SSL SEMINAR

Deep Learning Based Vehicle Recognition Sys.

Kumoh National Institute of Technology
System Software Lab.
20200573 서준혁

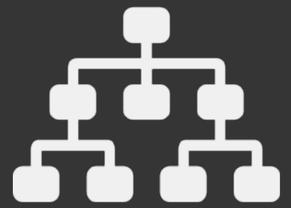
SSL SEMINAR

PART 01

DATASET CHECK

데이터셋 오류 확인 및 보완

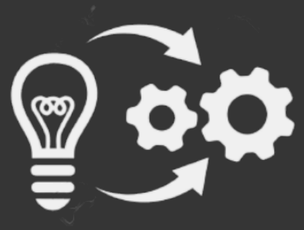
PART 02



DATASET 전처리

데이터셋 세부 내용 확인 및 전처리

PART 03



기타 사항

향후 계획 및 고민

SSL SEMINAR

PART
01

CHECK

데이터셋 오류 확인 및 보완

DATASET 오류 확인

저번 시간에 처리한 DATASET에서 처리한 이미지 개수와 라벨 개수가

서로 맞지 않는 것을 확인했었다.

따라서, 기존 코드에서 어떤 파일이 짝이 없는지 확인하기 위해

새로운 코드를 추가하여 파일 이름을 출력해서 확인할 수 있도록 했다.

```
Processed Image : 647
Processed Labels : 644
```

```python
# Read JSON Type Files Recursively
# Not Finished
def readJson(dir):
    labelCount = 0
    imageCount = 0
    labels = []
    images = []

    for filename in glob.iglob(dir + "**/**", recursive=True):
        if ("_T_" in filename and filename.endswith(".json")):
            f = open(filename, 'rt', encoding='UTF8')
            json_data = json.load(f)

            for item in json_data["learningDataInfo"]["objects"]:
                if item["classId"] == "P00.차량전체":
                    data = J2Text(item, json_data)
                    title = filename.replace(".json", ".txt")
                    writeTxt(data, title)
                    labelCount += 1
                    title = title.split("\\")[6]
                    labels.append(title.replace(".txt", ''))

        elif ("_T_" in filename and filename.endswith(".jpg")):
            image_read = np.fromfile(filename, np.uint8)
            image = cv2.imdecode(image_read, cv2.IMREAD_COLOR)
            title = filename
            writeImg(image, title)
            imageCount += 1
            title = title.split("\\")[6]
            images.append(title.replace(".jpg", ''))

    unfair = 0
    for i in range(0, 644):
        if images[i] not in labels:
            print("** ", images[i], " **")
            unfair += 1

    print("\nProcessed Image :", imageCount)
    print("Processed Labels :", labelCount)
    print("Unfair Count :", unfair)
```

DATASET 오류 확인

```
**  C_211119_BE_010_17_GR_A_T_03_013  **
**  C_211209_BM_038_20_GR_A_T_03_012  **
**  C_211220_SS_085_17_GR_C_T_03_006  **

Processed Image : 647
Processed Labels : 644
Unfair Count : 3
```

다음과 같은 파일들이 짝이 없는 것을 확인했으며 해당 파일에

어떤 문제가 있는지 직접 열어 확인해봤다.

문제가 있는 파일들은 공통적으로, 파일 이름에 전체 외관을 뜻하는

_T_가 포함되어 있지만, 전체 외관 정보가 없는 경우였다.

따라서 이렇게 짝이 없는 데이터를 삭제할 필요가 있어 보였다.


```
"learningDataInfo": {
    "path": null,
    "LearningDataId": "C_211119_BE_010_17_GR_A_T_03_013",
    "fileExtension": "json",
    "objects": [
        {
            "classId": "P07.앞도어",
            "annotation": "bbox",
            "coords": {
                "tl": {
                    "x": 758,
                    "y": 356
                },
                "tr": {
                    "x": 1118,
                    "y": 356
                },
                "bl": {
                    "x": 758,
                    "y": 696
                },
                "br": {
                    "x": 1118,
                    "y": 696
                }
            },
            "left": 758,
            "top": 356,
            "width": 360,
            "height": 340,
            "angle": 0
        },
```

실전 투입

지금까지는 샘플 데이터셋을 전처리해보았다. 짝이 없는 데이터 삭제는 추후 구현하기로하고,

실제 데이터셋 처리를 진행하는 중, 파이썬 구문 오류가 발생했다.

