3D Tumor Segmentation with U-Net: Analyzing MRI Scans for Medical Insights

E6691.2024Spring.JWSS.report.jws2215
Johan Willem Schulz Sweldens jws2215
Columbia University

Abstract

This project's goal is to explore the application of semantic segmentation models in tumor detection using medical imaging, with a focus on MRI scans. Leveraging the U-Net architecture, originally designed for biomedical image segmentation, we developed a model tailored to analyze 3D MRI data. The motivation behind this research lies in the potential to enhance medical diagnostics by automating tumor detection processes, enabling faster and more accurate assessments. By training the model on a dataset comprising 3D MRI scans and corresponding segmentation masks, we aimed to provide medical professionals with a tool for rapid evaluation of patient data. Results from the project demonstrated the original goal was accomplished and the feasibility of the approach, with the developed model accurately predicting tumor pixels and generating informative visualizations. However, differences in performance metrics compared to previous studies suggest areas for further investigation and refinement. Overall, this project represents a significant step towards harnessing computer vision technology to improve tumor detection and patient care in oncology.

1. Introduction

Developing applications in medicine has long been an area of exploration within computer vision. Tasks like classification, object detection, and segmentation have all been areas of interest that researchers have sought to combine the fields in. One task that researchers have experimented with is tumor detection. Being able to correct and quickly detect tumors from medical imaging provides a variety of benefits in the medical industry.

A computer vision model that could detect cancerous tumors could provide a great deal of real world benefits to oncologists. A model could process through a larger amount of patients data in a shorter period of time than a trained oncologist, and it could simplify a lot of tasks in the field..

One way medical professionals examine tumors is with an MRI (Magnetic Resonance Imaging). MRIs use strong magnetic fields to create a 3D scan of a subject. It can generate 3D images of the body to provide insights that cannot be seen with more simplistic methods. The images that are developed do not harm the patient since it does not use radiation, and they provide quality images of body soft tissue, instead of only bones like X-Rays.

Compared to classification, or object detection, image segmentation provides the most useful tool for attempting to detect the geometry of tumors in patients. A classification model would only be able to provide a statement on whether or not there is a tumor present in an MRI. A bounding box object detection model would provide insights on where the tumor is, but it would not be able to provide fine detail about the tumor's geometry. Semantic segmentation is a great tool for outlining tumor geometry since it goes through each pixel and provides a classification on them. This type of model allows for very practical applications of neural networks in the medical field.

This project follows the implementation of U-Net, a convolutional neural network purpose built for biomedical image segmentation. [3] The original University of Freiburg paper that introduced U-Net, uses it to do semantic segmentation of tissue cells in 2D images. However, this project replicates the use of their model but with 3D MRI scans instead of 2D tissue images.

The idea behind my project is to develop a tool for medical professionals to use when examining patients. They just need to enter the patient number into the program, and then they receive a 3D model of the tumor alongside a video providing an evaluation of the MRI scan layer by layer.

2. Summary of the Original Papers

2.1 Methodology of the Original Papers

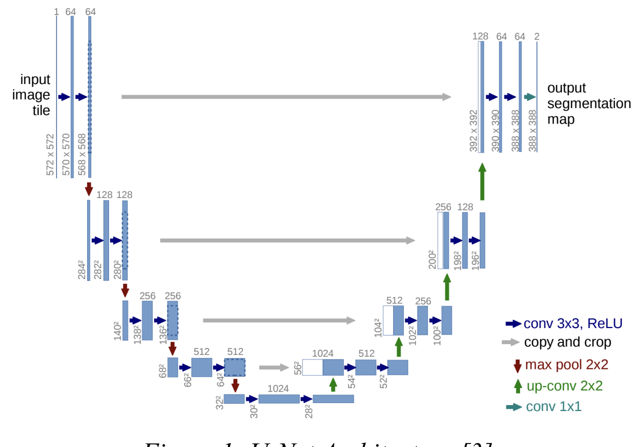

Figure 1: U-Net Architecture [3]

The main inspiration of this project originates from the original paper, "U-Net: Convolutional Networks for Biomedical Image Segmentation" from the University of

Freiburg [3]. It introduces a novel approach to biomedical image segmentation which both addresses issues with limited training data and the demand for precise localization. The main benefit of U-Net (named for the 'U' shaped architecture) and other semantic segmentation models is their ability to give each pixel a class label. The paper emphasizes the difficulty possessed by the scarcity of annotated training data in biomedical tasks, which limits the effectiveness of traditional approaches in semantic segmentation, resulting in the necessity of the U-Net. This architecture combines a contracting path of context with a symmetric expanding path for precise localization. Similar to fully convolutional networks, the novel architecture employs successive layers with upsampling operators to increase the output resolution. Notably, U-Net does not include fully connected layers and instead uses only valid convolutions. This enables a seamless segmentation of arbitrarily large images through an overlap-tile strategy.

