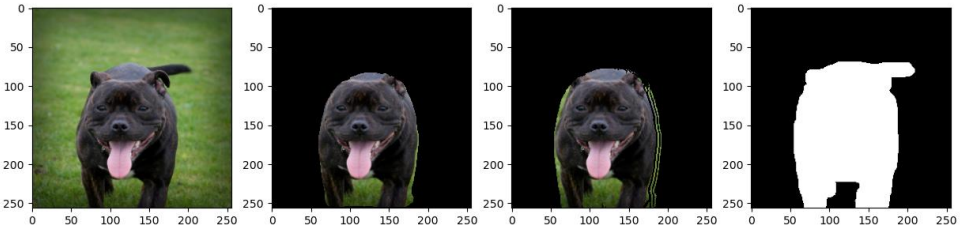
PA3 – Segmentation

20231126 황산하

1. Introduction

본 과제는 Simple Oxford Pet dataset을 이용하여 Image Segmentation을 진행하는 과정이다. 단순히 학습으로만 Image Segmentation을 하는 것이 아니라, CSPN과 DYSPN이라는 정교화 알고리즘을 이용하여 초기 예측한 마스크를 label이미지와 더 비슷하게 만드는 작업을 거친다.



[그림 1] CSPN 결과 예시, 순서대로 원본 이미지, coarse 마스크, CSPN 적용 후 마스크, 원본 마스크

본 과제를 위해 U-Net을 백본 아키텍쳐로 사용하였고, Unet을 통해 생성된 마스크를 CSPN과 DYSPN 알고리즘을 통해 각각 정교하게 만들었다. 해당 과제를 위해 Unet을 직접 pytorch를 이용해서 만들었고, 7 epoch를 학습하여 약 0.67의 miou성능의 마스크를 따로 저장하여 CSPN과 DYSPN알고리즘을 구현하였다.

2. Code

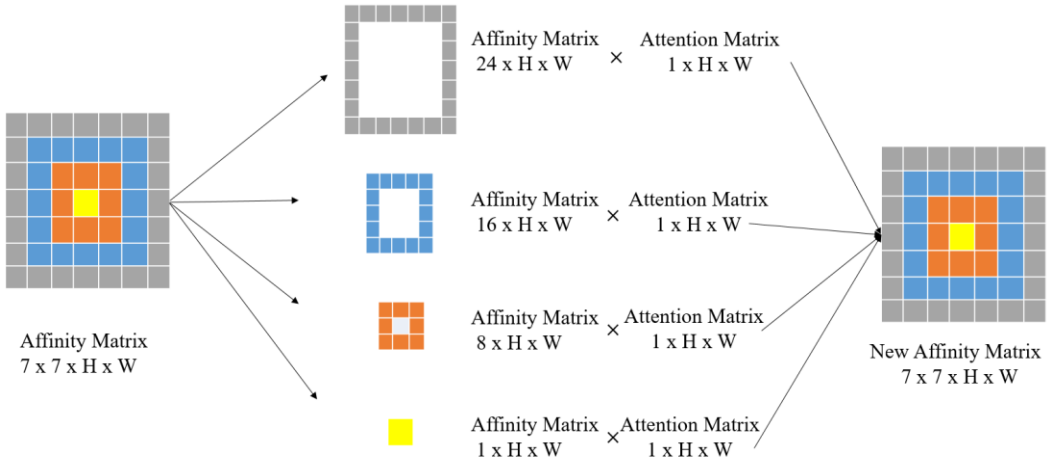
1) **coarse.py :** 이 스크립트는 초기 coarse segmentation 마스크를 만들기 위해 작성한 스크립트이다. Unet은 PPT의 Unet을 보고 구현했으며 이미지가 input으로 들어가면 Ground Truth 마스크와 Dice Loss를 구해 학습을 진행한다. 과제 초반에 제공해준 pytorch lightening으로 작성된 코드를 약간 변형했으며, eval()함수를 통해 train과 valid dataset의 coarse mask를 저장하는 코드로 구성되어있다.

2) **model.py:** 이 스크립트는 coarse.py에 작성한 Unet과 CSPN과 DYSPN알고리즘, 그리고 각각의 알고리즘에 사용된 UNet을 구현한 스크립트이다. 사용된 세개의 Unet이 output size가 달라지기 때문에 편의상 세개를 전부 구현하고 독립적으로 이용했다.

