CS230

Brain Tumor Segmentation on Clinical Datasets

Project Category: Healthcare

Scott Lee
wslee@stanford.edu

Nikesh Mishra
nikesh01@stanford.edu

Rodrigo Nieto
rjnieto@stanford.edu

Abstract

Brain tumor segmentation using deep learning is a helpful tool for physicians to rapidly diagnose brain tumors. In this project, an encoder-decoder architecture is utilized to perform brain tumor segmentation. In particular, we investigated several mode reduction techniques to try to limit the number of input modalities needed to achieve good performance. We utilized pre-processed datasets to train and test, and also implemented our own processing pipeline by stripping the skulls from brain images collected by Stanford Health Care (SHC).

1 Introduction

In the brain, different types of cancerous tumors can exist, and each type has a different prognosis and course of care. One type of tumor is known as a glioblastoma which is a high grade, aggressive glioma. These can grow quickly, and usually require surgery and chemotherapy; most people diagnosed with glioblasotoma live less than two years after their initial diagnosis[1]. Accurate and reproducible identification of these tumors can help doctors to rapidly detect glioblastomas, increasing treatment time and the chance of survival. The more rapid diagnosis also gives the patients and their families more time to prepare to deal with the disease. In the deep learning community, there has been a strong interest in accomplishing this task. The BraTS competition is held annually to move the field forward [2]. In the BraTS dataset, 4 imaging modalities are present: T1 (t1), T1 with contrasting agent (t1ce), T2 (t2), and Fluid Attention Inversion Recover (flair). There are three nested outputs: the necrotic and non-enhancing tumor core (NCR/NET - label 1), the peritumoral edema (ED - label 2), and the GD enhancing tumor (ET — label 4)[3]. Although the BraTS competition and state of the art brain imaging gives all four modalities, in a clinical setting less than four images are typically provided.

In this project, we develop a tumor segmentation algorithm for proprietary Stanford data that will identify whole tumor, enhancing tumor, and tumor core segments using all four modalities. We develop an approximation algorithm capable of estimating the t1 and t2 modes given the t1ce and flair modes as inputs. The segmentation algorithm should have optimal performance with all four modalities and gradually decreasing performance as two modalities are removed. A third deep learning algorithm is also implemented to strip the skulls from the images of Stanford data set. This project was done in collaboration with Professor Haruka Itakura at the Stanford School of Medicine.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related Work

In recent years, top BraTS competitors [4] have used an array of different deep learning architectures such as autoencoders [5], U-Nets [6], densely connected CNNs with label-uncertainty [7], and ensembles of various CNN architectures [8].

Many of the BraTS competitors construct ensembles of various models to boost their performance. Based on [4], the state of the art methods for single models were U-Nets and autoencoders. In this project, the team was most interested in using a variational autoencoder to regularize the learned parameters.

For the mode reduction, the main technique found in the literature was to process each channel individually at the early stages of the algorithm and recombine these using their mean and variance [9]. The team wanted to apply a similar technique to the autoencoder architecture.

3 Dataset and Features

The project utilizes multiple datasets. The first dataset is the BraTS competition data set, which consists of 285 training cases, 66 validation cases, and 191 testing cases [2,5]. The BraTS challenge data set was obtained from the University of Pennsylvania.

For the validation and testing cases, the labels are only available through the BraTS web portal, which was very slow. To iterate more quickly, the training data was randomly shuffled and 256 samples were saved for training while the remaining 29 were saved for a validation set. Each of these are encoded in a grayscale image where the shape equals (155, 240, 240) and each voxel corresponds to a volume of 1 mm^3. Before training, the data was pre-processed by downsampling the data from (4,155,240,240) to (4,80,96,64) to ease the computational burden. Next, the inputs were normalized by subtracting out the mean and scaling by the inverse of the standard deviation. Lastly, the downsized images were found to be noisy, with missing points scattered inside, so the data went through the signal processing steps of erosion followed by dilation. The labels were similarly downsampled and smoothed without the normalization. Figure 1 shows an example from the training set following pre-processing. The gap between the blue and the red regions are likely an artifact of the downsampling process.

