Final Project Report

Rishi Dadhich
Arizona State University
Team 7
rdadhich@asu.edu

Reed Axman
Arizona State University
Team 7
raxman@asu.edu

Harkirat Singh Chahal
Arizona State University
Team 7
hchahal1@asu.edu

Kaushal Rathi
Arizona State University
Team 7
krathi3@asu.edu

I. Abstract

With human doctors comes human error. Unfortunately this can have dire consequences when diagnosing patients based on medical imaging. Therefore, diagnostic accuracy may be able to be improved with the addition of machine learning algorithms. Transformers that benefit from global (long range) information modeling using self-attention mechanism have been successful in natural language processing and computer vision recently. Convolutional Neural Networks, capable of capturing local features, are unable to model explicit long-distance dependencies from global feature space. However, both local and global features are crucial for dense prediction tasks especially for 3D medical image segmentation. TransBTS provides a potential solution on extracting the local and global features. In progress check-1 we had presented 3 goals to achieve. They were to train the model, test the model, understand the inner detailed functioning of the model. Currently we have completed training, tested the models and have broken down the aspects of the model for easy understanding. Unfortunately, our BraTS2020 submission returned with an error and we were unable to obtain new results.

II. Introduction

As one of the leading forms of death, cancer causes a significant number of deaths every year. Given the type of cancer death can usually be avoided with early detection and treatment. Unfortunately, it can be difficult to read the size of cancerous lesions from medical imagine methods causing doctors to mis/under diagnose the spread of cancer in a given area. The manual process of lesion detection is also time consuming and heavily relies on learned experience which can exacerbate misdiagnosis further. Therefore, it stands that this process may benefit from a generalizable automated lesion detection process to supplement the physicians diagnostic process. Since lesion formation, size, and shape varies greatly it is difficult to find commonalities between subjects. Also, as not all lesions have distinct boundaries and have different structures it becomes very difficult to generate segmentation algorithms based on prior knowledge.

Currently Convolutional Neural Networks (CNNs) dominate visual modelling since they are quick to segment medical images. Unfortunately, CNN based methods have difficulty in preserving the explicit long-distance dependencies of small kernel size which limit the receptive fields of convolution. At the same time transformers can learn the long term dependencies easily learn long-distance dependencies but struggle to learn short-term dependencies. Li et. al. deploy a Convoutional Neural Network with a transformer in order to overcome problem associated with traditional segmentation techniques as aforementioned. This was done in hopes to create a generalizable cancer lesion detection algorithm to assist in diagnosis.

III. Methods

Training: The training method was supervised learning. We had to replicate a Github repository and to check if we got the output. The paper states that the training uses a K fold technique since there is no testing dataset for the BraTS 2019, BraTS 2020, LiTS 2017, and KiTS 2019 datasets. This is because these datasets are part of a world challenge hence the testing dataset is not released. Only the training and validation are released while the testing dataset is used as rubric to check the participants score. The training is done only a training dataset using the k fold cross validation technique (k=5) and the validation data itself is used for testing purposes to get an idea of the generalization of the training itself.

Testing: The best trained model is used for testing the data for all the datasets. Due to non-availability of the test data, we used validation dataset instead, for testing. 125 neuroimaging files after data augmentation and concatenation are fed into the trained model to give the segmented result.

The output can be visualized on the basis of multiple classes of segmented tumors namely WT (whole tumor), TC (tumor core) & ET (enhancing tumor), which can clearly be identified from the output image. Validation parameters like dice scores and Harsdorf distances need ground truth to be evaluated which in turn is computed by submitting our output *.nii* files to CBICA (University of Pennsylvania) dept (only applicable to BraTS2020 dataset).

Intermediate Layer Output Visualization; Extended Work: After training, we thought it would be valuable to visualize the data generated in the encoding and and decoding process. This is done in the decoding step where the encoding data from the skip connection is available alongside the decoded output at each upsampling step. To plot the results, a second tensor is created containing the same information of the mask or output. This tensor is then moved to the cpu before detaching the gradient plots to preserve the generated information. This tensor is then converted to a numpy array so the pixel data of each slice can be visualized with matplotlib.

