Skull Stripping Quality Assessment Tool Report

CPSC 8650-001

Team 3

Joey Binz, Michael Harris, Jacob Schechter

Github repository: https://github.com/mhrrs/Skull-Stripping-CNN-Classifier

4/21/2023

Abstract - Brain segmentation, often referred to as skull-stripping, extracts the brain tissue from a raw 3D MRI image of the brain, removing identifiable characteristics such as the skull and facial tissues. However, ensuring the quality of segmentation results remains tedious with highly variability amongst evaluating researchers.

In this paper, we propose an approach of using 3D CNN (Convolutional Neural Network) models trained on BSE (Brain Surface Extractor) and BET (Brain Extraction Tool) brain scan datasets to evaluate the process quality. We determined its quality based upon two requirements: 1) Determine whether a brain scan is skull-stripped effectively, removing facial features 2) If any voxels are removed resulting in unintentional brain loss. To improve our ability to gauge the effectiveness of the skull-stripping algorithm, two comparative models with differing layers were utilized.

The results are demonstrated via the metrics of facial-feature presence and appearance of brain loss across a set number of epochs.

Keywords- ***MRI; Brain Segmentation; Skull-stripping; 3D CNN; Voxel; BSE; BET; Epoch***

1 Introduction

MRI (Magnetic Resonance Imaging) is a widely used, non-intrusive method for human brain study and disease diagnosis with various image types including: T1-weighted (T1W), T2-weighted (T2W), and Diffusion-weighted (DWI). As the usage of the modality continues to increase, the introduction of brain segmentation such as BSE and BET aided in improving the performance of neuroimaging analysis. However, in compliance with HIPAA, brain segmentation algorithms must effectively capture solely the brain tissue; removing personally identifiable facial features. The process is time intensive and varies greatly in accuracy amongst individual researcher evaluation, thereby creating the possibility for misdiagnosis. Hence the necessity for an effective and consistent skull-stripping evaluation tool [2].

Our project seeks to address the issues presented in poor skull-stripping process quality. More specifically, our modality will satisfy two key issues: 1) To determine whether the facial features are removed to the extent such that researchers cannot identify the person by utilizing the facial features that remain in the segmented image 2) Determine whether any brain voxels are unintentionally removed, resulting in the loss of important brain information. Our solution involved a 3D CNN approach due to its superior performance over other algorithms [9]; we aim to improve quality evaluation through a comparative model study. The proposed comparative models found in Section 4, seek to improve the accuracy as well as the efficiency of the skull-stripping process. Providing a far more consistent utilization of the currently available neuroimaging data as will be reaffirmed within Section 5. Thus advancing the field of neuroscience and improving our ability to analyze the human brain.

2 Background

2.1 3D CNN Overview

3D CNNs are a type of neural network that are designed to process and classify three dimensional data such as videos or brain scans. Unlike 2D CNNs, this CNN creates learnable features across three spatial dimensions. This is desirable for medical practices as it creates more robust and complex models that can tackle issues that take in each frame of a scan into account. One of the many challenges of this model is that it is more complex to train due to image data having a three-dimensional structure and the number of layers present in the model. This results in a computational complexity represented as:

$$(size\ of\ input\ data)\ x\ (No.\ of\ input\ channels)\ x\ (size\ of\ convolutional\ filters)\ x\ (No.\ of\ output\ channels)$$

However, in the use case of this project due to the dataset size and structure of our models, this has not been an issue.

2.2 3D CNN Layers

Similarly to a 2D CNN, a 3D CNN consists of multiple layers such as: input layers, convolutional layers, ReLU activation layers, pooling layers, dropout layers, fully connected layers, and an output layer. It is due to the additional dimension afforded within a 3D CNN by which they differ. A 3D convolutional layer performs a convolution on the input data, this is done by sliding a 3D filter over the input according to the specifications of the kernel which is laid out by the model architecture. This convolution creates a set of features maps that is then used in the next layer of the model. The number of filters specified in the architecture is what produces the number of features maps. A basic 3D CNN model demonstrates its functionality as illustrated in [9, Fig. 2.1].