CSPN의 경우는 기존의 affinity를 구하는 과정과 다르게 Unet에서 Coarse마스크와 이미지를 입력으로 받아 출력하는 형태로 모델을 만들었다. Affinity는 픽셀간의 관계를 의미하는 행렬이기 때문

에, 특히 주변 픽셀의 정보를 집계한 행렬이기 때문에 주변 정보를 잘 학습한다고 알려진 CNN 기반의 아키텍쳐를 이용하면 이를 효과적으로 구할 수 있다. 따라서 여기서 구한 affinity를 이용하여 coarse 마스크를 곱하여서 더하는 방식으로 마스크를 업데이트하는 CSPN을 구현했다. 각 이미지마다 몇번 반복할건지 정할 수 있도록 알고리즘을 만들었다.

DYSPN은 CSPN보다는 좀 더 복잡한 알고리즘이다. 어텐션이라는 개념이 들어가는데, 논문을 보면 각 어텐션은 DYSPN의 반복횟수마다 다른 어텐션이 필요하다. 그리고 7X7의 커널을 이용하기 때문에 Unet의 최종 output size를 affinity의 채널 7*7 =49과,6번 반복마다 4개 채널의 어텐션이 필요하므로 6*4=24개 총 72의 채널을 갖는 output을 만들도록 한 후에 이를 적절히 나누어 학습에 이용했다. 73개가 아니라 72인 이유는 논문을 보면 가운데 채널은 이용하지 않기 때문에 여기는 어텐션만을 이용해서 연산했다. 또한 논문에서는 아래 그림 2 처럼 연산하도록 말하고 있기 때문에, 이를 위해 인덱싱해서 각각의 채널을 나누어서 연산했다.



$$h_{i,j}^{t+1} = \sum_{k \in \mathbb{Z}_+} \sum_{(a,b) \in N_{i,j,k}^t} \frac{\pi_{i,j,k}^t}{S'_{i,j}} w_{i,j}(a,b) h_{a,b}^t$$

$$+ \frac{\pi_{i,j,0}^t}{S'_{i,j}} h_{i,j}^t + (1 - \frac{S_{i,j}}{S'_{i,j}}) h_{i,j}^0$$

[그림 2] DYSPN의 어텐션 동작 방식

이 때 S'은 scale을 위해 절대값을 취해 합한 값이고 S는 값을 합한 값을 말한다. h는 업데이트 되

고 있는 마스크를 의미하고 π는 어텐션을 의미하며, w는 affinity를 의미한다. 따라서 저 수식을 풀어보면 아까 말한 affinity의 가운데 값은 사용하지 않고 어텐션만을 연산에 이용했으며 어텐션과 affinity 매트릭스를 각각 곱하고 이를 스케일링해주었다. 각 CSPN과 DYSPN에서 사용되는 UNet은 affinity를 잘학습하도록 학습이 되게 아키텍쳐를 완성했고, DYSPN같은 경우는 attention도 학습이 되도록 학습이 되도록 아키텍쳐를 구성했다.

3) **main.py** : model.py 에서 구현한 CSPN과 DYSPN을 직접 실행시켜서 결과를 받을 수 있는 스크립트이다. model.py 에서 구현되어 있는 CSPN들은 알고리즘 동작만하고 iteration은 해당 스크립트에서 진행해주었다. 따라서 model에서는 몇번 반복할건지 iteration을 설정할 수 있게 만들었고, model의 학습이 끝난 이후에 eval()함수를 통해 결과를 평가하고 저장하도록 구성했다.

4) **visualize.py** : 이 스크립트는 시각화를 위해 작성했다. 간단하게 결과를 체크해볼 수 있는 visual함수와 iteration마다 결과를 확인할 수 있는 iteration_vis 함수로 이루어져 있다. 본 과제를 위해 모든 결과물 형태를 npy 형식으로 저장했다. 특히 반복 마다 결과를 나타내기위해 각 반복마다 결과물을 리스트에 저장한 후, 이를 concatenate해서 시각화에 이용했다.