Since this is before any color processing is done, a intensity color map was chosen to show pixel intensity denoting the change in the image before and after being passed through the learning algorithm. The data used in this was the model trained after 25 epochs. Since we desired to visualize what may be occurring in the model, a non fully trained model was deemed appropriate to use.

IV. Implementation

Model : The TransBTS model (fig.1) relies on four python files which describe positional encoding, the transformer model, a U-Net CNN, and the main TransBTS file. It is helpful to think of this model as a multi-layered structure where each function initiates another layer. The first layer is the TransBTS function which sets the values of hyperparameters and returns the complete model to be passed to the GPU. The first step of this model is to encode the input image before being passed through the transformer. This is handled when the parent class TransformerBTS is initialized during the initialization of the child class BTS. As pytorch will automatically run each forward function within a class when initialized, the forward function in this class will provide the encoder filters and output to pass through the to the position encoder and transformer. The general flow of encoding is the input is passed through the U-Net CNN to encode and downsample until the tensor goes from a size (1, 32, 128, 128, 128) to (1, 128, 16, 16, 16). Downsampling is achieved by pairing encoding blocks with downsampling blocks per encoding step until the desired output size is reached. After an initial pass through a convolution and subsequent dropout layer, data is passed to encoding blocks which follow a cycle that uses batch normalization on the input, passes it through a rectified linear unit (relu) activation function, and uses this output as the input for a second convolution cycle. Once this is completed, the output is returned as a downsampled tensor as the input channels of the convolution layers increase over each downsample cycle. After 3 downsample cycles with the last tensor being passed through four more encoding blocks which do not change the tensor size, the encoding masks and output tensor are returned. The latter output tensor then undergoes batch normalization before being passed through a relu activation function. The output tensors sliding local blocks are then extracted to be passed through a linear encoder before the pixel positions are encoded.

From the positional encoding file it is possible to see that there are two types of encoding: fixed positional and learned positional. The downsampled data is then passed through a dropout regularization method before being used as the input for the transformer. This transformer weighs the significance of each part of the input data through a self-attention mechanism. The self attention mechanism normalizes the positional data, passes the normalized data through a dropout function, and then normalizes the data again. These layers are then sequentialized, returning the output layer and intermediate output layers. Once these layers are created, they are passed through

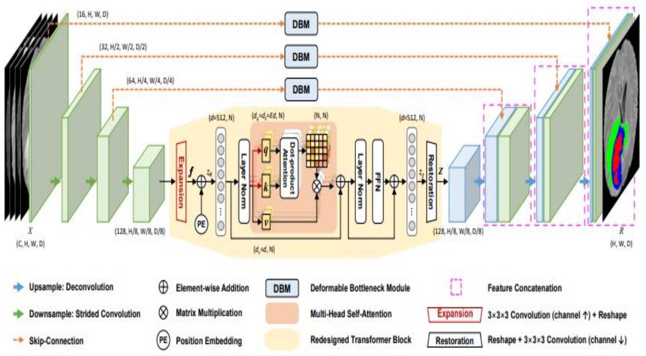


Fig. 1. TransBTSv2 Model [1]

a feed forward network for feature mapping before decoding and upsampling.

