Object Detection Algorithms _ mid ppt

20213056_강민정, 20213062_김성윤, 20203505_김정연, 20203508_오세범

팀원소개

20213056_ 강민정

20213062_김성윤

20203505_김정연

20203508_오세범(조장)

Object Detection이란?

OBJECT DETECTION은 이미지나 비디오에서 특정 객체를 식별하고, 그 객체의 위치와 경계 상자(BOUNDING BOX)를 결정하는 작업

최근 자율 주행 자동차, 보안 시스템, 생체 인식, 로봇 공학 등 다양한 분야에서 해당 기술이 많이 활용되기 시작하면서 더 정확한 OBJECT DETECTION 알고리즘의 중요성이 높아짐

개발 동기

OBJECT DETECTION의 가장 고질적인 문제인 작은 객체에 대한 정확도 저조, 분산되어있는 이미지에 대한 정확도 저조에 초점을 맞춰 코드 수정을 진행함.

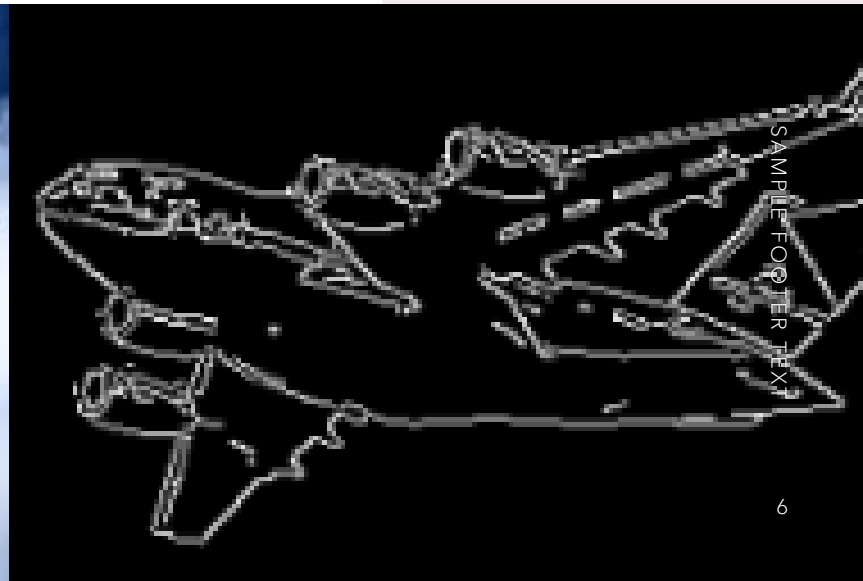
딥러닝 모델의 학습 및 추론에 사용하는 입력 이미지의 전처리 과정

```python
# 입력 이미지의 전처리를 수행하는 클래스
class DataTransform():
    """
    이미지과 어노테이션의 전처리 클래스. 훈련과 추론에서 다르게 작동한다.
    이미지의 크기를 300x300으로 한다.
    학습시 데이터 확장을 수행한다.

    Attributes
    ----------
    input_size : int
        리사이즈 대상 이미지의 크기.
    color_mean : (B, G, R)
        각 색상 채널의 평균값.
    """

    def __init__(self, input_size, color_mean):
        self.data_transform = {
            'train': Compose([
                ConvertFromInts(),  # int를 float32로 변환
                ToAbsoluteCoords(),  # 어노테이션 데이터의 규격화를 반환
                PhotometricDistort(),  # 이미지의 색조 등을 임의로 변화시킴
                Expand(color_mean),  # 이미지의 캔버스를 확대
                RandomSampleCrop(),  # 이미지 내의 특정 부분을 무작위로 추출
                RandomMirror(),  # 이미지을 반전시킨다
                ToPercentCoords(),  # 어노테이션 데이터를 0~1로 규격화
                Resize(input_size),  # 이미지 크기를 input_size × input_size로 변형
                SubtractMeans(color_mean)  # BGR 색상의 평균값을 뺀다
            ]),
            'val': Compose([
                ConvertFromInts(),  # intをfloatに変換
                Resize(input_size),  # 이미지 크기를 input_size × input_size로 변형
                SubtractMeans(color_mean)  # BGR 색상의 평균값을 뺀다
            ])
        }

    def __call__(self, img, phase, boxes, labels):
        """
        Parameters
        ----------
        phase : 'train' or 'val'
            전처리 모드를 지정.
        """
        return self.data_transform[phase](img, boxes, labels)
```

