

Instance Segmentation를 활용한

이미지 분석 및 가공

■
**목차
페이지**

SEGMENTATION

01

프로젝트
연구 주제

03

Annotation
시각화

05

Instance
Segmentation
Model

02

COCO dataset

04

데이터셋
전처리

01

프로젝트 연구 주제

01

프로젝트 연구 주제



사용자가 원하는 객체 혹은 객체 이외의
영역을 가공하여 이미지 재구성

01

프로젝트 연구 주제



사용자가 원하는 객체 혹은 객체 이외의
영역을 가공하여 이미지 재구성



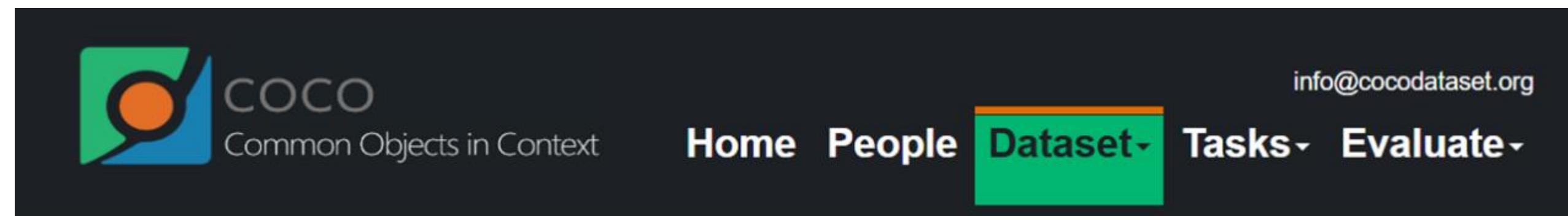
Instance Segmentation의 U-net 활용

02

COCO dataset 소개

02 | COCO Dataset

Object Detection, Segmentation, keypoint Detection 등의 Annotation 정보를 제공하는 데이터셋
YOLO v4, Deeplab 등 우수한 모델들의 학습 데이터로 사용



Tools

COCO API

Images

2014 Train images [83K/13GB]
2014 Val images [41K/6GB]
2014 Test images [41K/6GB]
2015 Test images [81K/12GB]
2017 Train images [118K/18GB]
2017 Val images [5K/1GB]
2017 Test images [41K/6GB]
2017 Unlabeled images [123K/19GB]

Annotations

2014 Train/Val annotations [241MB]
2014 Testing Image info [1MB]
2015 Testing Image info [2MB]
2017 Train/Val annotations [241MB]
2017 Stuff Train/Val annotations [1.1GB]
2017 Panoptic Train/Val annotations [821MB]
2017 Testing Image info [1MB]
2017 Unlabeled Image info [4MB]

02 | COCO Dataset 구성

Images

Train2017

Val2017

Test2017

Json 파일

JSON Annotation

Info

생성 일자 등을 가지는 헤더 정보

License

이미지 파일들의 라이선스 정보

Images

모든 이미지들의 id, 파일명, 이미지 너비, 높이 정보

Annotation

대상 image 및 object id

Segmentation, bounding box, 픽셀 영역 등의 상세 정보

Categories

80개 오브젝트 카테고리에 대한 id, 이름, Group을 가짐

02 | COCO Dataset 구성

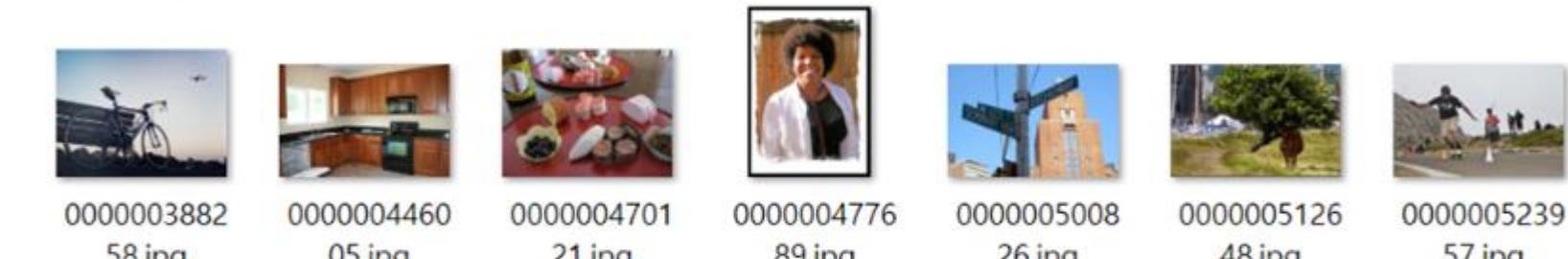
Training Data

- 2017 Train Images 사용
- 118,000장의 이미지 데이터로 구성



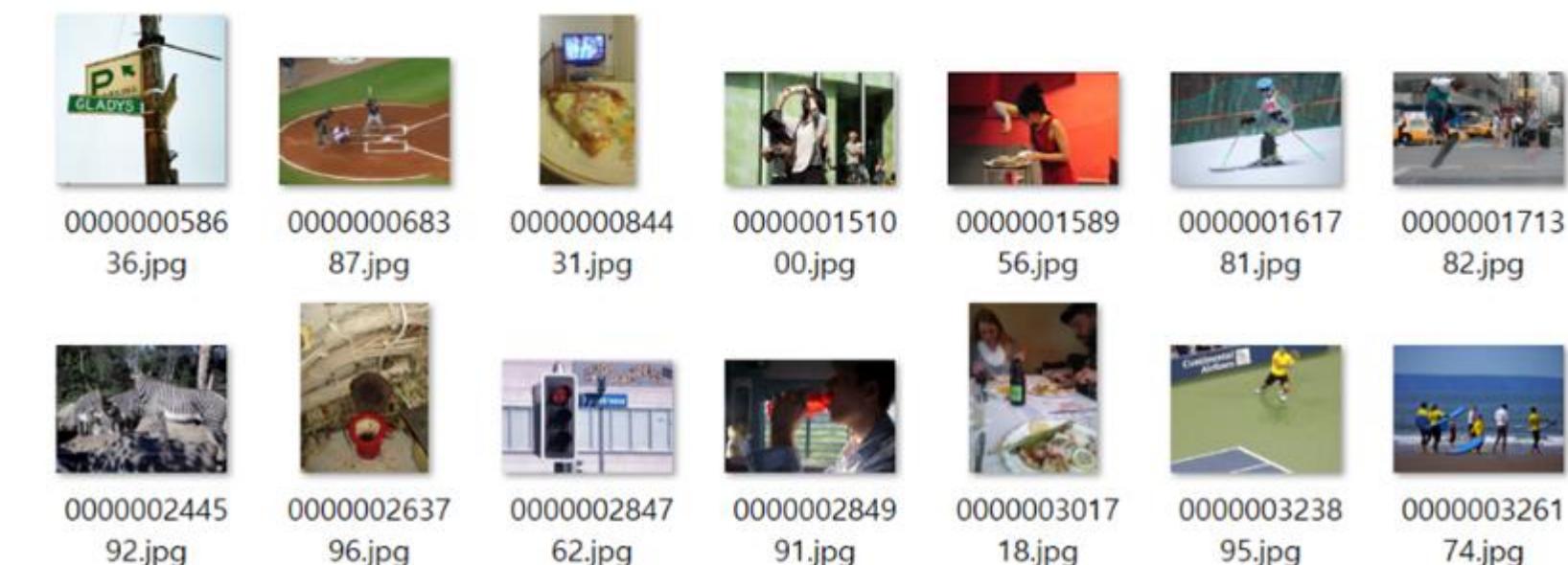
Validation Data

- 2017 Val Images 사용
- 5,000장의 이미지 데이터로 구성



Testing Data

- 2017 Test Images 사용
- 41,000장의 이미지 데이터로 구성



02 | COCO Dataset 구성

Training Data

- 2017 Train Images 사용
- 118,000장의 이미지 데이터로 구성

 person_keypoints_train2017.json	2024-05-14 오후 11:43	JSON 파일	233,286KB
 person_keypoints_val2017.json	2024-05-14 오후 11:43	JSON 파일	9,786KB
 captions_train2017.json	2024-05-14 오후 11:43	JSON 파일	89,713KB
 captions_val2017.json	2024-05-14 오후 11:43	JSON 파일	3,782KB
 instances_val2017.json	2024-05-14 오후 11:42	JSON 파일	19,520KB

Validation Data

- 2017 Val Images 사용
- 5,000장의 이미지 데이터로 구성

Testing Data

- 2017 Test Images 사용
- 41,000장의 이미지 데이터로 구성

03

Annotation 시작화

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

```
import os
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import matplotlib.colors as colors
import seaborn as sns
import numpy as np
import random

from random import shuffle
from PIL import Image

from pycocotools.coco import COCO
```

한 Image에 속하는 모든 annotation을 불러오는 코드 등이 포함된
COCO 데이터셋의 시각화 및 EDA 작업을 도와주는 함수

3-0. Pycocotools API

getAnnIds: Get ann ids that satisfy given filter conditions.

getCatIds: Get cat ids that satisfy given filter conditions.

getImgIds: Get img ids that satisfy given filter conditions.

loadAnns: Load anns with the specified ids.

loadCats: Load cats with the specified ids.

loadImgs: Load imgs with the specified ids.

showAnns: Display the specified annotations.

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-1. COCO dataset 다운로드

COCO dataset 다운로드 및 압축 해제

```
!wget -nc http://images.cocodataset.org/zips/train2017.zip
!unzip -o -q /content/train2017.zip
```

```
!wget -nc http://images.cocodataset.org/zips/val2017.zip
!unzip -o -q /content/val2017.zip
```

```
!wget -nc http://images.cocodataset.org/zips/test2017.zip
!unzip -o -q /content/test2017.zip
```

annotation 다운로드 및 압축 해제

```
!wget -nc http://images.cocodataset.org/annotations/annotations\_trainval2017.zip
!unzip -o -q /content/annotations_trainval2017.zip
```

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-2. 데이터셋 경로 지정 및 주석 설정

```
# 경로 지정
dataDir = '/content'
dataType = 'train2017'
annFile = '{}/annotations/instances_{}.json'.format(dataDir, dataType)
imageDir = '{}/{}/'.format(dataDir, dataType)

coco = COCO(annFile)
```

3-3. 데이터셋 카테고리 출력

```
# 카테고리 내용 출력
category_ids = coco.getCatIds()
num_categories = len(category_ids)
print('number of categories: ', num_categories)
for ids in category_ids:
    cats = coco.loadCats(ids=ids)
    print(cats)
```

