5주차 스터디

딥러닝 CNN 완벽 가이드 – 세션9, 10

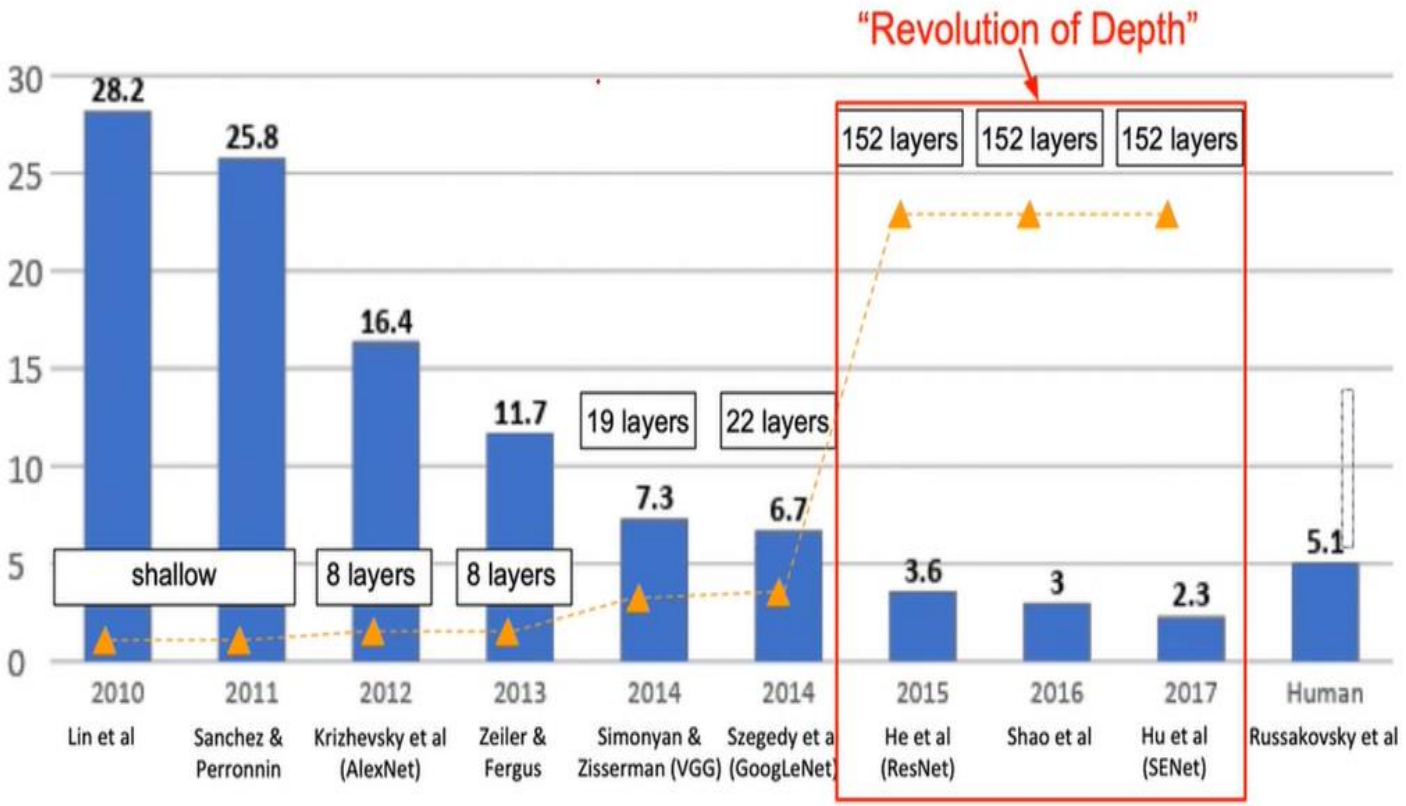
세션9 – Advanced CNN 모델 파헤치기 - AlexNet, VGGNet, GoogLeNet

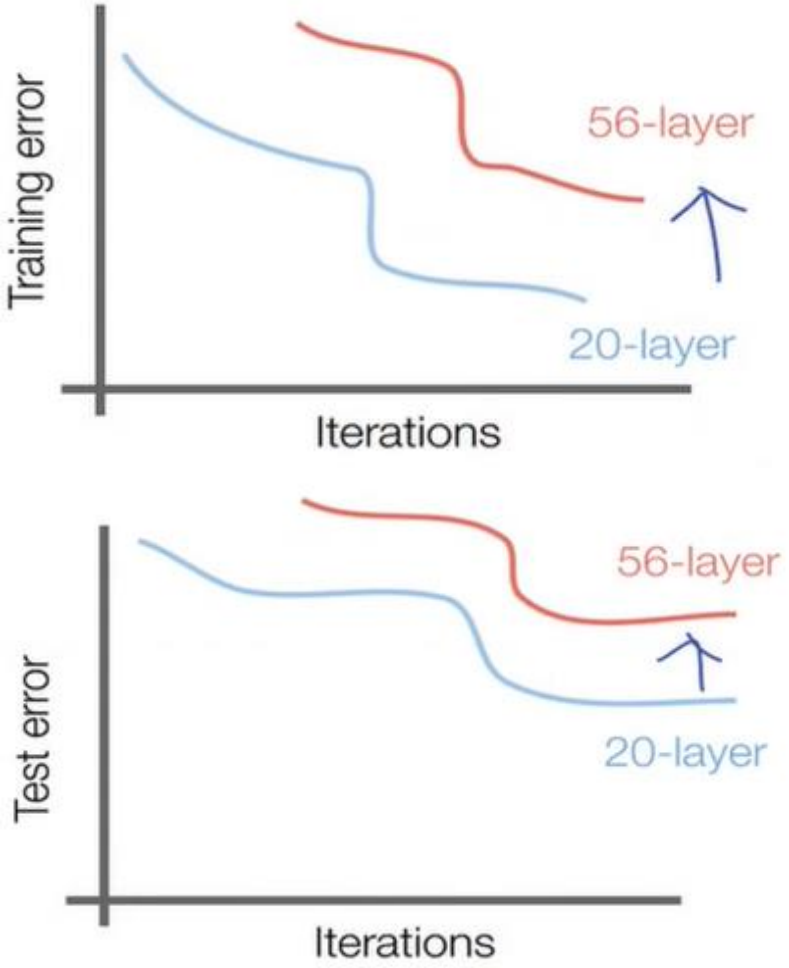
세션10 - Advanced CNN 모델 파헤치기 - ResNet 상세와 EfficientNet 개요

