Team 11 Project Presentation

Ashton Sobeck, Sushant Kot, Zack McGeehan

The Problem

- Medical researchers go through a tedious task of manually checking if skull-stripped MRI images are identifiable in any way.
- We were tasked with creating a model that takes in a skull-stripped MRI scan and detect if it is
  - Personally identifiable
  - Missing brain information
- This model could help save time of researchers and help them get back to the main focus of their job.

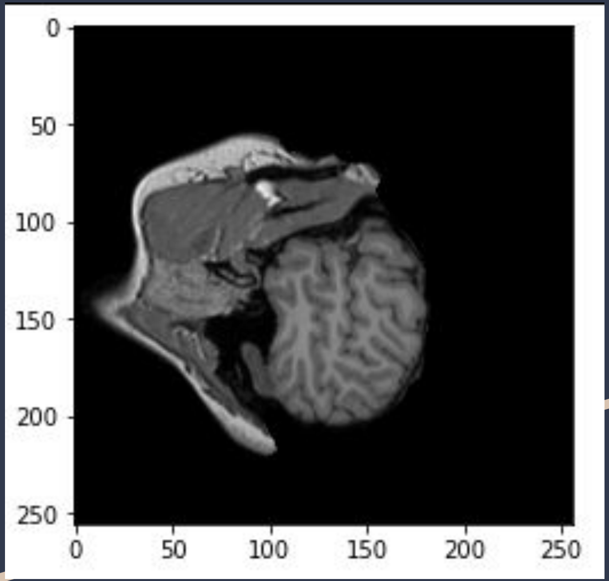
The Data

- Dr. Wang was very gracious in providing us with an extensive dataset of skull-stripped MRI scans.
- 1,311 scans.
- 3 classes:
  - Personally identifiable with no brain feature loss.
  - Not personally identifiable with brain feature loss.
  - Not personally identifiable with no brain feature loss.
- Each image within a scan was 256x256.
  - Variable depth with a maximum of 150.
- Separated into a labels file and a folder of .nii files.
  - .nii is an extension for MRI scans.

Programming

- We used the python library, keras, to construct our dataset and model.
- Keras allows for very intuitive construction of models and manipulation of data.
- The python library, nibabel, was used to retrieve the data from each .nii file
- We also used python's matplotlib library to plot charts and images that we accrued throughout the script.
- We stored our code in Github.
  - https://github.com/ashsobeck/CPSC8650GroupProjectTeam11

Data Preparation



- First, the dataset needed to be built for the model.
  - 85% of the scans were training data.
  - 15% of the scans were testing data.
- We needed to make sure that the file names from the label file corresponded to the data read in.
  - To do this, an array of images were constructed with another array with the corresponding class label.
- We used a class label of 0, 1, and 2.
  - 0 - Personally identifiable with no brain feature loss.
  - 1 - Not personally identifiable with brain feature loss.
  - 2 - Not personally identifiable with no brain feature loss.

Data Preparation (Cont.)

- At first, we attempted to use the full size of the image.
  - We ran into issues with tensorflow not being able to perform computations with 256x256x150 sized inputs.
  - We also had to omit data if we were to use the entire size because some scans had less than 150 depth.
- Our solution was to subsample the data down to a size of 150x150x75
  - This cuts the amount of data in half.
  - Also allows us to use all data because we can scale every size down or up to the desired size.

Our Model

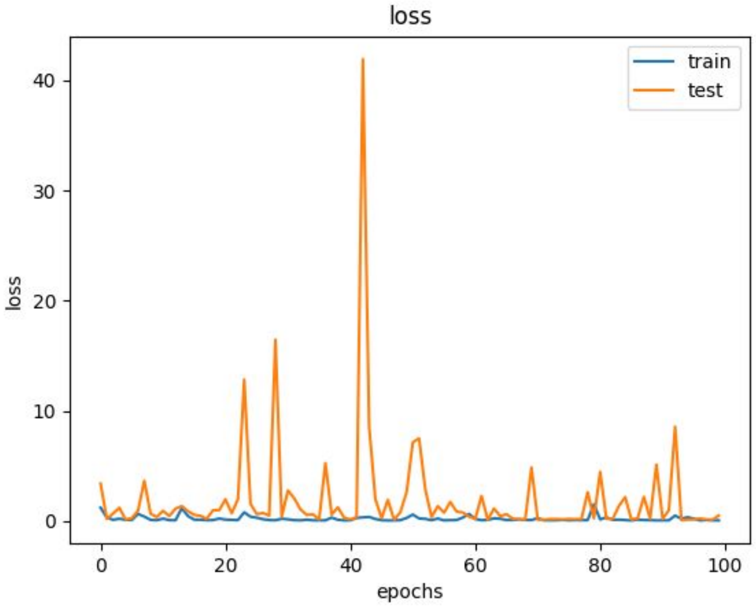
```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 75, 1)] 0
_________________________________________________________________
conv3d (Conv3D)              (None, 126, 126, 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
```

