Advanced Machine Learning Project – Spleen Segmentations with a 3D U-net

Martín José Fernandes Bolaños
King's College London
MSc Medtech Innovation and
martin.fernandes_bolanos@kcl.ac.uk

Abstract—**This report contains explanations for all the tasks carried out during my final project on Advanced Machine Learning. During this project, segmentations of the spleen were produced using a 3D U-net. Various concepts learned during the semesters were applied, among them, data augmentation, parameters optimization, training with unlabeled data, calculation and visualization of uncertainty, and ensembles of models.**

I. Background

In deep learning, more specifically in the field of computer vision, semantic segmentations were introduced in 2015 with the U-net architecture proposed in the paper *'U-Net: Convolutional Networks for Biomedical Image Segmentation'* where cell segmentations were achieved with a better performance compared to previous methods at that time [1]. Although the architecture was originally made for 2D images, the concept can be expanded to 3D images by using 3D convolutions; this approach was used during this project to segment the spleen from CT images. The U-net architecture consists of two main parts, an encoding and a decoding part. In the original paper, the decoding part consisted of a series of double convolutions followed by a max pool layer. On the contrary, the decoder side of the network followed a similar approach but to recover the size of the input image up-convolutions were used followed by double convolutions. Finally, a 1x1 convolution was used to produce an output segmentation map that can be used to generate segmentations. Another crucial component of the U-net architecture is that feature maps from the encoding part are concatenated with the feature maps of the decoding map in a process described as *'copy and crop'* [1]. This allows the network to recover information that was lost due to the process of encoding and thus, produced accurate segmentations. This project is divided into different tasks, each of which will be explained in the following sections of this report. A Jupyter Notebook will be submitted with the code implementations.

II. Task 1

During this task, an initial segmentation algorithm pipeline was implemented using a 3D U-net architecture. The first step of this section was creating a class for the dataset that was going to be considered. The CT images and segmentations used were the ones located in the *'imagesTr'* and *'labelsTr'* folders. The Dataset class that was created (*SpleenDataset*) can take as argument transformations for data pre-processing for both the images and the labels. The data was resized to a shape of 64x128x128 to have consistency with my other group members for this project and to make sure that the model could take a pre-determined tensor size to be trained with. Moreover, this resizing process was used to reduce the size of the input tensors to save memory, time, and resources during the training process. Another pre-processing step was to normalise all the data as in deep learning a better performance can be achieved when all training data have similar statistical characteristics in terms of pixel intensities. For the training process, the size of the available dataset was 41 data samples; therefore, 30 images were used for training while 11 were used for validation purposes.

To write the code for the 3D U-net, an approach based on the code from Pepe Cantoral [2] was implemented with some variations, the main one being, that for this project 3D convolutions were going to be used instead of 2D convolutions. The first step for writing an architecture was defining a class called *MyConv* which was simply going to be a 3D convolution but with predetermined padding of 1, kernel size of 3 and stride of 1. This type of convolution allows matching the size of the input image with the size of the produced segmentation, something that was done differently in the original U-net paper [1] as padding was not used. The second step was to create a class called *DoubleConv* that consists of two *MyConv* each followed by a Batch Normalization layer and a ReLU layer. In the original U-net paper Batch Normalization was not used as it was a concept that was introduced in the same year [3], and it has been shown to produce better results during training. For the encoding part of the network, a class called *DownConv* was used, similarly to the original paper it consists of a 2x2 Max Pooling layer followed by the *DoubleConv* class. For the decoding part, a class called *UpConv* was used, in this one, instead of using a 3D up-convolution an *nn.Upsample* layer with a scale factor of 2 was used followed by a 3D convolution. This was followed by the *DoubleConv* class. Finally, the U-net architecture was created in the class *UNET* by using 4 *DownConv* where the number of channels was increased by a factor of 2 for each down convolution and 4 *UpConv* which reduced the number of channels by a factor of 2 for each up convolution. To achieve the feature maps concatenation used by this type of architecture, the class 'UpConv' used *torch.cat* in its forward method to achieve a good reconstruction of the segmentations.