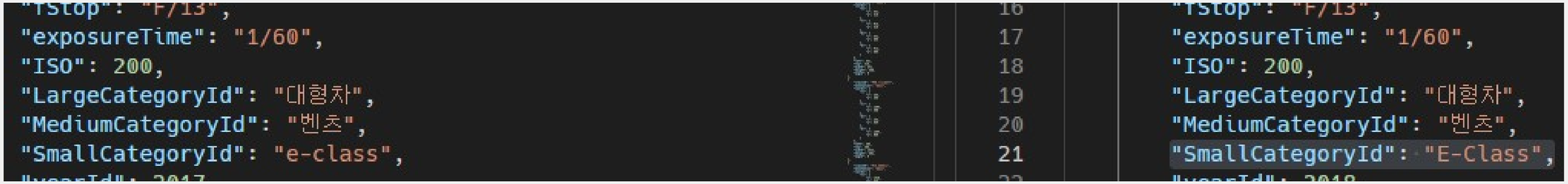
차량 모델을 적어놓은 리스트에서 이름을 찾지 못했다는 오류였다.

```
Traceback (most recent call last):
  File "C:\Users\ssam2\Desktop\yolov5\dataset\label.py", line 329, in <module>
    readJson(rootDir)
  File "C:\Users\ssam2\Desktop\yolov5\dataset\label.py", line 58, in readJson
    data = J2Text(item, json_data)
  File "C:\Users\ssam2\Desktop\yolov5\dataset\label.py", line 178, in J2Text
    labels = str(str(model[2]) + " " + str(x) + " " + str(y) + " " + str(width) + " " + str(height))
IndexError: list index out of range
```

실전 투입

확인해본 결과, 벤츠 E-CLASS 차량 파일 일부에서 소문자로 차량 모델명이 적혀있음을 확인했다.

다른 차종을 제외하고 해당 차종에서만 나타난 문제라 해당 부분을 무시할 수 있도록 코드를 수정했다.



실전 투입

약 1시간 30분 동안 데이터셋을 처리한 결과, 영상 데이터 322,664장 중, 98,882장이 처리된 것을 확인할 수 있었고,

라벨 데이터는 약간의 오차가 있는 98,395개로, 오차 개수는 729개임을 확인했다.

```
Processing C_211213_VV_095_18_BK_A_T_03_018.txt
Runtime : 5330.3 sec

Processing C_211213_VV_095_18_BK_A_T_03_022.txt
Runtime : 5330.3 sec

Processing C_211213_VV_095_18_BK_A_T_03_023.txt
Runtime : 5330.31 sec

Processing C_211104_VV_095_19_GR_A_T_02_007.txt
Runtime : 5330.31 sec

Processing C_211104_VV_095_19_GR_A_T_03_005.txt
Runtime : 5330.31 sec


Processed Image : 98882
Processed Labels : 98395
```

