

Spring 2020

Predicting Depression Progression Rates in Radiotherapy Patients

Ardella Phoa

Joshua Vincent

Shani Williams

Follow this and additional works at: https://scholarcommons.scu.edu/bioe_senior



Part of the [Biomedical Engineering and Bioengineering Commons](#)

Recommended Citation

Phoa, Ardella; Vincent, Joshua; and Williams, Shani, "Predicting Depression Progression Rates in Radiotherapy Patients" (2020). *Bioengineering Senior Theses*. 100.

https://scholarcommons.scu.edu/bioe_senior/100

This Thesis is brought to you for free and open access by the Engineering Senior Theses at Scholar Commons. It has been accepted for inclusion in Bioengineering Senior Theses by an authorized administrator of Scholar Commons. For more information, please contact rscroggin@scu.edu.

SANTA CLARA UNIVERSITY

Department of Bioengineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED
UNDER MY SUPERVISION BY

Ardella Phoa, Joshua Vincent, Shani Williams

ENTITLED

**PREDICTING DEPRESSION PROGRESSION RATES OF
RADIOTHERAPY PATIENTS**

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF

**BACHELOR OF SCIENCE
IN
BIOENGINEERING**



Thesis Advisor 6/9/20
Date



Department Chair 6/9/20
Date

**PREDICTING DEPRESSION PROGRESSION RATES OF
RADIOTHERAPY PATIENTS**

BY

Ardella Phoa, Joshua Vincent, Shani Williams

SENIOR DESIGN PROJECT REPORT

Submitted to

the Department Bioengineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements for the degree of

Bachelor of Science in Bioengineering

Santa Clara, California

Spring 2020

Predicting Depression Progression Rates of Radiotherapy Patients

Ardella Phoa

Joshua Vincent

Shani Williams

Department of Bioengineering

Santa Clara University

2020

ABSTRACT

Cancer is a highly prevalent disease that affects millions of people worldwide. In addition to the physiological effects of the disease, cancer patients are more likely to be diagnosed with Major Depressive Disorder (MDD). Unfortunately, prior research has shown that MDD can also decrease the efficacy of radiotherapy cancer treatments. Currently, there is no way to predict, prevent, or mitigate this comorbidity, preventing physicians from administering supplemental therapies. In this paper, we propose a low-cost and efficient computational tool that can be utilized to quantify a patient's likelihood of developing depression. To do so, we used PET images and a ResNet34 architecture to train a convolutional neural network to identify depression biomarkers in the brain. These brain PET images were taken from the Alzheimer's Disease Neuroimaging Initiative (ADNI) dataset and also provided information regarding the patient's depression at the time of the scan. We were then able to label and classify images in our dataset based off of this data. Although our model only yielded an accuracy of 54.25%, sensitivity of 56.25% and a specificity of 53.64%, a visual evaluation of our results (GradCAM) confirmed that our algorithm was able to detect the correct regions of interest in the brain, where depression biomarkers were found. This leads us to believe that our deep learning model, with improvement, can be used to effectively help classify depression progression rates in radiotherapy patients.

Acknowledgements

This project was completed at Santa Clara University Department of Bioengineering with Dr. Yuling Yan and conducted in collaboration with the University of California, San Francisco School of Medicine with Dr. Dugyu Tosun. We would like to thank our two advisors, Dr. Yan and Dr. Tosun, for access to hardware, constant encouragement, and propitious instruction. We would also like to thank Ryan Ellis for his guidance on our network architecture and the ADNI database for providing us with the necessary PET images and information to complete this project. Without these contributions, our project would not have yielded results within our given time frame.

TABLE OF CONTENTS

CHAPTER 1	8
INTRODUCTION	8
<i>1.1 Motivation</i>	8
<i>1.2 Cancer and Depression</i>	8
<i>1.2 Problem</i>	10
<i>1.3 Contributions</i>	10
CHAPTER 2	11
BACKGROUND AND SIGNIFICANCE	11
<i>2.1 Imaging Modality Background</i>	11
<i>2.2 Relevant Work</i>	12
<i>2.3 ResNet</i>	13
<i>2.4 Significance</i>	14
CHAPTER 3	16
SYSTEMS	16
<i>3.1 Image Preprocessing</i>	16
<i>3.2 Deep Learning Architecture</i>	16
<i>3.3 Evaluation</i>	17
CHAPTER 4	18
SUBSYSTEMS	18
<i>4.1 Data Collection</i>	18
<i>4.2 Subject Selection Criteria</i>	18
<i>4.3 Data Labeling</i>	19
<i>4.4 Data Augmentation</i>	19
<i>4.5 Class Distribution</i>	21
<i>4.6 Split</i>	21
<i>4.7 Deep Convolutional Neural Network</i>	22
<i>4.8 Statistical Evaluation</i>	22
<i>4.9 Visual Evaluation</i>	23
CHAPTER 5	24

SYSTEMS-INTEGRATION, TESTS, AND RESULTS	24
5.1 <i>Training</i>	24
5.2 <i>Statistical Evaluation</i>	25
5.3 <i>Visual Evaluation</i>	26
CHAPTER 6	29
PROFESSIONAL ISSUES & CONSTRAINTS	29
6.1 <i>Economics</i>	29
6.2 <i>Health & Safety</i>	29
6.3 <i>Ethics and Privacy</i>	30
6.4 <i>Technology</i>	30
6.5 <i>Usability</i>	31
CHAPTER 7	32
CONCLUSION & FUTURE WORK	32
7.1 <i>Conclusion</i>	32
7.2 <i>Future Work</i>	33

LIST of FIGURES

Figure 1: (Left) Original Image; (Right) Five degree rotation in the transverse plane.	18
Figure 2: Cropped Image	19
Figure 3: Loss	22
Figure 4: Accuracy	23
Figure 5: ROC Curve	24
Figure 6: Positive case CAM	25
Figure 7: Negative Case CAM	25
Figure 8: Oversaturated Image	68
Figure 9: Normal Image	68

LIST of TABLES

Table 1: Confusion matrix representing the model's performance on unseen images.	23
Table 2: Software	36
Table 3: Hardware	36

Chapter 1

Introduction

1.1 Motivation

In 2019 alone, 1.7 million new cases of cancer were diagnosed in the United States [1]. Additionally, cancer patients experiencing severe pain are 2-4 times more likely to develop and be diagnosed with Major Depressive Disorder (MDD) than other patients experiencing lesser degrees of pain [2]. MDD is a mental health disorder that can cause psychological and physical distress, detrimentally affecting a patient's overall quality of life. In a recent meta-analysis, Pinquart et al. established a positive correlation between MDD and mortality rate [3]. Indeed, MDD increases the mortality rates of cancer patients by up to 39% [3]. **Therefore, administering care for MDD alongside radiotherapy is paramount in increasing the efficacy of cancer treatment.**

1.2 Cancer and Depression

Given the overall prevalence of depression, many studies have been conducted to correlate the incidence of depression with other medical illnesses, namely cancer. Among various primary cancer tumor sites, depression rates vary. For example, pancreatic cancer has been proven to alter CNS serotonin receptors, leading to higher risk for depression [4]. However, regardless of cancer site or diagnoses, all occurrences still correlate to a higher risk of developing depression compared to the general population.

It is important to recognize that many cancer-related stressors can also contribute to increased risk of developing depression. For example, the severity of one's illness may affect individuals differently, depending on their pain tolerance or overall ability to overcome hardships [5]. Additionally, cancer patients may receive multiple and varying forms of treatment, such as radiotherapy, chemotherapy, or surgery. Thus, while the overall data still shows an increased risk of depression in cancer patients, it can be difficult to make direct correlations when there are typically many factors in play.

1.2.1 Cancer Treatment and Efficacy

Among all cancer patients, more than half receive radiotherapy as their primary form of treatment [4]. Given that certain cancers can be sensitive to radiation, radiotherapy offers an effective and less invasive way to shrink tumors. Depending on the type of cancer, chemotherapy or other drug-related treatments may also be combined with radiation therapy. In 2016, radiochemotherapy was shown to achieve around 80% tumor control [6]. Since the overall goal is to minimize cancerous growths while maintaining healthy normal cells, the practice of combining multiple cancer treatments is essential to increasing cancer survival rates.

Within the past couple of decades, cancer research has dramatically increased the efficacy of cancer treatments and therefore survival rates. However, such treatments that are essential for recovery may have detrimental interactions with other drugs or conditions, namely depression. Those who are diagnosed with depression are typically prescribed SSRIs (Selective Serotonin Reuptake Inhibitor). Individually, anticancer drugs and SSRIs are effective. However, unwanted drug interactions may occur, causing potential toxicity and lowering of drug efficacy [4].

1.2 Problem

Although there is ample research to indicate the correlation between cancer diagnoses and MDD, current literature lacks a method for quantifying and standardizing this predisposition. Additionally, access to patient information can be limited based on publicly available data and therefore a streamlined process for testing this relationship is challenging.

1.3 Contributions

In this project, we aim to utilize publicly available PET images to classify depression progression rates for cancer patients. Before receiving radiotherapy as treatment, patients are required to complete a full-body PET scan. Since almost half of cancer patients receive radiotherapy, we propose using brain PET images to monitor depression progression. Our work specifically uses certain biomarkers found in PET images that have been proven to correlate to MDD. With our solution, healthcare professionals could better personalize treatment options for cancer patients who are more likely to be diagnosed with MDD, thus maximizing their recovery and overall well-being.

Chapter 2

Background and Significance

2.1 Imaging Modality Background

Positron Emission Tomography (PET) is a common imaging modality for cancer and depression detection, as it measures glucose metabolism. In order to perform a PET scan, the radiologist injects a radiotracer, a positron emitting radionuclide attached to a glucose molecule, into the patient's bloodstream [6]. Tissues absorb these radiotracers as they metabolize glucose. As the radionuclides decay, they emit positrons, which annihilate with electrons and emit gamma rays. Sensors known as gamma cameras, surrounding the patient, detect these gamma rays, localize the source, and develop an image [6]. A tissue with a low concentration of radiotracer results in a low intensity in the image. Similarly, a tissue with a high concentration of radiotracer results in a high intensity in the image. Tissues that consume a lot of glucose consume a proportional amount of radiotracer. Therefore, these tissues appear as high intensity areas in the image. This quality is useful for cancerous tumor detection, as the tumors consume a lot of glucose. In addition, this quality is useful for depression biomarker detection, as it captures brain structure metabolism [7]. Because the PET imaging modality captures information through cancerous tumor metabolism and brain metabolism, it is possible to derive information about each disease with one scan.

Radiologists already order a full body PET scan in order to plan radiotherapy. Because of the PET imaging properties previously discussed, these scans may capture information about

both cancer and depression biomarkers. Recent meta-analyses established a set of structural and functional changes on brain PET images characteristic of depression biomarkers [7,8,9] These studies isolated brain regions that showed consistent changes in untreated depressed patients, including: (1) consistent atrophy in the amygdala and (2) changes in glucose metabolism in the subgenual cingulate cortex (sACC) [9]. It may be possible, through these biomarkers, to identify patients at risk for developing depression, especially during radiotherapy treatment.

Identification of these patients at risk may allow doctors to plan an inclusive cancer treatment that incorporates mental health care in order to increase the efficacy of radiotherapy treatment in this patient population.

