GoogLeNet (2014)

2021-12-05

1. Introduction

- 딥러닝은 idea, algorithm, network architecture로 발전해왔다.
- GoogLeNet은 적은 parameter로 더 정확도를 높였다.
- Efficiency : power와 memory 사용에서 효율성을 높였다.
- Deeper : Inception module 사용, network depth 증가

2. Related Work

- 당시 트렌드는 layer 수를 늘리고, dropout 을 하는 것이었다.
- Inception model에서 모든 filter는 학습한다.
- GoogLeNet은 22 layer
- Network-in-Network 에서 1x1 Conv: 연산 속도에 큰 손실을 주지 않으면서 depth와 width를 늘림
- R-CNN(Regions with CNN) (low-level 단서로 Detection 후보 찾기 + CNN Classifier) → 여기다 multi-box prediction + ensemble approach 추가

※ Network-in-Network에서 1x1 Conv

- 1x1 conv
- 28x28x192 input을 28x28x32로 줄이는 방법
  : 1x1x192 형태의 32개의 filter 이용

3. Motivation and High Level Considerations

- Network size 키울 때 문제
  : param 수가 많아져서 overfitting, computational resource 사용 증가
- 문제 해결을 위해 Sparsely connected architecture 사용
- Non-uniform sparse data structure을 사용하는 문제
  : 연산 시 비효율적이다.
- 따라서 Dense matrix의 연산을 이용하지만 실제론 sparsity 한 structure을 비슷하게 만들자.

➢ FC layer를 줄여서 sparsity를 만들고, Inception module 내부는 Dense하게 만들자.

4. Architectural Details

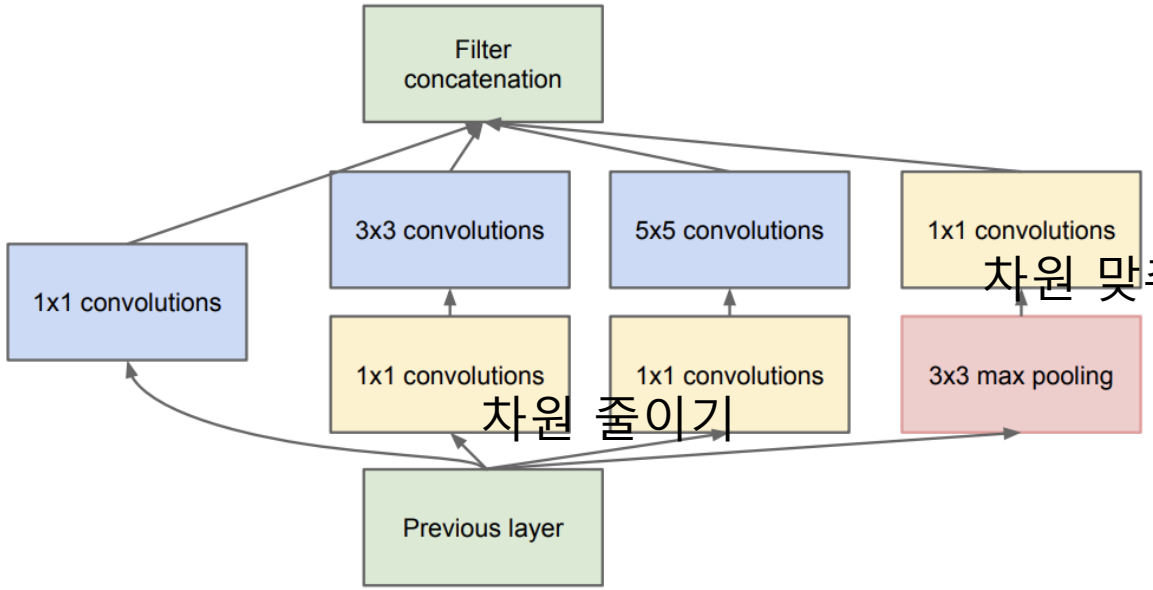
- Optimal local construction을 반복
- Layer-by-layer 구조 : 마지막 layer로 correlation statistics 분석하여 높은 correlation 가지는 unit으로 clustering(각 unit은 몇 몇 region에 해당
- Lower layer에서 correlated unit은 local region에 집중
- 각 stage마다 pooling을 해야 효과적
- Layer마다 output 과 연관성이 달라지므로 높은 layer일수록 spatial concentration이 낮아지므로 3x3, 5x5 Conv의 비율이 높아져야한다.

4. Architectural Details

- 5x5 Convolution과 pooling은 연산량이 많아짐
  : 이를 해결하기 위해 차원줄이기 적용
  : 연산많은 3x3 Conv, 5x5 Conv 전에 1x1 Conv로 차원축소
- 가끔 max-pooling을 stride 2로 해서 grid의 해상도를 절반축소
  (lower은 전통방식, higher은 inception layer 사용이 효율적?)
- 1x1 conv로 연산증가 막으면서 unit 수 크게 증가시킴
- 다양한 스케일로 진행하고 집계하기 때문에 다양한 스케일의 요약된 feature를 추출할 수 있다.
- 2~3배 빠르지만 수동으로 잘 디자인 해야한다.

