Using UNet neural networks to identify and segment brain tumors from MRI scans

Baiqing Lyu
Boston University
baiqing@bu.edu

Shengyao Luo
Boston University
jaxluo@bu.edu

Abstract

By using the a combination of brain MRI scans and manual FLAIR abnormality segmentation masks, we use the u-net architecture to construct a neural network to predict segments of tumor activity. We also assess the accuracy of this method through training it with a dynamic learning rate. We were able to achieve a dice score of 91.2, which is higher than the previously obtained dice score of 88 using the same data-set.

1 Introduction

The Cancer Imaging Archive provides data on patients from across hospitals and their medical scans. In this case, it was patient's brain MR images combined with a manually done tumor mask if there was presence detected by human doctors. All code, reports and supplementing information can be located at `https://github.com/BaiqingL/CS523-Final`

2 Background

One of the most anticipated areas of machine vision has been its advances in the medical field. As our demand for higher precision identification becomes ever growing, doctors needs to allocate their time wisely to address some of the most complex decision making processes for a patient's health. Tasks like tumor identification takes valuable time and effort away from health practitioners who could be spending it on higher priority tasks. Therefore, machine learning models have stepped in to assist doctors on saving them time by analyzing images magnitudes faster than the human eye could ever can at increasingly high accuracies.

Convolutional networks are some of the most commonly used techniques when it comes to computer vision. Since convolutional neural networks are generally understood as networks with the ability to pick out or detect patterns and make sense of them [1], it allows for a diverse amount of image classification tasks. In this case, we used a specially designed convolutional network to perform image segmentation.

3 Data processing

3.1 Data overview

The data-set contains brain MR images together with manual FLAIR abnormality segmentation masks. The images were obtained from The Cancer Imaging Archive (TCIA). They correspond to 110 patients included in The Cancer Genome Atlas (TCGA) lower-grade glioma collection with at least fluid-attenuated inversion recovery (FLAIR) sequence and genomics cluster data available. These patients comes from 5 different hospitals. Tumor genomics clusters and patient data is provided in data.csv file.

Group 1 CS 523, Boston University, (NeurIPS 2021), Boston, Massachusetts.

All images are provided in .tif format with 3 channels per image. For 101 cases, 3 sequences are available, i.e. pre-contrast, FLAIR, post-contrast (in this order of channels). For 9 cases, post-contrast sequence is missing and for 6 cases, pre-contrast sequence is missing. Missing sequences are replaced with FLAIR sequence to make all images 3-channel. Masks are binary, 1-channel images. They segment FLAIR abnormality present in the FLAIR sequence (available for all cases).

The data-set is organized into 110 folders named after case ID that contains information about source institution. Below are some example images in a patient's MRI, we display the independent MR images along with the tumor mask, and produced an overlap image of the brain with its tumor highlighted by the mask.

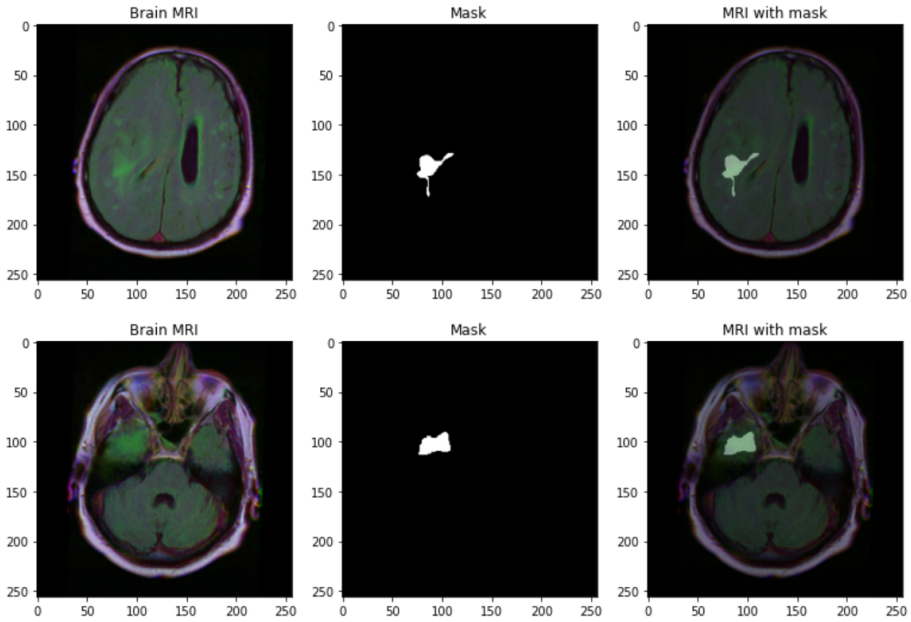

Figure 1: Brain MRI (left), Tumor mask (middle), Brain MRI with tumor highlighted (right)