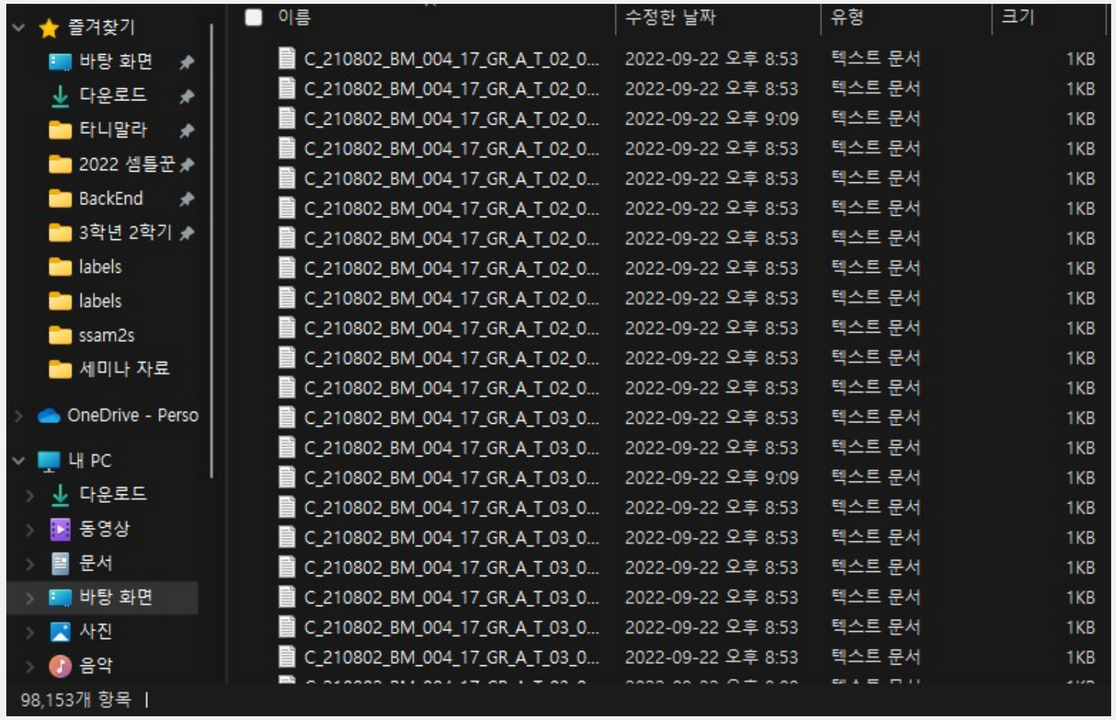
```
** C_211223_VO_075_17_GR_B_T_02_003 **

** C_211223_VO_075_17_GR_B_T_03_010 **

** C_211227_HO_003_17_GR_C_T_03_007 **

** C_211227_HO_003_17_GR_C_T_03_019 **

** C_211227_HO_003_21_GR_A_T_03_007 **

UNFAIR COUNT : 729
```

실전 투입

오차에 해당하는 729개의 이미지 및 라벨 데이터는 추가 코드 작성을 통한 삭제 과정을 거쳤고,

다음과 같이 초기 데이터셋을 성공적으로 구성하였다.

```
Processed Labels : 98153
Processed Images : 98153
```



SSL SEMINAR

PART 02

Dataset 전처리

실 데이터셋 전처리

데이터 분할

데이터셋을 제공한 AI-HUB의 데이터셋 활용 설명서를 읽어보면, TRAIN / VALID / TEST 데이터셋을

8 : 1 : 1의 비율로 구성하라고 적혀있는 것을 확인할 수 있다.

따라서, 전처리가 끝나고 한 폴더에 모여있는 196,306개의 데이터를 비율에 맞게 분류를 해보자.

	학습(Training)	검증(Validation)	시험(Test)
개요	– 모델 성능 지표를 올리기 위해 입력한 사진과 이에 해당하는 정답 출력값으로 구성된 데이터를 반복적으로 학습하는 과정 – 일반적으로 데이터를 2의 제곱 단위로 조금씩 묶어서(mini batch, 단위는 4,8,16, 32, ...) 학습에 적용하며, 학습에 사용하는 계산 자원이 클수록 큰 단위를 적용 – 반복 학습을 통해 성능 개선 – 딥러닝 모델은 일반적으로 데이터가 많을수록 성능이 개선됨	– 학습 도중 모델 성과 평가 및 비교 – 모델의 성과 지표는 알고리즘별로 설정된 학습 가이드 함수(loss function)과 실제 응용에서 고려할 평가 척도를 함께 살펴본다 – 모델의 학습이 더 필요한지(과소적합), 너무 학습을 많이 하였는지(과적합)를 성과 지표들을 통해 확인 – 일반적으로 과적합이 시작되는 시점 또는 성과 지표가 수렴하기 시작하는 시점의 모델을 선택	– 학습 완료된 모델 성능 시험 – 학습에 사용하지 않은 별개 데이터 적용 – 검증 단계에서 확인한 모델의 성과 지표뿐만 아니라, 실제 응용 단계에서 고려할 다양한 특성을 만족하는지 확인
필요 데이터	– 취득 데이터의 80%	– 취득 데이터의 10% – 차량 종류의 비율이 학습 데이터와 유사해야 함	– 취득 데이터 10% – 차량 종류 비율이 학습 데이터와 유사해야 함