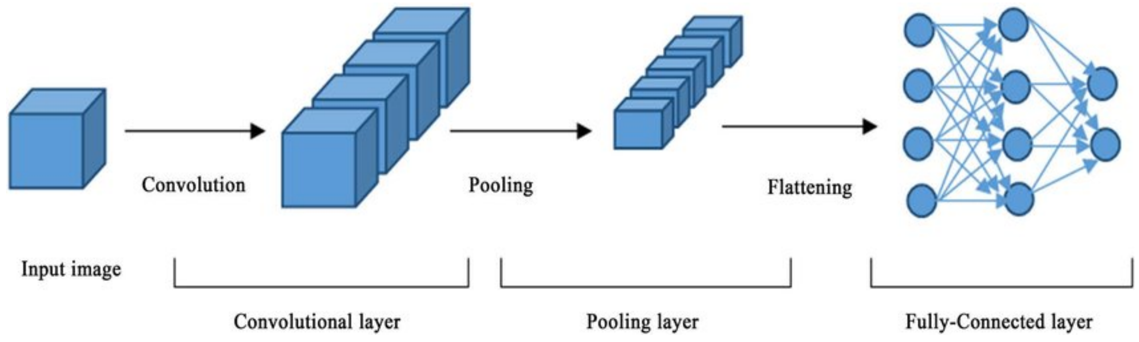


Figure 2.1: Basic 3D CNN architecture

3 Dataset & Data Pre-processing

3.1 BSE_BET IXI Dataset

The BSE_BET dataset contains skull stripped MRI brain scans of various quality and was the primary source of data for the project. The dataset contained 713 instances of recognizable facial features being left behind by skull stipping tools, 1332 instances of data loss, and 15 perfect instances of skull stripping as detailed in Figure 3.1.

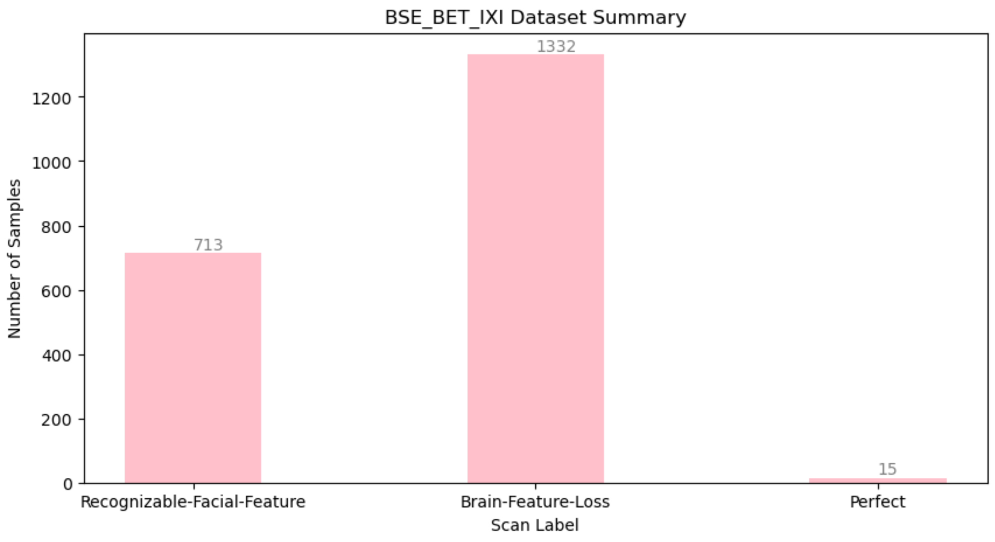


Figure 3.1: Bar Graph - BSE_BET Dataset Summary

3.2 Computational NeuroImaging Labs Perfect Dataset

An additional 125 MRI scans were taken from the NFBS dataset publicly available online. The scans had been hand edited after using skull stripping tools & contained no data loss or no recognizable facial features. They were added into the model's training data and properly extracted using scripts available in the project's github under the processing_scripts folder. Figure 3.2 below illustrates improvements to the dataset.

