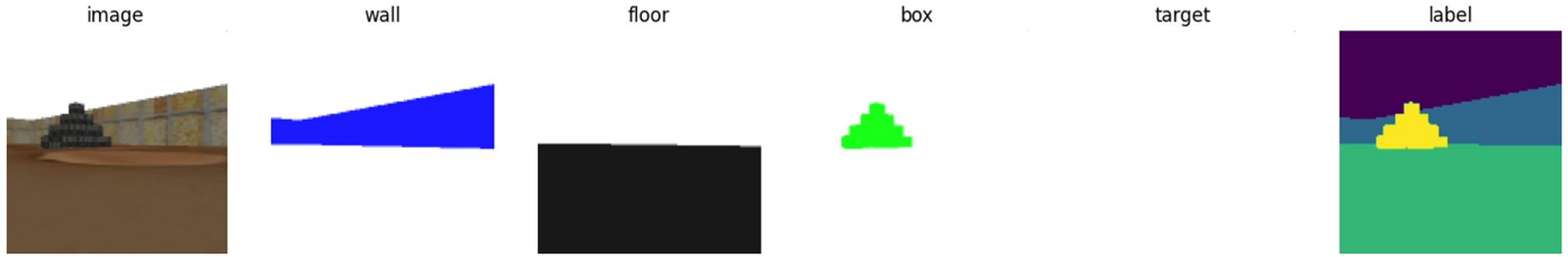
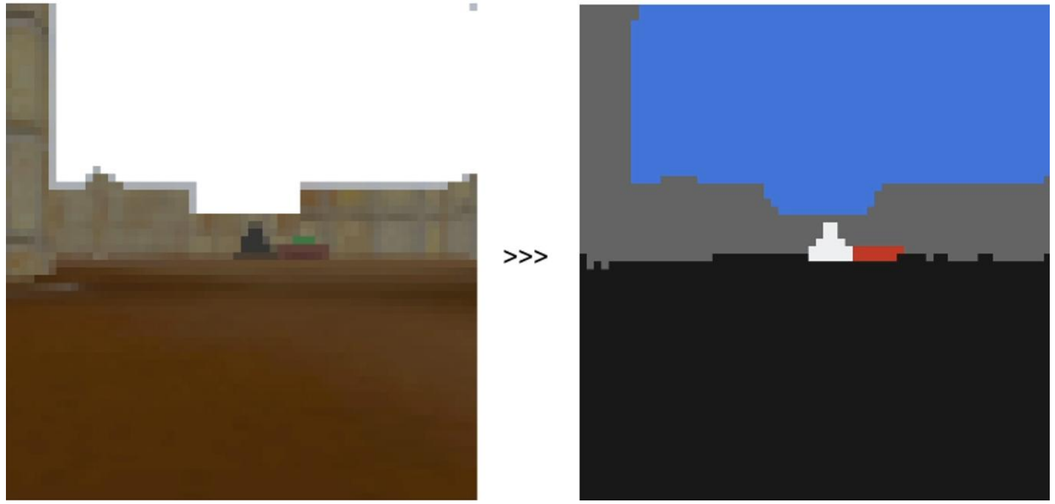
2023.05 - 2025.06

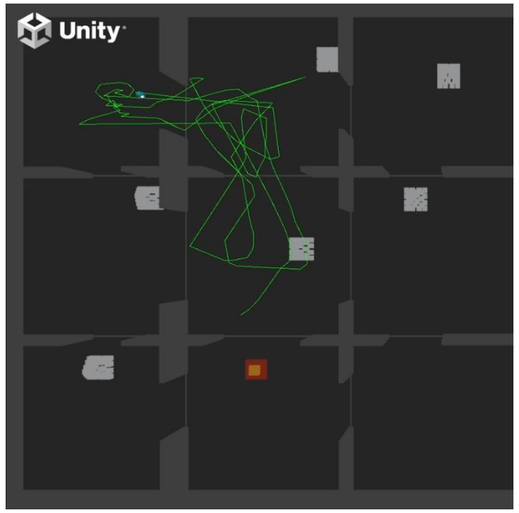
Artificial Hunter Vision #아주대학교 AI 집중교육 프로젝트 - 팀장



라벨 데이터 생성



실시간 이미지 세그멘테이션



강화학습

후기

높은 난이도의 기획과 구현을 처음부터 끝까지 주도한 프로젝트였습니다.

