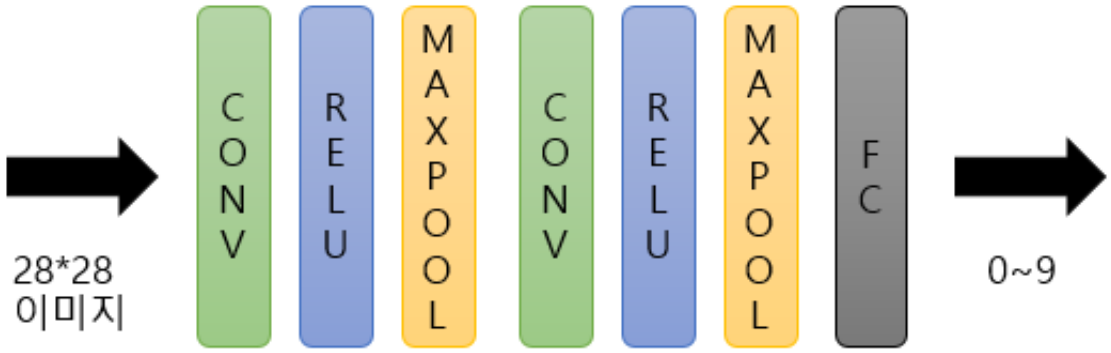
CNN model

0. List



1. Todo

CNN model

- AlexNet
- ZFNet
- GoogleNet
    - inception
- vgg 16, vgg 19
- ResNet

를 설명하면서 ILSVRC에서 2012-2015까지의 발전

좀 양이 많아서, ILSVRC기준. 혁명적이었던 이유 / 대표적 특징만 언급

사실 이 모델이 나오고 끝나는게아니고 추가적으로 더해지는 것들이 있었지만 주제가 'CNN을 이용한 모델'이기에 생략!

conv를 병렬로 쌓는 것과 직렬로 쌓는 것의 차이

2. Review : CNN

3. AlexNet

1) AlexNet?

a. AlexNet이란?

- 영상 DB기반으로 한 ILSVRC (ImageNet Large-Scale Visual Recognition Callenge) 2012에서 우승
- 논문의 제 1저자 Alex
- 그들이 개발한 CNN구조를 AlexNet

b. AlexNet의 효과?

- ILSVRC의 결과에서 보면 다른 것들에 비해 혼자 15% 대로 압도적 성능
- 이후 대회에서는 모두 AlexNet을 기준으로. 2013년부터는 이보다 더 좋은 결과들 나옴

c. AlexNet의 의의

- GPU를 사용 -> 이후 CNN구조 설계시 GPU사용이 대세가 됨
    - 알파고도 1920개의 CPU, 280개 GPU를 병렬적으로 사용

```
사실 이 전에 LeNet5라는 것이 존재.

여기서는 2개의 GPU를 기반으로 한 병렬 구조
```

2) AlexNet

a. 개요

- 5개의 conv
- 3개의 fully connect
- 1000개 카테고리로 분류 (softmax)

이 밖에도

- 65만개의 뉴런
- 6000만개의 파라미터
- 6억 3000만개의 connection
- -> 2개의 GPU.. (GTX580) : 3GB메모리.

b. 모델의 구조

- input : 227*227*3 (영상) - 매우 큰 편
- conv1 layer kernel : 11*11*3 - 큰 편
- stride 4

c. 중간에 질문 들어온다면

Q. 논문 이미지에는 224로 적혀있는데 왜 227이냐?

출력 크기 = { ([입력크기]-[커널사이즈]+[제로패딩*2]) /2 } + 1 = 출력크기

224를 넣으면 55가 나올 수가 없고 227을 넣어야 55로 떨어짐

- caffe에 기본 제공돼있는 AlexNet 모델의 경우 입력값 227
- 1 레이어에만 패딩이 없고 나머지는 있다

> if 가로 224*224
>
> 커널 11*11
>
> stride 4
>
> 1 필터 : 1 ~ 11
>
> 2 필터 : 5 ~ 15
>
> 3 필터 : 9 ~ 19
>
> ..
>
> n 필터 : 4(N-1) + 1 ~ 4(N-1) + 11
>
> - 총 55개의 필터
> - 217 ~ 227
> - 근데 ZFNet이후는 7*7 에다가 stride 4을 사용 (어느 블로그에선 7이라하지만 오타로 보임)
> - 그럴 경우
> - 1 ~ 7
> - 4(N-1)+1 ~ 4(N-1)+7
> - 마지막 픽셀 좌표 224 (아님ㅎ223)
> - field 3만큼 작기 때문에 영상 크기도 224

