환편기 중간 보고

포항공과대학교 인공지능연구원

- 유태욱, 최성우 -

목 차

데이터 전처리

초기 데이터 구성

정상 데이터 : 256 * 256 총 6526건

불량 데이터 : 512*512 8건 1024*1024 1건

미라벨 데이터 : 100*100 총 82054건

→ 데이터 별 픽셀이 다르기때문에 그냥 리사이징 한다고 해결되는 문제가 아님

따라서 각각의 데이터 픽셀을 일정하게 만들어주고 패턴 학습을 할 수 있도록 데이터 크기를 일정하게 맞춰줌

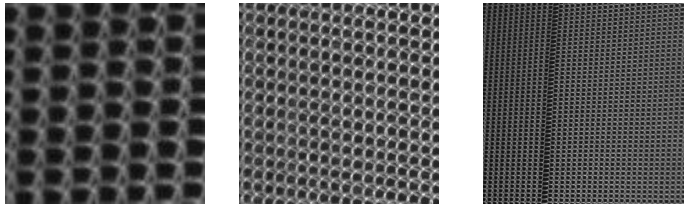
총 두가지 방안:

1. 정상 불량 데이터를 200*200 혹은 500*500 으로 맞춘다음 나눠서 전부 다 같은 픽셀로 맞춰주는 방법
2. 정상 불량 데이터를 128*128 으로 나눠서 100*100으로 맞춰주는 방안

→ 픽셀 손실이 적은 1번 방법을 선택하여 전처리를 진행함

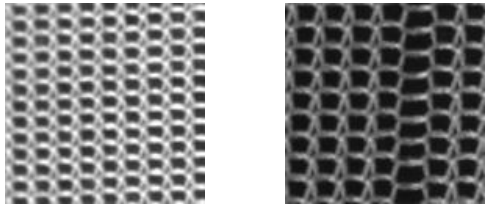
데이터 전처리

전처리 이전 - 미라벨 정상 불량



미라벨 정상 불량

전처리 이후 - 정상 불량



정상 불량

전처리를 통해 각 원단의 구멍의 크기를 비슷하게 맞춰줌(정확한 판단을 위해)

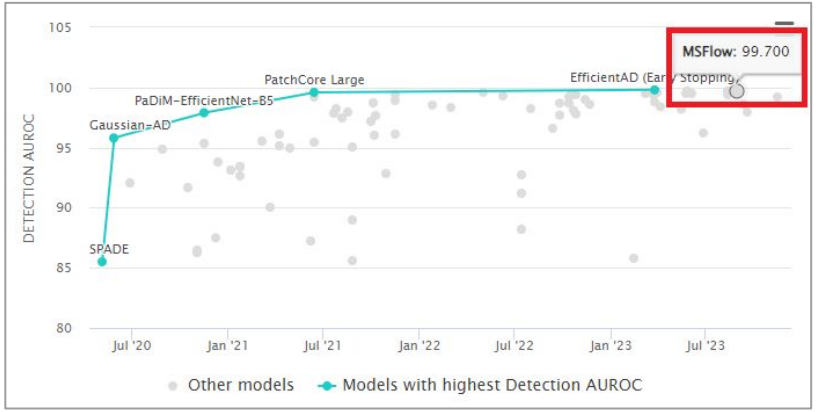
전처리 결과로 데이터 셋 재 구축
→ 원본 불량이미지를 자르고 나니 불량 이미지도 많이 생겼지만, 정상인 부분에 대해서 정상 데이터로 재라벨링
미라벨 데이터를 정상/불량으로 육안으로 직접 나눠서 불량 데이터 수를 늘림 (test 데이터에 활용)

최종적으로 train data set -> 정상데이터 총 22,930 건 구축
test -> 총 5,324 건 (불량/ 정상 비율 약 50%)