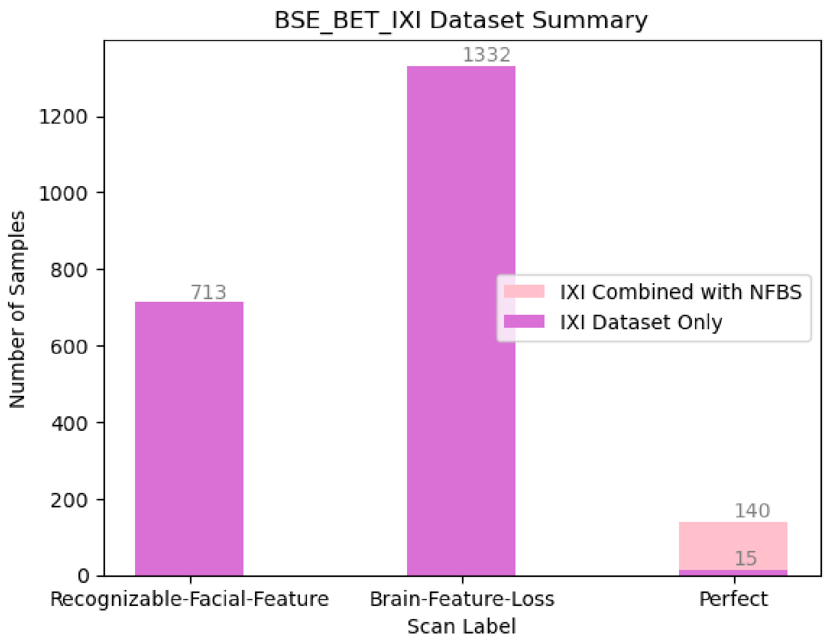


Figure 3.2: Bar Graph - Improved BSE_BET Dataset Summary

Examples of slices taken from the 3D data that were then projected as 2D images are demonstrated below in Figures 3.3 and 3.4

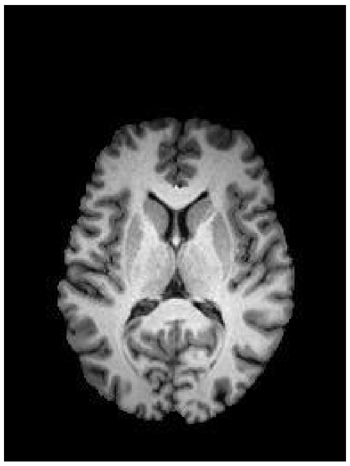


Figure 3.3: Perfect Brain Scan

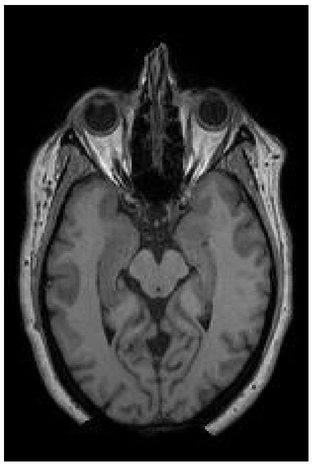


Figure 3.4: Brain Scan with Facial Features

Figures 3.3 and 3.4 respectively illustrate a perfect brain scan followed by a brain scan with recognizable facial features taken from the dataset.

3.3 Data Pre-processing Measures

Pre-processing the data involved using basic python scripts to first extract the two datasets & combine them into one folder with one unified label file in a proper format of (filename, yes/no, yes/no) where the yes/no responses stand for recognizable facial features and data loss respectively. After this our processing script reads through the data and separates each file into folders based on each file’s label (one folder for recognizable facial features, one folder for brain feature loss, and one for perfect scans). The data from each individual file is then loaded into a numpy array using python’s numpy library & the niibabel library [3].

From here the numpy arrays are stored in order to improve computation time in the future, and the nifti file data is resized to a desired dimensions of 64x128x128 in order to be processed using less memory. Additionally the data is normalized so that the 3dCNN is less affected by outliers.

Next the numpy arrays are shuffled to further generalize the data and are loaded (along with their corresponding labels) into training and testing dataloaders provided by the pytorch dataloader library [4]. The test and train split was 70% train and 30% testing. The model never encounters any of the testing data until it is evaluated for accuracy after training, in order to see how the model handles new data.

The data loader takes the numpy arrays and converts them into pytorch tensors with a batch size of 4, the batch size was kept low to help use less resources on the machine training the data since 3D data uses memory much faster than 2D. The data loader then feeds the data into the model directly during training and testing.