2.2 Relevant Work

Several algorithms to diagnose depression exist; however, these algorithms do not predict predisposition for depression. Recently, the World Health Organization (WHO) adopted a plan to address mental health and encouraged members of the tech industry to develop technologies to detect and treat mental health disorders [10]. The Association for Computing Machinery responded with the 2017 AVEC “Real-life Depression and Affect Recognition Workshop and Challenge.” In response to this challenge, Chlasta et al. implemented a convolutional neural network to diagnose depression based on audio samples [11]. The network learned patterns in the audio samples, characteristic of depression. However, patterns only appear when a patient already has depression. In other words, the algorithm only detects the clinical symptoms, not the pathological features. Therefore, the model cannot predict whether a patient will develop depression or not. Alhanai et al. implemented an LSTM model to diagnose depression from audio samples, which, as before, limited the anticipatory ability of the network [12]. Therefore, audio-based diagnosis methods lack the anticipatory ability necessary to predict whether a

subject is predisposed to developing depression or not.

Several algorithms to identify pathologies, such as Alzheimer's Disease (AD), through MRI imaging exist; however, MRI is not an optimal imaging modality for depression biomarker detection. Approaches for identifying AD biomarkers in structural MRI images often use convolutional neural networks (CNN). Farooq et al. classified AD by passing structural MRI images through ResNet, which we will explain in the next section [13]. The study achieved as high as a 98% accuracy with a ResNet-18 architecture. PET images share a similar format with MRI images; therefore, it is possible that this architecture will work for PET images as well [13]. Structural MRI images effectively capture structural information, but lack functional information, such as tissue metabolism. On the other hand, PET images capture tissue metabolism, as explained previously. Therefore, we may adopt a similar architecture and analyze functional images, which contain depression biomarkers.

2.3 ResNet

ResNet allows researchers to build deeper neural networks with improved accuracy. He et al. developed ResNet in 2015 [14]. Their work gained notice after their network won first place in the ILSVRC 2015 classification task [14]. ResNet is attractive as a CNN because it is deep. Depth only increases accuracy to a certain point in ordinary CNN's, known as the degradation problem [14]. ResNet overcomes this problem by implementing shortcut connections. The higher the number of layers of shortcut connections, the more accurate the network becomes and the less the amount of error rates. These shortcut connections allow the network to learn the residual, instead of a direct mapping between input and output. In theory, it is easier to optimize the residual mapping than the direct mapping [14]. These shortcut connections allow for deep architectures without the degradation problem.

2.4 Significance

As established in Chapter 1, MDD significantly decreases the efficacy of radiotherapy treatment and increases the mortality rate of cancer patients. It is necessary to anticipate depression development in order to plan mental health treatment as a part of overall cancer treatment. Supplementing radiotherapy with mental health treatment may increase efficacy and patient survival rate.

In 2010 a study conducted through the American Society of Clinical Oncology observed the effects of the supplemental treatment of depression with the treatment of cancer in relation to survival rates. A total of 64 of the 101 women that had metastatic breast cancer and depression symptoms in the study received a year of depression therapy along with their cancer therapy. This depression therapy is called supportive-expressive group therapy. After the start of the study, 4, 8 and 12 months, the Center for Epidemiologic Studies-Depression Scale was taken across 101 participants. Over 1 year that this study was conducted, a decreased (CES-D) score was able to increase a patient's longer survival rate. For women with decreasing CES-D scores over 1 year, overall median survival time was 53.6 months (n = 48), compared with 25.1 months for women with increasing scores (n = 53) [15]. This study is an example of how supplementing cancer therapy with mental health treatment could increase the efficacy and patient survival rate.

Chapter 3

Systems

3.1 Image Preprocessing

As mentioned in Chapter 2, PET images capture metabolic and volume changes in the brain. For this reason, we use brain PET images to identify depression biomarkers, established from prior research. Prior to training the algorithm on these images, we preprocessed, standardized, normalized, and labeled all of the images. Depending on the source of data, some PET images may be in a raw or processed form. Some processed images already standardize the image space, increase the resolution, and reduce noise. Thus, once obtaining the PET images, we must further standardize and normalize the images to improve our model's numerical stability. Additionally, we must have a method of determining if a patient is depressed or not that directly correlates to the PET image. Once we have this metric, we can then label each image with their respective diagnosis.

3.2 Deep Learning Architecture

After we acquired and preprocessed the images, we chose our algorithm architecture. As mentioned previously, we implemented a convolutional neural network, particularly a ResNet architecture. PET images contain three dimensions; therefore, we implemented deep convolutional layers that filtered across three dimensions. Next, we split our dataset into three portions: the training, validation, and test sets. The network only learned features from the

training set. The validation and test sets were reserved in order to evaluate the model's performance on unseen data.

3.3 Evaluation

Once we trained the algorithm on the testing set, we utilized evaluation methods, both quantitative and qualitative, to validate our results. In order to calculate the performance of the model, we compared the model's prediction to the actual outcome. Additionally, we continuously monitored the model's performance by plotting the progression of both the validation and training loss after every epoch. Finally, we utilized visual evaluation methods, which helped us identify the features, such as metabolic increases or volumetric decreases, that the model used to differentiate between positive and negative cases.

Chapter 4

Subsystems

4.1 Data Collection

We acquired brain PET images through the Alzheimer’s Disease Neuroimaging Initiative (ADNI). The ADNI study monitors neurodegeneration due to dementia and Alzheimer’s disease [16]. Study subjects periodically receive medical scans, such as PET scans. In addition, subjects answer a depression survey, the Geriatric Depression Scale (GDS), and receive a depression score, during their visit [17]. It is important to note that the images and GDS scores are taken at the same time; therefore, one image corresponds to one GDS score. We used these PET scans and GDS scores to train and test the neural network.

4.2 Subject Selection Criteria

We selected subjects to study through a strict set of criteria. Firstly, the subject received three or more PET scans. Secondly, the subject started the study without depression. The ADNI study sets a GDS score of six as the threshold for depression [17]. According to this threshold, a subject with a GDS score below six is not depressed; a subject with a GDS score of six or above is depressed. Thirdly, if the subject develops depression, they do not resolve. In other words, if the subject’s GDS rises above the threshold, it does not fall back below the threshold. The code for subject selection appears in Appendix A1.

4.3 Data Labeling

We assigned each patient a depression progression rate. The ADNI study provides GDS scores, magnitude of depression at a particular instant, not depression progression rates, rate of depression development over time. We obtained depression progression rates through a linear mixed-effects regression (LMER) model. The LMER model accounts for variance in the data due to random-effects, effects experienced in individual groups [18]. In this case, the subject is a random-effect. Therefore, there is variance in the data due to different subjects. We extracted the depression progression rates from the random-intercepts. A random-intercept of zero or below indicates that a subject does not develop depression over time; conversely, a random-intercept above zero indicates that a subject develops depression over time. As such, patients with a random-intercept of zero or below were labeled as nonprogressors, class 0, and patients with a random-intercept above zero were labeled as progressors, class 1. The code for generating labels appears in Appendix A2.

4.4 Data Augmentation

We standardized across all images. In addition, we normalized each image in order to prepare them for the neural network. Normalization limited the pixel values to a range between zero and one. In practice, normalization prevents pixel values from tending toward infinity as the image passes through the network.

Only 1428 images in the ADNI database met the selection criteria; in order to double the sample size, we performed data augmentation. We copied each image and slightly rotated each copy by five degrees in the transverse plane, as in Figure 1, a common data augmentation technique [19,20]. This resulted in 2856 usable images.

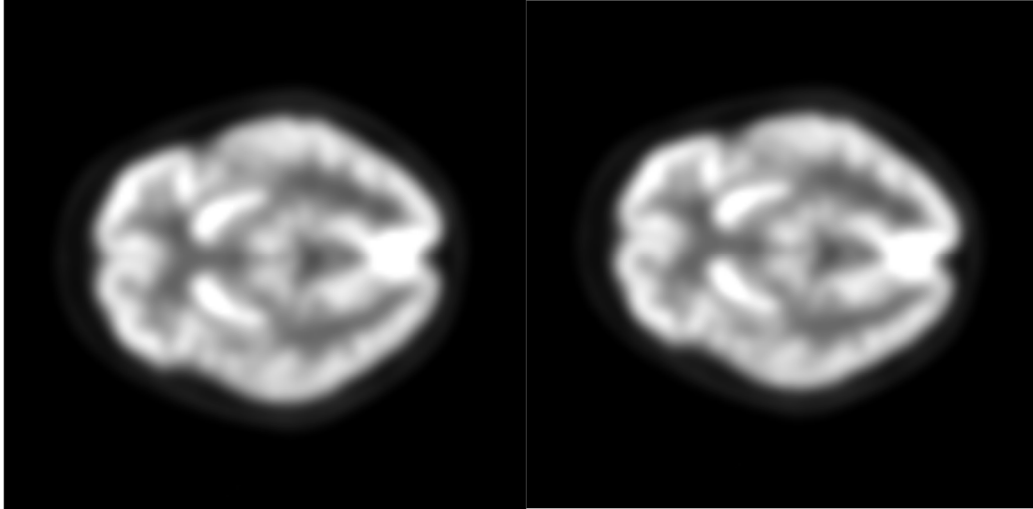


Figure 1: (Left) Original Image; (Right) Five degree rotation in the transverse plane.

Upon instantiation, we selected a specific subregion of the brain image. Recall from the background that the literature points to several specific brain structures as depression biomarkers, including the sACC and amygdala. The ADNI images contained null space around the brain. We removed the null space around the brain in order to reduce the image size from $96 \times 160 \times 160$ to $80 \times 120 \times 120$. After removing the null space, we downsampled the image. Essentially, we reduced the image resolution by a factor of two, which halved the image dimensions. Down-sampling allowed us to increase the batch size from two to sixteen. An increase in batch size increases training speed and improves model generalizability [21]. The code for data augmentation appears in Appendix A3.

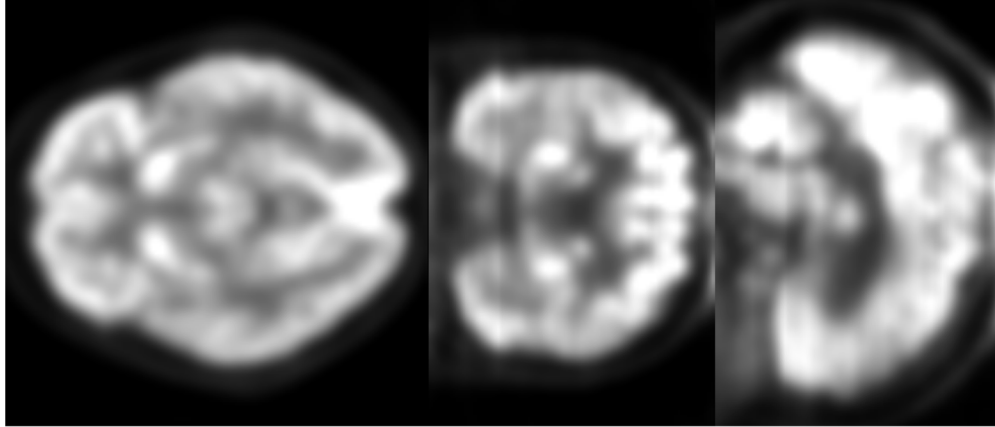


Figure 2: Cropped Image

4.5 Class Distribution

After subject selection and data augmentation, the dataset consisted of 2856 images; 63% of these images represented negative cases and 37% of these images represented positive cases. If we trained on this data distribution, we would achieve either a 63% accuracy or 37% by randomly guessing. In order to balance the class distribution and achieve a 50% accuracy by randomly guessing, we extracted a subset from the dataset. The subset included all positive cases and an equal number of negative cases. Therefore, the network would achieve a 50% accuracy through randomly guessing. Unfortunately, class balancing reduced the number of images available to train the network from 2856 to 2061 images. The code for subset generation appears in Appendix A4.

