

KAMP 제조AI데이터셋 활용 권한

No. GB20240229-PU1707379903-65055



△ 제조AI데이터셋 활용 주의사항

연구/공식적으로 사용하실 때에는 꼭 아래와 같이 'KAMP 출처(reference)'를 남겨주시고, 인용 시 활용한 내용과 문서 등은 아래 이메일로 보내주시기를 바랍니다.
(E-mail : kamp@kaist.ac.kr)

출처 표기 양식

중소벤처기업부, Korea AI Manufacturing Platform(KAMP), X-ray 검사장비 AI 데이터셋,
KAIST(주)임피스, 한양대학교 산학협력단, (주)아큐라소프트, 2020.12.14., www.kamp-ai.kr

제조AI데이터셋 개요

제조AI데이터셋명 : X-ray 검사장비 AI 데이터셋

업종 : 식품가공

목적 : 품질보증

유형 : bmp

알고리즘 : YOLOv3

사용조건 : 콘텐츠 변경허용

제공기관 : KAIST (수행기관: (주) 임피스/한양대학교 산학협력단/(주)아큐라소프트)

등록일 : 2020-12-14

이용자 정보

회원 ID : untjrgus

다운로드일 : 2024-02-29 20:12:26

활용목적 : 제조AI 교육 및 강의

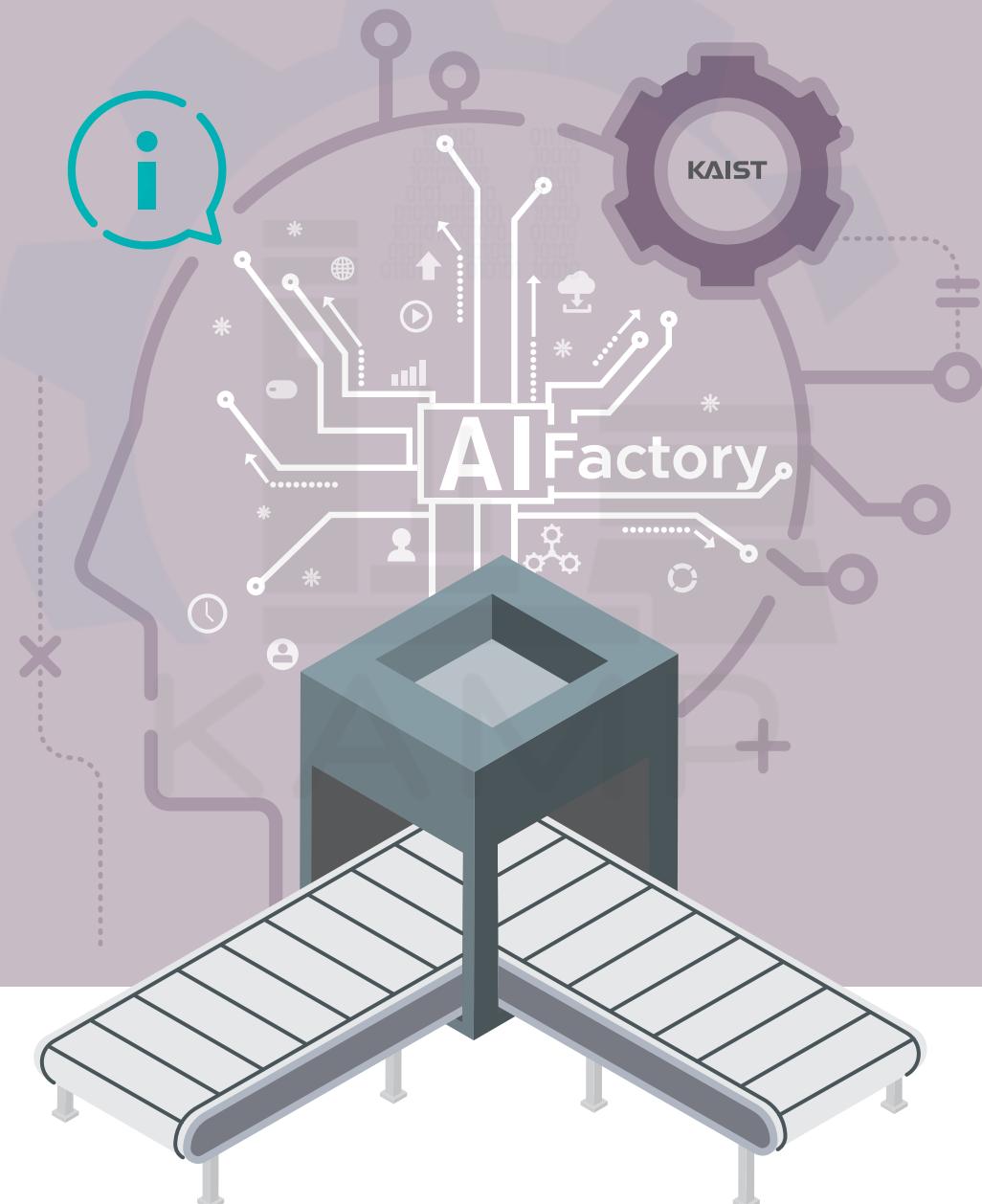
인공지능 중소벤처 제조 플랫폼(KAMP, Korea AI Manufacturing Platform)에서 다운로드한 제조AI데이터셋의 저작권을 위하여 문서 고유 번호와 워터마크를 부여하여, 해당 문서의 표준을 준수하였음을 증명합니다.

운영기관 : KAIST 제조AI빅데이터센터

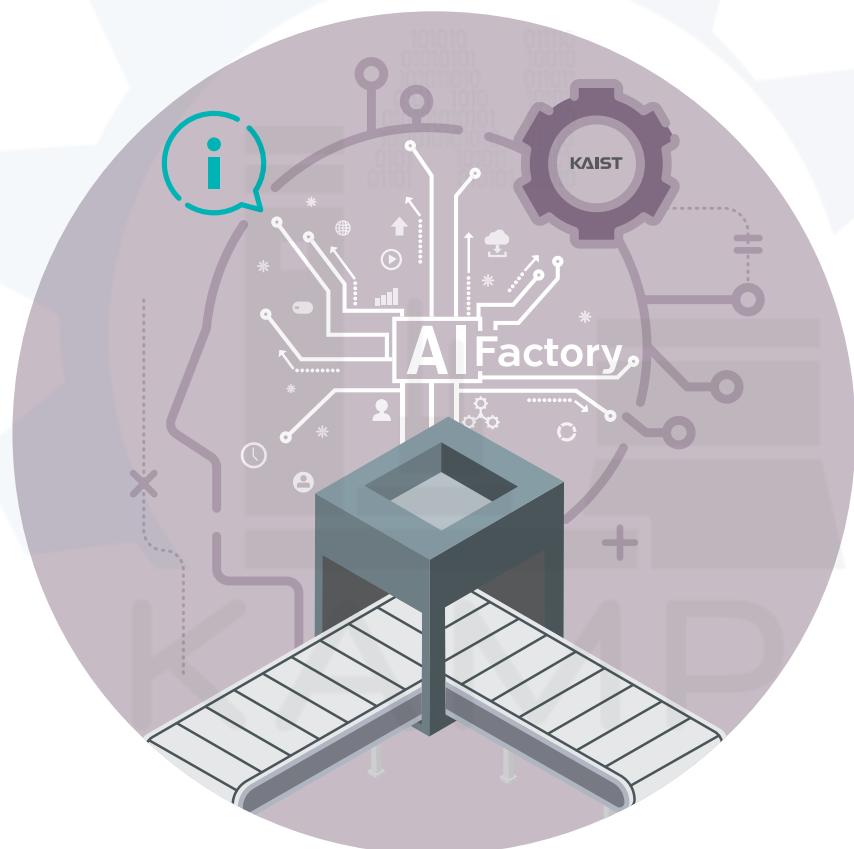
주소 : 대전광역시 유성구 문지로 193 KAIST 문지캠퍼스 행정동

전화번호 : 042-350-1331 | 이메일 : kamp@kaist.ac.kr

「X-ray 검사장비 AI 데이터셋」 분석실습 가이드북

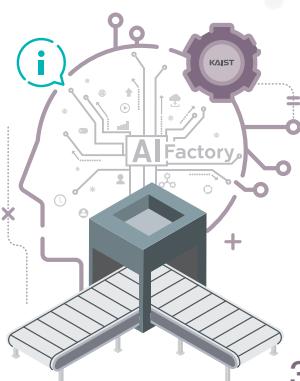


「X-ray 검사장비 AI 데이터셋」 분석실습 가이드북



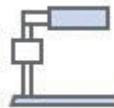
Contents

| | |
|---|-----------|
| 1 분석요약 | 04 |
| 2 분석 실습 | |
| 1. 분석 개요 | 05 |
| 1.1 분석 배경 | 05 |
| 1) 공정(설비) 개요 | |
| 2) 이슈사항(Pain point) | |
| 1.2 분석 목표 | 06 |
| 1) 분석목표 | |
| 2) 제조 데이터 정의 및 소개 | |
| 3) 제조 데이터 분석 기대효과 | |
| 4) 시사점(implication) 요약기술 | |
| 2. 분석실습 | 07 |
| 2.1 제조데이터 소개 | 07 |
| 1) 데이터 수집 방법 | |
| 2) 데이터 유형/구조 | |
| 2.2 분석 모델 소개 | 09 |
| 1) AI 분석모델 | |
| 2.3 분석 체험 | 16 |
| 1) 분석 단계별 Process | |
| 2) 분석 실습 | |
| [단계 ①] 라이브러리/데이터 불러오기 | |
| [단계 ②] 데이터 종류 및 개수 확인 | |
| [단계 ③] 데이터 정제(전처리) | |
| [단계 ④] 데이터 특성 파악 | |
| [단계 ⑤] 학습/평가 데이터 분리 | |
| [단계 ⑥] 모델 구축 | |
| [단계 ⑦] 모델 훈련 | |
| [단계 ⑧] 결과 분석 및 해석 | |
| 3. 유사 타 현장의 「X-ray 검사장비 AI 데이터셋」 분석 적용 | 50 |
| 부 록 | |
| 분석환경 구축을 위한 설치 가이드 | 51 |
| GPU 활용을 위한 CUDA 설치 | 62 |



「X-ray 검사장비 AI 데이터셋」 분석실습 가이드북

- 필요 SW :** Anaconda, Python 3.7.3, (or version of python 3.x.x)
- 필요 패키지 :** Numpy, Pandas, Matplotlib, Keras, Sklearn, Tensorflow, Ipython, Pydotplus, Os, Graphviz, Random, Subprocess, Sys
- 분석 환경 :** [CPU] i5-10400, [GPU] gtx 1660 super, [RAM] 32GB
- 필요 데이터 :** X선이물검출기(06.23_09.22) 폴더
- 주 관 기관 :** 한국과학기술원(KAIST)
- 수행 기관 :** (주)임픽스, 한양대학교 산학협력단, 주식회사 아큐라소프트



1 분석요약

| No | 구분 | 내용 |
|----|----------------------|---|
| 1 | 분석 목적 (현장 이슈, 목적) | <ul style="list-style-type: none"> - 포장공정의 마지막 단계에 완제품 내 금속 및 이물 포함 여부를 검출하기 위해 진행되는 X-RAY 비 전검사는 소비자에게 완제품이 전달되기 전 생산라인에서 진행되는 마지막 품질검사임에 따라 검사장비가 판정오류 없이 제대로 작동하는 것이 매우 중요하다. - X-RAY 검사기는 제조라인에서의 적용편의성, 인식률 등의 장점이 많은 반면, 이중·삼중 검사 시 종종 발견되는 미검·과검의 문제가 있어, X-RAY 검사 데이터의 AI 분석을 통해 장비셋팅, 검사기준 또는 여타 스펙을 개선할 수 있는 학습모델이 필요하게 되었다. |
| 2 | 데이터셋 형태 및 수집방법 | <ul style="list-style-type: none"> - 분석에 사용된 변수 : X-RAY상 결함(Defect)이 검출되어 표시된 이미지 - 데이터 수집 방법 : X선이물검출기의 SD Card - 데이터셋 파일 확장자 : 결함 이미지(bmp), 라벨링 데이터(jpg, txt) |
| 3 | 데이터 개수 데이터셋 총량 | <ul style="list-style-type: none"> - 데이터 개수 : 3,919개(bmp, jpg), 530개(txt) - 데이터셋 총량 : 419MB |
| 4 | 분석적용 알고리즘 | 알고리즘 YOLOv3 |
| | | <ul style="list-style-type: none"> - 객체 탐지(Object Detection)의 주요 방법론 중 하나로서 이미지 내의 개별 객체의 위치를 추적하고 분류하는 방법론이다. - 객체가 시야 안에 있을 때 바로 판단하는 사람의 시각 기관(Human visual system)과 유사하도록 모델을 단일 신경망(single neural network)으로 구성한다. - 객체 탐지를 단일 회귀문제로 재구성하여 분석한다. - 이미지에 대한 경계 상자(Bounding Box)로 영역을 설정하고, 영역 내 물체의 종류(Class)나 유무를 판단하여 분류 및 객체 탐지 문제를 한 번에 해결한다. |
| 5 | 분석결과 및 시사점 | <ul style="list-style-type: none"> - X-RAY 검사기를 사용하여 품질검사를 하는 검사공정에서 이미지 데이터의 분석 및 학습을 통해 AI 객체 탐지 분석모델을 구축하였다. - 개발된 AI 모델을 통하여, X-RAY검사 시 제조기업에서 흔히 발생하는 미검·과검 문제를 해결하여 비용절감을 할 것이라고 기대한다. |

2 분석 실습

1. 분석 개요

1.1 분석 배경

1) 공정(설비) 개요

- 공정(설비) 정의 및 특징

- X-RAY 검사기는 포장공정의 마지막 단계에 사용되는 검사장비로, 완제품을 개봉·침투하지 않고 엑스레이를 통한 투사 결과를 나타내어 내부 상태를 확인하여 금속 및 이물 포함 여부를 검출할 수 있게 한다.

2) 이슈사항(Pain point)

- 공정(설비)상의 문제현황

- 포장공정의 비전검사는 소비자에게 완제품이 전달되기 전 생산라인에서 진행되는 마지막 품질검사임에 따라 검사장비가 판정오류 없이 제대로 작동하는 것이 매우 중요하다.
- 포장공정 품질검사에 주로 사용되는 X-RAY 검사기는 제조라인에서의 적용편의성, 인식률 등의 장점이 매우 많음에도 불구하고, 이중·삼중 검사 시 종종 발견되는 미검·과검의 문제가 여전히 남아있다.

* 미검 : 불량이 있어도 발견하지 못하고 양품(OK) 판정
 과검 : 양품임에도 과도하게 검수하여 불량(NG) 판정

- 문제해결 장애요인

- 검사장비 자체의 한계로 인한 미검 및 과검 문제를 해결하기 위해서는 비전검사를 이중, 삼중으로 진행하여야 하는데, 이는 해당 라인의 생산흐름과 작업동선 등의 변화와 중복작업으로 인한 비용낭비를 야기할 수도 있다. 또한, 장비의 기술적 한계라는 근본적 원인은 여전히 남아있기 때문에 반복 검사를 통해서도 모든 불량이 발견되지 못했을 가능성은 여전히 남게 된다.

- 극복방안

- 검사장비 자체의 기술적 한계라는 근본적 원인은 생산현장에서 해결할 수 있는 문제는 아니나, 대신 장비에서 수집한 데이터의 AI 분석을 통해 이를 보완할 수는 있다. 검사장비에서 발생하는 X-RAY 이미지 데이터를 대상으로 AI 객체 탐지 분석모델을 개

발하여 장비가 자체적으로 발견하지 못한 불량을 발견하여 최종 제품 품질에 문제가 없도록 한다.

1.2 분석 목표

1) 분석목표

- 포장공정 품질검사에 주로 사용되는 X-RAY 이물검출기는 제조라인에서의 적용편의성, 인식률 등의 장점이 매우 많음에도 불구하고, 장비 자체의 기술적 한계 때문에 양품이 불량으로 판정되거나(과검), 불량이 있는데도 양품으로 판정되는(미검) 문제가 종종 발생한다. 검사장비를 교체하지 않는 한 해결될 수 없는 근본적 기술적 한계를 보완하기 위해 검사장비에서 수집되는 X-RAY 사진이미지를 라벨링하고 학습한 객체 탐지 분석 모델을 개발하여 이를 바탕으로 포장제품의 양품/불량품을 정확하게 판정하고자 한다.

2) 제조 데이터 정의 및 소개

- X-RAY 이물검출기는 장비를 지나가는 모든 포장제품의 내부를 X-RAY로 촬영하여 이물을 감지하고 불량여부를 판정한다. 이와 같은 검사과정에서 생성되는 사진 이미지를 활용하여 더욱 정확한 불량 판정을 하기 위한 머신학습 데이터이다.

3) 제조 데이터 분석 기대효과

- 비전검사에서 생성되는 이미지 데이터를 통해 불량여부를 판정할 수 있는 검사장비에 넓게 활용할 수 있을 것으로 판단된다. 본 분석을 통해 이미지 데이터의 결함(defect)을 라벨링하고 학습하여 비전 이물검사 단계의 정확성을 높일 수 있는 데이터셋을 구축한다. 또한, 이와 같이 구축한 AI 학습모델은 X-RAY 검사장비뿐만 아니라 다른 종류의 비전검사 장비에도 적용될 수 있을 것으로 기대된다.

4) 시사점(implication) 요약기술

- 식품제조업체 H사의 포장공정에서 사용하는 X-RAY 이물검출기는 장비 자체의 기술적 한계로 인한 미검·과검 문제 때문에 완벽히 제품 불량을 검출하기 어려웠다. 따라서 본 가이드북에서는 이물검사결과 이미지 데이터를 대상으로 한 AI 객체 탐지 분석모델을 기반으로 포장공정의 비전검사에서 보다 정확하면서 신속한 결함(defect) 검출을 도모한다.

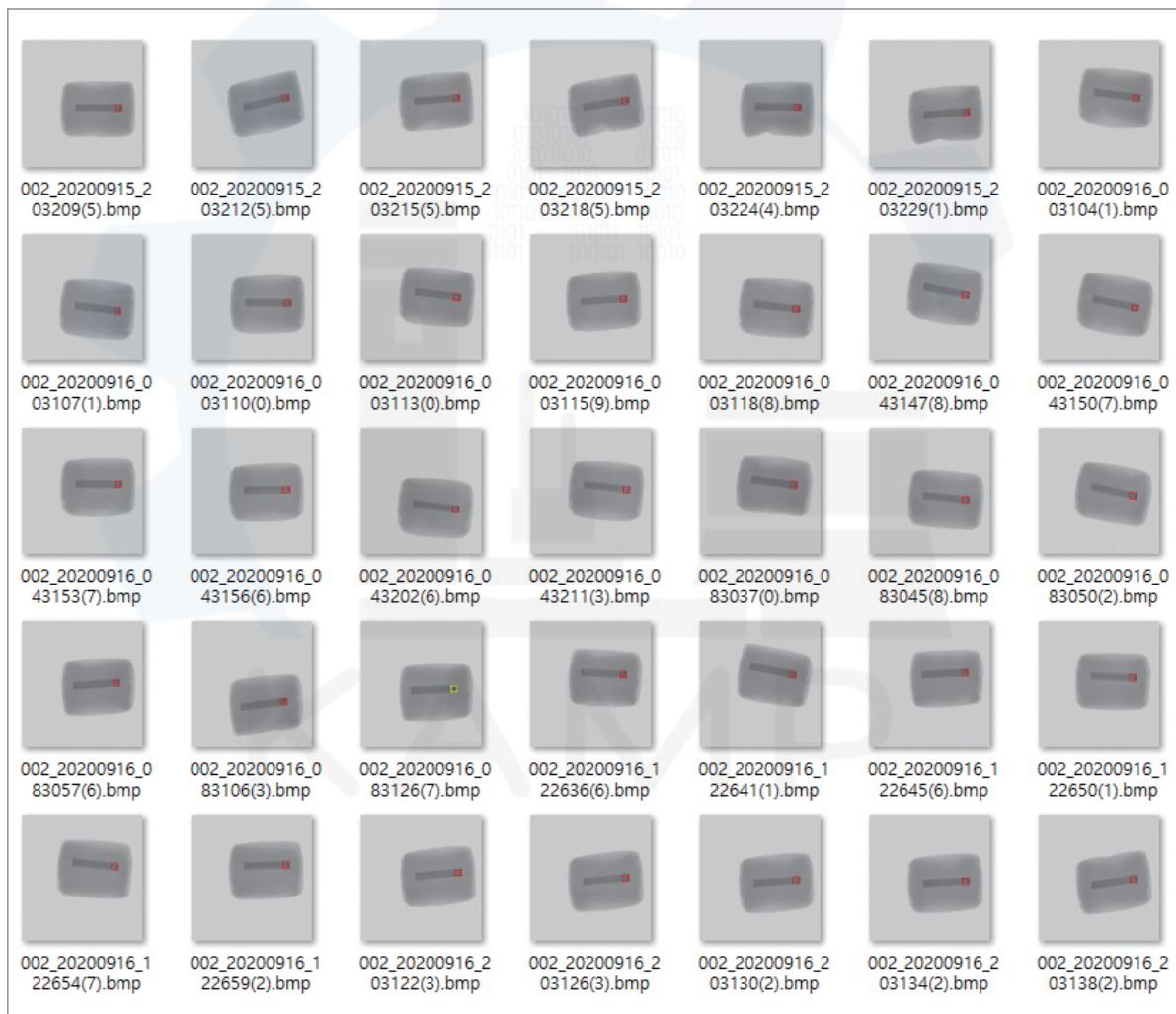
2. 분석실습

2.1 제조데이터 소개

1) 데이터 수집 방법

- 제조 분야 : 분무건조공법을 이용한 분말유크림 제조
- 제조 공정명 : 포장공정 비전검사 단계
- 수집장비 : X-RAY 이물검출기의 SD Card¹
- 수집기간 : 2020년 6월 23일 ~ 2020년 9월 22일 (약 3개월)
- 수집주기 : 불규칙 (※ 검사장비를 통과하는 제품이 불량일 경우에만 이미지를 저장한다.)

2) 데이터 유형/구조



[그림 1] X-RAY 이물검출기 이미지 데이터 스크린샷

1) 데이터는 X선이물검출기의 SD Card에 자동 저장되는 데이터를 PC로 열어보는 방식으로 수집한다.

• 데이터 크기, 데이터 수량 :

- [1호기] 폴더 - 1,031개, 112MB
- [2호기] 폴더 - 920개, 92.9MB
- [3호기] 폴더 - 858개, 210MB

* 데이터는 [X선이물검출기(06.23_09.22)] 폴더 내에 설비별로 [1호기], [2호기], [3호기] 폴더로 정리되어 있으며, 각 설비 폴더 안에는 일자별로 결함이미지(bmp) 폴더가 있다.

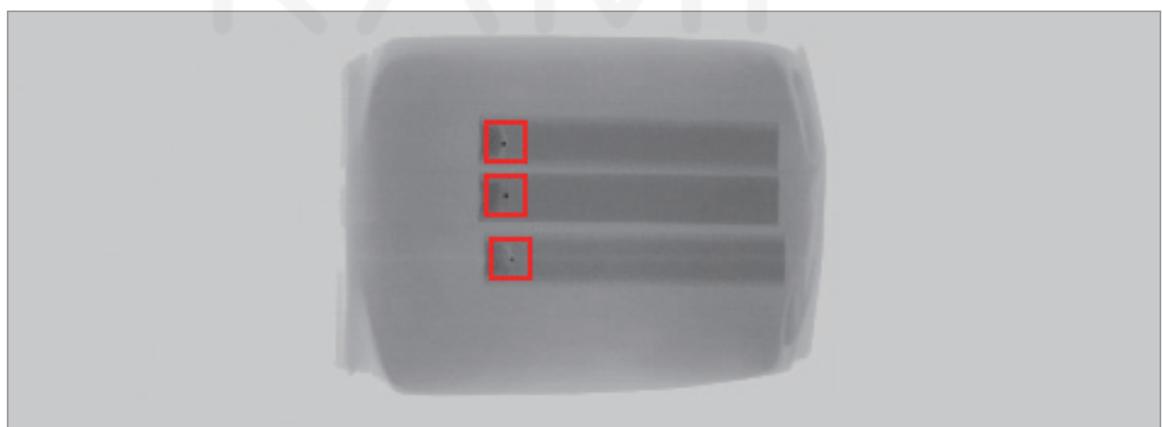
• 데이터 속성정의 표 :

| 변수명 | 설명 | 데이터 타입 |
|------------------|---|---------------------|
| SN_NgImage | X선이물검출기를 통해 수집된 X-RAY상 결함(defect)이 검출되어 표시된 이미지 | BMP형식 이미지 파일 (.bmp) |
| SN_NgImage_Label | 이미지의 결함 좌표(위치)를 나타내는 텍스트 파일 | 텍스트 파일 (.txt) |

- 데이터는 총 2,809개(415MB)의 이미지(BMP) 파일이며, 분석에 사용되는 데이터(이미지)의 개수는 라벨링 개수에 따라 달라진다. 이미지에 대한 라벨링(Labeling)을 통해 결함의 좌표를 얻는데, 모델 학습을 위한 라벨링 개수는 사용자 지정이며, 사용자가 라벨링을 하는 만큼 모델 학습을 위한 데이터를 사용하는 것이다. 라벨링을 많이 한다고 하여 모델 예측도가 반드시 다 상승하는 것이 아니므로 모델의 학습 성능 평가로 합리적인 라벨링 개수도 판단하고자 한다.

• 양품/불량품 데이터 설명 :

- X-RAY에서 결함(defect)이 검출된 제품의 이미지는 다음과 같다. 붉은색 상자로 표시된 부분의 중심의 검정색 부분이 제품의 결함 부분이며, 양품은 검정색 부분과 상자로 표시된 것이 없는 데이터이며 따로 수집되지 않는다.



[그림 2] 결함이 검출된 불량품의 X-RAY 사진 예시

• 독립변수/종속변수 정의 :

| 구 분 | 명 칭 | |
|------|-----------|--|
| 독립변수 | X-RAY 이미지 | 결함이 포함된 이미지(BMP) 파일 |
| | 결함 좌표 | 이미지 파일의 결함(Defect)에 대해 좌표 정보가 담겨있는 TXT(텍스트) 파일 |
| 종속변수 | 객체탐지 이미지 | 객체탐지(Object Detection) 박스가 표시된 이미지(JPG) 파일 |

* 독립변수란 다른 변수에 영향을 받지 않는 변수로, 입력값이나 원인을 나타낸다.

종속변수란 독립변수의 변화에 따라 어떻게 변하는지를 알고자 하는 변수를 말하며, 결과물이나 효과를 나타낸다.

- 본 분석의 독립변수는 X-RAY 검사장비에서 수집된 결함이 포함된 X-RAY 이미지 (BMP)와 결함 이미지에 대해 라벨링 작업을 통해 추출한 결함좌표 텍스트(TXT)이다. 학습모델에 이 두 종류의 입력값을 투입하여 모델을 학습한 후, 라벨의 위치가 포함되지 않는 데이터에 대하여 객체탐지를 수행하여 얻는 결과물인 객체탐지 박스가 표시된 이미지(JPG)가 종속변수가 된다.

