INTRARETINAL CYSTOID FLUID SEGMENTATION ON OCT IMAGES USING DEEP LEARNING

Gabriel Raya *Julius Mannes* *Tristan Payer* *Pedro Martinez*

Radboud University
Data Science Department,
6500 GL Nijmegen, The Netherlands

ABSTRACT

Segmentation of intraretinal cystoid fluid (IRC) is crucial for treating retinal diseases in patients. This paper presents an implementation of a convolutional neural network (CNN), with a Unet architecture, tasked to segment IRC in optical coherence tomography (OCT) images. The algorithm achieves a Dice coefficient score of 0.7497, evaluated on a dataset stemming from the ICON challenge. The results of our implementation exceed those of the our comparison baseline (Venhuizen et al., 2018). This shows the algorithm's capability of segmenting IRC and improving treatment costs and efficacy for patients suffering from retinal diseases.

***Index Terms*—** cyst fluid segmentation, deep learning, retina

1. INTRODUCTION

Optical coherence tomography (OCT) is an important non-invasive, high-resolution imaging technique capable of capturing micron-scale structure within the human retina [1]. This technique has profoundly disrupted conventional diagnostic and therapeutic strategies in clinical management and has led to paradigm shifts in the understanding of macular disease [2]. One such example is age-related macular degeneration (AMD), a retinal condition that when left untreated, constitutes the major cause of severe visual loss in older adults [3]. To determine the best treatment option, as well as efficacy, it is crucial to detect areas of fluid accumulation [4]. These fluids are classified as intraretinal fluid (IRF) and subretinal fluid (SRF) based on their location in the retina (inside or below the sensory retina, respectively). A distinction on IRF can be made between intraretinal cystoid fluid (IRC) or cysts, and diffuse non-cystic IRF. Intravitreal injection of anti-vascular endothelial growth factor (VEGF) agents is the current treatment of age-related macular degeneration. Despite encouraging results, anti-VEGF agents-based medication has a high cost and bears the risk of complications and infections that might lead to vision loss[5]. Other treatments like Treat-and-Extend (TE) apply re-treatment only in case of re-occurence of fluid accumulation [6]. Precise detection and segmentation of retinal fluids is crucial to estimate the necessity of re-treatment in regimes like TE. Computer-based algorithms have shown their potential in the automatic analysis of retinal images and, particularly, in SD-OCT volumes [7]. Convolutional neural networks (CNN) have rapidly become a methodology of choice for analyzing medical images [8]. In this work we aim to implement a CNN with the U-net architecture that is tasked with detection and segmentation of IRC[9]. Prior studies have had success applying similar network architectures to this task [6]. We will evaluate our implementation on the same dataset as Venhuizen et al. (2018) did and will take their results as a baseline.

2. METHOD

2.1. Image Dataset

The dataset is an anonymized collection of a total of 112 maculacentered OCT volumes of 112 patients from Medical University of Vienna (MUV) in Austria, and the Erasmus University Medical Center (ERASMUS). The contents of all volumes amount to 3,671 B-scans, where each volume contains from 18 to 38 B-scans. A B-scan is a 2D image and multiple B-scans make up a 3D representation (a volume). Figure 2, visualizes the structure of such a volume.[1] Half of the patients had macular edema secondary to AMD and half of them had edema secondary to RVO [6].

The data were randomly divided into two sets 80% for training, consisting of 98 SD-OCT volumes (2855 B-scans) and a validation set of 24 SD-OCT volumes (816 B-scans).

