BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
COMPUTER SCIENCE IN ROMANIAN SPECIALIZATION

DIPLOMA THESIS
Alzheimer's Disease Detection. Classification between the early and severe stages using a convolutional neural network based on 2D MRI scans

Supervisor
Prof.dr. Horia F. Pop

Author
Ichim Ștefan

2024

Abstract

Alzheimer's Disease (AD) is a neurodegenerative disorder characterized by progressive cognitive decline, affecting memory, thinking, and behavior. This work aims to enhance early detection of AD using advanced machine learning techniques, focusing on deep learning models such as Convolutional Neural Networks (CNNs). The study involves the analysis of a brain imaging modality, specifically the structural Magnetic Resonance Imaging(sMRI) scan, to identify biomarkers associated with AD. A comprehensive dataset from the ADNI Initiative is preprocessed through methods like skull-stripping and normalization. The proposed approach leverages a modified ResNet architecture to classify Alzheimer's stages and assess model performance through various experiments. Results indicate significant improvements in diagnostic accuracy, suggesting the potential of deep learning models in clinical applications for early AD detection. Future work will explore the integration of additional data types and further model optimization to enhance robustness and its level of generalization.

Contents

1 Introduction

There is no denying that humanity stands at a previously inconceivable point in healthcare and medicine, which naturally have led to hindrances in senescence, with populations increasingly reaching older stages of life. Furthermore, studies which take into account multiple case scenarios show that population is expected to reach 9.2 billion by the age of 2050, leading to an uprise of 21% in the elderly. [15]

With that being said, researchers' concern has taken a turn towards diseases occurring in these later parts of human lives, some of which are considered treatable while others less so. One of such disorders is Alzheimer's Disease, or AD, considered to be the most likely precursor of dementia. Alzheimer's Disease is a brain disease, neurodegenerative, which in time diminishes cognitive skills such as memory, thinking and speaking, and in due course even removes the ability of accomplishing simple activities vital to one's daily life.

On top of that, it is an incurable disorder, which only underlines even further the reasons why early detection is of such great importance, so that appropriate actions can be taken by both the medical team and the one diagnosed, along with their relatives and close ones.

1.1 Disease Summary

The brain of a healthy human represents a cluster of neurons by the number of billions, which together amount to what actions and reactions we have through a process of signal propagation. Through our sensory mechanism, which includes hearing and seeing, receptors carry out the tasks of sending signals (Fig 1) using designated channels all the way to the neurons inside the brain, where new specific signals are formed and sent back, resulting in what we call actions. [25]

Alzheimer's Disease intervenes in this process by gradually decreasing the utility function of each neuron, leading to the atrophy of the brain's proficiencies, as neurons imminently die one by one.

There are three major factors included in the dynamic between AD and neurons. First of all, a key advantage of neurons that many other cells lack, and which accomplishes their long survival depends on their ability to repair themselves, form new connections, or changing current ones' magnitude. Secondly, synaptic connections, which solidify the signal transmission process, and lastly, the intake of glucose and oxygen necessary for their normal functioning. It is believed these fundamental attributes of a healthy human receive considerable drawbacks upon the disease's presence. [21]

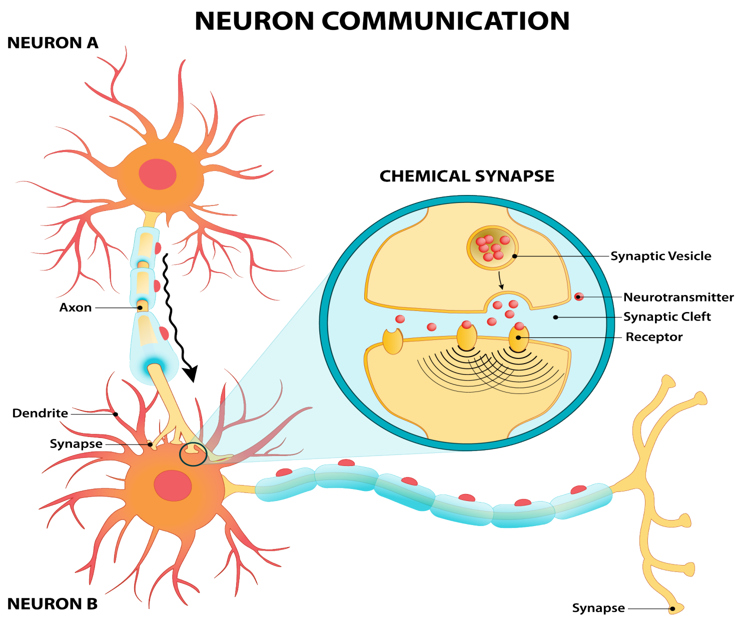


Figure 1: Communication between neurons. Source: MHA [18]

1.2 Causes

While the factors that lead to Alzheimer's Disease are not yet properly understood, past research and studies prove that some of the most commonly met criteria which lead to a diagnosis include genetic inheritance - chances of developing Alzheimer's Disease increase by 30% when another close relative suffers from it [12], lifestyle and environmental factors.

Genetical Inheritance

Genes represent instructions passed down from generation to generation, which contain information regarding how various cells need to behave. Some roles played by these include defining one's height, or the color of hair and eyes.

Advances in genetic research have led to the discovery of 80 genetic areas that can possibly play a part in AD development [20]. One of the more known genes which raises the risk of Alzheimer's Disease is the apolipoprotein E (APOE) gene, which comes in forms such as $\epsilon_2, \epsilon_3, \epsilon_4$. A pair of two such APOE genes, one from each parent, gets passed down to the next generation resulting in six possible cases. Among them, the (ϵ_4, ϵ_4) combination having the highest risk of AD, only increasing, not guaranteeing it, and in contrast, ϵ_2 provides a higher degree of protection against it.

External Factors

Besides genetic inheritance, researchers have drawn conclusions regarding causes of Alzheimer's Disease to contain a plethora of other outside factors, which we can have a higher influence on. Among these can be found vascular conditions - high blood pressure, heart diseases - and metabolic diseases - obesity and diabetes [20].

1.3 Symptoms

Before beginning the discussion about its effects, a noteworthy fact is that brain structure modifications, whether they may be neurofibrillary tangles or plaques of amyloid, occur several years before any cognitive issues manifest at all, a stage of the disease titled preclinical. With that being said, their presence does not inevitably lead to dementia.

Apart from the preclinical stage, AD has been classified into three others: mild, moderate, severe.

Early-stage (Mild)

A person which suffers from early-stage Alzheimer's Disease can still function normally on their own, without mandatory outside benefactors. However, changes appear in memory skills, starting to forget recently gained information, such as names at social gatherings, objects placements and losing the reasoninng behind starting certain activities.

It is important to understand these memory setbacks are hardly noticeable by the affected one, more commonly than not leaving it up to their surrounding group of people and friends to pinpoint them and initiate medical visits.

Middle-stage (Moderate)

Here, over the course of many years, cognitive skills start degrading, with the diagnosed person needing increasing help from other people. Previous rare memory losses become the norm, and even more proeminent. Not only that, disturbances in emotions begin escalating, some expressed in a stronger tone, while others hardly able to be exhibited at all.

Daily tasks must be simplified to the level the person with Alzheimer's can accomplish them, and as the external attention needed rises, place them in special care centers where experienced caretakers can easily reach out.

Late-stage (Severe)

This final stage of AD is categorized by vital losses in the ability to function at all. Patients stop reacting to outside factors altogether, and even initiating conversations. In due course, pain becomes impossible to verbalize, and as such, hourly check-ups are necessary. [2]

1.4 Diagnosis Process

AD diagnosis can only be carried out upon gathering a variety of complex data, which includes medical history, assessments of cognitive and physical skills, neurological exams, brain scans, blood tests and cerebrospinal fluid.

Medical history consists of modifications in how the patient behaves over the course of time, past and present medical concerns and even the undergoing medication. Other than these, information about other family members' health conditions is obtained, since, as previously mentioned, genes do play a role in increasing the risk, or protecting against Alzheimer's Disease.

An overall health status is evaluated, involving commonly met questions about diet, blood pressure and pulse, checking the quality of breathing and sample taking for testing.

Cognitive tests' purpose is to express a general view whether memory impairment takes a toll in the daily life of the diagnosed, and to shed light on the awareness of the disease. A number of tests are simple - tasks of remembering sequences of words, or mathematic operations, but there also exist those that take a longer period of time, alongside with raised levels of attention.

The neurological examination typically implies assessing the patient's nervous system, where a physician tries to distinguish between the possibilities of the disease to be a different brain disorder instead of AD - brain tumors or Parkinson's Disease.

Brain imaging is used to form 3D and 2D, functional and structural scans of the brain, through which experts can point out characteristics specific to Alzheimer's Disease. One such mark is the presence of higher concentrations than normal of amyloid beta ($A\beta$ or Abeta), peptides considered the essential part of amyloid plaques. This specific part of the diagnosis process typically serves only as a last resort. [3]

1.5 Imaging Modalities Involved in Alzheimer's Disease

Through recent technological advancements, brain imaging's role has shifted to a crucial one. By and large, imaging has expanded into various different modalities, each with their own strengths and weaknesses, but combined lead to a better analysis of AD's effects on the human brain. [13]

Amyloid PET

Amyloid Positron Emission Tomography is a non-invasive technique, which locates amyloid plaques. Due to its unavailability and expensive price, this method has not seen much popularity, but there is no denying its accuracy, past studies showing that 96% of the people who had performed an amyloid PET scan and were amyloid positive had been diagnosed with Alzheimer's Disease. [13]

FDG PET

Fluoro-deoxy-D-glucose (FDG) PET showcases synaptic activity, because the brain's primary energy comes from glucose. The Fluorine part of the FDG comes as a consequence of the convenient dynamic it has with Positron Emission Tomography, which easily detects it.

