Deep Learning Course Final Project

Attention U-Net:

Learning Where to Look for the Pancreas

https://arxiv.org/abs/1804.03999

Adi Froimovich

Abstract

This project presents a re-implementation and modification of the Attention U-Net architecture tailored specifically for pancreas segmentation tasks, with a focus on utilizing the CT-82 dataset.

Several key modifications, notably the integration of a larger U-Net architecture, the utilization of innovative attention mechanisms, and additional enhancements.

We provide thorough demonstrations of the model's performance, including evaluations of both baseline and modified versions.

Through comprehensive evaluation, we showcase the efficacy of our approach trying to improve pancreas segmentation metrics.

Additionally, we make our source code openly available to facilitate reproducibility and further advancement in the field of medical image segmentation.

Implementation

Main libraries

In our project, the main libraries utilized are PyTorch and MONAI (Medical Open Network for AI). MONAI is specifically tailored for medical image analysis tasks, offering a wide range of functionalities and utilities to streamline deep learning workflows in healthcare.

Here's a deeper explanation of the MONAI framework and the packages imported in our project:

MONAI Framework:

- **Purpose**: MONAI aims to simplify and accelerate the development of deep learning models for medical image analysis by providing a comprehensive set of tools, data structures, and pre-built components tailored to the unique requirements of healthcare imaging.
- **Features**:
    - **Efficient Data Handling**: MONAI offers efficient data handling capabilities, including data loading, transformation pipelines, and dataset management, optimized for large-scale medical imaging datasets.
    - **Standardized Transformations**: MONAI provides a rich library of standard transformations tailored for medical images, facilitating data preprocessing, augmentation, and normalization.
    - **Model Evaluation and Training**: MONAI includes utilities for model evaluation, training loop management, and integration with PyTorch, allowing seamless integration into existing PyTorch workflows.
    - **Metrics and Loss Functions**: MONAI offers a variety of common evaluation metrics and loss functions specifically designed for medical image analysis tasks, simplifying model evaluation and optimization.
    - **Inference Techniques**: MONAI provides specialized inference techniques optimized for medical image processing, such as sliding window inference and ensemble prediction strategies.
    - **Community Support**: MONAI has a vibrant community of researchers, developers, and healthcare professionals contributing to its development, documentation, and ecosystem expansion.

Data_transforms.py

train_transforms:

we performed several preprocessing steps on the training images.

1. **Normalize Intensity**: We normalized the intensity values of the images to ensure consistent and standardized input data across different scans. This step helps in reducing the impact of variations in image intensity on the training process.
2. **Crop Foreground**: We cropped the foreground of the images, focusing only on the relevant anatomical structures such as the pancreas. This helps in removing unnecessary background noise and clutter, improving the model's ability to discern important features.
3. **Orientation Correction**: We standardized the orientation of the images to a common coordinate system, specified by the RAS (Right, Anterior, Superior) axcodes. This step ensures uniformity in the orientation of the input data, facilitating easier interpretation and analysis.
4. **Spacing Adjustment**: We adjusted the spacing between voxels in the images to ensure a consistent resolution across different scans. This is important for maintaining spatial relationships between anatomical structures and for achieving accurate segmentation results.
5. **Spatial Padding**: We padded the images to a uniform spatial size of (96, 96, 96). This ensures that all input images have the same dimensions, which is necessary for feeding them into the neural network model during training. Additionally, padding helps in preventing loss of information at the boundaries of the images.

In our training pipeline, we applied transformations to augment the dataset and improve model generalization. Specifically, we utilized affine transformations and random crops. Affine transformations were employed to introduce variations in rotation and scaling of the images, enhancing the model's ability to handle different orientations and sizes of pancreas structures. Random crops were used to extract patches from the images, providing diverse perspectives and aiding the model in learning from different regions of interest. These transformations contribute to robust training by exposing the model to a variety of scenarios commonly encountered in medical imaging datasets.

val_transforms

Since these images are used for testing, we aim to keep the transformations relatively simple to avoid introducing unnecessary complexity or altering the data in ways that could affect the evaluation results.

