

Archive: 문서 정리

☰ 태그

사전조사

<프로젝트 명>

1. 어떤 데이터를 사용했는가?

DATA SET : 한국인 감정인식을 위한 복합 영상

2. 데이터를 어떻게 다뤘는가?

2-1.폴더 구조 변경

2-2. 데이터 전처리와 코드 분석

간략 코드 구조

tools

1. 어떤 데이터를 사용했는가?

DATA SET : 한국인 감정인식을 위한 복합 영상

<https://www.aihub.or.kr/aihubdata/data/dwld.do?currMenu=115&topMenu=100&dataSetSn=82&beforeSn=27716&inqrySeCode=&intrstDataAt=N&reloadYn=N&useAt=>

카테고리(종)	이미지 개수(개)	분포(%)
기쁨	70,735	14.47%
당황	70,457	14.41%
분노	68,835	14.08%
불안	69,965	14.31%
상처	70,103	14.34%
슬픔	70,508	14.42%
중립	68,173	13.94%
평균	69,825	14.28

데이터 구조

• 데이터 구성



파일명 형식: 게시자 ID 해시값 + "_" + 성별 + "_" + 연령 + "_" + 감정 + "_" + 배경 + "_" + 업로드 번호

예시) "0d907746925fb35712cca733630efe057352a022c5af5723fe93a525abb2f9e_남_30_불안_공공시설&종교&의료시설_001-001.jpg"

- 이미지 학습용 데이터의 원본 파일명에는 감정, 전문인/일반인 여부로 구별되며, 성별 및 연령대 정보는 '원본 파일명' 과 일치하는 JSON 파일 형태로 구성
- JSON 파일과 이미지 데이터는 1:N 관계로, 하나의 JSON 파일 안에 모든 이미지 데이터에 해당하는 메타데이터가 포함되어 있음

• 어노테이션 포맷

No	항목		길이	타입	필수여부	비고
	한글명	영문명				
1	성별	gender	1	String	Y	
2	나이대	age	3	Int	Y	
3	업로더 감정 정보	faceExp_uploader	10	String	Y	
4	업로더 배경 정보	bg_uploader	20	String	Y	
5	어노테이터 A bounding box 정보	faceBB_A	2	List	Y	boxes: [maxX, maxY, minX, minY: 좌표 값 / label: 감정 정보]
6	어노테이터 B bounding box 정보	faceBB_B	2	List	Y	위와 동일
7	어노테이터 C bounding box 정보	faceBB_C	2	List	Y	위와 동일

2. 데이터를 어떻게 다뤘는가?

2-1.폴더 구조 변경

- 일반적으로 YOLO 모델을 학습시키기 위한 데이터셋은 다음과 같은 구조를 가진다

```
dataset/
|
├── images/
|   |
|   ├── train/
|   │   └── image1.jpg
```

```

|
|   |
|   |   | image2.jpg
|   |   | ...
|   |   |
|   |   | val/
|   |   |   |
|   |   |   | image3.jpg
|   |   |   | image4.jpg
|   |   |   | ...
|   |   |
|   |   | labels/
|   |   |   |
|   |   |   | train/
|   |   |   |   |
|   |   |   |   | image1.txt
|   |   |   |   | image2.txt
|   |   |   |   | ...
|   |   |   |   | val/
|   |   |   |   |   |
|   |   |   |   |   | image3.txt
|   |   |   |   |   | image4.txt
|   |   |   |   |   | ...

```

```

dataset/
|
|   | images/
|   |   |
|   |   |   | train/
|   |   |   |   |
|   |   |   |   | anger/
|   |   |   |   |   |
|   |   |   |   |   | image1.jpg
|   |   |   |   |   | image2.jpg
|   |   |   |   |   | ...
|   |   |   |   |
|   |   |   |   | ...
|   |   |   |
|   |   |   | val/
|   |   |   |   |
|   |   |   |   | anger/
|   |   |   |   |   |
|   |   |   |   |   | image1.jpg
|   |   |   |   |   | image2.jpg
|   |   |   |   |   | ...
|   |   |   |   |
|   |   |   |   | ...
|   |   |   |
|   |   |   | test/
|   |   |   |   |
|   |   |   |   | anger/
|   |   |   |   |   |
|   |   |   |   |   | image3.jpg

```



```
/test_hap
/test_nor
/test_pai
/test_sad
기쁨.json
당황.json
분노.json
```

<FROM> - selected data

```
/images
  /anger
  /happy
  /sad
  /emb
/labels
  /anger
  /happy
  /sad
  /emb
/test_set
  /anger
  /anxiety
  /embarrass
  /happy
  /normal
  /pain
  /sad
  /anger.json
  /anxiety.json
  /embarrass.json
  /happy.json
  /normal.json
  /pain.json
  /sad.json
```

<TO>

```

/images
  /train
    /anger
    /happy
    /sad
    /emb
  /test
    /anger
    /happy
    /sad
    /emb
/labels
  /train
    /annotation.json
  /test
    /annotation.json

```

2-2. 데이터 전처리와 코드 분석

▼ 간략 코드 구조

```

est_wassup_03/
  configs/ # yolo yaml 파일
  core
    configs # config class
    datasets # custom dataset class
    models
    utils
  train_tools # 실제 돌아갈 script
  tools # data_tools로 변경 예정, data preprocessing관련
script
  inference # infer_tools로 변경 예정, inference관련 scr
ipt

