UNIVERSITY OF
CAMBRIDGE

UNet-based Segmentation of Kidney and Kidney Tumour in 3D Computed Tomography images

Abstract

This study introduces a deep learning approach to segmenting kidney and kidney tumors in 3D CT images using various configurations of the 3D UNet architecture trained on the KiTS19 dataset. We outline our methodology, beginning with the foundational setup and rationale behind our choices for data preprocessing, model architecture design (including plain, residual, and pre-activation residual 3D UNet), training strategies, and inference methods. Through rigorous hyperparameter tuning, we identified an optimal hyperparameter combination that includes patch generation, data augmentation, biased sampling, batch size, and learning rate scheduling. The residual 3D UNet emerged as the top performer in kidney tumor segmentation during 5-fold cross-validation, albeit displaying a tendency towards conservative predictions. Future improvements should prioritize reducing false negatives to enhance segmentation accuracy in clinical applications, thereby advancing the precision of diagnoses and treatments based on 3D CT scans of kidney tumors.

CRSid: yz870

Date: 30th June 2024

word count ≈ 6600

Table of Contents

1 Introduction

Each year, over 400,000 individuals are diagnosed with kidney cancer, and in 2022 alone, the disease resulted in over 150,000 deaths (World Cancer Research Fund 2022). Unlike other cancers where radiotherapy and chemotherapy are viable treatments, surgical resection remains the predominant therapeutic approach for kidney tumours (Henson et al. 2020). Among options for surgery, partial nephrectomy, where only the tumour is removed, has recently become the standard of care to better preserve renal function (Zabell et al. 2017). Furthermore, given the consensus that the majority of kidney tumours are indolent (Tsuzuki et al. 2018), active surveillance has emerged these years as a conservative treatment strategy for tumours.

In this context, the precise evaluation and treatment planning for kidney cancers relies extensively on detailed information regarding the positions, shapes, and sizes of kidneys and tumours. Although imaging techniques like Computed Tomography (CT) enable accurate tumour detection, without automatic delineation tools, clinicians often turn to nephrometry scoring systems which assess the complexity of kidney tumours based on various anatomical features observed (Kutikov and Uzzo 2009). However, these systems face challenges such as extensive manual involvement (Simmons et al. 2010), variability in interpretation among expert evaluators (Spaliviero et al. 2015), and limited predictive capability (Kutikov, Smaldone et al. 2011).

To address these problems, one promising approach is designing automatic semantic segmentation techniques, which can precisely assign a category label to each pixel of a medical image depicting organs or tissues of interest for a specific imaging modality. In the context of CT imaging, traditional methods such as thresholding, region-growing, and active contouring, though generally perform well in a single image, can struggle to adapt to diverse clinical scenarios dynamically and may not efficiently handle the growing volume and complexity of medical data (Zanaty and Ghoniemy 2016).

Recently, deep convolutional neural networks (CNNs) have garnered widespread recognition for their outstanding capability to extract and represent features when dealing with miscellaneous large-scale medical imaging (Liu et al. 2021). Architectures such as fully connected networks like DeepLab (L.-C. Chen et al. 2017), UNets (Ronneberger et al. 2015), and generative adversarial networks like SCAN(Dai et al. 2018) have seen increasing use in semantic segmentation tasks within the biomedical field.

Training a robust deep network for segmentation tasks requires a substantial amount of meticulously annotated image data. However, currently, the vast majority of publicly available datasets for kidney imaging only consist of scans from healthy patients, without cancerous tissue being presented. In response to this challenge, the KiTS19 dataset (Heller et al. 2019) provides 210 high-quality annotated 3D volumetric CT scans for training, including both images and labeled masks, along with 90 CT scans (images only) designated

for testing. This dataset will be utilized in this paper to develop automatic segmentation models tailored for kidneys and kidney tumors.

Numerous attempts have been made to implement 3D CNNs for biomedical volumetric data. Among these, the 3D U-Net architecture (Çiçek et al. 2016) and its variants have been predominantly employed due to their robust end-to-end learning capabilities, adaptability to various medical imaging tasks, and efficient training processes.

In this paper, we aim to build on the success of the 3D U-Net architecture to train a segmentation model on the KiTS19 dataset, with the goal of achieving high-quality segmentations for both kidneys and kidney tumours in unseen 3D CT scans. In the methodology section, summarized by the following flow chart, we will detail the basic setup and rationale behind our choices for data preprocessing, model architecture design, training strategies, and inference methods. We will also explain the key factors in data processing and training that were considered during our extensive hyperparameter tuning.

In the results section, we will first compare the performance of the model under different hyperparameters. Using the optimal setup, we will then train three 5-fold cross-validation models on three variants of the 3D U-Net architecture: plain, residual, and pre-activation residual, to determine the best-performing model. We will generate final predictions on the test dataset using an ensemble of the 5-fold cross-validation models and obtain the scores from the challenge website. Finally, we will analyze potential shortcomings of our model by closely inspecting the generated predictions.

In the conclusion and discussion sections, we will analyze the implications of our findings, discuss the strengths and limitations of our approach, and propose potential future directions for further improving kidney tumour segmentation using deep learning techniques.

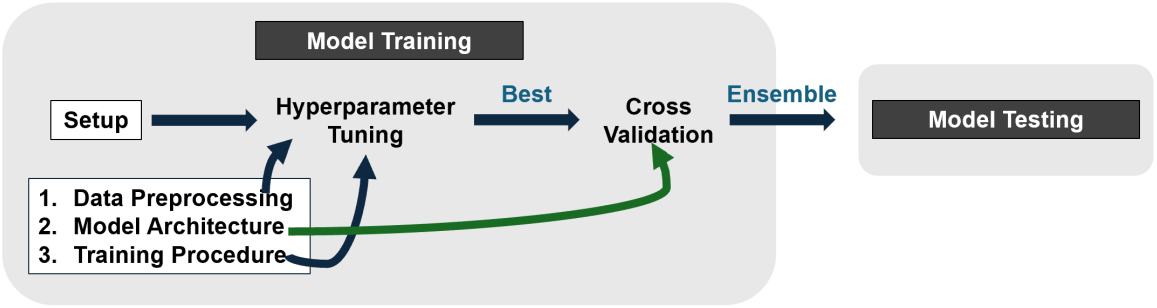

Figure 1: Analysis Pipeline

2 Methods

2.1 Preprocessing

According to Heller et al. 2019, the KiTS19 dataset includes 300 contrast-enhanced CT scans from patients who underwent partial or radical nephrectomy at the University of Minnesota between 2010 and 2018. Ground truth labels were manually generated by medical students under the supervision of an expert clinician, relying solely on transverse image planes, and labels for unannotated slices were computed through interpolation.

The dataset is stored in NIfTI (Neuroimaging Informatics Technology Initiative) format, which contains header information and raw voxel data Gorgolewski et al. 2016. From the provided metadata, the first noticeable aspect is that the images and masks are stored in a negative direction along the physical depth axis, necessitating a flip in the z-direction.

Additionally, these volumes have resolutions ranging from 0.437 to 1.04 mm and inhomogeneous voxel spacing, with slice thicknesses varying from 0.5 mm to 5.0 mm. Standardizing voxel spacing and intensities across all images is essential for CNNs to effectively process the images and enhances their generalization ability, particularly when dealing with data collected under varying conditions.

Voxel spacing refers to the physical distance between adjacent voxels (the 3D counterpart of pixels in a 2D image), and having consistent voxel spacing ensures that each voxel represents the same physical distance across all images. While smaller voxel spacing results in higher-resolution images, it also necessitates increased memory and computational resources for storage and processing. To achieve a balance between retaining sufficient contextual information and managing computational demands, we resample all cases to a uniform voxel spacing of $3.22 \times 1.62 \times 1.62$ mm using a linear interpolator.

In medical imaging, to ensure consistent visualization and analysis of images from different scanners or hospitals, intensity values are often clipped to highlight specific features within a predefined intensity range, typically customized for the organ under examination (Isensee and Maier-Hein 2019). Adopting this principle, we utilize a clipping range of [-79, 304]. Subsequently, we normalize intensity values by subtracting 101 and dividing by 76.9. This transformation standardizes intensity values to a range for better weight initialization, thereby enhancing training stability.

Even after standardizing voxel spacing and intensity ranges, the resulting images still vary in shape, and thus required to be convert into a collection of image patches with a fixed size. As described in Section 2.2, we target an input shape of $80 \times 160 \times 160$, achieved through three steps:

1. Crop out dark regions devoid of tissue to sharpen the model's focus on relevant body areas.

2. Padding the resulting volume if any dimension is smaller than the patch size.

3. Generate patches using random sampling, sliding window with overlaps, or sliding window without overlaps. Zero-padding the image if sliding windows move out of the original image.

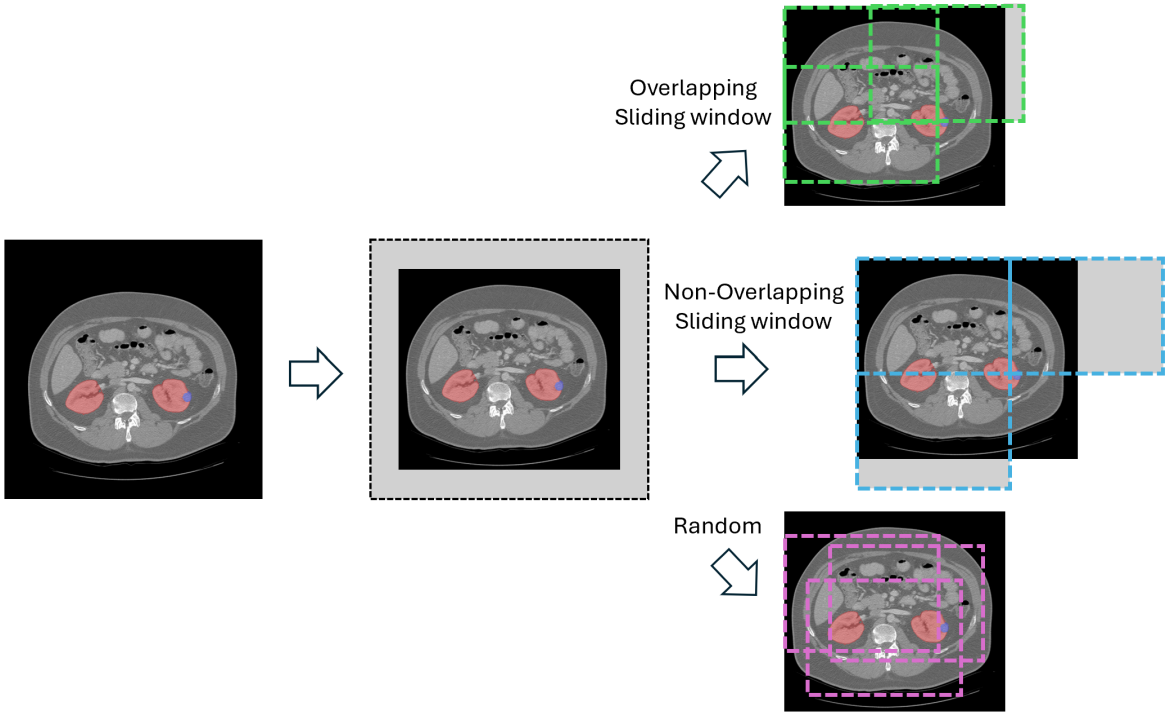

Figure 2: 2D demonstration of 3 patch generation methods

Figure 2 illustrates three 2D demonstrations of these patch generation methods: overlapping windows ensure repeated consideration of central image parts, enhancing robustness and minimizing the risk of overlooking critical details; non-overlapping windows prevent redundant processing and can offer computational efficiency; random sampling is straightforward and less prone to bias in sample selection. **Section 3.1.1** will provide a comparative analysis of these methods' effectiveness in our settings.