- Train의 compose를 사용해서
  여러 전처리 함수를 순차적으로 적용하는
  파이프라인 생성 → 이후 훈련과 추론 단계에서
  각각 다른 파이프라인을 사용할 수도 있음
- 해당 클래스에서 이미지와 이미지에 대한 추가 정보를 담고 있는 데이터를(annotation data)
  전처리 하여 모델에 입력으로 제공하기 위한
  변환 작업을 수행하기 위해 함수를 제공

딥러닝 모델의 학습 및 추론에 사용하는 입력 이미지의 전처리 과정 - 개선안

- 지난번 개인 과제에서 인식이 잘 되지 않았는데 경계선을 더 뚜렷하게 학습시키는 방향으로 생각하게됨
- 대상 픽셀을 강조하는 커널을 만들고, fliter2D를 사용, 이미지에 커널을 적용함
  이 때 중앙 픽셀을 부각하는 커널을 만들면 이미지의 경계선에서 대비가 더욱 두드러지는 효과가 생김
- 히스토그램 평활화를 사용하여 객체의 형태가 두드러지도록 만들어줌 → 관심 대상을 다른 객체나 배경과 잘 구분되도록 해줌
- 경계선 감지 기술 사용 → 정보가 적은 영역 제거 & 대부분의 정보가 담긴 이미지 영역 구분 가능



데이터 전처리과정 함수

- **이미지 회전** (Image Rotation): 이미지를 임의의 각도로 회전 → 객체의 다양한 방향에서 학습 가능
- **이미지 이동** (Image Translation): 이미지를 수평 또는 수직으로 랜덤하게 이동 → . 객체의 위치 이동에 대한 학습 강화 가능
- **이미지 자르기** (Image Cropping): 이미지에서 랜덤한 부분을 제거(자르기) → 객체의 일부가 가려진 상황에 대한 학습 강화 가능
- **가우시안 노이즈 추가** (Gaussian Noise): 이미지에 가우시안 분포를 따르는 잡음을 추가 모델의 노이즈 내성 & 일반화 능력 향상
- **데이터 축소** (Data Dropout): 랜덤하게 이미지를 제거 or 픽셀 값을 변경 데이터의 일부 정보를 제거 → 모델의 견고성 향상.
- **회색 음영** (Grayscale): 이미지를 회색 음영으로 변환컬러 정보를 제거 → 모델이 명암 정보를 주로 사용, 객체를 인식하게끔 함
- **배경 추가** (Background Addition): 이미지에 다른 배경을 추가 → 객체와 배경의 경계 뚜렷하게 함. 객체의 분리를 도와 모델이 객체를 정확하게 인식하게 함
- **배경 모델링** (Background Modeling): 객체를 다른 배경으로 이동 or 합성 → 다양한 배경에서의 객체 인식 강화 가능
- **스케일 조정** (Scale Variation): 이미지의 크기를 랜덤하게 조정 → 객체의 크기 변화에 대한 학습 강화 & 다양한 크기의 객체를 처리하게 보조

데이터 전처리과정 함수를 어떻게 확장할 것인가?
- VGG보다 더 나은 아키텍쳐는?

- ResNet
  - 깊은 신경망을 효과적으로 학습시키기 위해 도입된 잔차 학습 기법을 사용하는 네트워크 → 깊은 구조에서도 그래디언트 손실 완화 & 더 깊은 네트워크 구성 가능

