Modeling T1 Resting-State MRI Variants Using Convolutional Neural Networks in Diagnosis of OCD

Project Logbook

Tarun Eswar
Massachusetts Academy of Math and Science
STEM Project
Instructor(s): Kevin Crowthers, Ph.D., Nicholas Medeiros

Table of Contents:

GLP Record Keeping Contract

I, Tarun Eswar, commit to record keeping in accordance with Good Laboratory Practices.

- My experiments and records will be reproducible, traceable, and reliable.
- I will NOT write my notes on scraps of paper, post-it notes, or other disposable items. My notes will go directly into my laboratory notebook.
- My data will be recorded in real-time. If I cannot record data in real-time, I will record raw data as soon as physically possible.
- I will record both qualitative and quantitative observations in my laboratory notebook and laboratory reports.
- My laboratory notebook will include information on the materials and instruments utilized during experimentation.
- I will initial and date over the edge of any material that is taped into my laboratory notebook.
- I will provide a real-time record of any analysis I perform.
- I will use blue or black pen to make entries in my laboratory notebook. I will NOT use pencil.
- I will define ALL abbreviations.
- If I make a mistake in my laboratory notebook, laboratory worksheets, or other written material, I will not obliterate or obscure the mistake. Instead, I will cross out the mistake using a single line. Any empty spaces in tables or partially used notebook pages will be crossed out using a single diagonal line.
- If I record information online (ex. In Google Drive), I will login so that my contributions are traceable.
- I will initial and date each page in my notebook and the front of each laboratory report.

Printed name: **Tarun Eswar**

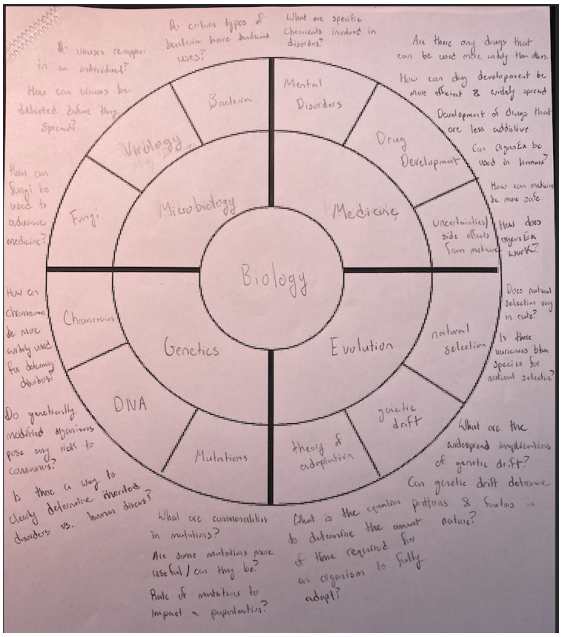
Signature 

Date: **9/02/2022**

Note: entries signed with last name as assumed to serve as a reference to the signature above.

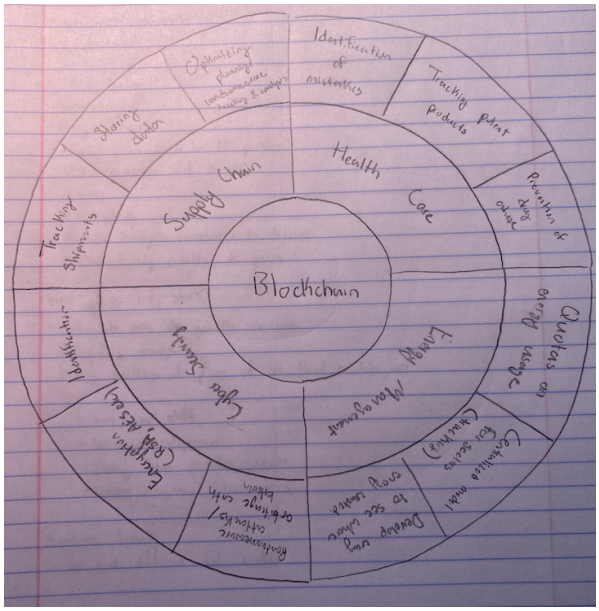
Brainstorming

Pie Diagrams



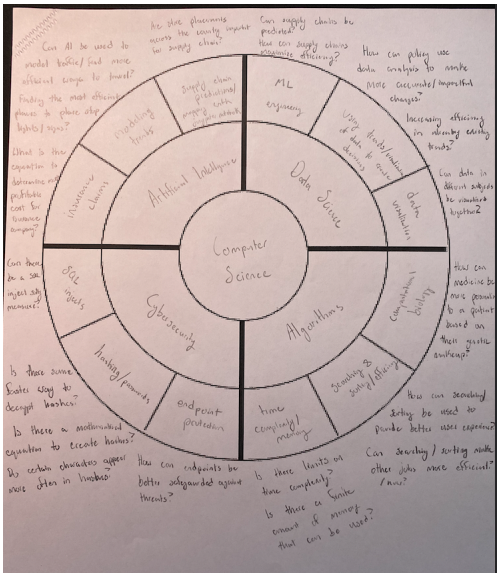
Eswar, 9/3/2022

I: Project idea development for obsessive-compulsive disorder



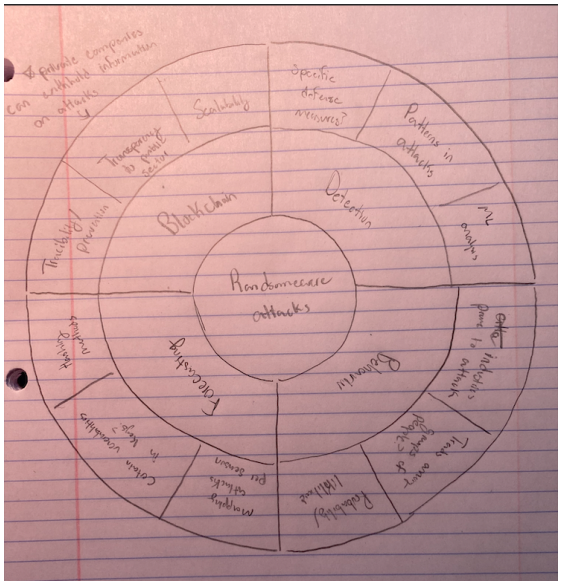
Eswar, 9/3/2022

II: Project idea development for blockchain in combination with supply chain or other sectors



Eswar, 9/3/2022

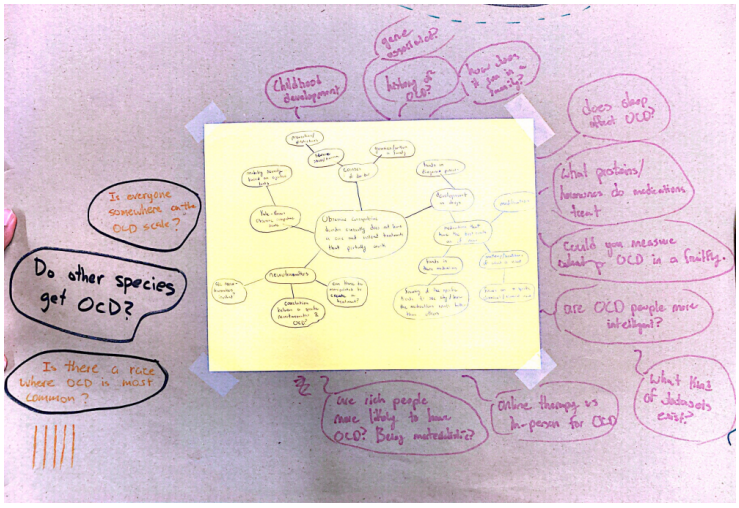
III: Project idea development for cybersecurity



Eswar, 9/3/2022

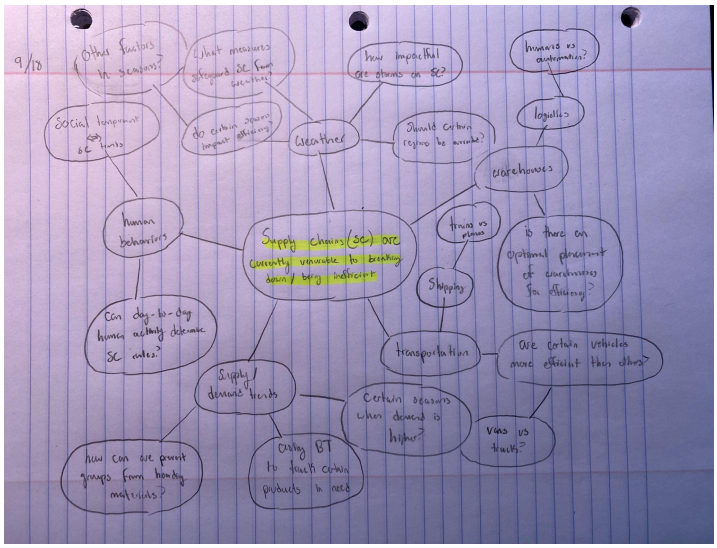
IV: Project idea development for trends in ransomware attacks

Mindmaps



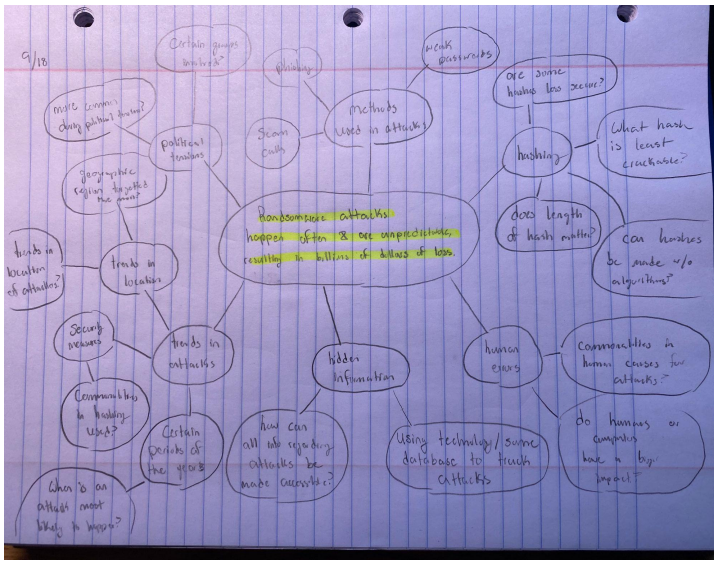
Eswar, 8/24/2022

Project idea development for OCD



Eswar, 9/28/2022

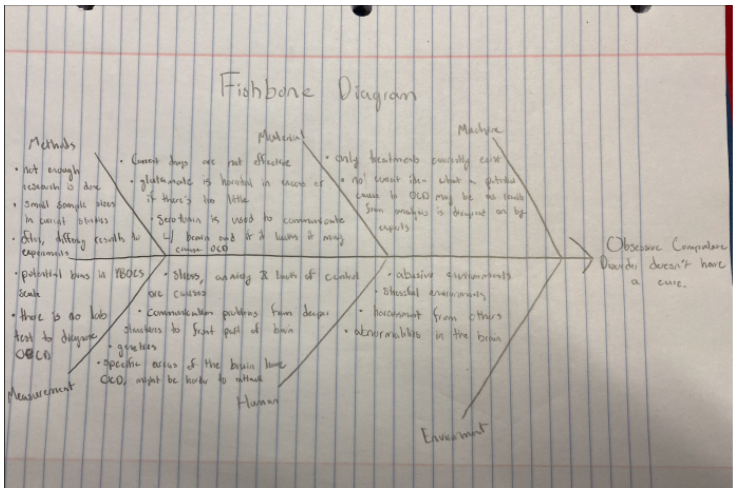
Project idea development for supply chain



Eswar, 9/28/2022

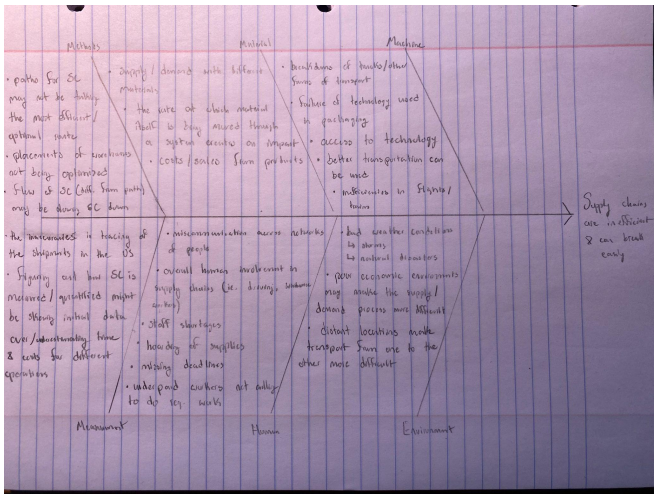
Project idea development for cyber security

Fishbone Diagrams



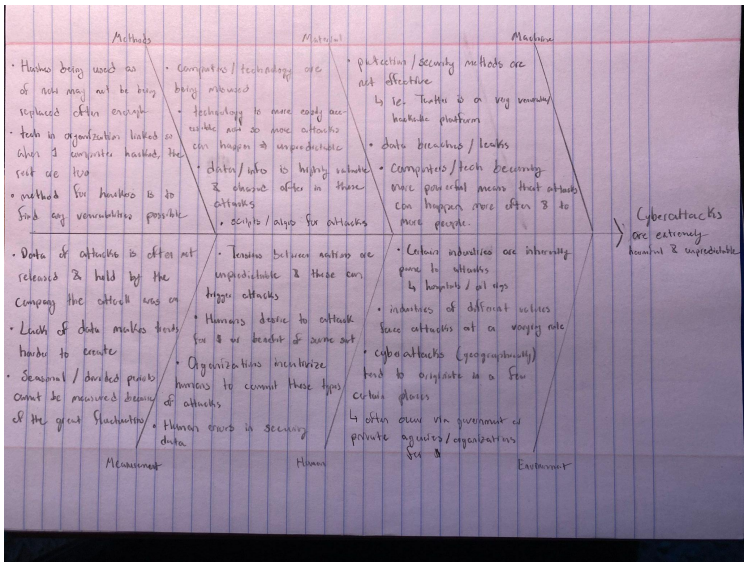
Eswar, 8/25/2022

Project idea development for OCD not having a cure



Eswar, 9/28/2022

Project idea development for supply chain



Eswar, 9/28/2022

Project idea development for cybersecurity

Five Whys

Eswar, 8/25/2022

Project idea development for OCD not having a cure

Subclause #1: there is no current lab test to diagnose OCD

Problem: Obsessive compulsive disorder doesn't have a cure

- Why? There is no current lab test to diagnose OCD
  - Why? There is no consensus among experts on the causes for OCD
    - Why? Differing lab results and experiments
      - Why? Differences in analysis and experiments
        - Why? Each study focuses on different aspects of causes

Potential Idea(s):

1. Looking at why there is no consensus on the cause of OCD
2. Which drug/idea had the greatest success and most consistency and why? Drug/idea might be a route to further look into

Subclause #2: serotonin is used to communicate with the brain and if it lacks it causes OCD

Problem: Obsessive compulsive disorder doesn't have a cure

- Why? Serotonin, when lacking, increases anxiety that fuels OCD
  - Why? Inherited disorders create this lacking amount
    - Why? Genes change due to DNA mutations
      - Why? Errors during cell division
        - Why? Issues regarding chromosomes

Potential Idea(s):

1. Is there a pattern between issues in chromes/cell division and OCD?
2. Can OCD be modeled on the basis of a family's history?

Eswar, 9/28/2022

Project idea development for supply chain

Subclause #1: distance and time required to travel between warehouses makes transport ineffective and difficult

Problem: Supply chains are inefficient and can break easily

↳ Why? Traveling for longer distances creates delays in the supply chain
↳ Why? Longer distances require more time to complete/travel
↳ Why? Transport within a supply chain can only move at a fixed rate, meaning that longer distances will increase the time needed.
↳ Why? Faster/more efficient methods of transit are not in place
↳ Why? Further funding would be required for new transport

Potential Idea(s):

1. Finding most optimal usage of planes, ships, and trucks to make transport of material as optimal as possible
2. Modeling when these different modes of transportation should/shouldn't be used

Eswar, 9/28/22

Project idea development for cybersecurity

Subclause #1: data of cyberattacks are often not tracible, leading to a lack of data and a difficulty in prediction/trend creation

Problem: Cyberattacks are harmful but are unpredictable

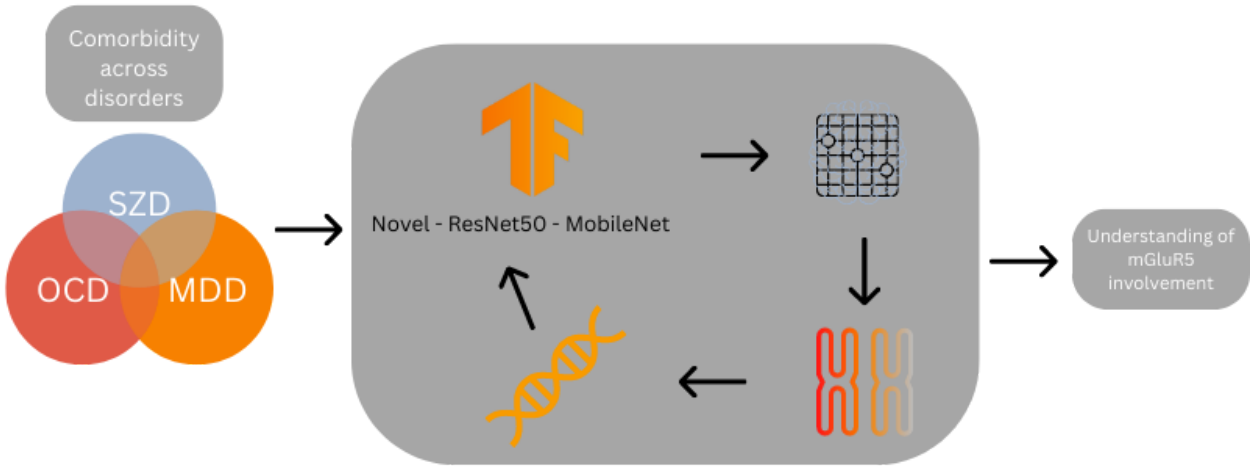
↳ Why? Cyberattacks are extremely sophisticated
↳ Why? Technology progressed over the past decade this point of sophistication
↳ Why? Increase in knowledge in the area of computers/technology has become more concentrated
↳ Why? More people are interested in exploring technology
↳ Why? Technology is relatively new and can be explored

Potential Idea(s):

1. Finding the trend between the increase in knowledge of technology with the developments of cyberattacks
   A. Using this to predict future attacks (i.e if one region is gaining more in knowledge of tech does this mean they are more prone to be the origin of a cyberattack?)
2. Modeling if security measures have progressed at a slower rate compared to cyberattack technology in the past decade

Systems Diagram

Eswar, 2/11/2023



Identification of the systemics diagram in process for this project. First, comorbid disorders are modeled via Tensorflow. Afterward, heatmaps were generated which were supposed to undergo gene expression analysis to understand the involvement of a glutamate receptor known as mGluR5 in OCD cases. This finding would potentially allow future research to narrow down the cause for OCD.

