Convolutional Neural Networks for prediction for individuals' age and cognitive ability using MRI

One Semester Individual Report
Final report

Student: Craig Winfield
Supervisor: Dr Matthias Treder

School of Computer Science & Informatics
Cardiff University
May 2019

Abstract

The average age of the population is steadily increasing; therefore, we need to look for earlier signs of deterioration in cognitive ability. The earlier that abnormalities are spotted, the sooner and better they can be treated, and MRIs can spot early signs of disease before any symptoms become visible. An MRI is a 3-dimensional medical imaging technique used in radiology to form pictures of the anatomy and the physiological processes of the body in both health and disease. MRI scanners use strong magnetic fields, magnetic field gradients, and radio waves to generate images of the organs in the body. Signs of physical changes include gray-matter loss, the brain shrinking, and the decline of white-matter.

Inspired by 2-dimensional and 3-dimensional convolutions, the aim of this project is to use machine learning on the Cam-CAN ageing neuroimaging dataset. Due to the non-linearity nature of ageing, it is hard to comprehensively model using standard linear statistical techniques such as GLM (Generalized Linear Models). The goal will be to predict the age of participants and physiological variables (cognitive ability) from MRIs. Augmentation of linear models by e.g. quadratic terms only partially address this problem since it fails to address other relationships. As an alternative, deep neural networks are known to learn arbitrary non-linear relationships. In recent years, Convolutional Neural Networks (CNNs) have been very successful at extracting features from raw images thereby circumventing the necessity for prior feature selection / dimensionality reduction through methods such as ICA (Independent Component Analysis)

Results are encouraging, with roughly 40% accuracy from a 7-class Classification model and achieving Mean Square Error and Mean Absolute Error of around 0.7, achieving better than the results of more basic models like Support Vector Machines. From these results, we believe Convolutional Neural Networks are a justified choice when mapping the relationships between MRIs and non-linear variables. However, we believe better results can be achieved if a larger dataset presents itself, and if greater computational power is available to handle the demanding nature of certain Convolutional Neural Network models. This is because hardware constraints and a limited dataset imposes restrictions on the type of Network architecture we can develop, such as the depth of the model.

Acknowledgments

I would like to thank my supervisor, Dr Matthias Treder, whose guidance and knowledge kept me in the right direction and made this an enriching and exciting project.

I would also like to thank my co-supervisor Dr. Kamen Tsvetanov (University of Cambridge).

Finally, I would like to thank Dr. Darren Price from the Cam-CAN project for help with data access and the whole Cam-CAN team

Table of Contents

Convolutional Neural Networks for prediction for individuals' age and cognitive ability using MRI 1
Abstract ... 2
Acknowledgments ... 3
Introduction ... 9
Background ... 12
- **Cam-CAN dataset** ... 12
- **MRI (Magnetic Resonance Imaging)** ... 12
- **Deep Learning in MRIs** ... 12
- **Machine Learning** ... 12
  - **Supervised Learning** ... 13
- **Fitting the model to the data** ... 13
  - **Underfitting** ... 14
  - **Overfitting** ... 14
- **Artificial Neural Network** ... 14
- **Basic Neural Network structure** ... 15
- **Training a neural network** ... 19
- **Optimal Performance** ... 19
  - **Calculating Loss** ... 19
  - **Loss Functions** ... 19
  - **Optimisation Function** ... 22
  - **Dropout** ... 24
- **Convolutional Neural Networks** ... **25**
  - **Building Blocks of a Convolutional Neural Network** ... 25
- **Batch Normalisation** ... 28
- **One-hot Encoding** ... 28
- **Performance Metrics** ... 29
  - **Accuracy** ... 29
  - **Confusion Matrix** ... 29
- **Convolutional Neural Networks vs Basic models** ... 29
  - **Naïve Bayes** ... 30
  - **Decision Trees** ... 30
  - **Support Vector Machine** ... 31
- **Software and Libraries used** ... 31

Python 31
Sci-kit Learn, NumPy, Pandas 31
NiBabel 31
Keras 32
Google Colab 32
Supercomputing Wales (SCW) 32
Tensorflow. 32
Approach 33
Business model approach & user interface 33
Data flows of the system 33
Float32 from Float64 39
2-dimensional vs 3-dimensional models 39
Version-Specific Approach 39
1-5 Hidden Layers 40
Final Version of Models 40
Implementation 42
General Implementation 42
Dataset Pre-processing 42
Cropping the MRIs 44
Training/Test Split 45
One-hot Vector Encoding 46
Convert 3-dimensional MRIs to 2-dimensional slices 46
Convolutional Neural Network 47
Section 1: The Network Model 48
Section 2: The Network Training, Evaluation, and Prediction 50
Section 3: The Network Main Function 51
Results and Evaluation 53
Evaluation Metrics and Methods used 53
Early Development Results 55
Effect of number of Layers 55
Effects of Dropout 55
Comparing Convolutional Neural Networks to Basic Models 57
Key Results 57
Note on Classification Results 58
2D Classification High-Resolution 58
2D Classification Low-resolution 59

3D Classification High-resolution 61
3D Classification Low-resolution 62
2D Regression High-resolution 63
2D Regression Low-resolution 64
3D Regression High-resolution 66
3D Regression Low-resolution 66
Conclusion of Results 68
Future Work 69
Architecture Optimisation 69
Better Classification Training/Test Split 69
Capsule Networks 69
Different Architecture Model 70
Mapping the Relationship between MRIs and other variables 71
Conclusion 72
Reflection 74
References 76

Table of Figures

Figure 1: Model Overfitting/Good fit/Underfitting (Bhande, 2018) 14
Figure 2 (Left) Feed-back and (Right) Feed-forward Networks [7] 15
Figure 3 Standard Artificial Neural Network [9] 16
Figure 4 Activation function plots, equation, and data range [12] 17
Figure 5 Gradient Descent Optimisation [15] 23
Figure 6 Effects of Learning Rate for Gradient Descent [15] 23
Figure 7 Application of Dropout [18] 25
Figure 8 Convolution Operation [20] 26
Figure 9 Max-Pooling of 4x4 Matrix [21] 27
Figure 10 Effects of Max-Pooling [22] 27
Figure 11 Confusion Matrix illustration [23] 29
Figure 12 Example Decision Tree [25] 31
Figure 13 Data Flow of the System 33
Figure 14 Histogram of the Age Distributions 35
Figure 15 3-dimensional to 2-dimensional 36
Figure 16 Possible Distribution of Classes 36
Figure 17 Convolutional Neural Network: Feature Extraction 38
Figure 18 Convolutional Neural Network: Classification and Regression 38
Figure 19 Dataset Pre-processing: Extract Cam-CAN MRIs 43
Figure 20 Data Pre-processing: Extract the labels 44
Figure 21 Code to crop the MRIs, ... 45
Figure 22 Cropped MRI 45
Figure 23 Training and Test split 45
Figure 24 One-hot Vector Encoding of Ages array 46
Figure 25 3-dimensional to 2-dimensional slices for low-resolution MRIs 46
Figure 26 2-dimensional MRI slice with constant z-axis value 47
Figure 27 Artificial Neural Network-Model definition 48
Figure 28 Convolutional Neural Network: Sequential model, Input Layer 48
Figure 29 Convolutional Neural Network: Max Pooling 48
Figure 30 Convolutional Neural Network: Dropout and Batch Normalisation 49
Figure 31 Convolutional Neural Network: Remaining Convolution and Max Pooling Layers 49
Figure 32 Convolutional Neural Network: Flattening layer 49
Figure 33 Convolutional Neural Network: Dense Layers 49
Figure 34 Convolutional Neural Network: Training 50
Figure 35 Convolutional Neural Network: Evaluation 50
Figure 36 Convolutional Neural Network: Prediction 50
Figure 37 Convolutional Neural Network: Main function 51
Figure 38 Convolutional Neural Network: Initialising Object 51
Figure 39 Convolutional Neural Network: Data Pre-processing 51
Figure 40 Confusion Matrix example 54
Figure 41 (Left) No Dropout, (Right) Dropout 56
Figure 42 Breakdown of models developed 57
Figure 43 Compiled Results of CNN models 58
Figure 44 Result: Confusion Matrix Classification 2D High-Resolution 58
Figure 45 Result: 2D Classification High-res Loss graph 59

Figure 46 Result: 2D Classification Low-res Confusion Matrix 60
Figure 47 Results: 2D Classification Low-res Accuracy and Loss 60
Figure 48 Results: 3D Classification High-res Confusion Matrix 61
Figure 49 Results: 3D Classification High-res Loss 62
Figure 50 Result: 3D Classification Low-res Confusion Matrix 62
Figure 51 Results: 3D Classification Low-res Loss 63
Figure 52 Result: 2D Regression High-res MSE & MAE 64
Figure 53 Result: 2D Regression High-res Loss 64
Figure 54 Result: 2D Regression Low-res MSE & MAE 65
Figure 55 Result: 2D Regression Low-res Loss 65
Figure 56 Results: 3D Regression High-res MSE and MAE 66
Figure 57 Results: 3D Regression High-res Loss 66
Figure 58 Result: 3D Regression Low-resolution MSE and MAE 67
Figure 59 Result: 3D Regression Low-res Loss 67
Figure 60 Future Training/Test split distribution 69
Figure 61 Inception Model [34] 70
Figure 62 Mini-network replacing 5*5 Convolution [35] 71

Table of Equations

Equation 1 Custom Loss Function 19
Equation 2 Mean Absolute Error 20
Equation 3 Mean Squared Error 20
Equation 4 Categorical Cross-Entropy 22
Equation 5 Naive Bayes Theorem 30

Introduction

In the UK, the life expectancy of an individual is now roughly 80 years[1]. Ageing is both chronological (number of years you have lived for) and biological (the age we appear to have). The chronological age of a person is usually assumed based on their appearance, whereas biologically speaking, ageing is a gradual process; it affects us physically, mentally, socially, and so on. However, the ageing process is different for everyone; the process can be slower or quicker for some. There are younger people who appear older, and vice versa. So, while the appearance of a person can give away their age, a person may not be as young or old as they appear. A procedure used by doctors around the world to examine the biological age of a person is the MRI (Magnetic Resonance Imaging) scan. MRIs produce detailed images of the organs and tissue in the body. For example, an MRI scan will be used to check for anomalies of the brain. These images can spot existing problems in the body as well as early signs of disease before any symptoms become visible. Figure 1 below is an example of a 2-dimensional Brain MRI.

