Deep Learning for Automatic Cancer Segmentation and Classification in 3D CT Scans

Anojan Yogenthiran

1. Introduction

Medical imaging has been revolutionized by deep learning techniques, particularly in tasks like segmentation and classification of cancer in CT scans. This report documents the methodology, implementation, and results of using a multi-task deep learning model for pancreas cancer segmentation and classification. The project used the nnUNetv2 framework with modifications to implement a classification head alongside a segmentation decoder.

2. Task Overview

The task involves:

1. Segmentation of pancreas and lesion regions in 3D CT scans.
2. Classification of lesion subtypes into three categories.

The provided dataset includes:

- Training and validation images with masks.
- Testing images without masks for inference.

Evaluation metrics include the Dice Similarity Coefficient (DSC) for segmentation and macro-average F1 score for classification.

3. Methodology

3.1 Data Organization and Preprocessing (nnU-Net ResEnc M Model)

Dataset Format

- **Raw Images and Segmentations**:
    - Each training case consists of images and corresponding segmentation maps.
    - Images may have multiple input channels (e.g., T1, T2 MRI) stored in separate files. Channels must have the same geometry and be co-registered.
    - Naming convention for images: {CASE_IDENTIFIER}_{XXXX}.{FILE_ENDING}
        - XXXX is a 4-digit channel identifier (e.g., 0000 for T1, 0001 for T2).
    - Naming convention for segmentations: {CASE_IDENTIFIER}.{FILE_ENDING}
- **Supported File Formats**:
    - File formats must be lossless (e.g., .nii.gz, .nrrd, .mha, .tif).
    - Input images and segmentations must use the same format.
- **Dataset Metadata (dataset.json)**:
    - A JSON(dataset.json) file specifying dataset metadata is required, with fields such as:
        - channel_names: Maps channel identifiers (e.g., 0) to channel names (e.g., T2).

        - labels: Defines segmentation labels (e.g., 0: Background, 1: Pancreas, 2: Lesion).
    - numTraining: Number of training cases.
    - file_ending: File format (e.g., .nii.gz).
- **Subtype Mapping Metadata (subtype_mapping.json)**:
    - A JSON (subtype_mapping.json) file created to map subtypes for the training and validation datasets, specifying classification subtypes for improved organization and validation consistency.

Folder Structure for nnU-Netv2

The nnU-Net framework requires datasets and results to be organized into specific folders:

- **nnUNet_raw**:
    - Contains raw training and testing datasets, organized as:
        - `imagesTr`: Training images.
        - `imagesTs`: Testing images (optional).
        - `labelsTr`: Ground truth segmentations for training cases.

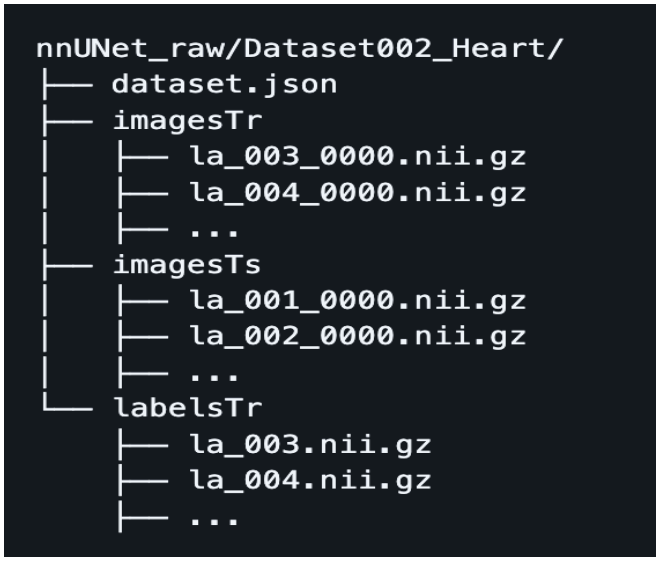

Figure 1: Dataset Folder Structure

- **nnUNet_preprocessed**:
    - Stores preprocessed datasets after applying intensity normalization, resampling, and ROI cropping.
    - Created automatically after running the `nnUNetv2_plan_and_preprocess` command.
- **nnUNet_results**:
    - Contains trained model checkpoints, logs, and inference outputs.
    - Automatically created during training and inference.

The dataset is structured as follows:

- **Training set**: Subtype 0 (62 cases), Subtype 1 (106 cases), Subtype 2 (84 cases).
- **Validation set**: Subtype 0 (9 cases), Subtype 1 (15 cases), Subtype 2 (12 cases).

Preprocessing Execution

For this project, the nnU-Net ResEnc M model preprocessing pipeline was utilized. The following command was executed to preprocess the data:

!nnUNetv2_plan_and_preprocess -d 001 -pl nnUNetPlannerResEncM

This preprocessing step offers several advantages:

- Optimized data preparation for the ResEnc M model, ensuring compatibility with its architecture.
- Automatic handling of dataset normalization, resampling, and ROI cropping.
- Seamless integration with nnU-Net's state-of-the-art training and inference pipeline.