Decision Matrix

Eswar, created 12/17/2023; entered 2/15/2023

3 OPTIONS / 9 CRITERIA	MobileNet	ResNet50	Novel 2D CNN
Accuracy (8x)	7	8	7
Time Complexity (6x)	9	8	6
Cost (2x)	9	9	9
Usability (4x)	2	4	1
Available Support (5x)	4	8	1
Frequency Used within Field (5x)	3	8	9
Heatmap Accuracy (1x)	6	8	4
GradCam compatibility (3x)	4	5	5
Novel Heatmap compatibility (1x)	5	4	4
EDITABLE TOTAL SCORE	194.0	253.0	187.0

Abstract

Eswar, 1/9/23; edited 2/12/23

Obsessive-compulsive disorder (OCD) presents itself as a highly debilitating disorder. The disorder has common associations to the prefrontal cortex and the glutamate receptor known as Metabotropic Glutamate Receptor 5 (mGluR5). This receptor has been observed to demonstrate higher levels of signaling from positron emission tomography scans measured by its distribution volume ratios. Though, studies are unable to verify the involvement of mGluR5. Computational modeling methods were used as a means of validation for previous hypotheses involving mGluR5. The inadequacies in relation to the causal factor of OCD were answered by utilizing T1 resting-state magnetic resonance imaging (TRS-MRI) scans of patients suffering from schizophrenia, major depressive disorder, and obsessive-compulsive disorder. Because comorbid cases often occur within these disorders, cross comparative abilities become necessary to find distinctive characteristics. After unique structures of tissues found in OCD TRS-MRI scans were identified, a gene expression analysis was conducted based on scan data output. Two-dimensional convolutional neural networks alongside ResNet50 and MobileNet models were constructed and evaluated for efficiency. Activation heatmaps of TRS-MRI scans were outputted, allowing for transcriptomics analysis. Though, a lack of ability of prediction of OCD cases prevented gene expression analysis. Across all models, there was an 88.75% validation accuracy for MDD, and 82.08% validation accuracy for SZD under the framework of ResNet50 as well as novel computation. OCD yielded an accuracy rate of ~54.4%. These results provided further evidence for the p factor theorem regarding mental disorders. Future work involves the application of transfer learning to bolster accuracy rates.

Keywords: Obsessive-compulsive disorder, magnetic resonance imaging, major depressive disorder, schizophrenia

Project Introduction

Eswar, 11/22/22

First Draft:

Problem. In the field of neurological disorders, Obsessive-Compulsive Disorder (OCD), is an anxiety disorder that lacks proper treatment because of comorbidity with other disorders. Singular causal effects are difficult to determine as a result, leading to patients with incurable symptoms.

Objective. The engineering goal of this project is to develop three novel 2D convolutional neural network models for Obsessive-Compulsive Disorder and two disorders of similarity—depression and schizophrenia. From there, the activation heatmaps will help determine the regions of interest for areas of difference between the disorders.

Eswar, 2/12/22

Final Draft:

Problem Statement. Determining the root cause of obsessive-compulsive disorder is highly difficult. Physicians are often unable to differentiate obsessive-compulsive disorder from major depressive disorder and schizophrenia. As a result, the overall aim of this project was to design models for each disorder, develop activation heatmaps, and extract regions of interest. These models serve as a stepping stone in reaching significance of GRM in OCD patients. In order to supplement these models, gene expression analysis was conducted afterwards in order to determine the involvement of mGluR5, encoded by GRM, in OCD patients.

Engineering Objective. Diagnosis of OCD is currently understudied and misunderstood within the field of neuroscience. On basis of these observations, a few main objectives were enacted:

Obj. 1a: Construct individual CNNs with guidance from pre-trained networks for OCD, MDD, and Schizophrenia respectively with accuracy rates of at least 80%.

Obj. 1b: Develop activation heatmaps, demonstrating regions of interest unique to each disorder.

Obj. 1c: Perform gene expression analysis on T1 resting-state MRIs with transcriptomics.

Obj. 1d: Provide an online web application to allow patients to receive data from the model's details in Obj. 1a, as well as to feed more data into the models.

Research on obsessive-compulsive disorder in regard to root causes is still misunderstood because of privatization of datasets and knowledge. As a result, Obj. 1d provides a method to aid future research in being able to expand upon past knowledge in a more accessible and reasonable manner. Obj. 1d will also allow for patients to receive helpful metrics free of charge.

Key Terminology

Eswar, 11/22/22

Below, the key terms involved in my project are defined:

- T1 resting-state MRI scan - these scans are a specific type of anatomy scan that maps out the structural tissues of the brain at a resting state phase. Typically, these are collected when the patients are at rest.

- Glutamate
  - neurotransmitter abundant in the brain responsible for a plethora of functions
  - serves to enhance learning and memory, energy sources, and pain signaling

- Metabotropic
  - An exciter transmitter encoded by GRM5 gene
  - Recently found that intervention of mGluR5 led to anxiolytic effects—an area of interest in finding a target for medications (Terbeck, 2015)

- Selective Serotonin Reuptake Inhibitors - type of drug typically used in treating disorders from MDD to other related anxiety disorders including OCD

- Comorbidity - a condition that occurs when multiple disorders are present in a patient and thus prevent classification of the specific disorder present in the patient.

- ResNet50 - feature extraction network that bolsters accuracy rates when applied to pre-existing novel networks in order to generate more accurate weighted heatmaps

- Weighted layers - a variable used and improved over epochs by a machine learning algorithm in order to learn the probability of decision in the next iteration.

- Epoch - epochs define the number of iterations of the model after being run for a cingular cycle.

- 2D CNN - deep learning architecture that allows for certain patterns or objects to be extracted from images and be identified as regions or key interests.

- Activation Heatmap - a map ranging from RGB 0,0,0 to 255,255,255 that places importance on certain regions of the map to demonstrate the areas prioritized by the algorithm.

- P factor - a theory that states that all neurological disorders are connected in some form, leading to a lack of feature extraction in resting-state MRIs or contradictions in past studies.

Background

Eswar, 11/22/22

Introduction

Currently, only 40-60% of patients suffering from obsessive compulsive disorder (OCD) can be cured with serotonin reuptake inhibitors (SSRIS) or other forms of behavioral therapy while the remainder suffer from the lasting consequences (Kellner, 2010). Recent studies in the field struggle to reach a common consensus on a causal factor for the observance of OCD. Thus, this project serves to create a model that narrows down the causal factor by cross-comparing OCD with other disorders of similarity.

Research Outline:

Recent studies have found that glutamate dysfunction possesses a high likelihood of indicating OCD cases. Building onto the notion that glutamic dysfunction is a primary contributor to the clinical diagnosis of OCD, the metabotropic glutamate receptor 5 (mGlru5) has been found to have implications with OCD based on animal studies as well as spectroscopy scans (Akkus, 2014). Activation levels of the distribution volume ratio (DVR) of mGlru5 between healthy and diagnosed patients are expected to deviate and result in a neurological impact on the patient. Therefore, the DVR will be modeled against one's Yale-Brown Obsessive Compulsive (Y-BOCS) score. Y-BOCS relies on user input to determine a score, so connecting it with a neurological tracker provides validation for the process as well as a method to determine a potential area of study for determining a greater root cause of OCD. There is expected to be a clear association with higher activation values for mGlru5 and more extreme severity of OCD that also have significant deviation from the Y-BOCS score. Furthermore, a neural network model will be used to determine case accuracy of DVR for mGlru5 against Y-BOCS. The result is expected to yield a statistically significant difference in accuracy rates based on the assumption that Y-BOCS are biased based on user input.

Professional Communication:

Professional Communication #1

Entered 9/28/2022, occurred 9/11/2022

I spoke with a family friend from Tufts University—Varshini Ramanathan—who is a biomedical engineering major. During our conversation, I presented my ideas regarding obsessive compulsive disorder (OCD) and glutamate. One of her primary focuses this past year involved glutamate and she was able to give me useful advice, explaining that focusing on a glutamate receptor rather than the neurotransmitter would be more valuable to my project. The reason for this specification was that glutamate is universally used, meaning that an association with OCD would not mean anything. Additionally, she provided me with more information on glutamate. Furthering my project, she also stated that she would contact her colleagues in order to see if lab data for glutamate in regards to OCD would be available. Finally, she concluded the call by explaining that I would be able to contact her in the future for any questions and concerns regarding glutamate and my project.

Professional Communication #2

Entered 9/28/2022, written 9/14/2022

Dear Dr. Rentizelas,

My name is Tarun Eswar and I'm currently a junior at the Mass Academy of Math & Science which is associate with Worcester Polytechnic Institute. Presently, I'm involved in a five-month individual science fair project with my core idea being centered around supply chain. Recently, I came across one of your articles entitled "Logistics issues of biomass: The storage problem and the multi-biomass supply chain," and I had a few questions regarding this article.

Primarily, I wanted to understand why previous studies referred to in this article neglected the idea of cost analysis in storage solution models. The study on biomass in the article specifically focuses on storage systems, and I wanted to uncover if this aspect of storage facilities was previously ignored because it was insignificant in the overall flow of the system or if it is a true knowledge gap that requires further analysis. Additionally, I also was left wondering about the impact of individual storage facilities. In my current study, I'm looking to find methods to create a more efficient and fail proof supply change. Thus, I wanted to know at a more focused level if individual facilities or the general trends of warehouses throughout a region are more pertinent. Finally, I wanted to determine if there is an environmental aspect involved with supply chain. While researching biomass, were there any correlation between that specific chain and emission rates? This is a potential path I am considering when looking at overall efficiency, fueling a desire further investigate this potential correlation.

Thank you for your time.

Best regards,
Tarun Eswar

Professional Communication #3

Entered 11/22/2022, occurred 11/5/2022

I spoke with Revathi Ravi of Massachusetts General Hospital who attended and graduated from Harvard Medical School in the past 3 years. After providing context on my project, she was able to guide me in direction for finding data for mGlru5. For instance, I was pointed in the direction of a few different databases/sources that potentially may possess the datasets I am looking for. The main issue I was having at this point was finding gene expression data that could be analyzed; thus, we now have a series of weekly meetings set up to discuss new findings for data as well as project direction.

Professional Communication #4

Entered 11/22/2022, occurred 11/21/2022

I met with Professor Skorinko at WPI to learn more about the psychological aspect of OCD. Previously, because of data issues, I was considering changing project topics to look at Y-BOCS, a measure for diagnosing OCD cases. As a result, I spoke with Professor Skorinko in order to present my current ideas as well as receive feedback. Furthermore, I was able to ask questions about the computational aspects regarding OCD. She provided insight on how I might go about using surveys to achieve my project goal as well as potential branching ideas that may help me develop a stronger project.

Professional Communication #5

Entered 1/9/2023, occurred from 12/10/2022 to 1/3/2023

I spoke with a classmate in my grade—Joseph Yu— who was able to provide me with feedback on my approach. From this point, he was also able to teach me the basics of machine learning required for my project. Any support and aid provided by Joseph can be found in the form of comments in my codebase. This form of communication occurred over the span of several months, in the form of meeting at a location in Shrewsbury or meeting via Zoom.

Materials and Methods

Eswar, 12/12/22

- **Software**
  - Python 3.10.0
  - Tensorflow 2.10.0
  - SimpleITK
  - Pandas
  - Matplotlib
  - NumPy
  - MedPy
  - Skimage
  - Seaborn
  - ResNet50
  - MobileNet
  - SkLearn 1.2.0
  - imaging-transcriptomics 1.1.10

- **Hardware**
  - 2020 Apple Mac Mini M1
  - 2022 Apple Macbook Pro M2

- **MRI Data**
  - OpenNeuro
    - Bezmaternykh D.D., Melnikov M.Y., Savelov A.A. et al. Brain Networks Connectivity in Mild to Moderate Depression: Resting State fMRI Study with Implications to Nonpharmacological Treatment. Neural Plasticity, 2021. V. 2021. № 8846097. PP. 1-15. DOI: 10.1155/2021/8846097
    - Grega RepovÅ¡ and Deanna M. Barch (2018). Working memory in healthy and schizophrenic individuals. OpenNeuro. [Dataset] doi: null
    - Bilder, R and Poldrack, R and Cannon, T and London, E and Freimer, N and Congdon, E and Karlsgodt, K and Sabb, F (2020). UCLA Consortium for Neuropsychiatric Phenomics LA5c Study. OpenNeuro. [Dataset] doi: 10.18112/openneuro.ds000030.v1.0.0
  - Past Literature
    - Data from: Alterations of gray and white matter networks in patients with obsessive-compulsive disorder: a multimodal fusion analysis of structural MRI and DTI using mCCA+jICA (Kim, Seung-Goo et al., 2015)

Daily Entries

12/02/2022 - Processing Datasets

Eswar, 12/2/22

The purpose of today's work was to extract the .gii files from the OpenNeru dataset and unfold them into variables as desired. Each dataset was broken into slices and then appended to variables. The result of this work allows the start of preprocessing data for the 2D convolutional neural network. Furthermore, this setup is similar for each framework: novel, ResNet, and MobileNet. Each setup process will be demonstrated as below and reused later in the project.

Novel

```
X = []

for i in range(9):
    t1 = sitk.ReadImage("sub-0" + (str(i+1)) +"/anat/sub-0" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)[70:110]
    X.append(t2)

for i in range(9,72):
    t1 = sitk.ReadImage("sub-" + (str(i+1)) +"/anat/sub-" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)[70:110]
    X.append(t2)

print(X)
```

ResNet50

```
X = []

for i in range(9):
    t1 = sitk.ReadImage("sub-0" + (str(i+1)) +"/anat/sub-0" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)

for i in range(9,72):
    t1 = sitk.ReadImage("sub-" + (str(i+1)) +"/anat/sub-" + (str(i+1)) + "_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)


print(X)
```

MobileNet

```
X = []

for i in range(9):
    t1 = sitk.ReadImage("sub-0" + (str(i+1)) +"/anat/sub-0" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)

for i in range(9,72):
    t1 = sitk.ReadImage("sub-" + (str(i+1)) +"/anat/sub-" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)


print(X)
```

12/05/2022 - Novel 2D Depression Model

Eswar, 12/5/22

Introduction

Previously as demonstrated by Korolev et al. in 2017, convolutional neural networks (CNN) have the ability to be applied to MRI datasets. In their study, a 2D CNN was applied to Alzheimer's patients. However, with clinical depression, studies typically use a support vector machine, making the use of 2D CNN in the realm of depression MRI data novel.

Objective

The objective of this model is to classify patients as either possessing clinical depression or being healthy based on their MRI results. This model can later be used to develop activation heatmaps to pinpoint the exact regions of irregularity that the model determines.

Methods

In order to create a novel model, tensorflow and sklearn were utilized for organizing and testing the dataset. Prior to feeding the dataset through the model, classification markers were applied using 0's and 1's. The X dataset contains the images from OpenNuero. The Y list contains a matching set of 72 0's and 1's that correspond to healthy and clinical depression patients. This was completed as follows:

```
y = np.concatenate(( [1]*51, [0]*21 ))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
random_state=44)

X_train = np.resize(X_train, (2560, 288, 288))
X_test = np.resize(X_test, (320, 288, 288))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train))
```

With the above implementation, the data is ready to be fed through the 2D CNN. Prior to fitting the data, the model layers need to be defined. The model will be defined as Conv2D layers, pooling layers, dense,

and dropout layers in order to condense each epoch into the necessary format for the sigmoid activation function in the final step.

```
model = tf.keras.models.Sequential()
model.add(layers.Conv2D(64,(12,12),activation='relu',input_shape=(288, 288, 1)))
model.add(layers.Conv2D(64,(8,8),activation='relu'))
model.add(layers.MaxPooling2D((3,3)))
model.add(layers.Conv2D(32,(6,6),activation='relu'))
model.add(layers.Conv2D(32,(4,4),activation='relu'))
model.add(layers.MaxPooling2D((3,3)))
model.add(layers.Conv2D(16,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(2048,'relu'))
model.add(layers.Dense(1024,'relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(512,'relu'))
model.add(layers.Dense(256,'relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(64,'relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(8,'relu'))
model.add(layers.Dense(1,'sigmoid'))
model.summary()
```

The layers above represent the model being used to analyze the t1 resting state MRI scans. The pre-labeled data labels alongside the t1 MRI scans can now be inputted into the model as follows:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.binary_crossentropy,
              metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=25, validation_data=(X_test, y_test))
```

Furthermore, to depict the accuracy and results of this model, a confusion matrix as well as train/val graph can be constructed with the following python implementations. The output will be shown in the analysis section.

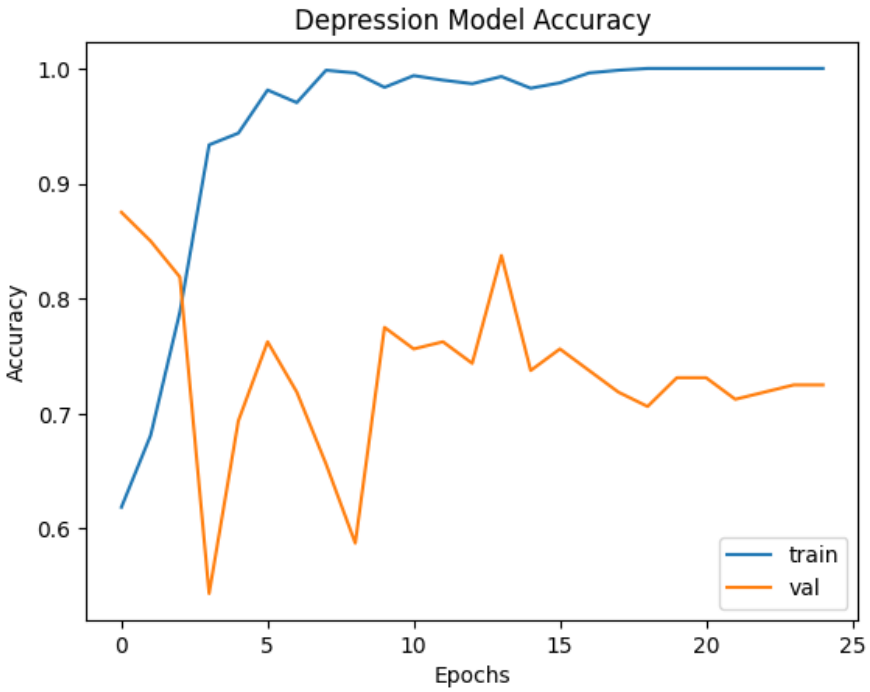
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Depression Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```