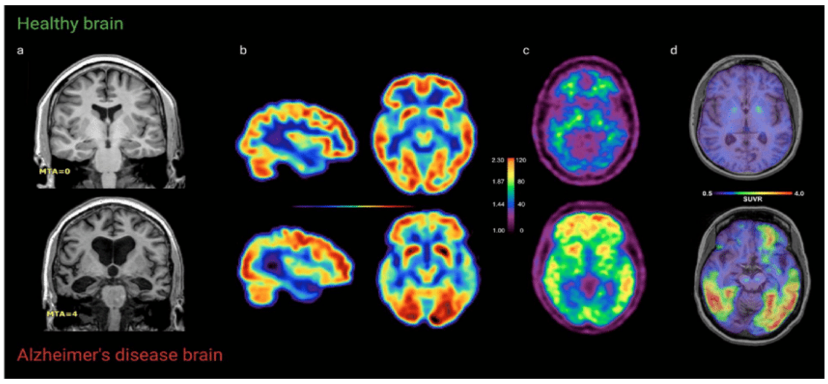


Figure 2: Neuroimages of the healthy versus the Alzheimer's disease (AD) brain. Neuroimaging with (a) structural MRI, (b) FDG-PET, (c) amyloid-PET with PiB and (d) tau PET with 18 F-AV1451 in both healthy and AD brains. Source: (Oostveen and Lange [22])

Structural MRI

Structual Magnetic Resonance Imaging (sMRI) is a non-invasive method applied to observe pathology and anatomy of the brain by emitting radiofrequency pulses in sequences. Its main purpose is to exhibit brain atrophy, associated with shrinking size due to neuron counts declining. One of its drawbacks is that, compared to PET imaging, hallmarks of AD cannot be detected, and also atrophy isn't specific to the disease discussed.

Functional MRI

As the previous method, the functional variant of MRI is non-invasive as well, but, on the other hand, provides scientists a neuronal activity mapping of the brain. A few of them require the patient to perform certain cognitive tasks during the scanning process, and there are also some which need the brain to be found in a specific resting state. Functional MRI's setback is the necessity of lack of motion, and any of the patient's movements could lead to faulty data.

Tasks

An important question to ask before traversing further, is what type of predictions are we demanding from ourselves? As aforementioned, Alzheimer's Disease classifies itself in three stages: Normal Cognition(NC), Mild Cognitive Impairment(MCI) - which further breaks down into progressive MCI and stable MCI, pMCI and sMCI respectively - and Alzheimer's Disease.

These various disease progressions have given birth to a collection of tasks for researchers. There are approaches which classify AD in two - NC vs AD, NC vs MCI, MCI vs AD -, three - NC vs MCI vs AD - and even four classes, including MCI's subclasses. On the other hand, regression problems build the percentage of a specific stage to progress.

Deep Learning

Recent advancements in deep learning, with a higher degree of respect to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks(RNNs), have had main roles in raising the speed and accuracy with which classification or regression task are accomplished.

Before 2013, the most commonly met algorithms included stacked Restricted Boltzmann machines and stacked autoencoders, but ever since, CNNs and RNNs have taken the spotlight, surprising the world with their results, especially when given inputs of MRI or PET [1].

Deep Learning's key characteristic is the power to find a hypothesis leading to the desired outcome unbeknownst to the humans of how it is realized, with the drawback of needing a high amount of data and proportionate computational resources.

Artificial Neural Networks

As their name suggests, ANNs consist of a collective of artifical neurons connected between eachother. They represent how researchers have tried to emmulate the biological neural brain, where the nodes play the role of neurons, and connections, or edges, between them that of synapses.

Furthermore these neurons are stratified into various layers, through which information passes, notably the initial and last layer have been titled as input and output accordingly, while the ones in-between called hidden layers.

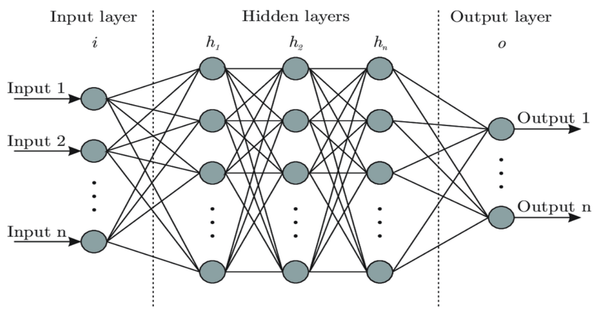


Figure 3: A general artificial neural network architecture. Source: Bre et al. [4]

Each neuron's meaning is to receive information and process it in order to be passed to the next layer through the array of synapses. Activation functions take place in the processing stage, in order for non-linearities to be applied to the hypothesis being built.

In the course of multiple iterations, these neurons adapt to the task given by the designer, adaptation formally known as weights.

Convolutional Neural Networks

These types of artifical neural networks are feedforward - input flows only in forward direction, without loops - and exhibit the capacity to extract features automatically. In such networks, convolutional layers (4) and pooling layers represent their main components. Convolutional layers transform the data by passing it through a kernel creating a feature map , and pooling layers finetune network parameters by taking these feature maps and reducing their size. This final product also takes the name of pooled feature map [17].

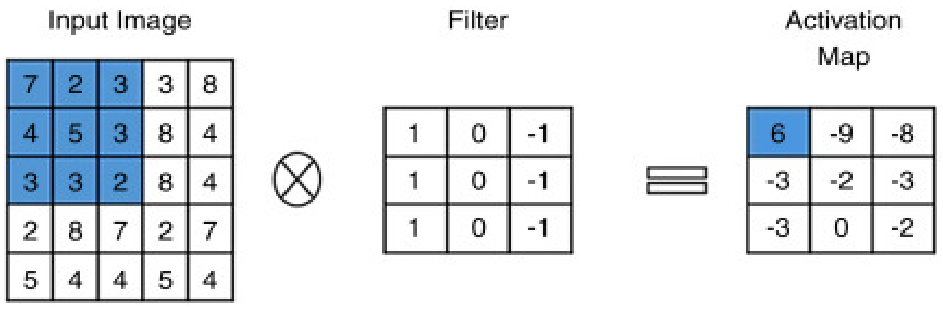


Figure 4: How a convolution filter is applied to an image channel. Source: ScienceDirect [24]

Recurrent Neural Networks

RNNs are the other types of neural networks, described best by how information propagates inside it, which compared to the single directioned CNNs, RNNs are bi-directional. Fig (5) displays for for each input x_t, information from the previous state also has agency over the prediction, just as this current state x_t will have impact for the following x_{t+1} state. This property creates the opportunity of some information to be used in more than just one place, with the justification that evaluations sometimes show improvements in results.

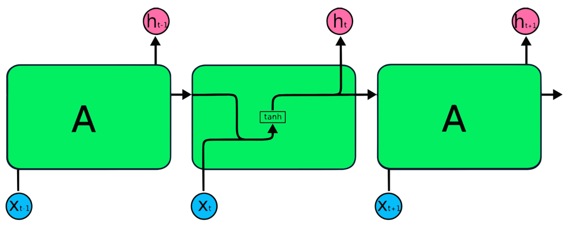


Figure 5: A simple RNN cell. Source: DataCamp [8]

1.6 Regions of Interests

This section serves as a guide to understanding why certain regions of the brain deserve more attention than others while studying the Alzheimer's Disease.

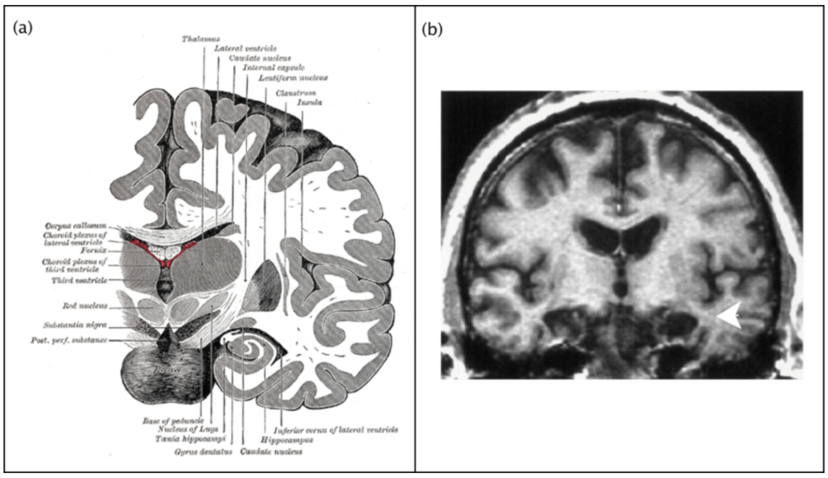


Figure 6: (a) Illustration showing the location of medial temporal lobe structures on coronal section (taken from Gray's Anatomy). (b) Coronal T1-weighted MRI scan of HM's brain, showing the absence of the hippocampus and related medial temporal lobe structures bilaterally (arrowhead). Source: Butler [5]

Medial Temporal Lobe (MTL)

MTL represents a brain region known for its capacities of handling memory abilities related to space and episodes [7]. Previous works prove that it is among the first to suffer from shrinkage over the course of AD's effect [9], mainly due to how soon tangles of neurofibrilaries develop (NFT). Upon the beginning of NFTs, they start spreading further in the network, ultimately reaching the neocortex, region which realizes reasoning, meaning and consciousness.

2 Related Work

Jong Bin Bae et al. [14] suggest a Convolutional Neural Network approach where, in contrast to other CNN approaches, they give as input 2D-MRI scans, with attention to the medial temporal lobe (MTL) from coronal slices, MTL being widely considered to be the area suffering most change. Even though other parts of the brain contain valuable information as well, they deem it unnecessary due to probabilities of altering the algorithm's outcomes. They make use of ADNI[1] and SNUBH[2] datasets in order to highlight the importance of each patient's background for the predictions, such as ethnicity and educational levels, by which the human brain is thought to vary accordingly. Their approach include evaluations through AUC (area under the receiver operating characteristic curve), accuracy, sensitivity and specificity - for ADNI: 88%, 83%, 76%, 89%, and for SNUBH: 89%, 82%, 79%, 85% . They have also underwent within and between dataset comparisons, having used 2 datasets: 91%-94% for within-dataset and 88%-89% for between.

