YOLO_V3 Mini Project

2020년 12월

Bit Academy
위 인 섭
박 병 호

목 차

1. 서론

1.1. 개발배경

YOLO_V3 coco data 내에 존재하는 객체가 아닌 새로운 객체를 YOLO에 학습시켜봄으로써 CNN을 통한 딥러닝학습을 경험해보고자 한다.

1.2. 개발의 필요성 및 목적

머신러닝에서 “차원의 저주”라는 말이 있다. 데이터에선 피처의 수가 곧 차원의 수이다. 데이터의 차원 즉, 피처의 수가 하나 늘어날수록 필요한 데이터의 양은 제곱으로 많아진다는 것이 바로 차원의 저주에 의미이다.

따라서 한정된 데이터로 효과적인 머신러닝 데이터를 얻으려면 데이터의 차원을 줄이려는 노력이 필수적이다. 이 과정이 바로 피처 추출이다

딥러닝 중 이미지 추출에 자주 사용되는 CNN(Convolution Neural Network)은 피처 추출을 위해 이미지 필터를 사용한다. 우리가 흔히 말하는 인스타 필터와 같은 뜻을 가진 필터이다. CNN은 다양한 컨볼루션 필터로 이미지를 단순화시키고(=Convolution) 변형(=activation)과 표본(=pooling)을 통해 이미지를 "축약"함으로써 저차원의 피처맵을 생성한다 이 과정이 CNN의 한 레이어에서 일어나며 여러 레이어를 거치며 점점 더 작은 피처가 생성되고 결국 마지막 예측까지 이루어지는 것이 CNN의 원리이다.

그렇다며 우리는 피처추출을 위해 이미지마다 어떤 필터를 적용해야 할까?

여기서 어떤 필터를 고를지 정하는 게 딥러닝이다 즉 컴퓨터가 행렬연산을 통해 이미지에 적합한 필터를 스스로 만들어내는 것이다.

우린 이번 미니프로젝트를 통해 머신러닝의 한 종류인 CNN을 이용하고 피처추출에 있어 필요한 양질의 데이터를 생산하고 데이터에 따라 변화되는 정확도를 비교해보고자 한다.

2. 개발방법

2.1. 개발환경

본 미니프로젝트는 Ubuntu 환경에서 진행하였으며 빠른 딥러닝 학습을 위해 GPU가 탑재된 데스크탑을 이용하였다.

YOLO의 버전은 V3를 이용하였으며 데이터의 Bounding Box 작업에는 YOLO_mark를 사용하였다. darknet 학습은 darknet53.conv.74가 이용하여 학습을 진행하였다.

2.2. 학습방법

학습에 사용된 객체의 종류는 고추 참치통조림, 아이폰 충전기, 라이터, 컵라면, 바나나우유, 가나 초콜릿, 칫솔 총 7개체를 선정하였으며 사진은 각각 A3 종이 위에서 촬영되었다 촬영된

이미지는 YOLO_mark를 이용하여 하나하나 수작업으로 Bounding Box를 입력해주었다

이후 yolov3.cfg파일을 우리가 학습하는 방식에 맞게 Batch, Width, Height, Filters, Classes, Random을 조정해주었으며 파일의 경로가 담긴 data파일과 객체의 이름이 담긴 names 파일을 작성하고 전처리된 이미지를 8:2 비율로 train.txt, validation.txt로 나눠주었다.

이후 학습 명령어인

./darknet detector train <data파일경로> <cfg파일경로> darknet53.conv.74 | tee backup/train.log 를 통해 학습을 진행하였다

3. 결과 및 고찰

3.1. 1차 학습 결과

- 학습시간 : 1hr
- 훈련횟수 : 1,000회
- 데이터 양 : 각 20장

1차 학습결과 객체를 잘 인식하지 못하고 log파일 확인결과 정확도가 40% 미만으로 매우 낮은값이 나왔다 1차 학습결과를 통해 충분한 학습시간이 필요하다고 생각 되었다

