Radiation Dose Prediction - MVA DLMI Course Team: CHAUVIN_MOUHLI

Paul CHAUVIN PAULCHAUVIN97@GMAIL.COM

Rayane MOUHLI RAYANE.MOUHLI@ENSAE.FR

Editors: Under Review for MVA DLMI Course 2023

Abstract

Radiation therapy is an efficient tool in the treatment of cancer. However, delivering high doses of radiation to cancerous tissues while minimizing the exposure of surrounding healthy tissues remains a significant challenge. In this work, we present a deep learning model to predict the radiation dose using a patient's CT scan and the contours of the targeted organs and areas to irradiate. We have implemented two models: a U-Net architecture and a U-Net++ to learn the complex relationships between the input features and the radiation dose. We trained our model on a large dataset of patients and evaluated its performance on an independent test set. The U-Net++ is our best model and achieves a score on CodaLab of 0.3305.

Keywords: Radiation therapy, U-Net, Generative Networks , Medical Imaging

1. Introduction

Radiation therapy is an effective treatment for cancerous tissues, but the treatment is only safe and effective if the surrounding healthy tissues are preserved and if the right amount of radiation is emitted. Therefore, predicting the radiation dose prior to the actual treatment is a critical step in designing treatment plans. In this competition, we aim to train a deep learning model that can predict the radiation dose using a patient's CT and the contours of the targeted organs and areas to irradiate. We use a U-Net-like architecture to achieve this goal.

U-Net (Ronneberger et al., 2015) is a convolutional neural network architecture that was initially proposed for biomedical image segmentation. It is designed to capture the high-resolution features of the input image while maintaining the spatial information of the image. We chose U-Net because it is well-suited for medical imaging tasks and has shown promising results in similar tasks. Even though it is more common to use this architecture to perform segmentation, we can also consider it as a generative network since it is built with an encoder (the descending blocks) and a decoder (the ascending blocks). An improved version of U-Net was published in 2018 called U-Net++ (Zhou et al., 2018). Hence, we decided to implement this model too, and compare both results.

2. Architecture and methodological components

The U-Net has an auto-encoder structure with skip connections. The encoder is designed to extract features from the input images, more precisely at each downsampling step we double the number of feature channels while we cut in half the spatial dimensions. The decoder is designed to construct the segmentation map by using those extracted spatial features. In our case, the encoder outputs a learned feature map from the CT scan and the masks, then the decoder constructs the dose prediction. The U-Net concatenates parts of the encoder features with the decoder, which is known as the skip-connection in ResNet (He et al., 2015). This concatenation operation is useful for the decoder to capture the possible lost features by the max-pooling. This model is interesting in our case since we want to learn different levels of a CT scan through the masks.