2.2. Data Inspection

All OCT volumes are stored in ITK MetaImage format containing images and corresponding annotations of IRC. The annotations show pixels that are either representing IRC or not, corresponding to two classes. Figure 1 displays a B-scan and the corresponding labeled IRC. The distribution of those two classes is heavily skewed, with 80 times more instances of pixels that do not represent fluid. This issue was

[1]Image retrieved from: https://retouch.grand-challenge.org/background/

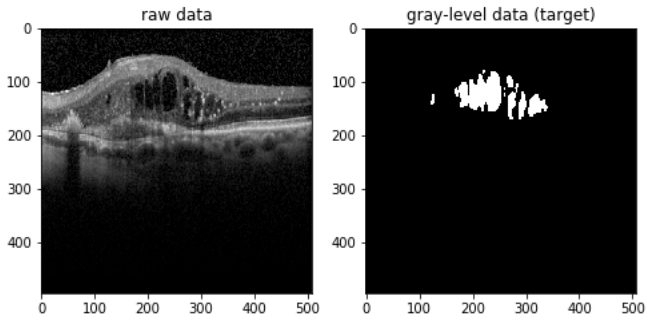


Fig. 1. Examples of B-scans from the database and the corresponding manual annotations: Input image at the left, image annotation is indicated at the right.

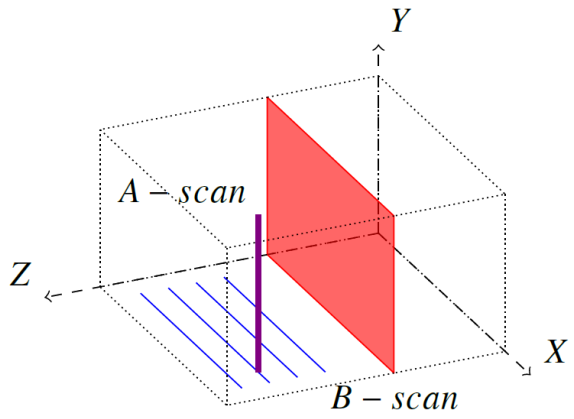


Fig. 2. OCT acquisition and the coordinate system. 1D axial scans (A-scan, purple) are combined to form a 2D cross sectional slice (B-scan, red) by scanning through the volume in a raster scan pattern (blue). Multiple B-scans are then combined to form a complete OCT volume.

addressed during training by sample weighting, which is further described in 2.5.

2.3. Preprocessing

The dimensionality of a B-scan varies depending on the volume it originates from. All scans were cropped to an image size of 448×448, so that the input to the network is of the same size. This size was chosen because it enables the depth which in the end produced best results in our experiments. We experimented with 256×256, but found that the results were worse than with image input dimensions of 448. Image pixel values were normalized to be in the range [0,1] as is common practice in image recognition tasks. The labeled annotations were one-hot encoded.

2.4. Model: U-net

A fully convolutional neural network (FCNN) was used with a similar architecture to the original U-net [9]. The main constituent of our network is a block of 2 convolutional layers of kernel size 3 with ReLU activation, followed by a batch normalization layer. The weights of the convolutional layers are initialized according to [10]. For reference purposes, we call this a UNet-block. The network consists of two paths, a contracting path and an expanding path. On the contracting path, a UNet-block is followed by a maxpooling-layer. The path ends with a single UNet-block. In total there are 6 UNet-blocks on the contracting path. On the expanding path an UNet-block is preceded by an upsampling-layer and a residual connection [11] to the corresponding level on the contraction path. In total there are 5 UNet-blocks on the expanding path. The last layer is a convolutional layer with the softmax activation function to produce the output. The initial filter size of a convolutional layer within an UNet-block is 32, subsequent levels of layers scale this value by scalars of 2, 4, 8 and 16. Such that the convolutional layers of the 6th UNet-block on the contracting path have 512 filters. The amount of UNet-blocks, as well as the amount of initial filters were subject to change in experiments 3. The described architecture yielded the best results. In total this architecture consists of 10,804,194 total parameters. Categorical Cross-Entropy was used as a loss function. Stochastic gradient descent (SGD) was applied as a optimization function with Nesterov momentum of 0.99. These settings for the optimization function were inspired by Venhuizen et al. (2018), which achieved favorable results.