데이터셋의 파일 리스트를 추출한 뒤에 대충 셔플하여 0.8, 0.1, 0.1로 각각 곱해 배분하면 안되나라고 쉽게 생각할 수 있지만,

그렇게 데이터셋을 분할했다가는 추후 학습을 거친 뒤에 모델에 OVERFITTING (과적합) 또는 UNDERFITTING (과소적합)이 생길 수 있다.

예를 들면 다음과 같은 경우이다.

0.8로 곱한 TRAIN 폴더에는 한 모델의 레드 색상만 들어가고, 0.1로 곱한 VALID 폴더에는 한 모델의 블루 색상만 들어간다면?

TRAIN, VALID 폴더에는 빨강, 파랑 모델이 섞여 들어갔지만, TEST 폴더에는 블랙 색상의 모델만 들어간다면?

학습하지 않은 색상에 대한 모델은 인식하지 못할 수도 있다.

이는 객체의 색상에 대한 AUGMENTATION이

제대로 이루어지지 않았음을 뜻한다.

따라서 모델의 각 컬러까지 골고루 분류할 필요가 있다.

TXT 파일은 STD 파일 입출력으로, 이미지 파일은 대용량 이미지 파일에 대한 처리가 빠른 NUMPY 배열을 이용해 구현했다. (스킵)

OS.LISTDIR() 함수를 통해 불러온 파일 이름 리스트인 DATALIST를 SETMODELS 함수에 전달하면,

반복문으로 돌면서 SPLIT 함수를 이용해 모델 번호를 추출한다.

추출한 아이디를 딕셔너리 타입인 MODELLIST에

모델 번호 : [파일명, 파일명, ...] 형태로 넣어 리턴한다.

```python
# Divide Files by Model Numbers
# Key : Model Numbers / Value : File Names(List)
def setModels(modellist, datalist):
    for name in datalist:
        # C_211201_BM_002_18_BL_B_T_03_004
        smallID = name.split("_")[3]

        if smallID not in modellist:
            modellist[smallID] = []

        modellist[smallID].append(name)

    return modellist
```

전달 받은 MODELLIST를 반복문을 이용해 돌면서, 각 파일명을 NAMELIST에 전달한다.

NAMELIST에 저장된 각각의 파일에 접근하여 해당 파일의 색상을 추출한다.

그리고 다시 각 색상에 맞게 파일명을 저장한다.

라벨과 이미지는 한 쌍이기 때문에 확장자를 떼고 이름만 하나 저장한다.

최종적으로 색상 : [파일명, 파일명, ...] 의 형태로 저장된다.

WH : [EX1, EX2, ...]
BK : [EX3, EX4, ...]

```python
for model in modellist:
    namelist = modellist[model]
    colors = {}

    for name in namelist:
        # C_211201_BM_002_18_BL_B_T_03_004
        color = name.split("_")[5]

        if color not in colors:
            colors[color] = []

        # Remove File Name Extension
        if name.endswith(".txt"):
            name = name.replace(".txt", '')
        elif name.endswith(".jpg"):
            name = name.replace(".jpg", '')

        colors[color].append(name)
```

