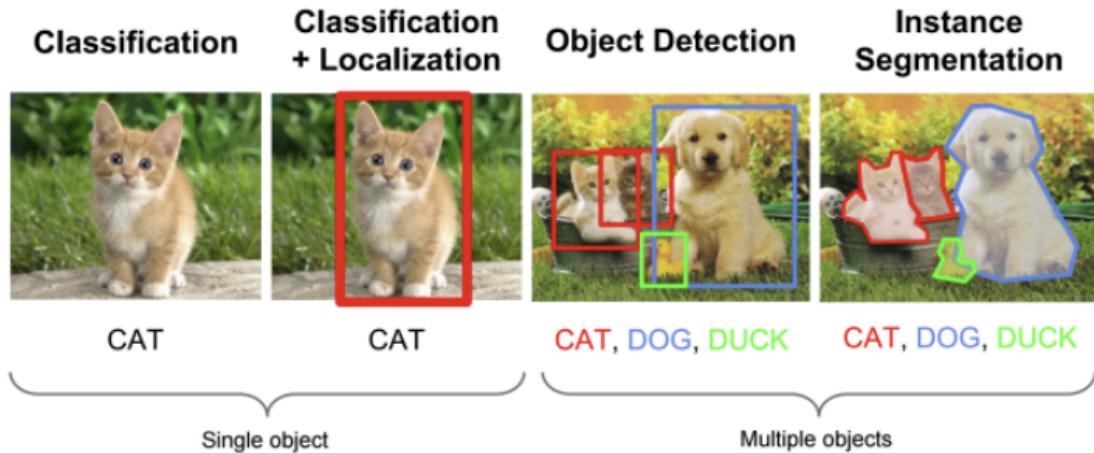


[논문리뷰] Segment Anything

? What is Segmentation

- Image Processing의 종류

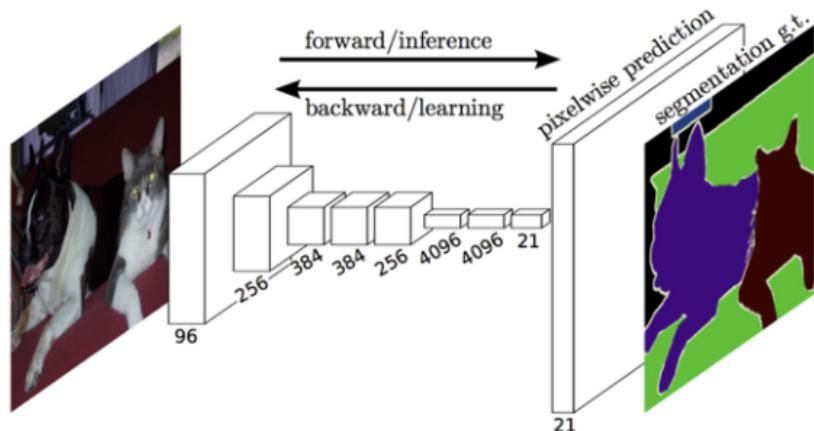


지난주에 다룬 VIT = Classification, YOLO = Object Detection 목적을 가짐

VIT는 이미지를 크게 분류하는 것에 강점을 가지고, YOLO는 객체가 어디에 있는지 빠르게 구분하는 것에 강점이 있는 것에 반해, SAM은 어떻게 나누는 것이 좋을지에 초점을 맞추기 시작함.

- 그래서 Segmentation 이란,

- 픽셀 단위로 어떤 클래스에 속하는지 분류하는 task로, 지난주에 다룬 모델들과는 다른 목적성을 가짐.





