

종합설계 프로젝트 수행 보고서

| | |
|-------|--|
| 프로젝트명 | 딥러닝을 적용한 CFRP 가공 결함 분석 |
| 팀 번호 | S2-6 |
| 문서 제목 | 수행계획서 () 2차발표 중간보고서 (○) 3차발표 중간보고서 () 최종결과보고서 () |

팀원 : 이형석 (조장)

최성훈 (팀원)

조현민 (팀원)

황규빈 (팀원)

지도교수 : 전광일 (인)

문서 수정 내역

| 작성일 | 대표작성자 | 버전(Revision) | 수정내용 | |
|------------|---------|--------------|-------|------|
| 2021.03.03 | 이형석(팀장) | 1.0 | 수행계획서 | 최초작성 |

문서 구성

| 진행단계 | 프로젝트 계획서 발표 | 중간발표1 (2월) | 중간발표2 (4월) | 학기말발표 (6월) | 최종발표 (10월) |
|------------|---|---|---|---|----------------|
| 기본양식 | 계획서 양식 | 계획서 양식 | 계획서 양식 | 계획서 양식 | 계획서 양식 |
| 포함되는 내용 | I. 서론 (1~6) II. 본론 (1~3) 참고자료 | I. 서론 (1~6) II. 본론 (1~4) 참고자료 | I. 서론 (1~6) II. 본론 (1~5) 참고자료 | I. 서론 (1~6) II. 본론 (1~7) 참고자료 | I II III |

이 문서는 한국산업기술대학교 컴퓨터공학부의
 “종합설계” 교과목에서 프로젝트
 “딥러닝을 적용한 CFRP 가공 결함 분석” 을 수행하는
 (S2-6 이형석, 조현민, 최성훈, 황규빈)들이 작성한 것으로
 사용하기 위해서는 팀원들의 허락이 필요합니다.

목 차

I . 서론

| | |
|------------------------|--------|
| 1. 작품선정 배경 및 필요성 | - 4 - |
| 2. 기존 연구/기술동향 분석 | - 5 - |
| 3. 개발 목표 | - 6 - |
| 4. 팀 역할 분담 | - 8 - |
| 5. 개발 일정 | - 9 - |
| 6. 개발 환경 | - 10 - |

II . 본론

| | |
|--------------------|--------|
| 1. 개발 내용 | - 11 - |
| 2. 문제 및 해결방안 | - 12 - |
| 3. 시험시나리오 | - 13 - |
| 4. 상세 설계 | - 14 - |
| 5. 프로토타입 | - 21 - |

| | |
|------------|--------|
| 참고자료 | - 36 - |
|------------|--------|

I . 서론

1. 작품선정 배경 및 필요성

| 기획 의도 | 내용 |
|----------------------------|---|
| 저비용의 CFRP 가공 결함 검출 | <ul style="list-style-type: none">- 항공, 우주, 자동차 등의 산업에 여러 가지 장점을 갖는 신소재로 CFRP가 떠오르고 있다.- CFRP 소재는 가공이 어려워 결함이 자주 발생하는데 CFRP 소재의 결함을 검출하는 장비는 매우 고가이기 때문에 규모가 작은 기업에서는 저비용의 결함 검출 방법이 필요하다. |
| 육안으로 결함을 검출하는 방법보다 더 빠른 방법 | <ul style="list-style-type: none">- 고가의 장비를 이용하지 않고 육안으로 검사하는 방법으로 결함을 검출한다면 상대적으로 많은 시간이 필요함.- 가공 후 즉시 결함 검출을 진행한다면 육안으로 검출하는 방법보다 훨씬 더 빠른 검사 진행이 가능하다. 때문에 가공 후 카메라를 이용한 결함 검출을 진행해 검사 시간을 줄이기 위함이다. |

2. 기존 연구/기술동향 분석

| 항목 | 내용 |
|---------|--|
| 기존 연구 | <ul style="list-style-type: none"> ■ 한국생산기술연구원 <ol style="list-style-type: none"> 1. 광학 스캐너를 활용해 가공된 부품 표면과 내부 불량을 1초 만에 파악하는 ‘3D 광학 고속 검사기술’을 세계 최초로 개발 ■ 코그넥스 <ol style="list-style-type: none"> 2. 자동화 프로세스를 위한 머신비전 시스템, 소프트웨어, 표면 검사 등을 제조 및 판매했다. ■ 다트비전 <ol style="list-style-type: none"> 3. 반도체, 디스플레이, 모바일 및 제조업과 다양한 분야에서 머신비전 솔루션을 공급하고 있다. ■ 엔비전 <ol style="list-style-type: none"> 4. 형상 및 결함 추적과 표면 검사, 색상 검사 머신비전 기술지원 서비스를 제공하고 있다. 5. XGS 센서 기반 카메라로 수치 측정과 같은 애플리케이션에 활용 |
| 기술동향 분석 | <ul style="list-style-type: none"> ● 딥러닝 기반 이미지 분석은 본질적으로 복잡한 성형 표면 및 성형 결함을 처리하는 데 탁월한 성능을 발휘한다. ● 머신비전 시스템은 제조된 제품에서 결함, 오염, 기능상 결점, 기타 이상 현상을 감지하는 검사를 수행하도록 컴퓨터를 트레이닝 했다. ● 기존의 규칙 기반 알고리즘으로는 처리하기 불가능하거나 어려운 복잡한 검사, 분류 및 위치 지정 애플리케이션을 해결한다. ● 국방, 항공 우주, 연구, 생체인식, 의료 등에서 머신비전을 활용하여 인간의 눈을 대신하고 있다. |

3. 개발 목표

| 항목 | 내용 |
|--------|--|
| 서비스 개발 | <ul style="list-style-type: none"> ● 웹, 앱을 통한 CFRP 가공 이미지 결함 검출 결과 확인 ● 핸드폰 카메라를 통한 간이 이미지 결함 검출 앱 개발 ● 검출된 결함에 대한 Box 표시 후 재가공 판단 |
| 웹 개발 | <ul style="list-style-type: none"> ● 페이지 관리자용 웹 개발 ● 사용자 결함 검출 이미지 확인용 웹 개발 ● 사용자 이미지 업로드 웹 개발 |
| 서버 개발 | <ul style="list-style-type: none"> ● 웹과 딥 러닝 프로그램 사이 이미지 전달 서버 개발 |
| 개발 방향 | <ul style="list-style-type: none"> ● 빠른 프로토타입 출시를 통해 애자일 방법론을 따라서 추가적인 요구 사항에 대처 ● 개발 시 선행 지식이 부족한 딥 러닝 Yolo v5 알고리즘을 클라우드 서비스에서 구현 ● 사용되거나 표시될 이미지의 서버와 프로그램 간에 송수신에 초점을 맞춘다. ● 데이터 송수신 이후에 웹과 앱의 UI에 대해 초점을 맞춘다. |

4. 팀 역할 분담

| 팀원 | 역할 |
|-----|---|
| 이형석 | <ul style="list-style-type: none"> - 프론트(웹 페이지) 구현 <ol style="list-style-type: none"> 1. 실시간 검출 히스토리 노출 - 서버 구현 <ol style="list-style-type: none"> 1. AWS EC2 서버 구축 |
| 조현민 | <ul style="list-style-type: none"> - 이미지 전처리 <ol style="list-style-type: none"> 1. YOLO MARK를 활용한 이미지 라벨링 |
| 최성훈 | <ul style="list-style-type: none"> - 알고리즘 모델 구현 <ol style="list-style-type: none"> 1. Yolo v5 기법 구현 및 활용 |
| 황규빈 | <ul style="list-style-type: none"> - 알고리즘 모델 구현 <ol style="list-style-type: none"> 1. Mask R CNN 기법 구현 및 활용 - 프론트(어플리케이션) 구현 <ol style="list-style-type: none"> 1. 이미지 업로드 기능 구현 2. 결과 반환 및 노출 |

4.1 팀원간 소통방법

개발 방법론

- 점진적 개발의 장점을 살리면서, 요구사항의 변화를 주기적으로 수용하는 애자일 방식 활용
- 2주 단위로 기존에 계획한 요구사항과 전 스프린트에서 나온 변경사항을 비교하고, 검토를 거쳐 우선순위화하여 반영
- 분석, 설계, 구현, 테스트 같은 공정들을 동시에 접근하여 중간 산출물을 최소화하면서 품질을 높임

프로젝트 진행관리

- 스프린트 관리를 위해 Trello 도구를 활용
- 협업 및 버전 관리를 위해 GitHub를 활용
- 팀원간 커뮤니케이션을 위해 Slack을 활용


