중간 보고서

Drone 영상에서의 Detection 및 Free-Space Segmentation

제출일	2023. 07. 31	팀명	FlyToTheSky
담당교수	박진선	팀장	김경현
		팀원	유일해, 신민건

목차

1. 요구조건 및 제약 사항 분석에 따른 수정사항

1.1 요구조건

본 과제에서는 야외 적치장 내의 오브젝트와 공간 구분을 자동화하여 효율적인 자원

관리를 목표로 한다. 이에 따른 요구조건으로 드론에서 실시간으로 전송되는 적치장의 공중 시점 영상(Aerial view image)을 분석한다. 사용가능한 공간과 적치된 오브젝트, 장비를 분류하여 적치장의 용적률을 계산하고 적치된 오브젝트를 인스턴스화 하여 출입하는 오브젝트에 대한 정보를 실시간으로 계산한다. 계산된 적치장 내부 정보를 바탕으로 적치장의 효율적인 관리를 위한 사용자 인터페이스를 제공하고자 한다.

1.2제약 사항 분석에 대한 수정사항

과제 진행 중 주요 제약 사항은 1) 비슷한 특징을 가지는 공간에 대한 구분 문제, 2) 데이터 부족, 3) 프로세싱 속도 제약, 4) 인스턴스 식별 문제, 5) 유사한 형태의 레이블 등이 존재한다. 본 과제에서는 이에 대한 대안으로 다음과 같은 방안을 적용하려 한다.

1) 비슷한 특징을 가지는 공간에 대한 구분 문제

적치장은 적치 가능한 공간이 기능적으로 나눠져 있지만, 이를 이미지가 가지는 정보를 통해 분류해내는 것은 쉽지 않다. 예를 들자면 적치장 내부와 적치장 외부는 이미지 상에서 비슷한 특징을 보인다. 하지만 적치장 내부는 용적률 계산에 포함되어야 하고 적치장 외부는 용적률 계산에 포함되지 않아야 한다. 그에 따라 라벨이 적치장 내,외부에 따라서 다른 라벨이 붙어있는 상황이다. 결론적으로 적치장 내부와 외부는 이미지 상에서 비슷한 특징을 보이지만 라벨이 달라 모델 입장에서 제대로 기능하지 못하는 문제가 있다. 때문에 End-to-End 방식의 모델이 해당 공간을 구분하는 것은 어렵다. 우리는 적치장은 고정되어 있고 내부 오브젝트의 변화만 존재한다는 적치장의 특성을 바탕으로 사용자가 입력을 통해 적치 가능한 공간을 분할하고 해당영역 내에서의 적치 정보를 제공하는 방법을 선정하였다. 이러한 방식을 통해 적치 가능한 공간의 변동에 대해 유연하게 대응할 수 있을 것으로 판단된다.

2) 데이터 부족

적용 대상인 야외 적치장에 대한 데이터가 부족하기 때문에, 모델 학습에 문제가 생길 가능성이 커 보인다. 우리는 이를 해결하기 위해 아래 2가지 방법을 사용하려 한다.

a. 데이터 증강 기법을 적용하여 모델을 학습
b. 전이학습을 이용하여 성능 향상

3) 프로세싱 속도 제약

실시간 계산 수행이라는 제약사항은 가벼운 모델을 필요로 한다. 하지만 용적률 계산이라는 현제 문제 상황은 실시간 계산 수행보다 정확한 공간 인지가 중요하다. 따라서 사용자의 편의나 기능 이용에 문제가 생기지 않을 정도의 딜레이를 둔 배치 프로세싱을

전재에 두고 모델을 개발하려고 한다.

4) 인스턴스 식별 문제

Semantic segmentation을 통해 분류 수행 시, 같은 클래스의 인스턴스들이 이미지에서는 떨어져 있음에도 불구하고 연결되어 나타났다. 떨어져 있기는 하나 매우 조금 떨어져 있기 때문에 위와 같은 현상이 발생한 것으로 추정하고 있다. 추가로 용적률 계산을 넘어 오브젝트 관리를 위해서는 인스턴스화 하는 작업을 필요로 한다. 우리는 distance based loss function을 이용하여 객체의 boundary 검출 성능을 향상시키고자 한다.

5) 유사한 형태의 레이블

Block, Equipment가 매우 유사한 형태를 띄고 있어 사람의 눈으로도 잘 구분할 수 없는 문제가 있다. 추후에 두 클래스의 라벨을 합쳐 모델에게 더 단순한 문제를 해결하는 방식으로 개선사항이 도출될 수 있는지 알아보려고 한다.

2. 과제 수행 내용 및 중간 결과

2.1 데이터 분석 및 전처리

서로 다른 야외 적치장 3곳에 대한 데이터가 주어진다. 각 적치장에 대해서 드론으로 영상을 찍은 3000 x 4000 크기의 이미지가 학습, 테스트시 사용된다. 또한 해당 이미지들을 모두 합친 전체 전치장 이미지가 제공된다. 이는 추후 마스크 결과를 표현하는데 사용될 수 있다. 이미지의 총 갯수는 00개로 semantic segmentation 을 하기에는 다소 적은 수의 데이터셋이다. 하지만 transformer류의 pretrain이 비슷한 도메인과 비슷하지 않은 도메인인 두가지 경우가 주어진 경우 성능차이가 그렇게 크지 않다는 [0] 논문에 의거하여 pretrained 모델을 적극 사용한다.

데이터셋의 레이블을 살펴보자면 1. 적치된 물품인 'block', 2. 현장에서 사용하는 장비인 'equipment', 3. 최종적으로 본 과제에서 구별해 내고 싶은 free-space 인 'space' 로 크게 3개의 레이블로 구성할 수 있다.