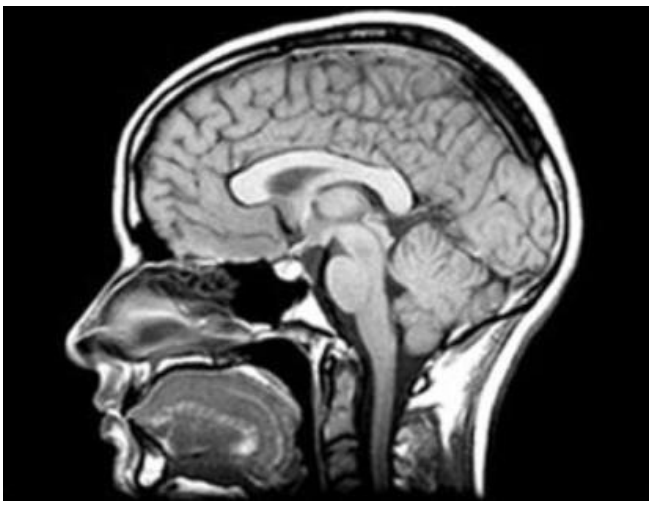


Figure 1 2-dimensional Brain MRI

The aim is to develop 2-dimensional and 3-dimensional Convolutional Neural Network (CNN) models for classification and regression, ones that can predict age and cognitive functionality from a given MRI. Ageing has a non-linear nature, making it hard to model comprehensively using standard linear techniques such as GLM (Generalized Linear Models). Deep neural networks are known to learn arbitrary non-linear relationships. In recent years, CNNs have been successful at extracting features (attributes) from raw images thereby circumventing the necessity for prior feature selection/dimensionality reduction through methods such as ICA (Independent Component Analysis), and have had great success in recent years in image

[1] https://www.ons.gov.uk/peoplepopulationandcommunity/birthsdeathsandmarriages/lifeexpectancies/bulletins/nationallifetablesunitedkingdom/2015to2017

processing competitions like the ImageNet competition [1]. The ImageNet competition is one where teams compete to achieve the highest accuracy in image classification.

The methodology used to tackle this project was Agile [2]. The first step of the approach is to define a model as simple as possible that can carry out regression or classification on the dataset. Next, we develop a solution that meets all minimum requirements for classification and/or regression and then test the model. Finally, we evaluate the performance of the software. After reflecting on the performance of the model, the software development lifecycle starts again, with each new iteration of the cycle building upon the results of the last iteration by working on the most relevant pieces of work. By taking this approach, we avoid carrying out unnecessary work now and focus on the top priority tasks to meet the requirements of the project in this iteration of the lifecycle. Figure 2 is an example of the methodology being adopted for the project.

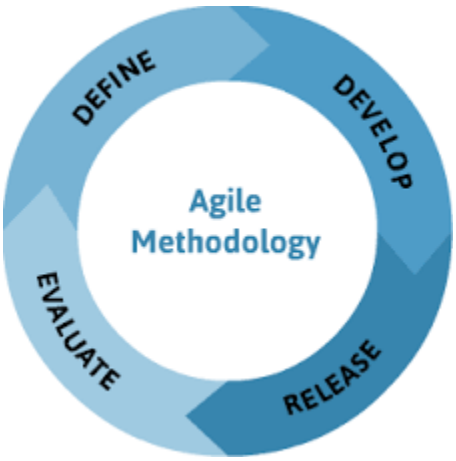


Figure 2 Agile methodology

In this project, the biological age of an MRI will reflect its perceived age and cognitive ability (memory score). The motivation for this project stems from the desire to see whether Convolutional Neural Networks are a suitable choice for modelling non-linear relationships from images compared to other methods available, therefore the scientific community benefits from this work. Looking further afield however, a well-developed solution would help doctors to make more decisive and accurate decisions, and patients could expect a higher standard of treatment as a result. The predictions themselves would reflect the person's health, and therefore, justification for succeeding action is given by the result from the prediction. These predictions would enable a doctor to take necessary measures with a patient more confidently, thus providing greater probability of successful treatment and another step forward in the preservation of cognitive ability into old age.

The network will be trained using the Cam-CAN (Cambridge Centre for Ageing and Neuroscience) dataset which contains 653 MRIs of high-resolution and low-resolution quality. It is assumed that a developed model will perform better on the high-resolution images due to the availability of more data in the high-resolution images. The dataset includes brain structure measures, general health measures, demographics data, and cognitive performance

from 653 individuals aged between 18-87 years old with different lifestyle choices and cognitive performance.

The results achieved have been great. All Convolutional Neural Network models performed better than more basic models (Support Vector Machines, Decision Trees etc.) and show an ability to correctly map MRI to non-linear variables, such as age. These results give justification for the use of CNNs to map non-linear relationships and will be discussed further on in the paper.

Background

Cam-CAN dataset

Cam-CAN is the Cambridge Centre for Ageing and Neuroscience, a large-scale collaborative research project. The Cam-CAN project is using epidemiological, behavioural, and neuroimaging data to understand how individuals can best retain cognitive abilities into old age. The Cam-CAN study is aiming to characterise age-related changes in cognition and brain structure and function, and to discover ways to retain cognitive functionality into old age [3]. We are using the Cam-CAN dataset for our Artificial Neural Network to be trained and predict a mapping between MRIs and values. The dataset contains 653 MRIs, with attributes such as age, memory, smoking, alcohol etc. The lifespan of the test subjects is 18-87 years old. For the purpose of this project, only the two following features are being focused on:

- Age
- Memory.

The Cam-CAN dataset contains 2 distinct sets of MRIs. These MRIs are divided into high-resolution and low-resolution data. The high-resolution images have a dimensionality of 181*217*181 and the low-resolution images have a dimensionality of 61*73*61. At the start of the project we were only given the low-resolution images. In the later stages of the project, we received access to the high-resolution images, but had to access them by way of importing from a shared directory on the Supercomputer Cluster. The MRIs have been processed through a pipeline into separate streams for different avenues of analysis, such as Voxel-based morphometry (VBM) for structural analysis [3]

MRI (Magnetic Resonance Imaging)

MRI uses a strong magnetic field and radio waves to create detailed images of the organs and tissues within the body. Since its invention, doctors and researchers continue to refine MRI techniques to assist in medical procedures and research. A doctor may order a brain MRI scan to look for Blood vessel damage, brain injury, cancer, multiple sclerosis, spinal cord injuries, and stroke. The detection of physical changes in the brain can allude to ageing, and can be detected by spotting grey-matter loss, shrinkage of the brain, and the decline of white-matter

Deep Learning in MRIs

The medical field has been exploring Convolutional Neural Networks for a number of years now., supposedly since the 1990s [4]. But the recent resurgence in Artificial Neural Networks has paved the way for cutting edge approaches towards neuroimages.

Machine Learning

Machine Learning is a field of Artificial Intelligence (AI) that provides systems the capacity to learn and improve from experience without being explicitly programmed. A computer

program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [5]. Once a Machine Learning model is developed, it can learn from data to discover useful insights and patterns that can reap future rewards. Through continuous development and research, the aim is to create more accurate Machine Learning software that can improve and learn with less human intervention.

Normally, Machine Learning algorithms fall into two different categories. These categories are supervised and unsupervised. Whereas unsupervised learning uses neither labelled nor categorized data, supervised learning uses labelled data to develop the Machine Learning model. Next, we determine the type of Machine Learning problem we are trying to solve; examples of supervised learning problems are Classification and Regression problems. Regression algorithms attempt to predict the mapping of input data to numerical or continuous output, e.g. age: 50 years old, 23 years old. Classification algorithms are almost the same, but instead attempt to predict the mapping of data to discrete or categorical output, e.g. fruit: “apple”, “banana”. With respect to this project, the solution will adopt a supervised learning approach that will solve a classification and regression problem.

Supervised Learning

Supervised machine learning algorithms take what has been learned from experience with labelled data to make predictions about future data or events. By learning from a labelled dataset, the algorithm maps input data to known targets. The learning algorithm can also compare its output with the correct, intended output and find errors so that it can modify the model accordingly. It can be best understood as the following equation

$$y = f(x)$$

where

- y is the predicted output
- f is the mapping function
- x is the input

Fitting the model to the data

The best designed network architectures are models that are deep enough to learn the relationships between the patterns and the data, but not too deep to learn the data’s noise and fluctuations. Poor performance of a Machine Learning algorithm can be pointed to overfitting and underfitting of the data, where a Machine Learning model’s performance can start to stagnate, or decline, after numerous iterations of model training, or show no signs of learning and improvement.