2.2 Network Architecture

The baseline model used for performing end-to-end segmentation of kidney tumours is the 3D U-Net, an extension of the 2D U-Net architecture (Ronneberger et al. 2015) into three dimensions, initially developed by Çiçek et al. 2016. Although the original 3D U-Net model marked a substantial advancement in volumetric segmentation within medical imaging, it incorporates only three downsampling operations. This might restrict the model's capability to capture the intricate and small-scale variations necessary for accurate segmentation of kidney tumours.

Therefore, we adopted the more sophisticated architecture outlined in Isensee and Maier-Hein 2019, illustrated in figure 3. This architecture employs 30 feature maps at the highest

resolution, which are gradually downsampled using strided convolutions. The number of feature maps is doubled with each downsampling operation, reaching a maximum of 320. Instance normalization replaces batch normalization, which is used in the original 3D U-Net design (Çiçek et al. 2016), as it effectively removes instance-specific contrast information from the content image (Ulyanov et al. 2016), thereby enhancing the network's invariance to contrast and appearance differences between images.

In the decoder, transposed convolutions are employed for upsampling, and skip connections between corresponding layers enable the preservation of fine-grained details while incorporating global contexts.

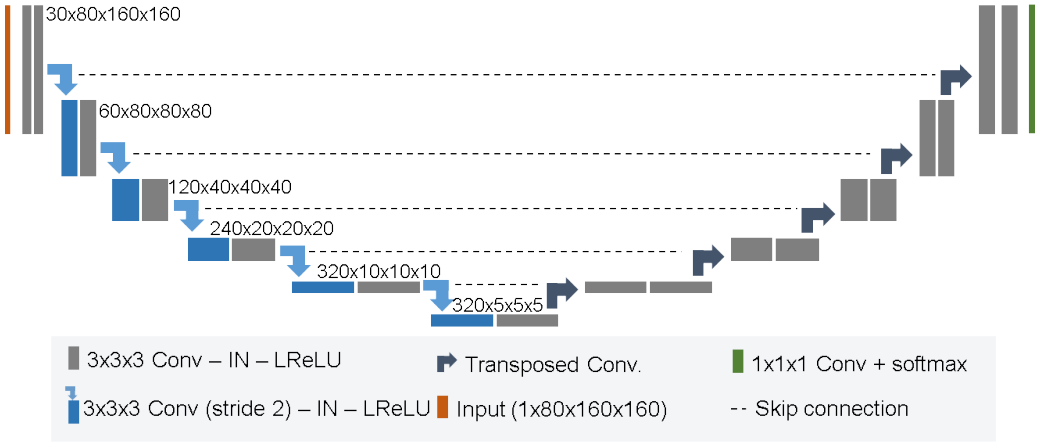

Figure 3: Plain 3D UNet architecture (Isensee and Maier-Hein 2019)

In deep learning, model performance can degrade with increasing network depth due to challenges such as vanishing gradients and the difficulty of learning complex mappings. Residual connections address these challenges by enabling the network to learn identity mappings, allowing the input signal to bypass certain layers (He et al. 2016a). ResNets, networks with residual connections, have a more stable gradient flow and an overall faster convergence speed.

We incorporated this idea into our original model architectue, utilizing the residual 3D U-Net architecture proposed in Isensee and Maier-Hein 2019, as shown in Figure 4. The general structure is similar to the baseline model but starts with 24 feature maps, includes residual connections in the downsampling blocks, and omits the second convolutional block after the transposed convolution block.

It is noticeable that the number of residual blocks increases as we gradually downsample the feature maps. This design is likely because downsampling decreases spatial resolution while increasing feature complexity and abstraction levels. Therefore, more residual blocks are needed to effectively capture and learn these increasingly complex features, enabling the model to capture nuanced details necessary for accurate segmentation.

Building on the foundation of ResNets, He et al. 2016b proposed a pre-activation alternative, which demonstrated a 1.1% performance improvement over its standard ResNet

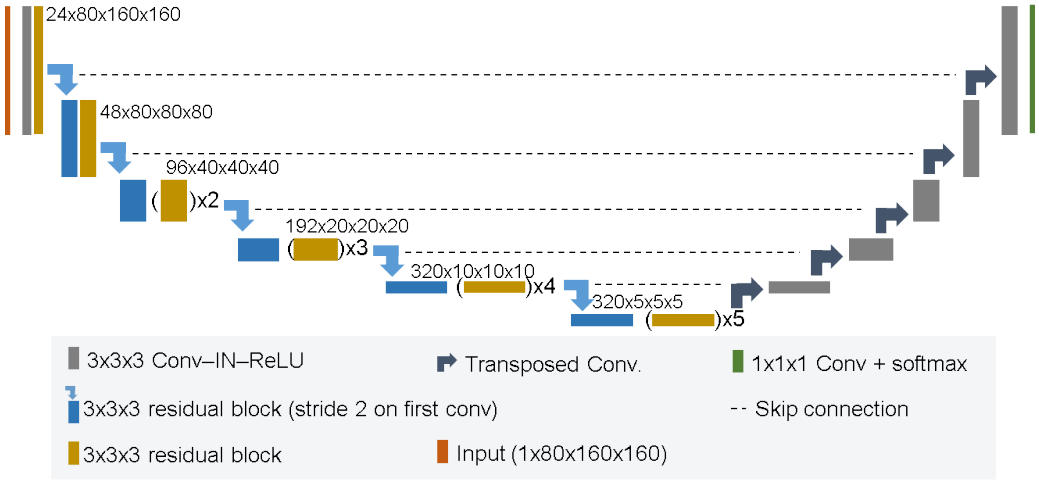

Figure 4: Residual 3D UNet architecture (Isensee and Maier-Hein 2019)

counterpart. Inspired by these findings, we also incorporated pre-activation residual connections into our model (Figure 4) to evaluate their impact on performance.

Figure 5 illustrates the differences between the original and pre-activation residual connections using the first downsampling block from Figure 4 as an example. The pre-activation variant first performs instance normalization and ReLU activation before entering the convolution. This ordering ensures that the input to each convolutional layer is normalized and activated, potentially leading to improved gradient flow and more efficient training.

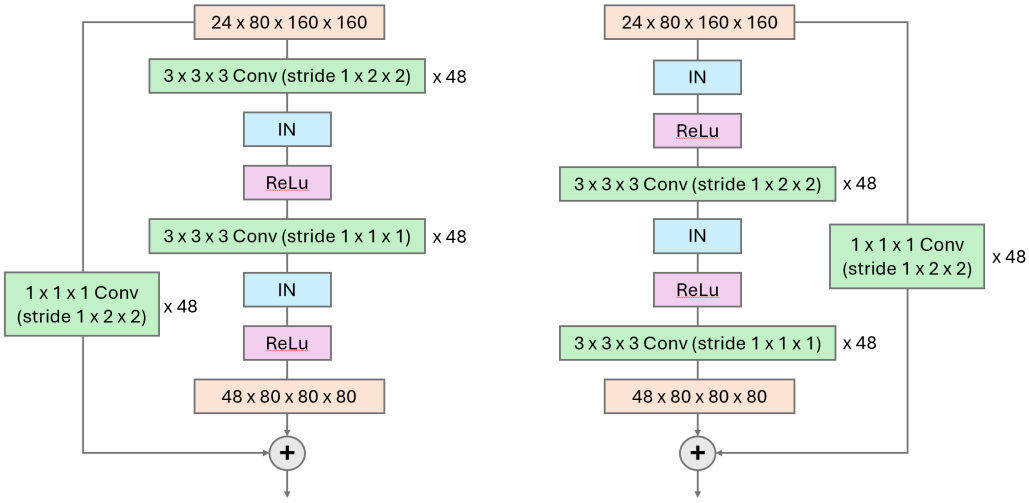

Figure 5: The first downsampling block in residual 3D UNet (left) and pre-activation residual 3D UNet (right)

Now, we constructed three different 3D U-Net models to be tested and compared: plain, residual, and pre-activation residual. Their performance on the KiTS19 dataset will be compared in **Section 3.2** after we determine the preferred training procedure, which will be discussed in detail in the subsequent sections.

2.3 Training

Recall from **Section 2.1**, after preprocessing, each training case has been converted into a set of $80 \times 160 \times 160$ patches. The number of patches per case is determined by the patch generation method and the image size after adjusting the voxel spacing. To maintain consistency in the number of patches within a batch and to ensure each batch contains information from different cases, a batch of size n is formed by selecting n patches randomly from n different cases.

A 5-fold cross-validation will be performed on the three U-Net variants mentioned above. This means that within each fold, 42 cases will be treated as the validation set, and the remaining 168 cases will be used for training. Hyperparameter tuning before the final cross-validation is primarily conducted on the first fold to identify patterns and reduce overall training time.

The training process is executed on a single NVIDIA A100 GPU with 80 GB VRAM. The model is trained for 600 epochs per fold, and a full 5-fold cross-validation on the final model takes approximately 6 days. Additional training settings are discussed in the following subsections.

2.3.1 Biased Sampling

In the segmentation mask, the pixel labels for background (0), kidney (1), and tumor (2) are extremely unbalanced. In the following figure, we outline the edges of the kidney and kidney tumor in the median transverse, coronal, and sagittal planes of the 50^{th} patient. It can be observed that only a small portion of the target labels are 1 (kidney), and an even tinier portion belongs to 2 (tumor).

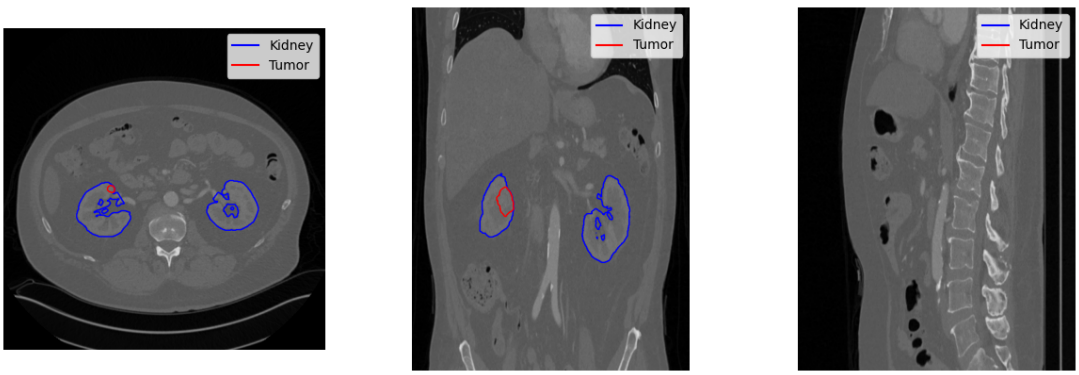

Figure 6: Median transverse (left), coronal (middle), and sagittal (right) planes of the 50th patient with kidney and tumour edges

This imbalance indicates that many generated patches may only contain parts of the kidney and tumour or even only contain the background. To ensure our model focuses on the primary features-the kidney and tumours-biased sampling is essential.

The first biased sampling approach discards patches generated far from the centre areas while using sliding windows on the image edges. Specifically, a patch is retained only if it contains more than half of its area from the original image in any of the three dimensions. When generating patches at the edge, we ensure that the remaining depth is greater than 40, the height is greater than 80, or the width is greater than 80. Otherwise, the patch would be filled with zero-padding and small edge images, likely not informative.