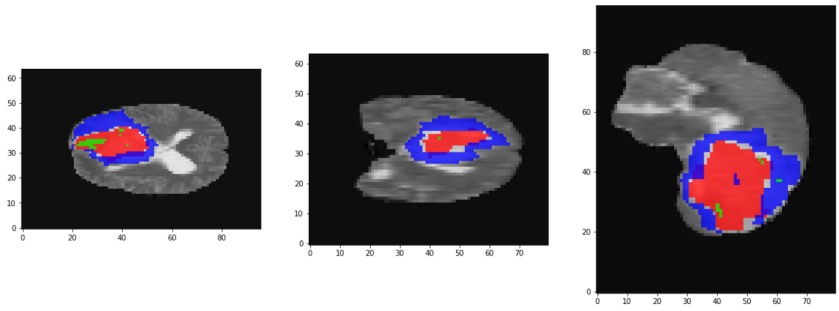


Figure 1: Example from training dataset following post processing step. Each subject had grayscale images that included four imaging modes (t1, t1c, t2, and flair). Red = NCR/NET , Green = ED, Blue = ET

The second dataset is from Stanford Hospital, and consists of about 460 patients with labeled imaging data. This set is proprietary Stanford data and can only be accessed on Stanford's Nero server.

For the skull stripping, the dataset from MICCAI MR Brain Segmentation Challenge 2018 [10] was used. The dataset consists of 30 subjects with fully annotated multi-sequence scans. For this dataset, the team requested the use and received permission from MRBrains to train the BraTS data for the skull stripping process.

4 Methods

4.1 Segmentation with autoencoder

A typical autoencoder consists of an encoder, a decoder, and a loss function. The encoder compresses the input, the decoder decompresses that output, and the loss function measures the distance between the input and output [11]. We used the autoencoder algorithm based on [5] that uses an encoder that compresses the input, a decoder that predicts the segment labels, and a variational autoencoder (VAE). The VAE takes the encoder output as an input and a random sample from a Gaussian distribution and reconstructs an image with the same input dimensions. This helps to regularize the shared encoder. Per [5], dropouts and batch normalizations were also used to further regularize the network. The architecture is shown in Figure 2 .

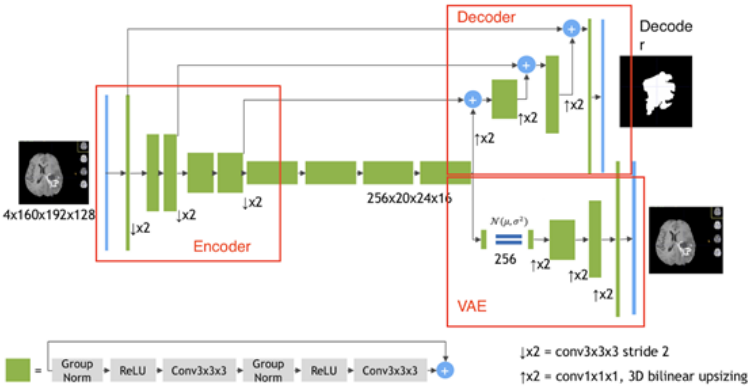


Figure 2: Visualization of the network architecture (image from [5])

Loss function is given by Equation 1.

$$\mathbf{L} = \mathbf{L_{dice}} + 0.1 \quad \mathbf{L_{L2}} + 0.1 \quad \mathbf{L_{KL}} \tag{1}$$

The first term called the dice loss $\mathbf{L_{dice}}$ represents how well the prediction matches the output and is given by Equation 2. The dice score is a standard mathematical technique that attempts to quantify the degree of similarity between an automatically segmented image and a given ground truth image by calculating the ratio of true positives to the sum of true positives, false positives, and false negatives. This term becomes increasingly negative as the fit improves; ϵ is used to prevent division by zero.

$$\mathbf{L_{dice}} = \frac{2 \quad \Sigma p_{true} \quad p_{pred}}{\Sigma p_{true}^2 + \Sigma p_{pred}^2 + \epsilon} \tag{2}$$

The $\mathbf{L_{L2}}$ and $\mathbf{L_{KL}}$ terms are loss terms related to the VAE branch and penalize mean and standard deviations that vary between the truth and prediction. These are given by Equations 3 and 4 below. For further detail, please refer to [5].

$$\mathbf{L_{L2}} = jjI_{input} \quad I_{pred}jj_2^2 \tag{3}$$

$$\mathbf{L_{KL}} = \frac{1}{N}\Sigma\mu^2 + \sigma^2 \quad \log\sigma^2 \quad 1 \tag{4}$$

The initial code was based on a Colab notebook [12]. We then wrote `train_model.py` (for code, see `/tumor-segmentation/auto_encoder/train_model.py`) as a scripted version of the notebook that we could run on AWS and Google Colab. For training, we used a batch size of one to fit within memory of the GPU used and 79 epochs.