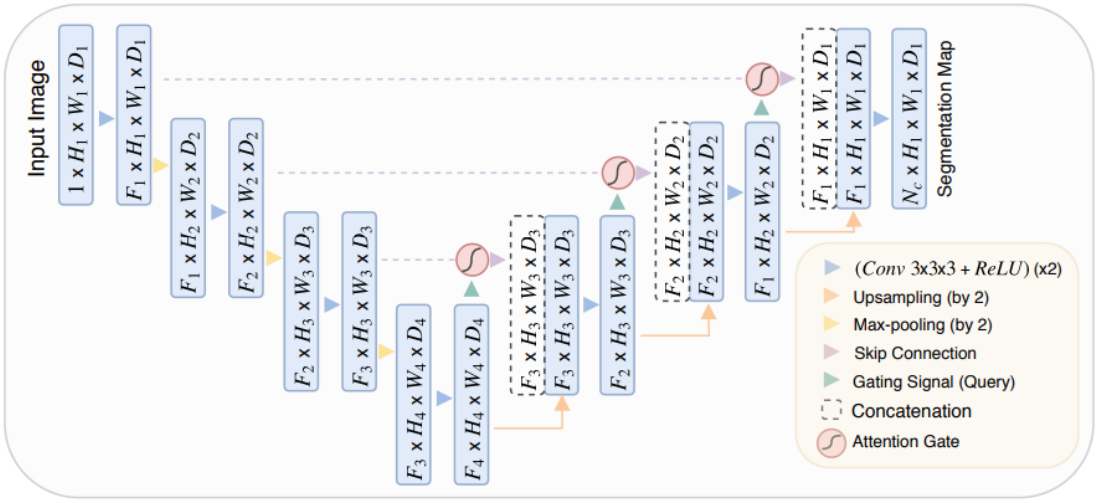
Here's an explanation of the transformations and their purpose:

1. **Load Image:** This step loads the image and its corresponding label into memory.
2. **Ensure Channel First:** Ensures that the channel dimension is the first dimension in the image and label tensors. This is a standard format required by many deep learning frameworks.
3. **Normalize Intensity:** Normalizes the intensity values of the images to maintain consistent and standardized input across different scans. Normalization aids in reducing the impact of variations in image intensity on the model's predictions.
4. **Crop Foreground:** Similar to the training set, this step crops the foreground of the images to focus only on the relevant anatomical structures, such as the pancreas. Removing unnecessary background noise helps in evaluating the model's performance on the relevant regions of interest.
5. **Orientation Correction:** Standardizes the orientation of the images to a common coordinate system, specified by the RAS (Right, Anterior, Superior) axcodes. This ensures uniformity in the orientation of the test data, facilitating consistent evaluation.
6. **Spacing Adjustment:** Adjusts the spacing between voxels in the images to ensure a consistent resolution across different scans. This is important for maintaining spatial relationships between anatomical structures during evaluation.

By keeping the transformations simple and consistent with those applied to the training data, we ensure that the test images are prepared appropriately for evaluation while minimizing any potential bias introduced by the preprocessing steps.

Model.py

The implemented network's diagram.



Attention_UNET Class:

1. **Initialization**:

   Upon instantiation, the **AttentionUNET** class initializes its parameters and layers. The constructor accepts parameters such as the number of input channels (in_channels), the number of output channels (out_channels), and the level of deep supervision (n_deep_supervision).

2. **Convolutional Blocks**:

   The architecture starts with a series of convolutional blocks (ConvBlock) that learn hierarchical representations of the input data.

   Initialization:

   - It takes the number of input channels (in_channels) and output channels (out_channels).
   - Constructs a sequence of operations including:
     - 3D convolutional layers with a 3x3x3 kernel, stride of 1, and padding of 1 for feature extraction.
     - Batch normalization layers for stabilizing training.
     - ReLU activation functions for introducing non-linearity.

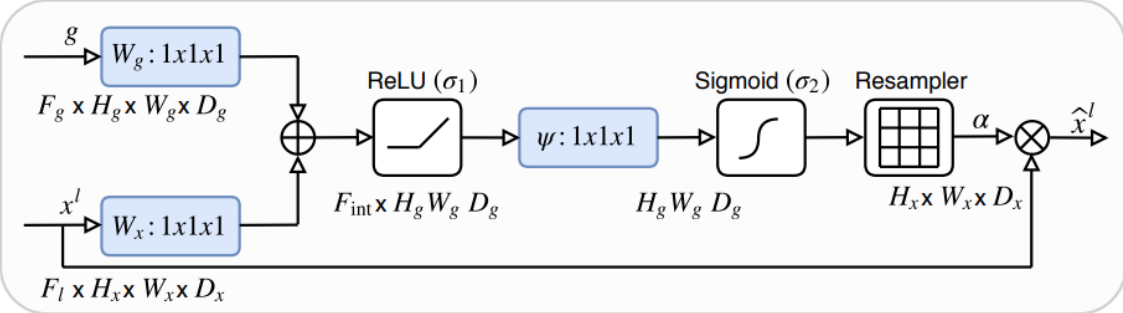
- Encapsulates these operations within a nn.Sequential module for ease of use.

3. **Downsampling Layers**:

   After each convolutional block, downsampling operations are performed to reduce the spatial dimensions of the feature maps while increasing their depth. These downsampling layers (MaxPool3d (kernel_size=2,stride=2), and MaxPool2d) help capture hierarchical features at different scales.

