

9 - b 잔차 연결

모듈화, 계층화 재사용을 잘 활용

그래디언트 소실 문제로 인해 층을 쌓는 것에 한계가 있다. 이로 인해 잔차 연결이 탄생

1. Output에 전의 Input을 더해서 소실 문제를 방지한다. (잔차 연결)
2. 일부를 제거하여 어디에서 성능 향상이 오는지 확인 (절제 연구)

잔차 연결

이전 입력에 담긴 정보를 유지하기 위해 Output에 Input을 더한다.

잔차 연결 의사 코드

```
x = ... # 입력 텐서
residual = x # 원본 입력값을 따로 저장 (잔차)
x = block(X) # x는 입력 값에 대한 output이 됨
x = add([x, residual]) # output에 원본 입력값을 더한다.
```

output과 input을 더할 때 shape을 맞춰주는 것이 관건

필터 개수가 변경되는 잔차 블록

```
# 필터 개수가 변경되는 잔차 블록
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs) # 1.
residual = x # 2.

x = layers.Conv2D(64, 3, activation="relu", padding="same")(x) # 3.
residual = layers.Conv2D(64, 1)(residual) # 4.
x = layers.add([x, residual]) # 5.
```

1. 1번 코드 수행 후 x 는 (None, 30, 30, 32) shape
2. residual 또한 (None, 30, 30, 32) shape
3. 3번 코드 수행 후 x 는 (None, 30, 30, 64) shape이 된다.
4. x (None, 30, 30, 64)와 residual (None, 30, 30, 32) 를 맞추기 위해 필터만 64 커널은 1
5. 크기를 맞췄으니 add해준다.

최대 풀링 (MaxPooling2D)를 가진 잔차 블록

```
# 최대 풀링 층을 가진 잔차 블록
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs) # 1.
residual = x # 2.
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x) # 3.
x = layers.MaxPooling2D(2, padding="same")(x) # 4.
residual = layers.Conv2D(64, 1, strides=2)(residual) # 5.
x = layers.add([x, residual]) # 6.
```

1. x 는 (None, 30, 30, 32) shape
2. residual 또한 (None, 30, 30, 32) shape
3. x 는 (None, 30, 30, 64) shape - same padding으로 인해 가로 세로는 안변함
4. x 는 (None, 15, 15, 64) shape
5. 최대 풀링으로 다운 샘플링된 x 와 shape을 맞추기 위해서 Conv2D에 strides=2를 사용
residual (None, 15, 15, 64)

6. 잔차연결

3,4 에서 padding = "same"을 하는 이유는 패딩으로 인한 다운 샘플링을 방지하기 위함이다.

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Rescaling(1./255)(inputs)
```

잔차 연결을 가진 합성곱 블록을 적용하는 유틸리티 함수. 선택적으로 최대 풀링을 추가합니다.

```
def residual_block(x, filters, pooling=False):
    residual = x
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    if pooling:
        x = layers.MaxPooling2D(2, padding="same")(x)
        residual = layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
        residual = layers.Conv2D(filters, 1)(residual)
    x = layers.add([x, residual])
    return x
```

Handwritten notes: (32, 32, 3) → (16, 16, 32) → (8, 8, 64) → (8, 8, 128). Pooling operation is noted as "최대 풀링을 사용하면 잔차를 원하는 크기로 투영하기 위해 스트라이드 합성곱을 추가합니다."

● 잔차 연결

```
residual = layers.Conv2D(filters, 1)(residual)
x = layers.add([x, residual])
return x
```

Handwritten notes: (8, 8, 128). Pooling operation is noted as "최대 풀링을 사용하지 않으면 채널 수가 바뀔 경우에만 잔차를 투영합니다."

① x = residual_block(x, filters=32, pooling=True) ----- 첫 번째 블록 (32, 32, 32)

② x = residual_block(x, filters=64, pooling=True) ----- 두 번째 블록. 블록마다 필터 개수가 증가합니다. (16, 16, 32)

③ x = residual_block(x, filters=128, pooling=False) ----- (8, 8, 64)

마지막 블록은 바로 다음에 전역 평균 풀링(global average pooling)을 사용하기 때문에 최대 풀링이 필요하지 않습니다.

```
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

①의 결과 (16, 16, 32)

정규화

샘플들을 균일하게 만드는 방법

1. data에서 평균을 빼서 데이터를 원점에 맞춘다.
2. data를 표준 편차로 나누어 분산을 1로 만든다.

중요한 것은 활성화 층 이전에 배치 정규화 층을 놓는 것이 좋다.

깊이별 분리 합성곱

깊이별 분리 합성곱 층에서는 채널 별로 따로따로 공간 방향의 합성곱을 수행한다.

각 채널마다 독립적으로 수행하여

각 채널마다 있는 두드러진 특성을 극대화한다.

즉, 채널 마다 독립적으로 공간 방향의 합성곱을 실행한다.

```
x = layers.SeparableConv2D(size, 3, padding = "same", use_bias=False)(x)
```

