Ashton Sobeck
Sushant Kot
Zack McGeehan
Team 11
CPSC 8650
20 April 2022

Team 11 Final Project Report

The Problem

In order for neuroscience researchers to study and conduct research on MRI brain scans, each scan needs to comply with HIPAA in order that the subjects are not able to be personally identified by anyone viewing the research. To make these MRI scans HIPAA compatible, researchers utilize a technique called skull-stripping to remove the identifiable data from the MRI scans. Many algorithms such as BET and BSE are used to strip the MRI brain scans of the necessary data. These algorithms help neuroscience researchers immensely, but sometimes have poor accuracy of removing identifiable information or even removing brain information. On top of these issues, researchers have to manually verify if the skull-stripping process was effective on the images passed in. This is time consuming for the researcher and takes away from the actual research that the researcher is conducting.

We have been tasked with creating a tool to help researchers in their efforts to identify the MRI scans that have been adversely affected by the skull-stripping process. By having a process to identify if a MRI brain image is personally identifiable or has brain feature loss, researchers will be able to spend less time parsing through their skull-stripped images, and more time conducting the research they are wanting to perform.

Our Programming

To solve this problem of alleviating the time taken for doctors to identify skull-stripped MRI scans, we used python, with the help of the libraries nibabel and keras. Nibabel is a python library that can open .nii files and extract the data from them. Keras is a deep-learning library for python and allows for intuitive creation of neural networks. Training is abstracted and the history of loss and accuracy during training is available to plot easily. To host our code, we utilized Github for easy access and updating between group members. The repository where our code is located is here: https://github.com/ashsobeck/CPSC8650GroupProjectTeam11 .

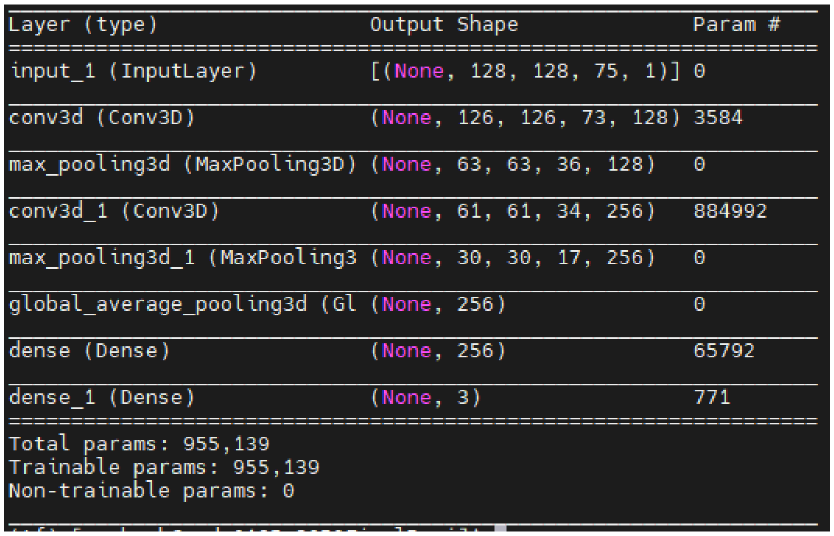
Our Data and Data Preparation

Dr. Wang supplied the class with a dataset of BET/BSE skull-stripped MRI scans. In total, there were 1,311 MRI scans that had one of three classes. The classes were personally identifiable with no brain-feature loss, not personally identifiable with brain-feature loss, and not personally identifiable and no brain-feature loss. Each scan had a height and width of 256, but a variable depth with a maximum of 150. The main MRI scan dimension was 256x256x150.

For the dataset, we utilized a 85/15 training and testing set split. This decision is based on how the MNIST dataset split. We created an array of labels and matched them up with the appropriate array of data from the files to create training images, training labels, testing images, and testing labels.

Initially, our plan was to process the full size of the MRI scan. Upon trying to train the model, we ran into overflow issues with tensorflow not being able to make computations with the full size MRI scan. Additionally, in order to use the full size of the data, we had omitted MRI scans that had a depth less than 150. To remedy these problems, we subsampled the data so that every MRI scan was 128x128x75. This cut the size of the data in half, and tensorflow was able to handle the computations. Subsampling also allowed us to scale depths less than 150 down to 75 since we were manipulating all of the data.