4. **Attention Mechanisms**:

   The **AttentionBlock** class implements an attention mechanism, specifically additive soft attention, within a grid attention block framework.



In order to try another attention mechanism, in one of our experiments we tried a new approach over the regular additive attention, utilizing convolution as a method for weighted aggregation of input (x) and gating signal (g). After applying 1x1x1 convolutions on X and g, we stack them together and instead of regular addition we apply additional 1x1x1 convolution to aggregate over the channels (can be used by setting conv_mapping to True, while False will use the original attention implementation)
The use of convolution is motivated by the intention to make a more informed selection of feature mixtures, incorporating both the

semantic information from the gating signal along with the spatial information coming from the input x

Here's a breakdown of its functionality:

1. **Initialization**:
    - The **AttentionBlock** class is initialized with parameters including the number of input channels (**in_channels**), gating channels (**gating_channels**), and intermediate channels (**inter_channels**).
    - It defines convolutional layers and activation functions used in the attention mechanism.
2. **Forward Method**:
    - During forward pass, the attention block takes two input feature maps: **x** and **g** (gating signal).
    - The **x** feature map undergoes a strided convolution to adjust its spatial dimensions to match that of **g**.
    - The **g** feature map is processed through a 1x1 convolution to align its dimensions with **x**.
    - Both processed feature maps are element-wise summed and passed through a ReLU activation function.
    - Subsequently, the resulting tensor is passed through another convolutional layer followed by a sigmoid activation function to generate attention coefficients.
    - These attention coefficients are upsampled to the original spatial dimensions of **x** using trilinear interpolation.
    - Finally, the attention coefficients are applied to **x** element-wise, scaling it based on relevance.
3. **Attention Gate Operation**:
    - The attention gate selectively focuses on informative regions of **x**, guided by the gating signal **g**.
    - It dynamically adjusts the importance of different spatial locations in **x** based on contextual information from **g**.

- Attention coefficients are learned during training, allowing the network to highlight relevant features and suppress irrelevant ones.
- The gate is trainable via backpropagation, enabling it to improve its focus over the course of training.

5. **Upsampling Layers**:

   Following the attention blocks, upsampling layers (UpBlock) are utilized to restore the spatial dimensions of the feature maps by scale factor of 2, while merging information from lower-resolution layers. This helps in recovering spatial details lost during downsampling.

6. **DeepSuperHead:**
   The **DeepSuperHead** class serves as the output layer for deep supervision in the model. Its purpose is to process one of the UNet levels' outputs by applying convolution followed by upsampling to match the spatial dimensions of the final image. This processing occurs before stacking the layers and calculating the deep supervision loss. The key functionality of this class lies in its **forward** method, which takes an input tensor and returns the processed output tensor.

7. **Deep Supervision Loss**:

   The **deep_supervision_loss** function calculates the loss for intermediate predictions in deep supervision. Here's a brief overview:

   1. If there are multiple predictions (**preds**) at different levels of the network, the function splits them.
   2. It assigns weights to each prediction based on its level, ensuring proper weighting – higher as it gets closer to the top.
   3. The function computes the loss for each prediction using the provided loss function.
   4. It combines the weighted losses to obtain the final deep supervision loss.

8. **Initialization of Weights**:

   Before training begins, the weights of the convolutional layers and batch normalization layers are initialized using the Kaiming and Normal initialization methods, respectively. This helps in stabilizing training and improving convergence.

training.py

main:

The main function orchestrates the training process of the model and monitors its performance.
Here are the key points about the **main** function

1. **Initialization**:
   - Initializes training and validation datasets (train_ds and val_ds) along with their respective data loaders (train_loader and val_loader).
   - Sets up the model (AttentionUNET or DeeperAttentionUNET), loss function (DiceLoss or DiceCELoss), optimizer (Adam), and evaluation metric.
   - Determines the device to use (GPU if available, otherwise CPU).
2. **Training Loop**:
   - Iterates over each epoch, printing the current epoch number.
   - Sets the model to training mode.
   - Iterates over batches in the training dataset, performs forward pass, calculates loss, and updates model parameters via backpropagation.
   - Computes and prints the average loss for the epoch.
   - Plots and saves a graph of the training loss curve after every epoch.
3. **Validation Interval**:
   - Executes validation after a specified interval of epochs (val_interval).
   - Sets the model to evaluation mode.
   - Evaluates the model on the validation dataset, computing the Dice metric for segmentation accuracy.
   - Saves the model checkpoint if the current Dice metric is the best observed so far.

4. **Checkpointing**:

- Optionally loads a checkpointed model and optimizer state if **load** parameter is set to **True**.
- Saves the model state, optimizer state, and other relevant training parameters to a checkpoint file after every epoch.