A large struggle in the biomedical imaging industry is a lack of training data. The authors of this paper did a variety of data augmentation techniques to provide a larger dataset to work with. Additionally, the introduced a weighted loss function to address challenges in cell segmentation tasks, particularly the separation of touching objects of the same class.

As part of the literature review, I also examined another paper, "Brain Tumor Segmentation using Enhanced U-Net Model with Empirical Analysis" from a Bangladeshi University, which provided insights on how to train a U-Net model on MRI datasets [4]. There were a few notable differences in approach in this paper and the previously mentioned one with regards to the training process and the model design. The U-Net model they use is slightly simplified compared to the original paper. They also use Adam and Categorical Cross entropy with over 235 epochs for training. Their model also used Tensorflow instead of pytorch for the implementation.

Algorithm 1 Algorithm for the Evaluation

```
1: load dataset
2: Prepare data generator
3: Split data into train and test sets
4: Load model
5: for each epoch in epochNumber do
6:     for each batch in batchSize do
7:         $y_{\text{predict}} \leftarrow \text{model}(\text{trainFeatureGenerator})$
8:         $\text{loss} \leftarrow \text{categoricalCrossentropy}(y_{\text{actual}}, y_{\text{predict}})$
9:         $\text{adamOptimizer}(\text{loss}, \text{learningRate})$
10:        evaluation()
11:    end for
12: end for
13: return
```

Figure 2: Training Algorithm from [4]

While this paper did discuss that it sliced empty space from a 2D image from the training loop when they are called in the data loader, the paper fails to mention how it compensated for the fact that MRI scans are produced in 3D and that often layers have no masks at all. This paper provided valuable insights in how to expand the objectives that U-Net can achieve, however it did not provide clarity on dealing with the issues that stem from working with a small amount of data that is trained on. The paper did perform one hot encoding and minmax scaling on the MRI 2D images and the segmentation masks. This paper also did not perform any data augmentation to increase the data size, nor did either paper mention what learning rate they used or recommended.

2.2 Key Results of the Original Paper

The *Nasim et al.* paper presented results of their model in the form of the IoU and Dice Coefficient. Here is how they calculate these metrics, "True Positive (TP) in the equations denotes a tumor match between the anticipated and actual tumor. True Negative (TN) indicates a match between the expected and actual non-tumor. False Positive (FP) denotes that the predicted tumor area is not the actual tumor, and False Negative (FN) means that the expected non-tumor is the actual tumor area." [4]

$$IoU = \frac{Area of Intersection(TP)}{Area of Union(TP + FP + FN)} \tag{1}$$

$$DiceCoefficient = \frac{2Intersection(TP)}{Intersection + Union(TP + FP + FN)} \tag{2}$$

Figure 3: Evaluation Metrics' Equations [4]

The result from [4] states that they achieved an Mean IoU of 0.9130, accuracy of 0.9981, sensitivity of 0.9971, specificity of 0.9991, and a Dice Score of 0.8409. These are the metrics that my model will also be evaluated on.

3. Methodology (of the Students' Project)

3.1. Objectives and Technical Challenges

The main objective is to create a well trained U-Net semantic segmentation model that scans 2D slices of 3D MRI scans and produces quality predictions based upon the input data. Then there was the data evaluation aspect which is to create a program that would allow a medical professional to enter in a patient's number and receive the results of the model being evaluated on the patient. The software product should create a video with an overlap of the predicted mask on top of the MRI scans and also create a 3D representation of the model that can be read into any standard visualization software like for example MATLAB.

There are quite a few technical challenges in this project that were both seen previously in the papers that were referenced and new ones that were discovered. Overall handling MRI scans is complicated due to how large each patient's file is and how few patients data there is to parse through. The dataset is massive when uncompressed, well over 60GB, but there are only 369 patients. [5] Overfitting is a very present challenge. To combat that, I implemented a quick line that saves the best model at each epoch. This is selected using the lowest validation loss. As I will discuss later in more depth, the model also has problems with training on blank data too early, which requires some modifications in the training code to prevent the model from predicting empty masks each time.

3.1. Methodology Compared to References

The methodology chosen in this project is a mix between the two implementations of U-Net that were referenced in the above methodology section. In general it can be summarized as follows, I follow the implementation of the original Freiburg paper by attempting to do Binary Semantic Segmentation and using their U-Net Architecture, however the hyperparameters and training techniques were selected from [4]. I also seek to replicate their method of quantifying the performance of the model with regards to the evaluation metrics chosen.

As described earlier the U-Net architecture is replicated in Pytorch in a simpler fashion from their published reference code. [3] However, this model will not do any data augmentation, outside of ensuring that the data is in the correct shape of 240x240. There is a virtual aspect which the custom dataset handles to stack all the different MRI mode .NIfTI files into one pytorch tensor for the input data. Additionally replicating the original U-Net paper, the preprocessing combines the different tumor classes into 1 to do binary semantic segmentation. This is done to both increase the overall accuracy of the model, replicate the implementation, and also provides a simpler representation in the 3D space when the predicted model is outputted in the Medical Visualization Tool. The MRI masks have to be one-hot-encoded as well to fit into the proper representation space that the model predicts from the input data. This allows for the Binary Cross Entropy loss function to be back propagated into the model correctly.