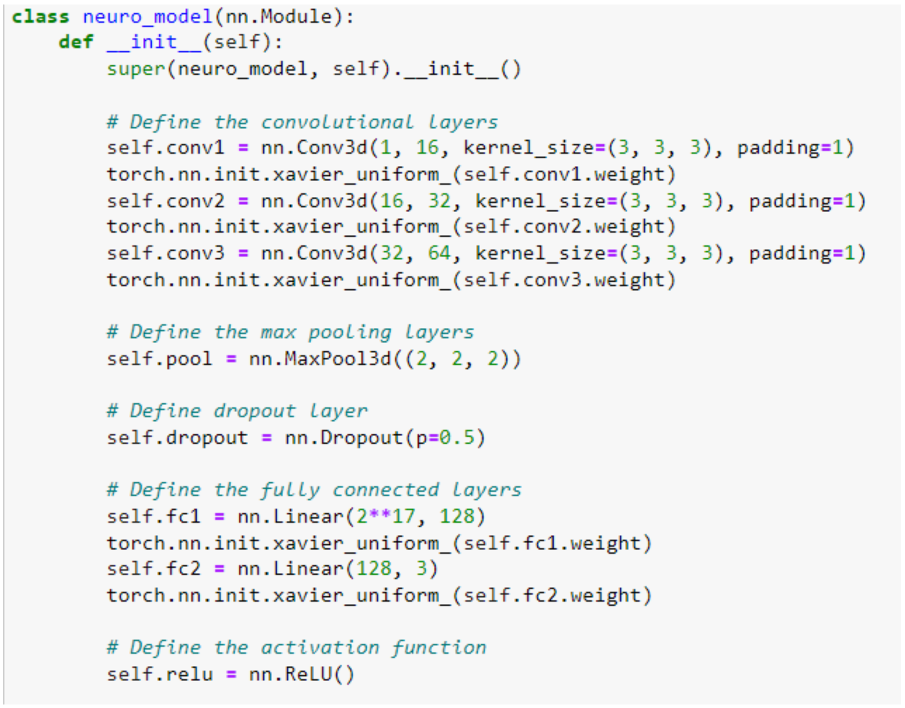
4 Model Details

This section lays out in detail the two models we used to solve the problem space mentioned earlier. Model 1 is a slightly simpler architecture with less optimizations in it’s training loop while model two attempts to add training optimizations and implements a slightly different model architecture.

4.1 Model 1

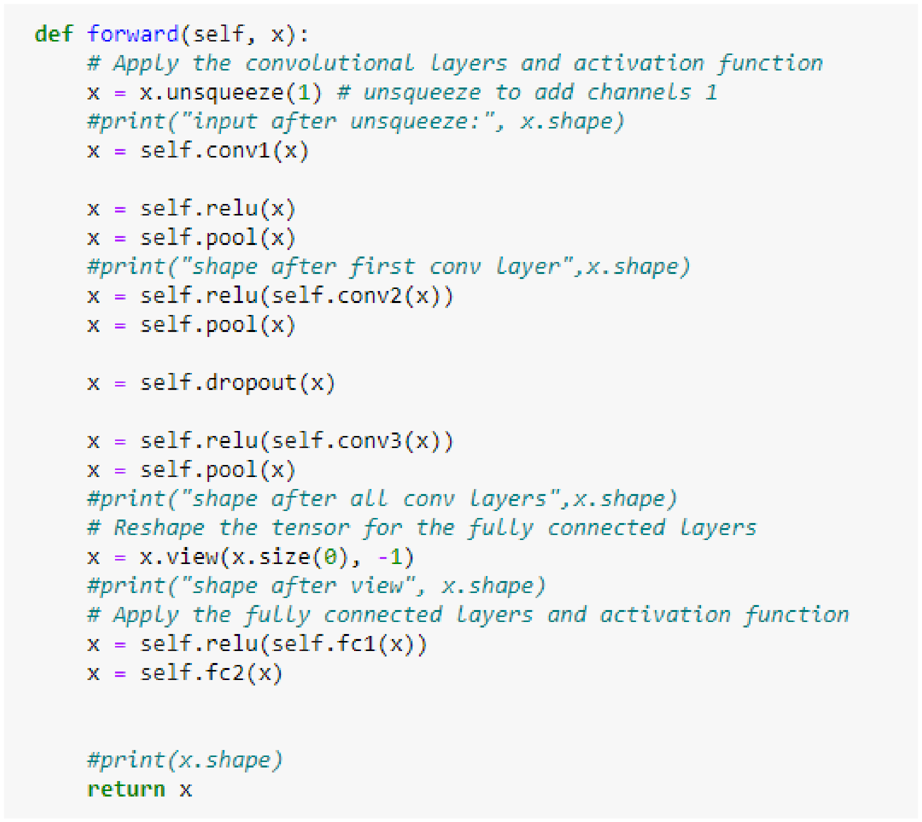
Model 1 was a 3D CNN consisting of three Conv3d layers all with kernel size of 3, followed by a max pooling layer (of size 2x2x2), a dropout layer with a rate of 0.5, and two fully connected layers. Additionally ReLU was used as the activation function. The convolutional layers had feature inputs and outputs of (1, 16) in the first layer, (16, 32) in the second layer, and (32, 64) in the third layer. The fully connected layer outputted a single dimension layer of size = 3 which represented the model's prediction. The model outputted binary labels where the index of the 1 output in the array signified the model’s prediction (index 0 predicts data loss, index 1 predicts recognizable facial features, and index 2 predicts a perfect scan). The dropout layer in the model was used to randomly dropout weights in the model with a probability of 50% in order to help generalize the model. A higher than usual probability for the dropout layer was chosen to compensate for the small dataset the model was trained on.

