6

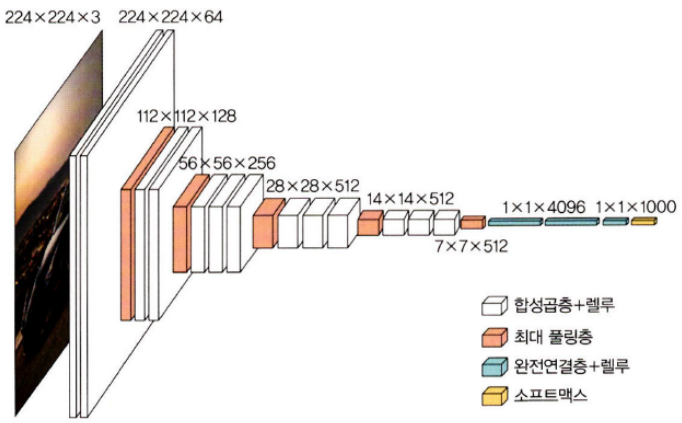
합성곱 신경망 II - Part 2

날짜	@2024년 11월 6일 → 2024년 11월 11일
범위	6.1.3~6.1.4장
상태	완료
주차	8주차

6.1.3 VGGNet

- VGGNet
  - 카렌 시모니안(Karen Simonyan)과 앤드류 지서만(Andrew Zisserman)이 2015 ICLR에 게재한 “Very deep convolutional networks for large-scale image recognition” 논문에서 처음 발표.
  - 합성곱층의 파라미터 수를 줄이고 훈련 시간을 개선하려고 탄생함. 즉, **네트워크를 깊게 만드는 것이 성능에 어떤 영향을 미치는지 확인**하고자 나온 것 ⇒ VGG
  - VGG 연구 팀은 **깊이의 영향만 최대한 확인**하고자 합성곱층에서 사용하는 필터/커널의 크기를 **가장 작은 3X3**으로 고정함.
  - 네트워크 계층의 총 개수에 따라 여러 유형의 VGGNet이 있음. (VGG16, VGG19 등)
- VGG16

❯ 그림 6-13 VGG16 구조



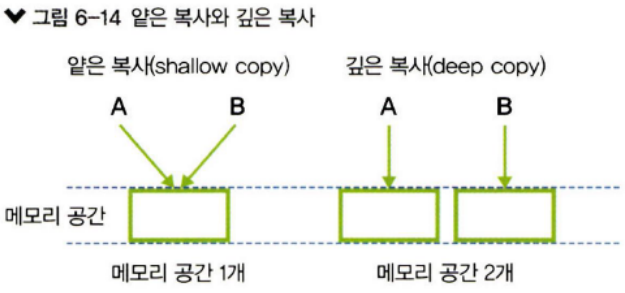
- 파라미터가 총 1억 3300만 개. 주목할 점은 모든 합성곱 커널의 크기가 3X3, 최대 풀링 커널의 크기는 2X2, 스트라이드는 2라는 것.
- 결과적으로 64개의 224X224 특성 맵(224X224X64)들이 생성됨.
- 마지막 16번째 계층을 제외하고는 모두 ReLU 활성화 함수가 적용됨.