2.2 분석 모델 소개

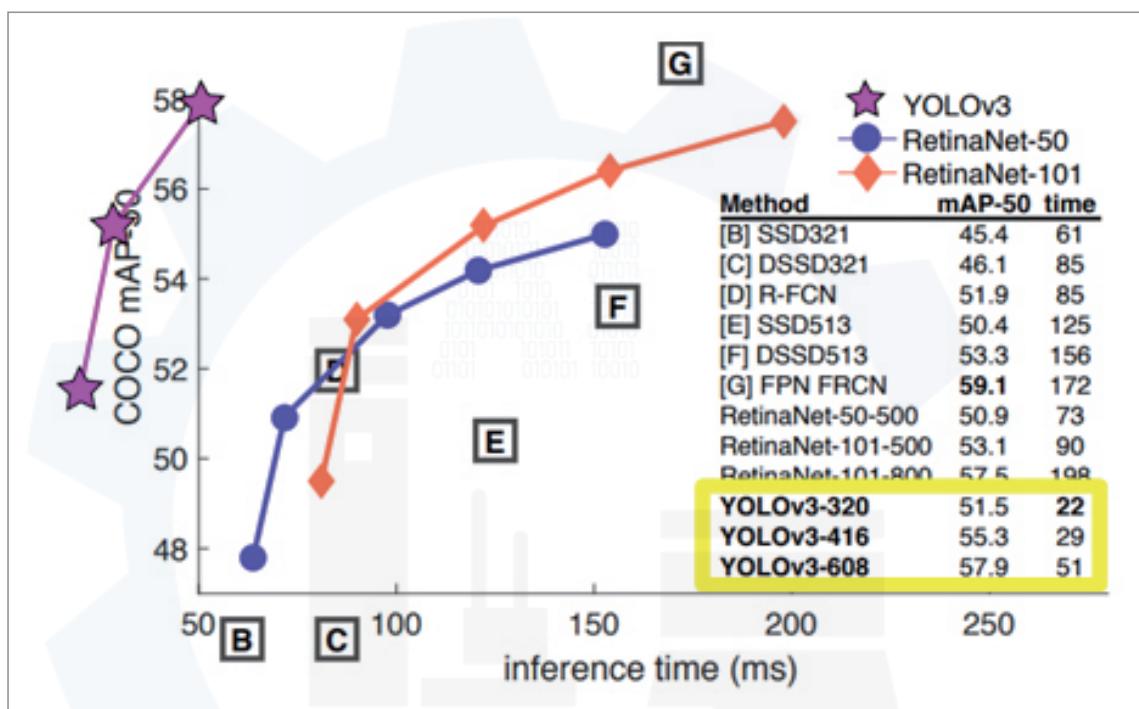
1) AI 분석모델

• 해당 AI 방법론(알고리즘) 선정 이유 기술

- X-RAY 이물검출검사는 포장공정의 마지막 단계이자, 완제품 생산의 마지막 단계이기 때문에 보다 정확하게 결함을 검출하여 판정오류를 방지해야 할 필요가 있다. 현재 가용한 방법으로는 이중·삼중 검사를 수행하는 것인데, 이는 라인의 생산 흐름의 버퍼와 작업 동선 등의 변화, 추가적인 작업으로 인한 비용 낭비 등을 야기한다. 이를 위해 보다 정확하면서 신속한 AI 모델의 보완이 필요하다고 판단된다.
- 찾아내고자 하는 결함에 대하여서는 X-RAY 이물검출기에서 1차적으로 수집하여 이미지로 보관한다. 각 이미지에는 검출된 결함이 표시되어 있으며, 이러한 이미지에 대하여 객체 탐지 방법론을 적용하여 X-RAY 이물검출기에서 확인된 결함을 객체로 탐지하여 불량을 구분해 낼 수 있다.
- 객체 탐지 인공지능 모델의 종류로써 YOLOv3는 객체(object)가 시야 안에 있을 때 바로 판단하는 사람의 시각 기관(human visual system)과 유사하도록 모델을 단일 신경망(single neural network)으로 구성하기에 이러한 방법론이 이중·삼중의 검사를 수행하는 인력을 대체할 수 있을 것으로 판단되었다.
- 모델의 기본적 동작 원리는 X-RAY 이물검출기에서 발생한 검사이미지에서 제품의 결함 위치 영역을 확인하고, 결함의 종류를 구분하는 데 적합하다고 판단되는데, 이는 객체를 둘러싸고 있는 경계 상자(bounding box)와 그 내부의 객체에 대한 종류(class)를 예측하는 회귀(regression) 문제로 객체 탐지(object detection)를 수행하는

모델이기 때문이다. 이러한 모델의 이론적 배경은 실시간으로 공정 X-RAY 검사설비에서 출력되어 나오는 제품 결함(defect)에 대하여 사람과 같이 신속한 판단과 결함 인지, 영역 추출 등을 수행하기에 적합하다고 판단되었다.

- 본 가이드북에서 활용된 YOLOv3는 보다 정확하고 신속한 결함 탐지를 수행하며 결함의 영역을 추출하여 시각화된 결과를 제시하는 장점도 가진다.
- 객체 탐지 모델로 많이 활용되는 R-CNN 관련 모델(Faster R-CNN 등의 모델)은 일정 수준의 정확도는 보장하나, 속도적인 측면에서 한계점이 있다. 반면, YOLOv3는 정확도의 성능은 유지하면서 빠른 속도를 낸다.



[그림 3] 정확도와 속도에 관하여 논문에서 주장하는 YOLOv3의 성능

* Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." arXiv preprint arXiv:1804.02767 (2018).

| Year | Object Detection Model | COCO Dataset (mAP@IoU=0.5:0.95) | Conference |
|------|------------------------|------------------------------------|------------|
| 2014 | R-CNN | - | CVPR |
| 2015 | Fast R-CNN | 19.7 | ICCV |
| 2015 | Faster R-CNN | 21.9 | NIPS |
| 2016 | YOLO v1 | - | CVPR |
| 2016 | SSD | 31.2 | ECCV |
| 2017 | DSSD | 33.2 | arXiv |
| 2017 | TDM | 37.3 | CVPR |
| 2017 | FPN | 36.2 | CVPR |
| 2017 | YOLO v2 | - | CVPR |
| 2017 | DeNet | 33.8 | ICCV |
| 2017 | CoupleNet | 34.4 | ICCV |
| 2017 | RetinaNet | 39.1 | ICCV |
| 2017 | DSOD | - | ICCV |
| 2017 | SMN | - | ICCV |
| 2018 | YOLO v3 | 33 | arXiv |

[그림 4] : 여러 학회에서 제시된 객체 탐지 모델의 종류와 COCO 데이터셋에 대한 성능

• 적용하고자 하는 AI 분석 방법론(알고리즘)의 구체적 소개

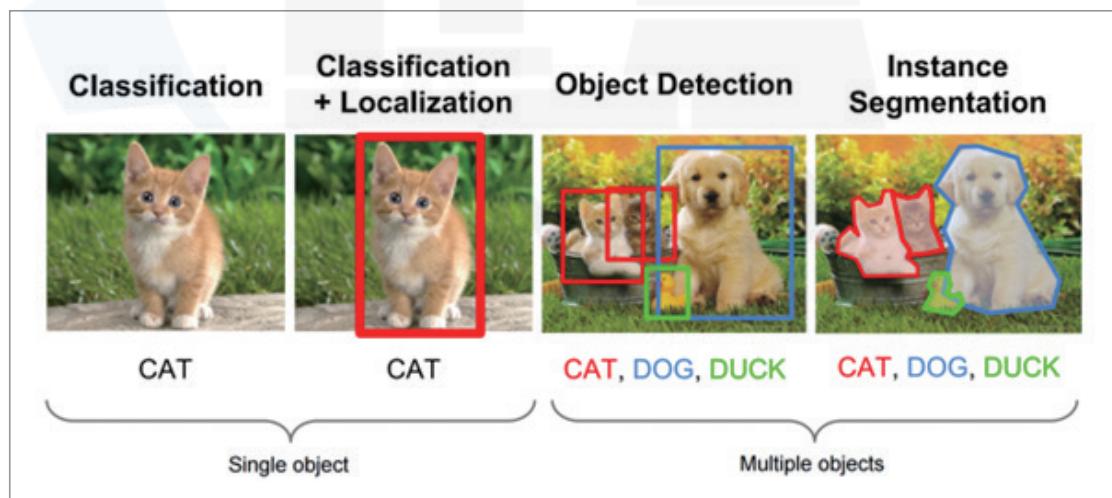
- 기본 개념 관련 참고 논문
 - You Only Look Once: Unified, real-time object detection(2015)
 - YOLO9000: Better, Faster, Stronger(2016)
 - YOLOv3: An Incremental Improvement(2018)
- 이러한 모델의 적합성은 상기 YOLO 기본 개념 논문에서도 확인할 수 있다. 논문 초록의 내용은 [그림 5]와 같다. 객체 탐지를 단일 회귀로 재구성하여 이미지에 대한 경계 상자(Bounding Box)로 영역을 설정하고, 영역 내 물체의 종류(Class)나 유무를 판단하여 분류와 객체 탐지를 한꺼번에 수행하는 방법론으로 YOLO 모델이 제안된 것이다.

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

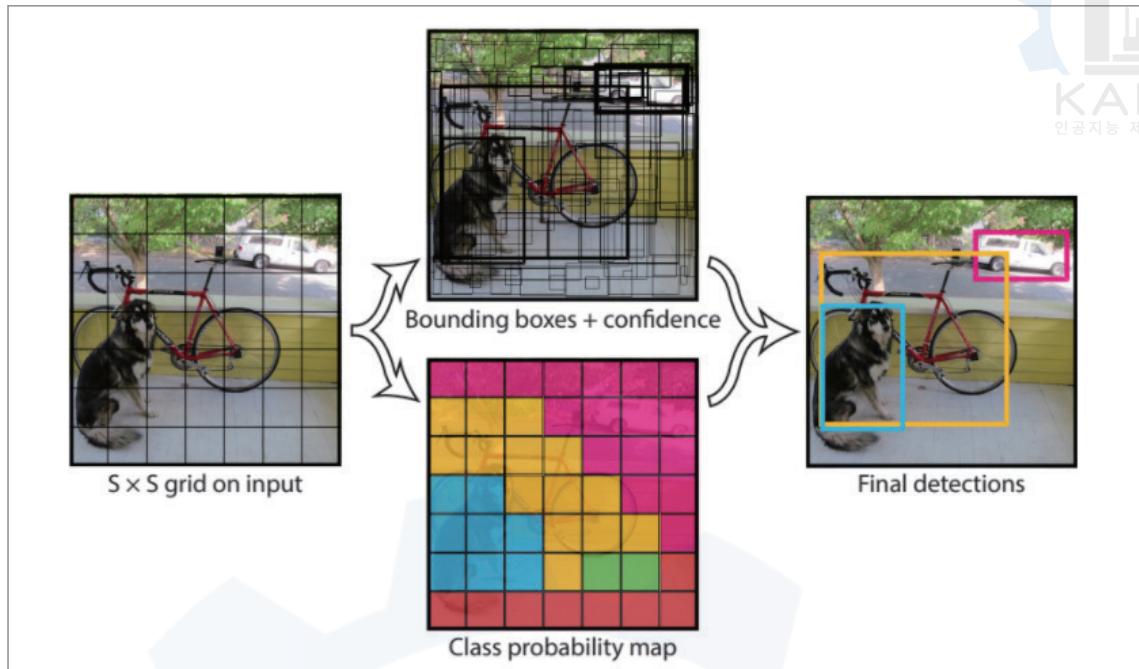
[그림 5] YOLO 기본 개념 논문 초록

- 심층신경망(Deep neural network)으로 경계상자 조정(bounding box coordinate)과 분류(classification)를 단일신경망(single neural network)을 통해 동시에 실행하는 통합 탐색(unified detection)을 구현 가능하게 하는 모델이다.
- YOLOv3는 크기가 256x256인 이미지에 대해 최신 GPU 기준으로 다음과 같은 성능을 가질 수 있기에 빠르고 정확한 결합 검출을 요구하는 이물검사공정에서 적합한 AI 모델로 판단되었다.
 - 170FPS(Frames per second, 170프레임/초)의 속도로 실행 가능하다.
 - mAP(Mean Average Precision, 정확도) 또한 Faster R-CNN과 유사하다.



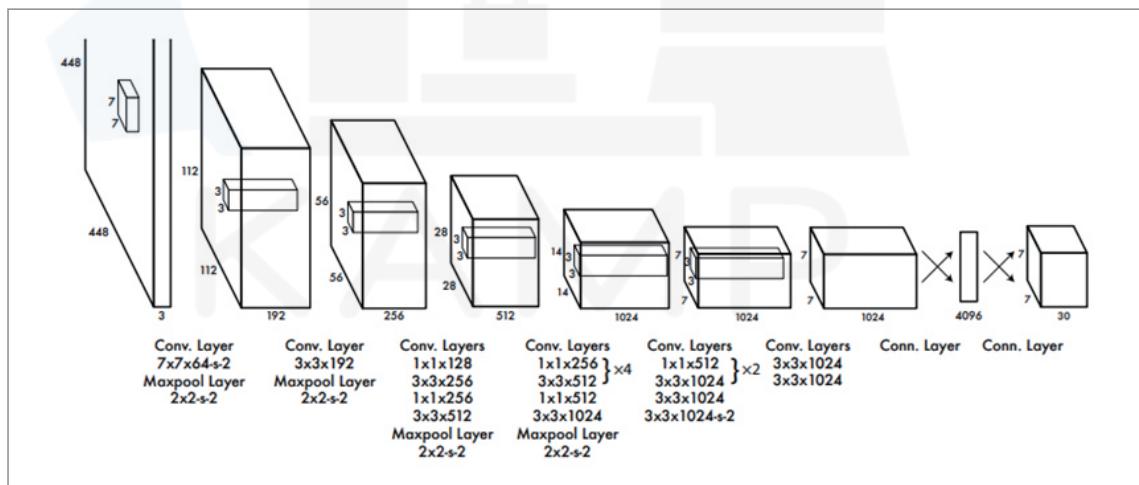
[그림 6] 객체 탐지에 관한 방법론의 구분

- [그림 6]은 논문에서 제시하는 핵심 아이디어로, 하나의 입력(input, image)이 모델에 투입되고 하나의 신경망을 통과하여 물체의 경계 상자(bounding box)와 종류(class)를 동시에 예측한다.



[그림 7] 슬라이딩 윈도우나 다른 복잡한 기법을 사용하는 대신 입력을 그리드로 나누어 그리드별로 작업 진행하는 YOLO 모델의 특징을 나타낸다.

- 반면 YOLO 모델은 신경망에 입력값(input, image)으로 특정 이미지 전체를 넣고, 특징 추출 후 경계 상자(bounding box)의 정보, 신뢰 점수(confidence score), 클래스(class) 확률 등을 포함하는 예측 텐서(Prediction Tensor)를 생성한다. [그림 8]과 같이 그리드별 예측 정보로 경계 상자(bounding box)를 조정하여 분류를 수행한다.



[그림 8] 그리드별 예측 정보로 경계상자를 조정하여 분류 수행

- 입력 이미지가 모델에 투입되면 합성곱 신경망들을 거쳐 말단의 모든 노드의 연결된 레이어로 재구성(Reshaping)을 진행한다. 다음으로는 예측 텐서(Prediction tensor)가 생성되며 분류 및 회귀를 수행한다. 작은 사각형은 합성곱 필터(Convolutional filter)로 특성맵(Feature map)을 추출한다.

- 예측 텐서(Prediction tensor)에 대해서는 다음과 같다. 클래스확률과 경계상자에 대한 좌표와 확률을 얻는다.

$$w \times h \times (\text{각각의 그리드에서 갖는 Bounding box의 후보 갯수}) \times 5 + (\text{Class의 갯수})$$

- 다음은 논문에 제시된 YOLO의 기본 학습 과정에 대한 수식이다.

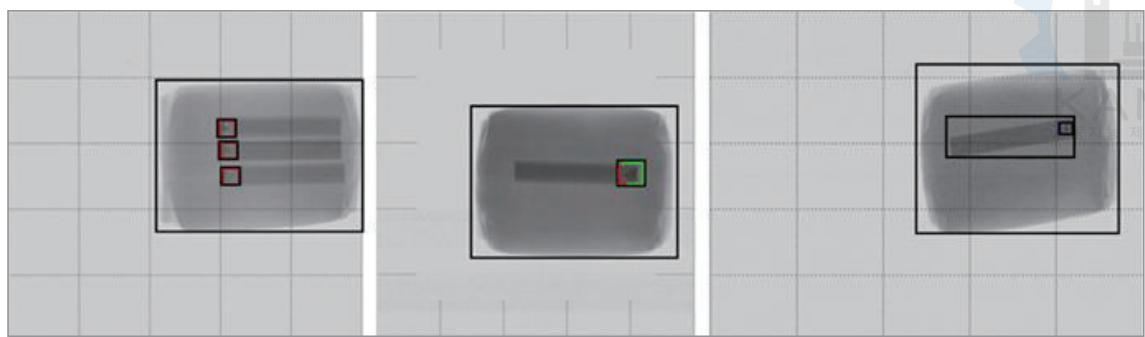
| | |
|--|--|
| <p>loss function:</p> $\lambda_{\text{coord}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$ $+ \lambda_{\text{coord}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$ <p>(Size & Location of Bounding box) (Sum squared error for optimize)</p> $+ \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$ $+ \lambda_{\text{noobj}} \sum_{i=0}^S \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$ <p>(Object detection)</p> $+ \sum_{i=0}^S \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$ <p>(Class)</p> | <p>Key Parameter:</p> <p>Coordinate Loss + Confidence-Score Loss + No-Object Penalties + Classification Loss</p> <p>w, h: 그리드 가로 세로</p> <p>S: Number of grid</p> <p>C: Confidence score</p> <p>B: 그리드 내 box 수</p> <p>x, y: Bounding box 중심좌표</p> <p>{p(c)}: i cell의 object 가 해당 Class에 속할 확률</p> <p>$\lambda_{\text{coord}}, \lambda_{\text{noobj}}$: Bounding box에 대한 값 조정</p> <p>$\mathbb{1}_i^{\text{obj}}, \mathbb{1}_{ij}^{\text{noobj}}$: 1 or 0 object 여부 확인</p> |
|--|--|

[그림 9] YOLO 기본 학습 과정 수식

* Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

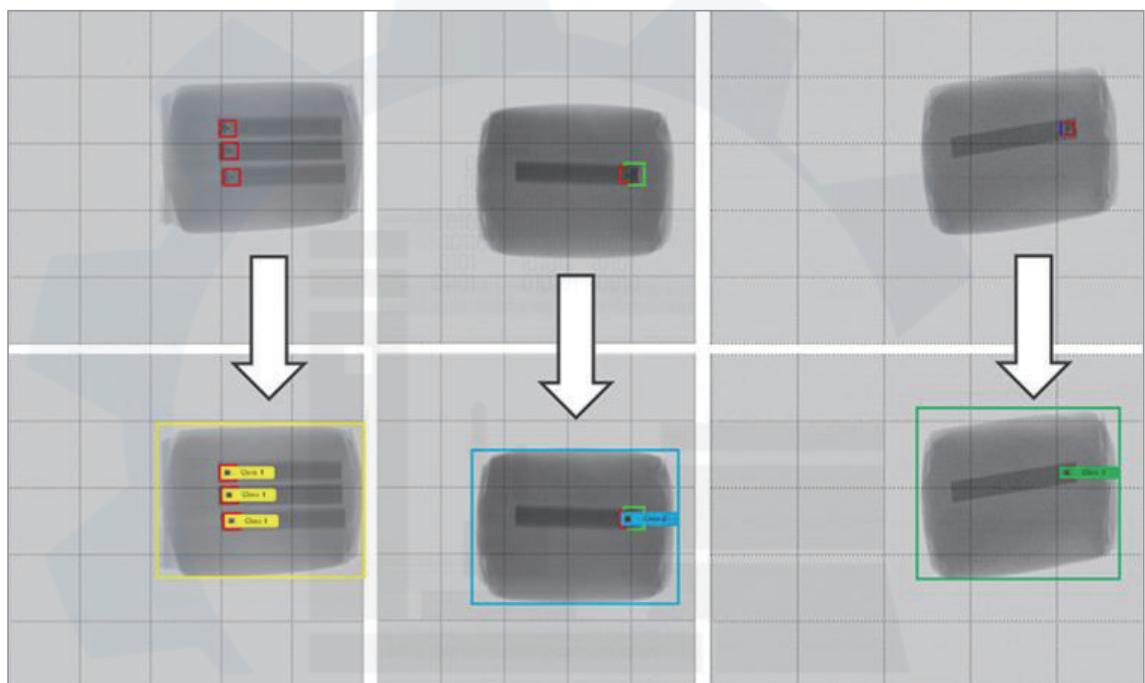
• AI 분석 방법론(알고리즘) 구축 절차 설명

- 그리드의 각 부분에 경계상자를 B개로 정의하고 다음 3가지를 예측하여 결과를 내는 것이 YOLO의 핵심 내부 프로세스가 된다.
 - 경계 상자(Bounding Box)의 중심과 너비와 높이
 - 경계 상자(Bounding Box)가 객체(Object)를 포함하고 있을 확률
 - 객체(Object)가 속한 클래스
- 실제 데이터에 대한 입력과 앞선 모델에서 수행하는 내부 프로세스 과정과 기대 결과 출력 샘플은 아래와 같다.



[그림 10] 라벨링 된 X-ray 투사 결과 이미지

- 라벨링 된 X-ray 투사 결과 이미지를 모델에 투입하여 그리드의 각 부분에 경계상자를 B 개로 정의하고 다음과 같이 3가지를 예측해 결과를 내는 것으로 진행한다.



[그림 11] YOLO 내부 프로세스에 따른 기대결과 출력 샘플

- 객체 탐지(Object Detection) 후 결과와 평가 기준들은 다음과 같다.

| 실제 | 객체 탐지 결과 | | | |
|-----|----------|----------|----------|----------|
| | 양 품 | | 불 량 | |
| | 양 품 | 불량정탐(TP) | 양품과검(FN) | 불량정탐(TN) |
| 양 품 | | | | |
| 불 량 | | | | |

- **불량정탐** : 옳은 검출
- **불량미검** : 검출되어야 할 것이 검출되지 않은 것
- **양품과검** : 잘못 검출된 것
- **양품정탐** : 검출되지 않아야 할 것이 검출되지 않은 것

- 정밀도 : 양품정탐을 양품정탐과 불량미검의 합으로 나눈 것으로, 모델이 양품이라고 분류한 것 중 실제 양품의 비율이다.

$$\text{정밀도}(\text{Precision}) = \frac{TP}{TP+FP} = \frac{TP}{\text{모든 검출}}$$

- 재현율 : 양품정탐을 양품정탐과 양품과검의 합으로 나눈 것이다.

$$\text{재현율}(\text{Recall}) = \frac{TP}{TP+FN} = \frac{TP}{\text{실제 모든 양품}}$$

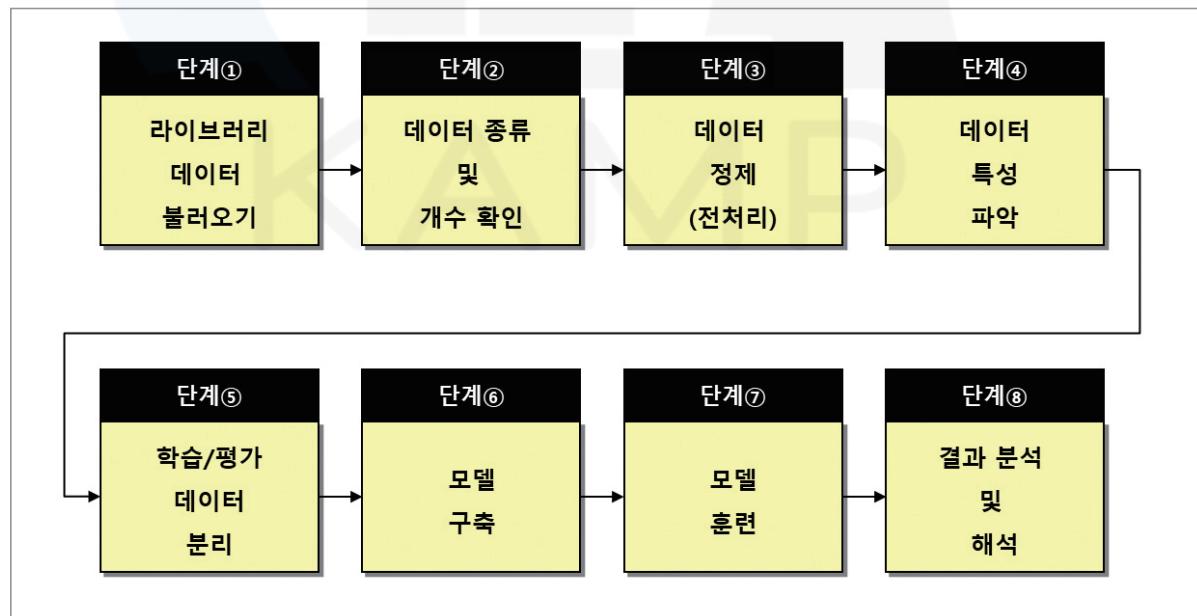
- F1 Score : 정밀도와 재현율의 조화평균을 이용하여 구한다. 값은 0~1 사이이며 높을 수록 좋다.

$$F1Score = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

- mAP(mean Average Precision) : 재현율 값들에 대응하는 정밀도 값들의 평균인 AP(Average Precision)를 객체(object)별로 구하고, 여러 객체 탐색된 결과에 대하여 평균(mean)값을 구한 것이 mAP이다.

2.3 분석 체험

1) 분석 단계별 Process



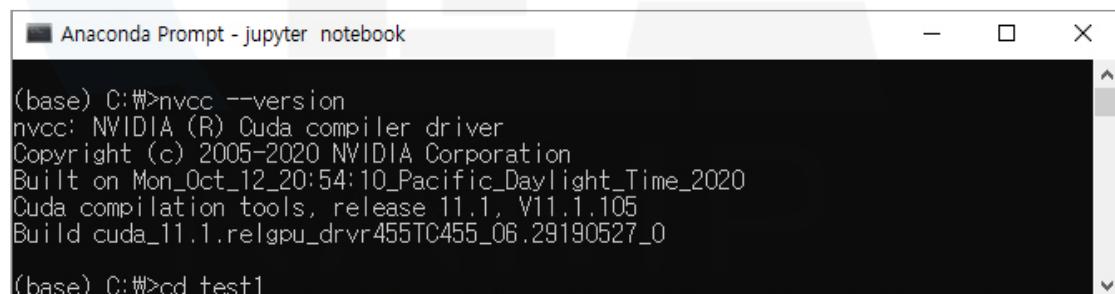
[그림 12] 분석 단계 Flow Chart

2) 분석 실습

[단계 ①] 라이브러리/데이터 불러오기

①-1. 라이브러리 설치 기본 사항 및 목록

- 필수 설치 SW : Anaconda, Python 3.7.3 등
- 필수 설치 라이브러리 : Numpy, Pandas, Os, Matplotlib, Graphviz, Sklearn, Tensorflow, Keras, Ipython, Pydotplus, Random, Subprocess, Sys, opencv-python, pytorch, pillow
- 분석에 필요한 데이터셋 : X선이물검출기 1호기, 2호기, 3호기에 대한 Defect 검출 데이터, BMP 파일 형식(.bmp)
- 실습을 위한 기본 필수 세팅
 “분석실습” 폴더 안에 있는 “test1” 폴더와 “OpenLabeling-master” 폴더를 로컬 디스크 (C:)로 옮긴다(사용자 지정 가능). 그리고 분석 실습을 위한 파이썬 주피터 노트북 파일 “yolov3_20201200.ipynb”는 “test1” 폴더의 “yolov3” 폴더 안으로 옮긴다. 그다음 Anaconda prompt를 실행 후 주피터 노트북을 실행하여 yolov3_20201200.ipynb 파일을 열어서 분석 실습을 진행할 수 있다. 이와 관련한 과정은 [1-2]에서 보다 자세하게 설명한다.
- GPU 활용을 위한 CUDA 설치 – 매뉴얼 하단 [부록] 참고
- CUDA의 설치가 완료되고 이를 확인하기 위해서는 Anaconda prompt창에 “nvcc-version”입력한다. 설치 여부와 CUDA 버전이 아래와 같이 나타난다.



