Untitled 5

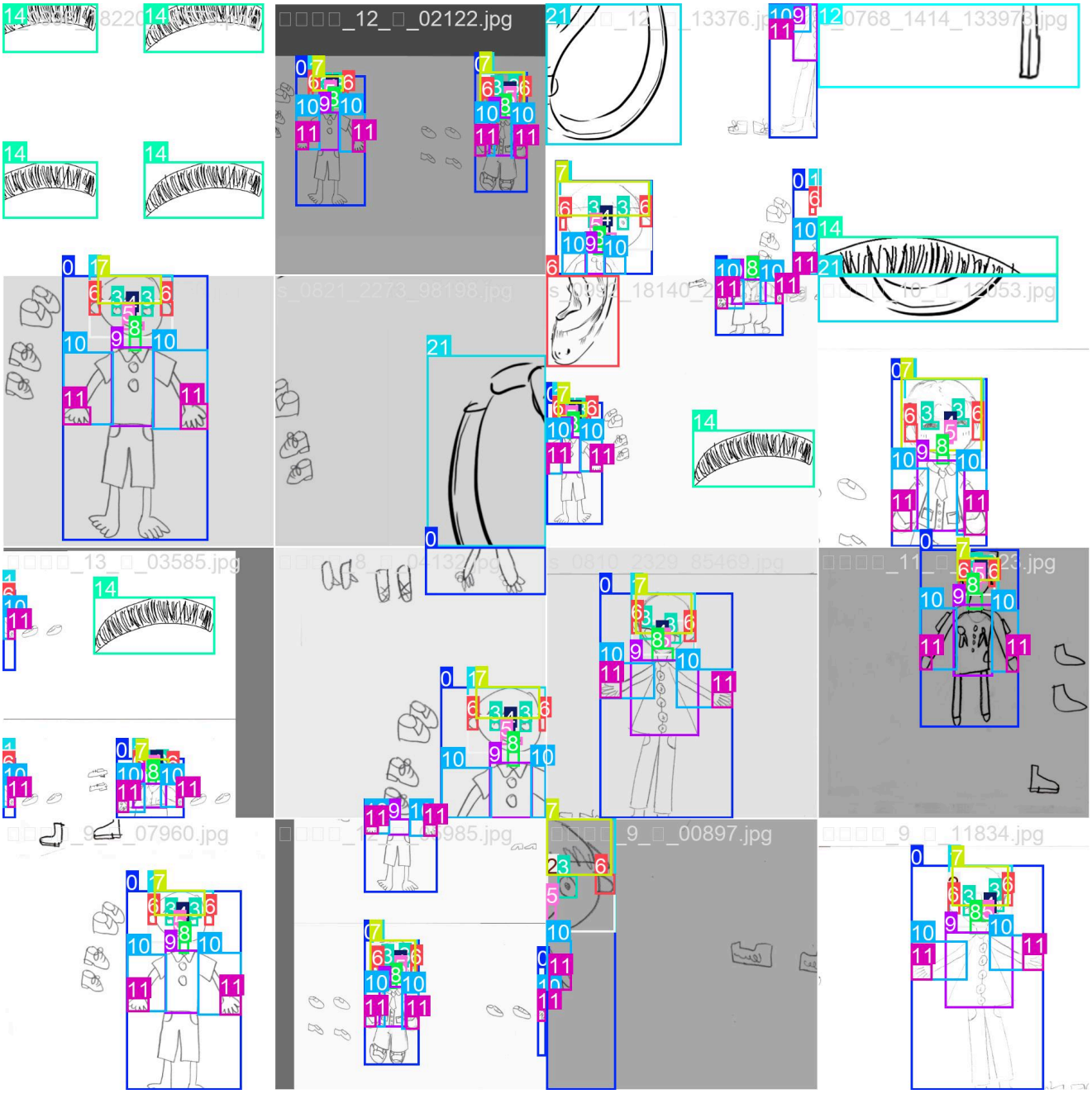
```yaml
# dataset.yaml
train: C:/capstone/data/train/images   # 학습용 이미지 폴더 경로
val: C:/capstone/data/validation/images     # 검증용 이미지 폴더 경로

nc: 27  # 클래스 수
names: ['사람전체', '머리', '얼굴', '눈', '코', '입', '귀', '머리카락', '목', '상체',
 '팔', '손', '모자', '안경', '눈썹', '수염', '벌린입(치아)', '목도리', '넥타이', '리본',
 '귀도리', '귀걸이', '목걸이', '장식', '머리장식', '보석', '담배'] # , '다리', '발', '단추
# +손, +귀, +코, +눈, +입, +눈썹, +수염, +팔, +얼굴, +1003,1004 사람전체
```