Figure 1 displays three graphs for three different models learning from the same dataset.

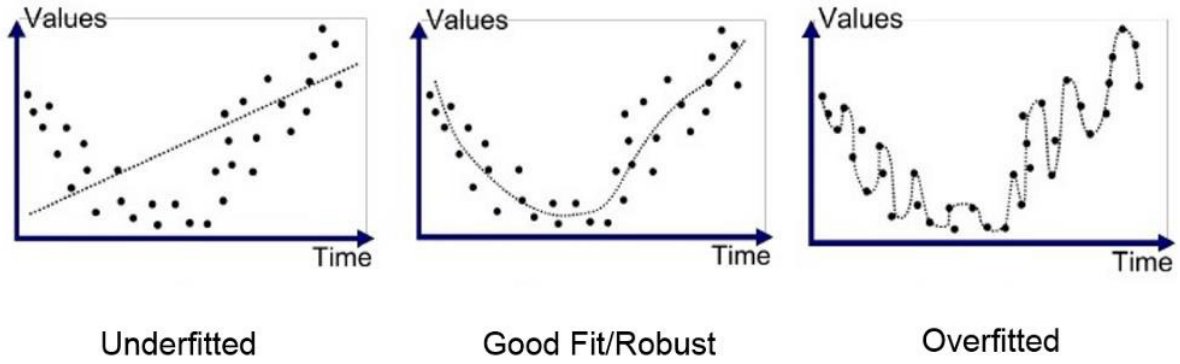


Figure 1: Model Overfitting/Good fit/Underfitting [6]

Underfitting

As the graph on the left of figure 1 shows, underfitting occurs when a model is unable to model the training data, and unable to generalise new data. The source of this problem can usually be attributed to a dataset that is too small. In other words, it is unable to learn anything from the training data. Just like overfitting, underfitting of a model is a detriment to the performance of a Machine Learning model. Underfitting is likely to be spotted earlier on because a model will have poor performance on training data and show no signs of an improvement in performance, where a good enough performance metric should be able to point out. The normal steps to take is to try a different algorithm and inspect for better performances.

Overfitting

As the graph on the right of figure 1 shows, overfitting happens when the model has studied the training data too much and learned every relationship to the point it decreases the performance. With this, the gap between training and test error is abnormally large, i.e. a high accuracy on the training data, but low accuracy on unseen data. While it is necessary to train a Machine Learning to perform as precise as possible with the training data, too much learning can result in the model learning all the detail and noise of the training data, meaning a negative impact on new data. This is because all the noise or fluctuations found in the training data is mistakenly taken as key aspects when discriminating against data. Actions to deal with overfitting include regularisation, more training data, and dimensionality reduction.

Artificial Neural Network

The Artificial Neural Network (ANN) [6] takes inspiration from biology and the brain, where the brain takes in information from the environment, performs hidden calculations, and produces a response. The brain is made up of billions of neurons that communicate with each other through electro-chemical signals., and constantly receive input to bring the body into action and produce an output.

If we wanted to map the relationship of smokers and risk of a health scare, we could construct a standard classification model to classify individuals who are at risk. As stated in the 'Introduction' section of this paper, Artificial Neural Networks can learn and model non-linear and complex relationships, Once the network has been adequately trained, predictions can be made from unseen data.

There are different types of Artificial Neural Networks, but they can be classed in either feed-forward or feed-back (recurrent) networks, and Deep or Shallow Neural Networks. In a feed-forward network, the input signal only travels in one direction, straight from the input layer all the way to the output layer. In a feed-back network, the neurons have signals that can go in both directions, and all possible connections between neurons are allowed. This results in a non-linear system that dynamically changes to reach a desired state of optimisation. Figure 2 below visualises the differences between feed-forward and feed-back networks.

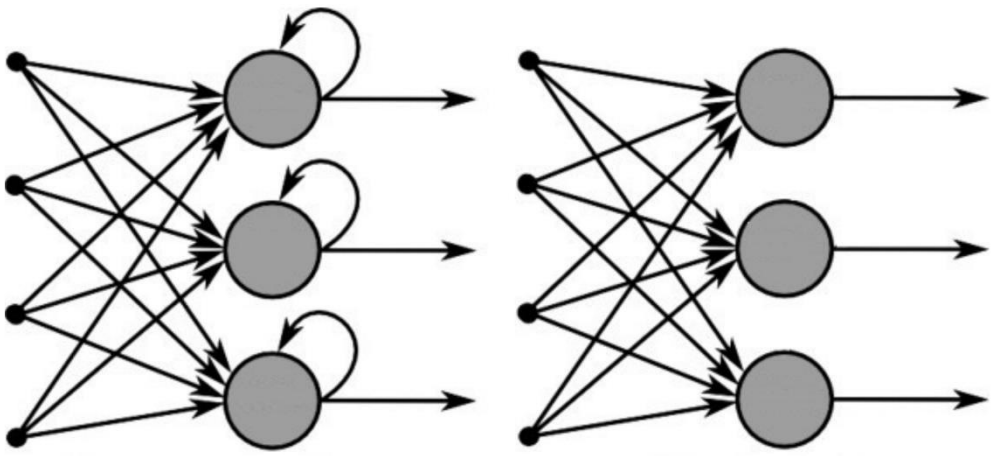


Figure 2 (Left) Feed-back and (Right) Feed-forward Networks [7]

It is hard to define what constitutes a Deep or Shallow Neural Network as there is no definition for it. The only agreement on the topic is that it is conditional on the number of hidden layers a Neural Network has. Some consider anything greater than 1 hidden layer a Deep Neural Network, others suggest a much larger number.

Basic Neural Network structure

Artificial Neural Networks are made up of stacks of layers, which contain a varying number of neurons. The most basic form of an Artificial Neural Network consists of an input layer, a hidden layer, and an output layer, in that order. These layers are a data-processing module [8] that take input from a previous layer, perform calculations on them, and output them to the next layer. We can visualise the structure described through Figure 3 below

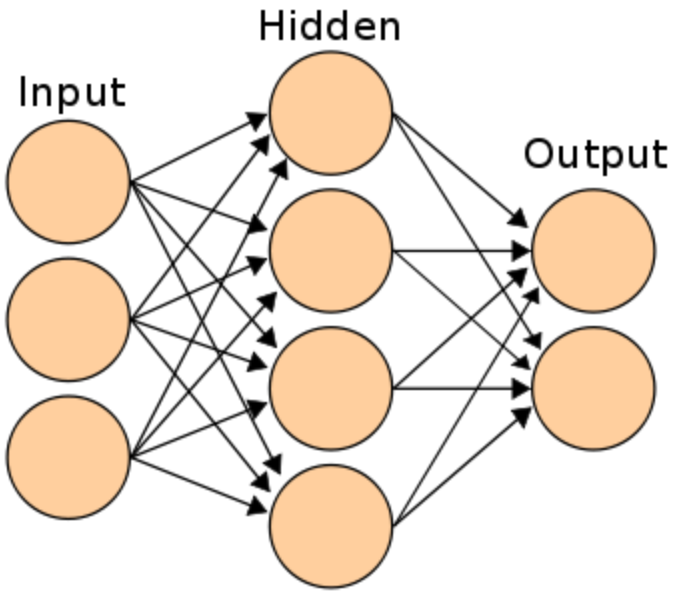


Figure 3 Standard Artificial Neural Network [9]

The Artificial Neural Network in Figure 3 is an example of a basic Neural Network to solve a Classification problem. The reason it is a basic model is because there is only 1 hidden layer. To solve a complex task, a greater number of hidden layers would be required, otherwise the model would have no chance of properly learning from the input data. The model could be classifying a dataset of animal features common to cats and dogs. The network would take properties of a cat and a dog as input and feed that data forward into the network, and eventually transmit into one of the two neurons in the output layer. The number of neurons for the output layer depends on the type of problem, e.g. regression, classification. If the network is modelling a regression problem, there will only be one neuron in the output layer to represent the predicted value of a sample, whereas a classification problem with *n* classes would have *n* neurons to represent the *n* different classes.

It is important to note that discrete values cannot be fed into a neural network, therefore values like 'Dog' and 'Cat', or 18-30 years old must be encoded before being fed into the input layer. The pre-processing of data is a necessary step to ensure that data is in the correct format for the network to learn from and will be covered later.

The Neural Network layers are made up of computational units called neurons [10]. These neurons are interconnected to each other and take in information from $N + 1$ values, where N is the number of neurons, and the +1 comes from a bias value b [11]. Bias b is a free variable that is also trained and gets initialised with a random variable. The neurons within the hidden layer of the network multiplies each of its input x_j from the previous layer's neurons by a weight w_j. The functionality of a neuron can be viewed as the following Equation 1:

$$\sum_{j=1}^{N} x_j \cdot w_j$$

Equation 1 Artificial Neuron: Weighted Sum of Input

where

- N is the number of neurons from the previous layer connected to a neuron.
- x_j are the input values.
- w_j are the weights

The values get summed together, and the sum gets passed to an activation function f. Activation functions are a key element of an Artificial Neural Network, as they are what enables them to learn from complex data and spot patterns, and to map non-linear relationships between the input and output; they create the non-linear property of our network. Without an Activation function, we end up with a simple linear model, and we would be unable to work with more complicated types of data like images and videos. There are different activation functions, but their job is to map the neuron's summed values to a desired range, such as between 0 and 1, or -1 and 1. Some of the more popular Activation functions include sigmoid, tanh, and ReLU. Figure 4 below provides a summary of some popular Activation functions.

