Deep Residual Learning for Image Recognition

He_Deep_Residual_Learning_CVPR_2016_paper.pdf

1. Introduction

논문에서 다루고 있는 주제가 무엇인지와 해당 주제의 필요성이 무엇인가

논문에서 제안하는 방법이 기존 방법의 문제점에 대응되도록 제안 되었는가

논문이 다루는 분야

- 딥러닝 기반 **이미지 인식(Image Recognition)**
- 특히, **심층 신경망(CNN)의 깊이가 성능에 미치는 영향**과 최적화 문제 해결

해당 task에서 기존 연구 한계점

- 깊은 네트워크(VGG, GoogLeNet 등)는 뛰어난 성능을 보였지만 층을 더 쌓으면 **기울기 소실/폭발** 문제가 해결된 이후에도 **Degradation Problem(층이 깊을수록 훈련/검증 정확도가 오히려 떨어짐)** 발생.
- 이는 단순히 과적합 문제가 아니라, 최적화가 제대로 되지 않는 근본적 한계.

논문의 contributions

- **Residual Learning Framework 제안**
  - 복잡한 함수 H(x)를 직접 학습하지 않고, 입력과 출력의 차이인 잔차 F(x)=H(x)−xF(x)를 학습.
  - 블록 출력은 y=F(x)+x 형태로, shortcut connection을 통해 입력을 직접 더함.
- **효율적 Shortcut Connections 설계**
  - 대부분은 파라미터 없는 항등 매핑(identity mapping) 사용 → 추가 연산 거의 없음.
  - 차원이 달라질 때만 projection(1×1 conv) 사용.
- **Extremely Deep Networks 성공적 학습**
  - ImageNet에서 152층까지 학습 성공 (당시 최심층).
  - CIFAR-10에서 1000층 이상의 네트워크도 학습 가능함을 입증.

2. Related Work

Introduction에서 언급한 기존 연구들에 대해 어떻게 서술하는가

제안 방법의 차별성을 어떻게 표현하고 있는가

- **기존 CNN (VGG, GoogLeNet 등):** 층을 깊게 쌓는 방식으로 성능 향상. 그러나 깊어질수록 최적화 문제 발생.

- **Highway Networks:** 게이트를 이용한 shortcut 연결 제안. 그러나 파라미터가 추가되고, 100층 이상에서는 뚜렷한 성능 개선이 없음.
- **ResNet의 차별성:**
  - 게이트가 아닌 **항등 shortcut**으로 단순하면서도 효과적.
  - 파라미터 증가 없음.
  - 매우 깊은 네트워크에서도 안정적인 최적화 가능.

3. 제안 방법론

Introduction에서 언급된 내용과 동일하게 작성되어 있는가

Introduction에서 언급한 제안 방법이 가지는 장점에 대한 근거가 있는가

제안 방법에 대한 설명이 구현 가능하도록 작성되어 있는가

Main Idea

- 층을 깊게 쌓으면 학습이 어려워지는 이유: 항등 함수(identity mapping)를 비선형 함수들의 조합으로 근사하기가 어렵기 때문.
- 이를 해결하기 위해 “항등 함수 + 작은 보정(잔차)” 형태로 학습을 단순화.

Contribution

- **Residual Block 설계:**
  - 기본 형태: $y=F(x,W)+x$
  - $F(x,W)$: Conv + BN + ReLU로 구성된 residual 함수
  - x: shortcut으로 직접 연결
- **Bottleneck 구조 제안:**
  - 깊이가 50, 101, 152층 이상이 될 때 효율성을 위해 1×1–3×3–1×1 구조 사용
  - 파라미터 수와 연산량 절약하면서 깊이를 크게 확장

4. 실험 및 결과

💡 Introduction에서 언급한 제안 방법의 장점을 검증하기 위한 실험이 있는가

Dataset

- **ImageNet 2012**: 1000 클래스, 128만 훈련 이미지, 5만 검증, 10만 테스트
- **CIFAR-10**: 10 클래스, 32×32 이미지