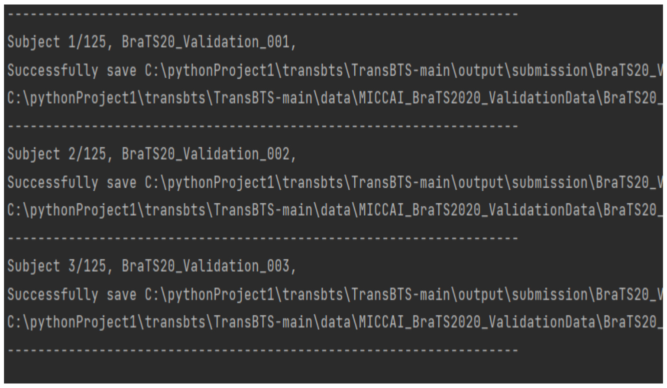
The encoder intermediate output layers are used to generate keys for the upsampling process. The difference between these two networks is the number of input channels where the first network takes 4 channels and outputs a single channel and the second network takes 4 channels and returns 4 channels. Each network performs the sequential process of passing the input through convolution then batch normalization, and finally through the relu activation function before using this output as input in a similar process for a second convolution. Once the output of the second network is obtained, this is passed through a similar version of the encoding process. The difference between the methods is that the decoding blocks perform the inverse sequential process of encoding and decrease the input channels over each upsample cycle. This is also where the skip connections are added by using the downsample mask to assist in mapping the features to upsampled images. The final output is then passed through a single convolution layer before being applied to a soft max algorithm which returns the final model.

Training : After progress check-1, the model was trained for 26 epochs on Google Colab. After training for 26 epochs on Google Colab the estimated time would have taken days beyond the submission deadline. Hence the training was shifted on to a faster GPU at the Agave Cluster server at Arizona State University for training. Fortunately for training on different server, hardware and across the other datasets did not require any large changes in the code. The code from Google Colab was directly applicable to the Agave cluster server with the only exception of changing the directories and extremely minor syntax changes. However, before the training a custom anaconda environment was created with all the necessary libraries. This included installing python 3.7, pytorch 1.6.0, torchvision 0.7.0, pickle, nibabel, pandas, numpy, tensorboardx.

On all the datasets the training was done for approximately 1000 epochs unlike in the paper which states training for 8000 epochs and at least on the Brats dataset with a warm up strategy of 60 epochs. This is because of the time constraints due to submission deadline and also hardware and time allocation constraints on the Agave cluster server. Only once we were able to get access to a GPU for 4 days of continuous training while the other times only 4 hours of wall time was possible because we needed at least 1 GPU which had at least 16 GB memory for training on batch size of 4 which is the minimum batch size possible. This meant implementing the warm up strategy was not possible. Warm up strategy is basically when the model start with a learning rate much smaller than the ”initial” learning rate and then increase it over a few iterations or epochs until it reaches that ”initial” learning rate. This helps in setting up the direction of the loss towards the global minima of the loss function. It must also be noted that the warm up strategy is important since continuous training leads to dropping of loss significantly at once. In terms of training it is essentially a process of training for 60 epochs before the model continues training from the epoch number saved last time.

Testing : Testing of the model is implemented in a GPU enabled local machine with CUDA 10.2. As the model needs to tested on validation data set for BraTS 2020, , LiTS 2017 and KiTS 2019 . These are a set of highly detailed 3D images, using latest torch vision CUDA version was ideal and less computationally expensive.

A virtual environment in the local machine needs to be established with the dependencies like cudnn 8.4.0, cudatoolkit 10.2.89, torchvision 0.7.0, pytorch 1.6.0, image.io, pickle etc. Changes corresponding to versions of libraries been used in existing code had to be done, to minimize the compatibility issues. Also, some alteration with respect to the global paths were to be taken care of as the testing is done in local machine and with the model we trained. Preprocessing the validation set wasn't necessary as the data recieved was already processed. Testing on the model requires the dataset to be uniform, for which we had to crop the image in identical sizes. Also due to scarcity of validation dataset, performing data augmentation to the processed images was vital. Techniques like randomcrop(), randomflip() and randomrotate() are being used to expand the dataset and in turn tried to enhance the testing accuracy.



```
-------------------------------------------------------------------
Subject 1/125, BraTS20_Validation_001,
Successfully save C:\pythonProject1\transbts\TransBTS-main\output\submission\BraTS20_V
C:\pythonProject1\transbts\TransBTS-main\data\MICCAI_BraTS2020_ValidationData\BraTS20_
-------------------------------------------------------------------
Subject 2/125, BraTS20_Validation_002,
Successfully save C:\pythonProject1\transbts\TransBTS-main\output\submission\BraTS20_V
C:\pythonProject1\transbts\TransBTS-main\data\MICCAI_BraTS2020_ValidationData\BraTS20_
-------------------------------------------------------------------
Subject 3/125, BraTS20_Validation_003,
Successfully save C:\pythonProject1\transbts\TransBTS-main\output\submission\BraTS20_V
C:\pythonProject1\transbts\TransBTS-main\data\MICCAI_BraTS2020_ValidationData\BraTS20_
-------------------------------------------------------------------
```

