14.4.7 ~ 14.5
SENet ~ 케라스를 사용해 ResNet-34 CNN 구현하기

2021-04-18 백관구

14.4.7. SENet

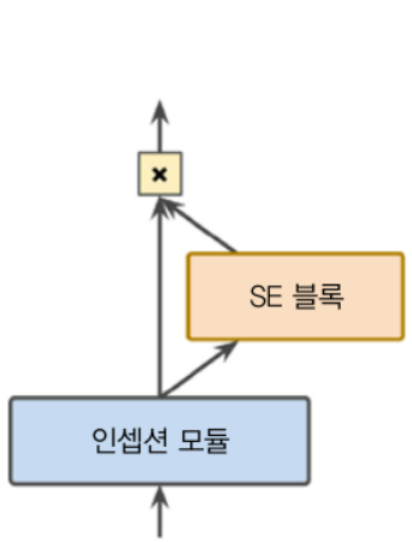
- SENet: Squeeze-and-Excitatioin Network (2019)
- 논문 링크: https://arxiv.org/pdf/1709.01507.pdf
- ILSVRC (Imagenet Large Scale Visual Recognition Challenge; 이미지넷 이미지 인식대회) 2017 대회 우승자
- 두 가지 버전: 핵심은 SE 블록
    1. SE-Inception
    2. SE-ResNet

Squeeze-and-Excitation Networks

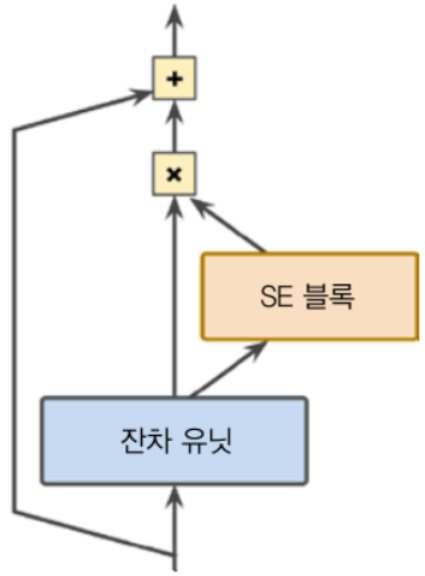
Jie Hu[0000−0002−5150−1003] Li Shen[0000−0002−2283−4976] Samuel Albanie[0000−0001−9736−5134]
Gang Sun[0000−0001−6913−6799] Enhua Wu[0000−0002−2174−1428]

Abstract—The central building block of convolutional neural networks (CNNs) is the convolution operator, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer. A broad range of prior research has investigated the spatial component of this relationship, seeking to strengthen the representational power of a CNN by enhancing the quality of spatial encodings throughout its feature hierarchy. In this work, we focus instead on the channel relationship and propose a novel architectural unit, which we term the "Squeeze-and-Excitation" (SE) block, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. We further demonstrate that SE blocks bring significant improvements in performance for existing state-of-the-art CNNs at slight additional computational cost. Squeeze-and-Excitation Networks formed the foundation of our ILSVRC 2017 classification submission which won first place and reduced the top-5 error to 2.251%, surpassing the winning entry of 2016 by a relative improvement of ∼25%. Models and code are available at https://github.com/hujie-frank/SENet.

Index Terms—Squeeze-and-Excitation, Image representations, Attention, Convolutional Neural Networks.



