Ứng dụng Deep Learning cho Phân đoạn Khối u Gan trên Ảnh CT 3D

Báo cáo khóa luận này trình bày về việc xây dựng hệ thống tự động phân đoạn gan và khối u gan từ ảnh CT 3D bằng các kỹ thuật học sâu. Mô hình sử dụng chiến lược Coarse-to-Fine với hai giai đoạn: giai đoạn phân đoạn gan và giai đoạn phân đoạn khối u trong vùng ROI gan.

Mục lục

---

1. Giới thiệu đề tài

- Phân đoạn gan và khối u gan trên ảnh CT 3D.
- Ứng dụng trong hỗ trợ chẩn đoán y tế.
- Sử dụng mô hình Coarse-to-Fine 2 giai đoạn (gan → ROI → khối u).
- Giai đoạn 1 phân đoạn gan trên ảnh CT3D
- Giai đoạn 2 phân đoạn khối u trên vùng ROI được cắt từ ảnh CT3D, ở giai đoạn này sử dụng mô hình 2d, giả định 3 lát cắt làm chiều sâu của ảnh

2. Hệ thống & Môi trường

- Yêu cầu hệ thống tối thiểu: Google Colab / GPU T4
- Ngôn ngữ : Python 3.10+, PyTorch 2.x
- Các thư viện liên quan: monai, torchvision, SimpleITK, requests, libtorrent, scikit-learn, medpy, PyYAML, scipy, scikit-image, nibabel

3. Cấu trúc thư mục

```
project/
├─ datasets/
│   ├─ get_datasets.py
│   ├─ lits.py
│   ├─ prepare_data.py
├─ init/
│   ├─ install_dependencies.py
├─ models/
│   ├─ layers/
│   ├─ ...
├─ notebooks/
│   ├─ ...
├─ processing/
│   ├─ augmentation.py
│   ├─ postprocessing.py
│   ├─ preprocessing.py
├─ utils/
│   ├─ ...
├─ parameters.yaml
└─ requirements.txt
```

- **Thư mục datasets**: gồm các file khai báo custom dataset, file chứa hàm download dataset từ source public
- **Thư mục init**: gồm file khai báo cài đặt các package cần thiết
- **Thư mục models**: gồm 2 phần chính, 1 là layers - chủ yếu chứa các block/module đơn vị nhỏ nhất có thể cấu thành model, 2 là các file chứa các custom model, các file model này sử dụng các block/module từ file layers
- **Thư mục notebooks**: Chứa các file notebooks, dùng để triển khai huấn luyện model, visualize kết quả...
- **Thư mục processing**: gồm 3 file, lần lượt là preprocessing - tiền xử lý, augmentation - tăng cường ảnh, postprocessing - hậu xử lý
- **Thư mục utils**: chứa các file/hàm chức năng hỗ trợ quá trình huấn luyện model
- **File parameters.yml**: chứa các siêu tham số, các path được khai báo mặc định
- **File requirements.txt**: chứa các package cần thiết trong quá trình huấn luyện, được gọi từ file trong thư mục init để cài đặt

4. Xử lý dữ liệu

- Dữ liệu: LiTS17 - https://www.sciencedirect.com/science/article/pii/S1361841522003085
- Giai đoạn 1: phân đoạn gan
    - Tiền xử lý: (chỉnh sửa file processing/preprocessing.py)
        - clip HU (hounsfield unit): -200:250
        - Normalize: zscore
        - resize: 128x128x128
    - Tăng cường dữ liệu: (chỉnh sửa file processing/augmentation.py)
        - random flip

        - random zoom
        - random rotate, scale
        - random shift intensity
        - random gaussian noise
        - random bias field
- Giai đoạn 2: phân đoạn u gan
    - Tiền xử lý: (chỉnh sửa file processing/preprocessing.py)
        - clip HU (hounsfield unit): 0:200
        - Normalize: zscore
        - resize: 3x256x256
    - Tăng cường dữ liệu: (chỉnh sửa file processing/augmentation.py)
        - random flip
        - random zoom
        - random rotate
        - random shift intensity
        - random gaussian noise
        - random bias field
- Cân bằng dữ liệu cho giai đoạn 2:
    - Tổng quan số lượng mẫu ở giai đoạn 2, sau khi cắt ảnh thành 3 slide với bước nhảy 2