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

데이터기반 인공지

```
number of categories: 80
[{"supercategory": "person", "id": 1, "name": "person"}]
[{"supercategory": "vehicle", "id": 2, "name": "bicycle"}]
[{"supercategory": "vehicle", "id": 3, "name": "car"}]
[{"supercategory": "vehicle", "id": 4, "name": "motorcycle"}]
[{"supercategory": "vehicle", "id": 5, "name": "airplane"}]
[{"supercategory": "vehicle", "id": 6, "name": "bus"}]
[{"supercategory": "vehicle", "id": 7, "name": "train"}]
[{"supercategory": "vehicle", "id": 8, "name": "truck"}]
[{"supercategory": "vehicle", "id": 9, "name": "boat"}]
[{"supercategory": "outdoor", "id": 10, "name": "traffic light"}]
[{"supercategory": "outdoor", "id": 11, "name": "fire hydrant"}]
[{"supercategory": "outdoor", "id": 13, "name": "stop sign"}]
[{"supercategory": "outdoor", "id": 14, "name": "parking meter"}]
[{"supercategory": "outdoor", "id": 15, "name": "bench"}]
[{"supercategory": "animal", "id": 16, "name": "bird"}]
[{"supercategory": "animal", "id": 17, "name": "cat"}]
[{"supercategory": "animal", "id": 18, "name": "dog"}]
[{"supercategory": "animal", "id": 19, "name": "horse"}]
[{"supercategory": "animal", "id": 20, "name": "sheep"}]
[{"supercategory": "animal", "id": 21, "name": "cow"}]
[{"supercategory": "animal", "id": 22, "name": "elephant"}]
[{"supercategory": "animal", "id": 23, "name": "bear"}]
[{"supercategory": "animal", "id": 24, "name": "zebra"}]
[{"supercategory": "animal", "id": 25, "name": "giraffe"}]
[{"supercategory": "accessory", "id": 27, "name": "backpack"}]
[{"supercategory": "accessory", "id": 28, "name": "umbrella"}]
[{"supercategory": "accessory", "id": 31, "name": "handbag"}]
[{"supercategory": "accessory", "id": 32, "name": "tie"}]
[{"supercategory": "accessory", "id": 33, "name": "suitcase"}]
[{"supercategory": "sports", "id": 34, "name": "frisbee"}]
[{"supercategory": "sports", "id": 35, "name": "skis"}]
[{"supercategory": "sports", "id": 36, "name": "snowboard"}]
[{"supercategory": "sports", "id": 37, "name": "sports ball"}]
[{"supercategory": "sports", "id": 38, "name": "kite"}]
[{"supercategory": "sports", "id": 39, "name": "baseball bat"}]
[{"supercategory": "sports", "id": 40, "name": "baseball glove"}]
[{"supercategory": "sports", "id": 41, "name": "skateboard"}]
[{"supercategory": "sports", "id": 42, "name": "surfboard"}]
[{"supercategory": "sports", "id": 43, "name": "tennis racket"}]
[{"supercategory": "kitchen", "id": 44, "name": "bottle"}]
[{"supercategory": "kitchen", "id": 46, "name": "wine glass"}]
[{"supercategory": "kitchen", "id": 47, "name": "cup"}]
[{"supercategory": "kitchen", "id": 48, "name": "fork"}]
[{"supercategory": "kitchen", "id": 49, "name": "knife"}]
[{"supercategory": "kitchen", "id": 50, "name": "spoon"}]
[{"supercategory": "kitchen", "id": 51, "name": "bowl"}]
[{"supercategory": "food", "id": 52, "name": "banana"}]
[{"supercategory": "food", "id": 53, "name": "apple"}]
[{"supercategory": "food", "id": 54, "name": "sandwich"}]
[{"supercategory": "food", "id": 55, "name": "orange"}]
[{"supercategory": "food", "id": 56, "name": "broccoli"}]
[{"supercategory": "food", "id": 57, "name": "carrot"}]
[{"supercategory": "food", "id": 58, "name": "hot dog"}]
[{"supercategory": "food", "id": 59, "name": "pizza"}]
[{"supercategory": "food", "id": 60, "name": "donut"}]
[{"supercategory": "food", "id": 61, "name": "cake"}]
[{"supercategory": "furniture", "id": 62, "name": "chair"}]
[{"supercategory": "furniture", "id": 63, "name": "couch"}]
[{"supercategory": "furniture", "id": 64, "name": "potted plant"}]
[{"supercategory": "furniture", "id": 65, "name": "bed"}]
[{"supercategory": "furniture", "id": 67, "name": "dining table"}]
[{"supercategory": "furniture", "id": 70, "name": "toilet"}]
[{"supercategory": "electronic", "id": 72, "name": "tv"}]
[{"supercategory": "electronic", "id": 73, "name": "laptop"}]
[{"supercategory": "electronic", "id": 74, "name": "mouse"}]
[{"supercategory": "electronic", "id": 75, "name": "remote"}]
[{"supercategory": "electronic", "id": 76, "name": "keyboard"}]
[{"supercategory": "electronic", "id": 77, "name": "cell phone"}]
[{"supercategory": "appliance", "id": 78, "name": "microwave"}]
[{"supercategory": "appliance", "id": 79, "name": "oven"}]
[{"supercategory": "appliance", "id": 80, "name": "toaster"}]
[{"supercategory": "appliance", "id": 81, "name": "sink"}]
[{"supercategory": "appliance", "id": 82, "name": "refrigerator"}]
[{"supercategory": "indoor", "id": 84, "name": "book"}]
[{"supercategory": "indoor", "id": 85, "name": "clock"}]
[{"supercategory": "indoor", "id": 86, "name": "vase"}]
[{"supercategory": "indoor", "id": 87, "name": "scissors"}]
[{"supercategory": "indoor", "id": 88, "name": "teddy bear"}]
[{"supercategory": "indoor", "id": 89, "name": "hair drier"}]
[{"supercategory": "indoor", "id": 90, "name": "toothbrush"}]
```

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-4. 이미지에 대한 모든 주석 출력

```
# 주석 불러오기
anns = coco.loadAnns(ann_ids)

# 이미지 불러오기
image_path = coco.loadImgs(imgId)[0]['file_name']
image = plt.imread(os.path.join(dataDir, dataType, image_path))
plt.imshow(image)

# 주석 표시
coco.showAnns(anns, draw_bbox=True)

# 카테고리 출력
category_names = [coco.loadCats(ann['category_id'])[0]['name'] for ann in anns]
for i, ann in enumerate(anns):
    plt.text(ann['bbox'][0], ann['bbox'][1] - 5, category_names[i], color='blue')

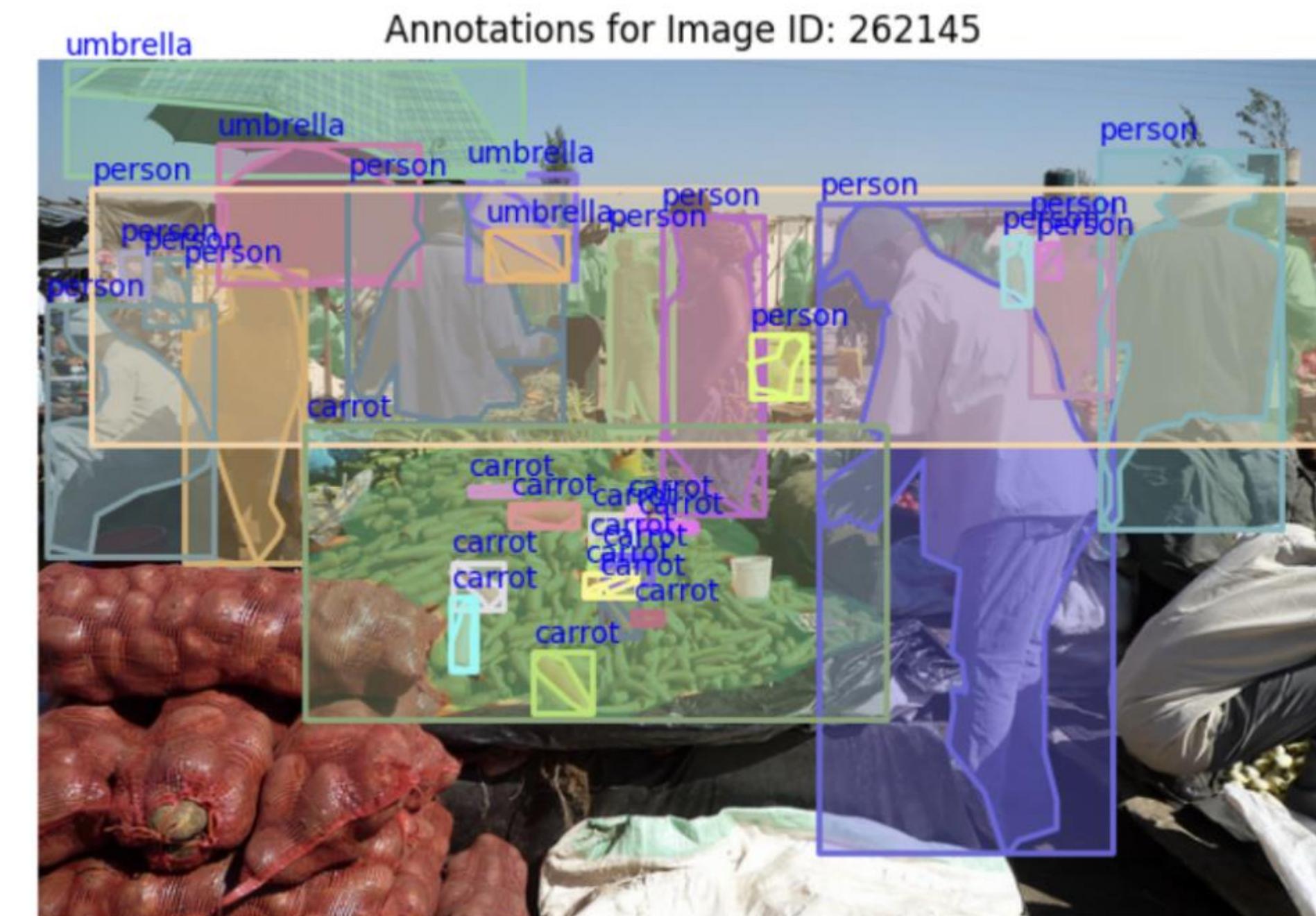
plt.axis('off')
plt.title('Annotations for Image ID: {}'.format(imgId))
plt.tight_layout()
plt.show()
```

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-4. 이미지에 대한 모든 주석 출력



03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-5. 특정 조건의 이미지와 주석 출력

```
# 클래스 지정
cat_ids = coco.getCatIds(catNms=['person'])

# 이미지 불러오기
img_ids = coco.getImgIds(catIds=cat_ids)

# 이미지 정보 불러오기
images = coco.loadImgs(img_ids)
print(len(images))
```

64115

```
anns = coco.loadAnns(ann_ids)

image_path = coco.loadImgs(imgId)[0]['file_name']
image = plt.imread(os.path.join(dataDir, dataType, image_path))
plt.imshow(image)

person_anns = [ann for ann in anns if ann['category_id'] == cat_ids[0]]
coco.showAnns(person_anns, draw_bbox=True)

for ann in person_anns:
    plt.text(ann['bbox'][0], ann['bbox'][1] - 5, f"person - {ann['id']}", color='blue')

plt.axis('off')
plt.title('Annotations for Image ID: {}'.format(imgId))
plt.tight_layout()
plt.show()
```

