CH6-1

CH6. 합성곱 신경망 2

6.1 이미지 분류를 위한 신경망

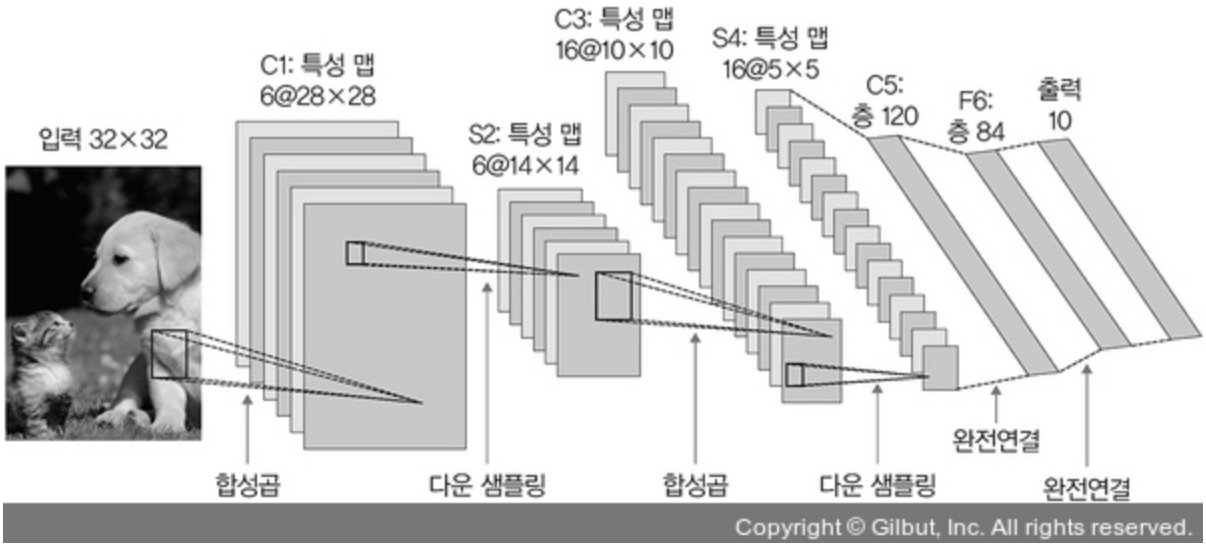
입력 데이터가 이미지인 분류는 특정 대상이 영상 내에 존재하는지 여부를 판단하는 것이다. 이미지 분류에서 주로 사용되는 합성곱 신경망의 유형을 알아보겠음.

6.1.1 LeNet-5

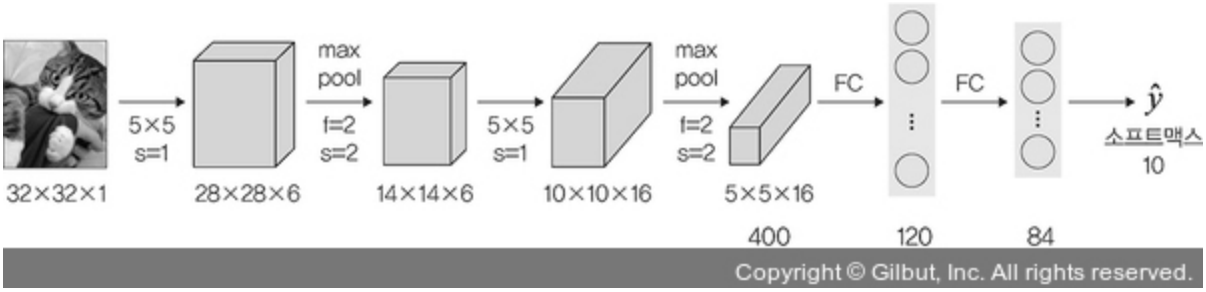
LeNet-5는 합성곱 신경망이라는 개념을 최초로 개발한 구조이다. 1995년 수표에 쓴 손글씨 숫자를 인식하는 딥러닝 구조 LeNet-5를 발표했는데, 그것이 현재 CNN의 초석이 됨. LeNet-5는 합성곱과 다운 샘플링(sub-sampling)(혹은 풀링)을 반복적으로 거치면서 마지막에 완전연결층에서 분류를 수행함.

아래 그림을 이용하여 구체적으로 살펴보면 C1에서 5×5 합성곱 연산 후 28×28 크기의 특성 맵(feature map) 여섯 개를 생성합니다. S2에서 다운 샘플링하여 특성 맵 크기를 14×14로 줄입니다. 다시 C3에서 5×5 합성곱 연산하여 10×10 크기의 특성 맵 16개를 생성하고, S4에서 다운 샘플링하여 특성 맵 크기를 5×5로 줄입니다. C5에서 5×5 합성곱 연산하여 1×1 크기의 특성 맵 120개를 생성하고, 마지막으로 F6에서 완전연결층으로 C5의 결과를 유닛(unit)(또는 노드) 84

개에 연결시킵니다. 이때 C로 시작하는 것은 합성곱층을 의미하고, S로 시작하는 것은 풀링층을 의미합니다. 또한, F로 시작하는 것은 완전연결층을 의미합니다.



LeNet-5을 사용하는 예제를 구현하면 다음과 같음. 앞 장에서 사용한 개와 고양이 데이터셋을 다시 사용함.

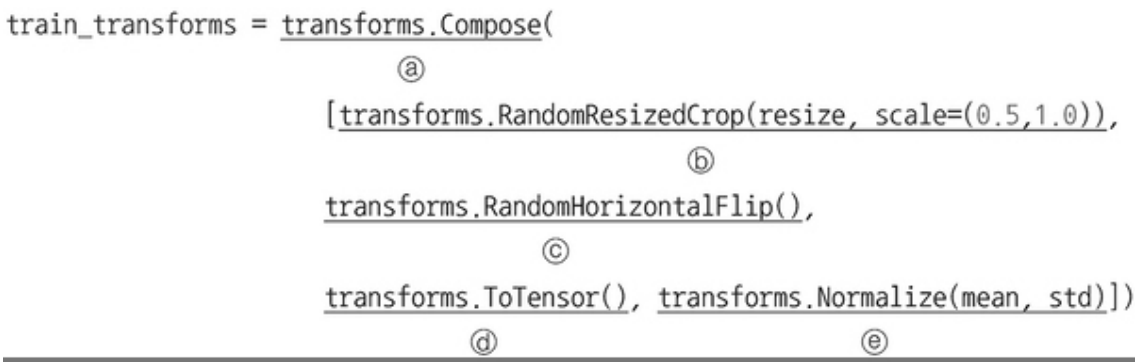


32×32 크기의 이미지에 합성곱층과 최대 풀링층이 쌍으로 두 번 적용된 후 완전연결층을 거쳐 이미지가 분류되는 신경망이다.

신경망에 대한 자세한 설명은 다음 표와 같음.

LeNet-5 예제 코드 - 복습

1. 모델 학습에 필요한 데이터셋의 전처리(예: 텐서 변환)이 필요함
    a. 토치비전 라이브러리를 이용하면 이미지에 대한 전처리 손쉽게 할 수 있음.