3. Conclusion & Limitation

CSPN과 DYSPN모두 이상하게 iou가 학습이 진행됨과 동시에 지속적으로 낮아지는 결과를 보였지만 시각화된 것을 보면 그래도 비교적 원본마스크를 잘 찾아갔다. 논문의 수식만 보고 구현했으면 어떻게 구현해야 할 지 막막했을 텐데, 수도코드가 주어져서 CSPN을 쉽게 구현할 수 있었다. DYSPN도 CSPN과 크게 다르지는 않았지만, 어텐션을 이용한다는 것과 affinity map에 추가적으로 처리가 들어간다는 점에서 조금 복잡했을 뿐 전반적으로 CSPN과 DYSPN의 구현에는 차이가 많이 없다는 것도 흥미로운 부분이다.

Unet만으로 학습했을 때 3 epoch 기준 0.4의 miou를 구할 수 있었는데, 초반 구현에서 miou가 학습할 때마다 줄어들었다. 줄어드는 원인에 대해 분석해보니, UNet으로 coarse 마스크를 구할 때에 비해서 iou가 0으로 나온 이미지가 꽤 많았다는 것을 알았다. 구현한 UNet이 문제인가 알아보기 위해 smp에서 제공하는 UNet으로 cspn과 dyspn을 알고리즘을 실행시켜 봤는데, 같은 현상을 볼 수 있다는 것을 깨달았다. 또한 DYSPN 같은 경우는 CSPN과 구현한 방법이 사실 다른데, 왜냐하면 수도 코드가 주어지지 않았기 때문에 어텐션과 affinity를 곱하는 방식만 추가하는게 잘못되었을 수도 있다는 가정을 하고 수식을 직접 구현했지만 원인을 해결하지못했다. 그럼에도 불구하고 miou가 떨어지기는 하지만 마스크를 확인해보니 대부분의 마스크의 성능은 사람이 보기에는 이전의 coarse 마스크와 비교했을 때보다 높아졌다는 걸 알 수 있었는데, 이는 affinity와 어텐션

을 충분히 학습하지 않아서 생긴 문제라고 결론 지었다. (10 epoch정도 학습했음)

또한 과제를 진행하면서 UNet만으로도 더 나은 마스크를 구할 수 있기 때문에 CSPN과 DYSPN을 굳이 왜 해야 하는 지에 대해 고민을 해볼 수 있었다. 지금은 딥러닝이 발전함에 따라, 딥러닝 만으로도 픽셀 간의 관계를 명확히 표현할 수 있지만, 이전에 detectron2로 image segmentation 을 했었던 경험을 되돌아 보면, 마스크의 경계 부분이 물결치는 등 정교함이 떨어지는 결과물을 얻는 경우가 꽤 많았던 기억이 떠올랐다. DYSPN과 CSPN을 이용함으로써 가지고 있는 마스크를 라벨에 비슷한 더 정교한 마스크를 표현할 수 있다는 점 때문에라도 이런 알고리즘들이 나올 수 밖에 없었다고 생각이 들었다.

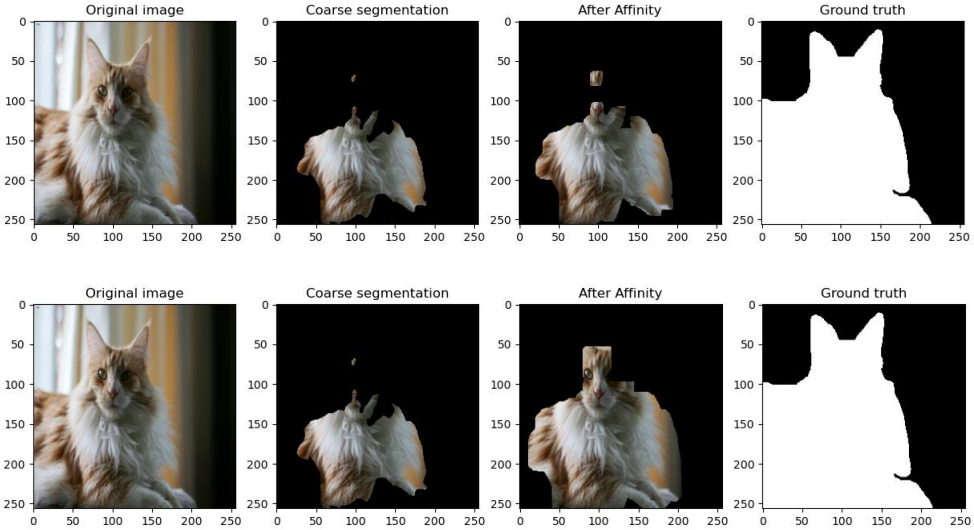
추가적으로 시도해본 결과는 DYSPN에서 사용하는 UNet은 총 72개의 채널을 가진 output을 출력하는데, 알고리즘 특성상 64로 채널이 줄었다가 마지막에 72개로 다시 늘어나는 방식을 취하고 있다. 그래서 이렇게 진행하면 sparse한 행렬을 받을 수도 있을거같아서 애초에 64로 줄이지 않고 바로 72까지만 줄여서 output을 출력하게 끔 해봤는데, 성능이 오히려 떨어진 점이 흥미로웠다. 알고리즘을 이해하고 구현하는데 시간이 생각보다 많이 들어서 많은 실험을 해보지는 못했지만 반복횟수를 늘리면 지속해서 마스크의 퀄리티가 좋아지는지 확인해보고 싶었고, 또한 Coarse 마스크를 구할 때 valid의 마크스가 아무것도 나오지 않는 현상이 꽤 오래 지속되었는데, train시에 각 epoch마다 validation의 성능을 평가할 때는 miou가 계속해서 잘 오르는 걸 보여주고는 train이 끝나고 마스크를 구해서 저장하는 과정에서는 왜 결과물이 안좋았는지 원인을 찾지 못했다. 해당 부분에서 꽤 많은 시간을 할애한 것이 이번 과제 수행에서 가장 아쉬웠던 부분이라고 생각한다. 그래도 CSPN과 DYSPN 모두 epoch가 반복될 때마다 성능이 오른 점은 고무적이다.