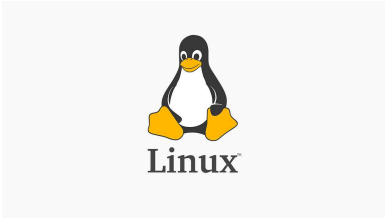
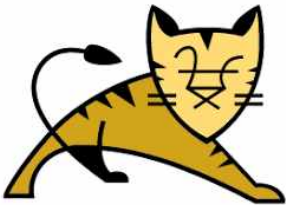


팀 룰



- 매주 수요일 스프린트 회의를 진행, 회의 간 프로젝트 진행 상태 및 문제점 파악

5. 개발 일정

| 항목 | 추진사항 | 12월 | 1월 | 2월 | 3월 | 4월 | 5월 | 6월 | 7월 | 8월 |
|-----------------------|-------------------------------------|-----|----|----|----|----|----|----|----|----|
| 사전 조사 및 요구사항 분석 | 사전조사 및 요구사항 분석 | | | | | | | | | |
| 시스템 설계 및 상세 설계 | 데이터 학습에 사용될 알고리즘 선정 | | | | | | | | | |
| | 웹 구조 설계 | | | | | | | | | |
| | 모바일 앱 구조 설계 | | | | | | | | | |
| | DB 설계 | | | | | | | | | |
| 데이터 수집 및 딥러닝 구현 | 결함이 있는 CFRP 소재의 이미지 데이터 수집 | | | | | | | | | |
| | 데이터 라벨링 | | | | | | | | | |
| | 라벨링된 데이터를 기반으로 학습 | | | | | | | | | |
| | 결함 검출 정확도 향상 | | | | | | | | | |
| 웹 및 모바일 앱 구현 | 모바일 UI | | | | | | | | | |
| | 모바일 앱 서버 업로드 및 검출 결과 출력 구현 | | | | | | | | | |
| | 웹 요청에 대한 검출 결과 응답 구현 | | | | | | | | | |
| | 검출 데이터 관리 기능 | | | | | | | | | |
| 테스트 | 기능 단위 테스트 | | | | | | | | | |
| 문서화 | 최종 보고서 작성 | | | | | | | | | |

6. 개발 환경

| | |
|------------------------|---|
| Server |  <p>Amazon EC2</p> <ul style="list-style-type: none"> ● Deep Learning AMI (Amazon Linux 2) Version 38.0 – p2.xlarge |
| OS |  <ul style="list-style-type: none"> ● Linux |
| Web Application Server |  <ul style="list-style-type: none"> ● Tomcat – 9.0.41 |
| Client OS |  <ul style="list-style-type: none"> ● Window 10 |
| 사용 도구 |  <ul style="list-style-type: none"> ● Chrome |

| | |
|--------------------|--|
| 개발 도구 및 언어 |  <ul style="list-style-type: none"> ● PyCharm community edition – 2020.1.3 |
| Web Server 개발환경 |  <ul style="list-style-type: none"> ● Spring Tool Suite 4 – 4.9.0 |

II. 본론

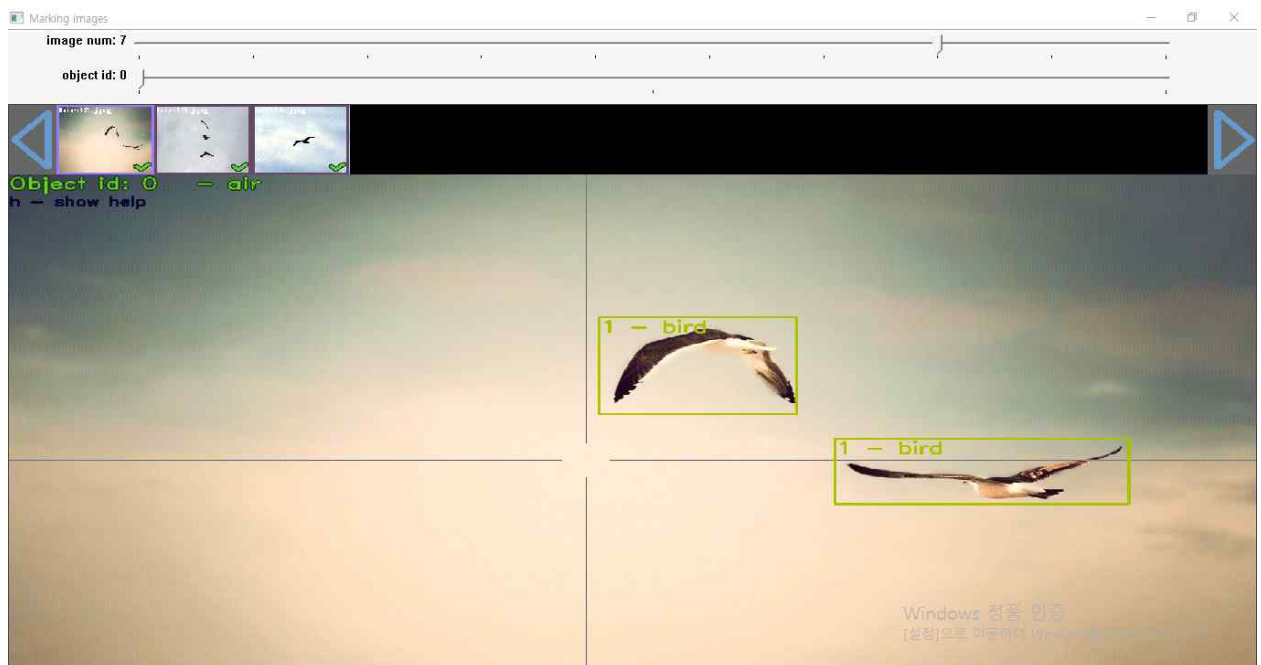
1. 개발 내용

1) Deep Learning Algorithm

- YOLO 알고리즘을 사용하기 위해서 먼저 라벨링을 위한 개발환경의 구축이 필요하다.
- Anaconda3, Python(3.5 Version이상), Pytorch, CUDA(9.0) OpenCV를 설치한다.

2) YOLO_Mark

- YOLO_Mark 프로그램을 이용하여 사진의 결함부분을 박스처리하고 해당 좌표를 txt파일로 생성한다.
- [그림1-1]은 예를 들어 새의 사진에 대해 라벨링을 할 때 마우스 드래그를 통해 박스처리를 한다. 이와 같은 방법으로 Training 이미지에 각각 박스처리하고 라벨링을 하여 좌표값을 저장한다.



[그림1-1]

3) YOLO_Mark 사용설명서

- Mouse Control

| Button | Description |
|--------|-------------|
| Left | Draw box |
| Right | Move box |

- 키보드 단축키

| Short Cut | Description |
|-----------|---------------------|
| → | 다음 이미지 |
| ← | 이전 이미지 |
| r | 선택한 상자 삭제(마우스를 가리킴) |
| c | 현재 이미지의 모든 표시 지우기 |
| p | 이전 마크 복사 |
| o | 개체 추적 |
| ESC | 응용프로그램 닫기 |
| n | 이미지 당 하나의 개체 |
| 0~9 | 개체 ID |
| m | 좌표 표시 |
| w | 선의 폭 |
| k | 개체 이름 숨기기 |
| h | 도움 |

4) YOLOv5

- YOLOv5의 training함수에서 YOLO_Mark로 라벨링한 좌표 데이터를 파라미터로 전달받아 학습시킨 데이터 .weight파일을 만든다.
- YOLOv5의 detection함수에서 학습된 weight파일을 통해 이미지를 전달 받으면 결함 부분을 검출해서 보여준다.

