```
import os
import cv2
import json
import torch
import pickle
import numpy as np
from copy import deepcopy
from pathlib import Path

from detectron2.projects import point_rend
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg


class ActModel:
    def __init__(
        self,
        model_path,
        simple_image_output_path,
        detail_image_output_path,
        simple_result_ratio_output_path,
        detail_result_ratio_output_path,
    ):
        self.detail_color_list = [  # 0925 수정
            [(172, 177, 176), 'Resin'],  # 0  Background
            [(7, 25, 82), 'Isotropic'],  # 1
            [(106, 27, 154), 'Fine_Mosaic'],  # 2
            [(255, 87, 34), 'Medium_Mosaic'],  # 3
            [(195, 2, 2), 'Coarse_Mosaic'],  # 4
            [(255, 235, 59), 'Fine_Leaflet'],  # 5
            [(2, 195, 25), 'Medium_Leaflet'],  # 6
            [(5, 101, 90), 'Coarse_Leaflet'],  # 7
            [(63, 238, 248), 'Mineral'],  # 8
            [(1, 196, 174), 'Fine_Flow'],  # 9
            [(146, 98, 17), 'Medium_Flow'],  # 10
            [(252, 7, 248), 'Coarse_Flow'],  # 11
            [(194, 252, 69), 'Fusinite'],  # 12
            [(8, 0, 15), 'Pore'],  # 13
            [(172, 177, 176), 'Resin'],  # 14
            [(245, 155, 147), 'Carbon_Deposition'],  # 15
            [(255, 255, 255), 'Incipient'],  # 16
        ]  # 클래스 색상값 정보
        self.simple_color_list = [  # 0925 수정
            [(172, 177, 176), 'Resin'],
            [(7, 25, 82), 'Isotropic'],
            [(106, 27, 154), 'Fine_Mosaic'],
            [(255, 87, 34), 'Medium_Mosaic'],
            [(195, 2, 2), 'Coarse_Mosaic'],
            [(2, 195, 25), 'Leaflet'],  # Fine Leaflet
            [(2, 195, 25), 'Leaflet'],  # Medium Leaflet
            [(2, 195, 25), 'Leaflet'],  # Coarse Leaflet
            [(63, 238, 248), 'Mineral'],
            [(146, 98, 17), 'Flow'],  # Fine Flow
            [(146, 98, 17), 'Flow'],  # Medium Flow
            [(146, 98, 17), 'Flow'],  # Coarse Flow
            [(194, 252, 69), 'Inert'],  # Fusinate
            [(8, 0, 15), 'Pore'],
            [(172, 177, 176), 'Resin'],
            [(194, 252, 69), 'Inert'],  # 원랜 Carbon Deposit Fusinate 색으로 통일했음
            [(255, 255, 255), 'Aniso_Fusinite'],  # Incipient
        ]  # 클래스 색상값 정보
        self.num_classes = 17  # 총 개수
        self.model_path = model_path  # Unet 모델경로
        self.detail_result_output_path = detail_result_ratio_output_path  # 예측결과 비율
```

```
저장경로 (detail)
        self.simple_result_output_path = simple_result_ratio_output_path  # 예측결과 비율
저장경로 (simple)
        self.detail_image_output_path = detail_image_output_path  # 예측결과 mask 파일 저
장경로
        self.simple_image_output_path = simple_image_output_path  # 예측결과 mask 파일 저
장경로
        self.device =  'cuda:0' if torch.cuda.is_available() else  'cpu'  # cuda or cpu

    def load_model(self, model_path):  # 딥러닝 모델 세팅 함수
        cfg = get_cfg()
        point_rend.add_pointrend_config(cfg)

        with open('./model/config.pkl', 'rb') as cfg_pickle_file:
            cfg = pickle.load(cfg_pickle_file)

        cfg.MODEL.WEIGHTS = model_path
        cfg.MODEL.DEVICE = self.device
        my_metadata = cfg['metadata']

        predictor = DefaultPredictor(cfg)

        return predictor, my_metadata

    def predict(self, input_path):  # 실제 예측함수
        Path(self.simple_image_output_path).mkdir(exist_ok=True, parents=True)
        Path(self.simple_result_output_path).mkdir(exist_ok=True, parents=True)
        Path(self.detail_image_output_path).mkdir(exist_ok=True, parents=True)
        Path(self.detail_result_output_path).mkdir(exist_ok=True, parents=True)

        predictor, metadata = self.load_model(self.model_path)  # 모델 load