COLORS에 저장된 각 색상에 접근하면서 라벨 이름과 이미지 이름이 같이 저장됐을 수 있으므로

SET 함수를 이용해 중복을 제거하고, RANDOM.SHUFFLE을 이용해

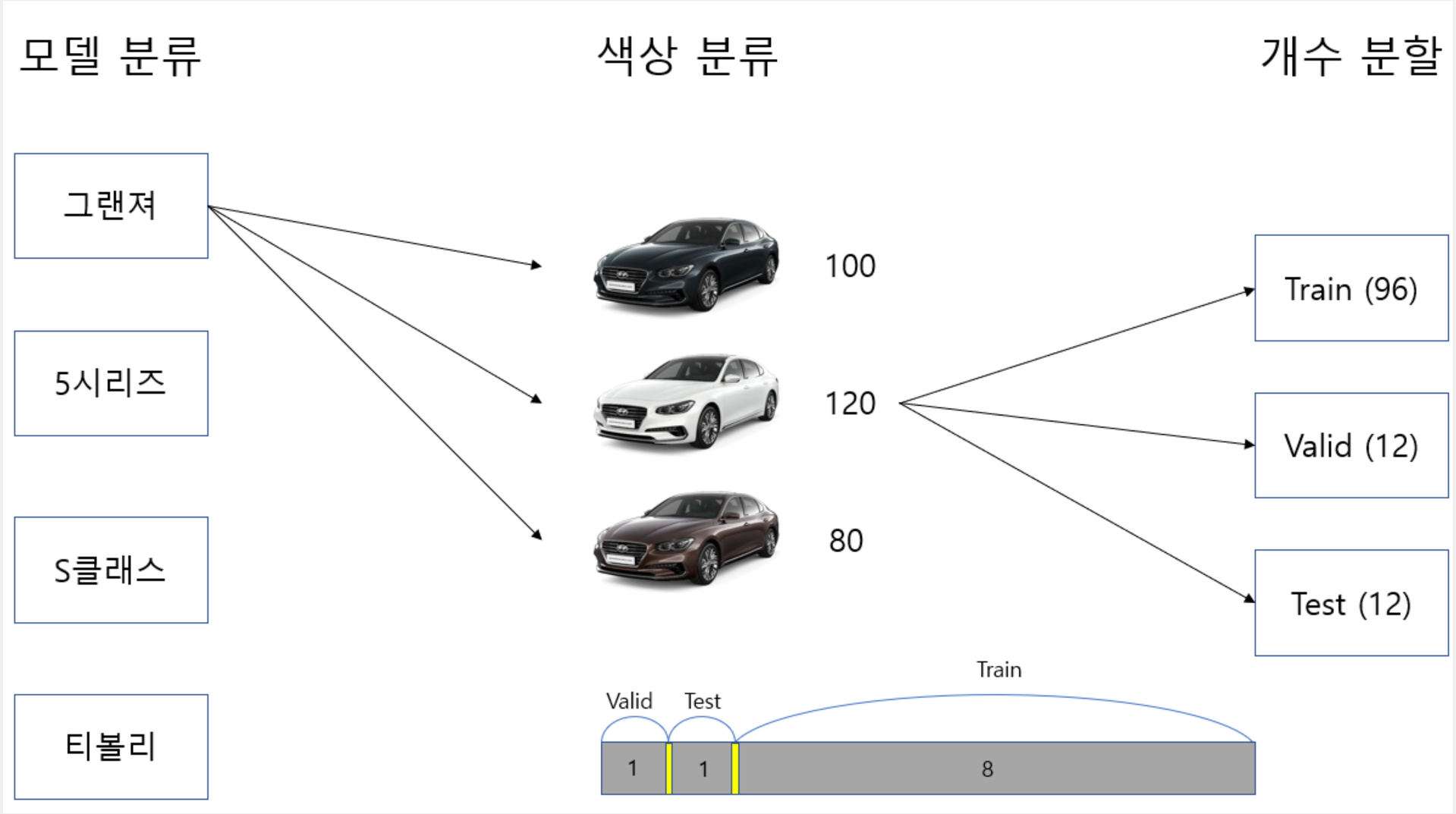
리스트를 한번 섞어준다. (사진 종류에 대한 AUGMENTATION 적용)

전체 사진 파일의 개수를 구하여 해당 개수의 0.1을 곱한 길이만큼

각각 VALID, TEST로 분류하고, 남는 0.8을 TRAIN으로 이동시키도록 한다.

```
for color in colors:
    # Remove Duplication
    colors[color] = list(set(colors[color]))

    # Shuffle List
    random.shuffle(colors[color])

    len_total = len(colors[color])
    len_valid = int(len_total * 0.1)
    len_test = int(len_total * 0.1) + len_valid

    list_valid = []
    list_test = []
    list_train = []

    for i in range(0, len_total):
        if i < len_valid:
            list_valid.append(colors[color][i])
        elif i > len_valid and i < len_test:
            list_test.append(colors[color][i])
        else:
            list_train.append(colors[color][i])
```

앞선 과정들을 시각화 하면 다음과 같다.

