Artificial Neural Network to Detect Alzheimer's in MRI Scans

James Poirier, Michael Tuttle

EE 462 Final Report

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University San Luis Obispo

June 2020

TABLE OF CONTENTS

LISTS OF TABLES AND FIGURES

ABSTRACT

Alzheimer's Disease ranks (AD) as one of the most common diseases in America. Currently, detecting Alzheimer's Disease relies upon reported symptoms, however changes in the brain can manifest years or decades before symptoms appear. In recent years, researchers have successfully utilized Artificial Neural Networks (ANN) for a variety of image classification tasks. Here, we train an ANN to detect Alzheimer's Disease using magnetic resonance imaging (MRI) brain scans. Giving an MRI to both a neural network and a doctor will allow the doctor to be more confident in their answer, as well as double-check their answer with an objective report from a trained network.

Classifying and diagnosing Alzheimer's disease has proven to be a problem in the past, with many cases going unnoticed until full cognitive impairment of the patient has occurred. Giving patients more time to prepare and delay the effects of Alzheimer's and Dementia allows for a more successful and invigorating life, while ensuring the proper counter measures are being implemented as appropriate. This project specifically addresses the need for more reliable Alzheimer's Disease detection. Because machine learning algorithms exceed human ability for detecting patterns in abstract data, neural networks could assist doctors diagnosing AD. Current detection relies primarily on user-reported symptoms. Using the trained network can provide an opportunity for early detection of Alzheimer's Disease, enabling those preventative measures.

In this project, we designed a convolutional neural network to detect Alzheimer's Disease from 3-D MRI images. To do so, we slice each 3-D scan into multiple 2-D images and input each one to the network individually. Using an adapted InceptionV3 architecture, we achieve an testing accuracy of about 72% on individual 2-D image slices. We then postprocess the results by averaging the classification scores over each slice of the MRI scan, which increases the classification accuracy to just over 88%.

Chapter 1. Introduction

Chapter 1 will include a deeper introduction into Alzheimer’s disease, Neural Networks, and the process in how detecting a disease such as Alzheimer’s through MRI scans begins. It will also discuss a few of the approaches that have been taken so far, and why this approach is the most promising and reliable version. This will be filled in more as the quarter goes on, and we are able to compare our results more accurately with other projects.

Alzheimer’s Disease (AD) is the most common form of Dementia, or the loss of cognitive functioning, accounting for approximately 60-70% of all cases [1]. AD typically appears in people over 60 years old, however early-onset Alzheimer’s can occur much earlier in life. Early detection of Alzheimer’s Disease could enable patients to start treatment early and make necessary arrangements and preparations, while also providing scientists with more opportunities to research the causes, development, and prevention of AD. Unfortunately, current AD detection relies on a doctor’s analysis of existing symptoms, with a definite diagnosis only possible postmortem [2]. Because of this, researchers have turned to the superhuman pattern-detection capabilities of machine learning systems as a possible diagnostic tool.

Advancements in machine learning algorithms over the past few decades have enabled the use of artificially intelligent systems for solving a wide variety of complex tasks. Algorithms providing the ability to train many-layered artificial neural networks (ANN) have led to these systems’ prominent use. In 2012, the system AlexNet [3] achieved state-of-the-art image classification accuracy using convolutional neural networks (CNN), spurring their widespread adoption for image recognition tasks [4].

Since AlexNet’s success, researchers have worked to adapt and apply CNNs in a medical imaging context. Prior work has compared the performance of different CNN architectures over a variety of medical image datasets of varying sizes and explored using semi-supervised learning for limited datasets [5], [6]. Looking specifically at neuroimaging, magnetic resonance imaging (MRI) data has proven useful to CNNs for detective brain abnormalities such as tumors [7].

Recently, published research has shown some success detecting AD using CNNs, however further improvements to these methods are necessary before applying these techniques traditionally [2]. This project seeks to improve upon previous attempts at AD detection to produce a system accurate enough for practical application through the use of more modern technologies and portable network creation tools.

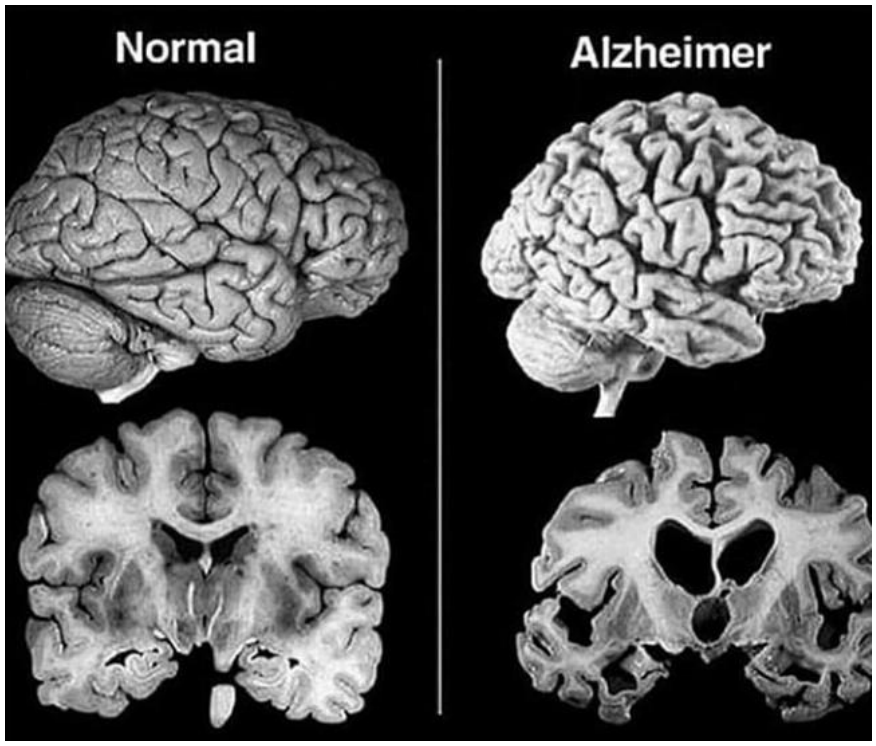

Figure 1: Comparison between Cognitively Normal and Alzheimer's Brains [21]

Figure 1 reveals a stark distinction between a cognitively normal brain and a brain with Alzheimer's Disease. The key features that distinguish the two are the extreme shrinkage of the hippocampus on the lower part of the brain, shrinkage of the cerebral cortex around the outside of the brain, and the enlarged ventricles in the center and lower portions of the brain [1]. These features are consistently present in all brains classified to have Alzheimer's Disease, and are the main aspects that the network will be attempting to analyze. Weighting each of these features and attributing the correct components to the classification of the brain was analyzed through various trainings to determine the most accurate classifier.

Previous networks have been successful in image recognition and citing similar patterns among various datasets, some of the most notable being AlexNet and Chellapilla's CNN on a GPU [7]. This project attempts to build off those previous networks and use a similar strategy to identify the specific areas in the brain that are most affected by Alzheimer's to then determine the level of cognitive impairment from an MRI scan. This project also contains the goal of being distributable and easy to use, so that anyone in the medical field can run the software and get a second opinion on an Alzheimer's diagnosis.

This project emphasizes the use of these new technologies to hopefully push the open-source movement further along and supply the AI community with the knowledge and possibilities of a few different ways to make neural networks more friendly to the rest of the world. Running these tools and educating others on the power behind neural networks will hopefully help drive the research and development of neural networks further along.

The following sections will discuss various previous methods, a background of Alzheimer’s disease, the approach that we took to conquer the problem, and the conclusions drawn and opportunities for future work. Chapter 2 will further discuss the reasons behind altering the previous approaches and emphasize the efforts made to create a more accurate, portable, and memory efficient network for detecting Alzheimer’s Disease. Chapter 3 will provide background information for Alzheimer's Disease and Artificial Neural Networks, to hopefully educate the reader enough to be able to internally reference this report for all questions. Chapter 4 will discuss the actual approach to the network, the performance of the network, and the various versions of the ANN that we tried, along with some of the results and comparisons between the different approaches.

Chapter 2. Literature Review

Chapter 2 includes a literature review on some of the previous methods and studies done in classifying Alzheimer’s Disease using Neural Networks. This will serve as a comparison between our study and other previous attempts at classification of Alzheimer’s Disease. It will go into the advantages and disadvantages of each method, and how our method plans to resolve as many of the previous issues as possible. Each researched method will be analyzed and compared to researched to provide us with the most well-rounded and modern ANN that we can develop. Finally, it will explain our method, and all the benefits and drawbacks that it brings to classification of Alzheimer’s Disease.

A network done by the Department of Electrical and Computer Engineering at McMaster University in Ontario Canada was able to get close to 100% accuracy using MRI scans from the ADNI dataset. Their network, known as Le-Net5 included two convolution, two pooling, and two FC layers in an optimized architecture to provide a binary output. This approach focused heavily on pre-processing images to provide as simple an image as possible. This included removing features such as the skull, eyes, and neck, as well as segmenting out grey matter, white matter, and fluid regions of the brain. This provided a more explicit and definitive approach to identifying the significant signs of Alzheimer's Disease.

Another network that was able to achieve a high level of accuracy was the Automatic Alzheimer's Disease Recognition from MRI Data Using Deep Learning Method from the School of Electrical Engineering and Computing at the University of Newcasstle, Callaghan, Australia, as well as the School of Computer Science from Shenzen University in Shenzhen China. This network was able to achieve a high level of accuracy through a 3D Convolutional Neural Network on small volumes of MRI scans using spatial redundancy. This network was also trained and run on ADNI, but featured significantly less pre-processing than the Le-Net5 network. Instead the patient sample would have fewer images and would go through numerous smaller series of pre-processing including Gradwarp.

Researchers developing machine learning systems often utilize TensorFlow for their work. Developed by Google, the success of Google Brain’s work using TensorFlow and the common usage among experts confirms the effectiveness of the TensorFlow framework. The TensorFlow creators also authored this whitepaper, making it the most reliable source for information on TensorFlow [9]. We chose TensorFlow because of its extensive documentation and references to utilize. This was a common framework used by many of the newer research papers and projects, and we found that they proved the most promising for the applications that we are using it for. TensorFlow also seemed to be easier to learn compared to some of its counterparts as shown through our research into the various platforms.

An Alzheimer's Disease Fact Sheet by the National Institute on Aging [2] provided us with the necessary background to understand the causes, symptoms, and effects of Alzheimer's Disease. The Fact Sheet was an excellent reference for us to learn about the basics of Alzheimer's Disease, and we encourage any persons reading this or a related study on Alzheimer's to give it a read. It goes into depth about the stages of the disease, effects, and current diagnoses. One notable point in this article is the lack of confidence when diagnosing Alzheimer's before a certain progression point of the disease. This is the difficult and frustrating part about this disease, and the part that this project aims at improving for doctors, patients, and the patients families.