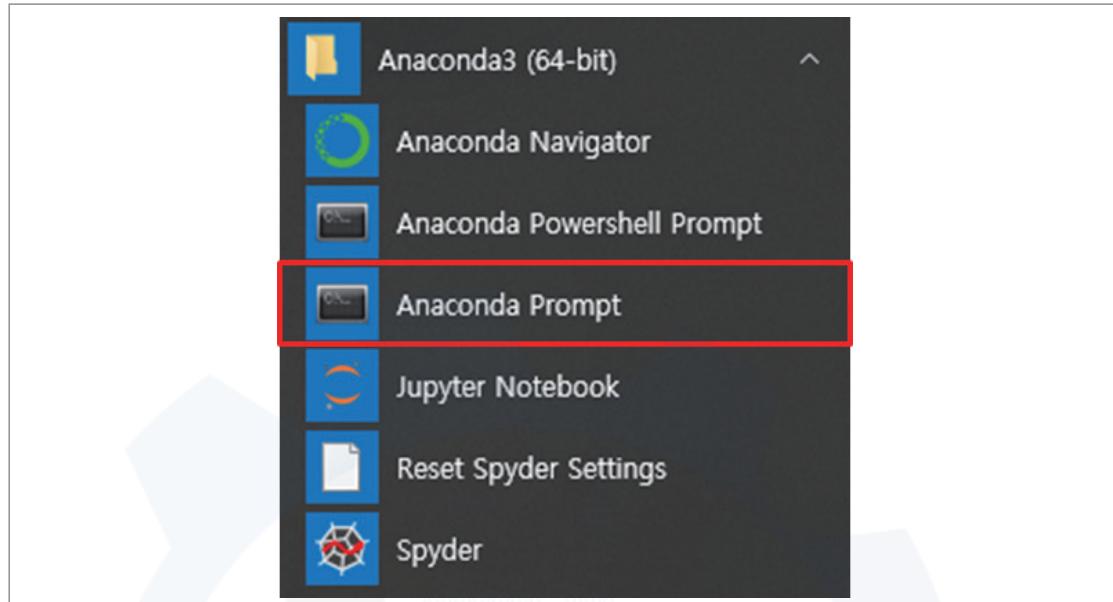
```
(base) C:\nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:54:10_Pacific_Daylight_Time_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.rhelgpu_drvr455TC455_06.29190527_0

(base) C:\cd test1
```

[그림 13] CUDA 설치 및 버전 확인

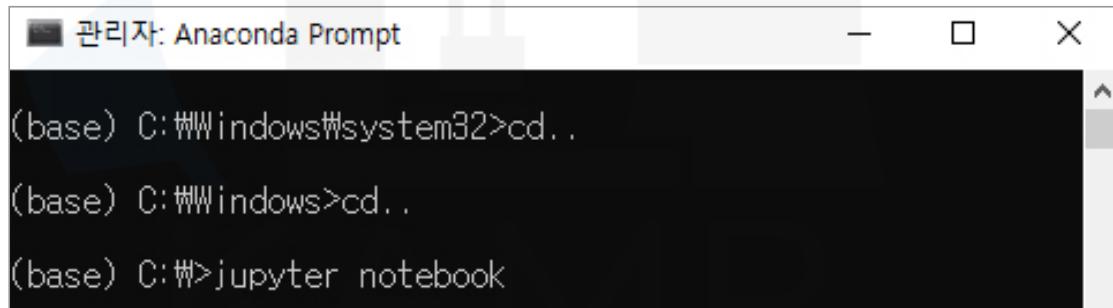
①-2. 필요 프로그램에 대한 기본 모듈 설치

- 기본 모듈 설치를 위해 Anaconda Prompt를 관리자 권한으로 실행한다. (마우스 우 클릭 > 관리자 권한으로 실행)



[그림 14] Anaconda Prompt 실행

- Anaconda Prompt를 실행 후 디렉토리 경로 설정을 위해 다음과 같이 입력하여 주피터 노트북을 실행한다.



```
(base) C:\Windows\system32>cd..
(base) C:\Windows>cd..
(base) C:\>jupyter notebook
```

[그림 15] Anaconda prompt 경로 설정 및 Jupyter notebook 실행 예시

- “분석실습” 폴더 안의 “test1” 폴더와 “OpenLabeling-master” 폴더를 “로컬 디스크 C: (C:\)”로 옮긴다.
- 주피터 노트북 창이 열리면 “test1” 폴더의 “yolov3” 폴더로 옮겨 놓은 “yolov3_20201200.ipynb” 파일을 더블클릭하여 실행한다. 이 파일에는 본 가이드북의 분석 실습 내용을 주피터 노트북에서 절차대로 진행하는 코드가 작성되어 있다.
- opencv-python:
본 분석을 위한 Labeling tool을 실행하기 위해 파이썬 모듈 opencv-python을 설치한다. Opencv는 이미지 프로세싱 및 영상 처리 등의 컴퓨터 비전을 수행하는 라이브러리이다.

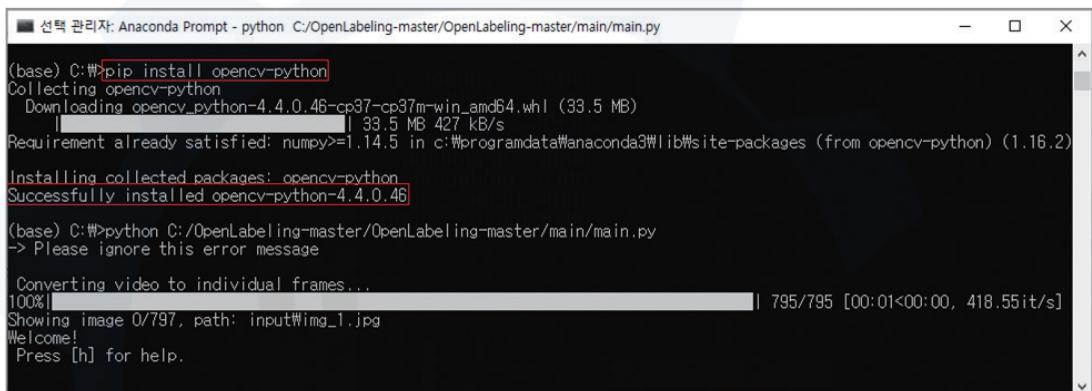
- 주피터 노트북에서는 다음의 코드를 실행하여 labeling tool을 설치한다.

```
# opencv-python 설치
!pip install opencv-python
```

[그림 16] Jupyter notebook에서 opencv를 설치하는 코드

- Anaconda prompt에서 수행 시에는 다음과 같은 순서로 진행한다.

- (1) Anaconda Prompt(또는 cmd)를 실행한다.
- (2) PIP(Python Package Index)를 통하여 파이썬으로 작성된 opencv 라이브러리 설치를 진행한다. 아래 이미지의 첫 번째 붉은색 박스로 표시된 부분을 그대로 입력 후 엔터를 누르면 labeling tool이 실행된다.
- (3) 설치완료 및 버전을 확인하고 설치를 종료한다.



```
선택 관리자: Anaconda Prompt - python C:/OpenLabeling-master/OpenLabeling-master/main/main.py
(base) C:\>!pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.4.0.46-cp37-cp37m-win_amd64.whl (33.5 MB)
    |████████| 33.5 MB 427 kB/s
Requirement already satisfied: numpy>=1.14.5 in c:\programdata\anaconda3\lib\site-packages (from opencv-python) (1.16.2)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.4.0.46
(base) C:\>python C:/OpenLabeling-master/OpenLabeling-master/main/main.py
-> Please ignore this error message

Converting video to individual frames...
100%[████████] 795/795 [00:01<00:00, 418.55it/s]
Showing image 0/797, path: input\img_1.jpg
Welcome!
Press [h] for help.
```

[그림 17] Anaconda Prompt에서 opencv를 설치하는 코드 및 결과

- PyTorch :

AI 모델 구현을 위해 파이썬 머신러닝 오픈소스 라이브러리인 PyTorch를 설치한다. 설치는 다음과 같은 순서로 진행한다.

- 주피터 노트북으로 수행 시에는 [그림 18]의 예시 코드 둘 중 하나를 실행하여 라이브러리 설치를 진행한다.

```
# pytorch 설치 (둘중 하나로 진행)
!conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch

# 위로 완료면 아래 방법으로
!pip install torch==1.7.0 torchvision==0.8.1 torchaudio==0.7.0 -f https://download.pytorch.org/wheel/torch_stable.html
```

[그림 18] Jupyter notebook에서 pytorch를 설치하는 코드

- Anaconda prompt로 수행 시에는 다음과 같이 진행한다.

- (1) Anaconda Prompt(또는 cmd)를 관리자 권한으로 실행한다.
- (2) [그림 19]의 코드 conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch를 그대로 입력 후 엔터를 누르면 설치가 진행된다.
혹은 다음의 코드를 입력해도 pytorch가 설치되며, [그림 20]과 같이 설치 결과가 화면

에 출력된다.

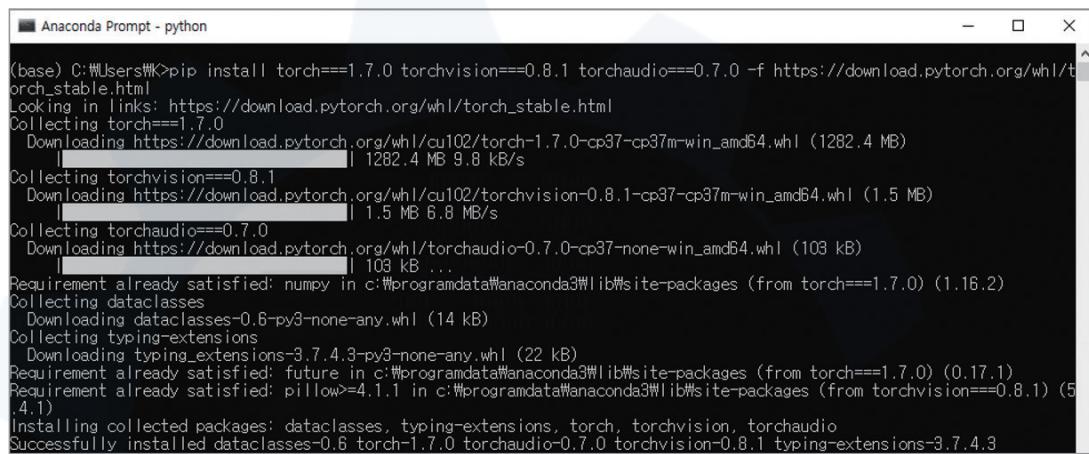
```
pip install torch==1.7.0 torchvision==0.8.1 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
pip install torch==1.7.0 torchvision==0.8.1 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
```

(3) 설치완료 및 버전을 확인하고 설치를 종료한다.



```
(base) C:\>conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch
```

[그림 19] PyTorch 설치 코드 예시1



```
(base) C:\Users\K>pip install torch==1.7.0 torchvision==0.8.1 torchaudio==0.7.0 -f https://download.pytorch.org/whl/torch_stable.html
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.7.0
  Downloading https://download.pytorch.org/whl/cu102/torch-1.7.0-cp37-cp37m-win_amd64.whl (1282.4 MB)
|██████████| 1282.4 MB 9.8 kB/s
Collecting torchvision==0.8.1
  Downloading https://download.pytorch.org/whl/cu102/torchvision-0.8.1-cp37-cp37m-win_amd64.whl (1.5 MB)
|██████████| 1.5 MB 6.8 MB/s
Collecting torchaudio==0.7.0
  Downloading https://download.pytorch.org/whl/torchaudio-0.7.0-cp37-none-win_amd64.whl (108 kB)
|██████████| 108 kB ...
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from torch==1.7.0) (1.16.2)
Collecting dataclasses
  Downloading dataclasses-0.6-py3-none-any.whl (14 kB)
Collecting typing_extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Requirement already satisfied: future in c:\programdata\anaconda3\lib\site-packages (from torch==1.7.0) (0.17.1)
Requirement already satisfied: pillow==4.1.1 in c:\programdata\anaconda3\lib\site-packages (from torchvision==0.8.1) (5.4.1)
Installing collected packages: dataclasses, typing_extensions, torch, torchvision, torchaudio
Successfully installed dataclasses-0.6 torch-1.7.0 torchaudio-0.7.0 torchvision-0.8.1 typing_extensions-3.7.4.3
```

[그림 20] PyTorch 설치 코드 예시2 실행 결과

- PyTorch 설치 및 CUDA 확인은 다음과 같다.

```
>>> import torch
>>> torch.cuda.is_available()
True
>>>
```

[그림 21] PyTorch 및 CUDA 설치 확인 코드 및 결과

- 이미지 확인 및 출력을 위한 pillow 라이브러리는 다음의 코드로 설치한다.

```
# pillow 설치
!pip install pillow
```

[그림 22] Jupyter notebook에서 pillow를 설치하는 코드

[단계 ②] 데이터 종류 및 개수 확인

②-1. 기존 데이터셋 확인

- 본 분석에서 사용하는 데이터셋은 “X선이물검출기(06.23_09.22)” 폴더의 설비 호기별 결함(Defect) 검출 데이터 이미지(BMP) 파일이다.²
- 주피터 노트북에서 원본 데이터 확인은 다음과 같이 진행한다. 먼저 [그림 23]과 같이 필요 라이브러리를 불러오는 코드를 실행한 후, image.open() 함수의 괄호 안에 이미지의 경로와 파일명을 입력하여 실행하면 해당 파일이 주피터 노트북의 결과 셀에 출력된다.

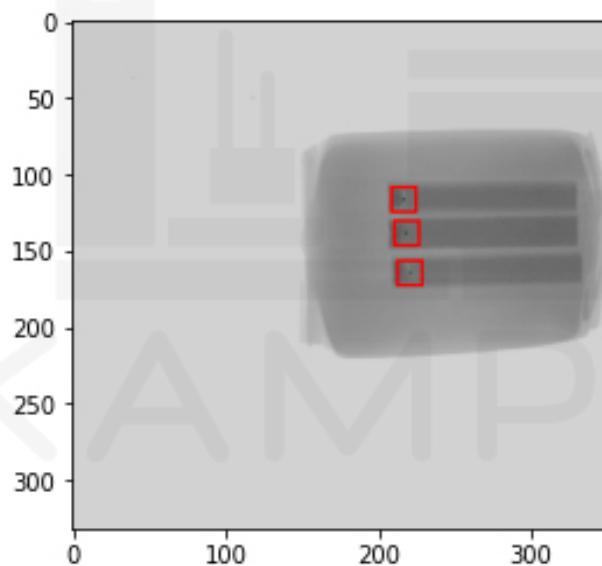
```
from PIL import Image
import matplotlib.pyplot as plt
```

[그림 23] Jupyter notebook에서 필요 라이브러리를 불러오는 코드

```
# 원본 데이터 이미지 확인
i = Image.open('C:/test1/yolov3/X선이물검출기(06.23_09.22)/1호기(2020.09.22)/SN77128_20200622_NetImage/002_20200622_203053(2).bmp')

plt.imshow(i)
plt.show()
```

[그림 24] Jupyter notebook에서 원본 이미지를 출력하여 확인하는 코드



[그림 25] Jupyter notebook에서 원본 이미지를 출력하여 확인한 결과

² “1호기” 폴더- 1,031개 파일/112MB, “2호기” 폴더- 920개 파일/92.9MB, “3호기” 폴더- 858개 파일/210MB = 총 2,809개 415MB 사이즈의 데이터셋

②-2. 데이터셋 재정리

```
# Raw Image에서 재정리합니다
import os
import shutil

image_dir = "C:/test1/yolov3/X선이물검출기(06.23_09.22)" # img_dir 저장될 위치에 맞
file= os.listdir(image_dir)

os.mkdir("C:/test1/yolov3/X선이물검출기")

for i in range(len(file)):
    os.mkdir("C:/test1/yolov3/X선이물검출기" + "/" + str(i+1) + "호기") # 저장할 주소

for a in file:
    path = image_dir + "/" + a
    file2 = os.listdir(path)
    for b in file2:
        if(b[17:]=="NgImage"):
            path2 = path + "/" + b
            file3 = os.listdir(path2)
            for c in file3:
                shutil.copy(path2 + "/" + c, "C:/test1/yolov3/X선이물검출기" + "/" + a[0] + "호기")
```

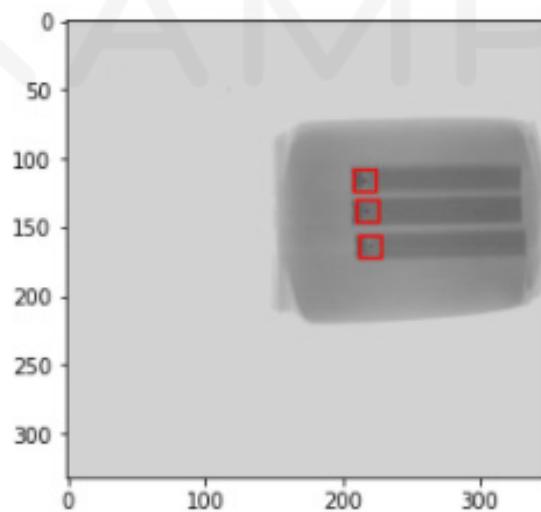
[그림 26] Jupyter notebook에서 원본 데이터를 재정리하는 코드

- 다음은 [그림 26]의 코드를 통해 원본 데이터를 재정리하여 설비 호기별로 이미지 파일을 정리한다.
- 재정리한 이미지는 다음과 같이 확인한다.

```
im = Image.open('C:/test1/yolov3/X선이물검출기/1호기/002_20200622_203053(2).bmp')

plt.imshow(im)
plt.show()
```

[그림 27] Jupyter notebook에서 재정리된 이미지를 출력하여 확인하는 코드



[그림 28] Jupyter notebook에서 재정리된 이미지를 출력하여 확인한 결과

- 다음은 [그림 26]에서 사용한 코드를 단계별로 상세히 설명하고자 한다. 먼저 데이터셋을 재정리하는데 필요한 라이브러리를 import를 통해 불러온다.

```
import os
import shutil
```

[그림 29] 필요 라이브러리를 불러오기 위한 코드

- 그다음 분석에 활용될 이미지 데이터셋이 저장되어 있는 로컬 디렉토리를 지정한다.

```
image_dir = "C:/test1/yolov3/X선이물검출기(06.23_09.22)"
file = os.listdir(image_dir)
```

[그림 30] 로컬 디렉토리를 지정하기 위한 코드

- 파일 내부의 호기별(1~3호기) 결함 검출 기록과 관련된 정보가 있는 개별 폴더에 접속한다.

```
for i in range(len(file)):
    os.mkdir("C:/test1/yolov3/X선이물검출기" + "/" + str(i+1) + "호기")
```

[그림 31] 원본 이미지 폴더에 접속하는 코드

- 호기별 폴더 내에서 결함 검출 이미지가 담겨있는 폴더로 진입하여 각 이미지를 재정리한다.

```
for a in file:
    path = image_dir + "/" + a
    file2 = os.listdir(path)
    for b in file2:
        if(b[17:]=="NgImage"):
            path2 = path + "/" + b
            file3 = os.listdir(path2)
            for c in file3:
                shutil.copy(path2 + "/" + c, "C:/test1/yolov3/X선이물검출기" + "/" + a[0] + "호기")
```

[그림 32] 원본 이미지를 호기별로 재정리하는 코드

- 위의 코드를 실행하여 기존 X-RAY 검사설비에서 추출된 데이터셋을 분석에 용이한 데이터셋으로 재정리한다. 각 폴더에 개별적으로 분류된 이미지들을 설비 호기별로 분류한 결과는 아래와 같다.



[그림 33] 설비 호기별로 분류 및 재정리한 데이터셋

- Anaconda Prompt에서 데이터셋 재정리 수행 시에는 다음과 같이 진행한다.



```
Anaconda Prompt  
(base) C:\test1\yolov3\python img.py
```

[그림 34] 데이터셋 재정리 파이썬 파일 실행 코드

- YOLOv3 모델은 객체를 탐지하는데 많이 활용된다. 본 분석 실습에서는 X-RAY 검사장비에서 검출된 결함을 객체로 하여 탐지하는 데 목적을 두며, 이를 위해서는 모델에 결함이 존재하는 이미지를 학습시키는 것이 필요하다. 학습을 위해서는 결함이 있는 원본 이미지와 모델이 결함 부분을 파악하여 학습할 수 있도록 결함 부분에 대한 좌표가 필요하다. 이를 위한 과정이 아래 [단계 ③] 데이터 정제(전처리) 과정이며, 이 과정을 거치면 그 결과([그림 44] 참조)로 좌표가 담긴 “라벨 텍스트 파일”이 출력된다.
- [단계 ③]부터 이어지는 분석 실습의 주 내용은 15개의 이미지에 대한 라벨링, 모델 훈련 및 탐지, 모델평가를 진행하지만, 더욱 다양한 실습을 경험하고자 하는 사용자들을 위해 본 가이드북은 결함이 있는 이미지 500개에 대하여 미리 라벨링 작업을 수행한 결과물(이미지 및 좌표 텍스트 파일)을 함께 제공한다. 이 자료는 “라벨링 6 종 세트” 폴더에서 찾아볼 수 있으며, 라벨링 개수에 차등을 주어 폴더별로 구분(15개, 50개, 100개, 200개, 300개, 400개)되어 있어, 더욱 다양한 분석실습을 수행할 수 있도록 하였다.
- 이와 같이 제공되는 이미지 개수별 라벨링 결과물 6종은 [단계 ⑧] 결과 분석 및 해석 과정에서 모델을 각각 학습하여 6개의 다른 결과를 도출하고, 이를 비교하여 어느 정도의 라벨링을 수행하는 것이 합리적인 의사결정인지를 판단하는데 사용할 수 있다.
- 또한, [단계 ③]의 라벨링 작업을 진행하지 않고, 바로 모델 학습을 실습하고자 하는 사용자들은 “라벨링 6종 세트”的 자료를 활용하여 바로 [단계 ④] 데이터 특성 파악 과정으로 진행할 수 있다.

[단계 ③] 데이터 정제(전처리)

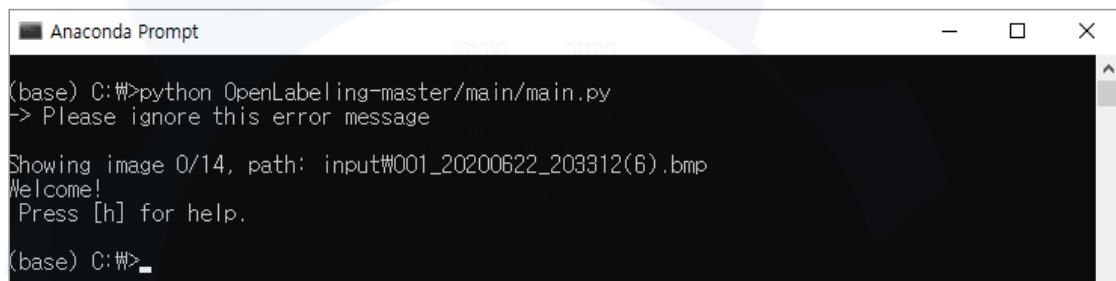
③-1. 결함 데이터 좌표 입력을 위한 Labeling 실행

- 주피터 노트북에서 실행하는 Labeling tool 코드는 다음과 같다. 먼저 로컬 디스크 C 드라이브에(혹은 지정한 경로에) Labeling tool 관련 폴더를 옮겨 놓아야 하며, [OpenLabeling-master > main > input] 폴더에 라벨링 작업을 할 이미지들을 옮겨 놓아야 한다.

```
# labeling tool을 실행합니다.
# (로컬디스크 C에 Labeling tool 폴더를 옮겨놓으셔야 합니다.)
# "C:/OpenLabeling-master/main/input" 경로에 라벨링할 이미지들을 옮겨 놓습니다.
!python C:/OpenLabeling-master/main/main.py
```

[그림 35] Jupyter notebook에서 Labeling tool을 실행하는 코드

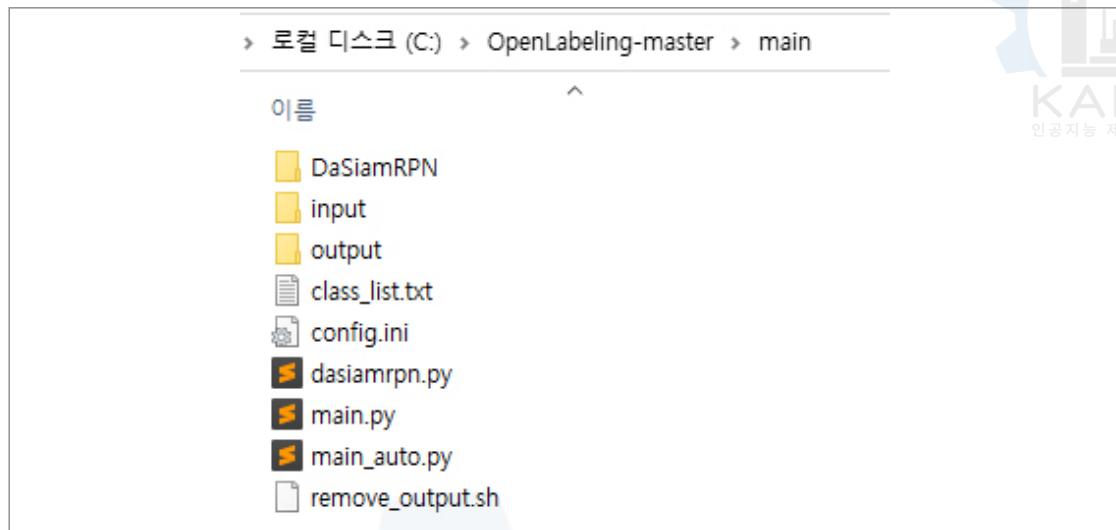
- Labeling tool의 결과물인 좌표 텍스트 파일은 [OpenLabeling-master > main > output > YOLO_darknet] 경로에 저장된다.
- Anaconda prompt에서 수행 시에는 다음과 같이 진행한다. Labeling tool이 저장된 디렉토리에 접근하여 실행한다. Anaconda Prompt에서 아래 볼드체 그대로 입력 후 엔터를 누르면 labeling tool이 실행된다.
Anaconda Prompt (or cmd) (base) C:\> python C:/OpenLabeling-master/main/main.py



[그림 36] Anaconda Prompt에서 Labeling tool을 실행하는 코드

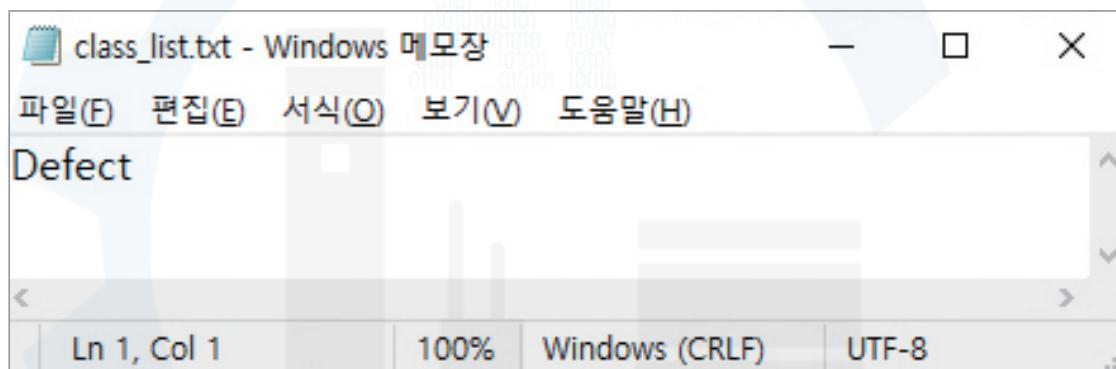
③-2. 학습 데이터셋을 위한 결함 이미지 라벨링 작업 수행

- 다음의 (1)~(4) 순서로 결함(defect)이 표기된 검출 이미지 데이터에 라벨(label) 좌표 값 저장 작업을 진행한다. 라벨의 종류와 4가지 좌표(x, y, w, h)는 하나의 텍스트 파일에 저장되어 있다.
 - (1) 파이썬 프로그램 다운로드 및 설치 (1-1. 설치 시 추가 필요 없음) (python3 -V, version 3 Python version 3 runtime interpreter)
 - (2) Local에 라벨 작업을 진행할 이미지 저장
 X선이물검출기(06.23_09.22, 1~3호기)
 - (3) (Openlabeling).py 소스코드 다운로드 및 파일 실행 데이터셋과 AI 모델 학습을 위한 Labeling Tool 다운로드
 다운로드 경로: <https://github.com/Cartucho/OpenLabeling>
 - (4) 로컬 이미지에 대한 라벨링 작업 진행
- 프로그램 로컬 경로를 설정한다(편의상 로컬 디스크 (C:)에 저장). 파일 다운로드 및 압축 해제 후, 원하는 로컬 디렉토리로 폴더를 이동한다(OpenLabeling-master > main 폴더).



