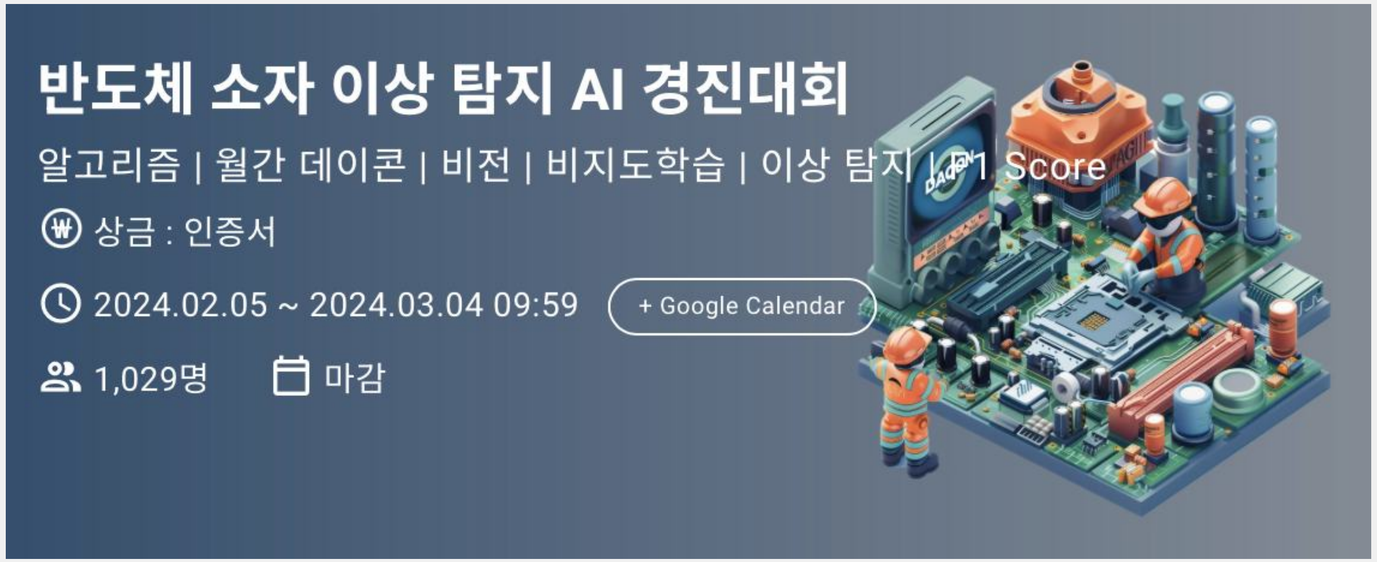
PROJECT REVIEW

중급 전서윤 팀

OUR MISSION



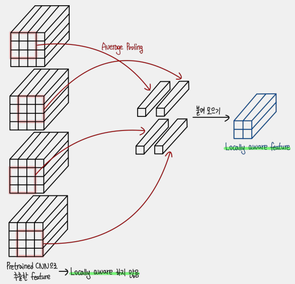
최근 몇 년간 인공지능(AI) 기술은 빠르게 발전하면서 다양한 산업 분야에 혁신을 가져왔습니다.
이러한 배경에서, 한 학기 동안 동아리에서 학습한 인공지능 지식을 바탕으로,
우리나라의 핵심 산업인 반도체 분야에 인공지능 기술을 적용하기로 했습니다.

따라서 데이콘의 반도체 소자 이상탐지 AI 경진대회를 참고하여
생산 공정에서 촬영된 반도체 소자 이미지로부터 비정상 샘플을 검출하는 비지도 학습 기반의 이상 탐지 AI 모델 개발하는 것을
목표로 프로젝트를 진행하였습니다.

PATCHCORE

이미지 출처: https://ffighting.net/deep-learning-paper-review/anomaly-detection/patchcore/