데이터 보강

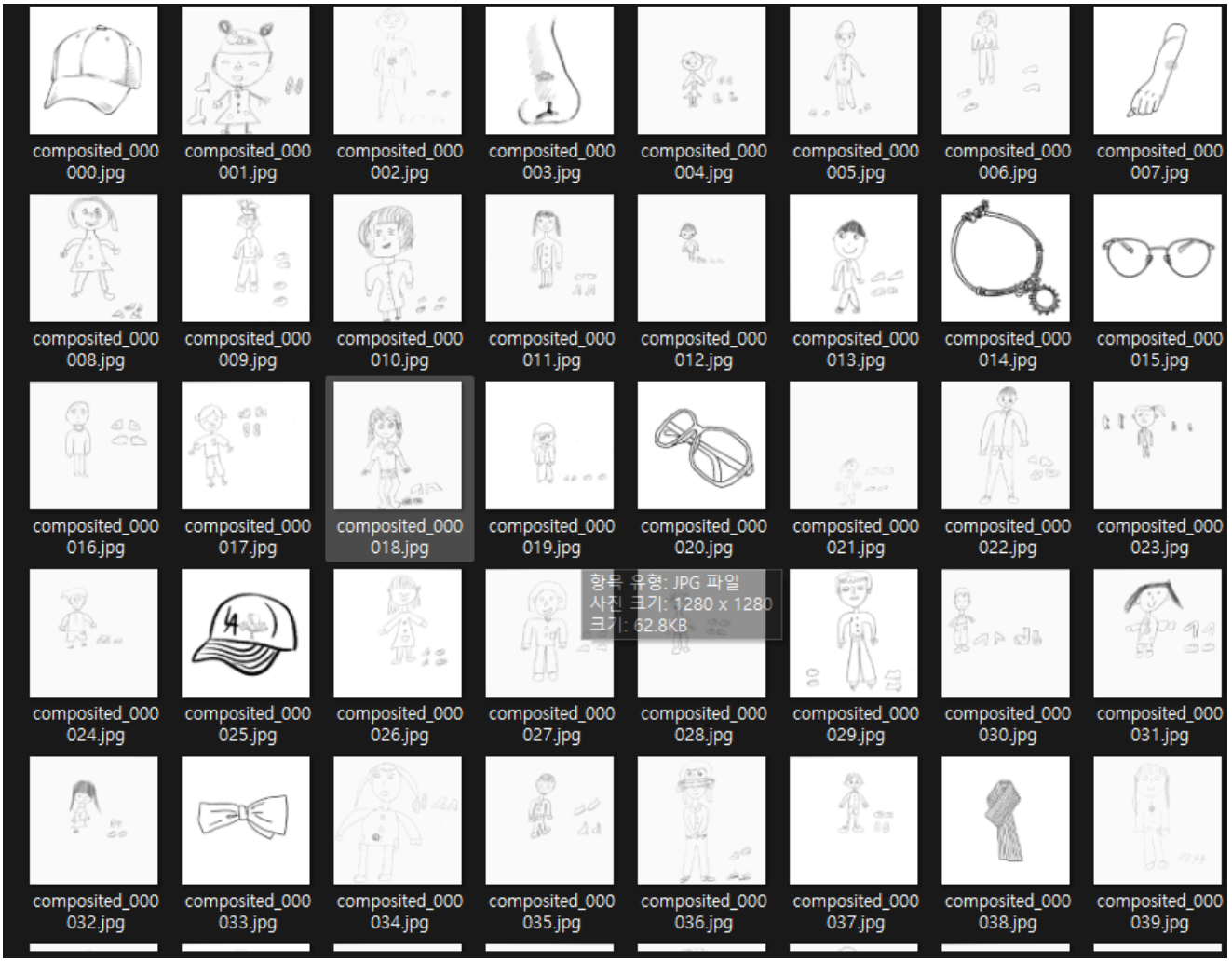
데이터 전처리

학습



결과 인식 못하는데 이유:'사람전체', '머리', '얼굴', '눈', '코', '입', '귀', '머리카락', '목', '상체', '팔', '손' 까지의 데이터는 한 이미지에 같이 있는데, 나머지 데이터는 단일로 존재하는 이미지에 각각 담겨있어서.