With regards to hyperparameters and criterion, the model will be using Adam like the *Nasim et al.* paper. Using SGD with momentum proved unreliable in early testing on a smaller dataset. It often did not converge or overfit very quickly. Additionally, I opted to use their learning rate of 1e-5 to mimic their approach as much as possible. I seek to replicate their similar results in Mean IoU., accuracy, sensitivity, specificity, and Dice Score. These metrics are not affected by using a single or multiple output channels since they are in totality.

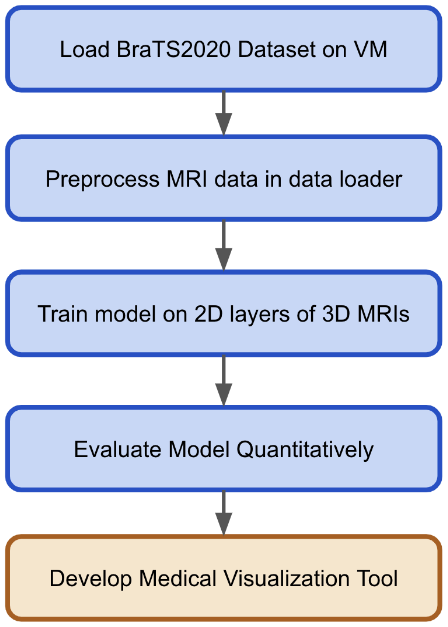
Load BraTS2020 Dataset on VM
Preprocess MRI data in data loader
Train model on 2D layers of 3D MRIs
Evaluate Model Quantitatively
Develop Medical Visualization Tool

Figure 4: Project Workflow [2]

3.2. Medical Visualization Tool

The Medical Visualization Tool is a python workbook that acts as a demonstration piece of the practical applications of this model. It utilizes a previously trained model to perform evaluation techniques on a patient's data and then visualize them in a way that is useful for a medical professional. It provides both a video demonstrating a loop of the 155 layers of an MRI scan alongside the predicted masks and a 3D data output file so that it can be visualized in any type of processing software like for example MATLAB.

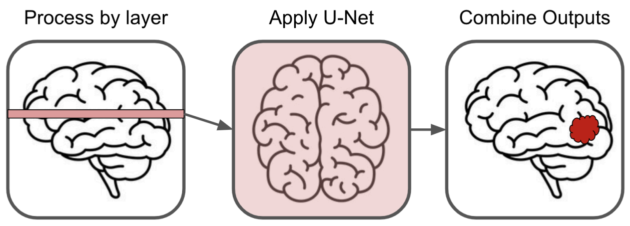

Figure 5: Medical Visualization Tool Block Diagram [2]

4. Implementation

In Section 4 Implementation, the organization provides a detailed overview of the implementation process. Subsection 4.1 covers the dataset used, including its source, contents, and splitting strategy for training, validation, and testing. It also discusses preprocessing steps applied to facilitate model training. Section 4.2

transitions to the deep learning network, focusing on the U-Net architecture, input shape, encoder-decoder structure, and output layer. Subsection 4.3.1 delves into software design, discussing dataset handling, manipulation, and custom PyTorch datasets. Subsection 4.3.2 elaborates on the training process, including challenges with 3D data and solutions devised. Lastly, Subsection 4.3.3 Evaluation Code presents evaluation metrics and tools for visualizing segmentation results in 2D and 3D formats, providing insights into model performance.

4.1 Data

The dataset that is used is the BraTS2020 which is provided by the University of Pennsylvania. [4] It is produced for the yearly Multimodal Brain Tumor Segmentation Challenge. I specifically downloaded this dataset not from the University of Pennsylvania webpage, but from the posting they made on Kaggle. The dataset is 4GB when compressed and downloaded. The dataset consists of 3D MRIs of 369 patients and includes the patients outcomes as well, however that part was not implemented in this model.

Each MRI scan of a patient is stored in a NIfTI file. The patients' scans consists of 4 image channels representing various MRI modalities: native (T1), post-contrast T1-weighted (T1Gd), T2-weighted (T2), and T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) volumes. Each of the 4 MRI modalities/channels are captured in the shape of [240, 240, 155] which represents the [height, width, depth] of the MRI scan. The height and width are self explanatory however the depth represents the vertical element of the 3D scan. Each complete 3D scan is effectively 155 equally spaced layers of 240x240 images. Obviously this represents a massive amount of data which enables great quality performance for the models. The segmentation masks for each patient are stored in a respective NIfTI file with the shape of [240,240,155].

