# Using ML to Detect and Classify Alzheimer's Data

Maya Gonzalez
mgonzalez3@oxy.edu
Occidental College

## Abstract

Machine Learning has been utilized to help solve problems faster and ideally more efficiently than humans. In one type of Machine Learning solution, Convolutional Neural Networks (CNNs) have demonstrated success in image processing and image classification due to the intrinsic properties of images. One such problem space CNNs are being utilized for is Alzheimer's Disease classification from brain scans. Although research has shown success in this task, more work needs to be done to assess if and how Machine Learning can be utilized within a real clinical setting.

## 1 Problem Context

Alzheimer's disease dramatically affects the individual themselves along with their family. There is currently no cure, but measures can be taken that can slow the disease's progression. Often times, the main indication is the recognition of memory loss and differences in environment interactions. It is also possible to analyze various brain scans and clinical data to determine the extent of a person's condition. The current process of diagnosis requires expensive scans, a specialized medical expert to analyze the images, and overall takes time.

Machine learning (ML) has been utilized in the medical field for various applications due to its ability to reach relatively accurate conclusions in complex problem spaces. One such application is in classifying and predicting Alzheimer's disease (AD) from brain scans and other clinical data. This has shown promising results in the detection of brain changes that are the result of dementia and the classification of Alzheimer's disease and other brain conditions.

Among other factors, loss of brain mass is one indicator of Alzheimer's Disease (AD), yet it is unclear exactly at what stage loss of mass takes place. This question is out of the scope for this project, but may be critical in understanding if detecting these anatomical changes can prevent early onset Alzheimer's or simply be used to diagnose. On a similar note, there are other factors that may contribute to AD that cannot be captured within an image.

Regardless, brain scans have been a popular modality to study and train a ML model on due to the physical indicators of cognitive impairment. Magnetic Resonance Imaging (MRI) is widely used for general medical imaging and is suitable for brain scans due to it's noninvasive nature. MRI scans use radio waves and magnetic fields to produce a detailed image of the brain structure and the brain regions. The clarity and detail of MRI scans make them an advantageous tool in early detection and diagnosis of brain related conditions, specifically Alzheimer's. MRI scans are able to show brain atrophy as well as structural abnormalities[2].

The Medial temporal lobe (MTL) consists of multiple structures that are crucial for memory function such as the formation of semantic and episodic declarative memory. Damage to these structures results in impaired learning and hinders recall of memories formed before the onset of damage.[1] Studies have identified decreased MTL activation in patients with mild-to-moderate dementia when attempting to learn new information in functional MRI scans. [3] MRI scans can show subtle anatomical changes that indicate brain atrophy from early stages of Alzheimer's disease. It is also important to note that there are various other conditions that may have similar or overlapping symptoms. Additional methods must be used to confirm AD or rule out other causes of dementia.

There are multiple modalities of MRI images, such as functional and structural MRI scans. Functional MRI scans measures changes in blood flow of active regions within the brain, whereas structural MRI scans show a static view of the brain's anatomy [8]. Currently structural MRI scans are the standard and recommended modality for use in clinical settings. [2]

## 2 Technical Background

In what is known as End-to-End Learning, Deep Learning (DL) models are able to use raw data as input, which enables features to automatically be learned. It also circumvents the need for specialized doctors to label datasets.

ML encompasses various types of learning such as neural networks. Neural networks are inspired by the architecture of the human brain and how the brain processes information. One such DL model is a type of artificial neural network: Convolutional Neural Networks (CNNs). Al-

though they were designed for general image usage, CNNs have gained popularity in the medical imaging analysis research community. CNNs have shown state-of-the-art performance in image classification problems. On such image classification task that has been widely explored is Alzheimer's Disease classification [9]. CNNs can classify brain scans as having Alzheimer's disease or not. The conception of CNNs was inspired by the visual cortex and how the brain processes visual information.

Compared to other neural network architectures, CNNs have demonstrated success in image processing primarily due to the use of convolutions and translational invariance, which refers to the idea of recognizing objects regardless of relative positions. This is possible with a CNN due to the nature of assessing sections of pixels together by applying a convolution on each section of pixels. CNNs take advantage of the intrinsic properties of images. It is based on the idea that adjacent pixels share similar properties and can be used to inform/influence the context of the surrounding pixels.

The goal of a CNN in image processing is to reduce the image dimension in such a way that avoids losing important information but reduces the pixel amount to reduce computational power. Relative to other ML models, CNNs have the ability to reach higher levels of abstraction and complexity, which enables them to detect subtle, scattered, and complex patterns in the data. CNNs have seen better results on image classification tasks compared to traditional ML methods.