03

Annotation 시각화

COCO dataset의 주석(annotation)
파일을 시각화하는 과정

3-5. 특정 조건의 이미지와 주석 출력

Annotations for Image ID: 262145



04

데이터셋 전처리

04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

4-1. 데이터셋 로드 및 준비

```
from tensorflow import keras

# annotation 파일의 경로 불러오기
ANNOTATION_FILE_TRAIN = '/content/annotations/instances_train2017.json'
ANNOTATION_FILE_VAL = '/content/annotations/instances_val2017.json'

# 클래스 지정
classes = ['person']

# train set의 API 초기화 및 카테고리 정보 로드
coco_train = COCO(ANNOTATION_FILE_TRAIN)
catIds_train = coco_train.getCatIds(catNms=classes)
imgIds_train = coco_train.getImgIds(catIds=catIds_train)
imgDict_train = coco_train.loadImgs(imgIds_train)

# val set의 API 초기화 및 카테고리 정보 로드
coco_val = COCO(ANNOTATION_FILE_VAL)
catIds_val = coco_val.getCatIds(catNms=classes)
imgIds_val = coco_val.getImgIds(catIds=catIds_val)
imgDict_val = coco_val.loadImgs(imgIds_val)

# 이미지 및 카테고리 수 출력
print(len(imgIds_train), len(catIds_train))
print(len(imgIds_val), len(catIds_val))

loading annotations into memory...
Done (t=44.93s)
creating index...
index created!
loading annotations into memory...
Done (t=1.20s)
creating index...
index created!
64115 1
2693 1
```

04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

4-2. 마스크 생성

```
# 생성할 훈련용 마스크를 저장할 디렉토리 생성
!mkdir mask_train2017

# count 변수 초기화
count = 0

# 훈련 이미지에 대한 마스크 생성
for ID in imgIds_train:

    # 마스크의 파일 경로 설정
    file_path = "/content/mask_train2017/train2017_{0:012d}.jpg".format(ID)

    # 훈련 세트에서 무작위 이미지 ID 가져오기
    sampleImgIds = coco_train.getImgIds(imgIds=[ID])
    sampleImgDict = coco_train.loadImgs(sampleImgIds[np.random.randint(0, len(sampleImgIds))])[0]

    # 이미지에 대한 주석 ID 및 주석 가져오기
    annIds = coco_train.getAnnIds(imgIds=sampleImgDict['id'], catIds=catIds_train, iscrowd=0)
    anns = coco_train.loadAnns(annIds)

    # 개별 인스턴스 마스크를 결합하여 마스크 생성
    mask = coco_train.annToMask(anns[0])
    for i in range(len(anns)):
        mask = mask | coco_train.annToMask(anns[i])

    # 마스크를 이미지로 변환하고 저장
    mask = Image.fromarray(mask * 255, mode="L")
    mask.save(file_path)

    count = count + 1
```

04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

4-3. 전처리 및 generator 생성

```
class CustomDataGenerator(keras.utils.Sequence):
    def __init__(self, images_path, masks_path, batch_size):
        self.images_path = images_path
        self.masks_path = masks_path
        self.batch_size = batch_size
        self.image_filenames = self.get_matching_filenames()
        self.mask_filenames = self.get_matching_filenames()

    def get_matching_filenames(self):
        image_files = set([os.path.splitext(filename)[0] for filename in os.listdir(self.images_path)])
        mask_files = set([os.path.splitext(filename[len('train2017_'):])[0] for filename in os.listdir(self.masks_path)])
        matching_files = list(image_files.intersection(mask_files))
        return matching_files

    def __len__(self):
        return int(np.ceil(len(self.image_filenames) / self.batch_size))
```

** 전체 데이터셋을 일정한 크기의 batch로 나누어 학습

- ▶ def Init : 클래스의 인스턴스 초기화
- ▶ def Get_matching_filenames : 이미지/마스크 파일 매칭
- ▶ def __len__ : 생성 가능한 배치수 계산

04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

4-3. 전처리 및 generator 생성

```

def __getitem__(self, idx):
    batch_filenames = self.image_filenames[idx * self.batch_size:(idx + 1) * self.batch_size]

    batch_images = []
    batch_masks = []

    for filename in batch_filenames:
        # 경로 설정
        image_path = os.path.join(self.images_path, filename + '.jpg')
        mask_path = os.path.join(self.masks_path, 'train2017_' + filename + '.jpg')

        image = Image.open(image_path)
        mask = Image.open(mask_path)

        # 이미지와 마스크의 크기가 동일한지 확인
        if image.size != mask.size:
            raise ValueError(f"Incompatible dimensions for image {image_path} and mask {mask_path}")

        # 이미지와 마스크의 크기를 128x128로 조정
        image = image.resize((128, 128))
        mask = mask.resize((128, 128))

        # 이미지와 마스크를 배열로 변환
        preprocessed_image = np.array(image)
        preprocessed_mask = np.array(mask)

        # 이미지 형태 확인
        if len(preprocessed_image.shape) == 3 and preprocessed_image.shape == (128, 128, 3):
            # 정규화
            preprocessed_image = preprocessed_image / 255.0
            preprocessed_mask = preprocessed_mask / 255.0

        # 전처리된 이미지와 마스크를 배치에 추가
        batch_images.append(preprocessed_image)
        batch_masks.append(preprocessed_mask)

    # 배치 이미지와 마스크를 넘파이 배열로 변환하고 반환
    return np.array(batch_images), np.array(batch_masks)

```

- ▶ def __getitem__: 주어진 인덱스에 해당하는 배치 생성 및 반환
- ▶ 전처리 진행 : 리사이징 및 정규화

04

데이터셋 전처리

COCO dataset의 이미지 파일을 전처리하는 과정

4-4. 전처리 결과 확인

04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

4-5. 전처리 전후 이미지 비교

원본 이미지

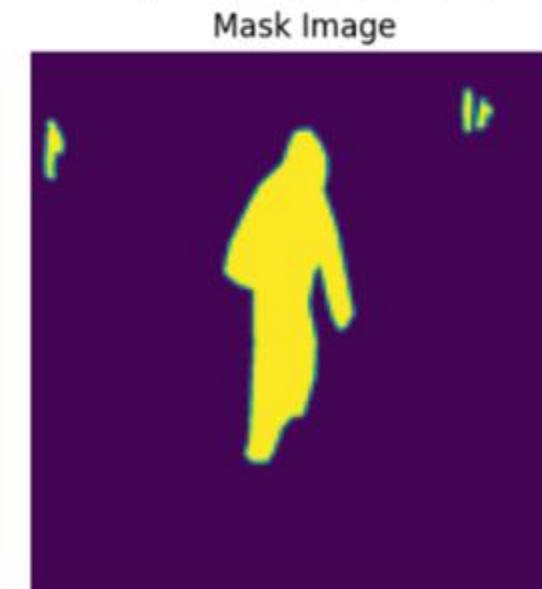
```
original_image = train_generator.image_filenames[0] + '.jpg'  
original_image_path = '/content/train2017/{}'.format(original_image)  
image = Image.open(original_image_path)  
  
print('Image ID:', train_generator.image_filenames[0] + '.jpg')  
print('Main Image Shape:', image.size)  
  
Image.open(original_image_path)
```



전처리된 이미지/마스크

```
batch_images, batch_masks = train_generator[0]  
  
preprocessed_image = batch_images[0]  
preprocessed_mask = batch_masks[0]  
  
plt.subplot(1, 2, 1)  
plt.imshow(preprocessed_mask)  
plt.title('Mask Image')  
plt.axis('off')  
  
plt.subplot(1, 2, 2)  
plt.imshow(preprocessed_image)  
plt.title(' Main Image')  
plt.axis('off')  
  
print('Image ID:', train_generator.image_filenames[0]+'.jpg')  
print('Mask Shape:', preprocessed_mask.shape)  
print('Main Image Shape:', preprocessed_image.shape)  
  
plt.tight_layout()  
plt.show()
```

Image ID: 000000050412.jpg
Mask Shape: (128, 128)
Main Image Shape: (128, 128, 3)



04

데이터셋 전처리

COCO dataset의 이미지 파일을
전처리하는 과정

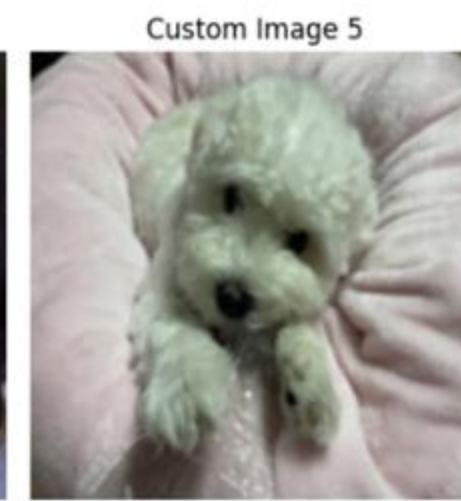
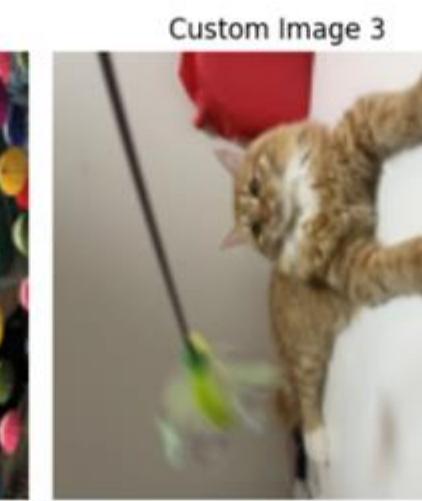
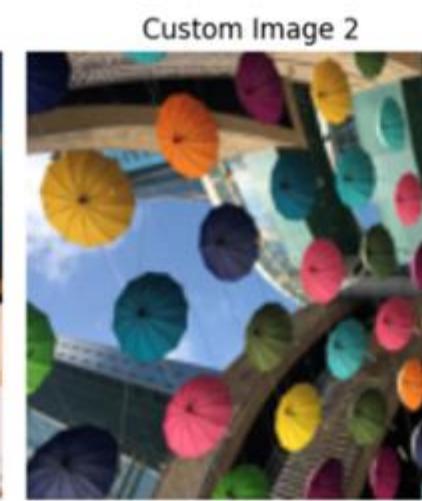
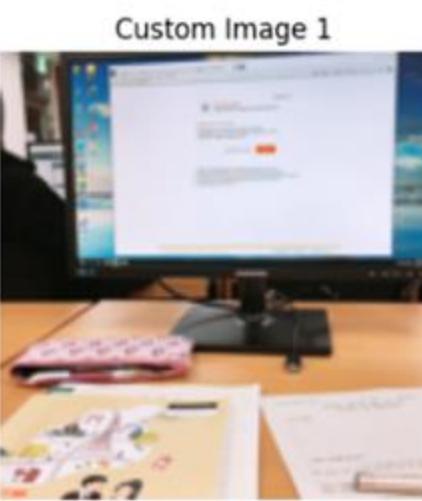
4-6. 직접 찍은 이미지 전처리

```
# Custom 이미지 시각화
def visualize_custom_images(custom_images):
    num_images = len(custom_images)
    fig, axs = plt.subplots(1, num_images, figsize=(15, 5))
    for i, image in enumerate(custom_images):
        axs[i].imshow(image)
        axs[i].axis('off')
        axs[i].set_title(f'Custom Image {i+1}')
    plt.tight_layout()
    plt.show()

# 경로 설정
custom_images_path = '/content/my_images'
custom_image_names = ['custom_1.jpg', 'custom_2.jpg', 'custom_3.jpg', 'custom_4.jpg', 'custom_5.jpg']
custom_image_paths = [os.path.join(custom_images_path, name) for name in custom_image_names]

# Custom 이미지 전처리
custom_images = process_custom_images(custom_image_paths)

# Custom 이미지 시각화
visualize_custom_images(custom_images)
```