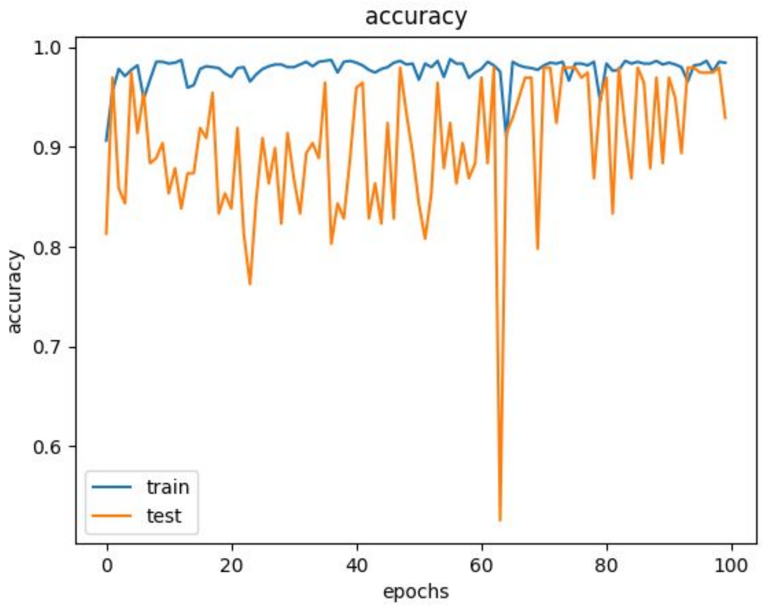
- We had 2 convolutional layers
  - First with 128 filters.
  - Then with 256 filters.
- After each convolutional layer, we had a pooling layer to reduce the dimension of the data.
- Once the input data had gone through the convolutional layers, it goes through a feed forward layer to output to classes.
- Outputs 3 numbers, each being the probability that the scan passed in is that class.

Training Parameters

- We trained for 100 epochs.
- Our model used the Adam optimizer.
  - .001 learning rate.
- For the loss function, we used Cross Entropy loss.
  - A very common loss for neural networks.
- We shuffled the data to make sure the network did not learn the order of the data and to add some randomness in.
- Our batch size was 1 due to data limits.
- We used Clemson’s Palmetto to train our model on.