CNNs are composed of various types of sequential layers. The three types of layers that constitute a CNN are the convolutional layer, pooling layer, and the fully-connected layer. The layers are 'stacked' so that the output of one layer is passed onto the subsequent layer. The process continues until the final layer is reached. The architecture of a CNN refers to how the types of layers, the ordering, and how many layers compose the CNN.

The convolutional layer plays a major role in CNNs. A convolution is a mathematical operation that essentially applies a filter to an image and performs a calculation. Convolutional layers are what enables feature extraction. Features may include, but are not limited to, edges, vertical lines, or more complex features such as a dog's eye. Feature extraction creates a set of the most important features and discards the original features from the raw data. This reduces the amount of data that isn't necessary for the model to learn. The goal of this step is to extract high-level features such as edges from the input image.

The pooling layer preserves the most crucial features. Similarly to a convolutional layer, the pooling layer performs a set of operations on the input data referred to as pooling operations. The step reduces the size of image by reducing the number of parameters and therefore dimensions. It creates smaller, more concise outputs, which improves performance. Ultimately, the CNN has less parameters to learn which also results in faster training. There are three types of pooling operations: min, max, and average pooling. Pooling layers also apply a filter over the image, derives a value used to populate the output matrix. The min pooling operations takes the minimum value within each each filter section and uses that value to set the respective output matrix value. The process repeats until the filter passes over the entire image and the output matrix is filled. The max and average pooling operations take the maximum and average value of each filter section, respectively.

Fully connected layers, or dense layers, are used as some of the final layers of the architecture. A dense layer changes the dimensions from the preceding layer. Each neuron of the dense layer is connected to every neuron of the preceding layer.

Convolutional layers and fully-connected layers have an activation function that can be specified. An activation function assesses a neuron's importance to the model's prediction by determining whether a neuron will be activated or not. There are various types of activation functions that can be used.

Any image can be represented as a grid of pixels. Each pixel can be translated to a value representing the color. CNNs process an image as three channels: red, green, and blue. Gray scale images are single channeled. For gray scale images, 0 represents black while 255 represents white. The variations of colors between black and white are represented as respective numbers between 0 and 255. For color images, the image is represented in a similar manner but this time consists of three values corresponding to the three channels. Each pixel is composed of a red, green, and blue intensity and thus can be represented as an array of three numbers within the range of 0 to 255.

The basic process of CNNs first starts with an input image of size nxn. Data flows from layer to layer in order, so that the output of one layer is fed into the next layer as input. Once the end of the network is reached, the error is calculated by comparing the output to the desired output. This iterative process begins with the input layer in what is referred to as forward propagation. With each output the error is calculated by comparing the output to the desired output. The derivative of the error is then calculated with respect to the given parameter. To adjust the parameter, the calculated derivative is subtracted from the parameter itself. The goal of parameter adjustment is to minimize the error via gradient descent. In order to update the parameters, it is necessary to compute the derivative with respect to the cost.

The convolutional step is a specialized process used within the convolutional layer. A filter, or kernel, of size fxf is utilized and a stride length, s, is determined. Odd numbered filters are the most common out of convention, since they allow us to refer to the output matrices in terms of a

central pixel. The filter begins the convolution step at the top left of the input matrix. The entire convolutional process involves he kernel moving over the image a set number of times. Each time, it takes the dot product of the filter and the portion of the input image that the filter is hovering over and calculates the sum. This sum becomes a value in the respective cell of the output matrix, or the feature map. Assuming a stride length of 1, the size of the output matrix is (n-f +1) since there are only a certain number of possible positions for the filter to traverse the input image. If a stride length is chosen that is greater than 1 the output size will be (n+2p-s)/s +1. To determine the next value in the output matrix, the filter moves to the right, determined by the stride length, and performs the same convolution over a new portion of the input image until it traverses the entire image width. Then, it will jump to the opposite side of the next row and repeat this process until it has traversed the entire image. Assuming only one filter was used, the results from each matrix multiplication are summed up with the bias, a non-linearity is applied, and the result is a one-depth channel Convoluted Feature Output. The goal of this step is to extract high-level features such as edges from the input image. The dimensionality of the each layer is determined by the pixel size and channel depth. For example, if you are using a RGB input image of 1000x1000, there will be 3 million input features. Instead of hand picking weights for the filter, it is possible to use each weight as a parameter and have the computer 'learn' the best weights through backpropagation. It has been found that this method can have better feature detection than researcher's chosen weights.

