Medical Imaging Segmentation

Mathieu Chevalley, Tristan Meynier, Daniel Garellick, Paul-Louis Delacour

May 1, 2020

1 U-net architecture

The architecture of our U-Net is similar as above (modulo $\pm$ 1 layers):

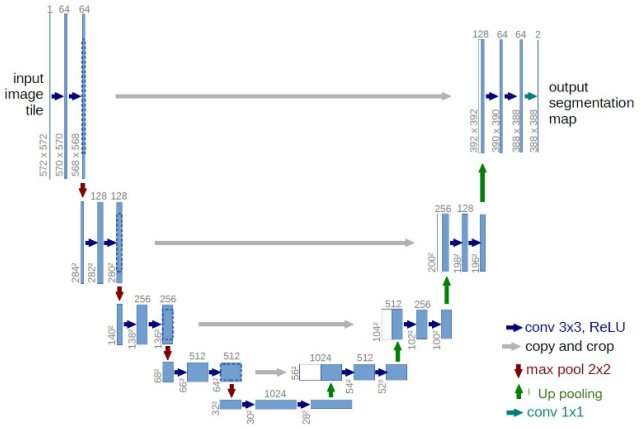

Figure 1: 4-layers U-Net as implemented (taken from project slides)

We used `LeakyReLU` instead of simple `ReLU` to prevent for the *dying* `ReLU` problem. Also we added some `Dropout` after each convolutional layer to prevent overfitting. We ran a Grid Search and performed 5-fold cross validation to select the hyperparameters and the number of layers of the network (see section 3). The activation function at the final layer is `softmax`.

2 Rotation of the training sample images

With one training sample we produced three additional rotated training samples. Each rotation angle is chosen uniformly between $-180°$ and $180°$. The addition of rotated training samples enables for data augmentation which increased the robustness of our model (see section 4). The following figure illustrates the output of a single rotation. In total we trained on the fifty original samples (unrotated) and an additional 150 rotated samples produced from the original ones.

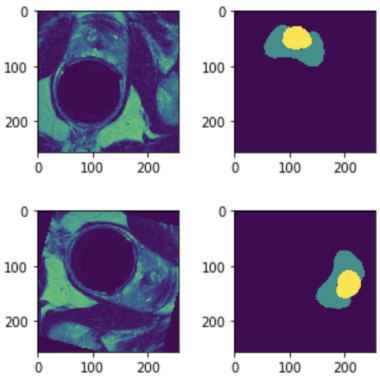

Figure 2: Visualization of a sample before (1st row) and after rotation (2nd row)

3 Grid Search

We need to find the best architecture (it terms of number of layers) and hyperparameters for the U-Net. Towards this we implemented a Grid Search. We restricted the search to 4 $\pm$ 1 layers and to the following combination of parameters:

- `start_neurons` = [32,64]
- `dropout` = [0.5, 0.35, 0.2]
- `learning_rate` = [0.0001, 0.00001]

Where `start_neurons` indicates the size of the first layer in the U-Net.

Due to the time constraint of this project it is impossible to consider all combinations of architectures and hyperparameters. However we observed (through a preliminary Grid Search) that the overall performance of the model improved with a `learning_rate` = 0.0001 and `start_neurons` = 64 independently of the number of layers. This finding enabled us to fix `learning_rate` and `start_neurons` when considering the effect of different architectures and dropout combined. This reduces the numbers of combinations to examine and so the running time.

As a side note we decided to keep as fixed the optimizer (`Adam`) and the kernel initialization (`glorot_uniform`).

We also attempted to initialize the weights using `he_uniform` however the results of the latter were significantly poorer than the former and hence to reduce the number of required fits we fixed the initialisation to `glorot_uniform` for all future grid-searches. The results of the grid-search with `he_uniform` are shown in the Appendix.

We did the grid-search with a 5-fold cross validation and kept the combination with the best mean accuracy. The results of each fold were also closely monitored to assess the variance as a significant variance across folds despite a high mean may be indicative of overfitting. This may be the case for the 5-layer architecture with 0.2 `dropout` which has a scores varying from 0.948 up to 0.971. The presented scores are the **Overall precision** (OP), **Per-class precision** (PCP) and **IoU**. The results of the Grid Search can be found at the end of the report. The model with the best mean accuracy has 4-layers and 0.35 `dropout`.