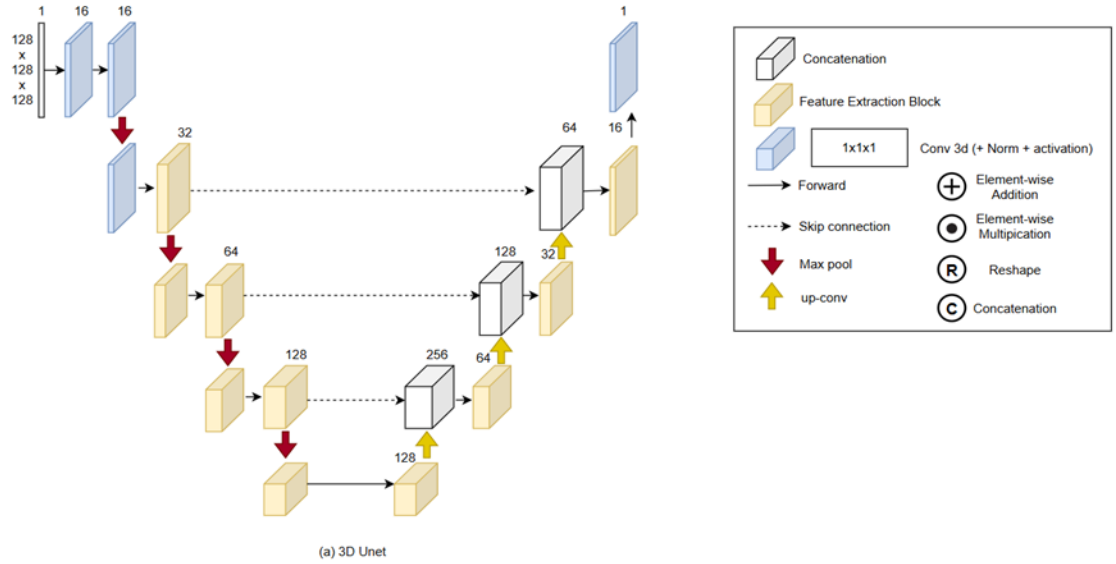
Tập dữ liệu	Tổng số mẫu	Mẫu chứa Gan	Tỷ lệ Gan (%)	Mẫu chứa Khối u	Tỷ lệ Khối u (%)
Huấn luyện	23380	7806	33,39%	3068	13,12%
Kiểm tra	5845	1892	32,37%	738	12,63%
Tổng cộng	29225	9698	33,18%	3.806	13,02%

        - Từ table trên ta có nhận xét là dữ liệu khối u chỉ chiếm ~13%, điều này dẫn đến việc dữ liệu cần phải được cân bằng trước khi huấn luyện nếu không kết quả sẽ bị lệch
    - Cân bằng dữ liệu (chỉ thực hiện trên tập huấn luyện):
        - undersampling - giảm mẫu:
            - Cách làm của bước này sẽ là lấy tổng số mẫu - (mẫu gan + mẫu u), sau khi ra được mẫu nền, mình chỉ lấy khoảng 30 - 40% số mẫu nền để kéo độ cân bằng về với gan và u, tránh bị lệch quá mức về nền
        - Weighted Sampling - Lấy mẫu có trọng số:
            - Thêm 1 cách nữa để tăng sự cân bằng cho dữ liệu là gán trọng số cao hơn (5) cho các mẫu có khối u, ngược lại thì các mẫu còn lại đều là (1)