Baseline

- Plain Network (VGG 스타일, shortcut 없음)
- 비교: 동일한 파라미터 수와 깊이에서 residual block 유무 차이

결과

- **ImageNet**:
  - Plain 34-layer < Plain 18-layer (Degradation 발생)
  - ResNet-34 > ResNet-18 (Residual Learning으로 성능 반전)
  - ResNet-152: Top-5 error 4.49% (단일 모델), 당시 모든 앙상블 모델보다도 우수
- **CIFAR-10**:
  - Plain: 깊어질수록 학습 에러 증가
  - ResNet: 깊어질수록 성능 향상 (110층에서 6.43% error)
  - 1000층 이상도 학습 가능 → 단, 작은 데이터셋에서는 과적합 문제 발생
- **Object Detection/Segmentation**:
  - Faster R-CNN의 backbone을 VGG-16에서 ResNet-101로 교체 시, COCO에서 mAP 대폭 향상 (+6.0%p, 28% 상대 개선).
  - ILSVRC & COCO 2015 대회 1위 석권.

5. 결론 (배운점)

💡 연구의 의의 및 한계점, 본인이 생각하는 좋았던/아쉬웠던 점 (배운점)

의의:

- Residual Learning은 네트워크 깊이 확장의 근본적 장벽(Degradation Problem)을 해결.
- 단순하면서도 일반화 성능이 뛰어난 구조 → 이후 딥러닝의 표준이 됨.

한계점:

- 너무 깊은 네트워크(예: 1202층)는 작은 데이터셋에서 과적합 문제 발생.
- Residual 구조 자체는 최적화를 돕지만, 일반화 성능은 데이터 크기/정규화 기법에 여전히 의존.

Q: "SGD+backprop이 항등함수(identity mapping)도 제대로 못 배우는데, 어떻게 '항등함수 + 잔차'를 배우는 게 더 쉽다는 거지?"

A: Plain Network(그냥 층만 쌓은 네트워크)에서 항등 함수 y=x를 표현하려면, 각 층이 **아무 일도 하지 않도록** 매개변수를 딱 맞게 조정해야 함.

하지만 각 층은 보통 `Conv + BN + ReLU` 같은 **비선형 함수**.

비선형 조합으로 정확히 "입력 그대로 출력"을 만들려면 아주 미세한 조율이 필요하고, SGD가 이걸 찾기가 어려움. 그 결과, 학습이 잘 안 되고 성능이 떨어짐.

- **항등은 shortcut이 자동으로 보장**. (입력 x를 그냥 더해버리니까)
- 네트워크는 그저 **얼마나 수정할지(잔차 F(x)만 배우면 됨.
- 따라서 SGD는 "복잡한 항등 근사" 대신 "작은 보정값 학습"만 하면 됨.

6. 공부 기록

1. 네트워크가 깊어질수록 생기는 문제

네트워크가 깊어질수록 **역전파(backpropagation)** 과정에서 기울기(gradient)가 전달되는 동안 문제가 생김

- **기울기 소실(Vanishing Gradient)**

  활성화 함수(예: sigmoid, tanh)는 출력 범위가 제한적이라서 미분 값이 0~1 사이의 작은 값.

층을 거듭 통과하면서 곱해지면 점점 0에 가까워져서, 앞쪽 층(입력 가까운 쪽)까지는 학습 신호가 거의 전달되지 않습니다. → 학습 정체.

- **기울기 폭발(Exploding Gradient)**

  반대로 특정 경우(가중치가 큰 값으로 초기화되었을 때 등) 기울기가 계속 곱해져서 **기하급수적으로 커짐**.

  → 가중치 업데이트가 불안정해지고 학습이 발산(diverge)합니다.

즉, 단순히 층을 많이 쌓으면 **이론상 표현력이 늘어나지만, 실제로는 학습이 불가능해지는 경우**가 많았음.