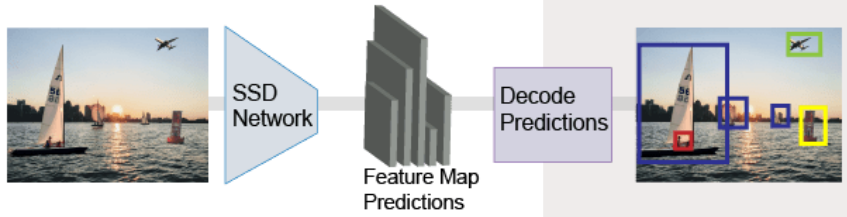
- MobileNet
  - 경량화된 네트워크 아키텍처 → 모바일 기기 & 임베디드 시스템에서의 실시간 객체 탐지에 특화
  - 작은 필터와 깊이 별 분리 합성곱을 사용 → 파라미터 수 & 연산량 감소 BUT 상당한 성능을 유지 = 실시간 객체 탐지에 적합한 네트워크

- EfficientNet
  - AutoML과 네트워크 스케일링 기법을 통해 자동으로 최적화된 네트워크 아키텍처를 생성하는 알고리즘 → 모델의 규모를 고려하여 깊이, 너비 및 해상도를 최적화함으로써 계산 비용을 줄이면서 성능을 향상

주어진 인공신경망구조

객체탐지를 위해 사용하는 SSD의 인공신경망 구조

- VGG **네트워크**
  - 34개의 레이어로 구성됨 (Convolutional Layers, Max Pooling Layers, Additional Layers, Convolutional Layers, ReLU Activation)
- **추가적인 합성곱 계층**
  - 패턴을 따라 채널 수와 커널 크기를 변환. 일부 층은 스트라이드와 패딩 사용 → 공간적인 크기를 조절한다.
- L2Norm **계층**
  - 특정 위치에서 특성 맵의 정규화를 수행
  - 모델이 안정적으로 학습할 수 있도록 하기위해 사용
- **객체의 위치와 클래스를 예측하는 합성곱 계층**



주어진 인공 신경망구조 설명
- 우리 팀만의 인공 신경망은?

- Backbone Network
  - 인공신경망의 백본 아키텍처를 변경하거나 개선 (EX - 더 깊은 or 넓은 컨볼루션 레이어 추가)
- Feature Extraction
  - 객체 감지 위한 특징 추출 단계를 수정하여 다양한 특징 추출
  - FPN, DFF등을 적용 → 더 다향한 특징 추출, 성능 향상
- Head Architecture
  - 객체 감지를 위한 헤드 아키텍처를 수정
  - 추가적인 컨볼루션 레이어, 다양한 스케일 및 비율의 앵커 박스, 다른 활성화 함수(ReLU, Mish, Swish 등) 사용 → 헤드 아키텍쳐 개선

- **손실 함수**
  - 새로운 손실 함수를 고려
  - 앞서 설명한 Smooth L1 Loss, Focal Loss, GIoU Loss 등을 사용 or 새로운 손실함수 개발
- **후처리 방법**
  - 비최대 억제 (Non-Maximum Suppression)
  - 객체 후보의 점수 조정, 객체 경계 상자의 세밀한 조정 등 수행 → 감지 결과의 정확성 향상
- **모델 크기 및 효율성**
  - 작은 모델 아키텍처 (Lightweight 모델)를 사용 + 모델 압축 기법 (프루닝, 양자화 등)을 적용
  → 모델의 크기를 줄이고 추론 속도를 향상

Training 함수에서 Optimizer 분석

```
# 4. 손실함수 및 최적화 기법의 설정

# 손실함수의 설정
criterion = MultiBoxLoss(jaccard_thresh=0.5, neg_pos=3, device=device)

# 최적화 기법의 설정
optimizer = optim.SGD(net.parameters(), lr=1e-3,
                      momentum=0.9, weight_decay=5e-4)
```

- 코드에서 손실 함수는 MultiBoxLoss를 사용하고 있다.
- 최적화 알고리즘은 SGD를 사용하고 있다.

