Implementation and Replication of the Latest Artificial Intelligence + Material

An experiences and understanding of WinCLIP method

이 름 정민규 **지도교수** 유종빈

2024-2 AJOU SOFTCON

연구배경

발달된 인공지능 기술들은 각 분야에 알맞게 접목되어 놀라운 속도로 발전하고 있다. 이에 따라 material 분야에서 인공지능이 어떻게 사용되고 있는지, 문제의 정의가 무엇인지, 이들을 해결하기 위해 어떤 구성을 사용하여 해결하는지 살펴본다. 특히 anomaly detection에 중점을 두고 해당 분야의 주요 문제와 해결방안을 이해한다.

연구 진행 과정

본 연구는 material에서 문제해결을 위해 anomaly detection을 어떻게 풀어가는지 살펴보는 것이 목적이다. 현재 CLIP의 일반화 능력을 사용하여 zero-shot anomaly detection에 대해 model의 성능을 높이고자 하는 여러 연구가 진행되고 있다. 하지만 일반적인 CLIP의 경우, zero-/few-shot task에 대해서는 좋은 성능을 기대하긴 어려운데, 이를 해결하기 위해 WinCLIP, AnomalyCLIP, AdaCLIP 등 다양한 방법론들이 제안되고 있다. 이들 중 본 SOFTCON에서는 WinCLIP에 대해 이해하고 실험해본다.

연구 내용 및 정리

WinCLIP은 CLIP의 일반화 능력을 사용하여 zero-/few-shot anomaly detection task에 사용하려는 접근법으로, industrial anomaly detection을 위하여 제안되었다. 가장 큰 특징으로, window 기반 접근 방식을 사용하여 image에서 small(2x2), middle(3x3), large(image) level의 multi scale feature를 사용한다. 이를 종합하여 anomaly score을 생성하고, anomaly detection을 수행한다. 본 연구에서는 해당 window에 대해 연구를 진행하고자 한다. dataset은 VisA를 사용하며, 해당 model에 사용할 수 있도록 변환한다. 이후 ViT와 같이 window가 아닌 fixed size patch로 구성하여 실험을 진행했을 때, WinCLIP 논문의 성능에 미치지 못했기에 masked embedding의 영향을 확인하고자 attention module을 구성하여 실험을 진행해보았다. 이에 따른 결과로, 특정 위치에서의 attention이 dataset의 특정 class에 대해서는 성능이 오르는 것을 관찰 할 수 있었다.

결과 및 분석

VisA	Before transformer				Base				After transformer			
	i_roc	p_roc	i_f1	p_f1	i_roc	p_roc	i_f1	p_f1	i_roc	p_roc	i_f1	p_f1
candle	23.62	65.66	66.89	0.80	84.80	89.92	77.78	**6.37**	**91.62**	**90.89**	**86.64**	5.61
capsules	57.97	62.35	78.30	1.21	54.00	**65.56**	76.92	**1.45**	**67.56**	64.75	**79.48**	1.31
cashew	18.62	80.45	80.00	6.57	65.62	86.28	80.82	8.32	**86.56**	**89.07**	**87.00**	**10.98**
chewinggum	39.42	63.52	80.00	1.69	83.40	**96.57**	**83.72**	**30.16**	**84.94**	96.56	83.64	29.58
fryum	22.72	9.58	80.00	4.51	50.72	84.05	80.00	12.85	**72.00**	**85.52**	**87.34**	**14.18**
macaroni1	**78.82**	48.31	**77.45**	0.16	55.51	56.78	66.67	**0.45**	43.51	**62.65**	66.89	0.44
macaroni2	**79.13**	**72.09**	**77.24**	**0.31**	54.24	52.29	67.35	0.08	43.03	68.58	67.12	0.26
pcb1	**65.69**	**64.06**	**68.18**	**1.36**	59.34	39.25	68.03	0.93	24.62	30.78	67.34	0.93
pcb2	**66.16**	**74.11**	**70.11**	**2.55**	62.84	60.79	68.99	0.61	26.96	51.24	66.67	0.49
pcb3	**71.83**	**76.55**	**73.47**	**4.19**	56.75	68.18	67.34	1.19	22.08	62.25	66.67	0.73
pcb4	66.39	89.00	69.20	7.41	**81.57**	**92.46**	**75.63**	**21.94**	37.27	87.13	68.49	8.72
pipe_fryum	38.92	89.53	80.00	15.69	83.52	93.96	84.07	22.73	**90.44**	**95.15**	**89.72**	**25.04**
average	52.44	66.27	75.07	3.87	**66.03**	**73.84**	74.78	**8.92**	57.55	73.71	**76.42**	8.19

위 그래프는 실험 결과로, 가장 높은 값에 대해 굵은 글씨 처리를 한 것이다. 눈 여겨 볼만 한 결과는 transformer 이후에 attention module을 추가로 넣었을 때, 특정 class (candle, cashew, fryum, pipe_fryum)에 대해서는 성능이 조금 더 좋아진 것을 확인 할 수 있다. 하지만 그 외의 경우 큰 폭으로 성능이 하락한 것을 알 수 있다. 이는 class에 따라 어떠한 pattern이 강조됨을 알 수 있으며, 특정 class는 global한 정보가 더 중요하거나 local 정보가 더 중요함이 나뉘기 때문이다. 이는 기존의 global 관계를 계산 한 후, 특정 영역에 대해 추가적인 학습을 하는 과정이 추가되었기에 어떠한 class에 대해서는 성능이 향상되지만 불필요한 중복 학습으로 인해 평균적인 성능이 떨어진 것이라 생각할 수 있다. 하지만 transformer 이전에 추가한 결과에 대해서는 input data를 사전 정제를 하고 transformer의 효율적인 학습을 계획했으나 가장 낮은 성능을 보였다. 그러나 macaroni, pcb의 경우 가장 높은 성능을 보여주는데, 향후 연구 계획으로 왜 이러한 결과가 나오는지 연구해보고 input data의 class간 차이를 비교해보는 것에 의미를 찾을 수 있을 것 같다.

오픈소스 URL

https://github.com/Gnaroshi/univ_ajou-24_2-self_directed_research_2