3.2 Data processing

Each patient had a corresponding image folder containing all of the sliced MR images, along with a mask containing the tumor location for that given slice. To format this data, each image is loaded into a custom dataset and dataloader processed with PyTorch. A train-test split of 85% train and 15% test is established to ensure that we can produce models that do not overfit to training data.

4 Experimentation

4.1 Brief overview

For our experiment setup, we used PyTorch along with one instance of a Nvidia Tesla V100 GPU to assist in training. The total training time spanned around 2 hours and took around 200 epochs for the dynamic learning rate to terminate continued training. We decided on using an u-net structure to tackle this problem as the ability for a u-net to extract patterns and features for segmentation has been used for the previous paper that also attempted to create mask segmentation. [2]

4.2 Algorithm decision

One common problem within training a data-set is determining the learning rate parameter and how many epochs to train the model for. To resolve this issue, we first decided to use a decaying learning rate, this would allow the training process to gradually fine tune itself in an attempt to find the best fit model.

However, there is also the possibility that due to the decaying learning rate the model fails to find better solutions and progressively gets stuck at the same location. To combat that issue, we decided to apply a spiking feature to the learning rate when our model stagnates. We first take the best fitted model, and each iteration of worse accuracies gets recorded as a failure. Every time a failure to produce better accuracy occurs we also decay the learning rate, until a threshold is hit and we spike the learning rate to its original value, boosting the model and encouraging it to find a better solution.

4.3 Implementation

Here is a brief psuedo-code overview for what we did in tuning the learning rate with a worse streak tolerance of 20 epochs and learning rate spike at 10 worse epochs:

Algorithm 1 Learning rate adjustment

learning rate $\leftarrow$ *1e-3*
highest dice score $\leftarrow$ *0*
worsen streak $\leftarrow$ *0*
if *worsen streak* $<$ *30* **then**
    train model for one epoch
    dice score $\leftarrow$ *model dice score*
    if *dice score* $>$ *highest dice score* **then**
        highest dice score $\leftarrow$ *dice score*
        worsen streak $\leftarrow$ *0*
        Save model checkpoint
    else
        worsen streak+=1
        Decay learning rate by one step
        if *worsen streak* == 5 **then**
            learning rate $\leftarrow$ *1e-3*
        if *worsen streak* == 15 **then**
            learning rate $\leftarrow$ *1e-3*

5 Results

5.1 Model performance

Out of all the mask pixels that was fed into the testing data-set, the model was able to correct predict $\frac{38596185}{38666240}$ pixels, which is an accuracy of 99.82%. However, this is not a good metric to measure performance of segmentation problems like this. The reason for that is because due to the fact that most of the brain did not have a tumor present, if the model were to predict no tumors for any given brain MRI it is still possible to achieve a relatively high accuracy on the pixel by pixel basis. Therefore, we need a better metric for accuracy measurements. To address this need to measure the performance of an image segmentation performance, we use the dice coefficient. The formula for calculating this value is as such:

$$\frac{2 \times intersection}{union + intersection}$$

An intersection is the areas where the model predicted mask overlaps with the actual mask, whereas the union is the sum of the computer segmented mask, overlapping mask and the ground truth mask. We first take a look at the dice coefficient of the model after training:

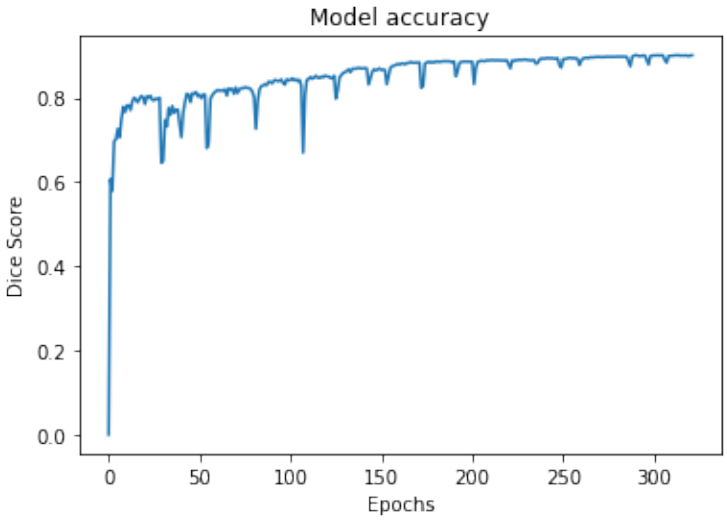

Figure 2: Dice coefficient of the model

One unique behavior that was observed with the dice coefficients is that every so often it seems like the accuracy would drop and then climb to a newer high. To further investigate this situation We also take a look at our applied dynamic learning rate adjustment to see the learning rate history throughout the course of training:

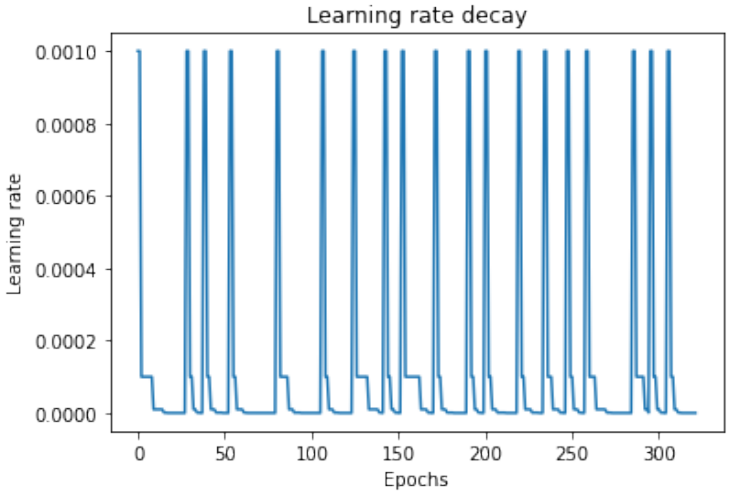

Figure 3: Learning rate of the model

We note that every time the learning rate spikes, the model seems to perform worse for a couple of epochs and then achieve a better accuracy. To confirm this suspicion, we normalized both values and combined the outcome on one figure:

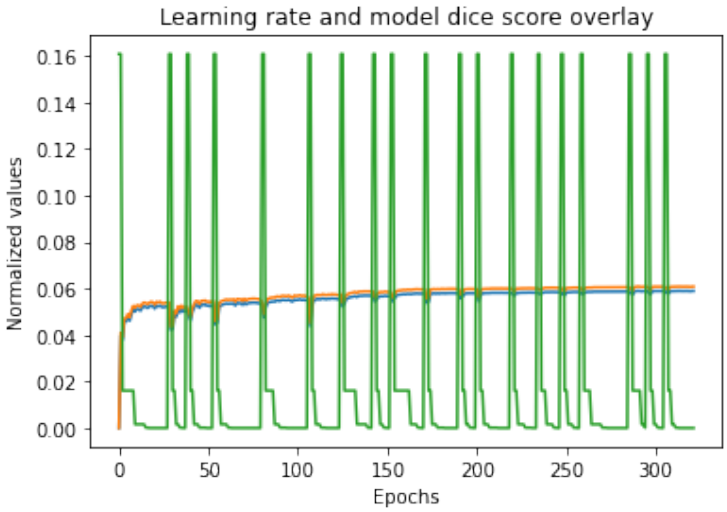

Figure 4: Dice coefficient and learning rate

5.2 Predicted tumor segments

In this section we show some examples of the trained model and its outputs on brain MRI. The inputs were only the images and the output was an array of values corresponding to the masks locating tumors. We first show an easy to identify tumor for common color analysis algorithms. Due to the large tumor size, our model was also able to pick up on the pattern and output the correct result with some minor imperfections.