제공된 데이터셋은 마이크로소프트에서 제공하는 레이블링 프로그램인 VoTT를 사용해 레이블링 되었다. 하지만 이와 같은 형식은 추후 mmseg, hugging face를 사용하는데 있어 호환성 문제로 제약사항이 존재했다. 따라서 VoTT 형식으로 된 json파일을 일반적으로 많이 사용하는 label2me 형식에 맞게 변환해 주는 과정을 거쳤다.

```
import numpy as np
import cv2
import json
from PIL import Image
import os, glob

with open('/home/gang/바탕화면/SHI_dataset/0613_1/vott-json-export/labeling_bongam1-export.json') as json_label:
    label = json.load(json_label)
```

그림 1. json2labelme (1)

그림 1과 같이 우선 필요한 패키지들을 import 해주고, 해당 VoTT 형식으로 작성된 json 파일을 불러 온다.

```
for i in label['assets']:
    img_name = label['assets'][i]['asset']['name']
    image = cv2.imread('/home/gang/바탕화면/SHI_dataset/0613_1/' + img_name)
    H, W, _ = image.shape
    seg_info = []
    for j in range(0, len(label['assets'][i]['regions'])):
        if label['assets'][i]['regions'][j]['tags'][0] != 'space':
            continue

        points_list = []

        for num in range(0, len(label['assets'][i]['regions'][j]['points'])):

            points = label['assets'][i]['regions'][j]['points'][num]
            points_list.append([points['x'], points['y']])

        seg_info.append({'label' : str(label['assets'][i]['regions'][j]['tags'][0]),
                         'points' : points_list,
                         'grounp_id' : None,
                         'shape_type' : 'polygon',
                         'flags' : {}})

    for j in range(0, len(label['assets'][i]['regions'])):
        if label['assets'][i]['regions'][j]['tags'][0] == 'space':
            continue

        points_list = []

        for num in range(0, len(label['assets'][i]['regions'][j]['points'])):

            points = label['assets'][i]['regions'][j]['points'][num]
            points_list.append([points['x'], points['y']])

        seg_info.append({'label' : str(label['assets'][i]['regions'][j]['tags'][0]),
                         'points' : points_list,
                         'grounp_id' : None,
                         'shape_type' : 'polygon',
                         'flags' : {}})
```

그림 2. json2labelme (2)

이후 해당 json의 asset 부분을 순환하면서, label2me 형식에 맞게 각 'name', 'regions' 양식을 변경해준다.

```
    with open(img_name[:-4] + '.json', 'w') as outfile:
        data = {'version' : '4.5.7',
                'flags' : {},
                'shapes' : seg_info,
                'imagePath' : img_name,
                'imageData' : None,
                'imageHeight' : H,
                'imageWidth' : W}

        json.dump(data, outfile, ensure_ascii=False, indent=4)
```

그림 3. json2labelme (3)

마지막으로 변경된 json을 새로 저장해주는 과정을 거치면 VoTT 형식으로 된 json 파일을 성공적으로 label2me 파일에 맞게 변환할 수 있다.

다음으로 필요한 과정은 label2me 형식의 json 파일의 mask 좌표를 시각화하여 loss 계산에 사용될 수 있도록 하는 것이다. 이 과정에서 space, equipment, block의 나열 순서가 space가 나중에 오게 된다면 equipment와 block의 영역을 덮어버리는 문제가 존재했다. 이런 현상의 원인은 VoTT 형식의 json 일 때 레이블을 지정해주는 순서에 의해 발생함을 알게 되었고, label2me 를 실행하여 해당 레이블들의 순서를 변경해 주는 식으로

해결했다.

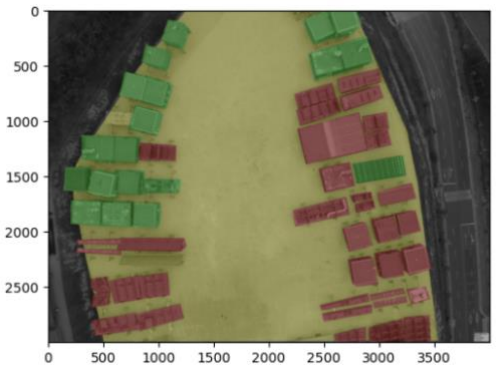


그림 4. Labeled Mask

그림4와 같이 간단히 GT를 살펴보면 빨간색으로 된 block, 초록색으로 된 equipment 그리고 아래에 노란색으로 표현되는 free-space가 존재한다. 추후 적재가 얼마나 이뤄졌는지는 mask 데이터에서 (#block pixel + #equipment pixel) / (#space pixel) 으로 간단하게 계산할 수 있다.

2.2모델 선정

드론 이미지가 가지는 특성은 위성 이미지와 유사하다고 판단하여 위성 이미지 데이터셋인 iSAD과 ADE20K 데이터셋에 대해 학습된 모델의 성능을 확인해 보고자 한다. 이와 비교하기 위해 일반적인 데이터셋에 학습된 모델과, 무작위 가중치 부여를 통한 학습 모델들과 비교하고자 한다.

비교에 쓰인 모델은 1) U-net, 2) Internimage-T, 3) BEiT, 4) Segformer 으로 4종류다.

1) U-Net

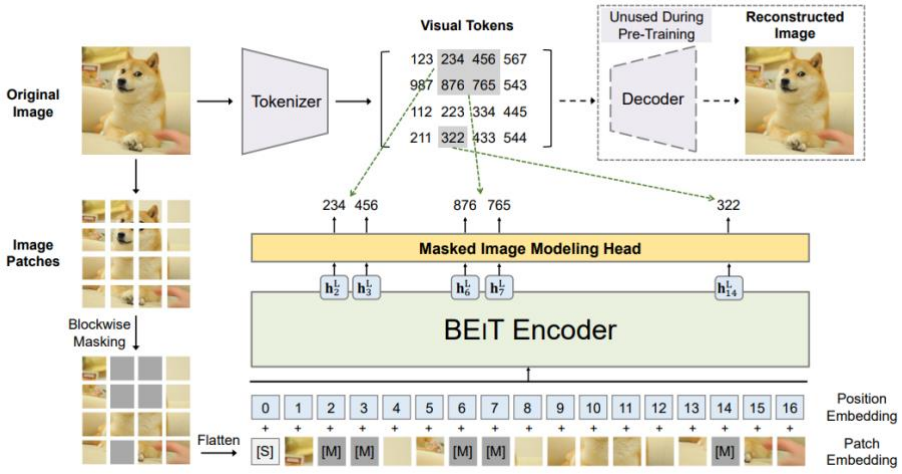
U-Net은 비교적 적은 파라미터 수를 가지면서도 좋은 성능이 입증된 모델이다. Encoder-decoder, skip connection으로 구성된 U-Net은 local, global context를 잘 잡아낼 수 있도록 고안되었다. 가벼운 모델에서의 성능이 문제 해결에 적합하다면 실시간 프로세싱도 가능할 것으로 생각되기에 이 모델을 선정하였다.

