4조 상세 보고서

전기정보공학부 20학번 김도엽 (조장), 전기정보공학부 20학번 김재진, 전기정보공학부 20학번 하태윤

[1차: Yolo를 이용한 Object Detection]

한 일: Yolov4 학습

(1) Darknet 학습 준비

Darknet 프레임워크를 다운로드 받고 YoloV4에 맞는 label을 사용하기 위해 제공받은 convert.py 코드를 이용하여 xml label을 txt label로 변환하였다. 제공받은 make_list_cur.py 코드를 이용해 train.txt와 test.txt파일을 생성하였다.

(2) 정확도 향상을 위한 작업

학습 시 정확도 향상을 위해 같은 object인 클래스들끼리 하나로 묶었다(boat1~boat14 -> boat). 이때 변경된 class의 개수에 맞게 yolov4 config 설정도 바꿔주었다. 정확도 향상을 위해, pre-trained model을 yolov4.conv.137로, Yolov4-tiny의 경우 yolov4-tiny.conv.29로 설정하였다.

(2) Yolo V4, Yolo V4-tiny 학습

- ./darknet detector train converted_data/yolov4_drone.data cfg/yolov4_drone.cfg yolov4.conv.137 –dont_show –gpus 1,2,3 명령어를 이용하여 YoloV4 학습을 실행하였다.
- ./darknet detector train converted_data/yolov4-tiny_drone.data cfg/ yolov4-tiny_drone.cfg yolov4-tiny.conv.29 –dont_show –map –gpus 1,2,3 명령어를 이용하여 YoloV4-tiny 학습을 실행하였다.

(3) 학습 결과

V4 (Total Detection Time: 12 Seconds)
for conf_thresh = 0.25, precision = 0.99, recall = 0.99, F1-score = 0.99, average IoU = 87.95 %
IoU threshold = 50 %, mean average precision (mAP@0.50) = 0.984523, or 98.45 %
V4-tiny (Total Detection Time: 4 Seconds)
for conf_thresh = 0.25, precision = 0.98, recall = 0.99, F1-score = 0.99, average IoU = 83.45 %
IoU threshold = 50 %, mean average precision (mAP@0.50) = 0.975940, or 97.59 %

Image Augmentation

-> Robustness을 높이기 위해서, ImgAug library(https://github.com/aleju/imgaug)를 이용하여 각 Class의 전체 이미지 개수가 10000개 근처가 되도록 데이터를 증강하였다.

공부한 내용: 딥러닝 기초 공부, Yolo, Yolo 9000, Yolo V3, Yolo V4, R-CNN, Fast R-CNN, Faster R-CNN 논문 공부

Yolo를 통하여 진행하는 1차였던 만큼 이가 정확히 어떤 메커니즘을 가지고 작동하는지 이해하기 위해서는 딥러닝에 대한 기본적인 지식이 필요했다. 이를 위하여 Sung Kim 교수님의 유튜브 강의 "모두를 위한 딥러닝"을 시청하여 머신러닝의 기초에서부터 딥러닝의 원리 (gradient descent, backpropagation)으로 넘어와 끝으로 단순 CNN과 RNN의 내용까지 공부를 하였다. 이후 1차에서 사용하게 된 Yolo V3와 Yolo V4를 이해하기 위해서 Yolo 원본 논문에서 V4 논문까지 리딩을 진행하였다. 그리고 그 과정 속에 R-CNN에서 처음 제시한 개념인 anchor box에 대해서 이해하기 위하여 R-CNN에서 Faster R-CNN 논문까지 추가적으로 리딩을 하였고 이를 통하여 1차에서 darknet 프레임 워크 상에서 우리가 어떤 알고리즘을 이용하고 있는지 이해할 수 있었다.

[2차: Ultra96V2 보드를 이용한 Object Detection]

Ultra96-V2 보드 세팅

이번 대회에서 사용되는 ZYNQ 기반의 프로세서 Ultra96-V2에는 Avnet에 맞는 PYNQ를 SD 카드에 image를 설치하여 기본 운영 환경을 조성하였다. DAC-SDC 대회의 평가 기준 중 하나인 energy measurement를 위해 기존 파일들을 추가적으로 대회에서 제공한 파일로 대체하는 작업을 수행하였다. 제공된 script가 제대로 동작하지 않아, 수동으로 ultra.conf 파일과 BOOT.bin, image.ub를 제공된 파일로 대체하였다.