```
correct_labels = np.array(tf.concat([item for item in y_true], axis = 0))
predicted_labels = np.array(tf.concat([item for item in y_pred], axis = 0))

confusion_mtx = tf.math.confusion_matrix(correct_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.set(font_scale=2)
sns.heatmap(confusion_mtx,
            xticklabels=["Healthy", "Depression"],
            yticklabels=["Healthy", "Depression"],
            annot=True, fmt='g', annot_kws={"size":40})
plt.xlabel('Prediction')
plt.ylabel('Truth')
plt.show()

print(confusion_mtx)
```

Observations

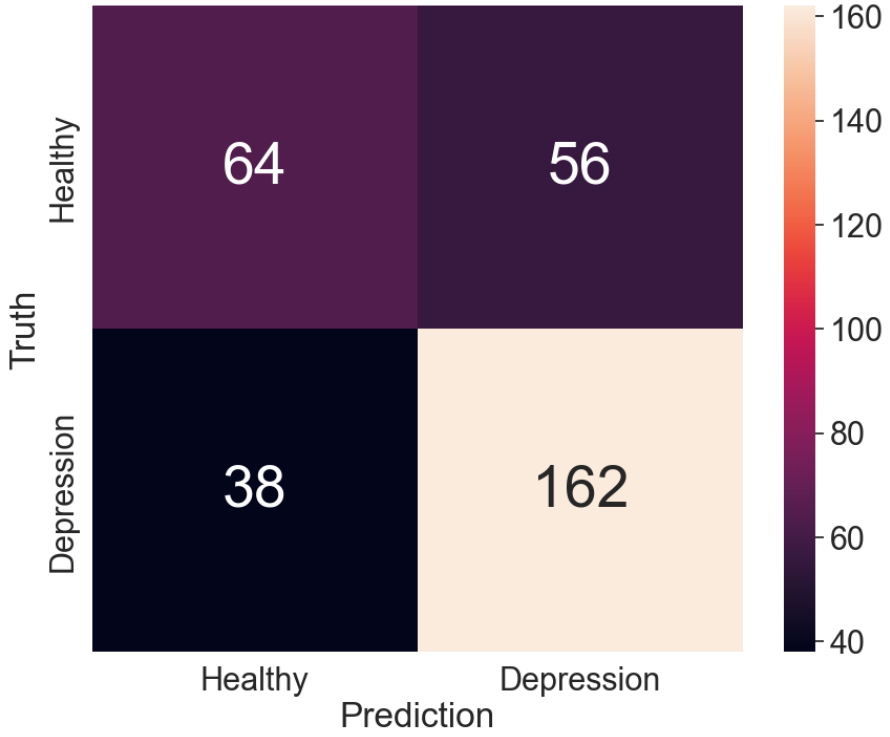
Through conducting this 2D CNN, I found that the entire dataset of patients provided cannot be used. Due to limitations of operating memory as well as inaccuracies when using all of the data, only the central slices of MRI were fed into the model with the X_train list. With the entire dataset, training accuracies of around 70% resulted whereas when only the central portion was used, the training accuracy spiked to 99% and the validation rates rose to 70%. Furthermore, I noticed that fMRI is not feasible with 2D CNNs because of size limitation and accuracy of the subsets for fMRI slices.

Analysis



As shown by the graph above, over the span of 25 epochs, the model produced a training accuracy of 0.9999 and a validation accuracy of 0.7255. Thus, at the training level, the results are highly accurate. However, if this model were applied to the general populous of unknown data, an accuracy of 0.7255 would be observed.

Furthermore, one source of limitation during the course of this work was the lack of testing data. The distribution provided by sklearn was roughly 75% training to 25% validation. With an increase in validation data tests, this accuracy could increase.



Additionally, a confusion matrix was generated to visualize the results of this model. Each tile either demonstrates a correct prediction or a false positive/false negative. For instance, the lower right tile would indicate that the model predicted 162 cases of depression and 162 of the patient dataset were truly depressive patients. However, the 38 in the bottom left tile would indicate that the model falsely deemed 38 depressive patients as healthy. With these metrics, the precision, recall, and f1 score can be calculated for the model:

Measure	Value
Sensitivity	0.6275
Specificity	0.7788
Precision	0.5818
Negative Predictive Value	0.8100
False Positive Rate	0.2212
False Discovery Rate	0.4182
False Negative Rate	0.3725
Accuracy	0.7290
F1 Score	0.6038
Matthews Correlation Coefficient	0.3990

(Confusion Matrix)

Based on evaluation the F1 score metric is the most important for analysis as it incorporates the precision and recall scores, providing a means for understanding the balances of false positives to false negatives in an unbalanced dataset. Given that the F1 score yields 0.60, it falls within the range of 0.5 to 0.8. Following the general rules of F1 scores, this value is decent at best.

However, based on the Matthews Correlation Coefficient of 0.3990, we can conclude that the model does in fact have strong positive correlation.

Concluding Remarks

Though the levels of accuracy seem decent, these metrics provide a proof of concept that novel models do in fact work. Levels of accuracy for both testing and validation would have been closer to 0.500 if the model was hypothetically randomly guessed. However, building onto this model for the remainder of work for the project is not feasible because these lower accuracy levels may tamper with activation heatmap reliability. Therefore, this model can be used as proof of concept for utilizing ResNet in the future. ResNet is composed of a pre-trained framework that can be used as a tool for t1 resting state MRI data. Because this model does appear to work, ResNet will bolster these observed accuracy levels and provide more relatable activation heatmaps in the future. Ultimately, these findings pave the way for the model involving OCD and schizophrenia. Both will most likely follow the path of utilizing a bare novel model for proof of concept and then converting to ResNet50 to bolster accuracy rates. Using ResNet will be the subject of future work for the depression model.

12/07/2022 - ResNet Depression Model

Eswar, 12/7/22

Introduction

In the previous daily entry, a novel neural network was constructed in order to demonstrate proof of concept for classifying between healthy and depressed fMRI scans from OpenNero. With this basis established, the successful proof of concept allows for the ability to expand observations to utilizing pre-trained models such as ResNet50. In this daily entry, the dataset will be trained based on the pretrained network of ResNet.

Objectives

The primary objective of the work is to create a secondary and more accurate model. Currently, with the novel neural network, constructing reliable activation heatmaps will be difficult due to the degree of variability present; however, once a model with higher levels of accuracy is built, accurate heat maps can be produced, allowing for cross-comparison in the future.

Eswar, 12/8/22

Methods

In order to create a novel model, the datasets were pre-filled based on the work completed on 12/2/2022. SkLearn was used once again in order to split the dataset into training and testing data. A random state of 44 was chosen for this model. From there, with the datasets loaded, the X_train and Y_train datasets were filled in order to conduct binary classification.

```
y = np.concatenate(( [1]*51, [0]*21 ))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=44)

# resize / filling
X_train = np.resize(X_train, (2560, 224, 224))
X_test = np.resize(X_test, (320, 224, 224))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train))
```

Afterwards, model creation using ResNet50 was completed. Based on the previous steps, the dataset yielded an input shape of 244, 244, 3. The weight used in this process was imagenet, a standard practice for image classification models. In order to use ResNet50, the preliminary epochs are conducted without any guidance from the pretrained network. The model was conducted in this fashion in order to better visualize the impact of ResNet on the model results.

```
base_model_resnet = tf.keras.applications.resnet50.ResNet50(input_shape=(224, 224, 3),
                                                 include_top=False,
                                                 weights='imagenet')
base_model_resnet.trainable = False
base_model_resnet.summary()
```

For verification of the model layers, a summary was produced, yielding the following.

```
Model: "resnet50"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================

 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv1_pad (ZeroPadding2D)      (None, 230, 230, 3)  0           ['input_1[0][0]']

 conv1_conv (Conv2D)            (None, 112, 112, 64  9472        ['conv1_pad[0][0]']
                                )

 conv1_bn (BatchNormalization)  (None, 112, 112, 64  256         ['conv1_conv[0][0]']
                                )

 conv1_relu (Activation)        (None, 112, 112, 64  0           ['conv1_bn[0][0]']
                                )

 pool1_pad (ZeroPadding2D)      (None, 114, 114, 64  0           ['conv1_relu[0][0]']
                                )
...
Total params: 23,587,712
Trainable params: 0
Non-trainable params: 23,587,712
__________________________________________________________________________________________________
```

As previously mentioned, a raw, non-ResNet model was first constructed, meaning the current trainable parameters under ResNet should be 0. With verification of the model completed, the model was executed.

```
# MODEL
history = model_resnet.fit(train_ds, y_train,
                    epochs=10,
                    validation_data=(test_ds, y_test))

# OUTPUT
Epoch 1/10
2022-12-17 22:10:50.848633: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
2022-12-17 22:10:52.555175: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin
optimizer for device_type GPU is enabled.
80/80 [==============================] - ETA: 0s - loss: 0.8897 - accuracy: 0.60352022-12-17 22:11:02.765608: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU
is enabled.
80/80 [==============================] - 13s 138ms/step - loss: 0.8897 - accuracy: 0.6035 - val_loss: 0.7077 -
val_accuracy: 0.6250
Epoch 2/10
80/80 [==============================] - 10s 129ms/step - loss: 0.5954 - accuracy: 0.6547 - val_loss: 0.6822 -
val_accuracy: 0.6250
Epoch 3/10
80/80 [==============================] - 10s 129ms/step - loss: 0.5803 - accuracy: 0.6613 - val_loss: 0.6295 -
val_accuracy: 0.6250
Epoch 4/10
80/80 [==============================] - 10s 129ms/step - loss: 0.5596 - accuracy: 0.6707 - val_loss: 0.7205 -
val_accuracy: 0.6250
Epoch 5/10
80/80 [==============================] - 11s 134ms/step - loss: 0.5419 - accuracy: 0.6855 - val_loss: 0.8453 -
val_accuracy: 0.6250
Epoch 6/10
80/80 [==============================] - 10s 129ms/step - loss: 0.5461 - accuracy: 0.6949 - val_loss: 0.6423 -
val_accuracy: 0.6250
Epoch 7/10
80/80 [==============================] - 10s 120ms/step - loss: 0.5212 - accuracy: 0.7047 - val_loss: 0.6416 -
val_accuracy: 0.6250
Epoch 8/10
80/80 [==============================] - 11s 131ms/step - loss: 0.5644 - accuracy: 0.6801 - val_loss: 0.4169 -
val_accuracy: 0.8094
Epoch 9/10
80/80 [==============================] - 11s 134ms/step - loss: 0.5650 - accuracy: 0.6910 - val_loss: 0.4471 -
val_accuracy: 0.8438
Epoch 10/10
80/80 [==============================] - 11s 132ms/step - loss: 0.5089 - accuracy: 0.7125 - val_loss: 0.7119 -
val_accuracy: 0.6250
```

As seen in the outputs, the raw accuracy without ResNet aid results in an accuracy of around 71.25%. From here, the training for ResNet50's fine tuning was turned on from epoch 10.

```
base_model_resnet.trainable = True

# Fine-tune from this layer onwards
fine_tune_at = 75

# Freeze all the layers prior to the `fine_tune_at` layer
for layer in base_model_resnet.layers[:fine_tune_at]:
  layer.trainable = False

# Model compilation
model_resnet.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

```
fine_tune_epochs = 10
total_epochs =  20 + fine_tune_epochs

history_fine = model_resnet.fit(train_ds, y_train,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=(test_ds, y_test), callbacks=[
                         tf.keras.callbacks.EarlyStopping(
                         monitor='val_loss',
                         patience=5,
                         mode='auto',
                         restore_best_weights=True,
)])

...

Epoch 13/30
80/80 [==============================] - 24s 296ms/step - loss: 0.4371 - accuracy: 0.7750 - val_loss: 0.4861 -
val_accuracy: 0.8000
Epoch 14/30
80/80 [==============================] - 26s 320ms/step - loss: 0.3746 - accuracy: 0.8105 - val_loss: 0.2936 -
val_accuracy: 0.8781
Epoch 15/30
80/80 [==============================] - 25s 306ms/step - loss: 0.3355 - accuracy: 0.8391 - val_loss: 1.4692 -
val_accuracy: 0.6469
Epoch 16/30
80/80 [==============================] - 24s 296ms/step - loss: 0.2631 - accuracy: 0.8922 - val_loss: 0.3263 -
val_accuracy: 0.8500
Epoch 17/30
80/80 [==============================] - 22s 277ms/step - loss: 0.2297 - accuracy: 0.8980 - val_loss: 0.4008 -
val_accuracy: 0.8313
Epoch 18/30
80/80 [==============================] - 22s 271ms/step - loss: 0.1741 - accuracy: 0.9301 - val_loss: 0.3502 -
val_accuracy: 0.8250
Epoch 19/30
80/80 [==============================] - 21s 262ms/step - loss: 0.1602 - accuracy: 0.9457 - val_loss: 0.4447 -
val_accuracy: 0.8031
```

With the model completed, visualization of the work is needed. The following lines utilize MatLab's python library in order to provide a graphical representation of the ResNet model.

```
# PRIOR TO RESNET

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
sns.set(font_scale=1)

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')
```

```
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()


# POST RESNET GRAPH
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']

initial_epochs = 10

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.5, 1])
plt.plot([initial_epochs,initial_epochs],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs,initial_epochs],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

# SAVING
model_resnet.save('depression_resting_state_dataset_t1_2d_resnet.h5')

# RESTORATION FOR FUTURE USE
new_model = tf.keras.models.load_model('depression_resting_state_dataset_t1_2d_resnet.h5')
loss, acc = new_model.evaluate(test_ds, y_test, verbose=2)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc))
```

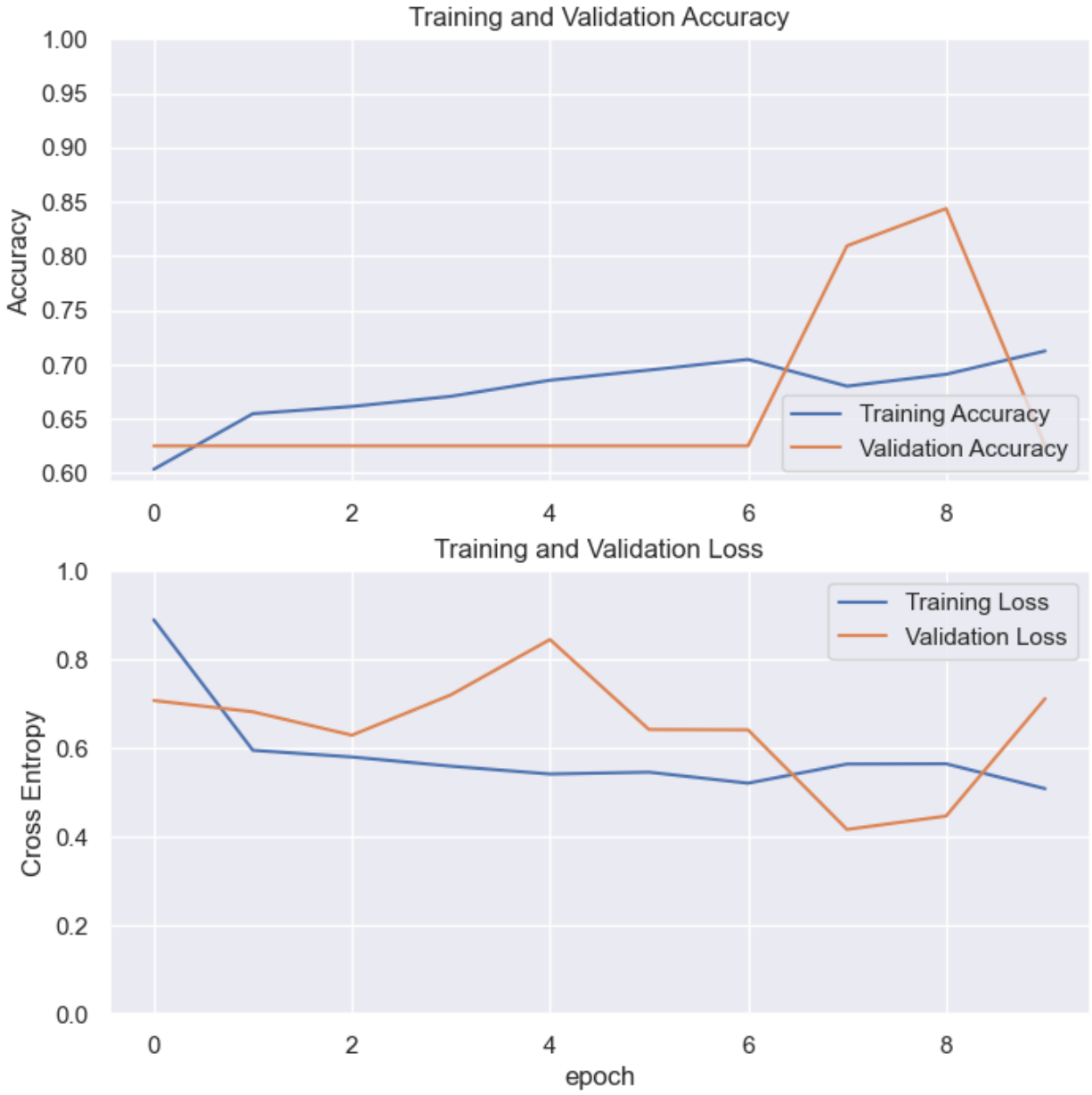
Eswar, 12/10/22

Observations

With the use of ResNet50, the accuracy rates of the model reached upwards of a high 90% range for testing and in the 80% range for training. These values are far more accurate as compared to the novel model, potentially suggesting that although differences exist, the support provided by ResNet is required in order to establish a model with clearer boundaries of healthy versus depressed.

Analysis

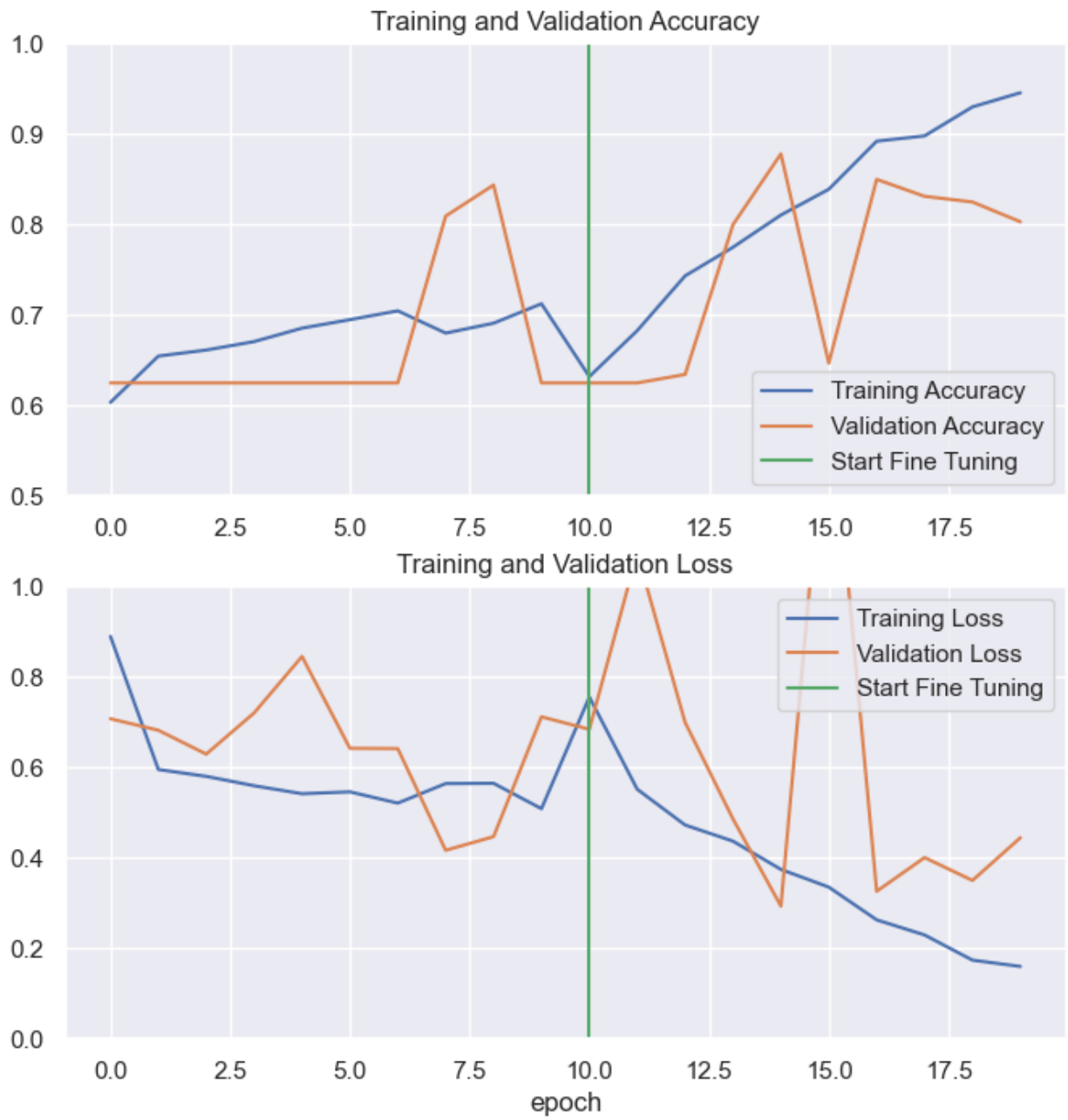
Prior to ResNet50



Model Accuracy. As demonstrated by the graphs above, the accuracy rates for depression fMRI image classification prior to using ResNet50 was fairly low. The training accuracy tended to remain between the 60% to 70% range and concluded with an accuracy of around 0.72. Furthermore, the validation accuracy graphs provided unusual results by reaching an accuracy of nearly 0.85 but then dropping back to 0.625.