Unity ML-Agents와 Python 기반 이미지 세그멘테이션 모델을 연동하며,강화학습 성능 저하의 원인이 정책이 아닌 지각 파이프라인에 있음을 스스로 분석하고 구조를 재설계했습니다.

데이터 생성, 모델 학습, 추론 서버와의 통신, 그리고 에이전트 관측 구조까지 전 과정을 반복적으로 개선하며 인공지능 시스템을 블랙박스가 아닌 엔지니어링 대상으로 다루는 경험을 할 수 있었습니다. 이 프로젝트를 통해 복잡한 문제를 끝까지 책임지고 해결하는 개발자로서의 태도와 기준을 확립했습니다.

작업

1. 강화학습 실험을 위한 학습 환경 구축 및 파라미터 튜닝
2. 이미지 세그멘테이션을 위한 데이터셋 구성 및 U-Net 모델 학습
3. Unity–Flask–PyTorch 간 이미지 스트림을 주고받는 실시간 추론 파이프라인 구현 및 에이전트 관측 입력 연동

소속 아주대학교 AI 집중교육

참여 2인 (개발자 2)

역할 팀장

기간 2023.07 - 2023.09

도구 #Pytorch # Unity ML-Agent # Flask

Artificial Hunter Vision

#1 고성능 AI에 대한 기술적 도전, Unity ML-Agent 플러그인을 사용한 강화학습 에이전트 생성

목표

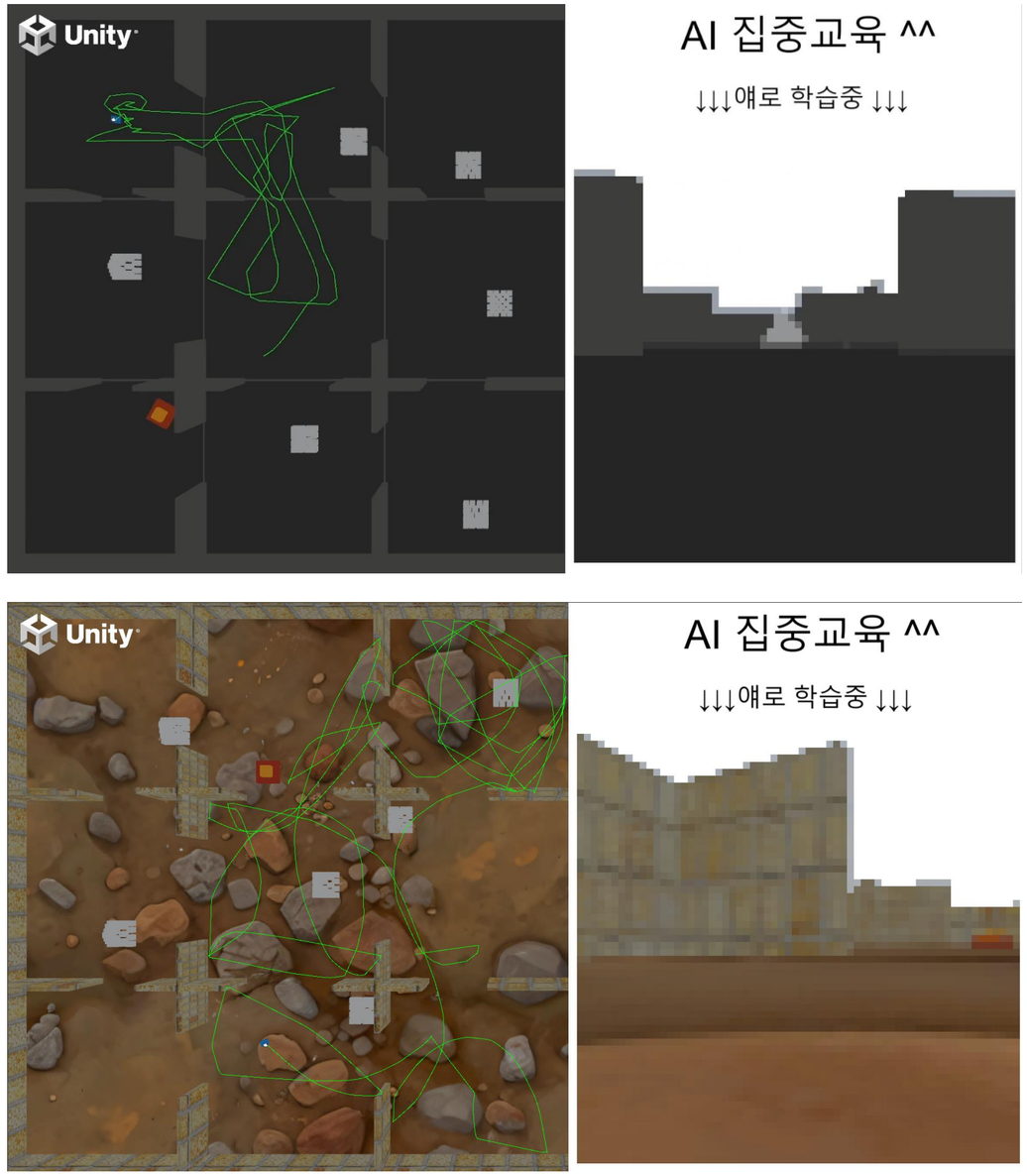
1. 사냥감을 추적하는 강화학습 에이전트를 소재로 게임을 만들어보고 싶었습니다.
2. 아주대학교 AI 집중교육 강의를 수강하며 프로토타입을 만들고자 했습니다.

