Medical Images Segmentation

Chaszczewicz Alicja, Heinke Simon, Sycheva Anastasia
SS 2020, Machine Learning for Healthcare, Project 3

***Abstract*— In Project 3 we have analyzed 3D prostate magnetic resonance images from two data sources: Boston University and Radboud University, Nijmegen Medical Centre in the Netherlands. Our task was to perform image segmentation to outline two non-overlapping adjacent regions of the gland: peripheral zone and central gland. This report describes the main steps of our analysis and summarizes our results. It is structured as follows: in Section I we introduce the dataset, Section II details the data preprocessing pipeline. In Section III we shortly describe the neural network architecture, whereas Section IV contains all the details of our training procedure. Section V describes some of our experiments and Section VI concludes.**

I. Introduction & Data Overview

Our training dataset comprises 3D prostate magnetic resonance images from 50 subjects. Each 3D image contains between 13 and 19 slices (17 on average) and has the shape 256 by 256 pixels. We familiarize ourselves with the dataset in the *data exploration.ipynb*. Plots in the notebook reveal that there is a severe class imbalance, whereby more than 92% of the voxels in the training set correspond to background. This imbalance will have serious implications for our subsequent analysis.

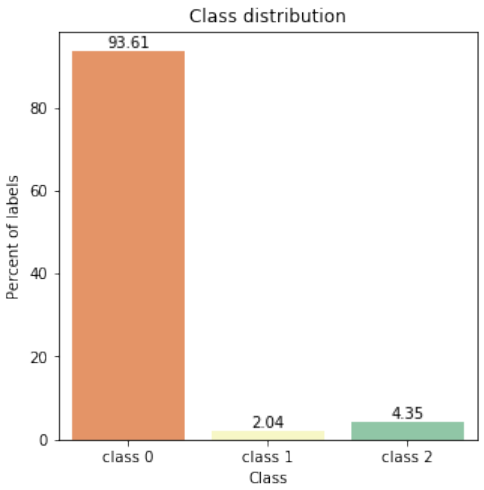

Fig. 1. Labeling pixels by class.

II. Data Preprocessing pipeline

In our review of medical image segmentation pipelines we have encountered three main approaches of feeding 3D MRI scans to the model: process individual slices, process small image patches and process 3D shapes. We have opted for the former. Furthermore to compensate for class imbalance we have experimented with extracting smaller patches that are more likely to contain the regions of interest and consequently yield more evenly distributed training set. However, this did not yield significant performance improvements.

According to task description, some of the images in the test set might be rotated by a random angle, thus our inference should be rotation invariant. To achieve this, we augment our dataset by adding random rotations of the original slices.

III. Network Architectures

As was suggested in the tutorial, we have focused our attention on the U-Net ([1]) architecture and tested its various modifications. We have also conducted a couple of experiments with other well-established models for image segmentation: VGG and FCN, however they did not yield better results than the U-Net type models.

IV. Training

UNET Image Segmentation notebook contains the pipeline we use for training our main models. The short description of the main components follows.

A. Loss functions

We have decided to mitigate the effects of severe class imbalance by selecting an appropriate loss function. First, we have used *weighted cross-entropy*, where class weights were inversely proportional to the number of occurrences in the training dataset. Additionally, we have used the

dice loss which is closely related to the IoU metrics that we want to optimize. Since this still did not yield the desired results we have modified the *dice* loss such that it ignored the information from the base class (i.e. background pixels). The final model was trained using a custom loss function that combined our version of *dice* loss with *weighted cross-entropy* with the weights 0.8 and 0.2 respectively.

B. Evaluation metrics

As was suggested during the tutorial, we compute overall precision, per-class precision and IoU. Due to considerable data imbalance and the presence of a strong background class, we put the main focus on IoU.

C. General

All models were trained with Adam optimizer, the learning rate varied accross different experiments between 0.0005 and 0.001. The final model was trained for 30 epochs.

V. Hyperparameter tuning

In *hyperparameter tuning.ipynb* we have slightly modified the previously described pipeline to facilitate our experiments. These experiments primarily focused on testing different neural network architectures and loss functions. We have considered three types of networks: classical U-Net model, VGGseg - extension of VGG net ([2]) adapted to image segmentation and FCN8 ([3]). Furthermore we have considered multiple modifications of U-Net, including but not limited to: "depth", i.e. number of contracting convolution layers, number of filters and types of up-sampling and down-sampling. We have also explored different loss functions, such as *focal loss* and tried different weights for *weighted cross-entropy* and *combined loss*. We have contemplated extending our data augmentation by adding 1) gaussian noise 2) random flips 3) zooming within the range 0.9 - 1.1. We though that 1) and 2) could help to account for the fact that images stem from two data sources and are of different quality. Our first experiments with this approach did not yield any improvement therefore we have decided not to pursue it further due to time constraints.

To save some time, we have trained the models for a small number of epochs (typically 10) and monitored the model progress using tensorboard. The results of some of our experiments are summarized in Table. Our main conclusions are:

- Selecting weights for weighted cross entropy is a delicate process, since there is a high risk that the predictions will degenerate into single class after several epochs.
- Combined loss tends to yield better results than weighted cross entropy
- The optimal number of filters in the first convolution layer for U-Net seems to be betweeen 8 and 16. The shallowest model with depth 2 performed markedly worse than other models considered
- FCN8 has too many parameters, would have needed much more epochs to produce reasonable results, by which time the checkpoints will probably occupy the entire RAM. We have tried it out as a proof of concept, however for the above mentioned reasons we did not explore it further.
- VGG net seemed to be promising, but the training progress was not very stable.
- Changing the loss function had a much stronger effect on the model performance than variations in architecture.
- The weighting of classes was crucial for the good IoU scores. However, the results were very sensitive to the exact class distribution. Validation sets have class distributions that are slightly different and suggest loss weighting that is not optimal overall. This was a main challenge for us with this type of data, since stratified validation set creation is not possible as number of classes is different per each image and full cross validation grid search is too expensive computationally.