On the other hand, Li et al. [16] provide a 3D-MRI approach using CNNs where the a region of the medial temporal lobe is taken into consideration this time, the hippocampus. The used datasets include 3 stages from the ADNI study: 1, GO & 2 and the AIBL[3]. Their main concern is that of predicting the progression from the moderate to servere stage along with a time estimation, underlining that previous MCI and AD comparisons only classify MCI in either progressive (pMCI) or stable (sMCI). Accuracies for each dataset are the following: ADNI with 76.2%, AIBL with 78.1%.

Ebrahimi and Suhuai [10] realize multiple trials with different architectures and input modalities, 2D and 3D MR scans, however making use only on a small amount of 132 subjects from the ADNI dataset before dividing in the corresponding purposes to the algorithm. Their best results come from SqueezeNet and ResNet-18 on 2D inputs using transfer learning. For measurements of accuracy, sensitivity and specificity, ResNet had achieved 84.38%, 87.5%, 81.25% respectively, while SqueezeNet's results were 90.62%, 81.25% and 100%.

Nguyen and colleagues tackled a Recurrent Neural Network method for the TADPOLE[4] Challenge [19]. Corresponding to the competition's recommendation, a set of 23 variables was used as inputs which included multi-modal imaging, cognitive test results and clinical diagnosis. The research had tested the minimalRNN architecture [6] which was modified properly in order for the problem to become 6-class labeling problem: NC stable, NC progressive, MCI recovered, MCI stable, MCI progressive and AD stable. Their trials were verified using multiclass area under the curve (mAUC)

[1]Alzheimer's Disease Neuroimaging Initiative
[2]Seoul National University Bundang Hospital
[3]Australian Imaging Biomarkers and Lifestyle Study of Aging
[4]The Alzheimer's Disease Prediction Of Longitudinal Evolution

and balanced class accuracy (BCA) with the scores of 94.5% and 88%. One of their unique noteworthy contribution is their ways of handling missing data at consecutive time points, which consisted of three variations: firstly, model-filling, done by the model during training and testing, and the next two were considered as preprocessing techniques: forward-filling and linear-filling using linear interpolation. Model-filling coming out on top for the minimalRNN solution.

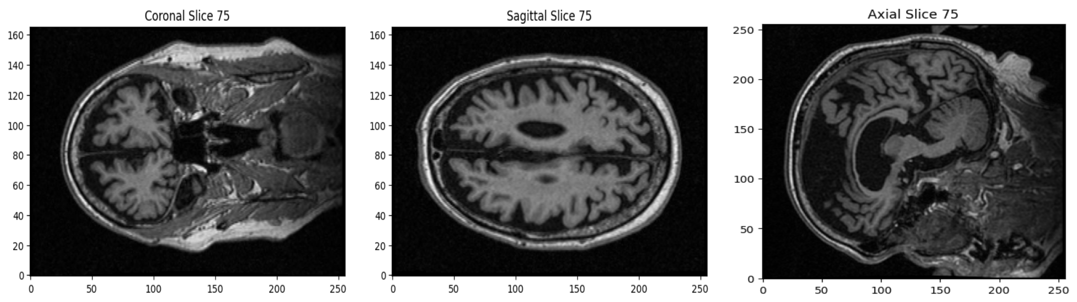
Article	Architectures	Performance Measures	Results
Jong Bin Bae et al.(2020)	2D-CNN	AUC, acc, sens, specif	88%, 83%, 76%, 89%
Li et al.(2019)	3D-CNN	acc	76.2%, 78.1%
Ebrahimi and Sunhuai (2021)	SqueezeNet Resnet-18	acc, sens, specif	84.38%, 87.5%, 81.25% 90.62%, 81.25%, 100%
Nguyen et al. (2020)	minimalRNN	mAUC, BCA	94.5%, 88%

Table 1: A breakdown of the state-of-the-arts

3 Dataset

The data used in this research has been taken from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database (adni.loni.usc.edu). The initiative started in 2003 led by Principal Investigator Michael W. Weiner, MD as a public-private partnership. Its primary objective has been to verify if serial magnetic resonance imaging (MRI), positron emission tomography (PET), clinical and neuropsychological assessments, as well as other biological markers can together measure the progression of mild cognitive impairment (MCI) and early Alzheimer's Disease (AD) [23].

Images are taken using structural MR imaging (sMRI) which display brain atrophies. These scans can be found in NiFTi format (.nii) and using the *nibabel* library in python, which was designed for the purpose of supporting neuroimaging file formats, they can be accessed and plotted as 2D images, according to a specific plane - coronal, sagittal or axial (Fig 7).



(a) Coronal Slice (b) Axial Slice (c) Sagittal Slice

Figure 7: Visualisation of an sMRI scan

Due to the large amount of size such complex scans come in, this study only takes into consideration the first phase of the ADNI Initiative, ADNI1 which included 448 unique subjects, with multiple visits, in total reaching up to 1512 entries. With that being said, only 852 entries also were accompanied by their respective mask used for preprocessing, which is later described. At the end, the data consisted of 381GB worth of NiFTi files.

Furthermore, images came in different shapes due to which scanner had been used: 52% entries followed [256 x 256 x 166] and 45% had the [240 x 256 x 160] size. Unfortunately, reshaping such scans to a standard cannot be achieved using simple algorithms due to how much each small pixel matters, point solidified by the reason for these files to be stored in a special format on their own.

FreeSurfer[5] is one such software which handles most problems in data preprocessing for MRI scans, which wasn't useable in this research due to its operating software limitations, along with required available space.

[5]Harvard [11]

Thus, the input had been reduced to 52% of valid entries, 300 cognitive normal scans and 187 from the AD group. As a consequence of this class imbalance, up-sampling was applied, which will be later described.

4 Preprocessing

In Data Science, preprocessing plays the valuable role of preparing data to be given to the learning algorithm. In order for the inference to unfold expectedly, the models expect inputs to follow their standards, and it becomes noticeable in their later evaluations if necessary preprocessing methods had not been utilised.

Out of the most commonly known and met preprocessing methods in computer vision, this study includes, skull-stripping, oftenly considered in any brain scans related machine learning algorithms, due to their high level noise reduction, min-max normalization with the main purpose of reducing pixel values to the interval of [0, 1] and finally, an up-sampling of a specific class in the dataset, required for balancing.

As a final note before detailing each technique, ADNI claims the segmentation masks provided may not also be the most accurate ones, resulting in research from the FreeSurfer team to come up with the highest quality brain extraction mechanisms. Even in fig 8 (c) it can be observed there still remain slight edges from the skull, and the final input scans will not only include the brain. Furthermore, another brain-related preprocessing step, image registration, could not be applied with the same reason of inability to use Harvard's software with complex algorithms.

4.1 Skull-Stripping

Firstly, as a preprocessing step, the skull-stripping technique takes place, which is realized by applying a segmentation mask to the original versions of the MRI. Segmentation masks represent black and white images, where the white area symbolizes the region of interest of the image to be applied on. These have been provided by the ADNI, and so only the algorithm which applies such a mask over an original scan needs to be written.

As it can be seen in fig 8, what is called background noise, or unnecessary additional pixels, get swept away, in our case being anything except of the human brain.

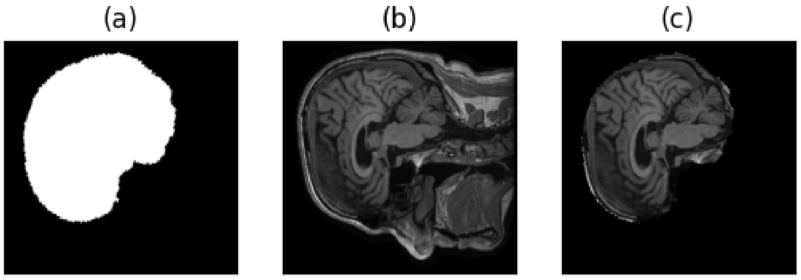


Figure 8: Skull-stripping using a segmentation mask over a sagittal slice: (a) the segmentation mask, (b) the original version of the scan, (c) the final result after applying the segmentation algorithm

4.2 Data visualisation

With the stripping step explained, we can now pay attention to a few differences of the brains between the cognitive normal and the Alzheimer's Disease group.