Training 함수에서 Optimizer 분석
- 하이퍼 파라미터 분석

- **학습률**
  - 최적화 알고리즘이 교육 중에 모델의 매개변수를 업데이트하는 단계 크기 제어 → 모델이 데이터에서 학습하는 속도 결정
  - 학습률이 너무 높으면 훈련이 불안정해질 수 있고, 학습률이 너무 낮으면 수렴이 느려지거나 지역 최소값에 갇힐 수 있음
- **배치 크기**
  - 최적화 알고리즘의 한 반복에서 처리되는 훈련 예제의 수 → 훈련 중 메모리 사용량과 매개변수 업데이트의 매끄러움 정도에 영향을 미침
  - 배치 크기가 클수록 업데이트가 더 안정적일 수 있지만 더 많은 메모리가 필요할 수 있으며 배치 크기가 작을수록 노이즈가 더 많이 발생하지만 수렴 속도가 빨라짐
- **가중치 감쇠** (= L2 **정규화**)
  - 모델에서 큰 가중치 값을 권장하지 않는 정규화 기술
  - 더 작은 가중치를 장려하는 손실 함수에 페널티 항을 추가하여 과적합을 방지하는 데 도움이 됩니다.
- **에포크 수**
  - 훈련 중에 모델이 전체 훈련 데이터 세트를 반복하는 횟수 & 데이터를 통과하는 전체 통과 횟수
  - Epoch가 너무 적으면 Underfitting이 발생할 수 있고 Epoch가 너무 많으면 Overfitting이 발생할 수 있음
- **드롭아웃 비율**
  - 모델의 무작위 단위가 일시적으로 "드롭아웃(탈락)"되는 정규화 기술 → 과적합을 방지하는 데 도움
- **네트워크 아키텍쳐**
  - 레이어의 수와 크기, 레이어 유형(컨볼루션, 순환 등), 활성화 기능을 포함
  - 아키텍처 선택은 특정 문제의 복잡성과 요구 사항을 기반으로 해야 함
- **초기화 방법**
  - 훈련 과정에 상당한 영향을 미칠 수 있음 → 모델의 수렴 속도 및 전체 성능에 영향

Training 함수에서 Optimizer 분석
- 더 나은 Optimizer가 있을까?(1) - 최적화 알고리즘/손실함수 변경

- Momentum
  - SGD에 관성을 추가한 알고리즘
  - 이전 업데이트의 방향 고려 → 일정한 방향으로 진행하도록 보조
  - Momentum은 지역 최솟값에서 탈출하고 빠른 수렴 가능
- AdaGrad
  - 학습률을 각 매개변수에 적응적으로 조정하는 알고리즘
  - 빈번하게 나타나는 매개변수에 대해서는 학습률을 감소 → 안정적인 학습 가능
  - BUT 오랫동안 학습된 매개변수의 경우 학습률이 지나치게 작음 → 학습이 느려질 수 있음
- RMSprop
  - AdaGrad의 단점을 보완하기 위해 개발된 알고리즘 → 더 최근의 그래디언트에 더 많은 중요성을 부여하여 학습률을 적응적으로 조정
  - 학습률 감소 문제 개선으로 AdaDelta 사용도 가능
- Adam
  - RMSprop과 Momentum을 결합한 알고리즘

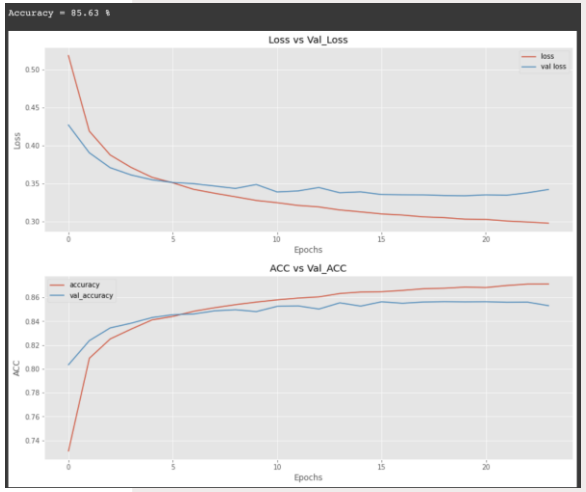
- MSE
  - 평균 제곱 오차는 회귀 작업에 널리 사용되는 손실 함수
  - 예측된 값과 실제 값 사이의 평균 제곱 오차를 측정
  - 예측된 값과 실제 값 사이의 평균 제곱 오차를 최소화하도록 함
- Smooth L1 Loss
  - 바운딩 박스 회귀를 포함한 회귀 작업에서 사용
  - MSE 이상치에 민감한 문제 해결
- Cross-Entropy Loss
  - 다중 클래스 분류 작업에 사용
  - 예측된 클래스 확률과 실제 클래스 레이블 사이의 차이를 측정