05

Instance Segmentation Model

05

Instance Segmentation Model

인스턴스 분할을 위한 모델 선정

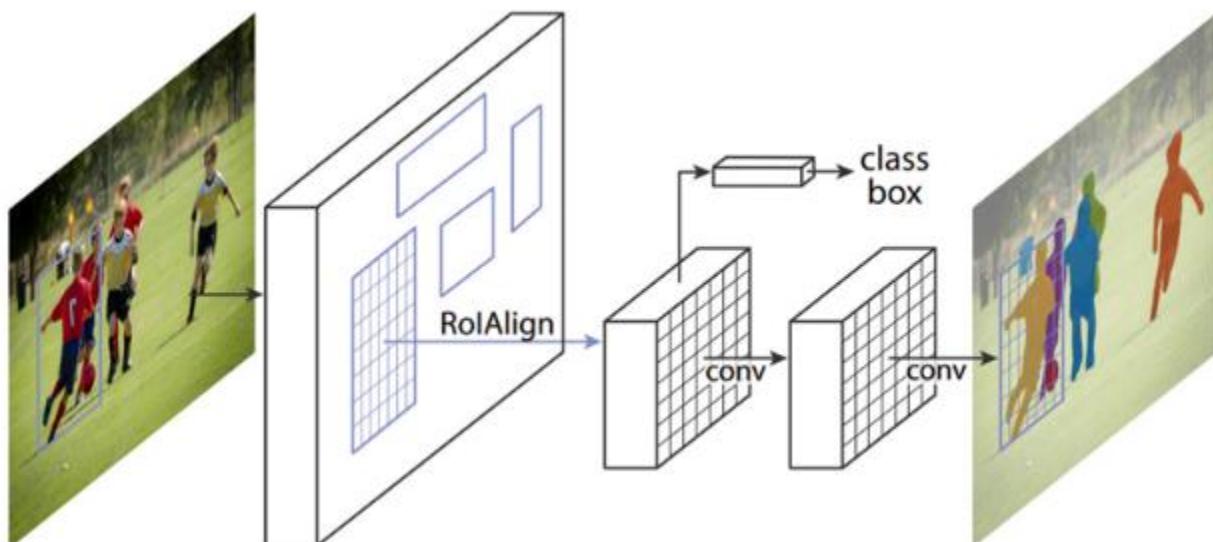
MMDetection(라이브러리)

Object Detection
/ Instance Segmentation 모델들

Architectures			
Object Detection	Instance Segmentation	Panoptic Segmentation	Other
<ul style="list-style-type: none">Fast R-CNN (ICCV'2015)Faster R-CNN (NeurIPS'2015)RPN (NeurIPS'2015)SSD (ECCV'2016)RetinaNet (ICCV'2017)Cascade R-CNN (CVPR'2018)YOLOv3 (ArXiv'2018)CornerNet (ECCV'2018)Grid R-CNN (CVPR'2019)Guided Anchoring (CVPR'2019)FSAF (CVPR'2019)CenterNet (CVPR'2019)Libra R-CNN (CVPR'2019)TridentNet (ICCV'2019)FCOS (ICCV'2019)RepPoints (ICCV'2019)FreeAnchor (NeurIPS'2019)CascadeRPN (NeurIPS'2019)Foveabox (TIP'2020)Double-Head R-CNN (CVPR'2020)ATSS (CVPR'2020)NAS-FCOS (CVPR'2020)	<ul style="list-style-type: none">Mask R-CNN (ICCV'2017)Cascade Mask R-CNN (CVPR'2018)Mask Scoring R-CNN (CVPR'2019)Hybrid Task Cascade (CVPR'2019)YOLACT (ICCV'2019)InstaBoost (ICCV'2019)SOLO (ECCV'2020)PointRend (CVPR'2020)DetectoRS (ArXiv'2020)SOLOv2 (NeurIPS'2020)SCNet (AAAI'2021)QueryInst (ICCV'2021)Mask2Former (ArXiv'2021)CondInst (ECCV'2020)SparseInst (CVPR'2022)RTMDet (ArXiv'2022)BoxInst (CVPR'2021)ConvNeXt-V2 (Arxiv'2023)	<ul style="list-style-type: none">Panoptic FPN (CVPR'2019)MaskFormer (NeurIPS'2021)Mask2Former (ArXiv'2021)XDecoder (CVPR'2023)	<ul style="list-style-type: none">Contrastive Learning<ul style="list-style-type: none">SwAV (NeurIPS'2020)MoCo (CVPR'2020)MoCov2 (ArXiv'2020)Distillation<ul style="list-style-type: none">Localization Distillation (CVPR'2022)Label Assignment Distillation (WACV'2022)Semi-Supervised Object Detection<ul style="list-style-type: none">Soft Teacher (ICCV'2021)

05 | Instance Segmentation Model

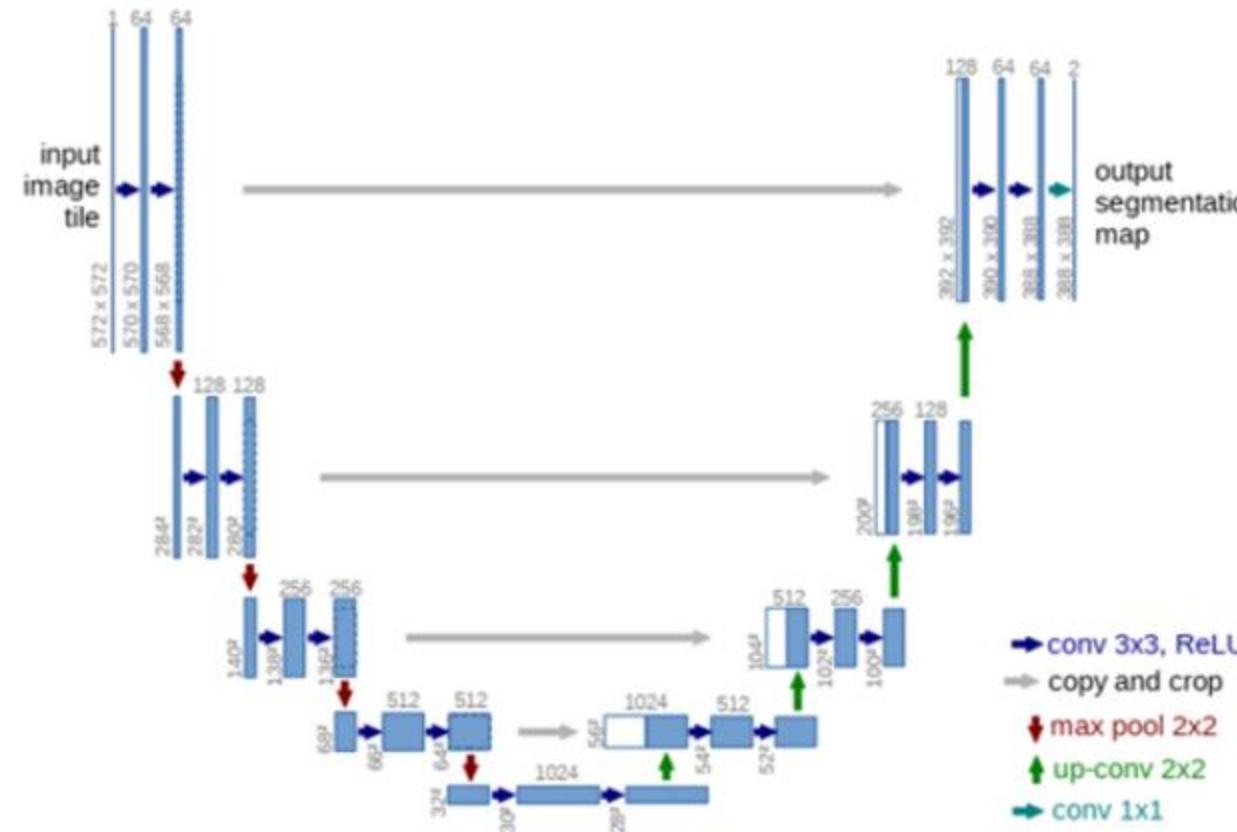
MASK R-CNN



- ▶ Faster R-CNN 기반
- ▶ 객체 감지와 정교한 픽셀 단위 마스크 생성을 동시에 수행
- ▶ 정교한 인스턴스 분할 가능
- ▶ 의료 영상 분석, 자율 주행 자동차, 로봇 비전 등 다양한 분야 활용
- ▶ 특히 복잡한 환경에서의 정확한 객체 인식 강점

05 | Instance Segmentation Model

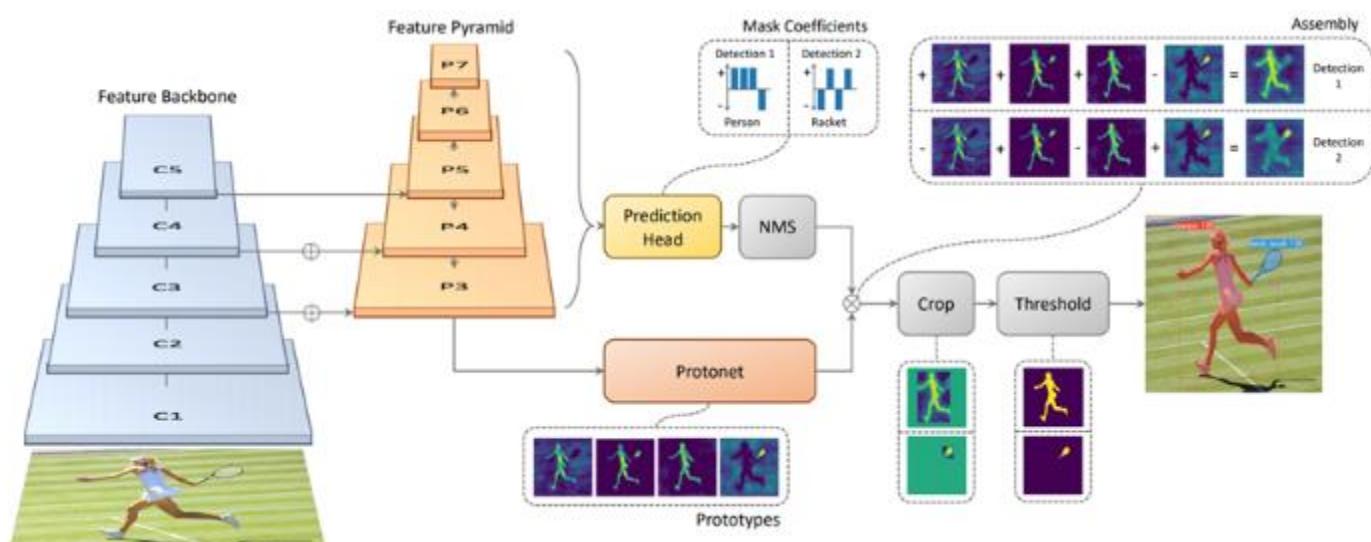
U-Net



- ▶ 영상 세그멘테이션 분야의 혁신적인 모델
- ▶ 특히 의료 영상 분석 등에 널리 활용되며, 정확성과 효율성이 뛰어남
- ▶ 인코더- 디코더 구조와 skip 연결을 통해 저해상/ 고해상도 특징을 효과적으로 융합
- ▶ 정밀한 세그멘테이션 결과 생성