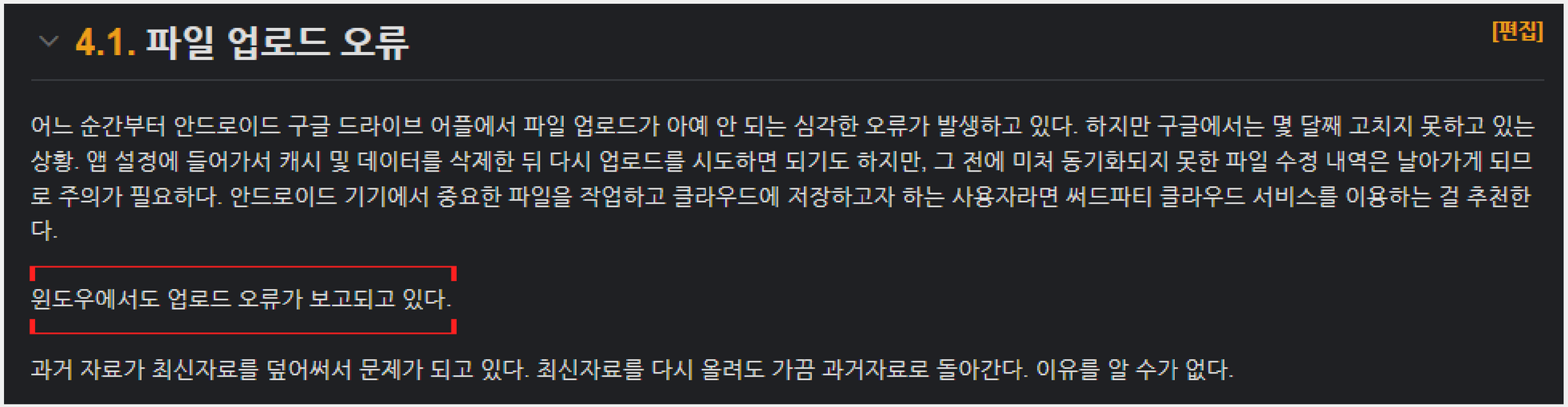


```
Train : 81  Valid : 9  Test : 8
70180 Images are Processed...

71.5 % Processed...  132047.36 sec


** 014 **
Train : 518  Valid : 64  Test : 63
Train : 738  Valid : 92  Test : 91
Train : 231  Valid : 28  Test : 27
Train : 603  Valid : 75  Test : 74
72784 Images are Processed...

74.15 % Processed...  138046.61 se


** 074 **
Train : 55  Valid : 6  Test : 5
Train : 154  Valid : 19  Test : 18
Train : 233  Valid : 28  Test : 27
Train : 26  Valid : 3  Test : 2
73360 Images are Processed...

74.74 % Processed...  139399.69 se


** 069 **
Train : 310  Valid : 38  Test : 37
```

약 38시간 동안 처리를 돌렸더니 74퍼센트가 진행되었고,

```
74.74 % Processed...  139399.69 sec


** 069 **
Train : 310  Valid : 38  Test : 37
Train : 313  Valid : 38  Test : 37
Train : 53  Valid : 6  Test : 5
Train : 22  Valid : 2  Test : 1
74222 Images are Processed...

75.62 % Processed...  141453.36 sec


** 099 **
Train : 342  Valid : 42  Test : 41
Train : 640  Valid : 79  Test : 78
Traceback (most recent call last):
  File "G:\내 드라이브\yolov5\dataset\split.py", l
    getColors(modellist)
  File "G:\내 드라이브\yolov5\dataset\split.py", l
    moveFiles(train, "train")
  File "G:\내 드라이브\yolov5\dataset\split.py", l
    cv2.imwrite(jpgName, image)
cv2.error: OpenCV(4.6.0) D:\a\opencv-python\opencv
Assertion failed) !_img.empty() in function 'cv::i

PS G:\내 드라이브\yolov5\dataset>
```

밤 사이 네트워크 오류로 인터넷 연결이 끊기면서 오류가 떠서 모두 날아갔다...