A classic paper by Krizhevsky and Sutskever on ImageNet Classification with Deep Convolutional Neural Networks provided a reference and good learning opportunity for the initial stages of this project. The paper provided a quick and concise summary of common techniques using Deep Convolutional Neural Networks; the type of network that we believed would be the most successful in this experiment and research because of its ability to analyze multiple layers of an image.

Researchers A Parvat, J. Chavan, S. Kadam, S. Dev and V. Pathak, gave an overview of many different deep-learning frameworks used train artificial neural networks. The authors describe each framework in detail and compare their functionality and APIs [10]. This paper was incredibly helpful in determining the best framework for this project and gave us a lot of insights into potential networks and configurations to try to implement. The most helpful section in this article was the discussion between the varying capabilities and performance specs for the tools and frameworks that they were analyzing. Since each tool and framework is only able to integrate with specific hardware's, and many AI technologies are considered "bleeding edge," these researches gave a great overview of relevant, widely used scalable machine learning and deep learning frameworks that will fit the challenges that we set for our personal implementation.

A patent was filed [6] by Zhou, Yi, Xiaodong, and others that describes an improved method for training neural networks used specifically for medical image classification. This paper includes limitations of current neural network training schemes, and how their approach is able to increase the productivity and reliability of the medical image classifications without making the program significantly more computationally heavy. We are following a similar format to their idea, and try to incorporate the concepts of divided pre-processing and alternative image resizing to make our images that are fed into the network as accurate as possible.

Focusing on the framework selected for this project, the whitepaper written by the developers of TensorFlow [9] provided an excellent overview of the functionality and performance of TensorFlow. Researching this paper and checking the referenced trainings provided an excellent background for our project, as well as showcasing the opportunities and features that we could utilize by picking to work in TensorFlow. TensorFlow is also extremely well known and documented, and because of this made it a strong contender to use as our framework to create our neural network in.

TensorFlow also has a Google Collab learning engine [9] that provided an easy to understand and useful approach to learning how to implement ANN's in TensorFlow. Using this engine gave this project a leg up at the beginning because it provided a much faster and easier way to get familiar with the TensorFlow software. A similar learning engine for Keras helped us figure out the basics behind using the common wrapper to simplify the training, implementation, and modification stages of the project. Keras was the next logical software to learn to increase the time spent training and testing models and decrease the time working on the setup and creation of each individual network.

NVIDIA's article on their "AI Supercomputer" [11] was an excellent reference for GPU distributed computing and showcased how training neural networks on a more powerful system can leave the user with impressive results, as well as impressive overfitting. This became especially apparent as we began training our network on a relatively powerful cloud computing instance provided by AWS and realized that our model was consistently overfitting the data. Thankfully, the article also included references on how to avoid this problem, and ways to alter your inputs to encourage the least amount of overfitting as possible.

Many of the initial models that were trained were found to be overfitting the data, and only increased in overfitting as we began to run our training in a cloud computing instance with greater epochs and a much larger batch size. The "AI Supercomputer" [11] research paper provided us with a resource that showed a few different options to avoid overfitting and some implementations that we were able to incorporate to change the results of our initial tests.

Upon our re-creation of the ANN after fitting the data consistently, we found that "simple model of spiking neurons," [8] an article by E. M. Izhikevich was an excellent reference for a new type of medical ANN architecture that we had not tried yet. After changing our architecture to mimic the one presented in this paper, we found that the model performed better both in accuracy and in time taken to run. This paired with the NVIDIA "AI Supercomputer" [9] paper gave us a better understanding of the effects of overfitting the data, and showed us more consistent and reliable ways to detect whether or not we began to overfit the data.

Focusing on learning about Dementia and Alzheimer's Disease, the World Health Organization detailed the prevalence of Alzheimer's Disease in the world currently, and showed the overall social and economic impacts of these diseases [1]. The World Health Organization was the general reference that we used to learn about the cause, signs, and effects of Alzheimer's Disease. They are members of the United Nations and dedicated to focusing on improving international health. In the event that this project becomes even more accurate and can begin to be implemented in hospitals and doctors office, the World Health Organization will be one of the first resources to get the product licensed and approved.

Following the approach by Eo, Jun, Kim, Jang, Lee, and Hwang in their paper on "cross-domain convolutional neural networks for reconstructing under sampled magnetic resonance images" [14] provided us with a few new ideas on stretching, reconstructing, and padding our images to both avoid overfitting and fix some of the poorly captured MRI scans that could potentially cause a poor training result due to unexpected features within the ANN. The approach they took towards revitalizing MRI images proved to be more challenging than we expected, but the concepts that they utilized were very helpful in the modification and implementation of our own cleaning, padding, and stretching algorithm for each image.

As the project continued, we realized that more and more images were sized differently, and we would need a more consistent sizing algorithm to ensure the most accurate training possible. The first half of a research paper by Jha, Kim, and Kwon [17] on using a Dual-Tree Complex Wavelet Transform with a Feed-Forward Neural Network had an impressive and simple way to ensure padding, while minimizing the effect of the modified images and their sizing differences between their actual brains. Their testing proved that it was better to just pad the image and have different sized observation zones for the actual brain than try to resize and modify the actual image. We ended up trying a few different methods as well that will be discussed in greater detail in Chapter 4, but found a similar conclusion where stretching and altering the image caused the ANN to begin focusing on parts other than those we wanted.

The approach presented in this project utilizes new and emerging technologies to create the most user-friendly, lightweight, free-to-use, and powerful neural network for rapid and automated detection of Alzheimer's Disease. Implementing new tools like TensorFlow and using a language like Python will allow a significantly more sharable and repeatable result, without the use of a paid software or a massive overhead in storage and computing resources. This project also takes in the entire 3D MRI image, and attempts to create a logical conclusion based on the whole input, instead of fragmenting it and reducing the scan size as has been previously been done before.

Chapter 3. Background

Chapter 3 will introduce some of the core concepts of neural networks and training algorithms, as well as explore the approach that we took to ensure the highest level of accuracy for this project. It will also include our training algorithms, databases that we used, and some pseudocode that provides insight to the inner workings of the network.

Databases Used:

Alzheimer's Disease NeuroImaging Initiative: adni.loni.usc.edu

The ADNI Database from USC is the main set for training and testing the current model. This database is being used for its excellent documentation and classification, as well as its depth in patients for each level of disease classification. The Alzheimer's Disease NeuroImaging Initiative has a database dating back to 2004, containing 3 subsets that are all utilized together in the final set ADNI-3, which is being used in this project. This database features almost 1000 control patient MRI scans, 500 confirmed patients with Alzheiner's Disease, 500 early-stage Cognitive Impairment and 300 late-stage Cognitive Impairment. This extensive collection and classification is the reason behind the choice to use the ADNI-3 dataset. Each of the photos is hand-selected and analyzed by multiple medical professional to ensure the accuracy of the diagnoses and the quality of the photos is up to their standard.

OASIS-3 AD Dataset: oasis-brains.org/#data

The OASIS Alzheimer's Disease dataset is a secondary subset that is going to be primarily used for training and testing the actual effectiveness of the network. The OASIS-3 AD Dataset has slightly more patients in the overall repository, with a classification for each patient similar to the ADNI dataset. OASIS does not filter their MRI images however, so they are sometimes blurry, off center, and corrupted. This lends itself to an excellent testing environment for the network, to see how the network performs against new factors and differing images that will ideally expose the limits of this project.

Software Packages and Languages Used

Python - https://www.python.org/

Keras - https://keras.io/

TensorFlow - https://www.tensorflow.org/

Git - https://www.github.com/

NiBabel - https://nipy.org/nibabel/

Python was chosen for this project because of its portability, universal compatibility, and lack of initial cost. Python is also relatively low overhead for storage requirements and can be easily run from a command line or an executable to provide the easiest possible user experience. Python is also quickly becoming the most popular coding and scripting language, and many computers come with quick installs for Python now, if it is not already installed. Python is also the language that we were most familiar with and thought that the tools provided and resources available with the language made it the most obvious choice for this project.

TensorFlow was chosen as the engine behind the network because of its extensive documentation, common trainings, and similarly small overhead when downloading. Utilizing TensorFlow over another tool like Matlab enables the code to be shared with any other developer or researcher who has knowledge pertaining to neural networks without requiring an expensive license. The ability to convert the program into an executable or a runnable script also allows for extremely easy sharing of the code, network, and results. This will be especially effective when sharing between hospitals and medical sectors because the effective cost of running the program is $0.

Keras was chosen for this project to make the actual neural network code more readable, more human-friendly, and more intuitive. It allowed for more time in research and development instead of learning the intricacies of TensorFlow. Keras does add some overhead in the development and running of a program, but the actual memory requirements for it are negligible compared to a standard program.

Git was chosen for its assistance in version control and ability to share and upload results efficiently and quickly. Having a URL dedicated to the project provides a single place for all the source code to be available to anyone with permission to access it.

NiBabel is a python package that provides functions for reading and writing to neuroimaging file formats. Because the MRI scan images used in this project are stored in the volumetric NIfTI file format commonly used for neuroimaging, this package is necessary to read the data into a format that can be input to the neural network model.

Neural Network Core Concepts

This project addresses the need for more reliable Alzheimer's Disease (AD) detection through the implementation of a neural network. Neural Networks are computational models based on the structure and functions of a living biological neural network. Every neural network follows the same basic structure: containing an input layer, hidden layer(s), and an output layer. The input layer feeds all of the inputs into the network. The hidden layer (or layers) process the inputs received from the previous layer. The output layer presents the processed data after the hidden layer has analyzed the inputs [13].