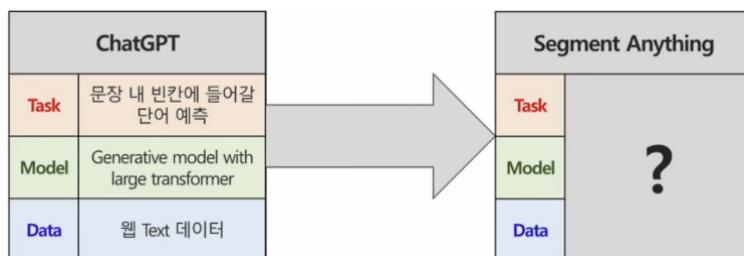
Introduction

- ✓ 이러한 발전 과정에도 불구하고, 이 모든 모델들의 한계 + 제안 배경

| 범용성 / Zero-shot 문제 해결(Promptable Segmentation)에 방점을 둔 모델 제안의 필요성

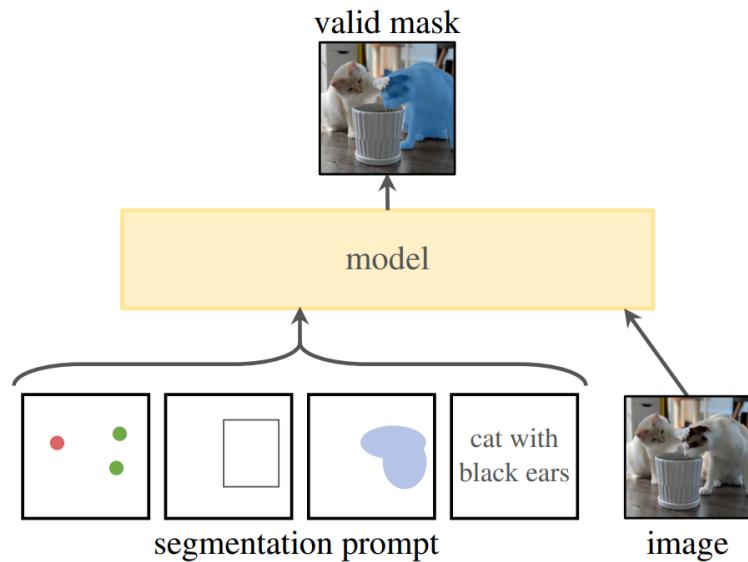
- 다양한 task에 적용 가능한 범용성 있는 Foundation Model로서의 성장이 부족하다는 것
- NLP에서의 prompting 아이디어 차용
"Segmentation 분야에서 ChatGPT와 같은 Foundation 모델을 만들 수 있을까?"

- ✓ 이에 대한 답을 찾기 위해 논문에서는 3가지 질문을 던진다.



- 1 Zero-Shot 일반화를 가능하게 할 Task는 무엇인가?
- 2 그에 대응하는 Model 아키텍처는 무엇인가?
- 3 이 태스크와 모델을 학습하는데 필요한 Data는 무엇인가?

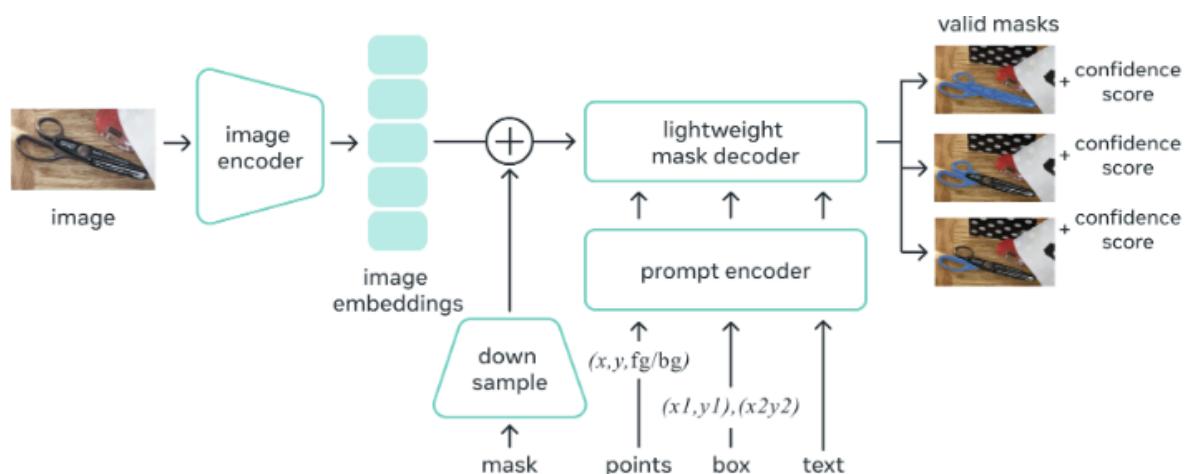
1 Task



(a) Task: promptable segmentation

- promptable segmentation ⇒ (정의) 어떠한 프롬프트를 주어도(point, bounding box, mask, text 등) → 반드시 하나의 mask를 반환 (이는 모호한 경우에도 반드시 출력값을 가진다는 task를 정의하면서.)
 - (1) 새로운 unseen data가 들어올 경우에도 처리 가능하다
 - (2) 다른 모델이 내는 output을 input으로서 넣을 수 있고, SAM이 내는 output이 다른 모델의 input이 될 수 있다는 모델의 연계성

2 Model



⇒ input 이미지/prompt의 특징을 추출하는 2개의 Encoder + 원하는 객체의 mask를 추출하는 1개의 Decoder로 구성

1. Image encoder

= 인풋 이미지를 고차원 feature embedding으로 변환

- Backbone Model = MAE로 ImageNet을 사전학습한 Vit-H/16
- 기존의 Transforemer Attention을 사용하면, 모든 픽셀 토큰끼리 다 참조해야 함. → 1024*1024 면 계산량이 폭발적인 문제.

