Final Test

딥러닝 응용

21500333

서남진

2020.1.9

1. Usage

* Requirement (Google Colab환경에서 실행 가능)

pytorch >=

torchvision

Pillow

numpy

matplotlib

* Directory

Train\_dir = ./train

: Train dataset이 클래스 별로 분류되어 저장되어 있다.

Valid\_dir = ./valid

: Validation dataset이 클래스 별로 분류되어 저장되어 있다.

Output\_dir = ./output (os.makedirs로 자동 생성, 코랩에서 엑세스 거부되는 경우 수동 생성)

: log figure, weights파일이 저장되는 디렉터리이다.

* Code

preprocess.py

: 데이터 전처리를 위한 함수가 있는 모듈이다. 현재 데이터를 카운트하거나, train set과 valid set을 나눌 때, 그리고 over sampling을 구현하였다.

train.py, train\_over\_sampling.py

: 위의 세팅을 마치고 option(type: Dict)를 설정한 후에 실행시키면 된다. train\_over\_sampling.py는 오버샘플 코드가 함께 있는 파일이다. (디렉터리가 변하므로, train directory를 복사해서 사용할 것.)

test.py (train.ipynb)

: test를 위한 코드이다. model.load\_state\_dict부분에 웨이트 파일 디렉터리를 바꿔주고 실행하면 된다.

1. Model design

각각의 샘플들의 크기가 일정하지 않으며, 오브젝트의 위치와 크기, 개수들이 일정하지 않다는 것을 확인하였다. 모델은 이 문제를 해결하기 위해 충분히 복잡해야 하며, 트레인 샘플의 개수가 20000개 보다 작고 컴퓨팅 리소스를 충분히 가지고 있지 않으므로 너무 복잡하지는 않는 모델을 피하였다.

이 보고서에서 사용한 GargLeNet은 GoogLeNet을 기반으로 만들었으며, 1개의 stem layer, 3개의 inception, 그리고 fully connected layer로 이루어져 있다. GoogLeNet과 달리 auxiliary branch를 사용하지 않았으며, 5\*5 커널대신 3\*3 커널을 두개 사용하며 컴퓨팅 리소스를 고려해 세개의 inception만 사용하였다.

1. Stem layer는 데이터가 inception에 입력되기 전, 채널을 충분히 늘려 bottle Neck이 발생하지 않도록 한다.
2. Inception layer는 각각 네 개의 branch마다 다른 size의 kernel을 사용하고 다시 네개의 branch의 아웃풋을 concatenation하여 forwarding한다. GoogLeNet 논문에서 사용한 컨볼루션 커널은 1\*1, 3\*3, 5\*5 세 가지이며, 각각 branch에 1\*1 컨볼루션 레이어를 더해 연산량을 줄인다.
3. Fully connected layer는 GAP와 Dropout을 통해 구현했으며, 마지막에 Linear layer를 통해 한개의 클래스 결과값을 제출한다.

음식, 실내, 플레이트, 테이블이(가) 표시된 사진

자동 생성된 설명음식, 테이블, 식물, 플레이트이(가) 표시된 사진

자동 생성된 설명

그림 poutine(왼쪽), axoloti(오른쪽)

GoogLeNet은 여러개의 커널을 통해 다양한 스케일의 샘플의 피처를 구할수 있으며, 본 보고서에서 사용한 모델은 작은 커널과 1\*1 커널을 통해 계산량을 줄였으므로, 문제에 적합한 모델이라고 생각하였다.

1. Training techniques

* Data Augmentation

가지고 있는 트레이닝 개수는 총 19640개로 Data가 문제의 난이도에 비해 상당히 부족하다고 생각하였다. 그렇기 때문에 트레이닝 샘플이 부족하여 모델이 오버피팅이 되는 것을 피하기 위하여 Data augmentation을 사용하였다. 뿐만 아니라 Data augmenation은 모델의 동적인 학습을 위해서도 도움이 되는데 본 보고서에서 시도한 방식은 크게 세가지로 color jitter, random horizontal flip, 그리고 random rotation이 있다. 위의 세가지 방식을 랜덤으로 선택하여 학습에 적용하였다.