```

```

utils
data # 원본 데이터(서버에서 원하는 비율로 sampling 된 )
features # 원본 데이터에서 구조, split, crop 등 어떻게든 전
처리된 것
results # train, eval, infer의 결과물 저장
    detect
        train
        eval
        infer
    classify
    2d_gen
    3d_gen

```

▼ tools

▼ convert_label_to_coco.py

→ 주어진 json 형식의 annotation 파일을 읽어와 coco 형식에 맞게 변환하는 코드이다

- **COCO_ANNOT** : coco 형식의 annotation을 초기화하는 딕셔너리다
- **CAT_MAPPER** : 얼굴 표정을 coco 카테고리 매핑하는 딕셔너리다

```

import argparse
from copy import deepcopy
# import cv2
from datetime import datetime
import json
import os
from os import walk
from os.path import isfile, exists
from PIL import Image
import sys

COCO_ANNOT = {
    "info": {
        "description": "facial-expression-classification",
        "url": "https://aihub.or.kr/aihubdata/data/view."

```

```

        "version": "1.2",
        "year": 2023,
        "contributor": "한국과학기술원",
        "date_created": "2023/10/10"
    },
    "images": [
        # {
        #     "id": 1, # "id" must be int >= 1
        #     "width": 426,
        #     "height": 640,
        #     "file_name": "xxxxxxxxx.jpg",
        #     "date_captured": "2013-11-15 02:41:42"
        # }
    ],
    "annotations": [
        # {
        #     "id": 1, # "id" must be int >= 1
        #     "category_id": 1, # "category_id" must be in
        #     "image_id": 1, # "image_id" must be int >= 1
        #     "bbox": [86, 65, 220, 334] # [x,y,width,height]
        # }
    ],
    "categories": [
        # {
        #     "id": 2, # "id" must be int >= 1
        #     "name": "happy"
        # }
        { "id": 1, "name": "anger" },
        { "id": 2, "name": "anxiety" },
        { "id": 3, "name": "embarrass" },
        { "id": 4, "name": "happy" },
        { "id": 5, "name": "normal" },
        { "id": 6, "name": "pain" },
        { "id": 7, "name": "sad" },
    ]
}

CAT_MAPPER = {

```



```

"분노": 1,
"불안": 2,
"당황": 3,
"기쁨": 4,
"중립": 5,
"상처": 6,
"슬픔": 7,
}

```

- `convert_origin_to_coco()`: 각 annotation의 박스 정보와 이미지 파일 정보를 coco 형식에 맞게 구성한다
- `convert_origin_to_coco()` 의 매개변수
 - `origin_annots`: 변환할 주석(annotations) 데이터의 목록을 나타내는 리스트
 - `img_root_dir`: 이미지 파일이 있는 루트 디렉토리의 경로
 - `coco_annot`: COCO(annotation) 형식의 데이터를 나타내는 딕셔너리
 - `change_img_name`: 이미지 파일 이름을 변경할지 여부를 지정하는 불리언 매개변수, True로 설정할 시 `img_names` 딕셔너리를 사용한다
 - `img_names`: 이미지 파일 이름을 변경하기 위한 딕셔너리입니다. 기본값은 `{"old": "new"}` 로, `old` 파일 이름을 `new` 로 변경

```

def convert_origin_to_coco(
    origin_annots:list, img_root_dir:str, coco_annot:dict=
    change_img_name:bool=False, img_names:dict={"old": "new"
}):
    if coco_annot == None: coco_annot = deepcopy(COCO_ANNOT)
    for annot in origin_annots:
        img_filename = ""
        try:
            img_filename = img_names[annot["filename"]]
        except KeyError:
            if change_img_name: continue
            else: img_filename = annot["filename"]

```

```

img_id = len(coco_annot["images"]) + 1
boxes = None

for annot_type in ["A", "B", "C"]:
    try:
        boxes = annot[f"annot_{annot_type}"]["boxes"]
        assert boxes["maxX"] > boxes["minX"] and boxes["maxY"] > boxes["minY"]
    except:
        boxes = None
        continue
    else:
        break

# when bbox errors, remove the image
if boxes == None:
    os.remove(f"{img_root_dir}/{img_filename}")
    continue

width = boxes["maxX"] - boxes["minX"]
height = boxes["maxY"] - boxes["minY"]

# img_height, img_width, _ = cv2.imread(f"{img_root_dir}/{img_filename}")
img_width, img_height = Image.open(f"{img_root_dir}/{img_filename}").size

# images
coco_annot["images"].append({
    "id": img_id,
    "width": img_width,
    "height": img_height,
    "file_name": img_filename,
    "date_captured": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
})

# annotations
coco_annot["annotations"].append({
    "id": len(coco_annot["annotations"]) + 1,
    "category_id": CAT_MAPPER[annot["faceExp_uploader"]],
    "image_id": img_id,

```

```

        "bbox": [
            boxes["minX"], # x
            boxes["minY"], # y
            width, # width
            height # height
        ]
    })

    return coco_annot

