

## TryOnDiffusion: A Tale of Two UNets 전처리 과정

인물과 옷 이미지에서 파싱 맵과 포즈 키포인트를 획득하고 복잡한 옷 및 포즈 키포인트를 정확하게 추출하는 전처리 과정 방법을 논문 토대로 정리

### 1. 인체와 의복 이미지로부터 파싱 맵과 포즈 키포인트를 추출하여 다양한 부위와 위치 정보를 확보합니다.

- Sp, Sg, Jp, Jg를 예측하여 인물과 옷 이미지에서 파싱 맵과 포즈 키포인트를 획득

Sp (separated person)	인간 파싱 맵, 이미지 내의 인 형상의 다양한 부분을 나타내는 레이블 정보를 포함하는 맵 (머리, 상체, 하체, 팔, 다리 등의 부위를 식별하는 역할 수행)
Sg (separated Garment)	옷 파싱 맵, 이미지 내의 의복의 다양한 부분을 나타내는 레이블 정보를 포함하는 맵으로 옷의 종류와 부위를 식별하는 데 사용
Jp (joint Person)	2D 포즈 키포인트, 인물 이미지에서 특정 부위의 위치를 나타내는 포즈 키포인트 정보
Jg (joint Germent)	옷 2D 포즈 키포인트, 옷 이미지에서 특정 부위의 위치를 나타내는 포즈 키포인트 정보

#### 1-1. Sp,Sg,Jp,Jg 예측 방법

(1) Sp와 Sg의 경우 : 휴먼 파싱(human parsing)

- **휴먼 파싱의 정의?**  
휴먼 파싱(Human Parsing)은 인간의 이미지나 영상에서 신체 부위와 의복 부위를 픽셀 단위로 분할하여 각 부위의 레이블 정보를 추출하는 작업을 의미
- 휴먼 파싱 방식 중 'Graphonomy: Universal Human Parsing via Graph Transfer Learning' 논문에 나오는 'Graphonomy'방식 이용
- 'Graphonomy' 작동원리 : 2가지 모듈로 구성되어 각각 그래프 내 추론, 그래프 간 전이 역할을 수행한다.

(1) Intra-Graph Reasoning (그래프 내 추론):

- 이미지를 그래프로 변환
- 각 픽셀은 노드, 관계는 엣지로 표현
- 픽셀 간의 관계를 활용하여 자세한 분할
- 인접한 픽셀 정보로 정확한 분할 및 레이블 할당

(2) Inter-Graph Transfer (그래프 간 전이):

- 다른 데이터셋의 그래프 가져옴
- 이미지를 그래프로 표현하여 새 데이터셋에 적용
- 학습된 그래프 지식 전이
- 새 데이터셋에서도 효과적인 휴먼 파싱 수행

(2) Jp와 Jg의 경우 : 포즈 추정(pose estimation)

- 포즈 추정의 정의?

주어진 이미지나 영상에서 사람이나 객체의 키포인트(관절) 위치를 예측하는 기술

- 포즈 추정 방식 중 'Towards Accurate Multi-person Pose Estimation in the Wild' 논문  
에 나오는 'multi-person pose estimation' 방식 사용

- multi-person pose estimation 의 작동 방식

(1) Faster RCNN 디텍터와 ResNet 활용:

- 사람 존재 가능성 있는 박스 위치 및 크기 예측 (Faster RCNN)
- 각 박스에 포함될 가능성 있는 사람의 키포인트 추정 (ResNet)

(2) 키포인트 추정 방식:

- 밀도 높은 키포인트 히트맵 및 오프셋 예측 (ResNet)
- 키포인트 유형별로 예측 결과 수집
- 새로운 집계 절차 도입하여 국부화된 키포인트 예측 획득

(3) 성능 향상 및 다양한 적용:

- 키포인트 기반 NMS 및 신뢰도 점수 계산 활용하여 정확한 결과 도출
- COCO keypoints 데이터셋에서 최첨단 결과 증명

## 2. 옷 분할 이미지(Ic)와 인물 분할 이미지(Ia)를 생성하여 의복 부분과 인체 형상을 따로 추출하기

- (1) Ic (Isolated clothing, 옷 RGB 이미지): 옷 이미지에서 실제 의복 부분을 따로 분할한 이미지. 옷 파싱 맵(Sg)을 사용하여 의복 부분을 분리하여 나타낸 것으로, 옷을 개별적으로 조작할 때 사용
- (2) Ia (Isolated atomy, 인물 RGB 이미지): 인물 이미지에서 옷 부분을 제거하고, 개인의 형상만을 보존한 이미지. 인물 파싱 맵(Sp)을 사용하여 옷 부분을 없앤 이미지로, 옷을 입지 않은 인물 이미지로 시착 결과의 정확성을 높이기 위해 사용