발표자 : 이현진, 임수진

3. ResNet

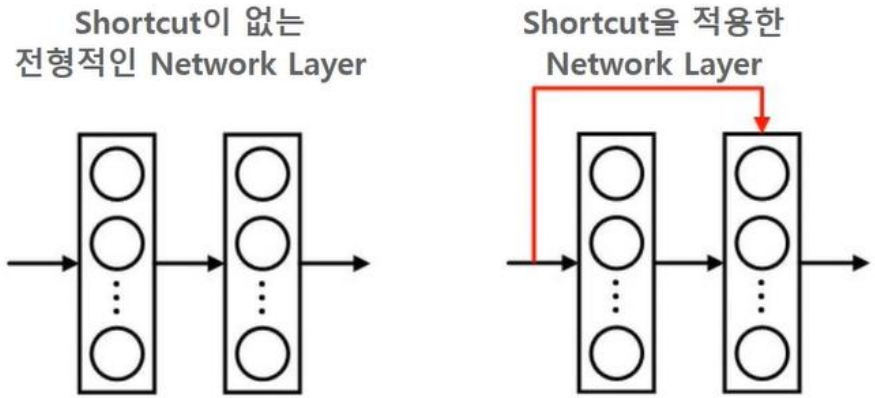
- 마이크로소프트에서 개발한 모델
- VGG이후 더 깊은 네트워크에 대한 연구 증가
- 깊은 층임에도 모델 오차율이 급격히 감소하는 등 안정적인 성능



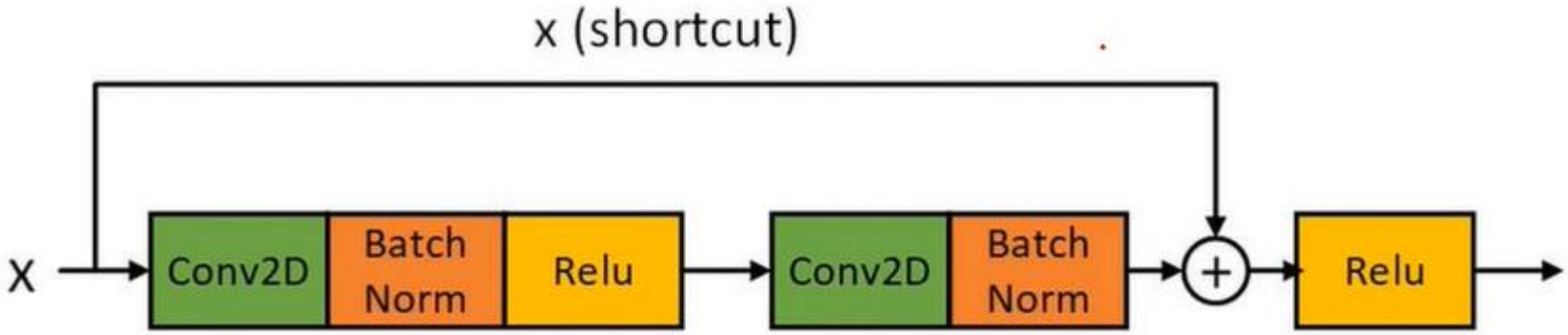


3. ResNet

- Shortcut : 이전 layer의 출력값을 conv layer로 거치지 않고 그대로 전달하는 구성

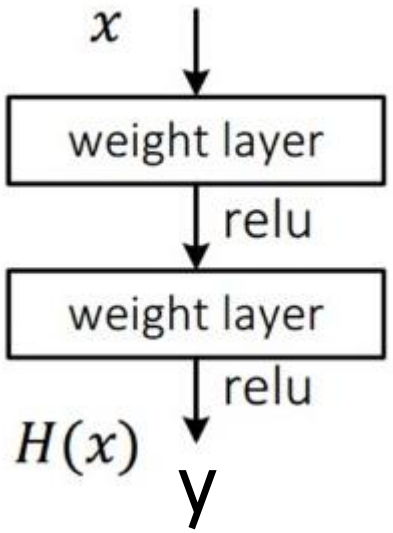


- Identity block : 'input값이 conv layer 거친 것 + 동일한 input값이 shourtcut으로 전달된 것' 으로 relu 적용