2.5. Procedure

The training procedure was monitored to automatically adjust the learning rate and stop training at convergence. The learning rate was initialized at 10^{-3} and decreased whenever the validation loss was plateauing (not decreasing in the last XX epochs), by a factor of 10 up until 10^{-6}. Training was ceased once the validation loss did not improve by 10^{-3} for consecutive 15 epochs. At every epoch a mini-batch of 6, 9 or 12 (depending on experiment condition) of randomly chosen images from the training data is given as input to the network. In order to increase robustness of the model, we applied augmentation, which is described in 3.1. As mentioned in 2.2, the data set is imbalanced. In such cases the network prefers to minimize the loss on the high-occurrence class. To prevent this, sample weighing was applied. During each mini-batch creation a weight map was created, assigning each of the two classes a weight according to occurrence of that class within the sample. The following formula was used for this:

$$weight_i = \frac{1}{2}\frac{amount_i + amount_j}{amount_i}$$

where $weight_i$ is the weight given to $class_i$ and $amount_i$ is the total amount of occurrence for $class_i$ in the given sample. We experimented with three conditions:

- no weight map
- weight map calculated on labels
- weight map calculated on input image pixel values

The first condition, as the name hints at, eschews entirely of such a weight map and serves as a baseline comparison.
In the second condition, weight maps calculated on labels, the weight that was applied to class 0 (no IRC) corresponds to the formula above with $amount_0$ being the total occurrences of class 0 and $amount_1$ the total occurrences of class 1 (IRC) in the sample.
The third condition, actually originated by accident, but was kept because it produces best results. In this case $amount_0$ in the above described example corresponds to the total occurrences of the normalized pixel value 0.0 in the input image, and $amount_1$ corresponds to the total occurrences of the smallest normalized pixel value, which is just about not 0.0. In our data, this was typically a pixel value of 0.00392157. An attempt to clear up the confusion behind this, as well as the results of the experiments undergone with regards to these conditions, can be found in the Appendix.

3. EXPERIMENTS

All code was written in Python 3.6 and for the creation of our CNN the Keras library was utilized. Execution of experiments was partly conducted on Google's Colaboratory platform, which provides the user with a Tesla K80 GPU and partly on an NVIDIA DGX-1. Training time of our models ranged from 3 to 5 hours. We conduct experiments using different configurations for the model.

In total we conducted about 20 experiments. The results of experiments which we deem interesting are shown in the Results section. We re-trained our final best model using L2 loss regularization [12] with a parameter of 0.001. This improved our final model as is visible in the Results section. Parameters that were subject to change in experiments include:

- Depth of the network (amount of U-net blocks, see 2.4)
- Amount of filters (filters used in each convolutional layer)
- Batch size
- Image size (cropping size)
- Image Augmentation (amount of augmentation / type of augmentation)

3.1. Data Augmentation

Image augmentation introduces more variability into the training set and can help the algorithm to find better parameters for generalization. It has been shown that augmentation improves performance in image recognition tasks [13]. When doing image augmentation a trade-off has to be made. On the one hand the images should still resemble images from the original dataset. On the other hand the images should introduce enough variability such that they help the network not to overfit on the training set. We experimented with the amount of total augmentations as well as which augmentation settings to use. Inspiration for which augmentations to use came from Venhuizen et al. (2018). Three different augmentations were experimented with and evaluated:

- rotating the image within a range of +10 and -10.
- mirroring the image on the Y-axis
- adding multiplicative speckle noise with a magnitude of 0.3

In the start we had used a rotation range of -15 to +15 degrees, similarly as in the work of Venhuizen et al. (2018). However, we found that performance was better with the smaller range of -10 to +10. A possible reason why 15 degrees was decreasing performance, is that IRC occurs exclusively on the upper half of the annotations in our dataset. A rotation by 15 degrees might modify this distribution too strong. Mirroring the image did decrease performance for our best models, while it improved performance on weaker models. We found that speckle noise did decrease performance on all models. A comparison is visible on Table 3 in the Results section. Hence, our final best model uses only augmentation of random rotations between the range of -10 and +10. Originally we augmented images inside of our batch creator. We found that augmenting images beforehand, and manually increasing the size of our training data, was improving performance. Hence, our best model was trained on training data, which was increased by 100%. That is, instead of 2855 training images, our algorithm had access to 5710 training images. Half of which are augmentations by random rotations in the range of -10 to +10. Experiments were evaluated using the Dice coefficient metric.