구현

1. Unity ML-Agents는 PyTorch 기반 딥러닝 모델과 PPO / SAC / POCA 강화학습 알고리즘을 API 레벨에서 처리하기에 강화학습 기술을 빠르게 실험·검증하기에 적합한 환경이었습니다.
2. ML-Agents 기본 예제를 기반으로 에이전트의 관측 데이터를 RenderTexture로 설정했습니다.
3. 단순한 환경에서는 에이전트가 맵을 최대한 넓게 탐색하는 경로를 스스로 형성하고, 탐색 중 빨간 버튼을 인식하면 즉시 접근하는 행동 패턴을 보여줘 기대 이상의 학습 성능을 보여줬습니다. (영상)
4. 이후 복잡한 텍스처가 적용된 환경에서도 학습을 진행했으나, 에이전트는 이미지로부터 안정적인 특징을 추출하지 못하는 인식 병목(perception bottleneck) 현상을 보였고, 그 결과 의미 있는 상태 판단이 어려워 주변을 회전하며 탐색하는 수준에 머물렀습니다. (영상)
5. 이를 통해 단순 CNN 기반 이미지 입력만으로는 복잡한 시각 정보 해석에 한계가 있음을 확인했고, 자율주행 분야에서 활용되는 이미지 세그멘테이션 기반 전처리 모델을 도입해 에이전트의 인지 성능을 향상시키는 방향을 검토했습니다.

복기

1. PPO 알고리즘 대신 탐험 학습에 적합한 SAC 알고리즘을 사용할 수 있었습니다.
2. 카메라의 높이를 높이면 에이전트 성능이 향상했을 수도 있습니다.



Artificial Hunter Vision

#2 이미지 세그멘테이션 기반 전처리

목표

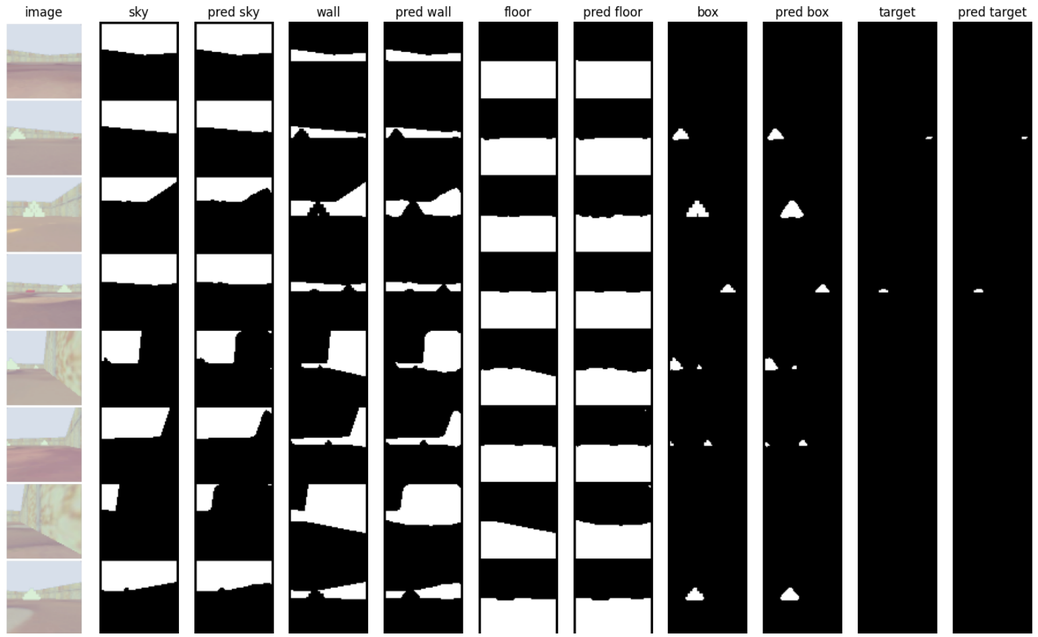
1. 복잡한 환경 이미지를 강화학습 에이전트가 효율적으로 인식할 수 있도록, 단순한 표현으로 전처리 가능한 이미지 세그멘테이션 모델을 확보하고자 했습니다.

