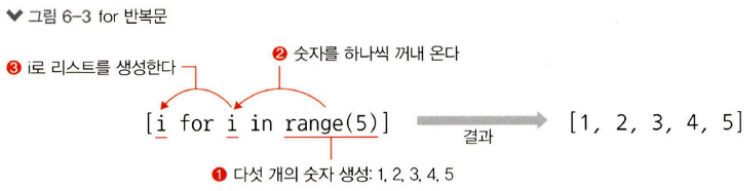
- 필요한 라이브러리 호출
  - '`tqdm`' : 아랍어로 progress(진행 상태). 즉, 진행 상태를 바(bar) 형태로 가시화하여 보여 줌. 주로 모델 훈련에 대한 진행 상태를 확인하고자 할 때 사용함.
- 이미지 데이터셋 전처리(텐서 변환)
  1. 토치비전(torchvision) 라이브러리 이용하면 이미지에 대한 전처리 손쉽게 할 수 있음.
     a. `transforms.Compose` : 이미지를 변형할 수 있는 방식들의 묶음.
     b. `transforms.RandomResizedCrop` : 입력 이미지를 주어진 크기(resize : 224×224) 로 조정함. 또한, scale은 원래 이미지를 임의의 크기(0.5~1.0)만큼 면적을 무작위로 자르겠다는 의미.
     c. `transforms.RandomHorizontalFlip` : 주어진 확률로 이미지를 수평 반전시킴. 이때 확률 값을 지정하지 않았으므로 기본값인 0.5 확률로 이미지들이 수평 반전됨. 즉, 훈련 이미지 중 반은 위아래가 뒤집힌 상태로 두고, 반은 그대로 사용함.
     d. `transforms.ToTensor` : ImageFolder 메서드를 비롯해서 torchvision 메서드는 이미지를 읽을 때 파이썬 이미지 라이브러리 PIL 사용함. PIL 사용해서 이미지 읽으면 생성되는 이미지 범위가 [0, 225]이며, 배열의 차원(높이 H x 너비 W x 채널 수 C)로 표현됨. 이후 효율적인 연산을 위해 torch.FloatTensor 배열로 바꾸어야 하는데, 이때 픽셀 값의 범위는 [0.0, 1.0] 사이가 되고 차원 순서도 (채널 수 C x 높이 H x 너비 W)로 바뀜. 이러한 작업 수행해 주는 메서드.
     e. `transforms.Normalize` : 전이 학습에서 사용하는 사전 훈련된 모델들은 대개 ImageNet 데이터셋에서 훈련됨. 따라서 사전 훈련된 모델을 사용하기 위해서는 ImageNet 데이터의 각 채널별 평균과 표준편차에 맞는 정규화(normalize) 해 주어야 함. 즉, Normalize 메서드 안에 사용된 mean,

std는 ImageNet에서 이미지들의 RGB 채널마다 평균과 표준편차를 의미함.

2. `__call__` 함수 : 클래스를 호출할 수 있도록 하는 메서드. `__init__` 은 인스턴스 초기화를 위해 사용한다면, `__call__` 은 인스턴스가 호출되었을 때 실행됨. 즉, 클래스 객체 자체를 호출하면 `__call__` 함수의 리턴(return) 값이 반환됨.

- 이미지 데이터셋을 불러온 후 훈련, 검증, 테스트로 분리

1. 고양이 이미지 데이터 가져오기
   a. `sorted` : 데이터 정렬된 리스트로 만들어서 반환함.
   b. `os.path.join` : 경로와 파일명을 결합하거나 분할된 경로를 하나로 합치고 싶을 때 사용함. 즉, cat_directory 디렉터리와 `os.listdir`을 통해 검색된 이미지 파일들(f)을 하나로 합쳐서 표시해 줌.
   c. `os.listdir` : 지정한 디렉터리 내 모든 파일의 리스트 반환함. 예제에서 사용하는 Cat 디렉터리의 이미지 파일 모두 반환함.
2. `images_filepaths`에서 이미지 파일들을 불러옴.
   a. for 반복문 이용하여 가져온 데이터에 대해 i 이용하여 리스트로 만듦. 즉, 그림과 같은 의미.