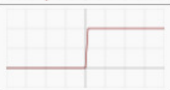
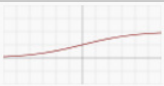
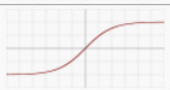
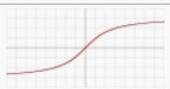
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 4 Activation function plots, equation, and data range [12]

The higher the value, the greater the activation. Depending on the strength of the neuron (output from Activation function), the value gets passed through to the next layer of neurons, until it reaches the output neurons. Finally, the difference in value between the predicted value from the output neuron against the actual value is propagated backwards into the network, adjusting the weight of each neuron in the network's hidden layer(s) that directly contributes to the output, with accordance to the neuron's contribution to the total loss. This process is done for each layer until all neurons in the network have a received a loss signal. The interaction between activation functions and the dot product of weights and input is shown in Equation 2 below:

$$f \sum_{i=0}^{N} \left((x_i \cdot w_i) + b \right)$$

Equation 2 Artificial Neuron: Weighted Sum of Input with Activation Function

To summarise what has been said about the role of Artificial Neurons, a neuron connected to N neurons from the previous layer receives $N + 1$ values from the previous layer, inclusive of bias b, where each x_i of input x gets multiplied by weights w_i of w. The sum of this function is then passed on to the activation function, and the output transferred to the next layer. Figure 5 below visualises the role of the artificial neuron in an Artificial Neural Network.

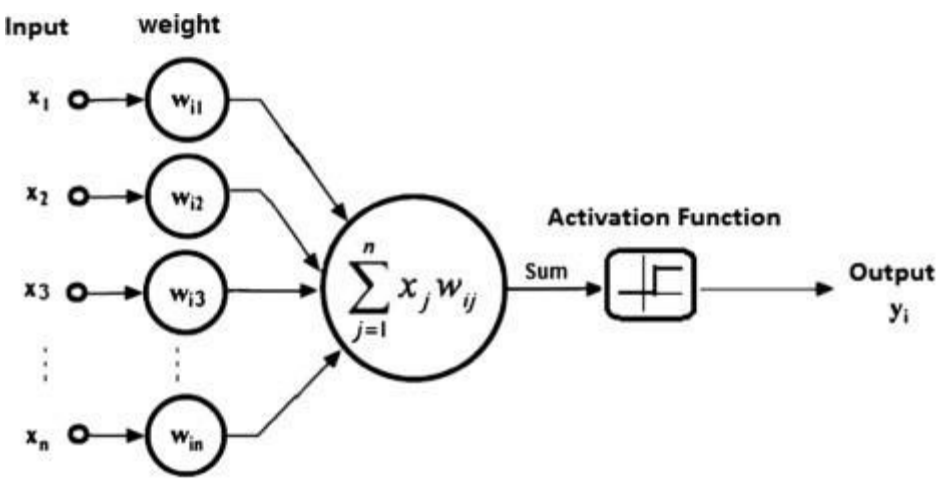


Figure 5 Artificial Neuron [13]

This Artificial Neuron model was inspired by the Perceptron model first proposed by Minsky and Papert [14] which introduced numerical weighting for inputs, and a mechanism for learning those weights.

Training a neural network

Optimal Performance

When an Artificial Neural Network starts off with a randomly assigned set of weights and biases, it learns to map a set of inputs to a set of outputs and makes its predictions, the network produces a Loss. This Loss is the distance score between the predicted values and the output values. Being too high or too low does not matter as much as how incorrect the model is. The model then performs backpropagation to propagate this loss to each neuron that has contributed to the output of the model, using this information to update the parameters (weight, bias) of the neural network with the intention of reducing Loss and producing a better model. These steps are repeated until an efficient model is developed.

Calculating Loss

Artificial Neural Networks can be viewed as an optimisation problem. To optimise the model, we need to reduce the Loss of a model. This can be achieved by readjusting the weight and bias parameters of the model's neurons. To do this, we rely on Loss functions. There are different Loss functions available, we can even create our own, but certain problems (classification, regression) rely on certain Loss functions.

Loss Functions

Loss functions are a key element in training a Machine Learning model. They are a method of evaluating how well an Artificial Neural Network models a dataset. When we adjust an Artificial Neural Network, we use the Loss functions as an indicator of steps in the right or wrong direction. Take the following as a functional example of calculating Loss for the ages of MRIs. The Loss function used in the following table is a custom-made Loss function that calculates loss as the absolute value difference between predictions and true value:

Network predictions	Actual Values	Calculated Loss
1st MRI: 23 years old 2nd MRI: 41 years old 3rd MRI: 70 years old	1st MRI: 26 years old 2nd MRI: 40 years old 3rd MRI: 73 years old	6 (we were off by 6 across all predictions)
1st MRI: 26 years old 2nd MRI: 40 years old 3rd MRI: 73 years old		0 (predictions were all correct)

Table 1 Custom Loss function

The Loss function above can be calculated by the following Equation 1:

$$\sum_{i=0}^{N}(|t_i - p_i|)$$

Equation 1 Custom Loss Function

where

- N are all the samples
- t_i is the true value
- p_i is the predicted value
- $|t_i - p_i|$ is the absolute value of $t_i - p_i$

Generally, for all Loss functions, they do not take into consideration whether incorrect a prediction was higher or lower than expected, and only cares about how far off the prediction is from the true value.

Mean Absolute Error

Mean Absolute Error (MAE), or L1 loss, measures the average magnitude of the errors (actual value – predicted value) in a set of predictions for a Regression model. A popular choice for Regression models, it is given by the following Equation 2:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Equation 2 Mean Absolute Error

where

- n is all training samples
- y_i are the true values
- $\tilde{y}_i$ are the predicted values

Mean Squared Error

We use a Mean Squared Error (MSE), or L2 loss, to calculate Loss for a Regression problem. Mean Square Error measures the average of the squares of the error. Its value tells you how close the set of points of a Regression model are to the Regression line. Arguably the most common Loss function, Mean Squared Error can be calculated using the following Equation 3:

$$\frac{1}{n}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2$$

Equation 3 Mean Squared Error

where

- N is all training samples
- y_i are the true values
- $\tilde{y}_i$ are the predicted values

Mean Squared Error vs Mean Absolute Error

MSE and MAE are both used for Regression problems. The question, then, is which Loss function should we use? Observe the values in the following table, where $|Error|$ is the absolute value:

$Error$	$\|Error\|$	$Error^2$
0	0	0
-0.5	0.5	0.0625
1.25	1.25	1.5625
10	10	100

$$\begin{aligned} Mean\ Absolute\ Error &= \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \\ &= \frac{1}{4}\sum_{i=1}^{4}|Error_i| \\ &= \frac{1}{4}\sum_{i=1}^{4}|0 + 0.5 + 1.25 + 10| \\ &= 2.975 \end{aligned}$$

$$\begin{aligned} Mean\ Squared\ Error &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \\ &= \frac{1}{4}\sum_{i=1}^{4}(Error_i)^2 \\ &= \frac{1}{4}\sum_{i=1}^{4}(0 + 0.5 + 1.25 + 10)^2 \\ &= 34.515625 \end{aligned}$$

The results show that Mean Absolute Error is less than Mean Square Error, suggesting that a model with this result has performed well (the closer to 0, the better). However, given that Mean Square Error squares the error, the value of $Error^2$, and in turn Mean Square Error, increases exponentially when $Error$ is greater than 1. This results in the model giving more weight to any outliers. In a dataset filled with outliers, Mean Absolute Error loss is more useful because it gives less weight to outliers than Mean Square Error.

One big problem in using MAE loss (for neural nets especially) is that its gradient is the same throughout, which means the gradient will be large even for small loss values. This isn't good for learning. To fix this, we can use dynamic learning rate which decreases as we move closer to the minima. MSE behaves nicely in this case and will converge even with a fixed learning rate. The gradient of MSE loss is high for larger loss values and decreases as loss approaches 0, making it more precise at the end of training (see figure below.)

In summary, outliers need to be considered when selecting a Loss function. If those outliers are an integral part of the dataset, then Mean Square Error should be adopted. But if the outliers represent corrupted data, we should use Mean Absolute Error

Categorical Cross-Entropy

Categorical Cross-Entropy loss, or Log loss, is a popular loss function for Multiclass classification. We use this Loss function when we are mapping samples to only one class out of a possible Nclasses. The Categorical Cross-Entropy is calculated for each prediction and label pair, and then the sum of loss for each pair gives the total Loss for the model. Categorical Cross-Entropy is given as Equation 4 below:

$$-\sum_{i=1}^{n} y_i log\widehat{y_i}$$

Equation 4 Categorical Cross-Entropy

Optimisation Function

The learning of an Artificial Neural Network comes from adjusting the weights w and bias b during training to reduce Loss on the batch. We start by training the network model on randomly initialised weights w and bias b, calculate the Loss, and then modify the weight and bias of the neurons in a model by using an Optimisation function to reduce Loss. The aim of an Optimisation function is to find the correct values of weights w and bias b that minimise Loss, therefore, we can derive the function $O(w, b)$.