4.6 Dataset Split

After correcting the class distribution, we split the dataset into three subsets: training, validation, and testing. The training, validation, and testing subsets contained 70%, 10%, and 20% of images, respectively. This is a common heuristic in machine learning. We trained the neural network on the training dataset. Essentially, the neural network adjusted its weights, or learned

features, based on the training set. The neural network did not adjust its weights based on the validation or testing set. After every epoch, the neural network evaluated its performance on the validation set, which contained unseen images. Essentially, the neural network evaluated the generalizability of learned features on the validation set after every update. The testing dataset was reserved until after training. We used the testing dataset to evaluate the final model's performance on unseen images.

4.7 Deep Convolutional Neural Network

We implemented a deep convolutional neural network to filter the images for potential biomarkers. In particular, we implemented a ResNet34 with projection connections, which comprises thirty-four convolutional layers, three projection connections, one global average pooling layer, and one fully connected layer [14]. The original ResNet processes 2D images. In order to process 3D images, we replaced 2D convolutions with 3D convolutions. The ResNet architecture uses striding instead of pooling in order to reduce image resolution [14]. In addition, we did not stride at the first convolutional layer, as we did not want to prematurely downsize the already small input image. The fully connected layer comprises two neurons, which represent nonprogression and progression probabilities, respectively.

4.8 Statistical Evaluation

We quantitatively evaluated the model's performance through standard statistical methods. Specifically, we looked at the confusion matrix, calculated the sensitivity and specificity, and analyzed the receiver operating characteristic (ROC) curve.

4.9 Visual Evaluation

Additionally, we qualitatively evaluated the model through class activation mapping (CAM).

Several CAM methods exist, particularly weight-based [22] and gradient-based [23,24]. We implemented a gradient-based CAM, GradCAM, because it is compatible with a wider variety of architectures than weight-based CAM [23,24]. GradCAM visually captures weight changes, particularly weight increases. Through GradCAM, we visually inspected which features contributed to a class prediction.

Chapter 5

Systems-Integration, Tests, and Results

5.1 Training

We trained the neural network on the training set using stochastic gradient descent (SGD) with a learning rate of 0.001 and a momentum of 0.9, a common momentum value [21], for 140 epochs. As in Figure 3, the training loss decreased significantly until it plateaued at approximately 0.31. However, the validation loss diverged from the training loss at this time. This divergence suggested that the network overfit to the training set. In other words, the model did not generalize to unseen images. Figure 4 reaffirms this as the training accuracy increases to 100%, but the validation accuracy remains around or lower than 50% on average.

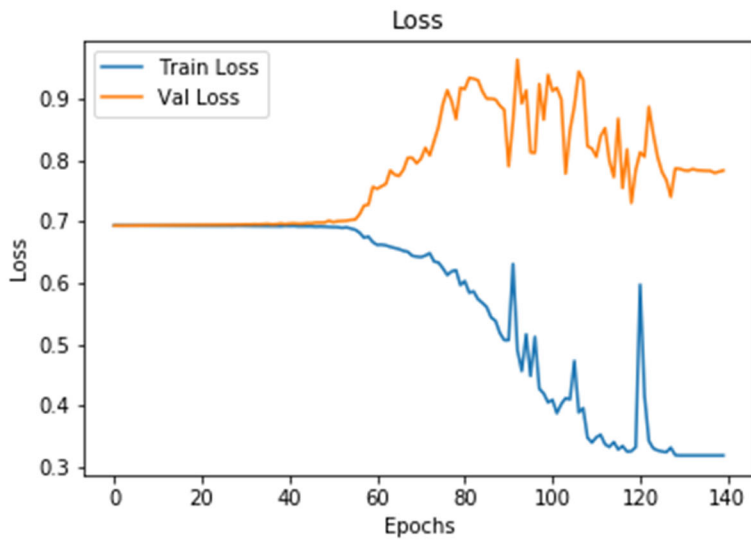


Figure 3: Loss

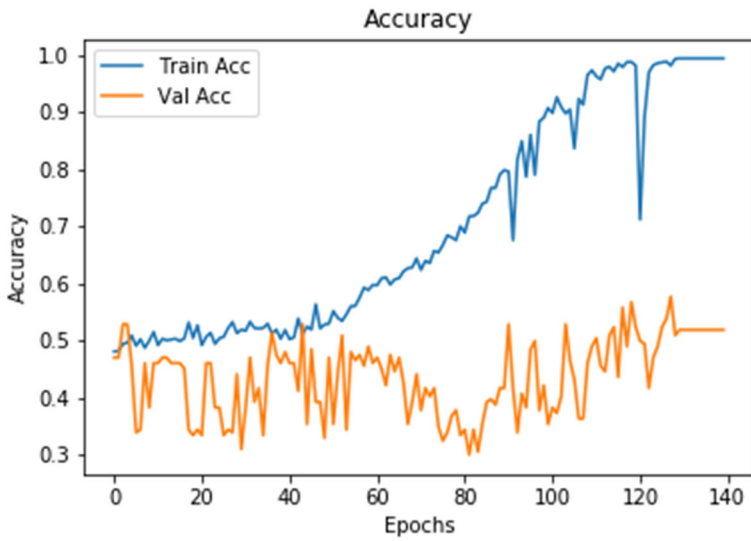


Figure 4: Accuracy

5.2 Statistical Evaluation

We loaded the weights from epoch 130, into the model and evaluated its performance on the testing set. We calculated the sensitivity and specificity of the model using the values from the following confusion matrix. The model only achieved a sensitivity of 56.25% and a specificity of

53.64%. These values suggest that the model was unable to discriminate between positive and negative cases in new cases. The model still randomly predicted the class in unseen cases. The ROC curve, Figure 5, is close to a 45 degree line to the horizontal, which reaffirms that the model randomly predicted the class in new cases.

Table 1: Confusion matrix representing the model's performance on unseen images.

Predicted/Actual	+	-
+	108	102
-	84	118

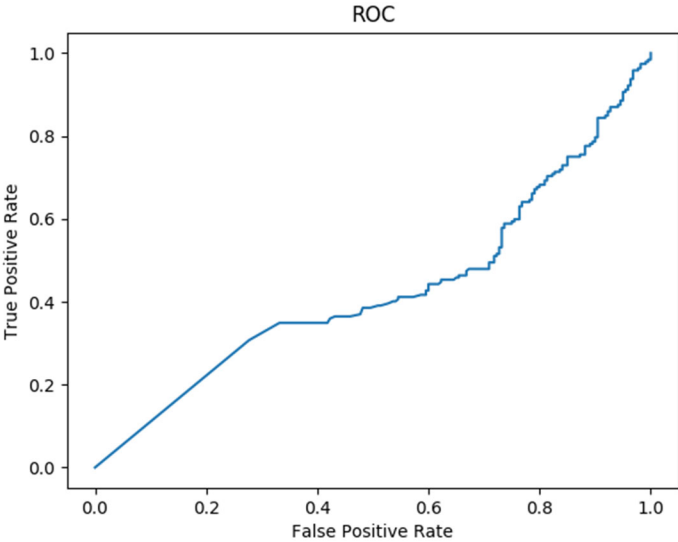


Figure 5: ROC Curve

5.3 Visual Evaluation

We visually inspected features that the model identified through GradCAM. For example, we generated a CAM in Figure 6, which was taken from a positive case. The CAM highlighted

regions of interest, including the sACC and the basal ganglia, characterized by light regions in Figure 6. Recall from the introduction that the sACC and basal ganglia are depression biomarkers according to the literature. Therefore, the model identified the sACC and basal ganglia in the image and used them to predict positive cases. We compared the positive case CAM to a negative case CAM. The negative case CAM, depicted in Figure 7, did not highlight the sACC or the basal ganglia. Therefore, the model did not use the sACC or the basal ganglia to predict negative cases.

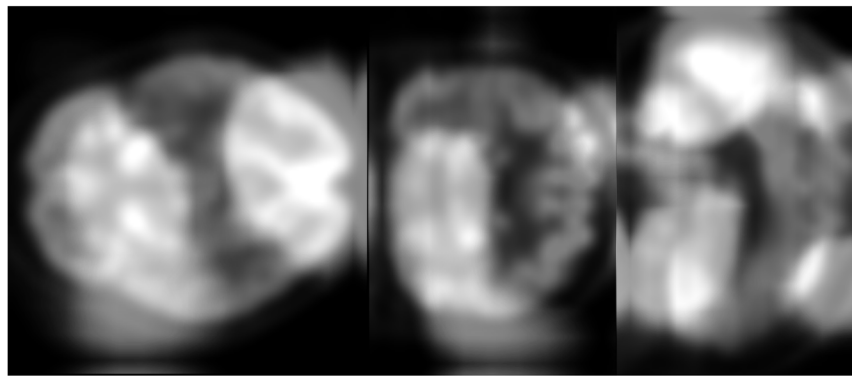


Figure 6: Positive case CAM

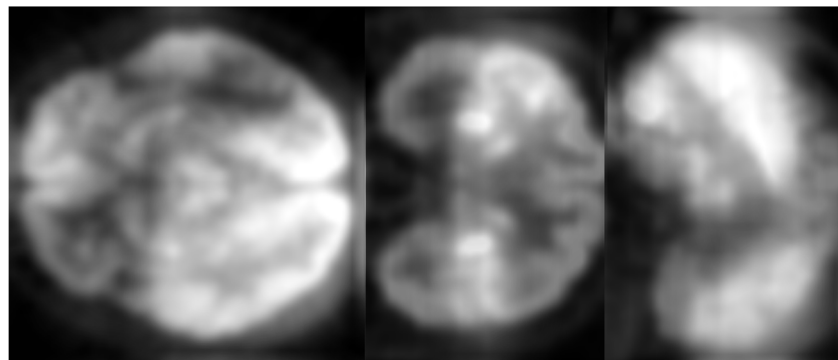


Figure 7: Negative Case CAM

Notice that the positive case CAM highlights the occipital region. Therefore, the occipital region contributed to a positive prediction. Indeed, the literature characterizes a phenomenon in the occipital lobe, known as occipital bending, as a MDD biomarker [25]. Occipital bending describes the phenomenon where one occipital lobe bends past the midline of the brain; a

decrease in functional connectivity may occur as well [26,27]. In rightward occipital bending, the left occipital lobe bends past the midline; in leftward occipital bending, the right lobe bends past the midline. The network may detect this occipital bending. However, the negative case CAM also highlights the occipital lobe. In their study, Fullard et al. observed that an equal number of MDD and normal subjects experienced right and left occipital bending [26]. Essentially, both MDD and normal subjects experience occipital bending. Therefore, it is present in both cases. The model focuses on occipital bending as a feature for identifying a positive case. However, the model may confuse negative cases as positive because they also contain occipital bending, contributing to false positives.