   b. 반복문 이용하여 `image_filepaths`에서 이미지 데이터 검색함.
   c. 조건문(if) : `cv2.imread()` 함수를 이용하여 모든 이미지 데이터를 읽어 옴. (not None 상태, 즉 '더 이상 데이터를 찾을 수 없을 때까지' 의미함.)
3. 넘파이 `random()` 함수 : 임의의 난수 생성. 이때 난수를 생성하기 위해 사용되는 것이 시드 값(seed value). 또한, `Numpy.random.seed()` 메서드는 상태를 초기화함. 즉, 이 모듈이 호출될 때마다 임의의 난수 재생성됨. 하지만 특정 시드 값을 부여하면 상태가 저장되기 때문에 동일한 난수를 생성함.

- 테스트 데이터셋 이미지 확인 함수

1. `cv2.cvtColor` : 이미지 색상 변경하기 위해 사용함.
   a. 첫 번째 파라미터 : 입력 이미지

b. 두 번째 파라미터 : 변환할 이미지의 색상 지정함. BRG(Blue, Green, Red) 채널 이미지를 RGB(컬러)로 변경하겠다는 의미.

2. 이미지 전체 경로를 정규화하고 분할을 위한 코드.
   a. `os.path.normpath` : 경로명을 정규화. 예를 들어 A//B, A/B/, A/./B 및 A//../B 모두 A/B로 경로 통일함.
   b. `split(os.sep)` : 경로를 / 혹은 \를 기준으로 분할할 때 사용함. 예를 들어, c:/temp/user/a.jpg라는 경로가 있을 때 split(os.sep)를 적용하면 ['c:', 'temp', 'user', 'a.jpg'] 처럼 분할됨. 또한, `split(os.sep)[-2]` 적용하면 'user' 반환할 것.
3. `predicted_label`에 대한 값 정의하기 위한 코드.
   a. predicted_labels 값이 있으면 그 값을 predicted_label로 사용함.
   b. predicted_labels에 값이 없다면 true_label 값을 predicted_label로 사용함.

- 테스트 데이터셋 이미지를 출력

  ⇒ 기본적인 데이터셋 준비 완료.

  ⇒ 데이터셋에는 학습할 데이터 경로 정의 → 그 경로에서 데이터를 읽어 옴. → 데이터셋 크기가 클 수 있으므로 `__init__` 에서 전체 데이터 읽어오는 것이 아니라 경로만 저장해 놓고, `__getitem__` 메서드에서 이미지 읽어 옴.(데이터 어디에서 가져올지 결정.) → 데이터로더에서 데이터셋 데이터를 메모리로 불러오는데, 배치 크기만큼 분할하여 가져옴.

- 이미지 데이터셋 클래스 정의
  - 예제 목적 : 다수의 개와 고양이 이미지가 포함된 데이터에서 이들을 예측하는 것.
  - 레이블(정답) 이미지에서 고양이와 개가 포함될 확률을 코드로 구현. 예를 들어 고양이가 있는 이미지의 레이블은 0, 개가 있는 이미지의 레이블은 1이 되도록 코드 구현함.

  1. 이미지 데이터에 대한 레이블 값(dog, cat) 가져옴.
     a. `img_path` : 이미지가 위치한 전체 경로 보여 줌.
     b. `img_path.split('/')[-1]` : 이미지 전체 경로에서 '/' 제거함.
     c. `split('.')[0]` : 마지막으로 `img_path.split('/')[-1]` 통해 얻은 결과인 'dog.113.jpg'에서 '.' 제거함. 이때 'dog.113.jpg'를 '.' 기준으로 분리, 분리된 값들에서 첫 번째 값을 가져오면 결과는 'dog'이 됨.

- 변수 값 정의
  - 전처리에서 사용할 변수에 대한 값을 정의함.
- 이미지 데이터셋 정의
  - 훈련과 검증 용도의 데이터셋 정의함.
  - 앞서 정의한 DogvsCatDataset() 클래스를 이용하여 훈련과 검증 데이터셋을 준비하되 전처리도 함께 적용함.
- 데이터로더 정의
  - 메모리로 불러와서 훈련을 위한 준비.