5. **Visualization**:
    - Displays a graph of the training loss curve using matplotlib after each epoch.
    - Saves the loss curve plot to the specified loss_curve_path.

Inference.py

In inference.py, the trained model is loaded and utilized to make predictions on validation data. By visualizing these predictions alongside the ground truth labels, the script enables a thorough qualitative evaluation of the model's performance. Here's an adjusted explanation for the updated code:

1. **Calculate Precision and Recall:**
    - The function calculate_precision_recall computes precision and recall metrics for a given prediction and ground truth labels. These metrics are essential for assessing the model's performance on positive labels.
2. **Validation:**
    - The validation function evaluates the model on the validation dataset. It computes various metrics including Dice coefficient and surface distance metrics. Additionally, it calculates precision and recall, providing insights into the model's performance beyond segmentation accuracy.
    - The function utilizes sliding window inference for predicting on large images, aggregates metrics, and optionally saves predicted images for qualitative assessment.

3. **Inference:**
    - The inference function orchestrates the inference process using the trained model. It loads the test data and the checkpointed model. Then, it performs inference using the validation function and reports various metrics including mean test Dice coefficient, mean surface distance, mean precision, mean recall, and the best validation Dice coefficient observed during training.
4. **Execution:**
    - In the __main__ block, the script sets up necessary paths and directories for saving inference outputs. It then calls the inference function with the provided data split and checkpoint paths.

utils.py

1. **save_checkpoint**:
    - This function saves the current state of the training process, including model weights, optimizer state, last epoch, best metric achieved, and epoch loss, The **checkpoint** dictionary is then saved to a file specified by **cp_path** using the **torch.save** function.
2. **load_checkpoint**:
    - This function loads a previously saved checkpoint from the file specified by **cp_path**, It retrieves the saved dictionary from the checkpoint file using **torch.load**.
    - The function then extracts and loads the model's state dictionary, optimizer's state dictionary, last epoch, and best metric from the loaded checkpoint.
3. **get_files**
   the function creates or loads data splits stored in a JSON file.
   If the file doesn't exist, it creates train-validation-test splits based on input image paths and labels.
   The splits are saved to the file.
   The function returns the training, validation, and test sets.

Usage instruction:

Use the provided requirements file to create a proper environment.

For training a model, first download the data from
https://www.cancerimagingarchive.net/collection/pancreas-ct/

Use the provided Dicom to nifti function:
seg_attention_unet/preprocess_utils/dicom_to_nifti.py

Then use the training.py file, provide root_dir containing a data folder with images and labels folders.

Choose a name for the experiment, set batch size, max epochs, learning rate etc.

Choose whether to use the original network or the deeper one by setting deeper_net to true or false accordingly in the main function. Use the CONFIG.py for the configuration of the network - number of channels, number of deep supervision heads and whether to use the modified attention by switching conv_mapping on or off. (The reason for this configuration file is to get a unified configuration for the parameters shared by training and inference).

Inference:
Set the right configuration for the trained model in the CONFIG file if needed, as well as the deeper_net parameter according to the model applied.

Choose the root_dir and the experiment_dir, our trained models can be downloaded from:
https://drive.google.com/drive/folders/1aS2IY39i4d7PGYNwwAU0R62AOjsrkSaU?usp=sharing

Experimentation

Experimental setup:

Datasets: CT-82 (the only public available Dataset). It's worth noting that despite its name implying 82 pictures, the CT-82 dataset actually contains 80 pictures.

We used the same train-test split ratio as in the paper:
Train size = 0.75(51 train, 9 validation).
Test size = 0.25(20).
We used the same split for all the models, ensuring proper comparison. (All models were training on the same train set, evaluated on the same validation set and same test set)

We trained all our models for 600 epochs, as we tried to fit in the timeline several experiments.
All set ups were independent, no changes were combined, as we wanted to separate the effect of each change to be able to gather insights specifically on each one.

Hyperparameters:

- Epochs = 600
- batch size = 2
- Optimizer = Adam
- Learning rate = 1e-4
- Loss Function = Deep supervision loss using DiceLoss function.
  Deep supervision involves introducing intermediate supervision signals at multiple layers of a neural network, allowing for better gradient flow and potentially faster convergence during training. Our loss function computes a weighted sum of the loss contributions from predictions made at each intermediate layer.