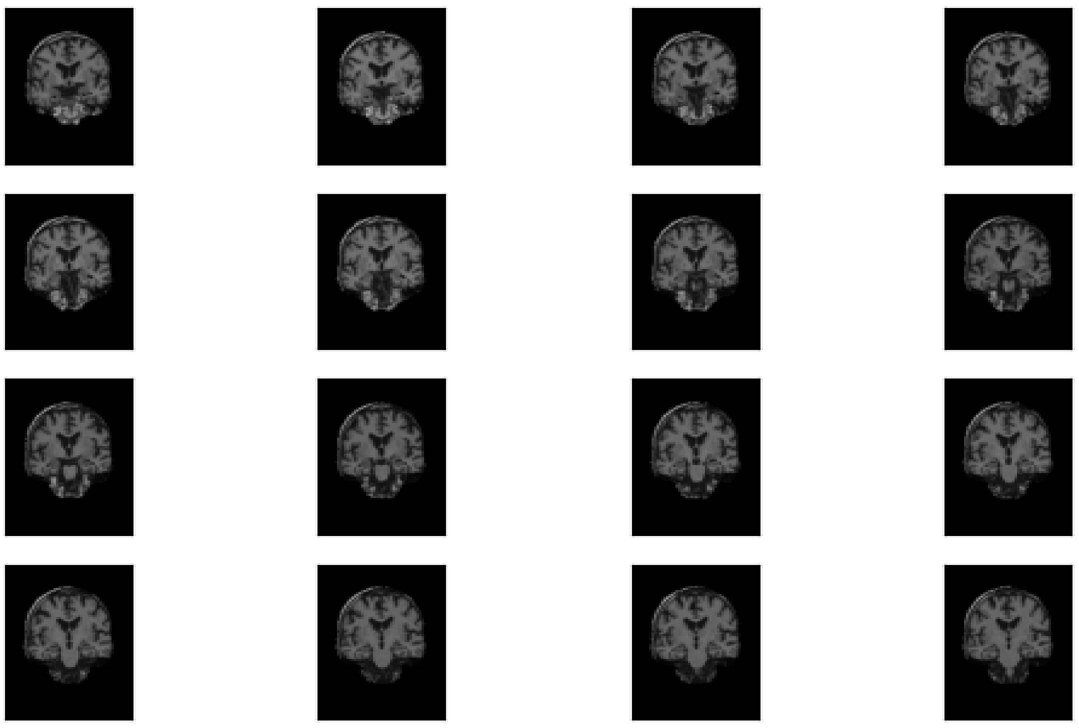


Figure 9: CN patient - 16 slices around the hippocampus

As it can clearly be noticed, brain atrophies at the hippocampus of an Alzheimer's Disease diagnosed person (fig 10) can be much more easily observed compared to that of a normally cognitive brain (fig 9). As previously noted, our main concern in these scans are about the Medial Temporal Lobe area, especially the hippocampus which resides in it. And as such, the models will consider the slices which surround a part of said hippocampus.

4.3 Min-Max normalization

Normalization lies at the foundation of machine learning preprocessing. Normalization, or scaling takes place in order for all input data to be bound to the same interval of values, and as little that interval is, the easier for the models to find similarities between each entry. With that being said, values most popuarly become scaled down to two interval values: [-1, 1] or [0, 1].

In this study, a min-max normalization algorithm is applied to the scan slices, which reduces pixels between 0 and 1. Each pixel going through the following formula:

$$x_{i,j} = \frac{x_{i,j} - x_{min}}{x_{max} - x_{min}}$$

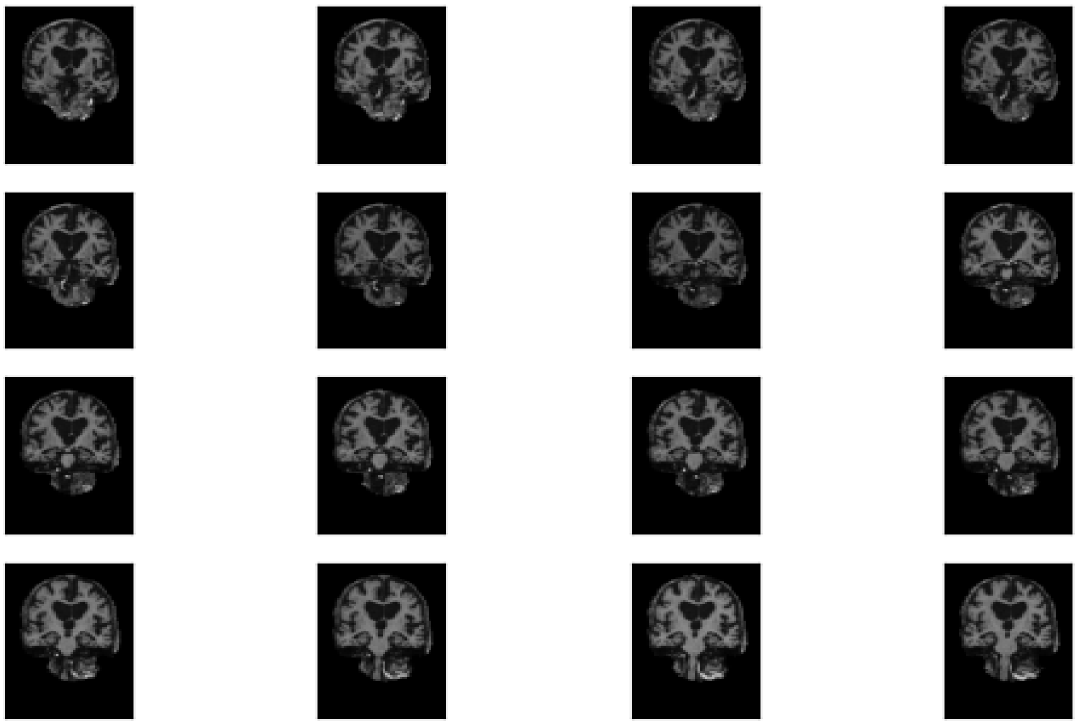


Figure 10: AD patient - 16 slices around the hippocampus

where x_{min} and x_{max} represent the smallest and largest pixel values of the original image.

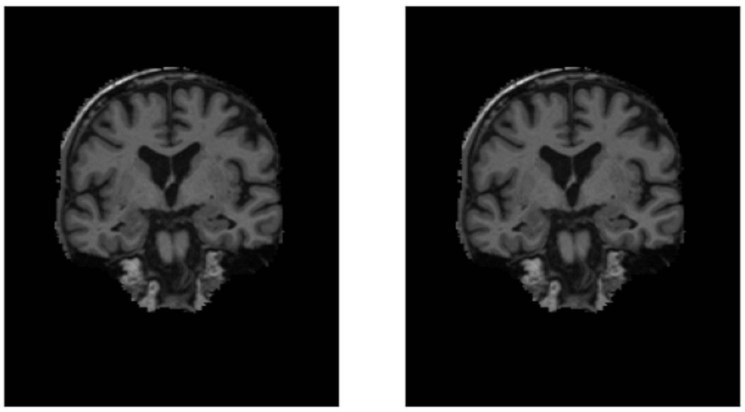


Figure 11: Min-max normalization, left image average pixel: 168.2, right image average pixel: 0.079

4.4 Up-sampling

Following up the imbalance in the number of diagnostics per class in the dataset, a certain method must be taken so that our dataset is equally balanced. At random, sufficient cases of Alzheimer's Disease group were duplicated up until the point of equality in cases with the Cognitive Normal class.

4.5 Image Registration

Image registration is the action of aligning each MRI scan to the same given coordinate space. It is under no circumstance possible for each subject to sit in the same position while scanners take the images. As mentioned before, FreeSurfer handles these complex pixel processing techniques due to their complex algorithms, and as such this step was skipped.

5 Proposed Approach

In this study, there were several experiments which had been carried out surrounding networks from the ResNet family.

Firstly, a ResNet101 modified architecture was tested, with multiple slices and even one slice in the center around the hippocampus serving as inputs. Having so many parameters, the multiple slices approach had no possibility of training and testing in appropriate time, thus an approach with only one slice as input was also attempted, which had led to higher training loop speed and better results.

Second of all, after having studied the modified ResNet101, its 18 layer counterpart was also given opportunity. One of the main advantages was that with the little amount of parameters ResNet18 comes with, the multiple slices choice had also become available which gave considerably better results compared to the single slice ResNet101. Notably, the single centric slice input for the ResNet18 had showed less significant results.

5.1 ResNet

Up until the mid 2010's, researchers had started exploring the impact of adding multiple layers in neural networks, and, to a certain degree of surprise, they found out that inreasing layer size will not necessarily lead to better results. This issue was mainly due to the appearance of vanishing or, in contrast, exploding gradients. These gradient events occur in the back-propagation step, when they suffer that big amount of changes that they span out of control, either too small to be of any impact - the vanishing gradient - or large enough to single-handedly distort the weights unbelievably so.

However, in 2015, Microsoft Research experts launched the Residual Network architecture. They introduced the concept of Residual Nodes (fig 12.b) which allows the model to skip through layers in both the inference and back-propagation steps resulting in the elimination of previous gradients concerns: vanishing or exploding.

For this model, each convolutional layer is followed up by a batch normalization, then the ReLu activation function. Conv layers serve to learn patterns and contours in the feeded images, batch normalization re-scales what results from the convolutions and finally the activation function has the purpose to introduce non-linearity at the layer outputs, which permits the model to distinguish even more complex connections between each given scan.

5.2 Modifications

Upon the collection of data from the ADNI Initiative, it was noticed, like other diagnostic-related databases, it didn't come only with 3D T1-weighted MRI scans and their respective label. Among the data, details for each subject about their age, sex

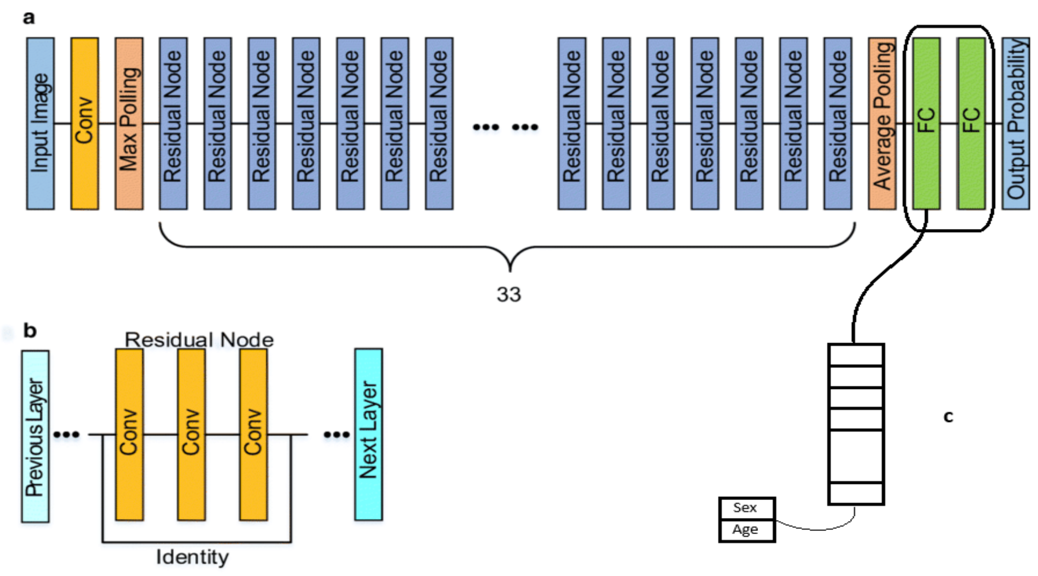


Figure 12: ResNet101 and its modifications, (a) - Model Architecture, (b) - Residual Nodes, (c) - Study modifications

and their visit count could also lead to an improved hypothesis for the studied model.