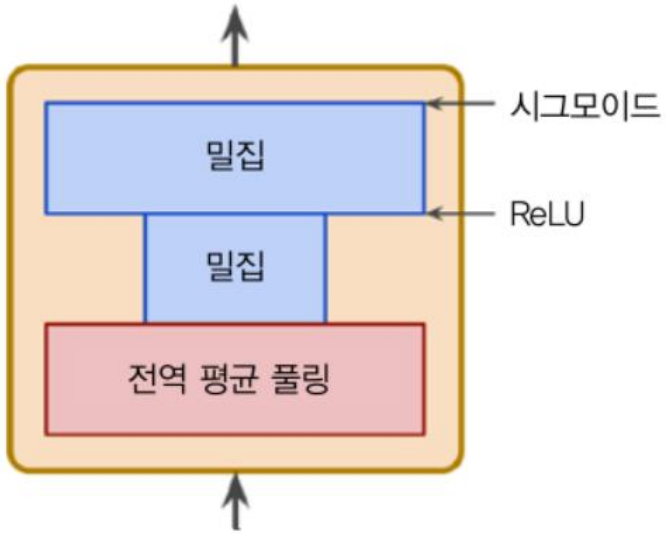
<SE-Inception>



<SE-ResNet>

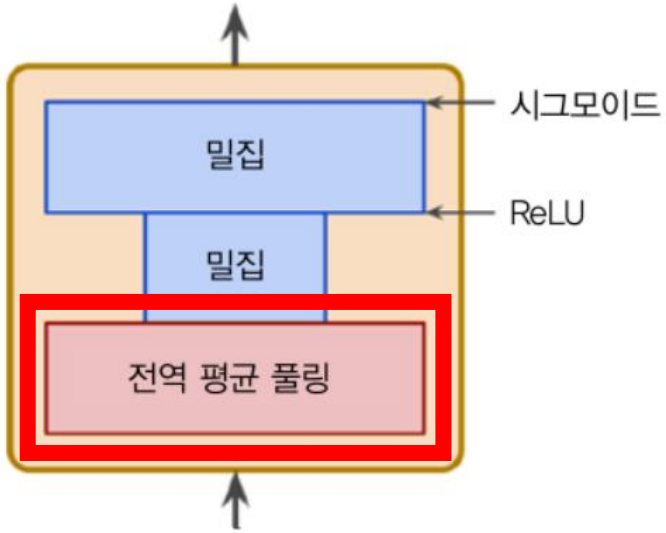
14.4.7. SENet

- SE 블록
- 구조: 전역 평균 풀링(Global Mean Pooling) → 압축(Dense+ReLU) → 출력(Dense+Sigmoid)



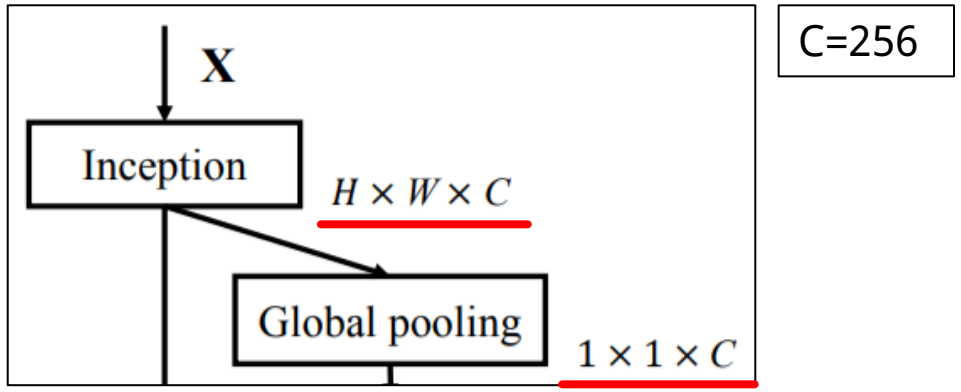
14.4.7. SENet

- SE 블록
- 구조: **전역 평균 풀링**(Global Mean Pooling) → 압축(Dense+ReLU) → 출력(Dense+Sigmoid)