Chapter 6

Professional Issues & Constraints

6.1 Economics

One PET scan costs thousands of dollars [28]. Cancer patients already receive a PET scan as a part of their cancer treatment. In the future, we plan to analyze the brains from these PET images, which will not require additional scans and will not place an extra economic burden on cancer patients.

6.2 Health & Safety

As discussed in the introduction, PET imaging is a radiation-based imaging modality. Therefore, the PET procedure delivers gamma radiation to the recipient. Like x-ray radiation, gamma radiation may produce deterministic or stochastic effects. A high radiation dosage over a short duration kills tissue, or produces deterministic effects. Continuous radiation exposure over an extended duration damages DNA and encourages cancerous growth, or produces stochastic effects [6]. As discussed, cancer patients already receive a full body PET scan prior to cancer treatment. We plan to analyze the brains from these PET images. Therefore, our project does not require additional PET scans and does not expose cancer patients to more radiation than the typical cancer treatment process.

6.3 Ethics and Privacy

We analyze medical information. Because we analyze medical information, we must comply with the Health Insurance Portability and Accountability Act of 1996 (HIPAA). The HIPAA Privacy Rule mandates that covered entities protect individually identifiable health information, also called protected health information (PHI). PHI includes demographic data, physical health conditions, mental health conditions, medical treatments, medical expenses, name, address, birthdate, and social security number. The HIPAA Security Rule mandates that PHI is protected from unauthorized use or disclosure [29]. Electronically, this may take the form of encryption. However, we did not access PHI from ADNI. We only accessed de-identified health information (DHI). DHI removes all identifiers from the PHI [30]. We only used PET images, GDS scores, exam dates, and subject numbers, which do not individually identify the patient. Therefore, we did not use personal information and could not identify individuals in the real world.

6.4 Technology

Deep learning attracted attention as a method to detect features in and extract features from medical images in the medical imaging community. Recent deep learning models show promise as a diagnostic tool. However, these models are not entirely explainable. Model interpretability tools, such as GradCAM, are in development in order to increase the interpretability of deep learning models and explain their predictions [23].

6.5 Usability

The algorithm must be autonomous. The model must receive images from the PET scan, analyze them, and return a prediction autonomously. An image processing pipeline, such as this, provides several benefits. For example, clinicians do not need to learn how to use the machine learning

interface. In addition, the pipeline immediately analyzes and provides a prediction for the patient. Therefore, the clinician can immediately provide this information to the patient.

Chapter 7

Conclusion & Future Work

7.1 Conclusion

We implemented a deep learning algorithm to identify depression biomarkers, such as metabolic increases, in PET brain images and deliver a diagnosis: not prone to depression development or prone to depression development. The network achieved a 100% accuracy on the training dataset; however, the validation accuracy decreased. Therefore, the model overfit to the training images. Essentially, the model learned discriminative features in the training images.

Unfortunately, those features did not generalize to new cases. We visually evaluated these features using gradient-based class activation mapping. Class activation maps revealed that the network identified and used depression biomarkers, the sACC in particular, to classify patients prone to depression development. From the literature, we anticipated that the network would identify the sACC as a depression biomarker. The class activation map also revealed that the network identified the occipital lobe as a depression biomarker. After review, we suspected that the network detected a phenomenon known as occipital bending in depressed subjects. Although rightward occipital bending is more frequent in depressed subjects, normal subjects still experience occipital bending, just at a lower frequency. We suspected that occipital bending may have contributed to false positives, as the occipital lobe appeared in several false positive class activation maps. Therefore, we will take this into consideration in future research. Overall, the

model identified key depression biomarkers in the PET brain images; however, these features did not generalize to unseen images.

7.2 Future Work

We plan to improve network generalizability and performance through several techniques. Firstly, we will increase the sample size through additional data augmentation techniques, such as Gaussian blur, sheering, five degree rotation in the counterclockwise, not just clockwise, direction, and so on. Secondly, we will preprocess over saturated images in our data set. These oversaturated images, as in Appendix C, bias the network filters during training. In order to reduce the number of oversaturated images in the dataset, we may use histogram truncation, where we trim the tails of the image histogram, the extreme pixel values, from the image. We may remove these oversaturated images from the dataset altogether, as well. Lastly, we plan to increase the GradCAM resolution. The current GradCAM produces low resolution heat maps, which makes it more difficult to detect the regions of interest with the naked eye. In the future, we will implement high resolution GradCAM methods, such as Res-3D-Grad-CAM-Shallow, which will crisply reveal regions of interest. Through these techniques, we hope to improve the generalizability, performance, and interpretability of the deep learning algorithm and provide an effective diagnostic tool to inform cancer treatment options for subjects prone to depression development.

Bibliography

- [1] R. L. Siegel, K. D. Miller and A. J. Jemal, "Cancer statistics, 2019," *CA: A Cancer Journal for Clinicians*, vol. 69, no. 1, pp. 7-34, 2019.
- [2] H. R. Smith, "Depression in cancer patients: Pathogenesis, implications and treatment (Review)," *Oncology letters*, vol. 9, no. 4, pp. 1509-1514, 2015.
- [3] M. Pinguart and P.R. Duberstein, "Depression and cancer mortality: a meta-analysis," *Psychol Med*, vol. 40(11), pp. 1797 - 1810, Nov. 2010.
- [4] J. D. Newport and C. B. Nemeroff, "Assessment and treatment of depression in the cancer patient," *Journal of Psychosomatic Research*, vol. 45, no. 3, pp. 215-237, 1998.
- [5] Y. Wang, J. Li, J. Shi, J. Que, J. Liu, J. M. Lappin, J. Leung, A. V. Ravindran, W. Chen, Y. Qiao, J. Shi, L. Lu, and Y. Bao, "Depression and anxiety in relation to cancer incidence and mortality: a systematic review and meta-analysis of cohort studies," *Molecular Psychiatry*, Nov. 2019, doi: <http://doi.org/10.1038/s41380-019-0595-x>.
- [6] N. B. Smith and A. Webb, "Nuclear medicine: Planar scintigraphy, SPECT and PET/CT," *Introduction to Medical Imaging: Physics, Engineering and Clinical Applications*, Cambridge, United Kingdom, Cambridge University Press, 2011, ch. 3.
- [7] L. Su, Y. Cai, Y. Xu, A. Dutt, S. Shi, and E. Bramon, "Cerebral metabolism in major depressive disorder: a voxel-based meta-analysis of positron emission tomography studies," *BMC Psychiatry*, vol. 14:321, Nov. 2014.
- [8] J. B. Savitz and W. C. Drevets, "IMAGING PHENOTYPES OF MAJOR DEPRESSIVE DISORDER: GENETIC CORRELATES," *Neuroscience*, vol. 164, pp. 300 - 330, 2009, doi:10.1016/j.neuroscience.2009.03.082.

- [9] J. Sacher, J. Neumann, T. Fünfstück, A. Soliman, A. Villringer and M. L. Schroeter, "Mapping the depressed brain: A meta-analysis of structural and functional alterations in major depressive disorder," *Journal of Affective Disorders*, vol. 140, no. 2, pp. 142-148, 2012.
- [10] World Health Organization (WHO), *Mental health action plan 2013-2020*, World Health Organization, 2013.
- [11] K. Chlasta, K. Wolk and I. Kreitz, "Automated speech-based screening of depression using deep convolutional networks," *Procedia Computer Science*, vol. 164, p. 618–628, 2019.
- [12] T. Alhanai, M. Ghasseni and J. Glass, "Detecting Depression with Audio/Text Sequence Modeling of Interviews," *Interspeech*, vol. 2522, pp. 1716-1720, 2018.
- [13] A. Farooq, S. Anwar, M. Awais and S. Rehman, "A deep CNN based multi-class classification of Alzheimer's disease using MRI," in *IEEE International Conference on Imaging systems and techniques (IST)*, 2017.
- [14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," Dec. 2015, arXiv:1512.03385v1.
- [15] K. Collie, J. Giese-Davis, H. Kraemer, E. Neri, K. Rancourt, "Decrease in Depression Symptoms is Associated with Longer Survival in Patients with Metastatic Breast Cancer: A Secondary Analysis," in *Journal of clinical oncology : official journal of the American Society of Clinical Oncology* vol. 29.4, pp. 413-420, 2011.
- [16] *ADNI*, LONI Image Data Archive. [Online]. Available:
<https://ida.loni.usc.edu/login.jsp?project=ADNI>.
- [17] "ADNI_GeneralProceduresManual," Available: http://adni.loni.usc.edu/wp-content/uploads/2010/09/ADNI_GeneralProceduresManual.pdf.

- [18] D. Bates, M. Maechler, B. Bolker, S. Walker, R. H. B. Christensen, H. Singmann, B. Dai, and F. Scheipl, "Package 'lme4'"
- [19] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning Augmentation Strategies from Data," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 113-123.
- [20] Z. Hussain, F. Gimenez, D. Yi, and D. Rubin, "Differential Data Augmentation Techniques for Medical Imaging Classification Tasks," *AMIA Annual Symposium Proceedings Archive*, pp. 978 - 987, Apr. 2018.
- [21] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," Sept. 2012, arXiv:1206.5533v2.
- [22] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, A. Torralba, "Learning Deep Features for Discriminative Localization," *CVPR*, pp. 2921 - 2929.
- [23] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," Dec. 2019, arXiv:1610.02391v4.
- [24] S. Ulyanin, "Implementing Grad-CAM in PyTorch," *Medium*, Feb. 2019, <https://medium.com/@stepanulyanin/implementing-grad-cam-in-pytorch-ea0937c31e82>.
- [25] J. J. Maller, R. H. S. Thomson, J. V. Rosenfeld, R. Anderson, Z. J. Daskalakis, and P. B. Fitzgerald, "Occipital bending in depression," *Brain: A Journal of Neurology*, vol. 137, pp. 1830 - 1837, Feb. 2014, doi:10.1093/brain/awu072.
- [26] K. Fullard, J. J. Maller, T. Welton, M. Lyon, E. Gordon, S. H. Koslow, and S. M. Grieve, "Is occipital bending a structural biomarker of risk for depression and sensitivity to

- treatment?” *Journal of Clinical Neuroscience*, vol. 63, pp. 55 - 61, Feb. 2019, doi:
<https://doi.org/10.1016/j.jocn.2019.02.007>.
- [27] C. Teng, J. Zhou, H. Ma, Y. Tan, X. Wu, C. Guan, H. Qiao, J. Li, Y. Zhong, C. Wang, and N. Zhang, “Abnormal resting state activity of left middle occipital gyrus and its functional connectivity in female patients with major depressive disorder,” *BMC Psychiatry*, vol. 18:370, 2018, doi: <https://doi.org/10.1186/s12888-018-1955-9>.
- [28] “PET Scans After Cancer Treatment.” Choosing Wisely: An Initiative of the ABIM Foundation. <https://www.choosingwisely.org/patient-resources/pet-scans-after-cancer-treatment/> (accessed May 20, 2020).
- [29] “Summary of the HIPAA Security Rule.” HHS.gov. <https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html> (accessed May 20, 2020)
- [30] Medical Imaging System Providing Disease Prognosis, by V. K. Ithapu, S. C. Johnson, O. C. Okonkwo. (2017, June 27). *US20160073969A1*. Accessed: May. 27, 2020 [Online]. Available:
<https://patents.google.com/patent/US20160073969?q=diagnostics+of+brain+imaging+using+machine+learning+approaches>
- [31] Method and System for Automated Brain Tumor Diagnosis Using Image Classification, by S. Bhattacharya, T. Chen, A. Kamen, S. Sun, S. Wan. (2016, October 6). *WO2016160491A1*. Accessed on: May 27, 2020 [Online]. Available:
<https://patents.google.com/patent/WO2016160491A1/en?q=medical+imaging+diagnostics+of+brain+using+machine+learning>
- [32] Systems and Methods for Brain Hemorrhage Classification in Medical Images Using an Artificial Intelligence Network, by S. Do, G. Gonzalez, M. Lev. (2019, March 14).