4. RESULTS

We do not provide results for all experiments conducted. Because we were trying to optimize our model and not surveying different architectures, we did not perform a systematic grid-search, which makes comparison between most experiments irrelevant. The results of a selection of experiments is displayed in Table 2. We added one experiment with L2 loss regularization with a parameter of 0.001 to our best performing model. This increased the score by a bit, as is visible in

Table 2. We also compare our best model to the results obtained by Venhuizen et al. (2018) in Table 1.

Table 1. Dice score comparison with baseline.

Model	Dice Score
Current	**0.7476**
Venhuizen et al. (2018)	0.698

Table 2. Dice score comparison of different hyperparameter settings. Depth indicates the amount of UNet-blocks used, see 2.4. Optimizer indicates the optimizer used, Batchsize the size of the (mini)-batches and Blocksize the amount of layers in one UNet-block.

Depth	Batchsize	Blocksize	Dice Score
11	6	2	0.7418
11	9	2	0.7202
11	12	2	0.7476
13	6	2	0.6561
13	6	3	0.6561
Best prediction using L2 regularization			
11	12	2	**0.7497**

5. DISCUSSION

The results show that our method compare favorably with state of the art techniques, and even seemingly outperforms the baseline we compare to. A batch size of 6 has shown to produce good results, whereas a size of 9 decreased the Dice score. The best performing batch size was 12. Also, increasing depth did not improve performance, as is visible in Table 2. This holds for both depth increase through additional UNet-blocks, as additional layers within a UNet-block. Applying L2 regularization did increase the Dice score a bit, warranting further experiments with networks that are regularized. However, the experiment with L2 regularization took substantially longer, compared to without it. Increase in training time was about 100%. This made it unfeasible for us to investigate further within the scope of the current project.

6. CONCLUSIONS

In this work tackled the task of IRC segmentation using a CNN with U-net architecture. We explained the details of our design and the motivation behind the choices we made. The algorithm was evaluated in different experiment conditions and compared the results of similar work from Venhuizen et al. (2018) on the same dataset. We were able to improve on their results. This was achieved however, only due to an accidental finding on how to calculate a weight map to counter class imbalance. Without this weight map computation, our algorithm performs a little worse than the comparator. The details of that are discussed further in the Appendix. Our results reaffirm the findings of Venhuizen et al. (2018), which shows that deep-learning techniques are close to diminishing the gap between human and machine performance in IRC segmentation.

7. FUTURE WORK

An interesting line of future work could be to integrate the best network architecture with the best settings that we found in a Generative Adversarial Network (GAN). Recent work has had success applying GAN's to lung segmentation tasks [14] and brain segmentation [15]. Further, we would like to investigate our findings with regards to the weight map, which we applied. A more in-depth description of what we found and an attempt of an analysis, can be found in the Appendix. Next to these, there remain some possible experiments that are possible to improve performance on the current architecture. Unfortunately, due to the scope of this project, we were not able to test out all possible configurations in terms of number of layers, amount of filters and other parameters.

8. ACKNOWLEDGEMENT

We would like to thank our supervisor Clarisa Snchez, which provided us with helpful directions and advice. Also, we would like to thank the Tecnolgico de Monterrey, Guadalajara, Mexico, for giving us the possibility to execute some of our experiments on their NVIDIA DGX-1 GPU. We thank Google as well, for making their Colaboratory platform free to use for everyone.