composited dataset 직접 구축



이제 다시 학습만 다시 시켜보면 됨.

```
import os

import glob

from PIL import Image

import cv2

import numpy as np

def convert_jpg_to_png_with_transparency(source_dir, target_dir):

    """

    source_dir의 모든 JPG 이미지를 PNG로 변환하고 배경을 투명하게 만들어 target_dir에 저장합니다.

    :param source_dir: JPG 이미지가 있는 디렉토리 경로

    :param target_dir: 변환된 PNG 이미지를 저장할 디렉토리 경로

    """
```

```
# 타겟 디렉토리가 없으면 생성
if not os.path.exists(target_dir):
    os.makedirs(target_dir)
# 모든 JPG 파일 찾기 (대소문자 구분 없이)
jpg_files = glob.glob(os.path.join(source_dir, '*.jpg')) + \
            glob.glob(os.path.join(source_dir, '*.jpeg')) + \
            glob.glob(os.path.join(source_dir, '*.JPG')) + \
            glob.glob(os.path.join(source_dir, '*.JPEG'))
count = 0
for jpg_file in jpg_files:
    try:
        # 파일명 추출
        file_name = os.path.basename(jpg_file)
        name_without_ext = os.path.splitext(file_name)[0]
        # PNG 파일 경로
        png_file = os.path.join(target_dir, f"{name_without_ext}.png")
        # 방법 1: 간단한 변환 (배경 투명화 없음)
        # Image.open(jpg_file).save(png_file, 'PNG')
        # 방법 2: OpenCV를 사용한 배경 제거 및 투명화
        # 이미지 로드
        img = cv2.imread(jpg_file)
        # 그레이스케일 변환
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        # 임계값 설정으로 마스크 생성 (밝은 배경 가정)
        _, mask = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY_INV)
        # 가장자리 부드럽게 처리
```

```
            kernel = np.ones((3, 3), np.uint8)
            mask = cv2.dilate(mask, kernel, iterations=1)
            mask = cv2.GaussianBlur(mask, (5, 5), 0)
            # 알파 채널 생성
            b, g, r = cv2.split(img)
            rgba = [b, g, r, mask]
            dst = cv2.merge(rgba, 4)
            # PNG로 저장
            cv2.imwrite(png_file, dst)
            count += 1
            print(f"변환 완료: {jpg_file} -> {png_file}")
        except Exception as e:
            print(f"변환 실패: {jpg_file}, 오류: {e}")
    print(f"\n총 {count}개의 이미지가 변환되었습니다.")
def convert_simple(source_dir, target_dir, subfolder_structure=False):
    """
    JPG 이미지를 PNG로 단순 변환합니다 (포맷만 변경).
    :param source_dir: JPG 이미지가 있는 디렉토리 경로
    :param target_dir: 변환된 PNG 이미지를 저장할 디렉토리 경로
    :param subfolder_structure: 하위 폴더 구조를 유지할지 여부
    """
    # 타겟 디렉토리가 없으면 생성
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)
    if subfolder_structure:
        # 하위 폴더 구조를 포함한 모든 JPG 파일 검색
```

```
    jpg_files = []
    for root, _, files in os.walk(source_dir):
        for file in files:
            if file.lower().endswith(('.jpg', '.jpeg')):
                jpg_files.append(os.path.join(root, file))
else:
    # 현재 폴더의 JPG 파일만 검색
    jpg_files = glob.glob(os.path.join(source_dir, '*.jpg')) + \
                glob.glob(os.path.join(source_dir, '*.jpeg')) + \
                glob.glob(os.path.join(source_dir, '*.JPG')) + \
                glob.glob(os.path.join(source_dir, '*.JPEG'))
count = 0
for jpg_file in jpg_files:
    try:
        # 상대 경로 계산
        rel_path = os.path.relpath(jpg_file, source_dir)
        dir_path = os.path.dirname(rel_path)
        # 파일명 추출
        file_name = os.path.basename(jpg_file)
        name_without_ext = os.path.splitext(file_name)[0]
        # 대상 디렉토리 생성
        if subfolder_structure and dir_path:
            target_subdir = os.path.join(target_dir, dir_path)
            if not os.path.exists(target_subdir):
                os.makedirs(target_subdir)
            png_file = os.path.join(target_subdir, f"{name_without_ext}.png")
```

```
            else:
                png_file = os.path.join(target_dir, f"{name_without_ext}.png")
            # 이미지 로드 및 변환
            img = Image.open(jpg_file)
            # RGBA로 변환 (알파 채널 추가)
            if img.mode != 'RGBA':
                img = img.convert('RGBA')
            # PNG로 저장
            img.save(png_file, 'PNG')
            count += 1
            if count % 10 == 0:
                print(f"{count}개 변환 완료...")
        except Exception as e:
            print(f"변환 실패: {jpg_file}, 오류: {e}")
    print(f"\n총 {count}개의 이미지가 변환되었습니다.")
```

실행 예제

```
if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description='JPG 이미지를 PNG로 변환')
    parser.add_argument('--source', type=str, default='.', help='소스 디렉토리 경로')
    parser.add_argument('--target', type=str, default='./png_output', help='타겟 디렉토리 경로')
    parser.add_argument('--mode', type=str, default='simple', choices=['simple',
'transparency'],
                        help='변환 모드: simple(단순 변환) 또는 transparency(배경 투명화 시도)')
    parser.add_argument('--subfolders', action='store_true', help='하위 폴더 구조 유지')
    args = parser.parse_args()
```

```
    print(f"변환 시작: {args.source} -> {args.target}")
    print(f"모드: {args.mode}, 하위 폴더 유지: {args.subfolders}")
    if args.mode == 'transparency':
        convert_jpg_to_png_with_transparency(args.source, args.target)
    else:
        convert_simple(args.source, args.target, args.subfolders)
import cv2
import numpy as np
import os
from PIL import Image
import random
import glob
import yaml
```

경로 정규화 함수 추가

```
def normalize_path(path):
    return os.path.normpath(path).replace("\", "/")
def load_image_safely(image_path):
    """
    여러 방법을 시도하여 이미지를 안전하게 로드합니다.
    OpenCV 실패 시 PIL을 사용합니다.
    """
    # 방법 1: OpenCV 직접 로드
    img = cv2.imread(image_path)
    if img is None:
        try:
            # 방법 2: PIL로 로드 후 OpenCV 형식으로 변환
```

```
            pil_img = Image.open(image_path)
            pil_img = pil_img.convert('RGB')  # RGBA가 있으면 RGB로 변환
            img = np.array(pil_img)
            img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)  # RGB에서 BGR로 변환
            print(f"PIL로 성공적으로 로드: {image_path}")
            return img
        except Exception as e:
            # 방법 3: 경로 인코딩 변경 시도
            try:
                encoded_path = image_path.encode('utf-8').decode('latin1')
                img = cv2.imread(encoded_path)
                if img is not None:
                    print(f"인코딩 변경으로 성공적으로 로드: {image_path}")
                    return img
            except:
                pass
            print(f"모든 방법으로 이미지 로드 실패: {image_path}, 오류: {e}")
            return None
    return img
```