        # 파일 or 디렉터리 구분
        if os.path.isfile(input_path):
            input_file_list =  [Path(input_path)]
        else:
            input_file_list = Path(input_path).rglob('*.*')

        detail_result_ratio = {'total': [], 'datalist': []}
        simple_result_ratio = {'total': [], 'datalist': []}

        for file_path in input_file_list:
            im = cv2.imread(str(file_path))
            outputs = predictor(im)

            x = outputs['sem_seg'].argmax(dim=0)

            label_class_predicted = x.cpu()

            # 마스크 만들기
            simple_mask_image = self.gray_to_color_class(
                label_class_predicted, self.simple_color_list
            )  # 컬러마스크 변경
            cv2.imwrite(f'{self.simple_image_output_path}/{file_path.stem}.png',
simple_mask_image)

            detail_mask_image = self.gray_to_color_class(
                label_class_predicted, self.detail_color_list
            )  # 컬러마스크 변경
            cv2.imwrite(f'{self.detail_image_output_path}/{file_path.stem}.png',
detail_mask_image)
```

```
            # 결과 뽑기  # 0925 S_colo_list 수정
            detail_result, simple_result = self.get_result_ratio(
                label_class_predicted,
                self.detail_color_list,
                self.simple_color_list,
                file_path,
                f'{self.simple_image_output_path}/{file_path.name}',
                f'{self.detail_image_output_path}/{file_path.name}',
            )

            # 결과 붙이기
            detail_result_ratio['datalist'].append(detail_result)
            simple_result_ratio['datalist'].append(simple_result)

        # datalist 를 통해 total 계산  # 0925 배열이었나 기억안남. 가서 확인해야함. 배열기준으로
cal_pore 짬
        simple_result_ratio['total'] =
[self.get_total(simple_result_ratio['datalist'])]
        detail_result_ratio['total'] =
[self.get_total(detail_result_ratio['datalist'])]

        # 0925 추가
        simple_result = self.calculate_without_resin_pore(simple_result_ratio,
simple_flag=True)
        detail_result = self.calculate_without_resin_pore(detail_result_ratio,
simple_flag=False)
        detail_result = self.get_cf_from_simple(detail_result, simple_result)

        detail_str_result = self.change_to_str(detail_result)
        simple_str_result = self.change_to_str(simple_result)

        print(detail_str_result)
        print(simple_str_result)

        with open(f'{self.detail_result_output_path}/{file_path.stem}.json', 'w') as
detail_f:  # 결과비율 저장
            json.dump(detail_str_result, detail_f, ensure_ascii=False, indent=4)

        with open(f'{self.simple_result_output_path}/{file_path.stem}.json', 'w') as
simple_f:  # 결과비율 저장
            json.dump(simple_str_result, simple_f, ensure_ascii=False, indent=4)

    def gray_to_color_class(self, gray_array, color_list):  # id 값으로 된 grayscale 결과
값을 컬러로 변경하는 함수
        h, w = gray_array.shape[:2]
        image = np.zeros((h, w, 3), dtype=np.uint8)

        for index, (color_code, class_name) in enumerate(color_list):
            image[gray_array == index] = [color_code[2], color_code[1], color_code[0]]

        return image

    def get_result_ratio(
        self,
        label_image,
        d_color_list,
        s_color_list,
        file_path,
        simple_output_file,
        detail_output_file,  # 0925 파라미터에서 s_color_list 만 추가
    ):  # 결과에서 각 클래스별 비율 계산 함수
        detail_result_ratio = {'ImageNo': file_path.stem, 'ImageOrg': str(file_path),
'ImageChg': detail_output_file}
```

```
        simple_result_ratio = {'ImageNo': file_path.stem, 'ImageOrg': str(file_path),
'ImageChg': simple_output_file}

        values, counts = np.unique(label_image, return_counts=True)
        counts.sum()