수행일정

- 실습실 컴퓨터 2대를 사용하여 한 대는 큰 함수 교체, 다른 한대는 파라미터, 변수 수정하는 방향으로 수행
- 최대한 자주 실습실에 방문하여 학습 진행상황 확인
- 새로 수정한 사항 있으면 클라썸에 게시, 대면/온라인 회의때 설명

기술개발 동향

Object Detection – try & error
- 최적화 알고리즘 교체



Adam 성능표

- 기존에 사용한 Adam을 rmsprop으로 교체

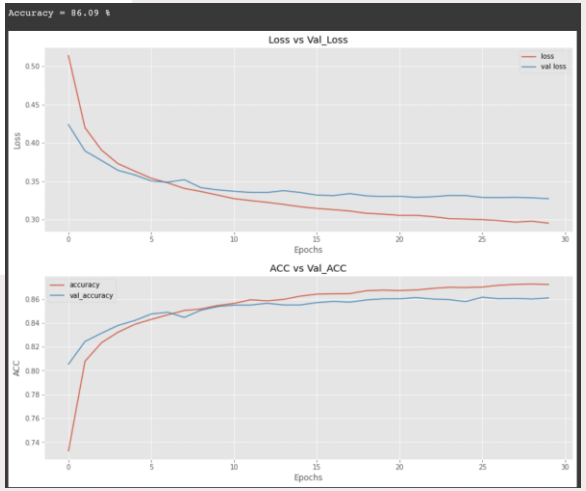
  → Adam 자체가 rmsprop과 Momentum의 결합 알고리즘이다 보니 성능이 낮아짐.

  높아진 loss를 확인, 다시 원래 최적화 알고리즘인 Adam으로 고정

- Adam을 NAdam으로 교체.

→ NAdam은 기존 Adam의 구조인 RMSProp + Momentum 대신 RMSProp + NAG을 사용한 것. NAG은 Momentum을 개량한 것으로 조금 더 공격적으로 하강하는 방식을 채득함. 찾아 본 자료에선 Adam 보다 좋은 성능을 보여준다고 되어있었으나 우리 코드에서는 성능이 20정도 떨어진 것을 확인(42.916)

→ 원인 : NAG의 계산법은 가중치와 관성을 바꿔가며 계산을 하는데 이는 역전파를 사용하는 목적과 정반대의 결과를 산출. 즉, 신경망에는 적합하지 않은 최적화 알고리즘



NAdam 성능표

Object Detection – try & error
- Default Box 수정

- Object Detection 모델이 작은 개체에 대해서는 정확도가 낮아지는 것을 확인, 해당 문제를 수정하기 위해 박스의 크기를 작게 조정하여 모델이 작은 개체를 더 잘 탐지하게끔 하고자 함.
- bbox_aspect_num: [4, 6, 6, 6, 4, 4] -> [3,4,4,4,3,3] 으로 교체

  → Tensor의 크기(expanded size)가 기존의 크기와 일치하지 않는 문제 발생.
- aspect_ratios: [[2], [2, 3], [2, 3], [2, 3], [2], [2]] -> [[2,3], [2, 3], [2, 3], [2, 3], [2,3], [2,3]] 으로 교체