```

- `main()`: 프로그램의 메인 함수로, 입력으로 주어진 디렉토리 내의 annotation 파일들을 읽어와 coco 형식으로 변환하고, 새로운 json 파일로 저장한다

```

def main(cfg):
    root_dir = cfg.dir_path
    new_coco_annot = {}

    if not exists(root_dir): raise FileNotFoundError(f"dir
    elif isfile(root_dir): raise FileNotFoundError(f"[{root
    else:
        for (root, dirs, files) in walk(root_dir):
            if root == root_dir: continue
            for file in files:
                with open(f"{root}/{file}", encoding="cp949") as f:
                    origin_annots = json.load(f)
                    new_coco_annot = convert_origin_to_coco(origin_a

    with open(f"{root_dir}/annotation.json", "w", encoding
        json.dump(new_coco_annot, f)

# if __name__ == "__main__":
#     parser = argparse.ArgumentParser()

#     parser.add_argument("--dir-path", type=str, default=

```

```
# config = parser.parse_args()
# main(config)
```

▼ sample_data_into_yolo_structure.py

→ 원본 데이터를 특정 비율로 선택하여 새로운 디렉토리 구조로 이동시키는 작업을 수행한다

선택적으로 일부 이미지를 샘플링 할 수 있다

- `create_features_dirs()` : 새로운 데이터 디렉토리 구조를 생성한다
- `create_features_dirs()` 의 매개변수
 - `dst_data_dir` : 생성될 데이터 디렉토리의 경로
 - `selection_ratio` : 데이터 세트의 일부 이미지를 선택하는 비율 (기본값: 0.8)

```
import argparse
from os import makedirs, listdir
from os.path import exists, isfile
from pathlib import Path
import random
import shutil
import warnings

EMOTIONS = ("anger", "anxiety", "embarrass", "happy", "n
DATA_TYPES = ("train", "test")
ROOT_PATH = Path(__file__).parent # est_wassup_03

def create_features_dirs(dst_data_dir:str, selection_rat
    """
    Returns:
        str: dst data dir
    """
    dst_data_dir = f"{dst_data_dir}_{selection_ratio}"
    new_num = 1
```

```

new_dst_data_dir = dst_data_dir
while exists(new_dst_data_dir):
    new_dst_data_dir = dst_data_dir + f"_{new_num}"
    new_num += 1
mkdirs(new_dst_data_dir)

dir_images = f"{new_dst_data_dir}/images"
dir_labels = f"{new_dst_data_dir}/labels"

# create directories
# check <TO> structure view above
mkdirs(dir_images, exist_ok=True)
mkdirs(dir_labels, exist_ok=True)

for d_type in DATA_TYPES:
    for emotion in EMOTIONS:
        mkdirs(f"{dir_images}/{d_type}/{emotion}", exist_ok=True)
        mkdirs(f"{dir_labels}/{d_type}/{emotion}", exist_ok=True)

return new_dst_data_dir

```

- `move_data_to_features()`: 기존 데이터 디렉토리에서 이미지와 레이블을 선택하고 새로운 구조로 이동한다
- `move_data_to_features()`의 매개변수
 - `src_data_dir`: 원본 데이터 디렉토리의 경로
 - `dst_data_dir`: 대상 데이터 디렉토리의 경로
 - `selection_ratio`: 데이터 세트의 일부 이미지를 선택하는 비율
- `train` 세트에서 선택된 비율만큼의 이미지를 랜덤하게 선택하여 새로운 위치로 이동한다

```

def move_data_to_features(src_data_dir:str, dst_data_dir:str):
    dir_images = f"{dst_data_dir}/images"
    dir_labels = f"{dst_data_dir}/labels"

```

```

for emotion in EMOTIONS:
    dir_emotion = f"{src_data_dir}/{emotion}"
    dir_test_set = f"{src_data_dir}/test_set/{emotion}"
    if not exists(dir_emotion): warnings.warn(f"directory {dir_emotion} does not exist")
    elif isfile(dir_emotion): raise warnings.warn(f"[{dir_emotion}] is a file")
    else:
        dir_img_train = f"{dir_images}/train/{emotion}"
        dir_label_train = f"{dir_labels}/train/{emotion}"
        dir_img_test = f"{dir_images}/test/{emotion}"
        dir_label_test = f"{dir_labels}/test/{emotion}"

        # empty the directory
        if len(listdir(dir_img_train)) != 0: shutil.rmtree(dir_img_train)
        if len(listdir(dir_label_train)) != 0: shutil.rmtree(dir_label_train)
        if len(listdir(dir_img_test)) != 0: shutil.rmtree(dir_img_test)
        if len(listdir(dir_label_test)) != 0: shutil.rmtree(dir_label_test)

        # select full*ratio number of images
        list_of_imgs = listdir(f"{dir_emotion}/raw/train")
        list_of_imgs = list(filter(lambda x: not x.endswith('.json'), list_of_imgs))
        selected_imgs = random.sample(list_of_imgs, int(len(list_of_imgs) * ratio))
        for img in selected_imgs:
            src_img = f"{dir_emotion}/raw/train/{img}"
            dst_img = f"{dir_img_train}/{img}"
            shutil.copy(src_img, dst_img)

        # copy to the directory
        # shutil.copytree(f"{dir_emotion}/raw/train", dir_img_train)
        # shutil.copytree(f"{dir_emotion}/labeled/train", dir_label_train)
        shutil.copyfile(f"{dir_test_set}.json", f"{dir_label_test}.json")
        shutil.copytree(dir_test_set, dir_img_test, dirs_exist_ok=True)

```

- `main()`: 프로그램의 메인 함수로, 주어진 소스 데이터 디렉토리에서 데이터를 추출하여 새로운 대상 데이터 디렉토리로 이동시킨다

```

def main(cfg):
    src_data_dir = cfg.src_data_path