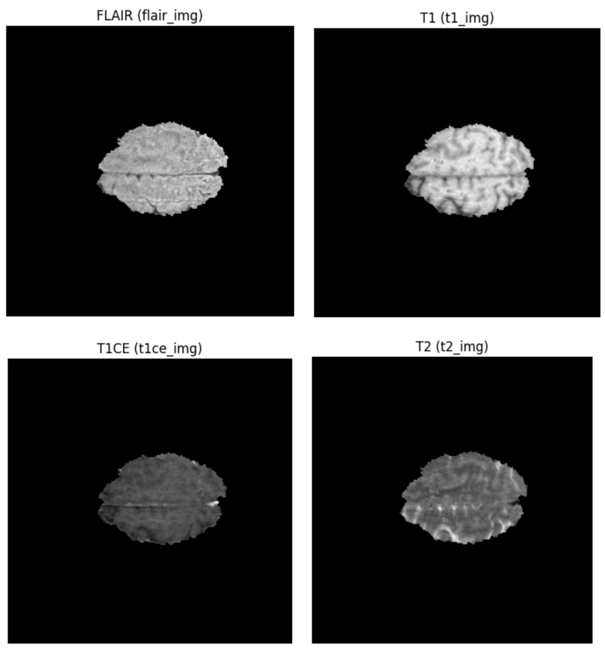

Figure 6: All 4 MRI modes of a single layer [4].

I choose to split the data by ⅔ would be training, ⅓ would be validation, and ⅓ would be testing. This is done by selecting the patients MRI scans and not by parsing through the individual layers of an MRI scan and then assigning them.

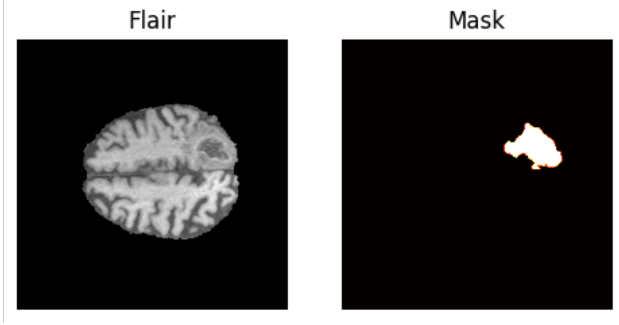

Figure 7: A T2-Flair and its respective mask [4].

4.2 Deep Learning Network

For my implementation I followed and replicated the original U-Net architecture from the Freiburg paper in Pytorch. However, there were some notable changes. I used an input shape of [16, 4, 240, 240], 16 representing the batch size, 4 the number of channels in the MRI, and 240x240 for the height and width.

Regarding the architecture of U-Net, in the encoder part, the input image, typically consisting of four channels (T1, T1ce, T2, and FLAIR MRI modalities), undergoes a series of convolutional operations to extract hierarchical features. [3] [1] The first convolutional block takes the input with 4 channels and outputs feature maps with 64 channels. Subsequent blocks increase the number of channels to 128, then 256, and finally 512, through additional convolutional layers. At each step, the spatial dimensions of the feature maps decrease due to max-pooling operations, facilitating the extraction of abstract features from the input image.

After reaching the bottleneck layer with 1024 channels, the network transitions to the decoder part, where the up-convolutional layers reverse the downsampling process of the encoder. The first up-convolutional layer doubles the spatial dimensions, transitioning from 1024 channels to 512 channels. Then, the feature maps are concatenated with the corresponding encoder feature maps and passed through convolutional blocks, reducing the number of channels to 256, 128, and finally 64. These operations progressively recover spatial details while integrating high-level context from the encoder.

The output layer consists of a 1x1 convolutional layer that reduces the number of channels to the desired number of output channels, typically corresponding to the number of classes in the segmentation task. The final output undergoes a sigmoid activation function to produce pixel-wise probability maps for each class, representing the model's segmentation predictions. Overall, the UNet architecture effectively balances feature extraction and spatial preservation, making it a robust choice for medical image segmentation. Figure 1 has a drawn out representation of what the block diagram looks like for this model from the Freiburg paper.

4.3 Software Design

4.3.1 Dataset and Dataloader

Parsing through the data and separating them into training, validation, and testing was handled by a workbook called **data_inspection.ipynb**. [1] It was a very simple program that randomly selected patients to assign to either of the groups based upon a random probability.

One of the biggest challenges when working with MRI scans is the question of how to parse through 3D images. As discussed earlier, the idea was to train the model to evaluate a single [240, 240, 4] image at a time instead of the full 3D [240, 240, 4, 155] scans at once. As such the dataset had each patient's files as one sample of ‘data’ that gets called by the function getitem.

The process begins with the creation of data dictionaries for both the training and validation datasets. These dictionaries organize the file paths and folder names of the MRI scans and their corresponding segmentation masks, allowing for easy access and manipulation. Each dataset is meticulously prepared using a custom `ImageDataset` class, which extends PyTorch's `Dataset` class. This class loads the MRI scans and segmentation masks, stacks the modalities into a single 4-channel image, and converts them into PyTorch tensors. Furthermore, the segmentation masks are converted into a binary classification problem by setting all non-zero values to 1 to match the implementation of the Freiburg paper.

