PANCREAS CANCER SEGMENTATION WITH A 3D U-NET

N. van den Hork, M. Mínguez Carretero, M. Moons, M. Schilpzand

Radboud University
Nijmegen, The Netherlands

ABSTRACT

In this work we describe our participation in the medical decathlon challenge for the specific task of pancreas segmentation. The objective is to segment pancreas and cancer from CT-Scans. In order to do this, Neural Networks have been used to create these predictions. Specifically, the 3D-Unet model has been used due to its state-of-the-art results. In total, the model has been trained for approximately 60 hours for a total of 22 epochs. An auxiliary dataset containing patches extracted from the original scans has been generated and used to train the model. The segmentation was somewhat successful. We obtain dice scores of 0.99, 0.63 and 0.28 for background, pancreas and cancer respectively on validation patches and similar scores with a less trained model on complete test volumes. Unfortunately, we were not able to submit predictions for the test volumes with our final model due to technical problems at grand challenge.

1. INTRODUCTION

The aim of this paper is to segment pancreas and cancer from CT-scans using the Pancreas Decathlon dataset [1]. The task of segmentation is best performed by convolutional neural models called U-Net models. Since the CT-scans are 3D images we use a 3D U-Net model for this segmentation challenge. The most difficult aspects of this challenge were to effectively preprocess the data for our model, post process the outcome of the model to useful predictions and handling the class imbalance in the data. Compared to the background voxels there are only very little pancreas and cancer voxels in the scans.

The Pytorch [2] framework has been used to create the neural network since it reduces the time needed to create a model and at the same time offers flexibility to modify the architecture to suit our needs. Furthermore, Pytorch allows us to train the models on the GPU using Google Colab platform [3].

[1]http://medicaldecathlon.com/
[2]https://pytorch.org/
[3]https://colab.research.google.com/

2. RELATED WORK

2.1. U-Net

The U-Net architecture is a network architecture that was developed in 2015 by Ronneberger et al.[1] for segmentation of neuronal structures in electron microscopic stacks. The U-net architecture has a distinct U-shape, which consists of a contracting path that is used to capture context and an expanding path that enables precise localization. This architecture depends on heavy data augmentation, but does not require a lot of training data. Furthermore, the U-Net is found to perform fast as it is capable of performing segmentation on 512x512 images in less than a second. The network is built on a fully-convolutional network architecture, but it is extended such that it requires few training images and yields more precise segmentations. In the upsampling part of the network there are a large number of feature channels to allow the network to propagate context information to higher resolution layers, which also results in the distinct U-shape of the network.

2.2. 3D U-Net

An extension of the U-Net of Ronneberger et al. was created by Cicek et al.[2], which replaces all 2D operations with 3D counterparts. Segmentation of volumetric biomedical data causes some difficulties, as neighbouring slices provide almost identical information. The architecture has a contracting and an expanding path with four resolution steps, just like the original network. The advantages of this 3D U-net are that it can be trained from scratch on sparsely annotated volumes and it can work on arbitrarily large volumes. A different 3D U-net architecture was designed by Isensee et al.[3], who designed a network architecture for brain tumor segmentation as a contribution for the 2017 brain tumor segmentation challenge. This network achieved state-of-the-art results on BraTS2015 and was one of the leading algorithms on the BraTS 2017 validation set after it was released. When classifying whole tumor, tumor core and enhancing tumor it achieved decent dice scores (i.e. respectively 0.896, 0.797 and 0.732 on the validation set and respectively 0.858, 0.775 and 0.647 on the test set). The network is similar to both [1] and [4], although there are some differences in design choices for among others the context pathway architecture, number of feature maps in the network and the structure of the upsampling pathway. The network uses twice as many filters as the network of [4], but it is trained with a slightly smaller input patch size and a larger batch size.

2.3. Medical Segmentation Decathlon

The Medical Segmentation Decathlon (MSD)[4] is a challenge that aims towards increasing the understanding in generalisability of proposed models for medical image segmentation. The aim is to shift the focus towards algorithms that can be used out-of-the-box on many tasks, as this would greatly benefit healthcare. A variant of the uNet that has also proven itself in the MSD challenge is the nnU-Net[5]. This network achieved the best results in the Decathlon Grand Challenge scoring a mean position of 1.5 which was 2.4 better than the second best model (the K.A.V.athlon). The fact that it performs so well on the MSD indicates that the network is applicable in multiple medical image segmentation domains with no modifications between these domains. In the spirit of the MSD, the authors present a robust and self-adapting framework rather than a specific network specified for a certain domain, arguing that one should focus on generalizability of a method.