❯ 표 6-3 VGG16 구조 상세

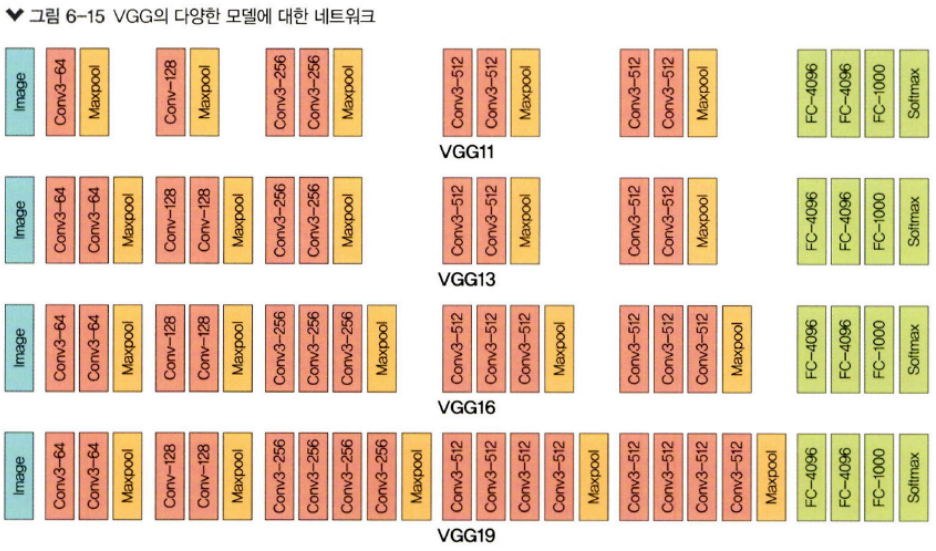
계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
이미지	1	224×224	–	–	–
합성곱층	64	224×224	3×3	1	렐루(ReLU)
합성곱층	64	224×224	3×3	1	렐루(ReLU)
최대 풀링층	64	112×112	2×2	2	–
합성곱층	128	112×112	3×3	1	렐루(ReLU)
합성곱층	128	112×112	3×3	1	렐루(ReLU)
최대 풀링층	128	56×56	2×2	2	–
합성곱층	256	56×56	3×3	1	렐루(ReLU)
합성곱층	256	56×56	3×3	1	렐루(ReLU)
합성곱층	256	56×56	3×3	1	렐루(ReLU)
합성곱층	256	56×56	3×3	1	렐루(ReLU)
최대 풀링층	256	28×28	2×2	2	–
합성곱층	512	28×28	3×3	1	렐루(ReLU)
합성곱층	512	28×28	3×3	1	렐루(ReLU)
합성곱층	512	28×28	3×3	1	렐루(ReLU)
합성곱층	512	28×28	3×3	1	렐루(ReLU)
최대 풀링층	512	14×14	2×2	2	–

계층 유형	특성 맵	크기	커널 크기	스트라이드	활성화 함수
합성곱층	512	14×14	3×3	1	렐루(ReLU)
합성곱층	512	14×14	3×3	1	렐루(ReLU)
합성곱층	512	14×14	3×3	1	렐루(ReLU)
합성곱층	512	14×14	3×3	1	렐루(ReLU)
최대 풀링층	512	7×7	2×2	2	–
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	4096	–	–	렐루(ReLU)
완전연결층	–	1000	–	–	소프트맥스(softmax)

- 예제 : VGGNet 중에서 가장 간단한 VGG11을 파이토치로 구현하기.
  - 필요한 라이브러리 호출
    1. `import copy` : 객체 복사를 위해 사용함. 객체 복사는 크게 얕은 복사(shallow copy)와 깊은 복사(deep copy)로 나뉨.
       - 단순한 객체 복사
       - 얕은 복사
       - 깊은 복사



  - VGG 모델 정의
  - 모델 유형 정의
    - VGG11, VGG13, VGG16, VGG19 모델의 계층 정리
    - 숫자(output channel, 출력 채널)는 Conv2d를 수행하라는 의미, 출력 채널(output channel)이 다음 계층의 입력 채널(input channel)이 됨.
    - M = 최대 풀링을 수행하라는 의미.



- VGG 계층 정의
  - VGG 모델에서 네트워크 정의. `vgg11_config` 사용함.

  1. 조건문 정의

     `assert c == 'M'` or `isinstance(c, int)`

     - 가정 설정문이라고 불리는 `assert` : 뒤의 조건이 True가 아니면 에러 발생시킴. 따라서 c = 'M'이 아니면 오류가 발생함.
     - `isinstance` : 주어진 조건이 True인지 판단함.

     ⇒ 위 코드의 의미는 c가 'M'이 아니거나 int 아니라면 오류가 발생함.

- 모델 계층 생성

  1. `batch_norm`(Batch Normalization) : 데이터 평균을 0, 표준편차를 1로 분포시키는 것. 각 계층에서 입력 데이터의 분포는 앞 계층에서 업데이트된 가중치에 따라 변함. 즉, 각 계층마다 변화되는 분포는 학습 속도를 늦출 뿐만 아니라 학습도 어렵게 함. 따라서 각 계층의 입력에 대한 분산을 평균 0, 표준편차 1로 분포시키는 것이 배치 정규화.