1. 파이토치의 데이터로더 : 배치 관리 담당. 한 번에 모든 데이터를 불러오면 메모리에 부담을 줄 수 있기 때문에 데이터를 그룹으로 쪼개서 조금씩 불러옴.
   a. 첫 번째 파라미터 : 데이터 불러오기 위한 데이터셋.
   b. batch_size : 한 번에 메모리로 불러올 데이터 크기, 여기에서는 32개씩 데이터 가져옴.
   c. shuffle : 메모리로 데이터를 가져올 때 임의로 섞어서 가져오도록 함.

- 모델의 네트워크 클래스
  - 데이터셋을 학습시킬 모델의 네트워크를 설계하기 위한 클래스 생성
  - Conv2d 계층에서 출력 크기 구하는 공식

$$\text{출력 크기} = (W - F + 2P)/S + 1$$

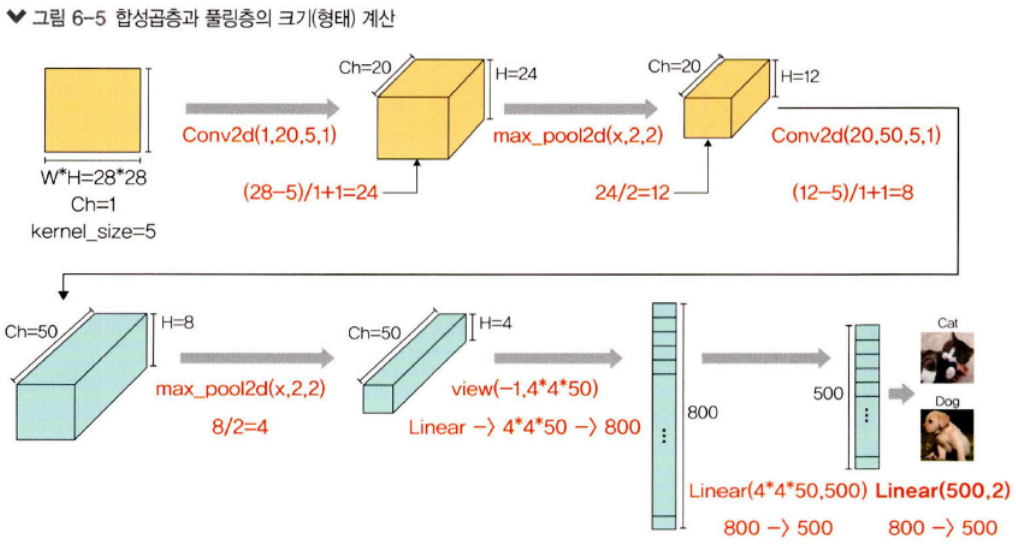
- W: 입력 데이터의 크기(input_volume_size)
- F: 커널 크기(kernel_size)
- P: 패딩 크기(padding_size)
- S: 스트라이드(strides)

  - MaxPool2d 계층에서 출력 크기 구하는 공식

$$\text{출력 크기} = IF/F$$

- IF: 입력 필터의 크기(input_filter_size, 또한 바로 앞의 Conv2d의 출력 크기이기도 합니다)
- F: 커널 크기(kernel_size)

  - 각각의 계층에 대한 결과



- 모델 객체 생성
  - LeNet()을 model이라는 이름으로 객체 생성하여 모델 학습 위한 준비.
- torchsummary 라이브러리 이용한 모델 네트워크 구조 확인
  - 출력 결과가 한눈에 들어오지 않는다면 torchsummary 라이브러리 사용.
  - keras와 같은 형태로 모델 출력해 볼 수 있는 라이브러리.

1. summary 통해 모델의 네트워크 관련 정보 확인할 수 있음.
   a. 첫 번째 파라미터 : 모델의 네크워크
   b. 두 번째 파라미터 : 입력 값으로 (채널(channel), 너비(width), 높이(height))가 됨. 여기서 (3, 224, 224)를 입력으로 지정함.