1. `transforms.Compose` 이미지를 변형할 수 있는 방식들의 묶음.
2. `transforms.RandomResizedCrop` 입력 이미지를 주어진 크기로 조정함. 또한, scale은 원래 이미지를 임의의 크기만큼 면적을 무작위로 자르겠다는 의미.
3. `transforms.RandomHorizontalFlip` 주어진 확률로 이미지를 수평 반전시킴. 이때 확률 값을 지정하지 않았으므로 기본값인 0.5의 확률로 이미지들이 수평 반전 됨. 즉, 훈련 이미지 중 반은 위아래 뒤집한 상태로 두고, 반은 그대로 사용함.
4. `transforms.ToTensor` ImageFolder 메서드를 비롯해서 토치비전 메서드는 이미지를 읽을 때 파이썬 이미지 라이브러리인 PIL을 사용함. PIL을 사용해서 이미지를 읽으면 생성되는 이미지는 범위가 [0,255]이며, 배열의 차원이(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 차원의 순서도(채널 수 C X 높이 H X 너비 W)로 바뀜. 그리고 이러한 작업을 수행해 주는 메서드가 `ToTensor()` 이다.
5. `transforms.Normalize` 전이 학습에서 사용하는 사전 훈련된 모델들은 대게 ImageNet 데이터셋에서 훈련됨. 따라서 사전 훈련된 모델을 사용하기 위해서는 ImageNet 데이터의 각 채널별 평균과 표준편차에 맞는 정규화를 해주어야 함. 즉, Normalize 메서드 안에 사용된 (mean: 0.485, 0.456, 0.406), (std: 0.229, 0.224, 0.225)는 ImageNet에서 이미지를 읽어 온다면 RGB 이미지가 아닌 BGR 이미지이므로 채널 순서에 주의해야 함.

b. `__call__` 함수는 클래스를 호출할 수 있도록 하는 메서드이다. `__init__` 은 인스턴스 초기화를 위해 사용한다면 `__call__`은 인스턴스가 호출되었을 때 실행됩니다. 즉, 클래스에 **call** 함수가 있을 경우 클래스 객체 자체를 호출하면 **call** 함수의 리턴(return) 값이 반환됨.

이미지가 위치한 디렉터리에서 데이터를 불러온 후 훈련용으로 400개의 이미지, 검증용으로 92개의 이미지, 테스트용으로 열 개의 이미지를 사용함.

예제를 진행하기 위한 시스템 성능이 좋을 경우 훈련용 2만개를 사용하면 모델 성능이 더 높아짐.

```
# 코드 6-3. 이미지 데이터셋을 불러온 후 훈련, 검증, 테스트로 분리

cat_directory = r'../chap06/data/dogs-vs-cats/Cat/'
dog_directory = r'../chap06/data/dogs-vs-cats/Dog/'

cat_images_filepaths = sorted([os.path.join(cat_directory, f) for f in
                               os.listdir(cat_directory)]) ------ ①
dog_images_filepaths = sorted([os.path.join(dog_directory, f) for f in
                               os.listdir(dog_directory)])
images_filepaths = [*cat_images_filepaths, *dog_images_filepaths]  ------ 개와 고양이 이미지들
을 합쳐서 images_filepaths에 저장
correct_images_filepaths = [i for i in images_filepaths if cv2.imread(i) is not None] ----
-- ②

random.seed(42) ------ ③
random.shuffle(correct_images_filepaths)
train_images_filepaths = correct_images_filepaths[:400] ------ 훈련용 400개의 이미지
val_images_filepaths = correct_images_filepaths[400:-10] ------ 검증용 92개의 이미지
test_images_filepaths = correct_images_filepaths[-10:] ------ 테스트용 열 개의 이미지
print(len(train_images_filepaths), len(val_images_filepaths), len(test_images_filepaths))
```

1. 고양이 이미지를 가져옴

```
cat_images_filepaths = sorted([os.path.join(cat_directory, f)
                               ⓐ            ⓑ
                               for f in os.listdir(cat_directory)])
                                                   ⓒ
```


   a. `sorted` 데이터를 정렬된 리스트로 만들어서 반환함

   b. `os.path.join` 경로와 파일명을 결합하거나 분할된 경로를 하나로 합치고 싶을 때 사용함. 즉, cat_directory 디렉터리(../chap06/data/dogs-vs-cats/Cat/)와 os.listdir을 통해 검색된 이미지 파일들(f)을 하나로 합쳐서 '../chap06/data/dogs-vs-cats/Cat/이미지파일명'(**예**

../chap06/data/dogs-vs-cats/Cat/cat.0.jpg)으로 표시해 줌.
또한, 아래와 같이 경로를 하나로 합칠 수 있음.

```
import os
list_path = ['C:\\', 'Temp', 'user']
folder_path = os.path.join(*list_path)
folder_path
```

다음은 실행 결과입니다. 다음과 같이 경로가 하나로 합쳐 있는 것을 확인할 수 있습니다.

`'C:\\Temp\\user'`

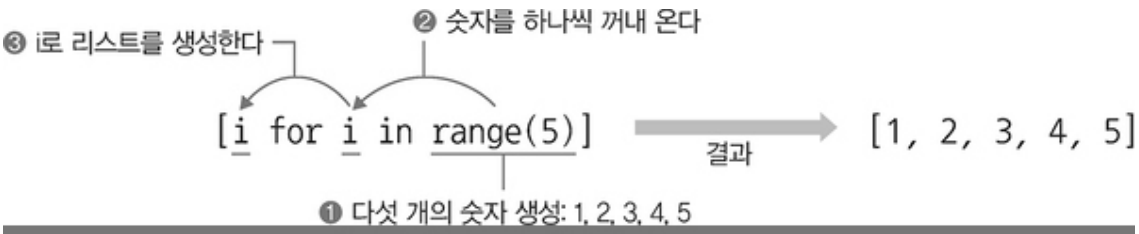
참고로 윈도 환경에서는 경로가 '\\'으로 표시됩니다.

c. `os.listdir` 지정한 디릭터리 내 모든 파일의 리스트를 반환함. 예제에서 사용하는 cat 디렉터리의 이미지 파일들을 모두 반환함.

2. `images_filepaths` 에서 이미지 파일들을 불러옴.

```
correct_images_filepaths = [i for i in images_filepaths
                            ⓐ       ⓑ
                            if cv2.imread(i) is not None]
                                    ⓒ
```


a. for 반복문을 이용하여 가져온 데이터에 대해 i를 이용하여 리스트로 만듦. 즉, 아래와 같은 의미를 가짐.



b. 반복문(for)을 이용하여 `images_filepaths` 에서 이미지 데이터를 검색함

c. 조건문(if)를 의미함. cv2.imread() 함수(cv2.imread(i))를 이용하여 모든 이미지 데이터를 읽어 옵니다(not None 상태, 즉 '더 이상 데이터를 찾을 수 없을 때까지'를 의미함).

3. 넘파이 random() 함수는 임의의 난수를 생성하는데, 이때 난수를 생성하기 위해 사용되는 것이 시드 값(seed value)이다. 또한, Numpy.random.seed() 메서드는 상태를 초기화합니다. 즉, 이 모듈이 호출될 때마다 임의의 난수가 재생성됩니다. 하지만 특정 시드 값을 부여하면 상태가 저장되기 때문에 동일한 난수를 생성합니다. 예를 들어 다음과 같습니다.

