2023 졸업프로젝트 진행 보고서

주제: 3D Object Detection

2023. 06. 23. ~ 2023. 07. 28.

신상윤, 윤성우

진행 사항

* SECOND 모델에서의 데이터 흐름을 통해 모델 파악하기
* 모델 변경 실습
* 아이디어 제시

SECOND 모델에서의 데이터 흐름을 통해 모델 파악하기

SECOND 논문의 3.1. Network Architecture 섹션에 모델의 구조가 소개되어 있고, 모델은 아래의 구성요소 순서로 데이터를 처리한다.

화살표로 표시된 항목은 각 구성요소를 OpenPCDet에서 구현한 클래스를 나타낸다.

3.1.1. Point Cloud Grouping

3.1.2. Voxelwise Feature Extractor

🡪 MeanVFE

3.1.3. Sparse Convolutional Middle Extractor

🡪 VoxelBackBone8x

🡪 HeightCompression

3.1.4. Region Proposal Network

🡪 BaseBEVBackbone

🡪 AnchorHeadSingle

실제 학습을 진행하면서 모델을 통과하는 point cloud data를 살펴보며 모델을 구성하는 각 모듈(클래스)의 역할을 파악하려 했다. 현재까지 파악한 내용은 아래와 같다.

MeanVFE

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

MeanVFE에서는 batch\_dict['voxels']를 입력받고, batch\_dict['voxel\_features']를 출력한다. (이하 batch\_dict는 편의를 위해 생략)

(위 사진의 ['vfe\_features']는 ['voxel\_features']의 오타이다.)

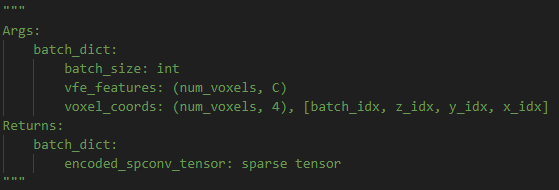
텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

['voxels']는 voxel 안에 들어있는 point 정보 (x, y, z, r)를 5개씩 (max\_points\_per\_voxel 개씩) 묶은 형태이고, ['voxel\_features']는 그 최대 5개의 (x, y, z, r)를 voxel 안에서 평균을 내어 저장한 것이다.

정리하자면, MeanVFE는 그저 각 voxel별로 point의 (x, y, z, r) 값의 평균만을 처리하는 모듈로, 논문에서의 voxel feature encoding 레이어에 해당하는 linear – batch normalization – ReLU 과정과 element-wise maxpool – locally aggregate 과정은 처리하지 않는다.

VoxelBackBone8x



VoxelBackBone8x에서는 ['batch\_size'], ['vfe\_features'], ['voxel\_coords']를 입력받고, ['encoded\_spconv\_tensor']를 출력한다.

(위 사진의 ['vfe\_features']는 ['voxel\_features']의 오타이다.)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

['voxel\_features']는 MeanVFE의 출력인 voxel의 point 정보 (x, y, z, r)을 가져온 것이고, ['voxel\_coords']는 그 voxel의 위치에 해당하는 coordinate (SECOND 논문에서의 spatial index)를 가리킨다.

참고로, point cloud의 x, y, z의 범위는 [0, 70.4], [-40, 40], [-3, 1]이고 voxel grid 간격은 각각 0.05, 0.05, 0.1이기 때문에, voxel grid의 크기 Depth Width Height는 1408 1600 40이 된다.

이 사실을 이용하여 위의 사진에서 ['voxel\_features'][0]의 좌표인 (12.0458, -1.9294, -0.2634)을 coordinate로 변환해보면

x 🡪 (12.0458 + 0) 0.05 = 240.916

y 🡪 (-1.9294 + 40) 0.05 = 761.412

z 🡪 (-0.2634 + 3) 0.1 = 27.366

라는 값이 나오기 때문에 ['voxel\_coords'][0]의 (x, y, z) coordinate가 (240, 761, 27)임을 알 수 있다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

voxel의 좌표 위치 정보가 담긴 ['voxel\_features']와 그 위치의 coordinate 정보가 담긴 ['voxel\_coords']를 이용하여 SparseConvTensor를 만든다. (참고: SECOND 논문 Figure 2.)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이렇게 만들어진 SparseConvTensor를 사용하여 sparse convolution을 수행한다.

참고로 각 sparse convolution layer는 sparse convolution – batch normalization – ReLU로 이루어져 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

각각의 sparse convolution의 수행 결과이다. channel을 늘리면서 spatial shape는 downsampling하는 것을 알 수 있다.