Gradients play a big role in how an Artificial Neural Network learns. A higher gradient provides a steeper slope that allows a model to learn faster. If the slope is zero, the model stops learning. Gradient descent [15] is a common optimisation strategy when training a Machine Learning model and involves iteratively adjusting values to minimise the Loss function. To find the local minimum, the algorithm takes steps proportional to the negative of the gradient of the function at the current point and tweaks parameters iteratively with each step down the gradient. See Figure 5 below for an illustration of this.

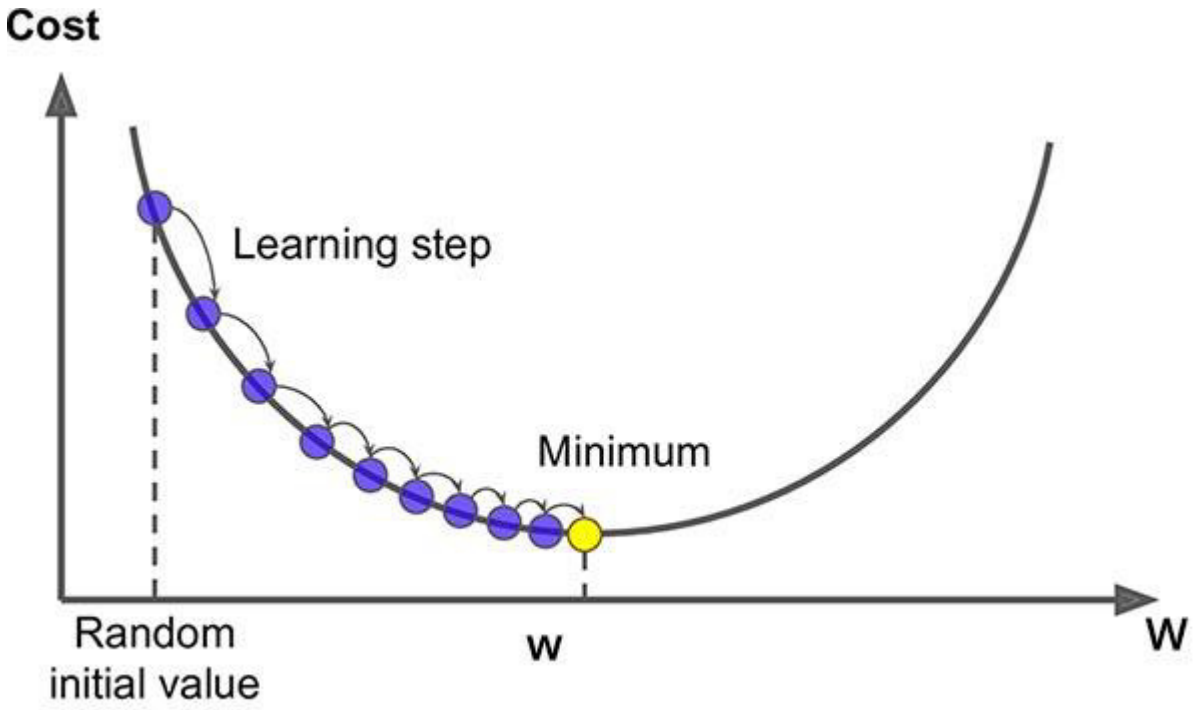


Figure 5 Gradient Descent Optimisation [15]

In Figure 5, we start at a random value, take the value of the Learning Step (how big we step) to move down towards the Global Minimum. As we move further down the slope in proportion to the negative of the gradient (where the line slopes downwards), we want to decrease the Learning Steps with each iteration but also adjust the parameters to move towards the global minimum. However, a Learning Step too big risks being trapped in local minima, and a Learning Step too large could result in the algorithm missing the global minimum all together. See Figure 6 below for an illustration of the effects of Learning Rate.

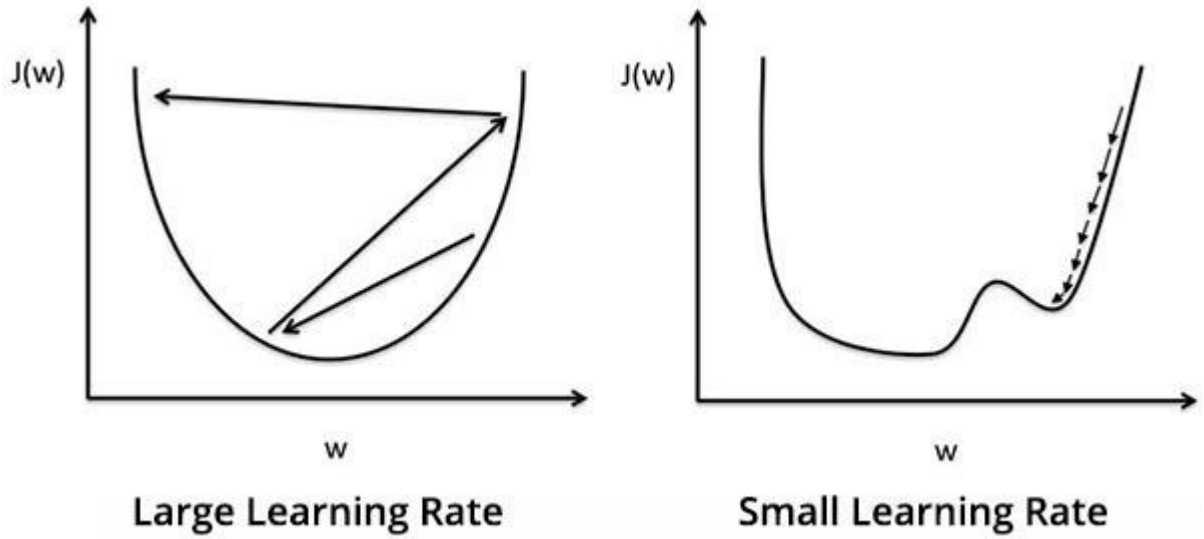


Figure 6 Effects of Learning Rate for Gradient Descent [15]

Stochastic Gradient Decent Algorithm

As we have already discussed, the aim of an Optimisation function is to minimise the Loss produced by the Loss function. An Optimisation function like Gradient Descent attempts to find the Global minima of a Loss function, and Gradient Descent does this by allowing a model to learn the direction on a gradient the model should take to reduce Loss and Global Minima. The model starts with randomly initialised values for parameters such as weights w and bias b, and calculates the Loss for the current weight and bias values. With each iteration of the model and further adjustments to the parameters, the model produces less and less Loss. The interaction between the Gradient Descent and Loss function is a feedback signal communicating how to improve performance.

Stochastic Gradient Descent calculates the error for each training sample in the dataset, updating the parameters for the samples one by one. Due to the frequent updates, this drives up the computational cost. Furthermore, the frequent updates can potentially result in noisy gradients, causing the error to become high then low, rather than slowly converging to a Global Minima

Adam Optimisation Algorithm

The Adam optimisation algorithm, derived from Adaptive Moment Estimation, is an adaptive learning rate method based on the Stochastic Gradient Descent algorithm used to update the network's weights and bias when training. The benefits of this algorithm [16] are stated from its paper with the following, "The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning". Adam has the advantage over the Stochastic Gradient Descent algorithm in that it scales the learning rate for individual parameters to reach the convergence of a gradient, but it is not always guaranteed to perform better than Stochastic Gradient Descent.

Dropout

Overfitting (explained earlier under "Background") is a common problem that neural network models must deal with. To counteract Overfitting, we implement the Dropout technique, an approach for regularisation in neural networks to reduce interdependent learning for a set of neurons. Dropout forces a neural network to learn more about the features more present in the random subsets of neurons and reduces training time. This technique involves temporarily dropping random Neurons in the network model during training, along with any of its inbound and outbound connection signals. Through Dropout, we construct numerous thinned networks where "a neural net with n units, can be seen as a collection of 2^n possible thinned networks" [17]. Once the testing comes around, all the neurons are used again but each activation is reduced by a factor of p to act as a scaled down version of the training weights. Figure 7 below is a visualisation of Dropout on a Feed-forward network.