Another biased sampling approach we implement is to give patches with tumours (label 2) 4 times the probability of being selected compared to other patches when we randomly select a patch from a case. This ensures that kidneys and tumors are seen more frequently by the model during training, and prevents edge cases from being completely overlooked. We will experiment with this biased sampling strategy in **Section 3.1.2**, alongside the data augmentation techniques mentioned below.

2.3.2 Data Augmentation

Deep learning in medical image analysis faces challenges due to limited and costly training data. Data augmentation addresses this by expanding and diversifying the training dataset, acting as regularization (Goodfellow et al. 2016) to reduce model overfitting and improve generalization to new data. Additionally, data augmentation helps mitigate class imbalance issues by generating more data for under-represented classes. We apply the following three categories of data augmentations to our images:

- Spatial transformation: apply random rotations within the range of -10 to 10 degrees and random scaling to size between 0.9 and 1.1 times the original size.
- Color adjustment: vary the brightness and contrast to their original values times a random number between 0.5 and 1.5.
- Noise addition: randomly add standard-normal distributed noise, scaled by a random factor between 0 to 0.3, to the original images.

These augmentations are applied independently to each image rather than sequentially. This approach is taken because deep neural networks can be sensitive to minor variations in data, and individual augmentations are expected to effectively desensitize models to different irrelevant changes.

All transformations are performed in 2D transverse planes, including spatial transformations, due to the anisotropic nature of the data where voxel spacing varies across dimensions. In addition, in Isensee, Petersen et al. 2018, it is argue that when the maximum edge length of input patches in a 3D U-Net exceeds twice the shortest dimension, 3D data augmentation might become suboptimal. Hence, for $80 \times 160 \times 160$ patches, we apply consistent 2D colour adjustments and spatial transformations, but different noise additions across all slices in the depth direction to simulate 3D transformations.

The following four figures illustrate the appearance of volumes after applying one of the three types of augmentations mentioned earlier. The number of augmentation types applied to each patch during training will be a hyperparameter tuned in **Section 3.1.2**.

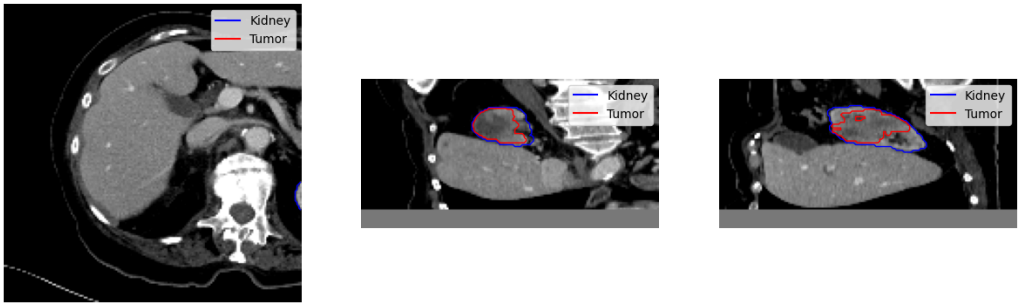

Figure 7: Original median transverse (left), coronal (middle), and sagittal (right) planes

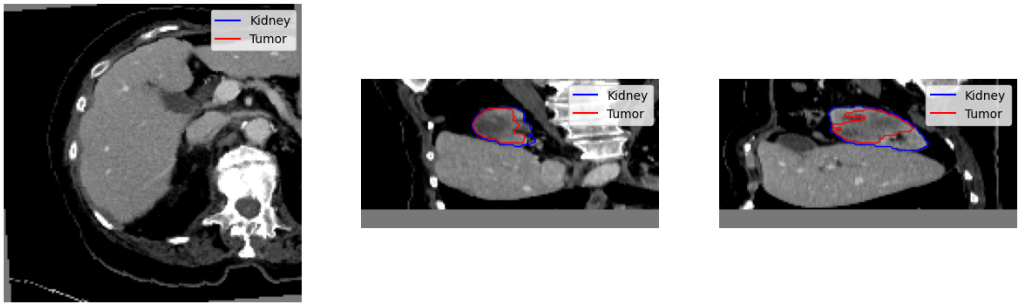

Figure 8: Medium planes after random rotation and scaling

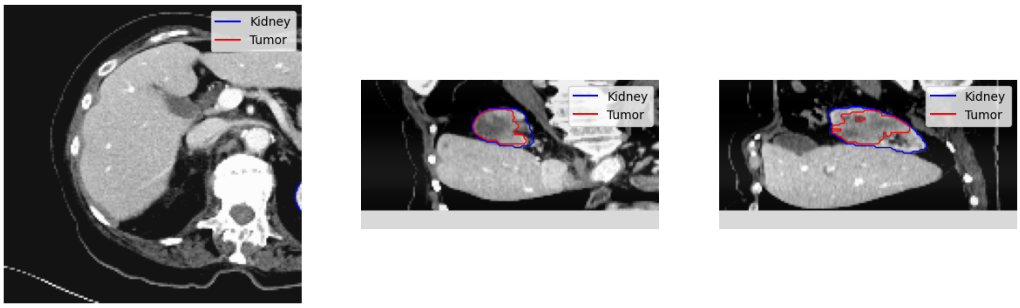

Figure 9: Medium planes after adjusting brightness and contrast

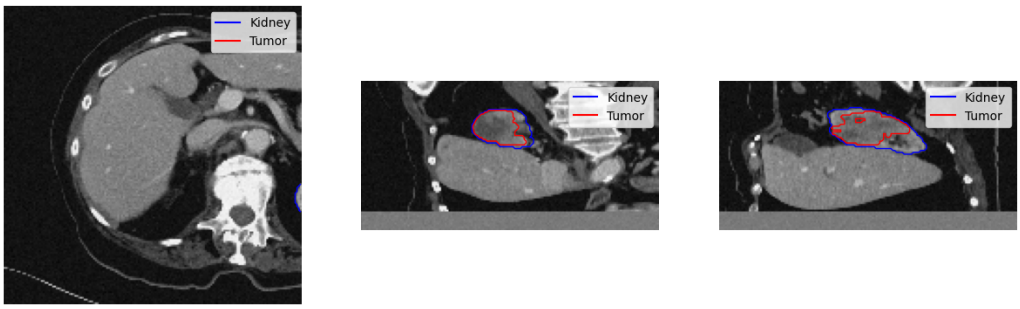

Figure 10: Medium planes after adding random Gaussian noise

2.3.3 Objective Function

For imbalanced classification tasks, selecting appropriate loss functions is crucial. If we rely on the accuracy, the proportion of pixels being correctly classified, the model can achieve almost perfect accuracy even when it fails to detect any Kidney. This situation, known as the "accuracy paradox" (Valverde-Albacete and Peláez-Moreno 2014), can be misleading. To mitigate this issue and provide a more comprehensive assessment of model performance, we opt for a loss function that combines cross-entropy (CE) loss and Dice score.

Cross Entropy (Eqn.1) loss penalizes misclassifications based on the logarithmic difference between predicted (p_{ic}) and true class probabilities (y_{ic}, $y_{ic} = 1$ only i is in class c) across N samples and C classes. It can handle class imbalance by assigning higher loss to misclassifications of the minority class. In our case, we use weights w_c of 0.5, 1, and 1.5 for background ($c = 0$), kidney($c = 1$) and kidney tumour ($c = 2$) respectively.

$$\text{Weighted CE Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} w_c \cdot y_{ic}\log(p_{ic}) \tag{1}$$

The Dice Score (Eqn.2) measures the overlap between predicted (p_i) and true (y_i) segmentation masks and thus is robust to class imbalance because it evaluates the similarity between segmentation masks regardless of the class distribution.

$$\text{Dice Score} = \frac{2\sum_{i=1}^{N} p_i y_i}{\sum_{i=1}^{N} p_i + \sum_{i=1}^{N} y_i} \tag{2}$$

$$\text{Dice Loss} = 1 - \text{Dice Score} \tag{3}$$

We compute the Dice score independently for kidneys and tumours. Since each pixel can only be assigned one label, but tumours (label 2) are actually a part of the kidney (label 1). Hence, Dice scores for kidneys were calculated by treating both kidney and tumour labels as foreground and all other labels as background. The Dice score for tumours is computed directly based on the tumour labels.

We compute the average of the kidney Dice score and tumor Dice score to form the composite Dice score (equation 4). We also include an additional term for the tumour Dice to force the model to focus more on the segmentation results of tumours. Then, the final loss function becomes:

$$\text{Composite Dice Score} = 0.5 \times (\text{Tumour Dice Score} + \text{Kidney Dice Score}) \tag{4}$$

$$\begin{aligned}\text{Loss} &= \text{Weighted CE Loss} + \text{Composite Dice Loss} + \text{Tumour Dice Loss} \\ &= \text{Weighted CE Loss} + 1 - \text{Composite Dice Score} + 1 - \text{Tumour Dice Score} \\ &= \text{Weighted CE Loss} + 2 - 0.5 \times \text{Kidney Dice Score} - 1.5 \times \text{Tumour Dice Score}\end{aligned} \tag{5}$$

2.3.4 Batch Size

Deep neural networks are often overparameterized, making them susceptible to overfitting. In addition to incorporating explicit regularizations in the loss function, several theoretical studies have highlighted that stochastic gradient descent (SGD) inherently provides an implicit regularization effect and a smaller batch size can intensify this effect (Li et al. 2018). However, opting for a small batch size may also increase the time required for training each epoch due to more frequent parameter updates.

As discussed earlier, a volume from a patient is initially split into multiple patches, with one of the patches being selected and undergoing several augmentations. These augmented patches, alongside the original patch, are then fed into the model simultaneously. The following diagram outlines the steps involved in creating a batch size of 2 using overlapping sliding windows and applying two random augmentations to each patch (rotation is in 3D for visualization purposes, but is implemented in 2D planes in practice).

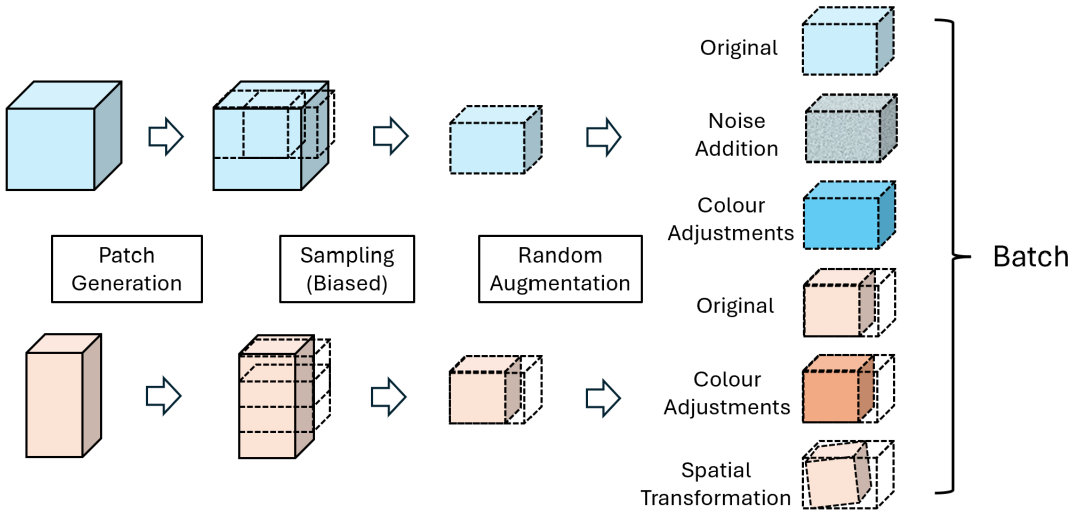

Figure 11: Example training batch (size = 2) using overlapping sliding window and two random augmentations on each patch

Following the steps shown above, a single case can create a maximum of 4 patches to be fed into the model. It has been tested that a maximum batch size of 4 (16 patches) is feasible in terms of memory usage, but we are still interested in exploring the effects of a smaller batch size, such as 2 (see **Section 3.1.3.**).