d. conv1

```
이것말고 이전에 LeNet이라는 것이 있었다
input 32*32 conv레이어는 모두 5*5kernel. 인풋이 흑백이라 depth 1
이게 conv를 거치면서  depth증가하게 됨
```

- 근데 AlexNet에서는 227*227*3 으로 매우 큼
- conv1에서는 kernel 11*11*3 인 큰 reception field 사용
- stride 4라서, 96개의 feature-map 을 생성하기 때문에 결과는 55*55*96 (48 두개)
- 즉
  - conv1 layer는 55*55*96 = 290,400개 뉴런
  - 각 커널은 11*11*3 = 363개의 weight + 1개의 bias = 한개의 커널당 364개 파라미터
  - 커널이 96개이므로 364*96 = 35,000 개 쯤
  - 커넥션은 290,400 * 364 = 1억개 쯤..
- conv1 한번 했을 뿐인데 1억개 이상의 conv생성
- conv1에서는 **maxpooling하지 않음.** 대신 stride를 4로 했기 때문에 (연산 수 줄이기 위해) pooling한 것처럼 영상의 크기 작아져 있었다.

d. conv2

kernel : 5*5*48

layer 55 * 55 * 48

stride ?

max pooling!

결과 : 27 * 27 * 128(256)

e. conv3

GPU1 , GPU2의 결과를 모두 섞음

3) AlexNet의 성능 향상을 위한 고려

- ReLU, overlapped pooling, response normalization, dropout
- 2개의 GPU사용

a. ReLU

AlexNet처럼 망의 크기가 큰 경우는 속도에 치명석 영향

여기서 ReLU사용

sigmoid, tanh쓸 때보다 학습속도 6배

b. overlapped pooling

- average pooling : window내의 픽셀의 평균
- max pooling : window내의 픽셀 중 최대값
    - average pooling보다 연산량이 더 많다
    - LeNet5에서는 average pooling. **AlexNet에서는 max pooling**
    - AlexNet
        - pooling : stride2, window 3*3

c. Local response normalization

- overfitting 피하기 위해 normalization 수행
- **ReLU를 쓰면 input의 normaization 필요없는게 특징**
- 하지만 output에선 입력에 비례하여 그대로 증가
- **conv1, conv2 output에서는 ReLU사용**
- max pooling 전에는 response normalization
- 같은 위치에서의 response를 인접한 다른 커널들의 결과와 비교해서 강한 자극만 전달될 수 있게 하는 원리
- **여기선 LRN이 필요하다고 주장하지만, 2년후 vgg에서는 LRN은 필요없다해서 거의 사용 안함**

d. overfitting 해결 - data augmentation

- 데이터 양 늘리기

- 작은 연산으로 학습 데이터 늘리는 2가지 방법
- GPU가 이전 이미지를 학습 하는 중에, CPU에서는 이미지를 늘려 디스크에 저장 X
    - ILSVRC 인풋은 256*256 -> 224*224(??? 227 아니고?)
    - 그렇게 1장의 학습 영상에서부터 2048개의 영상 얻음
    - test시에는 5개의 224*224(!!!!!), 수평 반전 가지고 10개로부터 softmax
- 각 학습 영상으로부터 RGB채널의 값을 변화
    - PCA라는 분석을 했다. 평균 0, 표준편차 0.1갖는 랜덤변수 곱해서 원래 픽셀값에 더해주는 방식
    - 다양한 영상 얻는다

e. overfitting 해결 - dropout

- AlexNet에서는 fc layer의 처음 2개 레이어에 대해서만 적용

4) 영향

AlexNet으로 인해 연구자들은 GPU로 관심 돌림

2012년에는 AlexNet설계자만 GPU사용. 2013, 2014에는 대부분 GPU사용 (표 참고)

