Deep Stroke Lesion Detection

CS230 Final Project Report - June 10th, 2018 - Roelant Kalthof, Andrew Lewis, Brian Triana

Background and motivation

Stroke is the leading cause of neurological disability and is estimated to be the leading cause of lost healthy life-years by 2020.[1] Currently, brain images are manually reviewed to identify strokes in patients presenting with concerning symptoms. Applying deep learning for segmentation of strokes provides two potential sources of value:

- Reliable segmentation of strokes can contribute towards generation of high quality data for clinical research purposes. Accurate identification of anatomical regions affected by stroke is key for creating accurate cohorts of patients to assess outcomes and the effect of potential therapies.
- Rapid segmentation can be directly applied to the clinical setting, where rapid diagnosis and treatment is critical for improving patient outcomes. Approximately 2 million neurons are permanently lost per minute in which a stroke is untreated. In other words, time is brain.[2] Rapid and accurate segmentation of stroke through deep learning can facilitate the clinical workflow and help improve patient outcomes after stroke.

Dataset description

We use the Anatomical Tracings of Lesions After Stroke (ATLAS) dataset (release 1.1) for both our training and testing[3]. This is an open-source dataset containing 229 T1-weighted MRIs with manually segmented lesions and additional metadata. Each MRI is represented as a stack of 188 2D 232x196 black-and-white axial image “slices”. Each patient also has 188 corresponding lesion masks with 232x196 binary labels for each lesion. The ATLAS dataset includes separate output images for patients with multiple non-overlapping lesions (one image per lesion), however we made the decision to merge these output images prior to training. Figure 1 shows two examples of axial MRI images and lesion masks.

Figure 1. Two examples of axial slices and their corresponding lesion masks

Initial model

Splitting the data

We had 229 MRIs with 188 slices each, i.e. 43,281 slices in total. Our training and development dataset contained 80%/20% of the data, respectively. We tried two ways of splitting this data: by slice and by patient.

Model architecture

Our initial model, based on existing code written by David Eng, consists of a convolution encoder (ending in a 256-dimensional encoding), and a convolution decoder.[4] It ended with a sigmoid function per pixel. The model architecture is visualized in figure 2. Drop-out regularization is applied to all layers denoted by an asterisk.

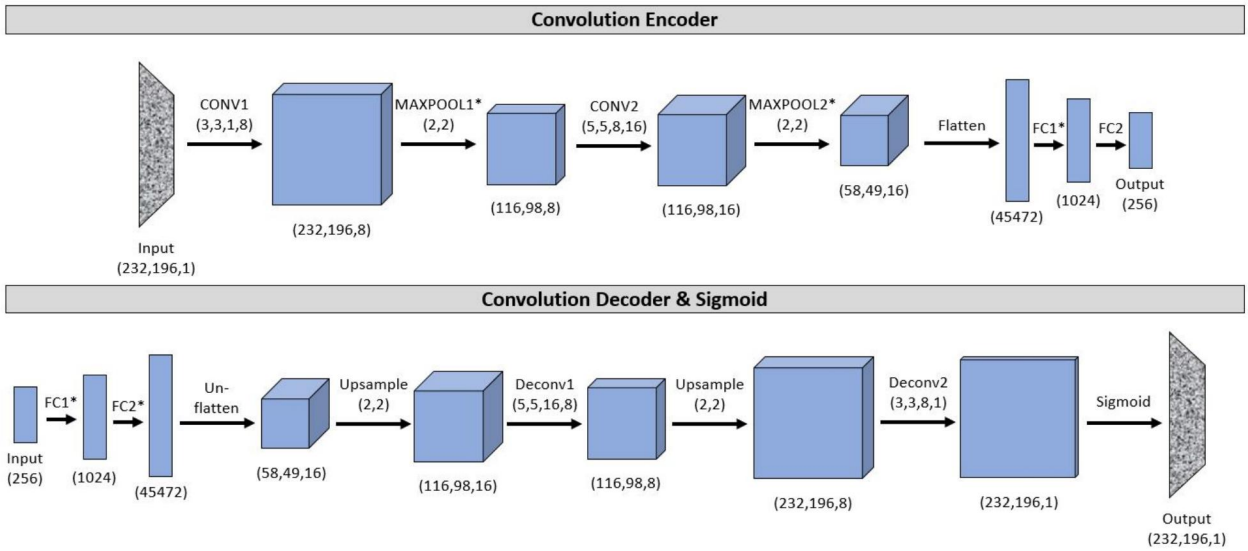

Figure 2: Diagram of the initial architecture: a convolution encoder, decoder & sigmoid

We trained the model with the following hyperparameters:

- Learning rate: 0.001
- Batch size: 100
- Drop-out rate: 15%
- Optimizer: Adam, with exponential smoothing parameter of 0.99
- Sigmoid classification threshold: 0.5
- Loss function: sigmoid cross-entropy per pixel
- Number of iterations: 10 Epochs

Training results

Figure 3 shows the training loss for both the model split by slice and the model split by patient. For the split by slice, the training loss decreases almost monotonically, which gave us confidence in the model performance. For the split by patient, the loss remained highly variable

and did not decrease steadily with increased epochs. We believe that this is a limitation of the sample size when split by patient.