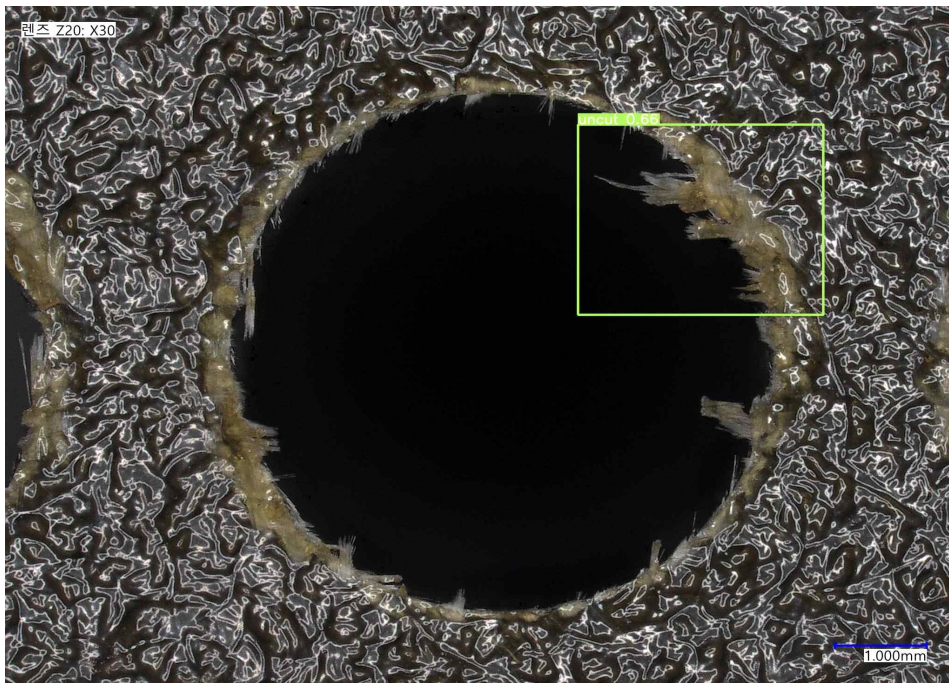
5) YOLOv5의 프로토타입

- 먼저 프로토타입으로 직접 라벨링한 500장 가량의 사진을 통해 학습을 시켰다.
- [그림2-1]은 학습된 이미지를 바탕으로 검출을 시도 하였는데 가공 결함이 나타나지 않고 정상 적으로 가공되었다는 결과가 나왔다.



[그림2-1]

- [그림2-2]는 검출 시도를 하였는데 미절삭이 발견되어 결함부분에 박스처리 된 결과가 나왔다.

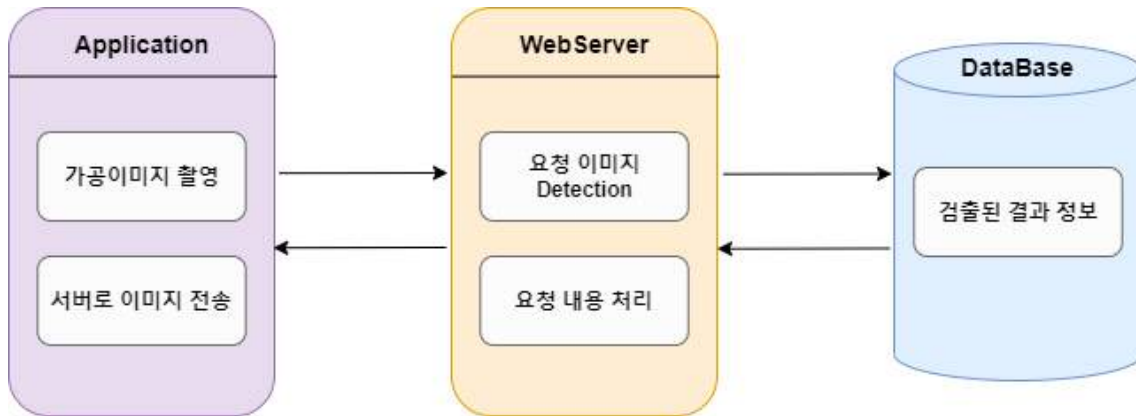


[그림2-2]

6) Application - Server 구성

- [그림3-1]에서 Application의 역할은 실제 가공 공장에서 사용하는 기계를 도입할 여건이 안되기 때문에 이 공정 과정을 애플리케이션을 통해 가상으로 동작을 보여주기 위해 APP을 도입하였다.

- APP을 통해 촬영한 이미지를 WebServer로 전송하면 서버에서 해당 이미지의 결함 여부를 판단하고 결과를 저장 및 알려주는 역할을 한다.

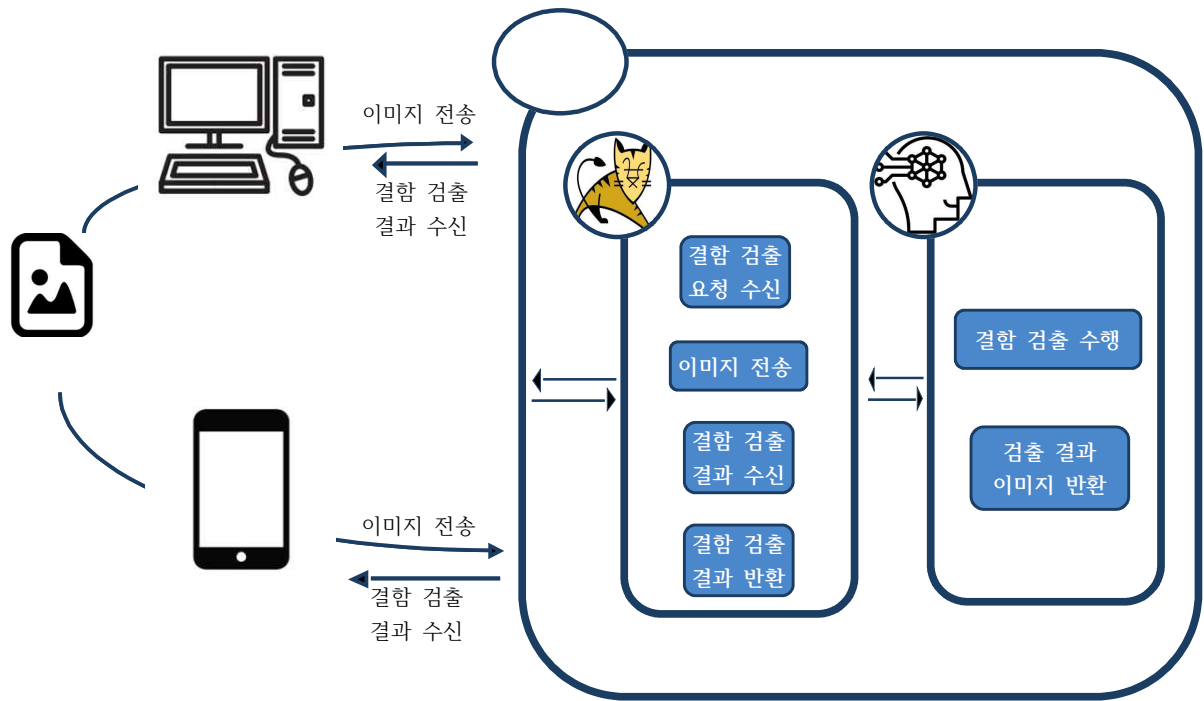


[그림3-1]

2. 문제 및 해결방안

- 부족한 Training Data로 인하여 작은 사이즈의 결함부분은 잡아내지 못하여 낮은 결함 검출률을 보인다.
 - 해결방안 : 현재 보유하고 있는 이미지의 양을 늘리기 위해 전처리를 하여 더 많은 Training Data를 만들어 다시 학습시키면 높은 결함 검출률이 나올 것이라고 기대된다.
- 더 높은 검출율과 빠른 응답시간을 구축하기 위해 여러 가지 머신러닝 기법들을 사용해보고 비교 분석하고 있다. 여기서 실제 머신러닝을 사용하는 회사의 전문가에게 피드백을 받은 기법 중 Mask RCNN의 기법이 있는데 이 기법은 라벨링을 할 때 결함 부분을 정확하게 라인을 그려야 하지만 우리가 검출하고자 하는 결함들의 경계선이 모호하여, 학습을 위해 이미지 라벨링을 하는 과정에서 영역 선정에 어려움을 겪고 있습니다
 - 해결방안 : 위의 문제에 대해 전문가에게 피드백 요청 또는 Mask RCNN처럼 매우 정확한 결함 위치가 도출 되어야 검출에 성공한 것인지 아니면 어느 정도 오차가 있어도 허용이 되는지 만약 오차가 있어도 된다면 허용 범위는 어느 정도인지 한국생산기술연구원의 조언을 구할 예정