Each epoch takes 500s on the Colab instance, and this gave a total training time of 12 hours. The training loss vs epoch is shown in Figure 3.

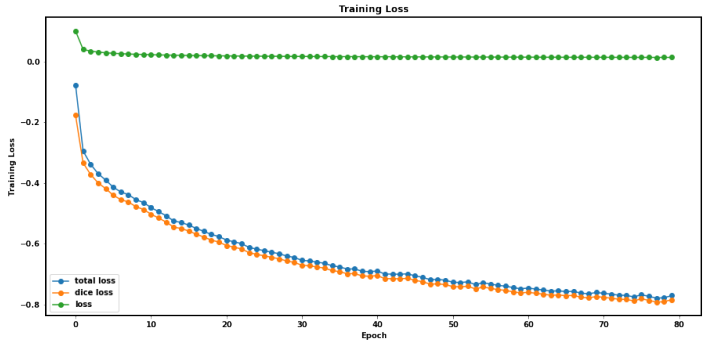


Figure 3: Model Training, Batch Size = 1, run on Google Colab

4.2 Mode Reduction

The goal of mode reduction was to eliminate the t1 and t2 input modalities with the least degradation to performance. The team observed that the VAE branch of the autoencoder model generated an output with the same dimensions as the input while reducing variance and mean. A separate VAE branch model was designed to implement this mode estimation. Since this model did not have the decoder or segmentation, the loss function did not contain the dice loss.

4.3 Skull Stripping

For the skull stripping process, we used a pre-trained extractor model that was developed using a U-net architecture[13]. The model is a CNN that was created on Tensorflow and uses verified labeled skull stripping datasets, including MRBrains, to conduct the training. The extractor also uses fewer parameters compared to similar models and uses data augmentation in the training process.

The images we skull stripped were a set of proprietary radiological brain scans from Stanford Hospital, and hosted on Stanford School of Medicine's secure cloud computing platform Nero. Our skull stripping pipeline entailed converting the provided three-dimensional image volumes from the standardized DICOM format into an alternative file type called NIfTI-1, which is a medical image storage format specific to neurological data. After processing the data, we fed our NIfTI-1 images through our skull stripping model to produce a total of 312 skull stripped images and masks.

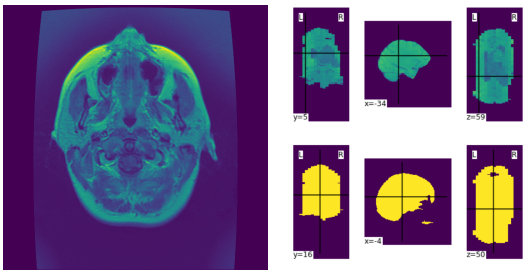


Figure 4: The left image shows a brain with it's skull; the right image shows a skull-stripped brain.

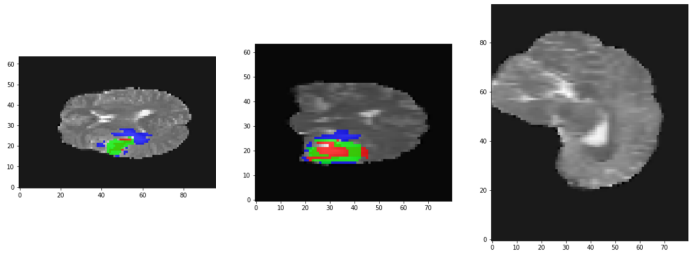
5 Experiments/Results/Discussion

The key hyperparameter for the algorithm was the learning rate used by the Adam optimizer. In [5], the learning rate was set to 10^{-4}. However, this learning rate caused the dice loss to start negative and then go to zero while the VAE cost gradually decreased to zero as well. Since the dice score

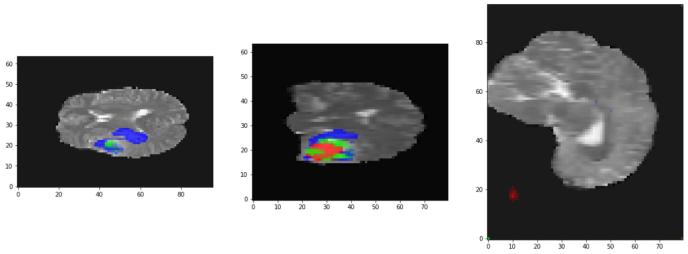
correlates with how well the model predicts labels, this led to a very broken model. By decreasing this to 10^{-5}, the dice loss started and became more negative throughout training.