- GPU왜 필요? 연산량 때문
    - CPU 14일 -> GPU 1일
    - CPU 3개월 -> GPU 일주일

```
ZFNet
생략쓰바갈쓰...일단........

- 빨리빨리..
- CNN의 구조 결정하는 가장 적합한 하이퍼파라미터 설정을 찾는건 어려운 일
- AlexNet 같은 경우 2개의 GPU로 학습하는 데 일주일 이상 소요
- 이를 해결하기 위해 Visualizing기법으로 해결
- 즉, ZFNet은 CNN구조를 가리키는 것이 아닌, CNN을 더 잘 이해할 수 있게 하는 기법

1) visualizing

- CNN의 중간 layer에서 feature의 activity가 어떻게 동작하는지를 알아야 한다
- 이 activity를 이미지 공간에 mapping시키는 기법
```

4. GoogLeNet(2014)

- 2014 ILSVRC는 구글의 GoogLeNet이 우승. 근소하게 옥스퍼드의 VGGNet이 2위
- **2014부터 CNN구조에 큰 변화**
- 이전에는 10layer 깊이. 2014부터는 deep해졌다
- GoogLeNet, VGGNet은 AlexNet에서부터 변화를 모색.
    - CNN을통한 학습이 더 커짐

1) deeper

a. deep

CNN의 성능을 향상시키는 가장 직관적인 방법 : 망의 크기를 늘리는 것

망의 크기를 늘린다?

- depth를 늘린다
- layer의 unit 수 (width)를 늘리는 것
- 대용량 데이터 사용

b. 역사

- 2013 까지는 CNN의 깊이 10 미만
- 2014인 GoogLeNet, VGGNet은 각각 22, 19까지 깊어짐
- AlexNet나온지 2년만에 에러율 10% 미만 달성
- 2015우승인 ResNet은 152레이어로 깊음. 3.57%로 내려감

c. depth가 너무 깊어지면 부작용? (side effect)

- 망이 커질수록 free parameter의 수가 증가
    - **overfitting에 빠질 가능성이 높아진다**
    - 이를 해결하기 위해선 아주 대량의 데이터에 라벨링이 필요한데, 이게 힘들다
- 연산량이 늘어난다
    - **필터 개수가 증가하면, 연산량은 제곱으로 늘어남**
    - ex) AlexNet
        - free parameter 수 6,000만 개
        - 6억 3,000만 개의 connection
        - 학습시간 1주일
        - 망이 깊어진다면 파라미터 더 많아지고 커넥션도 많아져서 시간 엄청 오래걸림
- 그럼에도 불구하고! deep하게 하는 이유?
    - 학습 능력이 증가하기 때문

2) Inception 개요

- 힘들어ㅓㅓㅓㅓㅓㅓ
- 그 인셉션 맞음
- 인셉션 영화(2010)

a. 개략

- 1*1 conv를 사용하여 차원을 줄이고 : reduce dimension
    - -> 망이 깊어졌을 때 연산량이 늘어나는 문제 해결
- GoogLeNet의 등장! <- 여기에 inception모듈이라는 것이 들어감
- 이후 inception이란 이름으로 논문 발표 : inception의 여러 버전 중 하나가 googLeNet
- AlexNet과의 비교
    - 훨씬 딥한데 free parameter의 수는 1/12
    - 전체 연산량도 적다