2.3.5 Optimizer & Learning Rate

Adam was used as the optimizer for stochastic gradient descent with an initial learning rate of 5×10^{-4} and l_2 weight decay of 3×10^{-5} for all experiments. Learning rate will be gradually reduced, enabling the model to fine-tune its weights more precisely after initial adjustments, thereby improving convergence and preventing overshooting.

To balance the need for rapid initial learning with the necessity of gradual refinement in later epochs, two different learning schedules were tested (results in **Section 3.1.4**): the first is based on short-term training loss reduction, where we reduce the learning rate if the training loss has not decreased in the last 10 epochs. The factor we multiply to the lr increases when the current composite Dice increases.

Algorithm 1 Learning Rate Scheduler 1

Input: epoch, train_losses, train_comp_dice, lr, min_lr
Output: lr
if epoch $>$ 11 and train_losses[-11] $\leq$ min(train_losses[-10:]) and lr $>$ min_lr **then**
    if train_comp_dice $<$ 0.5 **then**
        lr $\leftarrow$ lr $\times$ 0.2
    else if train_comp_dice $<$ 0.6 **then**
        lr $\leftarrow$ lr $\times$ 0.5
    else if train_comp_dice $<$ 0.8 **then**
        lr $\leftarrow$ lr $\times$ 0.8
    else
        lr $\leftarrow$ lr $\times$ 0.9
    end if
end if
return lr

The second learning rate adjustment method is based on the Exponential Moving Average (EMA) of training losses, like what has been done in Isensee, Petersen et al. 2018. Unlike immediate fluctuations in training losses, EMA provides a smoothed, longer-term perspective on model performance. This approach is less sensitive to short-term noise and offers a stable basis for making LR adjustments over epochs. The EMA at time step t for a sequence of training losses l_t is computed as:

$$\text{EMA}_t = \alpha \times l_t + (1 - \alpha) \times \text{EMA}_{t-1} \tag{6}$$

where we set $EMA_0 = 0$ and $\alpha = 0.1$. We record all the EMAs during training, and design the second the schedular as shown in Algorithm 2.

Algorithm 2 Learning Rate Scheduler 2

Input: epoch, train_emas, train_loss, lr, min_lr
Output: lr, train_emas
train_ema = 0.1 $\times$ train_loss + 0.9 $\times$ train_emas[-1]
train_emas.append(train_ema)
if epoch $>$ 31 and train_emas[-31] $\leq$ min(train_emas[-30:]) and lr $>$ min_lr **then**
    lr $\leftarrow$ lr $\times$ 0.2
end if
return lr, train_emas

***Early-stopping** We apply early stopping during the initial hyperparameter tuning phase to quickly assess model performance under different settings, halting training if the validation loss does not decrease in the previous 20 epochs. In the final cross-validation, early stopping is used until the learning rate drops below the minimum of 1^{-6}, at which point the model tends to overfit. More details are provided in the results section.

2.4 Inference

Due to the patch-based nature of our training, inference follows the same patch-based approach. Test images undergo preprocessing identical to the ones on training samples, and testing patches are generated using a sliding window with a stride of $60 \times 120 \times 120$, i.e., overlapping by a quarter of the patch size. All patches from a case are input into the model simultaneously, and the predicted probabilities after softmax across overlapping regions are averaged to produce the final prediction for the original image. If an ensemble of models are used to generate predictions, then prediction ensemble is also done via averaging probabilities predicted from every model in the ensemble.

During training, validation was conducted after each epoch using the first 10 cases from the validation dataset to optimize time efficiency. Validation metrics such as cross-entropy and Dice losses were computed based on the aggregated predictions from these cases. Following the completion of training for each fold in the cross-validation process, the model's performance was evaluated on the entire validation set to assess its generalization capability.

For predictions uploaded to the KiTS19 challenge, the predicted masks were adjusted back to each case's original voxel spacing using a K-nearest neighbors (KNN) interpolator. Additionally, the masks were flipped along the z-axis to account for the preprocessing steps applied during training. These adjustments ensured that the predictions accurately aligned with the original scan dimensions and orientation required for evaluation in the challenge.

3 Experiments & Results

3.1 Hyper-parameter Tuning

In this section, we detail the experiments conducted to determine the optimal combination of hyperparameters for training our 3D U-Nets to segment kidneys and kidney tumors. Specifically, we compare the training process using different patch generation methods, with and without biased sampling, varying the number of augmentations per patch, and employing different learning rate schedulers. All experiments described below utilize the plain U-Net architecture (Figure 3). We divided the dataset, using 1/5 of the training data as the validation set and the remaining 4/5 as the training set.

3.1.1 Patch Generation

Recall from **Section 2.1**, three methods for patch generation have been proposed: overlapping sliding windows, non-overlapping sliding windows, and random sampling. To evaluate the impact of different patch generation methods on the training process, biased sampling is excluded from this comparison, and we use one random augmentation (color, spatial, or noise) per patch. Training is conducted with a batch size of 2 using the first learning rate scheduler. Early stopping is implemented to save time. The loss and composite Dice scores are plotted below:

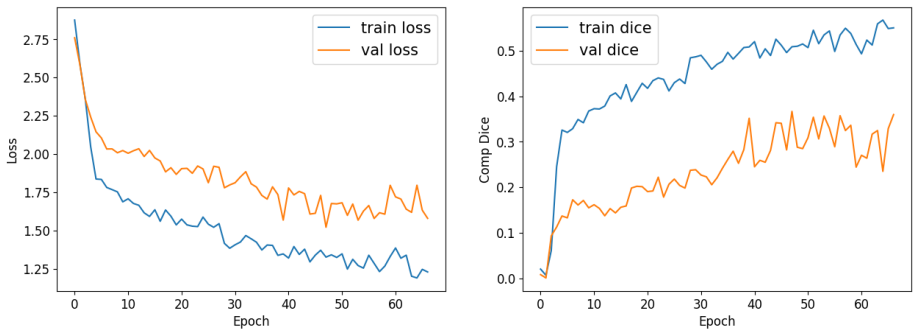

Figure 12: Losses and composite Dice scores using random patch generation

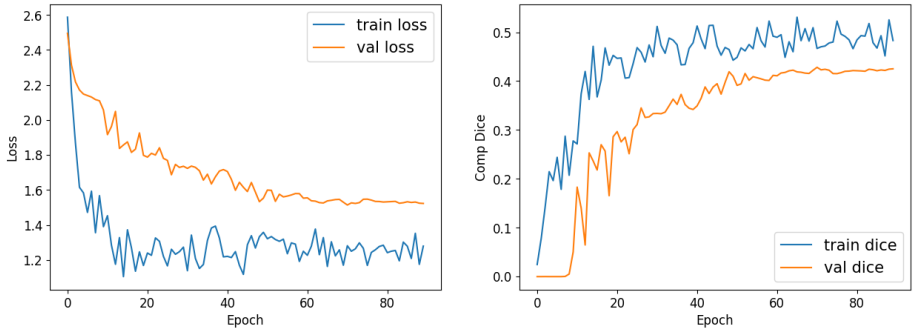

Figure 13: Losses and composite Dice scores using non-overlapping SW patch generation

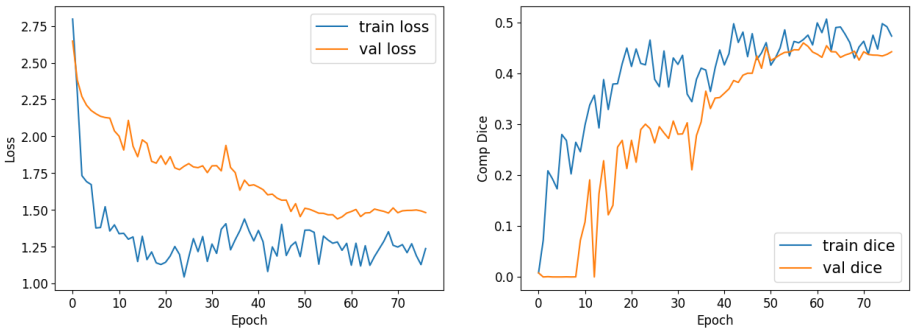

Figure 14: Losses and composite Dice scores using overlapping SW patch generation

In the case of random sampling, the training and validation losses decrease at a similar pace but maintain consistent differences. This is because, unlike sliding windows, random sampling only samples patches within the sample boundaries, which ensures patches never extend beyond the image edges. This results in smoother training curves, as the model rarely encounters 'surprise' instances where a tumour or kidney appears at the image corner and sees similar patches all the time.

However, it also hinders accurate predictions. Recall from **Section 2.4** that we perform patch-based inference using patches generated with overlapping sliding windows. During validation, the model encounters edge images with zero padding and possibly images with a kidney area in the corner for the first time. The model cannot generate accurate predictions for data forms it has never seen, leading to a consistent generalization gap between training and validation Dice scores.

When using sliding windows, the difference between the model's predictive ability on the train and validation set gradually decreases throughout the training process, with the overlapping method performing better. This outcome is logical because overlapping windows increase the likelihood of patches covering critical features of interest. Moreover, they enable the model to encounter various patch configurations, which resemble a blend of data augmentation and biased sampling techniques.

Therefore, we opt for using overlapping sliding windows to generate training patches. Similar to our inference method, we overlap the patches by 1/4 of the patch size, i.e., using a stride of $60 \times 120 \times 120$, in all the following sections.

3.1.2 Biased Sampling & Data Augmentation

In **Section 2.3.1**, we explained the method of biased sampling to be tested here, where patches containing tumours (label 2) are three times more likely to be selected for training (i.e., they have four times the probability of being chosen compared to other patches). Additionally, we defined 3 options for data augmentation: noise addition, colour adjustment, and spatial transformations.

Figures 12-14 illustrate that, without biased sampling and with only one instance of data augmentation, our model generally tends to overfit the data, maintaining a higher train Dice than test Dice throughout the training and causing the validation loss to plateau quickly. This underscore the necessity of both biased sampling and increased data augmentation for increasing generalization. Considering the additional computational resources and time required to generate and train more augmented data, biased sampling will be always used in the following experiments.

We will then investigate the optimal number of data augmentation types (**Section 2.3.2**) to implement by training four plain 3D U-Nets with no augmentation, one random type of augmentation, two random types of augmentation, and all three types separately. Other settings are not changed, the training processes are visualized below.