▼ 해상도 축소 : 인풋 이미지(1024*1024 이미지)로 들어가고, 아웃풋으로 16배 작은 feature 맵을 내기 위해 → window attention + global attention block 을 중간에 수행

- 이는 연산 효율 + feature map에 중요 정보 보존을 위해 특수 Attention을 사용하는 것
 - **Window Attention** (14*14) : 이미지를 작은 window 단위로 잘라서 14×14 토큰 범위 안에서만 self-attention을 수행
 - **Global Attention Block** (주기적 4회) : window 안에서만 보면 전체 맥락에 대한 정보가 부족하기 때문에, 일정 간격마다 Global Attention Block을 넣어 이미지 전체에 대한 맥락 보완

▼ 채널 축소 : 1×1 conv (채널 압축) → 3×3 conv (지역적 맥락 보강), 각 conv 뒤에는 LayerNorm 적용 ⇒ 256 차원

- SAM은 실시간성도 하나의 목표이기 때문에 채널 축소도 진행한다.
채널 축소를 반드시 해야하는건 아님.
채널을 크게 유지하면 더 많은 표현력, BUT 속도/메모리 손해가 있고, SAM은 실시간성을 목표로 하기 때문에 축소
 - 1*1 convolution : 해상도는 유지하면서, 채널 수(=특징 차원)을 줄임
 - 3*3 convolution : (압축만 하면 정보 손실 나니까) 주변 3*3 영역 픽셀을 보면서 특징 다시 채움
- 각 convolution 뒤에는 Layer Normalization 붙여서 학습 안정화
- 장점: 이미지는 1번만 Encoding → 이후에 다른 프롬프트가 들어와도 동일 Embedding 재사용 → 실시간 상호작용 가능

2. Prompt encoder

= 모든 프롬프트를 실시간으로 임베딩 벡터로 변환하는 목적의 encoder

▼ [종류1] 희소 프롬프트

사용자의 input을 embedding 하여서 image prompt에서 생성된 feature map 위에 prompt token으로 얹는다. 이때 생기는 점의 좌표는 공간 관계를 이해하지 못하는 모델에 그대로 넣을 수 없기 때문에, 위치 Embedding이 필요하다.

1. 점(Point)

input은 사람이 마우스로 클릭한 것이다. 최종 위치 Embedding을 생성할 때에는 좌표 정보 임베딩과, 배경 여부 임베딩을 같이 고려하는데, 이 2가지 정보를 종합한 E_{point} 를 계산한다.

$$E_{\text{point}} = PE(x, y) + LE_{\text{bg}}$$

$$E_{\text{point}} = PE(x, y) + LE_{\text{fg}}$$

- $PE(x, y)$ = Positional Encoding으로, 사용자가 클릭한 점의 좌표의 위치 임베딩
- $LE_{\text{bg}}, LE_{\text{fg}}$ = 이 점이 물체 안에 있는지, 밖에 있는지에 대한 추가정보(전경 또는 배경 여부)를 부여하는

2. 박스(Box)

$$E_{\text{tl}} = PE(x_{\text{tl}}, y_{\text{tl}}) + LE_{\text{top-left}}$$

$$E_{\text{br}} = PE(x_{\text{br}}, y_{\text{br}}) + LE_{\text{bottom-right}}$$

3. 텍스트(Text)

사용자가 단어 프롬프트 (ex. "cat", "person")를 입력하는 경우에는, 텍스트를 text encoder에 넣어 벡터 표현으로 변환한다. 보통 text encoder로는 CLIP모델의 Text Encoder와 같은 자유 형식의 텍스트를 지원하는 사전학습된 모델을 사용하여 임베딩을 생성한다.

▼ [종류2] 밀집 프롬프트 (마스크)

- 사용자가 직접 픽셀 단위의 마스크를 제공하는 경우(ex. 두루뭉실한 윤곽성 그림 등)의 prompt encoder

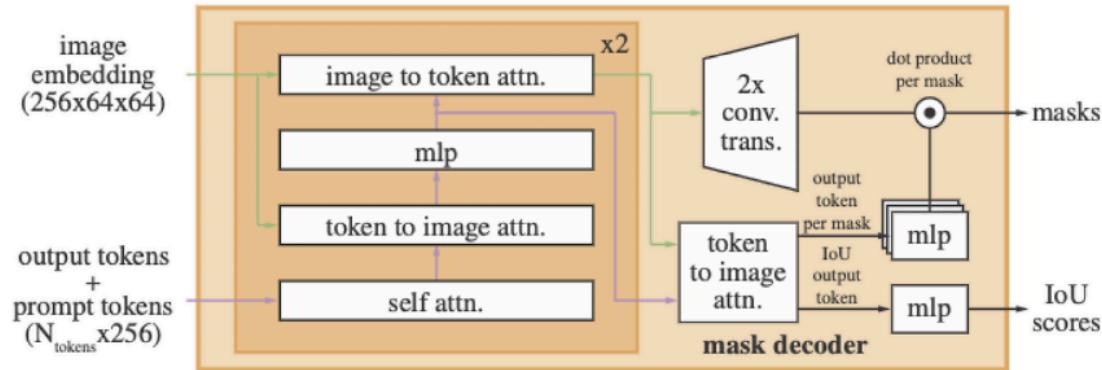
입력된 마스크 Image Encoder와 동일하게 convolution 연산을 통해 같은 해상도 (256차원 feature)로 변환(dense embedding map) →