Figure 5 shows a labeled sample image from the validation set as a qualitative indicator of the goodness of the segmentation. Table 1 shows that the loss went from -0.77 for the training set to -0.33 for the test set. This variance indicates that the model is overfitting to the training data.

For the mode reduction, inputting noise with same mean and standard deviation for modes t1 and t2 leads to the largest hit in performance. However, by estimating t1 and t2 using the VAE based model, much of the performance degradation can be recovered.



(a) Ground truth (validation set)



(b) Prediction (validation set)

Figure 5: Ground truth (a) and prediction (b) for a sample from the validation set.

	Total Loss	Dice Loss	VAE Loss
Training Set	-0.77	-0.78	0.013
Test Set	-0.33	-0.35	0.016
Test Set (missing modes: t1 and t2 set to noise)	-0.06	-0.18	0.12
Test Set (estimated modes: t1 and t2 estimated by VAE model)	-0.24	-0.25	0.011

Table 1: Summary of evaluated metrics for training set, test set, and test set with reduced input modes.

6 Conclusion and Future Work

Autoencoder architecture with VAE branch was implemented and learning rate was tuned to improve model training. Input reduction was investigated using the VAE branch only to estimate the missing modes. Skull stripping was accomplished using a separate deep learning model with pre-trained weights.

Future investigation may include actions to reduce the bias and the variance of the trained model. Bias could be reduced by training for more epochs ([5]] used 4 the number of epochs as our model). Variance could be reduced by data augmentation (i.e., horizontal flip of the data, cropping, adding small amount of noise). The results of further training and testing can be used to investigate ways for physicians to operate more effectively with deep learning models.

7 Contributions

Scott Lee set-up the Git repository, converted the initial model for the autoencoder from the publicly available Google Colab notebook to a usable script, trained the autoencoder model and tuned the hyperparameters (including implementing the discovery about the learning rate), and generated the VAE branch model to estimate missing inputs.

Nikesh Mishra conducted the initial debug of the autoencoder model in Google Colab, trained the autoencoder model, developed the loss functions used to evaluate the model, and wrote the code used in the skull-stripping pipeline.

Rodrigo Nieto found the model used to strip the skulls from the Stanford Hospital dataset, set-up the requisite tooling for the project on Stanford's Nero platform, and managed the team's procurement of data from external sources.

Dr. Haruka Itakura was the project mentor - she provided us with the idea for the project, helped get the team access to Nero, and answered the team's questions on tumor segmentation.

References

[1] https://theconversation.com/glioblastoma-topples-an-american-hero-but-researchers-will-continue-the-fight-102182

[2] B. H. Menze et al., "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)," in IEEE Transactions on Medical Imaging, vol. 34, no. 10, pp. 1993-2024, Oct. 2015, doi: 10.1109/TMI.2014.2377694.

[3]https://www.med.upenn.edu/sbia/brats2018/data.html

[4] Bakas, S., Reyes, M., Jakab, A., Bauer, S., Rempfler, M., & Crimi, A. (1811). Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the BRATS challenge. CoRR abs/1811.02629 (2018).

[5] Myronenko, A. (2018) 3D MRI brain tumor segmentation using autoencoder regularization. *CoRR* 1810.11654.

[6] Isensee, F., Kickingereder, P., Wick, W., Bendszus, M.,& Maier-Hein, K.H.: No new-net. In International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2018). Multimodal Brain Tumor Segmentation Chal- lenge (BraTS 2018). BrainLes 2018 workshop. LNCS, Springer (2018)

[7] McKinley, R., Meier, R., & Wiest, R. (2018, September). Ensembles of densely-connected CNNs with label-uncertainty for brain tumor segmentation. In International MICCAI Brainlesion Workshop (pp. 456-465). Springer, Cham.