⇒ 지금까지는 주어진 데이터셋을 훈련, 검증, 테스트 용도로 분리했는데, 테스트 용도의 데이터셋에 어떤 데이터들이 있는지 확인해보겠음.

```
# 6-4. 테스트 데이터셋 이미지 확인 함수

def display_image_grid(images_filepaths, predicted_labels=(), cols=5):
    rows = len(images_filepaths) // cols
    figure, ax = plt.subplots(nrows=rows, ncols=cols, figsize=(12, 6))
    for i, image_filepath in enumerate(images_filepaths):
        image = cv2.imread(image_filepath)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) ------ ①
        true_label = os.path.normpath(image_filepath).split(os.sep)[-2] ------ ②
        predicted_label = predicted_labels[i] if predicted_labels else true_label ------
 ③
        color = "green" if true_label == predicted_label else "red" ------ 예측과 정답(레이
블)이 동일하면 초록색으로 표시하고, 그렇지 않다면 빨간색으로 표시
        ax.ravel()[i].imshow(image) ------ 개별 이미지를 출력
        ax.ravel()[i].set_title(predicted_label, color=color) ------ predicted_label을 타이
틀로 사용
        ax.ravel()[i].set_axis_off() ------ 이미지의 축 제거
    plt.tight_layout() ------ 이미지의 여백을 조정
    plt.show()
```

⇒ 예측력이 좋지는 않음. 극히 일부의 데이터를 이용해 모델 학습을 진행했기 때문에 이런 예측력을 보이는 것.

지금까지 LeNet-5의 코드를 살펴보았고 이미지 분류에서 사용되는 또다른 모델인 AlexNet에 대해 살펴보겠음. 이제부터 AlexNet을 포함한 여러 유형의 컴퓨터 비전 모델들을 사폴 볼 텐데 LeNet에서 사용했던 코드와 유사한 부분이 많음.

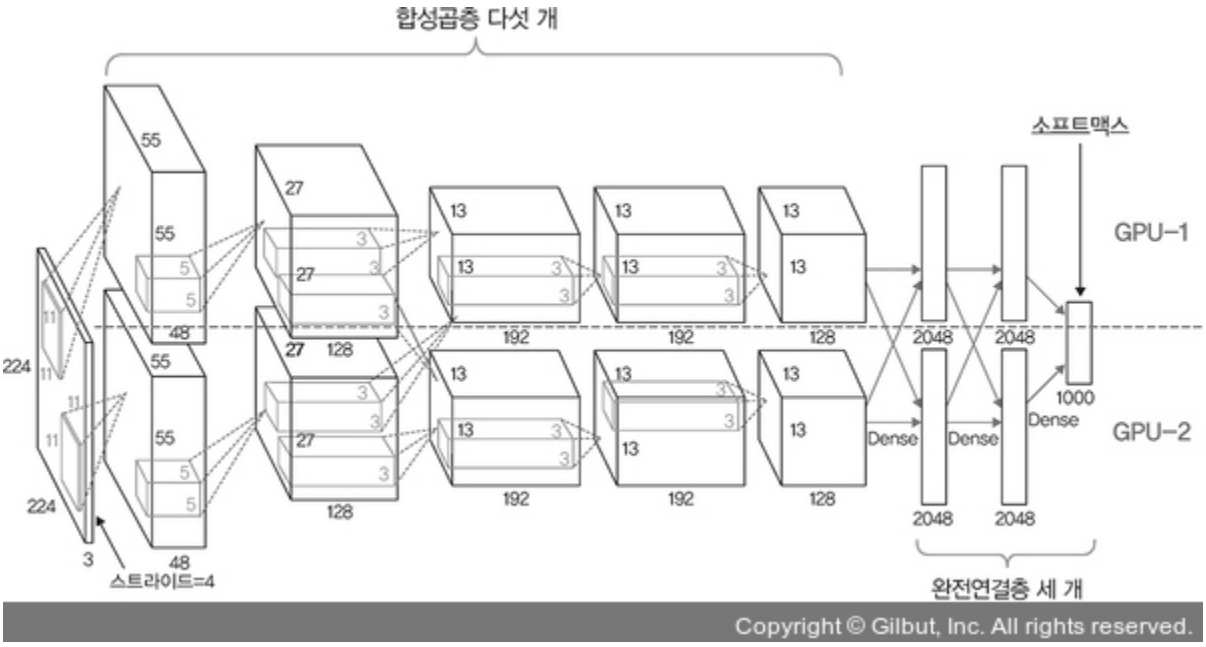
6.1.2 AlexNet

AlexNet은 ImageNet 영상 데이터베이스를 기반으로 한 화상 인식 대회인 'ILSVRC 2012'에서 우승한 CNN 구조이다.

AlexNet의 세부 블록을 이해하고자 CNN의 구조를 다시 살펴보겠음. CNN은 다음 그림과 같이 3차원 구조를 갖는다는 것을 이해해야 함.(이미지를 다루기 때문에 기본적으로 3차원 데이터를 다룸.

이미지 크기를 나타내는 너비(W)와 높이(H) 뿐만 아니라 깊이(D)를 가짐. 보통 색상이 많은 이미지는 RGB 세개 성분을 갖기 때문에 시작이 3이지만, 합성곱을 거치면서 특성 맵이 만들어지고 이것에 따라 중간 영상의 깊이가 달라짐.

이걸 이해하면, AlexNet 구조에 있는 숫자 의미에 대한 이해가 가능함. **AlexNet은 합성곱층 총 5개와 완전연결층 3개로 구성되어 있으며, 맨 마지막 완전연결층은 카테고리 1000개를 분류하기 위해 소프트맥스 활성화 함수를 사용함.** 전체적으로 보면 GPU 두 개를 기반으로 한 **병렬 구조**인 점을 제외하면 LeNet-5와 크게 다르지 않습니다.



AlexNet의 **합성곱층에서 사용된 활성화 함수는 렐루(ReLU)**이다.

네트워크에는 학습 가능한 변수가 총 6600만 개 있음. 네트워크에 대한 입력은 227×227×3 크기의 RGB 이미지이며, 각 클래스(혹은 카테고리)에 해당하는 1000×1 확률 벡터를 출력함.

AlexNet의 첫 번째 합성곱층 커널의 크기는 11×11×3이며, 스트라이드를 4로 적용하여 특성 맵을 96개 생성하기 때문에 55×55×96의 출력을 가짐. 첫 번째 계층을 거치면서 GPU-1에서는 주

로 컬러와 상관없는 정보를 추출하기 위한 커널이 학습되고, GPU-2에서는 주로 컬러와 관련된 정보를 추출하기 위한 커널이 학습됨.

이제 파이토치 코드로 AlexNet에 대해 살펴봅시다. 먼저 필요한 라이브러리를 호출함. 전반적인 코드는 LeNet과 크게 다르지 않기 때문에 주로 모델을 구성하는 네트워크가 어떻게 차이가 나는지 위주로 살펴보면 좋습니다.