이후에는 이미지 임베딩과 정보를 결합할 때 희소 프롬프트 방식과 다르게, image feature map 자체와 결합한다. ⇒ 이미지 임베딩과 element-wise sum

*예외적으로 prompt가 없는 경우에는 "no mask" 를 의미하는 특수 토큰을 추가하여서, decoder에는 항상 [Image features]+[Prompt tokens] 구조로 들어가도록 한다.

3. Mask decoder

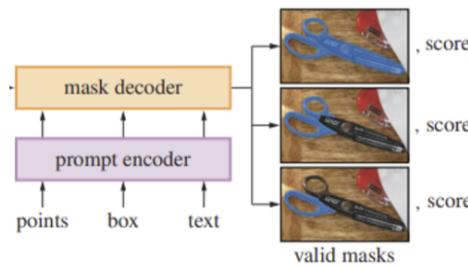
= 두 encoder로부터 나온 image embedding과 prompt embedding들의 attention (self-attention과 양방향 cross-attention) → masks + IOU scores 라는 두 출력 값을 내보내는 과정



▼ Decoder 2번 반복

- Self Attention :** Prompt tokens와 Output tokens 간의 self-attention으로, 여러 개의 prompt token (점+박스+텍스트) 관계를 반영해서 토큰 임베딩 (hidden state) 업데이트
 - Token → image Attention :** (cross-attention단계) 토큰 임베딩을 query로 삼고, 이미지 임베딩 벡터를 key와 value가 되어서, 각 토큰은 이미지 feature map을 보면서 “내가 가리키는 힌트가 이미지 공간 어디와 연관이 있는가?”를 참조 ⇒ 토큰 임베딩 업데이트
 - MLP :** 프롬프트 토큰을 작은 신경망(2-layer feedforward)으로 변환해서 정보 풍부하게 정리
 - Image → Token Attention :** 반대로 이미지 embedding이 토큰을 query로 삼아서 이미지 임베딩 토큰을 업데이트
- **출력 과정 (Output)**
 - **(1) masks 출력**
 - 업샘플링 :** 64*64 → 256*256 으로 해상도 복원

2. 출력 토큰 → MLP : 프롬프트 토큰을 작은 벡터로 변환
 3. Dot Product : 업샘플링 이미지 임베딩과 프롬프트 벡터를 곱해서 → 픽셀마다 객체 확률 map → threshold 0 or 1 = 최종 masks 생성
 - (2) IoU Score 출력 [MLP 2개]
- 모호성 문제 해결을 해결하기 위해 출력
 - 복수의 프롬프트가 주어지는 경우 모호성이 적기 때문에 단일 마스크를 출력하는데,
 - 단일점에 대해서는 바로 단일 정답을 내지 않고, IoU 예측 헤드를 이용해서 해결



- (1) 후보(3개 - 전체/부분/하위부분) 마스크를 동시에 출력 → (2) 학습에는 가장 ground truth와 가까운 마스크만 손실 계산에 사용 + 나머지는 무시 = 모델의 다양한 해석을 위함. → (3) 각 마스크마다 얼마나 정확할지를 추정하는 예상 IoU (Intersection over Union) 점수를 함께 출력해서 가장 신뢰도 높은 마스크 선택

Loss Function

- 손실함수
 - 마스크 손실 : Focal Loss + Dice Loss (20:1)
 - Focal Loss

$$L_{\text{BCE}}(p, y) = -[y \log(p) + (1 - y) \log(1 - p)]$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{if } y = 0 \end{cases}$$

$$L_{\text{Focal}}(p, y) = -\alpha (1 - p_t)^\gamma \log(p_t)$$

■ Dice Loss

$$\text{Dice}(y, \hat{y}) = \frac{2 \cdot |y \cap \hat{y}|}{|y| + |\hat{y}|} = \frac{2 \sum_i y_i \hat{y}_i}{\sum_i y_i + \sum_i \hat{y}_i}$$

$$\text{Dice loss} = 1 - \text{Dice}(y, \hat{y})$$

정답 마스크와 예측 마스크 간의 겹침 정도를 계산. Dice loss는 전경만 고려하므로 전체 부분에서 클래스 불균형에 강함. IoU보다는 오차에 더 민감.