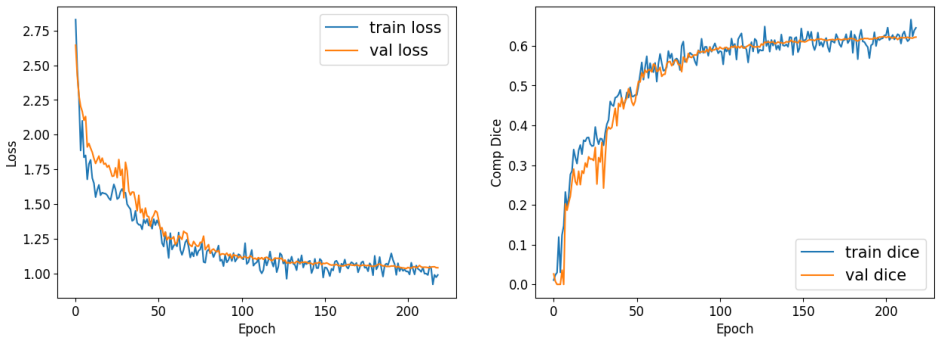

Figure 15: Losses and composite Dice scores without data augmentation

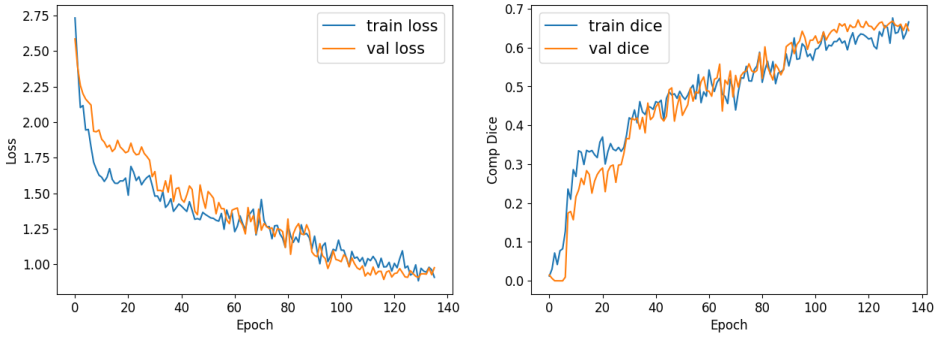

Figure 16: Losses and composite Dice scores with 1 data augmentation

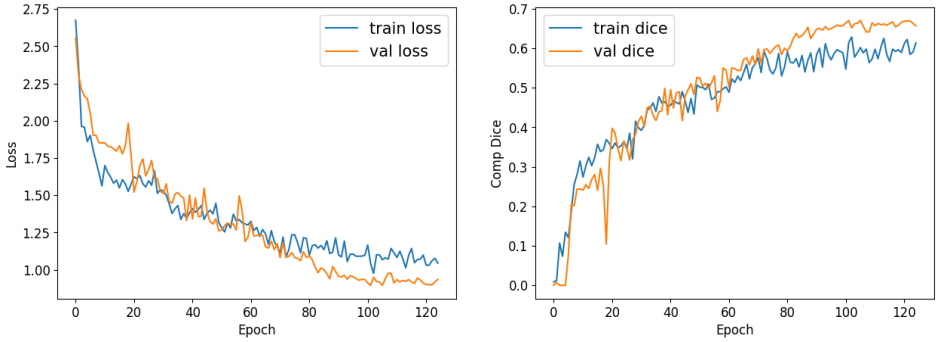

Figure 17: Losses and composite Dice scores with 2 data augmentation

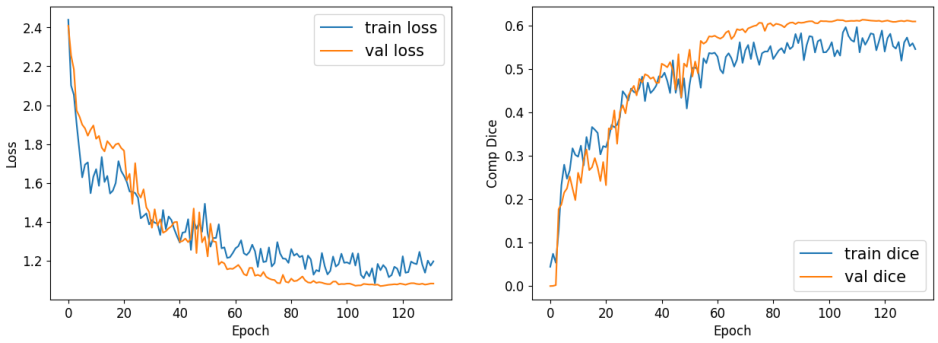

Figure 18: Losses and composite Dice scores with 3 data augmentation

From the above curves, it can be observed that even without any data augmentation, biased sampling significantly reduces the model's tendency to overfit, particularly during the early stages of training. When two or three augmentations are employed, the validation composite Dice score even surpasses the training Dice score. This does not necessarily imply better performance on the test sets since the training losses are computed on a patch basis. Due to biased sampling and augmentation, these patches frequently contain challenging-to-segment tumours, resulting in higher losses compared to those evaluated in a case-based manner while evaluation.

It is also noted that the training and validation losses converge to similar levels (around 1) by the end of training. However, this convergence is achieved most slowly in cases without augmentation and most quickly in cases with two or three augmentations. To evaluate the efficiency differences more thoroughly, we compute the time required for each epoch and assess the model's performance on the full evaluation set at the end of training (Table 1) for 4 models trained from varing data augmentation level.

Table 1: Comparison between different number of augmentations

# augmentations	time/epoch (min)	Dice tumor	Dice kidney	comp Dice
0	1.798	0.3120	0.8928	0.6024
1	2.431	0.3919	0.9082	0.6501
2	3.081	0.3424	0.9002	0.6213
3	3.713	0.2887	0.8835	0.5861

From the above table, we can see that adding one copy of augmented patches to the model results in an increase of around 40 seconds per epoch. Although the training losses still show potential for further improvement, and we plan to continue training for more epochs, the current evaluation results suggest that using one data augmentation yields the best results without significantly extending training time. Since it appears that the model has not yet focused on increasing the Dice score for tumors, we will first continue with one augmentation but remain open to increasing the number of augmentations if the model overfits the training set as we train for more epochs.

3.1.3 Batch Size

In the previous discussion of batch sizes, it has been stated that a batch size of 4 can be handled by our GPU VRAM and can potential reduce training time. Therefore, we now want to investigate whether increasing the batch size will affect our training performance. Using the same settings, we change the batch size to 4 and plot losses and composite Dice scores on the training and validation sets are as follows:

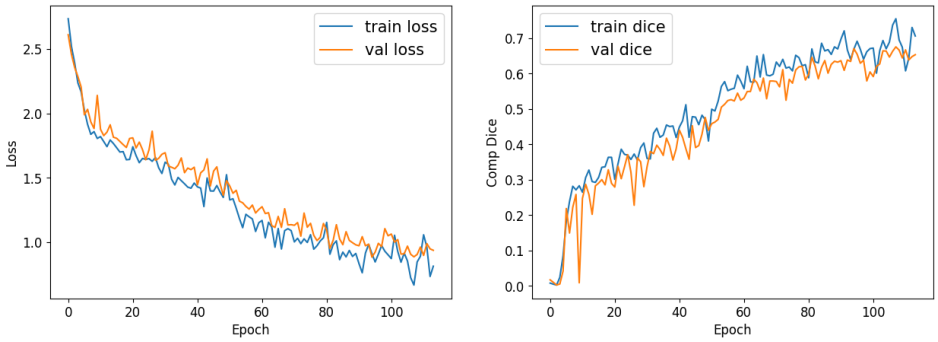

Figure 19: Losses and composite Dice scores using batch size 4

Compared to Figure 16, we observe that although the validation loss decreases at the same pace as the training loss, there is a noticeable difference between the training and validation performances. Aiming for better generalization, we will retain our original choice of a batch size of 2.

3.1.4 Learning Rate Scheduler

Up to now, we have consistently employed the first learning rate (LR) reduction method (Algorithm 1 in **Section 2.3.5**), which is based on immediate training loss changes, along with early stopping. This approach allows us to quickly compare the training progress with different hyperparameters. However, we have frequently observed that the training loss has not plateaued before the training be stopped prematurely due to transient improvements in validation performance observed a few epochs earlier.

To thoroughly evaluate the impact of two different LR reduction strategies (**Section 2.3.5**) over an extended training period, we will now discard early stopping and train the model using both LR schedules for 500 epochs. We start from an initial lR of 5×10^{-4} and gradually reduce it using 2 methods until it is below a minimum of 1×10^{-6}. The training procedure and how LR changes are visualized in Figures 20, 21, and 22.

From these plots, we observe two distinct patterns of loss reduction. Using LR Algorithm 1, although the validation loss decreases rapidly within the first 150 epochs along with the training losses, the model quickly overfits after the learning rate is reduced to the minimum. The validation composite Dice score remains around 0.6 until the end of training, while the training composite Dice improves to above 0.7.

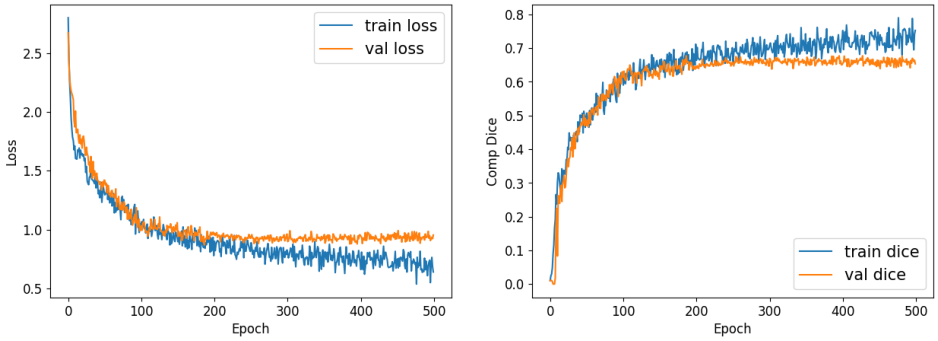

Figure 20: Losses and composite Dice scores using lr algorithm 1

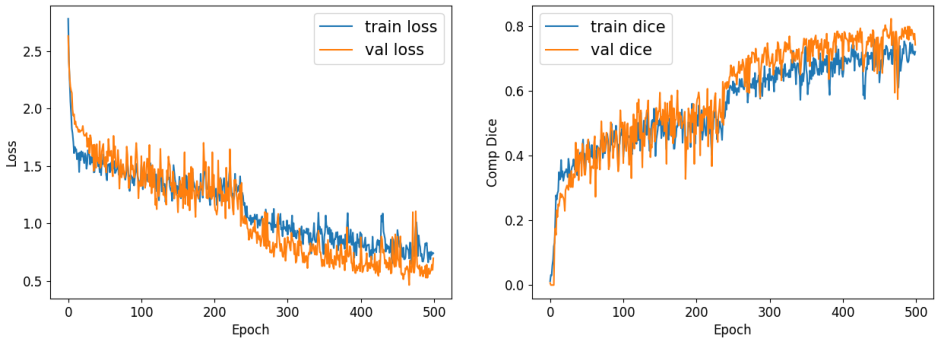

Figure 21: Losses and composite Dice scores using lr algorithm 2

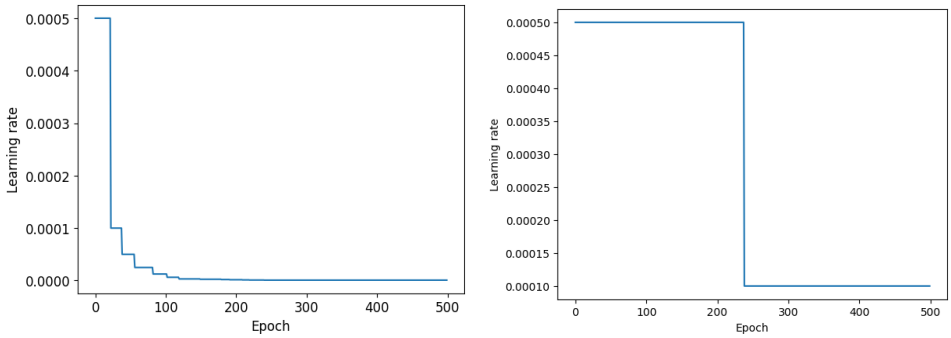

Figure 22: Learning rate changes using algorithm 1 (left) and algorithm 2 (right)

This behavior can be attributed back to the phase when a large learning rate is employed. During this phase, the parameter space is not thoroughly explored by our optimizer, leading to reduce the LR at a local minimum or saddle point on the loss surface. With a small learning rate, the model explores the parameter space very slowly, decreasing the likelihood of escaping local minima that represent overfitted solutions. Consequently, the model continues to adjust parameters within suboptimal subspaces.

