

---

# 2024년 빅데이터 기반 AI 개발 전문가 「차량 번호 인식 프로젝트」 포트폴리오

---

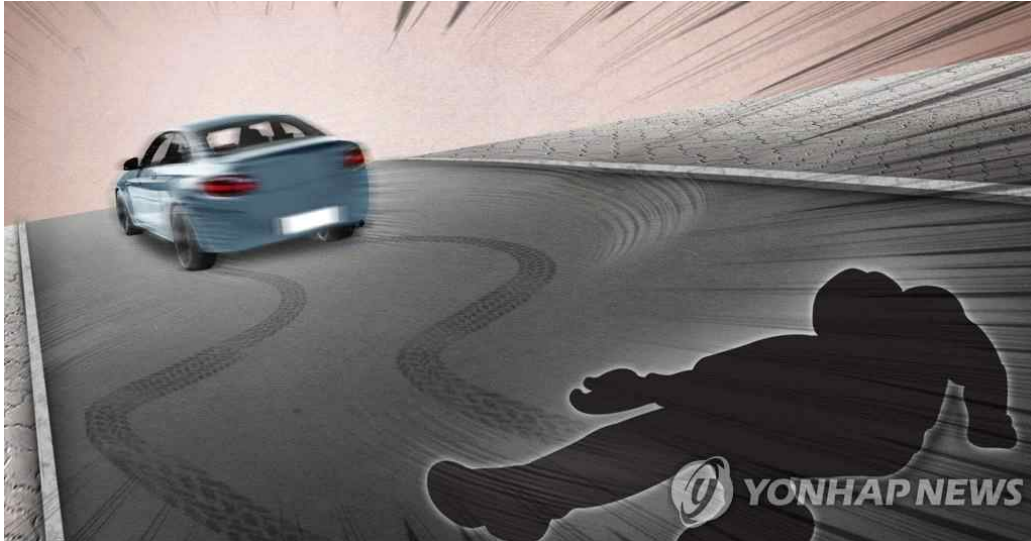
2024. 6

I. 개요	
1. 제안 배경 .....	<u>3</u>
2. 목적(필요성) .....	<u>5</u>
3. 하드웨어 사양 .....	<u>6</u>
4. 개발 기술의 독창성 및 차별성 .....	<u>7</u>
5. 학습 파라미터 .....	<u>7</u>
II. 프로젝트 목표	
1. 정성적 목표 .....	<u>8</u>
2. 성능 목표 .....	<u>8</u>
III. 학습 데이터셋	
1. 차량 번호판 데이터셋 .....	<u>9</u>
2. 차량 번호 인식 데이터셋 .....	<u>10</u>
3. 추가 학습 데이터셋 .....	<u>10</u>
4. 테스트 데이터셋 .....	<u>11</u>
IV. 개발 내용	
1. YOLOv5 전체 순서도 .....	<u>12</u>
2. YOLOv5 훈련 순서도 .....	<u>13</u>
3. FTP 전송 프로그램 .....	<u>15</u>
4. 센터 프로그램 .....	<u>19</u>
V. 추진일정 .....	<u>25</u>
VI. 기대효과	
1. 개발 산출물의 지속 가능성 .....	<u>26</u>
2. 데이터 활용에 따른 파급효과 .....	<u>26</u>
VII. 인력 구성 .....	<u>27</u>
VIII. 별첨	
1. [별첨 1] YOLOv5 요구사항 .....	<u>28</u>
2. [별첨 2] 번호판 인식 학습 결과 .....	<u>30</u>
3. [별첨 3] 문자 인식 학습 결과 .....	<u>31</u>

# I. 개요

## 1. 제언배경

- ① 요즘 사회 이슈  
뺑소니 교통사고 증가



지난 5년간 서울에서 발생한 뺑소니 교통사고 건수가 연평균 800건에 달하는 것으로 나타났다. 19일 도로교통공단 교통사고분석시스템(TAAS)에 따르면 2019년부터 작년까지 서울에서 발생한 뺑소니 교통사고는 3천965건이다.  
한 해 평균 793건으로, 하루 평균 2.2건 발생하는 셈이다.<sup>1)</sup>

## 교통 관리와 ITS

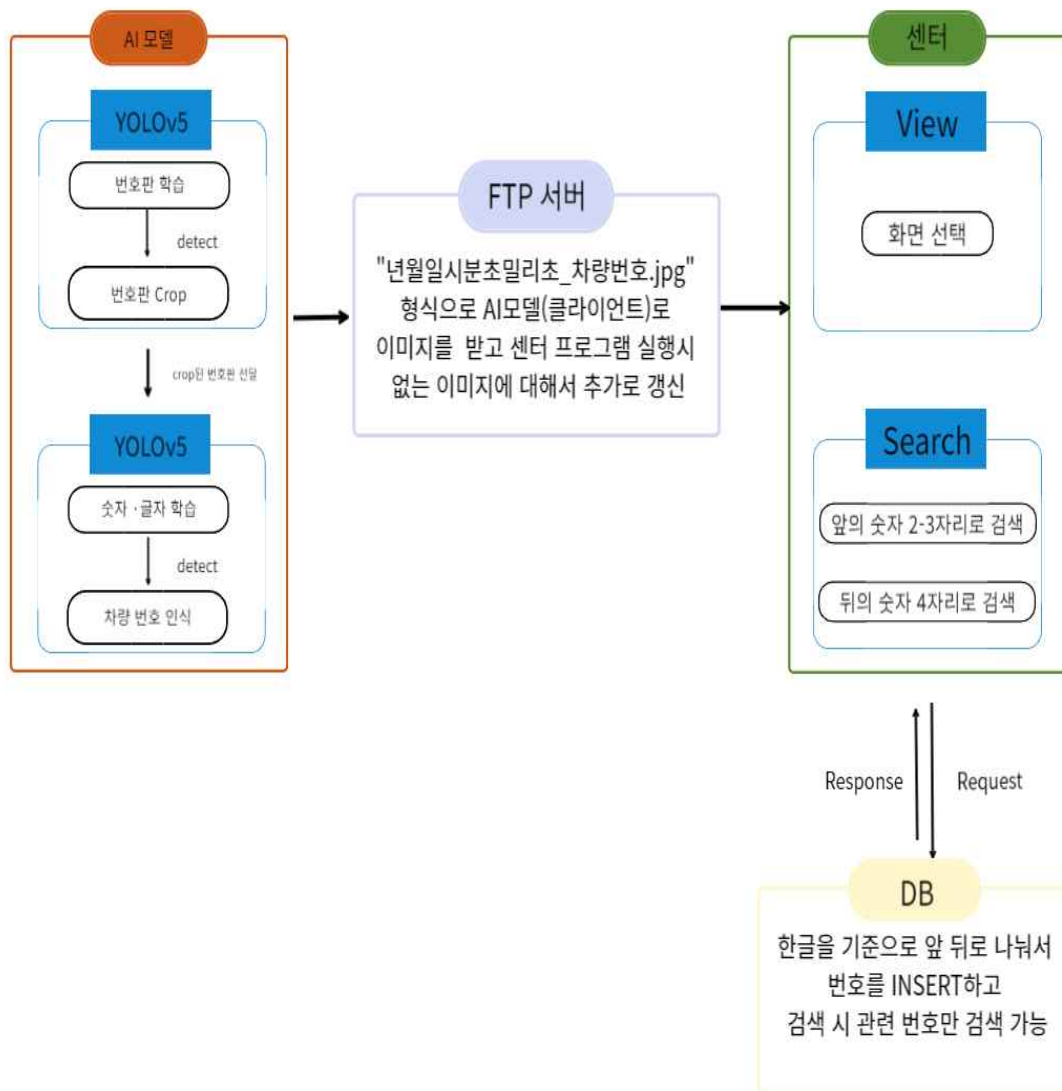


천안시가 주요 도로에 긴급차량 우선신호시스템 등 지능형교통체계(ITS)를 확대 구축한 결과 교통편의 만족도는 향상되고 교통흐름은 개선된 것으로 나타났다 7일 밝혔다.

시는 국토교통부 교통조사지침에 따라 운수종사자와 시민 100명을 대상으로 설문조사를 한 결과, 지능형교통체계 구축 전 신호체계 만족도는 평균 60.2점에서 67.8점으로 높아졌다.<sup>2)</sup>

## ② 전체 설명 및 순서도

차량 번호판 데이터셋을 수집하여 YOLOv5로 학습시켜서 차량 번호판을 인식합니다. 인식한 차량 번호판을 다시 YOLOv5로 학습시켜서 차량 번호를 인식하는 YOLOv5 프레임워크를 개발합니다. YOLOv5 프레임워크를 통해 인식된 번호판을 FTP를 통해서 “년월일시분초밀리초\_차량번호.jpg” 형식으로 센터 프로그램에 전송합니다. 센터 프로그램 실행 시 FTP 서버와 연결되어 있기 때문에 자동으로 이미지를 갱신하고 DB와 연동하여 원하는 이미지를 빠르게 검색할 수 있습니다.