Cross Entropy. The cross entropy graph provides more evidence for a rather inaccurate *training=false* raw model based on the rates continuously remaining above the 0.5 mark without ever leveling below even 0.25.

ResNet50 Results



Model Accuracy. ResNet's state of the art framework provided far more reliable results for classification. As stated during methodology, fine tuning began from epoch 10.0. In the graphs above, the fine tuning marker is depicted in green. After this stage began, the model began to drastically see more accurate results. The accuracy concluded at a level of 0.9457. 1.3 times more accurate as the pre-model work. The validation results demonstrate slight volatility but stabilize at around 80% accuracy to conclude.

Cross Entropy. The cross entropy graph provides strong evidence for an accurate model. The loss rates fall into the lowest quartiles nearing the end of testing. With more epochs, the model would observe a training loss of near, if not exactly 0.00.

Concluding Remarks

Based on the increased accuracy provided by ResNet50, this methodology presents itself as a promising avenue when analyzing obsessive-compulsive disorder as well as schizophrenia. Both disorders can utilize the similar method of preliminary proof of concept of a novel network with future implementations using ResNet. However, before confirming this proposed pathway, MobileNet must also be tested. After attempting to implement MobileNet, the three models can be compared for efficiency. As stated by Wang et al. 2019, MobileNet presents itself as an advantage given its need for fewer parameters and higher accuracy yield (Wang et al. 2019). Thus, on this basis, MobileNet will be implemented in the next entry. From there, a criteria matrix will be constructed to determine a consistent pathway for future works involving disorders of interest.

12/09/2022 - MobileNet Depression Model

Eswar, 12/9/22

Introduction

As demonstrated by Wang et al. 2019, MobileNet presents a viable method to increase accuracy rates while reducing parameters required for the model. These increased rates of accuracy have the potential for being an avenue for the final model. Thus, this entry seeks to implement MobileNet.

Eswar, 12/9/22

Objectives

The primary objective of this entry is to provide a third model that may have better accuracy rates. Though an objective for this entry, with two functioning models, this work serves as an extra avenue in order to bolster accuracy rates. If MobileNet provides sufficient accuracy as well as improvements in efficiency, it may be considered as the primary model as compared to novel or ResNet.

Eswar, 12/9/22

Methods

Similar to ResNet, the models were preloaded from the entry completed on 12/2/2022. The process to MobileNet implementation shares similarities with novel and ResNet. As demonstrated below, SkLearn splits the dataset in a similar fashion with random state of 44 and test size of 0.1 for memory purposes. Furthermore, the X_train, X_test, Y_train, and Y_test lists were all initialized within this step.

```
y = np.concatenate(( [1]*51, [0]*21 ))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=44)

X_train = np.resize(X_train, (2560, 224, 224))
X_test = np.resize(X_test, (320, 224, 224))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train))
```

A similar process for visualization of MobileNet was implemented in comparison to ResNet. Before turning on *fine_tuning* the model demonstrates the raw potential by turning the trainable parameter to false.

```
base_model_mobilenet = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),
                                                          include_top=False,
                                                          weights='imagenet')
base_model_mobilenet.trainable = False
base_model_mobilenet.summary()

...

Model: "mobilenetv2_1.00_224"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_3 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 112, 112, 32  864         ['input_3[0][0]']
                                )

 bn_Conv1 (BatchNormalization)  (None, 112, 112, 32  128         ['Conv1[0][0]']
                                )

 Conv1_relu (ReLU)              (None, 112, 112, 32  0           ['bn_Conv1[0][0]']
                                )

 expanded_conv_depthwise (Depth  (None, 112, 112, 32  288        ['Conv1_relu[0][0]']
 wiseConv2D)                    )

 expanded_conv_depthwise_BN (Ba  (None, 112, 112, 32  128        ['expanded_conv_depthwise[0][0]']
 tchNormalization)              )

 expanded_conv_depthwise_relu (  (None, 112, 112, 32  0          ['expanded_conv_depthwise_BN[0][0
 ReLU)                          )                                ]']

...
Total params: 2,257,984
Trainable params: 0
Non-trainable params: 2,257,984
```

From this outlook, all aspects of the model seem prepared to handle data input. The above layers depict the structure used in construction of the MobileNet model, though the remaining lines are cut out due to the amount of layers utilized by the framework. However, the above demonstrates the setup for the usage of the MobileNet Framework. With the pre-trained network setup, the raw comparison model can be built within TensorFlow using the keras API. This raw comparison makes use of the inputs and outputs as computed by the X/Y train/test from the initial setup. Following this computation, the model was fitted with Adam as the optimizer.

```
inputs = tf.keras.Input(shape=(224, 224, 3))
x = preprocess_input(inputs)
x = base_model_mobilenet(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = layers.Dense(1)(x)
model_mobilenet = tf.keras.Model(inputs, outputs)


base_learning_rate = 0.0001
model_mobilenet.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])

model_mobilenet.summary()


history = model_mobilenet.fit(train_ds, y_train,
                    epochs=10,
                    validation_data=(test_ds, y_test))

# OUTPUT

Epoch 6/10
80/80 [==============================] - 5s 57ms/step - loss: 0.5782 - accuracy: 0.6883 - val_loss: 0.6775 -
val_accuracy: 0.6281
Epoch 7/10
80/80 [==============================] - 4s 54ms/step - loss: 0.5808 - accuracy: 0.6891 - val_loss: 0.6488 -
val_accuracy: 0.6406
Epoch 8/10
80/80 [==============================] - 4s 54ms/step - loss: 0.5736 - accuracy: 0.6953 - val_loss: 0.6484 -
val_accuracy: 0.6375
Epoch 9/10
80/80 [==============================] - 4s 55ms/step - loss: 0.5736 - accuracy: 0.6980 - val_loss: 0.6545 -
val_accuracy: 0.6438
Epoch 10/10
80/80 [==============================] - 5s 59ms/step - loss: 0.5627 - accuracy: 0.6996 - val_loss: 0.6134 -
val_accuracy: 0.6813
```

From this point, MobileNet is ready to be implemented. MobileNet was fine tuned from layer 50 at epoch 10, a similar process as ResNet. Though for ResNet, the RMSprop optimizer was used instead.

```
# Setting training to true
base_model_mobilenet.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model_mobilenet.layers))

# Fine-tune from this layer onwards
fine_tune_at = 50

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model_mobilenet.layers[:fine_tune_at]:
  layer.trainable = False

# Model Compilation
model_mobilenet.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

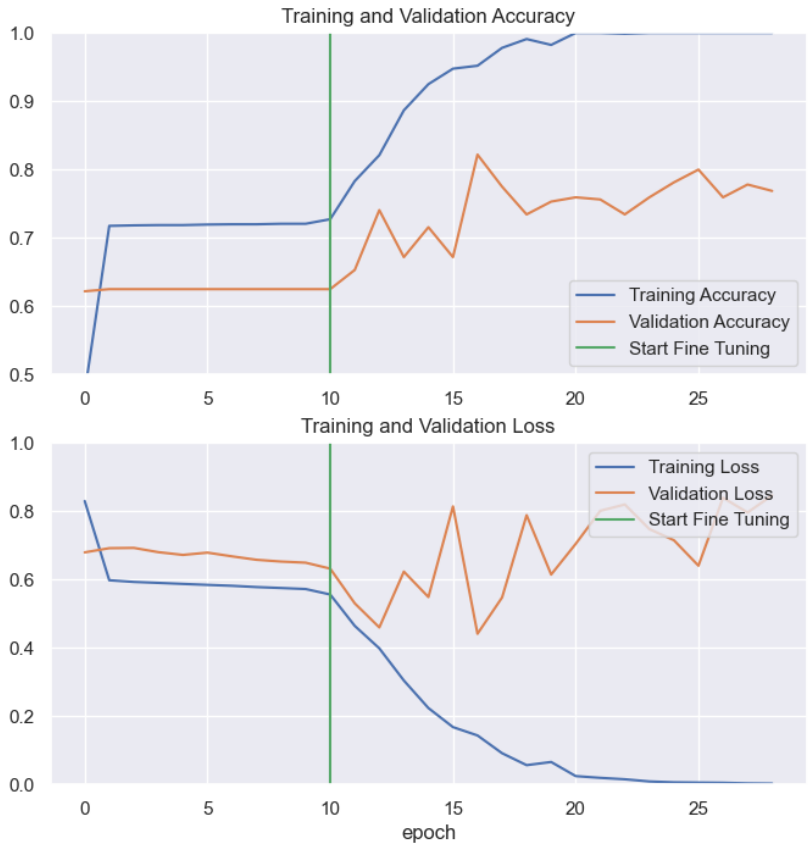
```
# Fine Tuning MobileNet at epoch 10
fine_tune_epochs = 10
total_epochs =  20 + fine_tune_epochs

history_fine = model_mobilenet.fit(train_ds, y_train,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=(test_ds, y_test), callbacks=[
                         tf.keras.callbacks.EarlyStopping(
                         monitor='val_loss',
                         patience=5,
                         mode='auto',
                         restore_best_weights=True,
)])
```

Furthermore, in order to graphically represent these results, the MatLab python API was utilized as follows. Accuracy graphs as well as cross entropy rates were yielded as a result.

```
# Prior to MobileNet Activation

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
sns.set(font_scale=1)

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

```
# Plotting with epoch 10 marker

acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']

initial_epochs = 10

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.5, 1])
plt.plot([initial_epochs,initial_epochs],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs,initial_epochs],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

# Saving the model

model_mobilenet.save('depression_resting_state_dataset_t1_2d_mobilenet.h5')

# Loading the model for future use

new_model = tf.keras.models.load_model('depression_resting_state_dataset_t1_2d_mobilenet.h5')
loss, acc = new_model.evaluate(test_ds, y_test, verbose=2)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc))
```

Analysis



Based on the resulting accuracy graphs provided by MobileNet, the model functions at an exceptional level:

Accuracy: Based on the graphs above, the model finished with a top accuracy of approximately ~82%. Though the model does not conclude with this level of accuracy, the configuration of the model enables it to take the performance at the point with the highest accuracy. As a result, the state of the model at epoch 16 is utilized.

Cross Entropy: The cross entropy graphs appear to be nearly normal with the reduction of training loss at a parabolic rate after epoch 10, signaling that the pretrained framework began to take effect. Unfortunately, more variance was seen with the validation loss, leading to an overall accuracy that may be able to be improved in the future. However, the pre-trained network does account for some of the overfitting patterns observed prior to epoch 10.

12/20/2022 - MDD Heatmap Creation

Eswar, 12/20/22

Introduction

With the models successfully computed, heatmaps were outputted in order to identify regions of interest in terms of analyzing MDD. In order to do so, confusion matrices can be utilized to illustrate the implicated regions of the resting-state MRI scans.

Eswar, 12/20/22

Objectives

The objective of this entry is to develop novel heatmaps for MDD and then to identify the key regions of the MRI scan being used by the ResNet50 model to classify patients. Furthermore, because several models were developed earlier on, heatmaps will be constructed to understand the variance of accuracy and consistency across each, including ResNet50, novel, and MobileNet. Understanding the generation of the heatmaps will provide a more consistent method for moving forward. From there, these regions can be further analyzed and cross-compared later on.

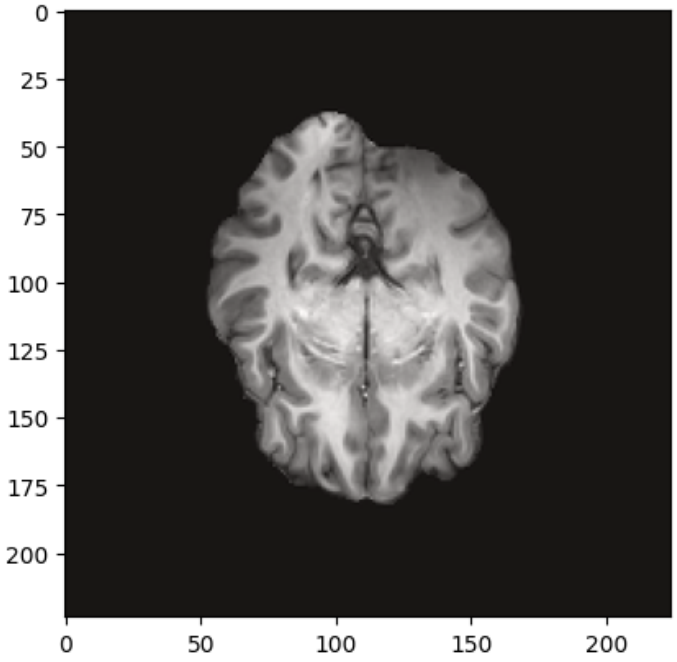
Eswar, 12/20/22

Methods

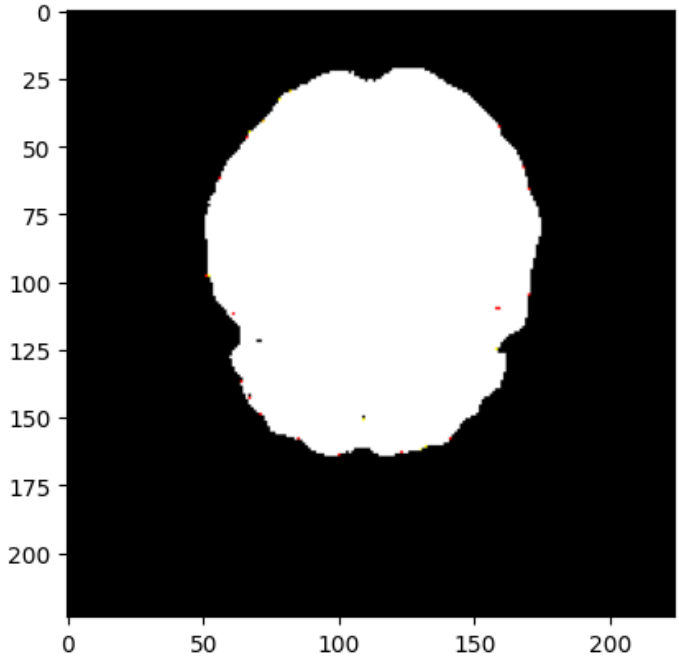
In order to analyze the existing model for MDD, the slices needed to be reloaded in. This process remains the same as the load in process done when constructing the models.

```
X = []

for i in range(9):
    t1 = sitk.ReadImage("../Data/sub-0" + (str(i+1)) +"/anat/sub-0" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)[70:110]
    X.append(t2)

for i in range(9,72):
    t1 = sitk.ReadImage("../Data/sub-" + (str(i+1)) +"/anat/sub-" + (str(i+1)) + "_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)[70:110]
    X.append(t2)

print(X)
}

X_train = np.resize(X_train, (2560, 288, 288))
X_test = np.resize(X_test, (320, 288, 288))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test
```

Once loaded in, the images were renormalized in order to gain features. Without the feature of the MRI, prediction may be inaccurate. To demonstrate the impact of normalization, the following slices demonstrate the difference.



Normalized T1 resting-state



Non-normalized T1 resting-state

The following function was utilized in order to normalize array slices. When they are not normalized, the values range from -255 to 255 for the stacked RBG. However, in order to see the features again, the range must be condensed into -1 to 1 as shown in the figure to the left. The function below demonstrates the computation to re-standardize the array.

```
def normalize(arr):
    arr = arr + abs(np.amin(arr))
    assert np.amax(arr) != 0
    arr = arr / np.amax(arr)
    return arr

X_train = normalize(X_train)
X_test = normalize(X_test)
```

After the array was normalized, another supplementary function was built in order to output the weights for an individual map. In order for this function to output the desired CAM, the last layer's weights are taken. From there, the weights are treated in accordance with the final conv2d layer which is then overlaid onto the original MRI with jetMap to show the regions of interest.

```
def get_class_activation_map(img):

    img = np.expand_dims(img, axis=0).astype(np.float32)

    predictions = model.predict(img)
    print(predictions)
    label_index = int(np.round(predictions[0][0]))
    print(label_index)

    class_weights = model.layers[-1].get_weights()[0]
    print(class_weights)
    class_weights_winner = class_weights
    final_conv_layer = model.get_layer("conv2d_4")


    get_output = K.function([model.layers[0].input],[final_conv_layer.output,
model.layers[-1].output])
[conv_outputs, predictions] = get_output([img])


    # squeeze conv map to shape image to size (7, 7, 2048)
    conv_outputs = np.squeeze(conv_outputs)
    print(np.shape(conv_outputs))

    # bilinear upsampling to resize each filtered image to size of original image
    mat_for_mult = scipy.ndimage.zoom(conv_outputs, (32, 32, 1), order=1)
    final_output = np.dot(mat_for_mult.reshape((288*288, 16)), class_weights_winner).reshape(288,288)

    return final_output, label_index
```

After confirming that the function worked as intended, another supplementary function was made in order to iterate through all of the images and output their respective heatmaps. All images are plotted on the same graph using plt.subplots. Additionally, alpha levels of 0.5 and 0.7 for the heatmaps and MRI respectively were selected for viewability purposes.