```

```

dst_data_dir = cfg.dst_data_path
selection_ratio = cfg.selection_ratio

if not exists(src_data_dir): raise FileNotFoundError(f"
elif isfile(src_data_dir): raise FileNotFoundError(f"[
else:
    # create est_wassup_03/features
    dst_data_dir = create_features_dirs(dst_data_dir, se

    # move data to features
    move_data_to_features(src_data_dir, dst_data_dir, se

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument("--selection-ratio", type=float, d
    parser.add_argument("--src-data-path", type=str, defau
    parser.add_argument("--dst-data-path", type=str, defau

    config = parser.parse_args()
    main(config)

```

▼ sample_data_into_yolo_structure.py 의 주요 매개변수

--selection-ratio: 데이터 세트에서 이미지를 선택하는 비율을 설정. 값은 0과 1 사이의 실수이며, 기본값은 0.8

--src-data-path: 원본 데이터셋이 위치한 디렉토리의 경로를 설정. 기본값은

`/home/KDT-admin/data`

--dst-data-path: 새로운 데이터셋이 생성될 디렉토리의 경로를 설정. 기본값은

`/home/KDT-admin/work/selected_images`

▼ 프로그램 실행 후 변환된 디렉토리 구조

```

"""
<FROM>

/anger
    /labeled

```

```

        /train
        /validation
    /raw
        /train
        /validation
/anxiety
    /labeled
        /train
        /validation
    /raw
        /train
        /validation
/test_set
    /anger
    /anxiety
    anger.json
    anxiety.json

```

<T0>

```

/images
    /train
        /anger
        /happy
    /test
        /anger
        /happy
/labels
    /train
        /anger
        /happy
    /test
        /anger
        /happy

```

```
"""
```

▼ create_feature_dataset.py

→주어진 디렉토리 구조에서 이미지와 레이블을 포함하는 데이터셋을 다른 디렉토리 구조로 이동하고, 이를 기반으로 새로운 coco 형식의 annotation을 생성하는 작업을 수행한다 (폴더 및 파일명에 한글이 포함 될 경우 오류가 날 가능성이 있기에 UUID로 변환 시켜준다)

- `dfs()` : 재귀적으로 디렉토리를 탐색하고 파일을 이동한다
- `dfs()` 의 매개변수
 - `content` : 디렉토리 내용을 저장하는 딕셔너리
 - `parent_path` : 탐색을 시작할 디렉토리의 경로
 - `filename_old_to_new` : 파일 이름을 새로운 이름으로 매핑하는 딕셔너리
 - `change_folder_name` : 폴더 이름을 변경할지 여부를 결정하는 불리언 값
 - `src_root_dir` : 소스 디렉토리의 루트 경로
 - `dst_root_dir` : 대상 디렉토리의 루트 경로
 - `mode` : 작업 모드("train" 또는 "test")
 - `progress_bar` : 진행 상황을 표시하기 위한 tqdm의 진행 표시줄 객체

```
import argparse
from convert_label_to_coco import convert_origin_to_coco
import json
from os.path import isfile, exists
from os import listdir, walk, makedirs
import shutil
from typing import Literal
from uuid import uuid4
import tqdm

def dfs(
    content:dict={"name": "root", "content": {}}, parent_path:
    filename_old_to_new:dict={}, change_folder_name:bool=False,
    src_root_dir:str="/home/KDT-admin/work/selected_images",
    progress_bar=None
):
```

```

for dir in listdir(parent_path):
    uuid = str(uuid4())
    if isfile(f"{parent_path}/{dir}"): # rename files
        if "." in dir:
            uuid = f"{uuid}.{dir.split('.')[-1]}"
        content["content"][uuid] = dir
        shutil.copy(f"{parent_path}/{dir}", f"{dst_root_dir}/{uuid}")
        if f"{src_root_dir}/images/{mode}" in parent_path:
            # TODO: 데이터 폴더 구조에 따라서 수정해야 할 수 있음
            # 주의!!!!
            # 이거 개발할 땐 data/images/anger... data/labels/a
            filename_old_to_new[dir] = f"{parent_path.replace(src_root_dir, dst_root_dir)}/{uuid}"
        if progress_bar:
            progress_bar.update(1)
    else: # rename dirs
        if change_folder_name:
            content["content"][uuid] = { "name": dir, "content": "" }
            new_path = f"{parent_path}/{dir}"
            new_dst_root_dir = f"{dst_root_dir}/{uuid}"
            makedirs(new_dst_root_dir)
            dfs(content["content"][uuid], new_path, filename_old_to_new)
        else:
            content["content"][dir] = { "name": dir, "content": "" }
            new_path = f"{parent_path}/{dir}"
            new_dst_root_dir = f"{dst_root_dir}/{dir}"
            makedirs(new_dst_root_dir, exist_ok=True)
            dfs(content["content"][dir], new_path, filename_old_to_new)