CNNs typically have more than one layer and must have a way to take the activations of one layer and map those to the activations of the next layer. The way this is done in CNNs is through forward propagation and back propagation which uses a convolution operation. The forward pass involves derivation of the gradients moving from left to right. At the end of the circuit, the loss is computed by using the loss function. Relating back to the filter, the convolution between the filter and the input derives the output. Backpropagation is used to derive the error between the expected output and the network's output. It measures the influence of every intermediary value in the graph on the final loss function. After the forward path, the backward path is derived by calculating the gradient of all the intermaries in the circuit from right to left until we are left with the gradients of the input. Since this is a recursive function, in the base case we consider the identity function, the gradient of f with respect to f, which has a gradient of 1. To compute the gradient of the input layer, the chain rule must be applied. Each intermediate value is positively influencing the loss if it is a positive gradient, whereas it is negatively influencing the loss if it is a negative gradient. The general backpropagation procedure calculates the error gap, changes the weights, and stops once the gap is no longer updated or when the differential value becomes 0. The least squares method can be used to calculate the error value. As an iterative process, the gradient descent function is used to calculate the weights of each layer. Training under gradient descent forces the network to learn to extract the most important features or features that will minimize the loss.

Another hyperparameter that can be specified is image padding. Image padding refers to how many empty pixels surround the image, if at all. There are a few issues of using the input exactly as it is without any surrounding empty pixels, the first being that every time a convolution is performed the output gets smaller, which means that in a network with many layers, the image will be very small. The next issue is that corner pixels from the input image are utilized less in the convolutions than middle pixels, which is a problem since the edges of the input image may contain important information. To fix the problems of shrinking output and the disregard for edge information, one method is to 'pad' the image before any convolutions are calculated. This adds additional pixels around the image border and therefore makes the image dimensions larger. The padding value is chosen based on the goal, the two most common choices are valid and same convolutions. Valid colvolutions mean that no padding is to be added. Same convolutions means that the output size is the same as the input size. The padding value varies depending on filter size and can be calculated by (f-1)/2.

Batch size refers to the number of images the model trains on at a time. Thus, the model will make a set number of iterations for each epoch based on the batch size. The number of iterations for each epoch can be calculated by dividing the training or validation dataset size by the batch size. The remainder of images will be added to their own batch and the number of iterations increases by one. In this scenario, the last batch size will not equal the set batch size. Literature suggests that a smaller batch size leads to improved accuracy. However, this suggestion is debated as researchers continue to investigate the optimal batch size. There is no ideal batch size for all datasets. [4] suggests trying out smaller batch sizes to determine the best batch size for your dataset. Batch sizes are commonly set to powers of 2 to best utilize the processing power of the GPU.

An epoch refers to when the entire dataset is passed to the network. One epoch accounts for both the forward and backward pass. Gradient descent is an iterative process that updates weights at each step to converge faster. Since the backpropagation step utilizes gradient descent, we want to increase the number of epochs to converge faster.

## 3 Prior Work

There are various methods to extract valuable information from MRI images. The main methods include voxel-based features, region-of-interest based features, and whole-image-based features. The most popular methods utilized in this research area include Singular Value Decomposition (SVD), Principal Component Analysis (PCA), and Convolution Neural Networks (CNN). Regardless of the chosen model, researchers experiment with data augmentation, and other methods to reduce overfitting. Data augmentation involves generating reflections and transformations of images. The network then makes a prediction on all these images. Another form of data augmentation is to alter the RGB intensities.

Although current research has shown promising results and some exhibit near perfect accuracy in ML models, ML is not widely used in actual clinical practice. This is partly due to the fact that most models are trained on one type of data, such as images, whereas there are additional factors that need to be considered in a real patient's case. Most existing regression methods only utilize a single modality of data without taking into account related information derived from different modalities. Specifically, most studies solely focus on disease classification or clinical score regression. There have been some researchers creating models that account for multiple modalities of data and have found interrelationships between the different types of data. These researchers have described the limitations in work that only considers low-level features such as gray matter tissue volumes from MRI data. There have been efforts to address both these tasks in a unified methodology. Such studies have found the features from both tasks are highly related as described in Zhang et. al [11], Liu et al. [6], and Suk et al.[7]. The multi-modality methods have generally found improved classification performance of AD.