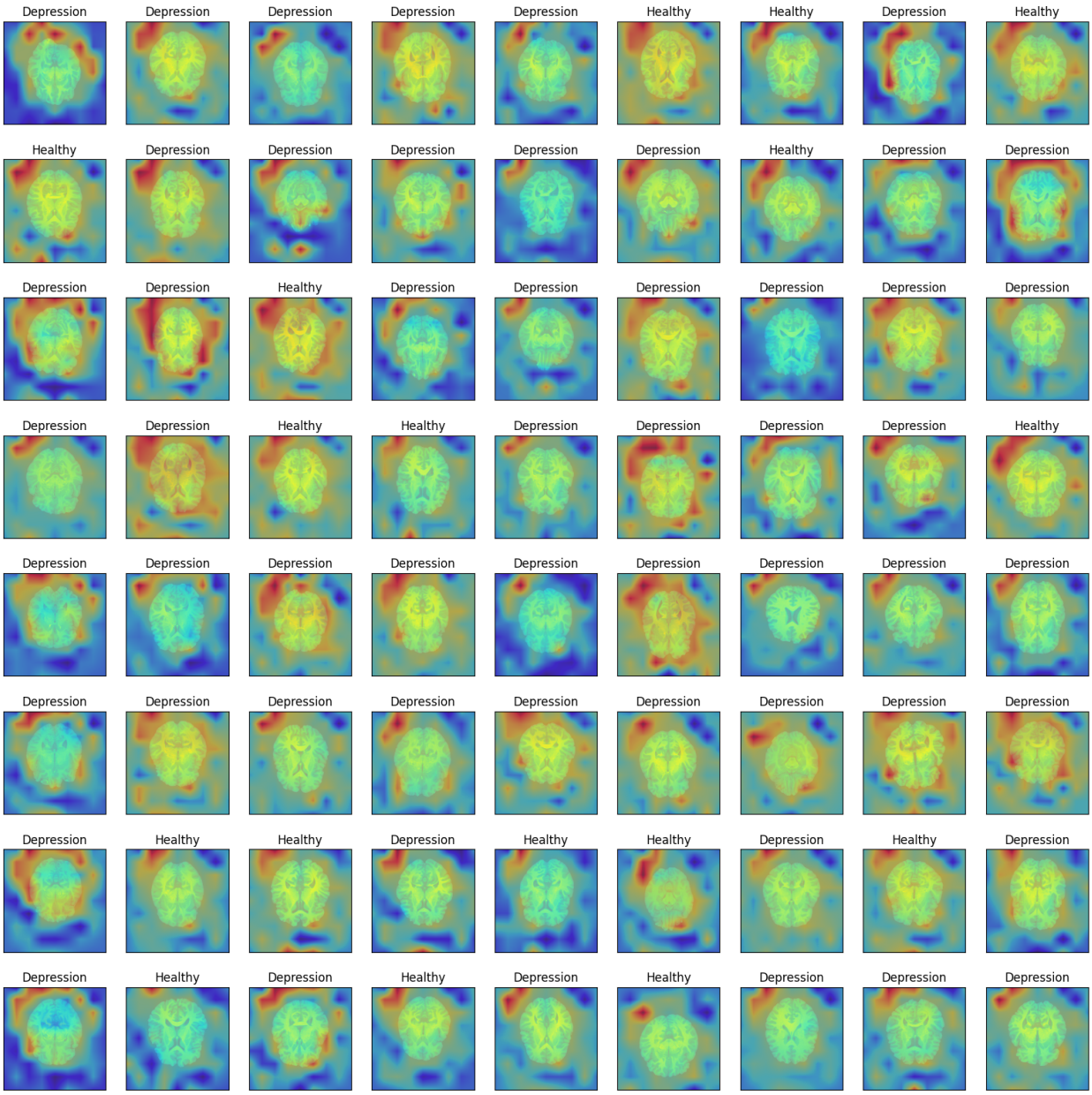
```
def plot_class_activation_map(images):

    figure, axis = plt.subplots(8, 9)

    figure.set_size_inches(20, 20)

    current_y = 0
    current_x = 0

    plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])

    for img in images:

        img = tf.squeeze(img)

        CAM, label = get_class_activation_map(img)

        # plot image
        axis[current_y, current_x].imshow(img, alpha=0.7)

        # plot class activation map
        axis[current_y, current_x].imshow(CAM, cmap='jet', alpha=0.5)

        if label == 0:
            class_label = 'Healthy'
        else:
            class_label = 'Depression'
        axis[current_y, current_x].set_title(class_label)

        if (current_x+1)%9 == 0:
            current_x = 0
            current_y +=1
        else:
            current_x += 1
```

```
plot_class_activation_map(img_list)
```

A similar process was conducted on both ResNet50 and MobileNet models to obtain heatmaps.

Eswar, 12/20/22

Analysis



Generated Heat Maps for Novel Network

From analyzing the output, we were able to construct novel heatmaps. Though, because this is a first attempt at heat map creation, many of the matrices were distorted, throwing the heatmap off the resting-state slice. Additionally, utilizing the middle most slice in analysis might need to be further examined in the future. The middle most slice was assumed to contain the most accurate representation of the MRI. However, surrounding slices may have more accurate depictions of the implications in MDD patients.

Based on analysis of this first attempt at heatmap creation, it seems that the frontal cortex is implicated in MDD patients; however, because of the matrice issues as noted earlier, this region may be inaccurate (U.S. Bureau of Labor Statistics, 2020). One potential source of error may be the fact that stacked np arrays are being fed into generate heatmaps. Instead, these stacked can be squeezed in order to extract a singular scan. Though the MRI data had been normalized, it may have not been inputted as such. Additionally, the scans may not be normalized, meaning that the features are lost during predictions as previously noted. These errors are important to consider in future work. However, for now, heatmaps have been generated to demonstrate proof of concept.

Note: Due to file corruption, the heatmaps for ResNet50 and MobileNet are not available but output similar results to the novel map as shown above.

1/05/2022 - Novel 2D Schizophrenia Model Based on Memory Controlled Data

Eswar, 1/05/2022

Introduction

With relatively successful construction of MDD models as noted by a previous entry, it was decided that the SZD model can be constructed at this point in time. Similar to MDD, multiple methods were tested out in order to determine the feasibility of utilizing 2D CNN's on SZD. Additionally, multiple datasets exist for SZD, meaning that this entry denotes the first of many future ones that may experience a change in dataset.

Eswar, 1/05/2022

Objectives

First and foremost, the main objective for this entry is to successfully construct a model for SZD. However, as seen with the MDD model, accuracy rates in the ~80% range are possible. As a result, I am to achieve a validation accuracy of at least 80% with this model. More specifically, I am prioritizing consistency in the graph output. I am hoping to see validation rates remain at a consistent level rather than alternating epoch per epoch.

Eswar, 1/05/2022

Methods

In order to conduct the novel model for this dataset, the same process from before was utilized.

```
model = tf.keras.models.Sequential()
model.add(layers.Conv2D(64,(12,12),activation='relu',input_shape=(288, 288, 1)))
model.add(layers.Conv2D(64,(8,8),activation='relu'))
model.add(layers.MaxPooling2D((3,3)))
model.add(layers.Conv2D(32,(6,6),activation='relu'))
model.add(layers.Conv2D(32,(4,4),activation='relu'))
model.add(layers.MaxPooling2D((3,3)))
model.add(layers.Conv2D(16,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(2048,'relu'))
model.add(layers.Dense(1024,'relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(512,'relu'))
model.add(layers.Dense(256,'relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(64,'relu'))
model.add(layers.Dropout(0.4))
model.add(layers.Dense(8,'relu'))
model.add(layers.Dense(1,'sigmoid'))
model.summary()
```

This model framework is derived from the same construction for the MDD models. In order to maintain consistency across this project, the same framework was used. However, due to unexpected results over the course of testing, this model was interrupted early.

Analysis

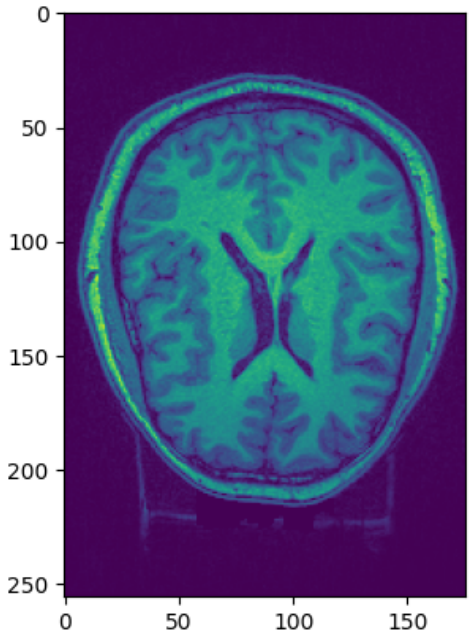
As stated in the objective statement, 80% accuracy range was the desired target. However, after analyzing the model performance during testing, it was noted that the model was unable to predict for SZD. With further analysis into the dataset, a few plausible theories arose.

First and foremost, the plain idea that there were no deviations in the TRS-MRI scans for SZD patients was considered. However, based on prior literature, it was noted that there must be some form of difference occurring. Based on this assumption, other routes were considered.

Analyzing the input of the data was another theory I had. If the data was incorrectly inputted or labeled, the model would not be able to predict. Though, after several reviews of the codebase, it appeared that the data was being inputted correctly. Furthermore, to validate the idea that the data was being inputted correctly, individual slices were displayed with *imshow*.

```
plt.imshow(t2[145])
```

In this particular instance of index 145, the following appeared. As a result, there seemed to be no issues in terms of data input as the scans appeared with clear imaging.



The final theory for the disruption of the model ended up being deemed correct. Because the study was focusing on memory control-based scans, the MRI data was selected to be used for memory purposes. As a result, the dataset being used was not representative of the base-level of a typical healthy

SZD brain. Future work involves looking for a new dataset for SZD as several exist. Once a new one is selected, future work involves ensuring that the dataset does not possess any potential biases that might lead to a lack of prediction or flawed results.

1/14/2022 - Novel 2D Schizophrenia Model Based on the UCLA Consortium

Eswar, 1/14/2022

Introduction

As previously noted, multiple datasets exist for schizophrenia. However, as noted by a previous entry the accuracy rate was not satisfactory. The dataset may have included other factors/conditions in the patients that created unintended consequences. As a result, this entry serves to test the model on a different dataset provided by the UCLA Consortium (https://openfmri.org/dataset/ds000030/). This study was deemed as satisfactory for the model as the data provided did not appear to have any biases. The study simply collected scan data across disorders to further understand the implications of these disorders. In particular, they had *T1-weighted Anatomical MPRAGE* slices, the model of slice being used in this project.

Eswar, 1/14/2022

Objectives

Similar to the first attempt at creating a model for SZD, the accuracy rate was the first priority. Given that the first model failed to predict, the expectations in terms of prioritization for this model was slightly lowered. Rather than aiming for 80%+ accuracy I instead hoped to reach at least a level of 75% accuracy.

Furthermore, I wanted to see a “normal” breakdown in terms of accuracy. By normal, I define it to be a consistent validation across the epochs in use for this model. A non-normal graph may allude to impacts of overfitting or inaccuracies in the model, leading to inaccurate results.

Eswar, 1/15/2022

Methods

Similar to prior methods, the data input process has remained static. Using the sitk attribute of readImage, an array of coordinate points was generated that depict the MRI. These were then zoomed to a scale of 244/288 to meet the ResNet50 input requirements

```
X = []
name = []

for i in range(3000):
    try :
        t1 = sitk.ReadImage("./../data/Healthy/sub-" + (str(i+10000)) +"/anat/sub-" + (str(i+10000)) +
"_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
        name.append(i+10000)
    except:
        continue
```

```
for i in range(100):
    try :
        t1 = sitk.ReadImage("./../data/Schizophrenia/sub-" + (str(i+50000)) +"/anat/sub-" +
(str(i+50000)) + "_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
    except:
        continue

print(len(X))
```

After the images were loaded in, they were analyzed for shape. Because shape must be homogeneous in order to convert to np array prior to compiling the model, the sessions that did not fit the required size were excluded. After analysis, it was found that the MRI at index 126 did not meet the input requirements. As a result it popped. Because only one MRI was popped in a dataset of 150+, it was decided that removing the singular slice would not impact the modal.

```
X.pop(126)

y = np.concatenate(( [1]*125, [0]*49 ))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=34)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Afterwards, the dataset was broken into the 70-30 configuration as used through all of the models with random state of 34.

```
X_train = np.resize(X_train, ((121*40), 199, 137))
X_test = np.resize(X_test, ((53*40), 199, 137))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train), np.shape(X_test),
np.shape(y_test))
```

```
X_train = normalize(X_train)
X_test = normalize(X_test)

model = tf.keras.models.Sequential()

model.add(layers.Conv2D(64,(4,4),activation='relu',input_shape=(199, 137,
1)))
model.add(layers.MaxPooling2D((3,3), padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64,(4,4),activation='relu'))
model.add(layers.MaxPooling2D((2,2), padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2), padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2), padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(16,(2,2),activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dense(1024,'relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(512,'relu'))
model.add(layers.Dense(256,'relu'))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(64,'relu'))
model.add(layers.Dense(16,'relu'))
model.add(layers.Dense(1,'sigmoid'))
model.summary()
```

```
model.compile(optimizer='adam',

 loss=tf.keras.losses.binary_crossentropy,
              metrics=['accuracy'])

print(np.shape(X_train))
model.summary()
```

Similar to previous models, maxpooling, batch normalization, conv, and dense layer. Afterwards Adam optimizer was utilized with measures for cross entropy and accuracy. Furthermore, the model outputted an overview of the layers with shape and params as displayed below. The total parameters for the model was 1,094,193 with only 480 parameters being non-trainable.

```
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)            (None, 196, 134, 64)      1088

 max_pooling2d_4 (MaxPooling  (None, 66, 45, 64)        0
 2D)

 batch_normalization_5 (Batc  (None, 66, 45, 64)        256
 hNormalization)

 conv2d_6 (Conv2D)            (None, 63, 42, 64)        65600

 max_pooling2d_5 (MaxPooling  (None, 32, 21, 64)        0
 2D)

 batch_normalization_6 (Batc  (None, 32, 21, 64)        256
 hNormalization)

 conv2d_7 (Conv2D)            (None, 30, 19, 64)        36928

 max_pooling2d_6 (MaxPooling  (None, 15, 10, 64)        0
 2D)
...
Total params: 1,094,673
Trainable params: 1,094,193
Non-trainable params: 480
```

Once the modal was computed, metrics for analyzing the performance were generated. The first metric used was the confusion matrix in order to provide a visual sense of the number of true neg/pos the model was outputting. The second metric demonstrates accuracy over epoch.

```
y_pred = []
y_true = y_test
# for i in X_test: print(i[:3000], "\n")

for i in X_test:
    i = np.expand_dims(i, axis=0)
    y_pred.append(np.round(model.predict(i)))

y_pred
```

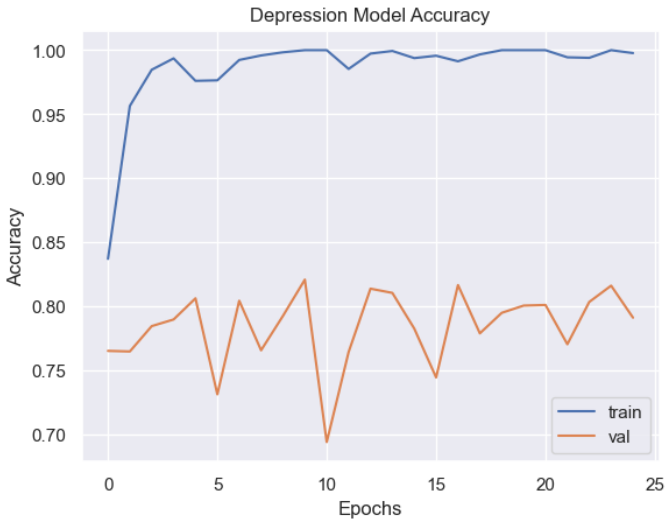
```
Output exceeds the size limit. Open the full output data in a text editor
1/1 [==============================] - 1s 654ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 14ms/step
```

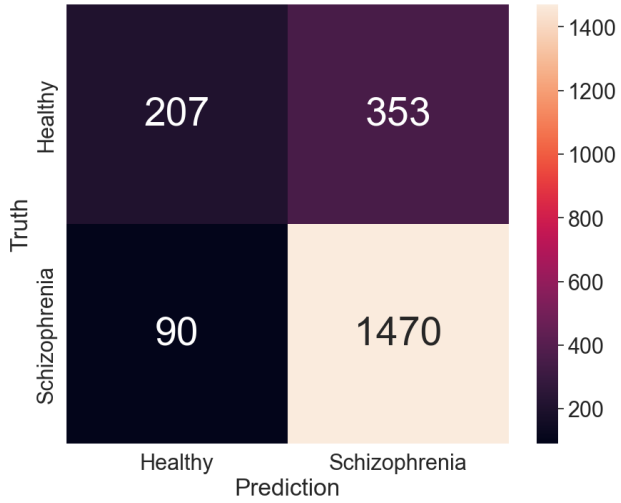
```
correct_labels = np.array(tf.concat([item for item in y_true], axis = 0))
predicted_labels = np.array(tf.concat([item for item in y_pred], axis =
0))
confusion_mtx = tf.math.confusion_matrix(correct_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.set(font_scale=2)
sns.heatmap(confusion_mtx,
            xticklabels=["Healthy", "Schizophrenia"],
            yticklabels=["Healthy", "Schizophrenia"],
            annot=True, fmt='g', annot_kws={"size":40})
plt.xlabel('Prediction')
plt.ylabel('Truth')
plt.show()

print(confusion_mtx)
```

```
sns.set(font_scale=1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Depression Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'val'], loc='lower
plt.show()
```

Analysis

The following charts demonstrate the performance of the model based on accuracy and misclassifications or correct classifications.





Overview: Based on the metrics above, the model had exceptional performance. In terms of model validation, the computation finished with an accuracy of 82.08%. Furthermore, the confusion matrix reflects this exceptional performance. The model had 1470 correct classifications for schizophrenia compared to 353 SZD misclassifications.

Objectives: All of the objectives were met; however, the validation consistency deviated heavily, resembling no consistency. Though, after epoch 15, the accuracy began to reach consistent levels, so this objective was considered to be satisfied as well.

Future work:
In terms of future work, the model will be improved with ResNet50. Based on previous testing with MobileNet, it has been observed that MobileNet does not provide any significant advantages for accuracy. As a result, only ResNet50 will be used for future testing as well as heatmap generation.

1/14/2022 - ResNet50 2D Schizophrenia Model Based on the UCLA Consortium

Eswar, 1/14/2022

Introduction

Based on previous results, ResNet50 can be utilized in order to increase accuracy levels. Previously, a novel neural network was constructed for this dataset; however, the accuracy rates peaked at around 82%. In order to provide a more accurate depiction of the MRI when creating heatmaps, upping the accuracy level should be prioritized. Using this line of logic, a ResNet50 model was constructed on the UCLA consortium dataset.

Eswar, 1/14/2022

Objectives

The main goal for this model is to increase the accuracy levels of the pre-existing novel network to surpass 80%. Furthermore, a consistent validation accuracy graph is another area of importance for this model. Previous novel models did not have a consistent accuracy graph, so this is an area that will be prioritized with this model.

Eswar, 1/15/2022

Methods

```
preprocess_input_resnet = tf.keras.applications.resnet.preprocess_input
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

X = []
name = []

for i in range(3000):
    try :
        t1 = sitk.ReadImage("./../data/Healthy/sub-" + (str(i+10000)) +"/anat/sub-" +
(str(i+10000)) + "_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
        name.append(i+10000)
    except:
        continue



for i in range(100):
    try :
        t1 = sitk.ReadImage("./../data/Schizophrenia/sub-" + (str(i+50000)) +"/anat/sub-"
+ (str(i+50000)) + "_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
    except:
        continue

print(len(X))
```

The reasoning for the loop ranges is based on the image data. As seen above, the loops have either 10000 or 50000 appended to the original value. This decision was made due to the nature of the dataset. The consortium contains several disorders. In order to identify the disorder, the first number of the file is utilized. In this case, the 1 digit of the identifier represents the healthy patients while the 5 represents SZD patients.

As noted in a previous entry, data with a non-homogenous shape cannot be utilized for classification learning. In this case, this led to the MRI at index 107 being excluded. Furthermore, standard functions are denoted before. These include the normalize function and image display function. The normalize function allows for features to be re-fixed to the MRI while the display image function was utilized to ensure the correctness of the dataset after removal of sub 107.