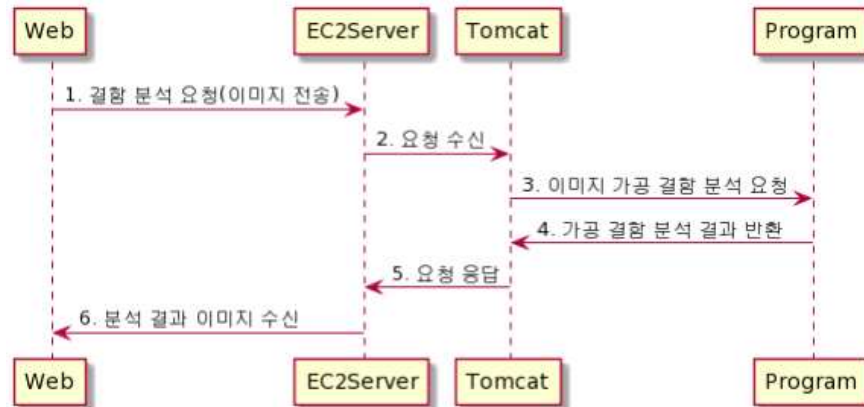
3. 시험 시나리오



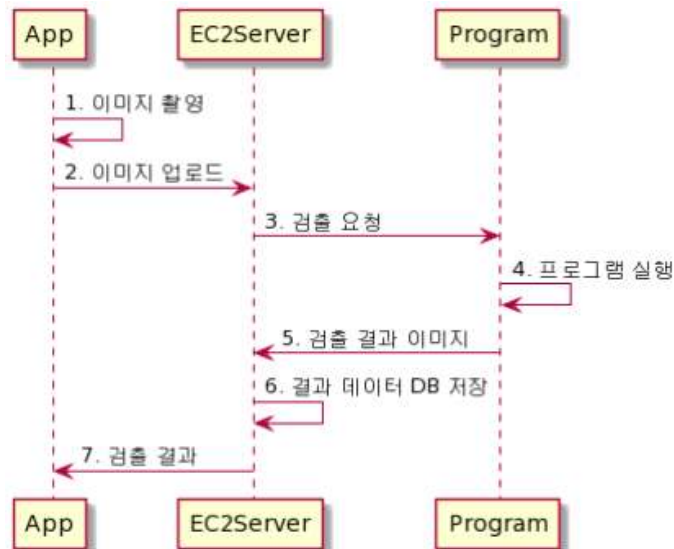
| 시나리오 명 | 내용 |
|------------|--|
| 서버 | <ul style="list-style-type: none"> - 모바일 앱의 요청을 처리. - 검출 프로그램을 사용해 업로드된 이미지의 결함을 검출하고 모바일 앱으로 응답을 전송. - 검출 결과를 데이터베이스에 저장 |
| 웹 | <ul style="list-style-type: none"> - 데이터베이스의 검출 결과를 바탕으로 분석, 통계 및 관리 페이지를 보여준다. |
| 모바일 앱 | <ul style="list-style-type: none"> - 카메라를 이용해 검출 대상 이미지를 얻고 서버에 검출 요청. - 서버에서 보낸 검출 결과를 수신 받고 화면에 출력. |
| 결함 검출 프로그램 | <ul style="list-style-type: none"> - 요청 받은 이미지의 결함을 검출해 이미지를 출력. |

4. 상세 설계

4.1 시퀀스 다이어그램



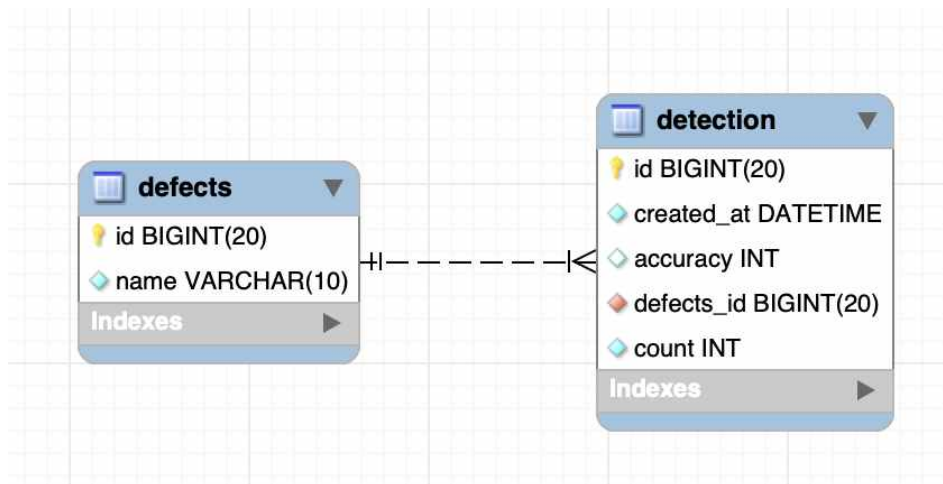
| 순서 | 내용 |
|----|--|
| 1 | 웹 페이지에서 이미지를 업로드 받아, 서버로 결함 분석을 요청 |
| 2 | 요청을 받은 서버는, 웹에서의 요청을 수행하기 위해 Tomcat으로 전달 |
| 3 | Tomcat에서 요청과 함께 전달된 이미지를, 분석을 위해 프로그램으로 전달 |
| 4 | 가공 결함 검출 결과를 Tomcat으로 응답 |
| 5 | 결함 검출 결과 요구에 대한 응답 |
| 6 | 서버는 결함 결과를 Web으로 응답 |



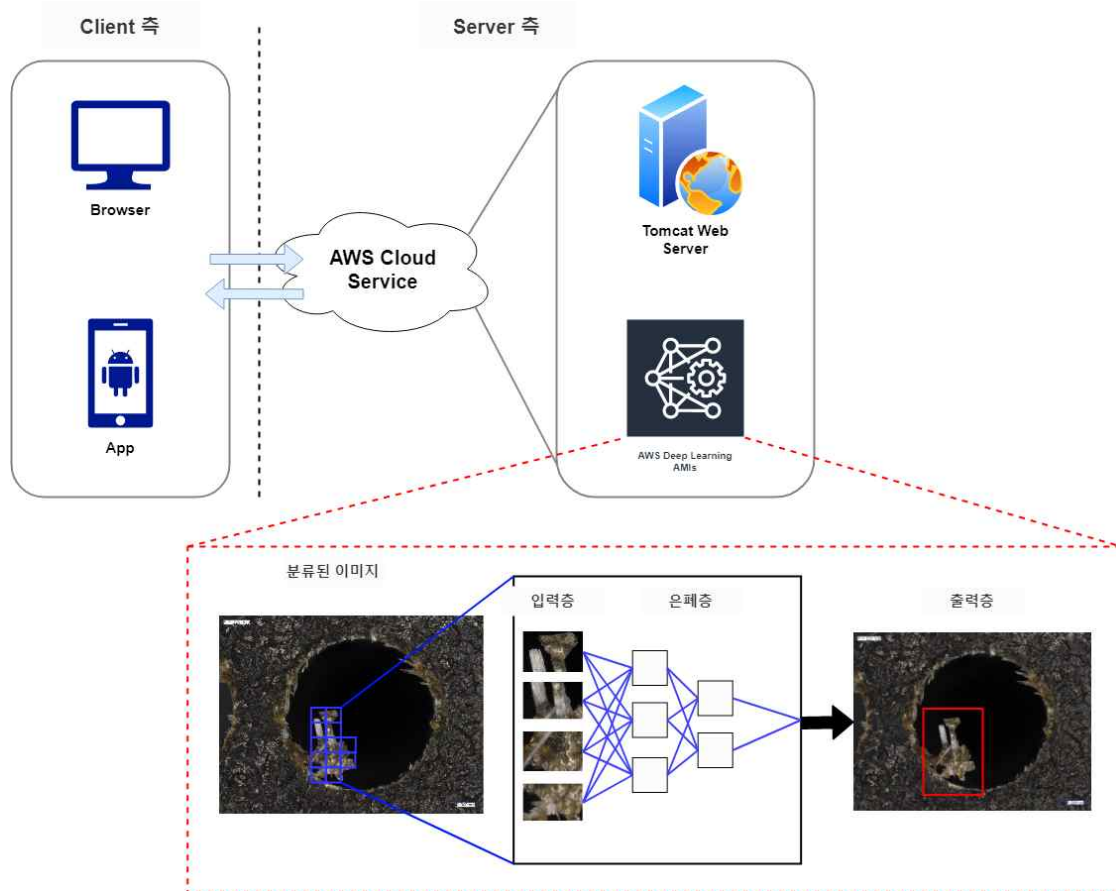
| 순서 | 내용 |
|----|------------------------------|
| 1 | 모바일 앱의 카메라를 이용해 결함 검출 대상을 촬영 |
| 2 | 모바일 앱에서 웹 서버로 이미지를 전송 |
| 3 | 서버는 결함 검출 프로그램을 실행 |

| | |
|---|---|
| 4 | 이미지에 결함 여부를 확인해 결함이 있다면 결함 부분에 바운더리 박스를 만들어 새로운 이미지를 생성 |
| 5 | 결함 검출 프로그램의 결과 이미지를 서버에 저장 |
| 6 | 서버는 결과 데이터를 데이터베이스에 저장 |
| 7 | 결함이 검출된 이미지와 검출 결과 데이터를 모바일에 응답 |