05 | Instance Segmentation Model

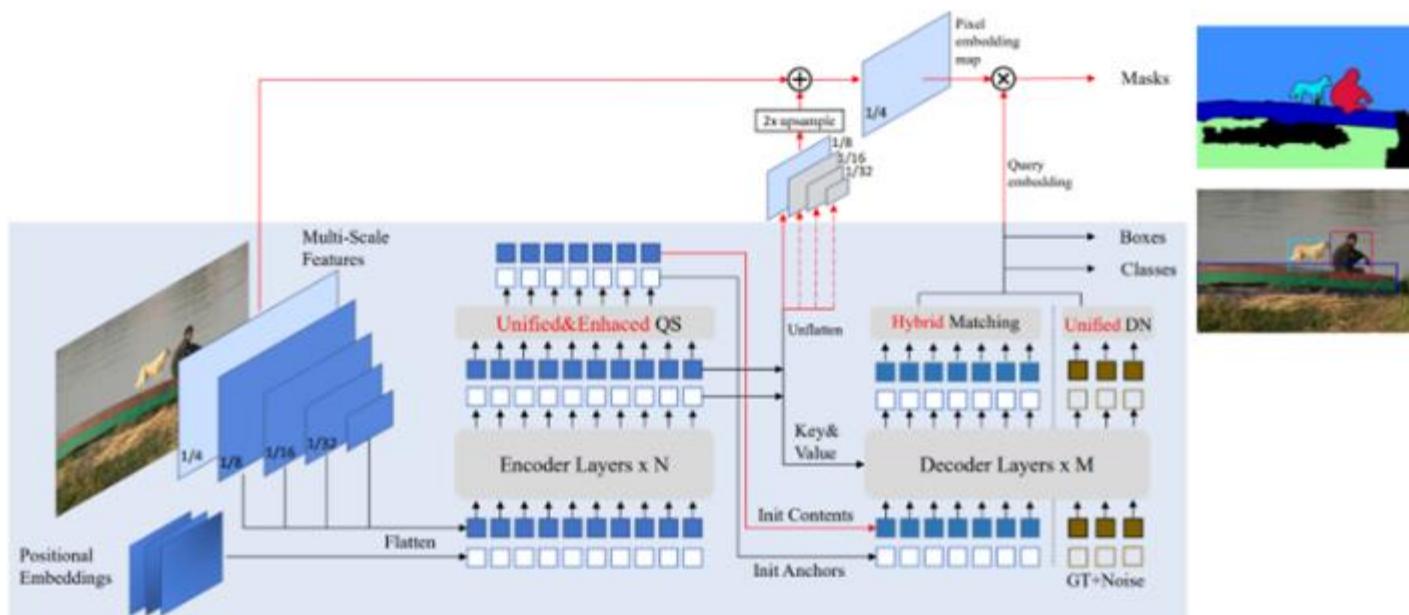
YOLACT



- ▶ 실시간 인스턴스 세그멘테이션의 혁신적인 모델
- ▶ 기존 모델에 비해 정확도와 속도가 매우 뛰어남
- ▶ 객체 탐지 / 마스크 생성을 병렬로 수행
- ▶ 고속 처리 가능하며, 복잡한 환경도 안정적인 성능
- ▶ 자율 주행, 영상 감시 등 실시간 인식이 필요한 분야 적용

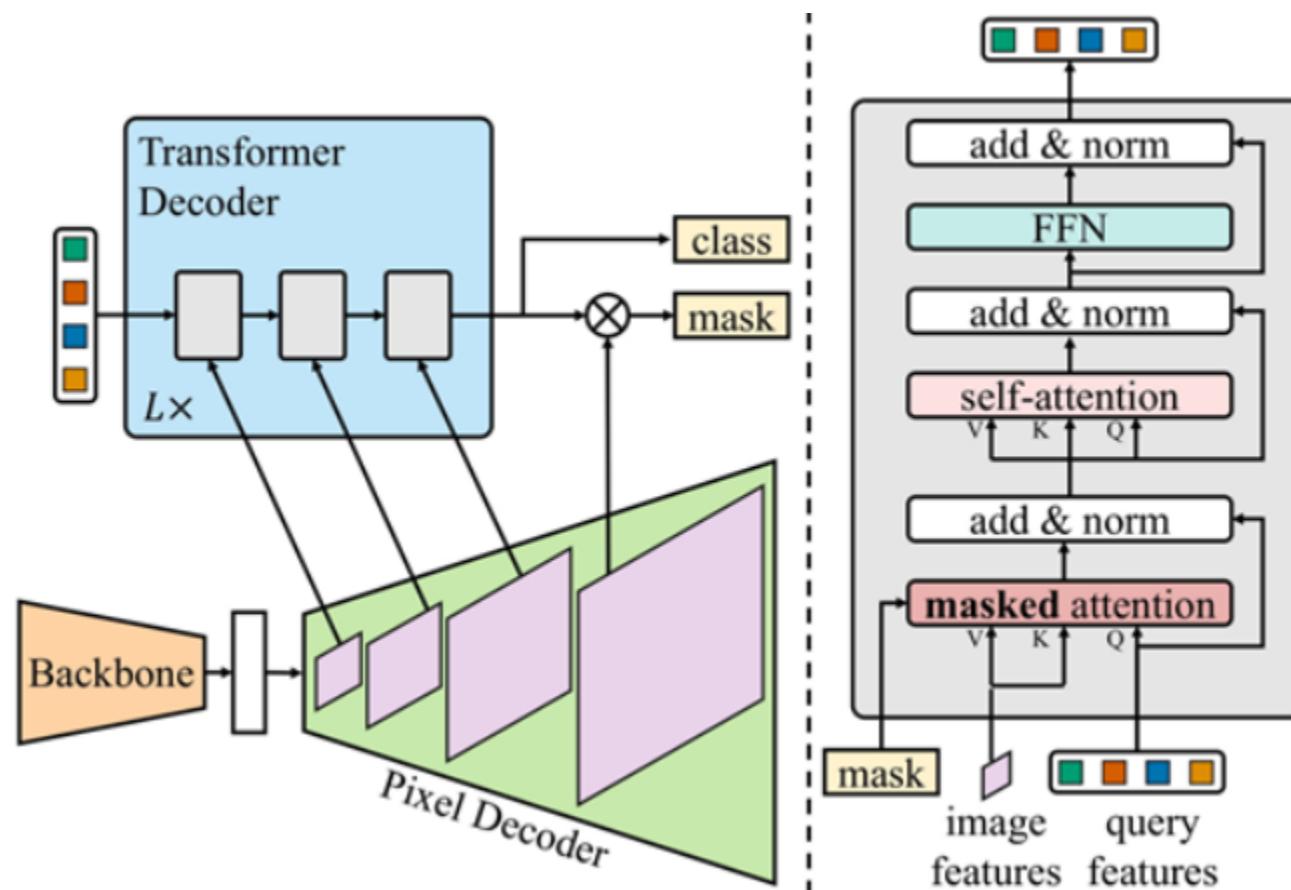
05 | Instance Segmentation Model

MASKDINO



- ▶ DINO 모델 기반
- ▶ 객체 감지와 정교한 픽셀 단위 마스크 생성 동시 수행
- ▶ 복잡한 환경에서도 정확한 인스턴스 분할 가능
- ▶ 광범위한 응용 분야 활용

05 | Instance Segmentation Model



Mask2former

- ▶ Transformer 기반 아키텍처
- ▶ 다양한 유형의 이미지 데이터에 유연하게 적용
- ▶ 기존 모델 대비 정확도와 효율성이 향상
- ▶ 광범위한 응용 분야 활용

05 | Instance Segmentation Model - 모델 선정

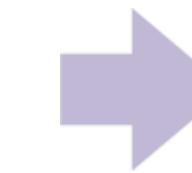
활용할 데이터셋



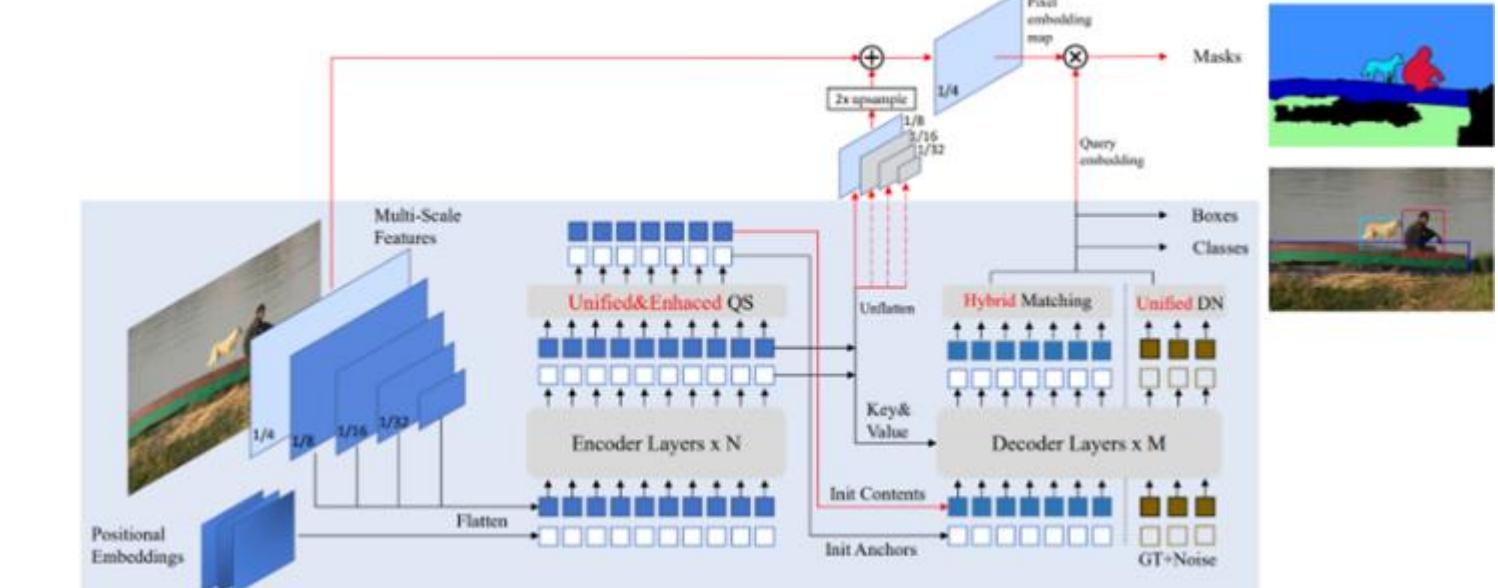
- ✓ 객체 감지와 정교한 픽셀 단위 마스크 생성 동시 수행
- ✓ 복잡한 환경에서도 정확한 인스턴스 분할 가능
- ✓ 광범위한 응용 분야 활용
- ✓ 설정 및 사용의 용이성