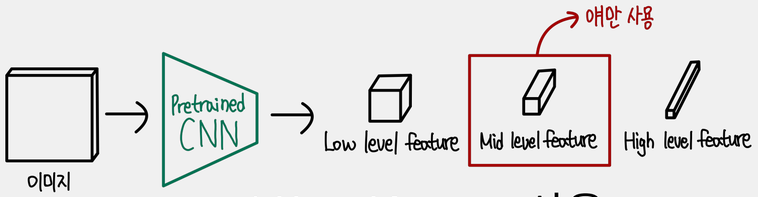
Local Patch Feature 1



- 이상치 탐지에 필요한 위치 정보
- 본인 정보만 가지고 있는 Patch

→ Sliding window 방식으로 주변부 패치 average pooling

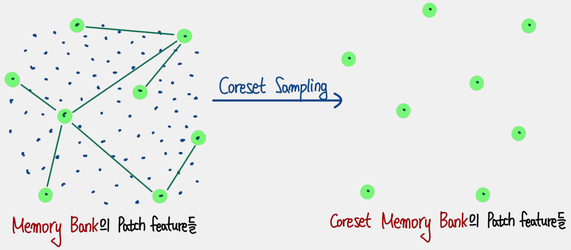
➔ locally aware한 특징 생성



- mid level feature 사용

위치 정보도 비교적 유지 & ImageNet에 덜 특화된 일반적인 특성을 구분

Coreset subsampling 2

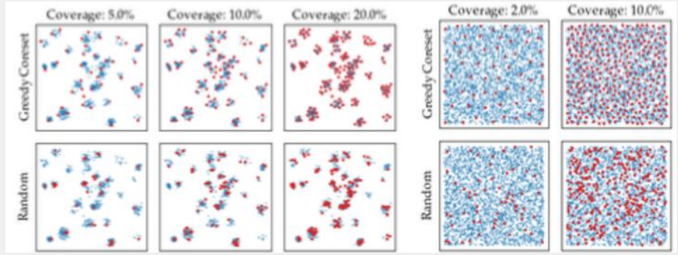


- 패치 특징이 많아지면 무거워짐

→ memory bank 전체 사용 X

핵심만 추출

➔ Coreset sampling

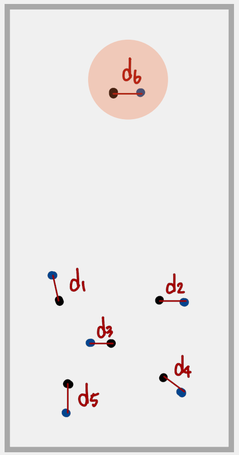


- Greedy Search 알고리즘 사용

→ 정상 feature가 뭉치지 않고 온전히 넓게 퍼져 선택됨

Detection and Localization 3

- 각 test Patch Feature별로 Coreset Feature와 비교
- K-NN → 가장 가까운 거리로 anomal score
- 측정한 거리에 가중치 부여



- 다른 patch feature들과 멀리 떨어진 샘플

➔ 더 큰 가중치

코드 흐름

1	**Cross-Validation 수행 (cv=5)** • KFold를 사용하여 데이터를 train, validation세트로 나눔 • 각 fold에 대해 2~4 과정을 수행
2	**Patchcore 모델 학습 및 예측** • 각 fold의 train 데이터로 Patchcore 모델을 학습 • 해당 fold의 validation 데이터에 대한 예측 점수 scores를 계산
3	**Validation 데이터에 대한 임계값 설정** • 각 fold의 validation 데이터에 대한 최대 예측 점수를 임계값 threshold로 설정 • 이 임계값은 정상 샘플과 이상 샘플을 구분하는 기준이 됨
4	**Test 데이터에 대한 예측 수행** • 각 fold의 test 데이터에 대한 예측 점수 scores를 계산
5	**최종 임계값 및 결과 계산** • 각 fold의 임계값 threshold들의 평균을 최종 임계값으로 설정 • 각 fold의 test 데이터에 대한 예측 점수 scores들 중 최대값을 선택 • 최종 임계값과 최대 예측 점수를 비교하여 이상 탐지 결과 prediction을 계산

PatchCore 모델

- **image size**: 224
- **backbone model**: wideresnet101
- **feature extract layer**: layer3
- **pretrained model embedding dimension** : 1024
- **target embedding dimension** : 1024
- **patch size**: 3
- **anomaly score NN**: 5
- **batch size**: 32
- **cv**: 5

UPDATES

한계점

- 알고리즘의 성능이 실행 환경에 따라 다르게 나타남

- Windows11 → F1 score = 1 (수상 결과)
- Ubuntu(Google Colab) → F1 score = 0.44 (직접 실행)

➔ 성능이 크게 낮아짐

- 환경에 따른 불가피한 결과

➔ 현재 환경에서의 F1 score=0.44를 기준으로 삼고
성능 향상을 위한 노력을 진행하기로 결정

문제 해결

가. ReConPatch

나. 모델 최적화

다. 기타 시도

가. RECONPATCH

Patchcore 단점

- 학습 데이터: 산업 현장 (X) 일반적인 ImageNet (O)
- 넓은 특징 분포 = 정상 제품 이미지들 간 편차 ↑

  ➔ 정상 제품 이미지도 높은 이상 점수로 측정 될 가능성

Reconpatch

- 목표: 유사한 패치 특징의 분산 ↓, 서로 다른 특징의 차이 ↑
- Patch-level 특징 표현 학습 접근법

  → 매우 유사한 특징을 집합화, 유사성이 낮은 특징은 배제
- 유사한 nominal representative 공유

  → nominal image patch에서 추출된 특징을

  비지도 학습 방식으로 가깝게 그룹화하는 표현 공간을 학습

Dimension	1024	512	256	128	64
PatchCore	**99.1**	98.66	98.45	98.54	97.75
ReConPatch	99.49	**99.56**	99.53	99.52	99.14