4.3.2 Training the Model

The training code, provided in **training.ipynb,** for the model is very similar to normal Pytorch implementations however there are some key differences dealing with the 3D aspect of the dataset. [1] To begin with the model is first initialized by calling the implementation the UNet implementation in **seg_models.py**. The loss is defined as Binary Cross Entropy and the optimizer is Adam with 1e-5 for the learning rate. The model was trained on 20 epochs, however the code I wrote is able to load in previous training runs, and continue from where they left off.

In general the training approach is very standard. Iterate for each epoch, and within each epoch iterate through each batch from the data loader. However there must be another for loop inside the trainloader loop to parse through each of the 155 layers of a patient's scan. Each 2D layer will be fed into the model and the model will train on that layer based upon the loss function and the optimizer. [1]

During my time experimenting with this program, I noticed that there would be a repeating problem where the model would simply not predict any tumors. This was due to the fact my loop started from the bottom of a brain and iterated upwards to the top. This means the first ~20% of the 2D scans had no tumors at all, resulting in the model learning to not predict anything, before it started to reach potential layers with the mask in it. Once the loop reached the mask, the model would begin to learn, however those efforts would be undone by the time the model evaluated the final ~20% of the top layers.

As such I wrote two functions to assist with learning. The first is an if statement that checks if the mask at the layer has any tumors at all. If it does not, the function skips that level and does not try to learn or evaluate at that depth. The second piece of code creates a random order (without replacement) of the depth layers to iterate through instead of going in order vertically. These changes were very significant in enabling the model to train. The model did not work without them.

Within each layer iteration, the model first extracts the current slice of the MRI and the mask given the depth index. The segmentation mask is then one hot encoded to match the output of the U-Net. The current MRI slice is fed into the model and the predicted outputs are generated.

At the end of each layer iteration, the model performs the backpropagation. This means that the batch size is effectively multiplied by 155 (the number of layers given no skips).

This process is repeated, but without the training and backpropagation for the validation part of each epoch. The training loop also keeps track of the best weights which are qualified by the lowest validation loss.

Algorithm 2 Training Loop

```
1:  for epoch ← 1 to num_epochs do
2:      for combined_mris, seg_imgs in train_loader do
3:          for depth_idx in rand_indices do
4:              current_slice ← combined_mris[depth_idx]
5:              current_seg ← seg_imgs[depth_idx]
6:              if No tumors in mask then
7:                  skip
8:              end if
9:              outputs ← model(current_slice
10:             loss ← compute_loss(outputs, current_seg)
11:             Back propagation with optimizer
12:         end for
13:     end for
14:     for combined_mris, seg_imgs in val_loader do
15:         for depth_idx in depth_indices do
16:             current_slice ← combined_mris[depth_idx]
17:             current_seg ← seg_imgs[depth_idx]
18:             outputs ← model(current_slice)
19:             loss ← compute_loss(outputs, current_seg)
20:         end for
21:     end for
22:     if best_val_loss > val_loss then
23:         best_val_loss ← val_loss
24:         best_model_weights ← model.state_dict()
25:     end if
26: end for
```

*Figure 8: Pseudo Code of the Training and Validation loop in the workbook **training.ipynb**. Note the additional 3rd for loop to handle the 3D dimension of the data. [1]*

4.3.3 Evaluation Code

The evaluation code has three parts: **model_dice_iou.ipynb** a python workbook which provides the quantitative evaluation metrics for the model, **medical_visualization_tool.ipynb** a python workbook which provides the ability to generate quick 2D visualizations and video overlay representations, and a small MATLAB program called **visualizer3D.m** which loads the output of the medical_visualization_tool and provides the ability to scroll through a 3D visualization of the tumor. [1]

The **model_dice_iou.ipynb** is a very simple notebook. It loads in the model and the data loader for the test set in a similar manner as the other workbooks. It calculates various evaluation metrics including the Dice coefficient, Intersection over Union (IoU) score, accuracy, sensitivity (recall), and specificity. The model predictions and ground truth are compared slice by slice through each depth of the MRI scans. For each depth slice, the metrics are computed and accumulated over all slices. Finally, the average scores for the entire validation dataset are calculated and printed out, providing insights into the overall performance of the model.

The **medical_visualization_tool.ipynb** has two main functions: patient_eval() and generate_3d_visualization(). They actually function in very similar ways; the only difference is the outcome. This script is designed to evaluate the performance of a semantic segmentation model on a given patient's MRI scan. It begins by initializing the model and loading the MRI scan and segmentation mask for the specified patient. The script then proceeds to visualize the segmentation process by displaying the flair image, mask, prediction, overlay of the prediction on the flair image, and a red overlay representing areas predicted with high certainty layer by layer. Depending on the specified parameters, the script can either display these visualizations individually or create a video summarizing the segmentation results. Additionally, the script includes a function to generate a 3D visualization of the stacked masks obtained from the segmentation process. Finally, it saves the stacked masks to a MATLAB .mat file for further analysis or visualization.