05 | Instance Segmentation Model - 모델 선정

활용할 데이터셋



MASKDINO



- ✓ 객체 감지와 정교한 픽셀 단위 마스크 생성 동시 수행
- ✓ 복잡한 환경에서도 정확한 인스턴스 분할 가능
- ✓ 광범위한 응용 분야 활용
- ✓ 설정 및 사용의 용이성

05 | Instance Segmentation Model - 모델 선정



- ✗ 모델 컴파일 오류
- ✗ 학습 오류
- ✗ 관련 참고자료 부족
- ✗ 코랩 메모리 부족
- ✗ 성능 확인 불가
- ✗ 오류 수정의 어려움

"모델 교체"

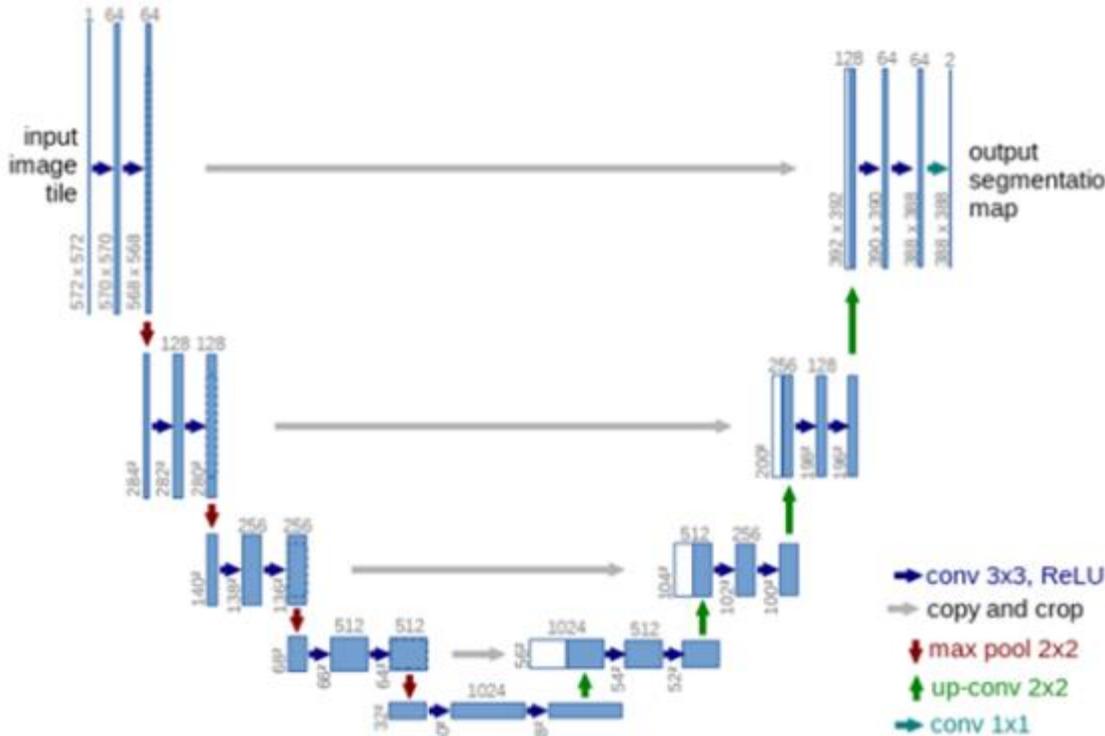
05 | Instance Segmentation Model - 모델 선정



이후 MaskRCNN / YOLOv7 / mask2former 등의 모델들을 시도하였으나 여러 오류들로 인해 모델 학습이 어려움

05 | Instance Segmentation Model - 모델 선정

U-Net



- ▶ 코코 데이터셋 : 다양한 객체와 복잡한 배경이 포함된 이미지로 구성, 정교한 이미지 분할 기술이 필요
→ U-NET의 우수한 분할 성능(정밀한 세그멘테이션 결과 생성 가능)
- ▶ U-NET은 학습 효율성이 뛰어나며, 상대적으로 적은 데이터로도 우수한 성능 코코 데이터셋의 방대한 크기와 복잡성을 고려할 때 매우 중요한 장점

05 | Model 학습

5-1. U-Net 모델 정의

```

import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.utils import Sequence

def unet_model(input_size=(128, 128, 3)):
    inputs = layers.Input(input_size)
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
    p1 = layers.MaxPooling2D((2, 2))(c1)

    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
    c2 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
    p2 = layers.MaxPooling2D((2, 2))(c2)

    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
    c3 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
    p3 = layers.MaxPooling2D((2, 2))(c3)

    c4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(p3)
    c4 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
    p4 = layers.MaxPooling2D((2, 2))(c4)

    c5 = layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(p4)
    c5 = layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(c5)

```

```

u6 = layers.Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = layers.concatenate([u6, c4])
c6 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(u6)
c6 = layers.Conv2D(512, (3, 3), activation='relu', padding='same')(c6)

u7 = layers.Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c6)
u7 = layers.concatenate([u7, c3])
c7 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(u7)
c7 = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c7)

u8 = layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c7)
u8 = layers.concatenate([u8, c2])
c8 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u8)
c8 = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c8)

u9 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = layers.concatenate([u9, c1])
c9 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u9)
c9 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c9)

outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)

model = models.Model(inputs=[inputs], outputs=[outputs])
return model

```

05 | Model 학습

5-1. U-Net 모델 정의

01 Input Layer

- (128, 128, 3) 크기의 RGB 이미지

02 Encoder

- 4단계의 Convolution과 MaxPooling layer로 구성
- 각 단계에서 필터 수: 64, 128, 256, 512
- 활성화 함수는 'relu', 패딩은 'same'으로 설정

03 Bottleneck

- 네트워크의 가장 좁은 부분으로, 전체 네트워크의 정보 요약본 포함
- 두 개의 Conv2D 레이어로 각각 1024개의 필터를 사용하여 가장 깊은 특성을 추출

04 Decoder

- Conv2DTranspose 레이어를 사용하여 특성 맵의 크기를 두 배로 증가
- 각 단계에서 필터 수는 512, 256, 128, 64
- 각 Decoder 블록은 순서대로 Encoder의 c4, c3, c2, c1과 결합

05 Output Layer

- 1개의 필터를 가진 Conv2D 레이어로 구성
- 1x1 커널과 sigmoid 활성화 함수를 사용하여 각 픽셀에 대해 클래스 확률 출력

05 | Model 학습

5-2. U-Net 모델 학습

```

import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint

# 모델 컴파일
model = unet_model()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# 체크포인트 콜백 정의
checkpoint_callback = ModelCheckpoint(
    filepath='/content/drive/MyDrive/데이터/checkpoint/Adam_best_model.h5', # 모델 파일을 저장할 경로
    monitor='val_accuracy', # 성능을 모니터링할 지표 (val_loss 또는 val_accuracy 등)
    save_best_only=True, # 가장 좋은 모델만 저장
    mode='max', # accuracy 모니터링 지표가 최대일 때 저장
    save_weights_only=True, # 가중치만 저장하려면 True, 전체 모델을 저장하려면 False
    verbose=1 # 로그 출력 설정
)

# 모델 학습
model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=10,
    callbacks=[checkpoint_callback]
)

```

- ✓ 손실 함수: binary crossentropy
- ✓ 평가 지표: accuracy
- ✓ ModelCheckpoint 콜백을 설정하여 모델의 학습 과정 중 성능이 가장 좋은 모델 저장
- ✓ 성능 지표: validation accuracy
- ✓ 학습 과정은 verbose=1로 설정하여 각 에포크의 결과를 확인할 수 있도록 함

05 | Model 학습

5-3. Optimizer 비교

Adagrad

학습률을 각 매개변수에 대해 적응적으로 조정하여 맞춤형 learning rate를 만듦

RMSProp

과거의 기울기를 적게 반영하고 최신의 기울기를 많이 반영하여 학습을 안정화하고 빠르게 수렴하도록 하는 AdaGrad의 개선된 최적화 알고리즘

Adam

RMSProp + Momentum을 결합한 방법으로, 1차 모멘트(기울기 평균)와 2차 모멘트(기울기 제곱 평균)를 동시에 추정하여 학습률을 적응적으로 조정

AdamW

Adam의 변형으로, L2 정규화 대신 가중치 감소(Weight Decay)를 사용하여 과적합을 방지

05 | Model 학습

5-3. Optimizer 비교

	accuracy	loss
Adagrad	0.8181	0.4569
RMSProp	0.8183	0.4042
Adam	0.8179	0.3970
AdamW	0.8212	0.3947

05 | Model 학습

5-4. Custom 이미지 예측

```
[ ] import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow import keras

# CustomDataGenerator 수정
class CustomDataGenerator(keras.utils.Sequence):
    def __init__(self, images_path, masks_path, batch_size):
        self.images_path = images_path
        self.masks_path = masks_path
        self.batch_size = batch_size
        self.image_filenames = self.get_matching_filenames()
        self.mask_filenames = self.get_matching_filenames()

    def get_matching_filenames(self):
        image_files = set([os.path.splitext(filename)[0] for filename in os.listdir(self.images_path)])
        mask_files = set([os.path.splitext(filename)[0] for filename in os.listdir(self.masks_path)])
        matching_files = list(image_files.intersection(mask_files))
        return matching_files

    def __len__(self):
        return int(np.ceil(len(self.image_filenames) / self.batch_size))

    def __getitem__(self, idx):
        batch_filenames = self.image_filenames[idx * self.batch_size:(idx + 1) * self.batch_size]

        batch_images = []
        batch_masks = []

        for filename in batch_filenames:
            image_path = os.path.join(self.images_path, filename + '.jpg')
            mask_path = os.path.join(self.masks_path, 'train2017_' + filename + '.jpg')

            image = Image.open(image_path)
            mask = Image.open(mask_path)

            if image.size != mask.size:
                raise ValueError(f"Incompatible dimensions for image [{image_path}] and mask [{mask_path}]")

            image = image.resize((128, 128))
            mask = mask.resize((128, 128))

            preprocessed_image = np.array(image)
            preprocessed_mask = np.array(mask)

            if len(preprocessed_image.shape) == 3 and preprocessed_image.shape == (128, 128, 3):
                preprocessed_image = preprocessed_image / 255.0
                preprocessed_mask = preprocessed_mask / 255.0

            batch_images.append(preprocessed_image)
            batch_masks.append(preprocessed_mask)

        return np.array(batch_images), np.array(batch_masks)
```

```
# Custom 이미지 전처리
def process_custom_images(custom_image_paths):
    images = []
    for image_path in custom_image_paths:
        image = Image.open(image_path).convert('RGB')
        image = image.resize((128, 128))
        preprocessed_image = np.array(image) / 255.0
        images.append(preprocessed_image)
    return np.array(images)

# Custom 이미지 시각화
def visualize_custom_images(custom_images):
    num_images = len(custom_images)
    fig, axes = plt.subplots(2, num_images, figsize=(15, 10))
    for i, (image, prediction) in enumerate(zip(custom_images, predictions)):
        axes[0, i].imshow(image)
        axes[0, i].axis('off')
        axes[0, i].set_title(f"Custom Image {i+1}")

    # three channels for RGB visualization
    if prediction.shape[-1] == 1:
        prediction_colormap = apply_colormap_to_prediction(prediction)

    # 원본 / 예측 불랜딩
    alpha = 0.5
    blended_image = (alpha * prediction_colormap + (1 - alpha) * image)
    axes[1, i].imshow(blended_image)
    axes[1, i].axis('off')
    axes[1, i].set_title(f"Prediction {i+1}")
    plt.tight_layout()
    plt.show()
```

```
# 예측 결과 시각화
def visualize_predictions(custom_images, predictions):
    num_images = len(custom_images)
    fig, axes = plt.subplots(2, num_images, figsize=(15, 10))
    for i, (image, prediction) in enumerate(zip(custom_images, predictions)):
        axes[0, i].imshow(image)
        axes[0, i].axis('off')
        axes[0, i].set_title(f"Custom Image {i+1}")

    # 원본 / 예측 불랜딩
    alpha = 0.5
    blended_image = (alpha * prediction_colormap + (1 - alpha) * image)
    axes[1, i].imshow(blended_image)
    axes[1, i].axis('off')
    axes[1, i].set_title(f"Prediction {i+1}")
    plt.tight_layout()
    plt.show()
```

```
# 예측 결과 시각화
visualize_predictions(custom_images, predictions)
```