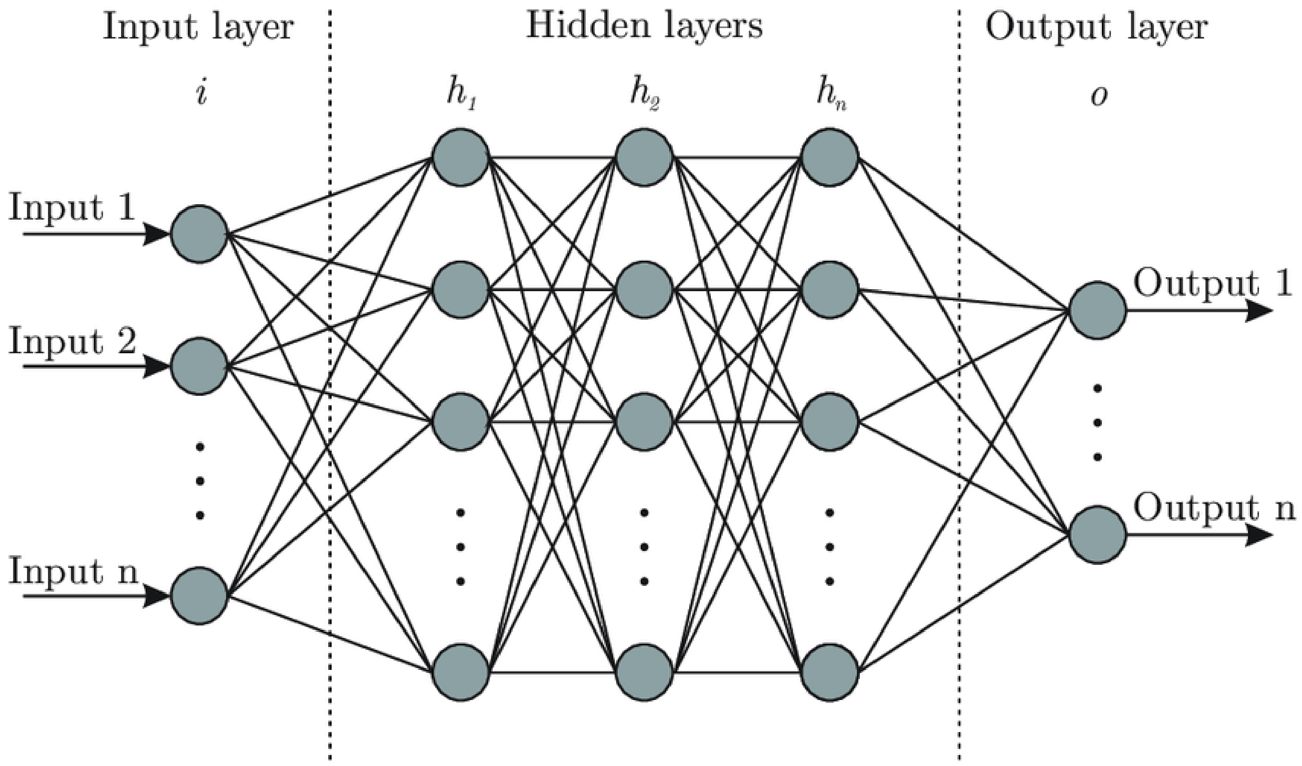

Figure 2: ANN Generic Structure [20]

Artificial Neural Networks got their start in 1943 when Warren McCulloch, a neurophysiologist and his partner Walter Pitts, a mathematician, wrote a paper on how neurons might work. They modeled a simple neural network with electrical circuits, forming the baseline for our current understanding of neurons and networks. Neural networks have since appeared in almost every aspect of our world; from intense theoretical computations to phone communications and everything in between.

Using brain scan images could provide an opportunity for early detection of Alzheimer's Disease, enabling preventative measures. Because machine learning algorithms exceed human ability for detecting patterns in abstract data, neural networks could assist doctors diagnosing AD. Because there exists a tradeoff between specificity and sensitivity, we must decide which metric to optimize. We prefer more specificity over sensitivity due to the undue stress caused by a false positive diagnosis and the existence of other detection methods that can compensate for false negatives. To better understand the customer needs, we spoke with medical professionals who might use this device.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific type of ANN architecture that has found widespread use in image classification and object recognition tasks. These models are significantly deeper than standard ANNs and are composed primarily of convolutional layers. In each of these layers, 2-D windows, called kernels, are trained to detect patterns in the input data. These kernels are shifted across the input, performing a 2-D correlation operation that maps the data to a new feature space in which regions containing similar patterns to the trained window are mapped to higher values than dissimilar regions. Each convolutional layer combines the features from the preceding layer to detect increasingly complex features from the original image. At the output of the network, one or two fully connected hidden layers are generally used to interpret the final feature space and perform the classification task. This technique yielded state-of-the-art results for image classification in AlexNet when trained on the ImageNet dataset. Since then, convolutional models have become standard in almost all neural network-based image classification tasks.

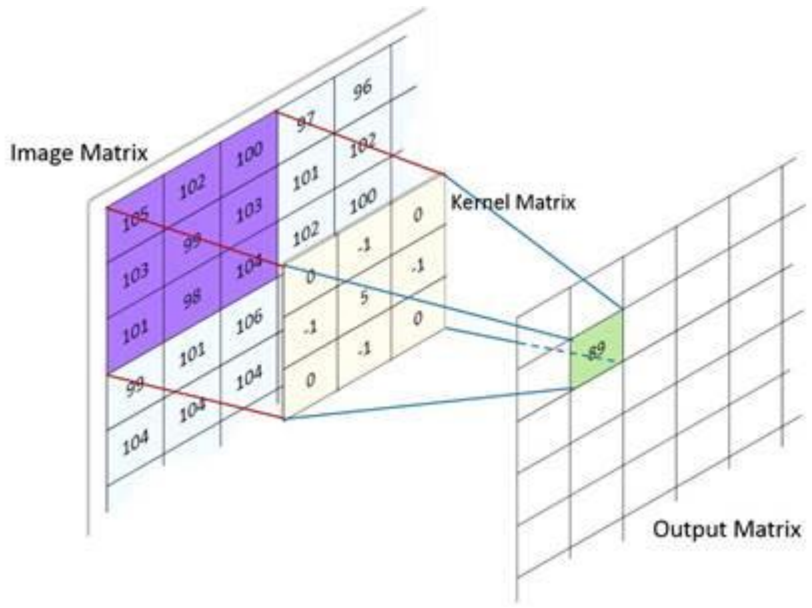

Figure 3: Visualization of 2-D Convolution Operation [19]

Chapter 4. Approach and Results

Chapter 4 will explain the approach that we took, and the structure behind each of our components. It will discuss the overall structure of the project using system block diagrams, computational flow charts, and pseudo code to further explain the logic behind some of our decisions. It will also include all data, plots, and tables that discuss the results of our tests.

Training Approach

To train our model, we gathered data from the Alzheimer's Disease Neuroimaging Initiative (ADNI). For consistency, all the data used came from the dataset "ADNI Complete: 1 Year 1.5 Tesla." This dataset consists of 2292 total MRI scans, of which 476 correspond to patients with Alzheimer's Disease (AD), 703 correspond to patients who are cognitively normal (CN), and 1113 correspond to patients experiencing Mild Cognitive Impairment (MCI). For simplicity, we looked only to distinguish between CN and AD patients, leaving us with a dataset of 1179 samples. The neural network model was then trained on 80% of the patients, with 10% used for the validation and testing sets each. Each patient was randomly assigned one of these three sets, and all MRI scans of the patient were then placed in that data set. This prevented any patient crossover between datasets. Because patients do not all have the same number of MRI scans, the actual percentages of training samples does not exactly match the patient percentages in each data set.

Each MRI scan image was stored in a NIfTI volumetric file format. Each dimension of these images varied slightly from patient to patient, however in general, the size of each dimension fell within the range of 166-256 pixels. Additionally, the number of pixels in each dimension was not always consistent, meaning the images could take on a variety of aspect ratios.. The initial plan was to use the entire volumetric image and input it into a 3-dimensional convolutional neural network for training, however this resulted in a worst-case input size of over 16 million voxels, which was far larger than could be reasonably used considering the available hardware. Instead, we took 17 planar slices from each brain scan (parallel to the top of the head) ranging from 16 voxels above the image center to 16 voxels below in steps of 2 voxels. These slices were zero padded to uniform dimensions of 256x256 and saved as PNG image using matplotlib's default viridis colormap before being input to the neural network. While training, each PNG image was considered as a separate training instance (i.e., postprocessing is not performed).

To train the network, we shuffled together the training images for all the training patients and input them to the network in random order. We then normalized the pixel values to be between 0 and 1 by dividing by 255. Random zooms were used to expand the apparent size of the dataset. The network then predicts the category of the image (CN or AD) and the prediction error is measured using the binary cross entropy loss function. Our initial attempts to train a model were limited by memory constraints that prevented the use of large batch size for training. This led us to use the stochastic gradient descent optimizer, which can update the network model based on an approximate the network's error gradient measured from a single sample or a small set of samples called a mini-batch. To train the model, we used a mini-batch size of 16 input images and trained the model for 100 epochs. Once we gained access to more computing resources, we switched to the Adam optimizer with a batch size of 32 input PNG images.

Training Algorithm

While training our model, we experimented with two different optimizers: stochastic gradient descent (SGD) and Adam. For SGD, we do not use a momentum term when updating the trainable parameters, so the update formula is simply:

$$\theta_t = \theta_{(t-1)} - \eta \cdot g_t$$

Where θ_t is a trainable parameter at time step t, η is the learning rate, and g is the estimated gradient of the objective function with respect to parameter θ averaged over each training sample in the mini-batch. For our model, we used a learning rate of 0.01 and a mini-batch size of 16.

The algorithm for the Adam optimizer is slightly more complicated. The algorithm is described in Adam: A method for Stochastic Optimization [24] and summarized on the next page.

Adam Algorithm [24]

```
**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)
```

For our training purposes, we used parameter values of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e - 7$. Each of these parameters were the default parameters in the Keras provided functions. In each of these algorithms, the binary cross-entropy loss function is optimized. This function is defined as follows:

$$E(y, \hat{y}) = -\frac{1}{N} \Sigma_{i=0}^{N} \big(y * log(\hat{y_i}) + (1 - y) * log(1 - \hat{y_i})\big)$$

Where N is the batch size (32), y is the instance label, and $\hat{y}$ is the output of the neural network from the sigmoid function (in the range (0, 1) non-inclusive).

Batch Normalization Algorithm

$$\text{Mean Computation: } \mu = \frac{1}{N \cdot H \cdot W} \sum_{b=1}^{N} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{bij}$$

$$\text{Variance Computation: } \sigma^2 = \frac{1}{N \cdot H \cdot W} \sum_{b=1}^{N} \sum_{i=1}^{H} \sum_{j=1}^{W} \left(x_{bij} - \mu\right)^2$$

$$\text{Normalization: } \widehat{x_{bij}} = \frac{x_{bij} - \mu}{\sigma}$$

Where N is the batch size of 32, (H, W) are the dimensions of the output feature space, and x_{bij} is a single output value of in the feature space for a single batch. Essentially, this finds the average and variance for each output channel of a layer by summing over all the output values in all the batches for that channel. A single mean and variance value are then used to normalize all the outputs in that channel.