결과적으로

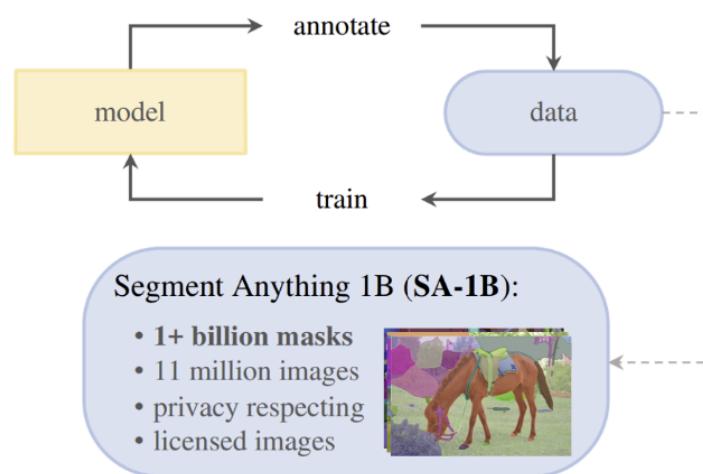
Focal Loss → 픽셀 레벨의 분류 능력 강화 (세부적인 경계, class imbalance 보정)

Dice Loss → 마스크 단위의 모양/겹침 최적화 (전경 전체 품질 개선)

따라서 두 Loss를 선형 결합하여 사용함으로써 픽셀 단위 정확도(local) + 전체 영역 정확도(global)를 동시에 잡을 수 있음

- IoU 예측 헤드 손실 : 예측 IoU와 GT IoU 간 MSE Loss
- 두 손실을 1:1로 더하여 사용

3 Data



(c) Data: data engine (top) & dataset (bottom)

Data Engine

- 보조 수동 단계 (Assisted manual stage)
- 반자동 단계 (Semi-automatic stage)
- 완전 자동 단계 (Fully automatic stage)

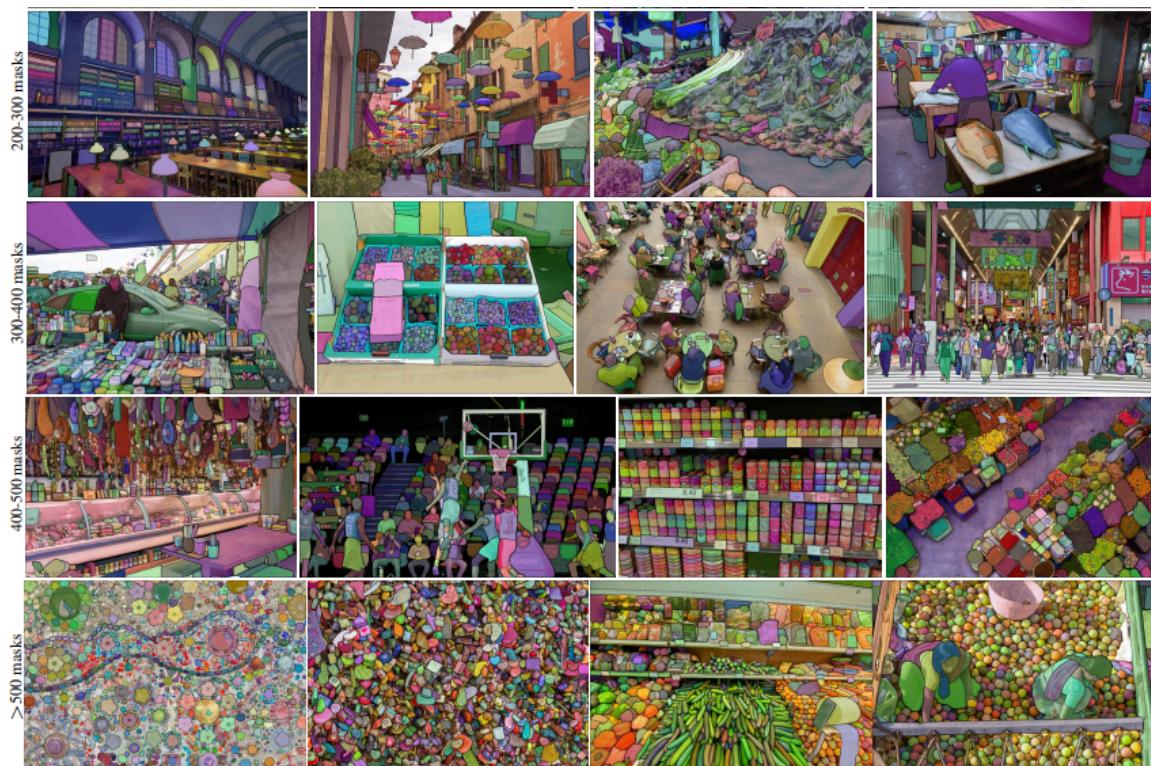
신뢰도 높고 안정적인 마스크들을 **NMS (Non-Maximal Suppression)**를 적용하여 중복되는 이미지를 제거. 작은 이미지의 안정성을 높이기 위해 **overlapping zoomed-in image crops**을 여러 번 적용

데이터셋 내 **1100만장의 이미지**에 대해 자동 마스크 생성을 적용하여 **11억 개의 고품질 마스크**를 생성.