05 | Model 학습

5-4. Custom 이미지 예측

```
[ ] import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from tensorflow import keras

# CustomDataGenerator 수정
class CustomDataGenerator(keras.utils.Sequence):
    def __init__(self, images_path, masks_path, batch_size):
        self.images_path = images_path
        self.masks_path = masks_path
        self.batch_size = batch_size
        self.image_filenames = self.get_matching_filenames()
        self.mask_filenames = self.get_matching_filenames()

    def get_matching_filenames(self):
        image_files = set([os.path.splitext(filename)[0] for filename in os.listdir(self.images_path)])
        mask_files = set([os.path.splitext(filename)[0] for filename in os.listdir(self.masks_path)])
        matching_files = list(image_files.intersection(mask_files))
        return matching_files

    def __len__(self):
        return int(np.ceil(len(self.image_filenames) / self.batch_size))

    def __getitem__(self, idx):
        batch_filenames = self.image_filenames[idx * self.batch_size:(idx + 1) * self.batch_size]

        batch_images = []
        batch_masks = []

        for filename in batch_filenames:
            image_path = os.path.join(self.images_path, filename + '.jpg')
            mask_path = os.path.join(self.masks_path, 'train2017_' + filename + '.jpg')

            image = Image.open(image_path)
            mask = Image.open(mask_path)

            if image.size != mask.size:
                raise ValueError(f"Incompatible dimensions for image {image_path} and mask {mask_path}")

            image = image.resize((128, 128))
            mask = mask.resize((128, 128))

            preprocessed_image = np.array(image)
            preprocessed_mask = np.array(mask)

            if len(preprocessed_image.shape) == 3 and preprocessed_image.shape == (128, 128, 3):
                preprocessed_image = preprocessed_image / 255.0
                preprocessed_mask = preprocessed_mask / 255.0

            batch_images.append(preprocessed_image)
            batch_masks.append(preprocessed_mask)

        return np.array(batch_images), np.array(batch_masks)
```

- ▶ Image file과 mask file의 이름을 추출하여 공통으로 존재하는 파일 이름

목록 반환

- ▶ 전체 데이터셋을 배치 크기로 나눈 값의 올림을 계산하여 총 배치 수 반환
- ▶ 현재 배치에 해당하는 이미지를 불러와서 128×128 크기로 리사이즈 하고,

픽셀 값을 [0, 1] 범위로 정규화

- ▶ 전처리된 이미지와 마스크를 numpy 배열로 변환하여 배치 데이터로 반환

05 | Model 학습

5-4. Custom 이미지 예측

```
# Custom 이미지 전처리
def process_custom_images(custom_image_paths):
    images = []
    for image_path in custom_image_paths:
        image = Image.open(image_path).convert('RGB')
        image = image.resize((128, 128))
        preprocessed_image = np.array(image) / 255.0
        images.append(preprocessed_image)
    return np.array(images)

# Custom 이미지 시각화
def visualize_custom_images(custom_images):
    num_images = len(custom_images)
    fig, axes = plt.subplots(1, num_images, figsize=(15, 5))
    for i, image in enumerate(custom_images):
        axes[i].imshow(image)
        axes[i].axis('off')
        axes[i].set_title(f'Custom Image {i+1}')
    plt.tight_layout()
    plt.show()

# 경로 설정
custom_images_path = '/content/drive/MyDrive/Colab Notebooks/데이터/팀플/custom'
custom_image_names = ['custom_1.jpg', 'custom_2.jpg', 'custom_3.jpg', 'custom_4.jpg', 'custom_5.jpg']
custom_image_paths = [os.path.join(custom_images_path, name) for name in custom_image_names]

# Custom 이미지 전처리
custom_images = process_custom_images(custom_image_paths)

# Custom 이미지 시각화
visualize_custom_images(custom_images)

# 불려온 모델을 사용하여 커스텀 이미지 예측
model = keras.models.load_model('/content/drive/MyDrive/Colab Notebooks/데이터/팀플/unet/AdamN_final_model.h5')
predictions = model.predict(custom_images)

# 클래스별로 색상을 다르게 하는 함수
def apply_colormap_to_prediction(prediction):
    unique_classes = np.unique(prediction)
    num_classes = len(unique_classes)
    colormap = plt.cm.get_cmap('jet', num_classes)

    # 컬러맵 적용을 위해 고유한 클래스 값을 0에서 num_classes-1로 매핑
    mapped_prediction = np.zeros_like(prediction[..., 0])
    for i, cls in enumerate(unique_classes):
        mapped_prediction[prediction[..., 0] == cls] = i

    colored_prediction = colormap(mapped_prediction / (num_classes - 1))
    return colored_prediction[..., :3] # Remove alpha channel
```

- ▶ 이미지를 128×128 크기로 리사이즈하고, 픽셀 값을 [0, 1] 범위로 정규화하여 numpy 배열로 변환
- ▶ 전처리된 이미지를 한 줄로 시각화
- ▶ 저장된 모델을 불러와 전처리된 이미지에 대한 예측 수행
- ▶ 예측 결과에 색상 맵을 적용하여 시각적으로 구분할 수 있도록 함
- ▶ 고유한 클래스 값을 0에서 num_classes-1로 매핑한 후, 컬러맵을 적용하여 RGB 이미지로 변환

05 | Model 학습

5-4. Custom 이미지 예측

```
# 예측 결과 시각화
def visualize_predictions(custom_images, predictions):
    num_images = len(custom_images)
    fig, axs = plt.subplots(2, num_images, figsize=(15, 10))
    for i, (image, prediction) in enumerate(zip(custom_images, predictions)):
        axs[0, i].imshow(image)
        axs[0, i].axis('off')
        axs[0, i].set_title(f'Custom Image {i+1}')

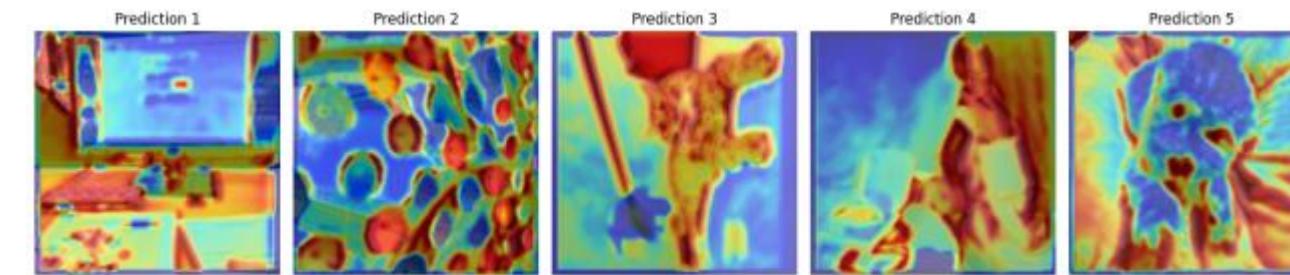
        # three channels for RGB visualization
        if prediction.shape[-1] == 1:
            prediction_colormap = apply_colormap_to_prediction(prediction)

        # 원본 / 예측 블랜딩
        alpha = 0.5
        blended_image = (alpha * prediction_colormap + (1 - alpha) * image)

        axs[1, i].imshow(blended_image)
        axs[1, i].axis('off')
        axs[1, i].set_title(f'Prediction {i+1}')
    plt.tight_layout()
    plt.show()

# 예측 결과 시각화
visualize_predictions(custom_images, predictions)
```

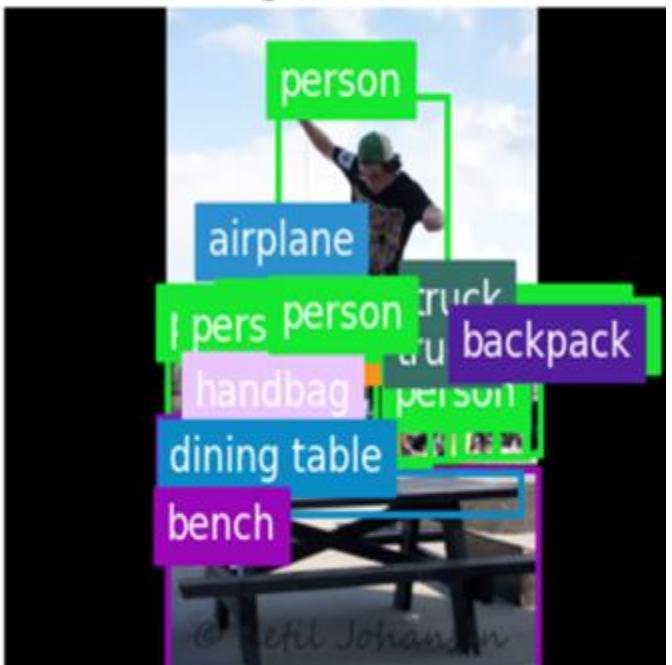
- ▶ 원본 이미지와 예측 결과를 블렌딩하여 하나의 이미지로 시각화
- ▶ 모델의 성능을 직관적으로 확인할 수 있도록 함