5. Kiến trúc mô hình

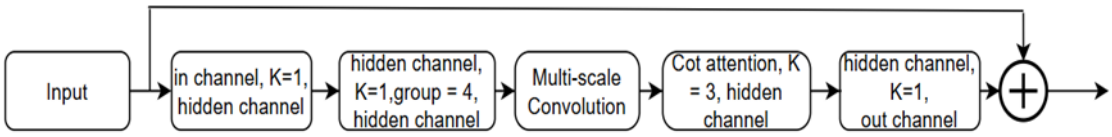
- Mô hình chung: UNet + feature extraction block (ResNeXt + Multi-Scale Convolution + CoTAttention)
    - Mô hình tổng quát (lấy mô hình 3d làm ví dụ)

- Unet

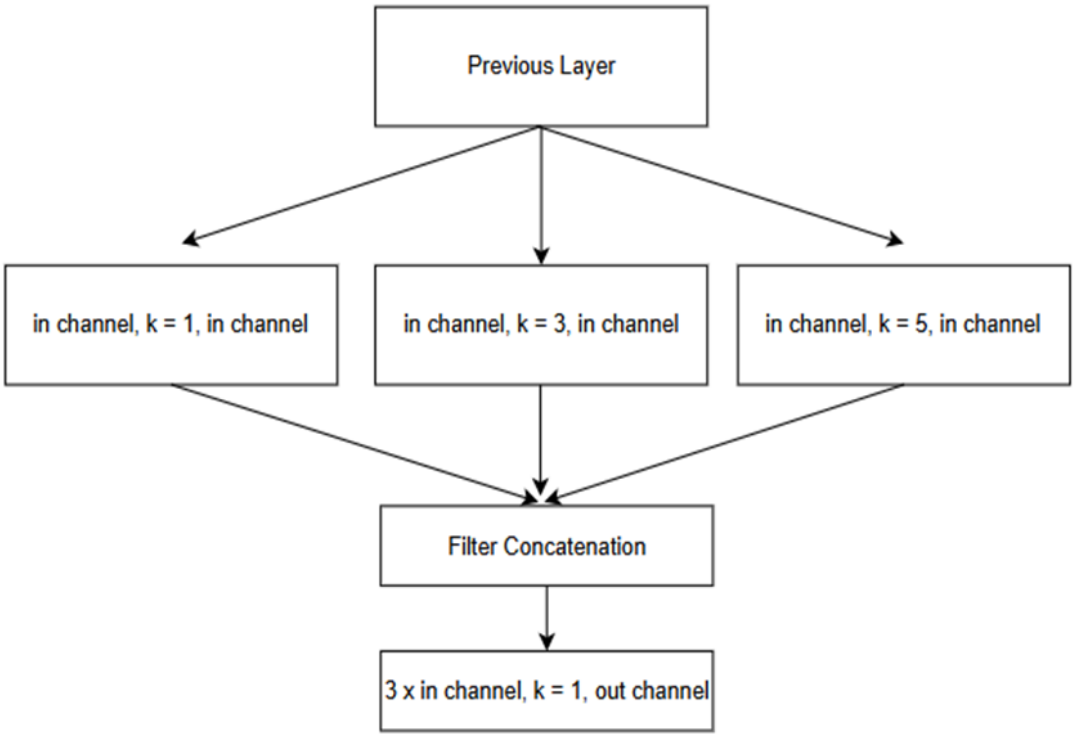


(a) 3D Unet

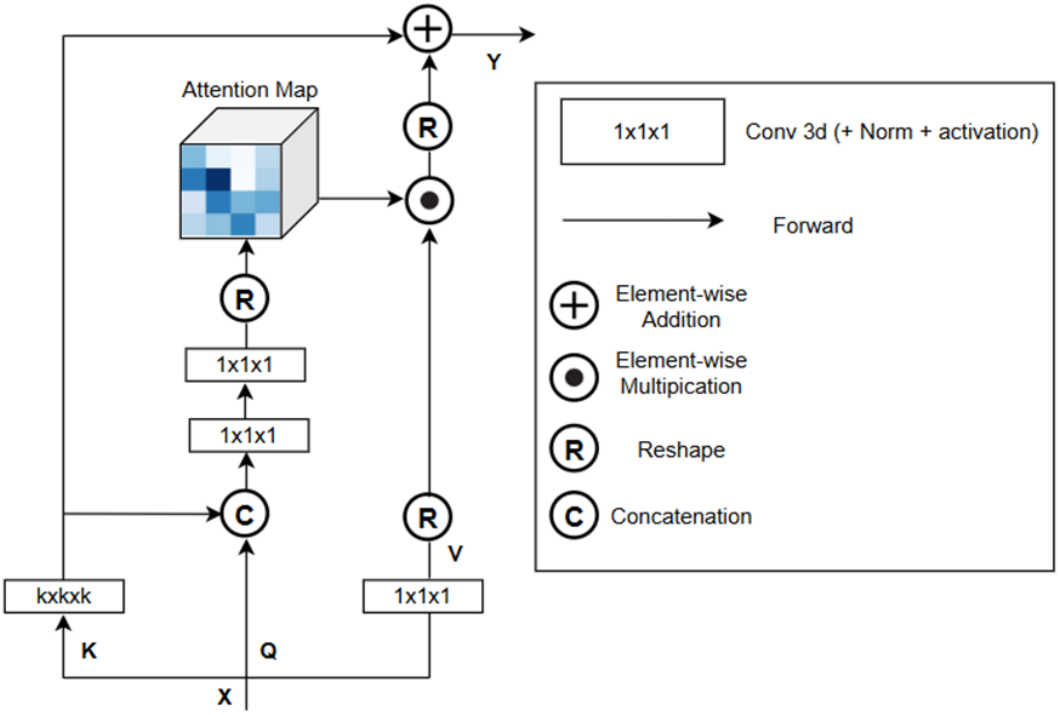
- Khối trích xuất đặc trưng



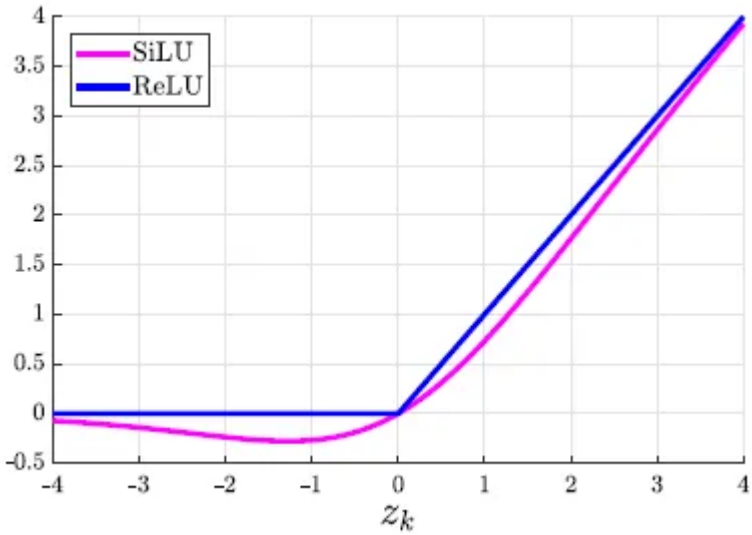
- Khối tích chập đa tỉ lệ



        - Khối attention



    - Giai đoạn 1: Sử dụng mô hình 3d
    - Giai đoạn 2: Sử dụng mô hình 2d
- Nhận xét thêm từ việc tune activation
    - Các block như ResNeXt, CNN khi kết hợp với Unet ở các papers gốc tác giả thường dùng ReLU làm activation chính, tuy nhiên khi mở rộng sang bài toán sử dụng ảnh CT, giá trị của ảnh CT có thể giao động từ -1000:1000. Khi tensor đầu vào qua lớp tích chập đầu tiên, nếu có sử dụng hàm ReLU thì các giá trị âm sẽ bị mất hết, điều này làm mất thông tin dữ liệu quan trọng.
    - Trong đề tài trên có sử dụng SiLU làm activation thay thế cho ReLU ở các layer cnn



    - Có thể sử dụng các activation khác có tính "mềm" với biên âm như Leaky ReLU, PReLU, ELU, SELU, GELU, SiLU