Krizhevsky et al.[5] created a deep CNN, AlexNet, that has outperformed other top models, achieved record error rates, and won the ImageNet classification challenge in 2012. The architecture has 60 million parameters, 650,000 neurons, five convolutional layers, and three fully-connected layers. The high performance is partially credited to the depth of the model. They found that removing any layers resulted in degraded performance of the model. The researchers identified the most important feature of AlexNet as the implementation of ReLU nonlinearity [5] . The standard way to model the output as a function of its input is through saturating linearity. Since gradient descent is used to train the CNN, the updates to the weights at each iteration are dependent on the gradient. The tanh(x) function plateaus anywhere beyond ±1, which means that the gradient of this function is basically 0 anywhere beyond this middle range. The weights will stop changing once a neuron's output reaches this range since the gradient will be in close proximity to 0. The other option is non-saturating linearity which has the function of f(x) = max(0,x). This function only plateaus for negative values. Non-saturating linearity is computationally much faster than that of saturating linearity. Neurons with this nonlinearity are referred to as Rectified Linear Units (ReLU). Overall, networks with ReLUs train several times faster than those with tanh units. It wouldn't be possible to work with such a large network if a ReLU was not implemented. In AlexNet, ReLU was applied to the output of every convolutional and fully-connected layer. It is important to note that saturating nonlinearities may work well for other smaller models trained on smaller datasets. Their results broke previous state-of-the-art records in competitions, as they achieved top-1 and top-5 error rates of 37.5% and 17.0%, respectively.

Some researchers attempt to improve accuracy metrics by implementing well-researched methods such as data augmentation and multi-modality training. Yoon et al. [10] achieved 98% overall accuracy by using AlexNet and Mini-batch Stochastic Gradient Descent. The researchers also explored data augmentation through rotations, which resulted in further improved accuracy. They concluded that their method was a good model for classifying NC, MCI, and AD in 18F-FBB amyloid PET brain images. Liu et al. [6] developed a convolutional neural network, DM2L, that incorporates both feature extraction and classification model training. Their joint classification and regression learning framework has an advantage as they incorporated patient's demographic information (age, gender, and education) into the learning process. The DM2L method performs better when compared to similar joint classification approaches. The researchers credit the superior performance to the inclusion of subject's demographic information as well as the simultaneous learning of MRI features along with the classifier and regressor.

## 4 Methods

The Alzheimer's Disease Neuroimaging Initiative (ADNI) offers access to multiple datasets including MRI scans. The number of patients represented in each data set varies, but it is generally less than 1,000 total patients. The limited amount of data available to train on poses performance issues for the classification of AD.

I used a dataset derived from ADNI that includes 5119 preprocessed structural MRI scans. Images are divided into four subfolders that represent the label. The labels include non-demented, very mild demented, mild demented, and moderate demented. The dataset contained 3200 images belonging to the non demented class, 2240 very mild images, 896 mild images, and 64 moderate images.

I used this entire pre-processed dataset and split it into

training and evaluation datasets. This is a common practice to test whether the model is overfitting to the training data. I used a 80/20 training to evaluation split, respectively. This is a common split in ML literature.

To build the model, I focused on designing the architecture to improve the accuracy metrics. I utilized Tensorflow, which is a Machine Learning platform. I used Tensorflow's Keras library to build the CNN. The code was written in Python. For most of my testing, I utilized a GPU to increase processing time. The built-in features of Keras enabled me to build the CNN efficiently while avoiding building the network from scratch. The use of an existing library allowed me to focus on research areas of interest such as architecture design.

For the architecture, my initial idea was to look at existing successful architectures and understand how their architecture decisions led to success. My idea was to replicate and tweak trends in layer order and hyperparameter selection that were seen in existing models. Although these architectures were able to achieve great results on their respective datasets, that success did not necessarily translate when training on a dataset of MRI images. For example, I tried training the AlexNet architecture on my dataset. Since this architecture achieved state of the art results classifying the ImageNet dataset, I thought it would perform well when tasked with classifying MRI images. Contrary to my hypothesis, AlexNet performed far worse than other simplified architectures that I was already experimenting with. This may be due to the fact that AlexNet could classify images that contained real life objects belonging to 1000 classes. AlexNet contained 1,281,167 training images and 50,000 validation images. In the MRI classification task, there are only four classes and the images look relatively similar since they are all brains. For example, two brains with differing gray matter volumes look more alike than when comparing a dog and a baseball bat. AlexNet may be more suitable for classification tasks where the images differ drastically.