## 2. 목적(필요성)

### ① 자동화된 차량 인식 및 데이터 관리

- 수작업으로 차량을 관리하는 방식은 오류가 발생할 수 있으며, 많은 인력을 필요로 합니다.
- 이를 자동화된 시스템으로 대체함으로써, 시간과 비용을 절감하고, 데이터의 정확성을 높일 수 있습니다.
- YOLOv5 프레임워크를 활용하여 90% 이상의 인식률을 달성함으로써, 신뢰성 높은 차량 인식 데이터를 제공하고, 이를 데이터베이스에 저장하여 필요시 언제든지 검색 및 조회할 수 있는 환경을 구축합니다.

### ② 실시간 데이터 전송 및 통합 관리

- 인식된 차량 데이터를 실시간으로 센터 프로그램에 전송하여, 모든 차량 출입 정보를 실시간으로 관리할 수 있습니다.
- FTP 전송 기능을 포함하여, 연계된 시스템 간 데이터 공유와 통합 관리가 가능하도록 설계하였습니다.

### ③ 사용자 중심의 검색 기능 제공

- 차량번호를 기준으로 인식된 차량 데이터를 검색할 수 있어, 필요한 정보를 빠르고 정확하게 조회할 수 있습니다.
- 이는 주차 관리, 출입 통제, 보안 감시 등의 다양한 분야에서 유용하게 활용될 수 있습니다.

### ④ 보안 강화 및 사건 추적 기능

- 특정 차량의 출입 기록을 정확하게 추적하여, 보안 사고 발생 시 신속하게 대응할 수 있습니다.
- 이는 시설 보안, 사건 조사, 교통 위반 단속 등 다양한 보안 관련 분야에서 필수적인 기능입니다.

### ⑤ 플레이백 기능을 통한 데이터 활용성 증대

- 저장된 차량 인식 데이터를 플레이백 기능을 통해 재생할 수 있어, 특정 시간대의 차량 출입 정보를 확인하거나, 과거 데이터를 분석하는 데 유용합니다.
- 이를 통해 차량 출입 패턴 분석, 교통 흐름 개선, 주차 공간 최적화 등 다양한 분야에서 데이터를 활용할 수 있습니다.

### ⑤ 미래 확장성 고려

- FTP 전송 기능을 포함하여, 다양한 시스템 간 데이터 공유와 통합 관리가 가능하도록 하였습니다.
- 이는 향후 스마트 시티 구축, 교통 관리 시스템 통합 등 다양한 확장 가능성을 열어줍니다.

### 3. 하드웨어 사양

Framework	YOLOv5		
하드웨어 사양	<div> 13th Gen Intel(R) Core(TM) i5-13400 2.50 GHz</div>	<div> NVIDIA GeForce RTX 3060</div>	<div> 32GB</div>
운영체제	<div></div>		
소프트웨어 사양	<div> python 3.9.19</div>	<div> PyTorch 2.3.0</div>	<div> ANACONDA 24.3.0</div>
cuda / cuDNN	<div> NVIDIA CUDA 12.1</div>	<div> cuDNN 8.9.6</div>	
DB	<div></div>		
IDE	<div></div>		
UI	<div> 5.15.0</div>		
Yolov5 요구사항	별첨 1 참조		

## 4. 개발기술의 독창성 및 차별성

□ Yolo 분석을 통한 객체 종류 분류 후 최적 알고리즘 선별 프로그램

딥러닝 구분	기본 사용	부가적 사용
차량번호판인식	YOLOv5, KNN	Yolo - 99%(mAP50 기준) KNN - 24%
차량번호인식	YOLOv5, KNN 문자인식	Yolo - 91%(mAP50 기준) KNN - 85%

## 5. 학습 파라미터

데이터셋	1504장
Epoch	(사전 학습) 2000회 (전이 학습) 2000회
이미지 사이즈	(차량 번호판) 640 (차량 번호 인식) 416
배치 크기	16
YAML 파일	데이터셋 경로 지정(train/val) (차량 번호판) 클래스 지정 번호판(car_plate) (차량 번호 인식) 클래스 지정 숫자 0~9, 한글 a1~a32 (가~주), d1~d3(하,허,호)
worker수	8
conf-thres	0.4 ~ 0.9 (오탐률과 누락률 조절하기 위해서)
iou-thres	0.4 ~ 0.9 (탐지 정확도 조절하기 위해서)
optimizer	Adam, SGD(default)

1) 이미령, “서울 뺑소니 교통사고 연평균 793건…강남·서초구 최다”, 연합뉴스, 2024년 05월 19일,  
<https://www.yna.co.kr/view/AKR20240518024100004?input=1195m>

2) 김인수, “천안시, 지능형교통체계(ITS)로 교통편의 만족도 향상”, 금강뉴스, 2024년 06월 09일,  
<https://www.ggilbo.com/news/articleView.html?idxno=1032921>

## II. 프로젝트 목표

### 1. 정성적 목표

- 센터 소프트웨어 확보
  - dataset 이미지 1000장 이상 확보
  - DB 연동으로 인한 빠른 번호판 검색 기능 추가
  - 로컬 프로그램과 센터 프로그램의 FTP 통신
  - 센터 프로그램 인터페이스가 사용자가 쉽게 사용할 수 있도록 직관적 설계
  - 데이터 전송 과정에서의 오류를 최소화
  - 자체 육안검사 또는 담임강사 또는 외부 강사 입회 하에 객관적 점수

### 2. 성능 목표

주요 성능지표	단위	개발목표	학습횟수	학습 결과	측정대상
번호판	%	95%	1회	별첨 1 참조	자체 검사
문자 인식	%	95%	3회	별첨 2 참조	자체 검사
프로그램 오류 비율	횟수	1회	-	-	외부 3자 입회하에 검수
센터에서 인식된 이미지 View	%	95%	-	-	외부 3자 입회하에 검수
FTP 통신 미전송	%	1%	-	-	외부 3자 입회하에 검수




### Ⅲ. 학습 데이터셋

#### 1. 차량 번호판 데이터셋

수집 방법	<p>유튜브에서 수집한 이미지</p> 
	<p>직접 수집한 이미지</p> <p>AIHUB에 있는 데이터셋들은 차량 번호판만 덩그러니 있는 데이터들만 있습니다. 저희는 차량 앞 혹은 뒤에 번호판이 있는 위치를 알게 하고 이미지의 맥락을 YOLOv5에 학습시키기 위해서 차량 번호판을 직접 수집하게 되었습니다.</p> 
평가 방법	<p>mAP50을 기준으로 모델의 성능을 평가</p> <p>다른 조의 데이터를 받아서 얼마나 잘 번호판을 인식하는지 확인</p>
AIHUB 데이터	



## 2. 차량 번호 인식 데이터셋

수집 방법	<p>차량 번호판에서 인식한 번호판을 crop하는 함수를 통해 detect했을 때 crop하는 이미지를 사용함</p> 
평가 방법	<p>YOLOv5가 인식한 차량 번호와 이미지 파일 제목(차량 번호)이 일치하는지를 확인</p> <pre>image 219/220 C:\study\yolov5-master_number\data\images\crop\960168.jpg: 128x416 1 0, 1 1, 2 6s, 1 8, 1 9, 1 a30, 11.5ms p.stem 97도 7637 global_list 97도 7637 198</pre> <p>전체 데이터셋에서 얼마나 차량 번호랑 일치하는지 비교하여 백분율로 표시</p>

## 3. 추가 학습 데이터셋

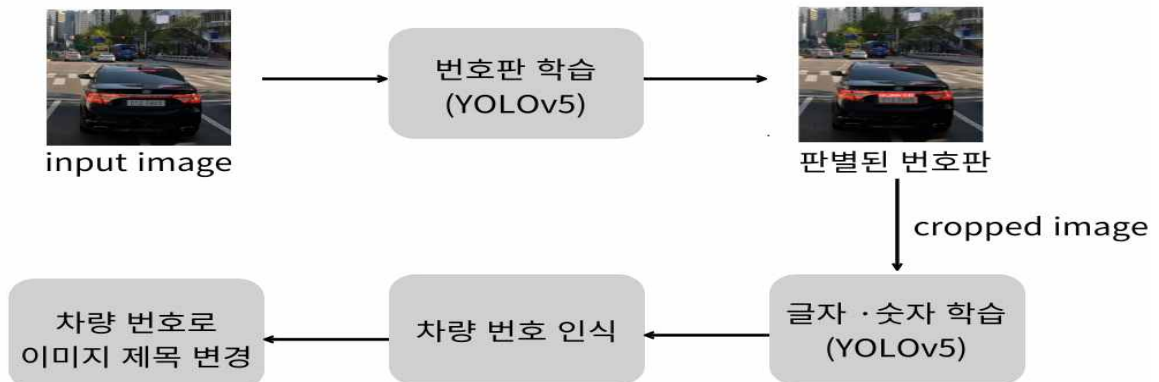
수집 방법	<p>차량 번호 인식 데이터셋 학습 후 테스트 데이터로 학습을 돌렸을 때 저녁이거나 번호판이 틀어졌을 경우 학습을 못했기에 AIHUB에 "자동차 차종/연식/번호판 인식용 영상" train 데이터에서 밤에 촬영된 이미지와 각도가 심하게 틀어진 이미지를 추가하여 학습을 진행</p> 
평가 방법	<p>테스트 데이터로 YOLOv5가 인식한 차량 번호와 이미지 파일 제목(차량 번호)이 일치하는지를 확인</p> <pre>image 219/220 C:\study\yolov5-master_number\data\images\crop\960168.jpg: 128x416 1 0, 1 1, 2 6s, 1 8, 1 9, 1 a30, 11.5ms p.stem 97도 7637 global_list 97도 7637 198</pre> <p>전체 데이터셋에서 얼마나 차량 번호랑 일치하는지 비교하여 백분율로 표시</p>