MSFlow

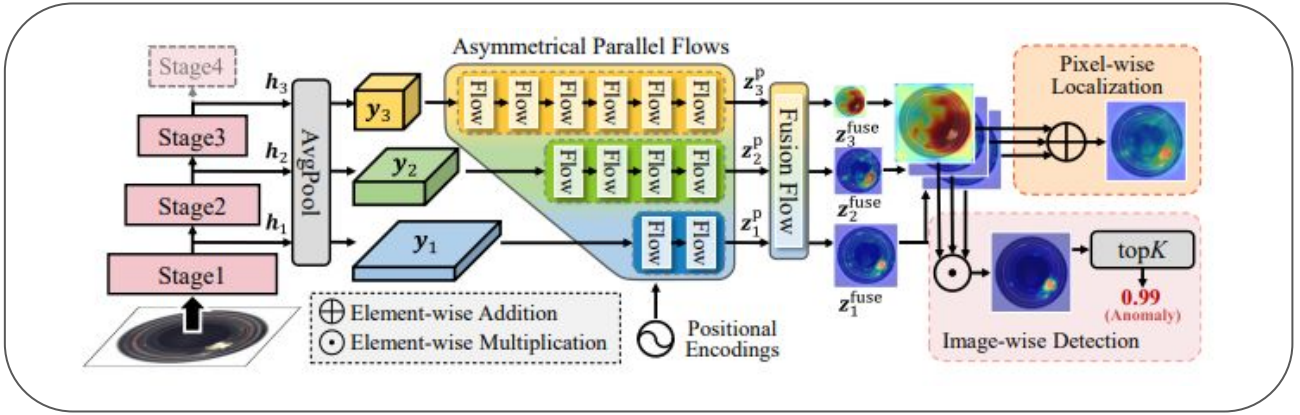
모델 선정 이유

1. 타 비교군의 모델 보다 우수
2. 오픈 소스 코드의 유무
3. 비교적 용이한 코드 수정 및 모델 구축



	정확도	속도
MSFlow	높음	빠름
AnoDDPM	높음	느림
CFA	높음	보통

MSFlow



모델 구조

크게 세 역할을 하는 딥러닝 모델이 합쳐져서 생성됨

1. Extractor: 입력 이미지에서 중요한 특징을 추출하는 역할. Resnet 구조를 기반으로 하며, 이미지의 고 수준 특징을 추출한다.
2. Parallel Flows: 추출된 특징을 다중 Scale에서 처리하고 학습. 중요한 정보를 다양한 방법으로 알아차리도록 돕는다. 여러 개의 SquenceINN 모듈로 구성
3. Fusion Flow: Parallel Flow들에서 생성된 정보를 결합하고 Scale을 조정해 모델의 최종 출력을 생성. GraphINN 모듈로 구성

MSFlow

주요 소스 코드 파일

- default.py: 모델 train 및 test 과정에 쓰이는 파라미터를 초기화 한 파일
- datasets.py: 모델의 train 및 test에 활용될 데이터 셋을 정의한 파일
    - torch.utils.data의 Dataset 클래스를 상속받아 만든 CKMDataset 클래스 정의 (환편기 이미지 데이터 셋 클래스)

      이미지의 경로를 통해 라벨을 지정해주고, 이미지를 전처리 후 반환해주는 기능까지 구현

- main.py: 터미널 환경에서 실행에서 매개변수를 파싱하고 train을 시작하는 파일
- train.py: 입력받은 파라미터와 데이터 셋으로 train 및 test 과정을 진행하는 파일
    - epoch를 진행하는 함수, 이미지 데이터를 모델에 입력으로 넣고 출력을 가져오는 함수 등이 존재
- post_process.py: 모델의 출력으로 받아온 확률맵 값과 Scale 크기를 가져와서 이상치 맵과 점수를 산출하는 함수가 초기화 되어 있는 파일