```

- `create_structure_dataset()`: 원본 디렉토리의 이미지 및 레이블을 새로운 디렉토리로 복사, 파일 및 디렉토리 이름을 변경한다
- `create_structure_dataset()`의 매개변수
 - `src_root_dir`: 원본 데이터셋의 루트 경로
 - `dst_root_dir`: 대상 데이터셋의 루트 경로
 - `filename_history`: 파일 이름 변경 내역을 저장하는 딕셔너리
 - `mode`: 작업 모드("train" 또는 "test")

```

def create_structure_dataset(src_root_dir:str, dst_root_dir:str,
                             filename_old_to_new = {},
                             new_coco_annot = None

    # TODO: progress bar 코드 정리
    tqdm_len = 0

    print(f"counting files in {mode} directory...")
    for e in listdir(f"{src_root_dir}/images/{mode}"):
        tqdm_len += len(listdir(f"{src_root_dir}/images/{mode}/{e}"))

    with tqdm.tqdm(total=tqdm_len, desc="copy renamed images"):
        dfs(filename_history["root"]["images"][mode], f"{src_root_dir}/images/{mode}")
    with tqdm.tqdm(total=len(listdir(f"{src_root_dir}/labels/{mode}")), desc="copy renamed labels"):
        dfs(filename_history["root"]["labels"][mode], f"{src_root_dir}/labels/{mode}")

    # TODO: 데이터 폴더 구조에 따라서 수정해야 할 수 있음
    # 주의!!!!
    # 이거 개발할 땐 data/images/anger... data/labels/anger...
    with tqdm.tqdm(total=len(listdir(f"{src_root_dir}/labels/{mode}")), desc="convert labels to coco"):
        for (root, dirs, files) in walk(f"{src_root_dir}/labels/{mode}"):
            if root == f"{src_root_dir}/labels/{mode}": continue
            for file in files:
                with open(f"{root}/{file}", encoding="cp949") as f:
                    origin_annots = json.load(f)
                    new_coco_annot = convert_origin_to_coco(origin_annots)
                    progress_bar.update(1)

    with open(f"{dst_root_dir}/labels/{mode}/annotation.json", "w") as f:
        json.dump(new_coco_annot, f)

```

- `main()`: 프로그램의 주요 작업을 실행하고 필요한 파일을 생성한다

또한 `argparse`를 사용하여 사용자가 지정한 디렉토리 경로를 파싱한다

그리고 `create_structure_dataset` 함수를 호출하여 주어진 작업을 수행한다

```

def main(cfg):
    src_root_dir = cfg.src_dir_path
    dst_root_dir = cfg.dst_dir_path

    filename_history = {
        "root": {
            "images": {
                "train": {"name": f"{src_root_dir}/images/train"}
                "test": {"name": f"{src_root_dir}/images/test"},
            },
            "labels": {
                "train": {"name": f"{src_root_dir}/labels/train"}
                "test": {"name": f"{src_root_dir}/labels/test"},
            },
        }
    }

    makedirs(dst_root_dir, exist_ok=True)
    print("getting ready...")
    if exists(src_root_dir) and not isfile(src_root_dir):
        if len(listdir(dst_root_dir)) != 0:
            shutil.rmtree(dst_root_dir)
        makedirs(f"{dst_root_dir}/images/train", exist_ok=True)
        makedirs(f"{dst_root_dir}/images/test", exist_ok=True)
        makedirs(f"{dst_root_dir}/labels/train", exist_ok=True)
        makedirs(f"{dst_root_dir}/labels/test", exist_ok=True)

    create_structure_dataset(src_root_dir, dst_root_dir, f)
    create_structure_dataset(src_root_dir, dst_root_dir, f)

    with open(f"{dst_root_dir}/filename_mapper.json", "w",
              json.dump(filename_history, f)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument("--src-dir-path", type=str, default=
    parser.add_argument("--dst-dir-path", type=str, default=

```

```
config = parser.parse_args()
main(config)
```

▼ create_feature_dataset.py의 주요 매개변수

- `--src-dir-path` : 원본 데이터셋이 위치한 디렉토리의 경로를 설정합니다. 기본값은 `/home/KDT-admin/work/selected_images`
- `--dst-dir-path` : 변환된 데이터셋이 생성될 디렉토리의 경로를 설정합니다. 기본값은 `../data`

▼ create_yolo_detection_dataset.py

→ 이미지와 해당 이미지의 주석 정보를 가지고 있는 데이터셋을 YOLO 객체 검출 알고리즘이 사용할 수 있는 형식으로 변환하는 작업을 수행한다

- `bbox_2_yolo()` : 주어진 바운딩 박스를 YOLO 형식으로 변환하여 반환한다
- `bbox_2_yolo()` 의 매개변수
 - `bbox` : 원본 바운딩 박스 정보 (xmin, ymin, width, height)
 - `img_w` : 이미지의 너비
 - `img_h` : 이미지의 높이
- 반환되는 값: YOLO 형식의 바운딩 박스 (centerx, centery, width, height)

```
import argparse
import json
from os import makedirs, listdir
import shutil
from typing import Literal

"""
<FROM>

/images
  /train
    /anger
```

```

        /happy
    /test
        /anger
        /happy
/labels
    /train
        /anger
        /happy
    /test
        /anger
        /happy

<T0>

/images
    /train
    /test
/labels
    /train
    /test
"""

DIR_LEVEL_1 = ("images", "labels")
DIR_LEVEL_2 = ("train", "test")

def bbox_2_yolo(bbox, img_w, img_h):
    x, y, w, h = bbox[0], bbox[1], bbox[2], bbox[3]
    centerx = bbox[0] + w / 2
    centery = bbox[1] + h / 2
    dw = 1 / img_w
    dh = 1 / img_h
    centerx *= dw
    w *= dw
    centery *= dh
    h *= dh
    return centerx, centery, w, h