#### 4. 테스트 데이터셋

수집 방법	<p>AIHUB에 “자동차 차종/연식/번호판 인식용 영상” validation 데이터를 활용하여 학습 데이터셋과 무관한 데이터셋으로 이미지를 테스트함</p> <div data-bbox="472 371 876 544">  </div> <div data-bbox="927 371 1364 544">  </div>
평가 방법	<p>테스트 데이터로 YOLOv5가 인식한 차량 번호와 이미지 파일 제목(차량 번호)이 일치하는지를 확인</p> <div data-bbox="486 647 1377 712"> <pre>image 219/220 C:\study\yolov5-master_number\data\images\crop\960168.jpg: 128x416 1 0, 1 1, 2 6s, 1 8, 1 9, 1 a30, 11.5ms p.stem 97도 7637 global_list 97도 7637 198</pre> </div> <p>전체 데이터셋에서 얼마나 차량 번호랑 일치하는지 비교하여 백분율로 표시</p>

## IV. 개발내용

### 1. YOLOv5 전체 순서도



#### ① Input Image (입력 이미지)

시스템에 입력된 원본 차량 이미지입니다. 이 이미지는 도로 위의 차량을 포함하고 있습니다.

#### ② 번호판 학습 (YOLOv5)

YOLOv5 모델을 사용하여 입력 이미지에서 차량 번호판을 탐지합니다. 이 모델은 차량 이미지에서 번호판이 위치한 부분을 정확하게 식별합니다.

#### ③ Output

탐지된 번호판이 포함된 이미지 영역을 출력합니다.

#### ④ Cropped Image (잘린 이미지)

YOLOv5 모델에 의해 탐지된 번호판 부분을 원본 이미지에서 잘라낸 결과입니다. 이 잘린 이미지는 번호판만 포함하고 있습니다.

#### ⑤ 글자 · 숫자 학습 (YOLOv5)

번호판의 잘린 이미지에서 개별 문자(글자 및 숫자)를 인식하기 위해 또 다른 YOLOv5 모델을 사용합니다. 이 단계에서는 번호판에 있는 문자를 하나씩 탐지하고 인식합니다.

#### ⑥ 차량 번호 인식

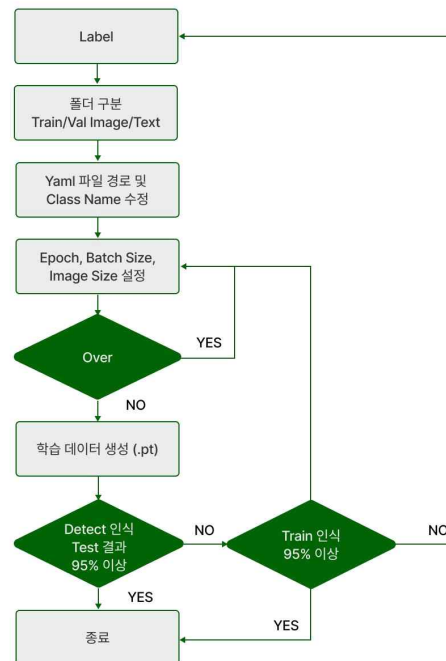
개별 문자 탐지 결과를 조합하여 최종적으로 차량 번호를 인식합니다. 이는 각 문자를 결합하여 완전한 차량 번호를 구성하는 단계입니다.

#### ⑦ 차량 번호로 이미지 제목 변경

인식된 차량 번호를 기반으로 원본 이미지의 제목을 변경합니다. 이 과정은 데이터 베이스나 파일 시스템에서 이미지 파일을 관리하는 데 유용합니다.

## 2. YOLOv5 훈련 순서도

Train Flow Chart



### ① Label (레이블링)

데이터에 레이블을 지정하는 과정입니다. 여기서 각 이미지에 대해 해당하는 객체를 식별하고, 그 정보를 저장합니다.

### ② 폴더 구분 (Train/Val Image/Text)

데이터를 학습용(Train)과 검증용(Validation)으로 구분하는 과정입니다. 이미지를 학습용 폴더와 검증용 폴더로 나누고, 각각에 해당하는 텍스트 파일을 준비합니다.

### ③ YAML 파일 경로 및 Class Name 수정

학습에 필요한 설정 파일인 YAML 파일을 수정합니다. 여기에는 데이터의 경로와 클래스 이름 등이 포함됩니다.

### ④ Epoch, Batch Size, Image Size 설정

학습에 필요한 파라미터들을 설정합니다. Epoch(에포크)는 학습 반복 횟수, Batch Size(배치 크기)는 한 번에 처리할 데이터 양, Image Size(이미지 크기)는 입력 이미지의 크기를 의미합니다.

### ⑤ Over 체크

학습이 완료되었는지 확인하는 단계입니다. 학습이 완료되지 않았다면 다음 단계로 넘어갑니다.

⑥ 학습 데이터 생성 (.pt)

학습을 통해 모델을 학습시키고, 결과를 .pt 파일로 저장합니다. .pt 파일은 PyTorch에서 사용하는 모델 저장 파일 형식입니다.

⑦ Train 인식 95% 이상 확인

학습된 모델의 성능을 평가합니다. 여기서는 학습 과정에서 95% 이상의 인식률을 달성했는지 확인합니다.

NO: 인식률이 95% 미만이면 파라미터(Epoch, Batch Size, Image Size)를 조정하고 다시 학습을 진행합니다.

YES: 인식률이 95% 이상이면 다음 단계로 넘어갑니다.

⑧ Detect 인식 Test 결과 95% 이상 확인

학습된 모델을 사용하여 검증 데이터에 대해 테스트를 수행합니다. 테스트 결과가 95% 이상의 인식률을 달성했는지 확인합니다.

NO: 테스트 결과가 95% 미만이면 학습 데이터를 재생성하고, 학습을 다시 진행합니다.

YES: 테스트 결과가 95% 이상이면 학습이 완료됩니다.

⑨ 종료

모든 과정이 완료되었고, 학습된 모델이 95% 이상의 인식률을 달성하여 학습 과정이 종료됩니다.

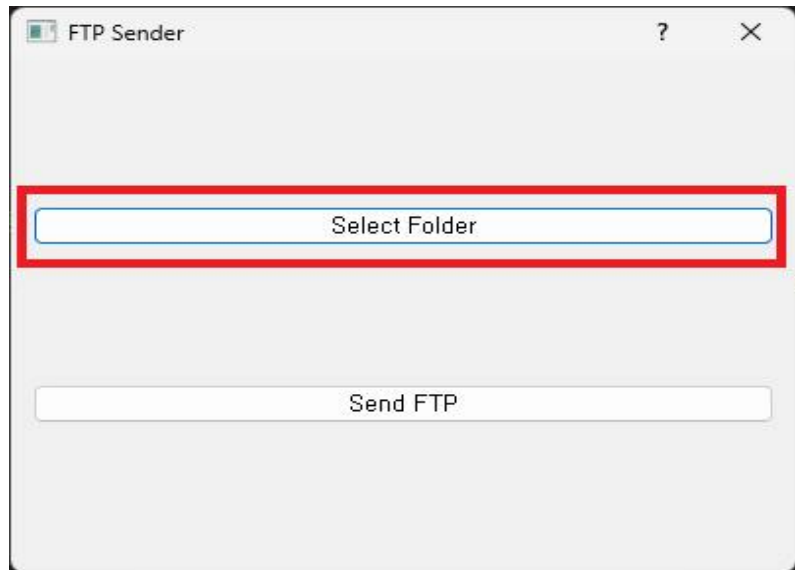
### 3. FTP 전송 프로그램

FTP 전송 프로그램은 원하는 폴더를 간단히 전송할 수 있도록 PyQt5를 사용하여 UI를 생성하여 쉽게 보낼 수 있도록 프로그램을 개발하였습니다.