⇒ **SA-1B**

Dataset

SA-1B : 1,100만 개의 다양한 고해상도 이미지, 데이터 엔진을 통해 수집된 11억 개의 고품질 분할 마스크로 구성



- **Image → Data Engine**을 통해 매우 많은 고품질의 분할 마스크를 생성했고 이를 연구에 사용할 수 있도록 공개했다

- **Mask** → 데이터 엔진은 11억 개의 마스크를 생성. 99.1%는 자동 생성됨. 이는 고품질이며 모델 학습에 유용하다는 것을 바로 아래와 7.에서 확인 가능. SA-1B에는 자동 생성된 마스크만을 포함.
- **Mask quality** → 500장의 이미지(약 5만 개의 마스크)를 랜덤 추출하여 annotators에게 픽셀 단위로 수정할 것을 요청. 자동 예측 마스크와 전문 교정 마스크를 IoU로 계산한 결과 94%의 쌍이 IoU 90% 이상, 97%의 쌍이 IoU 75% 이상을 기록.
- **Mask Properties (comparison)**
 - spatial distribution of object centers → 객체 중심이 이미지의 어느 부분에 많이 분포되어 있는지를 나타내는 그림. ADE20K와 SA-1B가 이미지 전체에 객체가 퍼져있는 데이터셋임을 알 수 있음.
 - sample size → 기존에 가장 많은 이미지와 마스크를 보유하고 있던 분할 데이터셋인 Open Images와 비교해도 SA-1B가 이미지는 11배, 마스크는 400배 더 많다. 또한 한 이미지 내의 마스크 수도 Open Images보다 36배 많다. 이미지 내 마스크 수가 가장 많던 데이터셋은 ADE20K이고 이보다도 3.5배 많다.
 - image-relative mask size → 다른 데이터셋에 비해 image-relative mask size가 적은 이미지의 비율이 높은 것을 확인 가능.
 - shape complexity

RAI Analysis

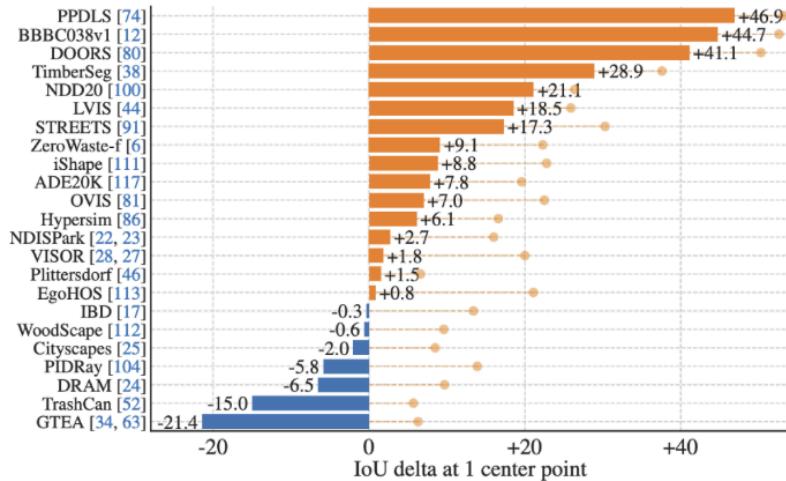
SA-1B와 SAM을 사용할 때 발생할 수 있는 공정성 문제와 편향을 조사하여 책임 있는 AI(Responsible AI, RAI) 분석을 수행. 지리적 및 소득 분포, 사람의 보호 속성(protected attributes)에 대해 공정하게 동작하는지에 착안.