Fig. 2. Snapshot of testing validation dataset

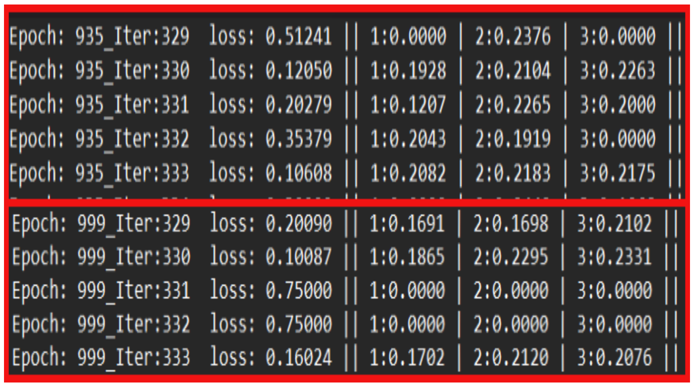
Testing for the validation dataset of 125 images can be seen inf Fig. 2. The validation of the images is computed through validatesoftmax() function which identifies the segmented tumors in the input images and further saves a set of segmented outputs. Outputs consists of two file types

(a) *.nii* : This format is a typical multidimensional neuroimaging file which consists of height, depth and width of each 3d image along with the details of each voxels. The format is to be validated by submitting the output on Upenn's school of medicines website which further will compare our output with the ground truth dataset convened by them.

(b) *.png* : This file format is for the visualization of the various tumors and can be pictorially compared from the input dataset.

V. Results

Training: 3 models were successfully trained on the, BraTS 2020, LiTS 2017 and KiTS 2019. As mentioned previously due to warm up strategy not being implemented properly one of the results that was observed is that the model of epoch 935 has lower training loss as compared to model of epoch 1000. To give a clarification on the impact when the training is resumed the weights initially allocated after which the loss drops is somewhere close to the weights of the model which was trained 60 epochs earlier. The loss does drop without the warm up strategy too however the training will take a longer time and it would probably never reach the absolute local minima. Hence one of the results observed is that the model of epoch 935 overall had a lower loss during training than model of epoch 999, which was the last epoch leading to the model of epoch 935 being used for testing. The lower loss can be observed since all the outputs in the console are actually logged in a text document [Fig. 3]. It can be seen that in epoch 935 the iteration 331,332,333 have lower losses than in epoch 999. And over the entire epoch 935 model has lower loss than the 999. For the other 2 datasets the 999 epoch models were used for testing.

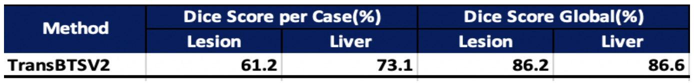


```
Epoch: 935_Iter:329  loss: 0.51241 || 1:0.0000 | 2:0.2376 | 3:0.0000 ||
Epoch: 935_Iter:330  loss: 0.12050 || 1:0.1928 | 2:0.2104 | 3:0.2263 ||
Epoch: 935_Iter:331  loss: 0.20279 || 1:0.1207 | 2:0.2265 | 3:0.2000 ||
Epoch: 935_Iter:332  loss: 0.35379 || 1:0.2043 | 2:0.1919 | 3:0.0000 ||
Epoch: 935_Iter:333  loss: 0.10608 || 1:0.2082 | 2:0.2183 | 3:0.2175 ||

Epoch: 999_Iter:329  loss: 0.20090 || 1:0.1691 | 2:0.1698 | 3:0.2102 ||
Epoch: 999_Iter:330  loss: 0.10087 || 1:0.1865 | 2:0.2295 | 3:0.2331 ||
Epoch: 999_Iter:331  loss: 0.75000 || 1:0.0000 | 2:0.0000 | 3:0.0000 ||
Epoch: 999_Iter:332  loss: 0.75000 || 1:0.0000 | 2:0.0000 | 3:0.0000 ||
Epoch: 999_Iter:333  loss: 0.16024 || 1:0.1702 | 2:0.2120 | 3:0.2076 ||
```