MSFlow

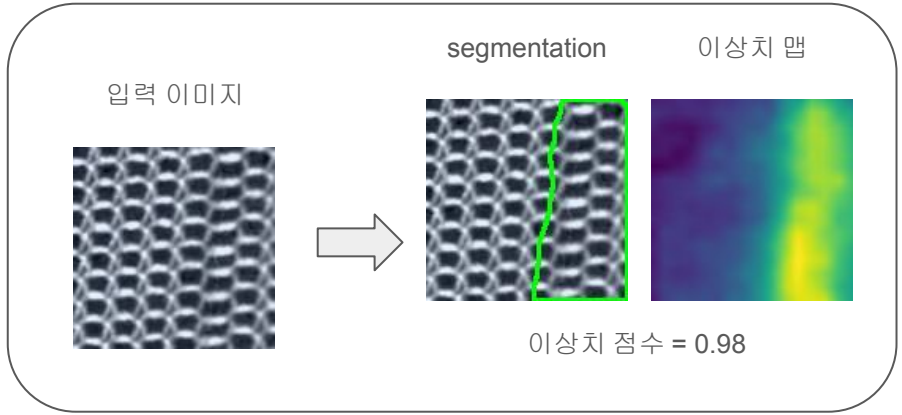
훈련 및 결과

- 전처리를 통해 구축된 데이터 셋 사용 (train - 22930, test - 5324)
- train 주요 하이퍼 파라미터
  - input size: 128*128
  - extractor: wide_resnet50_2
  - epoch(meta / sub): 25 / 4 (= total 100)

train 과정

```
[2023-10-23-15:07:53] Epoch 0.0 train loss: -4.641e+04  lr=3.00e-05
[2023-10-23-15:13:59] Epoch 0.1 train loss: -5.337e+04  lr=3.58e-05
[2023-10-23-15:20:06] Epoch 0.2 train loss: -5.536e+04  lr=4.17e-05
[2023-10-23-15:26:12] Epoch 0.3 train loss: -5.666e+04  lr=4.75e-05
[2023-10-23-15:26:57] Epoch 0   test loss: -3.963e+04   FPS: 118.0
Multi-scale sizes: [[16, 16], [8, 8], [4, 4]]
[2023-10-23-15:26:58] Epoch [0/24] Det.AUROC: last: 99.75       max: 99.75      epoch_max: 0
Saving weights to ./work_dirs\msflow_wide_resnet50_2_avgpool_pl258\textile\last.pt
Saving weights to ./work_dirs\msflow_wide_resnet50_2_avgpool_pl258\textile\best_det.pt

[2023-10-23-15:33:20] Epoch 1.0 train loss: -5.762e+04  lr=5.33e-05
```



MSFlow

경량화 방안

1. 가지치기를 통해 연산에 소요되는 GPU 줄이기
2. 메모리에 할당된 필요 없는 변수 해제함으로써 조금이라도 경량화 진행
3. 모델의 구조 중 extractor 구조를 조금 가벼운 resnet으로 바꿔 학습시키기