4.2 ER 다이어그램



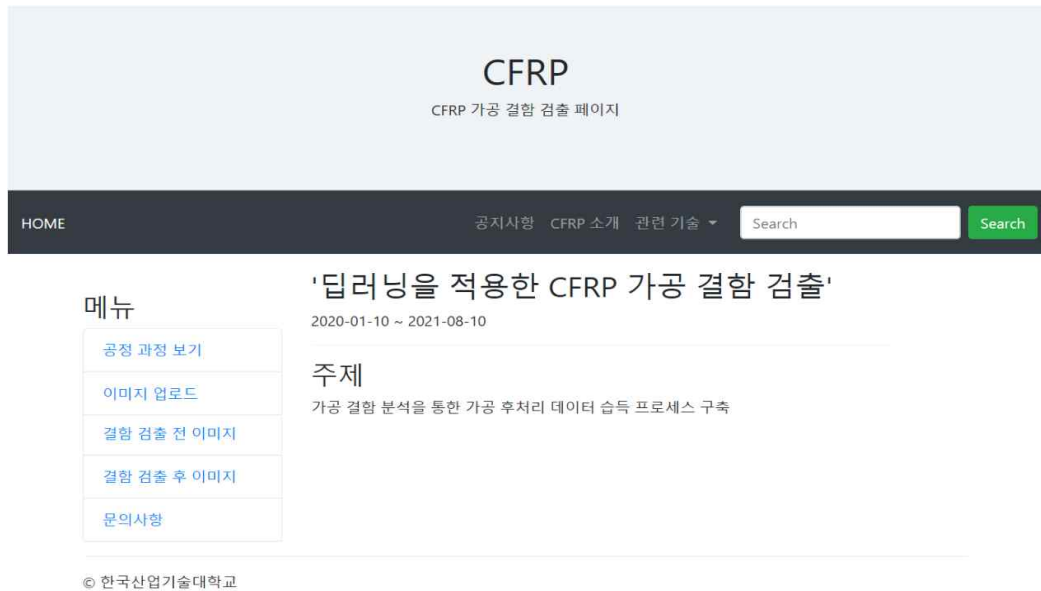
4.3 시스템 구성도



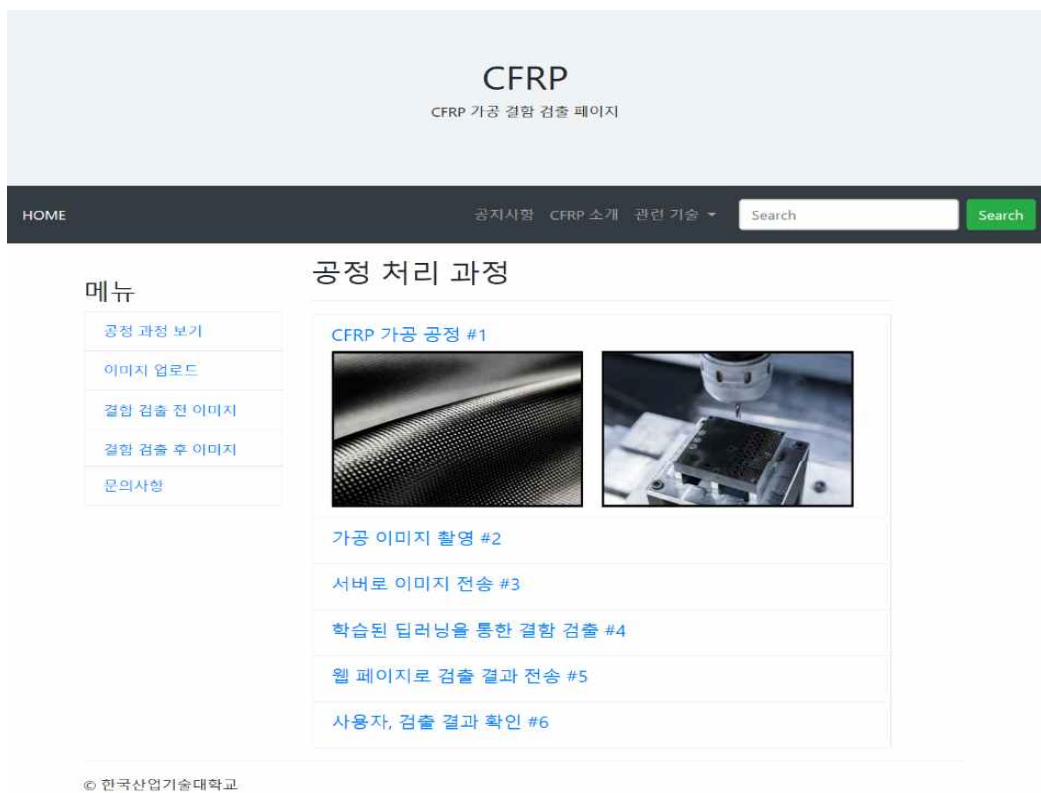
| 항목 | 내용 |
|----------------------|--|
| 웹 페이지 준비 | <ul style="list-style-type: none"> ● Apache Tomcat 9.0 서버에서 사용자에게 제공할 JSP 페이지 준비 |
| 앱 준비 | <ul style="list-style-type: none"> ● Android Studio를 통해 사용자가 이미지를 송신하고 수신받을 애플리케이션 준비 |
| AWS Cloud Service 준비 | <ul style="list-style-type: none"> ● AWS Cloud Server에는 Tomcat Server와 Deep Learning Server가 존재 ● Tomcat Server를 통해 웹 페이지 서비스 제공 ● Deep Learning Server를 통해 전송 받은 이미지 결함 검출 후 사용자에게 결과 전송 |
| 딥러닝 알고리즘 검출률 확인 | <ul style="list-style-type: none"> ● CNN Model의 Yolo v5 알고리즘으로 이미지 결함 검출 후 결함 부분에 Boundary Box 표시 |
| 결과 | <ul style="list-style-type: none"> ● CNN Yolo v5 알고리즘을 통한 딥러닝 훈련으로 낮은 검출률이지만 미절삭 섬유와 박리현상이 정확히 구분됨 ● 이미지를 송수신하기 위한 애플리케이션 구현, 내부 데이터 송수신 로직은 추후에 구현 |