Postprocessing Equations

$$\text{Average slice classification: } y_{avg} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

$$\text{Final output: } \begin{cases} 0, & if\ y_{avg} \leq 0.5 \\ 1, & if\ y_{avg} > 0.5 \end{cases}$$

Where y_i is the network output for a single PNG slice, and N is the total number of slices taken from each MRI scan (N = 17). Note that postprocessing is not applied during training, only during MRI classification after the system is fully trained.

System Architecture

Figure 4 shows the Alzheimer's Detection system's high-level block diagram. The user inputs a NIfTI data file of an MRI scan to the system, which then loads, slices, resizes, and normalizes the input data before inputting the image as a sequence of PNGs to the neural network. The neural network determines a classification score for each slice of the scan ranging from 0 (no Alzheimer's) to 1 (Alzheimer's detected) and outputs the most likely diagnosis based on the scores of each input slice using the postprocessing equation shown above. Table I summarizes the overall system functionality.

TABLE I

ALZHEIMER'S DISEASE DETECTOR HIGH LEVEL FUNCTION TABLE

Input	NIfTI MRI scan of resolution ≥ 166x166x166
Output	Classification indicating whether the network believes the patient has Alzheimer's Disease. The output is averaged classification score of the 17 input slices, thresholded with a value of 0.5. An output of 0 indicates a positive diagnosis for AD, while an output of 1 indicates a negative diagnosis.

The system decomposes into three basic systems: the image preprocessor; the neural network; and the display module. The preprocessor converts the input MRI scan image into a series of PNG images using matplotlib's viridis colormap. The image slices are taken along the xy plane ranging from +/- 16 voxels from the center of the z axis, in steps of 2 voxels, producing 17 output PNG images per MRI scan. These images are then symmetrically zero padded to a size of 256x256 pixels. Each PNG image is input the network for classification individually.

A convolutional neural network architecture is used for classification of the images. In this model, the input images pass through a series of convolutional layers, which extract key features from the images that are useful to the classification task. This is done by training convolutional kernels that are swept across the image performing a 2-D correlation operation that maps the input image to a set of feature spaces. Subsequent convolutional layers build off the feature spaces of the previous layers to extract increasingly complex patterns.

Max pooling layers are often used directly after convolutional layers to reduce the input size of the subsequent layers while preserving important information. This data reduction is necessary to reduce the computational overhead, as CNN models are generally relatively large compared to other ANNs. Finally, the network must convert the data into a 1-dimensional array to give each input neuron in the feedforward network one data point. The neural network then performs a series of computations and outputs the classification score. Table II shows the layer-by-layer breakdown of the initial convolutional neural network used in this project. Kernel sizes and strides were chosen to emulate the AlexNet architecture, however the total number of convolutional kernels and layers were reduced to accommodate the available computational resources. The neurons in the dense layer were also reduced from their AlexNet values.

The postprocessing module receives the classification scores from the 17 different slices of the input data, averages the scores, and thresholds the classification with a value of 0.5.

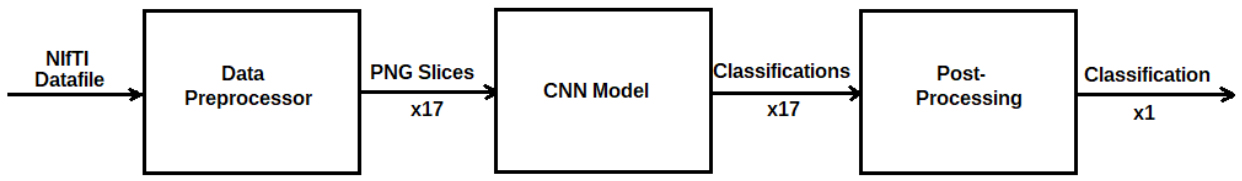

Figure 4. ANN for Alzheimer’s Disease Detection Inner Functionality

TABLE II

CONVOLUTION NEURAL NETWORK ARCHITECTURE

Layer	Parameters
2D Convolutional	Filters: 16 Kernel size: 11x11 Input shape: 256x256x1
2D Max Pooling	Kernel size: 3x3 Stride: 2
2D Convolutional	Filters: 32 Kernel Size: 5x5
2D Max Pooling	Kernel size: 3x3 Stride: 2
Flatten	None
Dense	Neurons: 512 Activation: Rectified Linear Unit (y = max(0, x))
Dense	Neurons: 512 Activation: Rectified Linear Unit (y = max(0, x))
Dense	Neurons: 1 Activation: Sigmoid

Sigmoid Activation Function Equation (y = output, x = weighted sum of inputs):

$$y = \frac{1}{1+e^{-x}}$$

Sample Photos

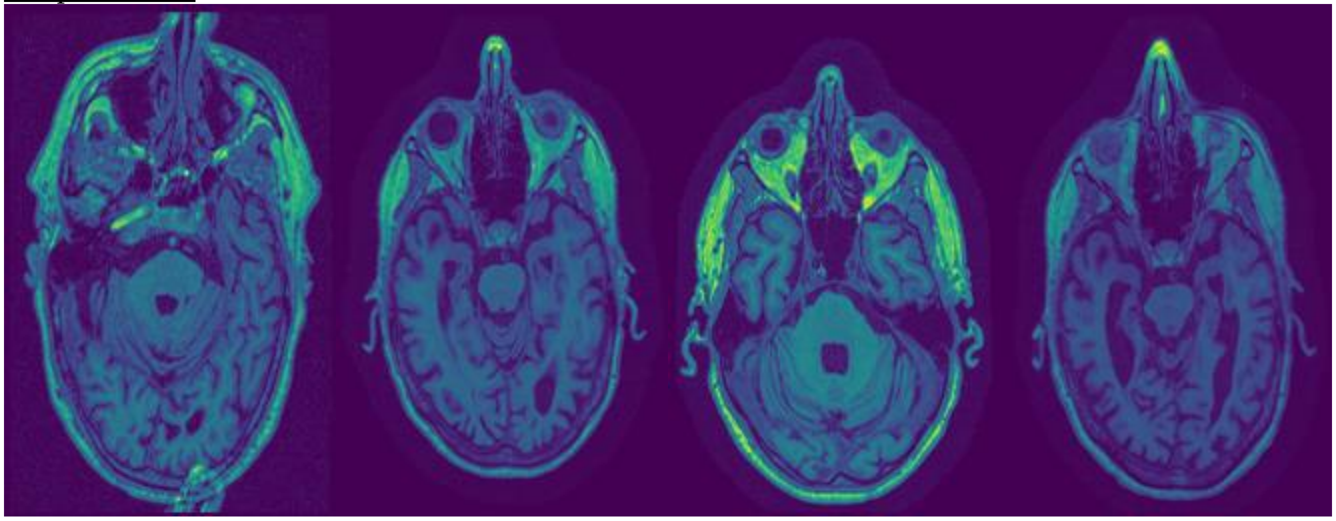
Figure 5: Sample AD Brain Slices

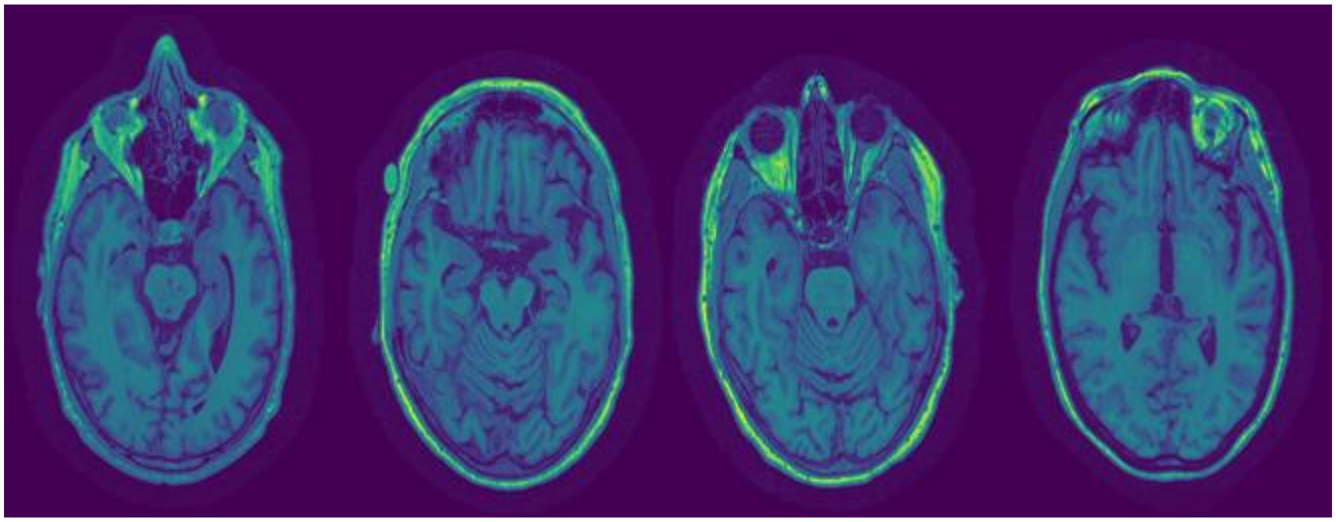
Figure 6: Sample CN Brain Slices

Results

Using the above system model, we input the MRI scan data to the system and trained the network for 100 epochs. Figure 6 shows the training and testing accuracy of the neural network plotted against the epoch number. From this graph, it can clearly be seen that the model is overfitting the training data while failing to classify the validation data set. While the training accuracy continues to increase and approaches 100% accuracy, the validation accuracy oscillates wildly between 40% and 60% between training epochs. This suggests that the model has memorized the input images rather than learned to detect key features that correspond to Alzheimer's Disease.

To prevent the overfitting on the training data, we augmented this network model. First, batch normalization was added after each convolutional layer. This operation shifts the mean of each output layer to 0 while normalizing the magnitudes by the standard deviation. These values are calculated using the batch normalization equations described on page 16.

A 40% drop out rate was also added to the fully connected layers of the network. This essentially removes a random subset of the neurons for each training step, which adds randomness to the model, making it harder for the network to memorize input images. 40% drop out was the most effective dropout rate of the range that were tested. After adding these measures to prevent overfitting, however, we still did not see a significant improvement in the network's validation data predictions. Thus, we decided to abandon our simple network model and try larger and more proven network models.