3. DATA

The pancreas data was collected from the Decathlon Challenge [6] and consists of 281 labeled training CT-scans and 139 unlabeled test scans. The labels consist of three classes: Background, pancreas and (pancreatic) cancer. The data is very imbalanced. For the training data the median percentages of background, pancreas and cancer are 99.8, 0.19 and 0.01 respectively. We counteracted this imbalance by using stratified sampling and by using appropriate loss functions. The CT-scans have a shape of $512 \times 512 \times Z$ voxels, with Z changing between volumes. The value for Z ranges from 37 to 192 with a median of 96. There is one outlier in the test set which has a Z of 683.

3.1. Preprocessing

Preprocessing has been one of the more time consuming tasks required to obtain good results. Many different aspects had to be considered, which are described in this section.

Metadata about a CT-scan is included for both images and labels. This metadata contains the voxel spacing in each dimension. Before we can use the data to train for segmentation purposes, the scans are resampled such that the voxel spacing is the same for every scan. The spacings for all volumes are set to [0.8, 0.8, 2.5], since this was close to the mean spacing of the volumes.

[4]http://medicaldecathlon.com/results.html

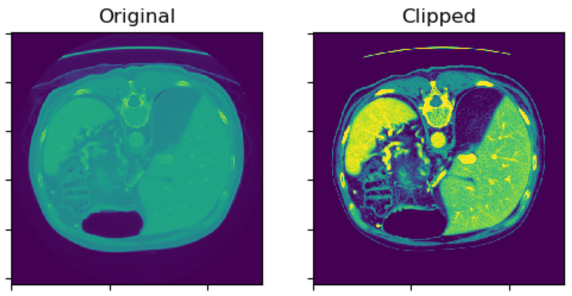

Fig. 1. On the left, the non clipped image. On the right, the image clipped at [-100, 200].

The images contain pixel values ranging from -1024 to 3071. As can be seen in the Hounsfield scale [7] many of these values represent irrelevant matter for our segmentation objective. The values outside the range of [-100, 200] are clipped. -100 Is in the range of fat tissue and 200 is slightly lower than the lowest Hounsfield value for bone. This clipping is done to get more resolution in the [-100,200] range, as this is where most tissue relevant to locating the pancreas. Figure 1 shows the difference of a sample with and without clipping the image values.

Since the images are very large and have different shapes, the neural network cannot process the CT-scans as they are presented in the original dataset. Processing these large scale images would additionally be computationally expensive and not efficient for training a network. A general solution to this problem is to sample fixed sized patches from the images. The shape of the patches was chosen as $128 \times 128 \times 64$. In case a CT-scan had less than 64 depth (Z axis) the patch is padded with the values of the edges to expand the size of the patch.

Sampling patches from the scans also allows us to control the class imbalance for the training set. Since the percentage of pancreas and cancer in the data is extremely small, a patch dataset would likely not consist of enough varied data containing pancreas or cancer. To compensate for the class imbalance the patches are sampled so that at least 30% of them contain pancreas voxels. For a higher percentage there is a risk that the distribution of the training data could become significantly different from the distribution of the test data. Another benefit of not using a higher percentage is that the model will learn to recognize other organs as background correctly.

In order to create a representative patch dataset, the volumes are given a sampling probability based on their size. These sampling probabilities are derived by dividing the number of voxels in a scan by the total number of voxels in the training set. When a new patch is made it samples from a volume according to these probabilities. This way the larger volumes are sampled from more often than the smaller volumes.

Lastly, a split of the training data into training and validation data is necessary for evaluation purposes. Since it is bad practice to have overlap between validation and training data, we designate 71 training volumes to be used for sampling validation patches and the other 210 to be used for sampling training patches. We then normalise the training and validation patches by subtracting the mean and dividing by the standard deviation of the training patch data set. The same normalisation is later applied to the test set as well.

3.2. Patch dataset

Since calculating the patches of the images is a costly task, it has been decided to create a dataset containing the necessary information to be used later in the training phase. Specifically, the data has been stored in a *hdf5* file because, instead of loading the entire dataset into memory, it allows us to retrieve patches at the moment we need them. There are separate *hdf5* files for training and validation patches. Every instance of such a file contains the image and label data of a patch. The image and label data is saved in small data types in order to reduce the dataset size.