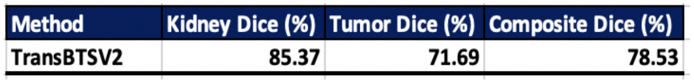
Fig. 3. Snapshot of difference in loss in iterations in epoch 935 and 999

Testing: Testing of 3 datasets was done successfully using the TransBTSv2 model. This includes the BraTS2020 dataset, LiTS 2017 dataset and kiTS 2019 dataset. The dice scores for LiTS 2017 and kiTS 2019 datasets are given below in Fig. 4 and Fig. 5 respectively. The testing output of BraTS 2020 dataset was sent to UPenn for evaluation, and there seems to be error in getting the dice scores back. The error originates from their server.



Method	Dice Score per Case(%)		Dice Score Global(%)	
	Lesion	Liver	Lesion	Liver
TransBTSV2	61.2	73.1	86.2	86.6

Fig. 4. Performance Comparison On LiTS 2017 Testing Set.



Method	Kidney Dice (%)	Tumor Dice (%)	Composite Dice (%)
TransBTSV2	85.37	71.69	78.53

Fig. 5. Performance Comparison On KiTS 2019 Testing Set.

The output of the test.py scripts is 2 folders: submission and visualization. The submission folder contains the model output in ".nii" format which is the most commonly used format for multi-dimensional neuroimaging data, this output folder was sent for Dice score calculations. The "visualization" folder contains the model output in ".png" format. Taking example of the BraTS dataset, we ran the model on 125 MRI scans, and the output was 159 images for each of the 125 MRI scans. As we can see in Fig. 6 the model gradually highlights the tumor in the brain MRI. The output images are initially completely black but as the image output increases from 1 to 159 more and more of the tumor gets highlighted, with the sharpest image of tumor being between the 75th & 80th image. The blue highlighted portion of the image in the tumor denotes the enhancing tumors, the red regions denote the non-enhancing tumors and the green region denotes the peritumoral edema. To get the understanding of the location of these regions they had be manually superimposed on an original sample of the training dataset. Fig. 7 and Fig. 8 show the original sample visualization in grayscale and location of the tumors after superimposition of the tumors sample output image onto the grayscale training sample image.

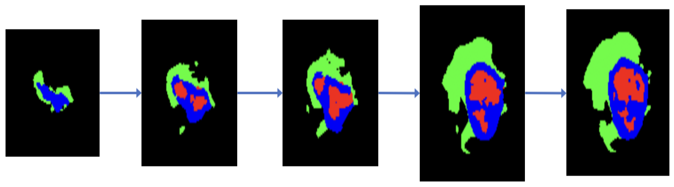


Fig. 6. Sample output images from testing. We can see the tumor gradually increase in the output image.

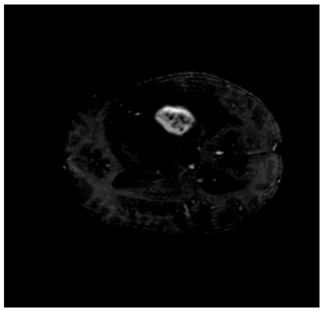


Fig. 7. Grayscale version of a sample training of training dataset showing the side view of the brain.