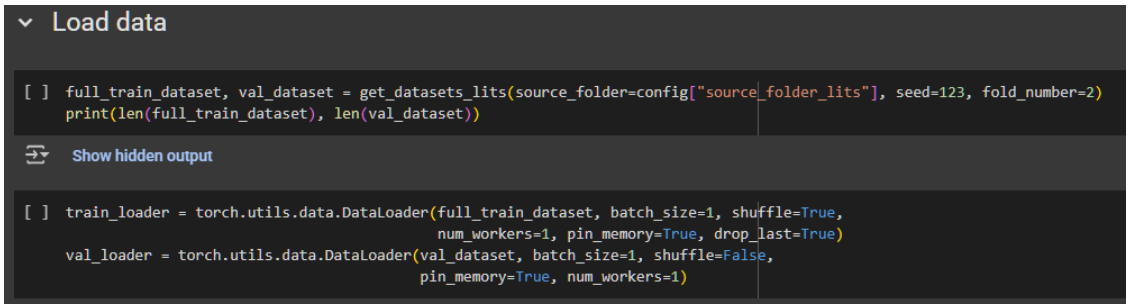
6. Huấn luyện & Đánh giá

- Loss: (chỉnh sửa file utils/metrics.py)
    - Giai đoạn 1:
        - Dice Loss + BCE, tỉ lệ 7/3

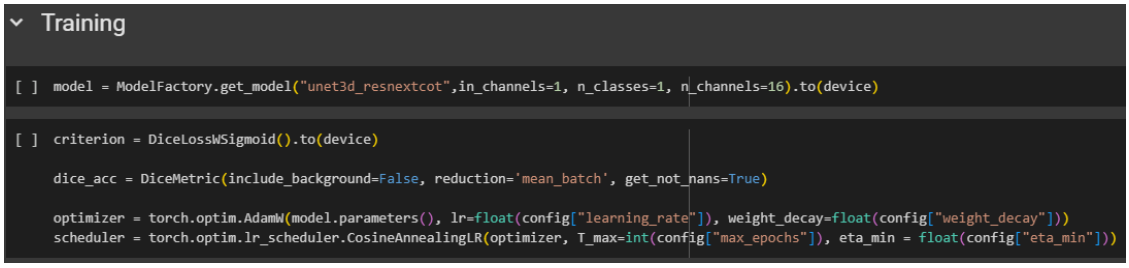
    - Giai đoạn 2:
        - Tversky Loss (alpha: 0.7, beta: 0.3) + BCE, tỉ lệ 7/3
- Metrics: Dice, IoU, Precision, Recall (chỉnh sửa file utils/metrics.py)
- Learning Rate: 1e-3
- Optimizer: AdamW; Weight Decay: 1e-5
- LR Scheduler: CosineAnnealingLR; eta_min: 1e-5

7. Cách chạy mô hình

- Giai đoạn 1:
    - Có 3 file notebooks: liver, liver_1, liver_2
    - File có chia các section con tương ứng với từng step:
        - Khai báo, cài package và tải dataset
        - load dataset



        - Visualize sample (option)
        - Training và đánh giá hiệu suất model + lưu weight vào google drive



```
[ ]  LOGGER.info("[TRAINER] Start TRAIN process...")

     (
         val_acc_max,
         best_epoch,
         dices_liver,
         loss_epochs,
         trains_epoch,
         ious_liver,
         precisions_liver,
         recalls_liver,
         time_tmp
     ) = trainer_stage1(
         model=model,
         train_loader=train_loader,
         val_loader=val_loader,
         optimizer=optimizer,
         loss_func=criterion,
         acc_func=dice_acc,
         scheduler=scheduler,
         batch_size=config["batch_size"],
         max_epochs = config["max_epochs"],
         start_epoch = config["start_epoch"],
         val_every=config["val_every"],
         path_save_model=config["path_save_model_state"],
         logger=LOGGER,
         save_model=True,
         post_fix="liver"
     )
```