- 앞선 모델 출력 결과와 비교하면 네트워크 내 파라미터 수와 구조가 이해하기 쉽게 표현되어 있는 것을 확인할 수 있음.
- 출력되는 정보 : 총 파라미터 수, 입력 크기, 네트워크 총 크기 등.

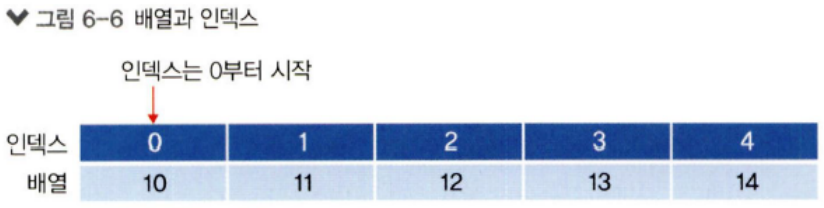
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 16, 220, 220]           1,216
              ReLU-2         [-1, 16, 220, 220]               0
         MaxPool2d-3         [-1, 16, 110, 110]               0
            Conv2d-4         [-1, 32, 106, 106]          12,832
              ReLU-5         [-1, 32, 106, 106]               0
         MaxPool2d-6           [-1, 32, 53, 53]               0
            Linear-7                 [-1, 512]      46,023,168
            Linear-8                   [-1, 2]           1,026
           Softmax-9                   [-1, 2]               0
================================================================
Total params: 46,038,242
Trainable params: 46,038,242
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 19.47
Params size (MB): 175.62
Estimated Total Size (MB): 195.67
----------------------------------------------------------------
```

- 학습 가능한 파라미터 수 확인
  - 모델의 학습 가능한 파라미터 수를 model.parameters() 이용하여 확인하기
  - 46,038,242개의 파라미터 학습해야 함을 확인할 수 있음.
- 옵티마이저와 손실 함수 정의
  1. 경사 하강법으로 모멘텀 SGD 사용하도록 함. 모멘텀 SGD : SGD에 관성이 추가된 것으로 매번 기울기를 구하지만 가중치를 수정하기 전에 이전 수정 방향 (+, -) 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법.
     a. 첫 번째 파라미터 : 경사 하강법을 통해 궁극적으로 업데이트하고자 하는 파라미터는 가중치(weight)와 바이어스(bias). 모델에 대한 파라미터는 총 46,038,242개.
     b. lr(learning rate) : 가중치를 변경할 때 얼마나 크게 변경할지 결정함.
     c. momentum : SGD를 적절한 방향으로 가속화, 흔들림(진동)을 줄여 주는 매개변수.
- 모델의 파라미터와 손실 함수 CPU에 할당
  - 파이토치는 GPU 할당이 자동화되어 있지 않기 때문에 모델의 모든 파라미터 및 손실 함수가 GPU 사용할 수 있도록 지정해 주어야 함.
- 모델 학습 함수 정의
  1. 오차와 입력을 곱하는 이유를 알아보기 위해 손실 함수의 reduction이라는 파라미터 이해할 필요 있음.