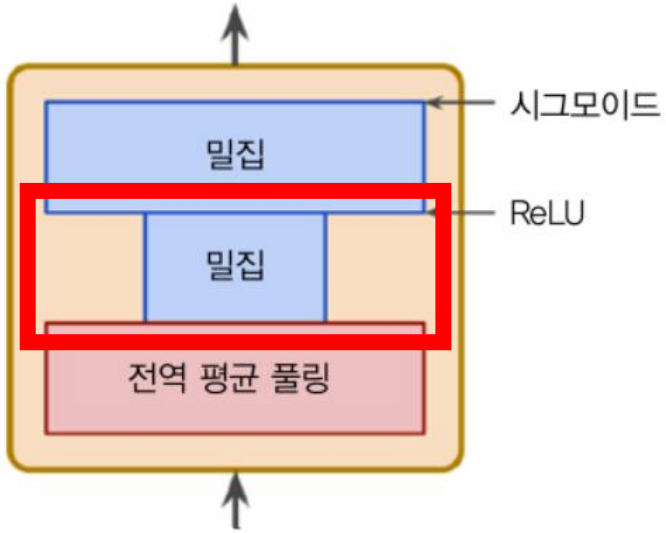
1. 전역 평균 풀링: 각 **특성** 맵(Channel)의 **평균 활성화 값**(노드 가중치) 계산

(예시) 256 채널의 특성 맵 → 전반적인 활성화 수준을 나타내는 256개의 숫자 벡터



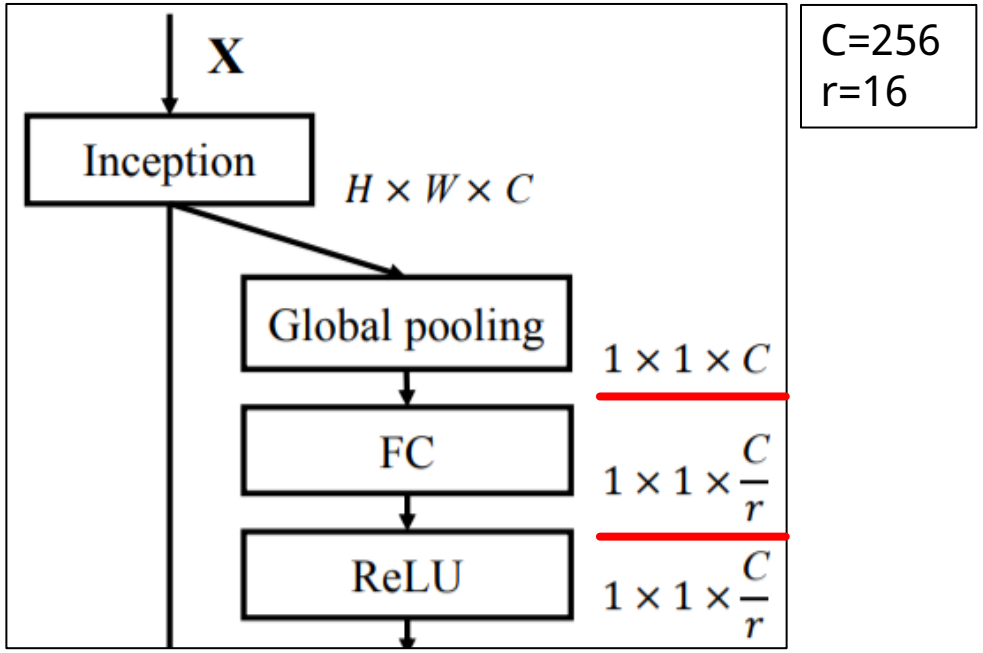
14.4.7. SENet

- SE 블록
- 구조: 전역 평균 풀링(Global Mean Pooling) → 압축(Dense+ReLU) → 출력(Dense+Sigmoid)