Experiments [5가지 TASK + Ablations]

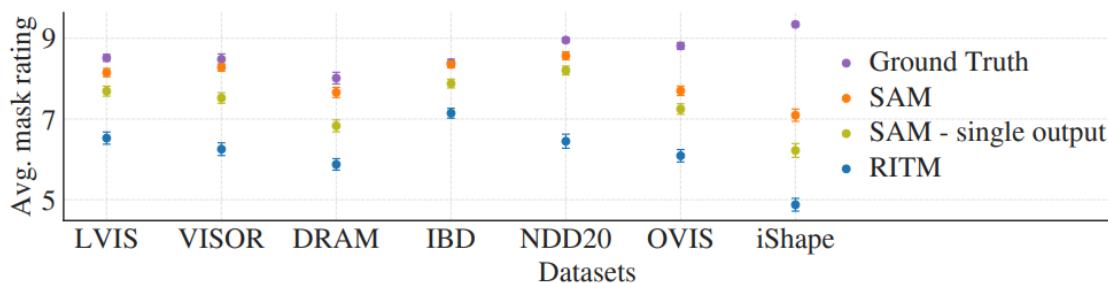
(7.1) Zero-Shot Single Point Valid Mask Evaluation (Zero-shot Segmentation)

- 이미지에 대해 점 하나를 찍어서 객체를 분할하는 태스크
- **SAM 모델 vs 기존의 SOTA인 RITM(강력한 대화형 분할 baseline) 비교**
- **결과**
 1. **mIOU 결과** : 23개 데이터셋 중 16개(70% 이상)에서 SAM이 좋은 성능을 보이고, ambiguity 특성까지 고려하여 Oracle 점수(주황 점선)를 봤을 때에는 모든 데이터셋에서 높은 성능을 보인다.



(a) SAM vs. RITM [92] on 23 datasets

2. 사람 평가 결과 : SAM 마스크가 더 깔끔하고 정확하다고 판단하며, 평균 7~9점으로 “객체가 잘 보이고 작은 오류가 있는 정도”라고 생각하는 수준이다. 1 center point에서 RITM보다 성능이 떨어진 데이터셋에서도 사람 평가 결과에서는 SAM이 일관되게 더 좋았음.



(7.2) Zero-Shot Edge Detection

엣지 검출 표준 데이터셋인 BSDS500 데이터셋 사용해서 object들의 edge를 검출하는 task이다. SAM은 학습 중에 BSDS500의 이미지나 엣지에 대해 전혀 노출되지 않았다.

- 결과

1. 정성적 결과 : SAM은 엣지 검출용으로 사전학습되지 않아 완전 제로샷 실험임에도 불구하고, 사진과 같이 정답과 가까운 엣지 맵이 생성되는 것을 눈으로 확인할 수 있다.



2. 정량적 결과

method	year	ODS	OIS	AP	R50
HED [108]	2015	.788	.808	.840	.923
EDETR [79]	2022	.840	.858	.896	.930
<i>zero-shot transfer methods:</i>					
Sobel filter	1968	.539	-	-	-
Canny [13]	1986	.600	.640	.580	-
Felz-Hutt [35]	2004	.610	.640	.560	-
SAM	2023	.768	.786	.794	.928

에지 검출 전용 최신 모델(발표 당시의)과 비교했을 때에는 성능이 떨어지나, 기존 전통적인 zero-shot transfer methods와 비교했을 때에는 높은 성능. R50에 주목했을 때 HED보다는 높고 EDETR과는 비슷한 지수. SAM의 Edge 검출은 마스크를 기반으로 하기 때문에 필요한 부분 이상으로 Edge를 검출하여 Precision이 낮고 Recall은 매우 높음.

(7.3) Zero-Shot Object Proposals

이미지 속에 객체가 있을 것이라고 생각하는 영역을 후보로 뽑아내는 task이다. 이 task를 위해서 기존의 SAM에는 최종 마스크 집합을 아웃풋으로 자동 산출하는 AMG 모듈이 있는데, 후보 마스크 세트를 제안하는 단계까지만 가져와서 사용한다.

- **SAM 모델 vs VitDet-H (Cascade Mask R-CNN + ViT backbone 기반 강력한 detection 모델) 비교**
- 결과:

method	all	mask AR@1000					
		small	med.	large	freq.	com.	rare
ViTDet-H [62]	63.0	51.7	80.8	87.0	63.1	63.3	58.3
<i>zero-shot transfer methods:</i>							
SAM – single out.	54.9	42.8	76.7	74.4	54.7	59.8	62.0
SAM	59.3	45.5	81.6	86.9	59.1	63.9	65.8

(7.4) Zero-Shot Instance Segmentation

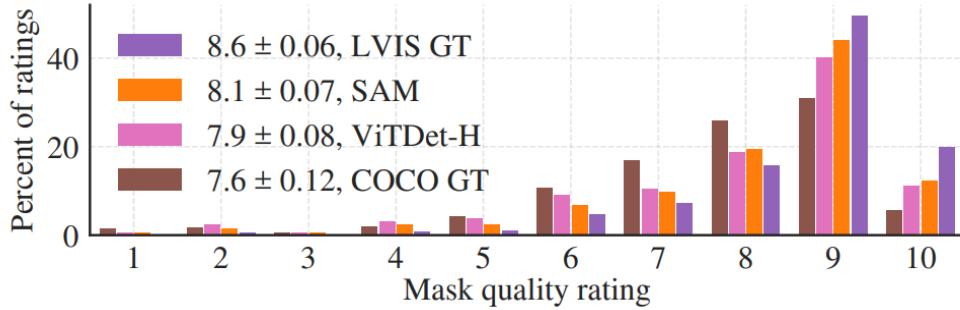
기존 객체 검출기 + SAM 조합으로 instance segmentation을 수행한다. 객체 검출기로는 이전에 사용한 VitDet를 이어서 사용하고, VitDet로 얻은 bounding box와 신뢰도가 가장 높은 SAM의 후보 마스크를 SAM의 프롬프트로 넣어 더 깔끔하고 정제된 마스크를 생성한다.