- VGG11 계층 확인
  - 배치 정규화(BatchNorm2d) 추가된 것을 확인.

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU(inplace=True)
  (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU(inplace=True)
  (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (13): ReLU(inplace=True)
  (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (17): ReLU(inplace=True)
  (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (20): ReLU(inplace=True)
  (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (24): ReLU(inplace=True)
  (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (27): ReLU(inplace=True)
  (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

- VGG11 전체에 대한 네트워크
  - 앞에서 정의했던 vgg11_layers, 완전연결층과 출력층(`(classifier): Sequential()`) 부분 합친 것을 확인.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace=True)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): ReLU(inplace=True)
    (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=7)
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

- VGG11 사전 훈련된 모델 사용
  - VGG 모델은 사전 훈련된 모델. 이미 누군가가 대용량의 이미지 데이터로 학습시키고, 최상의 상태로 튜닝을 거쳐 모든 사람이 사용할 수 있도록 공유함.

1. 배치 정규화가 적용된 사전 훈련된 VGG11 모델을 사용하기 위해서는 다음과 같은 파라미터 사용.
   a. `models.vgg11_bn` : `vgg11_bn`은 VGG11 기본 모델에 배치 정규화가 적용된 모델.
   b. `(pretrained=True)` : 사전 훈련된 모델을 사용(미리 학습된 파라미터 값들을 사용)하겠다는 의미.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (20): ReLU(inplace=True)
    (21): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (27): ReLU(inplace=True)
    (28): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

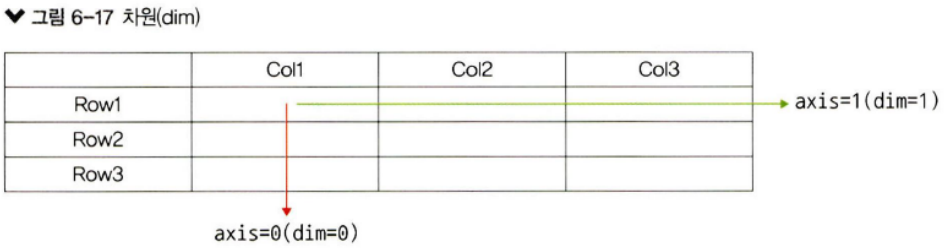
- 이미지 데이터 전처리
  1. transforms.Compose
     a. `transforms.Resize((256, 256))` : 이미지를 주어진 크기로 재조정.
     b. `transforms.RandomRotation(5)` : 5도 이하로 이미지를 회전시킴.
- ImageFolder를 이용하여 데이터셋 불러오기
  1. ImageFolder에서 사용하는 파라미터는 5장 참고. ImageFolder를 언제 사용하면 좋을까?

     ImageFolder : 계층적인 폴더 구조를 가지고 있는 데이터셋을 불러올 때 사용함.
- 훈련과 검증 데이터 분할

1. 파이썬의 `random_split()` : 훈련과 검증 데이터셋을 나누는 용도로 사용, 다음과 같은 파라미터 사용함. 데이터가 데이터로더로 넘어간 이후에는 분리가 불가능하므로 데이터셋(dataset) 단계에서 진행해야 함.
    a. `train_dataset` : 분할에 사용될 데이터셋
    b. `[n_train_examples, n_valid_examples]` : 훈련과 검증 데이터셋의 크기 지정. [훈련 데이터셋 크기, 검증 데이터셋 크기]

