CNN Architectures

1 LeNet-5

Conv - Pool - Conv - Pool - FC - FC

2. AlexNet : [Conv1 - MaxPool1 - Norm1 - Conv2 - MaxPool2 - Norm2 - Conv3 - Conv4 - Conv5 - MaxPool3 - FC6 - FC7 - FC8]

① ReLU 처음으로 사용

② LRN이라는 normalization 기법 사용 (핸즈온 p.563)

③ data augmentation

④ dropout 0.5 , batch 128 , SGD momentum 0.9 , lr 1e-2, manually reduce (/10) when plateaus , L2 decay 5e-4 . 7 CNN ensembles

⑤ Conv1 (55x55x96)은 2개의 GPU에 나눠서 train [55x55x48] x2

⑥ Conv 1, 2, 4, 5 : 같은 GPU 내의 feature map만 연결

⑦ Conv 3, FC 6, 7, 8 : 모든 앞선 feature map과 연결. 그니까 모든 GPU device와 communicate

→ ZFNet : AlexNet hyperparameter update

3. VGG : small filters, deeper networks

① 16 ~ 19 layers

② 3x3 , stride 1, pad 1, conv 2x2, stride 2 MaxPool

→ 왜 작은 filter 사용? 작은 filter 여러 개 사용이 큰 filter 1개 사용보다 (1) non-linearity 측면에서 더 효율적

(2) parameter 수가 적음 e.g. 3개 3x3 ≈ 1개 7x7

3x(3x3)xC < 1x(7x7)xC

③ Memory 대부분은 초기 conv layer에, parameter 대부분은 FC layer에

④ details :

(1) LRN 없음

(2) VGG 16 or VGG 19 : 19가 더 나은 performance, 더 많은 메모리 사용

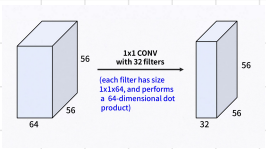
(3) 앙상블 이용

(4) FC7 (2번째 FC layer in VGG-16)이 다른 task에도 잘 generalize 됨.

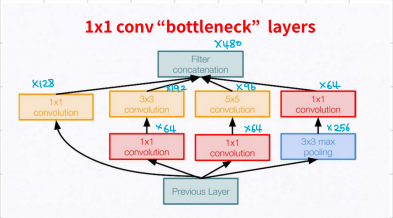
4 GoogLeNet : deeper networks, with computational efficiency

① 22 layers

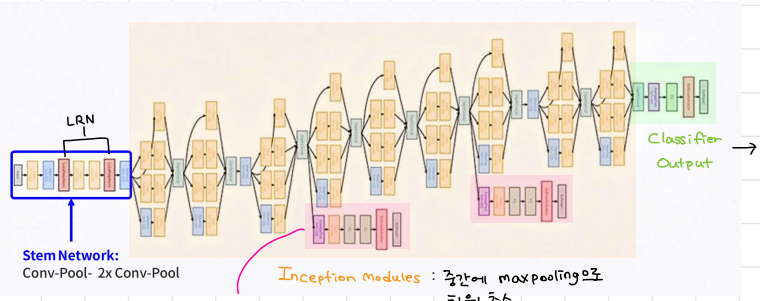
② "Inception" module 서브 네트워크



- 1x1 conv
  - dim reduction ⇒ 훈련 속도↑ , 일반화 성능↑
  - 합성곱 층이 1개에서 2개 ([3x3] → [1x1, 3x3], [5x5] → [1x1, 5x5])로 늘어서 더 복잡한 패턴을 감지할 수 있음.



⇒ 커널수는 hyperparameter. 즉 inception module 하나 추가 시 hyperparameter는 6개 추가됨.



→ 맨 앞에 Avg Pooling 사용 → FC layer 1개로만 구성 가능 → param # ↓↓

: 여기서 나온 loss를 더함 ← gradient vanishing 줄이고 (이전 layer들에 여기서 나온 gradient를 더하기 때문) regularization 하기위해

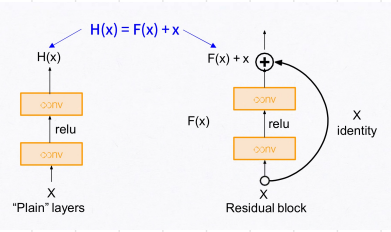
③ No FC layer

④ parameter 수 ↓↓ (5M)