구현

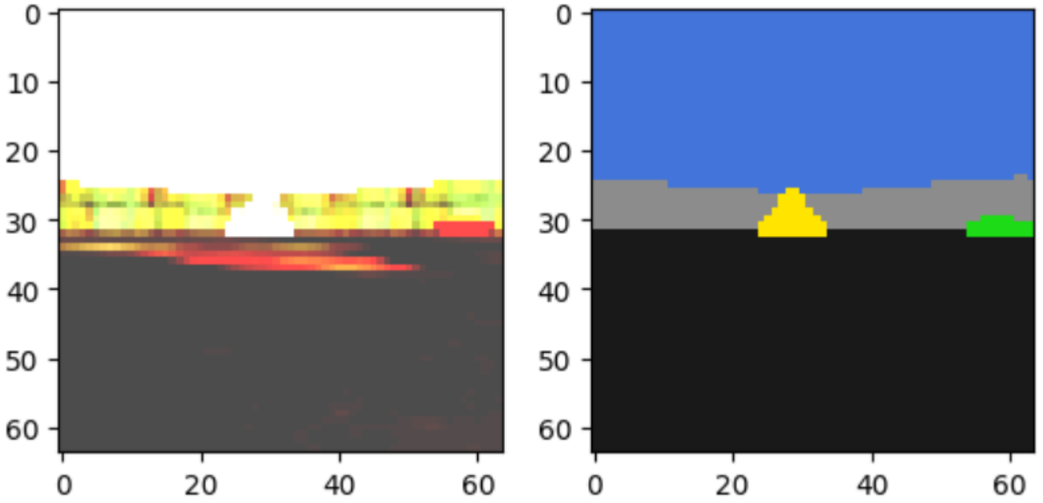
1. 모델 학습을 위해 대규모 라벨 데이터가 필요했습니다. Unity에서 오클루전 컬링(Occlusion Culling) 을 활용해 벽, 장애물, 버튼 오브젝트를 개별적으로 분리 렌더링한 뒤, 커스텀 셰이더를 적용해 단색 마스크 이미지로 변환하여 로컬 저장소에 저장했습니다.
2. 약 5만 장의 라벨 데이터를 생성했으며, 데이터의 분포 다양성과 순수성을 확보하기 위해 학습 환경 구성과 에이전트의 행동 패턴을 임의로 변경하며 반복 생성했습니다.
3. 세그멘테이션 모델로는 U-Net 구조에 MobileNet v2를 인코더로 적용해, 실시간 추론 환경에서도 사용할 수 있도록 연산량과 성능의 균형을 고려했습니다. 학습 과정에서는 Semantic label–preserving data augmentation을 적용해 특정 텍스처에 대한 과적합을 방지했습니다.

성과

1. 하늘·바닥·벽과 같은 주요 환경 구조는 안정적으로 분리할 수 있었습니다.
2. 허나 목표 오브젝트와 같은 소형 클래스에서는 예측 불안정성과 클래스 불균형 문제가 발생했습니다.
3. 이를 통해 강화학습 에이전트 성능 저하의 주요 원인이 지각(perception) 단계에 있음을 명확히 확인했고, 파라미터 튜닝과 라벨 구성 개선을 반복하며 전체 퍼포먼스를 점진적으로 향상시켰습니다.