2) Internimage-T

Internimage-T는 최근 주목받고 있는 Vision transformer 계열의 architec-ture가 아닌 CNN을 기반으로한 모델이다. 하지만 object detection, segmentation에서 부분적으로 SOTA 성능을 달성하였다. 해당 모델은 BEiT, Swin 모델에 비해 비교적 적은 수의 파라미터로 동등하거나 더 높은 성능을 보여주었다.

3) BEiT

BEiT는 vision transformer 계열의 모델이다. NLP분야에서 큰 영향력을 미쳤다고 평가받는 BERT의 훈련방식인 Masked language modeling을 이미지 모델에 적용했다. 기존의 이미지 모델이 BERT의 훈련방식을 채택할 수 없었는데 BEiT는 DALL-E에서 공개한 이미지 tokenizer를 이용해서 BERT방식의 discrete token을 예측하는 masked image modeling을 사용하여 좋은 성능을 보여주었기 때문에 실험을 진행했다. 아래 그림에서 모델 개요를 확인할 수 있다.



4) Segformer

Segformer는 이름 처럼, Transformer를 Semantic Segmentation task에 적용하기 위해 만들어진 모델이다. [3] 논문에서 나와 있듯, Segformer에는 2가지 특징이 있는데, 첫번째는 multiscale feature를 output으로 뽑는 계층적 구조의 Transformer encoder로 구성된다는 것이다. 이를 통해 각 patch에서 위치 정보를 위해 사용하는 positional encoding이 필요 없도록 하여 train에 사용하지 않은 image size를 test에 사용하더라도 interpolation 사용으로 인한 성능 하락을 피할 수 있다. 두번째는 복잡한 decoder를 사용하지 않고 MLP로만 구성된 MLP decoder를 사용해, feature 맵에서 global attention을 통해 강력한 representation을 얻을 수 있다. 해당 모델은 복잡한 디자인을 가지고 있지 않고 다양한 데이터셋에서 SOTA를 달성했을 뿐만 아니라 stron zero-shot robustness 를 보여준다. 하지만 CNN모델보다 적긴 하지만 아주 작은 메모리를 가지는 edge device에서 동작할 만큼의 가볍지는 않다. 본 과제에서는 edge device 환경에서 free-space를 산출해 낼 상황은 없다고 판단하고 본 모델을 선정했다.

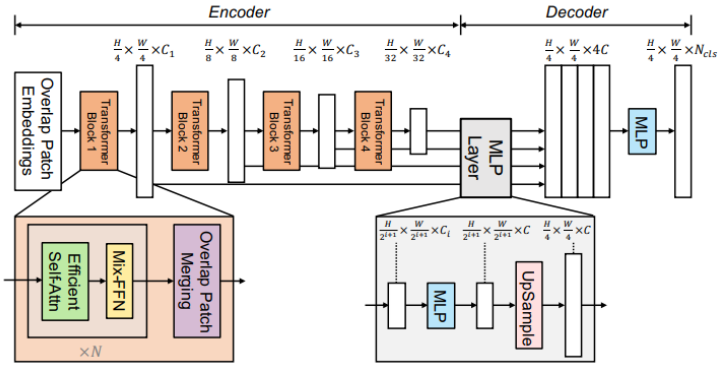


그림55. Segformer Framework

간략하게 Segformer의 프레임 워크를 살펴보면 Segformer는 그림5와 같이 2개의 메인 모듈로 구성되어 있다. Coarse 하고 fine feature를 추출하기 위한 Transformer encoder와 multi-level feature들을 directly fuse하고, semantic segmentation mask를 예측하기 위한 가벼운 All-MLP decoder. 위 그림에서 "FFN"이란 feed-forward network를 뜻한다.

2.3 모델 구현

1) U-Net

U-Net의 구조는 pytorch-UNet github(https://github.com/milesial/Pytorch-UNet)를 참조하여 구성하였으며 실험에 사용된 하이퍼 파라미터는 learningRate=0.005, batchsize=2, 옵티마이저는 Adam, Loss는 CrossEntropyLoss, Initial weight는 ran-dom weight를 선택하여 진행되었다. 데이터 전처리 과정에서 flip, crop을 이용하여 512x512 사이즈의 이미지를 학습하였다.

2) Internimage-T

Internimage-T는 Internimage github(https://github.com/opengvlab/Internimage)의 소스코드를 이용하여 학습을 진행하였다. 실험에 사용된 하이퍼 파라미터는 LearningRate=0.00006, batchsize=2, weight decay=0.05, 옵티마이저는 AdamW, Loss는 CrossEntropyLoss, Initial weight는 cityscape 512x1024 이미지에 대해 160,000 iterations 학습된 모델을 이용하였다. 데이터 전처리 과정에서 flip, crop을 이용하여 512x512 사이즈의 이미지를 학습하였다.

3) BEiT

Segformer에서 사용한 hugging-face 라이브러리를 사용하여 BEiT를 semantic segmentation task로 수행했다. 기본적으로 hugging-face를 사용하고 학습코드만 추가했다.