2. Stochastic Gradient Descent (SGD) with Backpropagation

- **Backpropagation (역전파)**

  출력 오차(error)를 계산하고, 이를 미분(gradient)해서 **가중치별로 어떻게 수정해야 하는지** 계산하는 과정.
  손실 함수(loss function)가 출력층에서 얼마나 잘못됐는지 계산한 뒤, **연쇄법칙(chain rule)**을 써서 각 층의 가중치에 대한 기울기(∂L/∂w)를 구하는 **알고리즘**

- **Gradient Descent (경사하강법)**

  계산된 기울기를 이용해서 가중치를 조금씩 수정 → 오차 최소화.
  Backpropagation으로 계산된 기울기를 이용해 **가중치를 실제로 업데이트하는 최적화 방법**

  $w \leftarrow w - \eta \cdot \nabla L(w)$

  여기서 η는 학습률(learning rate), ∇L(w)는 손실 함수의 기울기.

- **Stochastic Gradient Descent (확률적 경사하강법)**

  전체 데이터(1 epoch)를 다 쓰는 대신, **작은 미니배치(mini-batch)**를 뽑아서 학습.

  → 더 빠르고, 노이즈 덕분에 지역 최소값(local minima)에 빠지지 않고 잘 일반화.

3. Degradation Problem

- 네트워크가 **깊어질수록** 정확도가 점점 올라가야 정상.

  하지만 실제로는, 깊이를 늘리면 오히려 **훈련(training) 에러조차 증가**하는 현상이 나타남.

→ **Degradation Problem

- **overfitting(과적합)** 때문이 아님.

  과적합이라면 훈련 에러는 낮아지고, 검증 에러만 높아져야 하는데, 여기서는 **훈련 에러 자체가 더 커져버림** → 학습이 잘 안 되고 있다는 뜻

- 얕은 네트워크가 이미 좋은 해를 찾았다고 했을 때, 더 깊은 네트워크를 만들면, 그 얕은 네트워크 구조를 그대로 복사한 뒤, 추가된 레이어들은 **항등 함수(identity mapping, 즉 입력을 그대로 출력)**만 하도록 두면 됨.

  → 이 경우, **깊은 네트워크의 해는 최소한 얕은 네트워크의 해보다 나쁘지는 않아야** 함.

> 💡 **"이론적으로는 깊은 네트워크가 얕은 네트워크보다 나쁠 수 없는데, 실제로는 최적화 알고리즘이 항등 맵핑 해를 잘 못 찾아서 학습이 어려워지고 훈련 에러가 커지는 현상(Degradation Problem)이 생긴다"**
>
> SGD + Backprop은 아주 깊은 네트워크에서
>
> **항등 함수(identity mapping: H(x)=x 조차 근사하기 어려워 함.**

4. ResNet(Residual Network)

1. 원래 목표 (Underlying Mapping, H(x))

딥러닝 모델이 하고 싶은 건 어떤 함수 H(x)를 학습하는 것.

예를 들어, 입력 이미지 x가 들어오면 → 그에 맞는 정답 레이블을 출력하는 함수가 H(x)

일반적인 신경망에서는 층들을 쌓아서 이 H(x) 자체를 직접 근사(fit)하려고 함.

2. ResNet의 새로운 접근 (Residual Mapping, F(x))

ResNet은 이렇게 하지 않고, **차이를 학습하자**라는 접근을 함.

- 우리가 원하는 건 H(x).
- 그런데 얘를 바로 학습하는 대신, **잔차(residual)**를 학습:F(x):=H(x)−x

즉, "정답 함수 H(x는 사실 입력 xxx에 약간의 변화만 더해진 형태일 수 있다"

→ 그 **변화량(잔차, residual)**만 학습하자는 것.