### 2-1. Ia 생성 원리

- 인물 이미지에서 옷이 입혀져 있는 부분을 바운딩 박스로 마스킹 처리하여 가려준 후 다시 그 위에 머리, 손, 다리 같은 신체 부분을 복사해서 붙여 넣는다. -> 원래 옷의 정보는 완전히 사라짐!
- Ia의 필요성 : 어려운 인간 포즈나 느슨한 의복과 같이 복잡한 상황에서 의복 정보 노출을 줄이고 시착 결과의 정확성을 향상시키는 데 사용. 어려운 자세나 느슨한 옷을 입은 경우에도 옷 정보를 최대한 없앴으로써 시착 결과의 정확성을 더 높일 수 있다!

### (3) 포즈 키포인트 정규화

Jp와 Jg의 포즈 키포인트를 [0, 1] 범위로 정규화하여 네트워크 입력으로 사용

### (4) 시착 조건 입력 생성

ctryon = (Ia, Jp, Ic, Jg)로 정의된 시착 조건 입력 생성

---

preprocessing 작업 관련 내용 정리

## 현재 사용중인 데이터

C > 바탕 화면 > try_on > image-background-remove-tool > sampledata		
이름	수정한 날짜	유형
cloth	2023-08-29 오전...	파일 폴더
cloth_parse	2023-08-29 오전...	파일 폴더
cloth_pose	2023-08-29 오전...	파일 폴더
model	2023-08-29 오전...	파일 폴더
model_parse	2023-08-29 오전...	파일 폴더
model_pose	2023-08-29 오전...	파일 폴더

01. 의류 이미지
02. 의류 parse.json 데이터
03. Cloth\_pose.json 데이터
04. 인물 이미지
05. 인물 parse.json 데이터
06. 인물 pose.json 데이터

## human parsing 이용한 옷 정보 지운 모델 이미지 생성(1a)

방식 : model\_parse.json 파일 내에서 상의에 해당하는 category\_id 추출해서 상의에 해당되는 id 영역 정보와 배경을 제거, image-background-remove-tool 모듈 활용

모듈 설치 : <https://github.com/OPHoperHPO/image-background-remove-tool/tree/master>

환경 설정 :

- python3.8
- conda install pytorch torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia

## 01. category\_id 찾기

```
import os
import json

# JSON 파일들이 있는 폴더 경로
folder_path = './sampledata/model_parse/'

# 폴더 내의 모든 JSON 파일 찾기
json_files = [file for file in os.listdir(folder_path) if file.endswith('.json')]

# 모든 JSON 파일 내의 category_name과 category_id 추출
category_info = {} # 카테고리 정보를 저장할 딕셔너리

for json_file in json_files:
    json_path = os.path.join(folder_path, json_file)

    with open(json_path, 'r') as file:
        data = json.load(file)
        for region_key, region_data in data.items():
            if region_key != 'file_name':
                category_name = region_data.get('category_name')
                category_id = region_data.get('category_id')
                if category_name and category_id is not None:
                    category_info[category_name] = category_id

# 카테고리 이름과 ID 출력
print("Category information:")
for name, id in category_info.items():
    print(f"Name: {name}, ID: {id}")
```