4) Segformer

```
from torch.utils.data import Dataset, DataLoader
from transformers import AdamW
import torch
from torch import nn
from sklearn.metrics import accuracy_score
from tqdm.notebook import tqdm
import os
from PIL import Image
from transformers import SegformerForSemanticSegmentation, SegformerFeatureExtractor
import pandas as pd
import cv2
import numpy as np
import albumentations as aug
import matplotlib.pyplot as plt
```

```
WIDTH = 512
HEIGHT = 512
```

그림66. Segformer Implementation Code (1)

pretrained된 segformer를 사용하기 위해서 관련 라이브러리들을 불러왔다. 특히 'pip install --upgrade transformers'를 사용해 huggingface에서 지원하는 transformers 라이브러리를 설치해준다.

```
class ImageSegmentationDataset(Dataset):
    """Image segmentation dataset."""

    def __init__(self, root_dir, feature_extractor, transforms=None, train=True):
        """
        Args:
            root_dir (string): Root directory of the dataset containing the images + annotations.
            feature_extractor (SegFormerFeatureExtractor): feature extractor to prepare images + segmentation maps.
            train (bool): Whether to load "training" or "validation" images + annotations.
        """
        self.root_dir = root_dir
        self.feature_extractor = feature_extractor
        self.train = train
        self.transforms = transforms

        sub_path = "train" if self.train else "test"
        self.img_dir = os.path.join(self.root_dir, "JPEGImages")
        self.ann_dir = os.path.join(self.root_dir, "SegmentationClassPNG")

        # read images
        image_file_names = []
        for root, dirs, files in os.walk(self.img_dir):
            image_file_names.extend(files)
        self.images = sorted(image_file_names)

        # read annotations
        annotation_file_names = []
        for root, dirs, files in os.walk(self.ann_dir):
            annotation_file_names.extend(files)
        self.annotations = sorted(annotation_file_names)

        assert len(self.images) == len(self.annotations), "There must be as many images as there are segmentation maps"

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):

        image = cv2.imread(os.path.join(self.img_dir, self.images[idx]))
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        segmentation_map = Image.open(os.path.join(self.ann_dir, self.annotations[idx]))
        segmentation_map = np.array(segmentation_map)


        if self.transforms is not None:
            augmented = self.transforms(image=image, mask=segmentation_map)
            # randomly crop + pad both image and segmentation map to same size
            encoded_inputs = self.feature_extractor(augmented['image'], augmented['mask'], return_tensors="pt")
        else:
            encoded_inputs = self.feature_extractor(image, segmentation_map, return_tensors="pt")

        for k,v in encoded_inputs.items():
            encoded_inputs[k].squeeze_() # remove batch dimension

        return encoded_inputs
```

그림77. Segformer Implementation Code (2)

Segmentation 관련 커스텀 데이터셋을 사용하기 위해서 ImageSegmentationDataset 클래스를 정의한다. 본 클래스에서는 데이터셋에 대한 경로와 feature extractor, transforms 등을 정의하고, transformers 라이브러리 형식에 맞게 정리된 디렉토리 경로를 지정한다. 그리고 각 이미지 파일에 대한 파일명과 annotations를 멤버 변수인 image, annotations에 저장한다.

```
transform = aug.Compose([
    aug.Flip(p=0.5)
])
```

```
# root_dir = '../data/SHI/labelme_format/data_dataset_voc/'
root_dir = '/home/gang/바탕화면/GA2023/labelme_format/'

train_dir = root_dir + 'splited_train_valid_test/train'
valid_dir = root_dir + 'splited_train_valid_test/val'
test_dir = root_dir + 'splited_train_valid_test/test'


feature_extractor = SegformerFeatureExtractor(align=False, reduce_zero_label=False)

train_dataset = ImageSegmentationDataset(root_dir=train_dir, feature_extractor=feature_extracto
valid_dataset = ImageSegmentationDataset(root_dir=valid_dir, feature_extractor=feature_extracto
```

```
/home/gang/anaconda3/envs/openmmlab/lib/python3.8/site-packages/transformers/models/segforme
r/feature_extraction_segformer.py:28: FutureWarning: The class SegformerFeatureExtractor is d
eprecated and will be removed in version 5 of Transformers. Please use SegformerImageProcesso
r instead.
  warnings.warn(
```

```
print("Number of training examples:", len(train_dataset))
print("Number of validation examples:", len(valid_dataset))
```

```
Number of training examples: 45
Number of validation examples: 5
```

그림88. Segformer Implementation Code (3)

우선 학습을 위해서 디렉토리 경로를 저장해주고, feature_extractor는 segformer 레퍼런스에 따라서 SegformerFeatureExtractor(align=False, reduce_zero_label=Fasle) 로 설정해준다. 이후에 train, validation set을 나눠주고 갯수를 찍어보면 미리 놔눠 놓은 대로 9:1 비율의 train, validation set을 확인할 수 있다.

```
classes = pd.read_csv('/home/gang/바탕화면/GA2023/labelme_format/class_dict.csv')['name']
id2label = classes.to_dict()
label2id = {v: k for k, v in id2label.items()}
```

```
model = SegformerForSemanticSegmentation.from_pretrained("nvidia/mit-b3", ignore_mismatched_siz
                                                         num_labels=len(id2label), id2label=id2
                                                         reshape_last_stage=True)
```

```
...
```

```
optimizer = AdamW(model.parameters(), lr=0.00006)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
print(device)
print("Model Initialized!")
```

```
...
```

```
from torch.utils.tensorboard import SummaryWriter

writer = SummaryWriter('scalar/mit-b3_epoch_10000_0711/')
```

```
best_val_accuracy = 0.0
best_model_state_dict = None
```

그림99. Segformer Implementation Code (4)