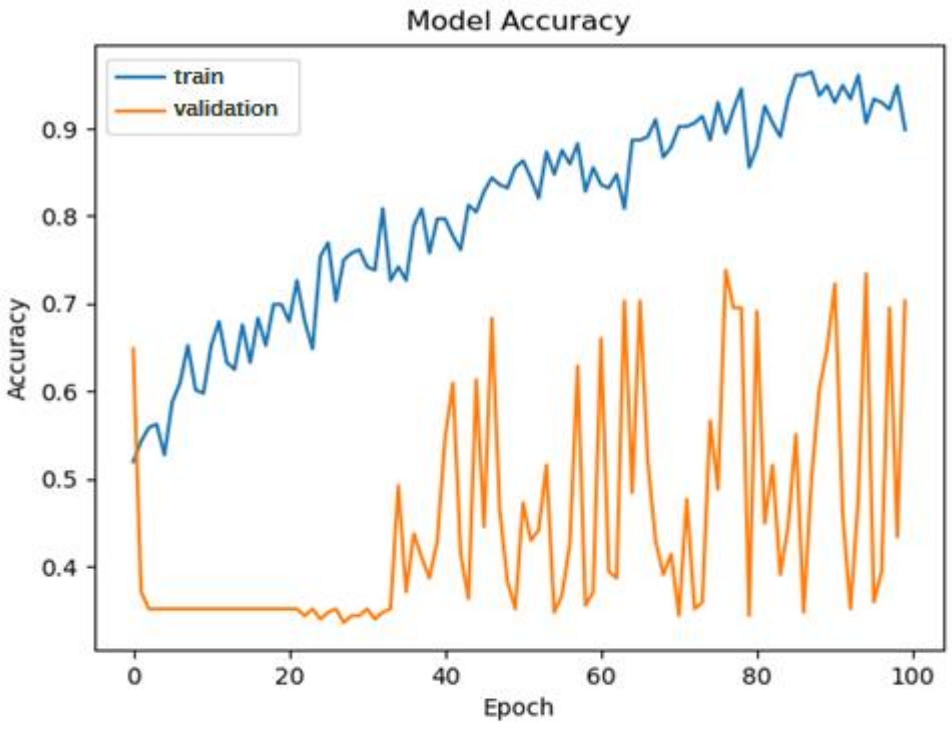

Figure 7: Model Accuracy vs Training Epoch

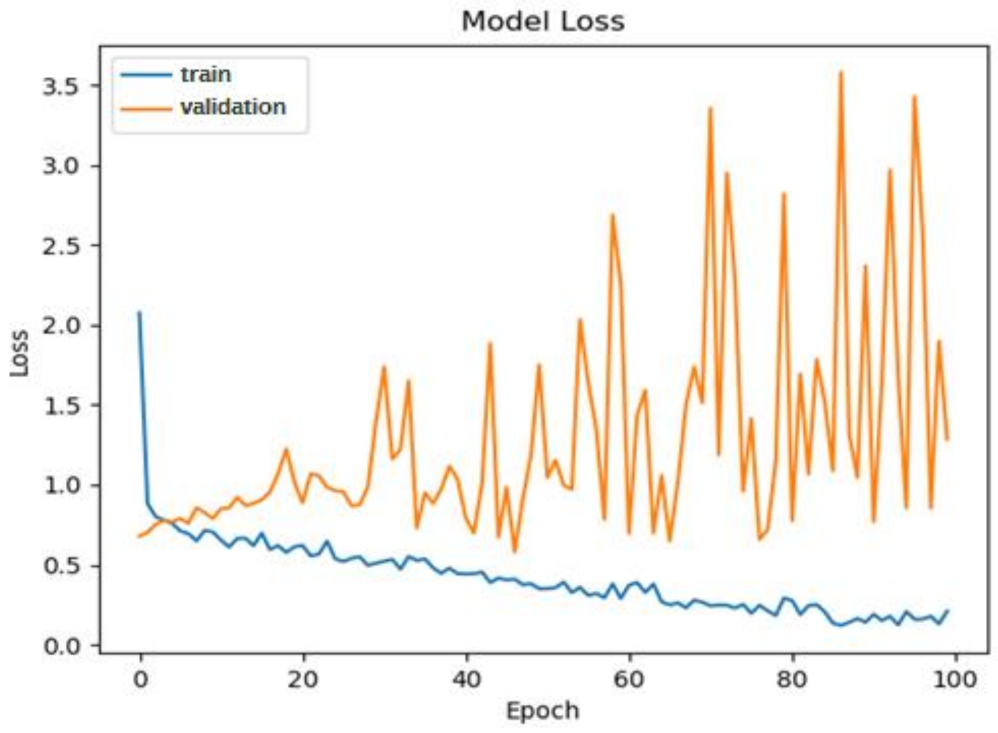

Figure 8: Model Loss vs. Training Epoch

The main concern we had with our previous network model from was that it was too limited in its size and number of convolutional kernels. Due to the hardware constraints of the laptops we were training the model on, we were unable to use a model even close to the size of robust CNNs such as AlexNet. In order to run the model, we had pared the convolutional layers down to about 10% the number of kernels as in AlexNet, with similar reductions to the fully connected layers.

To address this issue, we set up a remote server to train the network on that is significantly more powerful than our personal computers. This allowed us to try training a network that was much closer to the actual structure of AlexNet (except ours was grayscale and had 1 output node instead of 1000). Unfortunately, we found that this network quickly overfit the training data and was still unable to improve past the 60% validation accuracy.

Next, we attempted to utilize transfer learning to reduce overfitting and train the system faster. In this technique, we start with a system that is already pretrained on a different dataset, and we perform so retraining to repurpose it for our specific task. The idea behind this is that the features useful for one visual task can often be successfully applied in other contexts with only minor adjustments. For our transfer model we used the Inception V3 [19] architecture pretrained on the

ImageNet dataset. We removed the original 1000 node output layer and replaced it with our 1 node output for binary classification. The grayscale MRI images were converted to gray RGB images where the R channel equaled the G channel and equaled the B channel to be fed to the network. The Inception V3 model was chosen because there are less parameters and it was pre-trained.

In our first attempt, we froze the original model's parameters and trained only the weights between the last hidden layer and our new output layer. This meant that the exact features used to perform the ImageNet classification were being repurposed for Alzheimer's Detection. As could be expected, the accuracy plots shown in Figure 8 shows that this system was not able to effectively perform classification. With the training accuracy unable to increase past 70% and the validation accuracy stuck below 60%, we decided that the features used for ImageNet would not be useful for this classification task.

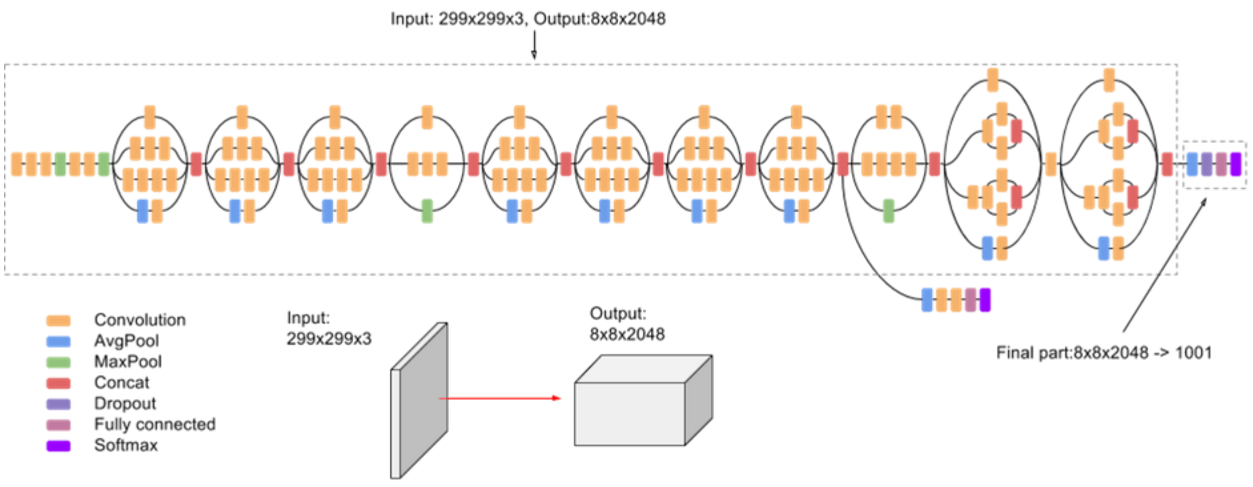

Figure 9: High-Level Architecture for Inception V3 model [24]

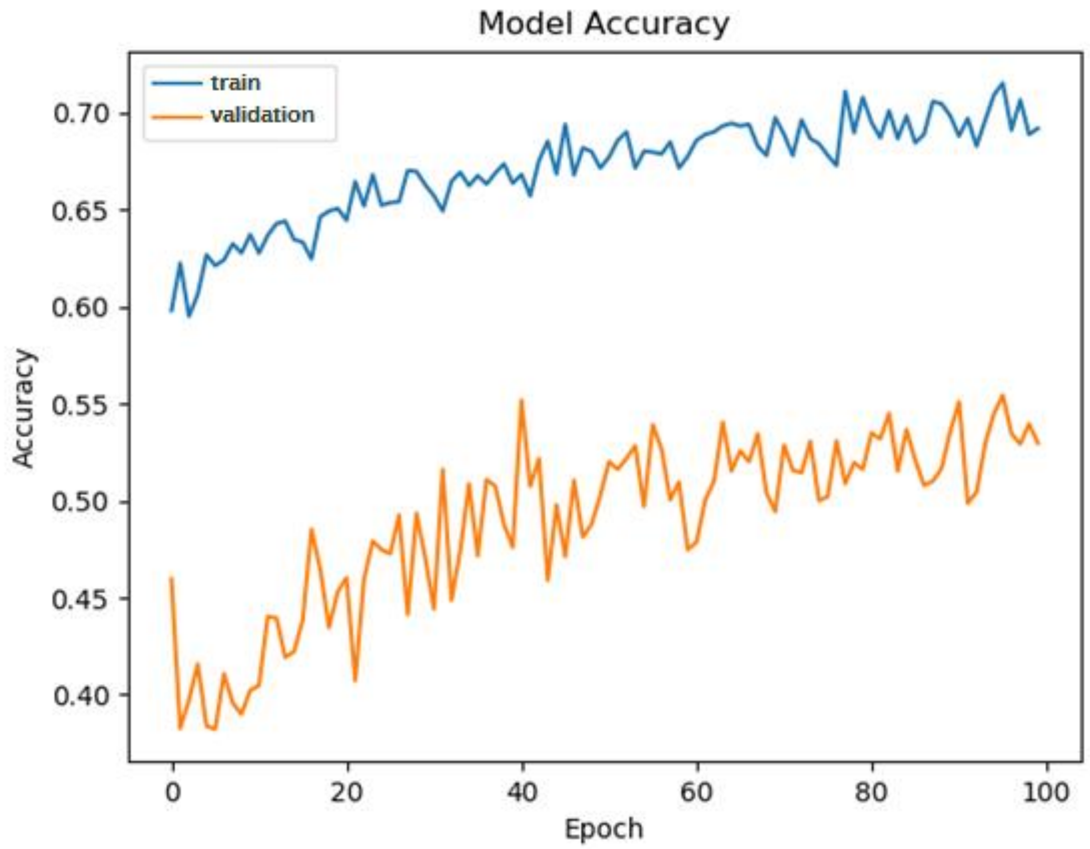

Figure 10: Model Accuracy vs Training Epoch

In our next attempt, we used the same model trained on ImageNet, however rather than freezing the model's parameters, we used the trained model as an initialized state but allowed for retraining of all parameters. This model once again overfit the training data, however using early stopping we were able to achieve slightly better results than with any previous method. Figure 9 shows the results of this training, yielding a peak validation accuracy just over 68%. While this is technically an improvement, it is still far below our target goal for the system.