```
# data filter
X.pop(107)

testX = []

for i in range(len(X)):
    print(str((len(X))) + " "+ str(len(X[i])) + " " + str(len(X[i][30])))


# normalize function
def normalize(arr):
    arr = arr + abs(np.amin(arr))
    assert np.amax(arr) != 0
    arr = arr / np.amax(arr)
    return arr

 # image display

plt.figure()

f, axarr = plt.subplots(6, 29, figsize=(20,10))


# use the created array to output your multiple images. In this case I have stacked 4
images vertically
for i in range(6):
    for j in range(29):
        axarr[i,j].imshow(X[(i*29)+j][30])
```

Once the data was validated for correctness in shape, the dataset was imputed into the model. Validation consisted of several components. The MRIs were checked first for shape. If the MRI was off-center, the algorithm would have the chance to bias results, providing a biased model. Furthermore, this similar theory of bias applied to color as well as other minor features within the scans that the model may have been relying on.

After these features were validated for, the dataset was resized for *X_train* and *X_test* with appropriate corresponding arrays for the *y_train* and *y_test*. The images were then stacked by 3 into the RGB test and train arrays as per ResNet50 requirements.

```
X_train = np.resize(X_train, ((121*40), 199, 137))
X_test = np.resize(X_test, ((53*40), 199, 137))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train), np.shape(X_test),
np.shape(y_test))
X_train_RGB = np.stack((X_train,)*3, axis=-1)

X_test_RGB = np.stack((X_test,)*3, axis=-1)

train_ds = preprocess_input_resnet(X_train_RGB)
test_ds = preprocess_input_resnet(X_test_RGB)
```

After data validation, no other metrics were required to be checked prior to model creation. However, ResNet was not activated to start with. The model was first run with no params for the first 10 epochs. After those epochs had been computed, the model switched to include ImageNet weights from ResNet, which bolster accuracy rates. These concepts are demonstrated below.

```
inputs = tf.keras.layers.Input((199, 137, 3))
x = base_model_resnet(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Flatten()(x)
predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)

# create the full network so we can train on it
model_resnet = tf.keras.Model(inputs=inputs, outputs=predictions)

base_learning_rate = 0.0001
model_resnet.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

model_resnet.summary()
```

```
---------

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 199, 137, 3)]     0

 resnet50 (Functional)       (None, 7, 5, 2048)        23587712

 global_average_pooling2d_1   (None, 2048)             0
 (GlobalAveragePooling2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
```

```
history = model_resnet.fit(train_ds, y_train,
                    epochs=10,
                    validation_data=(test_ds, y_test))

 ---------------

 Epoch 8/10
152/152 [==============================] - 47s 310ms/step - loss: 0.3692 - accuracy:
0.8407 - val_loss: 0.4526 - val_accuracy: 0.8005
Epoch 9/10
152/152 [==============================] - 47s 313ms/step - loss: 0.3599 - accuracy:
0.8450 - val_loss: 0.4493 - val_accuracy: 0.7915
Epoch 10/10
152/152 [==============================] - 48s 318ms/step - loss: 0.3527 - accuracy:
0.8473 - val_loss: 0.4501 - val_accuracy: 0.8000
```

After completing the first 10 epochs, the model had roughly an accuracy of ~80% with validation loss of 0.4501. Overall, at this point, the learning rates appeared to be roughly normal given the rate of 0.001 from the ResNet50 base. From the first 10 epochs, these results so far seemed to be reasonable. From this point onwards, the *trainable* attribute of ResNet was enabled

```
base_model_resnet.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model_resnet.layers))

# Fine-tune from this layer onwards
fine_tune_at = 10

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model_resnet.layers[:fine_tune_at]:
  layer.trainable = False

model_resnet.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer =
tf.keras.optimizers.Adam(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])

fine_tune_epochs = 10
total_epochs =  20 + fine_tune_epochs

history_fine = model_resnet.fit(train_ds, y_train,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=(test_ds, y_test), callbacks=[
                         tf.keras.callbacks.EarlyStopping(
                         monitor='val_loss',
                         patience=12,
                         mode='auto',
                         restore_best_weights=True,
)])
```

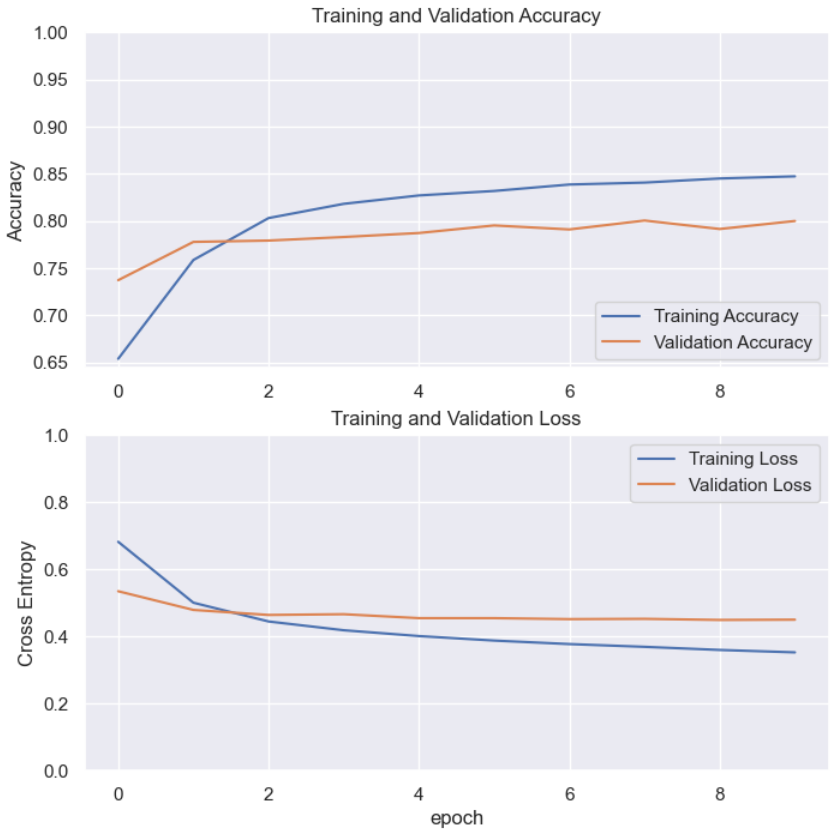
```
152/152 [==============================] - 111s 731ms/step - loss: 3.5118e-05 - accuracy:
1.0000 - val_loss: 1.3881 - val_accuracy: 0.8005
Epoch 21/30
152/152 [==============================] - 110s 725ms/step - loss: 2.8742e-05 - accuracy:
1.0000 - val_loss: 1.4182 - val_accuracy: 0.8000
Epoch 22/30
152/152 [==============================] - 112s 740ms/step - loss: 2.3570e-05 - accuracy:
1.0000 - val_loss: 1.4413 - val_accuracy: 0.7986
```

With the model reaching completion, the overall ending accuracy appeared to be in the ~79% range. In order to further analyze these results, confusion matrices and an accuracy over epoch graph were generated with the code shown below.

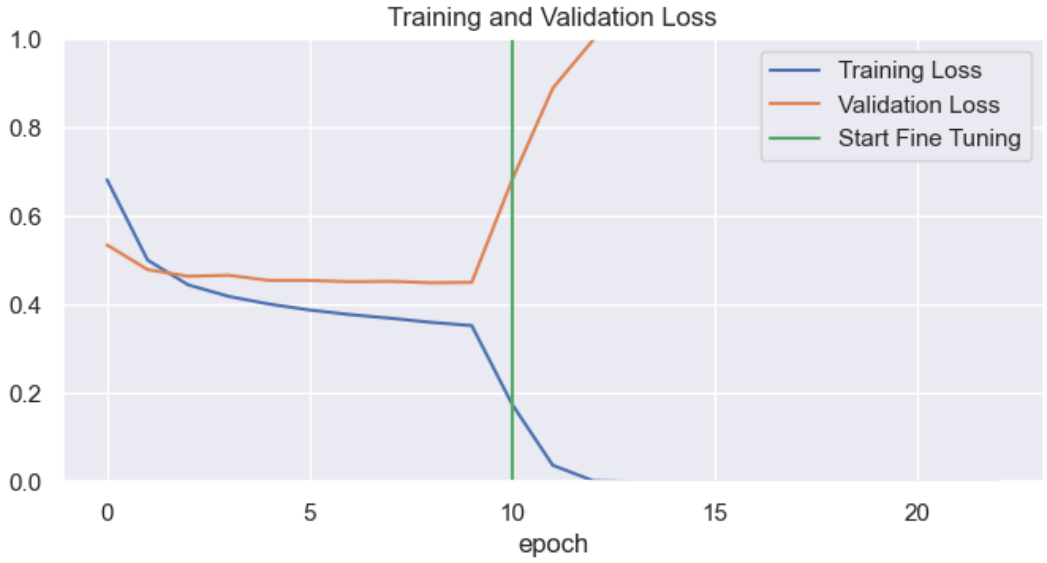
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']

initial_epochs = 10

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.5, 1])
plt.plot([initial_epochs,initial_epochs],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs,initial_epochs],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()

correct_labels = np.array(tf.concat([item for item in y_true], axis = 0))
predicted_labels = np.array(tf.concat([item for item in y_pred], axis =
0))
confusion_mtx = tf.math.confusion_matrix(correct_labels, predicted_labels)
plt.figure(figsize=(10, 8))
sns.set(font_scale=2)
sns.heatmap(confusion_mtx,
            xticklabels=["Healthy", "Schizophrenia"],
            yticklabels=["Healthy", "Schizophrenia"],
            annot=True, fmt='g', annot_kws={"size":40})
plt.xlabel('Prediction')
plt.ylabel('Truth')
plt.show()

print(confusion_mtx)
```

Eswar, 1/16/2022

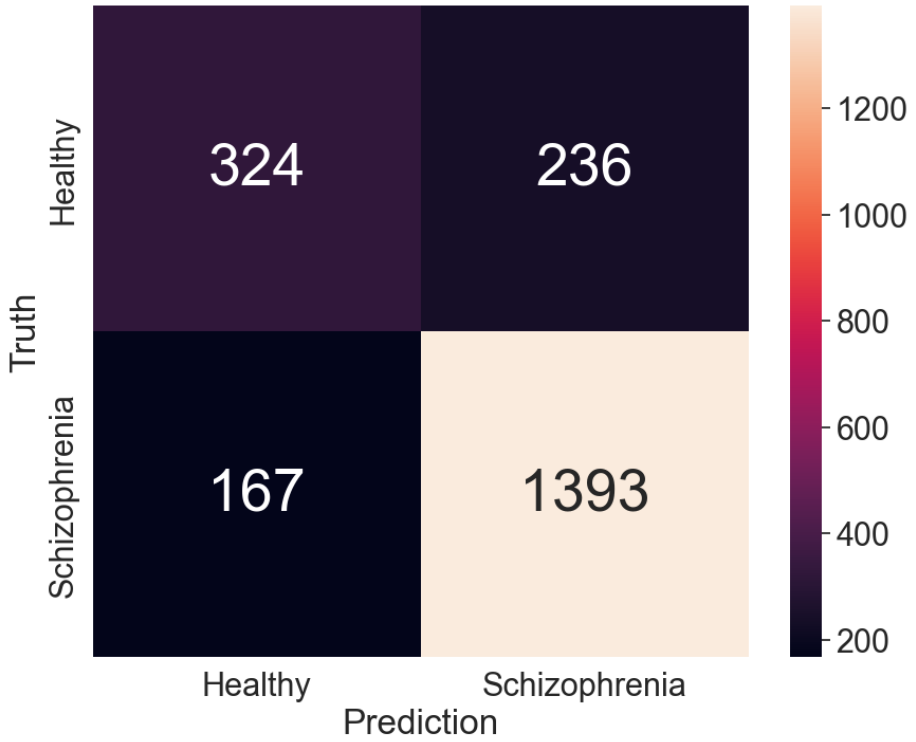
Analysis



This first figure represents the overall change in accuracy and loss over the interval of epochs. Because this duration did not make use of ResNet50, the model accuracy resembles a graph that is similar to the novel network. However, the more unexpected results appear after computing the ResNet50 model. It was assumed that a bolster of accuracy would be observed, though this was not the case.



As demonstrated above, the model did not experience any significant increase in accuracy after epoch 10. This may have been due to a few different factors; however, before looking at those specific factors, analyzing the confusion matrix is another area of interest.



The model was able to predict SZD cases at a remarkably high level. However, it often confused the SZD cases with healthy denoted by the corner with 236. The stats below generated by a confusion matrix calculator provide further insight on these results.

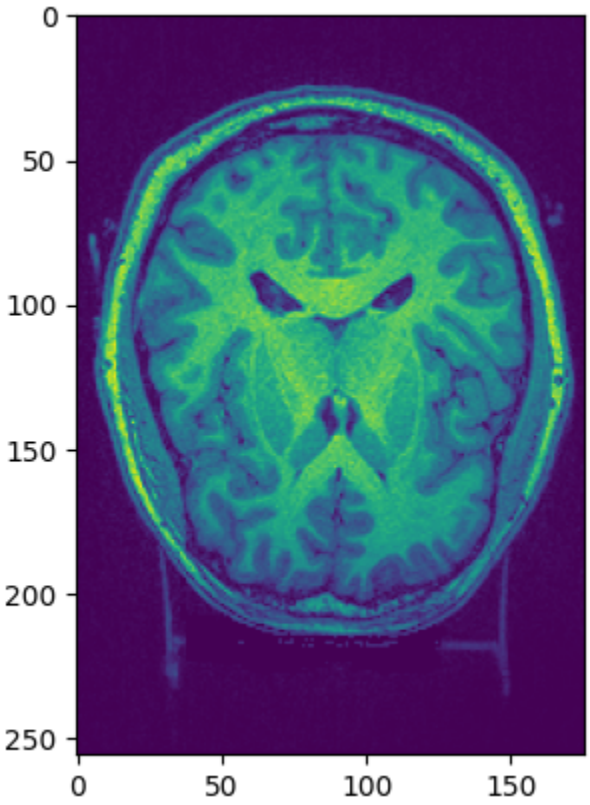
Sensitivity	0.6599
Specificity	0.8551
Precision	0.5786
Negative Predictive Value	0.8929
False Positive Rate	0.1449
False Discovery Rate	0.4214
False Negative Rate	0.3401
Accuracy	0.8099
F1 Score	0.6166
Matthews Correlation Coefficient	0.4928

(Confusion Matrix)

Demonstrated by the Matthews Correlation Coefficient and F1 scores, this model is most likely not suitable compared to the novel model. The F1 score ranks as decent at best with a value of 0.6166. Furthermore, Matthews Correlation Coefficient ranges from -1 to 1, with 1 indicating a perfect model. In this sense, the model ranks in the 75th percentile for match, another poor indication. This output demonstrates the misleading configuration of the confusion matrix. From a glance, one would assume

that the model was able to predict SZD well. However, based on the specificity scores and sensitivity scores, the model, in reality, had difficulty in predicting SZD patients and instead was able to determine healthy patients fairly well.

With these results in mind, it is important to acknowledge some potential sources of error to guide future work. One of the main issues that may have occurred comes from the potential of MRI data being unreasonable. However, this theory was ruled out after analyzing a few slices. For the most part, the MRI slices tended to appear as the following:



Overall, the scan is clear and should be easily readable by the model. As a result, this leads into the second theory. My second theory is that ResNet's model construction was at optimal performance. Rather, the novel network was constructed in such a thorough fashion that it outperformed this pre-trained network. Furthermore, ResNet has biases from other datasets whereas the novel network presents itself as an unbiased outlook into analyzing the images.

(Dataset demonstrating clear imaging)

Based on the fact that the novel network outperformed ResNet50, the novel has been selected to generate heatmaps in the future as well as be used in future applications. Future work aims to improve this model, though at this point, the work conducted on the novel model proves to be sufficient.

1/17/2022 - SZD Heatmap Creation

Eswar, 1/17/2022

Introduction

Based on the work of previous entries, a model has been confirmed in order to analyze SZD. As a result, with the model confirmed, heatmaps can be introduced to further analyze the disorders; however, as noted before, heatmaps can be difficult to work with due to weightings. As a result, in the creation of these heatmaps, a few different factors must be accounted for to provide more reliable results.

Eswar, 1/17/2022

Objectives

In the creation of these heatmaps, a primary objective is accuracy. I am to create accurate and reliable heat maps that will allow for insight and analysis into the disorder. Furthermore, I would like to create heat maps that are not distorted. As noted last time, the matrix multiplication of weighted layers disordered the jet map, and I am to mitigate such biases by utilizing different methods in this entry.