3. 최종 표현

원래의 함수 H(x)는 이렇게 다시 쓸 수 있음:

H(x) = F(x) + x

- 여기서 F(x): Residual function (잔차 함수)
- x: Identity mapping (입력을 그대로 전달하는 경로, shortcut connection)

즉, **신경망은 F(x)라는 "보정값"만 학습하면 되고, 나머지는 shortcut으로 그대로 더해줌.**

> 💡 ResNet은 신경망이 직접 원하는 함수 H(x)를 학습하지 않고,
> "입력과의 차이(residual)"인 F(x)=H(x)−x를 학습하도록 만들어,
> 최적화를 쉽게 하고 깊은 네트워크도 안정적으로 학습할 수 있게 한다.

Shortcut Connection (지름길 연결)

정의: 몇 개의 층(layer)을 건너뛰고 입력을 곧바로 출력 쪽에 더해주는 연결.

ResNet에서는 이 연결이 **항등(identity) 함수** 역할. 즉, $x \mapsto x$

- **스택된 층의 출력 = F(x)**

  → 네트워크가 학습하는 건 "입력과 목표 출력의 차이" **H(x) - x**
- **shortcut의 출력 = x**

  → 그냥 입력을 그대로 전달
- **최종 출력 H(x) = F(x)+x**

  → 즉, "입력 그대로" + "보정값"

> 💡 H(x)를 직접 학습하는 대신,
> →F(x) =H(x)−x (즉, **입력과 출력의 차이**)를 학습시키자.

Residual Block의 기본 구조

Residual Block은 다음 식으로 정의:

y=F(x,{Wi})+x

- x: 입력 벡터 (block으로 들어오는 값)
- y: 출력 벡터 (block을 지난 후 나가는 값)

- F(x,{Wi}): 여러 층(예: Conv + ReLU + Conv)을 통과해 얻은 **잔차 함수(residual function)**
- +x: 입력을 shortcut(항등 경로)으로 그대로 전달해서 더함

즉, **출력 = 입력 + 잔차**

ex) residual 함수 F가 2개의 레이어로 구성된 경우:

$F(x)=W_2\sigma(W_1x)$

- W_1, W_2: 가중치 행렬 (혹은 convolution filter)
- σ: 비선형 활성화 함수 (ReLU)
- Bias는 단순화를 위해 생략

따라서 최종 출력은:

$y=W_2\sigma(W_1x)+x$

Element-wise Addition (요소별 덧셈)

- F(x)와 x의 출력 차원이 같아야 element-wise(원소별)로 더할 수 있음.
- 만약 차원이 다르면?
    - shortcut에서 선형 변환 Ws를 적용해 차원을 맞춤.

$y=F(x,\{W_i\})+W_sx$

예: 채널 수가 변할 때 1×1 convolution으로 맞춤.

34-layer residual(오른쪽, ResNet-34)

· 가운데 plain과 뼈대(해상도 전환 지점, 채널 증감, 총 연산량)를 거의 동일하게 유지하되, 각 몇 개의 conv 묶음을 잔차 블록으로 바꾼다. 잔차 블록의 수식은 y=F(x)+x이며, F는 보통 3×3 conv 두 층으로 구성.