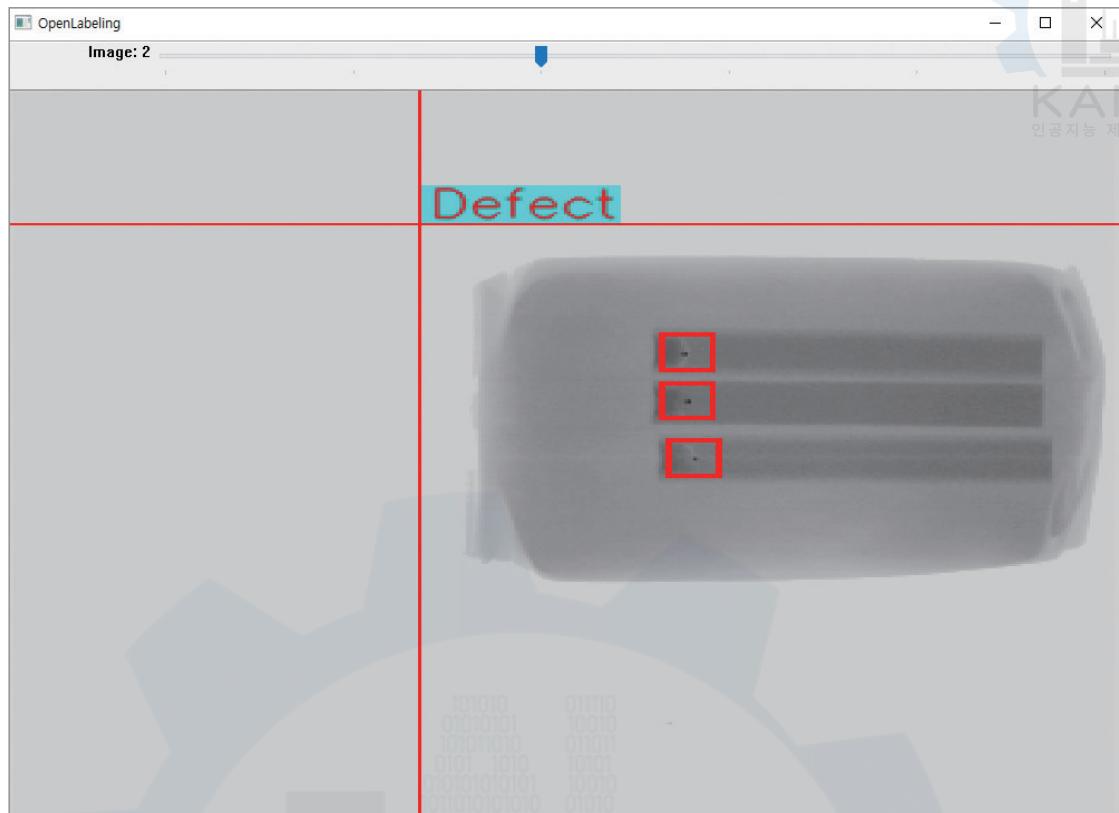
[그림 37] Labeling tool의 main 폴더

- class_list.txt 파일은 라벨링을 하고자 하는 클래스의 이름을 지정한다. 각 행이 클래스들을 의미하므로 다중 클래스의 경우 행으로 구분한다.



[그림 38] class_list.txt 파일

- Labeling tool 프로그램 실행 화면은 다음과 같다.

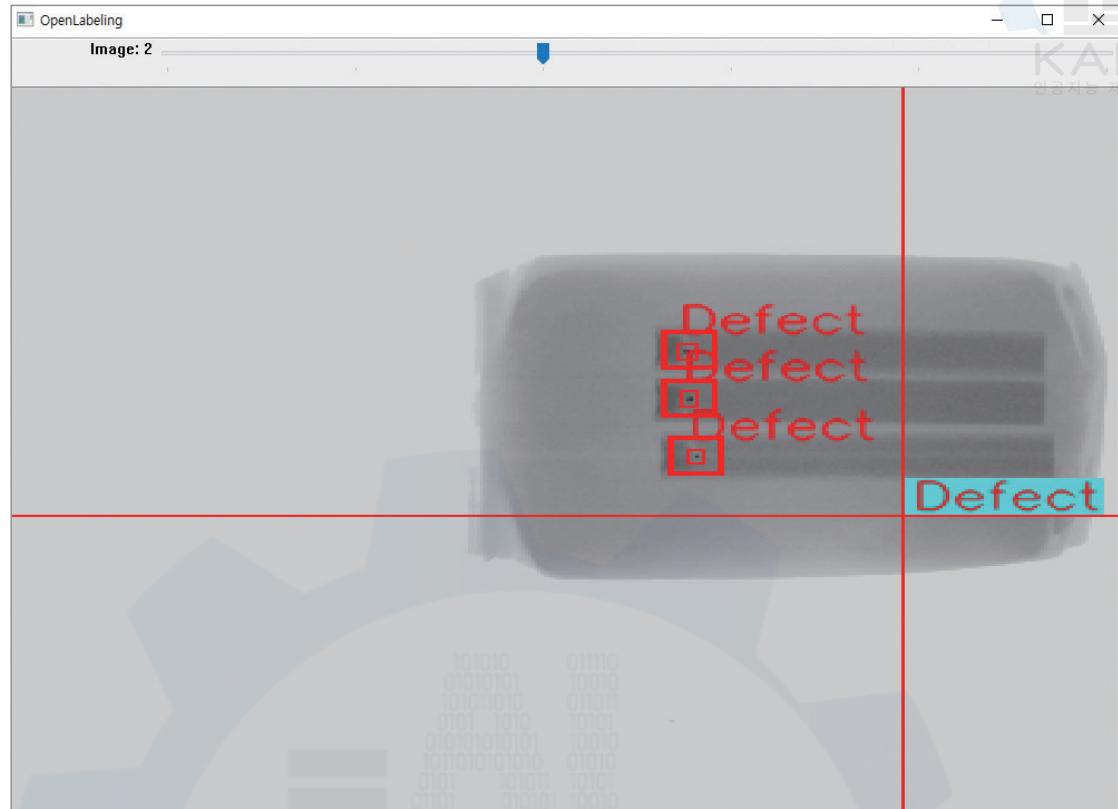


[그림 39] Labeling tool 프로그램 실행 화면

| 컨트롤 키 | | 기본 동작 내용 |
|-----------|--|----------------------------|
| D | | 다음 그림으로 이동한다 |
| A | | 이전 그림으로 이동한다 |
| S | | 다른 클래스로 변경한다 (다른 클래스로 라벨링) |
| Q | | 프로그램을 종료한다 |
| (마우스 우클릭) | | 라벨링을 제거한다 |

- Labeling tool로 이미지의 모든 결함(defect)에 라벨링을 수행한 결과는 다음과 같

다.



[그림 40] 이미지의 모든 결함(defect)에 라벨링을 수행한 결과

- Labeling tool로 이미지 라벨링 작업 시에 Anaconda Prompt에서 확인되는 결과는 다음과 같다. 라벨링 작업을 하고있는 데이터의 이름과 경로가 표기된다. (0~14로 총 15개의 이미지)

```
■ Anaconda Prompt - python C:/OpenLabeling-master/main/main.py
(base) C:\#>python C:/OpenLabeling-master/main/main.py
-> Please ignore this error message

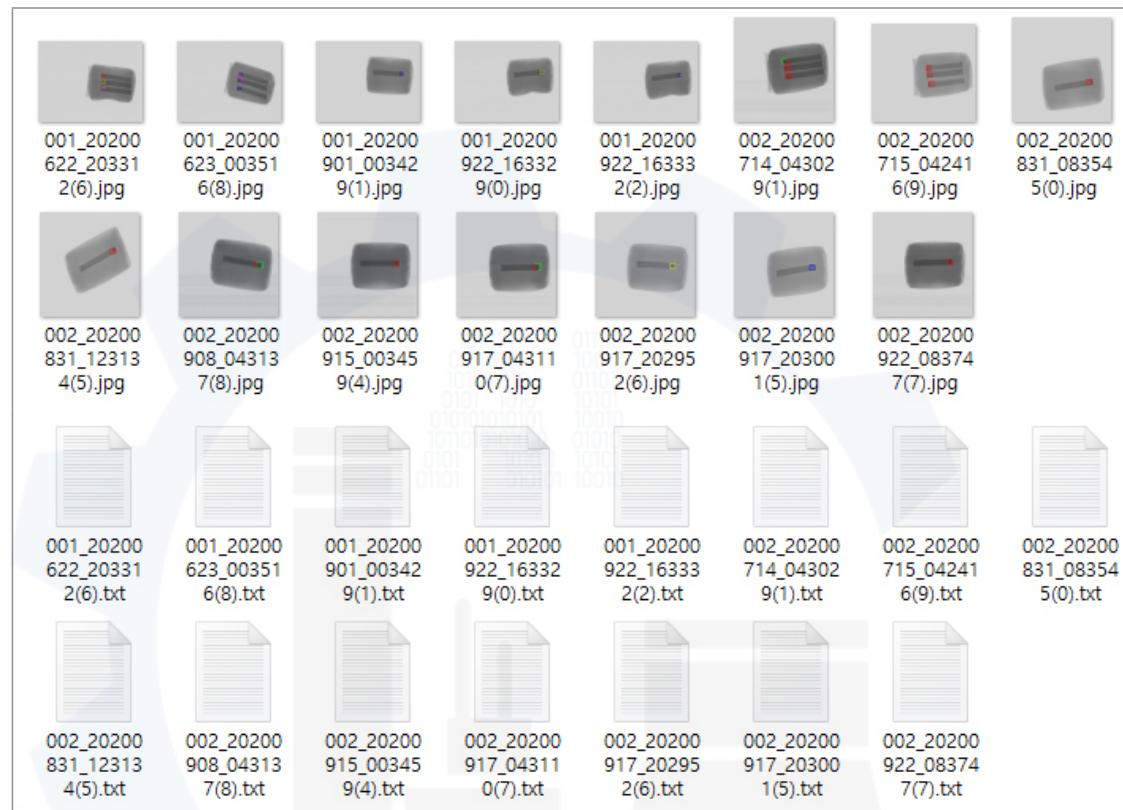
Showing image 0/14, path: input#001_20200622_203312(6).bmp
Welcome!
Press [h] for help.
Showing image 1/14, path: input#001_20200623_003516(8).bmp
Showing image 1/14, path: input#001_20200623_003516(8).bmp
Showing image 2/14, path: input#001_20200901_003429(1).bmp
Showing image 2/14, path: input#001_20200901_003429(1).bmp
Showing image 3/14, path: input#001_20200922_163329(0).bmp
Showing image 3/14, path: input#001_20200922_163329(0).bmp
Showing image 4/14, path: input#001_20200922_163332(2).bmp
Showing image 4/14, path: input#001_20200922_163332(2).bmp
Showing image 4/14, path: input#001_20200922_163332(2).bmp
Showing image 5/14, path: input#002_20200714_043029(1).bmp
Showing image 5/14, path: input#002_20200714_043029(1).bmp
Showing image 6/14, path: input#002_20200715_042416(9).bmp
Showing image 6/14, path: input#002_20200715_042416(9).bmp
Showing image 7/14, path: input#002_20200831_083545(0).bmp
Showing image 7/14, path: input#002_20200831_083545(0).bmp
Showing image 8/14, path: input#002_20200831_123134(5).bmp
Showing image 8/14, path: input#002_20200831_123134(5).bmp
Showing image 9/14, path: input#002_20200908_043137(8).bmp
Showing image 9/14, path: input#002_20200908_043137(8).bmp
Showing image 10/14, path: input#002_20200915_003459(4).bmp
Showing image 10/14, path: input#002_20200915_003459(4).bmp
Showing image 11/14, path: input#002_20200917_043110(7).bmp
Showing image 11/14, path: input#002_20200917_043110(7).bmp
Showing image 12/14, path: input#002_20200917_202952(6).bmp
Showing image 12/14, path: input#002_20200917_202952(6).bmp
Showing image 13/14, path: input#002_20200917_203001(5).bmp
Showing image 13/14, path: input#002_20200917_203001(5).bmp
Showing image 14/14, path: input#002_20200922_083747(7).bmp
Showing image 14/14, path: input#002_20200922_083747(7).bmp
Showing image 0/14, path: input#001_20200622_203312(6).bmp
Showing image 0/14, path: input#001_20200622_203312(6).bmp
```

[그림 41] 라벨링 작업 시 Anaconda Prompt에서 확인되는 결과 예시

[단계 ④] 데이터 특성 파악

④-1. 라벨링된 이미지 데이터 확인 및 라벨 관련 좌표 확인

- 라벨링 작업을 수행한 후 기존 이미지 파일과 라벨링 작업을 통해 구한 좌표 텍스트 파일은 다음과 같다. 상단은 원본 이미지의 일부로 [OpenLabeling-master > main > input] 폴더에 위치하고, 하단은 라벨링 좌표(x, y, h, w)가 포함된 텍스트 파일로 [OpenLabeling-master > main > output > YOLO_darknet] 폴더에 위치한다.

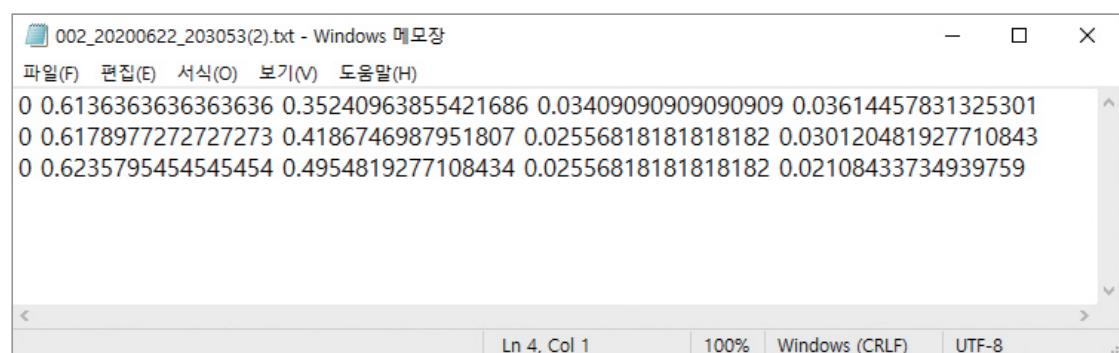


[그림 42] 라벨링 작업 수행 후의 원본 이미지와 라벨링 좌표가 포함된 텍스트 파일

- 결과 저장 디렉토리에서 라벨링 좌표 값을 확인한다.

C:\OpenLabeling-master\main\output\YOLO_darknet

Label, x, y, w, h 좌표의 예시는 다음과 같다.



```

002_20200622_203053(2).txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
0 0.6136363636363636 0.35240963855421686 0.03409090909090909 0.03614457831325301
0 0.6178977272727273 0.4186746987951807 0.025568181818182 0.030120481927710843
0 0.6235795454545454 0.4954819277108434 0.025568181818182 0.02108433734939759
  
```

[그림 43] 라벨링 좌표가 포함된 텍스트 파일 예시

④-2. 라벨링 결과물 이동 및 확장자 변환하기

- Labeling tool로 작업한 결과물인 이미지와 텍스트 파일(좌표)은 test1 > yolov3 폴더 내 “images”와 “labels” 폴더로 각각 옮긴다.
(이미지는 라벨링 작업을 한 이미지로, 원본과는 차이는 없다.)



[그림 44] 이미지와 텍스트로 구분된 폴더

- images 폴더로 이동한 이미지들은 현재 bmp 형식으로 되어있는데, 이를 모두 jpg 형식으로 변환시키는 작업을 실행한다. 이는 [단계 ⑤]에서 splitdata.py를 통해 train set과 test set을 나눌 때 지원하는 확장자가 jpg 파일이기 때문이다.
- 주피터 노트북에서 실행하는 코드는 다음과 같다. 우선 해당 데이터 파일을 [test1 > yolov3 > images] 폴더로 이동한 후, 파일 이름을 변경해주는 ren 명령어를 실행한다. 이때 파일 확장자 형식을 jpg로 모두 변환시킨다.

```
# 디렉토리 경로 설정
%cd C:/test1/yolov3/images
!ren *.* *.jpg*
```

[그림 45] Jupyter notebook에서 경로 및 이미지 확장자를 설정하는 코드

```
C:#test1#yolov3#images
```

[그림 46] Jupyter notebook에서 경로를 설정한 결과

- 다음은 위와 같이 [test1 > yolov3] 폴더로 옮겨지고 변환된 이미지 파일과 라벨 텍스트 파일로 모델 학습 및 검증을 수행한다.

[단계 ⑤] 학습/평가 데이터 분리

⑤-1. 학습 및 평가 데이터 구분

- 모델의 학습 및 평가를 위해서는 데이터셋을 분리해야 한다. 본 분석에서는 폴더 안에 포함되어 있는 이미지 데이터셋을 학습 데이터셋과 평가 데이터셋으로 구분하기 위해 각각의 고유한 이름을 텍스트 파일로 불러온 뒤 구분하여 모델로 불러와 학습 및 평가를 수행하도록 한다.
- 주피터 노트북으로 수행 시에는 먼저 경로를 다시 지정한다.

```
%cd C:/test1/yolov3
```

[그림 47] Jupyter notebook에서 경로를 재지정하기 위한 코드

```
C:#test1#yolov3
```

[그림 48] Jupyter notebook에서 경로를 재지정한 결과

- 그다음은 필요 라이브러리를 불러온 후, 데이터를 8:2로 분리하는 함수를 정의한다.

```
import random
import os
import subprocess
import sys
```

[그림 49] Jupyter notebook에서 필요 라이브러리를 불러오는 코드

```
def split_data_set(image_dir):
    f_val = open("test.txt", 'w')
    f_train = open("train.txt", 'w')

    path, dirs, files = next(os.walk(image_dir))
    data_size = len(files)

    ind = 0
    data_test_size = int(0.2 * data_size)
    test_array = random.sample(range(data_size), k=data_test_size)

    for f in os.listdir(image_dir):
        if(f.split(".jp")[1] == "g"):
            ind += 1

            if ind in test_array:
                f_val.write(image_dir+'/'+f+'\n')
            else:
                f_train.write(image_dir+'/'+f+'\n')
```

[그림 50] Jupyter notebook에서 학습/평가 데이터 분리 함수를 정의하는 코드

- 위와 같이 함수를 정의한 후, 아래와 같이 이미지 디렉토리를 저장하고, 위에서 정의한 학습/평가데이터 분리 함수에 디렉토리 경로를 입력하여 실행한다. 이미지 데이터를 나눈 결과는 텍스트 파일로 저장된다. 출력된 결과는 현재 디렉토리인 "C:/test1/yolov3/"에 "test.txt"와 "train.txt"로 생성되어 있다. 코드에 대한 보다 더 자세한 설명은 아래 Anaconda prompt 수행 예시 다음에 기재되어 있다.

```
image_dir="C:/test1/yolov3/images"
```

[그림 51] Jupyter notebook에서 이미지 경로를 지정하는 코드

```
split_data_set(image_dir)
```

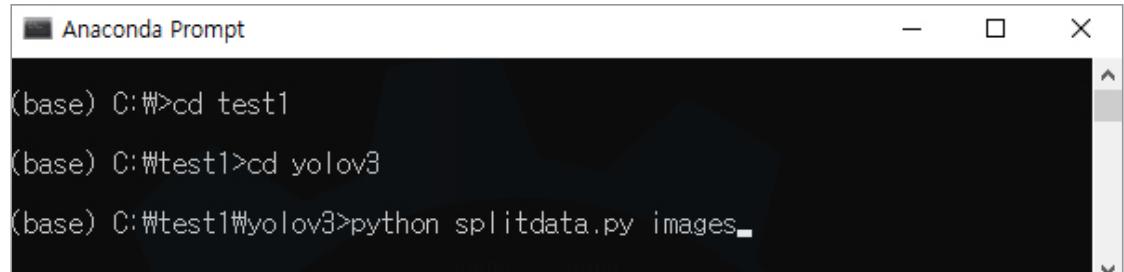
[그림 52] Jupyter notebook에서 학습/평가 데이터 분리 함수를 실행하는 코드

- 학습/평가 데이터 분리 작업은 아래와 같이 실행해도 되며, 결과는 위에서 수행한 방법과 동일하다.

```
!python splitdata.py images
```

[그림 53] Jupyter notebook에서 학습/평가 데이터 분리를 실행하는 다른 방법

- Anaconda prompt에서 수행 시에는 다음과 같이 진행한다.



```
Anaconda Prompt
(base) C:\>cd test1
(base) C:\test1>cd yolov3
(base) C:\test1\yolov3>python splitdata.py images
```

[그림 54] 데이터셋 분리를 위한 파이썬 파일 실행 코드

- [그림 49, 50]의 주피터 노트북 코드에 대한 더 자세한 설명은 다음과 같다. 먼저 학습/평가 데이터셋을 분리하기 위해 필요한 라이브러리를 import를 통해 불러온다.

```
import random
import os
import subprocess
import sys
```

[그림 55] 필요 라이브러리를 불러오기 위한 코드

- 분석에 활용될 이미지 데이터셋이 저장되어 있는 로컬 디렉토리를 지정한다.

```
def split_data_set(image_dir):
    f_val = open("test.txt", 'w')
    f_train = open("train.txt", 'w')

    path, dirs, files = next(os.walk(image_dir))
    data_size = len(files)
```

[그림 56] 학습 평가 데이터셋 로컬 디렉토리 지정 코드

- 파일을 만들고 경로를 받아 무작위로 이미지를 나누고, 훈련 데이터셋과 평가 데이터셋을 8:2의 비율로 나눈다. 아래 코드에서 int(0.2) 부분은 20%를 평가 데이터셋으로 구분한다는 의미이다.

```

ind = 0
data_test_size = int(0.2 * data_size)
test_array = random.sample(range(data_size), k=data_test_size)
  
```

[그림 57] 학습 평가 데이터셋 구분 코드

- 다음 코드에서는 “.jp” 형식의 이미지를 읽어 각각의 이미지 위치를 입력하고 각각의 데이터셋을 저장한다.

```

for f in os.listdir(image_dir):
    if(f.split(".jp")[1] == "g"):
        ind += 1

    if ind in test_array:
        f_val.write(image_dir+'/'+f+'\n')
    else:
        f_train.write(image_dir+'/'+f+'\n')
  
```

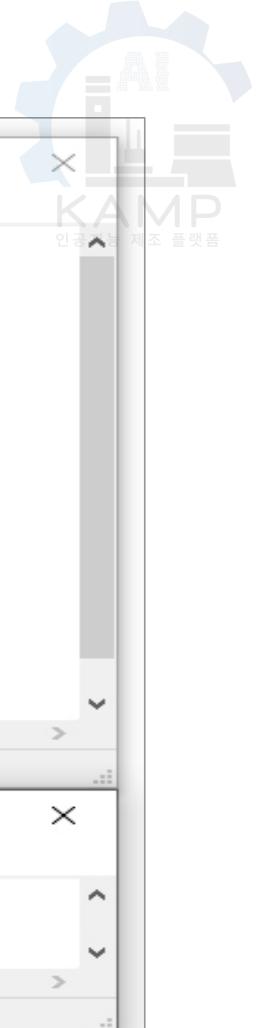
[그림 58] 이미지를 불러오는 코드

- 함수 정의 후, 다음의 코드를 실행하여 학습 및 평가 데이터셋이 분리된 것을 확인하면 “train.txt”와 “test.txt”로 나누어져 있다. (image_dir은 이미지 경로가 저장된 변수이다. [그림 51] 참조.)

```
split_data_set(image_dir)
```

[그림 59] 데이터 분리 실행 코드

- 모델에서는 이미지를 불러와 학습 및 평가를 수행해야 하므로 분리된 이미지에 대하여 결과를 저장해야 한다. 이러한 결과는 텍스트 파일(test.txt, train.txt)로 저장하며 다음과 같다.



[그림 60] 학습 및 평가용으로 구분된 이미지 파일명 예시

- 이렇게 구분된 이미지들의 이름을 모델이 각각 불러오며, 각각의 이미지와 좌표가 포함된 텍스트 파일을 각각의 데이터셋으로 학습 및 평가를 진행한다.