→ 위의 에러를 수정하기 위해 화면비를 수정해보고자 함. 여전히 같은 종류의 문제 발생함

→ 기존의 크기와 일치시키기 위해 코드의 다른 부분들도 수정했지만, 계속하여 같은 종류의 에러가 발생, 원래대로 복구

Object Detection – try & error - epoch 수정

- 기본 제공 코드에서 최대 epoch을 300으로 설정, 이후 다양한 epoch에서 vaild를 시도해봄

  → 150, 300일 때 약 69map 이 나왔고 200일때 약 70map 이 나옴

- 200을 넘어가면 과적합이 일어나는 것 같다고 판단, 조원들과의 회의 이후 모든 실험에서 epoch은 200으로 고정하여 실험함

Object Detection – try & error
- feature_map 수정

- Default Box 수정과 마찬가지로 작은 물체 탐지의 한계를 수정하고, 연산과정을 조금 더 단순화 하여 모델의 속도를 올리고자 feature_map을 조정함
- 19 x 19 feature 를 추가 → label로 수정 전에 49.498map 나오는 것을 확인
- 성능을 높여보고자 SSD300의 input size를 512로 수정, 19 x 19가 추가된 SSD512를 구현해 보고자 함 → 성능이 40.778 map가 나오는 것을 확인. 성능 개선이 어려울 것으로 예상되어 해당 feature는 다시 원상 복구함
- Feature 2개 추가를 시도 → 에러 내용이 너무 복잡하고 수정을 진행해보았지만 난이도가 있다고 판단 feature 추가는 포기

Object Detection – try & error
- Data agumentaition 수정

- nn.torch의 RandomRotation을 추가해보려함. 해당 함수는 이미지를 무작위로 회전시키는 증강기법. 학습 데이터를 다양한 각도로 변형하여 모델의 일반화 능력을 향상시키고자 함

  → 함수를 만들어내는 과정에서 많은 시간이 필요, box와 label control을 수정하는게 어렵다고 판단해서 포기.

- Data agumentation code에서 일부(ToAbsolutecoords, ToPercentscoords, Subtractmeans)를 제거

  → 셋 중 하나만 제거해도 map 성능이 현저하게 떨어짐 (0 또는 1점대)

Object Detection – try & error
- MultiBox_loss 수정

- 기존 MultiBox_loss 는 nn.CrossEntropyLoss()와 nn.L1Loss()를 사용하고 있었으나 Smooth L1 Loss 로 교체 시도함
- 기존 함수의 구조 전체를 이후 손실을 계산할때 output, target 에서 예상 손실, 예상 점수, 박스와 라벨을 인자로 받게 수정함.
- 하지만 이 수정사항의 경우 해당 MultiBox_loss 함수에 맞게 SSD 모델또한 수정할 필요가 있어 결국 수행되지 못함.

Object Detection – try & error
- Backbone 수정

- Backbone을 사전 학습된 resnet50으로 수정해보려 함
- chanel, kernel, weight size를 맞추는 과정에서 weight size 오류가 고쳐지지 않아 다시 원래로 수정.

Object Detection – try & error
- vgg map 수정

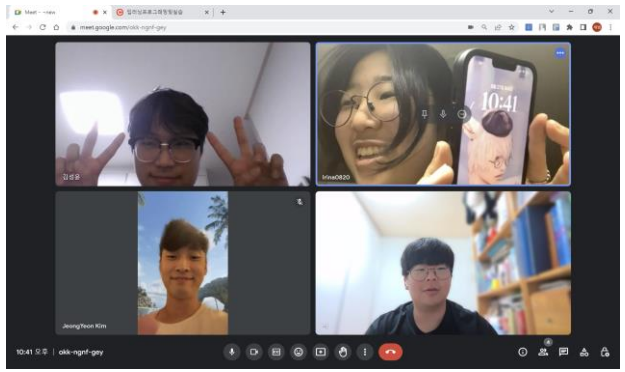
- 계산 복잡성을 줄이기 위해 깊이 별 컨볼루션 층을 VGGNet보다 더 많이 사용함.
- 이미지 shape 불일치: MobileNet은 VGGNet 보다 일반적으로 작은 차원의 입력이미지 사용 → 동일한 입력형태를 사용하자 불일치 에러 발생
- 레이어/파라미터 누락: VGGNet의 사전 훈련된 가중치를 로드하자 레이어, 파라미터의 호환 에러가 발생 → MobileNet에 맞는 가중치를 찾기에는 시간적, 정보적 한계가 있어서 결국 기존의 VGGNet 사용하기로 함