YOLO 형식으로 바운딩 박스 라벨 생성 함수

```
def create_yolo_annotation(class_id, x_center, y_center, width, height, img_width, img_height):
    # YOLO 형식: <class_id> <x_center> <y_center> (모두 정규화된 값)
    x_center_norm = x_center / img_width
    y_center_norm = y_center / img_height
    width_norm = width / img_width
```

```
    height_norm = height / img_height

    return f"{class_id} {x_center_norm:.6f} {y_center_norm:.6f} {width_norm:.6f} {height_norm:.6f}"
```

특정 조건에 맞는 신체 부위 찾기 (일반적인, 얼굴 제외 등)

```
def find_target_parts(base_labels, target_type, img_width, img_height, class_map):
    """
    base_labels: 기본 이미지의 YOLO 형식 라벨 목록
    target_type: 대상 지정 유형 (예: '입', '머리', '얼굴제외', '목_영역' 등)
    img_width, img_height: 이미지 크기
    class_map: 클래스 이름과 ID 매핑 딕셔너리
    """
    target_boxes = []
    face_box = None
    neck_box = None
    body_box = None
    # 모든 라벨 파싱
    parsed_labels = []
    for label in base_labels:
        parts = label.strip().split()
        class_id = int(parts[0])
        class_name = [k for k, v in class_map.items() if v == class_id][0]
        x_center = float(parts[1]) * img_width
        y_center = float(parts[2]) * img_height
        width = float(parts[3]) * img_width
        height = float(parts[4]) * img_height
```

```
            x1 = int(x_center - width/2)
            y1 = int(y_center - height/2)
            x2 = int(x_center + width/2)
            y2 = int(y_center + height/2)
            box_info = (x1, y1, x2, y2, x_center, y_center, width, height, class_name)
            parsed_labels.append(box_info)
            # 특정 부위 박스 저장
            if class_name == '얼굴':
                face_box = box_info
            elif class_name == '목':
                neck_box = box_info
            elif class_name == '상체':
                body_box = box_info
        # 타겟 유형에 따라 적절한 박스 선택
        if target_type == '얼굴제외':
            # 얼굴이 없으면 모든 박스 중에서 선택
            if face_box is None:
                target_boxes = [box for box in parsed_labels]
            else:
                # 얼굴 박스와 겹치지 않는 박스 찾기
                face_x1, face_y1, face_x2, face_y2 = face_box[:4]
                for box in parsed_labels:
                    x1, y1, x2, y2 = box[:4]
                    # 얼굴과 겹치지 않는 박스 중 큰 박스 선택 (상체, 팔 등)
                    overlap = (min(x2, face_x2) - max(x1, face_x1)) > 0 and (min(y2, face_y2) -
max(y1, face_y1)) > 0
                    if not overlap and box[8] in ['상체', '팔', '목', '머리카락', '손']:
```

```
                target_boxes.append(box)
    elif target_type == '목영역':
        # 목도리는 목과 상체 상단 영역에 배치
        if neck_box is not None:
            target_boxes.append(neck_box)
        elif body_box is not None:
            # 상체가 있으면 상단 부분을 목 영역으로 간주
            x1, y1, x2, y2, x_center, y_center, width, height, = body_box
            # 상체 상단 1/4 영역만 사용
            new_y1 = y1
            new_y2 = y1 + height / 4
            new_height = new_y2 - new_y1
            new_y_center = (new_y1 + new_y2) / 2
            modified_box = (x1, int(new_y1), x2, int(new_y2), x_center, new_y_center, width, new_height, '상체상단')
            target_boxes.append(modified_box)
    elif target_type in ['머리또는_머리카락']:
        # 머리나 머리카락 중 하나 선택
        for box in parsed_labels:
            if box[8] in ['머리', '머리카락']:
                target_boxes.append(box)
    else:
        # 특정 부위 찾기 (입, 눈, 귀 등)
        for box in parsed_labels:
            if box[8] == target_type:
                target_boxes.append(box)
    return target_boxes
```