For the training process of the network, a UNET with 8 initial channels was created. Differently from the original paper, 64 initial channels were not used as this approach would have taken more memory and more time to train. The optimiser used was Adam with a learning rate of 0.001 and the model was trained for 200 epochs. The instructions for this project were to use a Top-K loss when training, I tried to implement this loss function but my model was not optimising at all, therefore, as this is a function that can be used in models where cross-entropy loss is used I decided to continue the project forward using cross-entropy as I was using a lot of the available resources while trying to find the correct implementation for Top-K loss. It is important to note that the dice values that were printed during the training process are not correct, because when calculating them I forgot to use *argmax* when passing the prediction values to the dice function. However, in the code, the correct validation dice value is shown some cells below (right after I show the segmentations produced). The model performs with a validation dice of 0.8905266.

III. Task 2

During the second task of this project, data augmentation was performed. During this section, the *SpleenDataset* class was modified, so instead of taking as arguments transformations for images and labels independently, transformations could be applied equally to both images and labels. After this, a class called *CustomTransform* was created to perform augmentations. In this one rotation, translation, and shear were considered. I tried to include elastic deformations as well, but my code was not working, therefore, my augmentations for this task only include affine transformations. The parameters used for the augmentation was a range of ±10 degrees for rotations; the reason that a larger number was not used is that usually CT images are taken from the same direction, so it is difficult to find one that is rotated for example by 90 degrees. In the case of translations, I only chose a value of 10 pixels considering that the images were already resized to a shape of 64x128x128, so a number larger than this would have resulted in very large non-realistic distortions. Finally, for the shear, the angle range was ±10 degrees to avoid very large distortions.

For the training process using these augmentations, although more data was available, I used again 30 samples for training and 11 for validation. The only difference is that now data with affine transformations was going to be considered which makes the model more robust when passing images from other patients to it. The reason why no larger amounts of data were used was to avoid losing Google Colab resources and to not have an excessively long training time. To compare performance with the previous model, the same optimiser, loss function, and epochs were used. This time, again when printing dice values there is an error, as for the validation dice, I was printing the mean of all the dice coefficients being accumulated through the epochs. However, in some cells below, the validation dice score was calculated using the same validation data as the original model to make a fair performance comparison between models. This time the dice coefficient was similar to before but a bit lower with a score of 0.857428. The reason for this might be that this time the model was considering a lot of data with affine transformations, so the learning process of the network might be slower as it has to consider transformed data. To overcome this, probably more epochs of training would have been useful.

IV. Task 3

During this task optimisation of the network was carried out. As the network completed in task 1 was the one with a better validation dice, I chose to train this new network without the data augmentation that was carried out during task 2. When training the networks, I was noticing that for certain periods the training dice score was not getting better so I thought that it could have been because the algorithm was not converging rapidly to the solution as the learning rate was a bit high. Therefore, in this section, I decided to decrease the learning rate from 0.001 to 0.0001 so the model could converge better. However, this means that the model was going to also learn in a slower manner, due to this, to overcome this the second parameter that was optimised was the number of epochs which were increased from 200 to 250. Finally, the third parameter that was optimised was the number of initial channels/feature maps of the U-net. In the previous tasks, 8 channels were used. Ideally, 64 channels would have been great but that would have been very computationally expensive; because of this, the number of initial channels was increased to 10. After the training process was completed, using the same validation data that was used to evaluate performance in the networks of task 1 and task 2, the validation dice score that was obtained was 0.67880076. This is a lower score than the previous two networks and the reason for it could have been that by reducing the learning rate the network learns slower. Although the number of epochs was increased, to overcome this issue, the network could have been trained for even more epochs (maybe 400). However, due to limited computational resources, this was not done.

V. Task 4

During task 4 augmentation consistency was carried out. The theory behind it is that unlabeled data could be used to train the model. This approach is particularly useful when there is limited labelled data. Xie et al., use an approach where it is assumed that if unlabeled data is passed through a network, it should give the same output when the augmented version of the data is passed through the same model [4]. The loss between the output of the data and its augmented version is calculated and then weighted sum with the loss of the labelled data.