[단계 ⑥] 모델 구축

⑥-1. 모델 학습 수행을 위한 준비

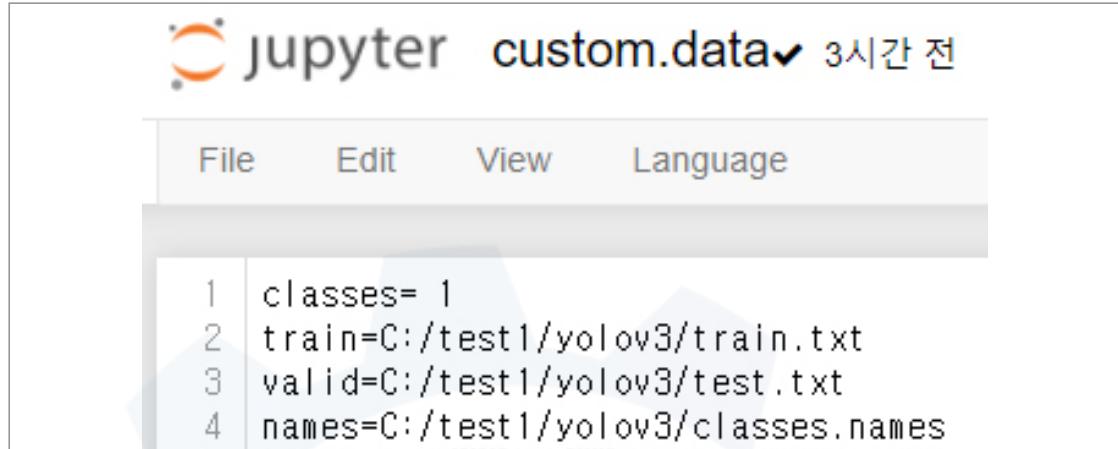
- 객체탐지를 위한 YOLOv3 학습 전에 다음 두 파일을 수정하여 데이터셋의 위치 설정과 클래스 개수 및 이름 설정을 진행해야 한다.

| | | |
|---------------|----------|-----|
| classes.names | NAMES 파일 | 1KB |
| custom.data | DATA 파일 | 1KB |

[그림 61] 모델 학습 설정을 위해 수정이 필요한 파일

- 주피터 노트북에서는 노트북 실행 시에 나타나는 디렉토리에서 “custom.data”와 “classes.names”를 각각 클릭하여 열어서 수정하면 된다.

- 다른 간단한 방법으로는 메모장으로 열기가 있지만 파일의 내용이 깨지는 경우가 가끔 발생한다.
- 주피터 노트북의 디렉토리에서 “custom.data”를 열어서 다음과 같이 내용을 수정 한다. 경로 정보는 yolov3가 위치하는 경로로 입력한다.



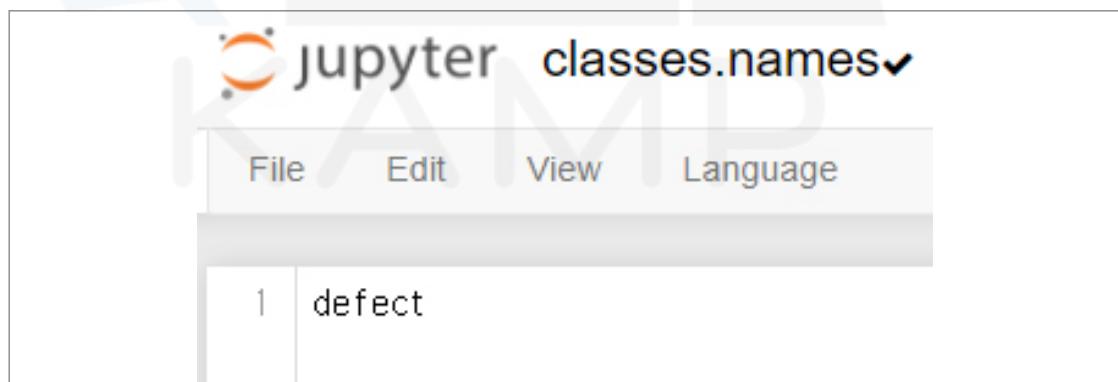
```

jupyter custom.data ✓ 3시간 전
File Edit View Language
1 classes= 1
2 train=C:/test1/yolov3/train.txt
3 valid=C:/test1/yolov3/test.txt
4 names=C:/test1/yolov3/classes.names

```

[그림 62] Jupyter Notebook에서 custom.data 파일 수정 예시

- 클래스 개수 설정
 - 학습 데이터셋 위치 설정
 - 평가(검증) 데이터셋 위치 설정
 - 클래스 name 위치 설정
- 검출할 결함(defect)에 대한 클래스 이름은 “classes.names” 파일에서 지정한다. 이는 YOLOv3 모델에서 객체탐지를 수행하여 결함을 구별할 때 구분 지어지는 이름으로, 본 분석에서는 결함의 종류가 하나이므로 하나의 클래스 이름만 지정한다.



```

jupyter classes.names ✓
File Edit View Language
1 defect

```

[그림 63] Jupyter Notebook에서 classes.names 파일 수정 예시

* 클래스 설정: 결과 시각화 시에 나타나는 객체 이름(사용자 지정)

- 다음은 YOLOv3 모델 구축을 위한 추가적인 모듈들을 한 번에 통합 설치하기 위해 주피터 노트북에서 다음 코드를 실행한다.

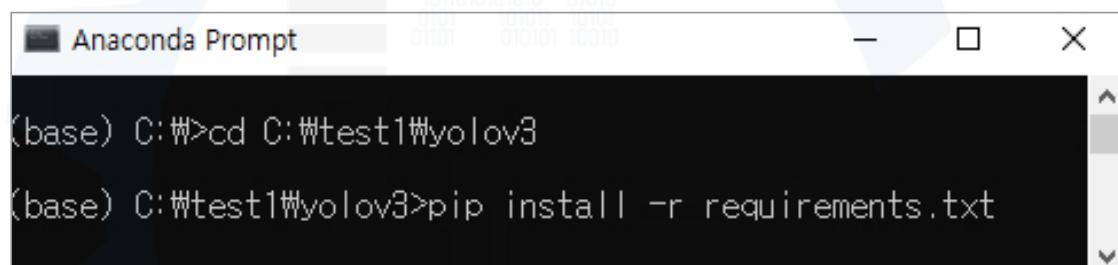
```
!pip install -r requirements.txt
```

[그림 64] Jupyter notebook에서 추가적인 모듈을 통합 설치하는 코드

```
Requirement already satisfied: Cython in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 4)) (0.29.21)
Requirement already satisfied: matplotlib>=3.2.2 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 5)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 6)) (1.19.2)
Requirement already satisfied: opencv-python>=4.1.2 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 7)) (4.4.0.46)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 8)) (8.0.1)
Requirement already satisfied: PyYAML>=5.3 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 9)) (5.3.1)
Requirement already satisfied: scipy>=1.4.1 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 10)) (1.5.2)
Requirement already satisfied: tensorflow>=2.2 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 11)) (2.4.0)
Requirement already satisfied: torch>=1.6.0 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 12)) (1.7.0)
Requirement already satisfied: torchvision>=0.7.0 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 13)) (0.8.1)
Requirement already satisfied: tqdm>=4.41.0 in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 14)) (4.50.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (2.8.1)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.2.2->-r requirements.txt (line 5)) (0.10.0)
```

[그림 65] Jupyter notebook에서 추가적인 모듈을 통합 설치한 결과

- Anaconda prompt에서는 다음과 같이 진행한다. requirements.txt를 실행하여 YOLOv3를 수행하기 위해 필요한 모듈을 설치한다.



```
(base) C:\>cd C:\test1\yolov3
(base) C:\test1\yolov3>pip install -r requirements.txt
```

[그림 66] 필요 모듈을 설치하기 위한 코드

```
(base) C:\MLP\yolov3>python splitdata.py images
(base) C:\MLP\yolov3>pip install -r requirements.txt
Requirement already satisfied: Cython in c:\programdata\anaconda3\lib\site-packages (from -r requirements.txt (line 4)) (0.29.6)
Collecting matplotlib>=3.2.2
  Downloading matplotlib-3.3.3-cp37-cp37m-win_amd64.whl (8.5 MB)
|██████████| 8.5 MB 261 KB/s
Collecting numpy>=1.18.5
  Downloading numpy-1.19.4-cp37-cp37m-win_amd64.whl (12.9 MB)
|██████████| 12.9 MB 726 KB/s
```

[그림 67] 필요 모듈 설치 결과

- YOLOv3 모델에서 수정해야 할 yolo-spp.cfg 부분은 다음과 같다.

```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear
```

[그림 68] 모델에서 학습의 특성을 위해 수정해야 하는 코드

```
[yolo]
mask = 6,7,8
anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198,
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

[그림 69] 모델에서 학습의 특성을 위해 수정해야 하는 코드

- 모델에서 class configuration을 변환하는 부분이다. 주어진 이미지에서는 결함끼리의 구분이 없이 하나의 결함만을 취급한다. 그렇기에 결함에 대한 구분은 결함의 유무로만 판단하며 결함 존재시 클래스는 1로 둔다: Class = 1 (defect). Yolo layer convolutional filter 개수를 $3*(5+class\text{ 개수}) = 18$ 로 변환한다.
- 검출할 결함(defect)에 대하여 클래스 이름 “classes.names”를 지정한다. 이는 YOLOv3 모델에서 객체탐지를 수행하여 결함을 구별할 때 구분 지어지는 이름으로, 본 분석에서는 결함의 종류가 하나이므로 하나의 클래스 이름만 지정한다.

[단계 ⑦] 모델 훈련

⑦-1. 모델 학습 수행

- AI 분석모델 학습 내용(학습-검증-평가 비율, 학습조건, 모델 튜닝)
 - epoch: 150 (사용자 지정, 모델 성능 확인 후 추가 조정 필요)
 - batch size: 5 (사용자 지정, 모델 성능 확인 후 추가 조정 필요)
- 모델을 학습(training)하여 YOLOv3를 데이터에 맞게 훈련시킨다.
- 주피터 노트북으로 수행 시 다음과 같이 진행한다. (이 부분은 Anaconda prompt로 진행하는 것이 좋다.)

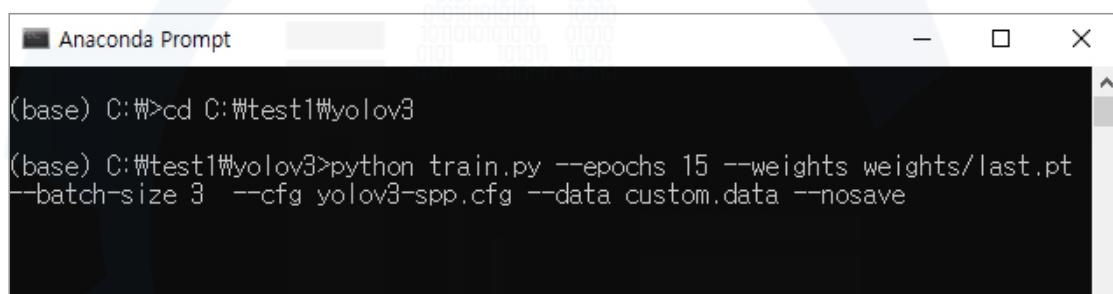
```
!python train.py --epochs 15 --weights weights/last.pt --batch-size 3 --cfg yolov3-spp.cfg --data custom.data --nosave --device cpu
```

[그림 70] Jupyter notebook에서 YOLOv3 모델 학습을 실행하는 코드

- 사용자의 필요에 따라 GPU 사용 시에는 위 코드에서 “--device cpu” 부분을 삭제하고, CPU 환경에서 실행 시에는 “--device cpu” 옵션을 추가한다.
- 본 가이드북에서는 초기 가중치(weights) 파일을 사용하여 모델을 학습하는 방식을 채용하며, 위 코드의 “--weights weights/last.pt” 부분이 해당 폴더에 저장된 초기 가중치 파일을 불러와 모델의 학습을 진행한다. 본 실습에서 사용하는 초기 가중치

파일은 object detection에 자주 사용되는 코코 데이터셋(COCO dataset)³⁾을 통해 사전에 학습하여 만들어진 가중치이다. 이같이 초기 가중치를 설정한 후 학습을 진행하는 이유는 제한된 컴퓨팅 파워를 통해 비용과 시간 측면을 고려하여 가장 효율적으로 필요한 모델 성능을 얻기 위해서이다.

- 모델 학습 완료 후, 학습의 결과물인 가중치는 마지막 epoch에 도출된 값이 초기 가중치와 동일한 경로(yolov3/weights)에 동일한 이름("last.pt")으로 저장되어 기존 가중치를 업데이트한다. 이는 이후 객체 탐지 수행 시에 사용된다.
- “라벨링 6종 세트”를 활용하여 학습을 진행하는 사용자는 실습 파일에 첨부된 각 세트에 해당하는 가중치 파일(last100.pt, last200.pt 등)을 [yolov3 > weights] 폴더로 이동 후 “last.pt”로 파일명을 변경하여 진행하면 된다.
- 최종적으로 얻은 가중치의 이름을 바꾸고 싶으면, yolov3 폴더의 train.py 코드 21번째 줄에서 “last = wdir + 'last.pt'”의 'last.pt' 부분을 원하는 이름으로 바꾸면 된다.
- Anaconda prompt에서 모델 학습 수행 시에는 아래의 코드를 prompt 창에 입력하여 진행한다.



```
(base) C:\test1\yolov3>python train.py --epochs 15 --weights weights/last.pt
--batch-size 3 --cfg yolov3-spp.cfg --data custom.data --nosave
```

[그림 71] Anaconda Prompt에서 모델 학습을 실행하는 코드

- YOLOv3 학습(train) 진행을 위한 코드에서 에폭(epochs)과 배치 사이즈(batch-size)는 사용자가 지정한다. 본 분석에서는 labeling 개수에 따른 비교 실험을 위해 에폭과 배치 사이즈를 각각 150, 5로 고정하였다.

```
python train.py --epochs 150 --batch-size 5 --weights weights/last.pt
--cfg yolov3-spp.cfg --data custom.data --nosave --device cpu
```

[그림 72] YOLOv3 모델 학습을 CPU에서 실행하는 코드

- 사용자의 필요에 따라 --device cpu 옵션을 추가하면 CPU 환경에서 실행할 수 있다.
- 모델 학습 과정 내용은 다음과 같다.

3) COCO (Common Objects in COntext) Dataset: 코코 데이터셋은 딥러닝 알고리즘 학습에 가장 많이 사용되는 오픈소스 객체 인식 데이터베이스 중 하나로, 대규모 객체감지, 세분화 및 캡션 데이터셋이다. 코코 데이터셋은 학습을 위해 이미 수백만 개 객체의 레이블이 지정된 수십만 개의 이미지를 포함하고 있다. 이같이 대용량의 레이블링이 잘 된 데이터셋을 기초 학습 자료로 사용했을 시, 이미지 처리 및 학습에 소요되는 시간이 크게 단축된다. (<https://deeppai.org/machine-learning-glossary-and-terms/Coco%20Dataset>)

```

Class Images Targets P R mAP@0.5 F1: 0%
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Class Images Targets P R mAP@0.5 F1: 100%
all 2 6 0.875 1 0.995 0.933
Epoch apu_mem Giou obj cls total targets img_size
0% 640/640 2.79G 1.06 0.046 0 1.1 6 448: 100%
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Class Images Targets P R mAP@0.5 F1: 0%
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Apex recommended for faster mixed precision training: https://github.com/NVIDIA/apex
Class Images Targets P R mAP@0.5 F1: 100%
all 2 6 0.724 0.833 0.835 0.775
149 epochs completed in 0.194 hours.

```

[그림 73] 모델 학습 과정

- 학습 과정을 완료하면 모델의 결과물로서 가중치(weight)가 출력되는데 이는 last.pt 파일로 저장된다. 모델 학습 후 가장 마지막 에폭에서 출력된 가중치가 저장된다.

| | | |
|----------------------------|-------|-----------|
| download_yolov3_weights.sh | SH 파일 | 1KB |
| last.pt | PT 파일 | 244,896KB |

[그림 74] 학습 결과(가중치)가 저장된 파일

[단계 ⑧] 결과 분석 및 해석

⑧-1. 객체 탐지 수행

- 위의 과정이 완료되면 학습된 YOLOv3 모델의 가중치가 weights 폴더에 last.pt 파일로 입력된다. 다음은 학습된 모델로 객체 탐지를 수행한다.
- 주피터 노트북으로 수행 시 다음과 같이 진행한다. 본 실습에서는 결함이 있는 15개 이미지에 대해 결함 객체 탐지를 진행하였다.

```
!python detect.py --weights weights/last.pt --source images --cfg yolov3-spp.cfg --names classes.names --output result --device cpu
```

[그림 75] Jupyter notebook에서 객체 탐지를 수행하는 코드

- 사용자의 필요에 따라 --device cpu 옵션을 추가하면 CPU 환경에서 실행할 수 있다. (GPU 사용 시 코드에서 “--device cpu”부분을 지우면 된다.)

```

Namespace(agnostic_nms=False, augment=False, cfg='yolov3-spp.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False,
          img_size=512, iou_thres=0.6, names='classes.names', output='result', save_txt=False, source='images', view_img=False, weights='weights/
          s/last.pt')
Using CUDA device0 _CudaDeviceProperties(name='GeForce RTX 2060', total_memory=6144MB)

Model Summary: 225 layers, 6.25733e+07 parameters, 6.25733e+07 gradients
image 1/15 images#001_20200622_203312(6).jpg: 416x512 Done. (0.026s)
image 2/15 images#001_20200623_003516(8).jpg: 416x512 Done. (0.026s)
image 3/15 images#001_20200901_003429(1).jpg: 416x512 Done. (0.026s)
image 4/15 images#001_20200922_163329(0).jpg: 416x512 Done. (0.025s)
image 5/15 images#001_20200922_163332(2).jpg: 416x512 Done. (0.026s)
image 6/15 images#002_20200714_043029(1).jpg: 512x512 1 defects, Done. (0.029s)
image 7/15 images#002_20200715_042416(9).jpg: 512x512 Done. (0.028s)
image 8/15 images#002_20200831_083545(0).jpg: 512x512 Done. (0.028s)
image 9/15 images#002_20200831_123134(5).jpg: 512x512 Done. (0.027s)
image 10/15 images#002_20200908_043137(8).jpg: 512x512 Done. (0.024s)
image 11/15 images#002_20200915_003459(4).jpg: 512x512 Done. (0.023s)
image 12/15 images#002_20200917_043110(7).jpg: 512x512 Done. (0.023s)
image 13/15 images#002_20200917_202952(6).jpg: 512x512 Done. (0.022s)
image 14/15 images#002_20200917_203001(5).jpg: 512x512 1 defects, Done. (0.023s)
image 15/15 images#002_20200922_083747(7).jpg: 512x512 1 defects, Done. (0.023s)
Results saved to C:\test1\yolov3\result
Done. (1.519s)
  
```

[그림 76] Jupyter notebook에서 이미지 결함 객체 탐지를 수행한 결과

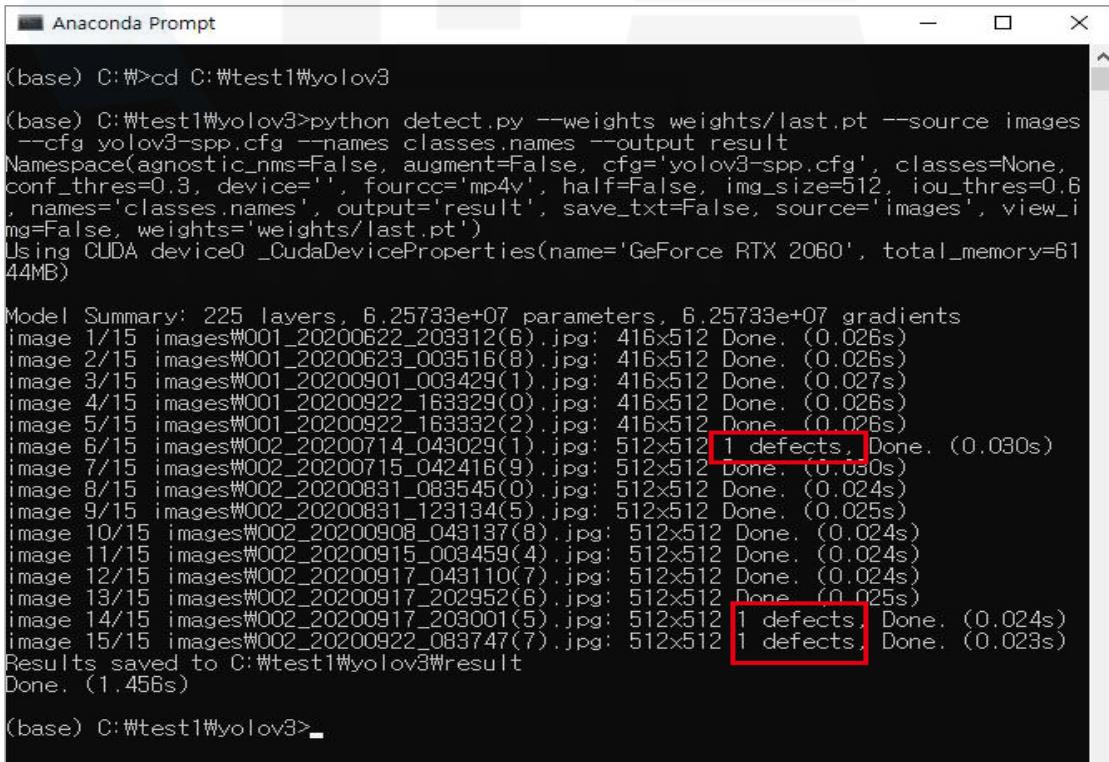
- 객체 탐지를 수행한 결과는 yolov3 폴더 안의 result 폴더 안에서 확인할 수 있다.
- Anaconda prompt에서 수행 시에는 다음과 같이 진행한다. 아래의 코드를 실행하여 result라는 폴더 안에 시각화된 결과가 출력되도록 한다.

```

python detect.py --weights weights/last.pt --source images1 --cfg
yolov3-spp.cfg --names classes.names --output result --device cpu
  
```

[그림 77] results 폴더 안에 시각화 결과를 출력하기 위한 코드

- 사용자의 필요에 따라 --device cpu 옵션을 추가하면 CPU 환경에서 실행할 수 있다.
- 객체 탐지가 수행되는 화면은 다음과 같다.



```

(base) C:\>cd C:\test1\yolov3
(base) C:\test1\yolov3>python detect.py --weights weights/last.pt --source images
--cfg yolov3-spp.cfg --names classes.names --output result
Namespace(agnostic_nms=False, augment=False, cfg='yolov3-spp.cfg', classes=None,
conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6,
names='classes.names', output='result', save_txt=False, source='images', view_i
mg=False, weights='weights/last.pt')
Using CUDA device0 _CudaDeviceProperties(name='GeForce RTX 2060', total_memory=61
44MB)

Model Summary: 225 layers, 6.25733e+07 parameters, 6.25733e+07 gradients
image 1/15 images#001_20200622_203312(6).jpg: 416x512 Done. (0.026s)
image 2/15 images#001_20200623_003516(8).jpg: 416x512 Done. (0.026s)
image 3/15 images#001_20200901_003429(1).jpg: 416x512 Done. (0.026s)
image 4/15 images#001_20200922_163329(0).jpg: 416x512 Done. (0.025s)
image 5/15 images#001_20200922_163332(2).jpg: 416x512 Done. (0.026s)
image 6/15 images#002_20200714_043029(1).jpg: 512x512 1 defects, Done. (0.030s)
image 7/15 images#002_20200715_042416(9).jpg: 512x512 Done. (0.030s)
image 8/15 images#002_20200831_083545(0).jpg: 512x512 Done. (0.024s)
image 9/15 images#002_20200831_123134(5).jpg: 512x512 Done. (0.025s)
image 10/15 images#002_20200908_043137(8).jpg: 512x512 Done. (0.024s)
image 11/15 images#002_20200915_003459(4).jpg: 512x512 Done. (0.024s)
image 12/15 images#002_20200917_043110(7).jpg: 512x512 Done. (0.024s)
image 13/15 images#002_20200917_202952(6).jpg: 512x512 Done. (0.025s)
image 14/15 images#002_20200917_203001(5).jpg: 512x512 1 defects, Done. (0.024s)
image 15/15 images#002_20200922_083747(7).jpg: 512x512 1 defects, Done. (0.023s)
Results saved to C:\test1\yolov3\result
Done. (1.456s)

(base) C:\test1\yolov3>
  
```

[그림 78] 객체 탐지 수행 과정 중 출력되는 내용

- [그림 78]에서 붉은색으로 표시한 부분을 확인하면 한 이미지에서 몇 개의 객체가 탐지되었는지 알 수 있다. (여기에서는 제품의 결함으로 클래스 이름을 설정하여 “defect”로 표기된다.) 탐지되지 않는 부분은 따로 숫자와 클래스가 표시되지 않으며, 가장 우측에는 이미지마다 객체 탐지에 소요되는 시간이 나타난다.
- 객체 탐지로 결함을 찾아내어 시각적으로 표현한 결과 중 일부는 다음과 같다.



[그림 79] 객체 탐지로 결함을 찾아내어 시각적으로 표현한 결과 예시

⑧-2. 모델 성능 평가

- 다음은 최종 출력된 모델의 성능을 평가한다.
- 주피터 노트북으로 수행 시에는 다음과 같이 진행한다. 아래 코드를 실행하여 출력되는 결과물에서 노란색으로 표시된 부분이 모델의 성능평가 결과이며, 순서대로 정밀도(precision), 재현율(Recall), mAP(mAP@0.5), F1 score를 나타낸다.

```
python test.py --cfg yolov3-spp.cfg --batch-size 3 --data custom.data --weights weights/last.pt --device cpu
```

[그림 80] Jupyter notebook에서 모델 평가를 수행하는 코드

- 사용자의 필요에 따라 --device cpu 옵션을 추가하면 CPU 환경에서 실행할 수 있다. (GPU 사용 시 코드에서 “--device cpu”부분을 지우면 된다.)

```

Namespace(augment=False, batch_size=3, cfg='yolov3-spp.cfg', conf_thres=0.001, data='custom.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False, task='test', weights='weights/last.pt')
Using CUDA device0 _CudaDeviceProperties(name='GeForce RTX 2060', total_memory=6144MB)

Model Summary: 225 layers, 6.25733e+07 parameters, 6.25733e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25465e+07 parameters, 6.25465e+07 gradients
all      3      5    0.619     0.6     0.58     0.61
Speed: 35.3/81.7/117.0 ms inference/NMS/total per 512x512 image at batch-size 3
  
```

[그림 81] Jupyter notebook에서 모델 평가를 수행한 결과

- Anaconda Prompt에서 수행 시에는, 다음의 코드를 실행한다. 최종 출력된 YOLOv3의 가중치(weight)에 결함(defect)을 검출하기 위한 데이터를 투입하여 객체 탐지를 수행하는 명령어이다.

```

python test.py --cfg yolov3-spp.cfg --data custom.data --weights
weights/last.pt --device cpu
  
```

[그림 82] 모델 평가 및 수행을 위한 파이썬 실행 코드

- 사용자의 필요에 따라 --device cpu 옵션을 추가하면 CPU 환경에서 실행할 수 있다.
- 다음과 같이 에폭(epoch), 배치사이즈(batch size), Iteration을 수행 과정에서 확인 할 수 있다

| Epoch | gpu_mem | GIoU | obj | cls | total | targets | img_size |
|---------|---------|------|---------|-------|-------|----------|---|
| 499/499 | 6.34G | 0.91 | 0.0541 | 0 | 0.965 | 9 | 416: 100% 13/13 [00:04<00:00, 3.10it/s] |
| Class | Images | | Targets | P | R | mAP@0.5 | |
| all | 7 | 21 | | 0.942 | 1 | F1: 100% | 2/2 [00:00<00:00, 3.82it/s] |

[그림 83] 객체 탐지 수행 과정에서 확인한 에폭, 배치사이즈, Iteration 결과

- 또한, 모델의 과정이 끝나면 yolov3 디렉토리에 “results.png” 파일이 저장되는데. 이 이미지에서 확인할 수 있는 Plot으로도 최종 결과물과 그 변화를 시각적으로 판단 할 수 있다.
- 추가적으로 객체 탐지시 결함의 영역을 표시하는 상자의 색을 바꾸는 부분은 다음과 같다.