액세서리 오버레이 함수 (확장 버전)

```
def overlay_accessory_advanced(base_image, accessory_image, accessory_class, target_type, base_labels, class_map):
    """
    base_image: 기본 이미지 (OpenCV 형식)
    accessory_image: 액세서리 이미지 (알파 채널 포함, RGBA)
    accessory_class: 액세서리 클래스 이름
    target_type: 액세서리 배치 유형 ('입', '얼굴' 제외', '목_영역' 등)
    base_labels: 기본 이미지의 YOLO 형식 라벨 목록
    class_map: 클래스 이름과 ID 매핑 딕셔너리
    """
    # 결과 이미지 (기본 이미지 복사)
    result_image = base_image.copy()
    img_height, img_width = result_image.shape[:2]
    # 대상 부위의 바운딩 박스 찾기
    target_part_boxes = find_target_parts(base_labels, target_type, img_width, img_height, class_map)
    if not target_part_boxes:
        return result_image, base_labels  # 대상 부위가 없으면 원본 반환
    # 대상 부위 선택 (여러 개일 경우 랜덤으로 선택)
    box_data = random.choice(target_part_boxes)
    x1, y1, x2, y2, x_center, y_center, width, height, part_name = box_data
    # 액세서리 크기 조절 및 위치 조정 (액세서리별 특화 설정)
    scale_factor = 1.0
    offset_x, offset_y = 0, 0
    # 액세서리 유형별 특화 설정
    if accessory_class == '안경':
```

```
        scale_factor = random.uniform(1.2, 1.5)
        offset_y = -height * 0.1
    elif accessory_class == '모자':
        scale_factor = random.uniform(1.0, 1.3)
        offset_y = -height * 0.8
    elif accessory_class == '귀걸이':
        scale_factor = random.uniform(0.3, 0.5)
        offset_y = height * 0.4
    elif accessory_class == '목걸이':
        scale_factor = random.uniform(0.8, 1.2)
        offset_y = height * 0.5
    elif accessory_class == '목도리':
        # 목도리는 목과 상체 상단 영역에 넓게 배치
        scale_factor = random.uniform(1.5, 2.0)  # 더 넓게 설정
        offset_y = height * 0.2  # 약간 아래로 위치 조정
        if part*name* == *'상체*상단':
            # 상체 상단인 경우 더 아래로 조정
            offset_y = 0
    elif accessory_class == '수염':
        # 수염은 입 아래에 배치
        scale_factor = random.uniform(0.8, 1.2)
        offset_y = height * 0.5
    elif accessory_class == '벌린입(치아)':
        # 벌린입은 입 위치에 정확히 배치
        scale_factor = random.uniform(0.9, 1.1)
        offset_y = 0
```

```
elif accessory_class == '리본':
    # 리본은 머리카락이나 옷에 배치
    scale_factor = random.uniform(0.2, 0.4)
    if part_name == '머리카락':
        offset_y = random.uniform(-0.2, 0.2) * height
    else:
        offset_y = random.uniform(-0.1, 0.1) * height
elif accessory_class == '귀도리':
    # 귀도리는 머리 주변에 배치
    scale_factor = random.uniform(0.8, 1.0)
    offset_y = 0
elif accessory_class == '머리장식':
    # 머리장식은 머리 상단이나 머리카락에 배치
    scale_factor = random.uniform(0.3, 0.6)
    offset_y = -height * 0.3 if part_name == '머리' else 0
elif accessory_class == '보석':
    # 보석은 다양한 위치에 배치 가능
    scale_factor = random.uniform(0.15, 0.3)
    offset_x = random.uniform(-0.3, 0.3) * width
    offset_y = random.uniform(-0.3, 0.3) * height
elif accessory_class == '담배':
    # 담배는 입 근처에 배치
    scale_factor = random.uniform(0.5, 0.8)
    offset_x = width * 0.5  # 입 옆으로 offset
    offset_y = 0
elif accessory_class == '장식':
```

```
        # 장식은 다양한 위치에 배치 가능
        scale_factor = random.uniform(0.2, 0.5)
        offset_x = random.uniform(-0.2, 0.2) * width
        offset_y = random.uniform(-0.2, 0.2) * height
    # 액세서리 이미지를 PIL로 변환 (알파 채널 처리를 위해)
    if isinstance(accessory_image, np.ndarray):
        accessory_pil = Image.fromarray(cv2.cvtColor(accessory_image,
cv2.COLOR_BGRA2RGBA))
    else:  # 이미 PIL 이미지인 경우
        accessory_pil = accessory_image
    # 크기 계산 - 여기서 액세서리 크기 조절이 이루어집니다
    new_width = int(width * scale_factor)
    aspect_ratio = accessory_pil.width / accessory_pil.height
    new_height = int(new_width / aspect_ratio)
    # 액세서리 크기 조절 - 고품질 리사이징 알고리즘 사용
    accessory_pil = accessory_pil.resize((new_width, new_height), Image.LANCZOS)
    # 변환과 회전 적용 (약간의 랜덤성 추가)
    rotation = random.uniform(-10, 10)
    accessory_pil = accessory_pil.rotate(rotation, expand=True, resample=Image.BICUBIC)
    # 합성 좌표 계산
    paste_x = int(x_center - accessory_pil.width/2 + offset_x)
    paste_y = int(y_center - accessory_pil.height/2 + offset_y)
    # 이미지가 경계를 벗어나지 않도록 조정
    paste_x = max(0, min(paste_x, img_width - accessory_pil.width))
    paste_y = max(0, min(paste_y, img_height - accessory_pil.height))
    # 액세서리 이미지 합성
    if accessory_pil.mode == 'RGBA':
```

```
    # 알파 채널이 있는 이미지를 NumPy 배열로 변환
    accessory_np = np.array(accessory_pil)
    # ROI 영역이 이미지 경계 내에 있는지 확인
    roi_height, roi_width = accessory_np.shape[:2]
    if paste_y + roi_height > img_height:
        roi_height = img_height - paste_y
    if paste_x + roi_width > img_width:
        roi_width = img_width - paste_x
    # 크기가 조정된 액세서리의 일부만 사용
    accessory_np = accessory_np[:roi_height, :roi_width]
    # 알파 채널과 RGB 채널 분리
    if accessory_np.shape[2] == 4:  # RGBA 확인
        rgb = accessory_np[:, :, :3]
        alpha = accessory_np[:, :, 3:4] / 255.0  # 알파 값을 0-1로 정규화
        # 합성 영역 선택
        roi = result_image[paste_y:paste_y+roi_height, paste_x:paste_x+roi_width]
        # 알파 블렌딩 적용
        if roi.shape[:2] == rgb.shape[:2]:  # 치수 확인
            # OpenCV는 BGR, PIL은 RGB로 작업하므로 변환
            blended = roi *(1 - alpha) + cv2.cvtColor(rgb, cv2.COLOR_RGB2BGR)* alpha
            result_image[paste_y:paste_y+roi_height, paste_x:paste_x+roi_width] = blended
# 새 액세서리의 바운딩 박스 생성
accessory_width = accessory_pil.width
accessory_height = accessory_pil.height
accessory_x_center = paste_x + accessory_width/2
accessory_y_center = paste_y + accessory_height/2
```

```
    # YOLO 형식으로 새 라벨 생성
    accessory_class_id = class_map[accessory_class]
    new_label = create_yolo_annotation(
        accessory_class_id,
        accessory_x_center,
        accessory_y_center,
        accessory_width,
        accessory_height,
        img_width,
        img_height
    )
    # 기존 라벨에 새 라벨 추가
    updated_labels = base_labels + [new_label]
    return result_image, updated_labels
```