b. NIN : Network In Network

- 일반적 CNN
  - **feature extraction (conv + pooling) + classifier (fully connected nerual network)**
  - conv와 pooling을 번갈아 사용하는 layer여러개를 사용하여 feature 추출하고
    - 최종 vector를 classifier역할하는 fc로 처리
  - CNN의 conv가 feature는 잘 추출하지만 filter특징이 linear하기때문에 non-linear한 성질을 갖는 feature를 추출하기엔 어려움 있어서 **feature-map개수를 늘려야하는 문제**에 주목 => 개수 느리면 연산량이 늘어난다
- micro neural network의 등장
  - conv filter대신에 MLP에서 feature추출하도록 함
- CNN과의 비교
  - CNN : filter의 커널을 입력 영상의 전체 영역으로 stride만큼 옮겨가며 연산
  - NIN : conv대신 MLP사용. 전체 영역을 sweeping하며 옮겨감 (CNN과 비슷)
  - CNN : fc가 있다
  - NIN : fc가 없다. 대신 Global average pooling사용. (**이미 feature vector잘 추출했기 때문에 풀링만으로도 충분**) -> average pooling만으로 classifier하기때문에 overfitting 피하고 연산량 줄어든다. googlenet도 global average pooling 사용
- MLP를쓰면 non-linear한 성질을 잘 활용할 수 있어서 feature추출 우수
- 1*1 conv를 사용해 feature map을 줄일 수 있도록 함

NIN에서는 망을 깊게 하기 위해 mlpconv layer를 여러개 쌓기 때문이며, 네트워크 안에 네트워크가 있다

GoogLeNet에서도 inception모듈을 9개 사용하기 때문에 개념적으로는 NIN과 비슷

c. 1*1 convolution (부연설명 필요)

- 사용 이유 : **차원 줄이기 위해**
- 여러개의 feature map으로부터 비슷한 성질들을 묶어낼 수 있고 -> feature-map의 수를 줄일 수 있다 -> 연산량 줄인다 -> 망을 더 깊게할 여유가 생긴다
- 논문에서는 1-layer fully-connected neural network라고 하는데 방식이 같기 때문

d. 구글의 inception

- 원래 버전
  - 1*1, 3*3, 5*5 conv, 3*3 max pooling
    - 3*3, 5*5는 많은 연산 필요
- 나중 버전
  - 3*3, 5*5 앞에 1*1 conv를 아래에 두고 1*1를 통해 차원 줄이면 여러 scale확보하면서도 연산 균형 맞춤
  - GoogLeNet이 22레이어까지 갈 수 있는 것도 이 덕분
- 이것이 인셉션 모듈
- 1*1, 3*3, 5*5 (파랑)을 가지고 다양한 scale의 feature를 추출하는게 가능해짐
- 1*1 conv를 통해 (노랑) 연산량 감소 -> 망 깊이, 넓이 증가 가능!
- 이 inception을 통해 NIN구조를 갖는 deep CNN구현이 가능해졌다

3) GoogLeNet

a. 구조

파라미터가 있는 layer기준으로 22개의 layer소유

거의 100개의 unit을 가짐

9개의 inception module 적용

- 파란색 : conv layer
- 빨강 : max-pooling unit
- 노란색 : softmax
- 녹색 : 기타 function
- 위의 숫자 : 각 단계에서 얻는 feature map의 수

표

- patch size/stride : kernel 크기, stride
- output size : 얻는 feature map의 크기 및 개수 n*n*x = x개의 feature map. n*n 필터
- depth : 연속적 conv layer
- #1*1 : 1*1 conv. 이걸 수행한 후 얻는 feature map 수
- #3*3 reduce: 3*3 conv 앞에있는 1*1 conv
- #3*3: 1*1에 의해 줄어든 feature map에 3*3 적용
- pool/proj : max pooling, max-pooling뒤의 1*1 conv.
- params : 해당 layer에있는 free parameter의 수
- ops : 연산의 수. feature map의 수와 입출력 feature map의 크기에 비례