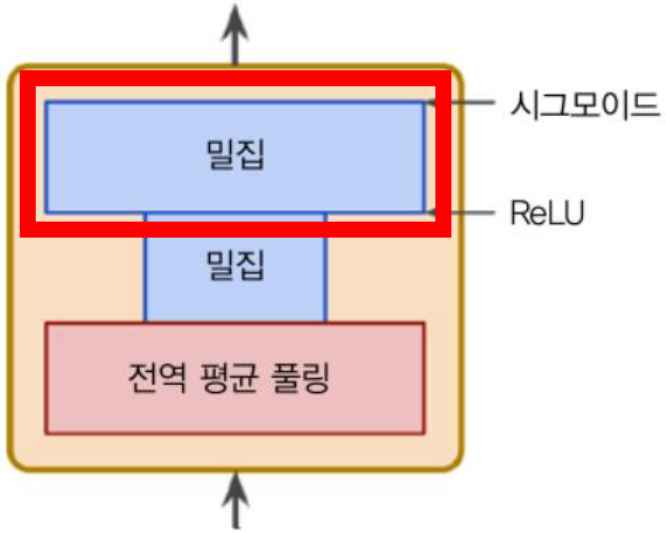
2. 압축: 특성 조합의 일반적인 표현을 학습 (17장에서 다룰 오토인코더와 관련)

(예시) 256개의 숫자 벡터 → 16개의 숫자 벡터



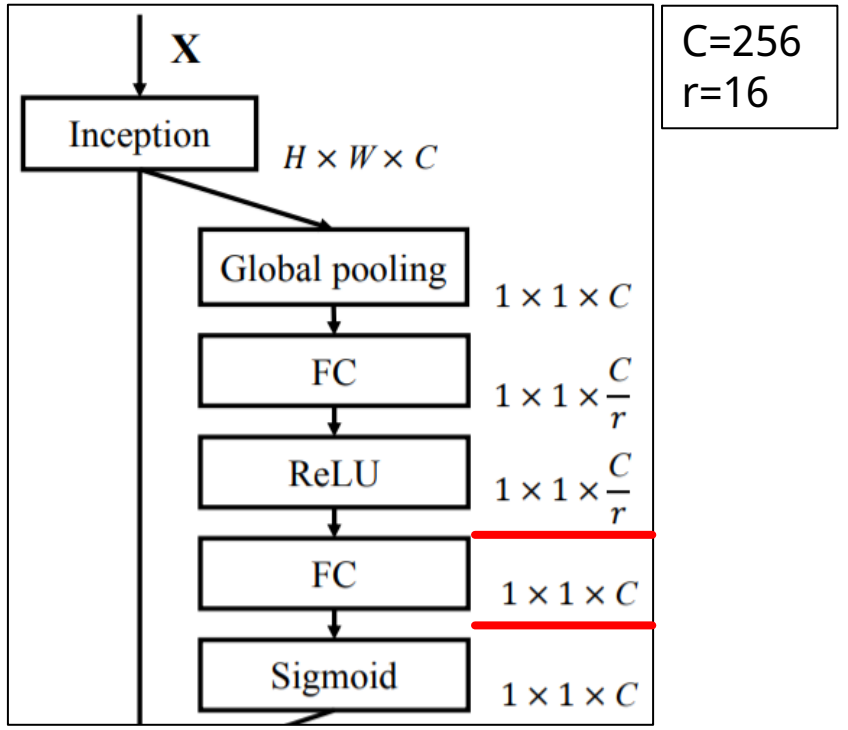
14.4.7. SENet

- SE 블록
- 구조: 전역 평균 풀링(Global Mean Pooling) → 압축(Dense+ReLU) → 출력(Dense+Sigmoid)



3. 출력: Sigmoid 함수를 적용함으로써 0 ~ 1 사이의 값으로 해당 특성의 중요도를 표현

(예시) 16개의 숫자 벡터 → 256개의 숫자(0 ~ 1) 벡터



14.4.7. SENet

1. 전역 평균 풀링: 각 특성 맵(Channel)의 평균 활성화 값(노드 가중치) 계산

    ➔ 256 채널의 특성 맵 → 256개 필터의 전반적인 활성화 수준을 나타내는 256개의 숫자 벡터

2. 압축: 특성 조합의 일반적인 표현을 학습 (17장에서 다룰 오토인코더와 관련)