Through more researching and my own experience, it became apparent that the model's performance is highly dependent on the dataset trained on. Thus, it may have been helpful in the initial stages to take inspiration from other successful architectures, but adjustments needed to be made based on my model's performance.

Next, I began by tweaking both the architecture and hyperparameters simultaneously. A better method that I adopted later on was to decide on an architecture and tweak hyperparameters after the architecture was solidified. Even still, I continued to modify the architecture by adding and deleting layers.

I experimented with the batch size in the initial stages and ultimately chose a batch size of 256. I kept this as a constant throughout the training to reduce the variables while I experimented with other architecture changes. I also experimented with filter size and stride. Based on my literature review, I experimented with increasing the filter size with each subsequent layer. I found this method to increase the accuracy. I also found that a constant stride length of 3 was optimal. I also implemented ReLU activation for the convolutional layers and the first fully-connected layer. Although I set training to 100 epochs, the training usually terminated around the 40th epoch.

Not only was I mindful of accuracy metrics, but I also wanted to reduce computation time and power. To avoid unnecessary training, I implemented callbacks to stop training when a certain condition was met. One such callback that was often utilized during training was an early stopping callback wihtin Keras that would end training when a set metric was no longer improving. The decision of these numbers was inspired by ML discussion boards as well as my own experimentation. I would tweak these parameters based on the model's performance. I ultimately decided on measuring validation loss. If the validation loss did not improve by 4 epochs, the training would end. Thus, the number of epochs the model would train on could fluctuate. The use of early stopping callbacks may prevent the model from overfitting. Overfitting refers to the trend when the model performs too closely to the training data and fails to generalize predictions accurately to the validation data.

I did controlled testing to determine the best set of layers and hyperparameters. One limitation of this method was that there may have been one parameter that resulted in poor performance independently. However, if this parameter was combined with additional parameters, then the result may improve. In my method, I could have missed out on entire areas of testing and exploration if I prematurely ended testing based on poor initial performance.

The architecture that I achieved the best results composed of three convolutional layers all followed by a max pooling layer. A flatten layer followed by two dense layers completed the model's architecture. For the convolutional layers, I utilized same padding to avoid reducing the image's dimensions. I also used a ReLU activation function for each convolutional layer. It was standard in other architectures to utilize the same activation function throughout the model. The motivation behind using the ReLU activation function was to make computation more efficient. T

## 5 Evaluation

The metrics I focused on were training loss, training accuracy, validation loss, and validation accuracy. In literature reviews, I did come across various other metrics used to assess the model's performance. I ultimately decided not to include these metrics, as it was enough for me to focus on accuracy and loss. It would be an interesting area to explore

in the future.

## 6 Results and Discussion

The architecture I implemented that had the best success achieved a 98.91% validation accuracy and a validation loss of .038. I was cognizant of overfitting, and the results indicate that the model was not overfitting the training data. One metric that indicated this was that the validation loss was higher than the training loss, which suggests that the model is performing better on the training data, as expected.

It is important to compare these results to other model's training results on the same dataset. Although the accuracy is high and the loss is minimized, a better picture is provided when comparing to others successes. By analyzing some of the posted solutions to this dataset from the Kaggle community, I was able to compare my results with results from published architectures. Some published architectures had achieved a validation accuracy as high as 99.84%, which was higher than mine. This comparison that was possible through the Kaggle community would have been more difficult if I utilized a raw dataset from ADNI. From my literature review, it seemed that ML researchers in this problem space did not always publish their architecture or code.

There is still a lot more to learn and explore when designing CNN architectures. I found myself falling into so called 'rabbit holes' when researching an aspect of CNNs. Once I thought I had a grasp on a concept, there was always something more to learn and delve into. For me, this work emphasized how complex ML problems are and how there is still so much to explore (in these problem spaces). I may have only been able to make implementations that scratched the surface of the designing CNNs, but I gained a broad understanding of CNNs and also went in-depth in certain research areas. was still able. I accomplished my initial goals of implementing my own architecture, achieving an accuracy above 95%, and understanding the role of layers and hyperparameter selection.

As far as next steps, I would like to utilize multiple modalities of data to create a more comprehensive model. Moreover, it would be worthwhile to explore disease progression of individual patients rather than analyzing a single image per patient. Some ADNI datasets take scans of patients at . These datasets can inform researchers about how the disease progresses. I would be curious to learn if there are any indicators? present in the early stages that indicate if a patient will or will not get moderate AD. This information could be used differently in a clinical setting. It would also have been interesting to try out processing the images myself instead of using a preprocessed dataset from Kaggle.