Regarding the test set, it was decided not to create an additional test set but to compute patches from the test images on the fly to then use them as the input of the trained model. Furthermore, the predictions are retrieved and stitched together to create the predictions for the 3D test volumes.

3.3. Stitching of patches

Since the model is trained exclusively on patches, additional steps must be taken in order to segment an entire volume. A volume is divided into patches and for each patch a prediction is made using the model. These predictions are then stitched together to construct the full volume. To prevent edge artifacts from occurring in the prediction volume, we use only the centers of the patches, this also means that any artifacts stemming from same convolution will be removed.

4. TRAINING

In this section we will discuss the model, the choice of loss function, evaluation metric and hyperparameter optimization. For the training of the model 8000 training patches are independently sampled from the first 210 scans and 2000 validation patches are sampled from the last 71 scans. Due to memory constraints a batch size of two is used for both training and validation. The Adam algorithm [8] is used for optimization with an initial learning rate of 0.001 and every epoch the learning rate decreases with a decay coefficient of 0.985.

4.1. Model Architecture

The architecture of the model that was used is a PyTorch implementation of the architecture described in the work of Isensee et al.[3], an implementation of it was published in a

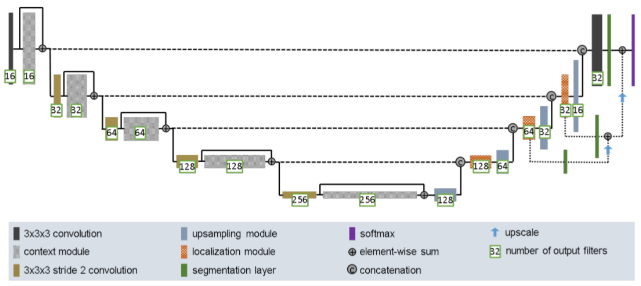

Fig. 2. A visual representation of the 3D U-Net architecture as described in section 4.1, this image was taken from [3].

Github repository by Po-Yu Koa[5]. Just like in the U-Net[1], the architecture (figure 2) consists of a context aggregation pathway that encodes representations of the input that become more abstract down this pathway, followed by a localization pathway that recombines such representations with shallower features to precisely localize these structures of interest. Higher levels in the U-Net (vertical depth of the U-shape) have lower spatial resolution, but higher dimensional feature representations. Activations in the context pathway are computed by context modules, where each of these modules is a pre-activation residual block with two 3x3x3 convolutional layers and a drop-out layer in between. The resolution of the feature maps between the context modules is reduced by connecting the context modules with 3x3x3 convolutions with input stride 2 while descending down the aggregation pathway. The localization pathway that is followed by the context aggregation pathway is designed to take features from lower levels of the network that encode contextual information at a lower spatial resolution and encode them. In order to achieve this, the feature voxels are upscaled twice in each spatial dimension, which is followed by a 3x3x3 convolution that halves the number of feature maps. The upsampled features are recombined with features from the same level of the context aggregation pathway via concatenation. After concatenating, features are recombined using a localization module that consists of a 3x3x3 convolution followed by a 1x1x1 convolution that halves the number of feature maps. All feature map convolutions in the original design of the network are computed using leaky ReLU nonlinearities with a negative slope of 0.01. Batch normalization was replaced with instance normalization because it was found that the stochasticity induced by small batches may destabilize batch normalization.

4.2. Loss Function

One of the main difficulties of this segmentation challenge is the large class imbalance as described in section 3. The choice of loss function is primarily motivated by this class

[5]https://github.com/pykao/Modified-3D-UNet-Pytorch

imbalance. For this segmentation task we try two loss functions: Weighted cross entropy and top-k cross entropy loss.

4.2.1. Weighted cross entropy loss

Weighted cross entropy loss is the same as cross entropy, but with the addition of class weights. The weight of a class increases the loss in case the prediction does not predict this class when it should have. This way it stimulates the model to predict large weight classes. For the weighted cross entropy loss we use the pytorch implementation, which combines log softmax and negative log likelihood. The loss of a patch is calculated as the mean of the loss of all voxels. The loss of a single voxel that has target class *C* is calculated as:

$$loss(x, C) = w_C\Big(-x[C] + \log\sum_i \big(\exp\left(x[i]\right)\big)\Big) \quad (1)$$

Where w_C is the class weight for class *C* and *x* is the one hot encoded prediction for the voxel.