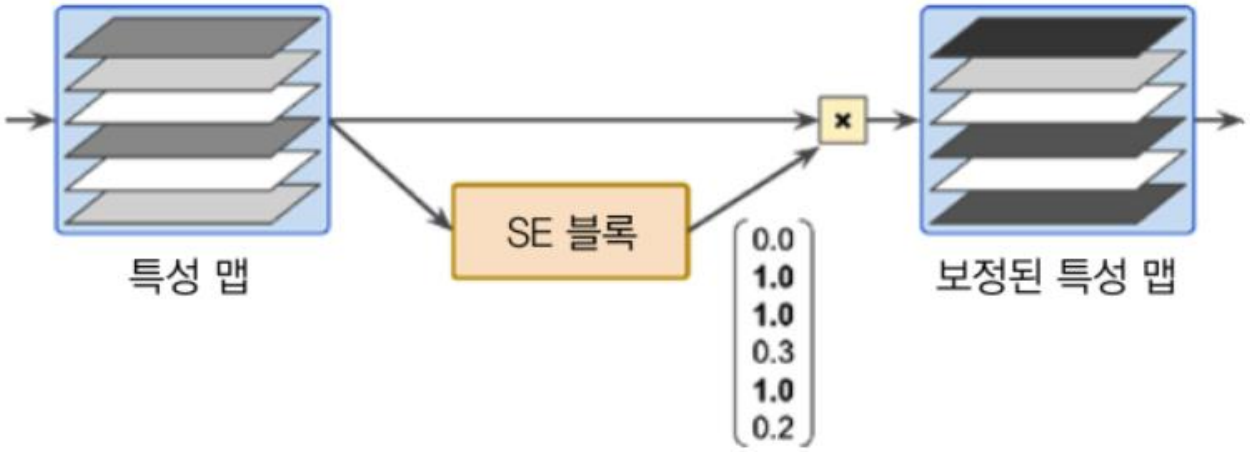
    ➔ 256개의 숫자 벡터 → 16개의 숫자 벡터

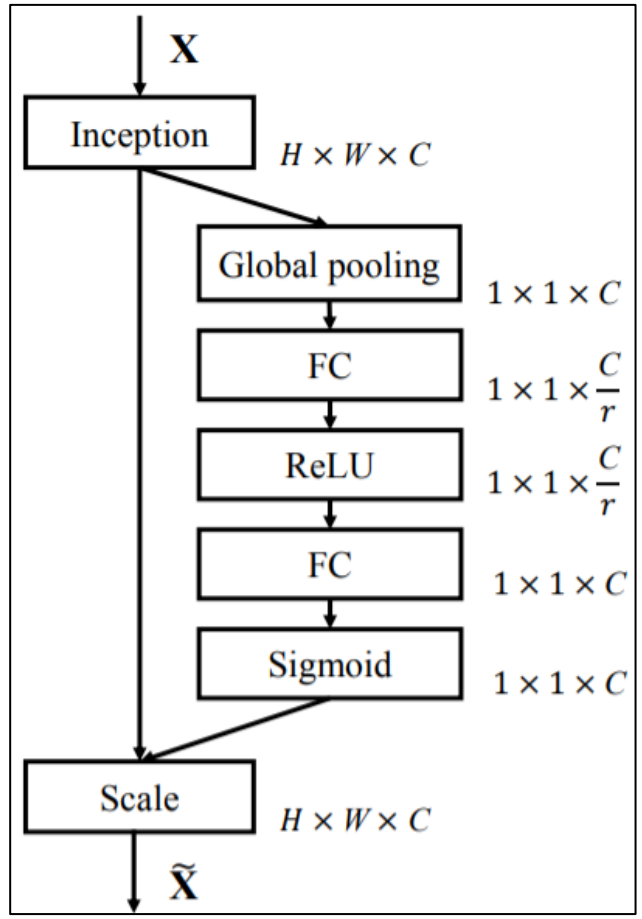
3. 출력: Sigmoid 함수를 적용함으로써 0 ~ 1 사이의 값으로 해당 특성의 중요도를 표현