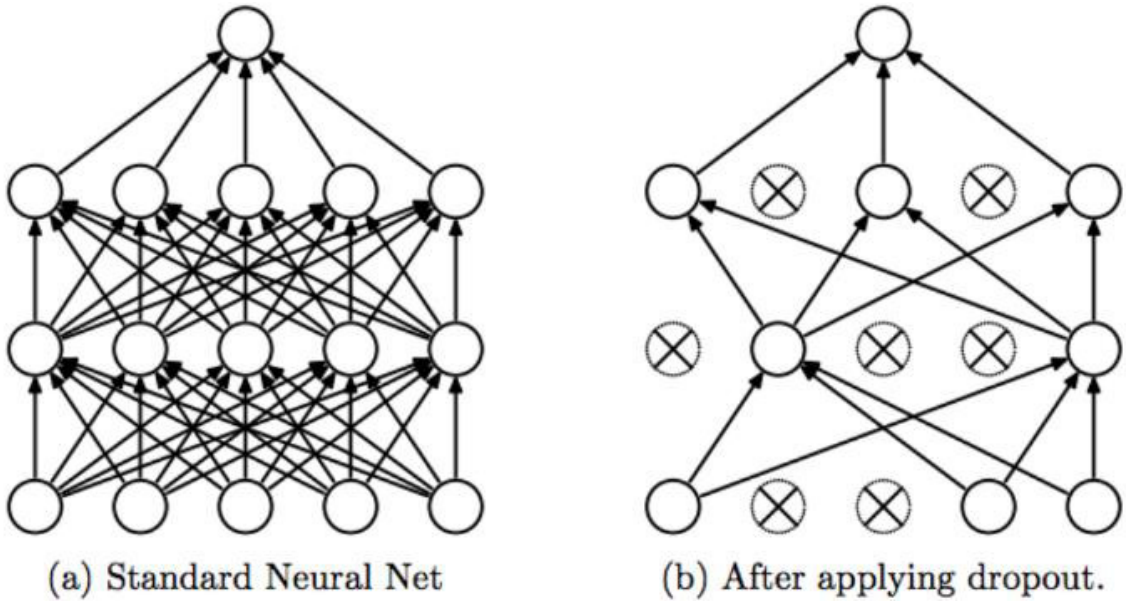


Figure 7 Application of Dropout [18]

Convolutional Neural Networks

The type of neural network used for this project is Convolutional Neural Networks (CNNs). They are frequently used in image processing, hence why they are adopted for this project, but can also be used for other problems like text classification. A popular starting point for many software engineers getting into CNNs is to develop a 2-dimensional Convolutional Neural Network for classifying the MNIST dataset [19]. Rather than studying a whole image all at once, CNNs can split the task up into smaller jobs by learning patterns of an image from its local segments to recognise it on any other part. E.g., a Convolutional Neural Network learns to recognise the digit '9' from the bottom right of an image and can now recognise it from any other part of an image. The general concept of CNNs is to recognise properties of images, such as detecting elements of interest or the pixel positions. By learning these patterns, a Convolutional Neural Network has a stronger chance of making these predictions because they learn from the local patterns rather than learning from the whole image.

Building Blocks of a Convolutional Neural Network

Convolutional Neural Networks are a class of deep, feed-forward neural networks made up of one or more convolutional layer. A CNN's first layer is always the Convolutional layer, but can go on to include other layers such as pooling, bath normalisation, and fully-connected.

Convolution Layer

Convolutional Neural Networks employ a mathematical operation called convolution to extract features from the input map, which is carried out by the convolution layer. The convolution function itself is an operation on 2 functions that produces a 3rd function.

Within a convolutional layer, features are extracted from an image whilst preserving the spatial relationship between pixels. This is accomplished by going over the full width and

height of the input image and calculating the dot product between a 1D or 2-dimensional filter (kernel) and the input at any position. Through each forward pass in the network, a 2-dimensional map storing the responses of the filter at every spatial position on the image is produced. Through learning, the network comes to learn more intuitively what filters activate when it sees a feature of interest such as an edge. The functionality of the convolutional layer is determined by the parameters, such as kernel size, number of filters, stride, and activation function. This is shown through figure 8 below.

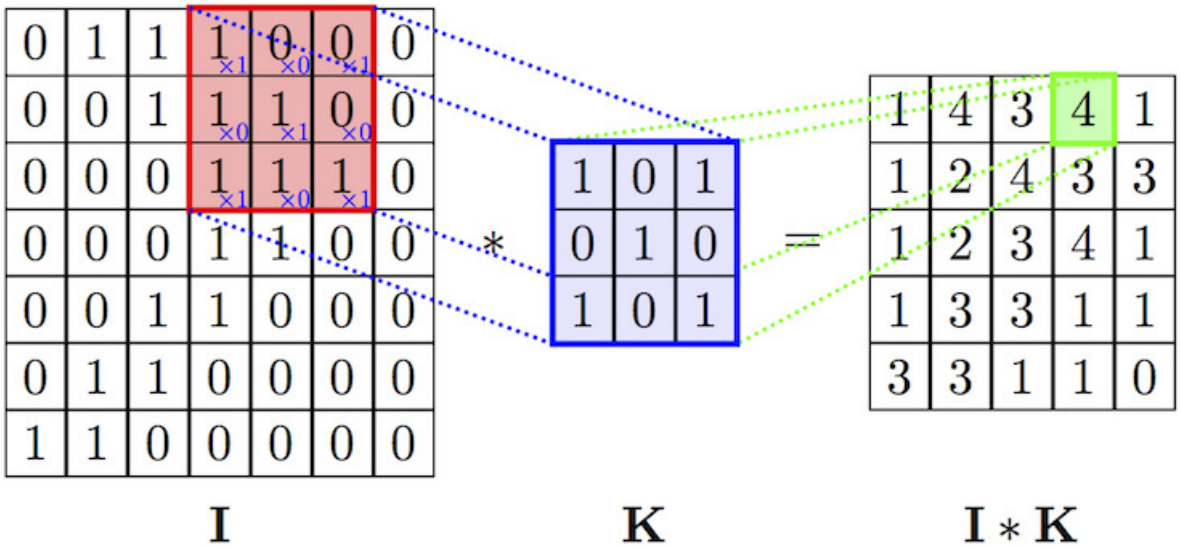


Figure 8 Convolution Operation [20]

Pooling Layer

Convolutional layers are commonly followed by pooling layers, which reduce the data dimensions by down sampling feature maps. Inserting these layers reduces the spatial size of the feature map to reduce the number of parameters and computation in the network and reduces the probability of overfitting. It also serves to allow for assumptions about the features contained in these regions, such as uncovering the nose, mouth, or eyes of a person's face. One of the most common pooling layers is max-pooling. The max-pooling layer acts as a window. Figure 9 is an example of a max-pooling layer operating on a 4x4 matrix, with a filter size of 2 and a stride of 2.

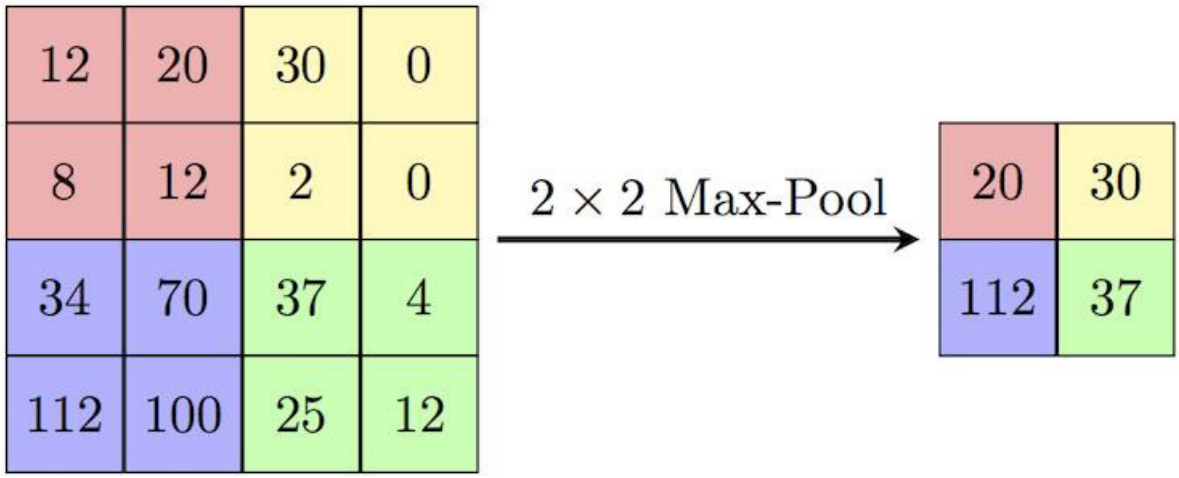


Figure 9 Max-Pooling of 4x4 Matrix [21]

In figure 9, the max-pooling layer will inspect the original 4x4 matrix as a 2x2 window at every iteration. Starting at the red region, it will select the element with the highest value and store that in the new output matrix. As we have a stride of 2, the max-pooling layer moves 2 pixels across to the yellow region and the same steps are followed. Eventually, the whole matrix is filtered out, and we are left with a 2x2 matrix with the maximum element values from the previous input. Observe Figure 10 below for a visualisation of the effects of max pooling on an image.

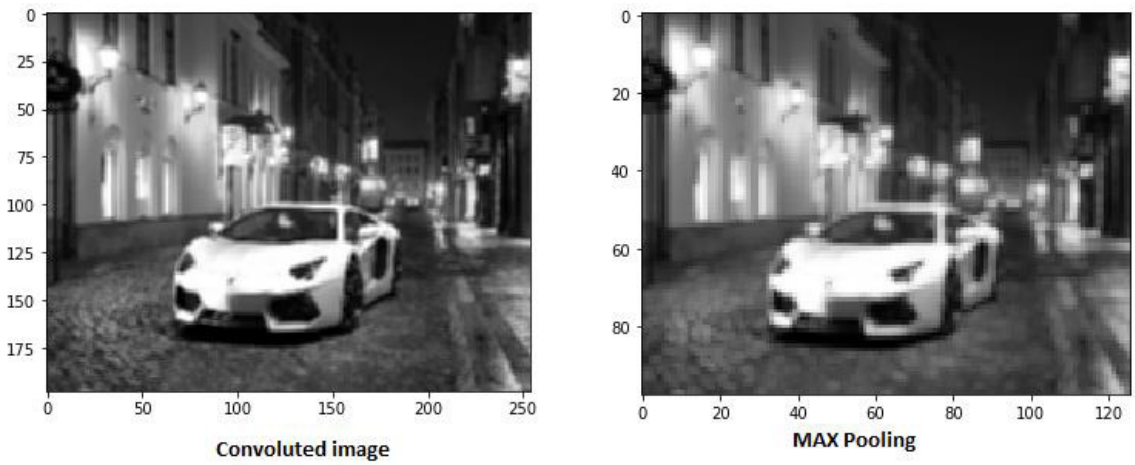


Figure 10 Effects of Max-Pooling [22]

Figure 10 is an example of what happens when we cut down on the dimensionality of an image via max pooling. The core outline and features of the original image are retained, and we cut down on the number of parameters in the network.