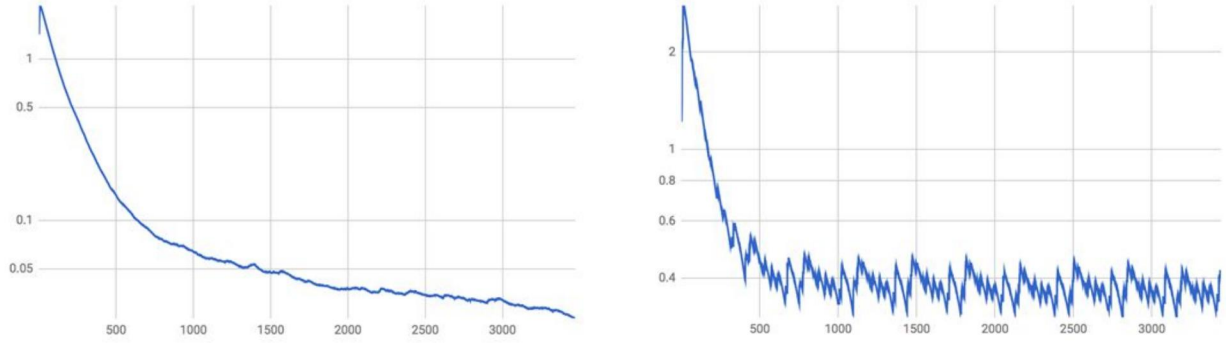

Figure 3: Training loss by iterations for split by slice (left) and split by patient (right)

Table 1 shows the DICE scores on the training and development set, for both the split by slice and by patient. The DICE scores for 'split by slice' are much higher than for 'split by patient'. This was to be expected since for 'split by slice', the model can learn from one slice of a certain patient and then apply those learnings to very similar slices of the same patient. That is an inherently easier task than predicting lesions for new patients. Unfortunately, we don't expect we can adequate results for the 'split by patient' model, since the lesions are far too varied and the sample size is too small to predict such a varied set of outputs. Because of these limitations of the dataset, we'll focus on improving the 'split by slice' performance.

On the 'split by slice' model, the variance was low (i.e. similar training and development DICE scores). Again, this was expected since this model can apply learnings from one slice in the training set to very similar slices (of the same patient) in the testing dataset.
However, there is substantial avoidable bias: the training DICE score was lower than expected human-level performance (probably closer to 1).

Table 1. Dice Scores for Training and Development Sets

	Split by slice	**Split by patient**
Training DICE score	0.115	0.009
Development DICE score	0.117	0.011

Figure 4 shows two example results for this model. In general, the model seemed to overpredict the size of the lesion.

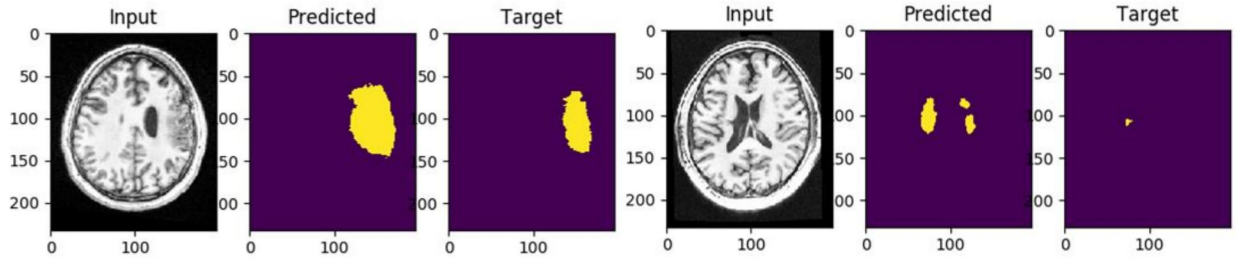

Figure 4: Example prediction results for the model split by slice

Model improvement

Based on the results of our initial model, we want to reduce the avoidable bias. Therefore, we focused on hyperparameter tuning and training a bigger model. Since we were already using an Adam optimizer, we didn't try to to use a better optimizer.