3.1. 2차 학습 결과

- 학습시간 : 2.5hr
- 훈련횟수 : 2,800회
- 데이터 양 : 각 20장

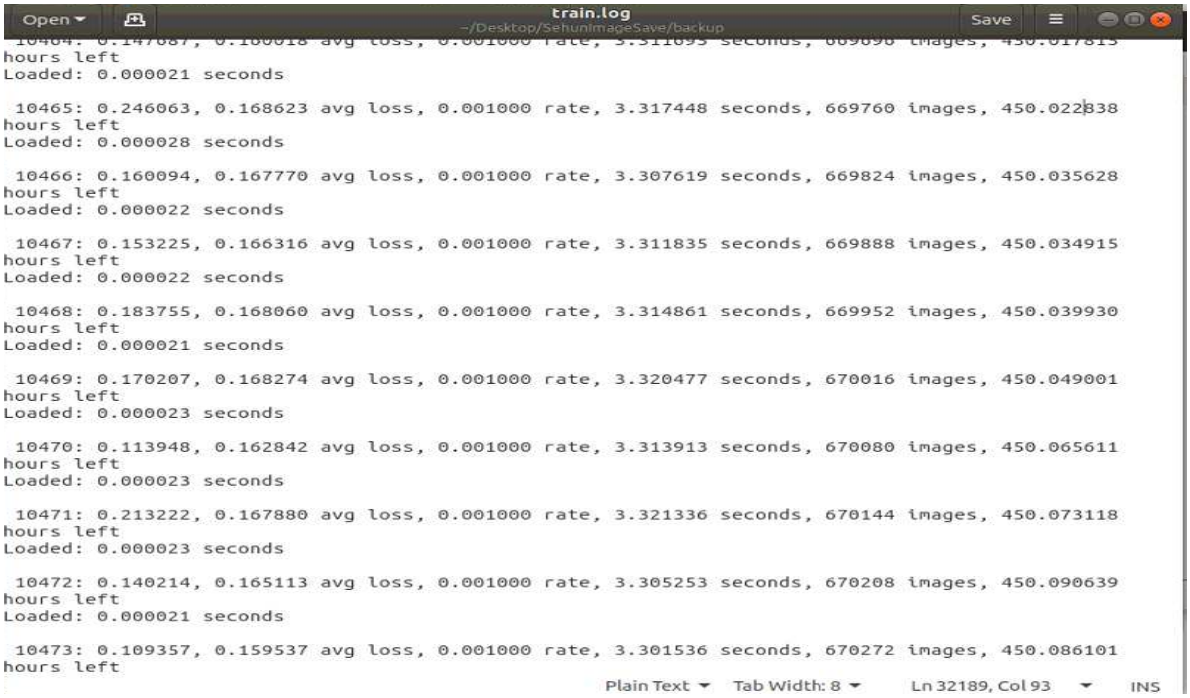
1차 학습결과를 바탕으로 시간을 늘려 학습을 진행하였다 그 결과 어느정도 객체를 인식하였으나 객체끼리 서로 헷갈리거나 각도에 따라 인식률이 떨어지는 문제점이 발생하였다

3.1. 3차 학습 결과

- 학습시간 : 7hr
- 훈련횟수 : 10,000회
- 데이터 양 : 600 ~ 1600 장

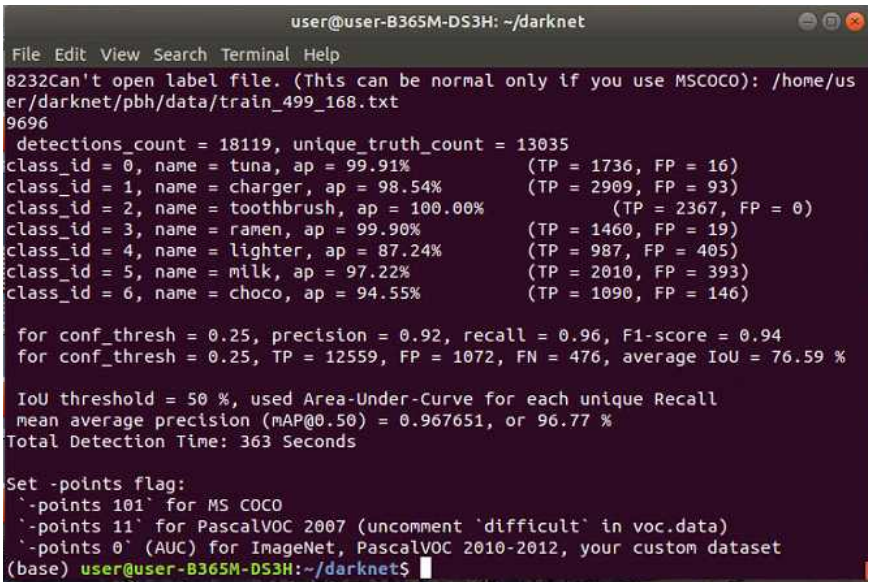
각도에 대한 문제와 정확도를 높이기위해 양질의 데이터가 더 필요하다고 느껴 데이터를 증식시키고 학습시간을 늘려 진행하였다 그 결과 1,2차와는 확연히 다른 인식률을 보여주었다

log 파일 확인 결과 정확도가 86 ~ 90% 사이로 계속 변했으며 마지막 log 파일 기록은 약 90%의 정확도를 가진 것으로 확인되었다.



정확한 정확도를 측정하기 위해 다음 명령어를 실행해보았다

./darknet detector map <data 파일경로> <cfg파일경로> <weights파일경로> <이미지경로>



4. 결론

이후 생성된 weights 파일의 성능을 확인하기 위해 웹캠을 통해 훈련한 객체를 인식시켜보았다. 전반적으로 객체를 잘 인식하였으나 칫솔의 경우 인식을 아예 못하거나 낮은 인식률을 보였으며 객체끼리 서로 붙어있을 경우 인식률이 떨어지는 경향을 보였다.

이는 다른 조의 학습 결과와 비교해볼 때 train Dataset 에 객체들이 함께 뭉쳐있는 사진이 적고 칫솔의 경우 더욱 다양한 Dataset 으로 추가학습이 필요한 것으로 생각된다.

데이터를 각도, 반전, 밝기 등의 변화를 주어 Dataset 을 늘린 뒤 기존 학습된 가중치 파일을 다음 명령어를 통해 재학습시킬 수 있다.

./darknet detector train <data 파일경로> <cfg파일경로> <weights 파일경로> -gpus 0,1,2,3

또한 학습된 가중치 파일은 다음의 명령어를 통해 직접 테스트해볼 수 있다.

사진으로 테스트

./darknet detector test <data 파일경로> <cfg파일경로> <weights파일경로> <이미지경로>

비디오로 테스트

./darknet detector demo <data파일경로> <cfg파일경로> <weights파일경로> <영상경로>