3*3 보다 5*5를 통해 얻는 feature-map의 개수가 작은 이유는 5*5가 더 많은 연산량 필요하기 때문

인풋 이미지의 크기가 이미 줄어든 상황에서는 3*3에서 얻는게 5*5보다 많기 때문

b. Auxiliary classifier (엌질리어리~)

deep해지면서 생기는 큰 문제 : vanishing gradient

- 이로인해 학습이 아주 느려지거나 overfitting이 발생
    - 뉴럴넷에서는 output의 error를 backpropagation하여 파라미터를 갱신
    - 근데 gradient가 0근처로 가면 학습이 엄청 느려지거나 param변화가 거의 없어 결과가 더 나빠짐
    - sigmoid를 활성함수로 쓰면 미분값이 0으로 수렴해서 학습속도가 느려짐
        - cross-entropy쓰면 개선은 되지만 본질적인 문제 해결은 아님
    - 요즘은 ReLU많이 쓰는데, 여러 레이어 거치면서 작은 값이 계속 곱해지면 결국 0 근처로 수렴
        - 더 깊어질수록 가능성이 커짐
- **이를 해결하기 위해 Auxiliary classifier 2군데 사용 : vanishing gradient 피함**

training deeper convolutional networks with deep supervision 논문 참고 (여기선 supervision이라 부름)

- **X4에 auxiliary classifier넣고 그 아웃풋으로부터 backpropagation결과를 결합**
- 결과를 보면 auxiliary가 없으면 0에 근접해지지만 있으면 다시 증가 : 안정적 학습
- **-> 2015년 논문에서 이게 regularizer와 같은 역할**

- 이게 batch-normalization되거나 dropout있으면 더 좋은 결과나온다는게 후에 나옴

학습할 때만 Auxiliary classifier넣고, 학습 후 DNN쓸 때는 제거!

c. 인수분해 (factorizing)

5*5 conv는 2단의 3*3conv로 구현 가능

free parameter 수 : 25 -> 9+9=18 : 절감

이와같이 대칭을 이용하지 않고 row / column 방향으로 인수분해도 가능

즉, 3*3 -> 1*3 / 3*1로 분해

n*n -> 1*n / n*1 로 바꿀 수 있으며 n이 커질수록 파라미터 절감 커진다

그림에서 n=3이면 인셉셜 모듈과 동일

한마디로

- **auxiliary classifier의 역할**
- 큰 filter갖는 conv kernel을 인수분해해서 **작은 크기를 갖는 conv로 대체하면 free parameter의 개수가 줄면서 연산량 절감을 가져옴**
- 큰 필터를 균일한 크기의 3*3로 표현하는건 VGG의 핵심 아이디어

d. grid

(여기는 좀 빠르게)

보통은 conv뒤에 pooling레이어 두고 grid줄였다

- 대표적인 방식 : conv 때 stride 1이상으로 하거나 pooling사용

conv - pooling / pooling - conv 뭐가 더 grid를 효과적으로 줄이는 것인가?

전자 -

- 큰 크기의 feature map에 conv적용했기 때문에 연산량은 4배 많다. 숨은특징 더 잘 찾아낸다

후자 - 최적으 아니다

- pooling 거치면서 숨은 정보 (representational concept)가 사라질 가능성 있다

-> 이건 좀 패스...일단.. [4] https://laonple.blog.me/220716782369

R-CNN

패스

https://laonple.blog.me/220731472214

5. VGGNet

2014 ILSVRC에서 GoogLeNet과 근소한 차이로 2위를 차지한 옥스포드의 VGGNet

구조적으로는 GoogLeNet보다 훨씬 간단한 구조를 가지고 있어 **이해가 쉽고 변형해보기도 쉬워 더 많이 사용**

실제로 inceptionV2, V3에서도 VGGNet꺼 일부 사용 중

1) 구조

AlexNet과 매우 유사

a. 층 보기

CNN은 보통 conv다음 pooling이 오는데 여기서는 3*3인 필터를 여러개 쌓는 구조 선택

- 3*3 2개 : 5*5 conv
- 3*3 3개 : 7*7 conv
- 이렇게 함으로 인해 파라미터 수 줄고 학습 속도가 빨라짐
    - 레이어 수도 많아지므로 -> 여러번 비선형 처리를 하므로 non-linearity가 증가. 더 좋은 feature추출 가능
    - 채널 개수 C일 때, 7*7 : 7^2*C^2 // 3*3 3번 : 3*(3^2*C^2)
- 실제로도 이렇게하는게 더 빠르다 (top-1 error에서 7%정도 상승)

b. 표

레이어 수에 따라 이름이 붙는다

- 표에서 D : vgg16
- E : vgg19
- GoogLeNet과 다른 점
    - fc가 있다
    - 파라미터가 매우 많다 (구글이 질투중)