Object Detection Model 선정 과정

Ultra-96 보드에서 앞선 1차에서 이용되었던 Yolo를 사용하고자 하였다. Ground up으로 진행하기에는 시간적인 제한이 존재하였기에 Yolo를 C에서 구현한 코드를 참고하여 진행하였다. 허나 ultra96 v2 상에서 본 코드를 작동시키는 것도 v1과 v2의 버전 차이에 의하여 다양한 디버깅 노력에도 진행이 불가능하였고, 나아가서 커스텀 Yolo 모델을 만들기 위하여 학습을 하는 것도 불가능하게 (즉 pretrained weight만 사용 가능) 하드코딩 되어있는 상태였기에 드론에서 찍은 이미지에 추가적인 학습을 원하는 입장에서 사용 불가능하다고 판단하였다.
이외에도 hackster.io에서 제공하는 caffe로 구현된 Yolov3 Tiny를 진행하였지만 이 또한 v1과 v2의 버전 차이에 의하여 진행이 어려웠고 1차에서 텐서플로우와 파이토치에 어느정도 익숙해진 반면 caffe라는 생소한 언어를 통하여 디버깅을 진행해야 한다는 것이 벽으로 다가와서 역시 힘들다는 판단을 내렸다.
이렇게 1차에서 이용했던 Yolo 모델을 활용하고자 다양한 방식으로 접근하였으나 다양한 난관에 부딪혔고 이 때문에 더 이상 Yolo에 묶이지 않고 다양한 모델들을 이용하는 방향으로 틀게 되었다.

처음으로 시도했던 오픈 소스 모델은 SkyNet이었다. 이를 이해하기 위하여 SkyNet: a Hardware-Efficient Method for Object Detection on Embedded System 논문을 리딩하였고 이후 공개된 코드를 이용하여 학습과 배포를 진행하고자 하였다. 허나 여기에도 다양한 난관이 존재하였다. 공개된 코드에는 최신 버전과 구 버전이 존재하였는데 최신 버전에서는 학습이 불가능하게 막혀 있어 구 버전을 이용하여 진행할 수밖에 없었다. 그러나 구 버전의 모델은 기본적인 오류가 다수 존재하였고 이를 디버깅하여 해결하려고 하였지만 서버에서 작동 시 CUDA 등의 버전이 맞지 않아 sudo 키워드가 막혀 있는 리눅스 서버 상에서 진행이 불가능하였다.

결과적으로 위의 troubleshooting을 1달 반 내외로 진행하였고 마감이 약 1달 정도 남았을 때 마지막으로 도전한 것이 본 모델이다. 오픈 소스로 공개된 rectangular-half-squeezenet라는 모델을 우리 상황에 맞게 integrate하여 진행하기로 결정했다.

모델 활용

(1) 준비
서버에서 GPU를 사용하여 구동하기 위하여 라이브러리 버전 관리와 데이터에 알맞게 작동하게 만들기 위하여 코드를 수정하였다.

(2) 모델 학습
python3 halfsqueezenet_objdetect.py --output="/home/aiproj_4/spooNN/recthalfsqznet/result/" –gpu=0,1 --data=/home/aiproj_4/darknet/data_training --dump2_train1_test0=1
을 통하여 학습을 진행하였다. 이 과정에서 epoch 수에 따라서 오버피팅이 진행되는 것을 확인하여 epoch와 학습률을 조정하여 가장 높은 결과가 나오는 경우로 선택했다. 본 학습을 통하여 params.h와 configs.h 파일을 얻었다. 즉 학습된 네트워크가 존재하며 weight 값들이 배열로 저장되어 있는 상태이다.

(3) Ultra 96-V2에서 구동
위 학습을 통해 얻은 결과를 Ultra96-V2에 옮기기 위하여 Vivado HLS를 진행하였다. export XILINX VIVADO를 통하여 설치된 Vivado HLS와 코드를 연동시켰고, 원하는 makefile 세팅에서 make를 진행하여 보드에서 이용될 수 있는 형태의 weights.txt 파일을 얻었다. 이후 Ultra96-V2 모델에 알맞은 설정으로 RTL와 bitstream을 얻기 위하여 Vivado HLS 스크립트를 사용하면 된다. RTL이 생성된 후에는 IP로 패키징 되는데 이를 extract하여 .bit과 .tcl 파일을 얻으면 된다.
마지막으로 2020년 대회 형식에 알맞도록 코드를 작성하여 이미지를 불러오고 이를 네트워크에 feed forward 할 수 있도록 하였다. 코드를 보드에서 예시 이미지를 이용해 구동한 결과, 1초 내외의 시간에 알맞은 위치를 잡아내는 결과를 확인했다.

```
print('Total energy: ' + str(total_energy),'J')
```

```
Got nn_ctrl!
total_num_img: 7
images_to_process_in_batch: 7
Total processed images: 7
Total time: 1.091902494430542 seconds
Total energy: 4.14581728354 J
```

(실행 결과는 앞서 제출한 실행 결과 보고서에 존재합니다)

발전 가능성
현재는 오픈 소스 모델을 활용하였는데 이 모델에 대회를 준비하면서 읽었던 논문(최신 Yolo 논문들과 SkyNet 논문)들에 나온 최신 기술들을 첨가하여 더 발전시키고 싶다. 이에 맞도록 코드를 수정하여 진행하며 Ultra96 – V2의 컴퓨팅 파워와 소비전력에 맞추어 최적점을 알아보기 위하여 모델의 레이어 수를 조정하면서 추가적인 실험들도 필요해 보인다.