[8] Zhou, C., Chen, S., Ding, C., & Tao, D. (2018, September). Learning contextual and attentive information for brain tumor segmentation. In International MICCAI Brainlesion Workshop (pp. 497-507). Springer, Cham. Chicago

[9] Havaei, M., Guizard, N., Chapados, N. & Bengio, Y. (2016). HeMIS: Hetero-Modal Image Segmentation. In Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016 (pp. 469–477). Springer International Publishing.

[10] Grand Challenge on MR Brain Segmentation, https://mrbrains18.isi.uu.nl/. Accessed 10 2020.

[11] https://blog.keras.io/building-autoencoders-in-keras.html

[12] Jadhav, Suyog, et al. "3D MRI Brain Tumor Segmentation Using Autoencoder Regularization." 3D MRI Brain Tumor Segmentation Using Autoencoder Regularization, 2020, https://github.com/IAmSuyogJadhav/3d-mri-brain-tumor-segmentation-using-autoencoder-regularization. Accessed 10 2020.

[13] Itzcovich, Ivan. "Deepbrain", 2019. https://github.com/iitzco/deepbrain. Accessed 10 2020.

Appendices

A Code Structure

Project Git Repo (Private Repo: username: cs230-grader, password: cs230_is_fun) https://github.com/stanford-cs230-2020/tumor-segmentation

Figure 6 shows the directory hierarchy of the git repo. For the main autoencoder with VAE, the model is in model.py and the training script is in train_model.py. model_reconstruct.py and train_model_reconstruct.py contains the model for the VAE branch of the code. The Colab notebooks contain "Example on BRATS2018.ipynb" which contains a Colab version of training the model. "Git Model Training.ipynb" contains a Colab version that pulls the latest train_model.py from git into Colab and trains the model. "Visualize Brain Data.ipynb" contains the code used to make the brain images in this paper.

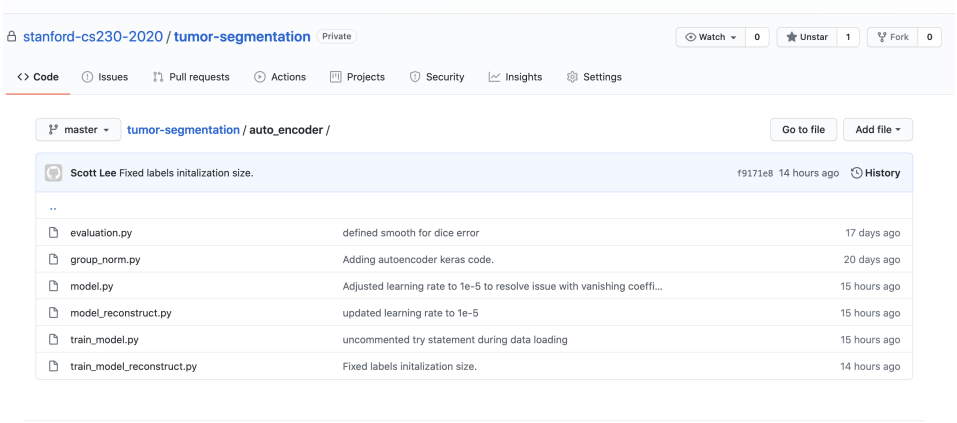


Figure 6: Directory tree for autoencoder and VAE brancc code in git.

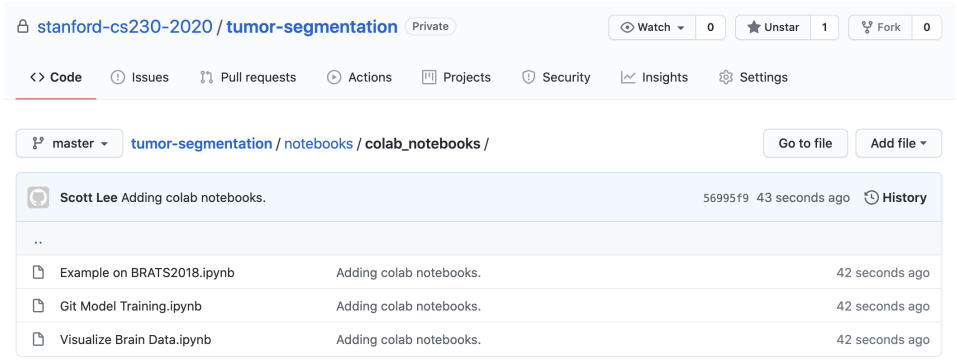


Figure 7: Directory tree for colab notebooks.