ILSVRC2012에서는 16레이어 이상은 별 도움 안되었다

학습 데이터 / 문제에 따라 레이어 몇개가 최적인지 결정

- 테스트 결과
    - 깊이가 깊어질 수록 error 감소 (19 이상에서는 감소하지 않지만, 데이터가 다르다면 괜찮을지도)
    - single scale보다 multiple scale이 더 낮은 에러율
    - multi-crop의 성능이 dense보다 성능이 더 좋으며 두개를 동시에 쓰면 더 좋은 성능

2) 특징

- 작은 필터 크기의 conv연산 (3*3)
    - AlexNet은 11*11 GoogLeNet은 7*7
- A-LRN에서는 LRN을 사용하고 이지만, 다른 모델에서는 사용
    - 별 효과가 없다고 판단
- 1*1이 없는건 아님
    - GoogLeNet이나 NIN처럼 차원 줄이기보다는 ReLU로 non-linearity확보 위함
- 11레이어를 먼저 학습 후 그 결과를 더 깊은 레이어의 파라미터 초기화에 이용 후 학습 (보충필요)
    - GoogLeNet에서는 auxiliary classifier로 해결

a. 단점

- 파라미터가 너무 많다
- 간단한 구조를 가졌지만 FC가 3개가 있고 pooling뒤에는 feature map이 2배로 커지면서 필요한 파라미터가 많아짐
- 파라미터가 많다 -> gradient vanishing, overfitting 발생 가능성이 크다

3) 의의

3*3 kernel사용했음에도 불구하고 좋은 성과

4) 기타

training할 때, test할 때 데이터를 어떻게 준비했는지는 생략

이미지넷일 경우 vgg16, vgg19비슷

6. ResNet

Residual learning

short connection, identity mapping

152레이어(ultra deep)

마이크로소프트

1) Intro

a. 망이 깊어지면 단점

- vanishing/exploding gradient 문제
  - CNN에서 parameter update를 할 때 gradient값이 너무 크거나 작은 값으로 되어 더 움직이지 않아 학습 효과가 없어지거나 속도가 아주 느려지는 문제
  - **이를 해결하기 위해 batch normalization, 파라미터 초기값 설정 등 여러가지 시도**
  - 하지만 **레이어가 많아지면 잘 해결되지 않음**
- 학습의 어려움
  - **파라미터 수가 매우 많아져 overfitting이 아니더라도 에러 발생**

b. ResNet team의 trial

- 56layer, 20layer 봐도 오히려 56layer가 더 오류 발생이 높음
  - 원래 레이어가 딥해지면 더 에러율이 낮아야하는거 아니었나!
  - **문제있다! 해결하자!**

2) Residual learning

깊어도 학습을 잘 할 수 있는 방법에 대해 제안 : residual learning

a. how to

기존 CNN : input x -> layer -> layer -> H(x) 를 얻는다

기존 뉴럴넷에서는 H(x) (output), y (target)의 차를 최소로 하는 방향으로 학습

- **여러층의 non-linear function이 identity mapping이 되도록 학습시키는 것이 쉽지 않다**
- H(x) = x가 되도록 학습할 것

만약 목표를 H(x)-x. 즉 output - input 을 얻게 학습을 하게 된다면 layer는 H(x)-x를 얻도록 학습이 되어야 한다.