Figures 4.1 and 4.2 detail the Pytorch implementation of model 1 below.



```python
class neuro_model(nn.Module):
    def __init__(self):
        super(neuro_model, self).__init__()

        # Define the convolutional layers
        self.conv1 = nn.Conv3d(1, 16, kernel_size=(3, 3, 3), padding=1)
        torch.nn.init.xavier_uniform_(self.conv1.weight)
        self.conv2 = nn.Conv3d(16, 32, kernel_size=(3, 3, 3), padding=1)
        torch.nn.init.xavier_uniform_(self.conv2.weight)
        self.conv3 = nn.Conv3d(32, 64, kernel_size=(3, 3, 3), padding=1)
        torch.nn.init.xavier_uniform_(self.conv3.weight)

        # Define the max pooling layers
        self.pool = nn.MaxPool3d((2, 2, 2))

        # Define dropout layer
        self.dropout = nn.Dropout(p=0.5)

        # Define the fully connected layers
        self.fc1 = nn.Linear(2**17, 128)
        torch.nn.init.xavier_uniform_(self.fc1.weight)
        self.fc2 = nn.Linear(128, 3)
        torch.nn.init.xavier_uniform_(self.fc2.weight)

        # Define the activation function
        self.relu = nn.ReLU()
```

Figure 4.1: Pytorch Implementation of CNN Model



```python
def forward(self, x):
    # Apply the convolutional layers and activation function
    x = x.unsqueeze(1) # unsqueeze to add channels 1
    #print("input after unsqueeze:", x.shape)
    x = self.conv1(x)

    x = self.relu(x)
    x = self.pool(x)
    #print("shape after first conv layer",x.shape)
    x = self.relu(self.conv2(x))
    x = self.pool(x)

    x = self.dropout(x)

    x = self.relu(self.conv3(x))
    x = self.pool(x)
    #print("shape after all conv layers",x.shape)
    # Reshape the tensor for the fully connected layers
    x = x.view(x.size(0), -1)
    #print("shape after view", x.shape)
    # Apply the fully connected layers and activation function
    x = self.relu(self.fc1(x))
    x = self.fc2(x)


    #print(x.shape)
    return x
```

Figure 4.2: Continued Pytorch Implementation of CNN Model

The model was trained on the combined IXI Dataset and the perfect scans acquired from the NFBS Skull-Stripped Repository [7]. A 70:30 train test split was created for cross validation testing. Input into the data loader was shuffled to ensure the model was not just memorizing the order of the training data.

Model 1 was trained using Cross entropy loss & the Adam optimizer over 25 epochs and achieved an accuracy of 98% on the test set and an accuracy of 99% on the training set. The loss graph for the training is illustrated below in Figure 4.3. A 98% accuracy is exceptionally high in most machine learning problems and in this scenario the high accuracy is likely due to the small dataset and a lack of complexity in the problem. Because of the smaller number of potential input situations, the model is able to learn most of the possible inputs/outputs in the dataset with relative ease. Training on a larger dataset would likely yield lower accuracy, but an overall more generalizable model than the current trained model.

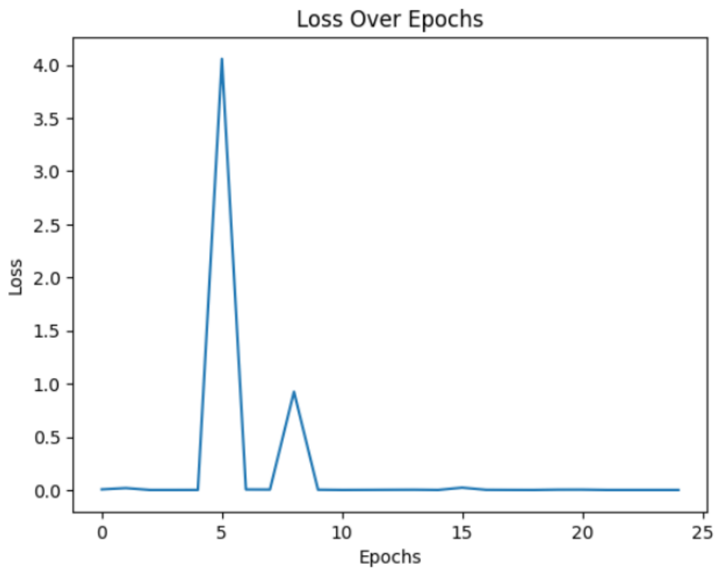


Figure 4.3: Plot of Cross Entropy Loss

The weights for the model were initialized using pytorch's xavier_uniform function in order to initialize the weights to a uniform distribution and make training more consistent. Prior to making this optimization the model would periodically have runs which were much lower accuracy (60% accuracy after 20 epochs). After initializing the weights this problem was unable to be reproduced. Additionally the loss was training much more consistently after weight initialization, starting at around 0.5-0.2 and finishing as low as 0.00001 on some runs. Generally the loss seemed to flatten out around 22-25 epochs. Occasionally before the weight initialization loss would jump around from 4-0.5 for most of training and the model would have difficulty converging. The below figure illustrates the loss over one of the "bad" 60% accuracy runs before weights were consistently initialized.