```

- `convert_anno()`: 주어진 데이터셋의 주석 정보를 읽어와 YOLO 형식으로 변환한 후 반환한다
- `convert_anno()`의 매개변수
 - `src_data_path`: 변환할 데이터의 소스 경로
 - `mode`: 변환할 데이터의 모드(train 또는 test)를 지정
- 반환되는 값: YOLO 형식의 주석 정보를 담은 딕셔너리

```
def convert_anno(src_data_path:str, mode:Literal["train"
    """
    Returns:
        { "image_id": [(image_name, category_id, yolobox)]
    """
    # read annotation data
    with open(f"{src_data_path}/labels/{mode}/annotation.j
        data = json.load(f)

    # create image info dictionary { image_id = image_info
    images = dict()
    for image in data['images']:
        id = image['id']
        file_name = image['file_name']
        w = image['width']
        h = image['height']
        images[id] = (file_name, w, h)

    # create yolo format annotations
    anno_dict = dict()
    for anno in data['annotations']:
        bbox = anno['bbox']
        image_id = anno['image_id']
        category_id = anno['category_id']

        image_info = images.get(image_id)
        image_name = image_info[0]
        img_w = image_info[1]
```

```

img_h = image_info[2]
yolo_box = bbox_2_yolo(bbox, img_w, img_h)

anno_info = (image_name, category_id, yolo_box)
anno_infos = anno_dict.get(image_id)
if not anno_infos:
    anno_dict[image_id] = [anno_info]
else:
    anno_infos.append(anno_info)
    anno_dict[image_id] = anno_infos

return anno_dict

```

- `write_yolo_annot_txt()` : YOLO 형식의 주석 정보를 텍스트 파일로 저장한다
- `write_yolo_annot_txt()` 의 매개변수
 - `src_data_path` : 원본 데이터셋의 경로
 - `dst_data_path` : 변환된 데이터셋의 경로
 - `anno_dict` : YOLO 형식의 주석 정보를 담은 딕셔너리
 - `mode` : 작업 모드 ("train" 또는 "test")

```

def write_yolo_annot_txt(src_data_path:str, dst_data_path:str):
    for k, v in anno_dict.items():
        emotion, origin_file_name = v[0][0].split("/")
        # copy img
        shutil.copy(f"{src_data_path}/images/{mode}/{emotion}/{origin_file_name}",
                    f"{dst_data_path}/images/{mode}/{emotion}/{origin_file_name}")

        # write yolo txt
        file_name = origin_file_name.split(".")[0] + ".txt"
        with open(f"{dst_data_path}/labels/{mode}/{file_name}", "w") as f:
            for obj in v:
                # category_id = obj[1]
                category_id = 0 # detect face only
                box = [' {:.6f}'.format(x) for x in obj[2]]
                box = ' '.join(box)
                f.write(f"{category_id} {box}\n")

```



```

line = str(category_id) + ' ' + box
f.write(line + '\n')

```

- `main()`: 전체 변환 작업을 수행하는 메인 함수로, 주어진 데이터셋의 주석 정보를 YOLO 형식으로 변환하고 텍스트 파일로 저장한다

```

def main(cfg):
    src_data_path = cfg.src_data_path
    dst_data_path = cfg.dst_data_path

    # create features folder if not exists
    makedirs(dst_data_path, exist_ok=True)

    # create destination directories
    for level1 in DIR_LEVEL_1:
        dir_level1 = f"{dst_data_path}/{level1}"
        makedirs(dir_level1, exist_ok=True)
        if len(listdir(dir_level1)) != 0:
            shutil.rmtree(dir_level1)
        for level2 in DIR_LEVEL_2:
            makedirs(f"{dir_level1}/{level2}", exist_ok=True)

    train_anns = convert_anno(src_data_path, "train")
    test_anns = convert_anno(src_data_path, "test")

    # save annotation txt files
    write_yolo_annot_txt(src_data_path, dst_data_path, train_anns)
    write_yolo_annot_txt(src_data_path, dst_data_path, test_anns)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument("--src-data-path", type=str, default="data")
    parser.add_argument("--dst-data-path", type=str, default="data")

```

```
config = parser.parse_args()
main(config)
```

▼ create_yolo_detection_dataset.py 의 주요 매개변수

- `--src-data-path`: 원본 데이터셋이 위치한 디렉토리의 경로를 설정. 기본값은 `../data`
- `--dst-data-path`: 변환된 YOLO 데이터셋이 생성될 디렉토리의 경로를 설정. 기본값은 `../yolo_detection_data`

▼ split_train_val.py

→ 주어진 데이터를 coco 형식 또는 yolo 형식으로 분할한다. 데이터를 분할하여 학습 및 검증 세트로 나누고 분할된 데이터를 각각의 형식에 맞게 저장한다

- import와 coco 및 yolo 데이터 포맷, 이미지 및 카테고리 정보가 포함되어 있다

```
import argparse
from copy import deepcopy
import json
from os import listdir, makedirs
from os.path import exists
from os.path import abspath
from pycocotools.coco import COCO
import random
import shutil
from typing import Literal
import yaml

"""
<COCO and YOLO classification data format>

/images
  /train
    /anger
    /happy
```

```

/test
  /anger
  /happy
/labels
  /train
    /anger
    /happy
  /test
    /anger
    /happy