Hyperparameter tuning

We started by tuning the hyperparameters of the model described in figure 2. We focused on the most important hyperparameters that could help us reduce bias: learning rate, dropout rate and mini-batch size. We randomly picked these hyperparameter values, according to table 2.

Table 2: Random selection of hyperparameters

	Minimum	**Maximum**	**Scale**
Learning rate	0.0001	1	Logarithmic
Dropout rate	0.0	0.3	Linear
Mini-batch size	32	256	Logarithmic

In total, we tested 27 different sets of hyperparameters and ran the model for 10 epochs on each trial. Table 3 shows selected results. The second row indicates the model with the best performance, i.e. with the highest development DICE score of 0.429.

Table 3: Selected results of hyperparameter tuning

Learning rate	**Batch size**	**Dropout**	**Training DICE score**	**Development DICE score**
0.002046	39	0.282	0.034	0.040
0.000487	70	0.046	0.436	**0.429**
0.000558	54	0.015	**0.442**	0.415

0.939068	207	0.212	0.027	0.026

For our best trial in the hyperparameter tuning exercise, we ran the model for an additional 20 epochs, however DICE score performance did not materially increase. Thus, we concluded that the training iterations are not a constraining factor.

We also ran a regression analysis on the hyperparameter tuning results. The conclusion from this analysis was that a decrease in any of our hyperparameters increases performance, with learning rate and dropout being the most impactful.

Table 5. Regression Analysis of Hyperparameter Tuning

	Coefficients	*Standard Error*	*t Stat*	*P-value*
Intercept	0.202007083	0.060728879	3.326375969	0.002937471
Learning Rate	-0.104948127	0.09630688	-1.08972616	0.287119911
Batch Size	-0.00049106	0.000374273	-1.312039257	0.202451746
Dropout	-0.268828325	0.258764173	-1.038893139	0.309653392

Training a deeper model

To decrease avoidable bias, we also trained a deeper model. We added a 512-dimensional FC layer to the encoder and decoder (between the 1024- and 256-dimensional FC layers). We used the optimized hyperparameters (i.e. row 2 of table 3).

This resulted in a training DICE score of 0.351 and a development DICE score of 0.332. Although these results are worse than our optimized shallower model, we expect hyperparameter tuning on the deeper model would result in improved performance, potentially surpassing the performance of the shallower model.

Tweaking the sigmoid threshold to increase specificity

Figure 4 indicated that our model usually over-predicts lesion regions. Further analyses indeed showed this is the case: when a lesion was actually present, we overpredicted lesion size by a factor 1.94 on average. To reduce this problem, we tweaked the sigmoid threshold from the current value of 0.5 to 0.7. We expect this increases the specificity (i.e. better predicting non-lesions), and therefore the DICE score. The main reason for this is that the threshold wasn't a trainable parameter in the model: the model trained on *sigmoid* cross-entropy loss, i.e. on the sigmoid output before the threshold function that transforms it into binary output. However, changing the threshold did not have a material impact on DICE score.

Conclusions & recommendations

It is promising that with hyperparameter tuning and deeper architectures, automated stroke lesion detection was improved. Although we increased performance, our model is not yet ready

for implementation: there is still avoidable bias that would need to be reduced. Also, our model is not accurate enough for new patients (we optimized the 'split by slice' model).

To further reduce the avoidable bias of our 'split-by-slice' model, we recommend a few steps:

- Tune hyperparameters for the deeper model to further increase its performance
- Try different proven model architectures, based on a few papers that implemented identification of key structures in similar brain MRI datasets [5,6,7]
- Try the state-of-the-art Detectron algorithm, which Facebook recently open-sourced [8,9]

To increase performance on the 'split-by-patient' model, collect more data. Ideally, this data would be less varied than the current dataset: e.g. focusing on particular types of lesions and particular brain locations. Increasing the number of observations and decreasing the variability will likely improve the performance on this task. To aid this data collection, we recommend using data augmentation. One could mirror the slices laterally (since the brain is laterally symmetric) or introduce distortions. Furthermore, instead of creating detailed segmentation masks, we recommend to just create bounding boxes and utilize a n R-CNN model to create detailed segmentation masks.[10]

Sources

1. https://academic.oup.com/ageing/article/39/1/11/40973
2. http://stroke.ahajournals.org/content/37/1/263
3. http://fcon_1000.projects.nitrc.org/indi/retro/atlas.html
4. https://github.com/gnedivad/atlas/tree/master/code
5. https://www.ncbi.nlm.nih.gov/pubmed/26808333
6. https://arxiv.org/abs/1610.07442
7. http://cs231n.stanford.edu/reports/2017/pdfs/512.pdf
8. https://research.fb.com/facebook-open-sources-detectron/
9. https://github.com/facebookresearch/Detectron
10. https://arxiv.org/abs/1711.10370