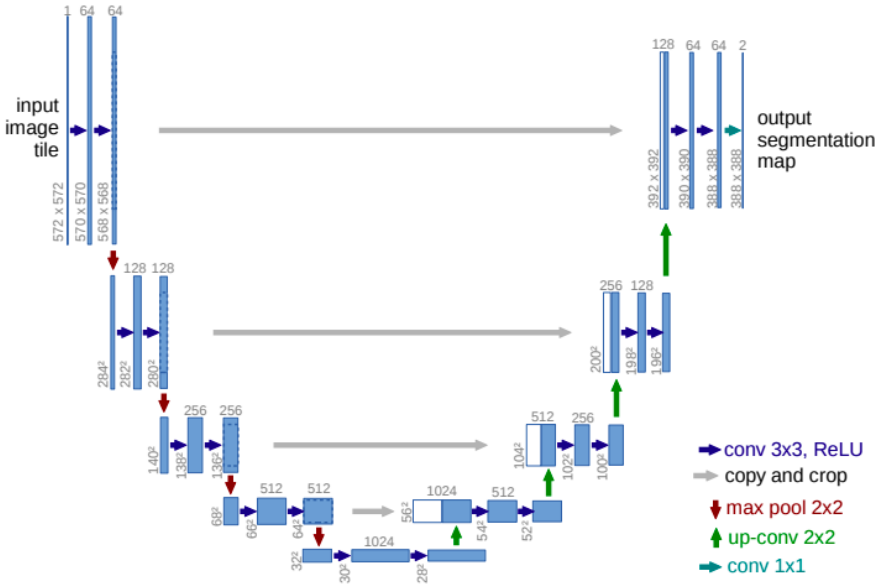

Figure 1: U-Net architecture

On the other hand, the U-Net++ architecture is an improvement over the U-Net. What distinguishes the U-Net++ from the classic U-Net is the way of connecting the encoder part with the decoder part. In U-Net, the feature maps of the encoder are directly inputted in the decoder, i.e there are skip connections. In U-Net++, the skip connections are replaced by convolution blocks. The convolution block brings the semantic level of the encoder feature maps closer to the feature maps in the decoder. The idea is that the optimization problem would be easier to solve if the encoder feature maps and the corresponding decoder feature maps are semantically similar. U-Net++ has been evaluated on multiple medical image segmentation tasks, such as nodule, nuclei, liver, and polyp segmentation, and the literature shows that it outperforms U-Net and wide U-Net (Xia and Kulis, 2017) architectures.

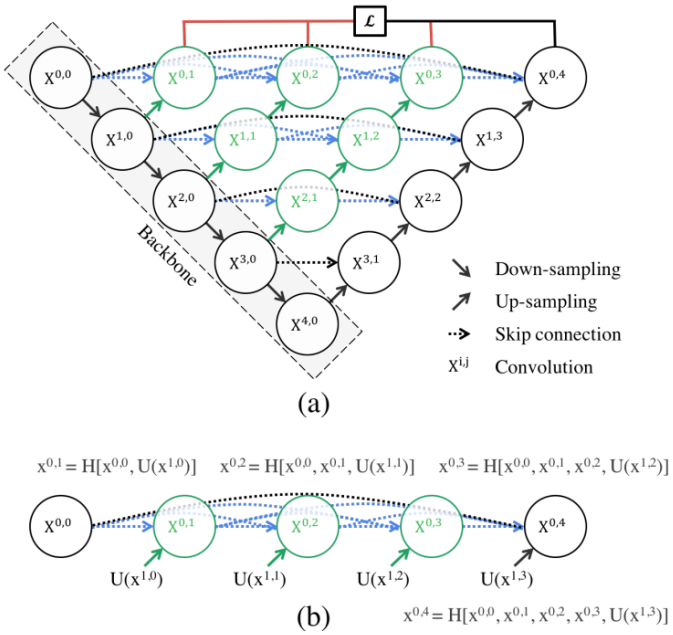

Figure 2: U-Net++ Architecture

The U-Net++ that we implemented has the same architecture as the one presented in 2. The blocks $X^{i,k}$ are VGG blocks : [Relu, 2D convolution, BatchNorm, 2D convolution, BatchNorm, SqueezeExcite]. We decided to add a SqueezeExcite layer (Hu et al., 2017) composed of an Adaptive Averaging Pool, a Linear and a ReLu layer. The idea of this layer is to add parameters to each channel of the VGG blocks so that the network can adaptively adjust the weighting of each feature maps. Finally, we add a convolutional layer as the output of our model. The output channels of the descending blocks are [64, 128, 256, 512, 1024].

We tried to modify the numbers of channels by decreasing them to [32,64 ,128 ,256 ,512] and to [16, 32, 64, 128, 256]. The number of trainable parameters are 2 324 081, 9 281 889 and 37 098 689 when we increase the numbers of channels. The consequence is that it has a huge impact on the computational time, for example with [32, 64, 128, 256, 512] the training time is about 2h30 while it is about 5h for [64, 128, 256, 512, 1024]. We trained the three models with all other things equal otherwise and we report the results in the table 1 after 40 epochs. For the last model, since we have an important number of parameters, we had to lower the batch size from 64 to 16 because of GPU limitations.

Dataset	**[16, 32, 64, 128, 256]**	**[32, 64, 128, 256, 512]**	**[64, 128, 256, 512, 1024]**
Train	0.31	0.25	0.13
Validation	0.46	0.41	0.38

Table 1: Comparison of the numbers of channels

We observe that the performance of the low channels numbers are worst than the last one. It can be explained by the fact that there are not enough channels to capture efficiently all the features of our model in the end. We couldn't experiment with a [128, 256, 512, 1024, 2058] configuration because of memory limitations.

To improve the performance, we tried to increase the numbers of epochs to force the model to overfit and then doing some regularization. To do so, we added 15 more epochs and dropout layers (Srivastava et al., 2014) after each VGG blocks to regularize the model. However with 55 epochs we still don't have an increasing validation loss and we can't get further than 55 epochs because of Google Colab GPU limitations.