    ➔ 16개의 숫자 벡터 → 256개의 숫자(0 ~ 1) 벡터

4. 특성 맵(256 채널) X 256개의 숫자 벡터 ➔ 보정된 특성 맵

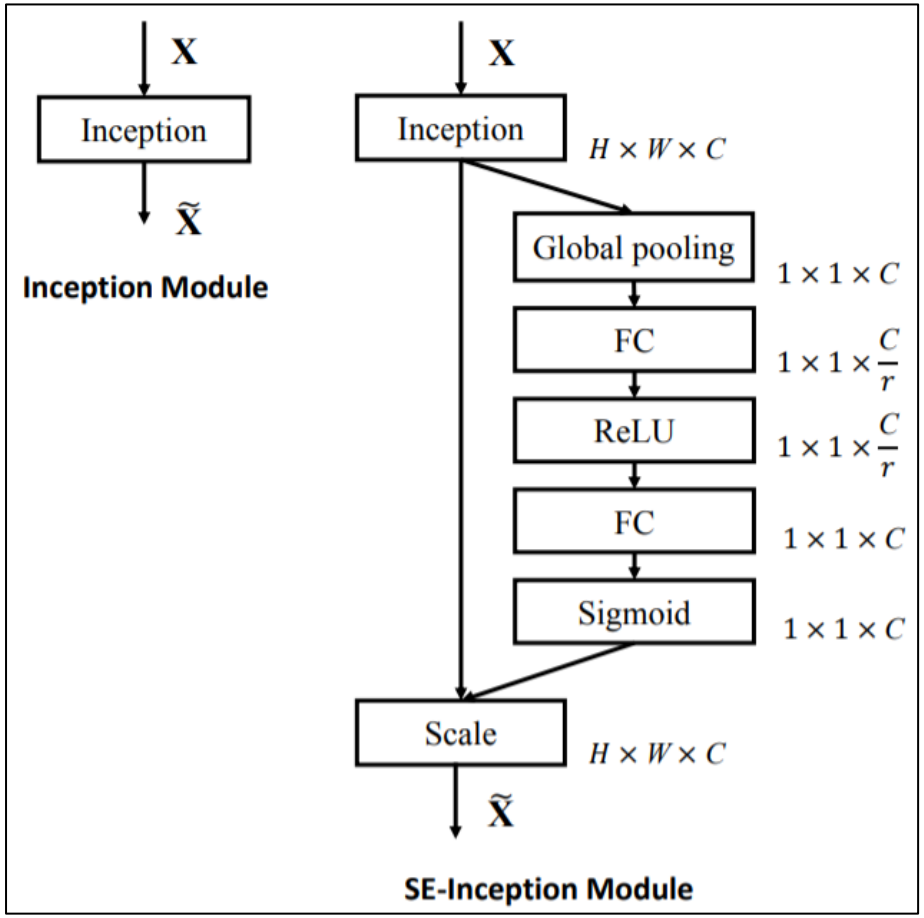
- 역할: 관련 없는 특성 맵(Channel-wise feature)의 값을 줄임