Considering that these models only want to classify whether an image should be categorized as a cognitive normal or Alzheimer's Disease type, and nothing related to time progression, they were modified to take into account the age and sex for each of the subjects, by expanding the final fully-connected layer with these 2 values, and also adding another one to follow it up, before predicting the probabilities for each class (fig 12.c).

In addition, the input channels for the first convolutional layer were also modified to the number of slices given as inputs, two models receiving one slice, and the other two 8 slices around the hippocampus region, each of these slices having the size [256 x 166].

5.3 Inference

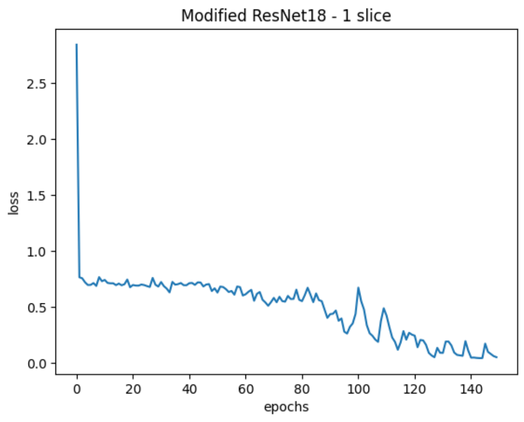
The models have been trained using an NVIDIA GeForce RTX 2060, with 6GB of GPU memory. The considered loss function was binary cross entropy(log) loss, given by:

$$BCE\_Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

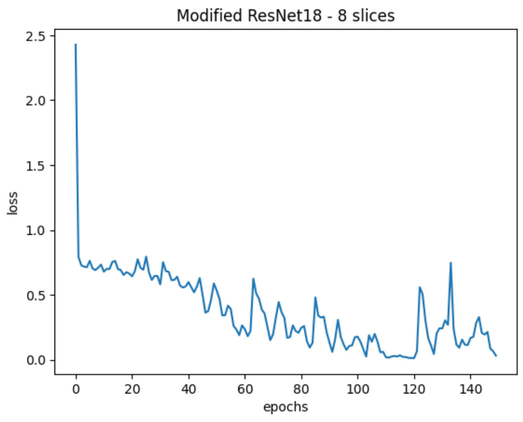
where y is the ground truth label, 0 for CN and 1 for AD, and $p(y)$ represents its respective probability.

As for the hyperparameters for all networks studied, the Adam optimizer was used with $\alpha = 0.01$ as learning rate, and the batch size of 32. The training loop was established to stop after 150 epochs.

Fig (13) consists of training losses side-by-side comparison between the 1 slice and 8 slices variants.



(a) 1 slice input modality



(b) 8 slices input modality

Figure 13: Loss progressions for the modified ResNet18 models

It can be noticed that the 8 slice modality [Fig 13 (b)] reaches better loss quicker than the 1 slice input model [Fig 13 (a)] , which in turn succeeded to bring forth better results at the performance measuring step.

6 Results and Experiments

Experiments were made on both ResNet101 and ResNet18 with the specified modifications, with the latter offering better results. The best performing model takes as input eight slices of the coronal region, at the hippocampus corpus and decides on the probabilities for each wanted class.

The data was shuffled and split using the 70/15/15 rule, using three sets: for training, developing and testing, meaning a set of 420 scans, and two sets of 90 respectively. The metrics used to evaluate the study's models were accuracy, sensitivity and specificity, with the best results being 93.2%, 95.8%, 93.5% for the eight slices input ResNet18 modified model, and 89.4%, 95.4%, 89.1% for the single slice version on the validation set.

Thus it can be observed that the eight slice model takes the upper hand compared to the single slice.

The results were used to build up a confusion matrix, from which the accuracy (1), sensitivity (2) and specificity (3) formulas were applied.

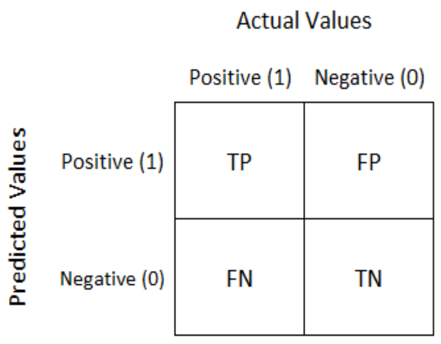


Figure 14: Confusion matrix

For a better understanding, TP refers to when the prediction predicts a positive correctly, FP when a positive is incorrectly predicted. TN and FN fall under the same rule, but with negatives.

Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of the total number of predictions. It provides a general idea of how well our models are performing overall. With that being said, it alone can be misleading, especially in cases where the data is imbalanced. This was, however, tackled using an up-sampling technique.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Sensitivity measures the ratio of actual positives that are correctly identified by the model. Its main attention is on the model's ability to correctly identify positive cases.

$$sensitivity = \frac{TP}{TP + FN} \tag{2}$$

In contrast to sensitivity, specificity measures the percentage of true negatives that are correctly identified.

$$specificity = \frac{TN}{TN + FP} \tag{3}$$

With the results calculated for each measure, the following comparison can be made with other state-of-the-art models.

Article	Architectures	Performance Measures	Results
This study	1-slice Modified ResNet-18 8-slice Modified ResNet-18	acc, sens, specif	89.4%, 95.4%, 89.1% 93.2%, 95.8%, 93.5%
Jong Bin Bae et al.(2020)	2D-CNN	AUC, acc, sens, specif	88%, 83%, 76%, 89%
Ebrahimi and Sunhuai (2021)	SqueezeNet Resnet-18	acc, sens, specif	84.38%, 87.5%, 81.25% 90.62%, 81.25%, 100%

Table 2: Comparison with the state-of-the-arts which give 2D input or use the same architecture for 3D

7 Application

7.1 Libraries

Throughout this study, a plethora of python frameworks were made use of which lead to an overall easier and more controlled environment. To begin with, one of the most popular libraries when working in python for any reason is numpy. For plotting loss progressions, matplotlib's pyplot was used and for working with file paths, the pathlib and os libraries were used, which were necessary for handling how the data was downloaded from the ADNI.

```python
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
from pathlib import Path
import os
```

For working with the NiFTi files, characteristic to how Magnetic Resonance Imaging scans are stored, nibabel took that role. NiBabel is used to access and preprocess a vast array of neuroimaging data files. Thus, it provides an interface to manipulate common neuroimaging file formats, by allowing easy reading and writing. Besides the used NiFTI file format, NiBabel allows work with formats such as: ANALYZE, MINC(Medical Imaging NetCDF), MGH/MGZ (formats used by the FreeSurfer), CIFTI and GIFTI.

NiBabel allows preprocessing of these complex files by loading them into numpy arrays, which one can modify with the neccessary changes in mind, according to the preprocessing steps taken before the model training loop.

Having loaded these files into numpy arrays, they can later be visualized either through slicing or 3D region extracting through careful index manipulation of these arrays.

```python
import nibabel as nib
```

Pandas came in handy with maneuvering .csv files, from which subject data was procured. Pandas is a powerful data analysis tool which enables high-performance data structures, making it easier to work with structured data.

```python
import pandas as pd
```

And finally, for the main driver of this study, in terms of the machine learning models, layer architectures, activation functions and optimizers, it was PyTorch. As well as sklearn for other a few evaluation methods.

```python
import torch
from torch.utils.data import DataLoader, Dataset, random_split

import sklearn
import sklearn.metrics
```

In summary, the complete list of library imports includes numpy, datetime, matplotlib.pyplot, pathlib, os, nibabel, pandas, torch and sklearn.

```python
    import numpy as np
    from datetime import datetime
    import matplotlib.pyplot as plt
    from pathlib import Path
    import os
    import nibabel as nib
    import pandas as pd

    import torch
    from torch.utils.data import DataLoader, Dataset, random_split

    import sklearn
    import sklearn.metrics
```

7.2 Dataset Manipulation and Preprocessing

Upon receiving the data from the ADNI website, it was observed it came with a .csv file, which would classify each subject and a respective scan to a certain group, along with other details regarding the pair such as sex, age, tools used for the scans and or if preprocessing had been applied.

```python
    dataset_dir = Path(os.getcwd()) / "Datasets" / "ADNI" / "ADNI1"

    imgs_dir = dataset_dir / "Images"
    masks_dir = dataset_dir / "SegmentationMasks"
    store_dir = dataset_dir / "InputImages"

    csv_files = dataset_dir.glob("**/*.csv")
    for csv_file in csv_files:
        csv_data = csv_file
```

Using Pandas, data was read from the .csv file, which was used to find scans and their respective segmentation masks, which if both existed would lead to applying the mask and creating a new image, stored in the /SegmentationMasks subdirectory.

Since mentioned previously, ADNI data came with original scans and segmentation masks. Thus, an algorithm needed to be used which would apply the masks over the original images and form the segmented scans.