Eswar, 1/17/2022

Methods

```
X = []
name = []

for i in range(3000):
    try :
        t1 = sitk.ReadImage("./../data/Healthy/sub-" + (str(i+10000)) +"/anat/sub-" +
(str(i+10000)) + "_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
        name.append(i+10000)
    except:
        continue
```

```
for i in range(100):
    try :
        t1 = sitk.ReadImage("./../data/Schizophrenia/sub-" + (str(i+50000)) +"/anat/sub-"
+ (str(i+50000)) + "_T1w.nii.gz")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(107,147):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
    except:
        continue

print(len(X))

X.pop(126)
```

Prior to conducting heatmaps on the model, it was loaded in. Furthermore, in this use case for heatmaps, gradCam was implemented. Rationale for utilization of gradCam will be provided later in the methodology or in the analysis of this entry. By using gradCam, the generation of heatmaps is dependent on a pre-existing set of images converted to L format due to the incapability of using .nii. As a result, the images were pre-stored into the file storage system as shown below in preparation for the heatmap work.

```
model =
tf.keras.models.load_model('../Novel/NOVEL_schizophrenia_resting_state_dataset_t1_2d.h5')

for i in range(174):
    im = Image.fromarray(X[i][20])
    im = im.convert("L")
    im.save("Images/sub" + str(i+1) + ".png")
```

Several of the processes when conducting heatmap work become repetitive. As a result, I created some functions to reuse for later. The images were previously stacked by 3; however, this dimensionality cannot be used for gradCam. As a result, an unstack function that moves the axis based on 0 was used. Furthermore, another function was utilized to provide an array representation of the image. Making use of these functions reduces computation time for later on.

```
def unstack(a, axis=0):
    return np.moveaxis(a, axis, 0)

def get_img_array(img_path, size):
    img = keras.preprocessing.image.load_img(img_path,
target_size=size)
    array = keras.preprocessing.image.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    return array
```

In order to create the gradCam, 3 functions served as the main basis for the model. The first function *make_gradcam_heatmap*, provides the heatmap for a singular sub-session. Architecture of the function is in accordance with gradCam requirements and documentation via keras.

```
def make_gradcam_heatmap(img_array, model, last_conv_layer_name, pred_index=None):

    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output,
model.output]

    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]

    grads = tape.gradient(class_channel, last_conv_layer_output)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
```

An additional method was implemented to create the gradCam as well as save it. As previously mentioned, gradCam heavily relies on the files saved within the directory, and this presents another instance when gradCam will use the directory to pull image/write image data.

```
img_array = preprocess_input(get_img_array(img_path, size=(199, 137, 1)))
img_array = tf.squeeze(img_array)
print(unstack(img_array, axis=2)[0].shape)

def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg",
alpha=0.85):
    img = keras.preprocessing.image.load_img(img_path)
    img = keras.preprocessing.image.img_to_array(img)

    heatmap = np.uint8(255 * heatmap)

    jet = cm.get_cmap("jet")

    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]

    jet_heatmap = keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap = keras.preprocessing.image.img_to_array(jet_heatmap)

    superimposed_img = jet_heatmap * alpha + img
    superimposed_img = keras.preprocessing.image.array_to_img(superimposed_img)

    superimposed_img.save(cam_path)
```

The above classes were primarily used in testing. The below functions demonstrate the core of the heatmap construction. Because the generator needs to iterate through 150+ sub-sessions, the heatmap is actively changing and needs to be adjusted in time before printing or the heatmaps will all be uniform. As a result, the *setHeatMap* function was created to make a gradCam heatmap based on the file input– a reactive change.

```
def setHeatMap(file, path):
    img_array = preprocess_input(get_img_array(path, size=(199, 137,
1)))print(file + " " + path)
    img1 = cv2.imread('Images/' + str(file))
    img1 = unstack(img1, axis=-1)

    # img_array = tf.squeeze(img_array)

    # img_array = np.expand_dims(img1, axis=-1)
    model.layers[-1].activation = None
    img_array = np.expand_dims(img1[2], axis=0)

    preds = model.predict(img_array)
    heatmap = make_gradcam_heatmap(img_array, model, "conv2d_9")

    return heatmap
```

With these support functions tested and working, the main function for heatmap generation was produced.

```
def plot_class_activation_map(images):

    figure, axis = plt.subplots(25, 7)

    figure.set_size_inches(15,55)

    current_y = 0
    current_x = 0

    plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])



    for i in range(images):
        print(i)
        img_path = keras.utils.get_file(

 origin="file:///Users/taruneswar/Documents/General%20Development/STEM%20Models/Schizophrenia/Heatmaps/
Images/sub" + str(i+1) +".png"
    )
        heatmap = setHeatMap("sub" + str(i+1) + ".png", img_path)

        save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg", alpha=0.7)

        image = plt.imread("cam.jpg")
        coordinatesList = [[0, 0]]
        tx, ty = coordinatesList[0]
```

```
        axis[current_y, current_x].imshow(image)
        axis[current_y, current_x].set_title("Patient " + str(i+1)).set_fontsize(10)

        # class_label = "Patient " + str(i+1)
        # axis[current_y, current_x].set_title(class_label)

        if (current_x+1)%7 == 0:
            current_x = 0
            current_y +=1
        else:
            current_x += 1


        #print("MRI number" + i + "produced")


    plt.show()
```

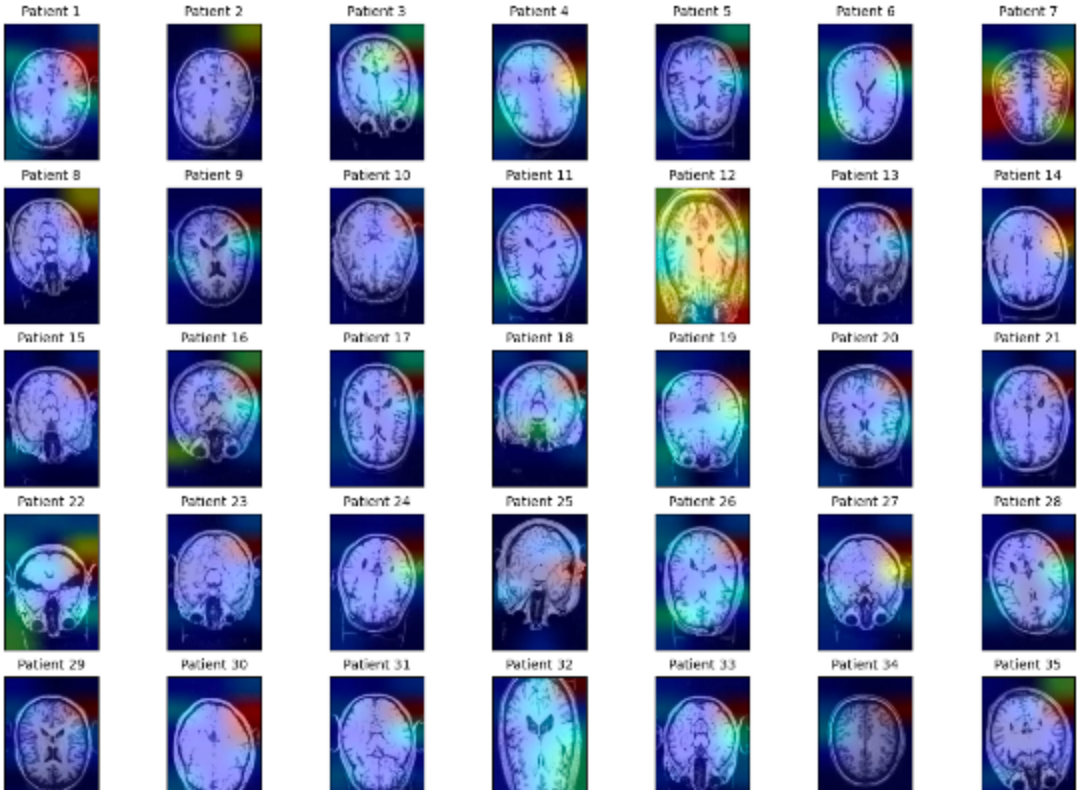
```
plot_class_activation_map(174)
```

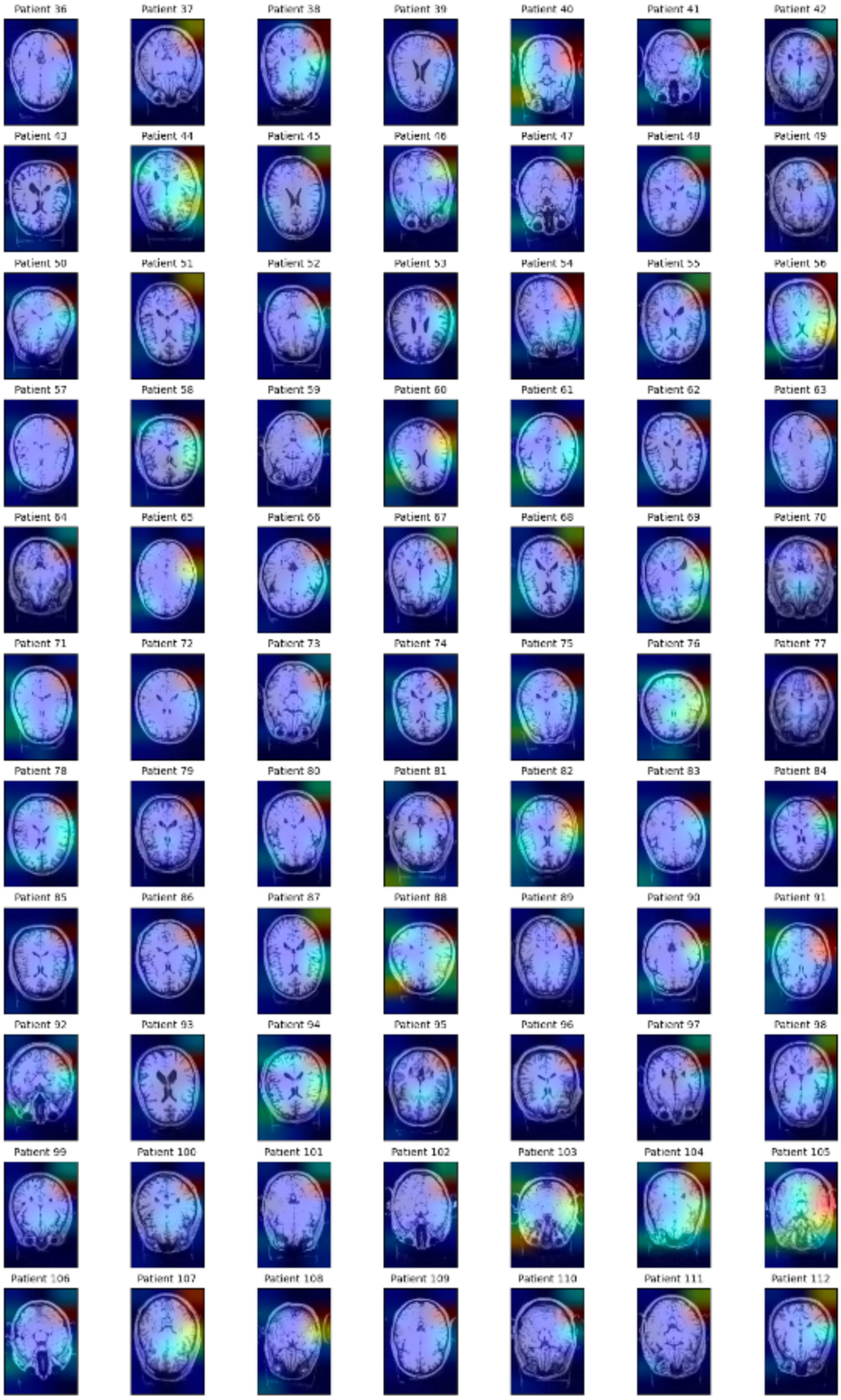
With the final function constructed, the heatmaps were called by passing in the size of the array as noted above.

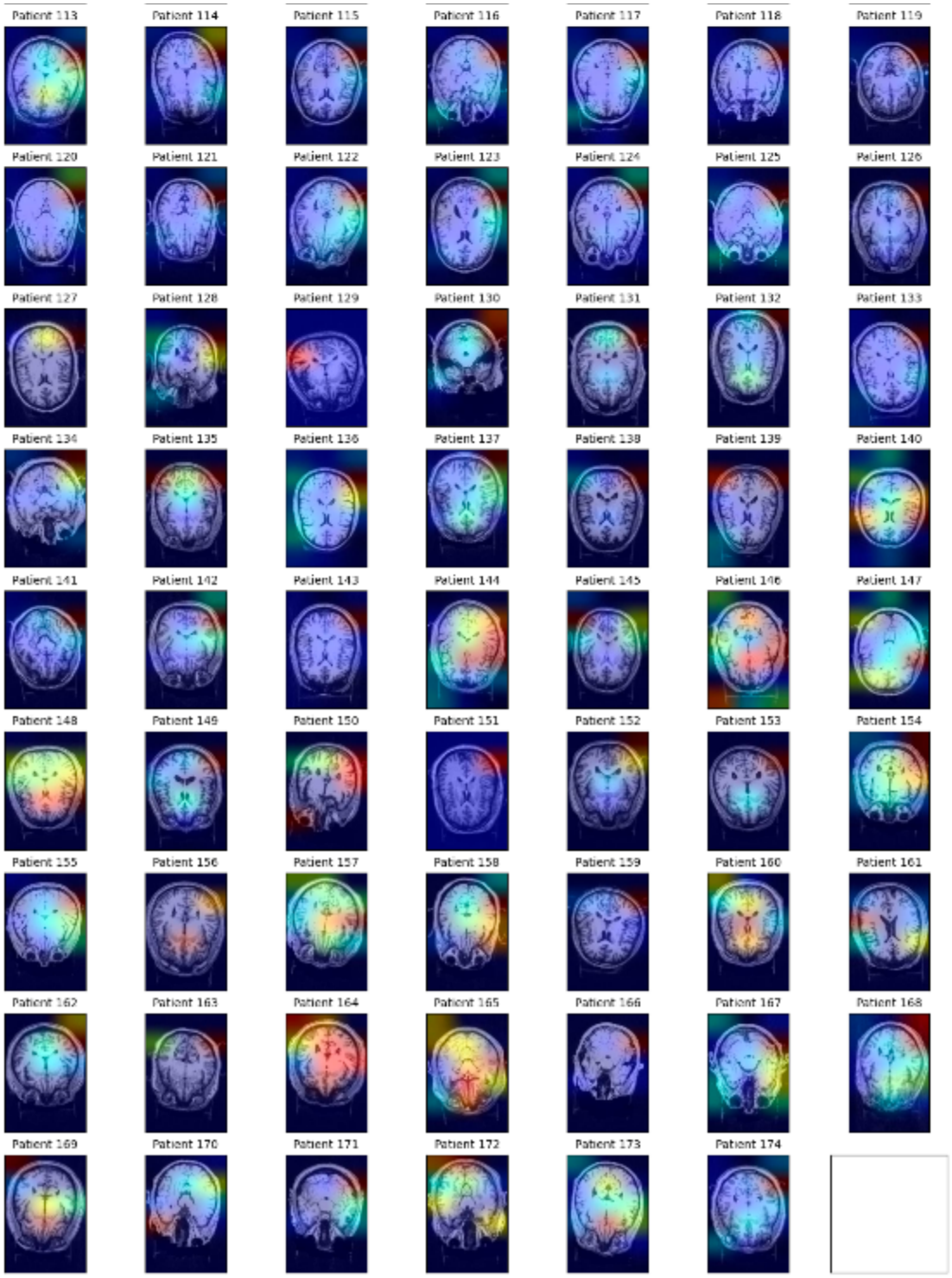
Note: failed or non-working attempts were unable to be displayed due to issues in storage with Github LFS.

Eswar, 1/18/2022

Analysis




Patient 36
Patient 37
Patient 38
Patient 39
Patient 40
Patient 41
Patient 42
Patient 43
Patient 44
Patient 45
Patient 46
Patient 47
Patient 48
Patient 49
Patient 50
Patient 51
Patient 52
Patient 53
Patient 54
Patient 55
Patient 56
Patient 57
Patient 58
Patient 59
Patient 60
Patient 61
Patient 62
Patient 63
Patient 64
Patient 65
Patient 66
Patient 67
Patient 68
Patient 69
Patient 70
Patient 71
Patient 72
Patient 73
Patient 74
Patient 75
Patient 76
Patient 77
Patient 78
Patient 79
Patient 80
Patient 81
Patient 82
Patient 83
Patient 84
Patient 85
Patient 86
Patient 87
Patient 88
Patient 89
Patient 90
Patient 91
Patient 92
Patient 93
Patient 94
Patient 95
Patient 96
Patient 97
Patient 98
Patient 99
Patient 100
Patient 101
Patient 102
Patient 103
Patient 104
Patient 105
Patient 106
Patient 107
Patient 108
Patient 109
Patient 110
Patient 111
Patient 112



As demonstrated above, 176 heatmaps were generated for the SZD dataset. Though at first, the heat maps seem incorrect, these TRS-MRI heat maps are correct. In some places, the dot product of the

matrices places the hotspot slightly off the MRI; however, this is correct. The hotspot is placed off of the MRI due to the *global average pooling* layer. In essence, the heatmaps are not mapping a specific variable, but rather an average of the weightings across all scans. Based on this process, some heatmaps will have slight variations from being perfectly centered on the MRI. In the ideal situation, these individual cases would be condensed into a singular MRI and the variances would be negligible; however, because of the differences in shapes of the MRIs, they cannot be combined and visual analysis must be made.

Based on the majority of the MRI data demonstrating implications in the right frontal lobe, it provides a safe mode of stating that the right frontal lobe is implicated in SZD. Scans that demonstrate differences in other areas represent a smaller percentage of the overall group, and therefore must not be weighted as heavily. Additionally, if we did have the opportunity to condense all of these images into 1 singular image, the number of MRIs with the hotspot in the right frontal cortex would combine to provide a stronger alpha value, showing the region as more prominent than others. These results are in support of prior results of implications in the right frontal cortex nonetheless (Mubarik & Tohid, 2016).

1/18/2022 - MDD Heatmap Correction

Eswar, 1/18/2022

Introduction

Based on the previous entry, it has been illustrated that more accurate heatmaps can be generated. The last time MDD heatmaps were created, they were highly inaccurate. As a result, this entry attempts to fix computation errors made in the previous version of MDD heatmaps.

Eswar, 1/18/2022

Objectives

The primary objective for this entry is to understand the previous mistakes made in prior heatmap generation processes. Once these errors are well-understood, the goal is to account for them when correcting the model.

Eswar, 1/18/2022

Methods

```
X = []

for i in range(9):
    t1 = sitk.ReadImage("../../Data/sub-0" + (str(i+1)) +"/anat/sub-0" + (str(i+1)) +
"_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)

for i in range(9,72):
    t1 = sitk.ReadImage("../../Data/sub-" + (str(i+1)) +"/anat/sub-" + (str(i+1)) + "_T1w.nii.gz")
    t2 = sitk.GetArrayFromImage(t1)
    t3 = []
    for j in range(70, 110):
        result = scipy.ndimage.zoom(t2[j], 224/288)
        t3.append(result)
    X.append(t3)


print(X)
```

```
inputs = tf.keras.layers.Input((224, 224, 3))
x = base_model_resnet(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)

# add a flatten layer
x = tf.keras.layers.Flatten()(x)

# and a fully connected output/classification layer
predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)

# create the full network so we can train on it
model_resnet = tf.keras.Model(inputs=inputs, outputs=predictions)
```

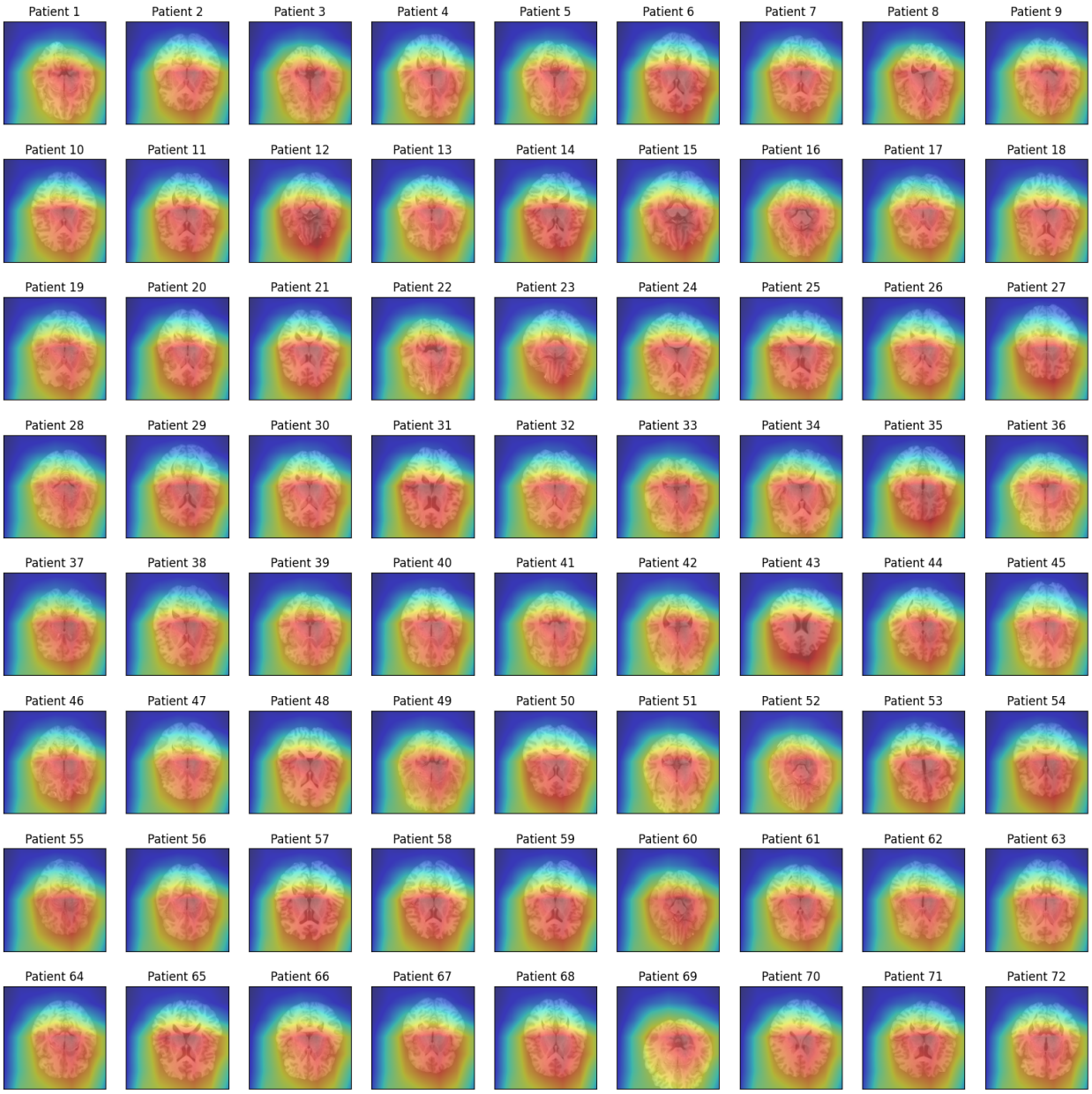
```
def plot_class_activation_map(images):

    figure, axis = plt.subplots(8, 9)

    figure.set_size_inches(20, 20)

    current_y = 0
    current_x = 0

    plt.setp(plt.gcf().get_axes(), xticks=[], yticks=[])

    i = 0

    for img in images:

        img = normalize(img)

        img = tf.squeeze(img)

        CAM, label = get_class_activation_map(img)


        coordinatesList = [[0, 0]]
        tx, ty = coordinatesList[0]
        axis[current_y, current_x].imshow(img, alpha=.6)
        axis[current_y, current_x].imshow(CAM, cmap='jet', alpha=0.5)
        class_label = "Patient " + str(i+1)
        axis[current_y, current_x].set_title(class_label)

        if (current_x+1)%9 == 0:
            current_x = 0
            current_y +=1
        else:
            current_x += 1
        i+=1

        #print("MRI number" + i + "produced")


    plt.show()
```

```
img_list = np.concatenate([test_ds[19::40],
print_class_activation_map(img_list)
```

Though difficult to demonstrate, the codebase was analyzed line-by-line with variables adjusted. After several takes of analyzing the data as well as speaking to experts in the field, nothing appeared to be correct. However, I noticed that there was an incorrect scale factor as well as lack of correct renormalization. As a result, the distorted heatmaps from earlier were produced. The heat maps that will be analyzed in the section following demonstrate the increase in accuracy of the heatmaps.