`reduction` 파라미터의 기본값은 'mean'. 'mean'은 정답과 예측 값의 오차 구한 후 그 값들의 평균을 반환함. 즉, 손실 함수 특성상 전체 오차를 배치 크기로 나눔으로써 평균을 반환하기 때문에 epoch_loss를 계산하는 동안 `loss.item()`과 `inputs.size(0)` 곱해줌.

- 모델 학습
  - 모델 학습을 위해 `train_model` 호출
  - 검증 데이터셋을 이용한 모델 학습 결과 최고 68%의 정확도 보이고 있음. 조금 더 정확한 결과를 원한다면 데이터셋을 늘려서 테스트하기.
- 모델 테스트를 위한 함수 정의
  - 훈련 데이터셋과 더불어 테스트 데이터셋을 모델에 적용하여 정확도 측정.
  - 측정 결과는 데이터 프레임에 담아 둔 후 csv 파일로 저장함.
  - 테스트 용도의 데이터셋을 이용하므로 model.eval() 사용함.

1. `torch.unsqueeze()` : 텐서에 차원을 추가할 때 사용함. 또한, (0)은 차원이 추가될 위치를 의미함.
   예를 들어 형태가 (3)인 텐서가 있다고 가정해보자. 0 위치에 차원을 추가하면 형태가 (1,3)이 됨. 즉, 행 한 개와 열 세 개의 구조를 갖는 텐서가 만들어짐.
   a. 형태가 (2,2)인 2D 텐서가 있을 때 0 위치에 차원을 추가하면 텐서 모양이 (1,2,2)가 됨. 이는 하나의 채널, 행 두 개와 열 두 개 의미함.
   b. 1 위치에 차원을 추가하면 (2,1,2) 형태가 되므로 채널 두 개, 행 한 개, 열 두 개가 됨.
   c. 2 위치에 차원을 추가하면 (2,2,1) 형태가 되므로 채널 두 개, 행 두 개, 열 한 개를 의미함.
2. 소프트맥스(softmax) : 지정된 차원(dim)을 따라 텐서의 요소(텐서의 개별 값)가 (0, 1) 범위에 있고 합계가 1이 되도록 크기를 다시 조정함.
   a. `F.softmax(outputs, dim=1)` : outputs에 softmax 적용하여 각 행의 합이 1이 되도록 함.
   b. `[:, 1]` : a 값 중 모든 행(:)에서 두 번째 칼럼(1번째 인덱스)을 가져옴.



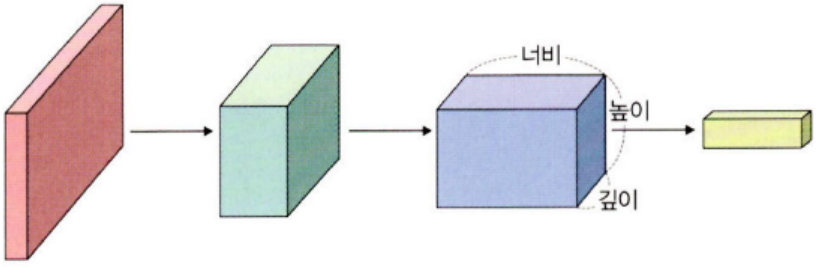
c. `.tolist()` : 배열을 리스트 형태로 변환함.

- 테스트 데이터셋의 예측 결과 호출
  - 테스트 데이터셋을 모델에 적용한 결과는 LeNet 파일로 저장해 둠.
  - 예측 결과 레이블이 0.5보다 크면 개를 의미, 0.5보다 작으면 고양이를 의미함.
- 테스트 데이터셋 이미지를 출력하기 위한 함수 정의
  - 예측 결과를 시각적으로 표현하기 위한 함수 정의함.
- 테스트 데이터셋 예측 결과 이미지 출력
  - 예측력이 크게 좋지는 않음.
  - 극히 일부의 데이터를 이용한 모델 학습을 진행했기 때문에 이와 같은 예측력 보임.

6.1.2 AlexNet

- AlexNet : ImageNet 영상 데이터베이스를 기반으로 한 화상 인식 대회 ' ILSVRC 2012'에서 우승한 CNN 구조
- AlexNet 구조
  - CNN 구조 : 3차원 구조(이미지를 다루기 때문에 기본적으로 3차원 데이터를 다룸.)
    - 이미지 크기를 나타내는 너비(width), 높이(height), 깊이(depth)
    - 보통 색상이 많은 이미지는 R/G/B 성분 3개를 갖기 때문에 시작이 3, 합성곱을 거치면서 특성 맵이 만들어지고 이것에 따라 중간 영상 깊이가 달라짐.

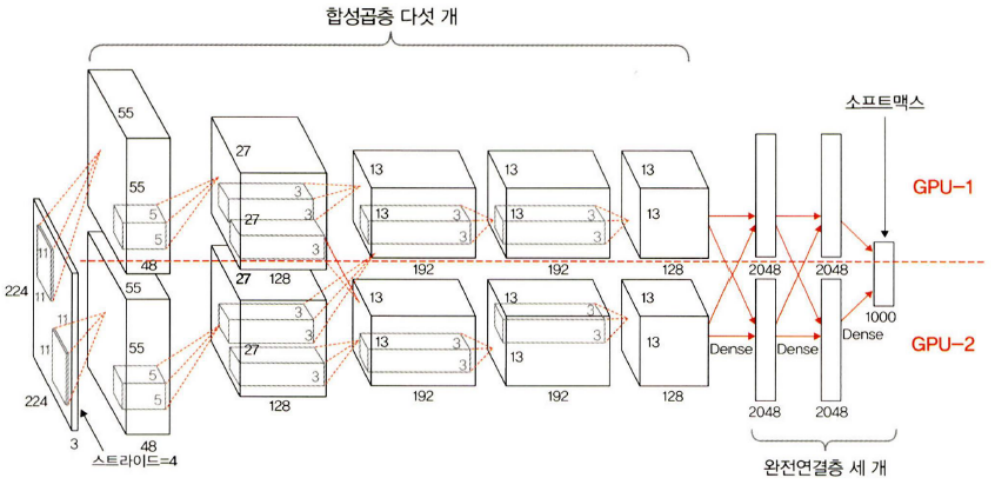
그림 6-8 CNN 구조