        # 0925 여기부터 return 까지 다 바뀌었음
        for d_key in d_color_list:
            detail_result_ratio[d_key[1]] = 0

        for s_key in s_color_list:
            simple_result_ratio[s_key[1]] = 0

        for result_index, color_index in enumerate(values):
            counter = counts[result_index]
            d_class_name = d_color_list[color_index][1]
            s_class_name = s_color_list[color_index][1]

            detail_result_ratio[d_class_name] += counter
            simple_result_ratio[s_class_name] += counter

        return detail_result_ratio, simple_result_ratio

    def get_total(self, datalist):  # 0925 다바뀜
        total = {}
        ctg_list = []

        for image_result in datalist:
            ctg_list += list(image_result.keys())

        ctg_list = list(set(ctg_list))

        for key in [value for value in ctg_list if 'Image' not in value]:
            total[key] = 0

        for class_key in image_result:
            if 'Image' in class_key:
                continue
            total[class_key] += image_result[class_key]

        return total

    def calculate_without_resin_pore(self, total_dict, simple_flag):  # 0925 추가
        result = deepcopy(total_dict)
        result['total'][0] = self.calculate_percentage(total_dict['total'][0],
simple_flag)

        for i, data in enumerate(total_dict['datalist']):
            result['datalist'][i] = self.calculate_percentage(data, simple_flag)

        return result

    def calculate_percentage(self, class_dict, simple_flag):  # 0925 추가
        result = deepcopy(class_dict)

        total_pixel = 0
        total_pixel_without_resin_port = 0
        cf_total = 0

        for key in class_dict:
            if 'Image' in key:
                continue

            total_pixel += class_dict[key]
```

```
            if key in [
                'Isotropic',
                'Fine_Mosaic',
                'Medium_Mosaic',
                'Coarse_Mosaic',
                'Leaflet',
                'Flow',
                'Aniso_Fusinite',
            ]:
                cf_total += class_dict[key]

            if key not in ['Pore', 'Resin']:
                total_pixel_without_resin_port += class_dict[key]

        for key in class_dict:
            if 'Image' in key:
                continue

            if key in ['Pore', 'Resin']:
                result[key] = class_dict[key] / total_pixel * 100
            else:
                result[key] = class_dict[key] / total_pixel_without_resin_port * 100

        if simple_flag:
            result['CF'] = 0.0

            for cf_key in [
                ['Isotropic', 1],
                ['Fine_Mosaic', 2.5],
                ['Medium_Mosaic', 3],
                ['Coarse_Mosaic', 4],
                ['Leaflet', 5],
                ['Flow', 3.5],
                ['Aniso_Fusinite', 2],
            ]:
                result['CF'] += (class_dict[cf_key[0]] / cf_total * 100) * cf_key[1]

        return result

    def get_cf_from_simple(self, detail_result, simple_result):  # 0925 추가 (total 안이
list인경우 기준임)
        detail_result['total'][0]['CF'] = simple_result['total'][0]['CF']

        for i, data in enumerate(simple_result['datalist']):
            detail_result['datalist'][i]['CF'] = data['CF']

        return detail_result

    def change_to_str(self, d):  # 0925 추가
        if isinstance(d, dict):
            for key, value in d.items():
                d[key] = self.change_to_str(value)
        elif isinstance(d, list):
            for i, data in enumerate(d):
                d[i] = self.change_to_str(data)
        elif isinstance(d, str):
            d = d
        else:
            d = str(round(d, 2))
        return d
```