Our Results


loss
train
test
40
30
20
10
0
loss
0
20
40
60
80
100
epochs


accuracy
1.0
0.9
0.8
0.7
0.6
accuracy
train
test
0
20
40
60
80
100
epochs

Our Results

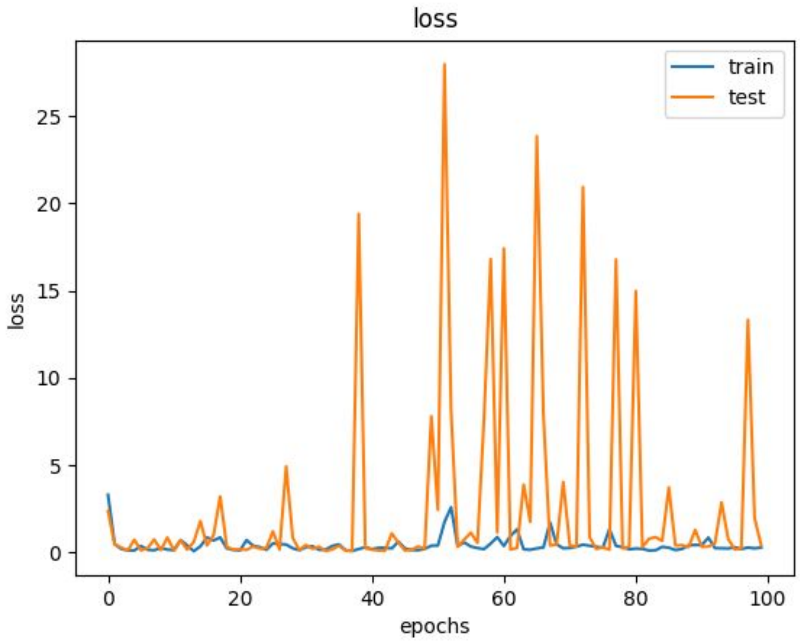
- Our testing accuracy was ~82% using keras' evaluate function with our testing dataset.
- There are large spikes in the testing loss and accuracy.
- Why?
  - We believe this to be the case because the model might have slightly overfit the training data.
  - Whenever the model predicts something wrong, it is trying to overcorrect with a large loss.

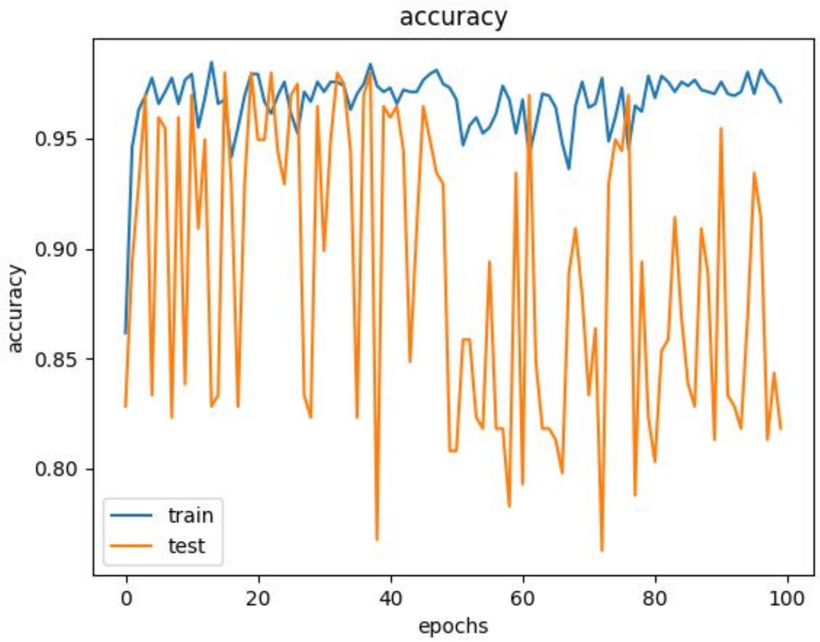
Potential Improvements

```
_________________________________________________________________
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
_________________________________________________________________
spatial_dropout3d (SpatialDr (None, 30, 30, 17, 256)   0
_________________________________________________________________
global_average_pooling3d (Gl (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 955,139
Trainable params: 955,139
Non-trainable params: 0
_________________________________________________________________
```

- There needs to be a sacrifice of training accuracy to help the model become better at recognizing new data.
- Adding in dropout layers to the model adds in some regularization in order to not overfit the training data.
  - This is the process of randomly dropping the value of nodes to ensure that the models do not overfit training data.
- Keep all of the same hyperparameters to see the difference in architecture.