AlexNet 예제 코드 - 복습

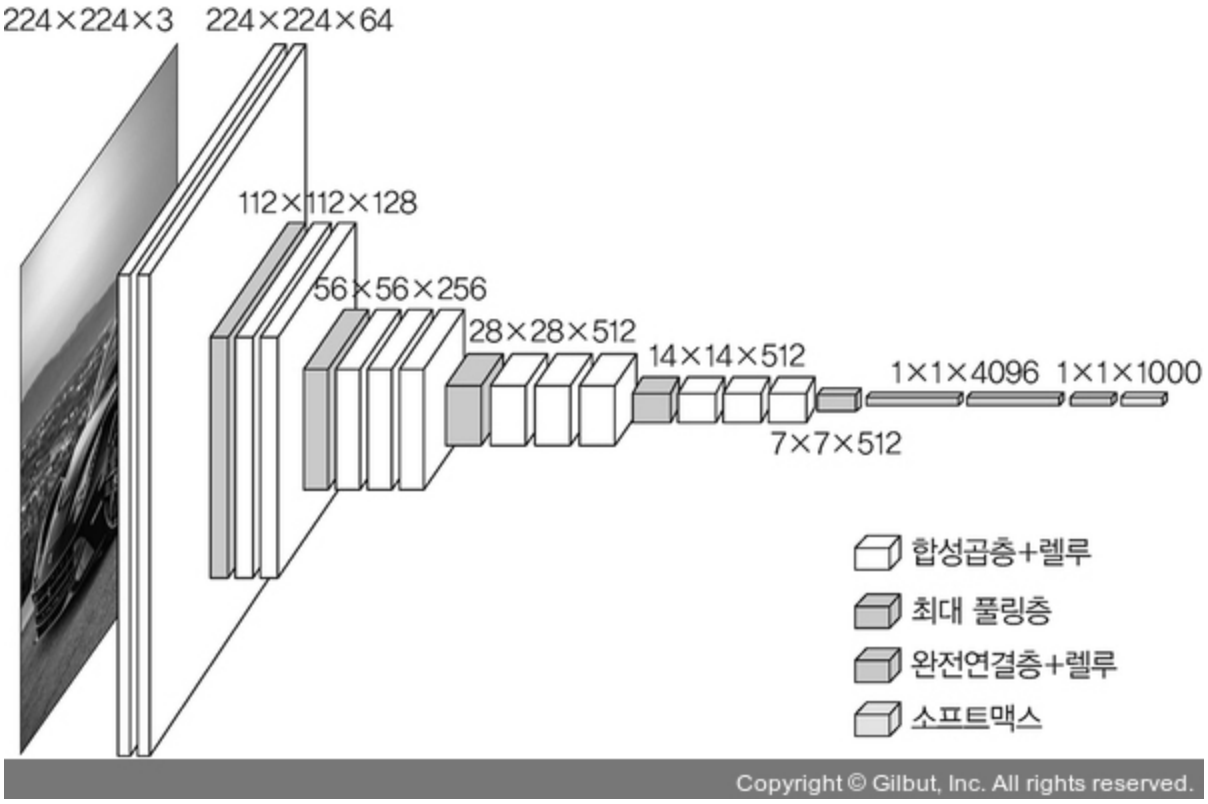
⇒ 역시 예측 결과 좋지 않음. 데이터셋이 많으면 성능 좋아질 수 있음. 파라미터 튜닝을 통해 성능 향상 할 수 있음.(8장)

6.1.3 VGGNet

VGGNet은 2015 ICLR에 게재한 “Very deep convolutional networks for large-scale image recognition” 논문에서 처음 발표 됨. VGGNet은 합성곱층의 파라미터 수를 줄이고 훈련 시간을 개선하려고 탄생함. 즉, 네트워크를 깊게 만드는 것이 성능에 어떤 영향을 미치는지 확인하고자 나온 것이 VGG. VGG 연구팀은 깊이의 영향만 최대한 확인하고자 합성곱층에서 사용하는 필터/커널의 크기를 가장 작은 3X3으로 고정함.

네트워크 계층의 총 개수에 따라 여러 유형의 VGGNet이 있으며, 이 중 VGGNet16 네트워크의 구조적 세부 사항은 아래 그림과 같음.

VGGNet16에는 파라미터가 총 1억개가 넘게 있음. 여기에서 주목할 점은 모든 합성곱 커널의 크기는 3×3, 최대 풀링 커널의 크기는 2×2이며, 스트라이드는 2라는 것입니다. 결과적으로 64개의 224×224 특성 맵(224×224×64)들이 생성됨. 또한, 마지막 16번째 계층을 제외하고는 모두 ReLU 활성화 함수가 적용됨.



VGG11 예제 코드 - 복습

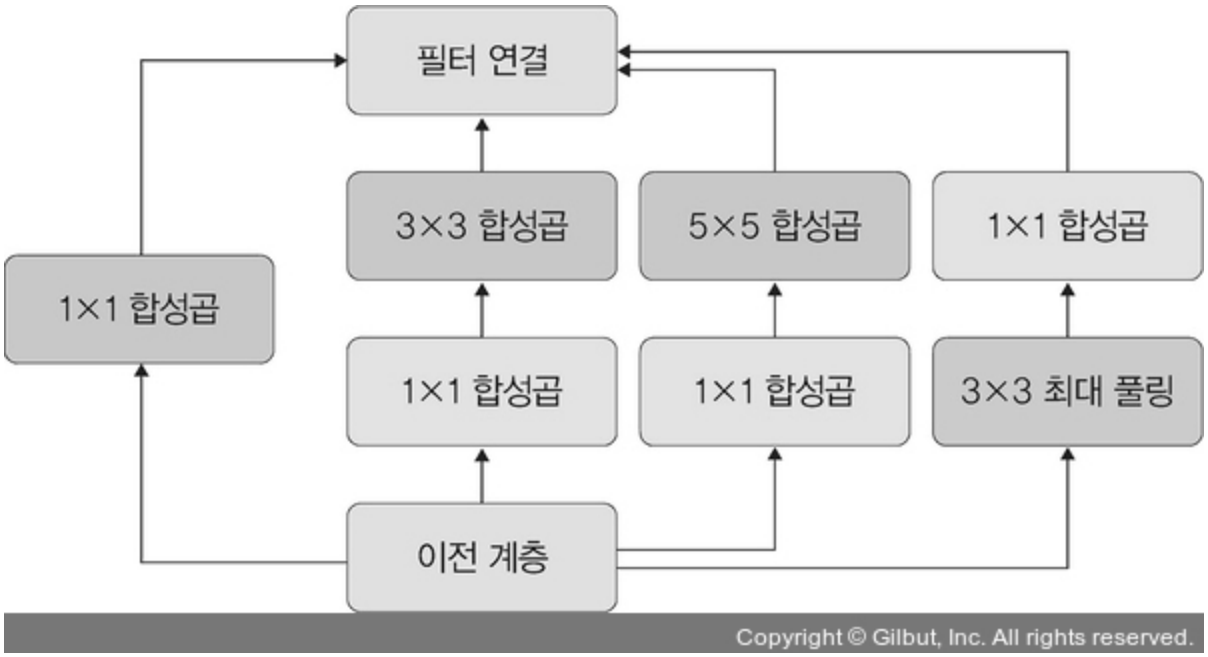
VGGNet 중에서 가장 간단한 VGG11을 파이토치로 구현함. 다른 것을 구현해보고 싶다면, VGG11에서 더 깊게 쌓아 올리면 됨.

⇒ 성능 좋지 않음. 데이터셋으로 사용되는 이미지 수가 매우 적으며, 에포크도 작게 설정되었기 때문. 이들에 대한 숫자를 늘리면 성능 좋아질 것.