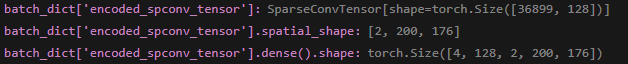








각 sparse convolution의 수행 결과를 sparse tensor가 아닌 실제 tensor로 살펴봤다. 각 convolution의 차원은 Batch Channel Height Width Depth인 것을 알 수 있다. (참고: SECOND 논문 Figure 3.)



VoxelBackBone8x의 출력인 ['encoded\_spconv\_tensor']의 형태이다. z-axis의 차원을 2로 줄여 2D로 변경하기 쉽게 만든 데이터를 출력한다.

HeightCompression

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

HeightCompression에서는 ['encoded\_spconv\_tensor']를 입력받고, ['spatial\_features']를 출력한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

['encoded\_spconv\_tensor']는 VoxelBackBone8x의 출력으로, Batch Channel Height Width Depth의 형태를 지닌 tensor이다. 이를 dense tensor로 바꾼 다음에, Channel(128)과 Height(2)를 곱함으로써 z-axis (Height) 차원을 없애고 새로운 Channel(256)을 가진 2D tensor를 만든다. 이것이 HeightCompression의 출력인 ['spatial\_features']로, Batch Channel Width Depth의 차원을 가진다.

BaseBEVBackbone

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

BaseBEVBackbone에서는 ['spatial\_features']를 입력받고, ['spatial\_features\_2d']를 출력한다.

(위 사진에서는 ['spatial\_features\_2d']가 표시되어 있지 않다.)



['spatial\_features']는 VoxelBackBone8x의 출력으로, Channel = 256이고 Heigh Width = 200 176인 tensor이다. 데이터가 2D로 바뀌면서 기존의 Width Depth 표현이 Heigh Width로 바뀐다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

위 사진은 BaseBEVBackbone의 구성을 보여준다. BaseBEVBackbone는 RPN의 feature map을 만들어주는 모듈이다. (참고: VoxelNet 논문 Figure 4.) 위의 구성의 list length가 모두 2이므로, 총 2개의 block을 거쳐 RPN의 feature map을 만든다는 사실을 알 수 있다.

첫번째 block에서는 stride = 1로 크기의 변함 없이, out\_channel = 128로 convolution을 5번 수행한 다음, 마찬가지로 stride = 1로 크기의 변함 없이 out\_channel = 256으로 transposed convolution을 수행한다는 것을 알 수 있다.

두번째 block에서는 stride = 2로 크기를 절반으로 줄인 다음, out\_channel = 256으로 convolution을 5번 수행한 다음, stride = 2로 크기를 두배로 늘린 다음 out\_channel = 256으로 transposed convolution을 수행한다는 것을 알 수 있다.



위 사진의 x.shape는 convolution의 결과를, ups[0].shape는 transposed convolution의 결과를 나타낸다.



위 사진의 x.shape는 convolution의 결과를, ups[1].shape는 transposed convolution의 결과를 나타낸다.



두개의 block에서 transposed convolution한 결과를 channel dimension으로 concatenate하면 RPN의 feature map이 완성된다. 위 사진은 concatenate된 RPN의 feature map을 나타낸다.



concatenate되어 만들어진 RPN의 feature map이 BaseBEVBackbone에서 출력된다.

AnchorHeadSingle



AnchorHeadSingle에서는 BaseBEVBackbone의 출력인 ['spatial\_features\_2d']를 입력받는다. 즉 RPN의 feature map을 입력받는다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

feature map을 이용하여 classification, bounding box, direction classification에 대한 prediction을 수행한다. 각 prediction은 모두 kernel size = 1인 convolution을 통해 수행된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위 사진은 prediction의 결과이다. classification의 class는 3가지 (Car, Pedestrian, Cyclist)이고, bounding box는 총 7가지 (x, y, z, w, l, h, ) 값을 가지고, direction classification은 0 degree, 90 degree로 총 2가지의 값을 가진다.

이때 위치별로 각각 anchor box가 6개를 가질 수 있으므로, classification은 63=18차원, bounding box는 67=42차원, direction classification은 62=12차원의 channel을 갖게 된다.

이렇게 구해진 prediction 값으로 loss를 구하여 학습을 진행한다.

anchor box가 6개인 이유는 아래와 같다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명



class는 총 3가지이고 direction은 총 2가지이기 때문에 32=6개의 anchor box를 가지게 되기 때문이다.

모델 변경 실습

기존 SECOND 모델의 ReLU activation 함수를 모두 4 Sigmoid activation 함수로 바꾸어서 학습시켜보았다.