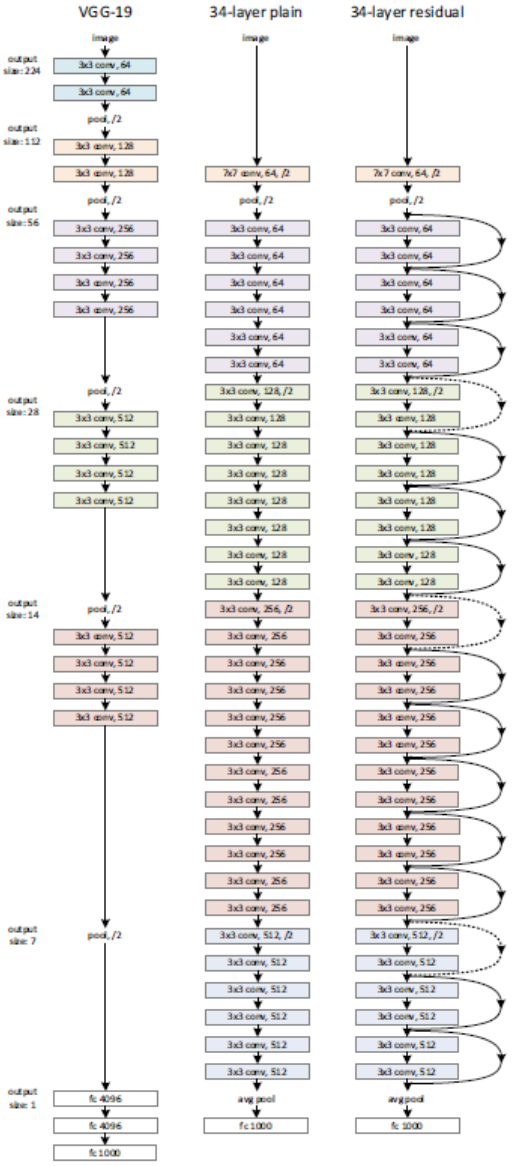
· 오른쪽으로 휘어진 화살표가 shortcut.

– 실선 shortcut: 입력과 출력의 차원이 같을 때 항등(identity) 경로로 그대로 더합다. 추가 파라미터와 유의미한 계산 증가가 없다.

– 점선 shortcut: 해상도나 채널이 바뀌는 경계에서 차원을 맞춰야 할 때.

A. 항등을 유지하되 채널 증가분은 0으로 패딩해 맞추는 방법(파라미터 증가 없음).

B. 1×1 conv(프로젝션, Ws)로 x를 선형 변환해 정확히 모양을 맞춘 뒤 더하는 방법(소량의 파라미터/연산 추가).



5. ImageNet 분류 실험(plain network vs residual network)

Plain Networks 결과

비교 대상:

- **18-layer plain net**
- **34-layer plain net**

결과:

- 34층 plain net이 오히려 18층보다 **validation error가 더 큼**.
- 즉, 깊어졌는데 성능이 나빠짐 → **Degradation Problem 발생**

- 학습 과정에서 34-layer plain은 훈련 에러조차 계속 18-layer보다 높음.

  즉, 단순히 과적합 문제가 아니라 **훈련 자체가 잘 안 되는 최적화 문제**임.
- 저자들의 주장:
  - 이건 **vanishing gradient(기울기 소실)** 때문은 아님.
  - 왜냐면 Batch Normalization(BN)을 썼기 때문에:
    - 순전파(forward) 신호는 항상 분산이 유지됨 → 0으로 죽지 않음.
    - 역전파(backward) 기울기도 norm이 정상적으로 유지됨.
  - 즉, BN 덕분에 gradient vanishing/exploding 문제는 완화됨.
- 저자들의 추측:
  - **깊은 plain net은 수렴 속도(convergence rate)가 지수적으로 느려진다.**
  - 그래서 충분히 학습을 진행해도 에러가 줄어드는 속도가 너무 느려서 성능이 떨어진다.

Residual Network(ResNet)

- residual net은 각 3×3 conv 쌍마다 shortcut connection 추가.
- **Shortcut 방식**: Option A → 항등(identity) 매핑 + 채널 수 늘릴 땐 zero-padding.

1. 깊이가 깊을수록 ResNet이 더 강력

- Plain net에서는 34층이 18층보다 오히려 성능이 떨어졌습니다(Degradation Problem).
- 하지만 ResNet에서는 상황이 반전:
  - 34-layer ResNet이 18-layer ResNet보다 **Top-1 에러가 2.8% 더 낮음**.
  - 즉, 깊어질수록 오히려 성능이 좋아짐.
- 또한 34층 ResNet은 훈련 에러가 크게 낮고 검증셋에서도 잘 일반화됨.
- 결론: **Residual Learning이 Degradation Problem을 해결했다.**

2. Plain vs ResNet 비교

- 34층 기준으로 비교하면:
  - **34-layer ResNet**이 **34-layer plain net**보다 Top-1 에러를 3.5% 낮춤.