9. REFERENCES

[1] Michael Pekala, Neil Joshi, David E. Freund, Neil M. Bressler, Delia Cabrera DeBuc, and Philippe M. Burlina, "Deep learning based retinal OCT segmentation," *CoRR*, vol. abs/1801.09749, 2018.

[2] Thomas Schlegl, Sebastian M. Waldstein, Hrvoje Bogunovic, Franz Endstraer, Amir Sadeghipour, Ana-Maria Philip, Dominika Podkowinski, Bianca S. Gerendas, Georg Langs, and Ursula Schmidt-Erfurth, "Fully automated detection and quantification of macular fluid in oct using deep learning," *Ophthalmology*, vol. 125, no. 4, pp. 549 – 558, 2018.

[3] Neil M. Bressler, "Age-Related Macular Degeneration Is the Leading Cause of Blindness . . .," *JAMA*, vol. 291, no. 15, pp. 1900–1901, 04 2004.

[4] Sebastian M. Waldstein, Jonathan Wright, James Warburton, Philippe Margaron, Christian Simader, and

Ursula Schmidt-Erfurth, "Predictive value of retinal morphology forvisual acuity outcomes of different ranibizumab treatment regimens for neovascular amd," *Ophthalmology*, vol. 123, no. 1, pp. 60 – 69, 2016.

[5] K. Ghasemi Falavarjani and Q. D. Nguyen, "Adverse events and complications associated with intravitreal injection of anti-vegf agents: A review of literature," *Eye*, vol. 27, no. 7, pp. 787–794, 7 2013.

[6] Freerk G. Venhuizen, Bram van Ginneken, Bart Liefers, Freekje van Asten, Vivian Schreur, Sascha Fauser, Carel Hoyng, Thomas Theelen, and Clara I. Sánchez, "Deep learning approach for the detection and quantification of intraretinal cystoid fluid in multivendor optical coherence tomography," *Biomed. Opt. Express*, vol. 9, no. 4, pp. 1545–1569, Apr 2018.

[7] Maximilian Wintergerst, Thomas Schultz, Johannes Birtel, Alexander Schuster, Norbert Pfeiffer, Steffen Schmitz-Valckenberg, Frank Holz, and Robert P. Finger, "Algorithms for the automated analysis of age-related macular degeneration biomarkers on optical coherence tomography: A systematic review," *Translational Vision Science Technology*, vol. 6, pp. 10, 07 2017.

[8] Geert J. S. Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez, "A survey on deep learning in medical image analysis," *CoRR*, vol. abs/1702.05747, 2017.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[12] Andrew Y Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 78.

[13] Jason Wang and Luis Perez, "The effectiveness of data augmentation in image classification using deep learning," *Convolutional Neural Networks Vis. Recognit*, 2017.

[14] Wei Dai, Nanqing Dong, Zeya Wang, Xiaodan Liang, Hao Zhang, and Eric P Xing, "Scan: Structure correcting adversarial network for organ segmentation in chest x-rays," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pp. 263–273. Springer, 2018.

[15] Pim Moeskops, Mitko Veta, Maxime W Lafarge, Koen AJ Eppenhof, and Josien PW Pluim, "Adversarial training and dilated convolutions for brain mri segmentation," in *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pp. 56–64. Springer, 2017.

10. APPENDIX

10.1. Sample Weight Accident

We used Keras' fit_generator() function (`https://keras.io/models/sequential/#fit_generator`) to train our model on data generated by a generator object (a batch creator). It is possible to pass a generator consisting of a third argument, next to the input and label, sample weights. Similarly to the class weights parameter of the same fit_generator() function it is a weight map, but provides weights specific to single batches. Essentially such a weight map can be described just as an array of, for example [0.3,1.7], which tells the algorithm that a loss on class 0 should be weighed only with 0.3. In fact this means that class 0 is less important than class 1 for which a loss is to be weighed 1.7 times more.