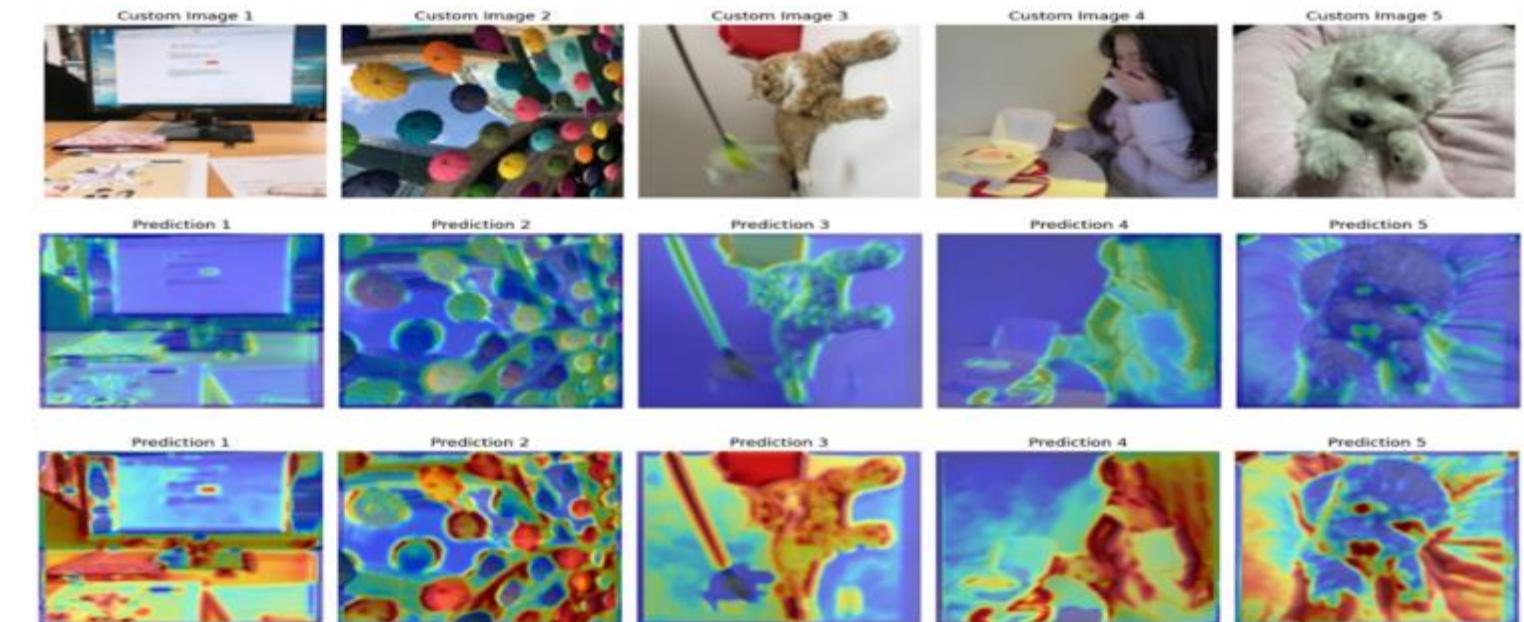
05 | Model 결과 해석

예상했던 결과

Image ID: 262148



실제 결과



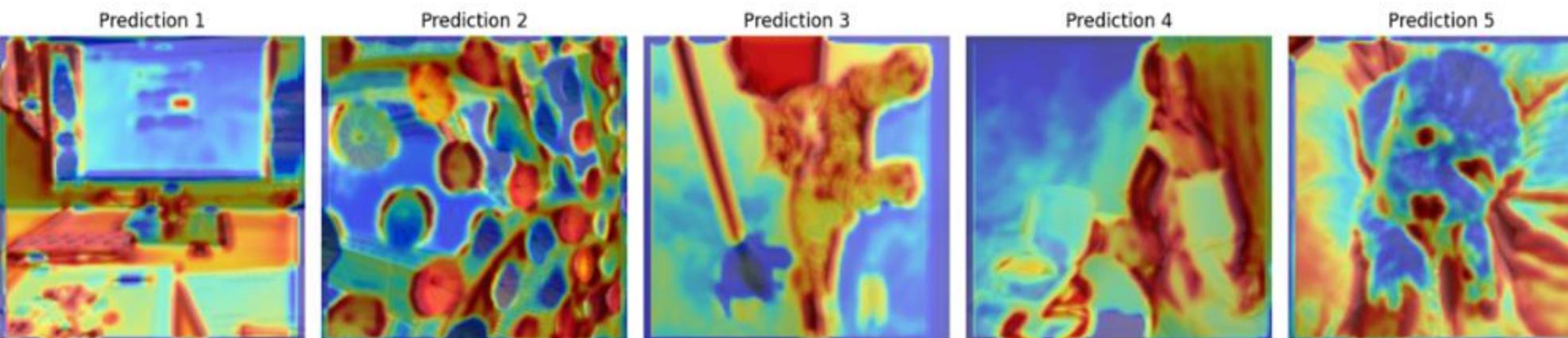
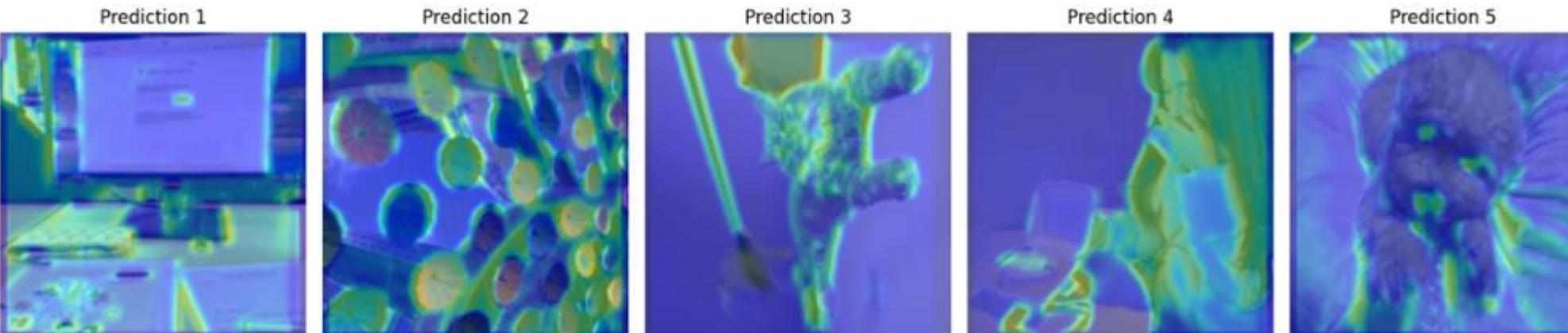
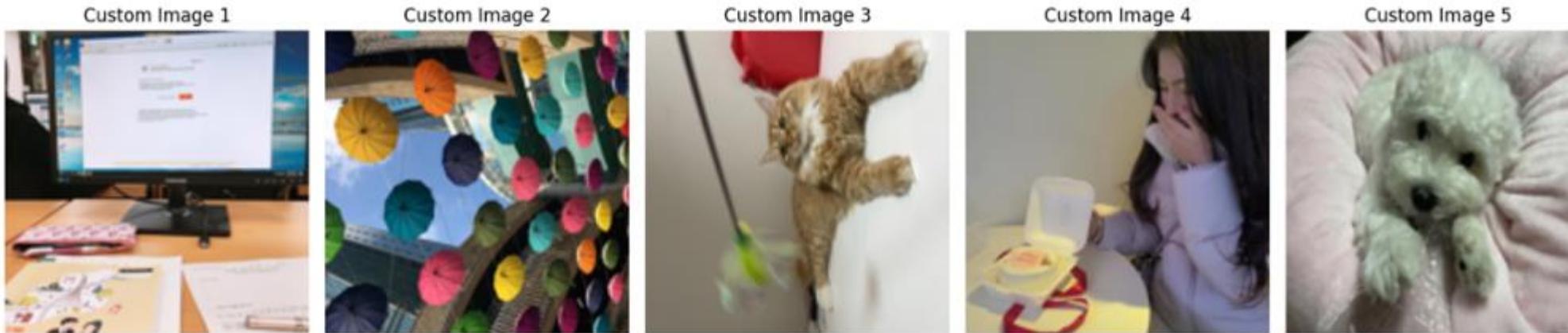
- ▶ Annotation 시각화 X
- ▶ 사람/동물에 비해 물건 / 배경의 인식 성능 저하
- ▶ 복잡하거나 색이 비슷한 경우 인식 성능 저하

- COCO와 달리 주석 파일 X, 새로운 주석 파일 필요
- 모델의 Epoch을 크게 설정하지 못함
- 세세한 인스턴스를 인식할 만한 모델의 성능 X

05 | Model 개선 방안

- ▶ 객체 인식 및 분할 기술 고도화
 - : U-NET 모델의 성능을 더욱 개선하여 정확도와 재현율을 높이고,
다양한 객체에 대한 세밀한 분할이 가능하도록 업그레이드
- ▶ 추가적으로 블러 처리 / 음영 조정 등의 보정이나 사진 편집 기능 추가
- ▶ Epoch 수 늘리기
 - : 구글 코랩 메모리 / 시간 상의 이유로 큰 epoch을 설정하지 못함 → 10 epoch 이상으로 늘려 모델 개선
- ▶ Attention 메커니즘 도입
 - : UNET 구조에 Attention 메커니즘을 적용하여 중요한 특징들에 더 큰 가중치
→ 객체 탐지와 경계 정확도를 향상

05 | Model 활용 방안



비교적 인물 / 동물 잘 인식
→ 사진 / 영상에서 인물 / 동물 인식 및 구별

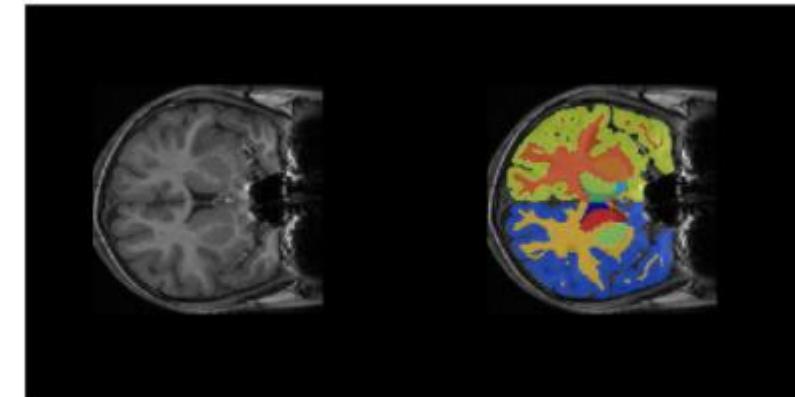
- ▶ 사진 및 비디오 분류
: 시스템에서 사람과 동물을 구별하여 자동으로 카테고리를 나누고 태그 설정

- ▶ 출입, 침입 관리
: 보안 카메라 시스템에서 사람과 동물 구별 알람 발생 / 동물의 출입을 방지

05 | Model 활용 방안

의료 영상 분석

정확한 장기 및 조직 구분으로 병변 탐지, 치료 계획 수립
MASKDINO, Mask2Former



자율 주행 및 로봇 비전

주행 환경의 복잡한 객체를 정확하게 인식하여 안전한 경로 계획
Faster R-CNN, YOLACT