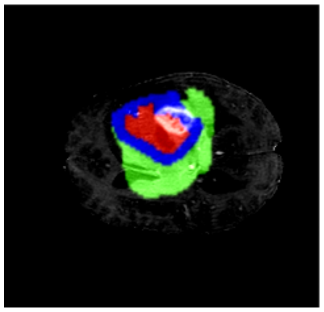


Fig. 8. Segmented output superimposed on the brain side view image to understand the location of the tumors.

Intermediate Layer Output Visualization: By visualizing the image output in the encoding and decoding steps it is possible to see the affect of the learning algorithm on the image [Fig. 9]. What is most interesting about these images is the ability to see how the boundaries change with the detection of different tissue edges. It is possible to see that the positional encoding of the pixels leads to the recreation of a nearly identical image in the decoding step by using a skip connection.

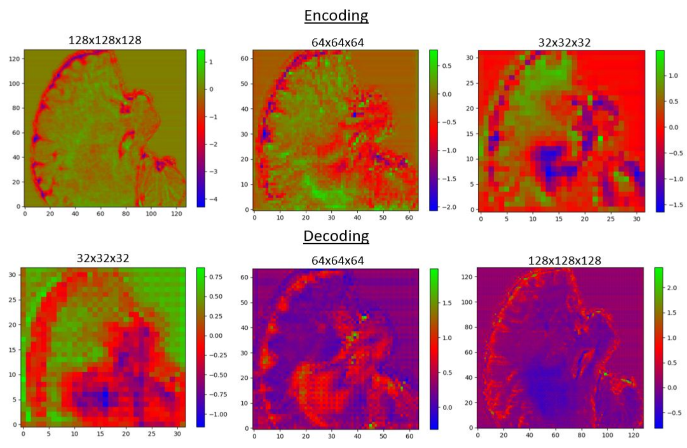


Fig. 9. Intermediate images from the encoding and decoding steps showing the affect of learning algorithms feature mapping

The most compelling images are the two 32x32x32 images which easily show the change to the image before and after the feature mapping process. In this image you can see the pixel values change creating more defined boundary while also highlighting inner areas of the image not previously highlighted. This shows the area and boundary detection occurring in the segmented image. By producing these images, one can better understand what the algorithm may be learning to detect and highlight in the training process.

VI. Discussion and Conclusion

A. Discussion

While it is unfortunate that we were not able to train for the entire 8000 epochs, learning was able to be shown within 1000. By showing intermediary images we could visualize the effect of the transformer on the learning algorithm.

From this project, we have learned that open-source machine learning projects are usually data intensive. Lot of hardware and time is required to train the model. Also, different techniques to handle the data (in our case the compression and decompression) may be required to handle the data when working with limited disk space.

Also, depending on the hardware, the code might need to changed or edited accordingly. Before we realised how memory intensive this algorithm was, steps were taken to lower the memory usage by decreasing the workers and batch size. Unfortunately, it was found that we would need 12 Gb of free RAM to run the training algorithm so we were forced to find an alternative to our local machines for training. With regard to python libraries, change in version can cause unprecedented number of errors and some of these maybe impossible to find a solution to.

Not only the hardware, even the OS used can cause significant impact on the working of the code and the time required to train the model. This culminated in a pytorch support issue between windows and linux as the pytorch 'nccl' backend used in this model is not supported on windows which is crucial since this backend supports distributed training and is much faster. Unfortunately, the 'gloo' backend was used initially when we tried to train on our local machines since were were not using distributed learning. However, once we switched to a colab workspace we changed the backend to 'nccl'.

As the entire project is dependent on such dynamic variables for the research to be reproducible the exact steps, versions, hardware, and software requirements need to be followed. If changes are needed one can expect discrepencies in results that might occur (like change in training time) due to the changes in any of the requirements.

Finally, the code structure of neural networks are fairly generalizable when it comes to a feed forward network. After learning these general code blocks, development becomes focused on how the data flows between these blocks rather than the actual neural network itself. This makes it much easier for us to take these general algorithm structures as inspiration for our own models.

If we were to do another large project such as this, the first priority would be to make sure we have enough hardware to train and test the model and also in a much easier way than using something like Google Colab as this has memory limits along with limited training time which deeply effected the ease of training.