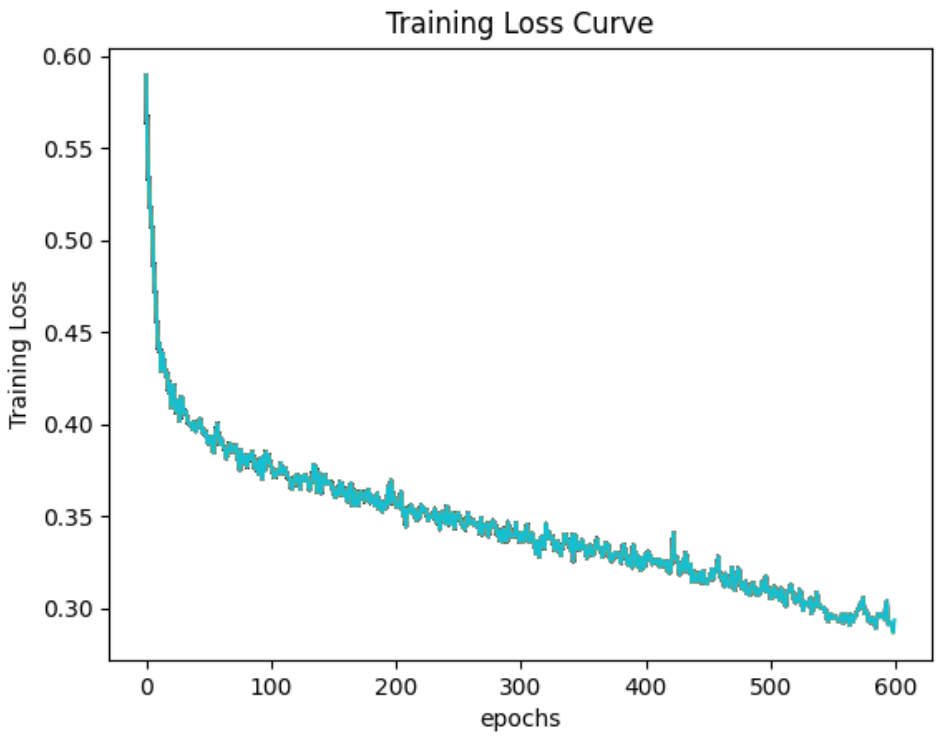
No dynamic learning rate scheduling because the loss curve seemed smooth enough, no aggressive oscillations observed.

Data preparation:
Converting DICOM images from the CT-82 dataset to NIfTI format enables compatibility with the MONAI library.

Evaluation metrics: Dice Score, Precision, Recall, surface to surface distance.

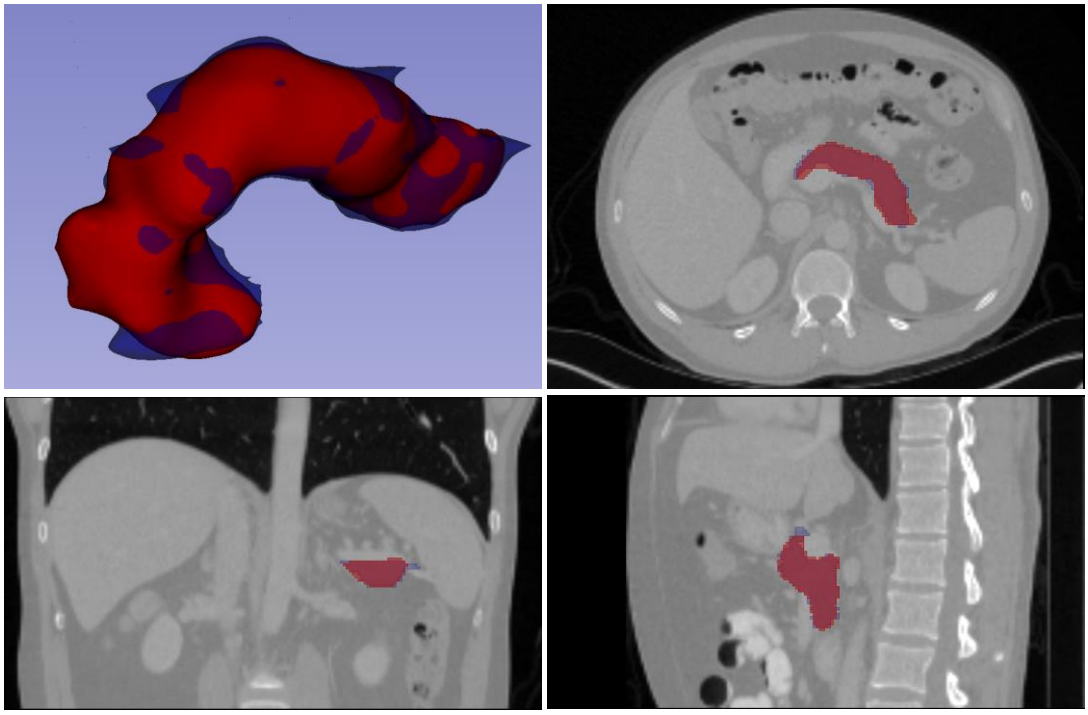
Baseline Evaluation:

Below, Training loss curve for our paper's model implementation:



Looking at the loss curve we can see a large drop at the beginning and almost linear decrease from epoch 100 and on. The training seems like it did not converge yet at 600 epochs, but for sake of experimenting with other training set ups we cut the training there. We might observe better results if we had more time to train that network, possibly reaching closer to the original results reported in the paper.