6.1.4 GoogLeNet

GoogLeNet은 주어진 하드웨어 자원을 최대한 효율적으로 이용하면서 학습 능력은 극대화할 수 있는 깊고 넓은 신경망이다.

깊고 넓은 신경망을 위해 GoogLeNet은 인셉션(inception) 모듈을 추가함. 인셉션 모듈에서는 특징을 효율적으로 추출하기 위해 1×1, 3×3, 5×5의 합성곱 연산을 각각 수행함. 3×3 최대 풀링은 입력과 출력의 높이와 너비가 같아야 하므로 풀링 연산에서는 드물게 패딩을 추가해야 한다. 결과적으로 GoogLeNet에 적용된 해결 방법은 희소 연결(sparse connectivity)입니다. CNN은 합성곱, 풀링, 완전연결층들이 서로 밀집(dense)(정교하고 빽빽하게)하게 연결되어 있습니다. 빽빽하게 연결된 신경망 대신 관련성(correlation)이 높은 노드끼리만 연결하는 방법을 희소 연결이라고 함. 이것으로 연산량이 적어지며 과적합도 해결할 수 있습니다.



인셉션 모듈의 네 가지 연산은 다음과 같습니다.

- 1×1 합성곱
- 1×1 합성곱 + 3×3 합성곱
- 1×1 합성곱 + 5×5 합성곱
- 3×3 최대 풀링(maxpooling) + 1×1 합성곱(convolutional)

딥러닝을 이용하여 ImageNet과 같은 대회에 참여하거나 서비스를 제공하려면 대용량 데이터를 학습해야 함. 심층 신경망의 아키텍처에서 계층이 넓고(뉴런이 많고) 깊으면(계층이 많으면) 인식률은 좋아지지만, 과적합이나 기울기 소멸 문제(vanishing gradient problem)를 비롯한 학습 시간 지연과 연산 속도 등의 문제가 있습니다. 특히 합성곱 신경망에서 이러한 문제들이 자주 나타나는데, **GoogLeNet(혹은 인셉션이라고도 불림)으로 이러한 문제를 해결**할 수 있다고 생각하면 됩니다.

6.1.5 ResNet

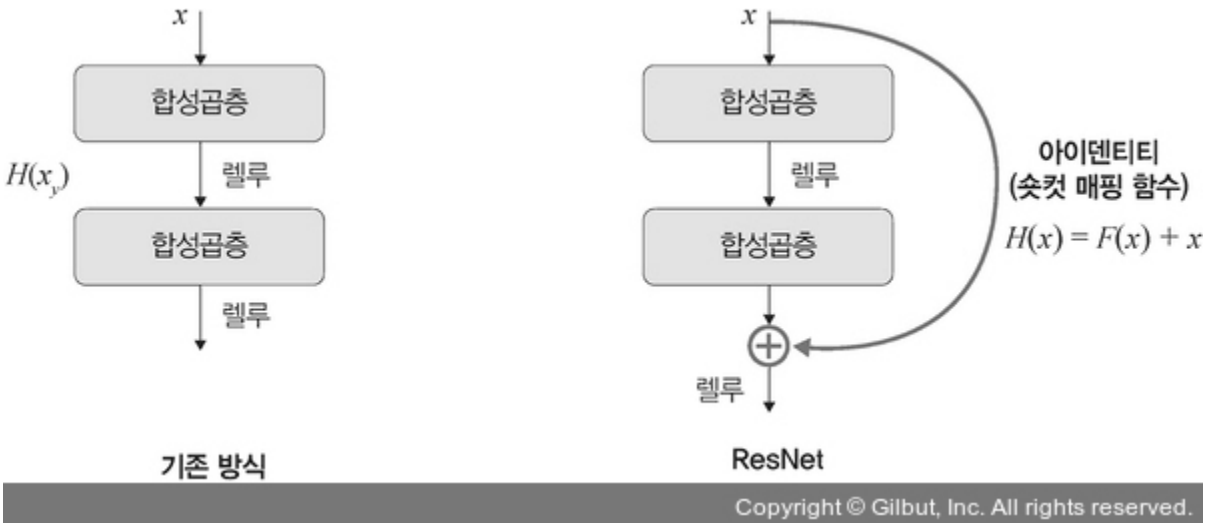
ResNet은 마이크로소프트에서 개발한 알고리즘으로 “Deep Residual Learning for Image Recognition”이라는 논문에서 발표되었습니다. ResNet 핵심은 깊어진 신경망을 효과적으로 학습하기 위한 방법으로 레지듀얼(residual) 개념을 고안한 것입니다.

일반적으로 신경망 깊이가 깊어질수록 딥러닝 성능은 좋아질 것 같지만, 실상은 그렇지 않습니다. “Deep Residual Learning for Image Recognition” 논문에 따르면, 신경망은 깊이가 깊어

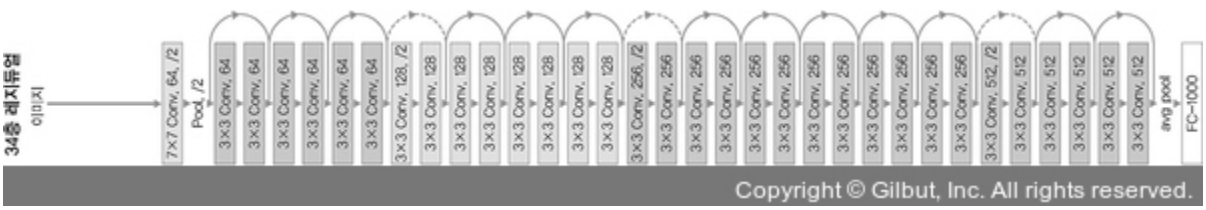
질수록 성능이 좋아지다가 일정한 단계에 다다르면 오히려 성능이 나빠진다고 합니다.

다음 그림과 같이 네트워크 56층이 20층보다 더 나쁜 성능을 보임을 알 수 있습니다. 즉, 네트워크 깊이가 깊다고 해서 무조건 성능이 좋아지지는 않는다는 것을 보여 주고 있습니다. ResNet은 바로 이러한 **문제를 해결하기 위해 레지듀얼 블록(residual block)을 도입**했습니다. **레지듀얼 블록은 기울기가 잘 전파될 수 있도록 일종의 숏컷(shortcut, skip connection)을 만들어 준다.**

이러한 개념이 필요한 이유는 2014년에 공개된 GoogLeNet은 층이 총 22개로 구성된 것에 비해 ResNet은 층이 총 152개로 구성되어 기울기 소멸 문제가 발생할 수 있기 때문입니다. 따라서 다음 그림과 같이 숏컷을 두어 기울기 소멸 문제를 방지했다고 이해하면 됩니다.