Also, more planning needs to be done in the initial stages of the project to make sure that deliveries are on time so the project can progress according to plan. If this was creating the algorithm from scratch, there would also need to be more planning done so everyone understands the flow of information through the model rather than a couple key people.

B. Conclusion

The model was successfully trained and tested across multiple datasets; checking the validity of the paper and research itself as a generalizable detection algorithm. While the TransBTSv2 model is generalizable, we did not have enough training time for the models to reach to their global minimas. In addition to this, since the dice score of the TranBTS2020 dataset were not received, it cannot be verified whether this technique is state-of-the-art or not.

References

[1] J. Li, W. Wang, C. Chen, T. Zhang, S. Zha, H. Yu, and J. Wang, "Transbtsv2: Wider instead of deeper transformer for medical image segmentation," 2022.

every team member is required to complete this form, which will be reviewed and signed by other team members
by signing this form, you agree that the student's descriptions of contributions are factual
effort/contribution levels determine an individual's score which can be different from the rest of the team
This form does not contribute to page count.

Final Report
Team # 7

Student name: Reed Axman	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to specific paragrphs, equation #, figure #, code line #'s sections, etc…)		Worked on implementing training and visualizing intermediary images of the model	implemented and generated intermediary images	Wrote implementation section for the model. Wrote sections in regard to visualizing intermediary images and proceeded to co-write: abstract, introduction, discussion and conclusions, and methods.	edited powerpoint and model and DBM added slides	Kaushal Rathi	Harkirat Singh Chahal	Rishi Dadhich

Student name: Kaushal Rathi	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to specific paragrphs, equation #, figure #, code line #'s sections, etc…)		Datapreprocessing which involved compression and decompression of data. Trained the model on all 3 datasets (BraTS2020, KiTS19,LiTS17) across multiple platforms like Local machine, Google Colab, Agave Cluster.	Trained the model on all 3 datasets for approximately 1000 epochs each. Created code for compression and decompression of data too.	Wrote implementation and methods "Training" subsection and lessons learned section. Proceeded to co-write: abstract, introduction, discussion and conclusion.	Edited powerpoint slides	Reed Axman	Harkirat Singh Chahal	Rishi Dadhich

Student name: Harkirat Singh Chahal	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3

specific & detailed evidence is required to support claims of contributions (make reference to specific paragrphs, equation #, figure #, code line #'s sections, etc…)		Worked on testing the model. Preprocessed the data before running the model on it. Ran and debugged the test.py file. Did testing on LiTS, KiTS & BraTS datasets. Did the testing on local machine, using PyCharm SDE and the required python libraries.	Ran the testing files concurrently with my teammate Rishi to fastrack the process. tested the models on all 3 datasets and generated ouputs in .nii format.	Wrote the results section of the report and contributed towrds the "testing" subsection in the "Implementation" section.	Made the "results" & "testing" slides in the final ppt.	Kaushal Rathi	Reed Axman	Rishi Dadhich

Student name: Rishi Dadhich	worked on literature	worked on implementation (data, platform, test run, debug, compatibility…)	generated results (run results, result data processing, presenting results	wrote report (Intro, method, result, discussions, …)	other significant contributions	peer approval 1	peer approval 2	peer approval 3
specific & detailed evidence is required to support claims of contributions (make reference to specific paragrphs, equation #, figure #, code line #'s sections, etc…)		Worked on testing the model on visualization dataset for LiTS, KiTS & BraTS datasets. Performed the testing on CUDA environment and recorded the visualization results for all the dataset. Also contributed towards data augmentation and tayloring the dataset before passing to the model.	Tested the model on LiTS, KiTS & BraTS datasets. Recorded the output .nii files and visualization files to be verfified from UPenn and compared our results with benchmark results.	Wrote implementation and methods "Testing" subsection and lessons learned section. Also contributed towards defining the result section in the report.	Edited powerpoint slides in testing and result section.	Kaushal Rathi	Reed Axman	Rishi Dadhich