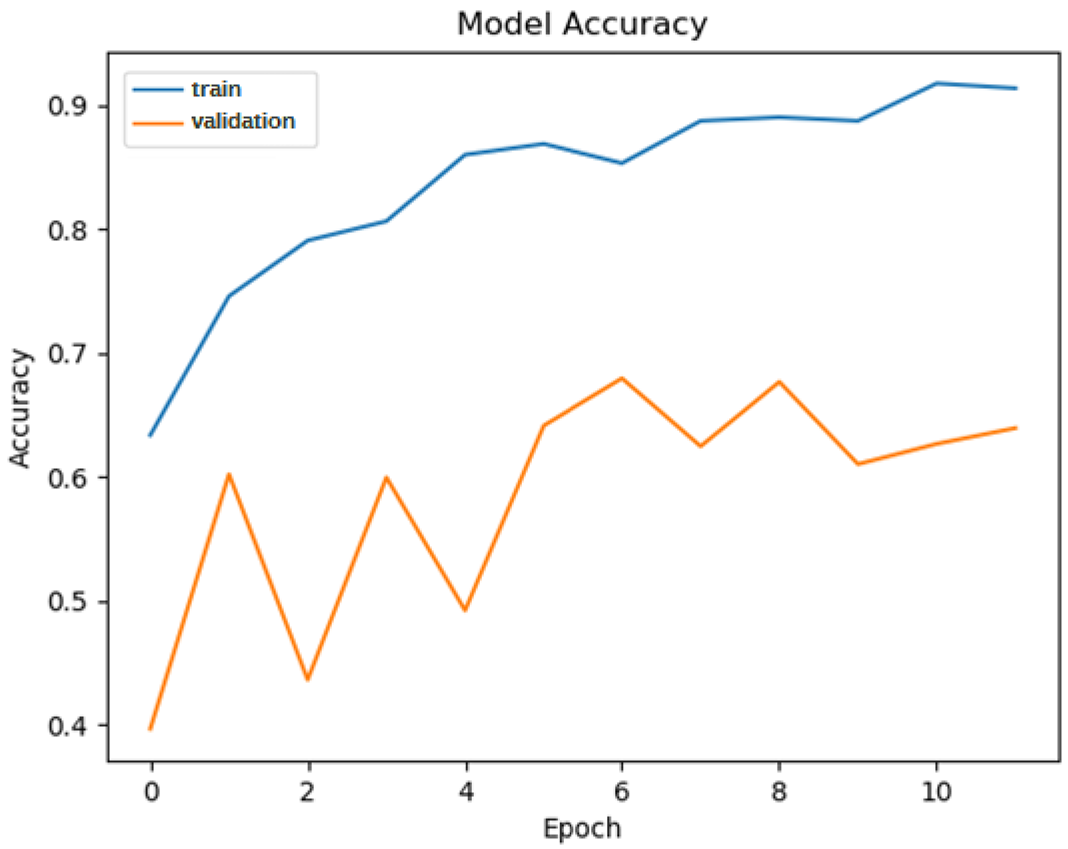

Figure 11: Model Accuracy vs Training Epoch

Our testing dataset accuracy was slightly higher however, at 72% accuracy without considering spatial redundancy. Table III shows the confusion of the raw PNG input classifications before redundancy. This shows the system has a sensitivity of 78%, while its specificity is just under 70% using the equations below.

$$\text{Sensitivity} = \frac{TP}{TP+FN}$$

$$\text{Specificity} = \frac{TN}{TN+FP}$$

When we incorporate the classification averaging postprocessing step, our accuracy climbs higher. The confusion matrix shown in Table IV summarizes the testing accuracy of the system with the redundancy postprocessor. Here, we see a sensitivity of 95% and a specificity of about 84%, for an overall accuracy of just over 88% on the 153 test MRI scans.

Tables V and VI show the confusion matrices for the training data with and without postprocessing. Interestingly, the accuracy on the training data is slightly lower than on the testing data, with 71.5% accuracy before postprocessing and slightly under 80% accuracy after processing. This could be because the testing data set is rather small, and could happen to have patients with more clear-cut cases of Alzheimer's Disease, while the training data set may contain a spectrum of severity in its cases.

TABLE III

TEST DATA CONFUSION MATRIX

	AD	**CN**
AD	**777**	**243**
CN	**477**	**1104**

TABLE IV

TEST DATA CONFUSION MATRIX WITH AVERAGING

	AD	**CN**
AD	**57**	**3**
CN	**15**	**78**

TABLE V

TRAIN DATA CONFUSION MATRIX

	AD	**CN**
AD	**5001**	**1119**
CN	**3419**	**6373**

TABLE VI

TRAIN DATA CONFUSION MATRIX WITH AVERAGING

	AD	**CN**
AD	**326**	**34**
CN	**162**	**414**

Chapter 5. Conclusions and Future Results

After multiple attempts at building a neural network model to accurately detect Alzheimer's Disease from MRI scan images, we achieved only somewhat satisfactory results. One consistent issue we are facing is our overfitting problem. Except for the transfer model with frozen parameters, all of our networks were capable of achieving close to 100% accuracy on the training dataset before significantly improving on the validation set. In our final model, we avoided this by using early stopping to end training after the validation data accuracy ceased to improve. This model proved to be our most successful classifier, achieving 88% test accuracy after averaging the results from multiple slices from each MRI image. Still, our single-image classification accuracy was still fairly low at about 72%, and our specificity metric consistently lagged behind our sensitivity.

One possible explanation for this inaccuracy is that our dataset is too small considering the amount of variation present within it. The MRI scans come with varying amounts and types of preprocessing performed on them, and differing resolutions and aspect ratios mean the brains appear different sizes after zero padding. These issues were apparently resolved by researchers at McMaster University[25] who performed extensive preprocessing on the ADNI dataset before inputting the MRI scans slice-by-slice into a CNN and using redundancy to determine the diagnosis. While these preprocessing steps are extremely computationally intensive and time consuming, this methodology yielded close to 100% accuracy. Researchers at universities in China and Australia [26] also achieved high accuracy (>90%) using a 3D CNN on a subset of the ADNI database that contained MRI scans of more uniform preprocessing.

For future work, we will continue to experiment with different preprocessing techniques to attempt to achieve higher overall accuracy. Additionally, transfer learning using more relevant datasets (e.g. brain tumor detection from MRI scans) may be another option to yield better results than using ImageNet-trained models.

References

Literature:

[1] *Dementia*, World Health Organization. Sep. 19, 2019. Accessed on: Nov. 11, 2019. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/dementia

[2] *Alzheimer's Disease Fact Sheet*, National Institute on Aging. May 22, 2019. Accessed on: Nov. 11, 2019. [Online]. Available: https://www.nia.nih.gov/health/alzheimers-disease-fact-sheet

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, Lake Tahoe, Nevada, 2012, pp. 1097-1105.

[4] A. A. M. Al-Saffar, H. Tao and M. A. Talab, "Review of deep convolution neural network in image classification," *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, Jakarta, 2017, pp. 26-31.

[5] H. Shin *et al*., "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning," in *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1285-1298, May 2016.

[6] Zhou, H. Yi, H. Xiaodong, L. Lui, Z. Li, C. Fan, S. Shanshan, and Ling, "Medical image segmentation with semi-supervised learning techniques," United States Patent 10430946, Mar. 14, 2019.

[7] H. E. M. Abdalla and M. Y. Esmail, "Brain Tumor Detection by using Artificial Neural Network," *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, Khartoum, 2018, pp. 1-6.

[8] E. M. Izhikevich, "Simple model of spiking neurons," in *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569-1572, Nov. 2003.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, D. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[10] A. Parvat, J. Chavan, S. Kadam, S. Dev and V. Pathak, "A survey of deep-learning frameworks," *2017 International Conference on Inventive Systems and Control (ICISC)*, Coimbatore, 2017, pp. 1-7.

[11] NVIDIA, "AI Supercomputer," DGX-1 Datasheet, Jun. 2017

[12] T. H. Oong and N. A. M. Isa, "Adaptive Evolutionary Artificial Neural Networks for Pattern Classification," in *IEEE Transactions on Neural Networks*, vol. 22, no. 11, pp. 1823-1836, Nov. 2011.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. New Jersey: Prentice Hall Press, 2010.

[14] T. Eo, Y. Jun, T. Kim, J. Jang, H.-J. Lee, and D. Hwang, "KIKI-net: cross-domain convolutional neural networks for reconstructing undersampled magnetic resonance images," *Magnetic Resonance in Medicine*, vol. 80, no. 5, pp. 2188–2201, 2018.

[15] Gunawardena, et al. "Applying Convolutional Neural Networks for Pre-Detection of Alzheimer's Disease from Structural MRI Data." An Introduction to Biometric Recognition - IEEE Journals & Magazine, Wiley-IEEE Press, 18 Dec. 2017, ieeexplore.ieee.org/document/8211486.

[16] L. Cao, L. Li, J. Zheng, X. Fan, F. Yin, H. Shen, and J. Zhang, "Multi-task neural networks for joint hippocampus segmentation and clinical score regression," *Multimedia Tools and Applications*, vol. 77, no. 22, pp. 29669–29686, 2018.

[17] Jha, D., Kim, J. and Kwon, G. (2017). Diagnosis of Alzheimer's Disease Using Dual-Tree Complex Wavelet Transform, PCA, and Feed-Forward Neural Network. *Journal of Healthcare Engineering*, 2017, pp.1-13.

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 2818-2826, doi: 10.1109/CVPR.2016.308.