Originally our plan was to calculate these weights using the actual number of occurrences of the classes in the annotated labels. However, due to a bug that we programmed into the algorithm quite early in the development process and did not discover until late, we came across an interesting finding. Instead of calculating the weights as originally planned, we calculated the weights using the occurrence of the normalized pixel value 0.0 of the input images as well as the occurrence of the next biggest pixel value (in our data typically a value of 0.00392157). So, in fact, instead of using the occurrences for the calculation of the weights for class 0, we used the occurrences of the pixel value 0.0 of the input image. For the weights of class 1, we used the occurrences of the pixel value, which is just bigger than 0.0.

After discovering this bug and fixing it, we found that performance did actually decrease. The weight maps calculated on those input image pixel values performed much better than the weight maps calculated on the annotated labeled classes.

To compare the performance we experimented on the same model with following conditions:

- No weight map.
- Weight map calculated on annotated labels

- Weight map calculated on smallest and next to smallest pixel value.

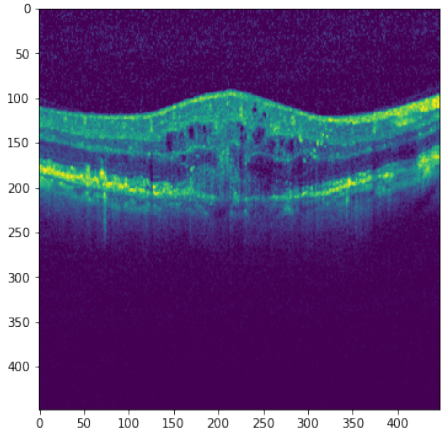
The obtained results can be inspected in Table 4 and indicate that our approach of calculating the weight map on input image pixel values, which was found by accident, is a large improvement.

Weight map	Dice score
no weight map	0.6571
label weight map	0.67798
pixel value weight map	0.7476

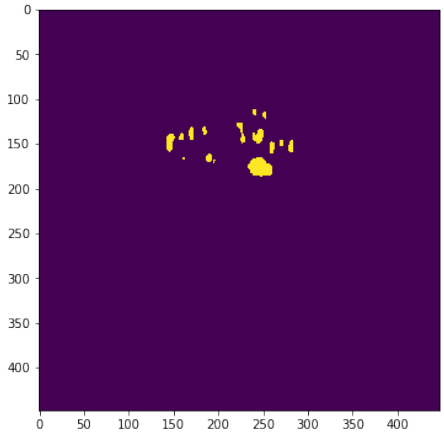
Table 3. Weight map comparison

We were curious about what these particular pixel values represent in the images. So, we investigated by setting all pixel values to 0 except the next to smallest value, which was set to 1. Figure 3 depicts the input image, the corresponding label, and a version of the input image in which only the next to smallest pixel value is set to 1 (yellow) and the rest is set to 0. This was an attempt to see if there is any structure in the distribution of that pixel value. Unfortunately, we cannot see any structure in picture (c) in Figure 3. We can only hypothesize that paying close attention to distinguish pixels in the highlighted area, benefits the segmentation performance for some reason. Investigating the reason and finding potentially even better pixel values to distinguish on when calculating sample weights, is an interesting task for future work.

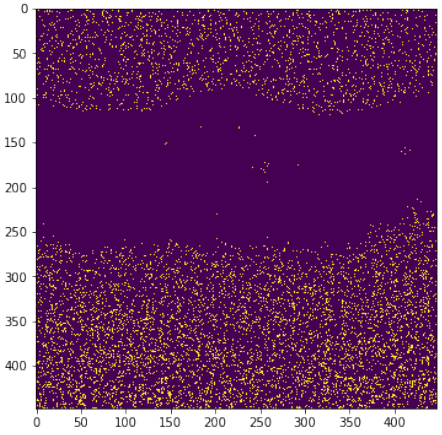
Fig. 3. Pixel Investigation.



(a) Input Image



(b) Annotation



(c) Highlighted Pixel Value in Question