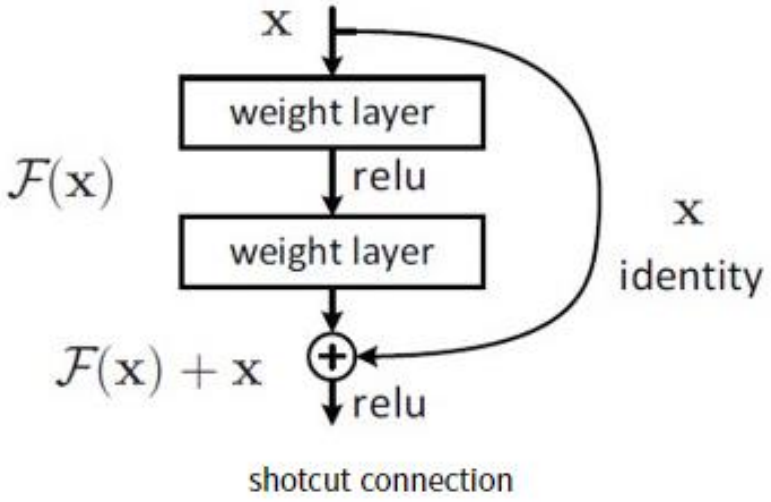


3. ResNet

- 기존 : x 를 타겟값 y로 mapping하는 함수 H(x)찾기 => H(x) – y 최소화 하는 학습
- X에 대한 타겟값 y는 x를 대변하는 것 => y와 x의 의미가 같도록 mapping => **H(x) – x 최소화 하는 학습**
  - Ex) 강아지 사진 pixel값이 input(x)이면 강아지 label인 1로 y를 mapping하는 것이 아니라 강아지 사진의 pixel값인 x로 y를 mapping하는 것
- F(x) = H(x) – x : 잔차 => 잔차 학습 Residual learning
- 기존 방식은 gradient vanishing 문제가 해결되지 않아 **네트워크는 0이 되도록 학습시키고 마지막에 x를 더한 것**
- x는 미분값이 1이기 때문에 각 layer마다 최소 gradien를 1로 갖도록 해서 gradient vanishing문제 해결



H(x) = y



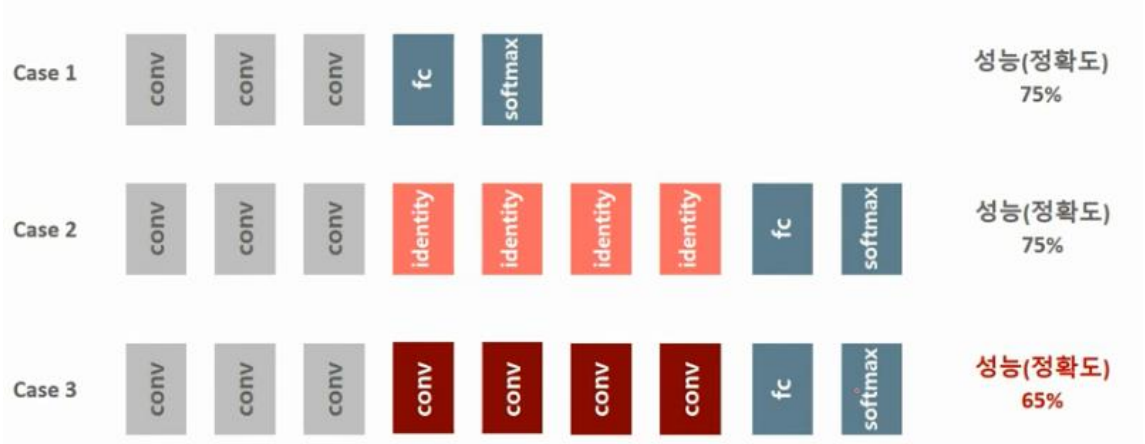
H(x) = F(x) + X

특징

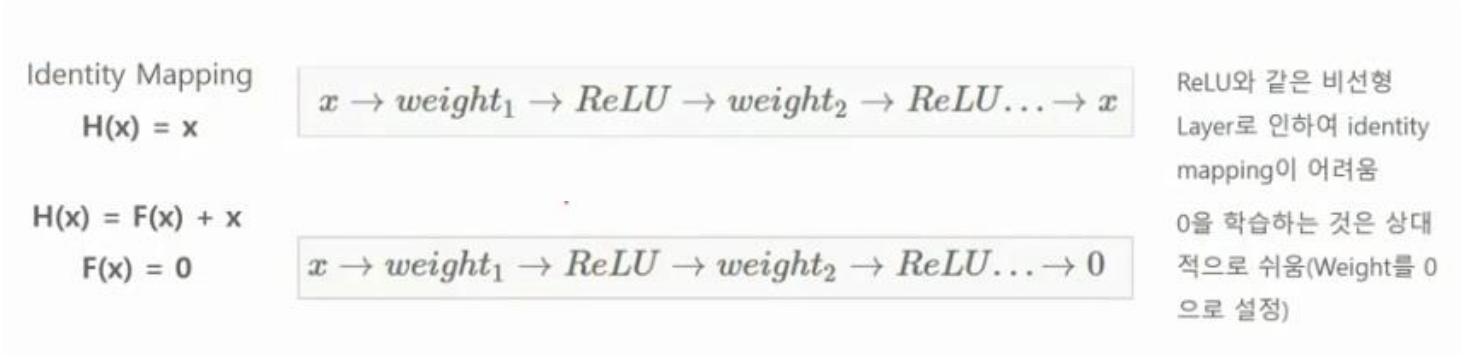
1. x가 그대로 skip connection이 되어 연산 증가 없다
2. F(x)가 몇 개 Layer포함할지 선택 가능
3. X와 H(x)의 차원 동일해야 한다.