Fully Connected Layer

Coming after a sequence of convolutional and pooling layers, the neurons in the fully connected (dense) layer has full connections to all activations of the previous layer and connects each of these neurons to every neuron in the next layer.

Batch Normalisation

The batch normalisation layer acts as a pre-processor on the previous layer's output. Imagine the difference between a grayscale image and a coloured image. The grayscale image will have pixel values ranging from 0 to 1, whereas the coloured image will have pixel values ranging from 0 to 255. To retain the similarity of the input, we normalise the data to ensure the values are within the same range, e.g. keep all data within the range between 0 and 1. Furthermore, batch normalisation allows layers in the network to be more independent of each other as it allows the adjustment of learning rates and reduces overfitting because it has a regularisation effect on the data. This is because it adds some noise to the activation functions.

One-hot Encoding

Discrete data, such as Age: 18-30, 31-45, 46-60 years old etc., cannot be directly fed into a neural network. To use the data, we encode it into an appropriate format. We can vectorise the data to accomplish this. One way of doing this is one-hot encoding [8]. This encoding method is a widely used format for categorical data, where the labels are labelled as an all-zero vector with a single 1 in the place of the label's index. This satisfies the requirement that classification models must be fed discrete data. The Keras framework has built-in methods to convert labels into one-hot encoding format. Table 2 below shows the conversion of three class labels (0, 1, 2) into one-hot vector encoding.

	Class label 0	*Class label 1*	*Class label 2*
Patient 1	*1*	*0*	*0*
Patient 2	*1*	*0*	*0*
Patient 3	*0*	*1*	*0*
Patient 4	*0*	*0*	*1*

Table 2 One-hot Vector Encoding for Patients' ages

The example from table 2 demonstrates one-hot encoding to represent continuous data as categorical data. Each sample in figure 3 is labelled as a vector of 0s and a single 1, where the index of the 1 represents the class of the label, so patient 1's class is 1. The one-hot vector encoding of Patient 1 is [1, 0, 0], and Patient 3 is [0, 1, 0].

The reason we cannot just use the numerical class labels of the data is because the Artificial Neurons function $f((x_i \cdot w_i) + b)$. The equation does not fairly represent the class labels, why should we apply the Artificial Neuron functionality to a class label of 0 or 1, let alone place greater magnitude on a higher-class label value?

Performance Metrics

To test the strength of a model, a model needs to go under performance evaluation. Different choices are available, but different problems call for specific choices when choosing a performance evaluation metric. These metrics can then be taken and compared against other models to see which approach is better.

Accuracy

Accuracy is defined as the number of correct decisions / total number of samples. It is calculated using the formula below:

$$accuracy = \frac{number\ of\ correctly\ predicted\ samples}{number\ of\ samples}$$

For example, if an algorithm correctly predicts 50 items out of 100 samples, this produces an Accuracy of 50%.

Confusion Matrix

A confusion matrix table focuses on the predictive capability of a classification model on test data where the actual values are known.

- True positive – The number of correctly predicted samples in the positive category
- True negative – The number of correctly predicted samples in the negative category
- False positive – The number of incorrectly predicted samples in the positive category
- False negative – The number of incorrectly predicted samples in the negative category

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Figure 11 Confusion Matrix illustration [23]

Convolutional Neural Networks vs Basic models

To justify the use of Convolutional Neural Networks against more basic models, we compare the results of our Convolutional Neural Network models against more basic Classification

models at a 2-dimensional level. The expectation is that the Convolutional Neural Network will outperform each model. The following basic models will be evaluated on the following problems:

- Naïve Bayes
- Random Forest
- Support Vector Machines

Naïve Bayes

Naïve Bayes [24] is a type of Supervised Learning conditional probability model to predict the class of a given sample. Naïve Bayes is probabilistic in that it calculates the probability of each data sample belonging to a given class, and then outputs the class prediction with the highest value. It is given by the following Equation 5:

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)}$$

Equation 5 Naive Bayes Theorem

where

- $P(A|B)$ is the probability of event A, given event B.
- $P(B|A)$ is the probability of event B, given event A.

Decision Trees

We use Random Forest Classifiers as another basic model, which employ Decision Trees. Decision Trees [25] can fit complex data sets while allowing a user to see how a decision was made. The Decision Trees answer sequential questions to go down a certain route of the tree to reach a suitable answer, e.g. Is it Sunny Is it cold? They are easy to interpret. Can handle numerical and categorical data, meaning they can be used for Classification and Regression, and are quite fast. However, they can be prone to Overfitting when the tree depth is quite large because a sample might meet the conditions for a tree at a deeper depth but not earlier in the tree, and the choice of algorithms implemented can lead to finding local optimums rather than global optimums.

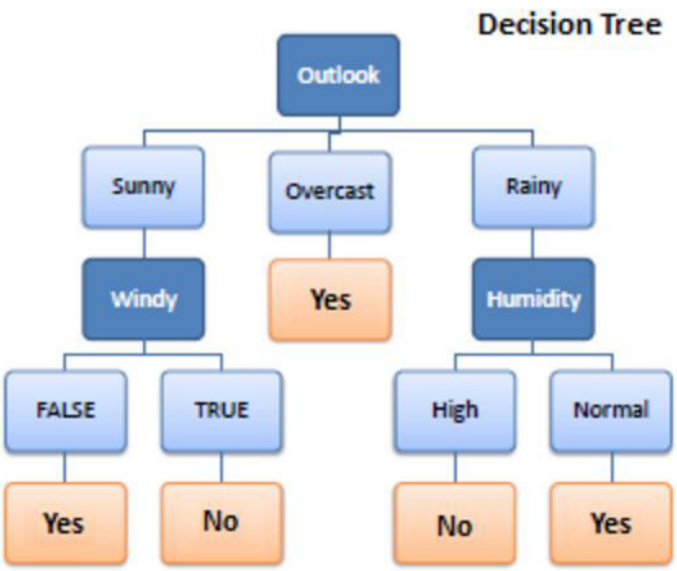


Figure 12 Example Decision Tree [25]

Support Vector Machine

Support Vector Machines are supervised learning models that can be used for Classification or Regression. Given a training x, and the training labels y, an SVM algorithm build a model to assign unseen data to one out of N classes.

Software and Libraries used

Python

The Python [26] programming language has fast become one of the most popular languages in AI. Its community support and ease of use as a high-level language make it a strong choice not only in research endeavours, but in the industry as well.

Sci-kit Learn, NumPy, Pandas

- Scikit-learn [26]: A collection of machine learning algorithms for supervised and unsupervised learning in Python. Provides efficient tools for data science tasks.
- NumPy [26]: Supports large, multi-dimensional arrays (tensors) and matrices in Python. Also comes with a wide range of mathematical functions to manipulate these data structures.
- Pandas: Useful for data manipulation, visualisation, and analysis, especially with tabular data.

NiBabel

NiBabel is a Python library [27] for read and write access to medical and neuroimaging file formats. It is used to extract meta information about MRIs and to access the image data of the MRIs. The image data is stored in a NumPy array.

Keras

Keras [8] is a high-level API framework written in Python for developing Neural Networks. In this project, Keras was used because the library makes it easy and quick to develop prototypes and supports CNNs. Keras runs seamlessly on CPU and GPU. For this project, Keras uses Tensorflow as the backend

Google Colab

Google Colab [28] is a free cloud service acting used for its python development environment that runs programmes through the web with either a CPU, GPU, or TPU processor. Software and libraries are easy to import. While the service is free, every instance of a virtual machine can only be run for 13 hours. This presents a problem because some Machine Learning jobs taker longer than 13 hours. Once the time expires, a new instance will have to be instantiated. This can be done an unlimited amount of times. Google Colab is best treated as a prototyping environment for less-demanding computational jobs which is why it will be used for modelling 2-dimensional MRIs.

Supercomputing Wales (SCW)

Supercomputing Wales [29] provides access to powerful computing facilities for computationally demanding tasks and problems, like the one being solved in this project. Access is strict and only given to authorised users. After successfully applying, SSH is used to connect to the facilities. The computationally demanding 3-dimensional Convolutional Neural Network model depends on running on the supercomputer cluster for GPU training of the Neural Network.

Tensorflow.

Tensorflow [30] is an open source Machine Learning framework to develop and train models. Tensorflow can be used to define a range of algorithms, including training and inference algorithms for deep neural network models, and it has been used developing Machine Learning systems for computer vision.

Approach

The aim of the system is to take a Machine Learning approach, more specifically, Convolutional Neural Networks, and predict the age and cognitive variables of a given 2-dimensional and 3-dimensaionl MRI. This section covers the design of the Artificial Neural Network, and the data pre-processing.