메인 함수: 합성 데이터셋 생성 (고급 버전)

```
def generate_advanced_composited_dataset(
    base_images_dir,
    accessories_dir,
    output_dir,
    class_map,
    accessory_target_map,
    num_combinations=1000
):
    """
    base_images_dir: 기본 신체 이미지가 있는 디렉토리
    accessories_dir: 액세서리 이미지가 있는 디렉토리 (하위 디렉토리로 분류)
```

```
output_dir: 출력 이미지와 라벨을 저장할 디렉토리
class_map: 클래스 이름과 ID 매핑 딕셔너리
accessory_target_map: 액세서리와 대상 신체 부위/위치 지정 매핑
num_combinations: 생성할 조합 수
"""
# 경로 정규화
base_images_dir = normalize_path(base_images_dir)
accessories_dir = normalize_path(accessories_dir)
output_dir = normalize_path(output_dir)
print(f"기본 이미지 디렉토리: {base_images_dir}")
print(f"액세서리 디렉토리: {accessories_dir}")
print(f"출력 디렉토리: {output_dir}")
# 출력 디렉토리 생성
os.makedirs(output_dir, exist_ok=True)
images_output_dir = os.path.join(output_dir, "images")
labels_output_dir = os.path.join(output_dir, "labels")
os.makedirs(images_output_dir, exist_ok=True)
os.makedirs(labels_output_dir, exist_ok=True)
# 기본 이미지 목록 가져오기
base_image_paths = glob.glob(os.path.join(base_images_dir, "*.jpg")) + \
                   glob.glob(os.path.join(base_images_dir, "*.png"))
print(f"기본 이미지 수: {len(base_image_paths)}")
# 액세서리 디렉토리 컨텐츠 확인
print("\n액세서리 디렉토리 내용:")
for item in os.listdir(accessories_dir):
    item_path = os.path.join(accessories_dir, item)
```

```
    if os.path.isdir(item_path):
        png_count = len(glob.glob(os.path.join(item_path, "*.png")))
        print(f"  - {item}: PNG 파일 {png_count}개")
# 액세서리 이미지 로드
accessories = {}
for accessory_class in os.listdir(accessories_dir):
    class_dir = os.path.join(accessories_dir, accessory_class)
    if os.path.isdir(class_dir):
        accessories[accessory_class] = []
        for img_path in glob.glob(os.path.join(class_dir, "*.png")):
            # 경로 정규화
            img_path = normalize_path(img_path)
            # RGBA 이미지로 로드 (알파 채널 포함)
            try:
                accessory_img = Image.open(img_path).convert("RGBA")
                accessories[accessory_class].append(accessory_img)
            except Exception as e:
                print(f"Error loading {img_path}: {e}")
# 로드된 액세서리 통계
print("\n로드된 액세서리 통계:")
total_accessories = 0
for acc_class, acc_list in accessories.items():
    print(f"  - {acc_class}: {len(acc_list)}개")
    total_accessories += len(acc_list)
print(f"총 액세서리 이미지: {total_accessories}개")
if total_accessories == 0:
```

```
        print("오류: 액세서리 이미지가 없습니다. 프로그램을 종료합니다.")
        return
    # 조합 생성
    count = 0
    for i in range(num_combinations):
        if count >= num_combinations:
            break
        # 무작위 기본 이미지 선택
        base_img_path = random.choice(base_image_paths)
        # 경로 정규화
        base_img_path = normalize_path(base_img_path)
        base_img = load_image_safely(base_img_path)
        if base_img is None:
            print(f"Failed to load image: {base_img_path}")
            continue
        # 해당 라벨 파일 로드
        label_path = normalize_path(os.path.splitext(base_img_path)[0].replace("images",
"labels") + ".txt")
        if not os.path.exists(label_path):
            print(f"Label file not found: {label_path}")
            continue
        with open(label_path, 'r') as f:
            base_labels = f.readlines()
        # 무작위 액세서리 선택 (1-3개)
        num_accessories = random.randint(1, 3)
        available_accessories = [acc for acc in accessories.keys() if acc in
accessory_target_map and accessories[acc]]
```

```
        if not available_accessories:
            print("No valid accessories found")
            continue
        accessory_classes = random.sample(available_accessories, min(num_accessories,
len(available_accessories)))
        result_img = base_img.copy()
        result_labels = base_labels.copy()
        # 각 액세서리 합성
        for acc_class in accessory_classes:
            if not accessories[acc_class]:
                continue
            accessory_img = random.choice(accessories[acc_class])
            target_type = accessory_target_map.get(acc_class)
            if target_type:
                try:
                    result_img, result_labels = overlay_accessory_advanced(
                        result_img, accessory_img, acc_class, target_type, result_labels, class_map
                    )
                except Exception as e:
                    print(f"Error overlaying {acc_class}: {e}")
                    continue
        # 결과 저장
        output_img_path = os.path.join(images_output_dir, f"composited{count:06d}.jpg")
        output_label_path = os.path.join(labels_output_dir, f"composited{count:06d}.txt")
        # 경로 정규화
        output_img_path = normalize_path(output_img_path)
        output_label_path = normalize_path(output_label_path)
```

```
        cv2.imwrite(output_img_path, result_img)
        with open(output_label_path, 'w') as f:
            f.writelines(result_labels)
        count += 1
        if count % 10 == 0:  # 더 자주 진행 상황 보고
            print(f"Generated {count}/{num_combinations} images")
    print(f"Dataset generation complete. Created {count} composited images.")
```