경량화 진행

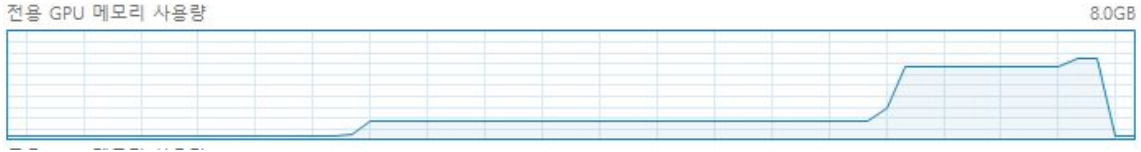
1. 가지치기를 통해 연산에 소요되는 GPU 줄이기

- 가지치기 진행 했을 때 연산속도, 모델의 크기, gpu 메모리 사용량에 있어서 바뀌는 부분이 없어, 다른 경량화 방안 우선 진행 후 재시도

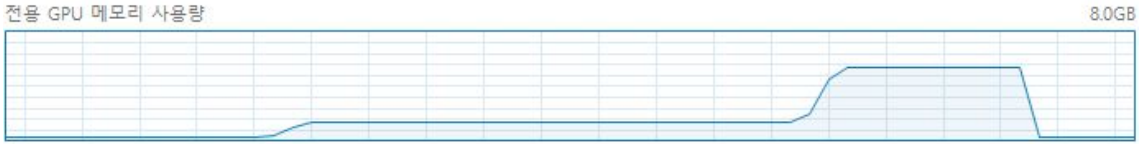
MSFlow

2. 메모리에 할당된 필요 없는 변수 해제함으로써 조금이라도 경량화 진행

- 기존 모델



- 경량화 진행 후 모델



-> post process(모델 출력값을 통해 이상치 맵과 점수를 계산하는 과정)에서 gpu 사용량이 급증하는 현상 완화

3. 모델의 구조 중 extractor 구조를 조금 가벼운 resnet으로 바꿔 학습시키기

- extractor 구조를 기존 wide_resnet50_2보다 가벼운 resnet18, resnet34 구조로 바꿔 학습 완료

MSFlow

성능 - fps 및 정확도

- fps: 초당 이미지 처리 수 (test 데이터 셋 크기 / 모델 forward)
- 정확도: AUROC 점수

※ 성능 측정은 batch size = 400 기준으로 수행

	AUROC	fps	
models (extractor)	-	gpu 0	gpu 1
resnet18	99.76	177	172
resnet34	99.86	180	180
wide_resnet50_2	99.73	149	141

MSFlow

성능 - GPU 사용률, 사용량 / CUDA 메모리 사용량

- GPU 사용률: GPUtil 모듈 활용한 GPU 메모리 사용률 모니터링
- GPU 사용량: GPUtil 모듈 활용한 GPU 메모리 사용량 모니터링
- CUDA 메모리 할당량: torch.cuda.memory_allocated() 활용한 CUDA의 GPU 메모리 할당량 모니터링

※ GPU의 종류가 같은데도 불구하고, 측정량에 차이가 있어 두 GPU 모두 모니터링

	GPU 사용률(%)		**GPU 사용량(GB)**		**CUDA 메모리 할당량(GB)**	
models (extractor)	gpu 0	gpu 1	gpu 0	gpu 1	gpu 0	gpu 1
resnet18	8 ~ 33%	11 ~ 93%	0.93 ~ 1.21	1.35 ~ 4.49	0.10 ~ 1.42	0.09 ~ 1.41
resnet34	7 ~ 29%	12 ~ 96%	0.93 ~ 1.18	1.41 ~ 4.44	0.12 ~ 2.74	0.12 ~ 2.40
wide_resnet50_2	8 ~ 31%	24 ~ 99%	0.93 ~ 1.10	2.92 ~ 6.14	0.74 ~ 2.93	0.75 ~ 2.93

MSFlow

향후 계획

- 가지치기 재 시도
- 추론만을 위한 함수 소스 코드 작성

EfficientAD

- 모델 선정 이유

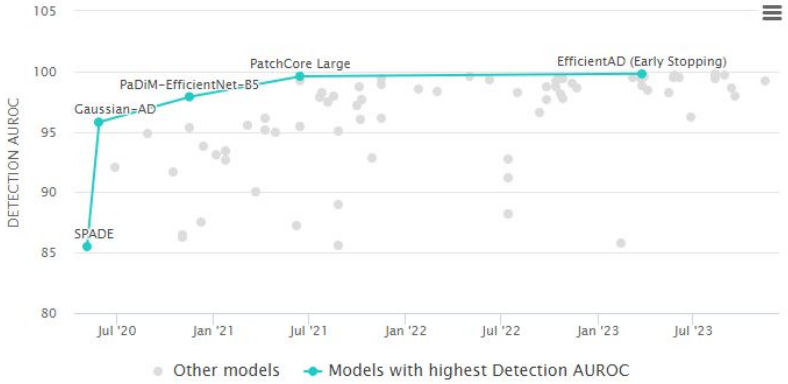
Anomaly Detection 과제에서 가장 중요한 건 Anomaly를 잘 판단하는 것과 얼마나 빠르게 추론하는 가가 중요 (ex> 농작물 밭의 금속 물체가 콤바인 수확기 내부로 들어가는 경우, 작업자의 신체에 칼날이 접근하는 경우 등)

