# Stroke Lesion Segmentation Final Report

Oscar Moses, Joey Dronkers, Jian Huang

## A. Introduction

Ischemic lesions in the brain arise as a result of stroke. As patients are admitted, doctors work against the clock to mitigate tissue damage from hypoxia. The motivation behind this experiment is to reduce the time taken to diagnose a stroke through automatic segmentation of affected areas.

As MRI scans are 3D images, this problem lends itself well to 3D convolution and segmentation. In this paper, we describe our methods and experiments for training a 3D CNN capable of classifying abnormalities in these brain scans. The goal evolved to become centered around modifications made to the model, loss function, optimization or data that yielded improvement. Ultimately, the network was successfully trained to find certain sized lesions to a relatively accurate degree, but segmentation and shape were inaccurate. This meant that the end result did not resemble anything close to a clinically reliable product. We don't doubt however, that with more data and clever engineering, it is possible to increase the accuracy further. The applications for an accurate model capable of aiding in the diagnoses of such conditions are significant. By providing further confidence to neurologists in a brief period of time, it's possible that strokes become easier and faster to treat. This applies in general to all conditions that are visibly identifiable through 2D or 3D scans. [4]

## B. Related Work

Here we summarize some of the work related to stroke segmentation, including 2D-based CNN, 3D-based CNN, and the traditional loss function used in segmentation.

Before deep learning became popular in recent years, hand-crafted feature based methods were used. Proposals for those methods were based on linear regression and relied on precise feature design. They had strong performance on small data sets, but failed to generalize on large data sets. [7]

Then, proposals were made based on 2D-based CNNs, which worked by converting the MRI data to 2D slices and applying 2D segmentation CNN for each slice. The 2D segmentation results were connected to generate 3D results. The limitation of working with 2D slices meant that the crucial 3rd dimension of context was ignored, which caused a lack of continuity in predictions. [7]

3D-based CNN proved to be much more effective. Two typical frameworks were fully convolutional networks (FCN) and U-Net. [6] Though FCN achieved end-to-end segmentation, the details of segmentation results were not ideal. U-Net, a variant of FCN, was made up of an encoder and a decoder, where the encoder summarized images into abstract feature representations and the decoder provided effective target information from the representations. U-Net was a powerful model because of its strong representational learning abilities, multi-layer abstracting and a massive number of parameters. U-net was what we used in our attempt.

Binary cross-entropy, a loss function widely used in segmentation tasks, calculates the gradient from the difference in probability distribution of each pixel in the predicted and real sample. Dice loss, also a common loss function, calculates the gradient through the dice overlap coefficient of the predicted and real sample. Different proposals tended to use different loss functions (usually a combination of CE loss and dice loss) that worked best according to their experiments. The loss function that we used was a combination of generalized dice loss and CE loss.

## C. Method

### C.1. Data Pre-processing

Data was split 80/20 into train and test segments. The model variance was largely due to the lack of large amounts of training data. Therefore, it was worthwhile training on a larger set even if the test set was significantly reduced.

Using 3D convolutional models resulted in numerous issues, one of which was time complexity. Large tensor size made this a genuine problem for training under time and hardware constraints. We solved this problem in two different methods.

The first method was to simply downsize the input data. This resulted in some success, but meant the model was training on data that was sized differently to what it would be tested with.

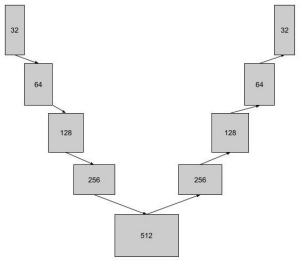
The second method was rejection sampling. This entailed creating a method for grabbing random NxN blocks within a datum. This method then tested how much of the target was contained within this chunk, and threw out blocks which contained a percentage less than a given thresholded. With the large size of the initial samples and our strict hardware limitations, we ended up using a threshold of 40 percent.

The data used was very similar in format, so we elected not to normalize.

### C.2. Model

As this is a segmentation task, one form of model was considered. This was U-Net. In each block, we considered using either 3D convolution or 2+1D convolution. Very quickly, it was found that 2+1D convolution often capped out its loss at an unsatisfactory level. As such, 3D convolution was what was used. Each block was 3D double con-

volution. A diagram of our model is presented below with numbers on each level annotating the width. Each level contains a convolution, a batch normalization, and a ReLu. All three of those processes occur twice in each level. [2]



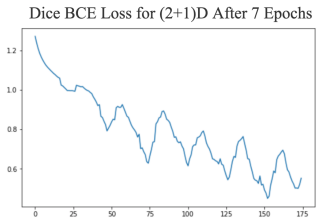

### C.3. Loss

Cross entropy is the pixel-wise classification loss and generalized dice is the multi-class extension of Dice loss.