4 Final results

Following the parameter search of section 3 we consider the model with 4-layers, `dropout` = 0.35, `start_neurons` = 64 and `learning_rate` = 0.0001. Also for the training we used the original data set along with the additional rotated samples from section 2. This allows the model to be rotation invariant. It is interesting to note that these additional rotated samples also improved the predictions scores of the non-rotated samples. As a general rule adding more samples improved all the predictions (rotated and non-rotated) and the robustness of the model. This is illustrated by the results below (in bold the best scores):

Dataset	Overall pre.	Per-class pre.	IoU
Original	o = 0.9489 r = 0.9341	o = 0.5845 r = 0.4543	o = 0.5247 r = 0.4091
Original + Rotat.	**o = 0.9588** **r = 0.9533**	**o = 0.6842** **r = 0.6442**	**o = 0.6025** **r = 0.5654**

Table 1: Results for 2 training data sets: {Original only} and {Original with additional rotated samples}. The letters **o** and **r** refer to the scores for the original (non-rotated) and rotated test samples respectively.

5 Conclusion and reproducibility

We compared our implementation and findings with [Olaf] [1] and observed that we get honorable results. One could possibly improve the results even further by adding more rotated (or even shifted) training samples.
The code for U-Net (section 1) can be found in `u_nn.py`, for the rotation of the samples (section 2) in `rotation.py`, for the Grid Search (section 3) in `GridSearch.py` and for the predictions (section 4) in `predictions.py`. The output files are also available.

[1] *U-Net: Convolutional Networks for Biomedical Image Segmentation* from Olaf Ronneberger, Philipp Fischer and Thomas Brox

6 Individual contributions

Tristan M. implemented the code for the U-Net (section 1) and for the metrics, contributed to some Grid Search runs (section 2) and to a significant part of the report.
Mathieu C. rewrote the U-Net to add LeakyReLu, debugged codes, ran the model and some GridSearch, tuned hyper-parameters and offered consulting sessions to my teammates
Daniel G. Wrote the code for the K-fold cross validation and Grid Search with assistance from Paul. Performed the majority of parameter tuning in the grid search. Added the results to the report and completed other relevant parts of it.
Paul L.D. Wrote the code for the random rotations of training data. Jointly wrote the different GridSearch codes with Daniel. Contributed to some GridSearch runs. Combined the result to create the code for training and testing our models. Combined some results to create comparative tables in report.

Parameters	Result per fold	Overall precision	Per-class precision	IoU
3 layers, 0.2 dropout	Mean = 0.95040 0.96747 0.95835 0.96087 0.95183 0.91349	0.95101	0.54653	0.49366
3 layers, 0.35 dropout	Mean = 0.95580 0.96294 0.94329 0.96673 0.96606 0.94006	0.95258	0.58122	0.51472
4 layers, 0.2 dropout	Mean = 0.94704 0.94329 0.94329 0.96686 0.96830 0.91348	0.94706	0.49201	0.44335
4 layers, 0.35 dropout	**Mean = 0.96769** 0.96889 0.96967 0.96751 0.96948 0.96291	**0.96562**	**0.72078**	**0.63547**
4 layers, 0.5 dropout	Mean =0.96219 0.96415 0.96663 0.96806 0.96798 0.94414	0.9596	0.62127	0.55187
5 layers, 0.2 dropout	Mean = 0.96406 0.96664 0.97064 0.96595 0.96887 0.94821	0.96312	0.66343	0.58967
5 layers, 0.35 dropout	Mean = 0.961834 0.95218 0.96444 0.96546 0.96473 0.96236	0.959996	0.653274	0.581955

Table 2: results of the Grid Search

A Appendix: he_uniform kernel initialization

Below we present the results for `he_uniform` initialzation in a grid search with `dropout` values of 0.1, 0.2, 0.3. We originally expected this to work better with the `LeakyRelu` activation but this was not the case.

Parameters	Result per fold	Overall precision	Per-class precision	IoU
4 layers, 0.1 dropout	Mean = 0.93097 0.93019 0.92818 0.93063 0.93451 0.93136	0.93591	0.33333	0.31197
4 layers, 0.2 dropout	Mean = 0.93084 0.93019 0.92739 0.93168 0.93095 0.93401	0.93591	0.33333	0.31137
4 layers, 0.3 dropout	Mean =0.9321 0.93019 0.93117 0.93408 0.93336 0.93168	0.93591	0.33333	0.31197

Table 3: Results of the Grid Search with he_uniform weight initialization