- Sau khi train xong thì lưu kết quả train vào trong file json để tiện tracking và đánh giá về sau

```
[ ]  import json
     path_save_result = f"/content/gdrive/MyDrive/KLTN/code/result_model_{model.__class__.__name__}_liver.json"

     results = {
         "dice_liver": [float(x) for x in np.array(dices_liver, dtype=np.float32)],
         "loss": [float(x) for x in np.array(loss_epochs, dtype= np.float32)],
         "iou_liver": [float(x) for x in np.array(ious_liver, dtype=np.float32)],
         "precision_liver": [float(x) for x in np.array(precisions_liver, dtype=np.float32)],
         "recall_liver": [float(x) for x in np.array(recalls_liver, dtype=np.float32)],
         "best_epoch": best_epoch,
         "time_train": time_tmp
     }
     with open(path_save_result, "w") as f:
         json.dump(results, f, indent=4)
```

- Visualize kết quả mẫu từ weight đã lưu ở trên với tập test
- Plot chart kết quả train từ file json ở trên (optional)

- Giai đoạn 2:
  - Có 2 file notebooks: tumor, tumor_1
  - File có chia các section con tương ứng với từng step sau:
    - Khai báo, cài package và tải dataset
    - Load model với weight của giai đoạn 1, khai báo model cho giai đoạn 2

```
Model

[ ]  weight_path = "/content/gdrive/MyDrive/KLTN/code/weight/best_metric_model_UNet3DWResNeXtCoT_liver.pth"
     model_stage_1 = ModelFactory.get_model("unet3d_resnextcot",in_channels=1, n_classes=1, n_channels=16).to(device)
     model_stage_1.load_state_dict(torch.load(weight_path, map_location=device))

     <All keys matched successfully>

[ ]  model = ModelFactory.get_model("unet2d_resnextcot",in_channels=3, n_classes=1, n_channels=32).to(device)
```

    - dùng model 1 dự đoán 1 lượt toàn bộ dataset để lấy bounding box vùng gan của từng ảnh CT rồi lưu lại, sử dụng bounding box trên, loop toàn bộ ảnh CT và cắt theo bounding

box đó, cuối cùng chuyển ảnh CT3D thành ảnh 2D hoặc 2.5D giả định nhiều lát cắt thành chiều sâu của ảnh => tạo thành dữ liệu 2D mới

Prepare Dataset 2.5D

```
[ ] liver_masks_bbox = get_liver_mask_bbox_v1(source=config["source_folder_lits"], model_stage_1=model_stage_1, device=device)
```

```
[ ] import json
    path_save_result = f"/content/gdrive/MyDrive/KLTN/bbox.json"


    patient_ids, bboxs = liver_masks_bbox
    results = [
        {
            "patient_id": int(pid),
            "bbox": [int(x) for x in bbox]
        }
        for pid, bbox in zip(patient_ids, bboxs)
    ]
    with open(path_save_result, "w") as f:
        json.dump(results, f, indent=4)
```

```
[ ] import json

    path_save_result = f"/content/gdrive/MyDrive/KLTN/bbox.json"
    def load_json(json_path):
        with open(json_path, 'r') as f:
            return json.load(f)

    liver_masks_bbox = load_json(path_save_result)
```

```
[ ] convert_to_2D_dataset_v1(source=config["source_folder_lits"], bbox=liver_masks_bbox, slides=3, stride=2, save_dir=config["path_save_dataset_2D"]) # 3 slide of 3d and stride 2
```

- Load dataset mới

Load data

```
[ ] full_train_dataset, val_dataset = get_datasets_2d_v1(source_folder=config["path_save_dataset_2D"], seed=123, fold_number=2)
    print(len(full_train_dataset), len(val_dataset))
```

Show hidden output

```
[ ] train_loader = torch.utils.data.DataLoader(full_train_dataset, batch_size=4, shuffle=True,
                                               num_workers=1, pin_memory=True, drop_last=True)
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=32, shuffle=False,
                                             pin_memory=True, num_workers=2, prefetch_factor=4)
```

- Rebalancing train dataset (undersampling + weighted sampling)