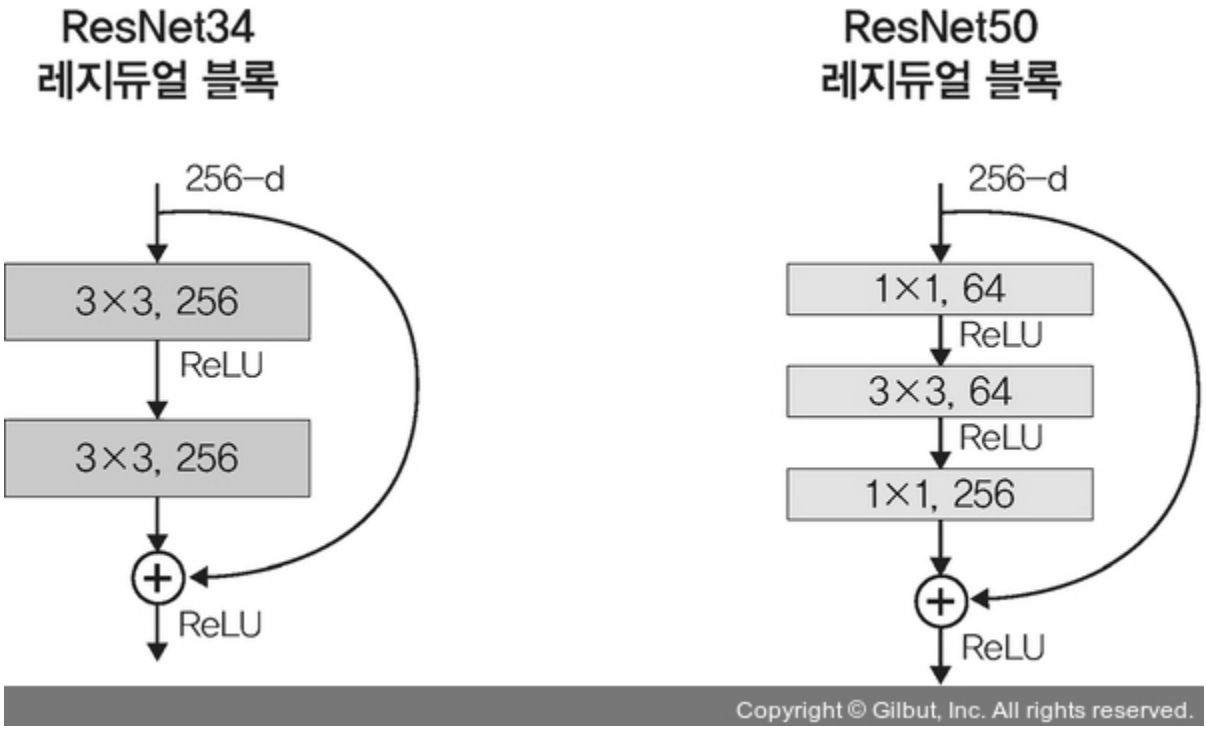
아직까지는 ResNet을 이해하기 어렵습니다. 차근차근 하나씩 살펴보겠습니다. 먼저 블록(block)이라는 개념에 대해 알아보겠습니다. 블록은 계층의 묶음입니다. 엄밀히 말해서 **합성곱층을 하나의 블록으로 묶은 것**입니다.
그림 6-26에서 색상별(보라색, 노란색 등)로 블록을 구분했는데 이렇게 묶인 계층들을 하나의 레지듀얼 블록(residual block)이라고 합니다. 그리고 ㅠ이라고 합니다.



하지만 이렇게 계층을 계속해서 쌓아 늘리면 파라미터 수가 문제가 됩니다. 계층이 깊어질수록 파라미터는 증가합니다. 예를 들어 ResNet34는 합성곱층이 34개와 16개의 블록으로 구성되어

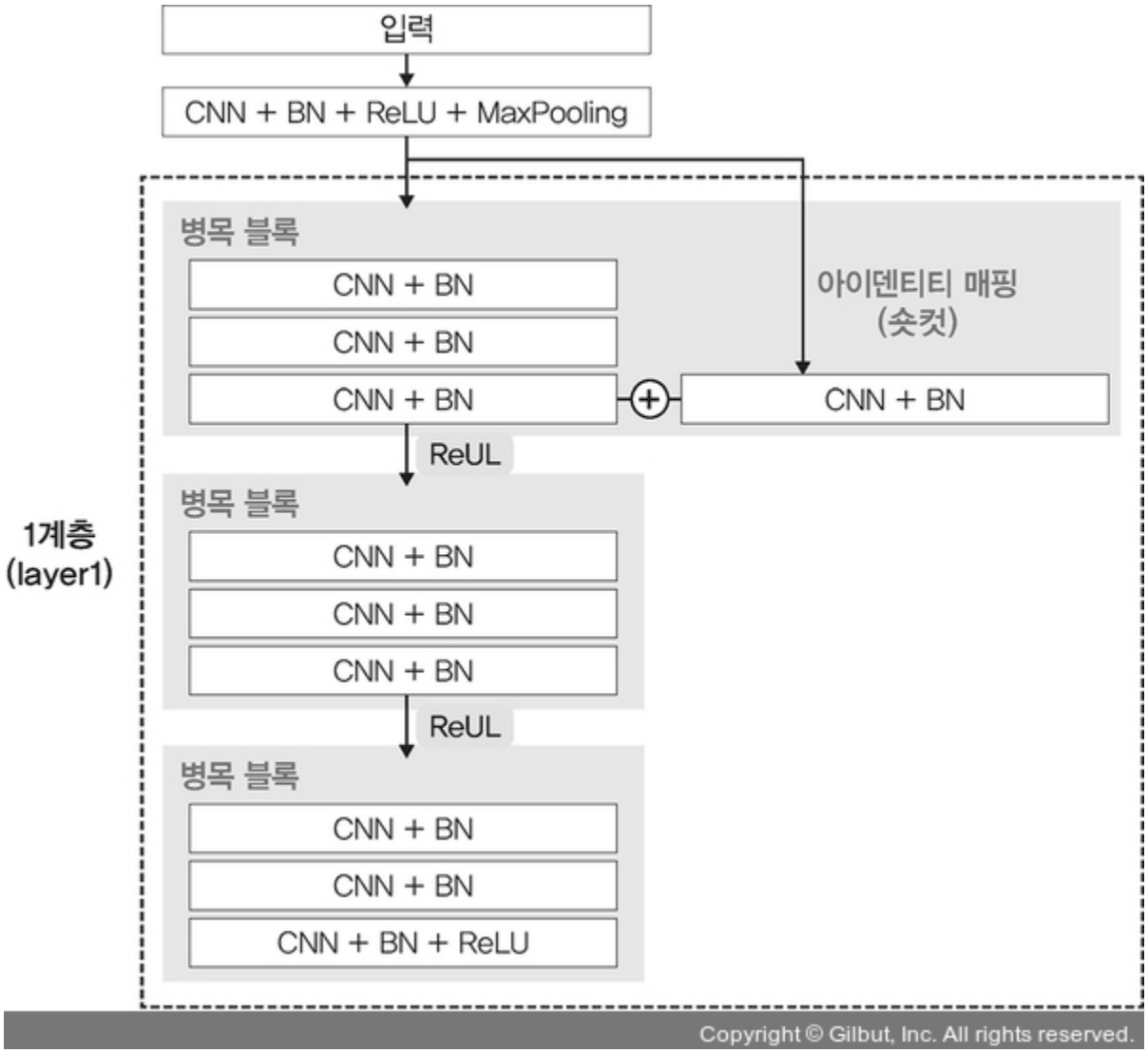
있습니다. 첫 번째 블록의 파라미터가 1152K라면 전체 파라미터 수는 2만 1282K입니다. 이와 같이 계층의 깊이가 깊어질수록 파라미터는 무제한으로 커질 것입니다. 이러한 문제를 해결하기 위해 병목 블록(bottleneck block)이라는 것을 두었습니다.