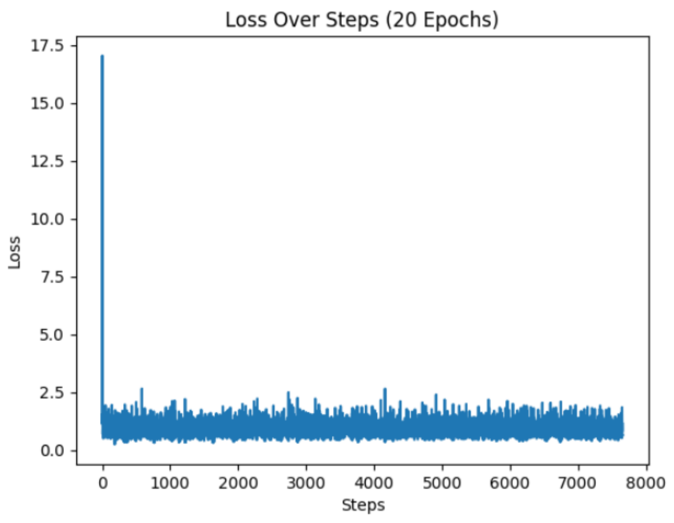
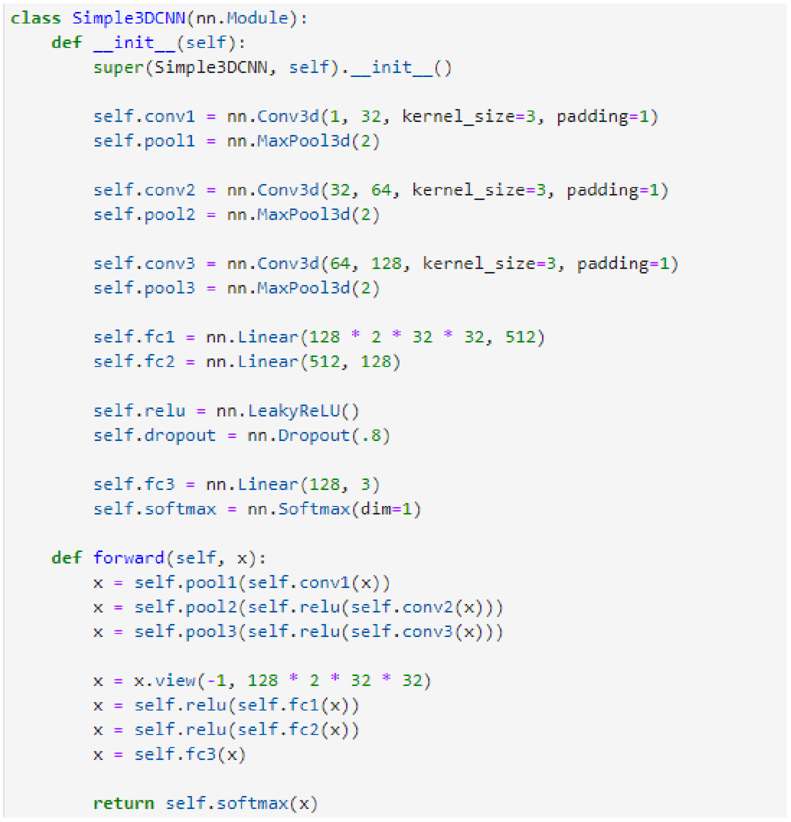