[19] *embARC Machine Learning Inference Library*, Synopsys, Inc., 2009. Accessed on: May 22, 2020. Available: https://embarc.org/embarc_mli/doc/build/html/MLI_kernels/convolution_2d.html

[20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. New Jersey: Prentice Hall Press, 2010 pp. .

[21] *Neuro Science News*, NeuroScience News Inc. Sept 21, 2016, Accessed on May 12, 2020. Available: https://neurosciencenews.com/alzheimers-memory-neurology-5096/

[22] Google TPU, Advanced Guide to Inception V3 on Cloud TPU May 29, 2020, Accessed on May 2, 2020. Available https://cloud.google.com/tpu/docs/inception-v3-advanced

[23] TensorFlow, TensorFlow Batch Normalization June 3, 2020, Accessed on April 22, 2020 Available https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization

[24] Adam: A method for Stochastic Optimization Jan 30, 2017, Accessed on March 13, 2020 Available https://arxiv.org/pdf/1412.6980.pdf

[25] Sarraf, DeSouza, Anderson, Tofighi (2017) DeepAD: Alzheimer's Disease Classification via Deep Convolutional Neural Networks using MRI and fMRI, Accessed on May 23, 2020 Available https://www.biorxiv.org/content/10.1101/070441v4.full.pdf

[26] Luo, S.H., Li, X.C. and Li, J.M. (2017) Automatic Alzheimer's Disease Recognition from MRI Data Using Deep Learning Method. Journal of Applied Mathematics and Physics, 5, 1892-1898. https://doi.org/10.4236/jamp.2017.59159

Appendix A. Code Used For this Project

Project Title: Alzheimer's Disease Detection using ANN

Student's Names: James Poirier, Michael Tuttle

Advisor's Name: Dr. Yu

<u>Current Code for the Neural Network</u>

```
import tensorflow as tf
import os
import sys
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Conv3D, Dense, Flatten,
MaxPooling3D, Dropout
from tensorflow.keras.layers import BatchNormalization, Activation,
GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import prep_data as prep
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import numpy
from tensorflow.keras.constraints import max_norm

# Default image directories if not specified
if len(sys.argv) < 2:
    path_to_photos = os.getcwd() + '/ADNI_Pictures'
else:
    path_to_photos = sys.argv[1] + '/ADNI_Pictures'




# Locations of train, test, and val datasets
train_dir = os.path.join(path_to_photos, 'train')
val_dir = os.path.join(path_to_photos, 'validation')
test_dir = os.path.join(path_to_photos, 'test')


num_AD_tr = len(os.listdir(train_AD_dir))
num_CN_tr = len(os.listdir(train_CN_dir))

num_AD_val = len(os.listdir(val_AD_dir))
num_CN_val = len(os.listdir(val_CN_dir))


# Define batch size and training epochs
batch_size = 32
epochs = 50
# input dimensions
dim1 = 256
dim2 = 256
dim3 = 3
```

```
imgShape = (dim1, dim2, dim3)

init_model = tf.keras.applications.InceptionV3(include_top=False,
                                               weights='imagenet',
                                               input_shape=imgShape)



new_input = init_model.input
print(new_input)

init_model.trainable = True

# Random zoom in from 0% to 30% zoom
train_image_gen = ImageDataGenerator(rescale=1/255.0, zoom_range=[0.7, 1])
val_image_gen = ImageDataGenerator(rescale=1/255.0)
test_image_gen = ImageDataGenerator(rescale=1/255.0)
# Flow from directories to get data
train_data_gen = train_image_gen.flow_from_directory(batch_size=batch_size,
                                                     directory=train_dir,
                                                     shuffle=True,
                                                     target_size=(dim1, dim2),
                                                     color_mode='rgb',
                                                     class_mode='binary')

print(train_data_gen.class_indices)

val_data_gen = val_image_gen.flow_from_directory(batch_size=batch_size,
                                                     directory=val_dir,
                                                     target_size=(dim1, dim2),
                                                     color_mode='rgb',
                                                     class_mode='binary')


test_data_gen = test_image_gen.flow_from_directory(batch_size=batch_size,
                                                     directory=test_dir,
                                                     target_size=(dim1, dim2),
                                                     color_mode='rgb',
                                                     class_mode='binary')

model = tf.keras.Sequential([
    init_model,
    GlobalAveragePooling2D(),
    Dense(1, activation='sigmoid')
    ])

print(model.summary())


# define early stopping parameters with patience 5
# Restore best weights when finished
es = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5,
restore_best_weights=True)

# Define SGD and Adam optimizer parameters
optim = tf.keras.optimizers.SGD(learning_rate=0.01)
optim2 = tf.keras.optimizers.Adam(learning_rate=0.001)
```

```
# Use binary crossentropy loss function
model.compile(optimizer=optim2,
            loss='binary_crossentropy',
            metrics=['accuracy'])

#class_weights = { 0:1.25, 1:0.83}

# Train the model
history = model.fit(
    train_data_gen,
    steps_per_epoch=batch_size,
    epochs=epochs,
    validation_data=val_data_gen,
    validation_steps=batch_size,
    callbacks=[es]
)

# Evaluate on test dataset
score = model.evaluate(test_data_gen, steps=100, batch_size=48)

# Print training progress to figures
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
idx = 0
fname = 'ADnetAccuracy'
while(os.path.exists(fname + str(idx) + '.png')):
    idx += 1
plt.savefig(fname + str(idx) + '.png')
plt.close()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
plt.savefig('ADnetLoss' + str(idx) + '.png')
plt.close()
# Save the model
model.save('InceptionV3')
```

Helper Function to get photos from embedded folders
```
import os, sys


def getPhotos(path, pic_dest):
    entries = os.listdir(path)
    if path == []:
        return
    for entry in entries:
        if os.path.isdir(path + '/' + entry):
            nextPath = path + '/' + entry
            getPhotos(nextPath, pic_dest)
        elif entry.lower().endswith('.nii'):
            moveTo = pic_dest + '/' + entry
            moveFrom = path + '/' + entry
            os.rename(moveFrom, moveTo)
    return

if __name__ == "__main__":
    dirpath = os.getcwd()
    #dirpath = dirpath + '/ADNI'
    pic_dest = dirpath + "/ADNI_Pictures"
    if not (os.path.exists(pic_dest)):
        os.mkdir(pic_dest)
    getPhotos(dirpath, pic_dest)
```

Helper Function to sort photos once they are in their new folder
```
import csv
import os
import re

def sortPhotos(path):
    filename = 'ADNI1_Complete_1Yr_1.5T_3_02_2020.csv'
    files = []
    with open(filename) as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        for row in readCSV:
            files.append(row)
    sub_ID = 1
    group = 2
    #path = os.getcwd()
    photo_fold = path + '/ADNI_Pictures'
    if not os.path.exists(photo_fold):
        os.mkdir(photo_fold)
    MCI_fold = photo_fold + '/MCI'
    train_fold = photo_fold + '/train'
    val_fold = photo_fold + '/validation'
    train_fold_AD = train_fold + '/AD'
    train_fold_CN = train_fold + '/CN'
    val_fold_AD = val_fold + '/AD'
    val_fold_CN = val_fold + '/CN'
    folds = [photo_fold, MCI_fold, train_fold, val_fold, train_fold_AD,
train_fold_CN, val_fold_AD, val_fold_CN]
    for item in folds:
        if not os.path.exists(item):
            os.mkdir(item)
    imageDir = path + '/ADNI_Pictures'
    pic_num = 0
    for line in files[1:]:
        pic_num += 1
        subject = line[sub_ID]
        #regegg = "[]*(" + subject + "){1}[]*"
        label = line[group]
        print(subject)
        images = os.listdir(imageDir)
        for image in images:
            if not os.path.isdir(imageDir + '/' + image):
                isMatch = re.search(subject, image)
                if isMatch != None:
                    if pic_num % 10 == 0:
                        data_set = '/validation/'
                    else:
                        data_set = '/train/'
                    fname = imageDir + '/' + image
                    if label != 'MCI':
                        newName = imageDir + data_set + label + '/' + image
                    else:
                        newName = imageDir + '/' + label + '/' + image
                    print(label)
                    os.rename(fname, newName)
```

<u>Helper Function to generate data files and interpolate mis-sized photos</u>

```
import numpy as np
import nibabel as nib
import tensorflow as tf
import matplotlib.pyplot as plt
import os, sys, re
import cv2


def getFileNames(folder):
    filenames = []
    for root, dirs, files in os.walk(folder):
        for fname in dirs + files:
            name = os.path.join(root, fname)
            if re.search(r".nii", name):
                filenames.append(name)
    return filenames

def getDataLabels(filenames, lab0, lab1):
    labels = []
    for fname in filenames:
        if re.search(lab0, fname):
            labels.append(0)
        elif re.search(lab1, fname):
            labels.append(1)
    #    print(labels)
    return labels

def resizeTo(img, dim1, dim2):
    x = len(img)
    y = len(img[0])
    padx = (dim1 - x)//2
    pady = (dim2 - y)//2
    img2 = cv2.copyMakeBorder(img, padx, padx, pady, pady, cv2.BORDER_CONSTANT, 
value=[0])
    return img2

def saveToPNG(filename, data):
    dim1 = len(data)
    midPoint = round(dim1/2)
    for shift in range(-16, 17, 2):
        imgArray = data[midPoint+shift]
        fname = filename[0:-4] + "_shift" + str(shift) + ".png"
        #img = cv2.resize(imgArray, dsize=(256, 256), 
interpolation=cv2.INTER_LINEAR)
        img = resizeTo(imgArray, 256, 256)
        #print("X: " + str(len(img)) + "    Y: " + str(len(img[0])))
        #print(fname)
        #print("xdim: " + str(len(img)) + "     ydim: " + str(len(img[midPoint])))
        plt.imsave(fname, imgArray)
    return

def genPNGimages(dirpath):
    data_path = dirpath + "/ADNI_Pictures"
    train_path = data_path + "/train"
    val_path = data_path + "/validation"

    train_path_AD = train_path + "/AD"
    train_path_CN = train_path + "/CN"

    val_path_AD = val_path + "/AD"
    val_path_CN = val_path + "/CN"

    train_files = getFileNames(train_path)
    train_labels = getDataLabels(train_files, r"[\\/]AD[\\/]", r"[\\/]CN[\\/]")
    val_files = getFileNames(val_path)
    val_labels = getDataLabels(val_files, r"[\\/]AD[\\/]", r"[\\/]CN[\\/]")
    #print(train_files)
    #print(train_labels)
    print(len(train_files))
    print(len(train_labels))


    for idx in range(len(train_labels)):
        fname = train_files[idx]
        label = train_labels[idx]
        data = nib.load(fname)
        data_np = data.get_data().astype('int16')
        saveToPNG(fname, data_np)
        #print("x: " + str(len(data_np)) + "   y: " + str(len(data_np[0])) + " 
z: " + str(len(data_np[0][0])))


    for idx in range(len(val_labels)):
        fname = val_files[idx]
        label = val_labels[idx]
        data = nib.load(fname)
        data_np = data.get_data().astype('int16')
        saveToPNG(fname, data_np)
```