We also experimented with increasing the number of epochs before reducing the learning rate, specifically waiting until the current loss had been achieved 20 or 30 epochs earlier. Despite this adjustment, the learning rate still decreased to its minimum value very rapidly. This observation suggests that Algorithm 1, which prioritizes short-term loss reduction, though aids the model in rapid learning during the initial training phase, it restricts the SGD to thoroughly explore the loss surface.

In contrast, as observed from Figure 21 and the right figure in Figure 22, despite experiencing more fluctuations in losses compared to those using Algorithm 1, there's no obvious overfit shown. By the end of training, the validation Dice score exceeded 0.7 even though the learning rate was reduced only once. The fluctuations in the loss and Dice curves and the sharp decline in loss at the moment of reducing the learning rate indicate that our optimizer explored various areas of the loss surface thoroughly before the rate reduction.

Since the training loss in Figure 21 plateaus for approximately 100 epochs before reducing the learning rate, we were curious about the possibility of earlier reduction. We modified the reduction criterion in Algorithm 2 to trigger when the training loss had not decreased for 20 epochs instead of 30. Subsequently, the training curve evolved as follows:

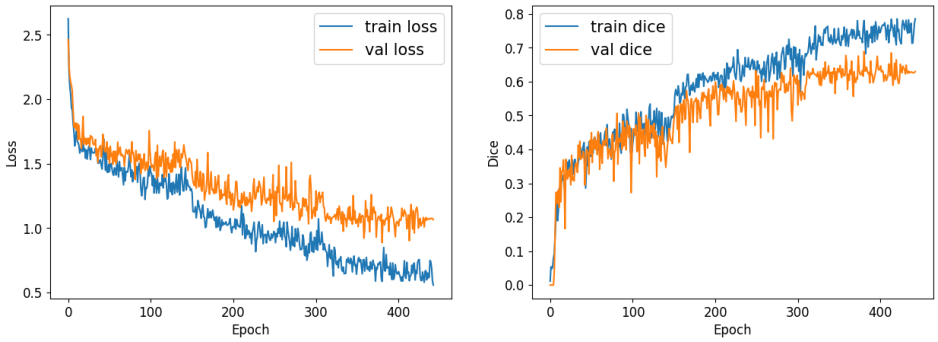

Figure 23: Losses and composite Dice scores using lr algorithm 2 (reduce faster)

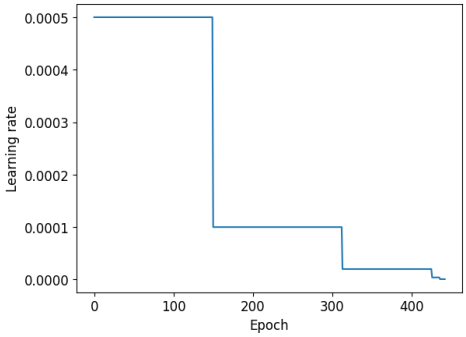

Figure 24: Learning rate changes using lr algorithm 2 (reduce faster)

The training procedure showed a noticeable decline compared to our previous observations, with the generalization gap widening each time the learning rate was reduced. Therefore, we decided to stick with the original setup for the LR reduction algorithm 2.

Additionally, it can be observed from Figures 20 and 23 that while the model may not overfit in the early stages of training, overfitting could occur once it has learned the main features of the kidney and starts refining its parameters to segment tumours more accurately. This could be attributed to a too-small learning rate, but more possibly due to the varying morphology of tumors across different cases. To mitigate this, we increased the types of random augmentations on each patch used to 2 during final cross-validation. By exposing the model to a greater variety of tumour shapes, we hope to reduce overfitting in the later stages of training.

3.2 Cross-Validation

Using the optimal hyperparameters determined in **Section 3.1**, we conducted 5-fold cross-validation on three variants of the 3D UNet. Each variant was trained for 600 epochs per fold. We compute the means and standard deviations of the final validation tumour, kidney, and composite Dice scores from 5 folds for each model, as shown below.

Table 2: Cross validation result

model	Dice tumor	Dice kidney	Dice comp
plain	0.4884 +- 0.0452	0.9339 +- 0.0100	0.7111 +- 0.0230
residual	0.5490 +- 0.0195	0.9449 +- 0.0083	0.7470 +- 0.0072
pre-activation	0.5269 +- 0.0376	0.9400 +- 0.0076	0.7334 +- 0.0195

From the table above, we observe that the tumor Dice score is the highest and exhibits the least variability across folds for the Residual 3D UNet model, followed by the Pre-Activation 3D UNet, with the Plain 3D UNet performing the poorest after 600 training epochs. In contrast, the Dice score for the kidney segmentation was similar across all three models, with the Residual UNet achieving the highest score again.

However, these performance rankings could change with additional epochs of training. Therefore, a closer inspection of the training process is warranted to determine if further training could enhance model performance. We plot how kidney and dice tumours change throughout the training of each fold of residual networks in Figure 25.

From Figure 25, we can observe that the validation Dice score for kidney segmentation consistently improves alongside the training validation Dice score. In folds 3 and 4, the validation kidney Dice score even surpasses the training kidney Dice score. As previously analyzed, this might be attributed to the differences between patch-based and case-based evaluation, yet it remains a good indicator of the model's capability in segmenting kidneys.

However, after approximately 500 epochs of training, the validation Dice score for tumour segmentation ceases to improve across all cases, with this plateau occurring even earlier in folds 4 and 5. Additionally, except for folds 1 and 2, there is a consistent discrepancy between the validation Dice score and the training Dice score. This persistent difference could be due to one or two particularly challenging cases in the 10 validation cases used for evaluation during training, or it may indicate a general lack of generalization ability in tumour segmentation across all cases.

We also notice more fluctuations in the tumour Dice scores, which is understandable given the smaller size of tumours. If the model adds the same amount of area to both the tumour and kidney in one weight update, the resulting change in the Dice score is more pronounced for the tumour than for the kidney.

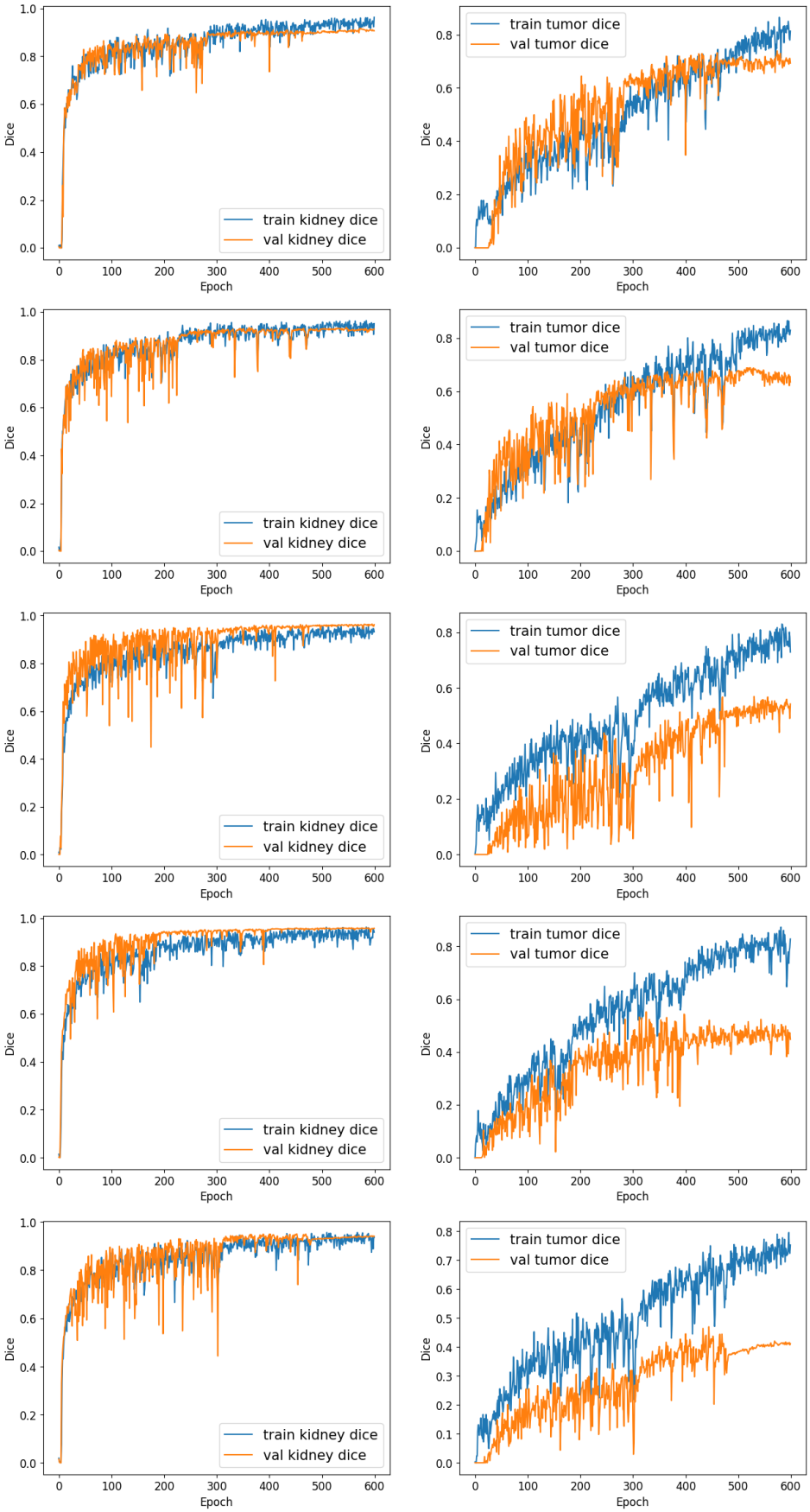

Figure 25: Kidney and Dice tumour Scores during training for residual 3D UNet fold 1 to 5 (from top to bottom)

3.3 Final Predictions

Similar trends, where the tumor Dice score stops improving after epoch 500, are observed in the training procedures of the Plain and Pre-Activation 3D UNet models as well. Therefore, we decided not to extend the training beyond 600 epochs. Instead, we selected the best-performing model, the Residual 3D UNet, to create an ensemble of five models trained in 5-fold cross-validation for predicting segmentations on the 90 test cases. The scores being obtained using our generated predictions from the challenge is **Mean Kidney Tumor Dice 0.7569 Kidney Dice 0.9360 Tumor Dice 0.5778**. Compared to Table 2, we can see that ensembling improves the model's ability to segment tumors.