멀티 클래스 세맨틱 세그멘테이션 결과 시각화



Data Augmentation 이미지도 세그멘테이션이 잘 된다

Artificial Hunter Vision

#3 Unity > Flask > Unit
소켓 통신을 열어 실시간 이미지 세그멘테이션으로 완성

목표

1. Unity와 Flask 서버 간 소켓 통신을 통해 실시간 이미지 스트림을 전달하고,
   Flask 서버에서 PyTorch 기반 세그멘테이션 추론을 수행하는 파이프라인을 구현했습니다.
2. 동시에 세그멘테이션 결과 이미지를 에이전트의 관측 입력값으로 활용했습니다.

구현

1. Unity Networking 라이브러리를 활용해 렌더링된 이미지 데이터를 HTTP POST 요청 형태로
   Flask 서버에 전달했습니다. (유니티 코드)
2. Flask 서버는 Unity로부터 수신한 이미지 데이터를 추출해 사전에 학습한 U-Net 기반
   세그멘테이션 모델에 입력하고, 추론 결과 이미지를 다시 Unity로 반환합니다. (Flask 코드)
3. Unity 측에서는 전달받은 세그멘테이션 결과 이미지를 강화학습 에이전트의 관측 입력으로
   제공하여 이후 행동 결정에 활용했습니다.

성과

1. 복잡한 텍스처 환경에서 인식 병목 현상을 보이던 에이전트는,
   세그멘테이션을 통해 의미 단위로 전처리된 이미지를 관측값으로 사용함으로써
   보다 안정적이고 목표 지향적인 행동을 보였습니다.

복기

1. 카메라 시야에 벽이나 바닥 텍스처가 과도하게 포함되는 경우, 세그멘테이션 모델은 전역적인 구조를
   충분히 인식하지 못하고 소형 객체(버튼, 장애물)를 과도하게 분리하려는 경향을 보였습니다. 이는
   시야 구성, 클래스 불균형, 국소적 feature 의존성에 따른 한계로 판단했습니다.

```
WWWForm form = new WWWForm();
form.AddBinaryData("image", imageData, "image.png", "image/png");

// POST 요청 보내기
using (UnityWebRequest www = UnityWebRequest.Post(serverUrl, form))
{
    yield return www.SendWebRequest();
```

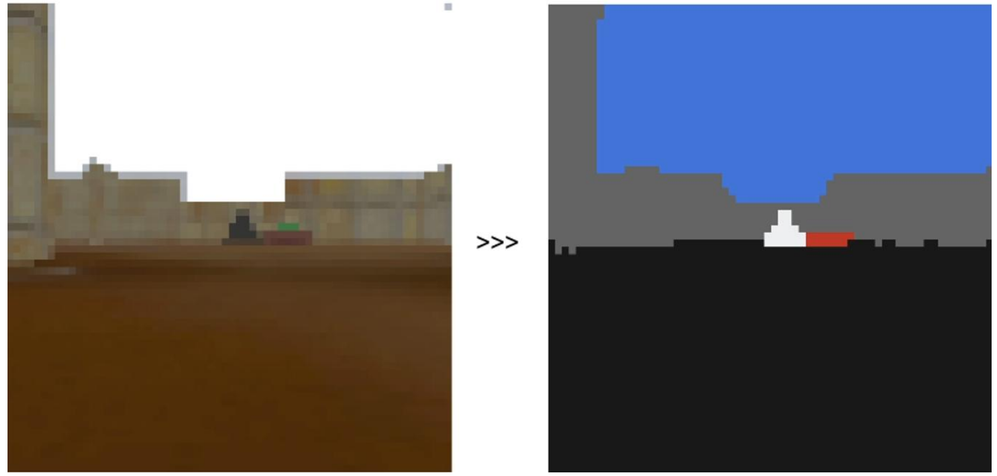
```
@app.route('/upload', methods=['POST'])
def upload_image():
    # POST 요청에서 이미지 데이터 가져오기
        # byte[]
    image_data = request.files['image'].read()
```

```
for l in range(5):
    map += (label == l).view(64,64,1) * torch.tensor(rgb[l]).view(1,-1)
map = map.int()
# print(f"map shape: {map.shape}")

label_map = map.detach().numpy()
# print(f"label_map shape: {label_map.shape}")
label_map = label_map.astype(np.float32)
label_map = Image.fromarray(np.uint8(label_map))

results = BytesIO()
label_map.save(results, format='png')
return results.getvalue()
```

```
Debug.Log("Image upload successful");
byte[] imageBytes = www.downloadHandler.data;
ConvertPNGToRenderTexture(imageBytes, renderTexture);
```



왼쪽 : 원본 / 오른쪽 : 에이전트가 보는 세상