Helper Function to Test Accuracy

```
import tensorflow as tf
import os
import sys
import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.math import confusion_matrix


model = tf.keras.models.load_model('ADnet_inception')

if len(sys.argv) < 2:
    path_to_photos = os.getcwd() + '/ADNI_Pictures'
else:
    path_to_photos = sys.argv[1] + '/ADNI_Pictures'


test_dir = os.path.join(path_to_photos, 'test')
test_dir_AD = os.path.join(test_dir, 'AD')
test_dir_CN = os.path.join(test_dir, 'CN')

num_AD = len(os.listdir(test_dir_AD))
num_CN = len(os.listdir(test_dir_CN))

test_image_gen = ImageDataGenerator(rescale=1/255.0)
batch_size = 32
dim1 = 256
dim2 = 256

test_data_gen = test_image_gen.flow_from_directory(batch_size=1,
                                                    directory=test_dir,
                                                    shuffle=False,
                                                    target_size=(dim1, dim2),
                                                    color_mode='rgb',
                                                    class_mode='binary')
numImgs = 2601
labels = test_data_gen.labels
predictions = model.predict(test_data_gen, batch_size=1, verbose=1,
steps=numImgs)
tn = 0
tp = 0
fp = 0
fn = 0
for idx in range(0, len(predictions), 17):
    img_preds = predictions[idx:idx+17]
    img_label = np.around(np.average(img_preds))
    if img_label == 1 and labels[idx] == 1:
        tn += 1
    elif img_label == 0 and labels[idx] == 0:
        tp += 1
    elif img_label == 1 and labels[idx] == 0:
        fn += 1
    elif img_label == 0 and labels[idx] == 1:
        fp += 1
```

```
conf_mat_avg = [[tp, fn], [fp, tn]]
print(conf_mat_avg)
conf_mat = confusion_matrix(labels, np.around(predictions[:numImgs]))
print(predictions)
print(conf_mat)
```

Helper Function to prep data and show file paths

```
import tensorflow as tf
import os
import nibabel as nib
import numpy as np

# x = iter(iterable)
# y = x.get_next().numpy()
# z = y.decode("utf-8")
# data = nib.load(z)
# data_array = data.get_data()

CLASSNAMES = np.array(['AD', 'CN'])

def getLabel(filename):
    fpath = tf.strings.split(filename, os.path.sep)
    return fpath[-2] == CLASSNAMES

def getDataArray(filepath):
    img = nib.load(filepath).get_data()
    return img

def process_path(filepath):
    print(filepath)
    label = getLabel(filepath)
    fname = tf.io.read_file(filepath)
    print(fname)
    img = getDataArray(fname)
    return img, label
```

Appendix B. Senior Project Analysis

1. Summary of Functional Requirements

This project takes in MRI brain scan images and returns a classification score indicating whether the subject has Alzheimer's Disease.

2. Primary Constraints

The limited amount of available labeled MRI images of the brain constrains the neural network's ability to detect Alzheimer's Disease. Because machine learning systems' effectiveness generally increases with an increased amount of training data, more data would likely increase the accuracy of this system. Additionally, the time required to train the system limits number of different architectures we can test when trying to find the optimal configuration.

This project must follow the requirements and specifications discussed in previous sections to ensure that the system as a whole is worthwhile to implement for doctors. If the system is unable to meet the specifications, then it will not be useful to doctors and therefore not used in the field. Meeting the constraints set out is a must in this project to ensure that the final product is usable.

3. Economic

This project has various potential economic effects involving the medical field. Because the system performs a job normally done by humans, it contributes to the medical field's automation, and may result in less demand for human capitol such as doctors in this field. This can save large amounts patient money but potentially harm the medical professionals' employment. Simultaneously, this project makes use of MRI images, which require a large and expensive MRI machine to generate. Patients that want to take the Alzheimer's test may end up paying the cost of these machines through hospital bills.

To produce this system, the costs include the cost of server computational resources that train the model. At $0.83 per hour of cloud resources over an estimated 270 hours, the total price of training the system comes out to about $220. As estimated in the Gantt Chart, the project requires about 18 weeks and 360 person-hours to complete. As a result, we expect the total production costs to near $11,000 at a pay rate of $30/hour.

4. If Manufactured on a Commercial Basis

Because this project requires no physical components, there exist no manufacturing costs. We would make this system available for download over the internet, charging a subscription fee for its use. Because hospitals would likely purchase this system, we would price its use at $100 per month. Assuming at least ten hospitals implement this system, that raises $12,000 revenue per year, all profits due to the lack of manufacturing and operating costs. Hosting can be done securely and privately on Github, where the code can be distributed over a SFTP server between consumers of our software.

5. Environmental Impacts

The primary environmental impacts of this project arise during the training stage. Training neural networks requires a substantial amount of processing power performing constant computations over many hours. This uses energy which likely comes from non-renewable resources, polluting the atmosphere and stripping the planet's natural resources. After training the model, system usage has comparatively minor environmental effects, as classifying images requires significantly less processing power than training the model. That said, if used on a large-scale, the combined energy from running the system as well as taking the MRI scans used as inputs becomes significant and could lead to the same environmental impacts as training. Processing power required to run this program is negligible once the network has been trained, since it is simply a complicated algorithm that will only take a few seconds to determine its result.

As for other species, this system could potentially reduce the number of number of animals used for medical testing. Should the neural network classify Alzheimer's Disease with sufficient accuracy, it could reduce the need for researchers to test other detection methods using animal subjects.

6. Manufacturing

This system consists of entirely software components, meaning the system will not face any challenges regarding manufacturing. Distribution of this system can occur online with no associated costs using Github and SFTP servers.

7. Sustainability

Once trained, this device should not require maintenance to continue performing at a constant level. However, as new data becomes available and researchers develop improved machine-learning models, updating this system may become necessary. Unfortunately, for new neural network models, updating the system requires retraining the entire system from scratch. This means repeating the energy-expensive training phase each time the system moves to a more effective model.

Updating this system would prove particularly troublesome, if the healthcare industry moved away from MRI scans as a diagnosis tool. This would require retraining the system on the new medical image types that hospitals offer patients. This means creating an entirely new set of labeled Alzheimer's brain scan images, which could take significant time.

8. Ethical

A Utilitarian perspective can provide insight into the ethical implications of this project. Under this ethical framework, one considers a project ethical if the total happiness or good created from it exceeds the total unhappiness created. This project attempts to provide a reliable method for detecting Alzheimer's Disease from MRI images of the brain. This could potentially aid in the prevention or mitigation of serious cognitive deterioration or death for many people. This prevents not only the misery of those spared of Alzheimer's Disease, but also those people's friends and families who would otherwise witness their loved one's slow mental decline.

On the other hand, any detection scheme that relies on machine learning models will doubtlessly output both false negatives and false positives sometimes False negatives may delay the treating of Alzheimer's Disease and result in a more severe condition some patients, while false positives have the potential to distress healthy patients that have no need to worry. Because of this, the system's accuracy is imperative in determining the ethics of its use. Furthermore, the system automates a sector of the healthcare industry that currently uses human expertise to diagnose Alzheimer's Disease. This could reduce the number of employment opportunities for doctors in this field, resulting in hardship for the doctors and their families.

To consider this Alzheimer's Detection system ethical under the Utilitarian perspective, the benefits of early Alzheimer's detection must outweigh both the harms of false diagnosis and the hardships faced by any doctors adversely affected by this system. Because the system specifications necessitate greater than 80% sensitivity and greater than 90% specificity, it remains reasonable to assume that the benefits from correct diagnosis far exceed the harms of false diagnosis. Furthermore, doctors currently rely on imperfect detection methods for diagnosing Alzheimer's so even a small improvement over current methods would yield net positive results. Finally, the system likely has minimal effects on doctors' employment in this field, as human diagnosis can confirm or correct the system's classification. Overall, a detection system meeting the accuracy specifications of this project almost certainly yields a net benefit to society.

The IEEE code of ethics also provides an effective framework for analyzing the ethical implications of this project. Because this project strives to provide the most accurate medical diagnosis based on the available data, it meets the code of ethics' first clause regarding the importance of public health and safety. The third clause also relates closely to this project, stating the importance of accurately portraying the system's capabilities. To avoid giving both doctors and patients undue confidence in the results output of this system, we measure the accuracy of this system on separate testing images and include the results of these tests in the project report. This report also helps the project meet the fifth clause of the code of ethics by discussing the social and ethical implications of implementing intelligent systems in human experts' place. Finally, this project addresses the IEEE code of ethics' seventh clause by crediting all sources influential the developing of this system. This group's members have also aligned our behavior to meet the IEEE code of ethics' remaining guidelines.

9. Health and Safety

As this project performs medical diagnosis, using this system creates health concerns. Specifically, the system requires accurate diagnosis to avoid potentially harming the patient's health. False negatives result in delayed detection of Alzheimer's Disease and potentially limit treatment options' effectiveness. On the other hand, false positives can cause undue distress for the patient and even trigger unnecessary and harmful treatment to take place. The system can also help catch Alzheimer's early however, and provide doctors a greater sense of confidence in their diagnosis when agreeing with a second party.

10. Social and Political

This system's creation carries with it some inadvertent social and political consequences. To start, this system automates a job normally done by trained medical professionals. This means potentially limiting the usefulness of doctors trained in Alzheimer's diagnosis and, thus, harming their employment. The direct stakeholders this project impacts include patients seeking early detection of Alzheimer's Disease and the doctors who provide this service. Indirect stakeholders include the hospitals containing the MRI equipment and the companies manufacturing MRI machines. Of these stakeholders, the doctors pay the highest cost, as they gain more effective diagnostic tools at the cost of becoming somewhat more replaceable. Patients gain the obvious direct benefit of Alzheimer's diagnosis, while hospitals may benefit from charging patients to get MRI brain scans. MRI companies may benefit indirectly from increased demand for their products.

11. Development

In the developing of this project, we learned to use a variety of different machine-learning tools, including Google's machine-learning framework, TensorFlow, and the high-level python library Keras. We also learned a variety of different neural network models that can effectively analyze and classify images. Through our literature search, we were able to discover other similar projects that attempted to fulfill the same goal as ours, and their recommendations and suggestions on how to further research in neural networks performing in the medical field.