To find the right weights we empirically tested with a small data set. Originally we used weights which were directly proportional to the inverse of the class distributions. This resulted in the class weights [1, 525, 9980]. This did not show good results and seemed to overvalue the pancreas and the cancer class. We eventually got decent results with weights [1, 10, 50] and decided to use these weights to train on the complete data.

4.2.2. Top-K Loss

This implementation of top-k loss is similar to what DeepMind used for clinical segmentation [9]. The voxel loss is calculated with equation 1 with equal class weights. However, in this case only the top-k voxel loss values are considered. These top-k loss values are again averaged to obtain a single loss value. This could compensates for the class imbalance since only the voxels with the highest cross entropy losses are considered. We considered a top-k percentage of 5% for our segmentation task.

4.3. Evaluation

To find the right hyperparameters and loss function for this segmentation task, we needed an evaluation metric different from the loss function itself. Due to the large class imbalance accuracy was not very helpful. Instead, the dice score per class was used as an evaluation metric. The formula for calculation of the dice score is given by:

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

Due to the class imbalance it can occur that there are no pancreas or cancer voxels in a batch. This results in a dice score of zero, even though the model might also predict no voxels in those classes, making it a perfect prediction. We should remark that an extremely small lower bound is used for the denominator to avoid division by zero. Since the dice score is also not the perfect metric, part of the predictions are also manually evaluated. This can quite quickly give an indication if the model is either predicting too little or too much of the low frequent classes.

4.4. Hyperparameter Optimization

Some of the hyperparameters of the network have been optimized using Bayesian Optimization (BO) from the Hyperopt library[10], which is an optimization technique that samples parameters within a designated space and tries to find the optimal configuration of parameters. As a new network has to be trained for every set of parameters, this optimization process can be very time-consuming. Since we want to keep the amount of evaluations as low as possible, BO is a useful technique for fine-tuning the parameters of neural networks, as previous parameter configurations are taken into account when sampling a new parameter configuration. This makes BO favourable over grid search and random search to estimate the optimal parameters for this network. A parameter space is created which contains all the parameters of the network that should be optimized and a range of values in which the optimal value for those parameters is expected to be in. The optimized parameter space is follows; dropout probability (0.2, 0.4 or 0.6, uniformly sampled), activation function (ReLU or Leaky ReLU, with uniform sampling) and the class weights (w2: 50-250, w3: 250-1000; both with uniform sampling). Due to limited time for parameter optimization, the learning rate was not optimized for the network in this process, as the performance of the learning rate appeared to be strongly dependent on the amount of epochs the network was trained on (a bias towards a high learning rate was expected with a low amount of epochs). For the first 15 evaluations, 15 random parameter configurations were sampled from the parameter space, similar to random search. Each network with these sampled configurations was trained using a training set of 100 patches and a validation set of 30 patches on 5 epochs. The performance of the network configuration was evaluated with the dice scores of these networks on pancreas and cancer tissue. The parameters of subsequent trials were sampled closer to the values of the earlier sampled configurations that were found to perform better using a Tree-of-Parzen Estimators algorithm[11] as search algorithm. After evaluating 95 different parameter configurations, the optimal configuration that was found consisted of the Leaky ReLU activation function, a drop-out of 0.6 and class weights of w2=83, w3=660. A second optimization was performed for optimizing the weights (w2: 2-25; w3:5-50) and k-value parameters (0.05, 0.1) for Top-K loss, after 200 evaluations (each consisting of 5 epochs) the best configuration was found to be k=0.05, w2=13, w3=43. Dice scores of pancreas and can-

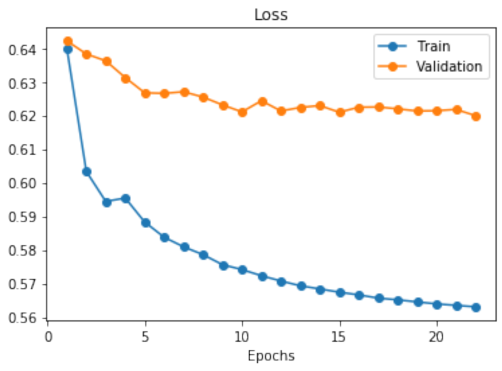

Fig. 3. Epoch loss for the training and validation set. The epoch loss is calculated by averaging all batch losses in an epoch.