3. ResNet

- Case2의 identity layer로 x가 들어오면 x가 출력되는 층을 쓴 것보다 Case3의 convolution을 사용한 모델 성능이 더 낮음
- Identity layer을 사용하는 것이 적어도 모델 성능을 더 떨어뜨리지는 않을 테니 input 값이 그대로 출력되는 층 생성하기



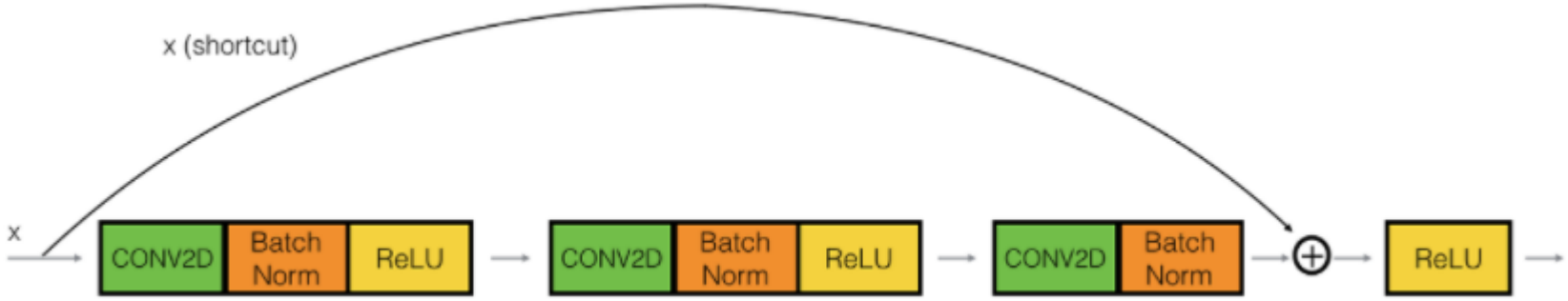
- Residual learning을 하는 이유는 Identity mapping이 생각보다 어려움