다음으로 classes 변수에 클래스를 담고 있는 csv의 이름들을 저장하고, 사용할 모델인 mit-b3 모델을 불러온다. optimizer는 AdamW를 사용하고 learning rate는 권장사항인 0.00006으로 설정해줬다. 이후 tensorboard 형태로 저장할 writer와 best 변수들을 초기화 해준다.

```
for epoch in tqdm(range(1, 10001), desc='epoch'):  # loop over the dataset multiple times
    print("Epoch:", epoch)
    pbar = tqdm(train_dataloader, desc='train')
    accuracies = []
    losses = []
    val_accuracies = []
    val_losses = []
    model.train()
    for idx, batch in enumerate(pbar):
        # get the inputs;
        pixel_values = batch["pixel_values"].to(device)
        labels = batch["labels"].to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward
        outputs = model(pixel_values=pixel_values, labels=labels)

        # evaluate
        upsampled_logits = nn.functional.interpolate(outputs.logits, size=labels.shape[-2:],
                                                     mode="bilinear", align_corners=False)
        predicted = upsampled_logits.argmax(dim=1)

        mask = (labels != 255) # we don't include the background class in the accuracy calculation
        pred_labels = predicted[mask].detach().cpu().numpy()
        true_labels = labels[mask].detach().cpu().numpy()
        accuracy = accuracy_score(pred_labels, true_labels)
        loss = outputs.loss
        accuracies.append(accuracy)

        losses.append(loss.item())
        pbar.set_postfix({'Batch': idx, 'Pixel-wise accuracy': sum(accuracies)/len(accuracies),
                          'Loss': sum(losses)/len(losses)})

        # backward + optimize
        loss.backward()
        optimizer.step()


        # if idx % 10 == 9:
        writer.add_scalar('training loss', loss, epoch * len(train_dataloader) + idx)
    else:
        model.eval()
        with torch.no_grad():
            for idx, batch in enumerate(tqdm(valid_dataloader, desc='vaild')):
                pixel_values = batch["pixel_values"].to(device)
                labels = batch["labels"].to(device)

                outputs = model(pixel_values=pixel_values, labels=labels)
                upsampled_logits = nn.functional.interpolate(outputs.logits, size=labels.shape[-2:],
                                                             mode="bilinear", align_corners=False)
                predicted = upsampled_logits.argmax(dim=1)

                mask = (labels != 255) # we don't include the background class in the accuracy calculation
                pred_labels = predicted[mask].detach().cpu().numpy()
                true_labels = labels[mask].detach().cpu().numpy()
                accuracy = accuracy_score(pred_labels, true_labels)
                val_loss = outputs.loss
                val_accuracies.append(accuracy)
                val_losses.append(val_loss.item())
                writer.add_scalar('validation loss', val_loss, epoch * len(valid_dataloader) + idx)

        val_accuracy = sum(val_accuracies)/len(val_accuracies)
        if val_accuracy > best_val_accuracy:
            best_val_accuracy = val_accuracy
            best_model_state_dict = model.state_dict()

    print(f"""    Train Pixel-wise accuracy: {sum(accuracies)/len(accuracies)}
    Train Loss: {sum(losses)/len(losses)}
    Val Pixel-wise accuracy: {sum(val_accuracies)/len(val_accuracies)}
    Val Loss: {sum(val_losses)/len(val_losses)}\n\n""")

writer.close()

torch.save(best_model_state_dict, '/home/gang/바탕화면/GA2023/savedmodel/epoch_10000_0711.pt')
```

그림1010. Segformer Implementation Code (5)

학습을 위한 코드는 위와 같다.

```
test_image_path = '/home/gang/바탕화면/GA2023/labelme_format/splited_train_valid_test/test/JPEGImages/'
test_mask_path = '/home/gang/바탕화면/GA2023/labelme_format/splited_train_valid_test/test/SegmentationClassPNG/'
test_gt_path = '/home/gang/바탕화면/GA2023/labelme_format/splited_train_valid_test/test/SegmentationClassVisualization/'

src_images = os.listdir(test_image_path)
src_masks = os.listdir(test_mask_path)
src_gt = os.listdir(test_gt_path)

src_images.sort()
src_masks.sort()
src_gt.sort()
```

```
for img, mask, gt in zip(src_images, src_masks, src_gt):
    image_img = Image.open(test_image_path + img)
    mask_img = Image.open(test_mask_path + mask).convert('L')
    gt_img = Image.open(test_gt_path + gt)

    feature_extractor_inference = SegformerFeatureExtractor(do_random_crop=False, do_pad=False)

    pixel_values = feature_extractor_inference(image_img, return_tensors="pt").pixel_values.to(device)

    model.eval()
    outputs = model(pixel_values=pixel_values)# logits are of shape (batch_size, num_labels, height/4, width/4)
    logits = outputs.logits.cpu()

    upsampled_logits = nn.functional.interpolate(logits,
                size=image_img.size[::-1], # (height, width)
                mode='bilinear',
                align_corners=False)

    # Second, apply argmax on the class dimension
    seg = upsampled_logits.argmax(dim=1)[0]
    color_seg = np.zeros((seg.shape[0], seg.shape[1], 3), dtype=np.uint8) # height, width, 3\
    for label, color in enumerate(palette):
        color_seg[seg == label, :] = color
    # Convert to BGR
    color_seg = color_seg[..., ::-1]

    # Show image + mask
    img = np.array(image_img) * 0.5 + color_seg * 0.5
    img = img.astype(np.uint8)

    fig, axs = plt.subplots(1, 3, figsize=(20, 10))
    axs[0].imshow(gt_img)
    axs[1].imshow(img)
    axs[2].imshow(color_seg)

    plt.savefig('./labelme_format/SegFormer_result/b3_epoch_10000_0711/'+mask, pad_inches=0)
```

그림1111. Segformer Implementation Code (6)

추론을 위해서 test를 위한 데이터셋이 위치한 디렉토리를 지정해주고 mask를 출력하는 것 까지 진행하면 ADE20K 데이터셋으로 pretrained 된 segformer를 사용할 수 있다.

2.4 결과

1) U-Net

U-net의 경우 iteration = 20,000에서 mAcc = 82.13, mIoU = 72.02로 가장 낮은 성능을 보여주었다. Equiment, block과 space 사이의 경계를 명확하게 구분하지 못하고 space 경계도 제대로 잡아내지 못했다.