<p><b>Import</b></p>	<pre>import os from ftplib import FTP, error_perm from PyQt5.QtWidgets import QApplication, QDialog, QFileDialog, QVBoxLayout, QPushButton from PyQt5.QtCore import pyqtSignal</pre> <p>① <b>os</b> 모듈은 운영 체제와 상호 작용하기 위한 다양한 기능을 제공합니다. 예를 들어, 파일 경로를 조작하거나 디렉토리를 탐색하는 기능이 있습니다.</p> <p>② <b>ftplib</b> 모듈의 FTP 클래스는 FTP 프로토콜을 사용하여 원격 서버와 통신하는 기능을 제공합니다. error_perm은 FTP 서버에서 발생하는 권한 오류를 처리하기 위한 예외 클래스입니다.</p> <p>③ <b>PyQt5.QtWidgets</b> 모듈은 다양한 GUI 위젯을 제공합니다. 이들 위젯을 사용하여 사용자 인터페이스를 구성할 수 있습니다.</p> <p>④ <b>PyQt5.QtCore</b> 모듈의 pyqtSignal 클래스는 PyQt5에서 이벤트를 처리하기 위한 시그널을 생성하는 기능을 제공합니다.</p>
<p><b>UI</b></p>	<pre>class FTPSender(QDialog):     selected_folder_signal = pyqtSignal(str) # 선택된 폴더에 대한 정보를 전달하는 시그널을 정의합니다. ①      def __init__(self):         super().__init__() # 부모 클래스의 초기화 메서드를 호출합니다.         self.setWindowTitle("FTP Sender") # 윈도우의 제목을 "FTP Sender"로 설정합니다. ②         self.resize(400, 300) # 윈도우의 크기를 너비 400, 높이 300으로 설정합니다.         self.ftp = FTP() # FTP 객체를 생성합니다. ③         layout = QVBoxLayout() # 수직 박스 레이아웃을 생성합니다. ④         self.select_button = QPushButton("Select Folder") # "Select Folder"라는 텍스트를 가진 버튼을 생성합니다.         self.select_button.clicked.connect(self.select_folder) # 버튼이 클릭되면 self.select_folder 메서드가 호출되도록 연결합니다.         layout.addWidget(self.select_button) # 버튼을 레이아웃에 추가합니다.         self.send_ftp_button = QPushButton("Send FTP") # "Send FTP"라는 텍스트를 가진 버튼을 생성합니다.         self.send_ftp_button.clicked.connect(self.send_ftp_folder) # 버튼이 클릭되면 self.send_ftp_folder 메서드가 호출되도록 연결합니다.         layout.addWidget(self.send_ftp_button) # 버튼을 레이아웃에 추가합니다.         self.setLayout(layout) # 현재 윈도우의 레이아웃을 위에서 생성한 레이아웃으로 설정합니다.         self.folder_path = None # 선택된 폴더의 경로를 저장할 변수를 생성합니다. ⑤</pre> <p>① selected_folder_signal 변수는 QFileDialog에서 선택된 폴더에 대한 정보를 클래스 FTPSender로 전송받습니다.</p> <p>② super().__init__()는 QDialog의 초기화 메서드를 호출하고 self.setWindowTitle()로 윈도우의 제목을 self.resize로 윈도우의 크기를 조절합니다.</p> <p>③ 클래스 내에 인스턴스 변수로 FTP 객체를 생성합니다.</p> <p>④ 레이아웃을 설정하는 부분입니다. self.select_button, self.send_ftp_button이라는 버튼을 생성하고 각각 맞는 함수와 연결시킵니다. 그리고 layout에 추가하고 setLayout으로 레이아웃 설정을 합니다.</p> <p>⑤ self.folder_path라는 변수를 생성하여 선택된 폴더의 경로를 저장할 인스턴스 변수를 생성합니다.</p>



## 폴더 선택

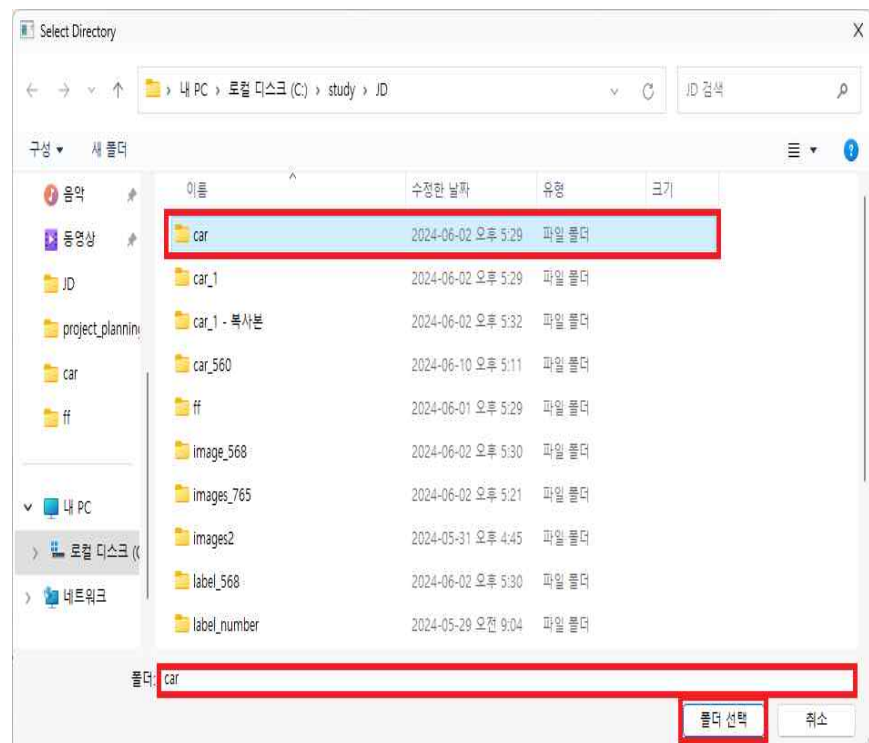


UI에서 Select Folder 버튼과 연결된 함수는 아래 함수와 같습니다.

```
def select_folder(self):
    self.folder_path = QFileDialog.getExistingDirectory(self, "Select Directory") # 사용자에게 디렉토리 선택을 요청하고 선택된 디렉토리의 경로를 가져옵니다.
    if self.folder_path: # 사용자가 디렉토리를 선택했다면
        self.selected_folder_signal.emit(self.folder_path) # 선택된 디렉토리에 대한 정보를 시그널로 보냅니다.
```

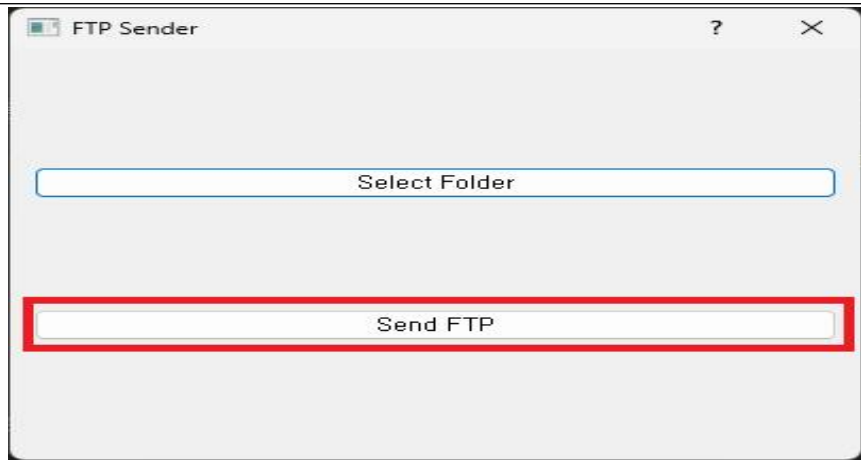
이 함수는 QFileDialog 위젯의 getExistDirectory 함수를 사용하여 사용자에게 디렉토리를 선택하도록 요청하는 대화 상자를 표시합니다.

아래 대화상자에서 사용자가 디렉토리를 선택했다면 선택된 디렉토리에 대한 정보를 시그널로 보냅니다.



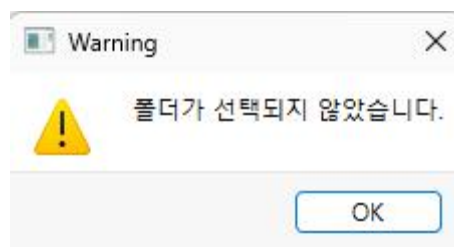


## FTP 전송



```
def send_ftn_folder(self):
    ① if not self.folder_path: # 폴더가 선택되지 않았다면
        QMessageBox.warning(self, 'Warning', '폴더가 선택되지 않았습니다.') # 경고 메시지 박스를 표시합니다.
        return # 함수를 종료합니다.
    ② try:
        self.ftp.connect('127.0.0.1', 21) # FTP 서버에 연결을 시도합니다. 서버 주소와 포트를 입력해야 합니다.
        self.ftp.login('admin', '1234') # FTP 서버에 로그인을 시도합니다. 사용자명과 비밀번호를 입력해야 합니다.
        self.ftp.cwd('/') # FTP 서버의 루트 디렉터리로 이동합니다.
        self.upload_folder(self.folder_path, '') # 선택된 폴더를 FTP 서버에 업로드합니다. 두 번째 인자는 원격 경로로, 여기서는 루트 디렉터리를 의미하는 빈 문자열을 사용합니다.
    except error_perm as e: # FTP 서버에서 권한 오류가 발생한 경우
        print(f"FTP error: {e}") # 에러 메시지를 출력합니다.
    finally:
        self.ftp.quit() # FTP 서버와의 연결을 종료합니다.
```

① 만약 Select Folder에서 폴더를 선택하지 않고 Send FTP버튼을 눌렀다면 "폴더가 선택되지 않았습니다."라는 경고 문구가 뜨면서 함수가 종료되도록 설계했습니다.