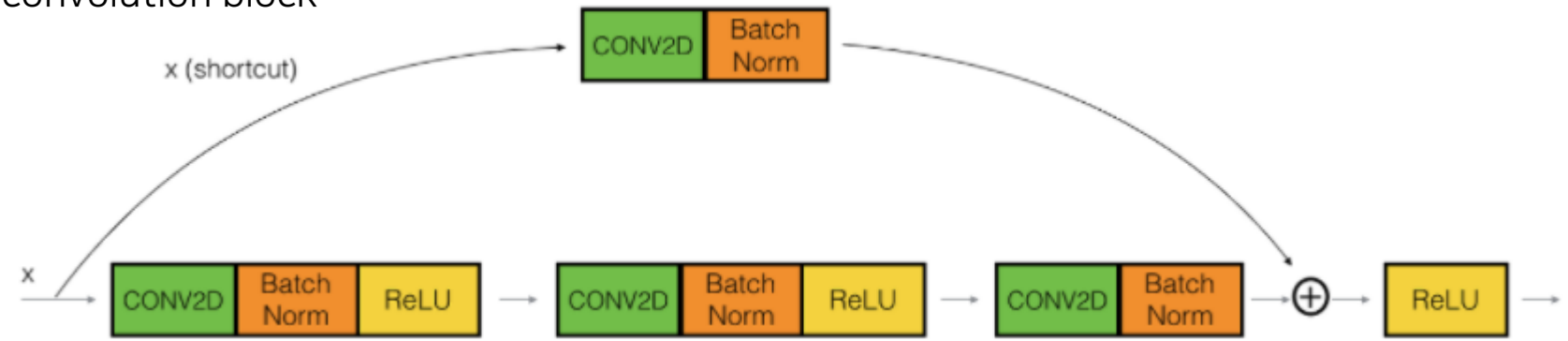
3. ResNet

- residual learning 사용

  - Identity block

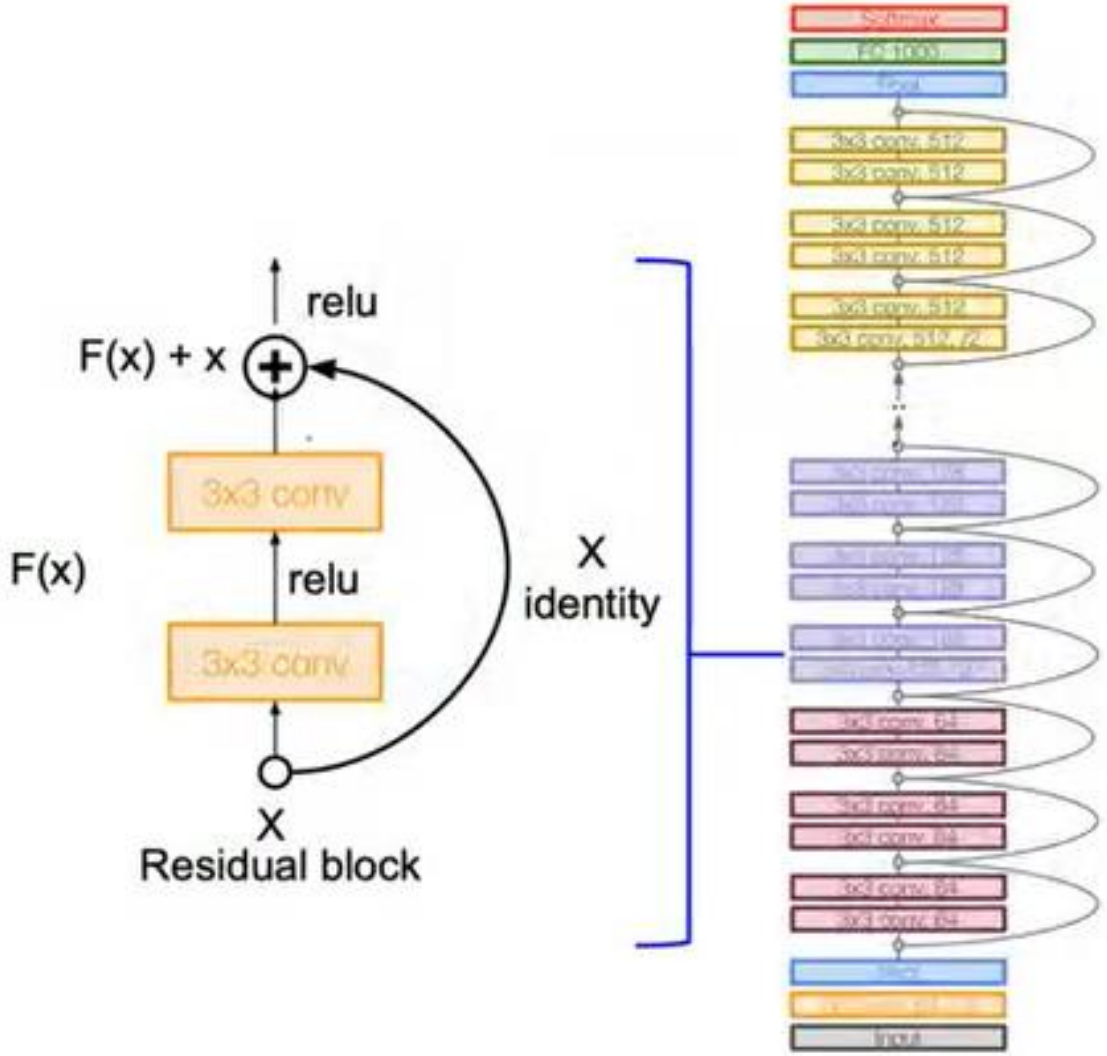


  - convolution block

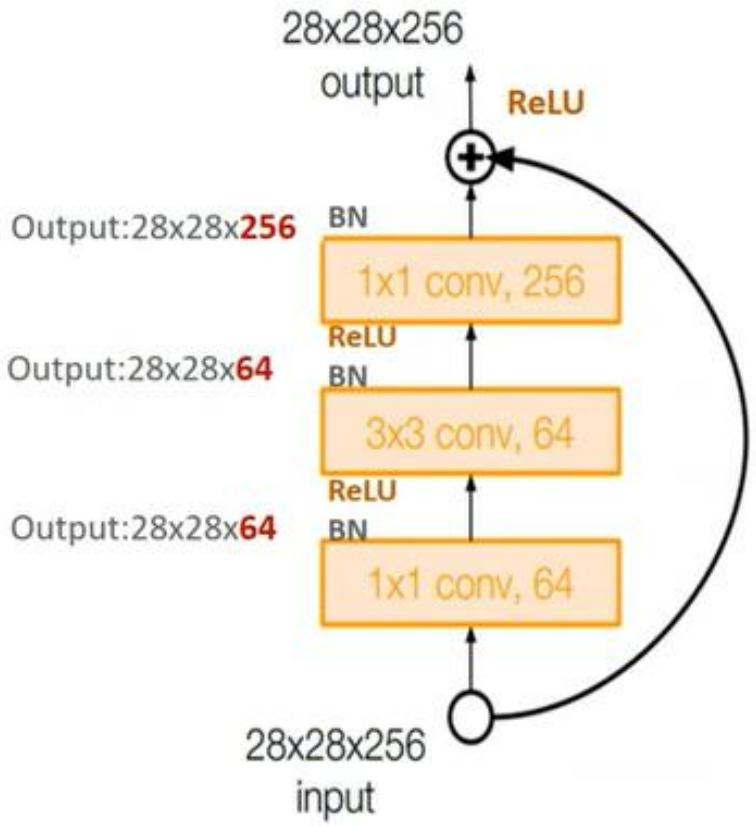


3. ResNet

- 3x3 Convolution 적용
- Residual block 내에서 CNN커널은 동일한 크기와 Depth유지
- Feature map output size를 절반으로 줄이면 layer의 연산량 보존 법칙으로 채널 수가 2배
- Feature map의 크기 줄일 때는 pooling 대신에 stride로 조정
- 해당 구조는 res34이고 res50 이상부터는 convolution이 달라진다