```

# Get names and colors
names = load_classes(opt.names)
colors = [[random.randint(0, 255) for _ in range(3)] for _ in range(len(names))]

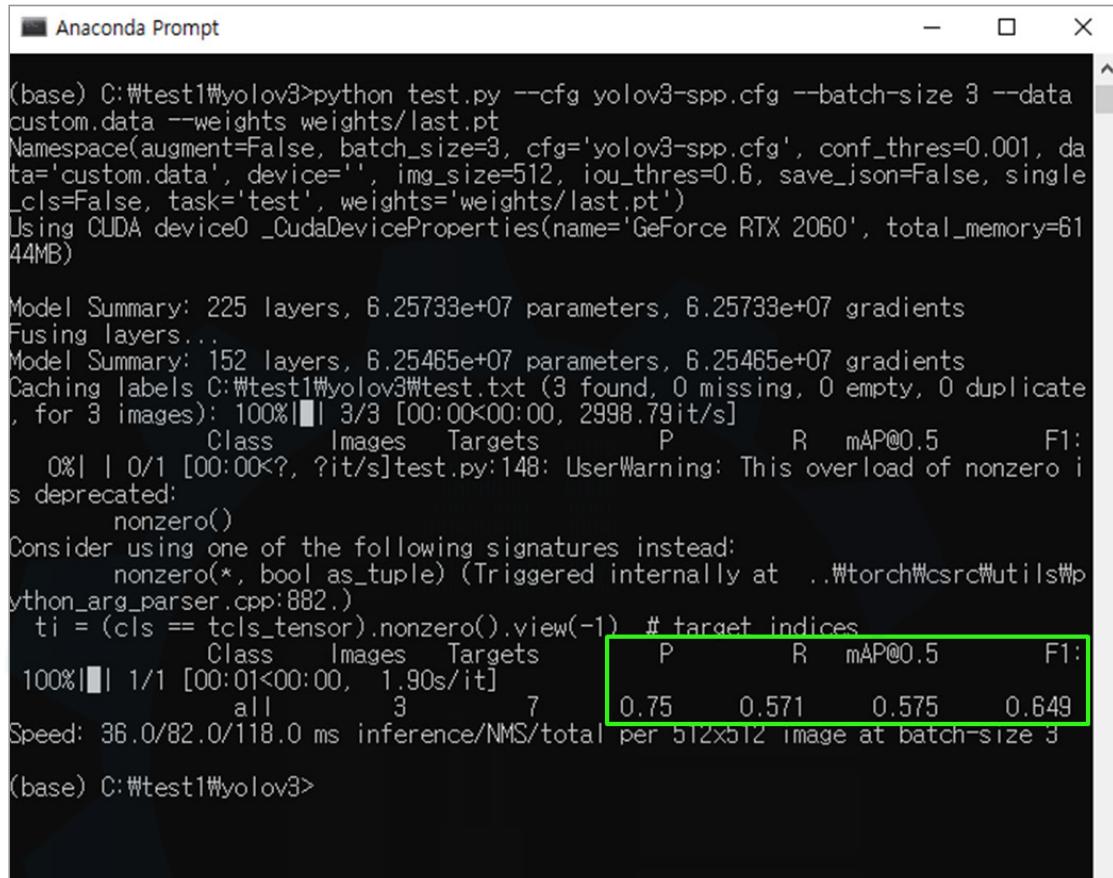
# colors = [[(255,0,0)] for _ in range(len(names))]
  
```

[그림 84] 결함 영역을 표시하는 상자의 색을 바꾸기 위한 코드

- 위 코드에서 “colors=” 부분을 확인하면 현재는 RGB 값의 0~255를 랜덤으로 3개 추출하여 객체 탐지 결함 영역 상자의 색을 정하도록 되어있다. 그 아래 줄 주석으로 되어있는 부분을 참고하여 다른 색으로 사용자가 지정할 수 있다.

⑧-3. 이미지 개수에 따른 객체 탐지 성능 확인

- 위 결과로 데이터에 대한 YOLOv3의 객체 탐지 결과의 성능 확인지표들인 정밀도(Precision)와 재현율(Recall), F1-score, mAP(mean average precision) 값이 출력된다. 각각 0과 1 사이의 값을 가지며 값이 높을수록 좋다.



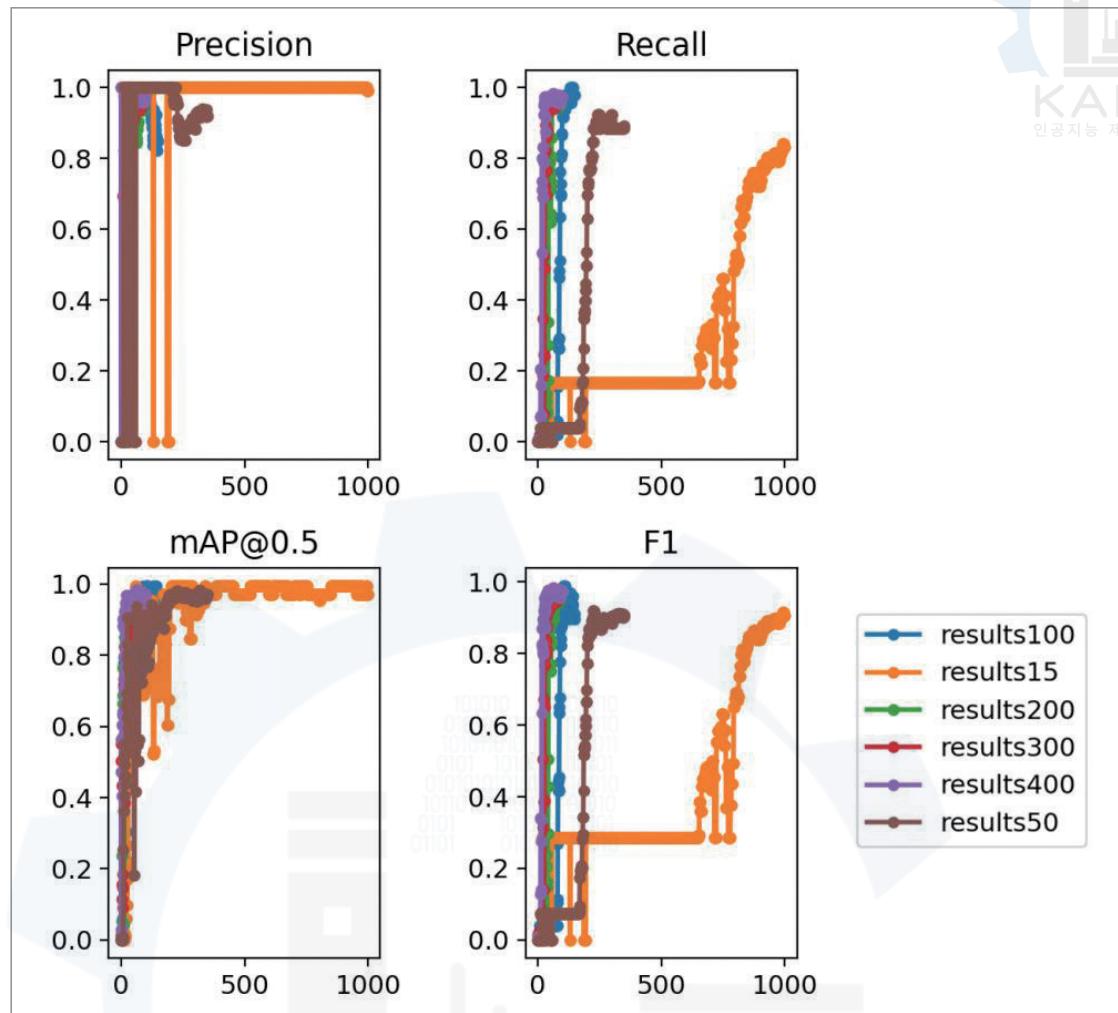
```
(base) C:\test1\yolov3>python test.py --cfg yolov3-spp.cfg --batch-size 3 --data custom.data --weights weights/last.pt
Namespace(augment=False, batch_size=3, cfg='yolov3-spp.cfg', conf_thres=0.001, data='custom.data', device='', img_size=512, iou_thres=0.6, save_json=False, single_cls=False, task='test', weights='weights/last.pt')
Using CUDA device0 _CudaDeviceProperties(name='GeForce RTX 2060', total_memory=6144MB)

Model Summary: 225 layers, 6.25733e+07 parameters, 6.25733e+07 gradients
Fusing layers...
Model Summary: 152 layers, 6.25465e+07 parameters, 6.25465e+07 gradients
Caching labels C:\test1\yolov3\test.txt (3 found, 0 missing, 0 empty, 0 duplicate
, for 3 images): 100%|████| 3/3 [00:00<00:00, 2998.79it/s]
      Class   Images   Targets      P      R    mAP@0.5      F1:
      0% | 0/1 [00:00<?, ?it/s] test.py:148: UserWarning: This overload of nonzero is deprecated:
      nonzero()
Consider using one of the following signatures instead:
      nonzero(*, bool as_tuple) (Triggered internally at ...#torch#csrc#utils#python_arg_parser.cpp:882.)
      ti = (cls == tcls_tensor).nonzero().view(-1) # target indices
      Class   Images   Targets      P      R    mAP@0.5      F1:
      100%|████| 1/1 [00:01<00:00,  1.90s/it]
          all       3        7      0.75    0.571    0.575    0.649
Speed: 36.0/82.0/118.0 ms inference/NMS/total per 512x512 image at batch-size 3

(base) C:\test1\yolov3>
```

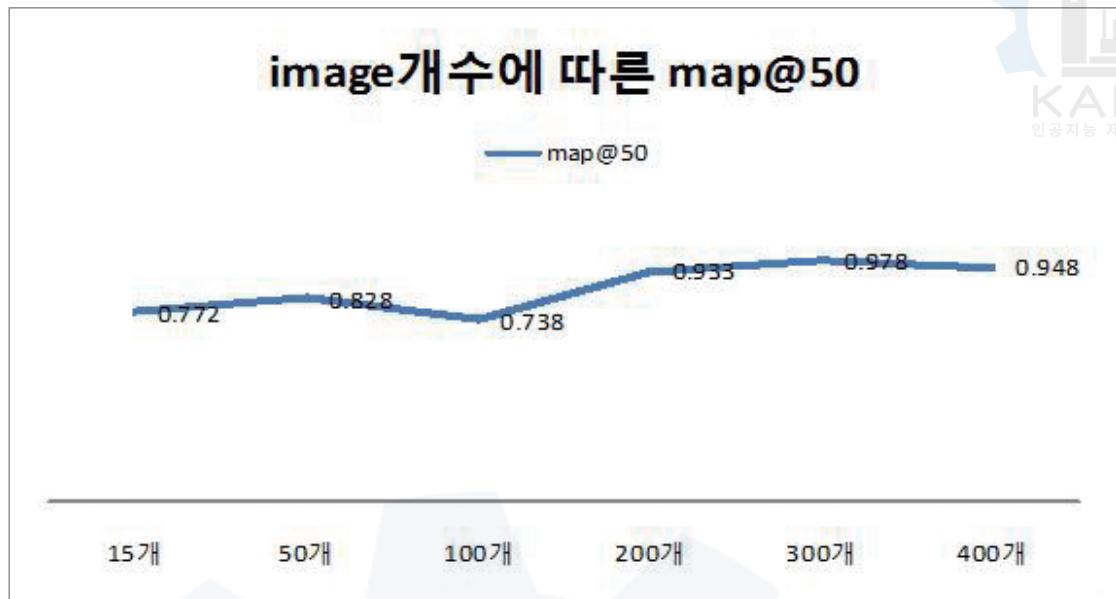
[그림 85] 객체 탐지 모델의 성능 확인 결과 예시

- 일반적으로 이미지의 개수가 많아지면 탐지 성능이 좋아지는 것은 사실이나 공정과정에서 데이터를 얻기 위한 시간이 길어지는 등 효율성이 떨어진다. 따라서 최적의 이미지 개수를 찾기 위해 효율성을 극대화하는 실험을 하였다.
- 이미지 결함에 대하여 얼마나 라벨링 작업을 해야 할지에 대한 실험을 진행하였다. 이미지 라벨링(이미지 개수)을 각각 15개, 50개, 100개, 200개, 300개, 400개로 수행하여 6개의 모델을 학습시키고, 각 모델의 성능을 평가하기 위해 앞서 제시한 4가지 성능지표인 정밀도(precision), 재현율(Recall), F1 score, mAP@50에 대하여 결과를 확인하였다. 모델 학습 과정에서 학습 데이터셋과 평가 데이터셋의 비율은 8:2로 동일하게 하여 진행하였다.
- 이와 같이 수행한 모델 성능평가 결과를 시각화한 결과는 아래와 같으며, 지표별로 이미지 라벨링 개수의 성능을 비교해볼 수 있다.



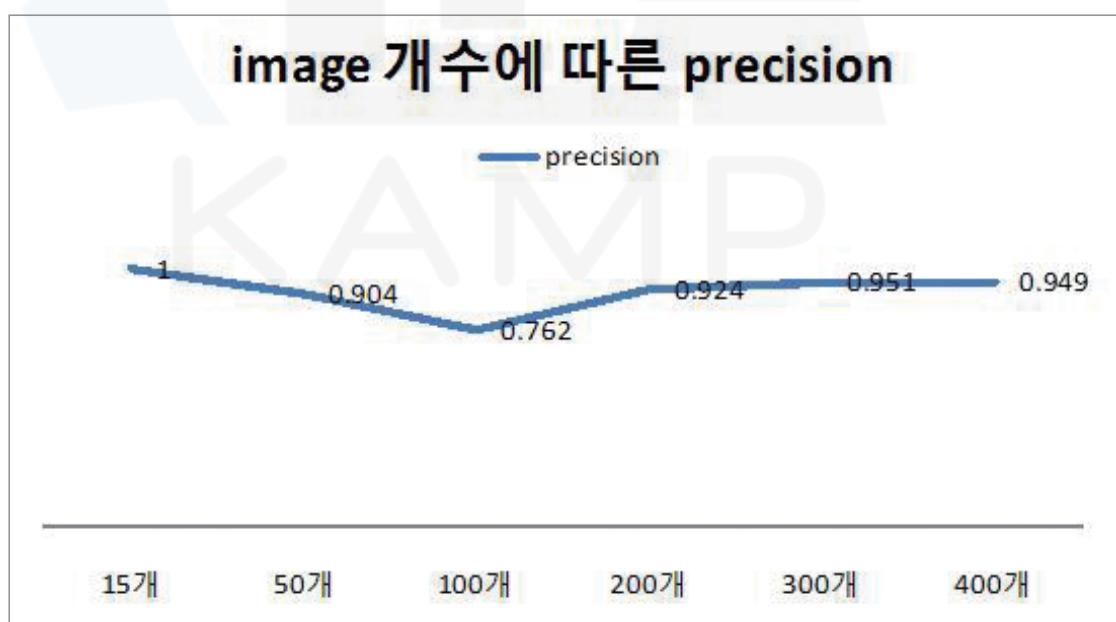
[그림 86] 객체 탐지 모델의 성능을 시각적으로 확인한 결과

- 에폭(epoch)이 진행됨에 따라 4가지 성능지표 모두 1과 가까워지는 것을 확인할 수 있다.
- 충분한 학습을 하기 위해 이미지 개수가 줄어들수록 더 많은 에폭을 진행하였고 4 가지 지표가 0.9 이상으로 도달하는 시점에 학습을 중단하였다.
- 학습을 모두 끝낸 후, 학습되지 않은 테스트 데이터셋 100개를 준비하여 각각의 mAP@50, F1 score를 비교하는 실험을 진행하였다.
- 다음은 mAP@50 결과이다. map@50은 대표적인 객체 탐지(object detection) 성능지표로 detection의 정확도를 나타낸다.



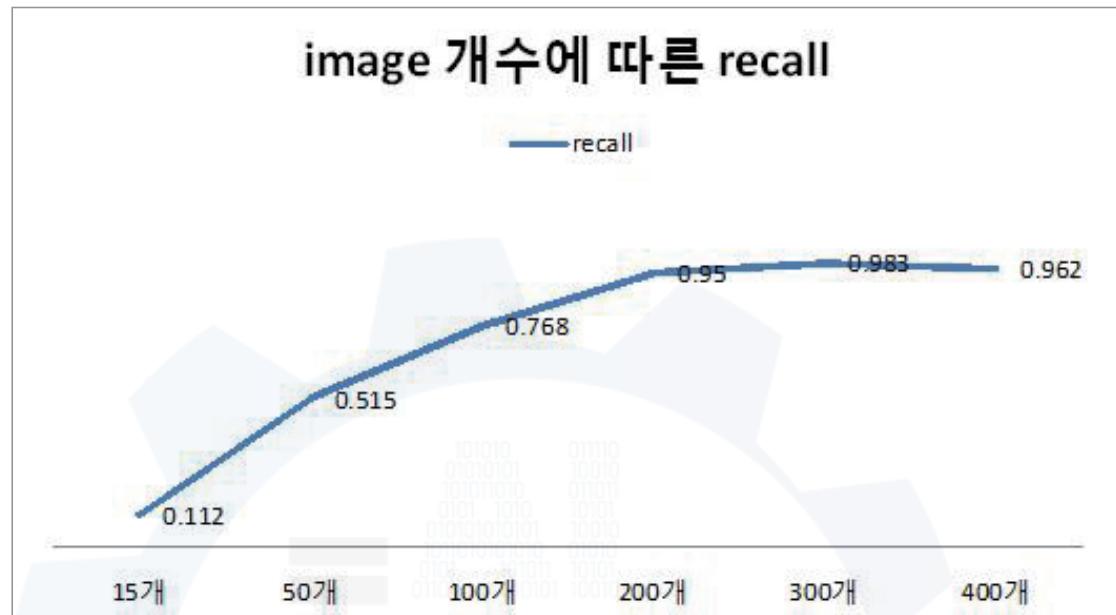
[그림 87] 이미지 개수에 따른 map@50 변화 추이

- 대체적으로 이미지의 개수가 증가하면 map@50이 증가한 것을 볼 수 있다. 이미지 개수가 200 이상이 넘어가면서부터는 0.9 이상으로 값이 수렴하였고 15~100개에서는 성능이 하락하는 것을 볼 수 있다. 다만 15개에서도 0.772의 준수한 map가 나타난 것으로 보아 결함을 감지한다면 IOU > 50으로 꽤 정확한 수준으로 탐지하는 것을 알 수 있다.
- 다음은 정밀도(precision) 결과이다. 이미지 개수에 상관없이 높은 수준의 정밀도 값을 보인 것으로 보아 결함 탐지를 한다면 실제 결함일 확률이 높다.



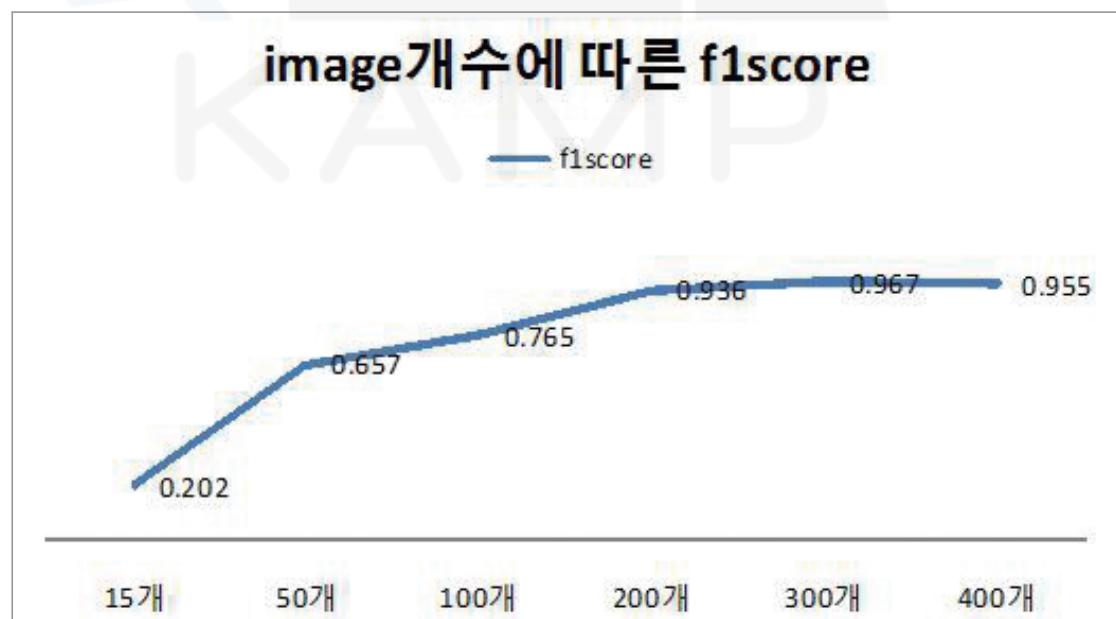
[그림 88] 이미지 개수에 따른 정밀도 변화 추이

- 다음은 재현율(recall) 결과이다. 이미지의 개수가 증가하면 재현율 값이 증가하였으며, 200개 이후부터는 값이 수렴하지만, 100개 이하로는 급격히 값이 하락하는 것을 확인할 수 있다. 즉, 이미지 개수가 줄어들 경우 실제 결함을 찾지 못할 확률은 급격히 증가하는 것을 확인할 수 있다.



[그림 89] 이미지 개수에 따른 재현율 변화 추이

- 다음은 f1 score 결과이다. 일반적으로 정밀도와 재현율의 값은 반비례하는 경향을 가지며 이를 동시에 감안하기 위해 조화평균인 F1 score를 대표적으로 사용한다. 모든 이미지 개수에서 정밀도는 높지만, 재현율 값은 100개 이하부터는 급격히 감소하기 때문에 재현율과 같은 경향성의 그래프가 그려진 것을 확인할 수 있다.



[그림 90] 이미지 개수에 따른 조화평균 변화 추이

- 이 경우에도 대체적으로 이미지 개수가 증가하면 F1 score 값이 증가하였으며, 200개 이상에서는 값이 수렴하지만, 200개 이하로는 급격히 떨어지는 것을 확인할 수 있다. 즉, 이미지 개수가 줄어들 경우 객체 탐지를 수행한다면 꽤 정확하게 탐지를 하지만, 결함을 감지하지 못하는 확률은 급격히 줄어든 것을 확인할 수 있다.
- 결론적으로 4가지 지표가 모두 1과 가까운 지점인 200개가 최적의 이미지 개수라고 할 수 있다.

⑧-3. 결함 영역의 시각적 확인

- 객체 탐지로 구한 이미지의 결함 영역을 시각적으로 확인하기 위하여 영역 이미지 자르기 코드를 수행한다. detect.py에서 자른 이미지 리스트(cropped image list)를 생성하는 것으로, [8-1]에서 객체 탐지 수행 시 detect.py 코드를 실행할 때 같이 실행된다.

```
cropped_images = []
```

[그림 91] 자른 이미지 리스트 생성 코드

- detect.py의 131번 줄에서 cropped_image는 잘린 이미지들의 x, y 좌표를 리턴해주는 함수이며 각각에 crop된 image들을 image list에 추가시켜준다.

```
if save_img or view_img: # Add bbox to image
    cropped_images.append(cropped_image(xyxy, im0))
    label = '%s %.2f' % (names[int(cls)], conf)
    plot_one_box(xyxy, im0, label=label, color=colors[int(cls)])
```

[그림 92] 잘린 이미지 좌표를 반환하여 리스트에 추가하는 코드

- 추가된 코드는 다음과 같다.

```
cropped_images.append(cropped_image(xyxy, im0))
```

[그림 93] 잘린 이미지 추가 코드

- cropped_image 함수는 utils.py에서 추가 구현되었으며 다음과 같다.

```
def cropped_image(x, img):
```

[그림 94] 잘린 이미지 함수화 코드

- 이 함수는 x, y 좌표와 img를 인자로 받는다.

```
cropped_image = img[int(x[1]):int(x[3]), int(x[0]):int(x[2])]
```

[그림 95] 이미지 좌표 인자를 받는 코드

- crop image를 얻기 위해 image에서 yolov3가 탐지한 x, y 좌표만 추출한다. 그 후 crop image를 리턴한다.

```
return cropped_image
```

[그림 96] 잘린 이미지 반환 코드

- detect.py의 146번 줄에서 crop image list를 세로로 합치기 위해 vconcat_resize_min 함수를 사용하여 이미지를 출력한다.

```
if save_img:
    if dataset.mode == 'images':
        im0 = vconcat_resize_min(cropped_images)
        cv2.imwrite(save_path, im0)
```

[그림 97] 잘린 이미지를 합쳐서 출력하게 해주는 코드

- 추가된 코드는 다음과 같다.

```
im0 = vconcat_resize_min(cropped_images)
```

[그림 98] 잘린 이미지 크기 조정 코드

- utils.py에서 구현한 vconcat_resize_min 함수는 다음과 같다. 이 함수는 im_list를 인자로 받는다.

```
def vconcat_resize_min(im_list, interpolation=cv2.INTER_CUBIC):
```

[그림 99] 이미지 조정 함수화 코드

- 각각의 이미지의 크기가 다르기 때문에 im_list 중 가장 작았던 사이즈로 크기를 똑같게 해주는 과정을 거친다.

```
w_min = min(im.shape[1] for im in im_list)
im_list_resize = [cv2.resize(im, (w_min, int(im.shape[0] * w_min / im.shape[1])), interpolation=interpolation)
                  for im in im_list]
```

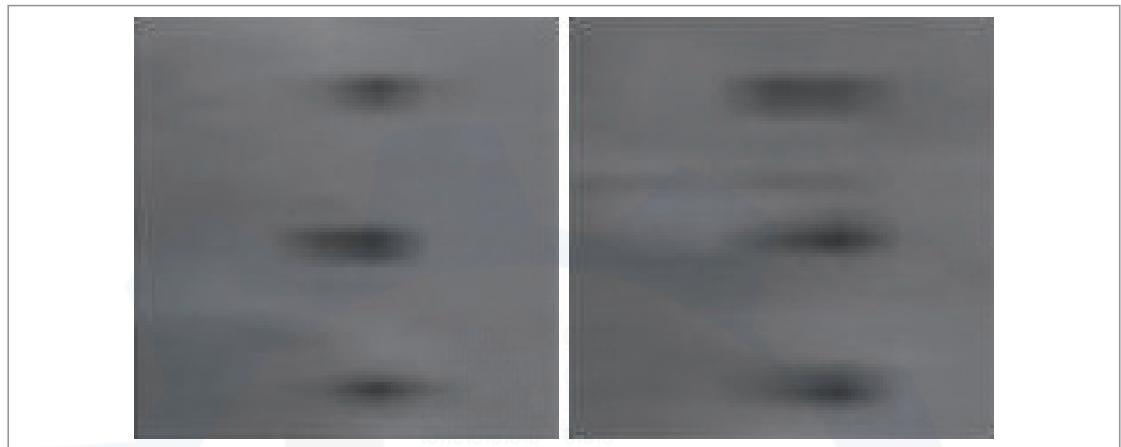
[그림 100] 이미지 크기 동일화 코드

- 그다음 cv2 라이브러리에 저장된 vconcat 함수를 사용하여 resize된 im_list를 합쳐준다.