The **visualizer3D.m** loads a 3D array from a MATLAB .mat file and creates a 3D isosurface plot based on the loaded data. The isosurface is generated using a desired threshold value. The plot properties are configured to set the face color to blue with partial transparency, remove the edges, and ensure equal scaling for all axes.. Finally, the plot is displayed. The program has an additional section which can produce a video of the tumor while rotating at an isometric view.

5. Results

5.1 Project Results

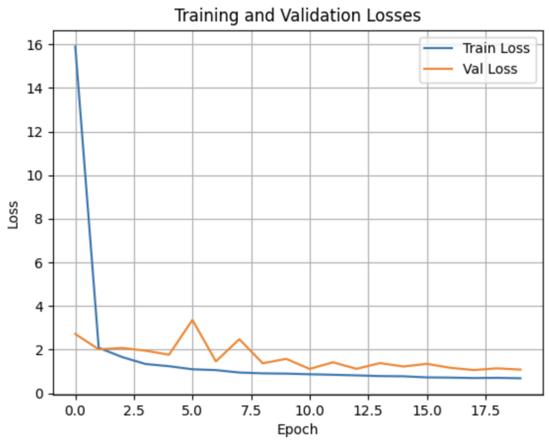

Figure 9: Training and Validation Losses. [1]

The model seemed to be training relatively well with the random weights initialization, and especially considering that the model did not use any pretrained weights. The training loss above is also not entirely one to one with the validation loss with regards to calculation metrics since the training loop skips layers that do not have any masks on them. This effectively ends up lowering the training loss. [1]

Overall, the model seems to be able to accurately predict the tumor pixels. The unique geometries were replicated. The model did well with regards to the metrics of accuracy and specificity. However, it did poorly with the Dice Score, mean IoU, and sensitivity. [4] These metrics are compared against the *Nasim et al.* paper below.

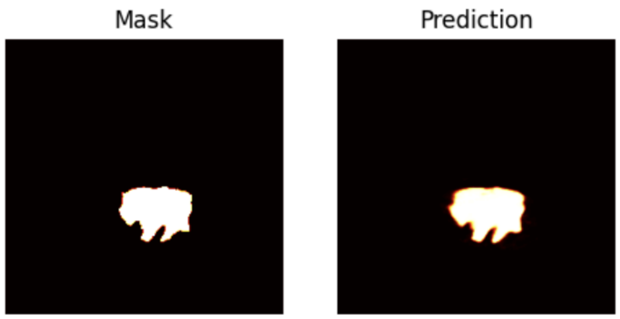

Figure 10: Ground truth tumor on left, model prediction on right for Validation Patient 10. [1]

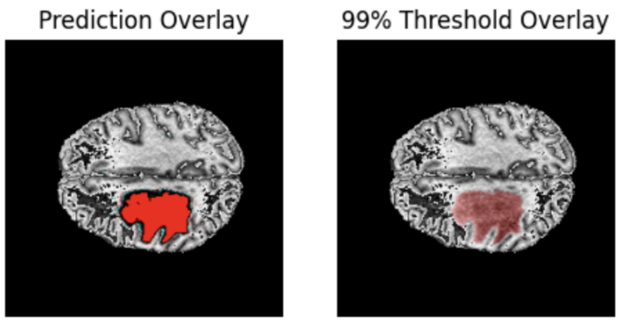

Figure 11: Mask overlay on the left scaling from blue to red with increasing confidence. The right has a 99% threshold confidence level on an opaque overlay. [1]

The Medical Visualization Tool works very well. It is able to load in a model and make quality overlay visualization videos from a given patient's data. Figure 11 above is an output from the tool. Additionally, it was able to successfully output the data to the MATLAB program, which created the 3D visualizations of the tumor. The Medical Visualization Tool demonstrated its ability to convey the complexities of the tumors in an easy to interpret 3D model and video. The result was a success.

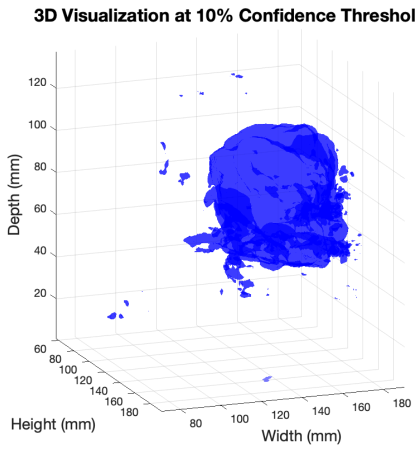

Figure 12: 3D Isosurface generated from the model prediction at 10% confidence level from Validation Patient 10. [1]