병목 블록을 두었을 때 어떤 현상이 발생할까요? 다음 그림은 ResNet34와 ResNet50입니다. ResNet34는 기본 블록(basic block)을 사용하며, ResNet50은 병목 블록을 사용합니다. 기본 블록의 경우 파라미터 수가 39.3216M인 반면, 병목 블록의 경우 파라미터 수가 6.9632M입니다. 깊이가 깊어졌음에도 파라미터 수는 감소한 것입니다.



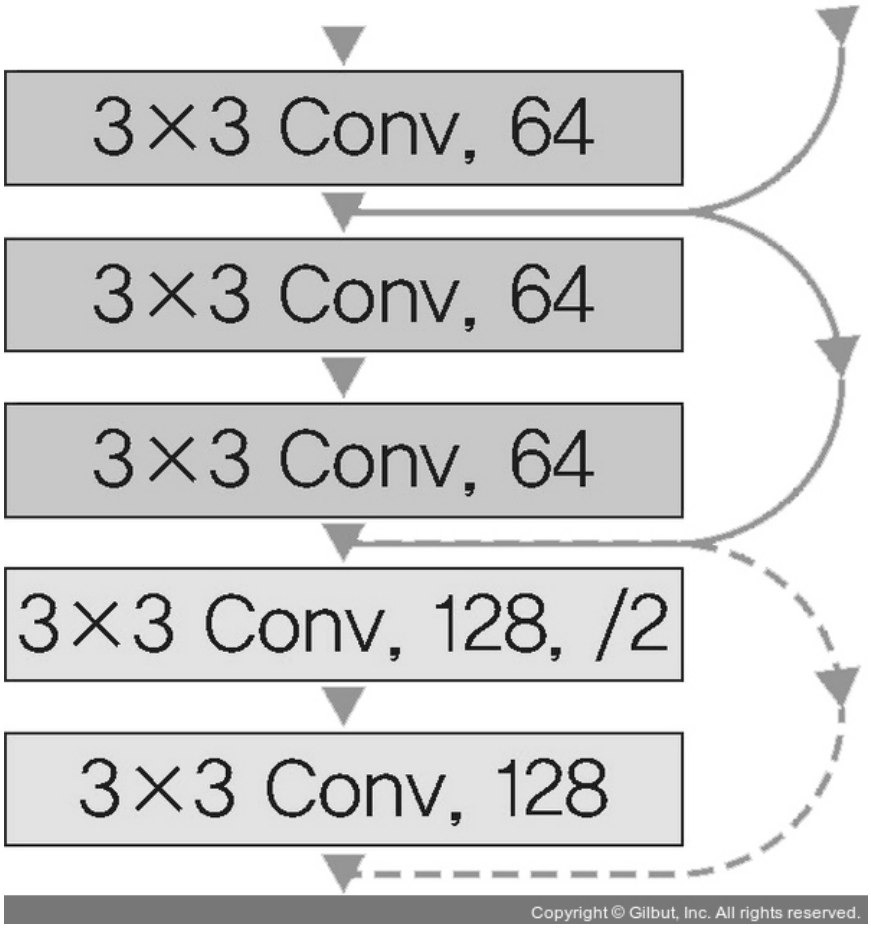
어떻게 가능한 것일까요? 앞에서 분명 깊이가 깊어질수록 파라미터 수가 증가한다고 했습니다. 하지만 병목 블록을 사용하면 파라미터 수가 감소하는 효과를 줄 수 있습니다. 합성곱층을 자세히 보면 ResNet34와는 다르게 ResNet50에서는 3×3 합성곱층 앞뒤로 1×1 합성곱층이 붙어 있는데, 1×1 합성곱층의 채널 수를 조절하면서 차원을 줄였다 늘리는 것이 가능하기 때문에 파라미터 수를 줄일 수 있었던 것입니다. 그리고 이 부분이 병목과 같다고 하여 병목 블록이라고 합니다.

이제 중요한 아이덴티티 매핑(identity mapping)(혹은 숏컷(shortcut), 스킵 연결(skip connection)이라고도 함)에 대해 알아보겠습니다. 그림 6-27의 아래쪽에 + 기호가 있습니다(기본 블록과 병목 블록 모두에서 사용됩니다). 이 부분을 아이덴티티 매핑이라고 합니다. 아이덴티티 매핑이란 입력 x가 어떤 함수를 통과하더라도 다시 x라는 형태로 출력되도록 합니다.



예시 코드

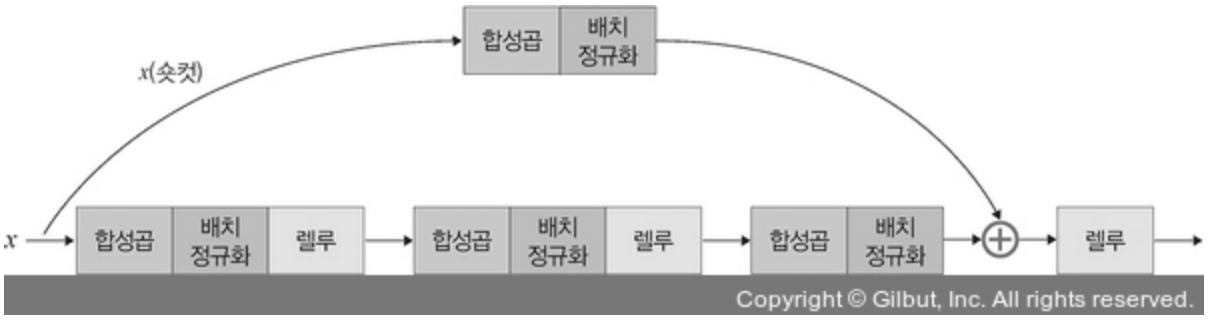
이번에는 또 다른 핵심 개념인 다운샘플(downsample)에 대해 알아보겠습니다. 다운샘플은 특성 맵(feature map) 크기를 줄이기 위한 것으로 풀링과 같은 역할을 한다고 이해하면 됩니다. 다음 그림은 ResNet 네트워크의 일부를 가져온 것입니다.



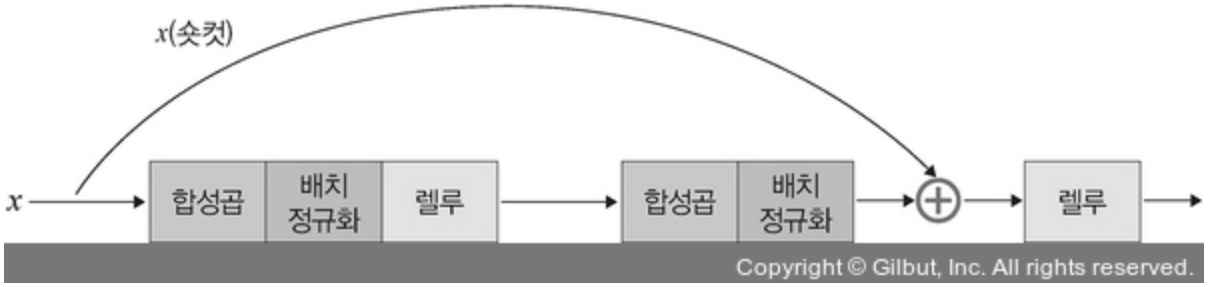
보라색 영역의 첫 번째 블록에서 특성 맵의 형상이 (28, 28, 64)였다면 세 번째 블록의 마지막 합성곱층을 통과하고 아이덴티티 매핑(identity mapping)까지 완료된 특성 맵의 형상도 (28, 28, 64)입니다. 이번에는 노란색 영역을 살펴볼까요? 노란색 영역의 시작 지점에서는 채널 수가 128로 늘어났고, /2라는 것으로 보아 첫 번째 블록에서 합성곱층의 스트라이드가 2로 늘어나 (14, 14, 128)로 바뀐다는 것을 알 수 있습니다.