```

<YOLO Detection data format>

```

/images
  /train
  /test
/labels
  /train
  /test
"""

```

```

COCO_ANNOT = {
    "info": {
        "description": "facial-expression-classification",
        "url": "https://aihub.or.kr/aihubdata/data/view.",
        "version": "1.2",
        "year": 2023,
        "contributor": "한국과학기술원",
        "date_created": "2023/10/10"
    },
    "images": [
        # {
        #     "id": 1, # "id" must be int >= 1
        #     "width": 426,
        #     "height": 640,
        #     "file_name": "xxxxxxxxx.jpg",
        #     "date_captured": "2013-11-15 02:41:42"
        # }
    ]
}

```

```

],
"annotations": [
  # {
  #     "id": 1, # "id" must be int >= 1
  #     "category_id": 1, # "category_id" must be in
  #     "image_id": 1, # "image_id" must be int >= 1
  #     "bbox": [86, 65, 220, 334] # [x,y,width,height]
  # }
],
"categories": [
  # {
  #     "id": 2, # "id" must be int >= 1
  #     "name": "happy"
  # }
  { "id": 1, "name": "anger" },
  { "id": 2, "name": "anxiety" },
  { "id": 3, "name": "embarrass" },
  { "id": 4, "name": "happy" },
  { "id": 5, "name": "normal" },
  { "id": 6, "name": "pain" },
  { "id": 7, "name": "sad" },
]
}

```

- 파일 복사 및 이동 함수, 주어진 이미지를 지정된 mode(train or val)에 따라 대상 디렉토리로 복사 또는 이동한다

```

def copy_file(
    src_root_dir: str, dst_root_dir: str, img_name: str,
    mode: Literal["train", "val"], do_for_label: bool=False
):
    shutil.copy(f"{src_root_dir}/images/train/{img_name}",
    if do_for_label:
        shutil.copy(f"{src_root_dir}/labels/train/{img_name.

def move_file(
    src_root_dir: str, img_name: str,

```

```

mode: Literal["train", "val"], do_for_label: bool=False
):
    if mode == "train": return
    shutil.move(f"{src_root_dir}/images/train/{img_name}",
    if do_for_label:
        shutil.move(f"{src_root_dir}/labels/train/{img_name}.

```

- `split_list_val_train()`: 입력된 이미지 리스트를 학습 및 검증 세트로 분할하는 역할을 한다
- `split_list_val_train()` 의 매개변수
 - `list_images`: 분할할 이미지 리스트
 - `trn_ratio`: 학습 세트에 포함될 이미지 비율을 나타내는 값으로, 0과 1 사이의 값
-

```

def split_list_val_train(list_images: list, trn_ratio: float):
    """
    Returns:
        tuple: ( trn_images, val_images )
    """
    list_images = list_images.copy()
    random.shuffle(list_images)
    middle = int(len(list_images) * trn_ratio)
    trn_images = list_images[:middle]
    val_images = list_images[middle:]
    return trn_images, val_images

def process_per_emotion(src_root_dir: str, dst_root_dir: str):
    """
    Returns:
        tuple: ( e_trn_images, e_val_images )
    """
    makedirs(f"{dst_root_dir}/images/train/{e}", exist_ok=True)
    makedirs(f"{dst_root_dir}/images/val/{e}", exist_ok=True)
    makedirs(f"{dst_root_dir}/labels/train/{e}", exist_ok=True)

```

```

makedirs(f"{dst_root_dir}/labels/val/{e}", exist_ok=True)

e_images = listdir(f"{src_root_dir}/images/train/{e}")
e_trn_images, e_val_images = split_list_val_train(e_images)
for img in e_trn_images:
    if not do_copy: break
    process_file_action(src_root_dir, dst_root_dir, f"{e}/{img}")
for img in e_val_images:
    process_file_action(src_root_dir, dst_root_dir, f"{e}/{img}")

return e_trn_images, e_val_images

def coco_annotation_split(annot_path: str, trn_images: list):
    coco_annot = COCO(annot_path)

    for img_id in coco_annot.getImgIds():
        img = coco_annot.imgs[img_id]
        ann = coco_annot.imgToAnns[img_id]
        img_name_only = img["file_name"].split("/")[1]
        if img_name_only in val_images:
            val_images.remove(img_name_only)
            val_annot["images"].append(img)
            val_annot["annotations"].append(ann)
        else:
            trn_images.remove(img_name_only)
            trn_annot["images"].append(img)
            trn_annot["annotations"].append(ann)

def write_yolo_dataset_yaml(dst_root_dir: str):
    dst_root_abs = abspath(dst_root_dir)
    with open(f"{dst_root_dir}/yolo-dataset.yaml", "w", encoding="utf-8") as f:
        yaml.dump({
            "path": f"{dst_root_abs}",
            "train": "images/train",
            "val": "images/val",
            "test": "images/test",
            "names": {
                0: "face"
            }
        }, f)

```

```

    }
    }, f)

def copy_test_set(src_root_dir: str, dst_root_dir: str):
    if src_root_dir != dst_root_dir:
        dst_images_test = f"{dst_root_dir}/images/test"
        dst_labels_test = f"{dst_root_dir}/labels/test"
        shutil.rmtree(dst_images_test, ignore_errors=True)
        shutil.rmtree(dst_labels_test, ignore_errors=True)
        shutil.copytree(f"{src_root_dir}/images/test", dst_images_test)
        shutil.copytree(f"{src_root_dir}/labels/test", dst_labels_test)