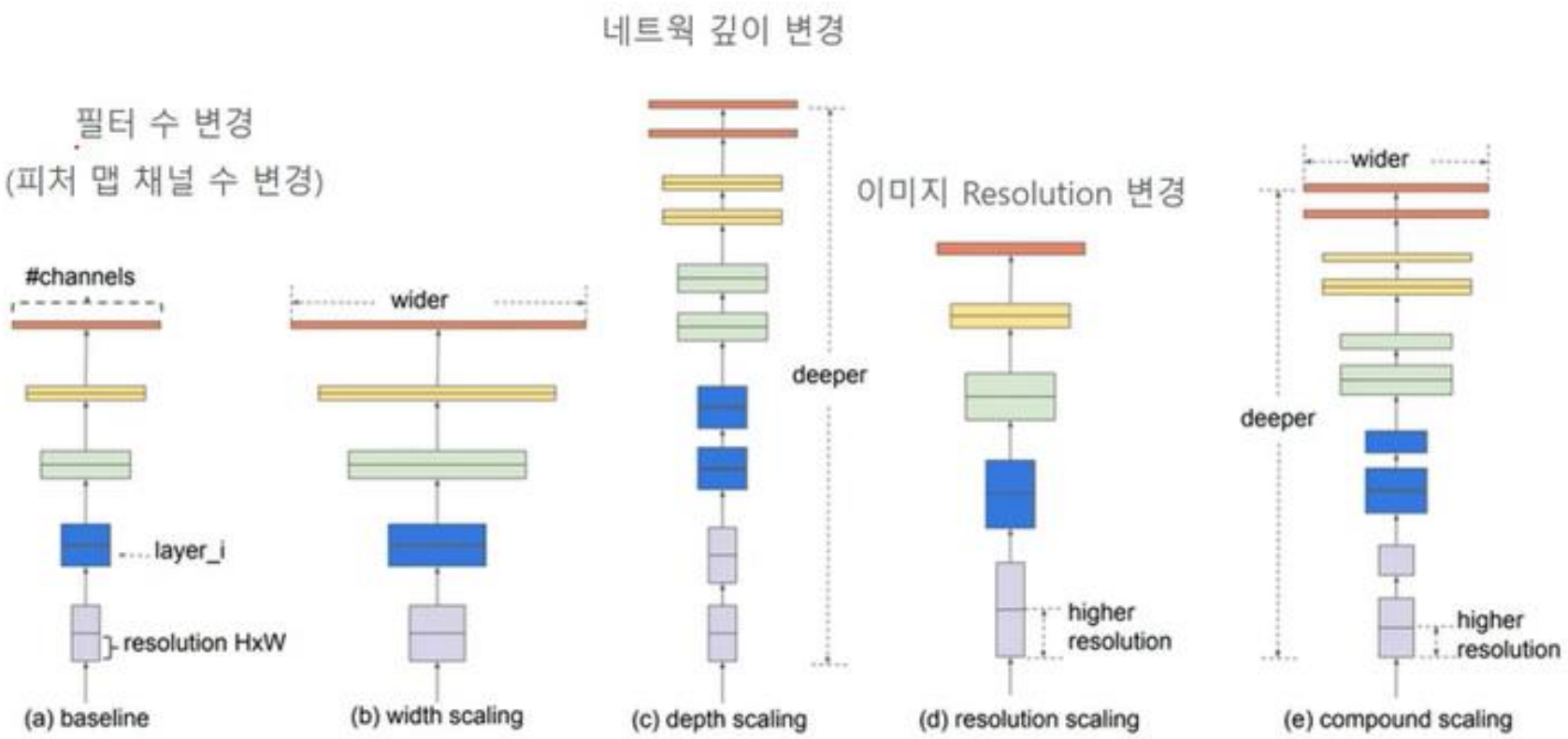
3. ResNet



- pooling 빼고 층 50층 이상 가진 resnet50, resnet101, resnet152
- 1x1으로 차원 압축해서 Bottleneck Layer이용하여 연산량과 파라미터 개수 줄인 후 3x3 적용하고 다시 1x1 적용
- Shortcut값과의 연산 위해 input이랑 동일한 크기 맞추기

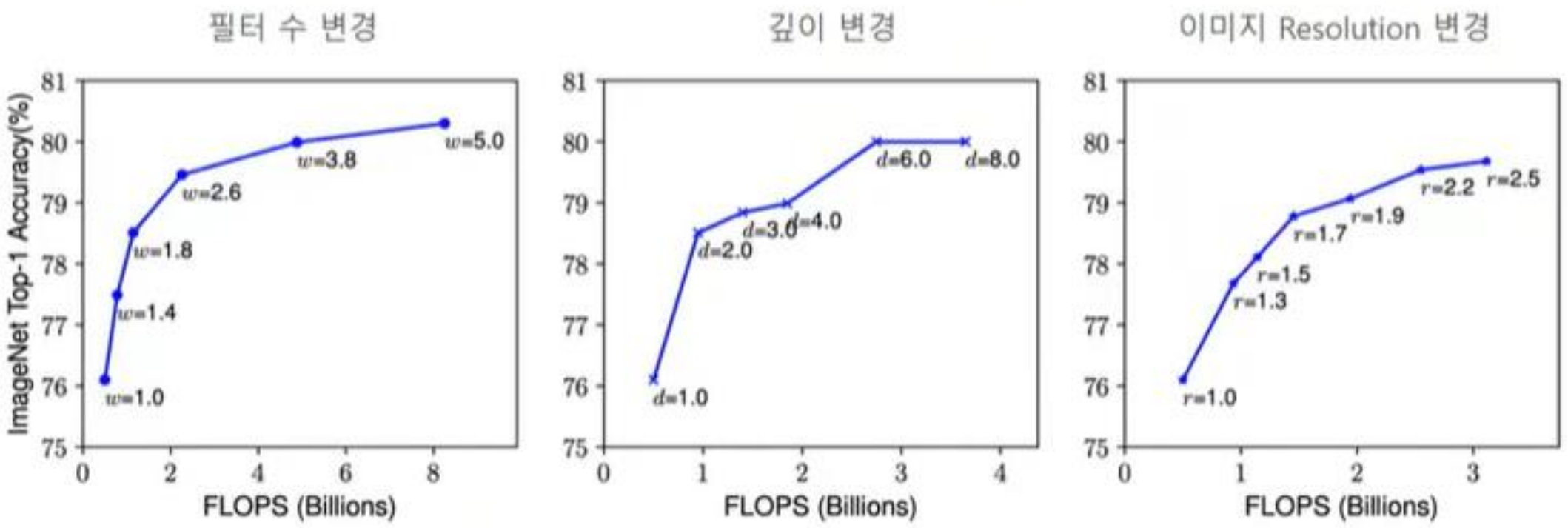
4. EfficientNet

- 네트워크 깊이, 필터 수, 이미지 해상도들의 최적의 조합을 auto-ml로 찾은 모델
- compound scaling방법을 써서 기존보다 훨씬 적은 파라미터 수로 더 좋은 성능을 낸 것