The apply_mask (img_arr, mask_arr) function takes as input the result of what NiBabel reads from NiFTi files for the original unprocessed image and also the segmentation mask corresponding to the original image. It was observed their axes didn't correspond, and as such a rotation had to take place, before applying the mask itself. Once the rotation takes place, a 180' around the third shape (index 2), the segmentation is applied using an array-level operation.

```python
def apply_mask(img_arr: np.ndarray, mask_arr: np.ndarray) -> np.ndarray:
    img_arr = nib.load(img_path).get_fdata()
    mask_arr = nib.load(mask_path).get_fdata()

    mask_arr = np.transpose(mask_arr, (2, 1, 0))
    segmented_arr = img_arr.copy()

    if img_arr.shape != mask_arr.shape:
        return None

    # Mask and image have different orientations
    for i in range(mask_arr.shape[2]):
        mask_arr[:, :, i] = np.rot90(mask_arr[:, :, i], k=2)

    segmented_arr[mask_arr == 0] = 0
    return segmented_arr
```

This part of the script iterates through the dataframe opened using Pandas and extracts the acquired date for each subject. It then proceeds to look for a scan and its mask corresponding to that pair of subject and date, which if found are used to form a new scan, where skull would be stripped away from the rest of the original version. Using NiBabel's save function, this outcome is stored back as a .nii file, in order for no significat voxel to be lost in this process.

```
for index, row in df.iterrows():
    # converting date formats to match the directory formats for easier finding
    acq_date = row["Acq Date"]
    date = convert_date(acq_date) # function converts to "yyyy-mm-dd"
    subject = row["Subject"]

    # skip already saved mask applications
    if os.path.exists(f"{store_dir}/{subject}__{date}.nii"):
        continue

    subject_img_dir = imgs_dir / subject
    subject_mask_dir = masks_dir / subject

    img_subdir = search_subdirectories(subject_img_dir, date)
    mask_subdir = search_subdirectories(subject_mask_dir, date)

    # skip if we don't find the mask for an image
    if img_subdir == None or mask_subdir == None:
        continue

    img_path = list(img_subdir.glob(f"**\*.nii"))[0]
    mask_path = list(mask_subdir.glob(f"**\*.nii"))[0]

    img_arr = nib.load(img_path).get_fdata()
    mask_arr = nib.load(mask_path).get_fdata()

    segmented_arr = apply_mask(img_arr, mask_arr)
    if segmented_arr is None:
        continue

    nifti_img = nib.Nifti1Image(segmented_arr, np.eye(4))
    nib.save(nifti_img, f"{store_dir}/{subject}__{date}.nii")
```

By the end of this script, we are expected to have our data stored in three separate subdirectories: /Images, /SegmentationMasks and /InputImages. The /Images and /SegmentationMasks subdirectories contain the original unprocessed scans, and some of their segmentation masks, which came through from the ADNI public dataset upon request. Once the .csv is iterated and groups of scans and masks are found, the apply_mask() algorithm outputs a resulting preprocessed scan, which would be serving as inputs later for the models.

With the segmentation masks applied on the original images, we can now separate the data. This was done using PyTorch's Dataset, which was later iterated through by DataLoaders. These two work together splendidly because the Dataset class provides access to data with a given index, and the DataLoaders can handle hyperparameters changes easily, such as batch sizes, or data shuffling type, while also iterating through the Dataset.

The following code defines a custom class 'ADNI1_Dataset' which inherits from 'torch.utils.data.Dataset', from the PyTorch framework. This class is designed to handle and preprocess a specific set of data. The constructor takes as parameters an arrays of strings, additional data which come in pairs of (sex, age) and lastly an array of labels corresponding to each previous image path and additional data. Also, label_dict is used for describing how the labeling of the two classes had been done from strings of characters to integers, and lastly input_shapes would partake in finding out which would be the index of the center coronal slice for each different types of shapes of the scans.

```python
class ADNI1_Dataset(Dataset):
    def __init__(self, img_paths, additional_data, labels, label_dict, input_shapes):
        self.img_paths = img_paths
        self.additional_data = additional_data
        self.labels = labels
        self.label_dict = label_dict
        self.input_shapes = input_shapes

    def __len__(self):
        return len(self.img_paths)

    def __getitem__(self, idx):
        img = read_image(self.img_paths[idx])
        coronal_slices = separate_coronal_slices_around(img, self.input_shapes[img.
    shape], 4)
        coronal_slices = torch.tensor(coronal_slices, dtype=torch.float)
        return coronal_slices, torch.tensor(self.additional_data[idx]), self.
    label_dict[self.labels[idx]]
```

'read_image(path)' is a simple function which uses NiBabel's load() and get_fdata() functions to store the NiFTi file found at the given 'path' into a numpy array.

```python
def read_image(path):
    input_img = nib.load(path).get_fdata()
    return input_img
```

The defined function 'min_max_normalize(image, new_min=0, new_max=1)' applies the min_max preprocessing algorithm, which bounds the pixel values between a given interval, in our case using the default values of the interval [0, 1].

```python
def min_max_normalize(image, new_min=0, new_max=1):
    min_val = np.min(image)
    max_val = np.max(image)
    normalized_image = (image - min_val) / (max_val - min_val) * (new_max - new_min) +
     new_min
    return normalized_image
```

The 'separate_coronal_slices_around()' takes as input an already read scan into a numpy array and returns 'slices_count' amount of slices around the 'slice_number'. In the '__getitem__(self, idx)' method of the custom 'ADNI1_Dataset', it is used after reading a segmented scan in order to obtain the coronal slices which will be used as inputs for the models. Before returning, they have to inevitably be converted into tensors, the equivalent of numpy arrays with which PyTorch's algorithms function.

```python
def separate_coronal_slices_around(img: np.ndarray, slice_number: int, slices_count:
    int) -> [np.ndarray]:

    slices = []
    for slice_no in range(slice_number - slices_count, slice_number + slices_count +
    1):
        slice = np.rot90(img[:, slice_no, :], k=2)
        slice = min_max_normalize(slice)
        slices.append(slice) # coronal section slice
    return slices
```

Before initializing one such custom Dataset, the constructor's parameters need to be built: the image paths, additional data for each subject, their respective labels, the class labeling and which index represents the middle coronal slice given a specific shape.

This was facilitated by iterating once again through the dataframe and only taking into account preprocessed scans which exist for all subjects in the .csv file. Furthermore, there was noticed an obvious class inbalance, specifically 300 cases of CN and 187 of AD, and as such two variables were used to determine an upsampling technique described later.

Given how our ResNet models only work with images with the same shape, it was decided that the shape with the highest scans to be used (256 x 256 x 166). Especially since resizing preprocessing wasn't available in this study.

```python
img_paths = []
labels = []
additional_data = []

input_shapes = {
    (256, 256, 166) : 128,
    # (192, 192, 160) : 86,
    # (240, 256, 160) : 128,
}

num_cn = 0
num_ad = 0

for index, row in df.iterrows():
    group = row["Group"]
    subject = row["Subject"]
    date = convert_date(row["Acq Date"])
    path = input_dir / f"{subject}__{date}.nii"
    input_img = read_image(path)

    # if image doesn't exist or doesn't follow shape criteria don't add it
    if input_img is None or input_img.shape not in input_shapes.keys():
        continue

    group = row["Group"]

    if group == 'CN':
        num_cn = num_cn + 1
    elif group == 'AD':
        num_ad = num_ad + 1

    sex = 0 if row["Sex"] == 'M' else 1
    age = row["Age"]

    additional_data.append((sex, age))
    img_paths.append(path)
    labels.append(group)
```

This upcoming section of the script applies the up-sampling technique with the 'num_cn'(CN cases number) and 'num_ad'(AD cases number) used for determining how much upsampling needs to be done for their amounts to be equivalent. An array is used to memorize which indices had been previously duplicated, so that only unique entries get doubled.

```
upsample_count = num_cn - num_ad
already_inserted = []

for idx, data in enumerate(img_paths):
    group = labels[idx]
    if group == 'AD' and upsample_count > 0:
        upsample_count = upsample_count - 1
        if idx not in already_inserted:
            already_inserted.append(idx)
            img_paths.append(img_paths[idx])
            labels.append(labels[idx])
            additional_data.append(additional_data[idx])
```

With the preparation finished, we can proceed in establishing our main dataset, which will be split using the 70/15/15 rule, in order for us to have one training set, on which the model will change his algorithm on, one development set which will be used to see after each epoch how it evaluates and one testing set used for a final performance measure.

```
label_dict = {
    "CN": 0,
    "AD": 1,
}

dataset = ADNI1_Dataset(img_paths, additional_data, labels, label_dict, input_shapes)

```

```python
train_size = int(0.7 * len(dataset))
dev_size = int(0.15 * len(dataset))
test_size = len(dataset) - train_size - dev_size

train_data, dev_data, test_data = random_split(dataset, [train_size, dev_size,
    test_size])

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
dev_loader = DataLoader(dev_data, batch_size=32, shuffle=True)
test_loader = DataLoader(test_data, batch_size=32, shuffle=True)
```

7.3 Model Architecture

The model's architectures depend on the number of layer in our ResNet. First of all, the 'BasicBlock' Class defines a basic residual block with only two convolutional layers, to which batch normalization is applied. The 'shortcut' is the key component of the ResNet architectures, its purpose being that of permitting the inputs to bypass certain amount of layers, which leads to the elimination of the gradient's problems of vanishing or exploding. In the forward pass, the outputs are activated using the ReLU function, which applies max(0, x) for all outputs.