☑ 웹 페이지 준비

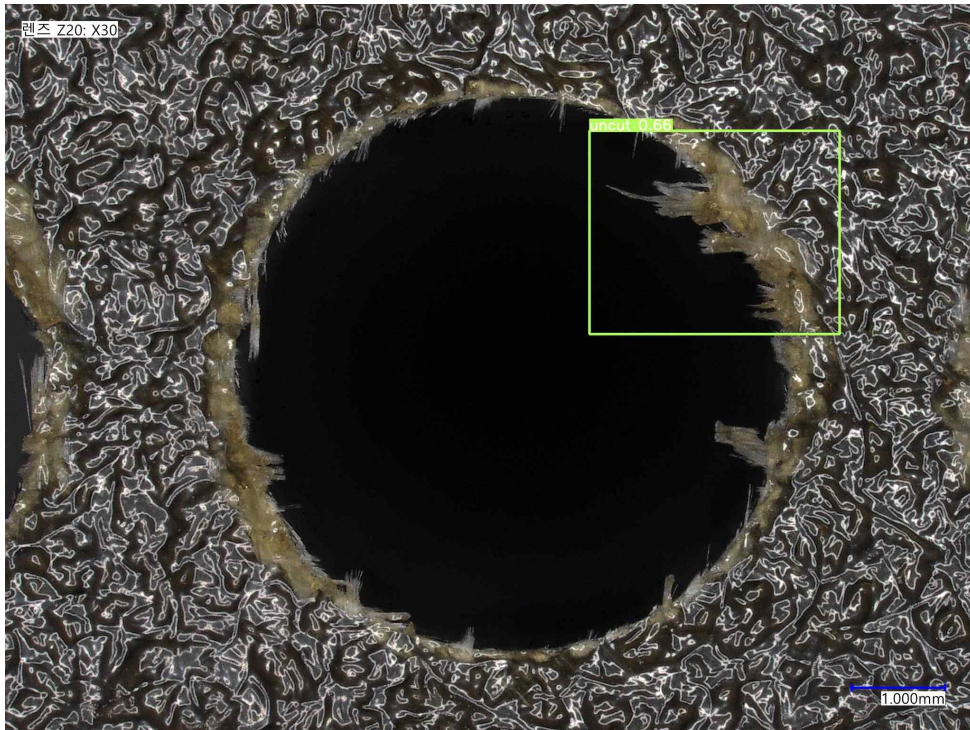
<메인 페이지>



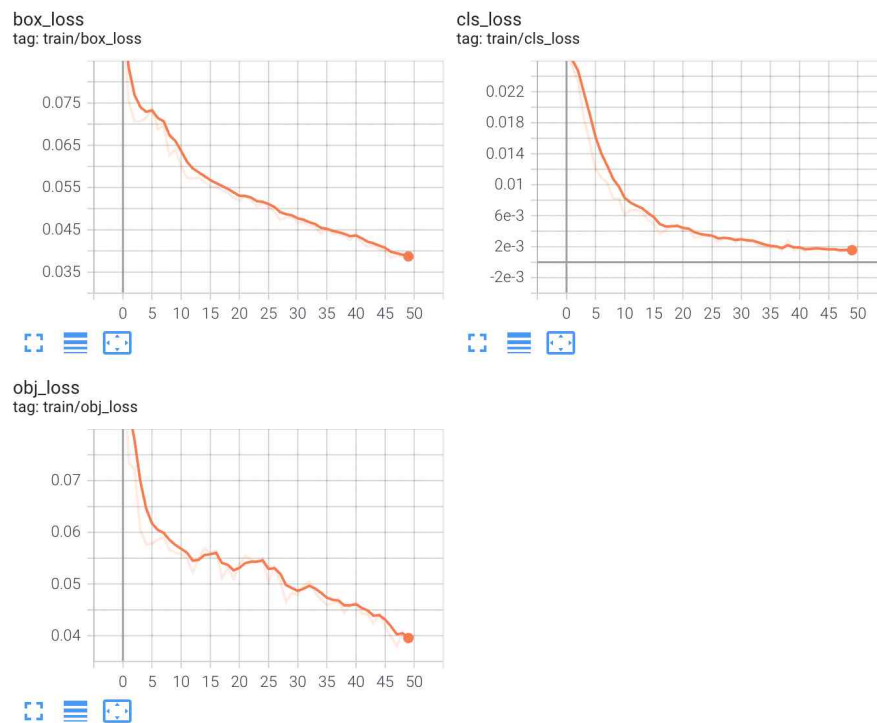
<공정 처리 과정 소개 페이지>



☑ 딥러닝 알고리즘 검출률 확인

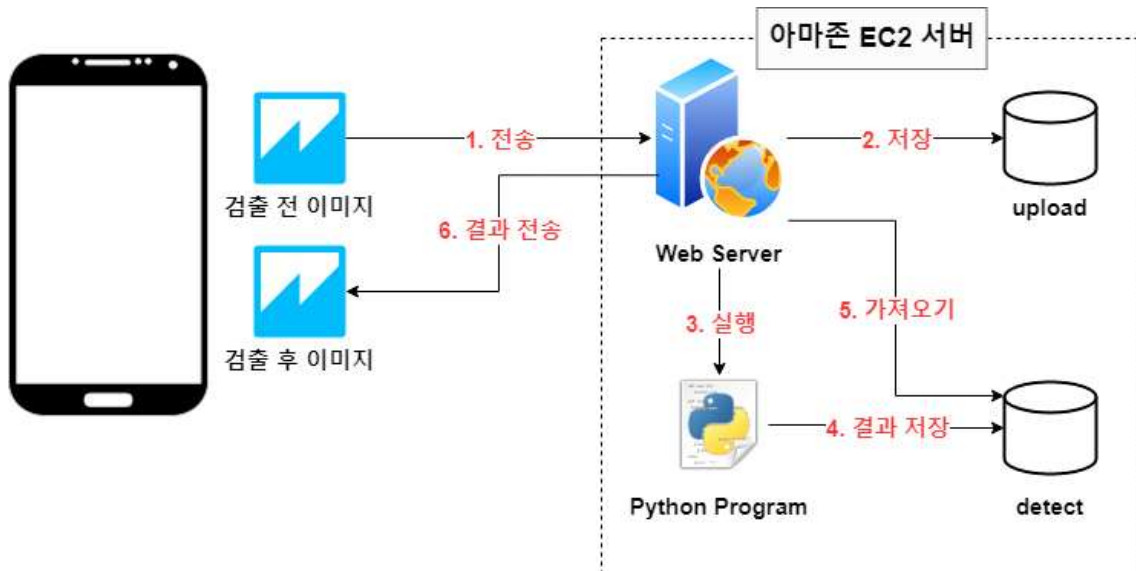


☑ 딥러닝 알고리즘 검출률 확인 - 손실 그래프



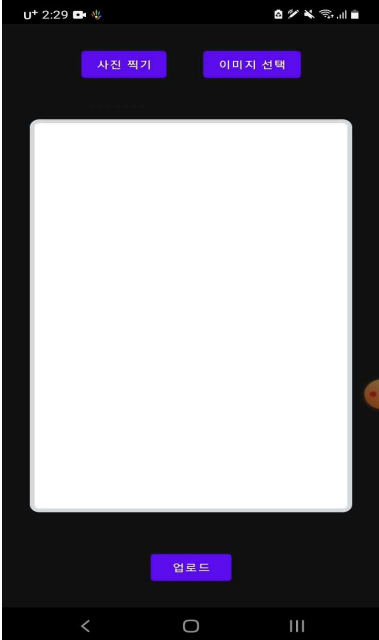
5. 프로토타입

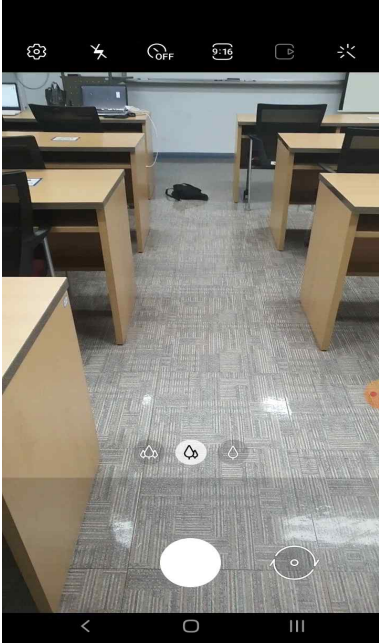
☑ 전반적인 앱-웹-파이썬 프로그램 흐름

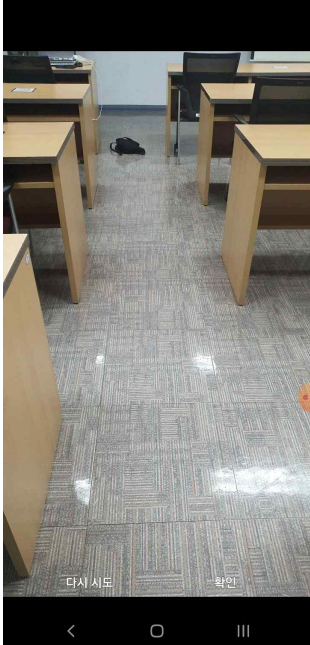


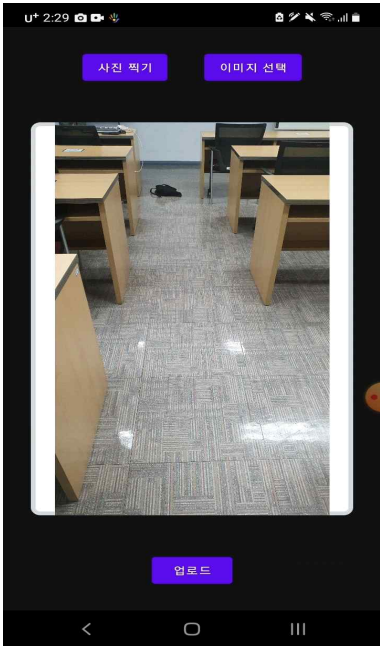
5.1 앱 프로토타입

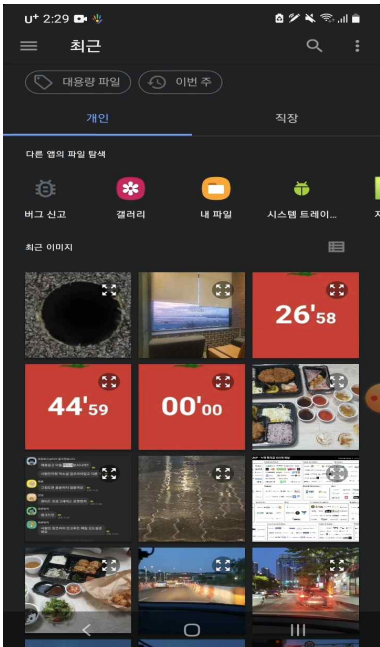
| 항목 | 내용 |
|----|--|
| | <ul style="list-style-type: none"> ● 앱 실행화면 |