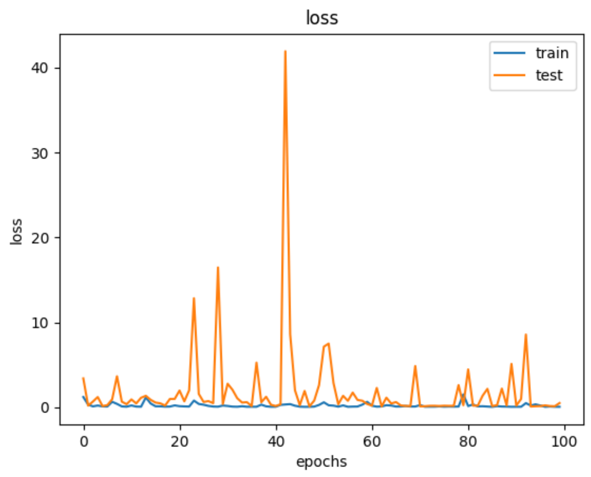
Our Model

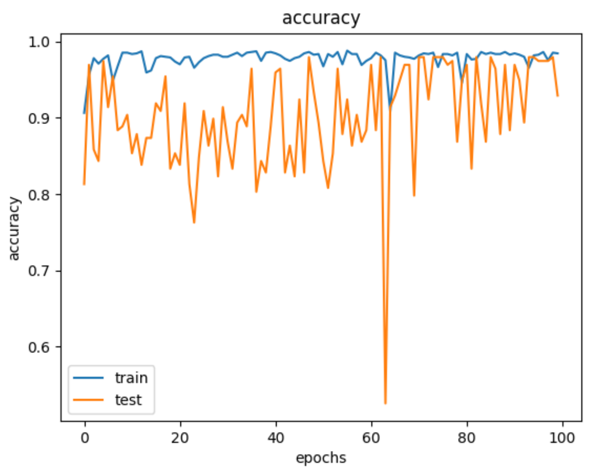


```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 75, 1)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 126, 126, 73, 128) 3584
_________________________________________________________________
max_pooling3d (MaxPooling3D) (None, 63, 63, 36, 128)   0
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 61, 61, 34, 256)   884992
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 30, 30, 17, 256)   0
_________________________________________________________________
global_average_pooling3d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 955,139
Trainable params: 955,139
Non-trainable params: 0
_________________________________________________________________
```

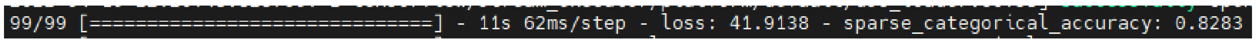
For our model architecture, we constructed a model of 2 three dimensional convolutional layers with max pooling layers in between. After the convolutional layers, we used a feed forward layer to classify the image in the three labels mentioned above. The convolutional layers and first dense layer used the Rectified Linear Unit activation function, while the classifying dense layer used the softmax activation function. The output of the model is a vector of size 3, with each number in the vector representing a probability that the input data is a certain label.

To start off, we trained our model for 100 epochs with the Adam optimizer, a .001 learning rate, cross entropy loss, and a batch size of 1. We used a batch size of 1 due to the data size issues that we had on Palmetto. In hindsight, this might have been able to be larger due to the subsampling of our data. We are able to observe how the testing loss and accuracy is doing during training by passing in validation data to the model after every epoch. Below is the loss and accuracy evaluations that were recorded after training:





Keras has an evaluate function to run test data on the model. Using the best weights found during the training, we had a testing accuracy of ~82%.