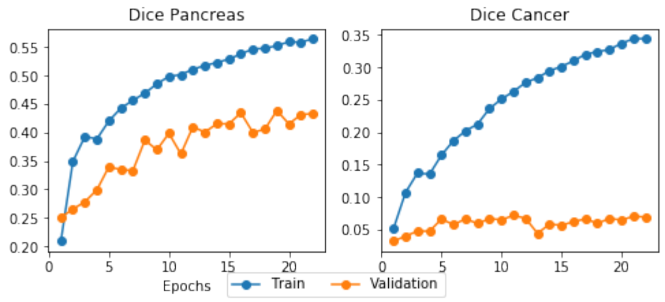

Fig. 4. Average dice scores of the train and validation set per epoch for the pancreas and cancer class.

cer were used as an evaluation metric, so the values for w2 and w3 that were found after the first run are likely to be too high estimates. Due to the low amount of epochs per network configuration evaluation, these results were used more as a guideline. As for the network using top-k loss, a network configuration without providing class weights was found to perform better than giving the network weights.

5. RESULTS

The training time of one epoch was around three hours. Therefore, we could not test as many parameter combinations and loss functions as we would like. We tested on smaller sets and used the Bayesian Optimization as an indication of what values to choose for training on the complete data. We extensively trained the most promising configuration which consisted of weighted cross entropy loss with class weights [1, 10, 50] for 22 epochs. The loss is shown in figure 3. Although it is hard to see because of the minor differences, the last epoch did have the lowest validation loss. Since it is unknown if the loss function with these weights is ideal for this segmentation task, we evaluate the dice score per class. The dice scores for pancreas and cancer are shown in 4. We did not add the background dice since this was generally around 0.99 and therefore had no added value. The validation dice scores of the last epoch are 0.43 for pancreas and 0.069 for cancer. These scores were obtained by averaging the dice scores over batches consisting of two patches. Note that, as was explained before, the dice score of a class is zero if there are no voxels of that class. This of course lowers the average dice score. To avoid this effect, the dice scores were also calculated over 100 validation batches (200 patches) at once. This resulted in a pancreas dice score of 0.63 and a cancer dice score of 0.28. The complete results of these validation predictions are shown in the confusion matrix in figure 5.

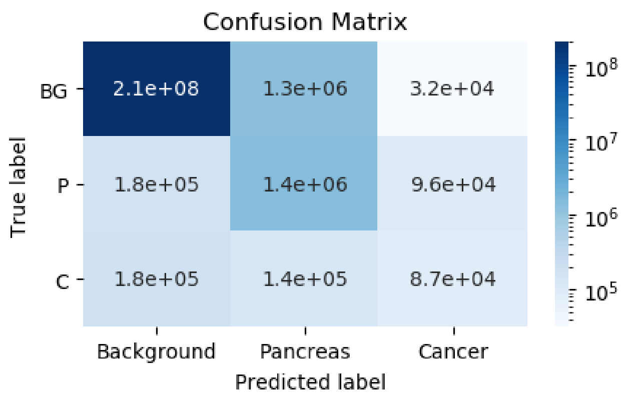

Fig. 5. The confusion matrix for 100 random batches of the validation set.

To give a visual indication of the model's performance figure 6 shows the predictions of three slices from the same training volume with the corresponding labels and images. We noticed that there are no signs of overfitting yet and the model's performance could still increase with more epochs. Unfortunately, we did not have the time and resources to train this model longer. In the end we save two models both with high dice scores to perform segmentation on the test scans. These are the models after 19 epochs and the model after the complete training (22 epochs).

5.1. Top K loss

We attempted to train a model with the top-k loss function with a top-k percentage of 5%. Unfortunately, this model predicted mainly background. To guarantee that the top-k voxel losses would contain pancreas and cancer, class weights [1,2,3] were added to the loss function. This led to extremely poor predictions. A lot of background was wrongly predicted as pancreas and cancer and also the pancreas and cancer dice scores itself were still relatively low. Lowering the top-k percentage and using no class weights might improve results, however, after the failed attempts we decided to focus our time on the weighted cross entropy implementation.