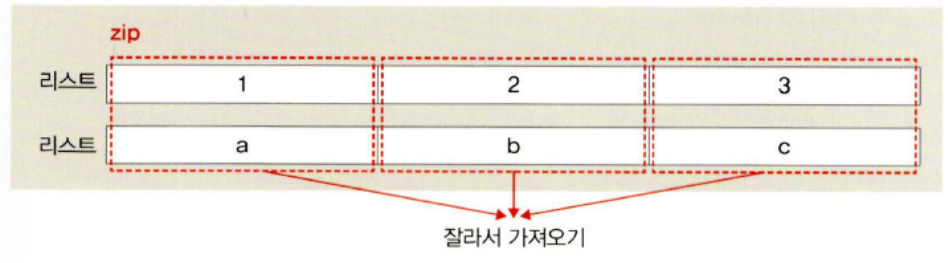
- 검증 데이터 전처리
    - `valid_data`를 `valid_data`라는 변수에 복사한 후 `'test_transforms'`로 전처리 적용함.
- 훈련, 검증, 테스트 데이터셋 수 확인
    - 훈련용 476개, 검증용 53개, 테스트용 12개
- 메모리로 데이터 불러오기
    - 데이터를 가져올 때는 배치 크기만큼 나누어서 가져옴.
- 옵티마이저와 손실 함수 정의
- 모델 정확도 측정 함수
    1. 예측이 정답과 일치하는 경우 그 개수의 합을 correct 변수에 저장함.
        a. `eq` : equal의 약자로 서로 같은지를 비교하는 표현식.
        b. `view_as(other)` : other의 텐서 크기를 사용하겠다는 의미. 즉, `view_as(other)`는 `view(other.size())`와 같은 의미. 따라서 `y.view_as(top_pred)`는 y에 대한 텐서 크기를 top_pred의 텐서 크기로 변경하겠다는 의미.
        c. `sum()` : 합계를 구하는 것. 여기에서는 예측과 정답이 일치하는 것들의 개수를 합산하겠다는 의미.
- 모델 학습 함수 정의
- 모델 성능 측정 함수
- 학습 시간 측정 함수
- 모델 학습
    - 성능이 좋지 않음.
    - 데이터셋으로 사용되는 이미지 수가 매우 적으며, 에포크도 매우 적게 설정.

    - 이들에 대한 숫자를 늘리면 성능은 좋아질 것. (데이터도 함께 증가시켜 모델 학습.)
- 테스트 데이터셋을 이용한 모델 성능 측정
    - 역시 결과가 좋지 않음.
    - 데이터셋을 늘리면 성능이 좋아지지만, 이미지 데이터가 늘어난 만큼 훈련 시간이 상당히 길어질 수 있음.
- 테스트 데이터셋을 이용한 모델의 예측 확인 함수
    1. `argmax` : 배열에서 가장 큰 값의 인덱스를 찾을 때 사용함.
        a. `1` : 행(axis=0) 또는 열(axis=1)을 따라 가장 큰 값의 색인을 찾음. 따라서 1은 열을 따라 가장 큰 값의 색인을 찾겠다는 의미.
        b. `keepdim` : `keepdim=True` 의 경우 출력 텐서를 입력과 동일한 크기로 유지하겠다는 의미.
    2. `torch.cat` : 텐서를 연결할 때 사용함.



        - dim=0은 행을 의미, dim=1은 열을 의미함.
- 예측 중에서 정확하게 예측한 것을 추출
    - 앞에서 정의한 get_predictions() 함수의 반환값을 각각 images, labels, probs에 저장하여 모델이 정확하게 예측한 이미지 추출함.
    1. max : 최댓값 반환, argmax : 최댓값을 갖는 인덱스 반환.
    2. zip() : 여러 개의 리스트(혹은 튜플)를 합쳐서 새로운 튜플 타입으로 반환함.

❤ 그림 6-18 zip()의 원리

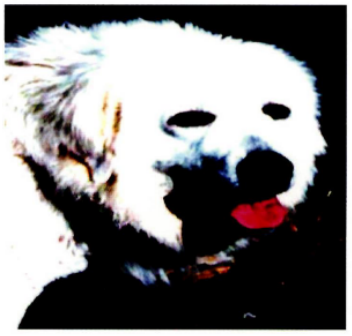


3. 데이터를 정렬하기 위해 sort() 메서드 사용함.
    a. reverse : 내림차순으로 정렬함.
    b. key : 데이터를 정렬할 때 key 값을 가지고 정렬하며 기본값은 오름차순. 또한, 여기서 사용되는 람다는 일종의 함수, 함수명 없이 사용 가능함.