```

```

def yolo_detection_split(src_root_dir: str, dst_root_dir: str)
    """
    this will reformat the source root dir if src_root_dir
    if not are same, this will copy images and labels to t
    """
    process_file_action = move_file
    do_copy = src_root_dir != dst_root_dir
    if do_copy:
        process_file_action = copy_file
        shutil.rmtree(dst_root_dir, ignore_errors=True)

    makedirs(f"{dst_root_dir}/images/train", exist_ok=True)
    makedirs(f"{dst_root_dir}/images/val", exist_ok=True)
    makedirs(f"{dst_root_dir}/labels/train", exist_ok=True)
    makedirs(f"{dst_root_dir}/labels/val", exist_ok=True)

    list_images = listdir(f"{src_root_dir}/images/train")
    trn_images, val_images = split_list_val_train(list_images)
    for img in trn_images:
        if not do_copy: break
        process_file_action(src_root_dir, dst_root_dir, img,
    for img in val_images:
        process_file_action(src_root_dir, dst_root_dir, img,

```

```

write_yolo_dataset_yaml(dst_root_dir)

def yolo_classification_split(src_root_dir: str, dst_root_dir: str)
    """
    this will reformat the source root dir if src_root_dir
    if not are same, this will copy images and labels to t
    """
    emotions = listdir(f"{src_root_dir}/images/train")
    val_annot = deepcopy(COCO_ANNOT)
    trn_annot = deepcopy(COCO_ANNOT)

    trn_images = []
    val_images = []

    process_file_action = move_file
    do_copy = src_root_dir != dst_root_dir
    if do_copy:
        process_file_action = copy_file
        shutil.rmtree(dst_root_dir, ignore_errors=True)

    for e in emotions:
        e_trn_images, e_val_images = process_per_emotion(src
            trn_images.extend(e_trn_images)
            val_images.extend(e_val_images)

    coco_annotation_split(f"{src_root_dir}/labels/train/an

    with open(f"{dst_root_dir}/labels/train/annotation.json"
        json.dump(trn_annot, f)

    with open(f"{dst_root_dir}/labels/val/annotation.json"
        json.dump(val_annot, f)

def coco_classification_split(src_root_dir: str, dst_root_dir: str)
    """
    this will reformat the source root dir if src_root_dir
    if not are same, this will copy images and labels to t
    """

```



```

emotions = listdir(f"{src_root_dir}/images/train")

val_annot = deepcopy(COCO_ANNOT)
trn_annot = deepcopy(COCO_ANNOT)

trn_images = []
val_images = []

process_file_action = move_file
do_copy = src_root_dir != dst_root_dir
if do_copy:
    shutil.rmtree(dst_root_dir, ignore_errors=True)
    process_file_action = copy_file

for e in emotions:
    e_trn_images, e_val_images = process_per_emotion(src,
trn_images.extend(e_trn_images)
val_images.extend(e_val_images)

coco_annotation_split(f"{src_root_dir}/labels/train/annota

with open(f"{dst_root_dir}/labels/train/annotation.json"
json.dump(trn_annot, f)

with open(f"{dst_root_dir}/labels/val/annotation.json"
json.dump(val_annot, f)

def coco_detection_split(src_root_dir: str, dst_root_dir
"""
this will reformat the source root dir if src_root_dir
if not are same, this will copy images and labels to t
"""
emotions = listdir(f"{src_root_dir}/images/train")

val_annot = deepcopy(COCO_ANNOT)
trn_annot = deepcopy(COCO_ANNOT)

trn_images = []

```

```

val_images = []

process_file_action = move_file
do_copy = src_root_dir != dst_root_dir
if do_copy:
    shutil.rmtree(dst_root_dir, ignore_errors=True)
    process_file_action = copy_file

for e in emotions:
    e_trn_images, e_val_images = process_per_emotion(src.
    trn_images.extend(e_trn_images)
    val_images.extend(e_val_images)

coco_annotation_split(f"{src_root_dir}/labels/train/an

with open(f"{dst_root_dir}/labels/train/annotation.js
    json.dump(trn_annot, f)

with open(f"{dst_root_dir}/labels/val/annotation.json"
    json.dump(val_annot, f)

```

```

def main(cfg):
    annot_format = cfg.annot_format[0]
    task = cfg.task[0]
    train_ratio = cfg.train_ratio
    src_root_dir = cfg.src_root_path
    dst_root_dir = cfg.dst_root_path

    splitter_map = {
        "classification": {
            "yolo": yolo_classification_split,
            "coco": coco_classification_split
        },
        "detection": {
            "yolo": yolo_detection_split,
            "coco": coco_detection_split
        }
    }

```

```

    }
}

splitter_map[task][annot_format](src_root_dir, dst_root_dir)
copy_test_set(src_root_dir, dst_root_dir)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    # 주의!!!!!!
    # split 하려는 src 폴더 포맷이 annot-format, task 와 같은지
    # Make sure there are two folders in the src-root-path
    parser.add_argument("--annot-format", choices=["coco",
    parser.add_argument("--task", choices=["classification
    parser.add_argument("--src-root-path", type=str, default=
    parser.add_argument("--dst-root-path", type=str, default=
    parser.add_argument("--train-ratio", type=float, default=

    config = parser.parse_args()
    main(config)

```