② 만약 폴더가 선택된다면 FTP 서버와 연결을 시도합니다. 서버 주소, 포트, 사용자명, 비밀번호는 고정되어 있습니다. FTP 서버의 루트 디렉터리로 이동합니다.

```
def upload_folder(self, local_folder_path, remote_folder_path):
    ③ for root, dirs, files in os.walk(local_folder_path): # 지정된 로컬 폴더 경로에서 모든 하위 디렉터리와 파일을 탐색합니다.
        # 디렉터리 생성
        for dirname in dirs: # 각 하위 디렉터리에 대해
            local_dir = os.path.join(root, dirname) # 로컬 디렉터리의 절대 경로를 생성합니다.
            relative_dir = os.path.relpath(local_dir, local_folder_path) # 로컬 디렉터리의 상대 경로를 생성합니다.
            remote_dir = os.path.join(remote_folder_path, relative_dir).replace('\\', '/') # 원격 디렉터리의 경로를 생성합니다.
            try:
                self.ftp.mkd(remote_dir) # 원격 디렉토리를 생성합니다.
                print(f"Directory created: {remote_dir}") # 디렉터리 생성 성공 메시지를 출력합니다.
            except error_perm as e: # 권한 오류가 발생한 경우
                print(f"Could not create directory {remote_dir}: {e}") # 디렉터리 생성 실패 메시지를 출력합니다.
    ④ # 파일 업로드
    for filename in files: # 각 파일에 대해
        local_file = os.path.join(root, filename) # 로컬 파일의 절대 경로를 생성합니다.
        relative_file = os.path.relpath(local_file, local_folder_path) # 로컬 파일의 상대 경로를 생성합니다.
        remote_file = os.path.join(remote_folder_path, relative_file).replace('\\', '/') # 원격 파일의 경로를 생성합니다.
        try:
            with open(local_file, 'rb') as file: # 로컬 파일을 바이너리 읽기 모드로 엽니다.
                self.ftp.storbinary(f'STOR {remote_file}', file) # 파일을 원격 서버에 업로드합니다.
            print(f"File uploaded: {remote_file}") # 파일 업로드 성공 메시지를 출력합니다.
        except error_perm as e: # 권한 오류가 발생한 경우
            print(f"Could not upload file {remote_file}: {e}") # 파일 업로드 실패 메시지를 출력합니다.
```

	<p>③ 지정된 로컬 폴더 경로에서 모든 하위 디렉터리와 파일을 탐색합니다. 각 하위 디렉터리에 대해 로컬 디렉터리의 절대 경로와 상대 경로를 생성하고, 원격 디렉터리의 경로를 생성합니다. FTP 서버의 루트 디렉터리를 생성하고 만약 디렉터리 생성이 성공하면 성공 메시지를 출력하고 만약 실패하면 생성 실패 메시지를 출력합니다.</p> <p>④ 선택한 폴더의 각 파일에 대해 로컬 파일의 절대 경로와 상대 경로를 생성하고 원격 파일의 경로도 생성합니다. 로컬 파일을 바이너리 읽기 모드로 읽어서 파일을 FTP 서버에 업로드하고 업로드가 성공하면 성공 메시지를 실패하면 실패 메시지를 출력합니다.</p>
--	--

## 4. 센터 프로그램

<p>Import</p>	<pre>import sys from PyQt5.QtWidgets import (QApplication, QLabel, QWidget, QPushButton, QLineEdit,                              QListWidget, QAbstractItemView, QListWidgetItem) from PyQt5.QtCore import Qt, QThread, pyqtSignal from PyQt5.QtGui import QPixmap, QIcon import os import sqlite3 import ftplib</pre> <p>① <b>sys</b> 모듈은 파이썬 인터프리터와 관련된 변수와 함수를 제공합니다. 예를 들어, 명령행 인자를 가져오거나 프로그램을 종료하는 기능 등이 있습니다.</p> <p>② <b>PyQt5.QtWidgets</b> 모듈은 다양한 GUI 위젯을 제공합니다. 이들 위젯을 사용하여 사용자 인터페이스를 구성할 수 있습니다.</p> <p>③ <b>PyQt5.QtCore</b> 모듈은 Qt의 핵심 기능을 제공합니다. 이에는 이벤트 처리, 시그널 및 슬롯, 스레드 관리 등이 포함됩니다.</p> <p>④ <b>PyQt5.QtGui</b> 모듈은 그래픽 관련 클래스를 제공합니다. QPixmap은 이미지를 표시하고 처리하는데 사용되며, QIcon은 아이콘을 표시하는 데 사용됩니다.</p> <p>⑤ <b>os</b> 모듈은 운영 체제와 상호 작용하기 위한 다양한 기능을 제공합니다. 예를 들어, 파일 경로를 조작하거나 디렉토리를 탐색하는 기능 등이 있습니다.</p> <p>⑥ <b>sqlite3</b> 모듈은 SQLite 데이터베이스를 사용하기 위한 인터페이스를 제공합니다. SQLite는 파일 기반의 경량 DBMS입니다.</p> <p>⑦ <b>ftplib</b> 모듈은 FTP 프로토콜을 사용하여 원격 서버와 통신하는 기능을 제공합니다.</p>
<p>UI</p>	<pre>def __init__(self):     ① super().__init__() # 부모 클래스의 초기화 메서드를 호출합니다.     self.setWindowTitle("Image Viewer") # 윈도우의 제목을 설정합니다.     self.resize(1920, 1080) # 윈도우의 크기를 설정합니다.     self.background = QLabel(self) # 배경을 위한 레이블을 생성합니다.     self.background.setAlignment(Qt.AlignCenter) # 레이블의 정렬을 중앙으로 설정합니다.     self.background.resize(1920, 1080) # 레이블의 크기를 설정합니다.     # 배경 이미지를 설정하고, 레이블 크기에 맞게 조정합니다.     self.background.setPixmap(QPixmap('background.jpg').scaled(self.background.size(),   Qt.KeepAspectRatio, Qt.SmoothTransformation))      ② self.split_mode = 1 # 이미지 분할 모드를 설정합니다. 1은 한 개의 이미지를 표시, 2는 두 개, 4는 네 개를 표시합니다.     self.selected_images = [] # 선택된 이미지를 저장할 리스트를 초기화합니다.     self.search_query = "" # 검색 쿼리를 초기화합니다.     self.search_query_end = "" # 검색 쿼리 끝을 초기화합니다.      self.logo = QIcon() # 로고 아이콘을 생성합니다.     self.logo.addFile('logo.jpg') # 로고 아이콘에 이미지 파일을 추가합니다.     self.setWindowIcon(self.logo) # 윈도우 아이콘을 설정합니다.</pre> <p>① PyQt를 사용하여 이미지 뷰어 윈도우를 초기화하는 부분입니다. 윈도우의 제목과 크기를 설정하고, 배경 레이블을 생성하여 중앙에 배치합니다. 마지막으로 배경 이미지를 레이블 크기에 맞게 조정하여 배경 레이블에 설정합니다.</p> <p>② 이미지 뷰어의 초기 설정을 담당합니다. 이미지 분할 모드를 설정하고, 선택된 이미지와 검색 쿼리를 초기화합니다. 그리고 로고 아이콘을 생성하여 윈도우 아이콘으로 설정합니다.</p>

```

# 각 버튼을 생성하고, 크기, 위치, 스타일을 설정합니다. 클릭 시 연결될 함수도 설정합니다.
self.one_button = QPushButton(self)
self.one_button.resize(72, 72)
self.one_button.move(1225, 866)
self.one_button.setStyleSheet("background-color: rgb(56,79,97);background: transparent;border: none;")
self.one_button.clicked.connect(lambda: self.set_selection(1))
self.one_button.clicked.connect(self.display_images)

self.two_button = QPushButton(self)
self.two_button.resize(72, 72)
self.two_button.move(1320, 866)
self.two_button.setStyleSheet("background-color: rgb(56,79,97);background: transparent;border: none;")
self.two_button.clicked.connect(lambda: self.set_selection(2))
self.two_button.clicked.connect(self.display_images)

self.four_button = QPushButton(self)
self.four_button.resize(72, 72)
self.four_button.move(1425, 866)
self.four_button.setStyleSheet("background-color: rgb(56,79,97);background: transparent;border: none;")
self.four_button.clicked.connect(lambda: self.set_selection(4))
self.four_button.clicked.connect(self.display_images)

self.clear_selection_button = QPushButton(self)
self.clear_selection_button.resize(112, 24)
self.clear_selection_button.move(278, 260)
self.clear_selection_button.setStyleSheet("background-color: rgb(56,79,97); background: transparent; border: none;")
self.clear_selection_button.clicked.connect(self.image_list.clearSelection)

```

③ PyQt를 사용하여 여러 버튼을 생성하고 설정하는 부분입니다. 각 버튼은 크기 위치 스타일을 설정하고, 클릭 이벤트에 대한 연결을 만듭니다. 각 버튼 클릭 시 `set_selection` 함수가 호출되어 이미지 분할 모드를 설정하고, `display_images` 함수를 통해 이미지를 표시합니다. 마지막으로 `clear_selection_button` 클릭 시 이미지 선택을 초기화합니다.

```

# 이미지 리스트 위젯을 생성하고, 크기, 위치를 설정합니다. 아이템 더블 클릭 시 연결될 함수도 설정합니다.
self.image_list = QListWidget(self)
self.image_list.move(105, 287)
self.image_list.resize(290, 652)
self.image_list.itemDoubleClicked.connect(self.add_image)
self.image_list.setSelectionMode(QAbstractItemView.ExtendedSelection) # 여러 아이템을 선택할 수 있도록 설정합니다.
self.image_list.setDragEnabled(True) # 드래그를 활성화합니다.

self.image_paths = None # 이미지 경로를 저장할 변수를 초기화합니다.

if hasattr(self, 'image_paths'): # image_paths 속성이 있는 경우
    self.image_list.addItem(self.image_paths) # 이미지 경로를 리스트 위젯에 추가합니다.