❤ 그림 6-19 일반 함수와 람다 함수

- 이미지 출력을 위한 전처리
    - 그대로 출력(왜곡)
    - 전처리 후 본래 이미지 출력

❤ 그림 6-20 본래 이미지 색상이 왜곡됨



❤ 그림 6-21 본래 이미지

1. `torch.add` : 말 그대로 더하라는 메서드.
    a. `torch.add`와 `torch.add_`의 차이점
        - `torch.add` : 새로운 메모리 공간이 할당되어 저장되기 때문에 x와 y가 같은지 물었을 때 False라는 결과 출력.

        - `torch.add_` : 새로운 공간 할당 없이 기존의 메모리 공간에 있는 값을 새로운 값으로 대체하겠다는 의미.
    - 모델이 정확하게 예측한 이미지 출력 함수
        1. `image.permute` : 축을 변경할 때 사용함.
    - 예측 결과 이미지 출력
        - 모델이 정확하게 예측한 이미지 출력을 위해 plot_most_correct() 함수를 호출함.
        - 호출할 때 모델이 정확하게 예측한 이미지(correct_examples)에 대해 출력함.

그림 6-22 테스트 데이터셋에 대한 VGG 모델의 예측 결과

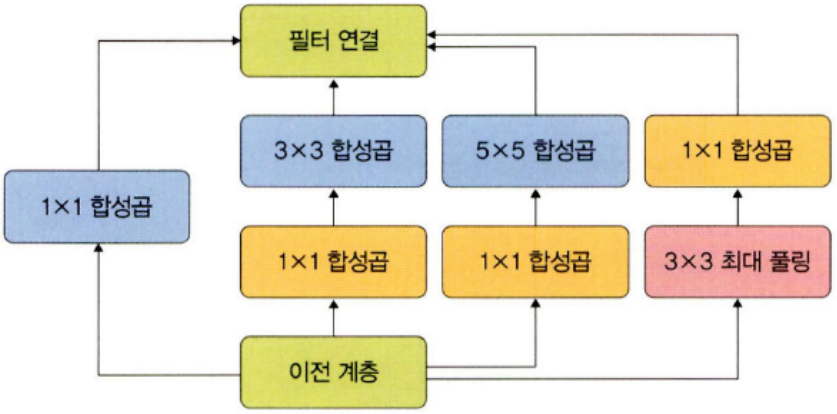
        - 결과의 정확도가 높지 않음.

6.1.4 GoogLeNet

- GoogLeNet
    - 주어진 하드웨어 자원을 최대한 효율적으로 이용하면서 학습 능력은 극대화할 수 있는 깊고 넓은 신경망.
    - 인셉션(inception) 모듈을 추가함.
        - 특징을 효율적으로 추출하기 위해 1x1, 3×3, 5×5 합성곱 연산을 각각 수행함.
        - 3×3 최대 풀링은 입력과 출력의 높이와 너비가 같아야 하므로 풀링 연산에서는 드물게 패딩을 추가해야 함.

        - 결과적으로 GoogLeNet에 적용된 해결 방법 → 희소 연결(sparse connectivity)
            - CNN은 합성곱, 풀링, 완전연결층들이 서로 밀집(dense)하게 연결되어 있음.
            - 빽빽하게 연결된 신경망 대신 관련성(connectation)이 높은 노드끼리만 연결하는 방법 → 희소 연결.
            - 연산량이 적어지고 과적합 해결 가능함.
- 인셉션 모듈 4가지 연산

그림 6-23 GoogLeNet의 인셉션 모듈



    - 1×1 합성곱
    - 1×1 합성곱 + 3x3 합성곱
    - 1×1 합성곱 + 5x5 합성곱
    - 3×3 최대 풀링 + 1x1 합성곱
- 딥러닝을 이용하여 ImageNet과 같은 대회에 참여하거나 서비스 제공하려면 대용량 데이터를 학습해야 함.
- 심층 신경망의 아키텍처에서 계층이 넓고(뉴런이 많고) 깊으면(계층이 많으면) 인식률은 좋아지지만, 과적합, 기울기 소멸 문제를 비롯한 학습 시간 지연과 연산 속도 등의 문제가 있음.
- 특히, 합성곱 신경망에서 이러한 문제들이 자주 나타남. GoogLeNet으로 이러한 문제 해결 가능함.