WO2019051271A1. Accessed on: May 27, 2020 [Online]. Available:

<https://patents.google.com/patent/WO2019051271A1/en>

Appendix A: Materials and Cost

We used the software and hardware listed in tables 1 and 2, respectively. The software was open-source and readily accessible.

Table 2: Software

Product	Cost
Python 3	N/A
Pip	N/A
NumPy	N/A
Scikit-Learn	N/A
Scikit-Image	N/A
PyTorch	N/A

Table 3: Hardware

Product	Cost
2 x NVIDIA GeForce GTX Titan X	\$2000

Appendix B1: Subject Selection Code

```
import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import json

root = os.getcwd()
fname = "GDSCALE.csv"
fpath = os.path.join(root, fname)

# Read into Pandas DataFrame, denoted as df
df = pd.read_csv(fpath, na_values = "-4")

# Cleaning
df.drop(labels = ["ID", "SITEID", "VISCODE", "VISCODE2", "USERDATE2",
"EXAMDATE", "GDUNABL", "GDUNABSP", "update_stamp"], axis = 1, inplace = True)
df.dropna(axis = 0, inplace = True)
df.reset_index(drop = True, inplace = True)

def to_years(date, delimiter = '/'):
    month, day, year = date.split(delimiter)
    m = float(month)*(1.0/12.0)
    d = float(day)*(1.0/365.0)
    y = float(year) + (2000.0)
    return (m + d + y)

class subject():
    def __init__(self, rid):

        self.rid = rid
        self.startdate = 0.0
        self.num_samples = 0
        self.meets_criteria = False

        # Exam dates and corresponding exam scores
        self.examdates = []
        self.examscores = []

    def append(self, date, score):
        self.examdates.append(date)
        self.examscores.append(score)
        self.num_samples += 1

    # Sort in chronological order
```

```

def sort(self):
    # Insertion Sort
    i = 1
    while(i < len(self.examdates)):
        j = i
        while(j >= 1):
            if(self.examdates[j] < self.examdates[j-1]):
                # Examdates
                temp = self.examdates[j]
                self.examdates[j] = self.examdates[j-1]
                self.examdates[j-1] = temp
                # Scores
                temp = self.examscores[j]
                self.examscores[j] = self.examscores[j-1]
                self.examscores[j-1] = temp
            j -= 1
        i += 1
    if(0 < len(self.examdates)):
        self.startdate = self.examdates[0]

# Normalize by the first year
# Use after sorting
def normalize(self):
    if(0 < len(self.examdates)):
        for i in range(len(self.examdates)):
            self.examdates[i] -= self.startdate

def get_years(self):
    return np.asarray(self.examdates)

def get_scores(self):
    return np.asarray(self.examscores)

def get_num_samples(self):
    return self.num_samples

def get_startdate(self):
    return self.startdate

def check_criteria(self, thresh = 6):
    meets_thresh = False
    rise_and_fall = False
    for i in range(len(self.examscores)):
        if(thresh <= self.examscores[i]):
            meets_thresh = True
        if(True == meets_thresh and thresh > self.examscores[i]):

```

```

        rise_and_fall = True
    # Includes both cases, where one stays under the threshold or continuously increases
    if(not meets_thresh and not rise_and_fall):
        self.meets_criteria = True
    if(meets_thresh and not rise_and_fall):
        self.meets_criteria = True
    return self.meets_criteria

# Hash table of subjects
subjects = [subject(i) for i in range(np.nanmax(np.asarray(df["RID"], dtype = np.int32)) + 1)]

for i in range(len(df)):
    subjects[np.int32(df["RID"][i]).append(np.float32(to_years(df["USERDATE"][i])),
np.float32(df["GDTOTAL"][i]))

for i in range(len(subjects)):
    subjects[i].sort()

for i in range(len(subjects)):
    subjects[i].normalize()

# Remove subjects with less than three data points
subset = []
for i in range(len(subjects)):
    if(3 > subjects[i].num_samples):
        for j in range(len(df)):
            if(i == df["RID"][j]):
                subset.append(j)

df.drop(axis = 0, index = subset, inplace = True)
df.reset_index(drop = True, inplace = True)

# Remove subjects that do not meet the criteria
subset = []
for i in range(len(subjects)):
    if(not subjects[i].check_criteria()):
        for j in range(len(df)):
            if(i == df["RID"][j]): # If the RID matches
                subset.append(j) # Append the row index

df.drop(axis = 0, index = subset, inplace = True)
df.reset_index(drop = True, inplace = True)

# Get start dates
startdates = [0.0 for i in range(len(subjects))]
for i in range(len(subjects)):

```

```

if(0 < len(subjects[i].get_years())):
    startdates[i] = subjects[i].get_startdate()

# Convert dates to years in df
df["Years"] = df["USERDATE"]
for i in range(len(df)):
    df["Years"][i] = to_years(df["USERDATE"][i]) - startdates[np.int32(df["RID"][i])]

for i in range(len(df)):
    if(0 > df["Years"][i]):
        df["Years"][i] = 0

# Write to csv file
df.to_csv("patients_that_meet_criteria.csv")

#####
# Transfer to R to make lmer model
# After modeling in R, transfer back

#####
# Create Labels
fname = "rates.csv"
fpath = os.path.join(root, fname)
df1 = pd.read_csv(fpath)

df1.head()

rids = []
rates = []
lbls = []
for i in range(len(df1)):
    rids.append(str(df1["Unnamed: 0"][i]))
    rates.append(df1["Years"][i])
    if(0 > df1["Years"][i]):
        lbls.append(0)
    else:
        lbls.append(1)

# Reformat rids
for i in range(len(rids)):
    while(4 > len(rids[i])):
        rids[i] = '0' + rids[i]

lbls_dict = dict(zip(rids, lbls))

fname = "revised_labels.json"

```

```
with open(fname, 'w') as fp:  
    json.dump(lbls_dict, fp)
```

Appendix B2: Labeling Code in R

```
# Depression rates for patients who develop depression only

library('lme4')
library('ggplot2')

# Load data
fname <- '~/Documents/SCU/SeniorDesign/patients_that_meet_criteria.csv'
fd <- read.csv(fname)
summary(fd)

# Initialize model
model <- lmer(GDTOTAL ~ Years + (Years|RID), fd)
summary(model)

# Extract individual slopes
slopes <- ranef(model)$RID

# Save output
fname <- "~/Documents/SCU/SeniorDesign/rates.csv"
write.csv(slopes, file = fname)
```

Appendix B3: Dataset Augmentation Code

```
import os
import numpy as np
from glob import glob
import nibabel as nib
from imgaug import augmenters as iaa
import imgaug as ia

cwd = os.getcwd()
contents = list(glob("norm_classes/*/*"))
output = "norm_classes-aug"

seq = iaa.Sequential([
    iaa.Affine(rotate=(5))
])

for i in range(len(contents)):
    img = np.load(contents[i])
    pat_id = contents[i].split('/')[-2]
    date = contents[i].split('/')[-1].split('.')[-2] + '_aug'
    path = os.path.join(output, pat_id)
    path2 = os.path.join(path, date) + '.npy'
    if os.path.exists(path) == 0:
        os.makedirs(path)
    if os.path.exists(path2) == 0:
        try:
            augd = seq(images=img)
            np.save(path2, augd)
        except:
            print('error adding:')
            print(pat_id, date)
```

Appendix B4: Dataset Code

```
import os
import sys
import json
import numpy as np
from sklearn import preprocessing
import skimage
from skimage.transform import downscale_local_mean
import torch
from torch.utils.data import Dataset

class v3(Dataset):
    def __init__(self, root_fpath, json_fpath, generate_subset = False, add_noise = False):
        super(v3, self).__init__()

        self.root = root_fpath
        self.children = os.listdir(self.root)
        self.generate_subset = generate_subset
        self.add_noise = add_noise

        self.imgs = [] # Individual image filepaths
        self.labels = [] # Labels per each image
        self.subjects = [] # Subject rid

        self.subset_imgs = []
        self.subset_labels = []
        self.subset_subjects = []

        self.progressors = 0
        self.nonprogressors = 0

        self.progressor_imgs = 0
        self.nonprogressor_imgs = 0

        self.subset_progressors = 0
        self.subset_nonprogressors = 0

        self.subset_progressor_imgs = 0
        self.subset_nonprogressor_imgs = 0

        with open(json_fpath, 'r') as fp:
            self.lbls_dict = json.load(fp)

        # As of v3, the root directory must contain a subdirectory for each patient
        # Ex: 0003, 0005, 0008, and so on
```



```

for i in range(len(self.children)):
    lbl = self.lbls_dict[self.children[i]]
    # On the subject level
    if(1 == lbl):
        self.progressors += 1
    else:
        self.nonprogressors += 1
    child = os.path.join(self.root, self.children[i])
    files = os.listdir(child)
    for j in range(len(os.listdir(child))):
        # On the sample level
        if(1 == lbl):
            self.progressor_imgs += 1
        else:
            self.nonprogressor_imgs += 1
        file = os.path.join(child, files[j])
        self.imgs.append(file)
        self.labels.append(lbl)
        self.subjects.append(self.children[i])

if(self.generate_subset):
    max_ = np.max(np.asarray(self.children, dtype = np.int32))
    represented = np.zeros(max_+1) # Hash table for represented subjects using subject rids
    switch = 1 # Switch between positive and negative cases, starting with positive cases
    np.random.seed(0)
    while(self.progressor_imgs > self.subset_progressor_imgs
          and self.progressor_imgs > self.subset_nonprogressor_imgs):
        idx = np.random.choice(len(self.children)) # Randomly select subject
        lbl = self.lbls_dict[self.children[idx]] # Grab label for subject
        if(switch):
            if(1 == lbl and 0 == represented[int(self.children[idx])]):
                self.subset_progressors += 1
                child = os.path.join(self.root, self.children[idx])
                files = os.listdir(child)
                for j in range(len(os.listdir(child))):
                    # On the sample level
                    self.subset_progressor_imgs += 1
                    file = os.path.join(child, files[j])
                    self.subset_imgs.append(file)
                    self.subset_labels.append(lbl)
                    self.subset_subjects.append(self.children[idx])
                represented[int(self.children[idx])] += 1
                switch = 0
            else:
                if(0 == lbl and 0 == represented[int(self.children[idx])]):
                    self.subset_nonprogressors += 1