4. Result

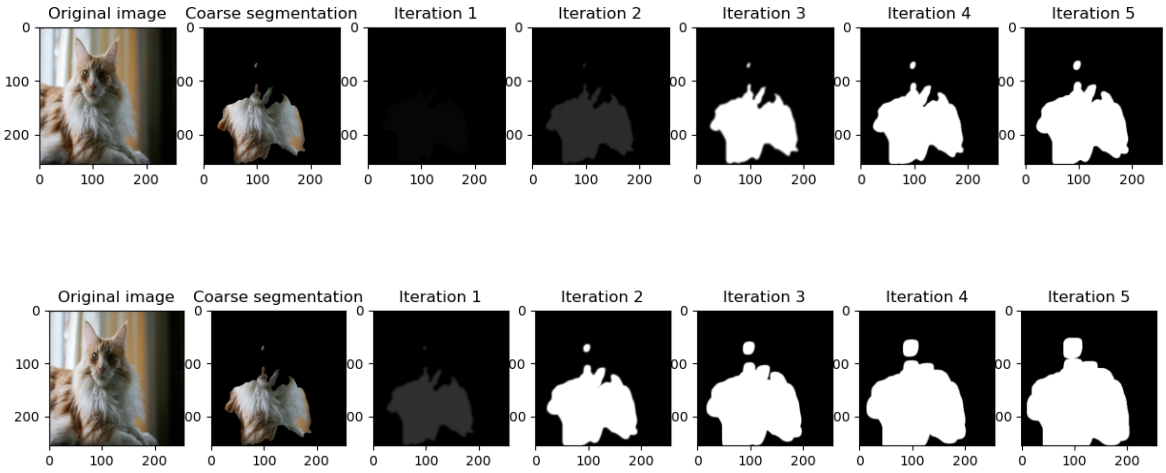
아래는 간단하게 결과를 나타낸 표이다. 시간관계상 5 epoch로만 돌려서 성능을 확인했고, validset에 대한 성능을 평가했다. 또한 몇가지 가시적으로 변화가 보이는 이미지들을 임의로 골라서 시각화한 후 보고서에 삽입했다. Result 폴더에 있는것들은 valid에 대해 알고리즘 실행한 것들 중 랜덤으로 선택해서 시각화해서 저장한 폴더이다.

[표 1] 알고리즘 성능표, 간단하게 성능을 나타낸 표이다. Coarse 마스크 같은 경우는 3 epoch로 고정했고 CSPN과 DYSPN의 성능을 각각 10 epoch에서 비교했다.

Miou (epoch)	Coarse segmentation (3 epoch)	CSPN	DYSPN
5 epoch	0.40	0.46	0.47



[그림 3] CSPN 결과(위), DYSPN결과(아래) 결과 예시 (valid 102 image)



[그림 4] DYSPN (아래) iteration 마다 보이는 마스크 결과 예시 (valid 102 image)