Possible alternative architectures that could have been used as the SegNet (Badrinarayanan et al., 2015) architecture, which is another encoder-decoder network that uses max-pooling indices from the encoder to perform upsampling in the decoder. We also thought about CycleGAN which is used for unpaired generative networks, to adapt it to our problem. However, we decided to focus on the U-Net like architecture because this model has already proved its worth.

3. Model tuning and comparison

Since we have already discussed about the blocks of the U-Net++ architecture, we will see the influence of the learning rate and the batch size on our prediction.

We chose to work with the Adam optimizer, but we also experimented the SGD. We didn't notice significant differences between both optimizers. So, we worked with Adam for all of our experiments.

Our best model runs with a learning rate of 10^{-4}. We tried to lower it to 10^{-2} and we reached a training loss of 0.14 and a validation loss of 0.4 while our best model has a slightly better results on the training and validation set with 0.13 and 0.38 respectively. After plotting the training and validation loss, we observed that the training loss has two main levels around 0.45 and 0.25. So, we decided to use a scheduler with the Pytorch function ReduceLROnPlateau. The idea is to decrease the learning rate when the loss stops decreasing. Indeed, in this case, we want to do smaller steps when the function doesn't decrease in order to not miss the minima. So, we started with a learning rate of 10^{-2} and we let the scheduler regularizes it. However, we didn't observe significant improvements of our model.

To tackle the dataset, we started by stacking the masks and the CT scan but we got poor results. So we decided to do some preprocessing in order to mix the mask information with the CT scan. To do so, we apply the possible_dose_mask on the CT scan to create a new CT scan. Then, we applied each mask on the new CT scan by doing a pixelwise multiplication. The size of the input data are (12, 128, 128). We tried to do data augmentation by transforming the data with horizontal/vertical flips, rotations and normalization but we

doesn't notice significant improvement on the results.

After experimenting many configurations, we will now present the model that gave us the best results on CodaLab. The model is a U-Net++ with [64, 128, 256, 512, 1024] as channel numbers with a batch size of 64. We trained our model during 40 epochs with the Adam optimizer and a learning rate of 10^{-4}. We reach a score of 0.3305 on CodaLab with this configuration. The training and the validation loss for this model are presented in the figure 3. The results are reported in the table 2 below.

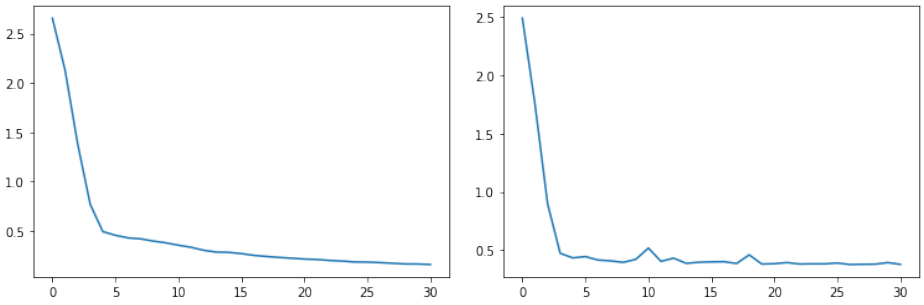

Figure 3: Training and Validation loss curves of the U-Net++ model

Our first idea was to implement a simple U-Net neural network built with 4 descending blocks (channels go from 64 to 512), a bottleneck, 3 ascending blocks and a convolution layer as output. We report the results in the table 2 below. We tried to add 1 block on the ascending and descending part, and we also experimented differents number of channels but the best configuration is the one we have just presented.

Dataset	**U-Net**	**U-Net++**
Train	0.27	0.13
Validation	0.56	0.37
Test (Codalab score)	0.37	0.33

Table 2: Comparison of the U-Net and the U-Net++

We observe that we have similar results on the training set, but the U-Net++ model has better performance when it comes to generalize to test and validation dataset.

Finally, we tested the robustness of our model. After the training, we computed the L1 loss on each batch of the validation set with shuffling to measure the standard deviation. We obtained a standard deviation error of 0.02 so we can conclude that the model has a low discrepancy.

References

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2015. URL https://arxiv.org/abs/1511.00561.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017. URL http://arxiv.org/abs/1709.01507.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Xide Xia and Brian Kulis. W-net: A deep model for fully unsupervised image segmentation, 2017. URL https://arxiv.org/abs/1711.08506.

Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation, 2018. URL https://arxiv.org/abs/1807.10165.