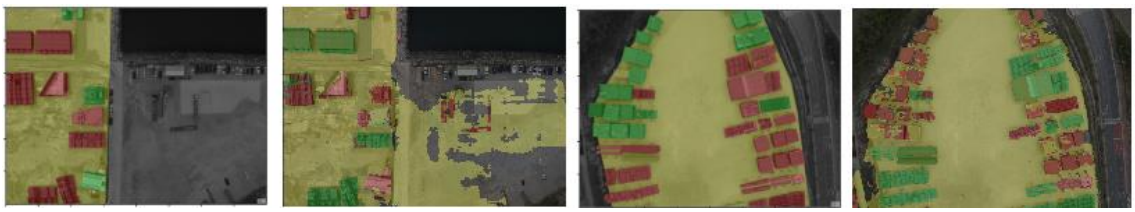


그림1212. U-Net Result (좌) GT, (우) pred

2) Internimage-T

Internimage-T의 경우 iteration = 128,000에서 mAcc = 89.37, mIoU = 81.31로 측정되었

다. Equipment, block 구분에 대한 정확도가 다소 낮고 붙어있는 object들에 대한 bound를 제대로 생성해내지 못하는 것으로 보인다. 또한 space와 background 구분에 대한 성능이 좋지 못하였다.

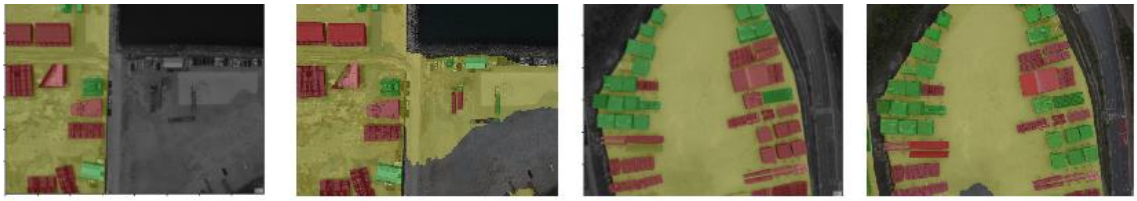


그림1313. Internimage-T Result (좌) GT, (우) pred

3) BEiT

Iteration = 10000, batch_size=5로 학습시켰으며 대략 200분 정도가 소요되었다. 특히 mAcc는 약 0.944 의 성능을 보인다. 하지만 아래 사진에서 알 수 있듯 다른 모델과 마찬가지로 bound를 제대로 생성하지 못하고 있다.

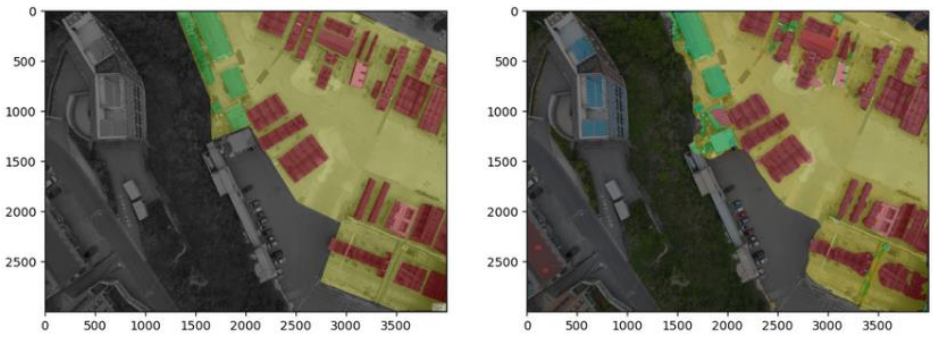


그림1414. BEiT Result (좌) GT, (우) pred

아래는 앞서 언급한 비슷한 "특징을 가지는 공간에 대한 구분 문제"에 해당하는 사진이다. 주차장은 적치장 외부로써 라벨링 되어있지만 현재 모델은 이를 구분하지 못하고 있는 상황이다. 또한 마찬가지로 깔끔한 bound를 찾지 못하고 있다.

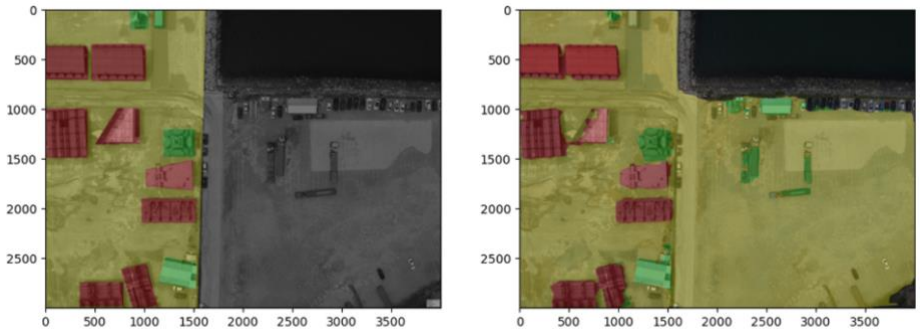


그림1515. BEiT Result (2) (좌) GT, (우) pred

4) Segformer

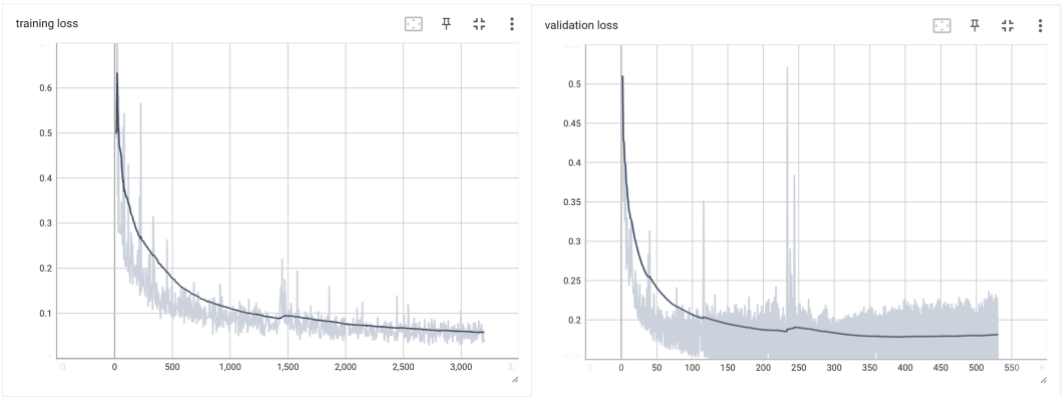


그림1616. Segformer Tensorboard Plot

위 tensorboard 결과에 따라서 epoch 300 이상 부터는 overfitting이 발생함을 알 수 있다. 이후에 실험에서는 epoch은 200-300 정도면 충분히 최상의 모델을 저장할 수 있을 것임을 알 수 있다.