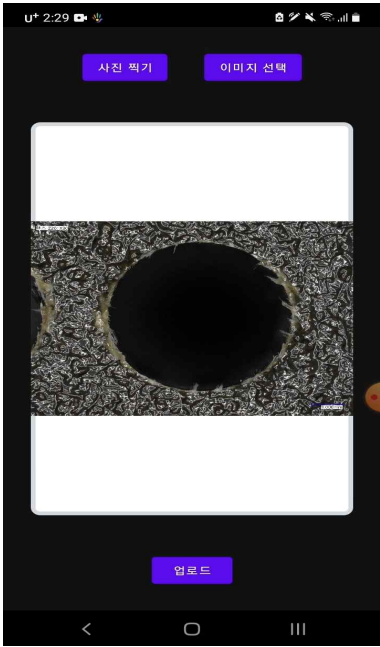
| 항목 | 내용 |
|---|---|
|  | <ul style="list-style-type: none"> ● 앱 메인화면 ● 사진찍기 기능 ● 갤러리에서 이미지 선택기능 ● 서버에 업로드 기능 ● 서버에 업로드 된 이미지는 딥러닝 알고리즘을 통해 결함검출 결과가 하얀 화면쪽에 출력됨 |

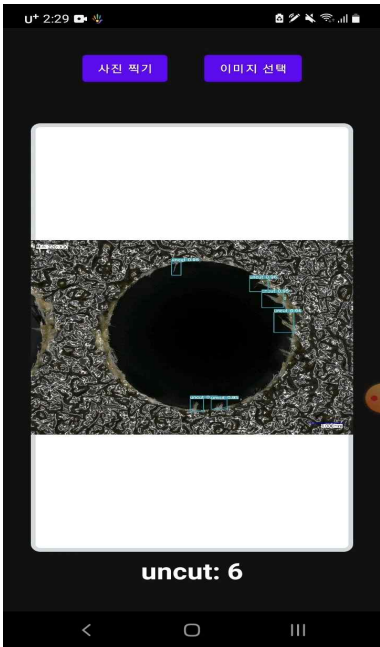
| 항목 | 내용 |
|---|--|
|  | <ul style="list-style-type: none"> ● 사진찍기 버튼을 누르면 해당 화면이 나옴 |

| 항목 | 내용 |
|---|--|
|  | <ul style="list-style-type: none"> ● 사진을 찍으면 다시 찍을지, 메인 화면에 찍은 사진을 업로드 할지 선택할 수 있다. |

| 항목 | 내용 |
|---|---|
|  | <ul style="list-style-type: none"> ● 사진을 찍고 확인을 누르면 메인 화면에 업로드 됨 |

| 항목 | 내용 |
|---|---|
|  | <ul style="list-style-type: none"> ● 이미지 선택 버튼을 누르면 갤러리의 이미지에서 선택해서 가져올 수 있음 |

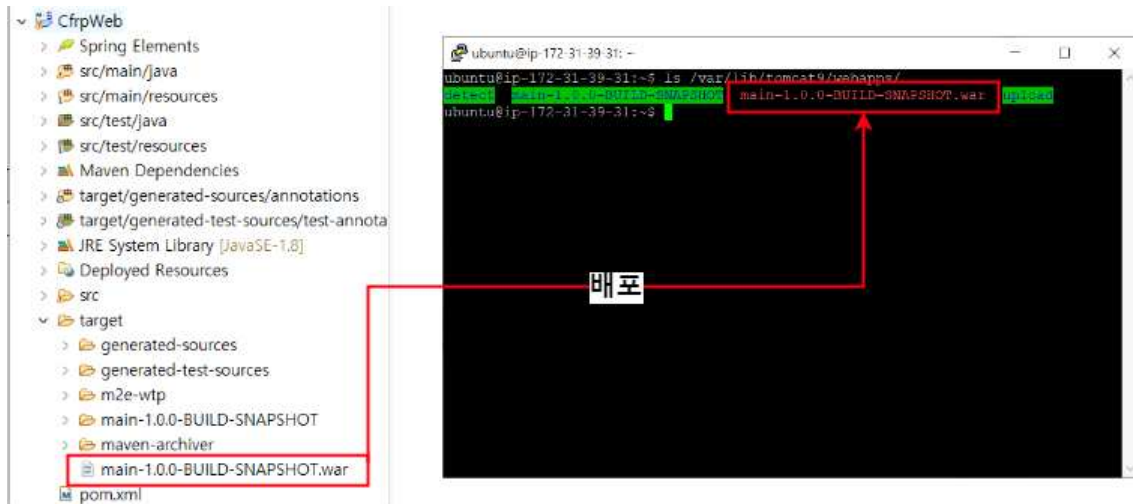
| 항목 | 내용 |
|---|--|
|  | <ul style="list-style-type: none"> ● 실제 테스트 이미지를 업로드 해보았음 |

| 항목 | 내용 |
|---|--|
|  | <ul style="list-style-type: none"> ● 테스트 이미지가 서버에 전송됨 ● 미리 학습시켜서 학습 모델을 만들어 놓았고 서버에서 딥러닝 프로그램을 통해 전송 받은 이미지의 검출을 실시함 ● 검출이 완료되면 서버에서 앱에 결과 이미지와 상태를 전송받음 (뒤의 서버 프로토타입에서 자세히 설명) ● 검출 결과 출력 |

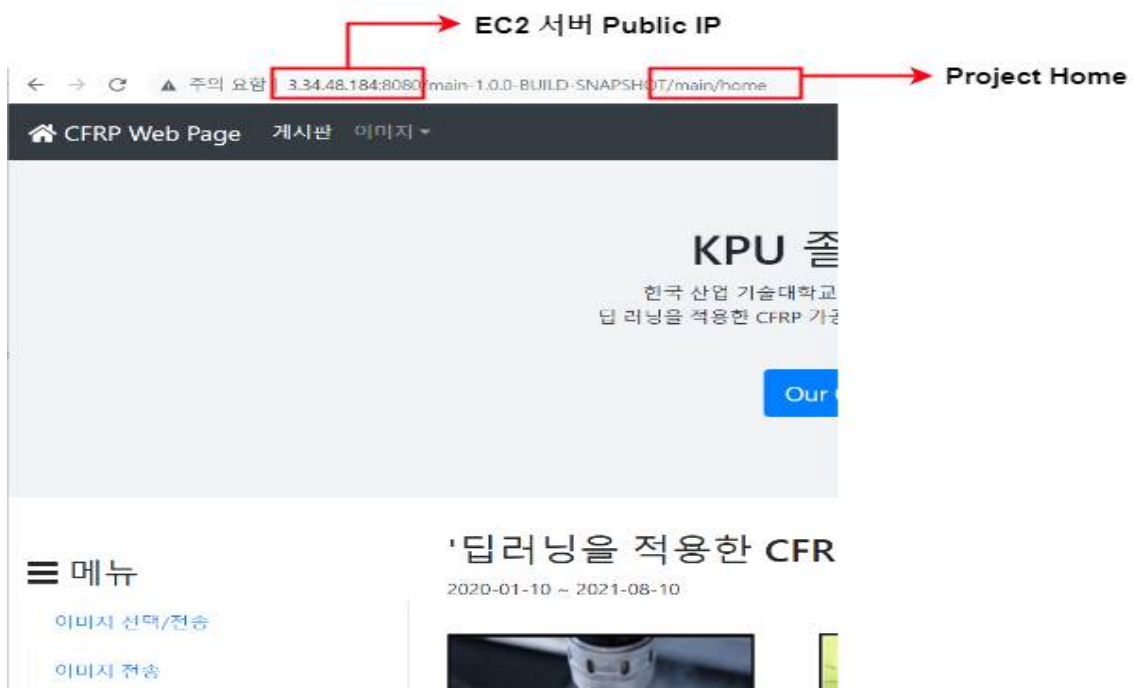
5.2 웹 서버 프로토타입

☑ FileZilla를 이용한 EC2 서버 Spring Project War 배포

spring project war file name : **main-1.0.0-BUILD-SNAPSHOT**



1. EC2 서버에 접속
2. Spring Project Maven install로 war 파일 생성
3. FileZilla Client 프로그램으로 War 파일을 EC2 서버에 배포
4. sudo service tomcat9 start로 웹 서버 실행
5. 웹 서비스 이용



1. Spring Project를 EC2 서버에 배포 후 웹 서버 실행
 2. <http://EC2서버IP+배포파일명+Controller메서드> 매핑명
- ex) 메인 : [/main/home](#)
ex) 검출 전 이미지 : [/selectImage](#)
ex) 검출 후 이미지 : [/detectImage](#)

☑ EC2 서버 내 이미지 검출 전, 후 이미지 저장 경로 설정[웹]

```
<beans:bean id="uploadPath" class="java.lang.String">
  <beans:constructor-arg value="/var/lib/tomcat9/webapps/upload/"></beans:constructor-arg>
</beans:bean>

<beans:bean id="detectPath" class="java.lang.String">
  <beans:constructor-arg value="/var/lib/tomcat9/webapps/detect/"></beans:constructor-arg>
</beans:bean>
```