➔ Patchcore보다 높은 성능

ReconPatch 모델 적용 시도 및 어려움

- PatchCore → ReConPatch 모델 변경하여 성능 향상 시도
- ReConPatch 알고리즘의 오픈소스 자료 부족

  → 공식 GitHub 레포지터리 & 논문 참고, ChatGPT 활용
- 여러 에러 발생

  → 결국 전부 해결하지 못해, ReConPatch 모델 적용 성공 X

MisconfigurationException, TypeError: 'module' object is not callable, IndexError: list index out of range, AttributeError: 'torch.Size' object has no attribute 'rank', ValueError: Input 0 of layer "dense_22" is incompatible with the layer: expected axis -1 of input shape to have value 150528, but received input with shape (32, 3, 224, 224), SyntaxError: incomplete input, ValueError: Length of values (4) does not match length of index (100), AttributeError: 'str' object has no attribute 'base_dtype', AttributeError: 'Variable' object has no attribute 'ref', ValueError: Length of values (20) does not match length of index (100)

나. 모델 최적화

BACKBONE 변경

모델의 주요 특징 추출 네트워크

기존 코드 wideresnet101 사용

이를 resnet50, efficientnet_b1, densenet121으로 바꿈

각 모델의 특징

WideResNet101: ResNet 계열의 일종, 넓은 네트워크로 더 많은 특징을 학습, 대규모 데이터셋에 적합하지만 계산량이 많음.

ResNet50: ResNet 계열의 일종, 적절한 깊이와 성능, 일반적인 이미지 분류 작업에 적합.

EfficientNet_B1: 연산 비용을 최소화하면서도 성능을 높이도록 설계됨, 자원이 한정적인 모바일 및 임베디드 장치에 적합, 파라미터가 적음.

DenseNet121: 특징 재사용을 극대화, 작은 데이터셋에서도 우수한 성능, 메모리 사용이 많을 수 있음.

결과

성능 비교 척도 F1 score

0에서 1 사이의 값을 가지며, 값이 클수록 모델의 성능이 좋은 것

	wideresnet101 (기존)	resnet50	efficientnet_b1	densenet121
public점수	0.35714...	0.3846…		
private점수	0.44	0.375		

resnet50과 비교 → public 점수에서는 성능이 비슷하지만 더 중요한 지표인 private 점수에서는 wideresnet101가 더 좋은 성능을 보임.

efficientnet_b1, densenet121과 비교 → 구글 코랩에서 사용할 수 있는 GPU 리소스가 고갈되었다는 문제가 해결되지 않아 실행결과를 얻지 못함.

다. 기타 시도

1. 교차 검증 최적화

이상탐지 문제에서의 정상/비정상 샘플 불균형 해결을 위해 StratifiedKFold 방식 시도

- StratifiedKFold: KFold 방식에서 데이터셋의 클래스별 비율을 유지하면서 폴드를 나누는 방식

그러나, train 데이터가 정상 샘플로만 구성되어 있어서 적용 불가

대안으로 KFold 방식을 사용할 수 있으나, 이미 기존 코드에서 사용한 기법이므로 추가적인 성능 향상 어려움

2. 리소스 부족 문제

그 외 성능 향상을 위한 시도

- 모델의 일반화 성능을 높이기 위한 데이터 증강(Data Augmentation)
- 하이퍼파라미터 최적화
- 최종 임계값 및 스코어 합산 방식 변경

그러나 Google Colab의 리소스 부족으로 GPU 사용 제한

→ TPU/CPU로 대체해봤으나, 지속적인 메모리 부족 문제(out of memory)로 테스트에 어려움 발생

결론

프로젝트 경험

1. 동아리에서 공부한 딥러닝 지식을 바탕으로 새로운 이상 탐지 기술 전반을 학습 및 실습
   - 최신 이상 탐지 알고리즘 - PatchCore, ReConPatch
   - 파이썬 라이브러리 - anomalib
2. 제조 환경에서 결함 데이터 수집의 어려움 인식 및 AI 기술의 반도체 산업 적용 가능성 탐구

한계점 및 향후 과제

- 우분투 환경에서 성능 저하 문제 발견
  → 리눅스 기반 실제 산업 환경에서의 적용을 위한 추가 연구 필요
- Googld Colab 리소스 제약으로 인한 실험 한계
  → 안정적인 컴퓨팅 자원을 확보하여 성능 개선 시도 재개하고자 함

감사합니다

중급 전서윤 팀