Below, we show one test example of the ground truth segmentation in blue, and our output segmentation of the baseline model – based on the paper.



baseline evaluation of our model's metrics compared to the paper's model metrics:

We compare our results to the reported results after training on the CT-82 dataset from scratch, denoted by SCR (another dataset used to train the model in the paper, which is not publicly available).

Method	Dice Score	Precision	Recall	S2S Dist
Paper's Attention U-NET	0.821 ± 0.057	0.815 ± 0.093	0.835 ± 0.057	2.333 ± 0.856
Project's Attention U-NET	0.7616	0.7046	0.8637	4.5603

Our project's results demonstrate decent performance, particularly considering the complexity of achieving high accuracy in pancreas segmentation tasks.
It's important to point out that our training did not converge, and the results might have been more aligned with those in the original paper if there had been more time. Despite some deviations from the metrics reported in the referenced paper, our model achieved a Dice Score of 0.7616, indicating substantial overlap between predicted and ground truth masks. While this falls slightly short of the paper's

reported score of 0.821, it's important to note the difficulty of reaching such high-performance levels. Additionally, our model exhibited strong precision (0.7046) and recall (0.8637) in correctly identifying pancreas regions, although precision was slightly lower compared to the paper's model. However, the Surface to surface distance of 4.5603 suggests room for improvement in localization accuracy compared to the paper's reported value of 2.333.

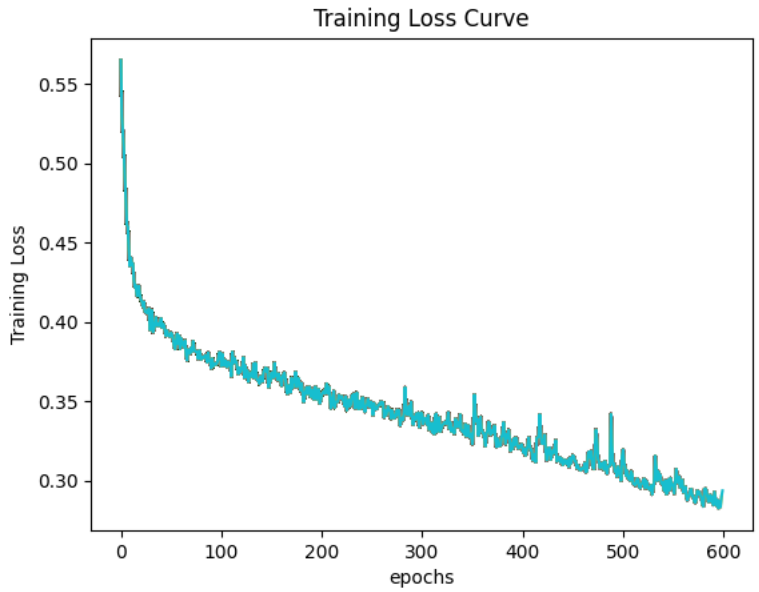
Variations and Experiments:

Deepen the model:

One notable adjustment involved deepening the network by increasing the feature channels in the final encoder layer from the standard 512 to 1024.

This modification aimed to assess the impact of enhancing the model's capacity by accommodating more feature maps.

Here, we delve into the rationale and potential implications of this architectural variation.



Seems to be very similar to our base model's loss.

Changed Loss Function to DiceCE Loss:

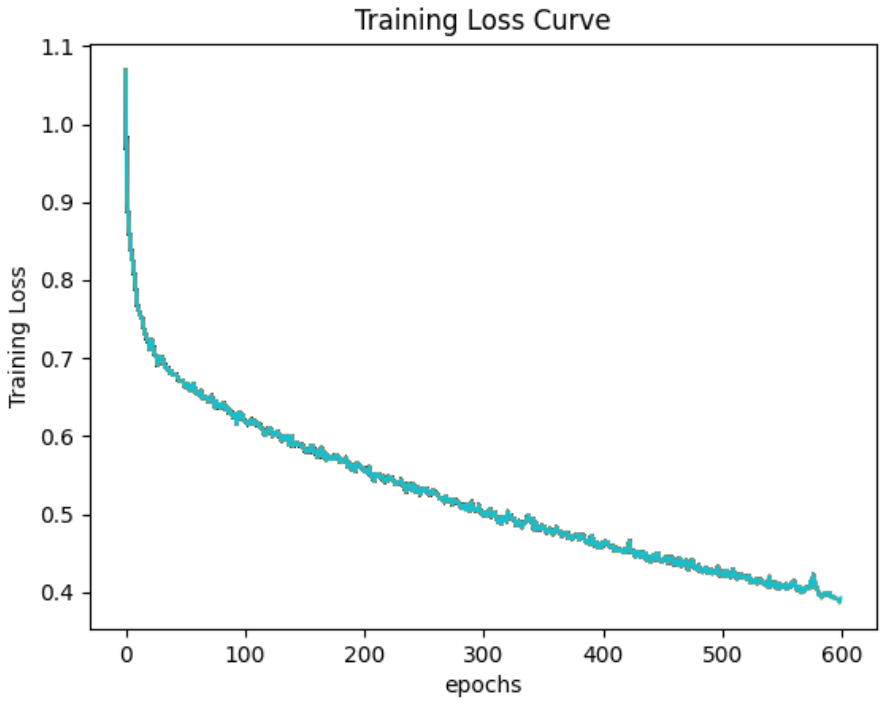
In an additional experiment, we incorporated a change in the loss function from Dice Loss to DiceCE Loss. The DiceCE Loss combines both Dice Loss and Cross

Entropy Loss, providing a more comprehensive measure of performance. By integrating both loss components, the model can simultaneously optimize for segmentation accuracy and pixel-wise classification.

While Dice loss focuses on the overlap between the segmentation masks, cross-entropy loss penalizes incorrect classifications at the pixel level, offering a more granular control over the model's predictions. This combination can be particularly effective in sharpening the segmentation boundaries and improving model performance on difficult cases.

The DiceCE Loss parameters allow for fine-tuning the balance between the two loss terms, providing flexibility in model optimization. We chose to use the default parameters which gives equal weight to both loss functions.

This change in the loss function was motivated by the desire to leverage the strengths of both Dice Loss and Cross Entropy Loss, ultimately enhancing the model's ability to accurately segment complex anatomical structures.



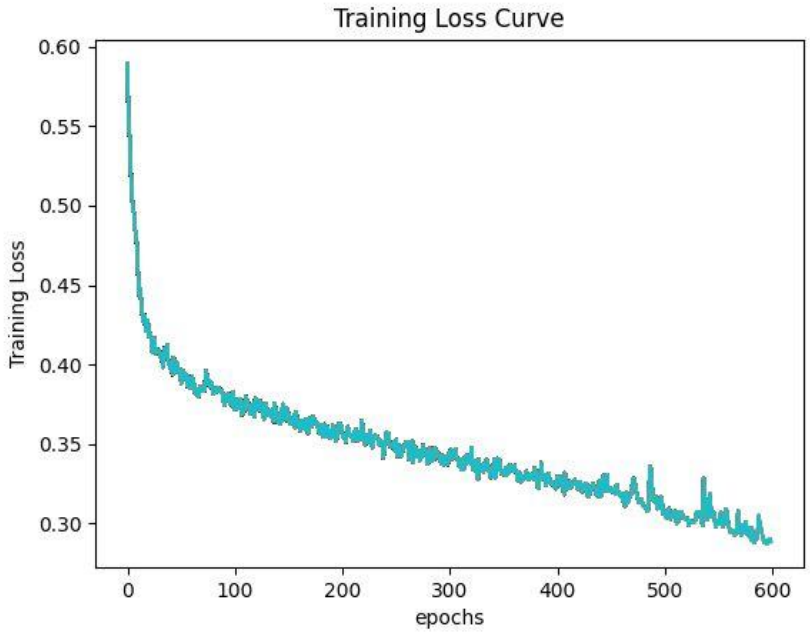
It is important to note that the higher loss is due to the combined loss functions. The behavior of the curve seems to follow a similar pattern as in previous trainings.

Attention mechanism change

In the AttentionBlock class, when conv_mapping is set to True, convolutional operations are utilized for weighted aggregation of the upper-layer and lower-layer feature maps, the purpose is to make the network to capture more complex relationships between them.

This approach should help the attention mechanism's ability to dynamically adjust the importance of spatial locations in the upper-layer feature map based on contextual information from the lower-layer feature map.

The aggregation convolutional layer combines the features from both layers to produce a fused representation, facilitating more intricate interactions and improving the attention mechanism's effectiveness.



Again, the behavior of the curve seems to follow a similar pattern as in previous trainings.