***Why Dice tumour is low ?** Although it is theoretically incorrect to evaluate model performance on images that have already be seen during training, we still generate predictions for our 210 training cases using the ensemble of five Residual UNets and compare these predictions with their ground truths to identify potential issues. The distribution of Dice scores for kidney and tumour on these 210 predictions are as follows:

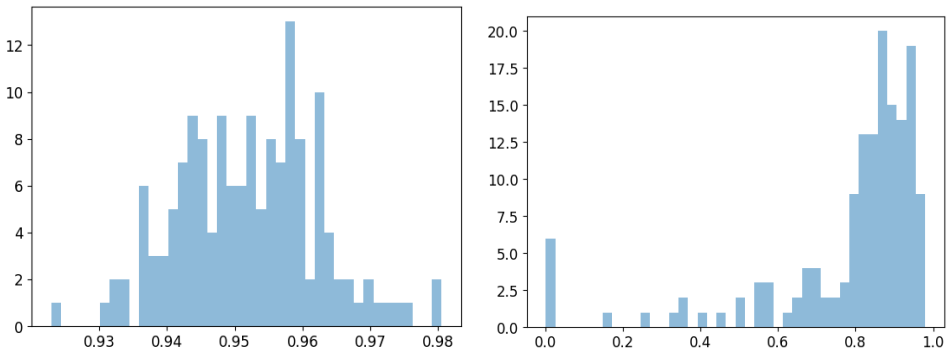

Figure 26: Distribution of kidney (left) and tumour (right) dice scores between truth and predicted segmentation on 210 training cases

The kidney Dice scores are consistently high and display a Gaussian-like distribution. However, the tumor Dice scores exhibit a left-skewed distribution, with the majority of predictions achieving over 80% overlap with the ground truth. Nevertheless, there are some cases where the predictions are almost or entirely incorrect. To better understand the model's performance, we visualized the predicted and true segmentations for cases with high (Figures 27-29) and medium (Figures 30-32) tumor Dice scores.

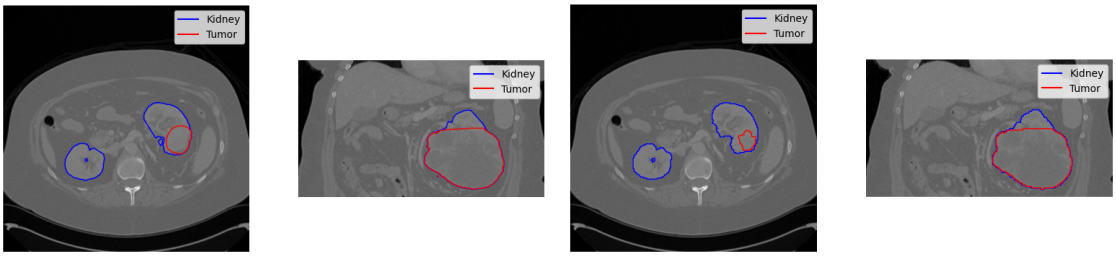

Figure 27: Ground truth (left) and predictions (right) in medium planes for case 67

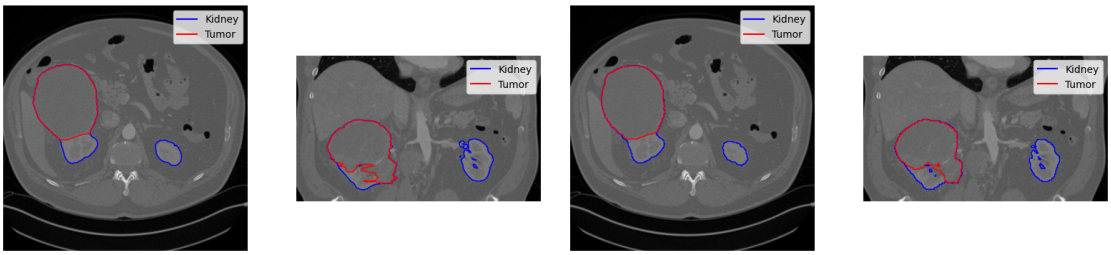

Figure 28: Ground truth (left) and predictions (right) in medium planes for case 114

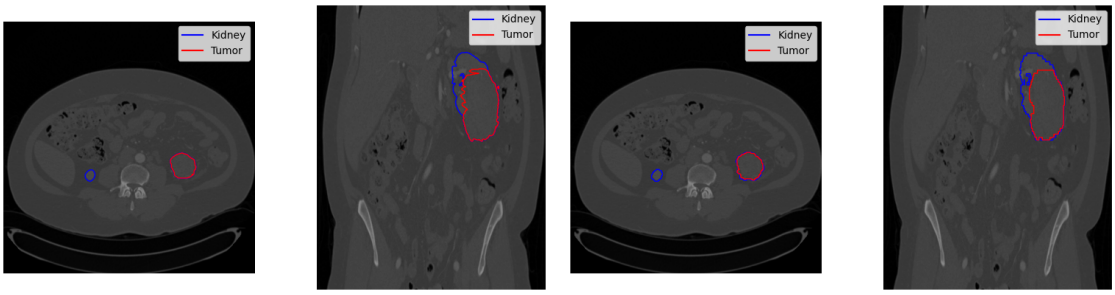

Figure 29: Ground truth (left) and predictions (right) in medium planes for case 135

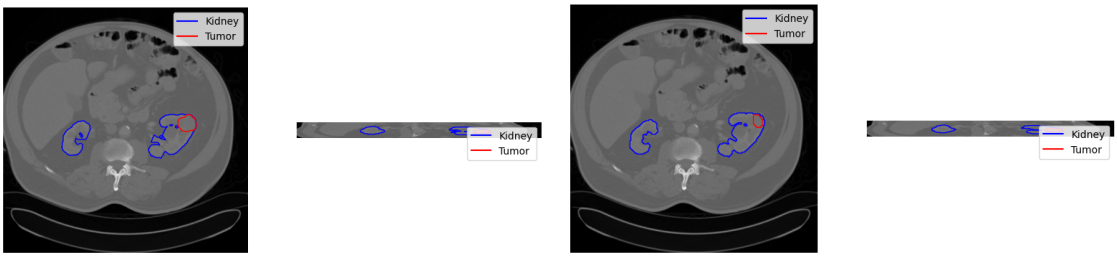

Figure 30: Ground truth (left) and predictions (right) in medium planes for case 110

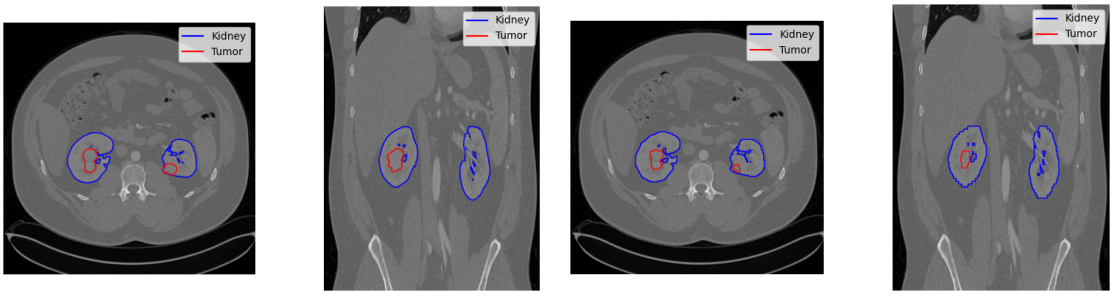

Figure 31: Ground truth (left) and predictions (right) in medium planes for case 118

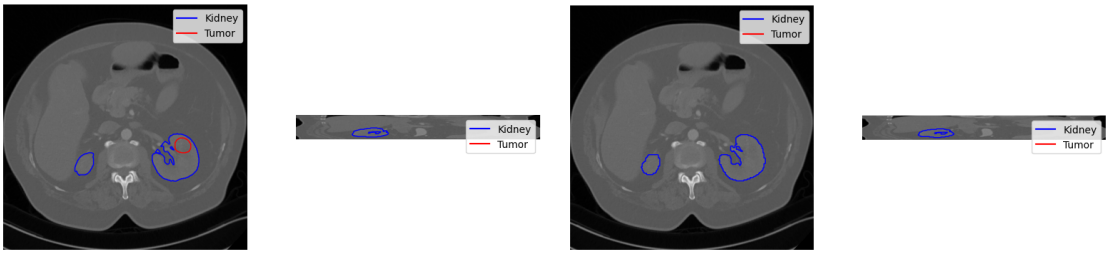

Figure 32: Ground truth (left) and predictions (right) in medium planes for case 122

From the above plots, we observe that cases with the highest tumour Dice scores (67, 114, 135) are those where large areas of tumours are present. Although exceptionally large tumours are more easily detected by the model, it still tends to generate smaller tumours (case 67) and smooth out the boundaries of tumours (cases 114 and 135). This conservativeness in our predictions is more apparent in cases where the model has an intermediate performance. For instance, in cases 110 and 118, the predicted tumour areas are visibly smaller than the ground truth. In case 122, the predicted tumour is even absent in the mid-transverse plane where it is supposed to be.

For cases with the lowest (actually zero) Dice scores, the tumours are usually very small and difficult to discern in the mid-transverse planes. Hence, we draw 2D slices with the largest tumour areas for cases 100, 85, and 123, where our model consistently fails to detect these small tumour portions hidden within the kidney.

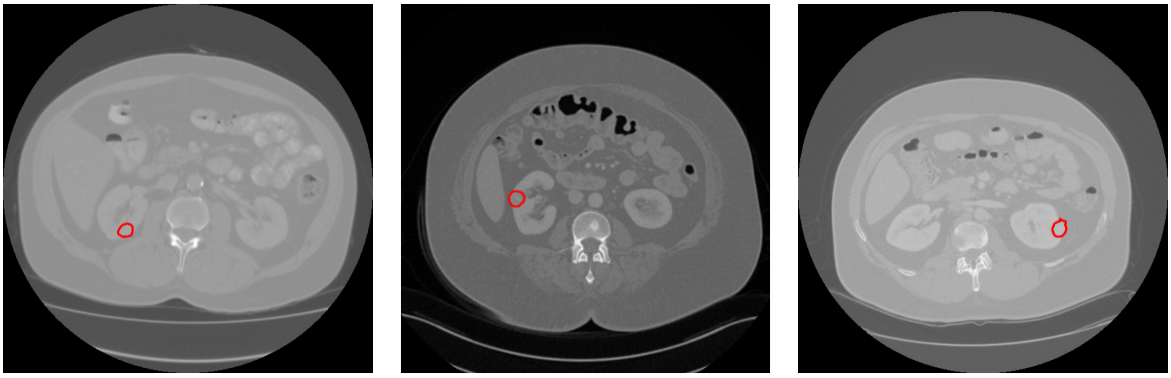

Figure 33: Ground truth slice with largest tumour area for case 85 (left), 100 (middle), and 123 (right)

4 Conclusion & Discussions

This study aimed to develop an effective model for segmenting kidney and kidney tumors in 3D Computed Tomography images. We utilized 210 training cases from the KiTS19 challenge to train three different versions of 3D UNet (plain, with residual connections, and with pre-activation residual connections). Thorough hyperparameter tuning for data preprocessing and the training procedure was conducted, and the optimal hyperparameters were used to perform 5-fold cross-validation on the three different UNet models.

In the methods section, we detailed the data preprocessing steps, which included adjusting voxel spacing and intensity ranges, and patch generation. We then introduced the basic setups of the three versions of UNet. For training, we discussed the reasons and methods for biased sampling, data augmentation, objective functions, batch size, and optimizer. We also explained our inference method, which involved averaging predicted probabilities from different patches and models in the ensemble.

In the results section, we sequentially explained the experiments conducted and determined the best hyperparameter combinations by comparing how composite Dice scores and loss changed throughout the training process. The optimal combination included overlapping sliding windows for patch generation, biased sampling, 2 random data augmentations on each patch, a batch size of 2, and a learning rate scheduler based on the long-term performance of loss decay.