기존 모델

* 학습시간: 186.80초 / epoch
* 성능
  + Car AP@0.70, 0.70, 0.70:

3d AP:88.4989, 78.2694, 77.0351

* + Pedestrian AP@0.50, 0.50, 0.50:

3d AP:56.5015, 53.2576, 47.7521

* + Cyclist AP@0.50, 0.50, 0.50:

3d AP:79.6927, 64.7743, 60.5385

변경 모델

* 학습시간: 354.73초 / epoch
* 성능
  + Car AP@0.70, 0.70, 0.70:

3d AP:87.0712, 77.5382, 76.3344

* + Pedestrian AP@0.50, 0.50, 0.50:

3d AP:52.7309, 49.7125, 45.9568

* + Cyclist AP@0.50, 0.50, 0.50:

3d AP:78.0391, 65.6828, 61.9870

평가

단순한 activation인 ReLU 대신에 복잡한 activation인 Sigmoid를 사용하여 학습시간이 2배 가량 늘어났다. 하지만 늘어난 학습시간에 비해 성능 개선은 일어나지 않았으므로, activation 함수는 ReLU에서 바꾸지 않는 것이 타당함을 알아냈다.

아이디어 제시

1. cost가 줄어드는 비율 그 자체를 학습에 넣으면 어떨까

ex1) 이번 iteration에서 cost가 줄어든 비율이 엄청 크다 (cost 2 🡪 1.5 : -25%) 🡪 현재 방향성을 더 살리자 (momentum 키운다 or RMSprop에서 speedup 한다)

ex2) 반대로 이번 iteration에서 cost가 오히려 늘어나거나 거의 차이 없다 (cost 2 🡪 2.1 : +5%) 🡪 방금 움직인 방향 반대로 간다

1. 깊이 깊어질수록 (input에 가까워질수록) gradient가 높아지도록 뭔가 조정하는 알고리즘 쓰면 vanishing gradient 효과가 줄어들지 않을까
2. 모든 gradient의 norm을 1로 고정시킨다
3. 마지막 레이어에 RNN 기법을 이용하여, 즉 마지막 레이어의 아웃풋을 recurrent하게 다시 마지막 레이어의 인풋으로 여러 번 넣는 짓을 하면 detection 작업이 좀 더 완고(robust)해지는 느낌이 나지 않을까, 약간 denoising 느낌으로, 아니면 약간 오버피팅 되는 느낌으로

마지막 레이어의 parameter들이 좀 더 오버피팅 될 것이기 때문에 좀 더 확신에 찬 느낌으로 detection을 수행할 것이라고 예상함

1. 코드의 vD, vH, vW를 0.05(x), 0.05(y), 0.1(z) 값을 바꾸면 어떤 영향이 생기는지 알아보기
2. 모델을 두개 이어붙인다.

첫 모델은 사람이 알아먹을 수 있는 feature를 내놓는 모델

두번째 모델은 그걸 가지고 detection 수행하는 모델

첫번째 모델이 vertical line detector 느낌 나게

(이러면 feature engineering한다는 소리인데 딥러닝의 의미가 없지 않나)

1. RPN을 2D로 다운시키지 말고, 3D 그대로 쓰면 어떨까

(우선 RPN이 뭔지부터 알아보자)

1. VoxelNet의 VFE에서는 point-wise feature에 inner-voxel 정보를 넣었다 (같은 voxel 안에 묶인 점들과 연관된 정보)

이를 확장해서, 주변 voxel들 (ex. 맞닿아있는 voxel 6개, 26개 등)과의 점들까지도 연관된 정보, 혹은 전체 점들과의 연관된 정보를 point-wise feature에 넣으면 어떨까

(Convolution과 의미가 겹쳐서 별로일 듯)

향후 계획

아직 살피지 못한 loss에서의 데이터 흐름을 추가로 파악해 SECOND 구조를 더 자세하게 분석한다.

제시된 아이디어 중 괜찮은 성능 개선이 기대되는 아이디어를 코드에 적용시켜본다.

질문

x, y, z에 해당하는 개념이 depth, width, height가 아니라 width, height, depth인가?

BEV는 2D data인가?

SECOND 논문에서 equation 5로 바꾸면 (summation index를 image pos가 아닌 kernel pos로 돌리면) GEMM이 안 된다는데, 이유가 뭔가?

해당 kernel pos (k)에 실질적으로 곱해지는, zero를 뺀 값들로만 이루어진 image col을 만들기 위해서? (empty point 아직 안 빼서 GEMM이 안 된다는 뜻??)

SECOND 논문 3.1.5 절에서 subscript t의 의미가 “여기서 사용할 target value”라는 뜻인가? (학습에서 나와야 할 goal value?)

g 첨자는 말 그대로 정확한 실제 값이고, a 첨자는 편의를 위해 고정시킨 대체 값 (ex. car = 1.6 x 3.9 x 1.56)인가?

코드 상의 vfe에서 도대체 무슨 일을 하는가, 그냥 말 그대로 point 평균만 치고 끝 아닌가?

FC 🡪 maxpool concatenate 🡪 maxpool 이 일을 어디서 하는가?