즉, 보라색과 노란색의 형태가 다른데 이들 간의 형태를 맞추지 않으면 아이덴티티 매핑을 할 수 없게 됩니다. 그래서 아이덴티티에 대해 다운샘플이 필요합니다.

참고로 입력과 출력의 형태를 같도록 맞추어 주기 위해서는 스트라이드(stride) 2를 가진 1×1 합성곱 계층을 하나 연결해 주면 됩니다. 이와 같이 입력과 출력의 차원이 같은 것을 아이덴티티 블록이라고 하며, 입력 및 출력 차원이 동일하지 않고 입력의 차원을 출력에 맞추어 변경해야 하는 것을 프로젝션 숏컷(projection-shortcut) 혹은 합성곱 블록이라고 합니다.

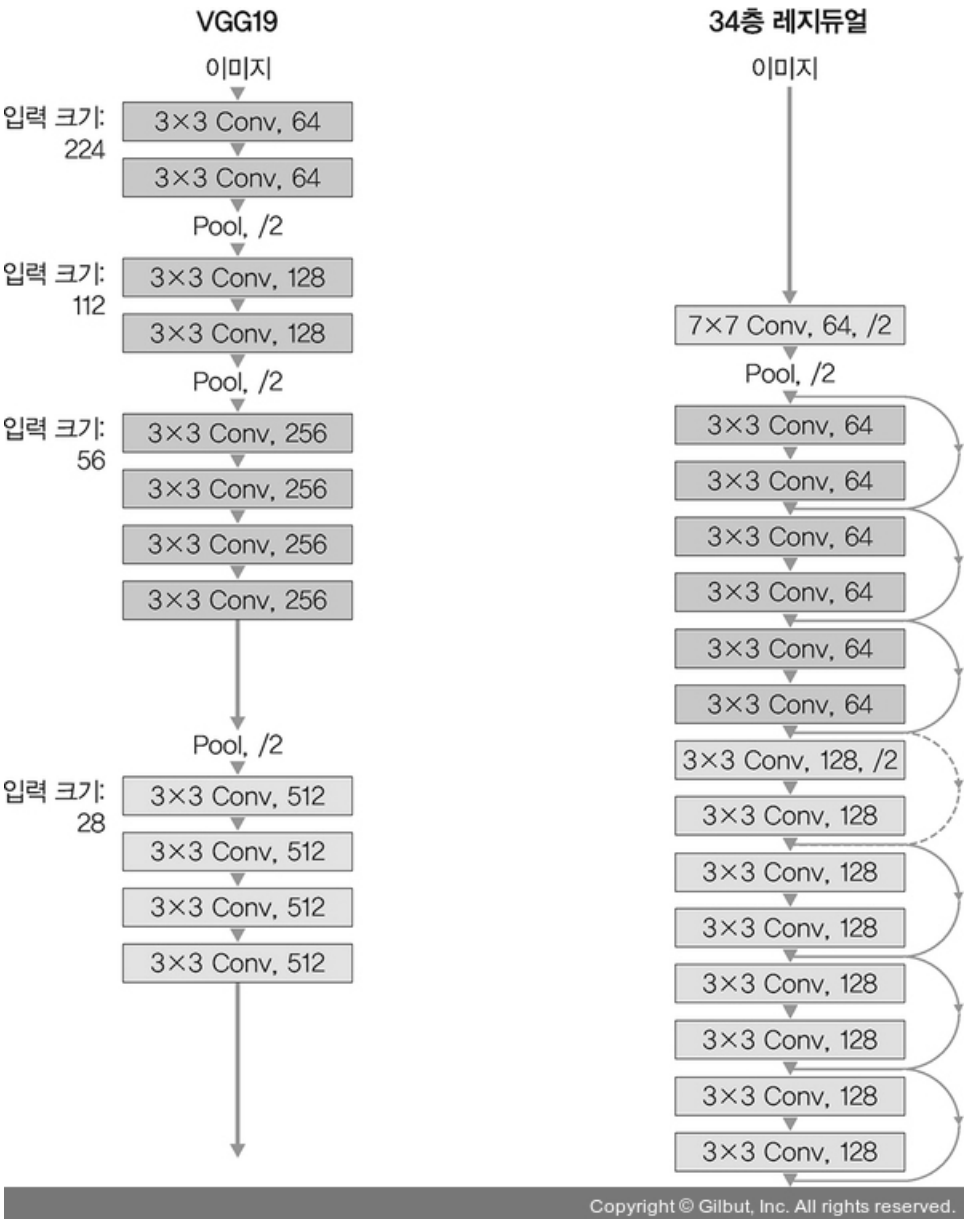


▲ **그림 6-30** 합성곱 블록



▲ **그림 6-31** 아이덴티티 블록

정리하면 ResNet은 기본적으로 VGG19 구조를 뼈대로 하며, 거기에 합성곱층들을 추가해서 깊게 만든 후 **숏컷들을 추가**하는 것이 사실상 전부라고 생각하면 됩니다.



▲ **그림 6-32** VGG19와 ResNet 비교

ResNet 예제 코드 - 복습

결국 아이덴티티 매핑과 병목 블록으로 ResNet 네트워크에 더욱 깊은 계층을 쌓을 수 있게 된 것입니다.

이제 ResNet 모델에 대한 네트워크를 정의해 봅시다. ResNet 역시 VGG처럼 다양한 모델이 있습니다.

ResNet 예제 코드 - 복습

⇒ 역시 정확도가 높지 않습니다. 계속 이야기하지만 성능을 향상시키려면 데이터를 더 늘려야 합니다. 책에서는 학습 용도로 CNN의 다양한 모델을 어떻게 사용하는지에 집중했습니다. 즉, 네트워크 위주의 학습을 위해 일부로 동일한 데이터셋을 사용했습니다. 따라서 이 장에서는 CNN의 다양한 모델의 네트워크 구성 위주로 학습하는 것을 권장합니다. 성능 향상 관련해서는 8장을 참고하세요.

컴퓨터 비전 분야에서 객체를 분류하는 방법에 대해 감이 좀 오나요? 중요한 것은 모델을 학습하기 위한 데이터입니다. 신경망(혹은 네트워크)은 이미 구현된 모델을 재사용할 수 있는 것이 많기에 우리는 단지 누군가가 만들어 놓은 신경망을 가져다 쓰기만 하면 됩니다. 중요한 점은 내가 가진 데이터에 가장 적합한 모델을 선택하는 것입니다. 이후에는 앞서 배운 전이 학습을 사용하여 약간의 튜닝만 진행하면 됩니다.