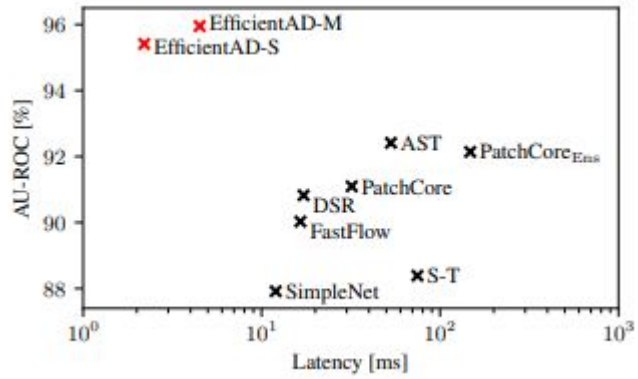
해당 모델의 논문에도 나와있듯이 EfficientAD모델이 다른 모델에 비해 정확도, 속도가 굉장히 우수

또한 Paperswithcode에서도 Anomaly Detection(MVtec 데이터셋 기준)에서 가장 성능이 좋은 모델이고 아직 공개되지않은 모델 CPR을 제외하고는 속도도 가장 빠름

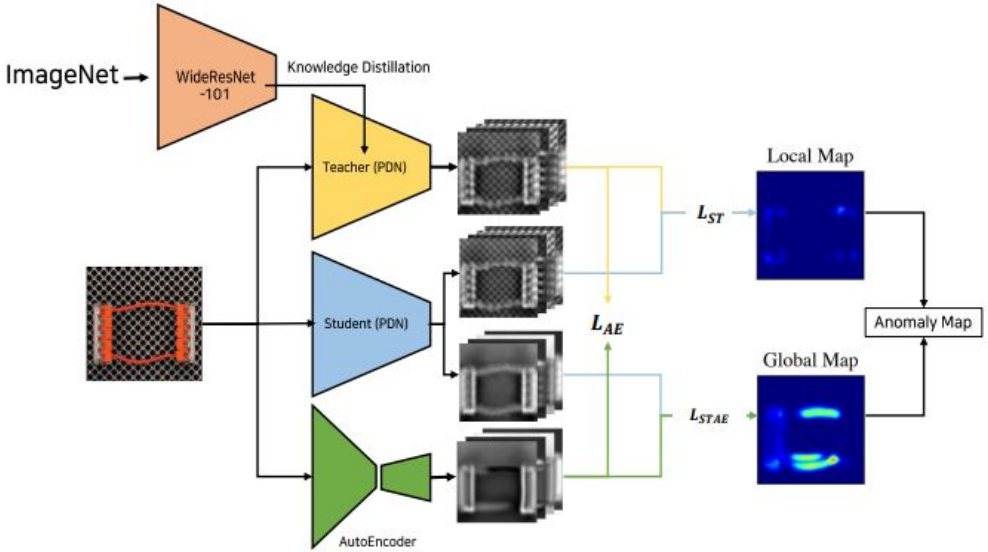
공식 FPS는 약 EfficientAD-S 기준 614

(하지만 측정 방식이 달라서 해당 FPS로 어떻게 계산하는 지는 파악되지 않는다.)





EfficientAD



- 모델 구조

Structural Anomaly & Logical Anomaly 를 둘 다 탐지 해 최종적인 Anomaly Map 완성

원본 이미지의 특징을 추출한 Teacher의 output을 student가 예측함으로써 이상 탐지(이상 있는 부분은 정상 데이터로 학습한 student가 특히나 따라할 수 없기 때문) AutoEncoder 도 비슷한 방법을 통해 Logical Anomaly Detection 탐지.

EfficientAD

- 훈련 과정

데이터 셋 (train - 22930, test - 5324)

Teacher는 원본 이미지를 통해 얼마나 특징을 잘 추출하는 가에 대해서 학습을 진행.

Student와 Autoencoder는 Teacher의 output을 예측하는 형식으로 학습을 진행.

학습 이미지 수에 따라 Anomalies에 대해서 Student가 Teacher를 너무 잘 따라하거나 학습이미지 수를 줄이는 경우 Normal image에 대한 학습 부족이 생김

적절하게 Normal Image의 feature를 잘 학습하는 것이 중요함.