* Over Sampling

스크린샷이(가) 표시된 사진

자동 생성된 설명

그림 train set 개수(preprocess.py로 실행)

Training sample을 보았을 때, 데이터의 개수가 많지 않은 것도 문제였으나, 무엇보다 모든 클래스의 개수가 많이 차이가 나는 것도 문제 모델의 적절한 트레이닝의 문제 중 하나였다. 특히, 샘플의 개수가 100개도 안되는 클래스는 거의 학습이 안되기 때문에 이를 해결하기 위해 over sampling기법을 시도하였다. 100개 이하의 클래스는 Random choice를 통해 랜덤으로 샘플을 복사하였다. 300개로 정한 기준은 트레이닝 샘플이 너무 많아 redundancy에 의해 오버피팅이 발생을 막기위해 너무 늘리지 않았고, 작은 클래스에 의해 전체 트레이닝 양상이 변화하는 것을 방지하기 다른 클래스 보다는 작게 정하였다.

Data Augmentation을 적용함에 있어서 로스가 안정될때까지는 주어진 데이터셋만을 사용하여 학습시켜보고 점차 Data의 변화율을 높여가며 데이터 난이도를 증가하는 방식으로 학습을 시켜보고 그렇지 않은 경우와 비교해 보기도 하였다.

* Learning curriculum

보통 학습을 할 때, 무언가에 대해 모르는 상태보다 조금이라도 아는 상태가 더 학습에 유리하며, 완전히 특정 지식에 갇혀있는 상태보다는 완전히 이해하지는 못하는 상태가 더 학습에 유리할 것이라고 생각했다. 그렇기 때문에 학습에 난이도를 주어 학습을 계획 하였고, 학습의 난이도는 Data Augmenation을 통해 난이도를 조정하였다. 두번째 학습시에는 Data augmentaion을 2개의 기법을 동시에 사용하며, random rotation 각도를 늘리고, horizontal 비율도 증가시켰다.

* Trial & Error

위에서 언급한 techinique뿐만 아니라, Cross validation을 통한 Ensemble기법, 입력 크기변화, 오버피팅 시킨후 재학습 등의 방법들을 하였지만, 결과를 발전 시킬만한 성과를 보이지 않았다.

1. Experiments



표 실험결과

실험결과는 위의 표1과 같다. 전체적으로 epoch이 증가하면 증가할수록 성능이 증가하는 것을 확인하였다. 정확도가 80%를 넘지 않는 것은 아직 학습이 덜 되었음을 확인 할 수 있다. 주목할 결과는 learning curriculum을 정해 점차 난이도가 증가하였을 때에 정확도는 87.7098%로 그렇지 않을 때(82.9581%)보다 훨씬 상승하였음을 확인하였다.

또한, over sampling을 하였을 때 정확도는 82.1007%로 같은 epoch 실험들과 비교하여 거의 7%P 가까이 증가하였음을 확인하였으며, 이는 epoch을 40번 돌린 결과와도 견줄만한 성능이었다.

1. Conclusion

본 실험에서는 35개 클래스의 이미지를 분류하는 분류기를 만드는 것을 목적으로 하였으며, 실험에서 사용한 모델은 GoogLeNet을 기반으로 하는 모델을 사용하였다. 실험 결과, 최고 정확도는 87.7098%로 epoch 40, learning rate 0.001, 그리고 learning curriculum을 정하여 학습한 모델이 가장 좋은 성능을 보였다. 본 실험을 통해, Data augmentation, learning curriculum과 over sampling방식이 모델의 학습에 큰 영향을 미친다는 사실을 확인할 수 있었다. 그러므로, over sampling 방식을 통해 클래스간 균형을 마추고 데이터를 늘린 후, Data augmentation을 통해 동적인 학습환경을 만들어주고, learning curriculum을 정해 점차 모델이 방향성을 가지고 학습을 하게 만든다면 좋은 성능을 기대할 수 있을 것이라고 생각한다.