  - AlexNet : 합성곱층 총 5개와 완전연결층 3개로 구성됨. 맨 마지막 완전연결층은 카테고리 1000개를 분류하기 위해 소프트맥스 활성화 함수 사용하고 있음.

- 전체적으로 보면 GPU 2개를 기반으로 한 병렬 구조인 점을 제외하면 LeNet-5와 크게 다르지 않음.

❯ 그림 6-9 AlexNet 구조



- 합성곱층에서 사용된 활성화 함수는 렐루(ReLU), 각 계층의 구조적 세부 사항은 표를 참고.

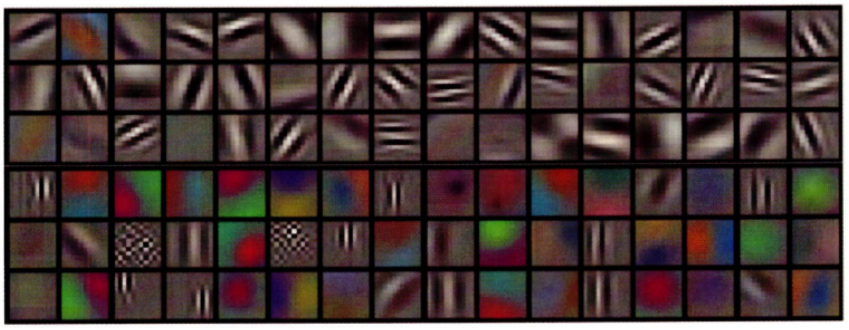
❯ 표 6-2 AlexNet 구조 상세

계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
이미지	1	227×227	–	–	–
합성곱층	96	55×55	11×11	4	렐루(ReLU)
최대 풀링층	96	27×27	3×3	2	–
합성곱층	256	27×27	5×5	1	렐루(ReLU)
최대 풀링층	256	13×13	3×3	2	–
합성곱층	384	13×13	3×3	1	렐루(ReLU)
합성곱층	384	13×13	3×3	1	렐루(ReLU)
합성곱층	256	13×13	3×3	1	렐루(ReLU)
최대 풀링층	256	6×6	3×3	2	–
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	1000	–	–	소프트맥스(softmax)

- 네트워크에는 학습 가능한 변수가 총 6600만 개. 네트워크에 대한 입력은 227x227×3 크기의 RGB 이미지. 각 클래스(혹은 카테고리)에 해당하는 1000x1 확률 벡터를 출력함.
- 첫 번째 합성곱층 커널의 크기는 11x11×3, 스트라이드를 4로 적용하여 특성 맵 96개 생성하기 때문에 55x55×96의 출력을 가짐. 첫 번째 계층을 거치면서

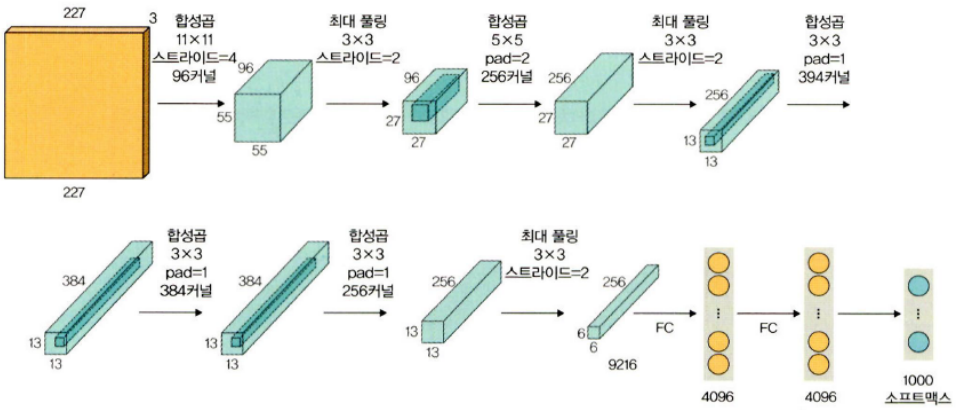
GPU-1에서는 주로 컬러와 상관없는 정보를 추출하기 위한 커널이 학습됨.
GPU-2에서는 주로 컬러와 관련된 정보를 추출하기 위한 커널이 학습됨.

❯ 그림 6-10 AlexNet GPU-1 · 2 적용 결과



- 파이토치 코드로 AlexNet 알아보기
  - 필요한 라이브러리 호출
    - 데이터셋은 앞서 진행했던 개와 고양이 이미지 사용.
    - 준비된 데이터를 이용할 네트워크 생성.
    - 원래 AlexNet의 마지막 계층에는 뉴런이 1000개 있지만 예제에서는 클래스 두 개만 사용함.

❯ 그림 6-11 AlexNet 예제 네트워크



  - 데이터 전처리
    - 앞선 코드와 동일.
  - 데이터를 가져와서 훈련, 검증, 테스트 용도로 분리
    - 이미지 위치한 경로에서 데이터 가져와 훈련, 검증, 테스트 용도로 분리.
    - 데이터가 위치한 경로에는 Cat, Dog 폴더로 구성되어 있음.

        - 우리가 사용하는 노트북(혹은 서버, 코랩)의 성능(CPU/GPU, 메모리)이 좋지 않다고 가정한 채 진행 → 훈련 데이터셋을 400개로 제한.
    - 커스텀 데이터셋 정의
        - AlexNet : 파라미터 6000만 개 사용하는 모델.
        - 이때 충분한 데이터가 없으면 과적합이 발생하는 등 테스트 데이터에 대한 성능이 좋지 않음.
        - 성능이 좋은 결과를 원한다면 충분한 데이터셋 확보하고 테스트 진행하면 됨.
        - `torch.utils.data.Dataset` 상속받아 커스텀 데이터셋(custom dataset) 정의함.
        - `torch.utils.data.Dataset` 클래스를 상속받아 커스텀 데이터셋 만들기