## 7 Ethical Considerations

There are various ethical concerns to consider when incorporating a Machine Learning (ML) model into the decision making process of a real clinical scenario. Using ML to diagnose Alzheimer's disease (AD) from image data is one problem space that has the potential to prevent unnecessary invasive procedures or to diagnose a patient before symptoms appear in hopes to slow down the progression. The concerns that are arguably most important are the ones concerning the safety of the patient. There are many factors and decisions being made in the overall process even before a patient is introduced into the equation. The general process includes data collection, research and model building, and then implementation into a clinical setting. It is necessary to rule out all possibilities of introducing a bias into the model before any ML model can be used in the decision making process of a real case. If not, these biases will be encoded into the final decision and can have life threatening consequences.

To start off at the beginning of the process, the data sets available are too small for training purposes. Current research mainly trains on one of several popular data sets, which contain only several hundreds of participants. It is also important to consider the specific data each model is trained on, as the model may become highly dependent on the data trained on. The model may perform well on a certain population group if that was the only data provided for training. It is acknowledged that most medical datasets are trained on a subset or specific group of the general population. The lack of racial and ethnic diversity in the participants that make up a dataset is a problem that extends into the space of brain scan databases. For example, the ADNI dataset is one of the largest AD datasets available and widely used in research. However, the participant data is biased, as demonstrated by the participant makeup. According to the reported ADNI participant demographics, of the 822 total subjects sampled from, 764 identify as White. Only 39 of the total participants identified as black or African American and 14 identified as Asian. Of the 822 participants, 21 identified as Hispanic or Latino. Males accounted for 478 of the participants while females made up 344 of the total participants. The lack of participant sampling which accurately reflects the general population reduces the ability to generalize the models trained on these data sets.

Regardless of the model's performance, there is a broader question of how the model would be implemented in a medical setting. More specifically, at what stage would the model's decision be incorporated? How much influence should the decision of a model have over that of the medical professionals? Most medical professionals and doctors involved in the decision making process may not have ex-

perience or familiarity with ML. They may see this tool as a black box, in that they have no idea how the decision is being reached and therefore may not trust the ML model. This may lead medical practitioners to adopt a biased view of the model's predictions.

Once deployed into a clinical setting, another issue emerges of who has access to this technology. MRI scans are an expensive procedure in the US healthcare system, so only individuals with the available money would be able to take advantage of this resource. This lack of accessibility further deepens the barrier of access to medical innovations. If a tool is only able to help a subset of the general population who have the available resources, then a question arises of how useful this will actually be.

Due to the outlined issues, it is not yet ethically feasible to use a ML model to assist in the decision making process of Alzheimer's disease. It is not worth risking a patient's life to make wishful strides in Machine Learning.

## References


[1] Peter J Bayley, Ramona O Hopkins, and Larry R Squire. "The fate of old memories after medial temporal lobe damage". In: *Journal of Neuroscience* 26.51 (2006), pp. 13311–13317.

[2] Avinash Chandra, George Dervenoulas, and Marios Politis. "Magnetic resonance imaging in Alzheimer's disease and mild cognitive impairment". In: *Journal of neurology* 266.6 (2019), pp. 1293–1302.

[3] Bradford C Dickerson et al. "Medial temporal lobe function and structure in mild cognitive impairment". In: *Annals of neurology* 56.1 (2004), pp. 27–35.

[4] Ibrahem Kandel and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset". In: *ICT express* 6.4 (2020), pp. 312–315.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[6] Mingxia Liu et al. "Joint classification and regression via deep multi-task multi-channel learning for Alzheimer's disease diagnosis". In: *IEEE Transactions on Biomedical Engineering* 66.5 (2018), pp. 1195–1206.

[7] Heung-Il Suk and Dinggang Shen. "Deep learning-based feature representation for AD/MCI classification". In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2013, pp. 583–590.

[8] M Symms et al. "A review of structural magnetic resonance neuroimaging". In: *Journal of Neurology, Neurosurgery & Psychiatry* 75.9 (2004), pp. 1235–1244.

[9] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9.4 (2018), pp. 611–629.

[10] Hyun Jin Yoon et al. *Classification of AD/MCI/NC from Amyloid PET Images using Deep Learning CNN Algorithm*. 2018.

[11] Yu-Dong Zhang, Shuihua Wang, and Zhengchao Dong. "Classification of Alzheimer disease based on structural magnetic resonance imaging by kernel support vector machine decision tree". In: *Progress In Electromagnetics Research* 144 (2014), pp. 171–184.