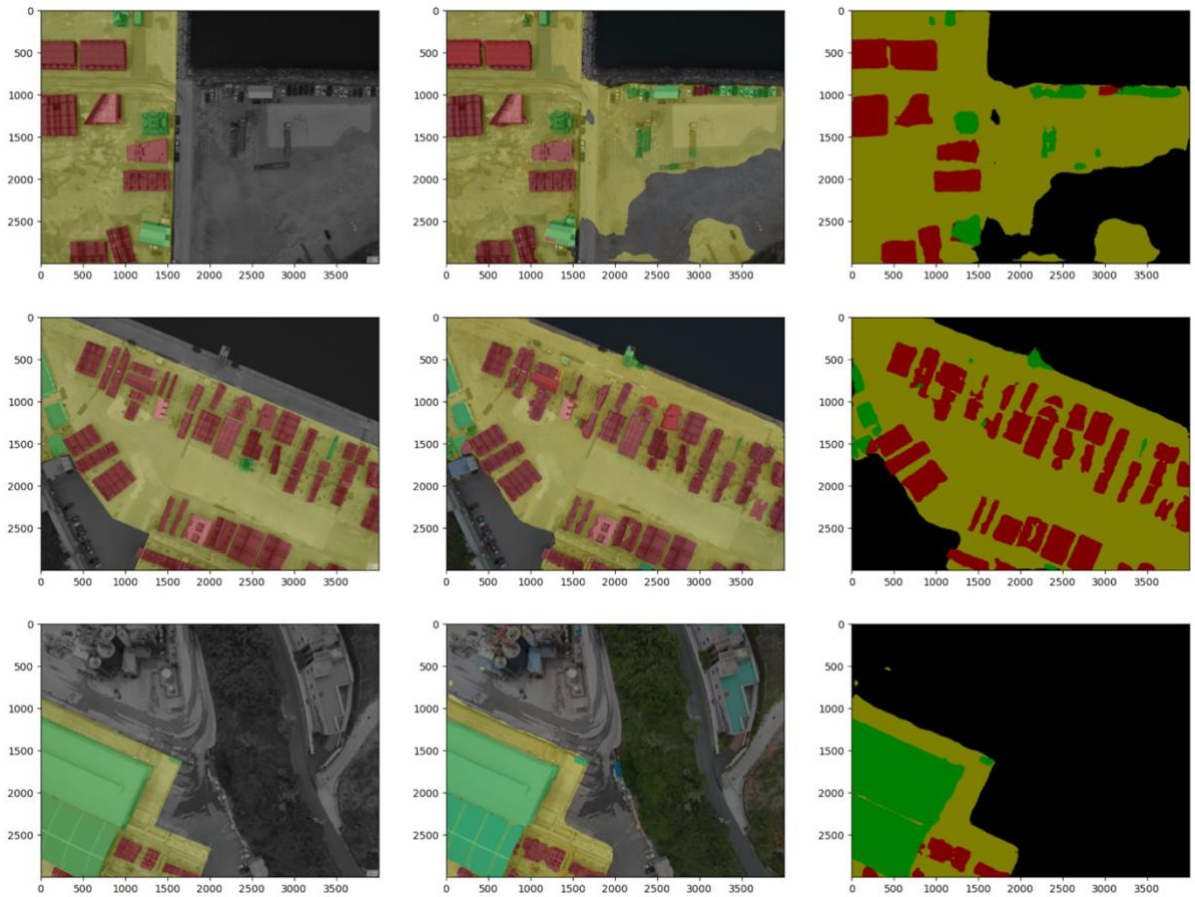


그림1717. Segformer Inference Results (좌) GT, (중간) pred, (우) pred mask only

위 결과로 알 수 있듯 비슷한 특징을 가지는 공간을 제대로 잡아내지 못하고 있다. 최상단 사진에서 주차장으로 인식되어야 할 부분이 추론 단계에서 free-space로 인식함을

알 수 있다. 이는 사람이 육안으로 보아도 구별이 잘 가지 않는 문제가 있다. 전반적으로 준수한 결과를 보여준다. 하지만 segmentation mask들의 edge가 매우 울퉁불퉁해 개별 인스턴스들은 준수하게 잡아낼 수 없음을 알 수 있다.

최종 결과는 아래와 같이 도표로 정리할 수 있다.

Model	mAcc	mIoU
U-Net	0.821	72.02
Internimage-T	0.894	81.31
BEiT	0.944	-
Segformer	**0.978**	-

표 1. Model Results

2.5 논의

실험에 사용된 모델이 공통적으로 분류해내지 못하는 상황은 아래와 같다.

1) 비슷한 특징을 가지는 공간에 대한 구분 문제

2) Block과 Equipment 분류 실패 문제

3) 여러 개의 인스턴스를 개별로 구분하지 못하고 하나의 큰 오브젝트로 분류하는 문제

문제를 해결하기 위해 아래의 방법을 시도할 예정이다.

1) 사용자의 input을 받아 적치장 내외부 boundary를 정의
2) Block과 Equipment의 클래스를 합병
3) Distance based loss function을 사용
4)

3. 이후 수행 목표

마지막으로 과제를 진행하는 도중에 다양한 문제점들을 만났다. 각각의 문제에 대해 여러 가지 해결 방안 제시 후 타당한 것을 여러 번의 회의를 통해 구체화하였다. 먼저 산업 도메인에서 중요한 이슈를 우선적으로 선별하고 순서대로 해결할 예정이다. 현재 식별한 우선순위는 아래와 같다.