```
return cv2.vconcat(im_list_resize)
```

[그림 101] 크기 조정된 이미지를 합치는 코드

- YOLOv3 모델이 경계 상자로 결함의 영역이라고 판단하여 추출한 이미지는 다음과 같다. 아래 두 개의 이미지는 각각 한 제품에 대한 것으로 각각 3개의 결함이 인식되어 추출되었으며 그 부분을 저장한 이미지이다.



[그림 102] YOLOv3 모델이 결함 영역으로 판단하여 추출한 이미지

⑧-4. 제조현장 관점에서 분석결과 해석하기

- 본 가이드북의 분석 예시를 통해 X-RAY 이물검출기에서 생성된 불량이미지를 대상으로 객체 탐지를 수행하여 결함(불량) 패턴을 분류하는 것이 가능할 뿐만 아니라, 정밀도 및 재현율 모두 상당히 높고, 각 이미지에 대한 분석 소요시간도 0.02~0.03초로 매우 짧아 X-RAY 이물검출기의 자체 분류 결과를 효과적으로 보완할 수 있다는 것을 확인하였다. 특히, 분석 설정 중 하나인 라벨링 개수(이미지 개수)를 200개로 설정하면 정밀도≈0.92, 재현율≈0.95, F1 score≈0.93으로 최적의 분석결과를 얻을 수 있음을 발견하였다. (개수를 이보다 늘리면 더 높은 성능을 얻을 수 있지만, 공정과정에서 데이터를 얻기 위한 시간이 길어지는 등 효율성 문제가 있다.)
- X-RAY 이물검출기의 미검·과검 문제를 보완하기 위해 이중·삼중의 검사를 수행하는 것보다 본 가이드북의 예시와 같이 객체 탐지 분석을 수행하는 것이 보다 더 정확하고, 빠르다는 것을 확인하였으나, 본 분석은 이물검출기에서 일차적으로 생성 및 보관된 이미지를 대상으로 하기 때문에 사후적 분석이라는 한계가 있다.
- 위와 같은 이유로 본 가이드북에서 개발된 분석 모델이 실제 공정에 실시간으로 적용되기 위해서는 지금과 같이 SD Card를 통해 X-RAY 이물검출기 데이터를 PC로 옮겨와 분석하는 방식이 아니라, 이물검출기에서 생성되는 데이터를 실시간으로 수집 및 관리하는 모듈과 이미지 처리 및 객체 탐지 딥러닝 알고리즘이 통합된 하나의 모듈을 구현하여 생산라인에 적용하여야 한다.

3. 유사 타 현장의 「X-ray 검사장비 AI 데이터셋」 분석 적용

3.1 본 분석이 적용 가능한 제조현장 소개

- 비전검사장비를 사용하는 대부분의 중소기업 제조업체에서는 본 가이드북에서 제시된 것과 동일한 문제를 안고 있을 것이다. 이와 같은 어려움이 있는 산업현장에서 만약 검사장비에서 검사결과 이미지 데이터의 추출이 가능하다면 본 분석은 적용 가능하다.

3.2 본 「X-ray 검사장비 AI 데이터셋」 분석을 원용하여 타 제조현장 적용 시, 주요 고려사항

- 비전검사장비의 종류 및 생산제품에 따라 분석결과가 달라지기 때문에 해당 장비 및 제품의 데이터로 분석모델을 다시 개발하여야 한다.
- 본 분석에서 활용한 특성값은 타 장비에서 수집되는 이미지 형식과는 맞지 않을 수 있기 때문에 현장 전문가의 의견을 반영하여 적용 여부를 결정해야 한다.

부록 분석환경 구축을 위한 설치 가이드

◎ Python을 사용하는 이유

- 데이터를 분석 및 예측하기 위해 Python 프로그래밍 언어를 사용하는 이유는 다음과 같다. 첫째, 비교적 읽고 쓰기 쉬운 프로그래밍 언어이며, 둘째, 효율적인 메모리 관리 기능을 갖추고 있고, 셋째, 머신러닝 프레임워크와 라이브러리가 풍부하다는 장점이 있다.

◎ Python 플랫폼 : Anaconda

- 먼저 Python 플랫폼인 Anaconda를 설치한다. Anaconda는 핵심적인 과학 및 수학 Python 라이브러리를 포함한 패키지이며, 머신러닝에 필수적인 툴이다. 쉽게 말해서, Anaconda를 설치하면 분석과 예측을 수행하기 위해 필요한 다양한 패키지들을 쉽게 설치할 수 있으며, 코드를 직접 입력하고 실행시킬 ‘개발환경’을 Anaconda Navigator라는 하나의 통합된 공간에서 편리하게 설치할 수 있다. 분석에 들어가기에 앞서 다음 순서에 맞춰 Anaconda를 설치한다.

◎ Anaconda Installer 설치

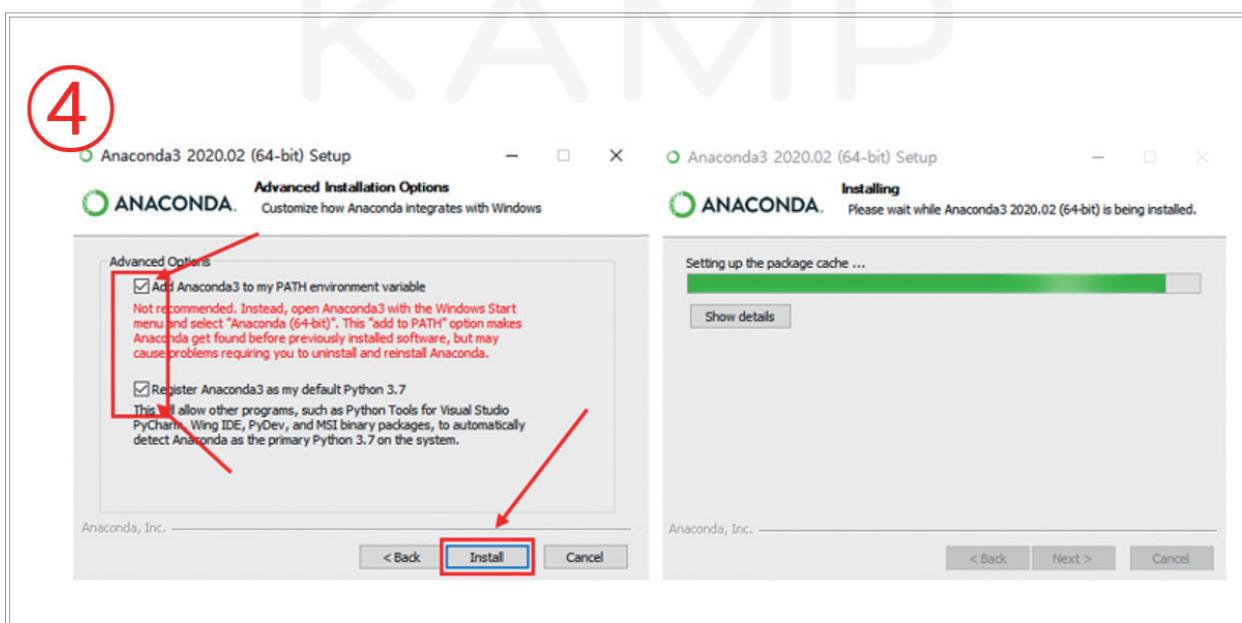
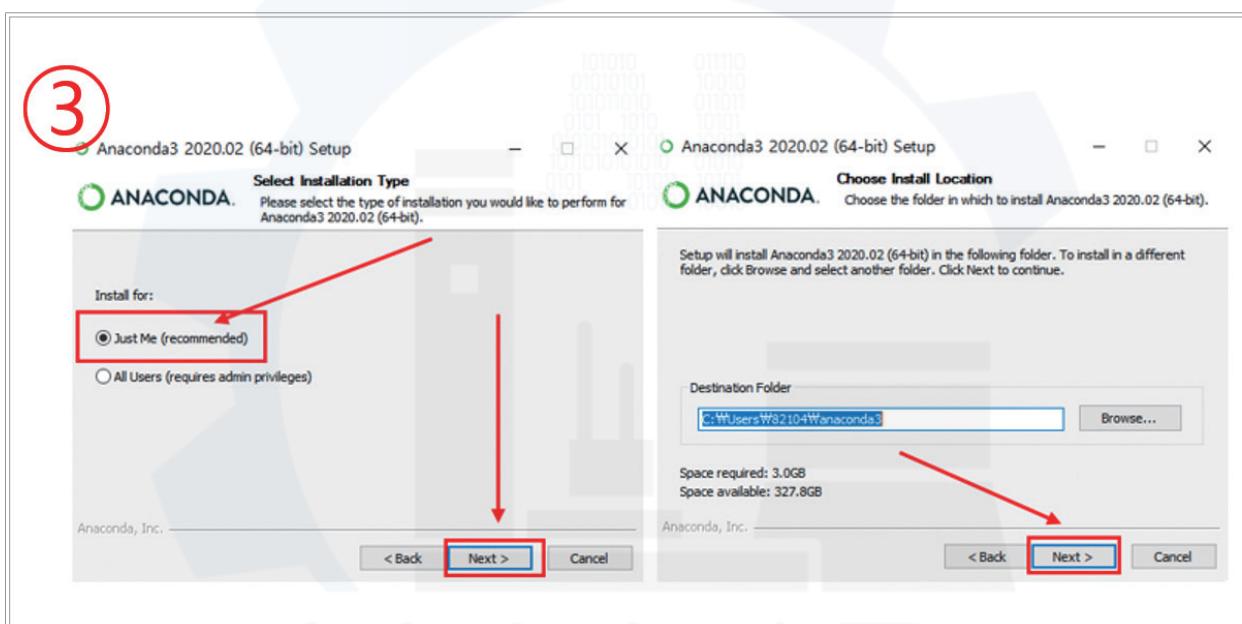
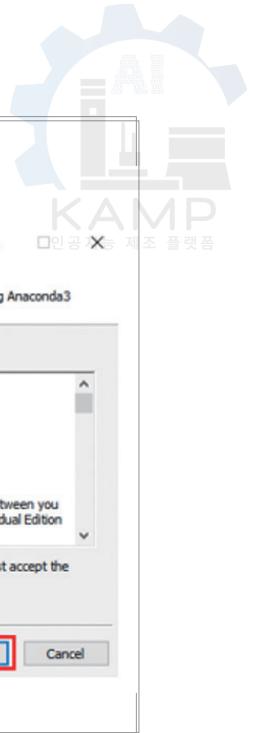


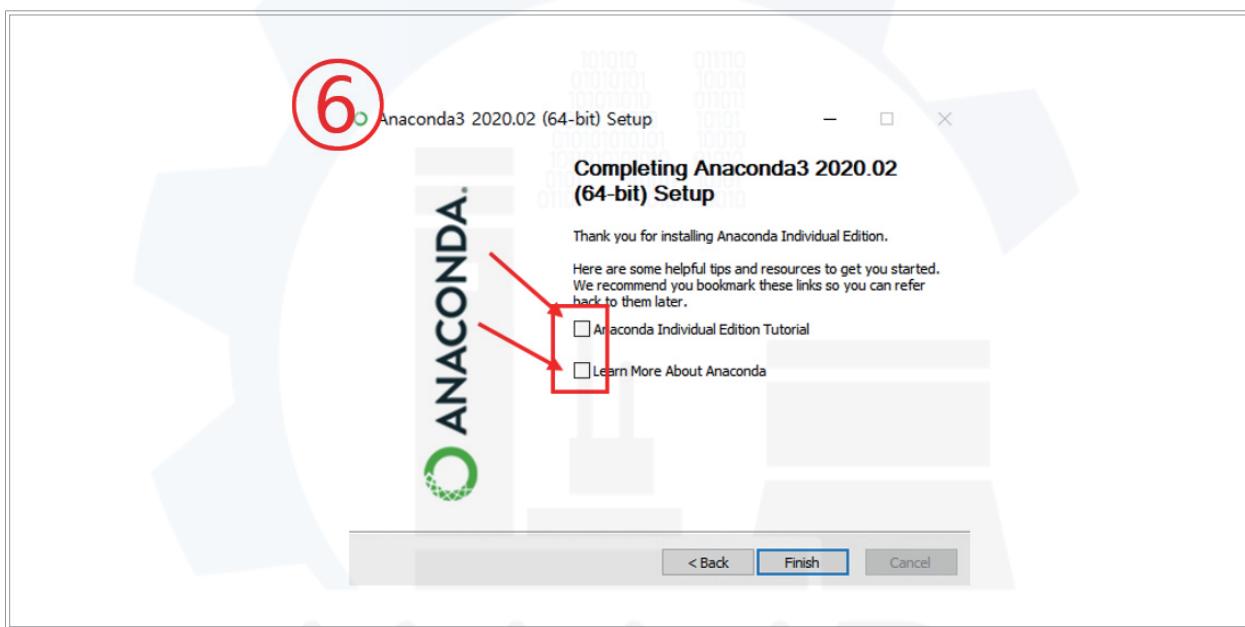
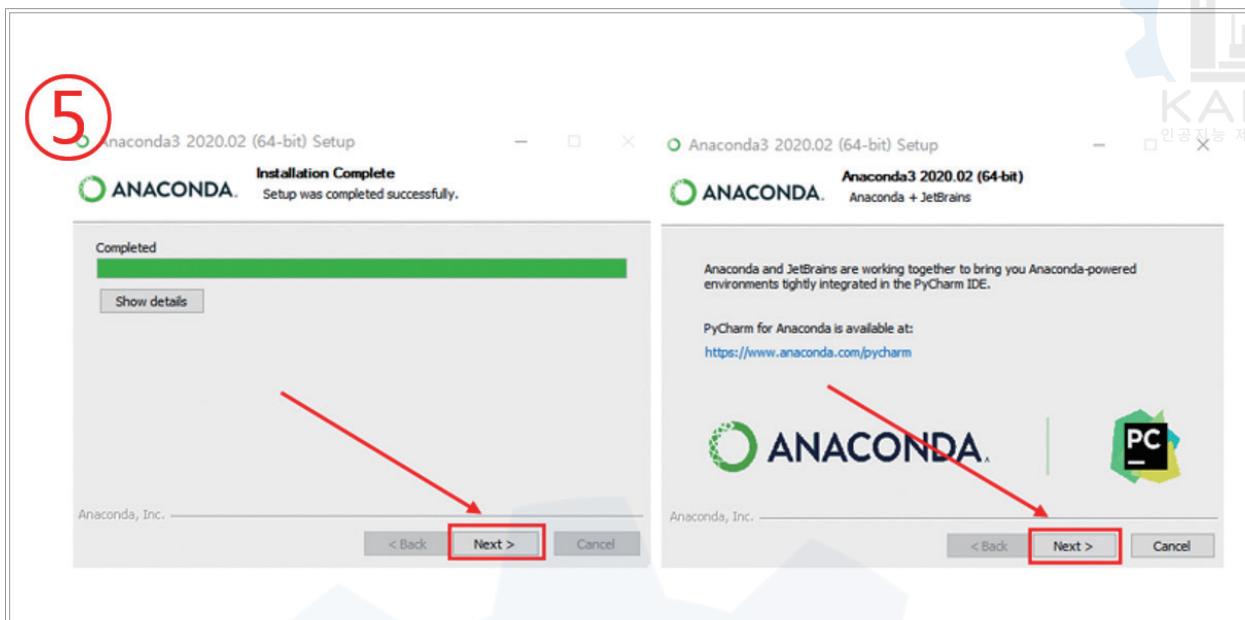


- ① www.anaconda.com/download 주소로 접속한다.
- ② 'Download' 버튼을 클릭한다.
- ③ Windows, MacOS, Linux 버전 중 자신의 PC 운영체제와 일치하는 버전을 선택하여 Installer를 다운로드 한다.

◉ Anaconda 실행파일 설치(Windows 64-bit 기준)

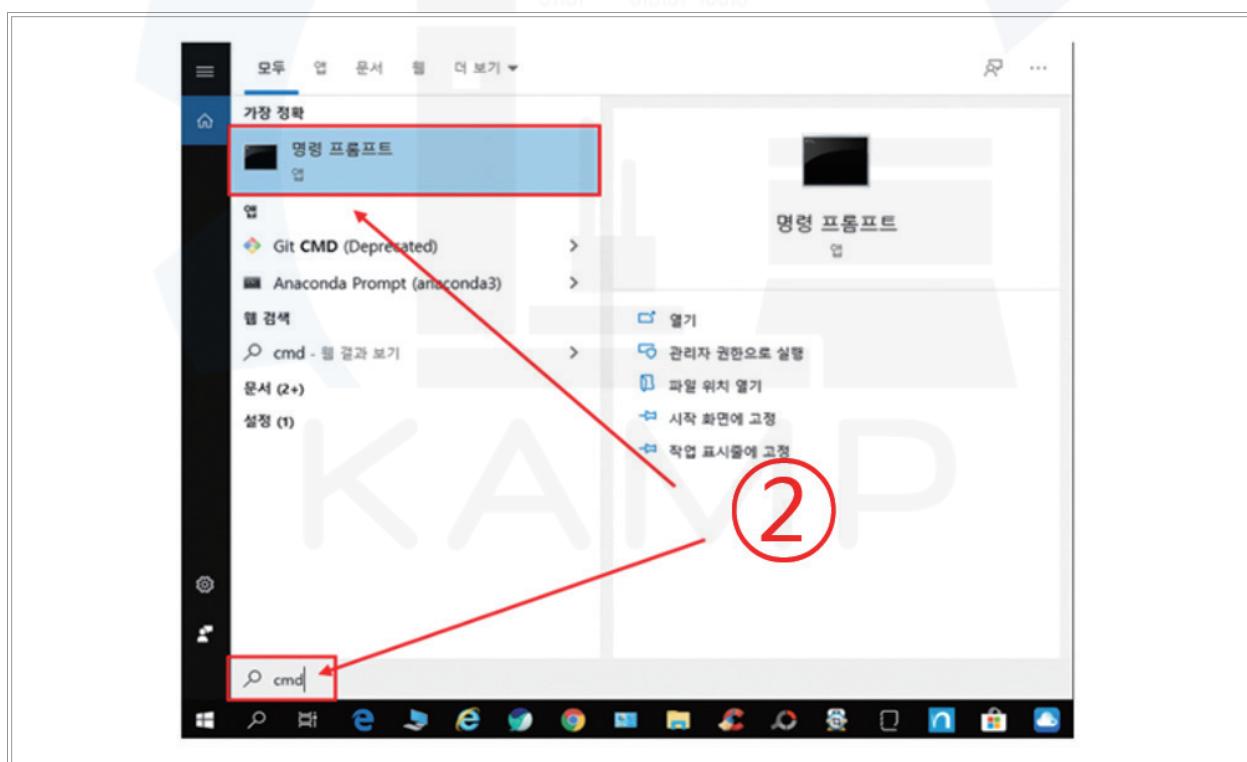
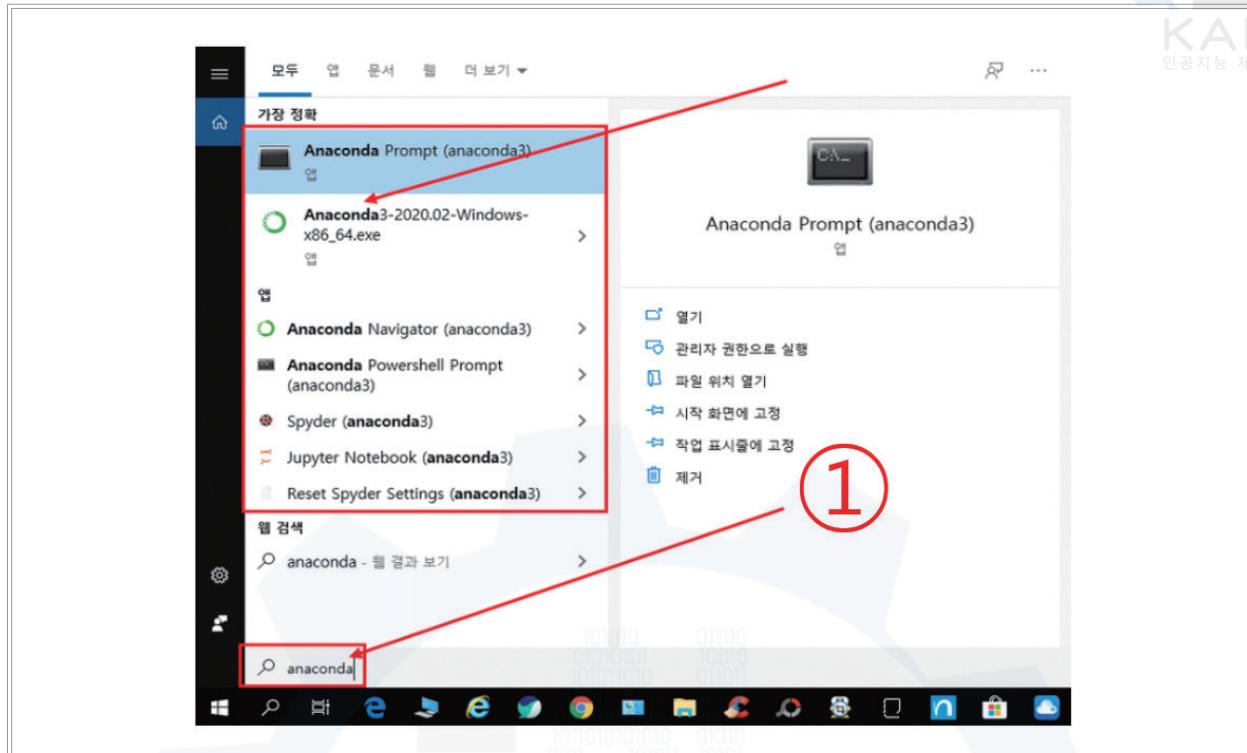




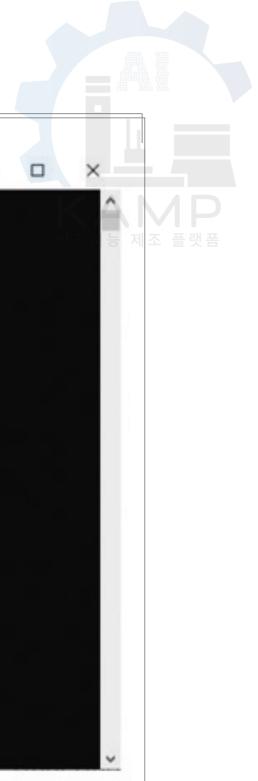


- ① 설치된 실행 파일을 열 때는 [마우스 오른쪽]-[관리자 권한으로 실행]을 클릭한다.
- ② 별도의 설정 없이 Next 버튼을 눌러 다음 단계로 진행한다.
- ③ 'Just me(recommended)'를 체크하고 넘어가서 다운로드 경로는 별도로 설정하지 않고 바로 넘어간다.
- ④ 체크박스를 모두 체크한 후 Install을 클릭한다.
- ⑤ 별도의 설정 없이 Next 버튼을 눌러 진행한다.
- ⑥ 두 가지 체크박스는 Anaconda와 관련된 튜토리얼과 교육자료에 관한 것을 설치 및 연결할 것인지 묻는 창이므로 체크를 전부 해제하고 Finish를 클릭한다.

① Anaconda 설치 확인



- ① 설치가 완료되면 Windows 좌측 하단의 돋보기를 클릭하여 Anaconda를 입력한 후 설치된 것을 확인한다.
- ② Anaconda를 통해 Python이 잘 설치되었는지 최종적으로 확인하기 위해 'cmd'를 입력한 후 '명령 프롬프트' 앱을 클릭한다.