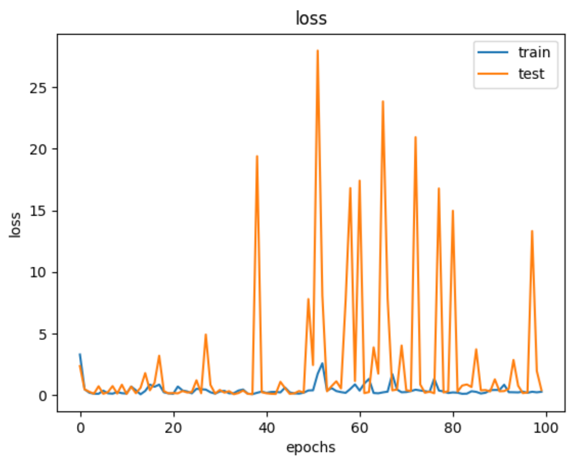


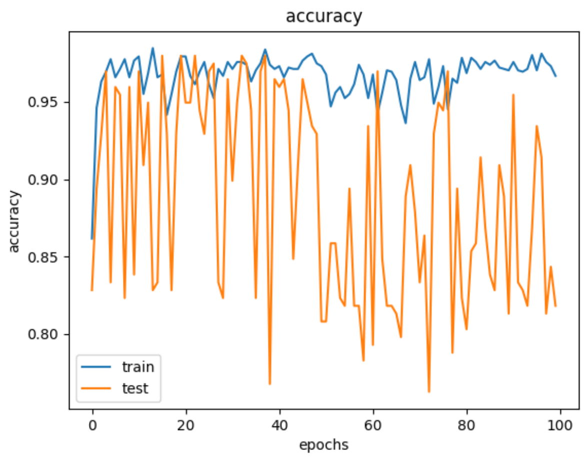
```
99/99 [==============================] - 11s 62ms/step - loss: 41.9138 - sparse_categorical_accuracy: 0.8283
```

However, we can observe that the testing loss and accuracy were very unstable during training. We believe that the testing loss and accuracy curves were so unstable because the model overfit the training data as the training curves are much more smooth. We were determined that we could do better and generate a smoother curve to convergence of our model.

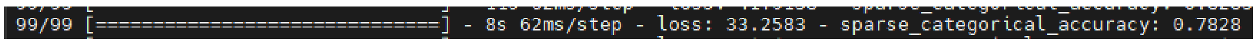
Our Iterations on the Model

Our first thought that would help us to remedy our issues with unstable graphs of loss and accuracy would be adding in some regularization to the model's architecture. To achieve this regularization, we added in dropout layers after the second convolutional layer and after the first dense layer. Dropout layers will randomly set a node of the model to 0 based off of a percentage threshold. Having this dropout helps prevent overfitting of the data. We did not want to change too much at once, so we kept the hyperparameters of batch size, learning rate and epochs the same. Below is the testing and training loss and accuracy curves from the next iteration of training that we did.





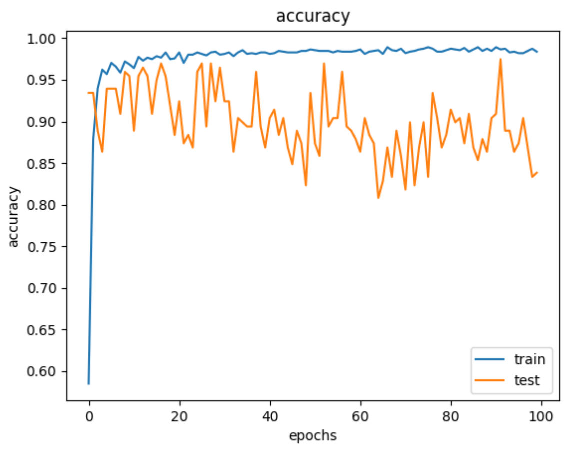
Using Keras' evaluate function, the testing accuracy dropped to 78%. The testing loss dropped by about 7 which is very desirable.

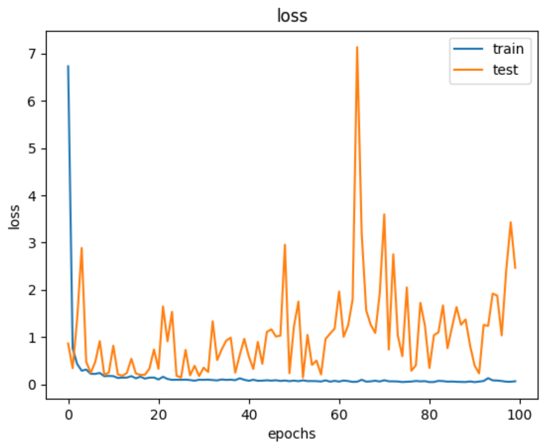


```
99/99 [==============================] - 8s 62ms/step - loss: 33.2583 - sparse_categorical_accuracy: 0.7828
```