    - 변수에 대한 값 정의
        - 전처리에 필요한 평균, 표준편차 등에 대한 변수 값 정의함.
    - 훈련, 검증, 테스트 데이터셋 정의
        - 훈련 데이터셋의 크기 및 레이블에 대한 출력 결과 : 훈련 데이터셋 크기 (3, 256, 256) ⇒ (채널, 너비, 높이)
    - 데이터셋을 메모리로 불러옴
        - 데이터셋을 데이터로더로 전달하여 메모리로 불러올 준비.
    - AlexNet 모델 네트워크 정의
        - AlexNet 모델 사용하기 위한 네트워크 : 사전 훈련된 네트워크와 유사하게 정의.
            - 합성곱(Conv2d) + 활성화 함수(ReLU) + 풀링(MaxPool2d) 다섯 번 반복된 후 두 개의 완전연결층과 출력층으로 구성됨.

        1. ReLU 활성화 함수에서 inplace 의미 : 연산에 대한 결괏값을 새로운 변수에 저장하는 것이 아닌 기존 데이터를 대체하는 것 의미. → 기존 값을 연산 결괏값으로 대체함으로써 기존 값들을 무시하겠다는 의미.
        2. `nn.AdaptiveAvgPool2d` : `nn.AvgPool2d`처럼 풀링을 위해 사용함.
            a. AvgPool2d : 풀링에 대한 커널 및 스트라이드 크기를 정의해야 동작함. 커널 크기, 스트라이드, 패딩을 지정해야 함. 예) `nn.AvgPool2d((3,2), stride=(2,1))` ⇒ 결과는 5x5 텐서 → 3x3 텐서, 7x7 텐서 → 4x4 텐서로 줄이는 효과.

$$H_{out} = \left[\frac{H_{in} + 2 \times \text{padding}[0] - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right]$$
$$W_{out} = \left[\frac{W_{in} + 2 \times \text{padding}[1] - \text{kernel\_size}[1]}{\text{stride}[1]} + 1 \right]$$

b. AdaptiveAvgPool2d : 풀링 작업이 끝날 때 필요한 출력 크기를 정의함. 출력에 대한 크기만 지정함. 출력 크기에 대한 조정이 상당히 쉬워짐. ⇒ 입력 크기에 변동이 있고 CNN 위쪽에 완전연결층을 사용하는 경우 유용함.

- model 객체 생성
  - AlexNet 모델에 대한 네트워크 구조 보여 줌.
- 옵티마이저 및 손실 함수 정의
  - 학습에서 사용될 옵티마이저와 손실 함수 정의.
- 모델 네트워크 구조 확인
  - torchsummary 이용하여 모델 네트워크 살펴보기
  - 모델의 네트워크를 파라미터와 함께 보여 주고 있기 때문에 내부적으로 어떤 일들이 이루어지는지 예측하면서 살펴볼 수 있음.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 63, 63]          23,296
              ReLU-2           [-1, 64, 63, 63]               0
         MaxPool2d-3           [-1, 64, 31, 31]               0
            Conv2d-4          [-1, 192, 31, 31]         307,392
              ReLU-5          [-1, 192, 31, 31]               0
         MaxPool2d-6          [-1, 192, 15, 15]               0