After this, the residual 3D UNet was selected due to its high and consistent predictive capability for kidney tumors in 5-fold cross-validation. We generated predictions for 90 test cases using the ensemble of 5 residual UNets, achieving scores of Tumor Dice 0.5778, Kidney Dice 0.9360, and Composite Dice 0.7569 in the competition. We finally examined the weaknesses of our model by comparing the predictions and ground truths of 210 training cases and found a general conservativeness in tumour segmentation.

To improve tumour segmentations and thereby improving the overall performance and robustness of the segmentation model, we should guide the model to be more stringent about false negatives (pixels misclassifies as backgroudns) rather than false positives (pixels misclassifies as foregrounds). The following methods might be helpful and warrant further exploration:

- Adjust Loss Function: One straightforward modification is to increase the weights for the tumor class in the cross-entropy loss more than currently applied, although its efficacy remains uncertain. Alternatively, we can employ advanced loss functions such as Tversky loss (Salehi et al. 2017), which generalizes the Dice score and the Jaccard index by introducing parameters to control the penalty for false positives and false negatives. Additionally, Focal Loss (Lin et al. 2017) can be used as an extension of the cross-entropy loss that includes a modulating term to emphasize learning on hard-to-classify examples.

- Incorporate Hard Negative Mining: Traditionally used to reduce false positives in training CNNs for non-uniformly distributed classes (Bucher et al. 2016), hard negative mining can be adapted to mitigate false negatives. This involves periodically retraining the model with a dataset enriched with hard negative examples, which are the particularly challenging cases where the model fails to detect the tumours correctly.

- Supervision at Different Resolutions: In a U-Net with multiple encoding and decoding stages, as implemented in Heller et al. 2019, supervision is applied not only at the final output layer but also at multiple intermediate layers, each corresponding to different resolutions of the input image. This approach ensures that the model learns from features at multiple scales. The total loss is then computed as a weighted sum of the losses at all resolutions, facilitating effective segmentation of objects of varying sizes.

- Postprocessing: Implement postprocessing techniques such as morphological operations to refine segmentation results. These methods are especially useful for correcting small areas of false negatives and improving the overall accuracy of the segmentation.

- Train a Separate Model for Tumor: Currently, the model's weights are tuned simultaneously for segmenting both kidneys and tumors, except for the final 1x1x1 kernel convolution block. Training a dedicated model specifically for tumor segmentation to prevent the model from prioritizing kidney predictions over tumors, thereby enhancing tumor detection specificity.

- Refine Model Architecture: Integrate advanced components such as attention layers or experiment with different architectures like DenseNets (Huang et al. 2017) or SE-Nets (Squeeze-and-Excitation Networks) (Hu et al. 2018). These enhancements can improve the model's focus on relevant features, thereby boosting segmentation performance.

References

Bucher, Maxime, Stéphane Herbin and Frédéric Jurie (2016). 'Hard negative mining for metric learning based zero-shot classification'. In: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*. Springer, pp. 524–531.

Chen, Liang-Chieh et al. (2017). 'Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs'. In: *IEEE transactions on pattern analysis and machine intelligence* 40.4, pp. 834–848.

Çiçek, Özgün et al. (2016). '3D U-Net: learning dense volumetric segmentation from sparse annotation'. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*. Springer, pp. 424–432.

Dai, Wei et al. (2018). 'Scan: Structure correcting adversarial network for organ segmentation in chest x-rays'. In: *International Workshop on Deep Learning in Medical Image Analysis*. Springer, pp. 263–273.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep learning*. MIT press.

Gorgolewski, Krzysztof J et al. (2016). 'The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments'. In: *Scientific data* 3.1, pp. 1–9.

He, Kaiming et al. (2016a). 'Deep residual learning for image recognition'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

— (2016b). 'Identity mappings in deep residual networks'. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, pp. 630–645.

Heller, Nicholas et al. (2019). 'The kits19 challenge data: 300 kidney tumor cases with clinical context, ct semantic segmentations, and surgical outcomes'. In: *arXiv preprint arXiv:1904.00445*.

Henson, Katherine E et al. (2020). 'Data resource profile: national cancer registration dataset in England'. In: *International journal of epidemiology* 49.1, 16–16h.

Hu, Jie, Li Shen and Gang Sun (2018). 'Squeeze-and-excitation networks'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141.

Huang, Gao et al. (2017). 'Densely connected convolutional networks'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

Isensee, Fabian and Klaus H Maier-Hein (2019). 'An attempt at beating the 3D U-Net'. In: *arXiv preprint arXiv:1908.02182*.

Isensee, Fabian, Jens Petersen et al. (2018). 'nnu-net: Self-adapting framework for u-net-based medical image segmentation'. In: *arXiv preprint arXiv:1809.10486*.

Kutikov, Alexander, Marc C Smaldone et al. (2011). 'Anatomic features of enhancing renal masses predict malignant and high-grade pathology: a preoperative nomogram using the RENAL Nephrometry score'. In: *European urology* 60.2, pp. 241–248.

Kutikov, Alexander and Robert G Uzzo (2009). 'The RENAL nephrometry score: a comprehensive standardized system for quantitating renal tumor size, location and depth'. In: *The Journal of urology* 182.3, pp. 844–853.

Li, Yuanzhi, Tengyu Ma and Hongyang Zhang (2018). 'Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations'. In: *Conference On Learning Theory*. PMLR, pp. 2–47.

Lin, Tsung-Yi et al. (2017). 'Focal loss for dense object detection'. In: *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988.

Liu, Xiangbin et al. (2021). 'A review of deep-learning-based medical image segmentation methods'. In: *Sustainability* 13.3, p. 1224.

Ronneberger, Olaf, Philipp Fischer and Thomas Brox (2015). 'U-net: Convolutional networks for biomedical image segmentation'. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, pp. 234–241.

Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus and Ali Gholipour (2017). 'Tversky loss function for image segmentation using 3D fully convolutional deep networks'. In: *International workshop on machine learning in medical imaging*. Springer, pp. 379–387.

Simmons, Matthew N et al. (2010). 'Kidney tumor location measurement using the C index method'. In: *The Journal of urology* 183.5, pp. 1708–1713.

Spaliviero, Massimiliano et al. (2015). 'Interobserver variability of RENAL, PADUA, and centrality index nephrometry score systems'. In: *World journal of urology* 33, pp. 853–858.

Tsuzuki, Toyonori et al. (2018). 'Renal tumors in end-stage renal disease: a comprehensive review'. In: *International Journal of Urology* 25.9, pp. 780–786.

Ulyanov, Dmitry, Andrea Vedaldi and Victor Lempitsky (2016). 'Instance normalization: The missing ingredient for fast stylization'. In: *arXiv preprint arXiv:1607.08022*.

Valverde-Albacete, Francisco J and Carmen Peláez-Moreno (2014). '100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox'. In: *PloS one* 9.1, e84217.

World Cancer Research Fund (2022). *Kidney Cancer Statistics*. Accessed: 2024-06-24. URL: https://www.wcrf.org/cancer-trends/kidney-cancer-statistics/.

Zabell, Joseph R et al. (2017). 'Renal ischemia and functional outcomes following partial nephrectomy'. In: *Urologic Clinics* 44.2, pp. 243–255.

Zanaty, EA and Said Ghoniemy (2016). 'Medical image segmentation techniques: an overview'. In: *International Journal of informatics and medical data processing* 1.1, pp. 16–37.

Appendix

A README

This project includes the code used to train 3D U-Nets on the KiTS19 dataset (210 training + 90 testing) and generate segmentations of kidneys and kidney tumours from any given 3D computed tomography images. The 3D U-Net architectures tested are based on the paper An attempt at beating the 3D U-Net, which explores plain, residual, and pre-activation residual 3D U-Nets. All the results shown in *report/yz870_project_28.pdf* can be reproduced by following instructions below.

Installation

To install this project, follow these steps:

1. Clone from the Gitlab Repository

```
$ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-science-mphil/projects/yz870.git
```

2. Create and activate new conda environment

```
$ conda env create -f yz870/environment.yml
$ conda activate cw_a2
```

3. Download the segmentations from the KiTS19 Data repository

```
$ git clone https://github.com/neheller/kits19
$ mv kits19/data yz870/data
```

4. Download the image data from a separate source

```
$ cd yz870
$ mv kits19/data yz870/data
```

Usage

Training: Before training, you need to amend *parameters.ini* according to the hyper-parameters you want to use (detailed explanation for what each parameter means is in *report/yz870_project_28.pdf*). To reproduce final cross-validation results, please keep other parameters as default and enter model name as '3d_unet_original' or '3d_unet_residual' or '3d_unet_pre_activation' and run:

```
$ python train_unet3d.py --param_file parameters.ini --gpu 0 --train_fold all
```

For parallel training, you need 5 GPUs for training 5 folds simultaneously, setting x to 0,1,2,3,4 correspondingly and run:

```
$ python train_unet3d.py --param_file parameters.ini --gpu x --train_fold x
```

Trained models will be saved in *models*, training logs will be saved in the *results* and visualized in the *figs* folder.

Inference: To generate predicted segmentations for a given 3D CT scan stored in NIFTI format (e.g., imaging.nii.gz), specify the model and its training configuration in the *parameters.ini* file, then execute:

```
$ python inference.py --param_file parameters.ini --image_file imaging.nii.gz --gpu 0
```

If the required models have been trained, predictions (prediction.nii.gz) will be generated upon execution. If no suitable model is found, an error will be raised, prompting you to train a model using the specified *parameters.ini* first.

The final 5-fold models for original, residual, and pre-activation 3d UNets can be downloaded via the link in *models/trained_model_link.txt*. After downloading the one you want to use, unzip the model archive and place it in the models folder. Then, you can generate predictions from pre-trained models by setting *parameters.ini* consistent with the subfolder name.

Development: To generate documentations, you need to install doxygen and run:

```
$ cd docs
$ doxygen
```

For further contributions, please install pre-commit hooks which can automatically check coding format.

```
$ pre-commit install
```

Data & Challenge result

210 training cases of the KiTS19 Challenge are used to train 3D UNets. Predictions for the 90 test cases using ensembled residual 3D UNet are generated in *predictions.zip*, uploading to the challenge, a final mean Kidney Tumor Dice of 0.7569 can be achieved.

GitHub Copilot

Copilot hasn't been shut down since it can enhance coding efficiency. However, it has been mostly used for generating repetitive codes, the overall structure and key ideas were predominantly created by myself. Codes fully generated by AI tools are all pointed out in the comments.

Running time

Running time changes when we change settings for number of epochs, batch size, number of augmentations, and etc. Using the optimal hyperparameters given in the default *parameters.ini*, a full 5-fold cross-validation on a single GPU takes about 6 days.

License

Author

Yichi Zhang (yz870)

29/06/2024

B Usage of generative AI tools

Github Copilot: Copilot hasn't been shut down since it can enhance coding efficiency. However, it has been mostly used for generating repetitive codes, the overall structure and key ideas were predominantly created by myself. Codes fully generated by AI tools are all pointed out in the comments.

ChatGPT: ChatGPT hasn't been used for any part of the code, except for some debugging and reformatting help. The report was first fully written without ChatGPT but further refinements in the delivery style of some paragraphs and the format of pseudocodes, tables and formulas were assisted by it. Instructions given to ChatGPT all fall into one of the following ways:

- Refine the following paragraph for clarity/conciseness.
- How to write a pseudocode in latex? please give an example.
- Change the following code for more readability.
- Does my understanding of something ... correct and comprehensive?
- Which comments in latex are for multiple-line formulas?
- ...

All the responses from these generative AI tools were carefully checked and fully understood before incorporating into the code/report.