```

```

        child = os.path.join(self.root, self.children[idx])
        files = os.listdir(child)
        for j in range(len(os.listdir(child))):
            # On the sample level
            self.subset_nonprogressor_imgs += 1
            file = os.path.join(child, files[j])
            self.subset_imgs.append(file)
            self.subset_labels.append(lbl)
            self.subset_subjects.append(self.children[idx])
        represented[int(self.children[idx])] += 1
        switch = 1

def __len__(self):
    if(False == self.generate_subset):
        return self.progressor_imgs + self.nonprogressor_imgs
    else:
        return self.subset_progressor_imgs + self.subset_nonprogressor_imgs

def __getitem__(self, idx):
    # Note : Normalize and standardize images before saving in order to expedite loading
    if(False == self.generate_subset):
        img = np.load(self.imgs[int(idx)]).astype(np.float32)
        if(self.add_noise):
            img = skimage.util.random_noise(img, mode = "gaussian", seed = None)
        cube_shape = (80, 120, 120)
        img_cube = self._get_cube(img, cube_shape)

        label = torch.Tensor([self.labels[int(idx)]).long() # Compatible with CrossEntropyLoss()
        subject = self.subjects[int(idx)]
    else:
        img = np.load(self.subset_imgs[int(idx)]).astype(np.float32)
        if(self.add_noise):
            img = skimage.util.random_noise(img, mode = "gaussian", seed = None)
        cube_shape = (80, 120, 120)
        img_cube = self._get_cube(img, cube_shape)

        label = torch.Tensor([self.subset_labels[int(idx)]).long()
        subject = self.subset_subjects[int(idx)]

    # Downsample image resolution in order to increase batch size
    img_cube = downscale_local_mean(img_cube, (2,2,2))

    img_cube = torch.Tensor(img_cube).float() # Convert to torch.float32

    img_cube = torch.unsqueeze(img_cube, dim = 0) # Insert channel dimension
    img_cube.requires_grad = True

```

```

return (img_cube, label)

def _get_cube(self, img, cube_shape):
    depth_offset = (img.shape[0] - cube_shape[0])//2
    width_offset = img.shape[1]//8
    height_offset = img.shape[2]//8
    img_cube = np.zeros(cube_shape)
    for i in range(cube_shape[0]):
        for j in range(cube_shape[1]):
            for k in range(cube_shape[2]):
                img_cube[i][j][k] = img[depth_offset + i][width_offset+j][height_offset+k]
    return img_cube

def __repr__(self):
    if(False == self.generate_subset):
        print("Progressors : Nonprogressors")
        print("%d : %d" % (self.progressors, self.nonprogressors))
        print("Progressor_imgs : Nonprogressor_imgs")
        print("%d : %d" % (self.progressor_imgs, self.nonprogressor_imgs))
    else:
        print("Progressors : Nonprogressors")
        print("%d : %d" % (self.subset_progressors, self.subset_nonprogressors))
        print("Progressor_imgs : Nonprogressor_imgs")
        print("%d : %d" % (self.subset_progressor_imgs, self.subset_nonprogressor_imgs))
    return str(self.__class__)

# For splitting indices
# Split indices and load into PyTorch Sampler
# Load PyTorch Sampler into dataloader
def split(length, val_prct, test_prct):
    train_sampler = np.random.permutation(int(length - (length*val_prct) - (length*test_prct)))
    val_sampler = np.random.permutation(int(length*val_prct))
    test_sampler = np.random.permutation(int(length*test_prct))
    # Make sure subjects do not appear in more than one subset
    for i in range(len(val_sampler)):
        val_sampler[i] += len(train_sampler)
    for i in range(len(test_sampler)):
        test_sampler[i] += len(train_sampler) + len(val_sampler)
    return train_sampler, val_sampler, test_sampler

def check_distro(dataLoader):
    class0 = 0
    class1 = 0
    for i, data in enumerate(dataLoader):
        imgs, lbls = data

```

```
for j in range(len(lbls)):
    if(0 == lbls[j]):
        class0 += 1
    else:
        class1 += 1
return class0, class1
```

Appendix B5: Deep Learning Algorithm

```
#####
import os

#####
# PyTorch modules
import torch
import torch.nn as nn
from torch.nn import Conv3d
from torch.nn import BatchNorm3d
from torch.nn import ReLU
from torch.nn import AdaptiveAvgPool3d
from torch.nn import Linear
import torch.nn.functional as F

#####
# Metrics modules
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

#####
# Image modules
import skimage
import skimage.transform

#####
# Medical Image Formatting
import nibabel as nib

#####
# ResNet34-Based Architecture with adjustments for 3D images and CAM compatability

class v2(nn.Module):

    def __init__(self):
        super(v2, self).__init__()

        # Width parameter in Deep Double Descent by Nakkiran et al.
        k = 64

        # Skip the first convolution, which reduces img size
        # self.conv11 = Conv3d(1, 64, 3, 2, 0) # double filter dimension, halve img dimensions
```

```

# TODO : Split into feature extractor and classifier

self.conv11 = Conv3d(1, k, 3, 1, 1) # Increase filter dimension

self.block11 = self._basic(k, 3, 1, 1)
self.block12 = self._basic(k, 3, 1, 1)
self.block13 = self._basic(k, 3, 1, 1)

self.conv21 = Conv3d(k, 2*k, 3, 2, 0) # increase filter dimension, halve img dimensions
self.conv22 = Conv3d(2*k, 2*k, 3, 1, 1)

self.proj21 = Conv3d(k, 2*k, 3, 2, 0) # increase upstream img dimensionality to match
downstream img dimensionality

self.block21 = self._basic(2*k, 3, 1, 1)
self.block22 = self._basic(2*k, 3, 1, 1)
self.block23 = self._basic(2*k, 3, 1, 1)

self.conv31 = Conv3d(2*k, 4*k, 3, 2, 0) # increase filter dimension, halve img dimensions
self.conv32 = Conv3d(4*k, 4*k, 3, 1, 1)

self.proj31 = Conv3d(2*k, 4*k, 3, 2, 0)

self.block31 = self._basic(4*k, 3, 1, 1)
self.block32 = self._basic(4*k, 3, 1, 1)
self.block33 = self._basic(4*k, 3, 1, 1)
self.block34 = self._basic(4*k, 3, 1, 1)
self.block35 = self._basic(4*k, 3, 1, 1)

self.conv41 = Conv3d(4*k, 8*k, 3, 2, 0) # increase filter dimension, halve img dimensions
self.conv42 = Conv3d(8*k, 8*k, 3, 1, 1)

self.proj41 = Conv3d(4*k, 8*k, 3, 2, 0)

self.block41 = self._basic(8*k, 3, 1, 1)
self.block42 = self._basic(8*k, 3, 1, 1)
self.block43 = self._basic(8*k, 3, 1, 1)

self.pool = AdaptiveAvgPool3d(1) # Output : (D, H, W) : (1, 1, 1)

# self.linear1 = Linear(512, 256, bias = False) # Arbitrary decision for hidden neurons
# self.linear2 = Linear(256, 128, bias = False)
# self.linear3 = Linear(128, 2, bias = False)
self.linear3 = Linear(8*k, 2, bias = False)
# self.linear3 = Linear(128, 2, bias = False) # 1*1*1*num_channels

```

```

# Input shape is the number of channels of the output

# Variable to store the gradients from the previous convolution
self.gradients = None

# Variable to store activation maps from the previous convolution
self.activations = None

def forward(self, img):

    img = self.conv11(img)

    # Begin solid skip connections
    img = F.relu(self.block11(img) + img)
    img = F.relu(self.block12(img) + img)
    img = F.relu(self.block13(img) + img)
    # End solid skip connections

    # ---- #

    # Begin dotted skip connection
    temp = F.relu(self.conv21(img))
    temp = F.relu(self.conv22(temp))
    img = self.proj21(img)
    img = img + temp
    # End dotted skip connection

    # Begin solid skip connections
    img = F.relu(self.block21(img) + img)
    img = F.relu(self.block22(img) + img)
    img = F.relu(self.block23(img) + img)
    # End solid skip connections

    # ---- #

    # Begin dotted skip connection
    temp = F.relu(self.conv31(img))
    temp = F.relu(self.conv32(temp))
    img = self.proj31(img)
    img = img + temp
    # End dotted skip connection

    # Begin solid skip connections
    img = F.relu(self.block31(img) + img)
    img = F.relu(self.block32(img) + img)
    img = F.relu(self.block33(img) + img)

```

```

img = F.relu(self.block34(img) + img)
img = F.relu(self.block35(img) + img)
# End solid skip connections

# ---- #

# Begin dotted skip connection
temp = F.relu(self.conv41(img))
temp = F.relu(self.conv42(temp))
img = self.proj41(img)
img = img + temp
# End dotted skip connection

# Begin solid skip connections
img = F.relu(self.block41(img) + img)
img = F.relu(self.block42(img) + img)
img = F.relu(self.block43(img) + img)
# End solid skip connections

# For GradCAM
if((not self.training) and img.requires_grad):
    # Grab the activation maps
    self.activations = img

    # Hook the gradients after the last convolutional layer
    handle = img.register_hook(self._hook)
    # register_hook is a tensor method
    # register_hook takes a function as an argument to grab the gradients
    # register_hook returns a handle?

# Pooling
pool = self.pool(img)
pool = pool.view(img.shape[0], -1) # Flatten
# N x C

# logits = self.linear1(pool)
# logits = self.linear2(logits)
# logits = self.linear3(logits)
logits = self.linear3(pool)
probs = F.softmax(logits, dim = 1) # Softmax applied across columns
return probs

#####
# from Kaiming He's initial publication on residual networks
def _basic(self, channels, kernel_size = 3, stride = 1, padding = 0):
    # If you want to maintain the input image dimensions, set padding = (kernel_size - 1)//2

```



```

        return nn.Sequential(Conv3d(channels, channels, kernel_size, stride, padding), #
Conv3d(in_channels, out_channels, kernel_size, stride, padding)
            ReLU(), # Do I need this ReLU?
            Conv3d(channels, channels, kernel_size, stride, padding)) # maintain image
dimensions via padding = (kernel_size-1)//2

    def _bottleneck(self, io_channels, intermediary_channels, kernel_size = 1, stride = 1, padding
= 0):
        # If you want to maintain the input image dimensions, set padding = (kernel_size - 1)//2
        return nn.Sequential(Conv3d(io_channels, intermediary_channels, kernel_size, stride,
padding),
            ReLU(),
            Conv3d(intermediary_channels, intermediary_channels, kernel_size, stride,
padding),
            ReLU(),
            Conv3d(intermediary_channels, io_channels, kernel_size, stride, padding))

#####
# Implemented with batch normalization
def _basic_bn(self, channels, kernel_size = 1, stride = 1, padding = 0):
    # If you want to maintain the input image dimensions, set padding = (kernel_size - 1)//2
    return nn.Sequential(Conv3d(channels, channels, kernel_size, stride, padding),
        BatchNorm3d(channels),
        ReLU(),
        Conv3d(channels, channels, kernel_size, stride, padding),
        BatchNorm3d(channels))

#####
# Hook to grab gradients during GradCAM
def _hook(self, grad):
    self.gradients = grad

#####
# Network information
def __repr__(self):

    state_dict = self.state_dict()
    layers = list(state_dict.keys())
    params = list(state_dict.values())

    num_params = 0

    for i in range(0, len(params)):
        count = 1
        for j in range(0, len(params[i].shape)):
            count *= params[i].shape[j]