Eswar, 1/19/2022

Analysis



Heatmap output after corrections

General heatmap analysis: Based on the outputs provided above, the heatmaps appear to be far more accurate compared to the earlier versions. Though the hotspot does leak off the MRI, this will be discussed later in the analysis. As for the overall shape, the placement and intensity of the heatmaps are exactly as expected. Furthermore, rather than using gradCam as done for SZD,

a different, novel approach, for heatmap generation was utilized as shown by the classes in the methodology. Cross-comparison of heatmaps from the same technique of creation may lead to bias because if the technique had a flaw, it would not be noticeable. As a result, this second technique on generation provides more consistency later on when drawing results.

Understanding significance: Previous literature typically utilizes biological modeling to understand the implicated regions of the MRI. However, in this approach, ResNet50 is utilized. Though other models do exist for MDD, this model provides a new take in the sense that the model is created as well as utilized to draw conclusions to specific regions of the MRI. Furthermore, the region noted as implicated—the corpus callosum—does align with more biological methods of the past, demonstrating further support for irregularities in the region (van Velzen et al., 2020). The power of the model is also seen in comparison to van Velzen et al. In their study, they state "The largest differences were observed in the corpus callosum and corona radiata." Their study required 20 international cohorts to understand the same implications that this methodology was able to uncover from only 72 patients. As a result, this model serves as viable usage in environments where less samples are present. In terms of MRI scanning, datasets typically come with fewer patients. As a result, this same process for understanding the MRI for a different dataset demonstrates a proof of concept to be utilized in the future.

Future work: Understanding the full complexity of the model architecture will need to be done in the future. Currently, because of lack of understanding the processes being used in the conv layers, the model resembles a black box. Future work should analyze this model to understand the accuracy it provides. If the accuracy in terms of concepts utilized are considered correct, the model may be applied to alike disorders or varying datasets. These ideas prove to be a major finding if future work can validate the model. Disorders would have the potential to be modeled with as few as 72 MRI scans, or potentially less.

1/25/2022 - OCD Model Creation

Eswar, 1/25/2022

Introduction

With the successful construction of models for SZD and MDD, it was determined that OCD was feasible to conduct. Proof that structural differences exist in MDD and SZD patients paves the way for this model. However, previous works have conflicting conclusions on the implications of TRS-MRI for OCD patients. As a result, analyzing the output of this model in particular will be important.

Eswar, 1/25/2022

Objectives

The main objective for this model is to create a model for OCD utilizing ResNet50 with a validation accuracy of ~75%. Furthermore, consistency of validation graphs is desired in order to ensure the results are not skewed or biased from epoch to epoch in any way.

Eswar, 1/25/2022

Methods

Due to the static nature of the methodology used in model creation, several parts of the methodology below will be left without explanations as they were entailed in previous entries.

```
y = np.concatenate(( [0]*34, [1]*30 ))

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=34)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
preprocess_input_resnet =
tf.keras.applications.resnet.preprocess_input
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

X = []
name = []

for i in range(500):
    try :
        t1 = sitk.ReadImage("./../data/old_data/FA" +
str(1000+i) + ".nii")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(25,65):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)
```

```
for i in range(500):
    try :
        t1 = sitk.ReadImage("./../data/old_data/FA" +
str(2000+i) + ".nii")
        t2 = sitk.GetArrayFromImage(t1)
        t3 = []
        for j in range(25,65):
            result = scipy.ndimage.zoom(t2[j], 224/288)
            t3.append(result)
        X.append(t3)

    except:
        continue

print(len(X))
```

The datasets either had 1000 or 2000 appended to the original value depending on healthy/OCD.

```
X_train = np.resize(X_train, ((40*44), 85, 71))
X_test = np.resize(X_test, ((40*20), 85, 71))

temp_y_train = np.array([])
for i in range(len(y_train)):
    for j in range(40):
        temp_y_train = np.append(temp_y_train, y_train[i])

temp_y_test = np.array([])
for i in range(len(y_test)):
    for j in range(40):
        temp_y_test = np.append(temp_y_test, y_test[i])

y_train = temp_y_train
y_test = temp_y_test

print(np.shape(X_train), np.shape(y_train), np.shape(X_test),
np.shape(y_test))


X_train_RGB = np.stack((X_train,)*3, axis=-1)

X_test_RGB = np.stack((X_test,)*3, axis=-1)

train_ds = preprocess_input_resnet(X_train_RGB)
test_ds = preprocess_input_resnet(X_test_RGB)
```

```
base_model_resnet = tf.keras.applications.ResNet50(input_shape=
(85, 71, 3),

include_top=False,

weights='imagenet')
base_model_resnet.trainable = False

inputs = tf.keras.layers.Input((85, 71, 3))
x = base_model_resnet(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Flatten()(x)
predictions = tf.keras.layers.Dense(1, activation='sigmoid')(x)
model_resnet = tf.keras.Model(inputs=inputs,
outputs=predictions)

base_learning_rate = 0.0001
model_resnet.compile(optimizer=tf.keras.optimizers.Adam(learnin
g_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

model_resnet.summary()
```

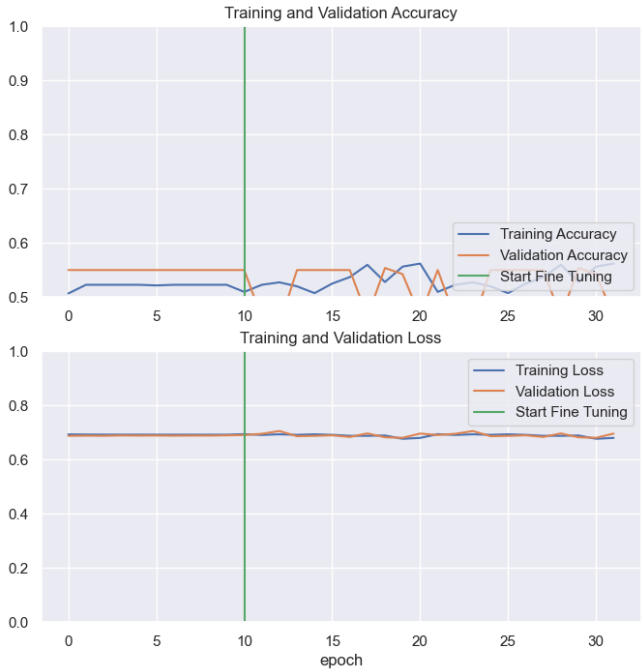
```
55/55 [==============================] - 12s 214ms/step - loss:
0.6891 - accuracy: 0.5278 - val_loss: 0.6825 - val_accuracy: 0.5537
Epoch 19/20
55/55 [==============================] - 12s 211ms/step - loss:
0.6769 - accuracy: 0.5563 - val_loss: 0.6806 - val_accuracy: 0.5425
Epoch 20/20
55/55 [==============================] - 12s 213ms/step - loss:
0.6802 - accuracy: 0.5619 - val_loss: 0.6962 - val_accuracy: 0.4650
```

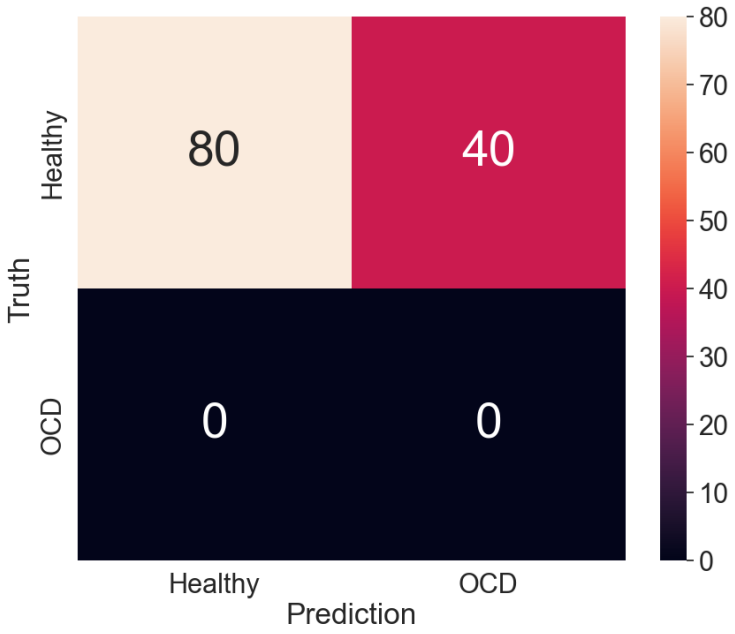
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']

initial_epochs = 10
```

```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.5, 1])
plt.plot([initial_epochs,initial_epochs],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs,initial_epochs],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

```
correct_labels = np.array(tf.concat([item for item in y_true], axis =
0))
predicted_labels = np.array(tf.concat([item for item in y_pred], axis
= 0))

confusion_mtx = tf.math.confusion_matrix(correct_labels,
predicted_labels)
plt.figure(figsize=(10, 8))
sns.set(font_scale=2)
sns.heatmap(confusion_mtx,
            xticklabels=["Healthy", "Depression"],
            yticklabels=["Healthy", "Depression"],
            annot=True, fmt='g', annot_kws={"size":40})
plt.xlabel('Prediction')
plt.ylabel('Truth')
plt.show()

print(confusion_mtx)
```

Analysis





General analysis: based on the accuracy and cross entropy graphs, the model was unable to make predictions. Though the ResNet50 application appeared to create some change in the accuracy and cross entropy, the overall idea is that the model was unable to predict.

Implications: Through the model was unable to predict, several possibilities exist for why the model was unable to predict:

1. There exists no structural differences in TRS-MRI OCD scans
2. The scans were faulty and as a result the model was unable to predict
3. OCD has multiple implicated regions and the model cannot predict as a result

Future work: The primary piece of future work from this entry will be to figure out the underworkings of this model.

2/3/2022 - Verifying FSL Correction for OCD TRS slices

Eswar, 2/3/2023

Introduction

As seen in a previous entry, the OCD model was unable to predict. As a result, a basis for understanding why it was unable to predict is needed. FSL is a common tool used for potential distortions in MRI data to correct them. Therefore, this entry attempts to delve more into utilized FSL.

Eswar, 2/3/2023

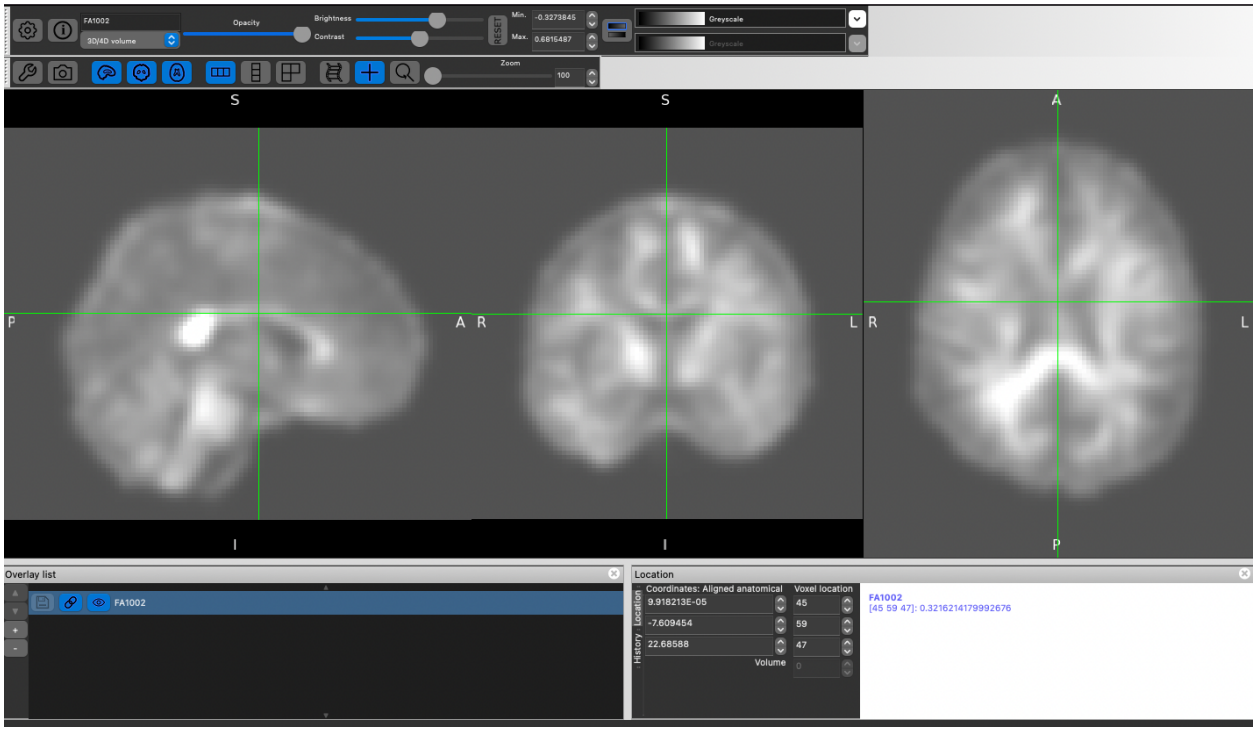
Objectives

The main objective was to investigate the dataset with FSL. due to vague wording in the readme.md file provided by the authors of the dataset. Flawed data is difficult to determine. As a result, FSL will be utilized to understand this specific issue further.

Eswar, 2/3/2023

Methods

The dataset was analyzed with FSL eyes to understand potential distortions. Furthermore, the scans were also put through eddy correction to understand if any difference occurred in the scans before and after the correction. The following demonstrates the usage of FSL eyes.



Note: FSL eddy correction relies on bash scripts and therefore were not recorded

Eswar, 2/3/2023

Analysis

Main takeaway: Based on the analysis conducted through FSL, the data was not distorted as the primary authors of the article already conducted corrections. As a result, there is reason to believe that several regions for the MRI scan are implicated in OCD rather than one singular reason.

Future-analysis based work: Though not recorded in the logbook, future work will involve further research validating these results. At the moment, the results do appear to make logical sense. Several different students have identified different regions of the MRI as implicated in OCD. If several regions are implicated, the model should be unable to predict. The conclusion of this project will rely on validating findings with past sources.

References

Akkus, F., Terbeck, S., Ametamey, S. M., Rufer, M., Treyer, V., Burger, C., Johayem, A., Mancilla, B. G., Sovago, J., Buck, A., & Hasler, G. (2014). Metabotropic glutamate receptor 5 binding in patients with obsessive-compulsive disorder. *International Journal of Neuropsychopharmacology*, *17*(12), 1915–1922. https://doi.org/10.1017/S1461145714000716

Bezmaternykh D.D., Melnikov M.Y., Savelov A.A. et al. Brain Networks Connectivity in Mild to Moderate Depression: Resting State fMRI Study with Implications to Nonpharmacological Treatment. Neural Plasticity, 2021. V. 2021. № 8846097. PP. 1-15. DOI: 10.1155/2021/8846097

Confusion Matrix—Online Calculator. (n.d.). Retrieved February 12, 2023, from https://onlineconfusionmatrix.com/

Kellner, M. (2010). Drug treatment of obsessive-compulsive disorder. *Dialogues in Clinical Neuroscience*, *12*(2), 187–197. https://doi.org/10.31887/DCNS.2010.12.2/mkellner

Kim, Seung-Goo et al. (2015), Alterations of Gray and White Matter Networks in Patients with Obsessive-Compulsive Disorder: A Multimodal Fusion Analysis of Structural MRI and DTI Using mCCA+jICA, PLOS ONE, Article-journal, https://doi.org/10.1371/journal.pone.0127118

Mubarik, A., & Tohid, H. (2016). Frontal lobe alterations in schizophrenia: A review. *Trends in Psychiatry and Psychotherapy*, *38*, 198–206. https://doi.org/10.1590/2237-6089-2015-0088

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *2017 IEEE International Conference on Computer Vision (ICCV)*, 618–626. https://doi.org/10.1109/ICCV.2017.74

van Velzen, L. S., Kelly, S., Isaev, D., Aleman, A., Aftanas, L. I., Bauer, J., Baune, B. T., Brak, I. V., Carballedo, A., Connolly, C. G., Couvy-Duchesne, B., Cullen, K. R., Danilenko, K. V., Dannlowski, U., Enneking, V., Filimonova, E., Förster, K., Frodl, T., Gotlib, I. H., … Schmaal, L. (2020). White matter disturbances in major depressive disorder: A coordinated analysis across 20 international cohorts in the ENIGMA MDD working group. *Molecular Psychiatry*, *25*(7), Article 7. https://doi.org/10.1038/s41380-019-0477-2

Wang, W., Li, Y., Zou, T., Wang, X., You, J., & Luo, Y. (2020). A Novel Image Classification Approach via Dense-MobileNet Models. Mobile Information Systems, 2020, e7602384. https://doi.org/10.1155/2020/7602384

Code Availability

All code and resources utilized for this project is available at https://github.com/Tarune28/Modeling-T1-Resting-State-MRI-Variants-Using-Convolutional-Neural-Networks-in-Diagnosis-of-OCD.