5. ResNet : Very deep networks using residual connections

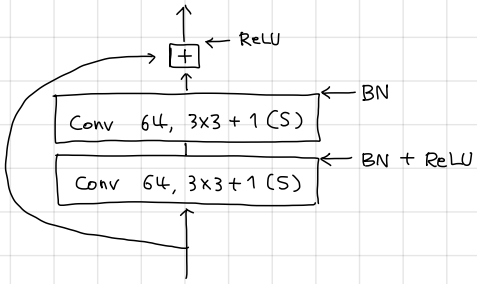
① 깊은 model이 얕은 모델보다 안 좋은 성능을 내는 경우

②



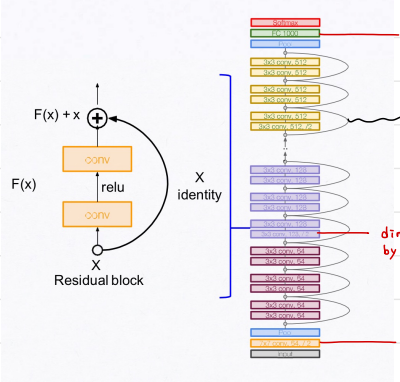
이미지 자체가 class를 대변

→ H(x) - y 가 아니라 H(x) - x를 최소화하는 방향



<ResNet의 residual block>

③ ResNet : Stack residual blocks

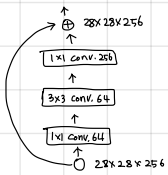


이전에 Pooling 해서 FC 1000 한 개로 출분

deep network에서는 여기에 bottleneck layer (1x1 conv) 추가해서 efficiency 향상

dim reduction by using stride 2 (일부 residual block에서) ⇒ map 수는 x2 , 각 map의 size ↓ → 이 경우 입·출력의 크기가 다르므로 1x1 conv 를 선형 연결에 추가해준다.

Beginning conv layer



(1) set weights to 0 → block computes I
↳ 필요없는 layer를 알기 쉬움
(2) L2 reg. drive all the params to 0.
ResNet must는 desirable thing!
(3) gradient highway → train easier & faster
→ converge well

④ Details :

(1) Conv 뒤에 BN

(2) ReLU 사용하므로 Xavier/2 initialization

(3) SGD + Momentum (0.9)

(4) lr : 0.1 로 시작. /10 at plateau ← BN 했기 때문에 lr 크게 설정 가능

(5) batch size : 256 , weight decay 1e-5

(6) dropout ⊗

6. Other architectures

① NiN (Network in Network) : Conv layer에 micro conv layer (Mlpconv.) 추가해서 더 abstract feature 뽑아냄

i.e. 1x1 conv

② Improving ResNets

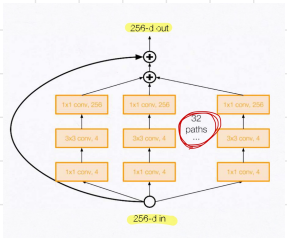
(1) skip해서 더한 후 ReLU (기존) → skip path에 ReLU 하고 더함 ; performance ↑
(activation)

(2) Wide Residual Networks : use wider residual blocks.

F filter ⟶ FxK filter ; filter 수 ↑ ⇒ 50-WideResNet이 152-ResNet 보다 좋은 성능

width를 늘리는 것이 depth 늘리는 것보다 computationally efficient (∵ parallelize 가능해지므로)

(3) ResNeXt :



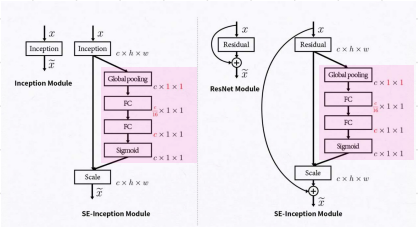
병렬적 process → performance ↑

(4) Deep networks with Stochastic Depth : train 시 random하게 residual block skip (Identity만 사용해서 넘어가는), test 시에는 다 사용.

⇨ gradient vanishing 완화, train 빠르게

(5) Multi-scale ensembling : Inception, Inception-ResNet, ResNet, WideResNet 앙상블

(6) SENet :



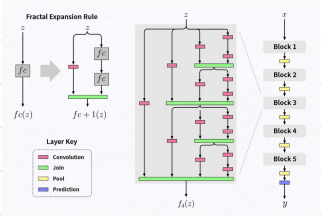
SE block을 Inception / Residual block에 추가해서

각 feature map의 weight (가중치, 중요도..? 정도로 이해)를 결정

↳ 관련 없는 feature는 값을 줄임.

③ Beyond ResNet

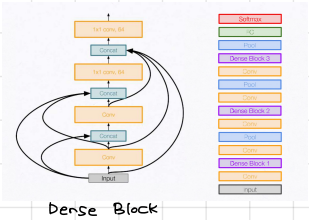
(1) Fractal Net : Sub-path drop해서 train. test 때는 전부 사용



→ shallow & deep path 사용

∵ shallow → deep transition이 중요한 것이라 주장

(2) Densely Connected CNNs



Dense Block

→ vanishing gradient 완화.

feature propagation 강화

feature reuse ↑

④ Efficient networks

(1) SqueezeNet : Alex Net과 비슷한 정확도, param 수, model size ↓

⑤ Meta-learning

(1) NAS : search space에서 architecture 뽑음 → train to get accuracy R → dp 구하고 R로 scale 해서 controller update
(w/ probability P) NAS

(2) NASNet : CIFAR-10 같이 작은 데이터셋에서 NAS 하고 그 모델을 ImageNet에 적용.