새로운 output F(x) = H(x)-x 라면, H(x) = F(x)+x가 된다. 즉 F(x)+x를 H(x)에 근사

그렇게해서 **변한 CNN구조가 residual learning의 기본이 된다**

F(x)를 학습한다는 것은, H(x)에서 x를 뺀 '나머지'(residual)를 학습한다는 것

- 변화
  - input에서 바로 output으로 연결되는 shortcut 연결이 생긴다. (더하기 연산만 수행)
  - identity shortcut connection
    - +연산만 추가되는 것이기 때문에 **파라미터 수에 영향X.**
    - **몇개의 레이어 건너뛰면서 input, output연결되기 때문에 forward, backward path단순**
      - 몇 개의 레이어를 skip하는 것이기 때문에 shortcut connection이라 부름
      - layer를 지난 출력과 element-wise addition
      - input, output dimension다르면 dimension맞추기 위해 parameter를 추가해서 학습
    - 즉, 깊은 망도 쉽게 최적화 가능. depth를 늘려 정확도 개선 가능
  - 이젠 F(x) = H(x)-x를 얻기 위한 학습을 함
    - 최적 = F(x)는 0. 학습할 방향이 미리 정해져 pre-conditioning을 함
    - F(x)가 0 방향으로 학습을 하면 작은 움직임을 검출.

b. info

- ILSVRC 2015 우승 3.57%
- 2014의 GoogLeNet보다 성능 두배. 망의 깊이 7배 이상
  - **2016년에 역전당함**

3) ResNet

a. 구조

- ResNet 설계시엔
  - 대부분 conv layer : 3*3 kernel
  - 연산량 줄이려고 max-pooling(1곳 제외), hidden fc, dropout 사용 안함
    - **입력, 출력의 feature map size가 동일한 곳에서 identity shorcut사용**
    - feature map size 증가시엔 둘 중 하나를 따름 (stride는 둘 다 2)
- **plain network 설계시엔 VGGNet 철학 많이 이용.**
  - output feature-map 크기가 같은 경우 / 해당 모든 layer는 모두 동일한 수의 filter를 갖는다
  - feature-map 크기가 절반으로 작아지면 / 연산량 균형 위해 filter 수를 두배 늘린다.
    - feature-map 크기 줄일 땐 pooling사용하는 대신 conv때 stride 2배
  - 이렇게 하면 망의 **깊이는 VGG보다 깊지만, filter수 줄이고 복잡도 낮춰서 연산량 줄임**

- 34layer의 plain network, residual network 비교
  - residual은 **2conv마다 shortcut connection**
  - **residual이 더 결과 좋고** residual은 34layer가 18layer보다 더 좋은 결과 (top-1 error기준)
    - plain34, resnet34에서 보면 resnet이 깊은 망에서 더 압도적 성과
    - plain18, resnet18에서 보면 비슷해보이지만 resnet이 더 빨리 수렴
  - **ResNet이 plain보다 더 수렴 속도가 빨랐다** (더 좋은 결과, 더 빠른 결과)
- 하지만 50, 101, 152layer에서는 구조 조금 더 바꾼다
  - 3*3 -> 1*1, 3*3, 1*1 : 병목처럼.
    - 차원을 줄였다가 다시 shortcur connection위해 원복
  - **연산시간을 줄이기 위해**
  - 1*1 쓰는건 dimension줄이기 위해
  - 3*3 후 다시 1*1은 dimension을 확대(??)시키기 위한 결과

b. 결과

- 152라는 아주 깊게 레이어를 짜도 좋은 결과.
- 이렇게 깊어도 결과 잘 얻을 수 있다

c. 나중에

- CIFAR데이터 (32*32 작은 영상 데이터)에서 쓰려고 좀 더 발전
- **1000레이어 넘기도 함**
  - CIFAR영상크기 작기때문에 feature map크기 조정. filter개수 조정
- 이모든건 classification!

4) 2016의 ResNet

2015땐 대회때문에 시간 제한.

pre-activation적용한 ResNet

- x(l+1) = f(h(xl) + F(xl))
  - xl : input / x(l+1) : output
  - h(x) : shortcut
  - F(x) : H(x) - h(x)
    - H(x) : 원래꺼
  - f(x) : ReLU
- 원랜 h(x)는 shortcut connection이었지만, 실제로는 identity 함수와 residual함수의 합을 더하고 ReLU를 수행하기 때문에 저런식으로 표현해줘야 한다
- 이 shortcut connection (h(x))이 어떤 경우에 가장 좋은 결과를 얻을 수 있는지