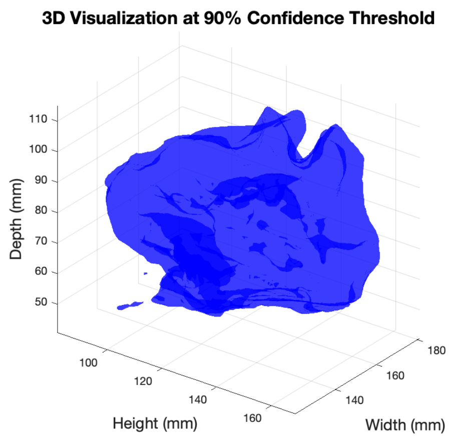

Figure 13: 3D Isosurface generated from the model prediction at 90% confidence level from Validation Patient 10. [1]

5.2 Comparison of the Results Between the Original Paper and Students' Project

Metrics	**Paper's Results**	**My Results**
Dice Score	0.8409	0.2889
Mean IoU	0.9130	0.2524
Sensitivity	0.9971	0.2935
Specificity	0.9991	0.9986
Accuracy	0.9981	0.9975

Figure 14: Evaluation Metrics Comparison between Nasim et al. and my results. Generally worse metrics were observed. [1] [4]

5.3 Discussion / Insights Gained

There are likely a few factors contributing to worse performance metrics on my model versus the *Nasim et al.* papers result. [4] The first possible point of divergence is the different model that was used. I followed the model from the original Freiburg paper which is better suited for binary semantic segmentation. [3] The choices in the learning hyperparameters and the optimizer were the same. *Nasim et al.* used a batch size of 1, while my model and the Freiburg model used a batch size of 16. However the number of epochs is very different. My model trained for only 20 epochs while theirs trained for 250. My model was already overfitting at 10 epochs, so clearly there was a significant difference in the training regiments.

The largest difference between the *Nasim et al* paper and my implementation is likely in how each model handles the 3D data. As stated earlier, my model trained by randomly selecting non empty layers within a batch. However, their paper never discussed how it dealt with the 3D data and whether it processed each layer individually or together in a group of 155 somehow with a custom loss function.

6. Future Work

The most important effort that would have to be in the future, is to experiment with a variety of different hyperparameters to try to achieve better results. This could also mean using potentially using a vision transformer for semantic segmentation and comparing that to the results from U-Net

In the future it would be interesting to explore using multiple output channels. I was unfortunately unable to explore that route due to time constraints and difficulties having the model train with the multiple output channel mode. Visualizing the different types of tumors would have been an interesting challenge in the MATLAB program.

A necessary addition to the Medical Visualization Tool before it would be considered as a product is an opaque representation of the entire MRI and not just the tumor. It would be very impactful and practical if it would be possible to see both the brain in a transparent manner and also the tumor highlighted in a different more filled in color.

7. Conclusion

This project aimed to develop a robust tool for tumor detection in medical imaging, particularly focusing on MRI scans, using deep learning techniques. Leveraging the U-Net architecture, originally proposed by the University of Freiburg for biomedical image segmentation, this project adapted the model to handle 3D MRI data for tumor detection. The objectives included creating a model capable of accurately segmenting tumors, developing a user-friendly medical visualization tool for practical application, and achieving performance metrics comparable to existing literature. [3]

The project capitalized on the U-Net architecture's ability to provide precise localization by assigning class labels to individual pixels. By focusing on semantic segmentation, the model could outline tumor geometry effectively. [5] Training the model on the BraTS2020 dataset, consisting of 3D MRI scans, presented several technical challenges, including data size, scarcity, and overfitting.

The implementation adopted a hybrid approach, combining elements from previous U-Net implementations while also integrating insights from related literature. [4] The training process involved careful handling of 3D data, with considerations for layer-wise processing and skip connections to mitigate early layer biases. The software design incorporated data preprocessing, model training, and evaluation components, culminating in a medical visualization tool for practical evaluation.

While the model demonstrated the ability to accurately predict tumor pixels and generate meaningful 3D visualizations, performance metrics such as Dice Score and Mean IoU fell short compared to benchmarks set by existing literature. [4] [1] Insightful observations highlighted the impact of hyperparameters, training regimen, and handling of 3D data on model performance. Despite limitations, the project provided valuable insights into the challenges and opportunities in medical image segmentation.

Future research directions include exploring alternative architectures such as vision transformers, experimenting with multiple output channels for tumor classification, and optimizing hyperparameters to improve model performance. Enhancements to the medical visualization tool, such as incorporating opaque representations of the entire MRI and integrating additional diagnostic features, could further enhance its practical utility in clinical settings. Overall, the project sets a solid foundation for continued exploration and innovation in the intersection of deep learning and medical imaging.

8. Acknowledgement

I am grateful to the TAs in EECS6691 and EECS6692 for their invaluable support throughout the semester. Their guidance and expertise have greatly enriched my learning experience. Thank you for your dedication and assistance.