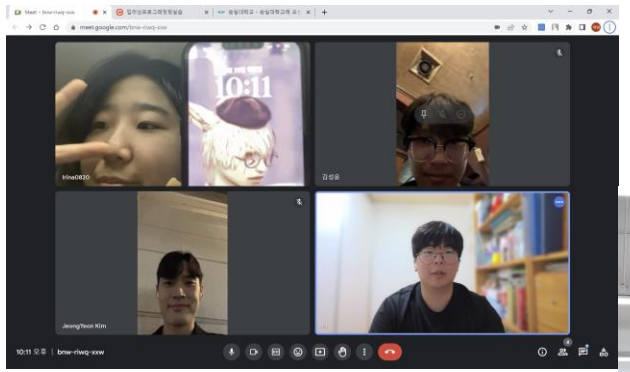
Object Detection – try & error
- ReLU6 추가

- ReLU는 입력 값이 양수인 경우에는 그대로 출력하고, 음수인 경우에는 0을 출력하는 활성화 함수로 계산이 간단하고 효율적이라는 장점이 있음
  하지만 음수 입력에 대해서는 뉴런층이 활성화 되질 않아 이런 뉴런들이 많아지면 표현능력이 제한됨
- ReLU6는 음수 입력에 대해서도 일정 범위 내에서 비선형성을 유지할 수 있게끔 음수 입력 값을 일정 범위 내에서 제한하여 비선형성을 유지하도록 ReLU함수를 개량한 것으로 수 입력에 대해서도 일정 범위 내에서 비선형성을 유지할 수 있음
  물론 음수 값을 손실하는 문제는 동일하지만 논문을 참고한 결과 ReLU6의 성능이 더 좋음을 알 수 있었음
- 하지만 실제로 적용해 보았을때 weight 파일이 비어있다는 오류가 계속하여 발생.
  → scale이 너무 커져서 발생하는 에러로 생각됨. 결국 다시 ReLU 로 고정

최종 결론

- 여러가지의 시도를 해보았지만 기존에 제공된 코드의 함수를 아예 교체하거나, 새로운 손실함수를 사용하는 것은 성능 향상에 큰 영향을 끼치지 못했음
- 오히려 영향을 끼친 부분은 학습횟수, 가중치, 학습률 변화로 관찰됨
- 이미 전체적인 코드의 구조가 SSD 구조와 연관되어있는데 사전에 조사한 손실, 최적화 함수들은 우리 코드의 SSD 구조에 적용시키기에는 어려움이 컸음.
- 데이터 증강또한 많이 할 수록 학습에 도움이 되었기에 삭제하기보다는 새로운 함수를 추가해보려는 방향으로 진행했으나 마찬가지로 전체 구조에 맞지 않아 기존의 코드와 많이 달라지지 않았음





대면 회의록

1) 대면 미팅 1시간 진행 이후 시간이 부족하다고 판단 되어 온라인 미팅으로 보강

2) 강민정 학우님 제외 모두 실습실에서 개발환경 구축
서로 이해가 안가는 부분 혹은 어려운 부분은
다른 팀원들이 도와주었음

3) 서로 시간이날때마다 실습실에서 모여서 코드를 수정함

개발하면서 느낀점

구현할 때 최근에 나온 논문들을 많이 참고 했는데 그중 mobilenetv2를 사용해보고 싶었으나 ssd 구조가 많이 달라 결국 적용에 어려움을 겪었음.

해당 아키텍쳐가 높은 탐지 성능을 보여줄 것으로 예상되었는데 적용하지 못해 아쉬움이 남았음

역할 분담

- 강민정 학우님 - PPT 제작, Multibox_loss 함수 수정, 프로젝트 관련 논문 검색
- 김성윤 학우님 - 전체 전반적인 코드 수정, 성능 관측, 코드 설명, 파라미터를 다양한 방식으로 수정, 성능 측정, 가장 먼저 새로운 코드 성능 측정
- 김정연 학우님 - VGG 아키텍쳐 교체, ReLU6 교체 집중, 이외에도 다양한 시도
- 오세범 학우님 - 최적화 알고리즘 Default Box 수정, 이외에도 다양한 시도

출처

1. Adam & NAdam 성능표 - https://velog.io/@hyunicecream/4가지-옵티마이저에-따른-성능평가-비교Adam-vs-Nadam-vs-RMSProp-vs-SGD
2. ReLU6 비교 - https://arxiv.org/abs/1801.04381
3. 논문 해석 참고 - https://minimin2.tistory.com/43