Besides the 'BasicBlock', used for the 18 layered version, the 'BottleNeck' class is a more complex variant of a residual block, used in deeper architectures of residual networks, such as the 101 layer version. These however include three convolutional layers, each followed by a batch normalization. And again, the important shortcut, specific to residual networks.

```python
# module: /models/ResNet.py
import torch
import torch.nn as nn
import torch.nn.functional as F

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,
    padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1,
    bias=False)
        self.bn2 = nn.BatchNorm2d(planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=
    stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride, padding
    =1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion*planes, kernel_size=1, bias=
    False)
        self.bn3 = nn.BatchNorm2d(self.expansion*planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion*planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion*planes, kernel_size=1, stride=
    stride, bias=False),
                nn.BatchNorm2d(self.expansion*planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

Having defined our residual blocks, 'BasicBlock' for the 18 layer version, and 'Bottleneck' for ResNet101, we can proceed with defining a general 'ResNet' class, which will be used in two functions, depending on which type of architecture we require.

Our model begins with a 7 x 7 convolutional layer, followed by a batch normalization and maxpooling. From here on, 4 residual blocks are created through which a downsampling happens thanks to the stride values. At the end of these residual blocks, an average pooling takes place.

Coming up are the modifications made to the original architecture, which can be described by firstly expanding the supposedly last fully-connected layer with our additional data, and another fully-connected layer after which the final outputs are generated.

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=1000):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(512*block.expansion, 512*block.expansion)
        self.fc2 = nn.Linear(512*block.expansion + 2, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x, y):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.maxpool(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avgpool(out)
        out = torch.flatten(out, 1)
        out = F.relu(self.fc1(out))

        # append (age, sex) to this layer
        out = torch.cat((out, y), dim=1)

        out = self.fc2(out)
        return out
```

These next two functions are used for swift model creation, which return different types of ResNet's depending on the name of the function. It can be observed the 101 architecture receives as 'num_blocks' the values of [3, 4, 23, 3], which lead to the creation of 101 layers, while the 18 architecture receives as 'num_blocks' [2, 2, 2, 2], which in turn builds up only 18 layers.

```
def ResNet101(num_classes):
    return ResNet(Bottleneck, [3, 4, 23, 3], num_classes)

def ResNet18(num_classes):
    return ResNet(BasicBlock, [2, 2, 2, 2], num_classes)
```

7.4 Training Loop

Before the training loop, the model needs to be initialized. Importing the ResNet module leads to access to the defined functions which facilitate that very model creation we desire. Upon loading, the first convolutional layer is modified to take as input 8 channels, for our 8 slices, and the last fully-connected layer is modified to output 2 classes, as per our problem requires: AD or CN. It is finally sent to the device available.

```python
import models.ResNet as RN

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

num_classes = len(label_dict.keys())
num_slices = 8

model = RN.ResNet18()
model.conv1 = torch.nn.Conv2d(num_slices, model.conv1.out_channels, kernel_size=model.
    conv1.kernel_size, stride=model.conv1.stride, padding=model.conv1.padding, bias=
    model.conv1.bias)
model.fc2 = torch.nn.Linear(in_features=model.fc2.in_features, out_features=
    num_classes, bias=True)

model.to(device)
```

As for the training loop, it is firstly preceeded by setting up hyperparameters. As specified, our models will be trained for 150 epochs, with the Adam optimizer having the 0.01 learning rate, and our loss function will be CrossEntropy. 'train_losses' and 'dev_losses' will be used for plots of the loss progression throughout the epochs.

```python
num_epochs = 150
optimizer = torch.optim.Adam(params=model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()

train_losses = []
dev_losses = []
```

The training loop itself begins by obtaining consecutive batches from the training dataloader. From here, our necessary inputs and their ground truth labels are extracted. Our model makes predictions for each entry in the current batch, and the back propagation step follows: first, every gradient gets cleaned, due to the optimizer calling 'zero_grad()'. Secondly, the loss is computed and by calling 'backward()' the gradients are calculated for each neuron. This being followed up by the optimizer's 'step()' leads to also apply these calculated gradients and thus modifying our weights.

At the end of each batch, accuracy is shown, as well as a confusion matrix, which was available through sklearn's 'confusion_matrix' method from 'metrics'.

A similar process applies to the development set before moving on to the next epoch, except that the optimizer will not apply the computed gradients.

A techinque called early stopping is used, which saves the weights of the last highest performing model on the dev set during the training iterations. At the end of the training loop, a comparison is made between the final state and the last highest performing on the development set, with the better performing one on the test set remaining as the final state.

```python
prev_best_eval_acc = -1
version = 1
iter = 0

for epoch in range(num_epochs):
    print(f"epoch = {epoch + 1}/{num_epochs}")
    train_running_loss = 0
    n = 0
    acc = 0
    conf_matrix = np.zeros((2, 2))
    for batch_idx, data in enumerate(train_loader):
        n = n + 1
        if batch_idx % 10 == 0:
            print(f"batch id = {batch_idx}")

        imgs, additional_data, labels = data[0].to(device), data[1].to(device), data[2].to(device)
        outputs = model(imgs, additional_data)

        optimizer.zero_grad() # cleans gradients
        loss = criterion(outputs, labels)
        loss.backward() # computes gradients
        optimizer.step() # applies gradient modifications

        train_running_loss += loss.item()
        acc = acc + torch.sum(torch.argmax(outputs, dim=1) == labels) / len(labels)
        preds = torch.argmax(outputs, dim=1)
        conf_matrix += sklearn.metrics.confusion_matrix(labels.cpu().numpy(), preds.cpu().numpy(), labels=np.arange(2))

    print(conf_matrix)
    train_losses.append(train_running_loss / n)
    print(f"train loss = {train_running_loss / n}, acc = {acc / n}")


    dev_running_loss = 0
    conf_matrix = np.zeros((2, 2))
    n = 0
    acc = 0
    for i, data in enumerate(dev_loader):
        imgs, additional_data, labels = data[0].to(device), data[1].to(device), data[2].to(device)

        outputs = model(imgs, additional_data)
        loss = criterion(outputs, labels)
        dev_running_loss += loss.item()

        acc = acc + torch.sum(torch.argmax(outputs, dim=1) == labels) / len(labels)
        conf_matrix += sklearn.metrics.confusion_matrix(labels.cpu().numpy(), torch.argmax(outputs, dim=1).cpu().numpy(), labels=np.arange(2))

    print(conf_matrix)
    print(f"dev loss = {dev_running_loss / n}, acc = {acc / n}")
    dev_losses.append(dev_running_loss / n)
    if acc / n > prev_best_eval_acc:
        print(f"new best acc: {acc / n}")
        version = version + 1
        prev_best_eval_acc = acc / n
        dt_save_path = f"weights/model_resnet_{iter}_{version}.pth"
        torch.save({
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'epoch': epoch,
            'loss': loss
        }, dt_save_path)
```

At the end of the training, our model is saved at the post training 'pt_save_path' location.

```
    pt_save_path = 'model_resnet18_8slices.pth'
    torch.save({
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'epoch': epoch,
        'loss': loss
    }, pt_save_path)
```

7.5 Results

Thus, with our model finishing training, we can move on to iterating over the test set using the test dataloader. In an alike manner, data is extracted from the current batches, which are used for obtaining our model predictions. Using these predictions and our ground truth labels, we form a confusion matrix from which the TP, TN, FP, FN values are extracted and used in formulas of the accuracy, sensitivity and specificity.

```
conf_matrix = np.zeros((2, 2))

n = 0
acc = 0
sens = 0
spec = 0

for i, data in enumerate(test_loader):
    if i % 10 == 0:
        print(f"batch_id = {i}")
    n = n + 1

    imgs, additional_data, labels = data[0].to(device), data[1].to(device), data[2].to
    (device)

    outputs = model(imgs, additional_data)

    conf_matrix += sklearn.metrics.confusion_matrix(labels.cpu().numpy(), torch.argmax
    (outputs, dim=1).cpu().numpy(), labels=np.arange(2))

    TN = conf_matrix[0, 0]
    FP = conf_matrix[0, 1]
    FN = conf_matrix[1, 0]
    TP = conf_matrix[1, 1]

    sens = sens + TP / (TP + FN)  # True Positive Rate (TPR)
    spec = spec + TN / (TN + FP)  # True Negative Rate (TNR)
    acc = acc + torch.sum(torch.argmax(outputs, dim=1) == labels) / len(labels)

print(conf_matrix)
print(f"eval loss = {test_running_loss / n}, accuracy = {acc / n}")
print(f"sensitivity = {sens / n}, specificity = {spec / n}")
```

7.6 Deployment and Demo

The models studied in this article take part in the demo application using the 'gradio' python library. This framework creates the opportunity to integrate models using a graphic user interface, where necessary inputs are passed onto a prediction of our model. A label will be outputed to the user's eye, which represents the model's prediction.

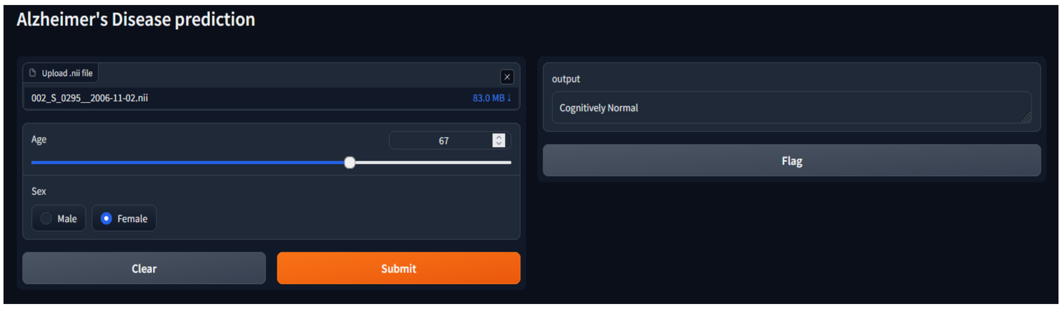
This code section displays how one such interface can be specified to have certain inputs, with gradio's Interface properties. An array of inputs is specified, one for our MRI scan, in the form of a NiFTi file, one for the subject's age for whom we will predict and finally a radio button for the sex.

```python
import gradio as gr

demo = gr.Interface(
    fn=predict,
    inputs=[
        gr.File(label="Upload .nii file"),
        gr.Slider(1, 100, step=1, label='Age'),
        gr.Radio(['Male', 'Female'], label='Sex', type='index')
    ],
    outputs='text',
    title="Alzheimer's Disease prediction"
)

demo.launch()
```

The application's graphic user interface is shown in Fig (15), where each specified input can be seen. Upon the pressing of the submit button, the function passed to the 'fn' property of gradio's Interface, will take as parameters the three inputs, and its output will be displayed in the right-hand side textbox.