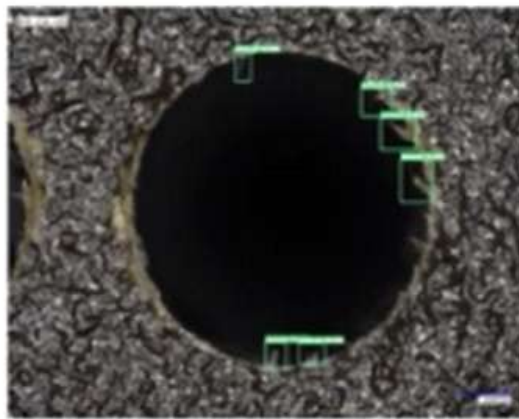
1. uploadPath는 이미지 검출 전 앱이나 사용자에게 전송 받은 이미지를 저장하는 빈
[EC2 서버의 /var/lib/tomcat9/webapps/upload/ 절대 경로로 명시](#)
2. detectPath는 이미지 검출 후 파이썬 프로그램으로부터 받은 이미지를 저장하는 빈
[EC2 서버의 /var/lib/tomcat9/webapps/detect/ 절대 경로로 명시](#)

☑ 검출 전 이미지 : EC2 [/var/lib/tomcat9/webapps/upload/](#)



1. 웹 서버는 앱 또는 사용자에게 전송 받은 이미지를 uploadPath에 저장한다.
2. upload 폴더에 존재하는 이미지를 owl-carousel css를 통해 슬라이드 형식으로 보여 준다.
3. 사용 가능 기능
 - 이전, 다음으로 이미지 전환
 - 마우스 드래그로 넘기기

☑ 검출 후 이미지 : EC2 /var/lib/tomcat9/webapps/detect



1. 파이썬 프로그램은 검출 결과 이미지와 text 파일을 detect 폴더에 저장한다.
2. detect 폴더에 존재하는 이미지를 owl-carousel css를 통해 슬라이드 형식으로 보여 준다.
3. 사용 가능 기능
 - 이전, 다음으로 이미지 전환
 - 마우스 드래그로 이미지 전환

5.3 서버 구현 내용

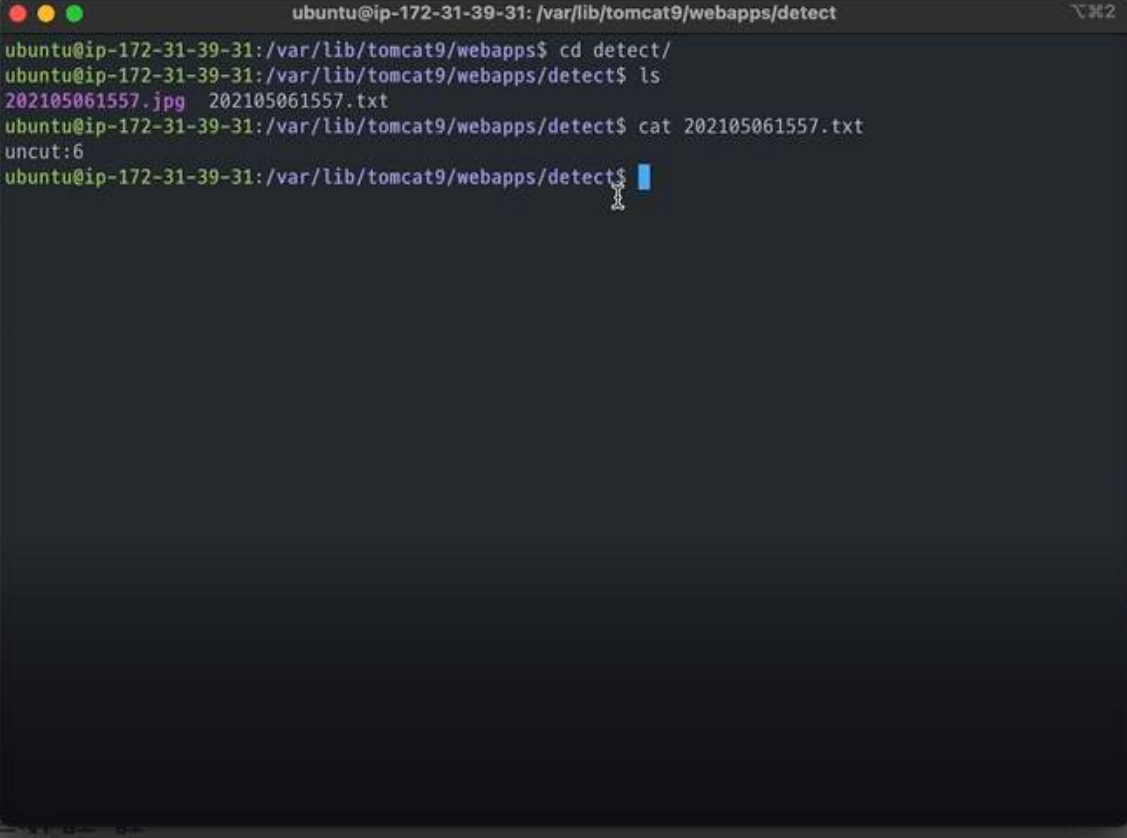
☑ 앱에서 업로드한 이미지 저장 경로

```
ubuntu@ip-172-31-39-31: /var/lib/tomcat9/webapps/upload
ubuntu@ip-172-31-39-31:~$ sudo service tomcat9 status
● tomcat9.service - Apache Tomcat 9 Web Application Server
   Loaded: loaded (/lib/systemd/system/tomcat9.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2021-05-06 17:40:26 UTC; 42s ago
     Docs: https://tomcat.apache.org/tomcat-9.0-doc/index.html
   Process: 2609 ExecStartPre=/usr/libexec/tomcat9/tomcat-update-policy.sh (code=exited, status=0)
   Main PID: 2623 (java)
     Tasks: 14 (limit: 1160)
    Memory: 185.6M
   CGroup: /system.slice/tomcat9.service
           └─2623 /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Djava.util.logging.config.file=/va

May 06 17:40:27 ip-172-31-39-31 tomcat9[2623]: APR/OpenSSL configuration: useAprConnector [false],
May 06 17:40:27 ip-172-31-39-31 tomcat9[2623]: OpenSSL successfully initialized [OpenSSL 1.1.1f 3
May 06 17:40:28 ip-172-31-39-31 tomcat9[2623]: Initializing ProtocolHandler ["http-nio-8080"]
May 06 17:40:28 ip-172-31-39-31 tomcat9[2623]: Server initialization in [1,294] milliseconds
May 06 17:40:28 ip-172-31-39-31 tomcat9[2623]: Starting Service [Catalina]
May 06 17:40:28 ip-172-31-39-31 tomcat9[2623]: Starting Servlet engine: [Apache Tomcat/9.0.31 (Ubu
May 06 17:40:28 ip-172-31-39-31 tomcat9[2623]: Deploying web application archive [/var/lib/tomcat9
May 06 17:40:33 ip-172-31-39-31 tomcat9[2623]: At least one JAR was scanned for TLDs yet contained
May 06 17:40:33 ip-172-31-39-31 tomcat9[2623]: INFO : org.springframework.web.context.ContextLoade
May 06 17:40:34 ip-172-31-39-31 tomcat9[2623]: INFO : org.springframework.web.context.ContextLoade
ubuntu@ip-172-31-39-31:~$ cd /var/lib/tomcat9/webapps/upload/
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps/upload$ ls
202105061557.jpg
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps/upload$
```

1. 앱에서 이미지를 업로드하면 /var/lib/tomcat9/webapps/upload 폴더에 저장된다
2. 이 경로에 있는 이미지들은 /home/ubuntu/yolov5 폴더 내의 detect.py 프로그램에 의해서 결함검출을 실시한다.

☑ 검출을 마친 이미지와 결과 저장경로



```
ubuntu@ip-172-31-39-31: /var/lib/tomcat9/webapps/detect
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps$ cd detect/
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps/detect$ ls
202105061557.jpg  202105061557.txt
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps/detect$ cat 202105061557.txt
uncut:6
ubuntu@ip-172-31-39-31:/var/lib/tomcat9/webapps/detect$
```

1. 이미지 검출 결과는 /var/lib/tomcat9/webapps/detect 폴더에 저장된다.
2. 검출 결과 이미지와 txt파일로 저장된다.
3. txt파일에는 어떤 결함이 있는지 작성되어있다.
4. 이 두 가지 파일은 앱에 전송된다.