```

④ PyQt를 사용하여 이미지 리스트를 생성하고 설정하는 부분입니다. 리스트 위젯의 위치 크기를 설정하고, 아이템 더블 클릭 이벤트와 선택 모드, 드래그 기능을 설정합니다. 그리고 이미지 경로를 저장할 변수를 초기화하고, 이미지 경로가 이미 설정되어 있다면 리스트 위젯에 추가합니다.

```

# 이미지를 표시할 레이블을 생성하고, 크기, 위치를 설정합니다. 이미지 스케일링을 활성화합니다.
self.one_label = QLabel(self)
self.one_label.setScaledContents(True)
self.one_label.setFixedSize(1350, 718)
self.one_label.move(486, 151)

# 두 개의 이미지를 표시할 레이블을 생성하고, 크기, 위치를 설정합니다.
self.two_labels = [QLabel(self), QLabel(self)]
for label in self.two_labels:
    label.setScaledContents(True)
    label.setFixedSize(675, 718)
self.two_labels[0].move(486, 151)
self.two_labels[1].move(1161, 151)

# 네 개의 이미지를 표시할 레이블을 생성하고, 크기, 위치를 설정합니다.
self.four_labels = [QLabel(self), QLabel(self), QLabel(self), QLabel(self)]
for label in self.four_labels:
    label.setScaledContents(True)
    label.setFixedSize(675, 359)
self.four_labels[0].move(486, 151)
self.four_labels[1].move(1161, 151)
self.four_labels[2].move(486, 510)
self.four_labels[3].move(1161, 510)

```

⑤ PyQt를 사용하여 이미지를 표시할 레이블들을 생성하고 설정하는 부분입니다. 각 레이블은 크기와 위치를 설정하고, 이미지 스케일링을 활성화합니다. 레이블들은 각각 하나, 두 개, 네 개의 이미지를 표시할 수 있도록 설정되어 있습니다.

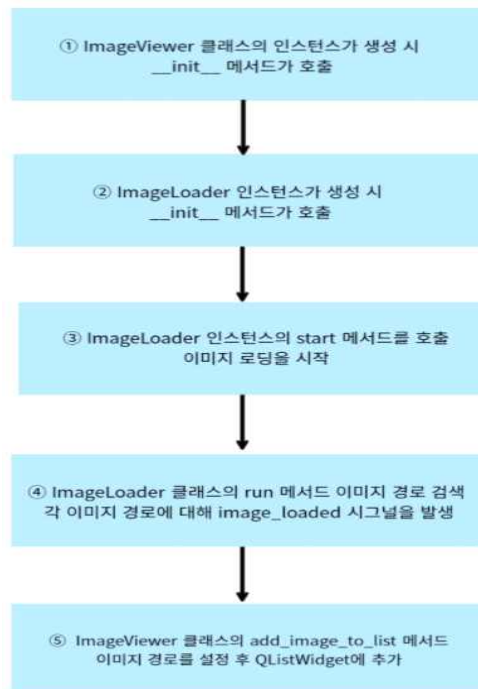
```
# 검색 입력 필드를 생성하고, 크기, 위치를 설정합니다. 텍스트 변경 시 연결될 함수도 설정합니다.
self.search_input_front = QLineEdit(self)
self.search_input_front.move(141, 183)
self.search_input_front.resize(245, 30)
self.search_input_front.setPlaceholderText("Search by front")
self.search_input_front.setStyleSheet("background: transparent; border: none;")
self.search_input_front.textChanged.connect(self.search_images)

self.search_input_back = QLineEdit(self)
self.search_input_back.setPlaceholderText("Search by back")
self.search_input_back.move(141, 229)
self.search_input_back.resize(245, 30)
self.search_input_back.setStyleSheet("background: transparent; border: none;")
self.search_input_back.textChanged.connect(self.search_images)
```

⑥ PyQt를 사용하여 두 개의 검색 입력 필드를 생성하고 설정하는 부분입니다. 각 입력 필드는 위치, 크기, 플레이스홀더 텍스트(사용자가 아무런 입력을 하지 않았을 때 QLineEdit 위젯에 표시되는 텍스트를 설정), 스타일을 설정하고, 텍스트 변경 이벤트에 대한 연결을 만듭니다. 텍스트가 변경되면 search\_images 함수가 호출됩니다.

해당 이미지처럼 QListWidget에 이미지 경로를 로딩하는 방법은 다음과 같습니다.

## 이미지 경로 (QListWidget)



① ImageViewer 클래스의 인스턴스가 생성될 때 \_\_init\_\_ 메서드가 호출됩니다. 이 메서드에서는 QListWidget와 QLineEdit 등의 위젯이 초기화되고, ImageLoader 클래스의 인스턴스인 self.loader가 생성됩니다.

```
self.loader = ImageLoader(self.image_path, self.search_query, self.search_query_end, self.image_list,
                           self.add_image_to_list, self.search_input_front, self.search_input_back)
```

② ImageLoader 인스턴스가 생성될 때 \_\_init\_\_ 메서드가 호출되며, 이 메서드에서는 ImageViewer 클래스의 add\_image\_to\_list 메서드를 image\_loaded 시그널에 연결합니다.

```
self.loader.image_loaded.connect(self.add_image_to_list)
```



③ ImageLoader 인스턴스의 start 메서드를 호출하여 이미지 로딩을 시작합니다. 이 메서드는 QThread의 start 메서드를 오버라이드하며, 이 메서드를 호출하면 QThread의 run 메서드가 호출됩니다.

```
self.loader.start()
```

④ ImageLoader 클래스의 run 메서드에서는 이미지 경로를 검색하고, 각 이미지 경로에 대해 image\_loaded 시그널을 발생시킵니다. 이 시그널은 ImageViewer 클래스의 add\_image\_to\_list 메서드에 연결되어 있으므로, 이 메서드가 호출됩니다.

```
class ImageLoader(QThread):
    image_loaded = pyqtSignal(QPixmap)

    def __init__(self, image_path, search_query, search_query_end, image_list, add_image_to_list, search_input_front, search_input_back):
        super().__init__()
        self.image_path = image_path
        self.search_query = search_query
        self.search_query_end = search_query_end
        self.image_list = image_list
        self.add_image_to_list = add_image_to_list
        self.search_input_front = search_input_front
        self.search_input_back = search_input_back
        self.image_pathss = None

    def run(self):
        while True:
            try:
                self.ftp = ftplib.FTP('127.0.0.1', user='admin', passwd='1234', timeout=30)
                print(f"FTP initialized: {self.ftp}") # Debugging line
                self.load_ftp_image_paths()
                self.image_paths = self.load_image_paths() # Save the paths to an instance variable
                self.image_pathss = self.image_paths # Update image_pathss with the loaded image paths
                self.search_images()
                for image_path in self.image_paths:
                    if not os.path.isfile(image_path):
                        print(f"Failed to load image: {image_path}")
                        continue
                    self.add_image_to_list(image_path) # Call the add_image_to_list method with the image path
                    QThread.msleep(30)
            except ftplib.all_errors as e:
                print(f"FTP connection error: {e}")
```

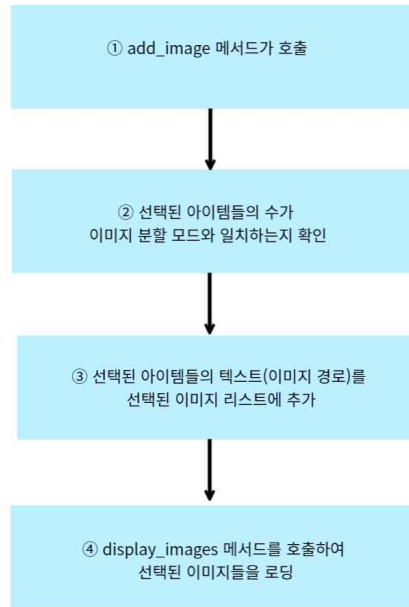
⑤ ImageViewer 클래스의 add\_image\_to\_list 메서드에서는 QListWidgetItem을 생성하고, 이 아이템에 이미지 경로를 설정한 후, 이 아이템을 QListWidget에 추가합니다.

```
def add_image_to_list(self, image_path):
    # QListWidget에 이미 아이템이 있는지 확인합니다.
    for i in range(self.image_list.count()): # QListWidget의 아이템 수만큼 반복합니다.
        if self.image_list.item(i).text() == image_path: # 현재 아이템의 텍스트가 입력된 이미지 경로와 같은지 확인합니다.
            return # 이미 아이템이 QListWidget에 있다면, 다시 추가하지 않고 함수를 종료합니다.

    # 아이템이 QListWidget에 없다면, 추가합니다.
    item = QListWidgetItem(image_path) # 새 QListWidgetItem을 생성하고, 이미지 경로를 텍스트로 설정합니다.
    self.image_list.addItem(item) # 생성한 아이템을 QListWidget에 추가합니다.
```

## 이미지 로딩 (QLabel)

이미지 로딩하는 순서는 다음과 같습니다.