Our Results Upon Adding Dropout

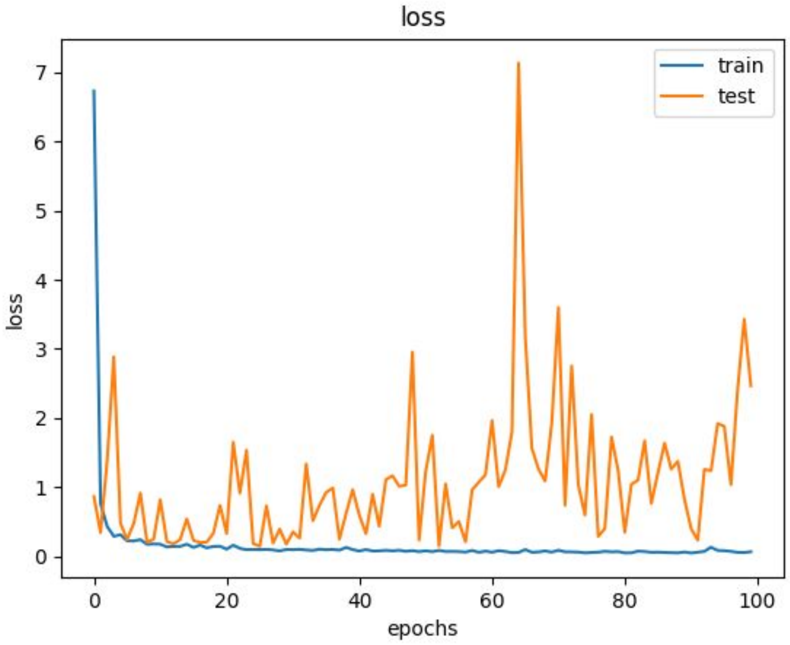


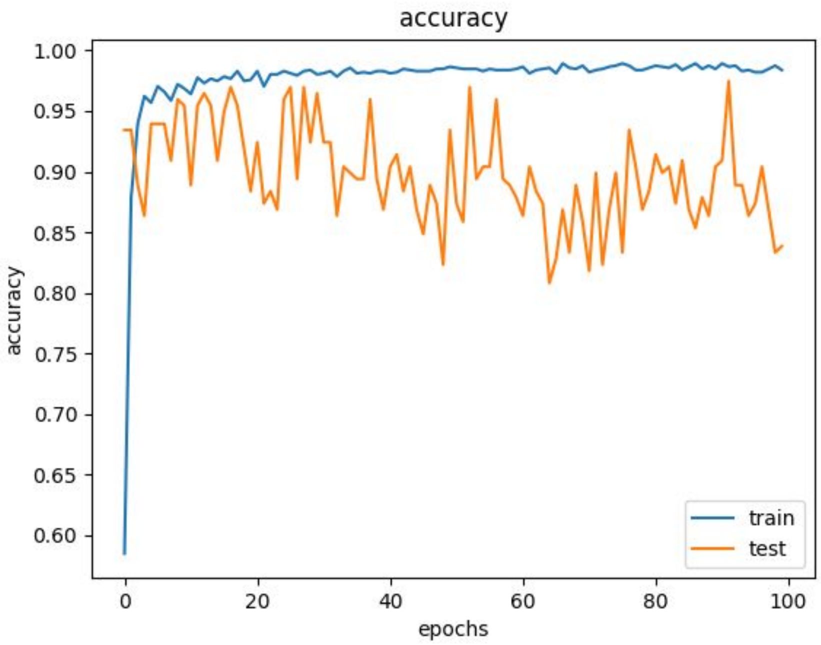


New Results

- Testing loss and accuracy more sporadic and jumpy than before.
- Dropout must have had an ill effect on our model.
- What could be the cause of spikes in loss and accuracy?
  - Large differences in the test and training data.
  - Would need to investigate further to truly determine.
  - Hard problem for the CNN to solve.
- We tried one more approach to stabilize the spikes from the test accuracy and loss.
  - Adjusting the hyperparameters.
  - Decrease learning rate from .001 to .0001 and increase batch size to 2.

Next Iteration Results





Next Iteration Results

- While still not very smooth, the curves of both training and testing loss and accuracy are much smoother than previous versions of the model.
- There are still spikes in the loss and accuracy of the testing, but not as wide of a range of values.
- We believe that the decrease in learning rate helped the model learn in a smooth manner which helped the large spikes of loss.
- This shows the importance of picking the correct parameters.
-

What We Learned

- Throughout this process, we learned many things.
- First, we learned how to create a dataset that a model would input.
  - We also learned how to subsample data and how that can help in data computation constraints.
- We also learned how powerful CNNs are as image classifiers.
  - They are a powerful tool that can help the medical researchers not waste time on a tedious task.
- The way that the hyperparameters and model architecture is chosen is very important when it comes to high accuracy in testing new data.

Questions?