VI. Final results / Conclusion

Our best performing model has the U-Net architecture and was trained with a customized loss that combined *weighted cross-entropy* and *dice* loss. The final architecture was found by a validation set based search over different weights balancing the two losses' types and classes. [TODO correct final resuls for rotated and non rotated images] It achieves an overall precision of 0.95, per clas precision of 0.73 and IoU of 0.57 on the test set.

Visual comparison between some of our predictions and the ground truth reveals that our model can reasonably match the shape of PZ and CG areas.

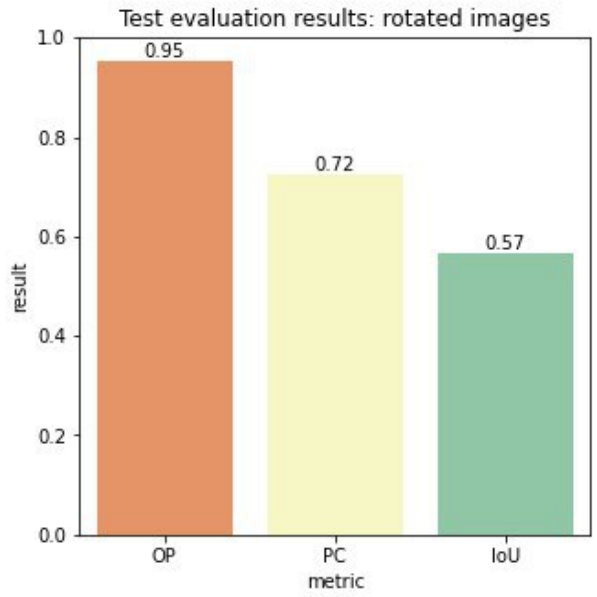

Fig. 2. Labeling pixels by class.

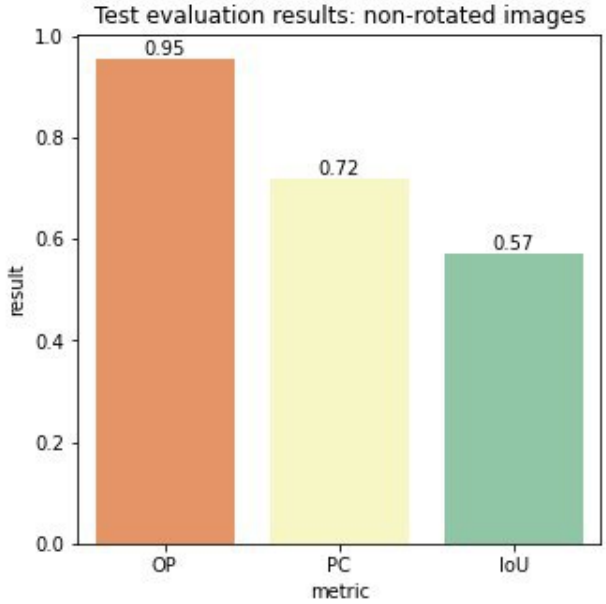

Fig. 3. Labeling pixels by class.

REFERENCES

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

[2] Simonyan, Karen, and Andrew Zisserman. 2014, "Very deep convolutional networks for large-scale image recognition."

[3] Pathak, Deepak, et al.2014, "Fully convolutional multi-class multiple instance learning."

VII. APPENDIX A

On the following page, you may find a (incomplete) list of the experiments that we ran during optimization process.

Architecture	Number of Filters	Number of Parameters	Loss Function	Loss Weights	Learning Rat Rate	Number of Epochs	IoU	Overall Precision	Per class Precision
unet depth=2	16	540'115	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	10	0.3828	0.7709	0.5523
unet depth=2	16	540'115	combined loss	$[0.2, 0.8]$	0.0005	20	0.4658	0.9366	0.5095
unet depth=2	32	540'115	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	20	0.3865	0.7525	0.5587
unet depth=2	32	540'115	combined loss	[0.2, 0.8]	0.0001	20	0.4792	0.9405	0.5681
unet depth=4	4	93'647	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	10	0.3190	0.8387	0.3295
unet depth=4	4	93'647	weighted cross entropy	$[0.3, 0.3, 0.3]$	0.0001	4	0.3071	0.9213	0.3071
unet depth=4	8	372'891	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0005	10	0.5037	0.8862	0.6081
unet depth=4	16	1'488'179	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0005	10	0.4758	0.8702	0.5704
unet depth=6	8	1'602'843	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	10	0.4849	0.8924	0.4848
unet depth=6	8	1'602'843	focal loss	NA	0.0001	5	0.3071	0.9213	0.3321
unet classic	4	89'215	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	7	0.3270	0.8439	0.3287
unet classic	8	355'835	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	10	0.4313	0.8329	0.4313
unet classic	16	1'421'299	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	10	0.5156	0.9010	0.5774
unet classic	32	5'681'123	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	29	0.3864	0.8679	0.4365
VGG	NA	11'585'411	combined loss	[0.2, 0.8]	0.0001	29	0.5321	0.9464	0.7006
VGG	NA	11'585'411	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	20	0.3921	0.8194	0.6653
FCN8	NA	134,276,585	weighted cross entropy	$[0.1, 5.0, 1.6]$	0.0001	19	0.2255	0.4813	0.4084