Alzheimer's Disease prediction

Upload .nii file: 002_S_0295__2006-11-02.nii 83.0 MB ↓

Age: 67

Sex: Male [✓ Female]

output: Cognitively Normal

Figure 15: Demo GUI

The following section represents our prediction function, which, as mentioned, will take three parameters, taken from the user interface after pressing the orange 'Submit' button.

As a consequence of the fact that our inputs are special type of files which are required to be read with the NiBabel library, we will have to reopen the file using the 'read_image()' function. With the file being loaded properly, we can check if it matches our model input shapes. If this check is also bypassed, the coronal slices will be extracted, transformed into tensors, and treated as if we would send to the model a batch size of one input, by unsqueezing the first dimenison of our tensor.

Importantly to not forget, the pair of sex and age also take the same route, before both will be sent into a model inference iteration. With the outputs at hand, we will return to the GUI a string, using the 'predict_labels' dictionary, which returns "Cognitively Normal" if our prediction is 0, or "Alzheimer's Disease" otherwise.

```
predict_labels = {
    0: "Cognitively Normal",
    1: "Alzheimer's Disease",
}

input_shapes = {
    (256, 256, 166) : 128,
}

def read_image(path):
    input_img = nib.load(path).get_fdata()
    return input_img

def predict(file, age, sex):
    img = read_image(file.name)
    if img.shape not in input_shapes:
        return "Incorrect file dimensions, try another"
    coronal_slices = separate_coronal_slices_around(img, input_shapes[img.shape], 4)
    coronal_slices = torch.tensor(coronal_slices, dtype=torch.float).to(device)

    x = coronal_slices.unsqueeze(0)
    y = torch.tensor((sex, age)).unsqueeze(0).to(device)
    output = model(x, y)
    return predict_labels[torch.argmax(output).item()]
```

8 Discussion and Future Work

This study explores ResNet18 and ResNet101 approaches with a few adaptations in their architectures, which classify a given amount of slices at the center of the brain, the hippocampus into 2 classes, Normal Cognition and Alzheimer's Disease.

To begin with, these models predict only one probability for every given amount of slices. One other attempt could be to obtain a prediction for each slice and then apply a mean for a final prediction.

Improvements can also be made upon the preprocessing step, with a proper utilization of the FreeSurfer software, of which many studies in the state-of-the-art make use. Firstly, an accurate segmentation would take place, where no margins from the skull could be left standing. And secondly, the image registration and scaling would lead to more useable images, and more accurate, positioned at the same coordinates.

A final clear future work would be to take into consideration more phases of the ADNI or even other dataset altogether, which would increase the cases of CN and AD for the model to train on.

9 Conclusions

Using data from the ADNI Initiative from its first phase, coronal slices from 3D T1-weighted MRI scans go through a skull-stripping and normalization step, then fed to modified ResNet architectures, which also considers additional details from each subject such as sex and age.

Experiments were made on two different architectures, with two different input modalities: ResNet18 and ResNet101, which were given as inputs a singular coronale slice at the center of the hippocampus, and eight slices around the same area. While being heavier computationally, the latter input modality of eight coronal slices proved to be more performant, especially for the 18 layer counterpart of the residual network.

The data is reduced from 1512 scans, after cleaning, to 487 useable, and in necessity of a class balancing method before training, for which up-sampling was opted.

Within 150 epochs of training, the model reaches its peak performance, which was evaluated using accuracy, sensitivity and specificity as metrics, having achieved 93.2%, 95.8% and 93.5% respectively.

10 Acknowledgements

This work is the result of my own activity, and I confirm I have neither given, nor received unauthorized assistance for this work.

I declare that I did not use generative AI or automated tools in the creation of content or drafting of this document.

Data collection and sharing for the Alzheimer's Disease Neuroimaging Initiative (ADNI) is funded by the National Institute on Aging (National Institutes of Health Grant U19 AG024904). The grantee organization is the Northern California Institute for Research and Education. In the past, ADNI has also received funding from the National Institute of Biomedical Imaging and Bioengineering, the Canadian Institutes of Health Research, and private sector contributions through the Foundation for the National Institutes of Health (FNIH) including generous contributions from the following: AbbVie, Alzheimer's Association; Alzheimer's Drug Discovery Foundation; Araclon Biotech; BioClinica, Inc.; Biogen; Bristol-Myers Squibb Company; CereSpir, Inc.; Cogstate; Eisai Inc.; Elan Pharmaceuticals, Inc.; Eli Lilly and Company; EuroImmun; F. Hoffmann-La Roche Ltd and its affiliated company Genentech, Inc.; Fujirebio; GE Healthcare; IXICO Ltd.; Janssen Alzheimer Immunotherapy Research & Development, LLC.; Johnson & Johnson Pharmaceutical Research &Development LLC.; Lumosity; Lundbeck; Merck & Co., Inc.; Meso Scale Diagnostics, LLC.; NeuroRx Research; Neurotrack Technologies; Novartis Pharmaceuticals Corporation; Pfizer Inc.; Piramal Imaging; Servier; Takeda Pharmaceutical Company; and Transition Therapeutics.

References

[1] Mohammed G. Alsubaie, Suhuai Luo, and Kamran Shaukat. Alzheimer's disease detection using deep learning on neuroimaging: A systematic review. *Machine Learning and Knowledge Extraction*, 6(1):464–505, 2024.

[2] Alzheimer's Association. Stages of alzheimer's, 2024. URL https://www.alz.org/alzheimers-dementia/stages.

[3] Alzheimer's Association. Medical tests for diagnosing alzheimer's, 2024. URL https://www.alz.org/alzheimers-dementia/diagnosis/medical_tests.

[4] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 2017.

[5] Christopher Butler. Transient epileptic amnesia. *Practical Neurology*, 6:368–371, 12 2006. doi: 10.1136/jnnp.2006.107227.

[6] Minmin Chen. Minimalrnn: Toward more interpretable and trainable recurrent neural networks. *ArXiv*, abs/1711.06788, 2017.

[7] Vassilis Cutsuridis and Motoharu Yoshida. Editorial: Memory processes in medial temporal lobe: Experimental, theoretical and computational approaches. *Frontiers in Systems Neuroscience*, 11, 2017.

[8] DataCamp. Recurrent neural network tutorial, 2022. URL https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network.

[9] Robin de Flores, Sandhitsu R. Das, Long Xie, Laura E. M. Wisse, Xueying Lyu, Preya Shah, Paul A. Yushkevich, and David A. Wolk. Medial temporal lobe networks in alzheimer's disease: Structural and molecular vulnerabilities. *Journal of Neuroscience*, 42(10):2131–2141, 2022.

[10] Amir Ebrahimi and Suhuai. Convolutional neural networks for Alzheimer's disease detection on MRI images. *Journal of Medical Imaging*, 8(2):024503, 2021.

[11] Harvard. Free surfer, 2023. URL https://surfer.nmr.mgh.harvard.edu/fswiki/FreeSurferWiki.

[12] Medical School at Harvard. Alzheimer's in the family, 2019. URL https://www.health.harvard.edu/mind-and-mood/alzheimers-in-the-family.

[13] Keith A. Johnson, Nick C Fox, R. Sperling, and W. Klunk. Brain imaging in alzheimer disease. *Cold Spring Harbor perspectives in medicine*, 2 4:a006213, 2012.

[14] Jong Bin Bae, Subin Lee, and Wonmo Jung. Identification of alzheimer's disease using a convolutional neural network model based on T1-weighted magnetic resonance imaging. *Scientific Reports*, 10:22252, 2020.

[15] Samir KC and Wolfgang Lutz. The human core of the shared socioeconomic pathways: Population scenarios by age, sex and level of education for all countries to 2100. *Global Environmental Change*, 42:181–192, 2017.

[16] Hongming Li, Mohamad Habes, David A Wolk, and Yong Fan. A deep learning model for early prediction of alzheimer's disease dementia based on hippocampal magnetic resonance imaging data. *Alzheimer's & dementia : the journal of the Alzheimer's Association*, 15(8):1059–1070, 2019.

[17] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[18] MHA. Neurons: How the brain communicates, 2024. URL https://mhanational.org/neurons-how-brain-communicates.

[19] Minh Nguyen, Tong He, Lijun An, Daniel C. Alexander, Jiashi Feng, and B.T. Thomas Yeo. Predicting alzheimer's disease progression using deep recurrent neural networks. *NeuroImage*, 222:117203, 2020.

[20] National Institue of Aging NIH. Alzheimer's disease genetics fact sheet, 2019. URL https://www.nia.nih.gov/health/genetics-and-family-history/alzheimers-disease-genetics-fact-sheet.

[21] National Institue of Aging NIH. What happens to the brain in alzheimer's disease?, 2024. URL https://www.nia.nih.gov/health/alzheimers-causes-and-risk-factors/what-happens-brain-alzheimers-disease.

[22] Wieke Oostveen and Elizabeth Lange. Imaging techniques in alzheimer's disease: A review of applications in early diagnosis and longitudinal monitoring. *International Journal of Molecular Sciences*, 22:2110, 02 2021. doi: 10.3390/ijms22042110.

[23] R. C. Petersen, P. S. Aisen, and L. A Beckett. Alzheimer's disease neuroimaging initiative (adni): clinical characterization. *Neurology*, 74(3):201–9, 2010.

[24] ScienceDirect. Convolutional layer, 2022. URL https://www.sciencedirect.com/topics/engineering/convolutional-layer.

[25] Athira Sivadas and K. Broadie. How does my brain communicate with my body? *Frontiers for Young Minds*, 8, 2020.