```

```

        num_params += count
        # print(str(layers[i]) + " : " + str(params[i].shape))

print("Number of Parameters : %d\n" % num_params)

return str(self.__class__)

#####
# These functions must accept the model as an argument in order to be compatible with
nn.DataParallel
# It is assumed that the model is wrapped in nn.DataParallel prior to using these functions

#####
# Save current model
def save_DataParallel(model, path):
    # Save DataParallel model generically, so I can load to any device desired
    torch.save(model.module.state_dict(), path)

#####
# Load previous model
def load(model, path):
    model.load_state_dict(torch.load(path))

#####
# Single prediction / set of predictions
def predict(model, sample, device):
    probs = model(sample.to(device))
    preds = torch.argmax(probs, dim = 1)
    return probs, preds

def _train_step(model, train_loader, device, loss_func, optimizer):
    running_train_loss = 0.0
    ground_list = []
    pred_list = []

    for i, data in enumerate(train_loader, 0):
        imgs, labels = data # Split images and labels
        imgs = imgs.to(device)
        labels = labels.to(device)
        labels = torch.squeeze(labels, dim = 1)

        optimizer.zero_grad() # Zero the gradients for the next pass

        probs = model(imgs)
        preds = torch.argmax(probs, dim = 1)
        # max_probs, preds = torch.max(probs, dim = 1)

```

```

    loss = loss_func(probs, labels) # Calculate loss
    loss.backward() # Calculate gradients
    optimizer.step() # Perform back pass/update weights
    running_train_loss += loss.cpu().detach().numpy()

    # Is it a problem that the tensors are on the gpu at this point?
    for j in range(len(preds)):
        ground_list.append(labels[j].cpu().detach().numpy())
        pred_list.append(preds[j].cpu().detach().numpy())

    running_train_loss = running_train_loss/len(train_loader)
    accuracy = accuracy_score(ground_list, pred_list)

    return running_train_loss, accuracy

def _val_step(model, val_loader, device, loss_func):

    running_val_loss = 0.0
    ground_list = []
    pred_list = []

    for i, data in enumerate(val_loader, 0):
        imgs, labels = data
        imgs = imgs.to(device)
        labels = labels.to(device)
        labels = torch.squeeze(labels, dim = 1)

        probs = model(imgs)
        preds = torch.argmax(probs, dim = 1)

        loss = loss_func(probs, labels)
        running_val_loss += loss.cpu().detach().numpy()

        for j in range(len(preds)):
            ground_list.append(labels[j].cpu().detach().numpy())
            pred_list.append(preds[j].cpu().detach().numpy())

    running_val_loss = running_val_loss/len(val_loader)
    accuracy = accuracy_score(ground_list, pred_list)

    return running_val_loss, accuracy

#####
# Training

```

```
def fit(model, train_loader, val_loader, device, epochs, loss_func, optimizer, writer, save,
save_path):
```

```
    model.train() # set to training mode
```

```
    path1 = "train_loss.txt"
    path2 = "val_loss.txt"
    path3 = "train_acc.txt"
    path4 = "val_acc.txt"
```

```
    # Clear files if they already exist
```

```
    fp1 = open(path1, 'w')
    fp2 = open(path2, 'w')
    fp3 = open(path3, 'w')
    fp4 = open(path4, 'w')
```

```
    fp1.close()
    fp2.close()
    fp3.close()
    fp4.close()
```

```
    for epoch in range(epochs):
```

```
        epoch_train_loss, epoch_train_acc = _train_step(model, train_loader, device, loss_func,
optimizer)
```

```
        epoch_val_loss, epoch_val_acc = _val_step(model, val_loader, device, loss_func)
```

```
        # Write to TensorBoard
```

```
        writer.add_scalar("Train/Loss", epoch_train_loss)
        writer.add_scalar("Val/Loss", epoch_val_loss)
        writer.add_scalar("Train/Acc", epoch_train_acc)
        writer.add_scalar("Val/Acc", epoch_val_acc)
```

```
        fp1 = open(path1, 'a')
        fp2 = open(path2, 'a')
        fp3 = open(path3, 'a')
        fp4 = open(path4, 'a')
```

```
        # Save to csv files
```

```
        fp1.write(str(epoch_train_loss) + ',')
        fp2.write(str(epoch_val_loss) + ',')
        fp3.write(str(epoch_train_acc) + ',')
        fp4.write(str(epoch_val_acc) + ',')
```

```
        fp1.close()
        fp2.close()
```

```

fp3.close()
fp4.close()

if(True == save):
    # Saves a DataParallel model
    save_DataParallel(model, os.path.join(save_path, str(epoch) + ".pth"))

print("Epoch : %d" % (epoch))
print("Train Loss : %f\tValidation Loss : %f" % (epoch_train_loss, epoch_val_loss))
print("Train Acc : %f\tValidation Acc : %f\n" % (epoch_train_acc, epoch_val_acc))

fp1 = open(path1, 'r')
fp2 = open(path2, 'r')
fp3 = open(path3, 'r')
fp4 = open(path4, 'r')

train_loss = fp1.read()
val_loss = fp2.read()
train_acc = fp3.read()
val_acc = fp4.read()

fp1.close()
fp2.close()
fp3.close()
fp4.close()

train_loss = train_loss.split(',')
val_loss = val_loss.split(',')
train_acc = train_acc.split(',')
val_acc = val_acc.split(',')

train_loss.pop()
val_loss.pop()
train_acc.pop()
val_acc.pop()

train_loss = np.asarray(train_loss, dtype = np.float32)
val_loss = np.asarray(val_loss, dtype = np.float32)
train_acc = np.asarray(train_acc, dtype = np.float32)
val_acc = np.asarray(val_acc, dtype = np.float32)

plt.figure()
plt.plot(np.arange(epochs), train_loss, np.arange(epochs), val_loss)
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Train Loss", "Val Loss"])

```

```

plt.savefig("Losses.png")

plt.figure()
plt.plot(np.arange(epochs), train_acc, np.arange(epochs), val_acc)
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Train Acc", "Val Acc"])
plt.savefig("Accuracy.png")

#####
# Testing
def evaluate(model, test_loader, device):

    model.eval() # set to eval mode

    # For sklearn roc curve generation
    path1 = "grounds.txt"
    path2 = "predictions.txt"
    path3 = "probabilities.txt"

    fp1 = open(path1, 'w')
    fp2 = open(path2, 'w')
    fp3 = open(path3, 'w')

    fp1.close()
    fp2.close()
    fp3.close()

    fp1 = open(path1, 'a')
    fp2 = open(path2, 'a')
    fp3 = open(path3, 'a')

    with torch.no_grad():
        for i, data in enumerate(test_loader, 0):
            imgs, labels = data
            imgs = imgs.to(device)
            labels = labels.to(device)
            labels = torch.squeeze(labels, dim = 1)

            probs = model(imgs)
            # preds = torch.argmax(probs, dim = 1)
            corresponding_probs, preds = torch.max(probs, dim = 1)

            # Write necessary data to csv files
            for j in range(len(preds)):
                fp1.write(str(labels[j].cpu().detach().numpy()) + ',')

```

```

fp2.write(str(preds[j].cpu().detach().numpy()) + ',')
fp3.write(str(corresponding_probs[j].cpu().detach().numpy()) + ',')

fp1.close()
fp2.close()
fp3.close()

# Reload
fp1 = open(path1, 'r')
fp2 = open(path2, 'r')
fp3 = open(path3, 'r')

# Loads string
grounds = fp1.read()
preds = fp2.read()
probs = fp3.read()

fp1.close()
fp2.close()
fp3.close()

# Converts to list
grounds = grounds.split(',')
preds = preds.split(',')
probs = probs.split(',')

# Pop off empty element
grounds.pop()
preds.pop()
probs.pop()

# Convert to numpy array
grounds = np.asarray(grounds, dtype = np.int8) # Just zeros and ones
preds = np.asarray(preds, dtype = np.int8) # Just zeros and ones
probs = np.asarray(probs, dtype = np.float32)

# Metrics
tn, fp, fn, tp = confusion_matrix(grounds, preds).ravel() # ravel flattens the array
cm = np.array([[tp, fp],
               [fn, tn]])
print(cm)

fpr, tpr, thresholds = roc_curve(grounds, probs, pos_label = 1)

plt.figure()
plt.plot(fpr, tpr)

```

```

plt.title("ROC")
plt.ylabel("TPR")
plt.xlabel("FPR")
plt.savefig("figures/figure1.png")

accuracy = accuracy_score(grounds, preds)
print("Accuracy : %f" % accuracy)

```

To view tensorboard run this command on the command line: tensorboard --logdir=runs

GradCAM

```
def GradCAM(model, imgs, device, name, save = False, root = '.');
```

```
    imgs = imgs.to(device)
```

```
    # Enables gradient calculation for the tensor
```

```
    if(not imgs.requires_grad):
```

```
        imgs.requires_grad = True
```

```
    # Predict
```

```
    probs, preds = predict(model, imgs, device) # Make a prediction
```

```
    # Note : tc.argmax() is a nondifferentiable function and sets requires_grad to False
```

```
    prob = probs[:,preds] # Isolate the predicted class
```

```
    # Get the gradients of the predicted class w.r.t. the parameters of the model
```

```
    prob.backward() # Computes the gradient of current tensor w.r.t. graph leaves.
```

```
    # Note : backward() throws an error with multidimensional tensors
```

```
    # The tensor must be a zero rank tensor / a scalar
```

```
    # Extract gradients from the model
```

```
    gradients = model.gradients
```

```
    # Pool the gradients
```

```
    pooled_gradients = torch.mean(gradients, dim = (2, 3, 4))
```

```
    # Extract feature maps
```

```
    activations = model.activations.detach()
```

```
    # Extract number of channels
```

```
    ch_dim = 1 # ch_dim is 1 in PyTorch
```

```
    num_channels = activations.shape[ch_dim]
```

```
    # For each channel
```

```
    for i in range(num_channels):
```

```
        activations[:,i,::,:] *= pooled_gradients[:,i]
```



```

# Take the average across the channels
heatmap = torch.mean(activations, dim = ch_dim)

# Apply ReLU. Only positive contributions pass.
rectified_heatmap = torch.nn.functional.relu(heatmap)

# Normalize the heatmap for each sample in the batch
for i in range(imgs.shape[0]):
    rectified_heatmap[i] /= torch.max(rectified_heatmap[i])

# Resize each heatmap
resized_heatmap = np.zeros((imgs.shape[0], imgs.shape[2], imgs.shape[3], imgs.shape[4]))
for i in range(imgs.shape[0]):
    resized_heatmap[i] = skimage.transform.resize(rectified_heatmap[i].cpu().numpy(),
    (imgs.shape[2], imgs.shape[3], imgs.shape[4]))

# Overlay the heatmaps on the original images
# super_imposed = np.zeros((imgs.shape[0], imgs.shape[2], imgs.shape[3]))
# for i in range(imgs.shape[0]):
#     super_imposed[i] = resized_heatmap[i] + imgs[i][0].detach().numpy()

# Save CAM in nifti format
# Can read in MRICron
if(save):
    nifti = nib.Nifti1Image(resized_heatmap[0], np.eye(4))
    nib.save(nifti, os.path.join(root, name + '.nii.gz'))

return resized_heatmap

# Check to see if GPU is available
def get_device():
    if(torch.cuda.is_available()):
        return torch.device("cuda:0")
    else:
        return torch.device("cpu")