- ViT 자체의 성능 vs VitDet를 앞단 Detector에 쓴 SAM 비교
- 결과
 - 두 데이터셋(COCO, LVIS) 모두 SAM이 ViTDet보다 점수는 낮다. 하지만, 실제 마스크된 결과를 사진으로 확인하면 SAM이 경계를 더 깔끔하게 출력하고 있다.

method	COCO [66]				LVIS v1 [44]			
	AP	AP ^S	AP ^M	AP ^L	AP	AP ^S	AP ^M	AP ^L
ViTDet-H [62]	51.0	32.0	54.3	68.9	46.6	35.0	58.0	66.3
<i>zero-shot transfer methods (segmentation module only):</i>								
SAM	46.5	30.8	51.0	61.7	44.7	32.5	57.6	65.5



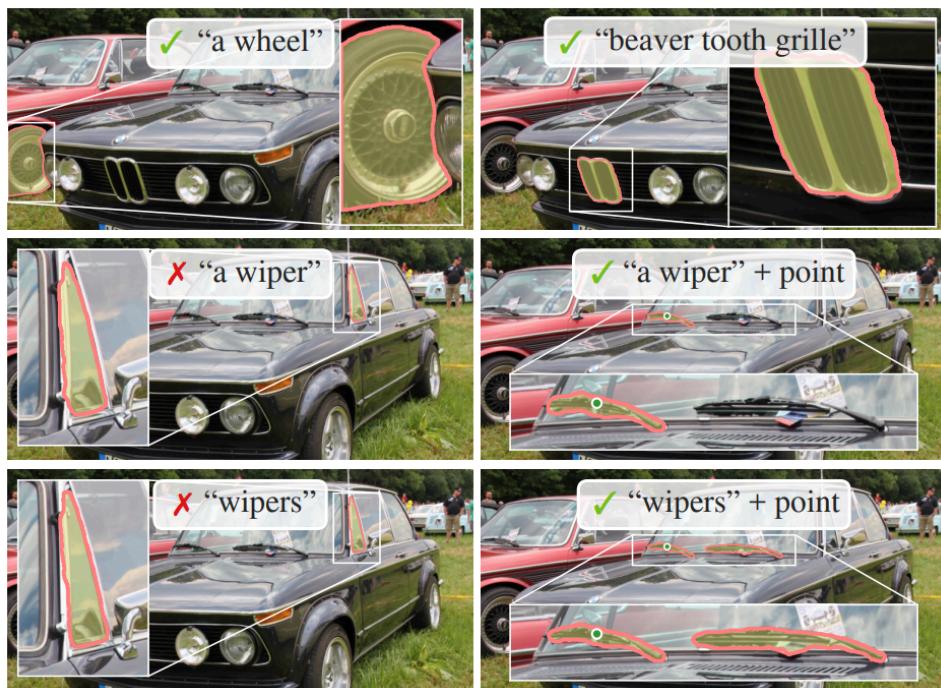
- 또한, 1~10 척도로 사람평가를 수행한 결과, SAM이 VitDet보다 우수하다고 한다.



정량적 결과와 정성적 결과가 다른 원인으로는, VitDet 모델이 데이터셋의 편향성을 반영하는 점에서 비롯된 것으로 추정하고 있다.

(7.5) Zero-Shot Text-to-Mask

마지막으로 텍스트 프롬프트에 대하여 객체 분할 성능을 평가. 학습 절차를 약간 수정. 기존 수동 마스크를 CLIP 이미지 인코더에 입력해서 이미지 임베딩을 추출하고, 이를 기반으로 CLIP이 뽑아낸 벡터가 의미 정보(이미지 임베딩과 텍스트 임베딩의 정렬)를 담고 있기 때문에, 이를 SAM에 넣어서 마스크 생성에 사용하는 방식으로 학습



- 결과: 여기서는 정량적 지표 없이 정성적인 결과만을 확인하는데, SAM은 text prompt를 기반으로 객체를 분할하는 것이 가능하며, text prompt만으로는 올바른 예측을 못하는 경우에는 추가 점(point prompt) 정보를 줌으로써 해결 가능하다고 한다.