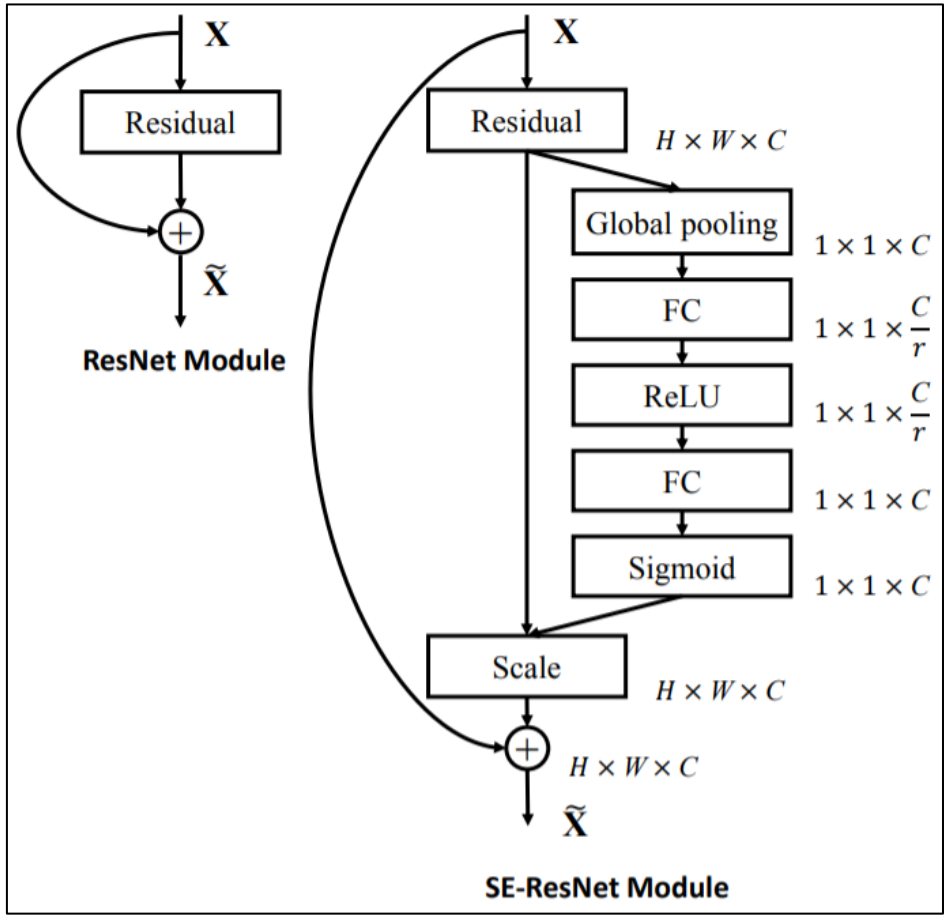




14.4.7. SENet

- SE-Inception, SE-ResNet 논문 그림





14.4.7. SENet

- SE 블록을 활용한 응용 구조

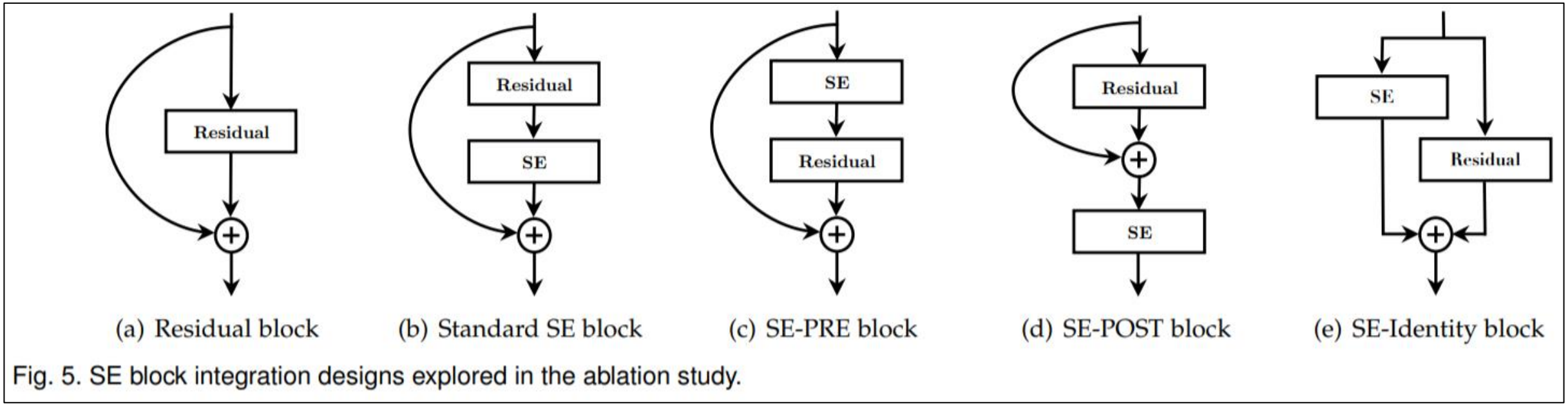


Fig. 5. SE block integration designs explored in the ablation study.