9. References

[1] Sweldens, "e6691-2024spring-project-jwss-jws2215," GitHub, 2024. [Online]. Available: https://github.com/ecbme6040/e6691-2024spring-project-jwss-jws2215.

[2] "E6691.2024Spring.jwss.jws2215.presentationFinal," Google Slides, 2024. [Online]. Available: https://docs.google.com/presentation/d/1Nv8Sxs5Y-Zh7Rh9WM4GNe3_OuQN03GxECzAbjy_NXTs/edit#slide=id.p1.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," arXiv:1505.04597 [cs.CV], May 2015. [Online]. Available: https://arxiv.org/abs/1505.04597

[4] M. A. Al Nasim, A. Al Munem, M. Islam, M. A. H. Palash, M. M. A. Haque, and F. M. Shah, "Brain Tumor Segmentation using Enhanced U-Net Model with Empirical Analysis," arXiv:2210.13336 [eess.IV], Oct. 2022. [Online]. Available: https://arxiv.org/abs/2210.13336

[5] S. Bakas et al., "Multimodal Brain Tumor Segmentation Challenge 2020: Data," Perelman School of Medicine at the University of Pennsylvania, CBICA, 2020. [Online]. Available: https://www.med.upenn.edu/cbica/brats2020/data.html.

[6] PARISA KARIMI DARABI, "Medical Image DataSet: Brain Tumor Detection," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/parisakarimidarabi/medical-image-dataset-brain-tumor-detection. [Accessed: 2024].

10. Appendix

10.1 Individual Student Contributions in Fractions

	jws2215
Last Name	Sweldens
Fraction of (useful) total contribution	1/1
What I did	Entirety

Figure 15: Teammate contribution table

10.2 Support Material

Before using the BraTS2020 dataset, I used a more informal dataset from Kaggle during the original project proposal [6]. This dataset used three channel RGB images of MRI scans as the input with square segmentation masks. As part of my learning how to work with U-Net and other fully convolutional neural networks, I developed my training techniques on that dataset before moving onto a larger, 3D dataset.. The small Kaggle dataset also was only 2D and lacked the depth to the scans. In the Github repository, there is a folder called legacy_code with a workbook called **display_data.ipynb** which contains my first efforts towards this project [1]. Even though segmentation masks were improperly labeled as square bounding boxes around the tumors, my model did train very well after 40 epochs and showed great promise. My initial efforts in this project in that workbook for the project proposal demonstrated that I could achieve my intended final goals with regards to training a model to detect tumors and building a visualization tool.

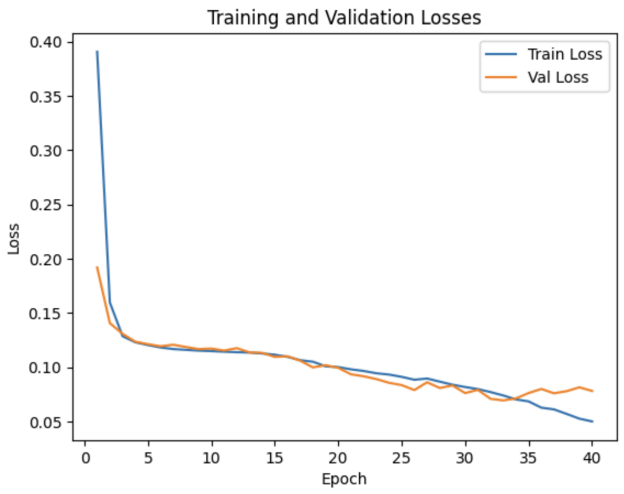

Figure 16: Training and Validation Losses from the legacy implementation of U-Net. [1]

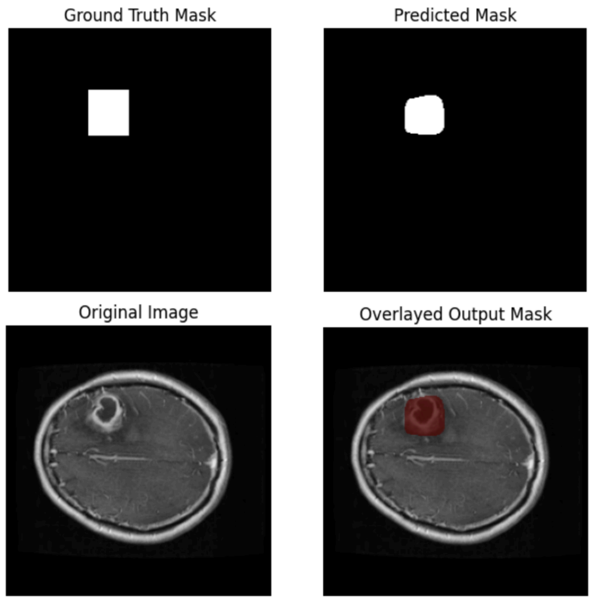

Figure 17: An example of an image with its ground truth mask, predicted mask, and overlaid output from legacy code. [1]