1. 적치장 내, 외부 구분 문제 해결
2. 객체의 명확한 edge을 따기 위한 방법 고안
3. UI 구현
4. Block, Equipment 라벨링 병합

각각의 해결 방법은 앞서 설명했다. 이 우선순위를 바탕으로 이후 수행 목표를 세우고자 한다.

우선 2.5에서 논의한 바와 같이 적치장 내&외부 boundary를 정의하는 것으로 관심있는 free-space 이외의 구역에 대해서는 추론과 학습이 이뤄지지 않도록 하는 것이 첫번째 목표가 될 것이다. 따라서 사용자가 정의한 boundary에 맞춰서 추론이 이뤄질 수 있도록 시스템을 구축 해야한다.

두번째로 edge가 매끄럽지 못한 것을 매끄럽게 설정해주는 것이 free-space 정확도 향상에 도움을 줄 것이다. 해당 작업으로 개별 인스턴스에 대한 뚜렷한 구별 또한 해줄 수 있을 것으로 보인다. edge가 매끄럽지 못한 것은 첫번째 이유로 labeling 자체가 정확하게 세밀하게 되어 있지 않은 것으로 보고 있고 다른 이유로는 semantic segmentation을 진행하기에 50장이라는 데이터셋이 턱없이 부족한 것이다. 추후에 추가적으로 제공되는 데이터셋을 계속 학습에 추가하고, labeling을 세밀하게 하거나, edge를 깔끔하게 segmen-tation 하는 방법에 대해서 추가적인 논의가 필요하다.

또한 해당 드론 이미지에서의 free-space 추론 시스템을 웹이나 앱으로 구현하여 사용자가 드론으로 찍은 사진을 실시간으로 전송한다면 trained model이 이를 계산하여 적재율을 계산해주는 것이 마지막 목표이다. 웹으로 구현하기에는 도메인 등의 제약사항이 따라 간단히 윈도우 컴퓨터 상에서 동작할 수 있는 프로그램 형식으로 구현할 것이다. 이는 외부인이 간단하게 접속할 수 없도록, 회사 내 컴퓨터 프로그램으로 인증을 받아 사용하는 방식을 가정한다. 앱을 구하는 것은 2가지 방법으로 생각해 볼 수 있는데, 첫번째는 실시간으로 드론이 이미지를 전송하면 자동으로 model이 background에서 추론을 실시하고, 계산이 완료되면 결과 버튼이 출력되어 해당 버튼을 누르면 전체 적치장의 masked 된 이미지와 적재율을 계산해주는 것이다. 두번째 구현 방식은 전체 야외 적치장 사진을 먼저 띄워놓고, 실시간으로 3000 x 4000 부분씩 mask로 변환하여 출력하는 것이다. 두 가지 방법이 큰 차이는 없지만 시각적으로 눈앞에서 mask가 변환되는 모습을

보는 것이 더욱 현실감 있다고 생각하여 두번째 방법으로 앱을 구현하는 것이 목표이다.

마지막으로 실제 정확도 향상에 기여할 지는 모르지만 block과 equipment 레이블링을 통합하여 background, space, block 총 3가지의 레이블링을 사용한다면 데이터가 50장 내외로 적은 환경에서 좀 더 정확도를 올릴 수 있을 것이라고 생각했다. 비교적 가장 간단한 작업이기 때문에 추후 앱까지 구현이 완료된 이후에 추가 정확도 향상이 필요하다고 생각되는 경우에 적용해볼 수 있는 방안이다.

4. 갱신된 과제 추진 계획 및 역할 분담

4.1개발 일정

<table>
<tr><th colspan="3">6월</th><th colspan="5">7월</th><th colspan="5">8월</th><th colspan="4">9월</th></tr>
<tr><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>5주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td></tr>
<tr><td colspan="3">segmentation 스터디</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td colspan="4">기존 모델 적용</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="3">야외 적치장 데이터셋 모델 성능 확인</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">최적 모델 확인</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">중간 보고서 작성</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">모델 최적화 앱 개발</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">github 연동</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">최종 보고서 준비</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">포스터 제작</td></tr>
</table>

표 2. 개발 일정

5. 구성원별 진척도

5.1개별 진척도 내용

개별로 모델을 맡아 아래 모델들에 대해 전이학습을 구행, 성능평가를 마쳤다. 각각의 모델에서 두드러지는 특징이나 이슈상황을 정리하고 회의 후 교수님과 미팅을 가지며 이슈에 대한 방안을 논의했다. 본 과제에서 크게 고려해본 모델은 아래와 같다.

- U-Net
- Internimage-T
- BEiT
- SegFormer

구성원 별로 아래와 같은 내용을 맡아서 진행했으며, 공동으로 작업한 내용 또한 아래 표2에서 확인할 수 있다.

이름	진척도
김경현	• 데이터 활용 방법 조사 • 모델 개발 및 평가 • 프로젝트 매니징 • SegFormer 학습, 평가 수행
유일해	• 선행 연구 조사 • 데이터 관리 및 전처리 • 모델 개발 • U-Net, Upernet-Internimage-T 학습, 평가 수행
신민건	• 도메인 분석 • 데이터 전처리 • 모델 개발 • BEiT 학습, 평가 수행
공동	• 데이터 및 과제 분석 • github 관리 • 보고서 작성 및 포스터 제작 • 발표 및 시연 준비

표 3. 개별 진척도 리스트

6. 참고자료

[1] Bao, Hangbo, et al. "Beit: Bert pre-training of image transformers." *arXiv preprint arXiv:2106.08254* (2021).

[2] Wang, Wenhai, et al. "Internimage: Exploring large-scale vision foundation models with deformable convolutions." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.

[3] Xie, Enze, et al. "SegFormer: Simple and efficient design for semantic segmentation with transformers." *Advances in Neural Information Processing Systems* 34 (2021): 12077-12090.