① add\_image 메서드가 호출됩니다. 이 메서드에서는 QListWidget에서 선택된 아이템들을 가져옵니다.

```
# QListWidget에서 선택된 아이템들을 가져옵니다.
selected_items = self.image_list.selectedItems()
```

② 선택된 아이템들의 수가 이미지 분할 모드와 일치하는지 확인합니다. 일치하지 않으면 함수를 종료합니다.

```
# 선택된 아이템들의 수가 이미지 분할 모드와 일치하는지 확인합니다.
if len(selected_items) != self.split_mode: # 선택된 아이템의 수가 이미지 분할 모드와 다르다면,
    return # 함수를 종료합니다.
```

③ 선택된 아이템들의 텍스트(이미지 경로)를 선택된 이미지 리스트에 추가합니다.

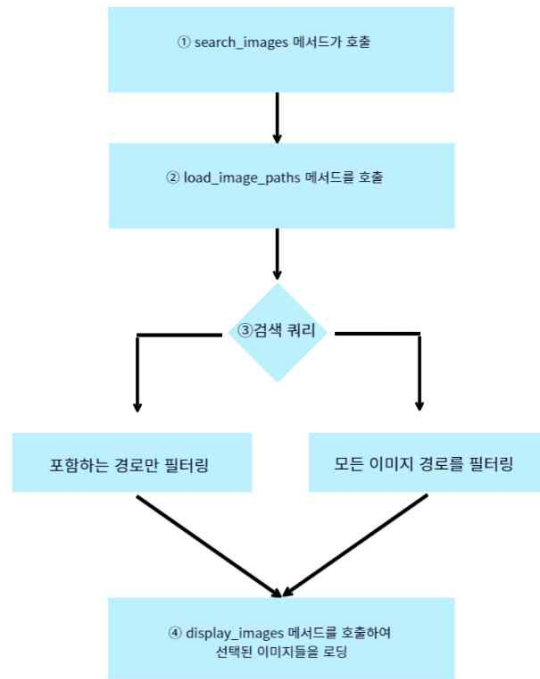
```
# 선택된 아이템들의 텍스트(이미지 경로)를 선택된 이미지 리스트에 추가합니다.
self.selected_images = [item.text() for item in selected_items]
```

④ display\_images 메서드를 호출하여 선택된 이미지들을 로딩합니다. 이 메서드에서는 선택된 이미지 리스트에 있는 각 이미지 경로에 대해 QPixmap을 생성하고, 이 QPixmap을 QLabel에 설정합니다.

```
def display_images(self):
    if self.split_mode == 1: # 이미지 분할 모드가 1인 경우,
        if self.selected_images: # 선택된 이미지가 있다면,
            # 선택된 이미지를 레이블에 표시
            self.one_label.setPixmap(self.selected_images[0])
            # 표시된 이미지를 제거
            self.selected_images.pop(0)
        else:
            # 이미지가 없으면 레이블을 비움
            self.one_label.clear()
            self.one_label.update()
    elif self.split_mode == 2: # 이미지 분할 모드가 2인 경우,
        for i, label in enumerate(self.two_labels): # 두 개의 레이블에 대해 반복합니다.
            if i < len(self.selected_images): # 선택된 이미지가 충분하다면,
                # 선택된 이미지를 레이블에 표시
                label.setPixmap(self.selected_images[i])
            else:
                # 이미지가 없으면 레이블을 비움
                label.clear()
                label.update()
    elif self.split_mode == 4: # 이미지 분할 모드가 4인 경우,
        for i, label in enumerate(self.four_labels): # 네 개의 레이블에 대해 반복합니다.
            if i < len(self.selected_images): # 선택된 이미지가 충분하다면,
                # 선택된 이미지를 레이블에 표시
                label.setPixmap(self.selected_images[i])
            else:
                # 이미지가 없으면 레이블을 비움
                label.clear()
                label.update()
```

## 검색 기능

본 화면과 같이 검색 기능은 다음과 같은 순서로 함수들이 실행되어 검색 기능을 수행합니다.



① search\_images 메서드가 호출됩니다. 이 메서드에서는 검색 입력 필드의 텍스트를 소문자로 변환하여 검색 쿼리를 설정합니다.

```

query_front = self.search_input_front.text().lower() # 검색 입력 필드(앞)의 텍스트를 소문자로 변환하여 저장합니다.
query_back = self.search_input_back.text().lower() # 검색 입력 필드(뒤)의 텍스트를 소문자로 변환하여 저장합니다.
self.search_query = query_front if query_front else None # 검색 쿼리를 설정합니다.
self.search_query_end = query_back if query_back else None # 검색 쿼리 끝을 설정합니다.
  
```

② load\_image\_paths 메서드를 호출하여 이미지 경로를 로드합니다. 이 메서드에서는 검색 쿼리가 있으면 해당 쿼리로 이미지 경로를 검색하고, 없으면 모든 이미지 경로를 검색합니다.

```

self.image_paths = self.load_image_paths() # 이미지 경로를 로드합니다.

def load_image_paths(self):
    with sqlite3.connect('image_db.db') as conn: # SQLite 데이터베이스에 연결합니다.
        c = conn.cursor() # 커서를 생성합니다.
        if self.search_query: # 검색 쿼리가 있으면,
            c.execute(f"SELECT path FROM Images WHERE front LIKE '{self.search_query}%'") # 해당 쿼리로 이미지 경로를 검색합니다.
        elif self.search_query_end: # 검색 쿼리 끝이 있으면,
            c.execute(f"SELECT path FROM Images WHERE back LIKE '{self.search_query_end}%'") # 해당 쿼리로 이미지 경로를 검색합니다.
        else: # 검색 쿼리가 없으면,
            c.execute("SELECT path FROM Images") # 모든 이미지 경로를 검색합니다.
        image_paths = [row[0] for row in c.fetchall()] # 검색 결과를 리스트로 저장합니다.
    return image_paths # 이미지 경로 리스트를 반환합니다.
  
```

③ 검색 쿼리가 있으면 이미지 경로 중 검색 쿼리를 포함하는 경로만 필터링하고, 없으면 모든 이미지 경로를 사용합니다.

```

if self.search_query or self.search_query_end: # 검색 쿼리가 있으면,
    # 이미지 경로 중 검색 쿼리를 포함하는 경로만 필터링합니다.
    filtered_paths = [path for path in self.image_paths if query_front in os.path.basename(path).lower() and query_back in os.path.basename(path).lower()]
else: # 검색 쿼리가 없으면,
    filtered_paths = self.image_paths # 모든 이미지 경로를 사용합니다.
  
```

④ 이미지 리스트를 초기화하고 필터링된 이미지 경로를 이미지 리스트에 추가합니다.

```

self.image_list.clear() # 이미지 리스트를 초기화합니다.
self.image_list.addItems(filtered_paths) # 필터링된 이미지 경로를 이미지 리스트에 추가합니다.
  
```



## V. 추진일정

w1	w2	w3
2024/05/27 - 2024/06/14		
학습 데이터 set 수집		
Label 작업		
	학습 진행	
	인식률 확인	
	센터 프로그램	
		서류 작업 및 최종확인

## VI. 기대효과

### 1. 개발 산출물의 지속가능성

#### 1.1. 시스템 유지보수 용이성

- ① 모듈화된 설계: 독립적 유지보수 가능, 전체 시스템 영향 최소화
- ② 오픈소스 프레임워크 사용: 커뮤니티 지원 및 업데이트로 지속성 보장
- ③ 문서화: 체계적 문서화로 유지보수와 기능 확장 지원

#### 1.2. 확장 가능성

- ① 모듈 확장성: 기능 추가 및 개선 용이
- ② API 및 인터페이스: 다른 시스템과 연동 용이

#### 1.3. 경제적 지속 가능성

- ① 비용 효율성: 인력 비용 절감, 운영 비용 낮춤
- ② 지속적인 ROI(Return on Investment): 효율적 차량 관리와 데이터 분석으로 지속 수익 창출

### 2. 데이터 활용에 따른 파급효과

#### 2.1. 교통 관리 및 정책 개선

- ① 교통 패턴 분석: 도시 교통 정책 및 도로 확장 등에 활용
- ② 교통 혼잡 완화: 실시간 데이터로 교통 흐름 최적화

#### 2.2. 보안 강화

- ① 출입 통제: 중요한 시설 출입 관리 및 비인가 차량 차단
- ② 사건 대응: 차량 이동 기록 분석으로 사건 해결 지원

#### 2.3. 환경 보호

- ① 탄소 배출 관리: 교통 혼잡 줄여 탄소 배출량 관리
- ② 친환경 정책 지원: 전기차 충전소 배치 및 정책 수립 지원

#### 2.4. 공공 서비스 개선

- ① 응급 서비스 지원: 실시간 데이터로 응급 차량 이동 경로 확보
- ② 공공 안전 강화: 범죄 예방 및 공공 안전 향상

## **VII. 인력 구성**

- 소프트웨어 엔지니어 및 시스템 관리자 (김동욱)

  - 시스템 통합

  - FTP 전송 구현

  - 데이터베이스 연동

  - 사용자 인터페이스 개발

  - 서버 및 네트워크 관리

  - 시스템 유지보수

- 데이터 과학자 및 머신 러닝 엔지니어 (조재경)