    - 이는 곧 residual connection 덕분에 훈련 에러가 확 줄었기 때문(Fig. 4 오른쪽 vs 왼쪽 비교).
- 결론: **Residual connection이 깊은 네트워크 최적화에 효과적임이 검증됐다.**

3. 얕은 네트워크에서는 큰 차이 없음

- 18층 모델의 경우: plain과 residual의 최종 정확도는 거의 비슷.
- 하지만 **18-layer ResNet은 수렴 속도가 더 빠름** (훈련 초반에 에러가 빨리 떨어짐).
- 해석: 얕은 네트워크는 SGD로도 충분히 학습이 가능하지만, Residual 구조가 들어가면 **학습 안정성이 올라가고 초기 수렴 속도가 빨라진다.**

Shortcut 구현 방식

1. **Option A (Identity + Zero Padding)**

- shortcut은 그냥 항등 매핑(identity)
- 다만 채널이 부족할 땐 0을 채워서 맞춤 (zero-padding)
- 장점: 파라미터 추가 없음, 연산량도 거의 없음.

2. **Option B (Projection for size change only)**

- 해상도나 채널이 바뀌는 경우엔 1×1 conv(= projection)으로 변환해서 맞춤.
- 나머지 경우에는 항등 shortcut 그대로 사용.

3. **Option C (Projection everywhere)**

- 모든 shortcut을 projection(1×1 conv)으로 구현.
- 즉, 항등 대신 항상 가중치 있는 1×1 conv를 둠.
- 파라미터와 연산량이 많이 늘어남

💡 결과 정리

- 세 가지 옵션(A, B, C) 모두 **plain network보다 훨씬 성능이 좋음**
  - → 즉, residual 구조 자체가 핵심.
- B가 A보다 약간 더 좋음 → 이유: A에서는 zero-padding된 채널은 사실상 "학습이 안 되는(dead channel)" 부분이라 residual 학습 효과가 없음.
- C가 B보다 아주 조금 더 좋음 → 하지만 차이는 크지 않음. 성능 향상은 projection이 아니라 residual 구조 덕분.
- 따라서 projection shortcut은 꼭 필요하지 않음.

Bottleneck

- 4-layer까지는 **Residual Block**을 2층 구조(3×3 conv → 3×3 conv)로 설계했음.
  - 그런데 ImageNet에서 100층 이상으로 가면, 이 구조를 그대로 쓰면 **계산량과 파라미터 수가 너무 커짐**.
  - 그래서 ResNet 저자들은 **"Bottleneck 블록"**이라는 더 효율적인 구조를 고안.
  - Bottleneck 설계 덕분에 **계산량은 줄이고 깊이는 늘릴 수 있게** 됨.
- 구조

  하나의 Residual Block = **3층 구조**:

  1. **1×1 conv (차원 축소)**
     - 입력 채널을 줄여서, 다음 3×3 conv 연산량을 줄임.
  2. **3×3 conv (핵심 연산)**
     - 가장 중요한 feature extraction.
     - 채널 수가 줄어든 상태에서 수행하므로 연산량이 적음.
  3. **1×1 conv (차원 복원)**
     - 다시 원래 채널 수로 늘려 shortcut과 더할 수 있도록 함.

  👉 즉, **1×1 → 3×3 → 1×1** 구조가 Residual Function F(x).

  👉 "병목(bottleneck)"이라는 이름은, 중간 3×3 단계가 **좁은 차원(채널 수 적음)**에서 수행되기 때문

- Bottleneck 블록의 shortcut을 projection(1×1 conv)으로 하면, **양쪽 고차원 feature map을 다 변환해야 해서** 파라미터와 연산량이 거의 2배가 됨.
    - 따라서 bottleneck 구조에서는 **항등 shortcut(Identity)**이 훨씬 효율적