Method	Dice Score	Precision	Recall	S2S Dist
Paper's Attention U-NET	0.821 ± 0.057	0.815 ± 0.093	0.835 ± 0.057	2.333 ± 0.856
Project's Attention U-NET	0.7616	0.7046	0.8637	4.5603
Deeper Attention U-NET	0.7566	0.6583	0.9088	1.3498
DiceCE Loss	0.7930	0.7558	0.8598	1.9074
Modified Attention	0.7500	0.6771	0.8742	1.4986

We observed the best generalization with the model that was trained on the Dice and cross entropy loss. Possibly indicating that a pixel level loss was needed.

Challenges:

- Working with a relatively small dataset from the CT-82 dataset posed challenges in model generalization and robustness due to limited training examples.
- Not all details regarding preprocessing steps were explicitly mentioned in the article, such as the specific size of patches cut from the images, introducing ambiguity in reproducibility and parameter optimization.
- The article did not provide explicit information on the weight booting strategies employed for attention mechanisms and U-Net architecture, as well as crucial hyperparameters such as learning rate and number of epochs, complicating reproducibility, and benchmarking of results.

Addressing these challenges necessitated extensive experimentation with various hyperparameters, and data augmentation techniques to attain results closely resembling those reported in the paper.

This process of iterative refinement was a significant challenge, requiring meticulous tuning and validation to achieve comparable performance metrics while ensuring reproducibility and robustness of the segmentation model.

Discussion on insights

1. **Baseline Model Performance**: It's noteworthy to highlight that our base model exhibited a decent performance, particularly in relation to the metrics reported in the referenced paper. With access to more computational resources, and more time to train and fine tune, there exists a promising opportunity to further optimize our base model and potentially reach or even surpass the performance reported in the referenced paper. By extending the training duration, exploring larger model architectures, and fine-tuning hyperparameters, we could potentially unlock additional gains in segmentation accuracy.
2. **Deeper Attention U-Net**: Despite employing a deeper attention mechanism, our model exhibits a slightly lower Dice Score (0.7566) and worse precision recall values, compared to the baseline. This suggests that while deeper net may enhance certain aspects of segmentation, it may not necessarily improve overall performance.
   Having said the, on a larger dataset, we might see an improvement, as we use more parameters, we probably need more data for such complicated network.
3. **DiceCE Loss**: Introducing the DiceCE loss function results in a marginal increase in Dice Score (0.7930), indicating its potential effectiveness in enhancing segmentation accuracy mixed with higher precision and slightly lower recall. Overall, a very decent performance for the DiceCE loss function!
4. **Different Attention**: Implementing a different attention mechanism yields varied outcomes. While the Dice Score (0.7500) is comparable to the baseline, both Precision (0.6771) and Recall (0.8742) show worse values.

In summary, these additional experiments highlight the nuanced impact of attention mechanisms and loss functions on pancreas segmentation performance, underscoring the importance of carefully selecting and fine-tuning these components to achieve optimal results.

From our experiments, it seems like working with the DiceCE loss could even enhance our model performance, although that its hard to conclude when the models did not converge.

Suggestions for further improvements

1. Extended Training Duration: Given sufficient computational resources and time, extending the training duration could significantly enhance model performance. A longer training period would afford the model more opportunities to refine its learned representations, optimize segmentation parameters, and adapt to the intricacies of pancreas segmentation. This prolonged exposure to training data could help capture subtle anatomical variations and improve the model's ability to generalize to unseen images. Thus, investing in longer training times, if feasible, holds the potential to yield substantial improvements in segmentation accuracy and clinical utility.
2. Ensemble Learning Approaches: Leveraging ensemble learning techniques, such as model averaging or stacking, by combining multiple independently trained models can improve segmentation accuracy and robustness. Ensemble methods can help mitigate the inherent variability in model predictions and enhance the overall performance by leveraging diverse sources of information.
3. Pre training on a larger dataset: Utilizing a larger dataset is essential as the models were trained only on 60 images. We can pretrain on a large dataset and then finetune on the target dataset. This approach can help tailor the model to the intricacies of pancreas segmentation.
4. Integration of Weakly Supervised Learning: Exploring weakly supervised learning approaches, such as semi-supervised learning or self-supervised learning, can help alleviate the burden of manual annotation and facilitate scalability to larger datasets. By leveraging weak supervision signals, such as image-level labels or anatomical priors, we can effectively train the model with limited labeled data and improve segmentation performance.
   In the lack of enough annotated samples, we can use self-supervised approaches to capture the distribution of medical images, with proxy tasks such as rotating an image, then letting the model predict the angle of rotation.

5. Clinical Validation and Deployment: Conducting extensive clinical validation studies in collaboration with domain experts can validate the utility and efficacy of the model in real-world clinical settings. Assessing the model's performance against clinically relevant endpoints and conducting prospective validation studies can provide valuable insights into its impact on patient care and clinical decision-making.