            Conv2d-7          [-1, 384, 15, 15]         663,936
              ReLU-8          [-1, 384, 15, 15]               0
            Conv2d-9          [-1, 256, 15, 15]         884,992
             ReLU-10          [-1, 256, 15, 15]               0
           Conv2d-11          [-1, 256, 15, 15]         590,080
             ReLU-12          [-1, 256, 15, 15]               0
        MaxPool2d-13            [-1, 256, 7, 7]               0
AdaptiveAvgPool2d-14            [-1, 256, 6, 6]               0
          Dropout-15                 [-1, 9216]               0
           Linear-16                 [-1, 4096]      37,752,832
             ReLU-17                 [-1, 4096]               0
          Dropout-18                 [-1, 4096]               0
           Linear-19                  [-1, 512]       2,097,664
             ReLU-20                  [-1, 512]               0
           Linear-21                    [-1, 2]           1,026
================================================================
Total params: 42,321,218
Trainable params: 42,321,218
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.75
Forward/backward pass size (MB): 10.90
Params size (MB): 161.44
Estimated Total Size (MB): 173.10
----------------------------------------------------------------
```

- 모델 학습 함수 정의
  - LeNet 사용했던 코드와 동일.
- 모델 학습
  - 훈련 위해 정의된 train_model() 함수를 호출하여 모델 학습.
  - 에포크는 10.
- 모델을 이용한 예측
  - 모델 예측에 대한 실행 결과 → 예측 결과를 CSV 파일로 저장하는 함수이기 때문에 결과는 큰 의미 없음. 단순 처리 완료된 결과 보여 줌.
- 데이터 프레임 결과 확인
  - label 값이 0.5보다 크면 개, 0.5보다 작으면 고양이로 예측했음을 의미.
- 예측 결과를 시각적으로 표현하기 위한 함수 정의
- 예측 결과에 대해 이미지와 함께 출력

❯ 그림 6-12 테스트 데이터셋에 대한 AlexNet 모델의 예측 결과

- 예측 결과가 좋지 않음.
- 데이터셋이 많으면 성능이 좋아질 수 있음. 예제들은 실습 PC(노트북) 성능을 고려하여 데이터셋 제한시킴.
- 성능은 파라미터 튜닝을 통해서도 향상시킬 수 있음.