4. Architectural Details

- Inception 모듈
  - 작은 여러개 필터로 연산량 줄임?



차원 맞추기 (pooling은 필터 개수 바꾸지 못함)

차원 줄이기

5. GoogLeNet

- 앙상블 기법 사용했다.
  : 7개 모델 중 6개 모델은 같은 구조, 다른 샘플링 방식 사용
- 차원축소 layer는 ReLU사용
- Classifier 전에 average pooling + extra linear layer 사용
  : FC 대신 average pooling → top-1 accuracy증가, dropout사용
- Auxiliary classifier로 lower stage 차별 줄임
  : loss는 discount weight로 얻어짐(0.3)
  : inference시 버림
    - Average pooling, 1x1 conv, FC layer(dropout 70%), softmax loss

6. Training Methodology

- DistBelief(분산 ML 시스템)사용, GPU로 학습
- Asynchronous sgd w/ 0.9 momentum
  8 epoch마다 4% 씩 learning rate 줄이기
  최종 모델의 inference time에는 Polyak averaging 사용
- 합친 모델들은 다양한 옵션으로 학습됨
  - 다양한 크기의 patch
    (crop, 사이즈 8%~100%, Aspect ratio 3/4 와 4/3 로 랜덤)
  - Photometric distortion 사용 (overfitting 해소)
  - Random interpolation method 사용(resize 위해)
  - 결론 : 뭐가 우리 최종 결과에 좋은 영향을 미쳤는지 모름

7. ILSVRC 2014 Classification Challenge Setup and Results

- 7버전으로 학습해서 합침
  initialization, learning rate policy 같게
  sampling method, input image order 다르게
- Cropping approach를 다양하게
  - 실제 적용에서는 불필요할 것
- Multiple crop과 각각의 classifier의 평균으로 최종 predict
- Top-5 error가 6.67%(당시 1등)

8. ILSVRC 2014 Detection Challenge Setup and Results

- Inception model로 인해 region classifier 처럼 동작
- Selective Search로 높은 recall 얻음
- Proposal 수 줄이기 : superpixel size 2배, 60%만 사용
- 6개 convNets을 앙상블 해서 accuracy 높임
- Bounding box regression, localization data 사용 안함
- 다른팀들 전략 : external data, ensemble, contextual model
  - External data : detection data로 정리된 classification data로 model pre-training

9. Conclusion

- Dense block으로 Optimal sparse structure
  : 연산에 효율적, 실현 가능, 유용
- Context와 bounding box regression 없이도 detection 성능이 나온것을 보면 inception architecture의 힘을 알 수 있다.

※ 궁금한점

- 데이터가 잘 나눠지면 sparse한건가? 데이터가 잘 나눠질때 sparse connected architecture를 사용할 수 있는건가?
- 현재 최적의 모델을 찾아주는 여러 모델들은 어떤 것이 있고 어떻게 구현돼 있을까?
- Lower layer 에서 traditional conv 스타일로, higher layer에서 inception module 사용하는게 좋은것은 higher layer에서 correlation이 높은 데이터끼리 모이기 때문인가?
- Localization data가 뭔가? Detection model을 pretrain 하는 부분이 잘 이해가 안된다.
- FC layer에서만 drop out을 하는 이유가 뭘까?
- Detection을 평가기준과 proposal을 찾는 방법에 대해서도 공부해야겠다.

※ 느낀점

- Auxiliary classifier로 vanishing gradient(lower layer까지 gradient가 전달되지 않는것)을 줄이는 것이 신기하다.
- CNN이 이미지의 feature을 점점 요약하는 느낌이다.
- 너무 다양한 시도를 해서 뭐가 우리 최종 결과에 좋은 영향을 미쳤는지 모른다고 쓴 것이 인상적이었다.
- Detection은 결국 proposal을 모은다음에 Classification 하는거였는데, proposal을 어떻게 모으는지 궁금하다.
- 당시에 효율성과 성능을 크게 잡은 것이 대단하다고 생각된다.
- 테스트를 할때 crop해서 넣는 것이 좀 특이했다.
- 아키텍처가 좋다는 것을 어떻게 판단할 수 있을지 궁금했다. 이 논문에서는 성능향상을 위해 너무 많은 것을 넣었기 때문에 성능 향상이 아키텍처의 영향인지 알수가 없다는 생각이 들었다. 뭔가 좋을거 같으니 해봤는데 좋더라의 느낌이었다. 아니면 일반인도 읽을 수 있게 쉽게 쓰다보니 설명이 다 안된건지 궁금해졌다.