
3D U-NET FOR PROSTATE MRI SEGMENTATION

Daniel Homola
dani.homola@gmail.com

1 Introduction

This mini project implements the fully convolutional 3D U-Net segmentation network [1] and train/evaluate it on the NCI-ISBI 2013 Challenge: Automated Segmentation of Prostate Structures. This dataset consists of 80 patients' 3D MRI scans from their prostate region. The model was implemented using core TensorFlow (i.e. without Keras) and Python. The main focus of this work was establishing preliminary benchmark performance along with developing a package that enables us to quickly iterate and try various model architectures (through hyperparameter tuning) and evaluation methods. Therefore, the project, in its current form is not intended to be a rigorous and thorough evaluation of the algorithm and it is merely a platform for future work and experimentation.

2 Data preparation

The dataset's train and leadership samples were combined to form a training set of 70 samples. The number of scans, their dimensions and various other relevant sample metrics were explored using a jupyter notebook. Several the scans and corresponding segmentation data were examined from several patients using tiled plots and animations.

Finally, the scans were preprocessed using the following steps:

- All MRI scans were rescaled to be between zero and one.
- Although the 3D U-Net is a fully convolutional network architecture (which is therefore dimension agnostic), the training scans were down-sampled to 128 x 128 size to reduce the required memory during training. The test dataset was not resized to ensure we measure test performance on the original resolution.
- To form batches of the data at training time, the number of scans (i.e. the depth of input tensors) had to be unified across patients. Therefore each patient's data was maximised to be no more than 32 scans. Then, at training time, patients with less scans are padded with zeros.
- An input size of [batch, 32, 128, 128, 1] was also desired as the network has shortcut connections between the analysis and synthesis paths, which necessitate compatible tensor dimensions at equal depths of the architecture. Maximum depth of 32 was chosen, as the network's three dimensional max-pooling and up-convolutional operations both shrink and expand the data by a factor of 2 respectively. Therefore, an input tensor dimension that is the power of two guarantees error free batches without masking.
- Unfortunately, no time was left for incorporating data augmentation into the input pipeline which surely did hurt the performance of the trained model(s).

3 Model architecture

The original implementation was followed as closely as possible, including convolutional, max-pooling and transposed convolutional layer parameters. Batch normalisation [2] and softmax with weighted cross-entropy loss were also used as in the original paper. Due to the flexible parametrisation of experiments in this project, several crucial hyperparameters of the model architecture can be easily changed by creating changing a model's `params.json` file. These parameters include:

- depth of the architecture,
- number of 3D convolutional filters to use in the first layer,
- whether to use batch normalisation or not.

Furthermore, we can set the batch size, training steps, learning rate, class weights, train and test datasets to use.

Due to time constraints only a few models were trained for this exercise (see below), and they were only trained for a short period of time (100 epochs). To see the full hyperparameter list of these referenced models, please check the `params.json` file in the project's `models`.

- `base_model`: Mimics the original paper's architecture: `depth=4`, `n_base_filters=32`, `BN=True`.
- `base_model_no_bn`: Same as above but without batch normalisation.
- `deeper_model`: `depth=6`, `n_base_filters=8`, `BN=True`.

4 Model performance

The performance of these models were assessed on the test dataset's 10 patients, using the mean Intersection of Union (IoU) across the three classes, defined as $\frac{TP}{TP+FP+FN}$, where TP: true positives, FP: false positives, FN: false negatives. Due to time constraints, proper cross-validation was not carried out and the networks were only tested on this one test set. Similarly (and unfortunately), the other interesting evaluation methods of the paper, such as artificially un-labelling some of the data and assessing model performance as a function of labelled slices, were also left out from this initial work.

Both the original test dataset (without resizing to 128x128) and the resized version were used for evaluation. Unfortunately, with model `deeper`, the original 400x400 scans of the test dataset resulted in incompatible layer sizes, that could not be concatenated, hence the missing evaluation for this model-testset pair. As could be clearly seen from Table 4, there is quite a lot of room for improvement, compared to the authors' results, however, given the low number of epochs. Nonetheless, and not surprisingly, the results demonstrate, that deeper architectures and batch normalisation get better results.

Unfortunately these three models were built and trained with an oversight in their code, that ultimately prevented me from extracting predictions that could be matched with and plotted next the corresponding ground truth segmentation. This happened as I did not save the labels at the model's prediction phase, only the predicted classes. Due to the inherent randomness of the `tf.data.Iterator`, once the prediction phase has completed, I could not match the saved predictions to their ground truth. To overcome this, a `small_model` was also trained last minute and used to generate some example prediction plots, see Figure 1.

Model	Mean IOU	
	Original	128 x 128
Base with BN	0.3384	0.4731
Base w/o BN	0.3081	0.3284
Deeper with BN	-	0.5223
Small with BN	-	0.4514

Table 1: Model performance on test dataset

5 Software engineering decision

Stanford's CS230 class

6 Future work

References

- [1] Ozgun Cicek et al. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". In: *Lecture Notes in Computer Science* (2016), pp. 424–432. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46723-8_49. URL: http://dx.doi.org/10.1007/978-3-319-46723-8_49.
- [2] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

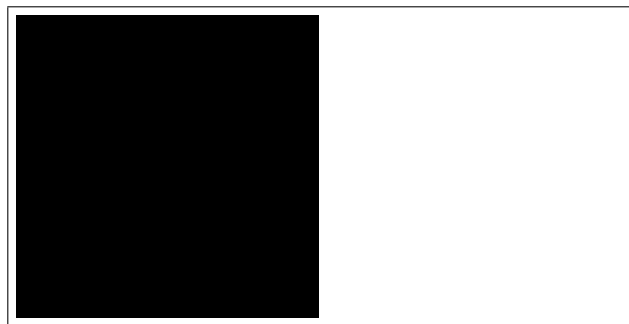


Figure 1: Sample figure caption.