Custom train dataset

```
[ ] import random

    indices_liver = []
    indices_tumor = []
    indices_background = []

    for i in range(len(full_train_dataset)):
        sample = full_train_dataset[i]
        label = sample["label"]
        liver = sample["liver_mask"]

        if label.sum() > 0:
            indices_tumor.append(i)
        elif liver.sum() > 0:
            indices_liver.append(i)
        else:
            indices_background.append(i)
    random.seed(42)
    selected_background = random.sample(indices_background, k=int(len(indices_background) * 0.4))
    final_indices = indices_tumor + indices_liver + selected_background
```

```
[ ] from torch.utils.data import Subset
    balanced_dataset = Subset(full_train_dataset, final_indices)
```

WeightedRandomSampler

```
[ ] import torch
    from torch.utils.data import WeightedRandomSampler

    weights = []
    for i in range(len(balanced_dataset)):
        sample = balanced_dataset[i]
        has_tumor = sample["label"].sum() > 0
        weights.append(5.0 if has_tumor else 1.0)
```

```
[ ]     sampler = WeightedRandomSampler(weights, num_samples=len(balanced_dataset), replacement=True)
        train_loader = torch.utils.data.DataLoader(balanced_dataset, batch_size=16, sampler=sampler, shuffle=False, pin_memory=True, num_workers=2, drop_last=True, prefetch_factor=4)
```

- Training và đánh giá hiệu suất model + lưu weight vào google drive

Training

```
[ ] criterion = TverskyLossWSigmoid().to(device)

    dice_acc = DiceMetric(include_background=False, reduction='mean_batch', get_not_nans=True)

    optimizer = torch.optim.AdamW(model.parameters(), lr=float(config["learning_rate"]), weight_decay=float(config["weight_decay"]))
    scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=int(int(config["max_epochs"]) * 0.75), eta_min = float(config["eta_min"]))
```

```
[ ] LOGGER.info("[TRAINER] Start TRAIN process...")

    (
        val_acc_max,
        best_epoch,
        dices_tumor,
        loss_epochs,
        trains_epoch,
        ious_tumor,
        precisions_tumor,
        recalls_tumor,
        time_tmp
    ) = trainer_stage2(
        model=model,
        train_loader=train_loader,
        val_loader=val_loader,
        optimizer=optimizer,
        loss_func=criterion,
        acc_func=dice_acc,
        scheduler=scheduler,
        batch_size=config["batch_size"],
        max_epochs = int(int(config["max_epochs"]) * 0.75),  # reducing epochs to 75% for faster training
        start_epoch = config["start_epoch"],
        val_every=config["val_every"],
        path_save_model=config["path_save_model_state"],
        logger=LOGGER,
        save_model=True,
        post_fix="tumor_2d"
    )
```

- Sau khi train xong thì lưu kết quả train vào trong file json để tiện tracking và đánh giá về sau

```
[ ] import json
    path_save_result = f"/content/gdrive/MyDrive/KLTN/code/result_model_{model.__class__.__name__}_tumor_2d.json"

    results = {
        "dice_tumor": [float(x) for x in np.array(dices_tumor, dtype=np.float32)],
        "loss": [float(x) for x in np.array(loss_epochs, dtype= np.float32)],
        "iou_tumor": [float(x) for x in np.array(ious_tumor, dtype=np.float32)],
        "precision_tumor": [float(x) for x in np.array(precisions_tumor, dtype=np.float32)],
        "recall_tumor": [float(x) for x in np.array(recalls_tumor, dtype=np.float32)],
        "best_epoch": best_epoch,
        "time_train": time_tmp
    }
    with open(path_save_result, "w") as f:
        json.dump(results, f, indent=4)
```

- Visualize kết quả mẫu từ weight đã lưu ở trên với tập test
- Plot chart kết quả train từ file json ở trên (optional)

8. Hạn chế & Hướng mở rộng

- Chưa đánh giá cross-dataset
- Chưa áp dụng mô hình SOTA mạnh cho bài toán segmentation (vd như InternImage), có thể áp dụng các khối attention mechanism tối ưu hơn