사용 예시

```
if name == "main":
    # 클래스 맵핑 정의
    class_map = {
        '사람전체': 0, '머리': 1, '얼굴': 2, '눈': 3, '코': 4, '입': 5, '귀': 6, '머리카락': 7,
        '목': 8, '상체': 9, '팔': 10, '손': 11, '모자': 12, '안경': 13, '눈썹': 14, '수염': 15,
        '벌린입(치아)': 16, '목도리': 17, '넥타이': 18, '리본': 19, '귀도리': 20, '귀걸이': 21,
        '목걸이': 22, '장식': 23, '머리장식': 24, '보석': 25, '담배': 26
    }
    # 액세서리와 대상 신체 부위 매핑 - 확장 버전 (목도리 추가)
    accessory_target_map = {
        '모자': '머리',
        '안경': '눈',
        '눈썹': '눈',
        '귀걸이': '귀',
        '목걸이': '목',
        '목도리': '목_영역',     # 목과 상체 상단에 배치
        '리본': '얼굴_제외',     # 얼굴 빼고 아무데나
        '넥타이': '목',
```

```
        '수염': '입',           # 입 주변에 위치
        '벌린입(치아)': '입',     # 입에 위치
        '장식': '얼굴_제외',      # 얼굴 빼고 아무데나
        '귀도리': '머리',         # 머리에 위치
        '머리장식': '머리또는머리카락',  # 머리나 머리카락에 위치
        '보석': '얼굴_제외',      # 얼굴 빼고 아무데나
        '담배': '입'            # 입에 위치
    }
    # 실행
    generate_advanced_composited_dataset(
        base_images_dir="C:/capstone/data/validation/images",
        accessories_dir="C:/capstone/data/validation/accessories_png",
        output_dir="C:/capstone/data/validation/composited",
        class_map=class_map,
        accessory_target_map=accessory_target_map,
        num_combinations=2000  # 생성할 이미지 수
    )
import json
import os
import glob
# 최종 통합 데이터셋에서 '안경' 클래스 번호 (dataset.yaml에서 'x'는 n-1번)
class_mapping = {
    "담배": 26
}
```

두 디렉토리 (안경 JSON과 선글라스 JSON 파일이 있는 경로)

input_dirs = [

#r "C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\2.Validation\라벨링데이터\VL1\ABSTRACT_SKETCH\L1_8\L2_58\L3_1003",

r"C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\2.Validation\라벨링데이터\VL1\ABSTRACT_SKETCH\L1_3\L2_24\L3_305"

]

#output_dir = "C:/capstone/data/train/labels"

output_dir = "C:/capstone/data/validation/labels"

os.makedirs(output_dir, exist_ok=True)

라벨 변환 규칙: "안경테"와 "선글라스" 모두 "안경"으로 변환

def map_category(label):

    if label in ["스냅백", "베레모", "중절모", "비니", "털모자", "모자(캡)", "카우보이모자", "학사모", "왕관"]:

        return "모자"

    return label

def map_category(label):

    if label in ["입술/입", "이빨/치아"]:

        return "벌린입(치아)"

    return label

def map_category(label):

if label in ["앉아있는 사람", "걷는 사람"]:

return "사람전체"

return label

```
def map_category(label):
    if label == "담배":
        return "담배"
    return label
```