```
명령 프롬프트 - python
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\woory\python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

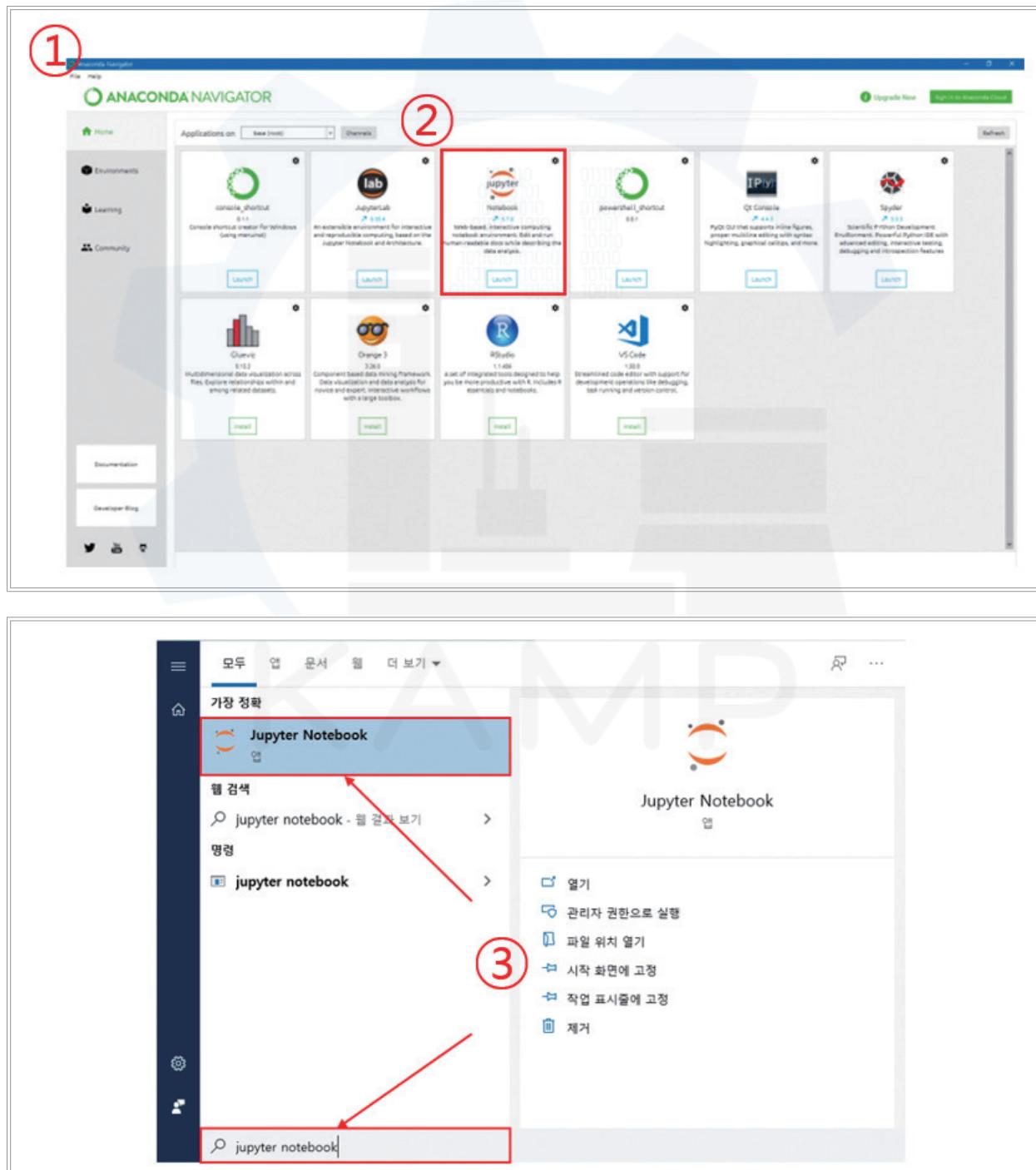
Type "help", "copyright", "credits" or "license" for more information.
>>> print('아나콘다 설치가 잘 되었습니다.')
아나콘다 설치가 잘 되었습니다.
>>>
```

- ① 'python'을 입력한 후 Enter를 눌러 Python이 실행되는 모습을 확인한다.
- ② 최종적으로 코드 실행이 정상적으로 작동하는지 확인하기 위해 '>>>' 옆에 'print('아나콘다
가 설치가 잘 되었습니다.')'를 입력하고 Enter를 눌러 해당 문자열이 그대로 화면에 출력되는
모습을 확인한다.

◎ Python 개발환경 : Jupyter Notebook

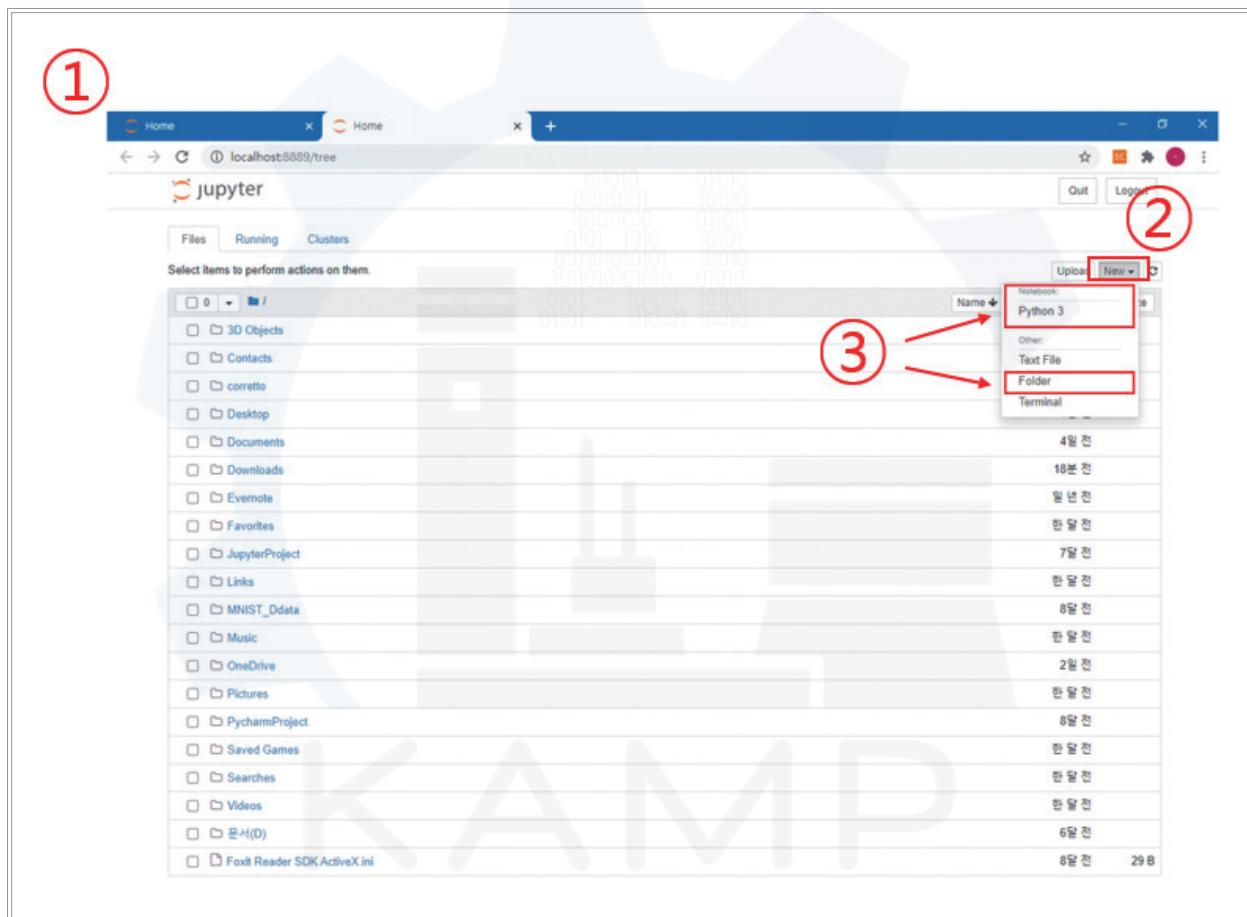
- Jupyter Notebook이란 ‘컴퓨터에게 Python 언어로 명령을 내렸을 때 즉각적인 응답이 오는 대화형 개발환경’이다. 여기서 말하는 ‘Notebook’은 일종의 ‘브라우저 웹페이지’ 형태이다. 즉, Jupyter Notebook이라는 웹페이지 상에서 Python 코드를 한 줄씩 입력하여 시행하고 분석 결과를 얻는다.

◎ Jupyter Notebook 설치



- ① Anaconda Navigator를 검색 후 실행한다.
- ② Jupyter Notebook이 'Launch' 상태인지 확인한다.
 (※ 'Install' 상태일 경우, 클릭하여 설치해야 한다.)
- ③ Windows 화면 좌측 하단의 돋보기 버튼을 눌러 'jupyter notebook'을 입력하고 실행 앱이 있는지 확인한다.
 (※ Anaconda Navigator를 매번 실행하지 않아도 Jupyter Notebook 앱을 통해 곧바로 실행이 가능하다.)

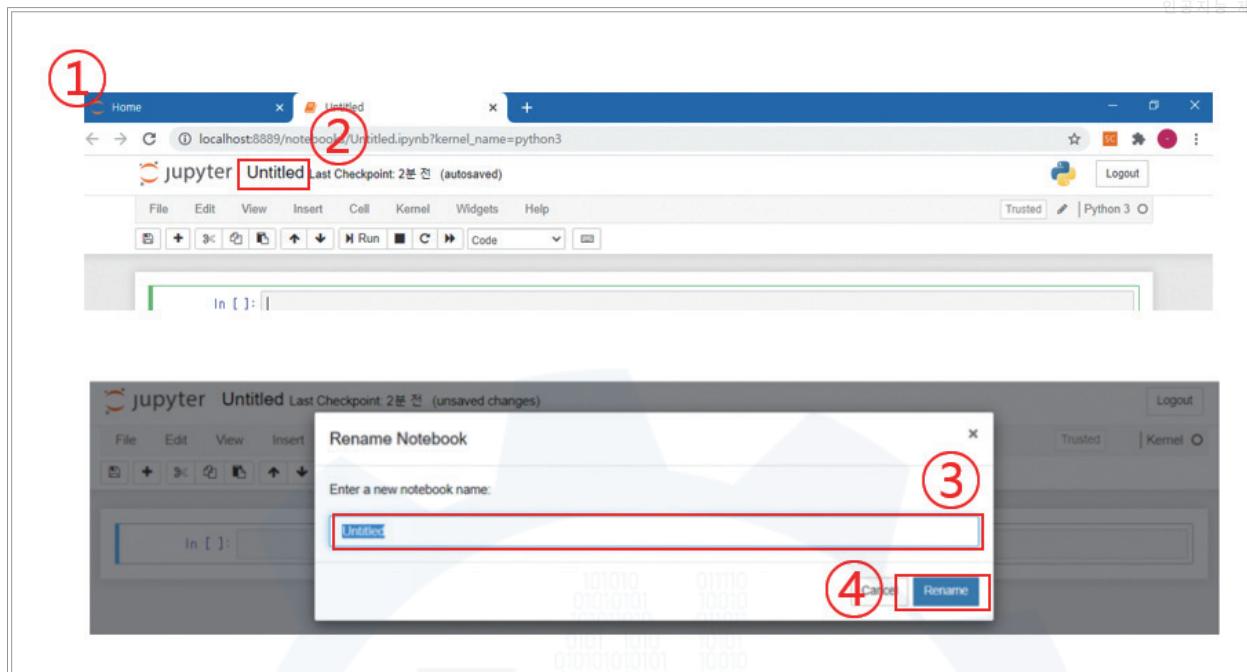
◉ Jupyter Notebook 실행 후 Notebook 생성



- ① Jupyter Notebook 앱을 클릭하여 웹페이지가 실행되는 것을 확인한다.
- ② notebook을 하나를 생성하기 위해 'New' 버튼을 클릭한다.
- ③ 현재 위치에 코드 파일 notebook을 바로 생성하려면 'Python 3'을 클릭하고, 새로운 폴더를 생성 후 그 안에 생성하려면 'Folder'를 클릭한다.
 (※ 새로운 폴더 생성 후 진행할 경우, 그 폴더 안에서 [New]-[Python 3]를 동일하게 클릭한다.)

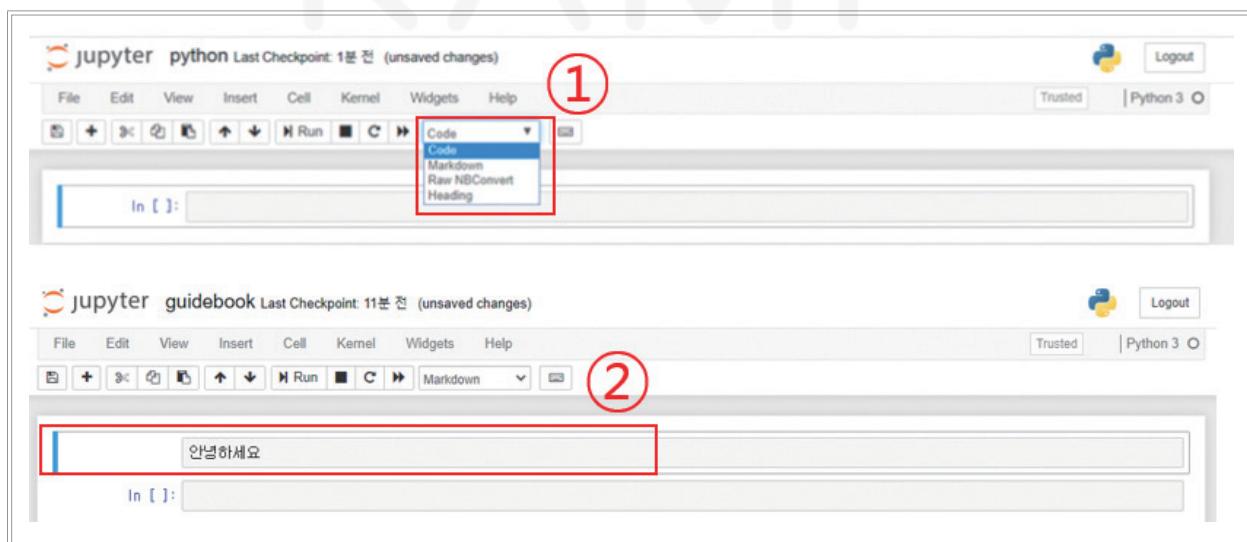
◉ Notebook 기본 활용법 : 제목 설정 및 주석 사용

- 먼저 제목을 설정할 경우 다음과 같은 과정을 거친다.



- ① 새롭게 생성된 notebook을 연다.
- ② 처음 생성한 notebook의 제목을 확인하고 수정할 경우 제목을 두 번 클릭한다.
- ③ Rename Notebook 창에 수정하고자 하는 notebook의 제목을 입력한다.
- ④ 수정을 완료하기 위해 'Rename' 버튼을 클릭한다.

- 다음으로 코드 내에 주석(설명)을 기입할 경우 다음과 같은 과정을 거친다. ‘마크다운(Markdown)’을 이용하여 코드 실행 줄을 메모장 형식으로 변형하거나 이러한 변형 없이 코드를 입력한 실행 줄 내에 곧바로 기입할 수 있다.



```
jupyter guidebook Last Checkpoint: 11분 전 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 Logout
In [1]: 안녕하세요
NameError: Traceback (most recent call last)
<ipython-input-1-041a565a82b3> in <module>
      1 안녕하세요
    NameError: name '안녕하세요' is not defined
```

```
jupyter guidebook Last Checkpoint: 13분 전 (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3 Logout
In [1]: 안녕하세요
NameError: Traceback (most recent call last)
<ipython-input-1-041a565a82b3> in <module>
      1 안녕하세요
    NameError: name '안녕하세요' is not defined
In [2]: #안녕하세요
In [ ]:
```

- ① 마크다운을 사용하기 위해 코드 실행 줄 하나를 선택해두고 ‘Code’로 설정되어 있는 버튼을 클릭하여 ‘Markdown’을 선택한다.
- ② 메모장 형식으로 코드 실행 줄이 변경된 것을 확인하고 주석 내용을 입력하여 Shift+Enter 키를 눌러 주석을 설정한다.
- ③ 마크다운으로 변경하지 않은 코드 실행 줄에 주석을 입력할 경우 오류가 발생하는 것을 확인 할 수 있다.
- ④ 마크다운 변경 없이 주석을 기입할 경우 ‘#’을 먼저 입력한 후 그 뒤에 설명을 기입한다.

◉ Notebook 기본 활용법 : 단축키

- 자주 활용하는 단축키는 해당 코드 실행 줄을 클릭해놓은 상태에서 활용한다.

- ① Shift + Enter : 해당 코드 줄을 실행(Run)한다.
- ② m : 해당 코드 줄을 마크다운으로 자동 변경한다.
- ③ y : 해당 마크다운 줄을 코드 줄로 자동 변경한다.
- ④ a : 바로 위에 코드 줄을 추가한다.
- ⑤ b : 바로 아래에 코드 줄을 추가한다.
- ⑥ x : 해당 코드 줄을 삭제한다.

◎ Python installation – Anaconda

다운로드: <https://www.anaconda.com/products/individual>
 Download > Window python 3.8.X(or version of python 3.x.x)
 > 64-Bit Graphical Installer or 32-Bit Graphical Installer



[그림 103] Anaconda

PC사양 확인 후 (내 PC > 우클릭 > 속성 > 시스템 종류 확인)



[그림 104] 로컬 PC사양 확인

(최초 실험 환경: Python 3.7.3 runtime interpreter)

설치 후 응용 프로그램 Anaconda Prompt 확인

◎ CUDA 다운로드: <https://developer.nvidia.com/cuda-toolkit>

◎ CUDA 설치 과정 1 – CUDA 설치 홈페이지

위 홈페이지에 접속하여 “Download Now”를 클릭한다.

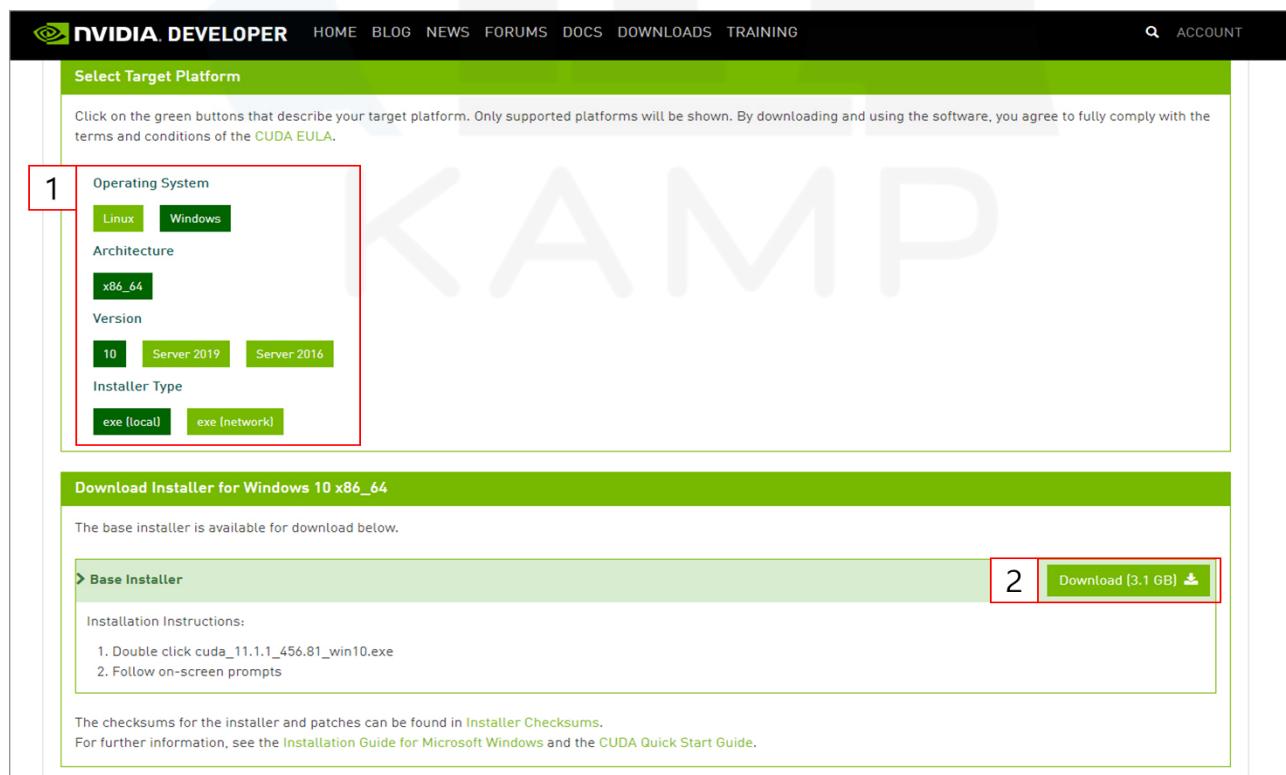


The screenshot shows the official NVIDIA CUDA Toolkit landing page. At the top, there's a banner for Jensen Huang's GTC Keynote. Below it, a navigation bar includes 'Home', 'High Performance Computing', and 'CUDA Toolkit'. The main content area is titled 'CUDA Toolkit' and describes it as a development environment for GPU-accelerated applications. It highlights support for various architectures like x86, Arm, and POWER, and mentions built-in capabilities for distributing computations across multiple GPUs. A large green 'Download Now' button is centered at the bottom of the main text block.

[그림 105] CUDA 홈페이지의 다운로드 위치

◎ CUDA 설치 과정 2 – CUDA 다운로드

위 “Download Now”를 클릭 후 로컬 PC에 맞는 조건을 입력하여 해당 CUDA 설치 파일을 다운 받는다.

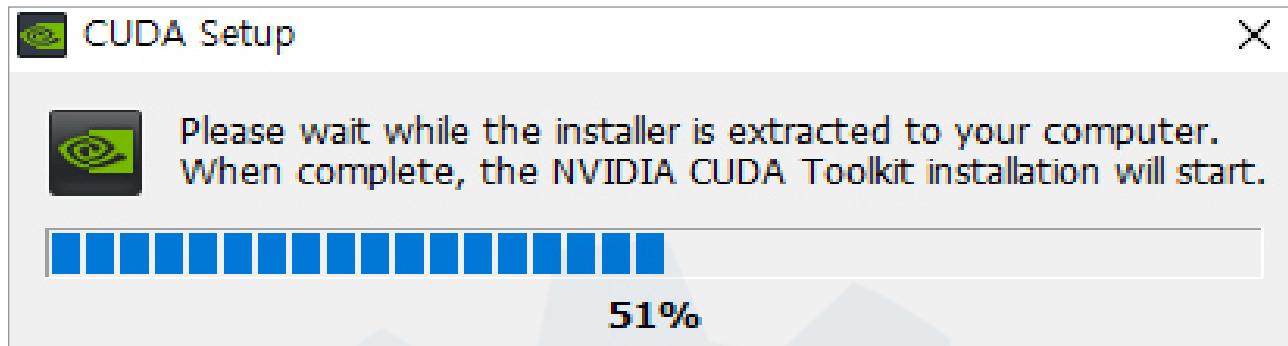


This screenshot shows the NVIDIA Developer website's CUDA download interface.
 Step 1 (left side): A red box highlights the 'Select Target Platform' section where users can choose their operating system (Linux or Windows), architecture (x86_64), version (10, Server 2019, Server 2016), and installer type (exe (local) or exe (network)).
 Step 2 (right side): A red box highlights the 'Download Installer for Windows 10 x86_64' section, which displays the 'Base Installer' and its download link ('Download [3.1 GB]').
 Below the download link, there are installation instructions and checksum information.

[그림 106] 로컬 PC 조건에 맞게 다운받을 CUDA 설정 예시

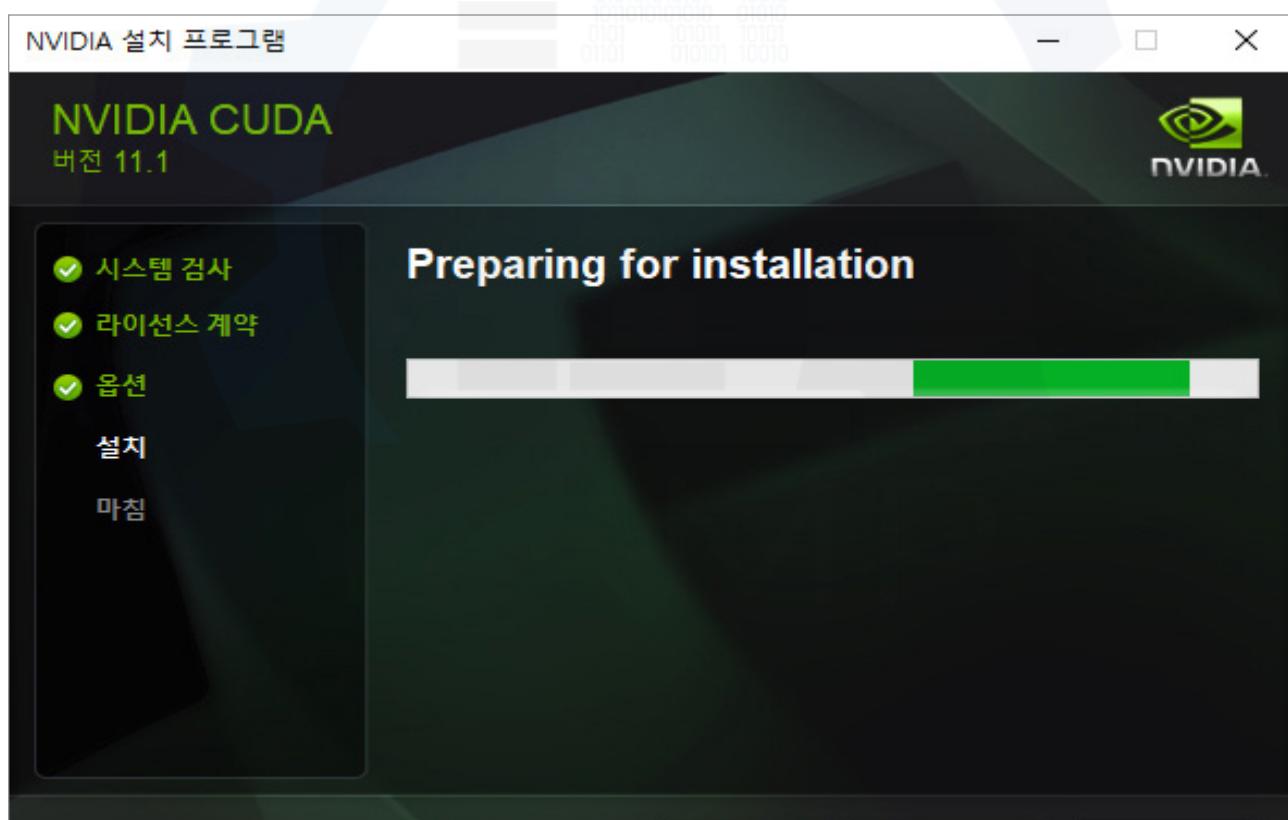
분석 실습을 수행하고자 하는 로컬 PC 사양에 맞게 CUDA를 다운받는다. PC의 사양은 내 PC(혹은 내 컴퓨터) 아이콘에 마우스 우클릭 후 “속성”을 누르면 확인 가능하다. ([그림 104] 참조)

◎ CUDA 설치 과정 3



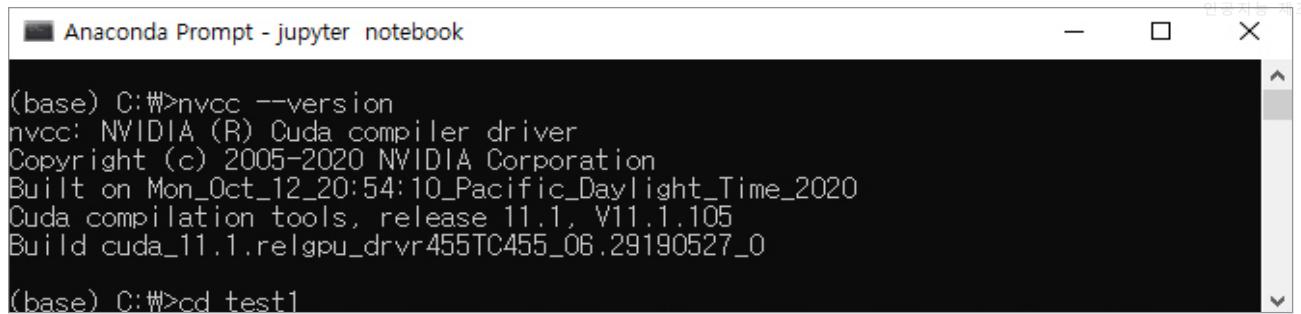
[그림 107] CUDA 설치 진행 과정

◎ CUDA 설치 과정 4



[그림 108] CUDA 설치 진행 과정

- CUDA의 설치가 완료되면 Anaconda prompt창에 “nvcc--version”입력하여 설치 여부와 CUDA 버전을 확인한다.

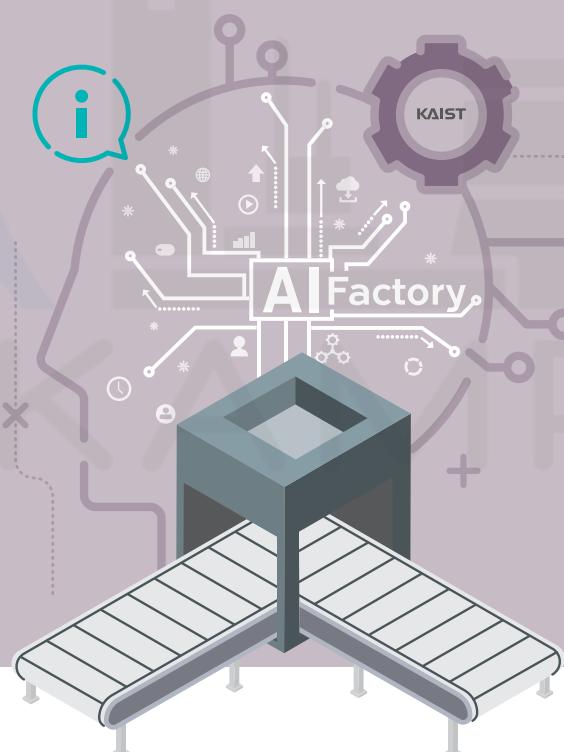


```
(base) C:\>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Oct_12_20:54:10_Pacific_Daylight_Time_2020
Cuda compilation tools, release 11.1, V11.1.105
Build cuda_11.1.relgpu_drvr455TC455_06.29190527_0

(base) C:\>cd test1
```

[그림 109] 로컬 PC사양 확인

「X-ray 검사장비 AI 데이터셋」 분석실습 가이드북



중소벤처기업부



스마트제조혁신추진단



34141 대전광역시 유성구 대학로 291 한국과학기술원(KAIST)
T. (042)350-2114 F. (042)350-2210(2220)