따라서 Teacher를 잘 따라할 수 있음과 동시에 Anomaly Image에 대한 일반화를 피할 수 있게 하기 위해 Hard Feature loss를 통해 학습

또한 Student-Teacher Loss로 Loss penalty term을 추가(normal part가 아닌 부분을 따라하는 것을 방해)

코드 파일 내부에서 EfficientAD-small.py 혹은 EfficientAD-Medium으로 학습이 가능

EfficientAD

- 훈련 과정

EfficientAD_S 와 Efficient_M은 같은 구조를 가지고 있지만 Efficient_M 같은 경우 Efficient_S의 구조를 확장하여 컨볼루션 레이어의 커널 수를 두배로 늘리고 두번 째 풀링 레이어와 마지막 컨볼루션 레이어 뒤에 1*1 컨볼루션을 추가.

추가적인 컴볼루션과 커널 수 증가로 인해 성능이 늘지만 학습시간에서 약 3배 이상 차이

EfficeintAD_S 훈련시간 : 약 1시간 30분

EfficientAD_M 훈련시간 약 5시간

오히려 EfficientAD_M이 S보다 낮은 성능으로 인해 EfficientAD_S를 최종으로 채택

EfficientAD

- 추론 파일

need_module.py : Inference 하기 위한 함수들을 모은 파일,

- ImageFolderWithPath : 이미지를 경로와 함께 받아오는 class
- predict : 모델을 통해서 output을 받고 해당 아웃풋을 통해 anomaly_map을 완성하는 코드
- visualize_one_sample : Anomaly_map을 통해서 예측한 값을 각각 정상, 불량으로 판별하고 예측 class를 반환
- anomaly : 원본이미지에 anomaly_map을 통해서 알아낸 Anomaly Part를 직접 Segmentation하고 해당 이미지를 폴더에 저장

Inference.py : 추론에 직접적인 파일

- 각각 모델들을 불러와 이미지를 모델에 맞게 transform 한다음 추론, 만약 불량인 경우 원본이미지에 불량인 부분까지 표시해서 Segmentation가능.

EfficientAD

- 성능 및 결과

훈련시간 약 1시간 30분

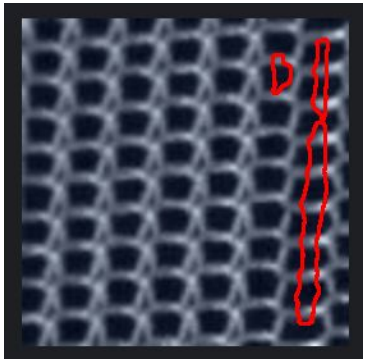
```
Final image auc: 99.8683
```

훈련 코드 제거 후 inference , need_module 두 개의 파일로만 구동이 가능하도록 변경 완료.

원본 이미지에 anomaly 한 부분 Detection해서 표시하는 코드까지 완성

현재 코드상 추론 성능도 매우 우수하고 FPS 를 따져보았을 때 장당 약 100FPS 이상까지도 가능한 것으로 확인 (장당 추론시간 약 0.005초)

GPU 사용량을 따졌을 때 최대 10%만 활용하는 것으로 보아 GPU 사용량도 상당히 좋음



```
GPU percentage usage
Minimum GPU usage: 1.0%
Maximum GPU usage: 10.0%
Average GPU usage: 3.5%


GPU memory usage
Minimum GPU memory usage: 0.82 MB
Maximum GPU memory usage: 1.28 MB
Average GPU memory usage: 1.07 MB
```

EfficientAD

- 향후 계획

추가적으로 TensorRT를 진행 중이나, 구동이 잘 안되는 부분이 존재해서 추가적으로 오류를 해결하고 구동해볼 예정

하지만, 이미 가볍게 나온 모델을 경량화 가능한 지는 잘 모르겠음

→ GPU 메모리 크기와 호스트 메모리 크기가 다른 것에 대해서 오류가 발생, 하지만 변경해줘도 똑같이 계속 오류가 남.