  - 데이터셋 준비 및 전처리

  - Yolov5 모델 학습 및 최적화

  - 데이터 분석 및 시각화

  - 모델 배포 및 유지보수

## [별첨 1] YOLOv5 요구사항

```
Base -----
gitpython>=3.1.30
matplotlib>=3.3
numpy>=1.23.5
opencv-python>=4.1.1
pillow>=10.3.0
psutil # system resources
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
thop>=0.1.1 # FLOPs computation
torch>=1.8.0
torchvision>=0.9.0
tqdm>=4.64.0
ultralytics>=8.0.232
#protobuf<=3.20.1
# https://github.com/ultralytics/yolov5/issues/8012
Logging-----
# tensorboard>=2.4.1
# clearml>=1.2.0
# comet
#
Plotting -----
pandas>=1.1.4
seaborn>=0.11.0

Export -----
# coremltools>=6.0 # CoreML export
# onnx>=1.10.0 # ONNX export
# onnx-simplifier>=0.4.1 # ONNX simplifier
# nvidia-pyindex # TensorRT export
```

```

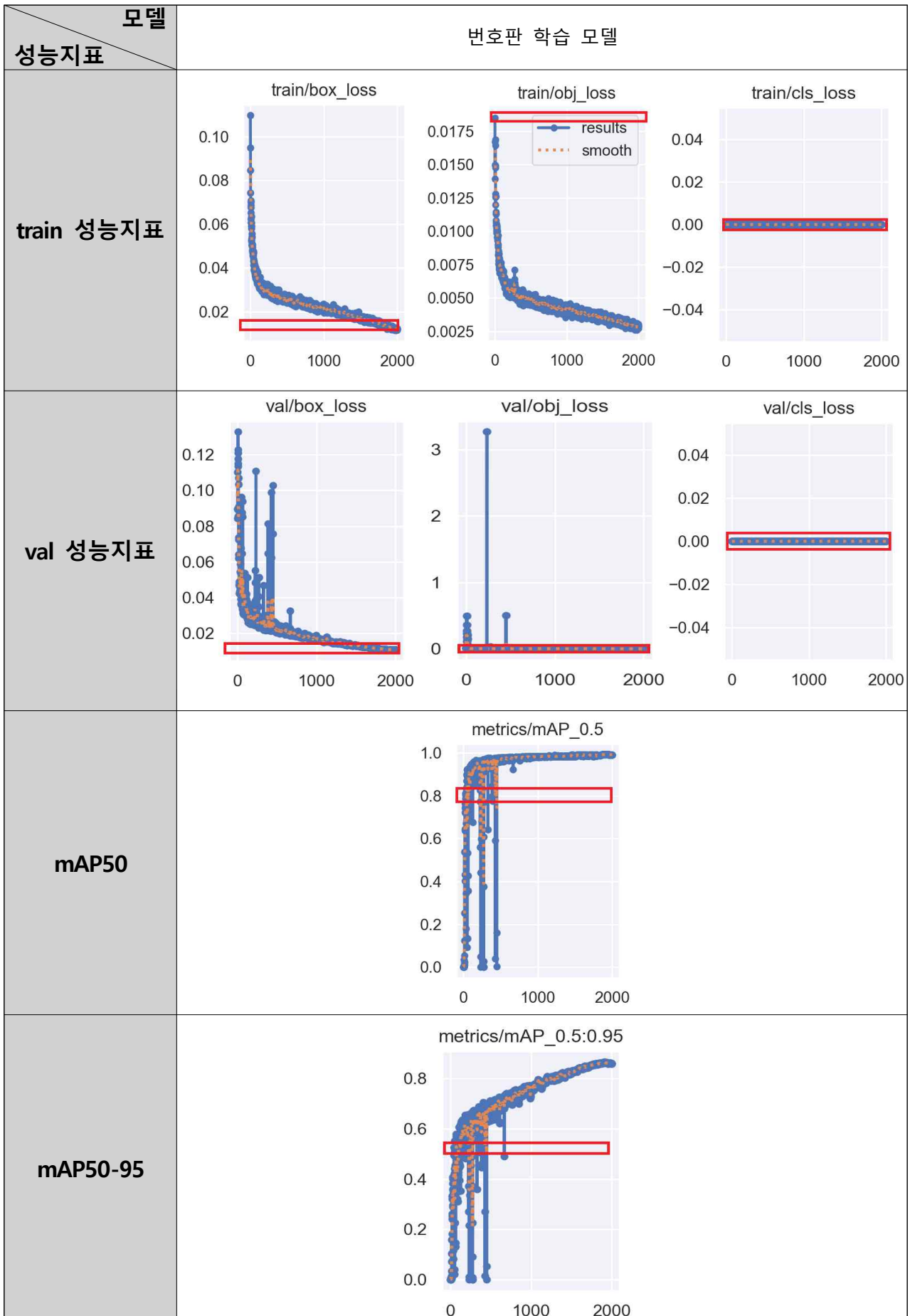
# nvidia-tensorrt # TensorRT export
# scikit-learn<=1.1.2 # CoreML quantization
# tensorflow>=2.4.0,<=2.13.1 # TF exports (-cpu, -aarch64,
-macos)
# tensorflowjs>=3.9.0 # TF.js export
# openvino-dev>=2023.0 # OpenVINO export
#
-----
setuptools>=65.5.1 # Snyk vulnerability fix
# tritonclient[all]~=2.24.0
#
-----
# ipython # interactive notebook
# mss # screenshots
# albumentations>=1.0.3
# pycocotools>=2.0.6 # COCO mAP
wheel>=0.38.0 # not directly required, pinned by Snyk to avoid a
vulnerability

```


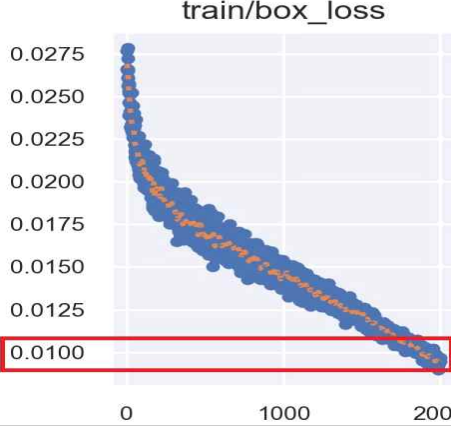
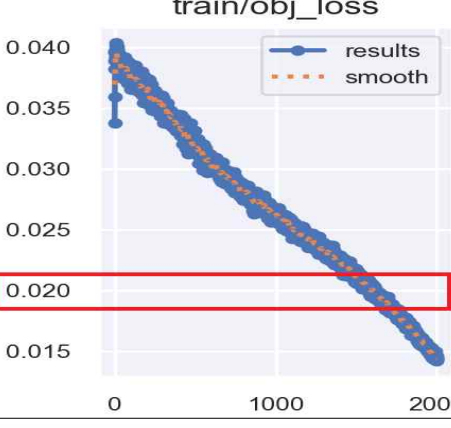
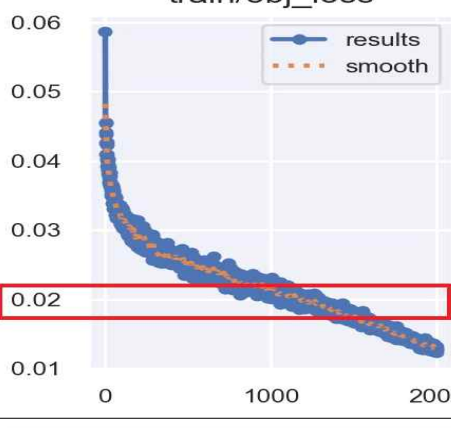
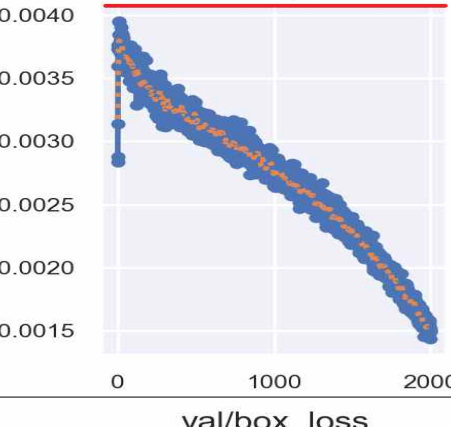
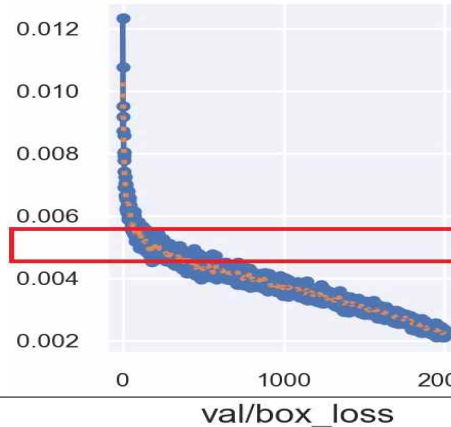
Deploy

Extras

## [별첨 2] 번호판 인식 학습 결과



### [별첨 3] 문자 인식 학습 결과

성능지표 \ 모델	전이 학습	추가 학습
train_box_loss		
train_obj_loss		
train_cls_loss		
val_box_loss	