변환된 파일 개수를 카운트할 변수

```
converted_count = 0
```

각 입력 디렉토리의 JSON 파일들을 순회

```
for input_dir in input_dirs:
    json_files = glob.glob(os.path.join(input_dir, "*.json"))
    for json_file in json_files:
        with open(json_file, "r", encoding="utf-8") as f:
            data = json.load(f)
        # abstract_image 정보에서 이미지 해상도와 경로, 바운딩 박스 추출
        abs_img = data.get("abstract_image", {})
        abs_width = abs_img.get("abs_width", 300)
        abs_height = abs_img.get("abs_height", 300)
        abs_path = abs_img.get("abs_path", "default.jpg")
        img_file_name = os.path.basename(abs_path)
        txt_file_name = os.path.splitext(img_file_name)[0] + ".txt"
        output_file_path = os.path.join(output_dir, txt_file_name)
        # abs_bbox는 문자열 형태로 제공되므로 파싱 (예: "[20,57.48957824707031,263,177]")
        abs_bbox_str = abs_img.get("abs_bbox", None)
```

```
if abs_bbox_str is None:
    continue
try:
    bbox_values = json.loads(abs_bbox_str)
    x, y, w, h = bbox_values
except Exception as e:
    print(f"Error parsing bbox in {json_file}: {e}")
    continue
# 중심 좌표 계산 및 정규화 (YOLO 포맷: [cx, cy, w, h] - 모두 0~1 범위)
cx = x + w / 2.0
cy = y + h / 2.0
cx_norm = cx / abs_width
cy_norm = cy / abs_height
w_norm = w / abs_width
h_norm = h / abs_height
# category 정보에서 레벨3 라벨 추출 및 변환
cat = data.get("category", {})
ctg_label = cat.get("ctg_nm_level3", None)
if ctg_label is None:
    continue
mapped_label = map_category(ctg_label)
if mapped_label not in class_mapping:
    continue  # 최종 통합 클래스에 없다면 무시
class_id = class_mapping[mapped_label]
# YOLO 형식의 txt 파일로 저장
with open(output_file_path, "w", encoding="utf-8") as f_out:
```

```
            f_out.write(f"{class_id} {cx_norm:.6f} {cy_norm:.6f} {w_norm:.6f} {h_norm:.6f}\n")

        converted_count += 1

        print(f"변환 완료: {json_file} -> {output_file_path}")

print(f"총 {converted_count}개의 파일이 변환되었습니다.")

import json

import cv2

import os

import numpy as np

# JSON 파일 경로 (실제 경로로 수정)

json_file = r"C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\1.Training\라벨링데이터\TL3\ABSTRACT_SKETCH\L1_8\L2_58\L3_996\s_0996_18224_211666.json"

# 이미지 파일 경로를 코드에서 직접 설정 (JSON 파일의 경로와 무관하게)

img_path = r"C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\1.Training\원천데이터\TS4\ABSTRACT_SKETCH_2\L1_8\L2_58\L3_996\s_0996_18224_211666.jpg"

# JSON 파일 읽기

with open(json_file, "r", encoding="utf-8") as f:

    data = json.load(f)

# abstract_image 섹션에서 바운딩 박스 정보 추출

abs_img = data.get("abstract_image", {})

abs_bbox_str = abs_img.get("abs_bbox", None)

if abs_bbox_str is None:

    print("바운딩 박스(abs_bbox) 정보가 JSON에 없습니다.")

    exit()

try:

    bbox = json.loads(abs_bbox_str)
```

```
    # bbox 값: [x, y, w, h]
    x, y, w, h = bbox
except Exception as e:
    print(f"바운딩 박스 파싱 오류: {e}")
    exit()
# 이미지 로드
try:
    img_array = np.fromfile(img_path, np.uint8)
    image = cv2.imdecode(img_array, cv2.IMREAD_COLOR)
    if image is None:
        raise Exception("이미지 디코딩 실패")
    cv2.imshow("Loaded Image", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
except Exception as e:
    print(f"이미지 로드 실패: {img_path}\n오류: {e}")
# 바운딩 박스 좌표 계산 (좌측 상단, 우측 하단) 및 정수 변환
x1, y1 = int(x), int(y)
x2, y2 = int(x + w), int(y + h)
# 이미지에 바운딩 박스 그리기 (녹색 테두리, 두께 2)
cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
# 이미지 표시
cv2.imshow("Image with Bounding Box", image)
cv2.waitKey(0)
```

cv2.destroyAllWindows()

import json

import os

import glob

입력 JSON 파일들이 있는 폴더 경로 (실제 경로로 수정)

input_dir = r"C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\2.Validation\라벨링데이터\VL1\ABSTRACT_SKETCH\L1_6\L2_46\L3_816"

수정된 JSON 파일을 저장할 출력 폴더 경로 (실제 경로로 수정)

output_dir = r"C:\049.스케치, 아이콘 인식용 다양한 추상 이미지 데이터\01.데이터\2.Validation\라벨링데이터\VL1\ABSTRACT_SKETCH\L1_6\L2_46\L3_816_hlaf_width"

os.makedirs(output_dir, exist_ok=True)

modified_count = 0

json_files = glob.glob(os.path.join(input_dir, "*.json"))

for json_file in json_files:

    with open(json_file, "r", encoding="utf-8") as f:

        data = json.load(f)

    # category 정보 확인: '귀걸이'인 경우에만 처리

    category = data.get("category", {})

    ctg_nm_level3 = category.get("ctg_nm_level3", "")

    if ctg_nm_level3 != "귀걸이":

        continue

    # abstract_image의 abs_bbox 값 추출 (예: "[78,41.48957824707031,142,213]")

    abs_img = data.get("abstract_image", {})

    abs_bbox_str = abs_img.get("abs_bbox", None)

    if abs_bbox_str is None:

        continue

```
    try:
        # 문자열을 리스트로 파싱 (형식: [x, y, w, h])
        bbox = json.loads(abs_bbox_str)
        if len(bbox) != 4:
            print(f"{json_file} - 예상과 다른 bbox 형식입니다.")
            continue
        x, y, w, h = bbox
        # 가로 길이(w)를 절반으로 줄임
        new_w = w / 2.0
        new_bbox = [x, y, new_w, h]
        # 수정된 bbox를 다시 문자열로 저장
        abs_img["abs_bbox"] = json.dumps(new_bbox)
        data["abstract_image"] = abs_img
    except Exception as e:
        print(f"{json_file} 처리 중 오류 발생: {e}")
        continue
    # 출력 폴더에 같은 파일명으로 저장
    output_file_path = os.path.join(output_dir, os.path.basename(json_file))
    with open(output_file_path, "w", encoding="utf-8") as f:
        json.dump(data, f, ensure_ascii=False, indent=4)
    modified_count += 1
    print(f"변환 완료: {json_file} -> {output_file_path}")
print(f"총 {modified_count}개의 '귀걸이' JSON 파일이 업데이트되었습니다.")
```