4.1. 파일 업로드 오류 [편집]

어느 순간부터 안드로이드 구글 드라이브 어플에서 파일 업로드가 아예 안 되는 심각한 오류가 발생하고 있다. 하지만 구글에서는 몇 달째 고치지 못하고 있는 상황. 앱 설정에 들어가서 캐시 및 데이터를 삭제한 뒤 다시 업로드를 시도하면 되기도 하지만, 그 전에 미처 동기화되지 못한 파일 수정 내역은 날아가게 되므로 주의가 필요하다. 안드로이드 기기에서 중요한 파일을 작업하고 클라우드에 저장하고자 하는 사용자라면 써드파티 클라우드 서비스를 이용하는 걸 추천한다.

윈도우에서도 업로드 오류가 보고되고 있다.

과거 자료가 최신자료를 덮어써서 문제가 되고 있다. 최신자료를 다시 올려도 가끔 과거자료로 돌아간다. 이유를 알 수가 없다.

알고 보니 윈도우에서 알려진 업로드 오류가 있었고, 장시간 업로드를 하면서 중간에 에러가 발생했던 것 같다.

또한, 구글 드라이브는 네트워크 요청/응답을 거친 후 SSD가 아닌 HDD 클라우드 스토리지에서 파일 입출력을

진행하기 때문에 처리 속도가 상당히 느린 점도 사례가 많았다..

따라서, 파일 분할을 로컬에서 진행하고 구글 드라이브로 업로드하기로 방법을 변경했다.

약 40시간 동안 50퍼센트를 처리했던 것과 달리, 로컬에서 진행한 결과

약 8000초 (약 2시간 10분) 만에 처리를 끝냈다...

별도로 제작한 개수 카운팅 프로그램을 돌린 결과 각 폴더에 적절한 비율로

파일들이 배치된 것을 확인할 수 있었다.

```
** 081 **
Train : 26  Valid : 2  Test : 1
98153 Images are Processed...

100.0 % Processed...  8135.25 sec


!! 196306 !!


!! 98153 !!

MODELS COLORS ARE SAVED TO LIST...
```

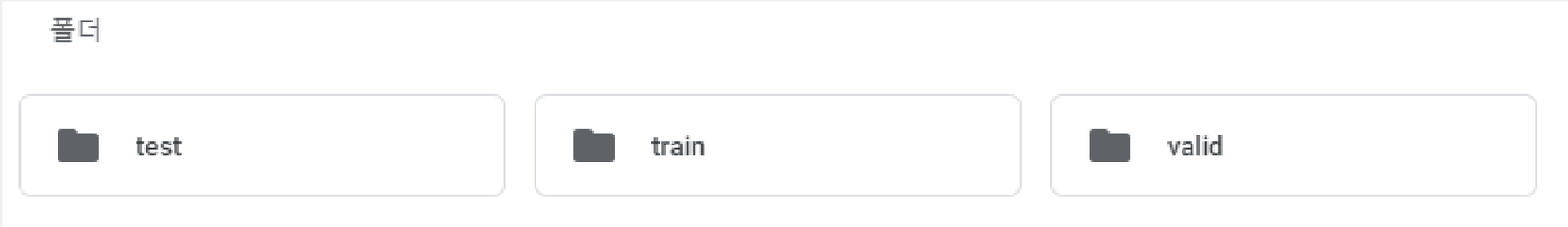
```
PS C:\Users\ssam2\Desktop\yolov5\dataset> python .\finalcheck.py
Train image count : 79186
Train label count : 79186
Valid image count : 9660
Valid label count : 9660
Test file count : 9307
```

느낀 점

매일 밤마다 처리를 돌려놓고 오류가 발생하고를 반복하면서 많이 힘들었지만

그래도 문제를 찾고 해결하면서 뿌듯했던 것 같다.

이제 데이터셋 구축은 끝났으니, 학습을 준비해보아야겠다.



SSL SEMINAR

Deep Learning Based
Vehicle Recognition Sys.

Contact

발표자 | 서준혁

TEL. 010-9802-5053 **E-mail** ssam2s@naver.com