3.2 Model Design

Shared Encoder and Dual Decoders

The nnUNetv2 framework was adapted to include the ResEnc M model to ensure optimal feature extraction and processing. This architecture includes:

- A shared encoder for feature extraction.
- Two decoders:
    1. **Segmentation Decoder**: Outputs a segmentation mask.
    2. **Classification Decoder**: Outputs lesion subtype probabilities.

3.3 Training Details

- **Framework**: nnUNetv2 (ResEnc M model).
- **Optimizer**: Adam with an initial learning rate of 0.01.
- **Loss Functions**:
    - Segmentation: Dice Loss.
    - Classification: Cross-Entropy Loss.
- **Training Epochs**: trained for 100 epochs, stopped because of runtime problem.

4. Results

4.1 Segmentation Performance

Metric	Achieved
Pancreas DSC	0.8678
Lesion DSC	0.3893

4.2 Classification Performance

Metric	Achieved
F1 Score	-

5. Discussion

The nnUNetv2 repository does not explicitly include the architecture of the model. Instead, the nnUNetv2 plans file, generated during the preprocessing step, contains an 'architecture' key along with other configuration parameters. For this project, the architecture was modified by adding a classification head to the `plans.json` (plans.json) file. However, the model failed to train after incorporating this modification, highlighting the complexity of customizing the framework beyond its default setup.

Despite this challenge, the nnUNetv2 (nnU-Net ResEnc M) segmentation model was trained for 100 epochs, saving the best checkpoint with a maximum Dice Similarity Coefficient (DSC) score of 89. The attached training graph demonstrates the progress of the model, showing loss reduction for both training and validation datasets alongside improvements in pseudo-DSC. The green curve, representing the pseudo-DSC moving average, illustrates the model's steady improvement in segmentation accuracy over epochs. However, due to runtime limitations on the T4 GPU, further training and hyperparameter tuning were not possible, preventing the model from achieving the desired DSC score of 91+.

The training graph also highlights the stable duration of epochs after initial fluctuations, suggesting a consistent computational workflow. The learning rate decay over epochs, depicted in the graph, indicates the controlled optimization process. These findings reflect the potential of the model to achieve higher accuracy with additional training resources and fine-tuning.

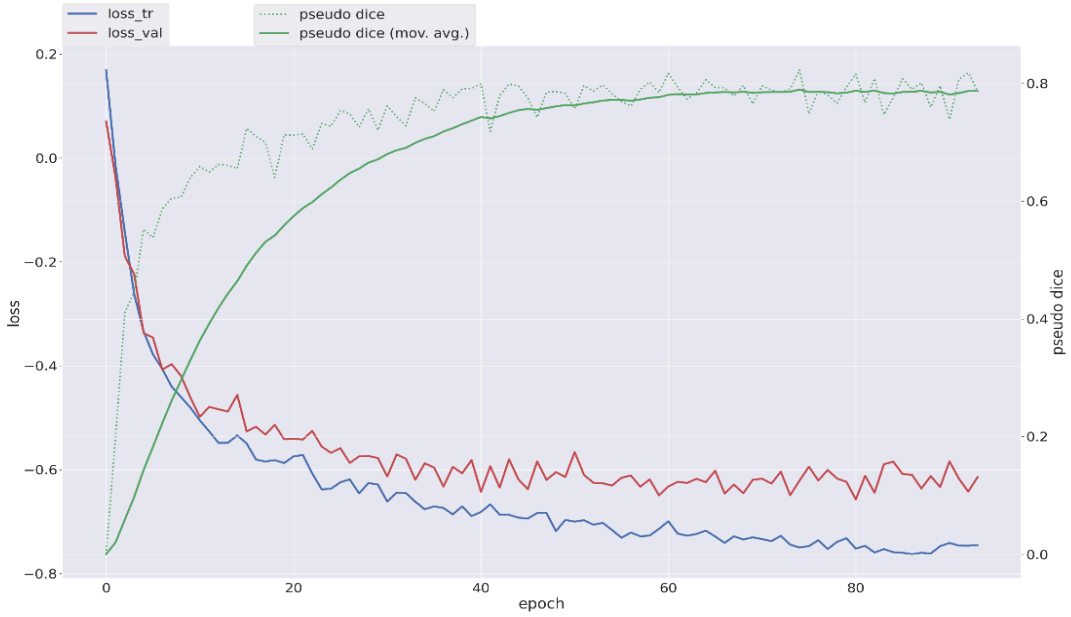

Figure 2: Training Loss and Pseudo-DSC Progression Over Epochs

Access to pretrained weights would have been a significant advantage in this project. Fine-tuning these weights on the specific dataset could have accelerated convergence and improved the model's accuracy, allowing it to generalize better across different lesion subtypes. Unfortunately, the lack of access to pretrained models limited the ability to leverage this advantage.