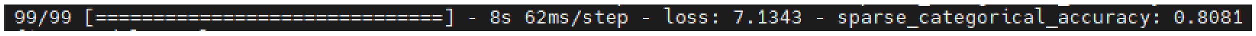
From the graphs, there are still large spikes in the testing loss and accuracy. While there are still large spikes, the range of these spikes are much lower. The maximum testing loss before the dropout layers was 40, and now the maximum testing loss is ~27. While the ranges are lower, the model is more sporadic and unstable than it was before. There are just as many if not more spikes in the testing loss and accuracy, and the training loss and accuracy curves are not as smooth as before. We believed that there was more performance to be achieved with this model.

To try and smooth out the curves, we looked at adjusting the hyperparameters. For our next iteration of training, we decreased the learning rate from .001 to .0001 and increased the batch size from 1 to 2. Now that we have improved our architecture, we aim to improve our hyperparameter choice. Below is the training and testing loss and accuracy curves from our next iteration of training.





The changes in the hyperparameters helped our model immensely. The training curves on both loss and accuracy smoothed out almost completely. The test loss and accuracy curves are still unstable and spiking, but the ranges are now very small. The maximum loss is down from 40 initially to 7. The training accuracy curve does indicate that there could be some overfitting due to how high the accuracy is, but the overall result is much better than our first attempt at training the model. Using Keras' evaluate function, the testing accuracy rose back to 80%, and the test loss decreased to 7.



```
99/99 [==============================] - 8s 62ms/step - loss: 7.1343 - sparse_categorical_accuracy: 0.8081
```

This small change in our model shows how important hyperparameter choice is.

Lessons Learned

Throughout the process of creating this model to classify an MRI brain scan, we have learned many lessons.

We first learned how to explore data, and how to create a dataset from an existing dataset that could be input into a neural network. To create our input dataset, we first needed to extract the data from the .nii files and map them to the correct label that each MRI had. While extracting the data, we learned how to subsample the data successfully in order not to run into computation issues when using an image that is too large. Through this dataset creation process, we learned how important it is to know the data that one is using. Without extensive knowledge of the data that is being used, the performance of the task that is being conducted might not be as great as it could be.

Another lesson we learned is about how powerful convolutional neural networks are at image classification. We were surprised at how effective and powerful these neural networks could be when trained for just a short amount of time. The potential for researchers to utilize these models is boundless, and neuroscience researchers will be able to save ample time with a very well trained convolutional neural network to detect if an MRI scan is personally identifiable or has brain feature loss.

Lastly, we learned how important the choice of hyperparameters and architecture of the model that is being constructed is. Throughout the iterations of our model, subtle changes in both the architecture and hyperparameters of our model greatly helped the accuracy and performance of our classification. By adding in dropout layers, we reduced the chance of overfitting our training data. In doing so, this made the model more unstable than it previously was, but added peace of mind about our overfitting concerns. After changing the architecture, we tweaked the hyperparameters of our model. By decreasing the learning rate to .0001 and increasing the batch size to 2, we were able to smooth out the curves of both the training and testing loss and accuracy. While the testing accuracy and loss curves were still spiking slightly, the range of spikes compared to our initial iteration was a great improvement. We certainly learned the importance of choosing appropriate architecture and hyperparameters.

Conclusion

If the scope of the project were longer, we would have liked to figure out the source of the testing accuracy and loss spiking very sporadically. Perhaps a large batch size would have helped, or even subsampling the MRI scans to a smaller size. Another approach would be to expand the architecture of the model to have more convolutional layers with larger amounts of filters. The size of the stride and kernel could be tweaked as well to see how it affects performance. On top of that, training with larger amounts of memory could help alleviate the problem of small batch size.

Overall, this project allowed us to have a glimpse of what data mining looks like in application. Throughout the entire semester we have been equipped with great tools to solve this problem. Our group has been able to apply many of the tools and techniques that we have learned over the course of this semester to the problem of helping researchers save time.