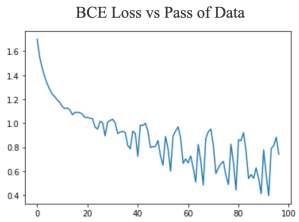
Generalized dice can balance the loss of the foreground and background according to the pixel number. However, it does nothing when the prediction does not overlap with the foreground ground truth. Thus, the cross entropy loss is introduced. The combination of generalized dice and cross entropy complement each other to produce a better result. In our code, we take the sum of two loss functions, which in combination gave us a performant loss, while in Song's paper, log was used. [3]

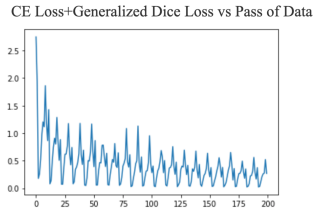
## D. Experiments

The hypothesis for this experiment was that by splitting 3D convolution into two distinct processes, memory usage and performance would be improved. This has been used in practice and research to classify video data by letting the third dimension represent temporal changes. (2+1)D convolutional blocks achieve this by using a 3 x 3 x 1 filter to convolve along the height and width axis, followed by a 1 x 1 x 3 to include depth. In the context of this assignment, the memory limitation of Google Colab meant that scans needed to be downsized to 64 x 64 x 64 before training. However, (2+1)D allowed for data to be passed through the network at twice the resolution as before, 128 x 128 x 128. Thus the hypothesis was correct, although it is unclear if this improved accuracy. More voxels means that the shape of the target is more detailed and this might have affected accuracy measurements by the Dice coefficient. Additionally, whilst higher resolution images were now viable, the batch size had to be smaller. [5]



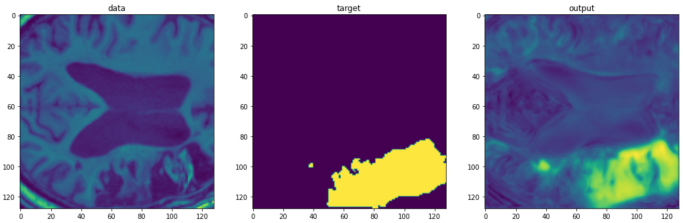

In the base code, the loss function that we used was DiceBCE, which gave an average loss of .69 on the test set. [1] To optimize the losses, we implemented a new loss function by combining CE loss with generalized dice loss, which gave an average loss of 0.0093 on the test set. The performances of both functions on the training data can be found in the graphs below. However, the predicted images of this combined loss function are less accurate than DiceBCE alone. One reason could be that our training set is too small due to a limited number of samples and the 80/20 train/test split.




## E. Conclusions

This model is effective for finding the general area of where the lesion is. However, it is extremely poor in finding exact edges and will sometimes yield false positives. Shown below is the performance of a model trained in 3D with DiceBCE loss and 64x64 sections on an image in the test set in comparison to the target area where the lesion should be. At a visual glance, this image is reasonably consistent with the rest of the images in the 3D test tensor.




Our model does find lesions to a reasonable degree. However, the flaws of not using a pretrained model become clear. The model, despite hours of training, is poor at removing noise and rendering clean edges. Overfitting was a relatively insignificant issue. While this model after 20

epochs recorded an average in-sample DiceBCE loss of .66, it recorded an out of sample loss of .69. Outliers yielded significant misclassification; the difference in loss on certain datums compared to others reached as high as .72 in the training dataset. Some form of further data processing may have benefited this model. As such, this would be a strong emphasis for future work along with using a pretrained model from other neurology work.

## F. Contribution

- Introduction - Oscar
- Related Work - Jian
- Methods - Oscar, Joey, Jian
- Experiments - Oscar and Jian
- Conclusions - Joey

https://github.com/Oscar-Mo/Stroke-Lesion-Segmentation

## References

[1] Loss function library - keras pytorch, kaggle. kaggle. 2022. 2

[2] J. Howard. Unet3d/unet3d.py at main · jphdotam/unet3d, github. 2022. 2

[3] Tao Song. Generative model-based ischemic stroke lesion segmentation. 2019. 2

[4] E. Tarakci. Example usage of bidsio for input + output for use with the isles 2022 challenge, github, 2022. 1

[5] D. Tran. A closer look at spatiotemporal convolutions for action recognition. 2017. 2

[6] Yue Zhang. Application of deep learning method on ischemic stroke lesion segmentation. 2022. 1

[7] Yongjin Zhou. D-unet: A dimension-fusion u shape network for chronic stroke lesion segmentation. 1