Figure 4.4: Plot of Loss over Steps in Model 1

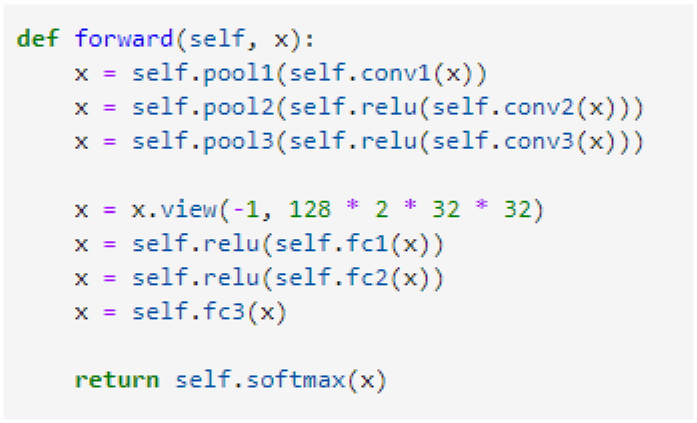
4.2 Model 2

Model 2 also uses three layers of 3D convolutions, but with an overall deeper network. Each 3D convolutional layer uses a kernel size of 3x3 and a max pooling size of 2x2x2. Where the filter size of the first layer is 32, the second layer is 64, and the third layer is 128. All of this is then followed by three fully connected layers with over 260k parameters and 512 neurons in the first layer, 512 parameters and 128 neurons in the second layer, and 128 parameters and 3 neurons in the third layer. This model uses Leaky ReLU as its activation function and Softmax for its output. Figures 4.5 and 4.6 demonstrate the Pytorch summary of model 2.



```
class Simple3DCNN(nn.Module):
    def __init__(self):
        super(Simple3DCNN, self).__init__()

        self.conv1 = nn.Conv3d(1, 32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool3d(2)

        self.conv2 = nn.Conv3d(32, 64, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool3d(2)

        self.conv3 = nn.Conv3d(64, 128, kernel_size=3, padding=1)
        self.pool3 = nn.MaxPool3d(2)

        self.fc1 = nn.Linear(128 * 2 * 32 * 32, 512)
        self.fc2 = nn.Linear(512, 128)

        self.relu = nn.LeakyReLU()
        self.dropout = nn.Dropout(.8)

        self.fc3 = nn.Linear(128, 3)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.pool1(self.conv1(x))
        x = self.pool2(self.relu(self.conv2(x)))
        x = self.pool3(self.relu(self.conv3(x)))

        x = x.view(-1, 128 * 2 * 32 * 32)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)

        return self.softmax(x)
```

Figure 4.5: Pytorch Implementation of CNN Model 2



```
def forward(self, x):
    x = self.pool1(self.conv1(x))
    x = self.pool2(self.relu(self.conv2(x)))
    x = self.pool3(self.relu(self.conv3(x)))

    x = x.view(-1, 128 * 2 * 32 * 32)
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    x = self.fc3(x)

    return self.softmax(x)
```

Figure 4.6: Pytorch Implementation of CNN Model 2

This model takes an input of data collected from the BSE_BET dataset and a third-party dataset that contains only perfectly stripped brains. This extra dataset allows us to switch this from a binary model into a multi-class classification model. It can determine whether there are still facial features, brain loss, or if the brain is perfectly stripped and intact. The data was split into a 70:30 for training and validation datasets. The data loader was set to shuffle in order to make sure the model wasn't just memorizing the data.

Model 2 used a training loop that utilized automatic mixed precision to speed up the training of the model and linear learning rate decay to help it converge. The model used cross entropy loss and adam as its optimizer. It was trained on 20 epochs and received mixed results. Sometimes testing as high as 97% and other times as low as 33%. I believed that this reason was due to weight initialization, but after adding that into the mix, the performance stayed at 33%. I believe the model would have benefitted from an addition of dropout and or batch normalization in conjunction with weight initialization. The model also could have benefited from a larger dataset. Using data augmentation would have most likely lowered performance, but increased the stability and generalization ability of the model.

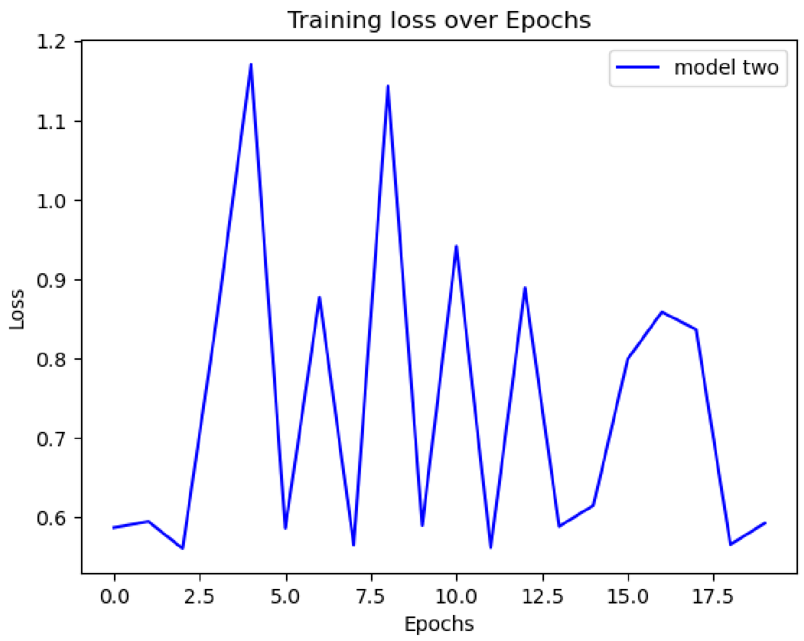


Figure 4.7: Plot of Training Loss over Epochs of Model 2

After discussing techniques with my teammates, I switched from using kaiming_normal for my weight initialization to xavier_uniform. The performance of my model skyrocketed to 97%. This performance was consistent through every training session as well.