4. EfficientNet

- 개별 scaling요소에 따른 성능 향상 테스트
- 약 80%정확도에서 개별 scaling요소를 증가시켜도 성능 향상 어려움



4. EfficientNet

- 3가지 scaling factor를 동시에 고려하는 compound scaling 적용
- Depth, width, resolution에 따른 FLOPS(부동소수점 수)(연산량) 변화 기반 도출
- Depth는 2배가 되면 FLOPS도 거의 2배로 선형 관계
- Width, resolution은 2배가 되면 FLOPS는 4배가 되어서(면적이기 때문) 제곱!

$$depth : d = \alpha^{\phi}$$

$$width : w = \beta^{\phi}$$

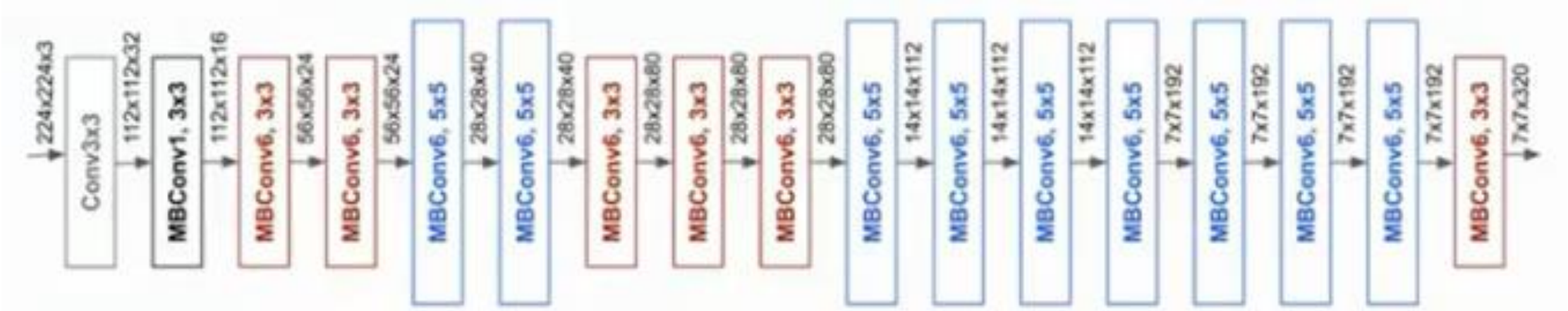
$$resolution : r = \gamma^{\phi}$$

$$\alpha \cdot \beta^2 \cdot \gamma^2 \simeq 2 \qquad (\alpha \geq 1, \beta \geq 1, \gamma \geq 1)$$

최초에는 φ를 **1로** 고정하고 grid search 기반으로 α, β, γ의 최적 값을 찾아냄. EfficientNetB0의 경우 α =1.2, β=1.1, γ=1.15
다음으로 α, β, γ을 **고정하고** φ을 증가 시켜가면서 EfficientB1~ B7까지 Scale up 구성

4. EfficientNet

- 그림은 EfficientNet-B0
- B0모델에서 B1~B7으로 갈수록 depth, width증가시켜서 모델 생성



4. EfficientNet

- mobileNetV1

- Depthwise Separable Convolution =
  DeopthwiseConv + Pointwise Conv

  - DepthwiseConv :

    모든 채널에 한 번에 Conv 연산 적용하는 것 대신

    이미지나 피처맵을 각채널별로 쪼개서 적용(RGB)

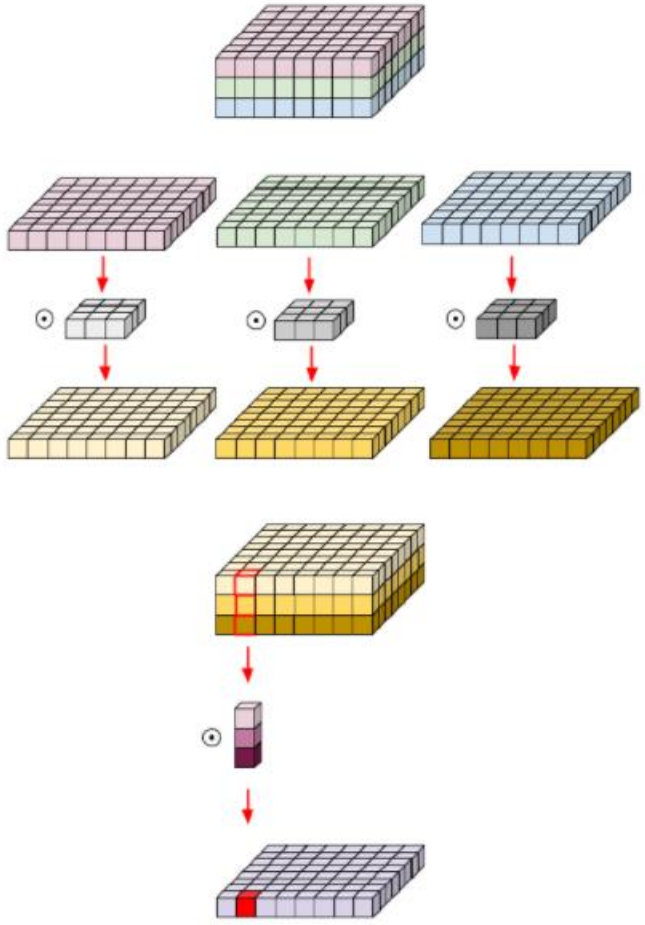
    input의 특징을 그냥 Conv보다 더 적은 양의 파라미터로!