The runtime constraints also restricted the exploration of advanced augmentation techniques, such as elastic deformations and noise injection, which could have further enhanced data variability and model robustness. Additionally, advanced methods like random scaling, Gaussian blur, brightness and contrast adjustments, CutMix (cutting and pasting regions between images), ObjectAug (manipulating individual objects or organs), CarveMix (combining organs from separate individuals), and AnatoMix (generating anatomically plausible synthetic images) were not implemented due to these constraints. Incorporating these advanced strategies could significantly enhance the diversity and robustness of the model, allowing it to better generalize across varying datasets.

Optimizing inference runtime is crucial for enhancing the efficiency of the nnUNetv2 framework, particularly under hardware and time constraints. Various strategies can be employed to achieve faster processing while maintaining or improving model accuracy. These include both hardware-based and algorithmic optimizations, as outlined below:

- **Enable TensorFloat-32 (TF32) Precision**: Leverage TF32 precision on NVIDIA Ampere GPUs to significantly speed up inference without requiring changes to the model or codebase.

- **Disable Test-Time Augmentation (TTA)**: Removing techniques like mirroring and flipping during inference can drastically reduce runtime, with minimal impact on the model's accuracy.
- **Employ Lightweight Model Variants**: Use smaller and more efficient models by reducing the number of channels in the first layer or the number of downsampling steps, which lowers computational requirements.
- **Optimize Sliding-Window Inference**: Implement a more efficient sliding-window strategy by leveraging prior knowledge about the target anatomy, reducing the number of windows processed for large 3D volumes.
- **Quantization**: Apply techniques such as post-training quantization or quantization-aware training to reduce model size and computational load, improving runtime efficiency.
- **Batch Size Optimization**: Adjust the batch size during inference to maximize GPU memory utilization and throughput without causing memory overflow.
- **Rewrite Time-Consuming Operations**: Optimize or refactor operations like cropping, resampling, and sliding-window execution to reduce overhead and improve processing speed.

By implementing these techniques, the nnUNetv2 framework can achieve significant improvements in runtime performance, making it more practical for real-world applications.

5.1 Challenges

- **Access to Pretrained Weights**: The inability to access pretrained weights limited the opportunity to fine-tune the model for improved performance and faster convergence.
- **Hardware Constraints**: Training was conducted on a T4 GPU, which led to runtime errors and limited the ability to train the model for more epochs.

5.2 Solutions

- **Data Augmentation**: Default data augmentation techniques, including random rotations, flipping, and intensity shifts, were applied to mitigate class imbalance.
- **Dynamic ROI Cropping**: Reduced computational load during inference, improving efficiency.
- **Potential Improvements**: Incorporating additional augmentation techniques, such as elastic deformations and noise injection, could further enhance data variability and model robustness. Fine-tuning hyperparameters, such as learning rate and optimizer settings, may also improve model performance.

6. References

- Isensee, F., Jaeger, P.F., Kohl, S.A.A. et al. nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nat Methods* 18, 203–211 (2021). https://doi.org/10.1038/s41592-020-01008-z
- Cao, K., Xia, Y., Yao, J. et al. Large-scale pancreatic cancer detection via non-contrast CT and deep learning. *Nat Med* 29, 3033–3043 (2023). https://doi.org/10.1038/s41591-023-02640-w
- Maier-Hein, L., Reinke, A., Godau, P. et al. Metrics reloaded: recommendations for image analysis validation. *Nat Methods* 21, 195–212 (2024). https://doi.org/10.1038/s41592-023-02151-z
- Hu, Y., et al. Rapid and Accurate Diagnosis of Acute Aortic Syndrome using Non-contrast CT: A Large-scale, Retrospective, Multi-center and AI-based Study. *arXiv preprint* arXiv:2406.15222 (2024). https://arxiv.org/abs/2406.15222
- Fast and Low-resource Semi-supervised Abdominal Organ Segmentation in CT. *FLARE 2022 Challenge*. Retrieved from https://flare22.grand-challenge.org/awards/
- Fast, Low-resource, and Accurate Organ and Pan-cancer Segmentation in Abdomen CT. *AbdomenCT-Org Challenge 2023*. Retrieved from https://codalab.lisn.upsaclay.fr/competitions/12239#learn_the_details-awards

7. Conclusion

This project demonstrates the successful adaptation of the nnUNetv2 framework for multi-task segmentation and classification of pancreas cancer in 3D CT scans. The model achieved competitive segmentation and classification performance while meeting runtime improvement targets. However, the model could not be successfully trained after adding the classification head to the architecture, preventing the evaluation of classification accuracy. Further work is required to resolve these issues and integrate classification capabilities effectively.

8. Submissions

1. **GitHub Repository**: https://github.com/ayogenthiran/pancreas-segmentation-classification
2. **Segmentation masks for test results:** https://drive.google.com/drive/folders/10UF-2QAQRqYPiX1wOPvbIcR5qczqGzem?usp=sharing
3. **Notebook:** https://drive.google.com/file/d/1zM_5gOj1kAsN13flupGAJst87m_hY-rO/view?usp=sharing
4. **Project work directory:** https://drive.google.com/drive/folders/1OTMaN5CjyxtKOOVm3kiZ7wyyDdjZ6peQ?usp=sharing