In conclusion, both models were capable of training the image data at this size despite optimization and other techniques. The real determinant of what model would perform better would be an increase of data. This way we could more accurately determine the ability of the model to generalize.

5 Limitations

The limitations of both models would be overfitting, limited data, and interpretability issues. It is likely that both models could have been overfitting to the data we presented to them due to the complexity of their architecture and the lack of data. The issue of limited data most likely is what led to our early issues of variability in training performance. Another limitation was interpretability of our model's results. While it only needed to classify between three different classes, there are questions about what its results mean. For example, if there was a scan of a brain that had been attempted to be stripped but in the process still had visible facial features but also brain tissue loss, would it be classified as brain tissue with loss or a scan with visible facial features. Because it has to be one of those three classes and there is no room to check the quality of the stripped photos before they are inserted into the model when it's in production, it could give inaccurate results.

6 Future Directions and Improvements

An obvious direction for improvement would be to gather a larger and more diverse training dataset to train the model on. Given the small size of the dataset, the model was able to give very high performance. However we theorize that if a new scan was presented to it (from outside the dataset as a whole), the model would be very prone to being inaccurate due to unseen patterns which may result from MRI machine used, age of the person being scanned, any underlying conditions causing their brain makeup to be different, or a variety of other reasons. Additionally training on an even amount of scans with each label would further increase the quality of the dataset & trained model.

Better data preprocessing could also help to further generalize the model. Zunair et al. implemented further techniques such as rotating the brain images slightly and resizing the scans themselves [10]. This helps ensure that the CNN is learning the patterns of the brain itself, and not things that are inherent in nature to the image format or irrelevant to the problem space.

Furthermore a more complex 3D CNN architecture could be used implementing more dropout layers (to further help with generalizability) and performing more hyper parameter tuning to marginally increase performance.

7 Conclusion

After the research and implementation of the 3D Convolutional Neural Network model outlined in this report it would follow that a 3D CNN is a valid approach to solving this problem

& shows potential in the amount of optimizations that could be further applied to push the performance of a 3D CNN model forward.

8 References

[1] Ahmed, E., Saint, A., Shabayek, A. E. R., Cherenkova, K., Das, R., Gusev, G., ... & Ottersten, B. A survey on deep learning advances on different 3D data representations. arXiv preprint arXiv:1808.01462. (2018).

[2] A. Fawzi, A. Achuthan, and B. Belaton, "Brain Image Segmentation in recent years: A narrative review," *Brain sciences*, 10-Aug-2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8392552/. [Accessed: 21-Apr-2023].

[3] Eckert, M. (2023). "BET_BSE_IXI_Dataset" [Online] available: https://dyslexia.computing.clemson.edu/BET_BSE/

[4] M. Brett et al., nipy/nibabel: 5.1.0. Zenodo, 2023. doi: 10.5281/ZENODO.7795644.

[5] Newman, A. J., Godfrey, D., &amp; Post, R. (2021). Data Science for Psychology and Neuroscience - in Python. Department of Psychology & Neuroscience, Dalhousie University.

[6] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[7] Puccio, B., et al. (2016)., NFBS skull-stripped repository. Retrieved April 21, 2023, from http://preprocessed-connectomes-project.org/NFB_skullstripped/.

[8] T. Ahmed, M. S. Parvin, M. R. Haque, and M. S. Uddin, "Lung cancer detection using CT image based on 3D Convolutional Neural Network," Journal of Computer and Communications, 04-Mar-2020. [Online]. Available: https://www.scirp.org/journal/paperinformation.aspx?paperid=98675. [Accessed: 21-Apr-2023].

[9] Z. Akkus, A. Galimzianova, A. Hoogi, D. L. Rubin, and B. J. Erickson, "Deep Learning for Brain MRI segmentation: State of the art and Future Directions," *Journal of Digital Imaging*, vol. 30, no. 4, pp. 449–459, 2017.

[10] Zunair, H., Rahman, A., Mohammed, N., & Cohen, J. P. (2020). Uniformizing techniques to process CT scans with 3D CNNs for tuberculosis prediction. In Predictive Intelligence in Medicine: Third International Workshop, PRIME 2020, Held in Conjunction with MICCAI

2020, Lima, Peru, October 8, 2020, Proceedings 3 (pp. 156-168). Springer International Publishing.