## 02. 배경와 의류 이미지 제거(ia)

```
import os
import json
from PIL import Image, ImageDraw
import torch
from carvekit.api.high import HIInterface

# JSON 파일들이 있는 폴더 경로
json_folder_path = './sampledata/model_parse/'
input_folder = './segment_output'
output_folder = './delected_cloth_model' # 변경 가능한 출력 폴더 경로
```

```

# 폴더 내의 모든 JSON 파일 찾기
json_files = [file for file in os.listdir(json_folder_path) if file.endswith('.json')]

# 원하는 category_id들
desired_category_ids = [9, 7, 8]

# CarveKit 인터페이스 설정
interface = HiInterface(object_type="object",
                        batch_size_seg=5,
                        batch_size_matting=1,
                        device='cuda' if torch.cuda.is_available() else 'cpu',
                        seg_mask_size=640,
                        matting_mask_size=2048,
                        trimap_prob_threshold=231,
                        trimap_dilation=30,
                        trimap_erosion_iters=5,
                        fp16=False)

# JSON 파일을 이용하여 이미지 처리 및 저장
for json_file in json_files:
    json_path = os.path.join(json_folder_path, json_file)

    with open(json_path, 'r') as file:
        data = json.load(file)

        jpg_filename = json_file[:-5] + '.jpg' # JSON 파일 이름으로부터 JPG 파일 이름 생성
        image_filename = os.path.join('./sampledata/model/', jpg_filename) # JPG 파일 경로 구성

        if os.path.exists(image_filename):
            # CarveKit를 이용하여 배경 제거
            input_path = os.path.join(input_folder, jpg_filename)
            images_without_background = interface([input_path])
            cat_wo_bg = images_without_background[0]

            # Ensure that the image mode is 'RGB' before drawing
            if cat_wo_bg.mode != 'RGB':
                cat_wo_bg = cat_wo_bg.convert('RGB')

            # 이미지에 해당하는 카테고리 영역을 desired_category_ids의 색상으로 채우기
            draw = ImageDraw.Draw(cat_wo_bg)

            # 가라가라
            for region_key, region_data in data.items():
                if region_key == 'file_name':
                    continue
                category_id = region_data.get('category_id')

                if category_id in desired_category_ids:
                    segmentation = region_data.get('segmentation', [])

                    for segment in segmentation:
                        points = [(x, y) for x, y in segment]
                        draw.polygon(points, outline=None, fill=(128, 128, 128, 0)) #

```

회색(투명)으로 설정

```
# 이미지 저장
output_path = os.path.join(output_folder, jpg_filename)
cat_wo_bg.save(output_path, format='JPEG') # JPG 포맷으로 저장
print('Processed image saved as', output_path)
else:
    print(f'Image not found: {image_filename}')
```

결과



## 배경 지운 옷 이미지 생성(lc)

```
import os
import torch
from carvekit.api.high import HiInterface
from PIL import Image

input_folder = './sampledata/cloth'
output_folder = './germent_noBG' # 변경 가능한 출력 폴더 경로

if not os.path.exists(output_folder):
    os.makedirs(output_folder)

interface = HiInterface(object_type="object",
                        batch_size_seg=5,
                        batch_size_matting=1,
                        device='cuda' if torch.cuda.is_available() else 'cpu',
                        seg_mask_size=640,
                        matting_mask_size=2048,
                        trimap_prob_threshold=231,
                        trimap_dilation=30,
                        trimap_erosion_iters=5,
                        fp16=False)

jpg_files = [file for file in os.listdir(input_folder) if file.endswith('.jpg')]

for jpg_file in jpg_files:
    image_path = os.path.join(input_folder, jpg_file)
    images_without_background = interface([image_path])
    cat_wo_bg = images_without_background[0]

    # Ensure that the image mode is 'RGB' before saving as JPEG
    if cat_wo_bg.mode != 'RGB':
        cat_wo_bg = cat_wo_bg.convert('RGB')

    output_path = os.path.join(output_folder, jpg_file)
    output_path = os.path.splitext(output_path)[0] + '.jpg' # Change the extension to '.jpg'
    cat_wo_bg.save(output_path)

print("이미지 처리 및 저장 완료")
```



결과



키 포인트 정보(jp, jg) 정규화

```
import os
import json

def normalize_keypoints(keypoints, image_width, image_height):
    normalized_keypoints = []

    for i in range(0, len(keypoints), 3):
        x = keypoints[i]
        y = keypoints[i + 1]
        v = keypoints[i + 2]

        normalized_x = x / image_width
        normalized_y = y / image_height
```

```
        normalized_keypoints.extend([normalized_x, normalized_y, v])

    return normalized_keypoints

input_folder = './sampledata/model_pose'
output_folder = './jp' # 변경 가능한 출력 폴더 경로

if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# 폴더 내의 모든 JSON 파일 찾기
json_files = [file for file in os.listdir(input_folder) if file.endswith('.json')]

for json_file in json_files:
    json_path = os.path.join(input_folder, json_file)
    output_path = os.path.join(output_folder, json_file)

    with open(json_path, 'r') as file:
        data = json.load(file)

        image_width = data['image_size']['width']
        image_height = data['image_size']['height']

        normalized_keypoints = normalize_keypoints(data['landmarks'], image_width,
image_height)
        data['landmarks'] = normalized_keypoints

    with open(output_path, 'w') as file:
        json.dump(data, file)

    print(f'Normalized keypoints saved to {output_path}')
```