14.5. 케라스를 사용해 ResNet-34 CNN 구현하기

```
class ResidualUnit(keras.layers.Layer):
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = keras.activations.get(activation)
        self.main_layers = [
            keras.layers.Conv2D(filters, 3, strides=strides,
                                padding="same", use_bias=False),
            keras.layers.BatchNormalization(),
            self.activation,
            keras.layers.Conv2D(filters, 3, strides=1,
                                padding="same", use_bias=False),
            keras.layers.BatchNormalization()]
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [
                keras.layers.Conv2D(filters, 1, strides=strides,
                                    padding="same", use_bias=False),
                keras.layers.BatchNormalization()]

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```

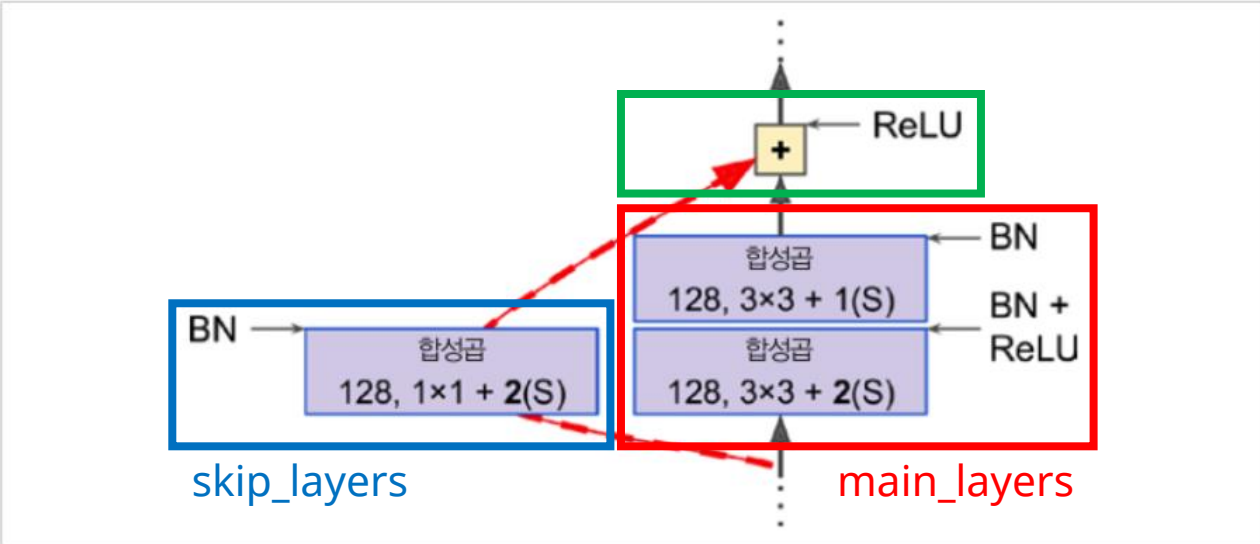


그림 14-18 특성 맵의 크기와 깊이가 바뀔 때 스킵 연결

- __init__: 필요한 층 생성
- call: 생성된 층을 적용하여 작동하게 함

14.5. 케라스를 사용해 ResNet-34 CNN 구현하기

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(64, 7, strides=2, input_shape=[224, 224, 3],
                              padding="same", use_bias=False))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation("relu"))
model.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="same"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    model.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
model.add(keras.layers.GlobalAvgPool2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation="softmax"))
```

layer name	output size	18-layer	34-layer	50-layer	101-layer
conv1	112×112			7×7, 64, stride 2	
conv2_x	56×56			3×3 max pool, stride 2	
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax			
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9