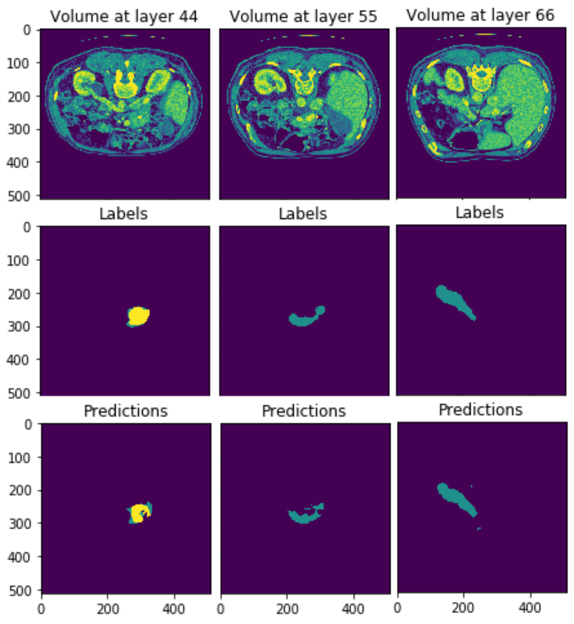

Fig. 6. Segmentation for the pancreas. (Green is pancreas, yellow is cancer, purple is background)

5.2. Grand Challenge

We submitted our predictions for the test scans to the decathlon-10 challenge[6] for evaluation. Unfortunately, our submissions as well as those from other fellow students from the ISiMI course were removed and new ones were not added to the leaderboard. This might be due to our submission containing only relevant data for the pancreas segmentation and not for the other nine organs. We submitted test predictions with one of our earlier models, after training for 12 epochs, and this resulted in a dice score 0.60 for pancreas and 0.30 for cancer. There were not many (pancreas) submissions at this point and this placed us third for pancreas dice and first for cancer dice score. In later stages of training we were able to obtain better average validation scores, therefore, we expect that this would also translate into better results on the decathlon grand challenge leaderboard. However, the test dice scores were very similar to the scores obtained by training the final model on the 100 validation batches. Unfortunately, there is no way to figure out if the final model would perform better on the test scans.

6. CONCLUSIONS

Performing segmentation on large 3D volumes with substantial class imbalance is not an easy task. With a 3D Unet and weighted cross entropy loss we managed to somewhat successfully segment pancreas and cancer tissue from 3D CT-scans. After 60 hours of training we obtained a validation dice score of 0.99, 0.63 and 0.28 for background, pancreas and cancer respectively on patches and earlier iterations of our model obtained dice scores of 0.60 and 0.30 for pancreas and cancer on complete test volumes. Our validation results do no show clear signs of overfitting yet which indicates that the model could perform better with more training time. Unfortunately, we were not able to train our model longer due to time constraints.

[6]https://decathlon-10.grand-challenge.org

7. REFERENCES

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[2] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger, "3d u-net: learning dense volumetric segmentation from sparse annotation," in *International conference on medical image computing and computer-assisted intervention*. Springer, 2016, pp. 424–432.

[3] Fabian Isensee, Philipp Kickingereder, Wolfgang Wick, Martin Bendszus, and Klaus H Maier-Hein, "Brain tumor segmentation and radiomics survival prediction: contribution to the brats 2017 challenge," in *International MICCAI Brainlesion Workshop*. Springer, 2017, pp. 287–297.

[4] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt, "Cnn-based segmentation of medical imaging data," *arXiv preprint arXiv:1701.03056*, 2017.

[5] Fabian Isensee, Jens Petersen, Andre Klein, David Zimmerer, Paul F Jaeger, Simon Kohl, Jakob Wasserthal, Gregor Koehler, Tobias Norajitra, Sebastian Wirkert, et al., "nnu-net: Self-adapting framework for u-net-based medical image segmentation," *arXiv preprint arXiv:1809.10486*, 2018.

[6] Amber L. Simpson et al., "A large annotated medical image dataset for the development and evaluation of segmentation algorithms," *CoRR*, vol. abs/1902.09063, 2019.

[7] Richard Wendt, "The mathematics of medical imaging: a beginner's guide," 2010.

[8] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] Stanislav Nikolov, Sam Blackwell, Ruheena Mendes, Jeffrey De Fauw, Clemens Meyer, Cían Hughes, Harry

Askham, Bernardino Romera-Paredes, Alan Karthikesalingam, Carlton Chu, et al., "Deep learning to achieve clinically applicable segmentation of head and neck anatomy for radiotherapy," *arXiv preprint arXiv:1809.04430*, 2018.

[10] James Bergstra, Dan Yamins, and David D Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in science conference*. Citeseer, 2013, pp. 13–20.

[11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.