During this task, I applied a modified version of augmentation consistency. Instead of performing data augmentation to the labelled data, I passed the unlabeled images through the network that I created in the network of task 1 (which was my best model so far with a dice score of 0.89) and created segmentations from it. Then, I trained a model with the same parameters as the first network with both the labelled data and unlabeled data which was now labelled. Similarly to Xie et al. approach, this assumes that the new network will give the same output segmentations as the ones calculated using the network of task 1. When training this network with this approach I only used 100 epochs due to limited resources. Also, during the training process at some point, the network stopped optimising correctly and the training dice score was being almost 0. This persisted until around epoch 68 when the network's training dice was increasing again. This problem could have occurred due to the network's weight initialisation as sometimes when running the networks in the previous tasks the same issue was occurring, but I restarted the training. For this task training was not restarted as the resource availability of Google Colab was being low. However, in the future to overcome this problem more epochs could be used to let the network optimise properly or to perform a pre-determined way to initialise the weights before training. After the training process was completed, the validation dice obtained was 0.10887915 which is very low compared to the previous tasks.

VI. Task 5

In this task, the uncertainty of the model is measured to understand how confident the segmentation predictions are. To do this, the *UNET* class was modified and a dropout with a probability of 0.5 was applied for each *UpConv* and *DownConv*. Also, the Batch Normalisation layers were removed from the *DoubleConv* class to avoid a loss in performance as dropout is already being used. Once the training was finished with 200 epochs, the dice score was 0.8245312 which is a bit lower than the dice score of task 1 as dropout makes the training slower. To calculate the uncertainty, the validation data was passed to the model and the mean and the variance were calculated to measure the uncertainty. The uncertainty mean of the model was 0.0044 while the uncertainty variance was 0.0043. When visualising the uncertainty, an image from the validation was passed 10 times through the model while dropout was on and then the segmentations were summed and visualised with *plt.imshow*. By summing the segmentations, brighter values are obtained where the model is less uncertain, for instance, in the centre of the spleen, while darker values are seen where the model is more uncertain, for example, in the borders of the segmentation (as shown in Figure 1). The relationship between uncertainty and error is that a model with higher uncertainty will lead to a higher error. The reason for this is that a model with high uncertainty for a particular pixel has a higher chance of assigning the wrong label to it resulting in an error.

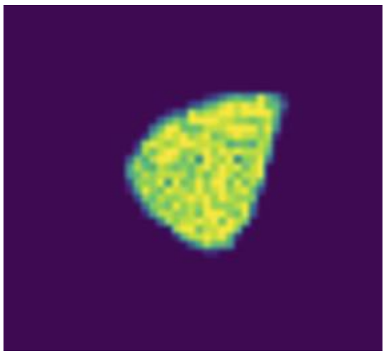

Figure 1. Visualisation of the uncertainty of the model of Task 5.

VII. Task 6

During this task, the segmentations were shared among the team members. To create an ensemble, model the segmentations were summed pixel-wise. As there were only 3 members who shared segmentations in my group (including myself) every pixel with a value of 2 was considered part of the spleen and every pixel that was 0 or 1 was not considered part of the spleen. This way a simple majority voting model was created. However, I think when I was saving my segmentations there was a problem with the saving order, therefore my segmentations do not match with the ones of my classmates, however, they are still good. Because of this, the ensembled model was only considering my classmates' segmentation in the majority voting and thus, the performance of this ensemble is worse than my model’s. This problem could be overcome if more segmentations from other members were available.

VIII. Task 7

In this final task, an ensembled model was created again; however, because everyone’s model performs differently, a weighted sum of the segmentations was carried out instead of a majority vote like in the previous task. The weights chosen were the dice scores reported by the people who shared their segmentations. This way models that perform better have a higher weight than those that perform worse. In this case, a threshold of 1.70 in the pixel value was used to determine whether a pixel is part of the spleen or not. This model performs exactly the same as the previous task and the reason for it again is that my segmentations were saved in a different order. However, I am almost certain that with more segmentations from different people, the model would have performed better.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015, pp. 234-241.

[2] P. Cantoral, "Segmentación Semántica - U-NET desde cero con PyTorch - Parte 2," YouTube, 2022. [Online]. Available: https://www.youtube.com/watch?v=x_cY9l1cwj0. [Accessed: Mar. 20, 2023].

[3] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in Proceedings of the International Conference on Machine Learning (ICML), 2015, pp. 448-456.

[4] Q. Xie, M. Luong, E. Hovy and Q. V. Le, "Unsupervised Data Augmentation for Consistency Training," in Proceedings of the Conference on Neural Information Processing Systems (NeurIPS), 2019, pp. 8393-8403.