  - PointConv :

    필터 크기가 1로 고정되어 있는 걸로

    여러 채널을 하나의 채널로 합치기

    채널 수 조절 -> 차원 감소 -> 연산량 감소

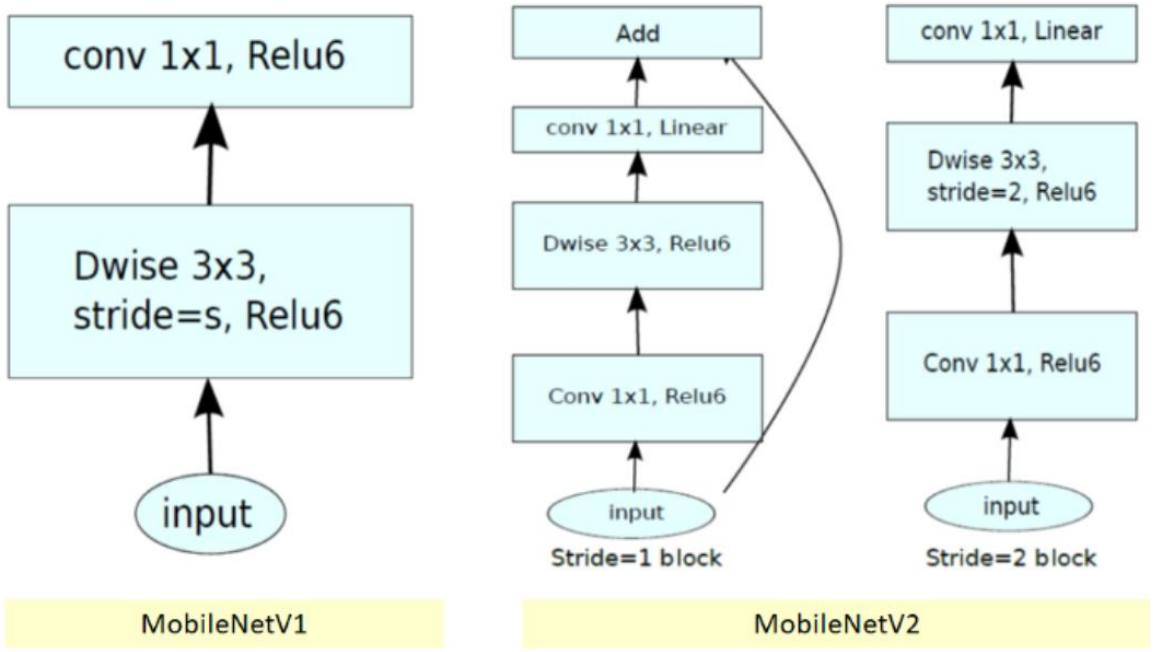


Depthwise Separable Convolution

4. EfficientNet

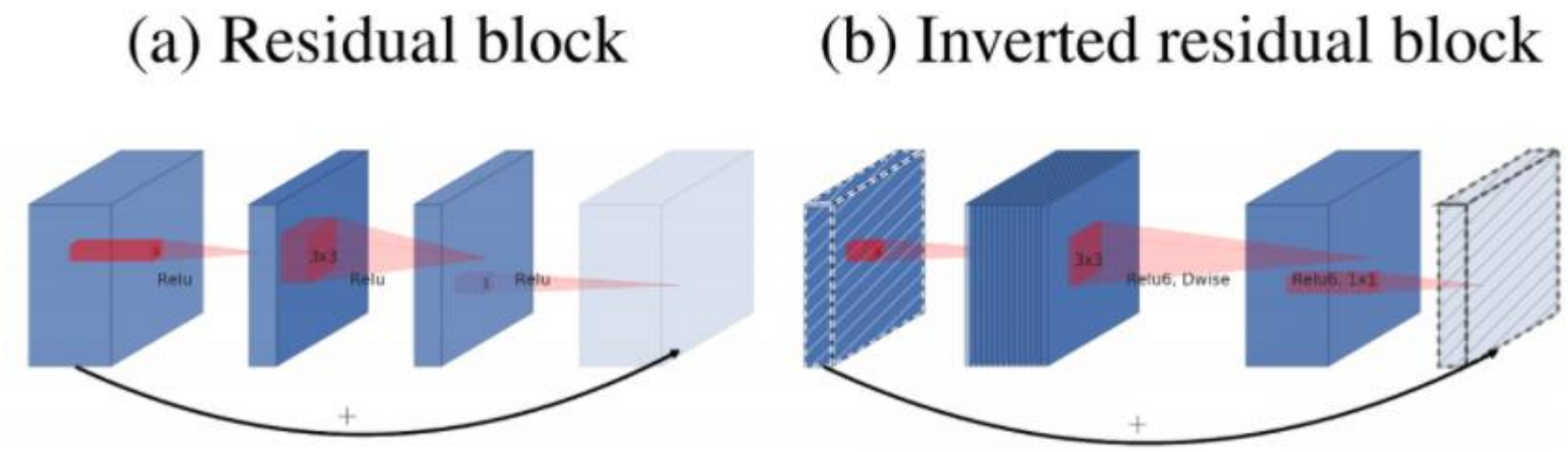
- mobileNetV2

- Stride = 1 , Stride = 2 두 가지
- 첫 번째 layer : pointwise convolution에 ReLU를 적용으로 시작
- 두 번째에 depthwise convolution을 적용



4. EfficientNet

- Inverted residual block
  - 채널증가 -> 학습 -> 채널감소 형식
  - 버려지는 피처들까지 고려
  - block에서 채널을 늘리기 위해서는 pointwise(1x1) convolution이 사용되었고
    경량화된 convolution 연산을 위해 depthwise separable convolution 사용
  - 기존의 mobilenetV2랑 유사



감사합니다