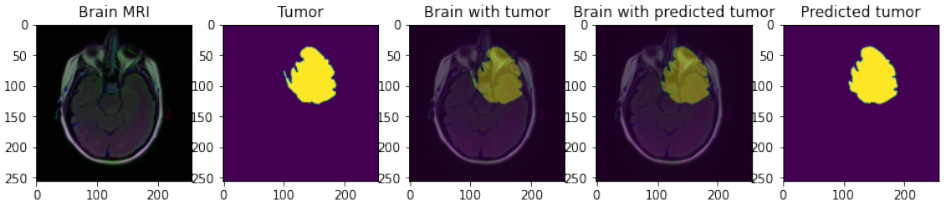

Figure 5: An easy to identify segmentation problem

Now we show a tumor segment that is harder to identify with common non machine learning algorithms. The left and right brain does not show any noticeable color differences to the human eye, which will also pose a problem for doctors and require some effort. The model is again able to pick up on this detail and output a rough estimation of the tumor location as well.

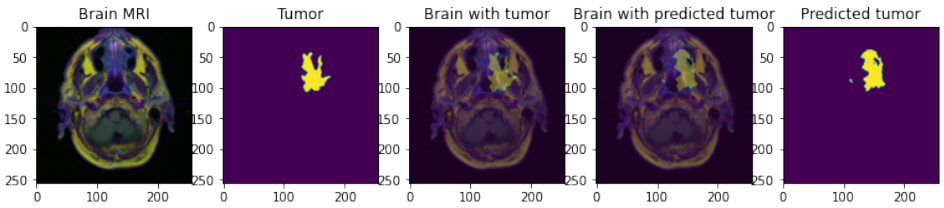

Figure 6: A difficult to identify segmentation problem

However, the model is far from perfect, here is an example of a tumor segment that was not predicted by the model. The color differences and patterns was too hidden for the model to pick up on, which posed a difficult challenge to correctly segment.

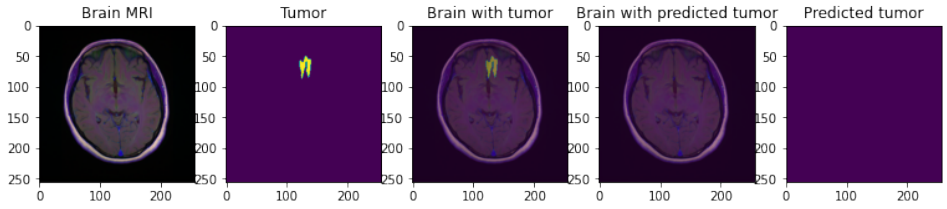

Figure 7: An incorrect identification of the segmentation problem

5.3 Comparisons to previous work

The latest paper published using this data-set has achieved a maximum reported dice coefficient of 82%. Given that we used the same ground technique to train the data-set, we were able to achieve a higher dice coefficient of 91.2%. This result could be credited to our application of a dynamic learning rate algorithm, which helped boost our results from many local maximas in attempt to find the best possible fit with our current abilities.

The trained model is uploaded as a github submodule soft linked to our main repo, it is trained on one instance of a Nvidia Tesla V-100 and total training time spanned around 2 hours for this specific run.

6 Limitations

6.1 Data size

Current data size for the brain MRI sets is still relatively small, given that there are only a handful of patients inside the entire set. If more hospitals can participate in the study it would be possible for the model to recognize more and more unique patterns of what dictates the definition of a brain tumor.

6.2 Network architecture

The U-net architecture seems to be unable to distinguish some details regarding brain tumors that have weak color differences or any noticeable patterns. This could be potentially improved with more training data, however improvements to learning some semantics of how a brain tumor should be defined and applying some sort of transfer learning can also help.

7 Additional work

7.1 Model packaging

The current model seems to already perform quite well in segmenting brain tumors, it would be beneficial to package the model into a FaaS on platforms like AWS lambda or local programs to assist in hospital use. This would require more extensive testing on new data in order to be completely sure of the effectiveness of the current model.

7.2 Closer implementation

The current U-net is closely implemented to the original paper, the main difference is that instead of expanding the image after each convolutional layer, we decided to crop it instead. Luckily since our training images did not require scaling so this feature should not impact overall performance, but should be modified to be an one to one implementation.

7.3 Learning rate tuning

There seemed to still be improvements available in tuning our learning rate adjustment parameters to allow for a better performing model. Due to the time constraint of this project, we have settled at the current best model.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. DOI:https://doi.org/10.1145/3065386

[2] Mateusz Buda, Ashirbani Saha, Maciej A. Mazurowski. 2019. Association of genomic subtypes of lower-grade gliomas with shape features automatically extracted by a deep learning algorithm. Computers in Biology and Medicine (June 2019), 218-225. DOI:https://doi.org/10.1016/j.compbiomed.2019.05.002