```

Appendix B6: Training Code

```
import os

import numpy as np
import torch as tc
import torchvision as viz
from torchvision.transforms import Normalize
from torch.utils.data import SubsetRandomSampler
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter # available as of PyTorch version 1.2

import json

# Custom Modules
import datasets
import models2
from models2 import get_device

def main():

    # Get current working directory
    root = os.getcwd()
    # Set data directory and label file
    parent_dir = os.path.join(root, "norm_classes")
    lbl_file = os.path.join(root, "revised_labels.json")

    # Load dataset
    ds = datasets.v3(parent_dir, lbl_file)

    path1 = "train_sampler.npy"
    path2 = "val_sampler.npy"
    path3 = "test_sampler.npy"

    if(os.path.exists(path1) and os.path.exists(path2) and os.path.exists(path3)):
        train_sampler = np.loadtxt(path1)
        val_sampler = np.loadtxt(path2)
        test_sampler = np.loadtxt(path3)
    else:
        # Split indices for training, validation, and testing
        train_sampler, val_sampler, test_sampler = datasets.split(len(ds), 0.10, 0.20)
        # Only run for new permutations
        np.savetxt(path1, train_sampler)
        np.savetxt(path2, val_sampler)
        np.savetxt(path3, test_sampler)

    # Load into PyTorch Sampler
```

```

trainSampler = tc.utils.data.SubsetRandomSampler(train_sampler)
valSampler = tc.utils.data.SubsetRandomSampler(val_sampler)
testSampler = tc.utils.data.SubsetRandomSampler(test_sampler)
# Load data into PyTorch DataLoader using respective sampler
batch_size = 16
train_loader = tc.utils.data.DataLoader(ds, batch_size, False, trainSampler)
val_loader = tc.utils.data.DataLoader(ds, batch_size, False, valSampler)
test_loader = tc.utils.data.DataLoader(ds, batch_size, False, testSampler)

# Creates new SummaryWriter
# Writes to runs
# If a runs directory does not exist, it will create one
writer = SummaryWriter()

# Get GPU if available
device = get_device()

# Load model
model = models2.v2()
# Check number of GPU's available
# If > 1, split training
if(tc.cuda.device_count()):
    model = tc.nn.DataParallel(model)
    print("Using %d GPU's." % tc.cuda.device_count())
model = model.to(device) # Send model to GPU if available

loss_func = tc.nn.CrossEntropyLoss()

lr = 0.001
momentum = 0.9
optimizer = tc.optim.SGD(model.parameters(), lr, momentum)

epochs = 200
save = True
save_path = os.path.join(root, "session5")
models2.fit(model, train_loader, val_loader, device, epochs, loss_func, optimizer, writer, save,
save_path)
return 0

if(__name__ == "__main__"):
    status = main()
    print(status)

```

Appendix B7: Statistical Evaluation Code

```
import os

import numpy as np
import torch as tc
import torchvision as viz
from torchvision.transforms import Normalize
from torch.utils.data import SubsetRandomSampler
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter # available as of PyTorch version 1.2

import json

# Custom Modules
import datasets
import models2
from models2 import get_device

def main():
    # Get current working directory
    root = os.getcwd()
    # Set data directory and label file
    parent_dir = os.path.join(root, "norm_classes-aug")
    lbl_file = os.path.join(root, "revised_labels.json")

    # Load dataset
    ds = datasets.v3(parent_dir, lbl_file, True)

    path3 = "test_sampler.npy"

    if(os.path.exists(path3)):
        test_sampler = np.loadtxt(path3)
    else:
        print("No test_sampler provided.")

    # Load into PyTorch Sampler
    testSampler = tc.utils.data.SubsetRandomSampler(test_sampler)
    # Load data into PyTorch DataLoader using respective sampler
    batch_size = 16
    test_loader = tc.utils.data.DataLoader(ds, batch_size, False, testSampler)

    # Check to see if GPU is available
    device = get_device()

    # Load model
    model = models2.v2()
```

```
models2.load(model, "session5/130.pth")
# Check number of GPU's available
# If > 1, split training
if(tc.cuda.device_count()):
    model = tc.nn.DataParallel(model)
    print("Using %d GPU's." % tc.cuda.device_count())
model = model.to(device) # Send model to GPU if available
# model.eval()

# Evaluate the model
models2.evaluate(model, test_loader, device)
return 0

if(__name__ == "__main__"):
    status = main()
    print(status)
```

Appendix B8: Graphical Evaluation Code

```
#####
import os
import sys

#####
# PyTorch modules
import numpy as np
import torch
import models2

#####
# Medical Image Formatting
import nibabel as nib

def get_device():
    if(torch.cuda.is_available()):
        return torch.device("cuda:0")
    else:
        return torch.device("cpu")

def get_cube(img, cube_shape):
    depth_offset = (img.shape[0] - cube_shape[0])/2 # Take the difference and center it
    width_offset = img.shape[1]//8
    height_offset = img.shape[2]//8
    img_cube = np.zeros(cube_shape)
    for i in range(cube_shape[0]):
        for j in range(cube_shape[1]):
            for k in range(cube_shape[2]):
                img_cube[i][j][k] = img[depth_offset + i][width_offset+j][height_offset+k]
    return img_cube

def main(argc, argv):

    if(3 > argc):
        print("Usage: python3 script.py image.nii.gz name")
        return -1

    cwd = os.getcwd()
    # nifti = nib.load(os.path.join(cwd, argv[1]))
    # img = nifti.dataobj
    img = np.asarray(np.load(os.path.join(cwd, argv[1])), dtype = np.float32)

    # Assuming that we use the original size (96 x 160 x 160)
    cube_shape = (80, 120, 120)
    # Should I downsample like I do when training?
```

```

img_cube = get_cube(img, cube_shape)
nifti_cube = nib.Nifti1Image(img_cube, np.eye(4))
nib.save(nifti_cube, os.path.join(cwd, argv[2] + '_cuboid.nii.gz'))

img_cube = torch.Tensor(img_cube)
img_cube = torch.unsqueeze(torch.unsqueeze(img_cube, dim = 0), dim = 0)
img_cube.requires_grad = True

device = get_device()
# device = "cpu"
model = models2.v2()
model.eval()
model.to(device)
models2.load(model, "session5/130.pth")
_ = models2.GradCAM(model, img_cube, device, argv[2] + '_GradCAM.nii.gz', save = True)
print("GradCAM saved.")

return 0

if(__name__ == "__main__"):
    status = main(len(sys.argv), sys.argv)
    print(status)

```

Appendix C: Oversaturated Versus Saturated Image

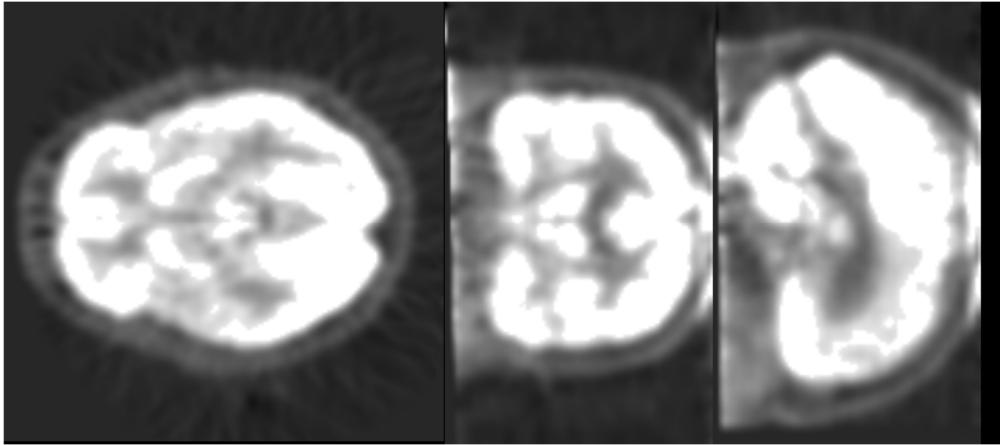


Figure 8: Oversaturated Image

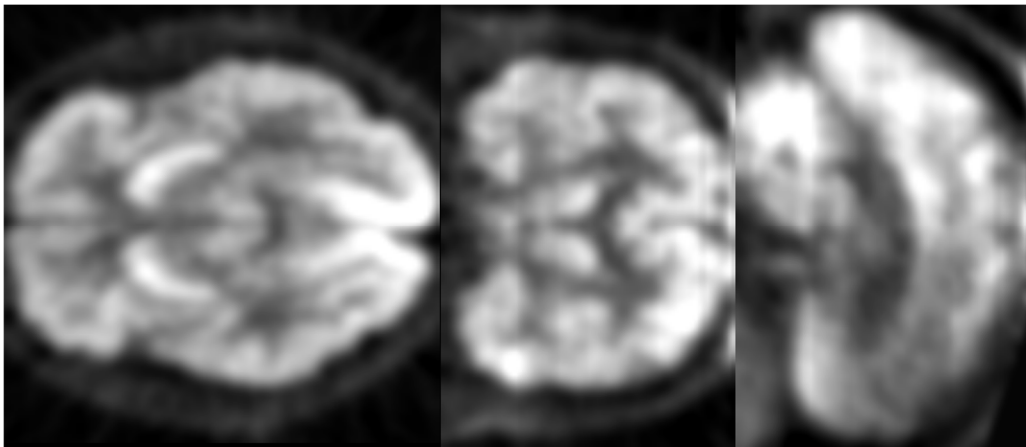


Figure 9: Normal Image

Appendix D: Patent Search

We performed a patent search for machine learning algorithms similar to ours. Specifically, we searched for patents that were representative of machine learning algorithms that performed classification tasks. The following are the machine learning systems that are representatives by the few patents that we found:

1. Medical Imaging System Providing Disease Prognosis
2. Method and System for Automated Brain Tumor Diagnosis Using Image Classification
3. Systems and methods for brain hemorrhage classification in medical images using an artificial intelligence network.

The Medical Imaging System Providing Disease Prognosis patent portrays a machine learning system that can input medical imaging data of a few samples of high dimensionality to perform a classification task. This system is very generalizable. The algorithm of the medical imaging system can perform classification tasks to aid in diagnosing and monitoring cognitive diseases like Alzheimer and dementia. Unlike in our deep learning algorithm, this algorithm system uses an autoencoder as its neural network rather than a ResNet [30].

The Method and System for Automated Brain Tumor Diagnosis Using Image Classification patent portrays a machine learning method and system that uses learned features of a brain tumor to make classification decisions. This classification is done on confocal laser endomicroscopy (CLE) images of the brain. CLE is a medical imaging technique that provides microscopic information of tissue in real-time on cellular and subcellular levels. The classes used are malignant or benign tissues. Any specific classifier and any type of machine learning based classifier may be used: Support Vector Machine (SVM) or random forest classifier [31].

The Systems and Methods for Brain Hemorrhage Classification in Medical Images Using an Artificial Intelligence Network to help characterize and classify brain hemorrhaging. An image from an x-ray computed tomography ("CT") imaging system is fed into the machine learning algorithm of a convolutional neural network. The output of the system is a class activation map that indicates at least one type of brain hemorrhage and a confidence value of each labeled region of where the hemorrhaging may be occurring representing a corresponding at least one brain hemorrhage condition [32].

Based on the patents above, creating a patent for our machine learning algorithm is promising. Sometimes algorithms are considered as computational and therefore non patentable. However, since we are applying our algorithm as a part of a technical solvency, our project will be patentable. Lastly, our system and method around our algorithm solves a unique classification problem with regards to identifying a predisposition rather than identifying a sign of diagnosis.