Business model approach & user interface

This solution has been conducted in the interests of scientific research to see if the proposed approach is suitable for tackling the project's problem. The intended approach has been created for someone with enough command-line experience to run the solution. Should the results of this project prove successful and show potential for commercial uses, the solution can be updated to provide better human-computer interaction for inexperienced users. One improvement would be to include a graphical user interface (GUI) for users not experienced with using the command-line interface.

Data flows of the system

At a high-level, this solution is a Machine Learning approach to predict variables from a 2-dimensional and 3-dimensional MRI. The system takes as input a multi-dimensional NumPy array containing all the un/cropped MRIs and, depending on a regression or classification problem, the dataset labels in the form of a NumPy file containing a vector of integer labels, or a multi-dimensional array containing the one-hot vector encoding of classes The output generated is the evaluation metrics generated from the training, such as Accuracy, Confusion Matrix, MAE, and MSE. Each of these parts are carried out in different Python files. For a visual explanation of the general system data flow, follow Figure 13 below.

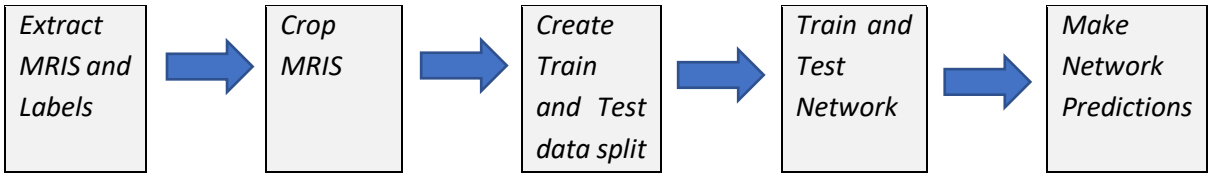


Figure 13 Data Flow of the System

Data pre-processing

The MRI dataset is stored in a shared directory on the supercomputer cluster and must be extracted from there. We extract the MRIs from the dataset's directory and they ultimately end up as a NumPy array. There are 653 unique MRIs of high-res or low-res quality. The 2-dimensional images are used with Google-Colab for the earlier part of the project, but we move to the supercomputer cluster for the 3-dimensional images in the later stages; using the 3-dimensionalimages requires more computational power.

The cognitive variables for each unique MRI are stored in a file called 'CC700_mt.txt'. To extract the labels for the dataset, we take the unique ID for every instance of an MRI, look it up in the text file, and store the value of the required label. The valued is then inserted into an array, and every subsequent value stored is added to the end of the array. Doing this ensures that the labels are stored at the same array index as the MRI data, making it easier to access the related MRI data and label.

In their original format, the labels extracted from the text file are ready to be used for a Regression model. But since the data types of the labels (age, memory) are in numerical format, we need to make modifications to the data to make it compatible with a classification problem. We go through the values of the variables and assign a numerical class label to represent the group that the specific MRI belongs to. Once we have done this, we use one-hot encoding8 to represent the class labels as vectors. After these steps are completed, the data can be fed into the classification model.

$Full\ Dataset - Classification$		
$Class\ Label$	$Samples$	$Representation$
$0\ (18 - 30\ years\ old)$	80	12.25%
$1\ (31 - 40\ years\ old)$	105	16.07%
$2\ (41 - 50\ years\ old)$	101	15.46%
$3\ (51 - 60\ years\ old)$	101	15.46%
$4\ (61 - 70\ years\ old)$	104	15.92%
$5\ (71 - 79\ years\ old)$	100	15.31%
$6\ (80 - 87\ years\ old)$	62	9.49%

The table shows that some classes have a higher frequency than others, but since there is no great disparity of the distribution, there is no need to make any adjustments to the dataset. Choosing to have a 71-79 and 80-87 may seem odd, but this was to increase the representation of the class by including 80-year olds into the class, going against the pattern set by the previous class labels (upper bound as a multiple of 10). Figure 14 is a histogram visualisation of the data split mentioned above.

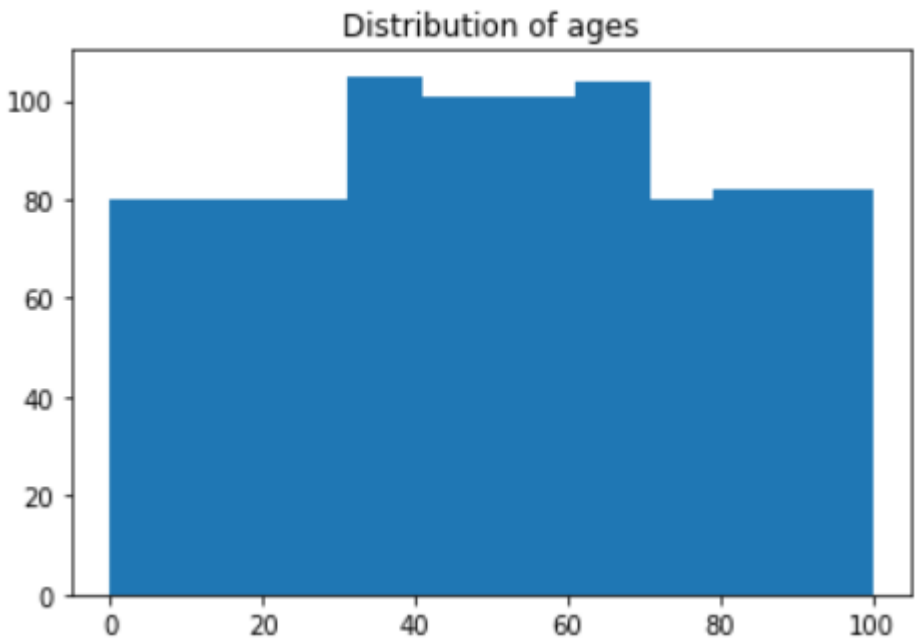


Figure 14 Histogram of the Age Distributions

Cropping the MRIs (conditional).

This step is required when training the model on the supercomputer cluster with the 3-dimensional high-res MRIs. To improve performance, and work with hardware constraints, we cut the size of each high-res MRI by cropping the images. The MRIs are reduced from their original size of 181x217x181 to 96x112x96. Cutting down on data from the MRIs removes redundant data, but also helps remove noise. After cropping the images, each MRI is inserted to the end of a NumPy array. The order of the array elements is in accordance with the NumPy array containing the labels for the dataset.

One-hot vector encoding (conditional)

This step is only initiated when creating a classification model. The dataset labels from the first step are passed into this step for one-hot vector encoding. When designing a classification model, we use one-hot vector encoding on the NumPy array of labels to turn the values into vectors of 0s and 1s. If we label all MRIs with a class label between 1 and 5, and take MRI instances with the class labels 4, 5, 1, and 2, the instances will have the one-hot encodings of [0,0,0,1,0], [0,0,0,0,1], [1,0,0,0,0], and [0,1,0,0,0]

Convert 3-dimensional MRIs into 2-dimensional slices (conditional)

The MRIs in their original state are 3-dimensional volumetric images (x*y*z). Each axis is made up of multi-dimensional vectors containing float32 bits. The full scope of multi-dimensional vectors is accessed when used at a 3-dimensional level. If we want to access the MRI at a 2-dimensional level, we must select a specific axis index. Take figure 15 belowas an example. Let the 3-dimensional shape be represented as an array image[:, :, :], where : represents all values of the axis, and the 2-dimensional shape represented as an array image[2, :, :]

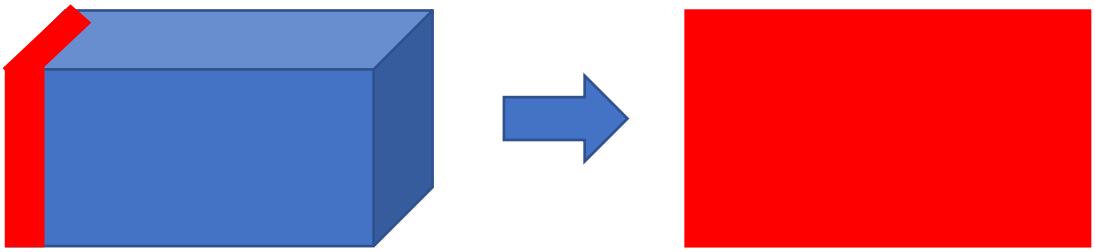


Figure 15 3-dimensional to 2-dimensional

The reason that the shape on the left-hand side is currently a 3-dimensional shape is because every single multi-dimensional array in the MRI is in use. But the shape on the right-hand side is 2-dimensional because not every tensor is used; we use one multi-dimensional array at one of the axes while the rest of the arrays are omitted. This is what converts the 3-dimensional MRIs into 2-dimensional.

Train/test data split

The original dataset contains 653 unique MRIs. To split the dataset, the Scikit-learn library is used. These MRIs, along with their labels, are stored in NumPy arrays. This data is split in two subsets: training data and test data. We train our model on the data, and the model makes predictions on the test data. The data split ratio is 80/20, that is 521 MRIs for training, and 132 for testing. This ratio was chosen because there's not a great deal of data available, so we need to ensure there is enough data to train the model, while still providing enough data to test the model. The frequency of each class label will be kept roughly the same as the frequency of the full dataset, and the split can be viewed below. Figure 16 below is an example of what the training/test ratio distribution could be.

Training Set				Test Set		
Class Labels	Samples	Representation		Class Labels	Samples	Representation
0	64	12.28%		0	16	12.12%
1	84	12.28%		1	21	15.90%
2	80	15.35%		2	21	15.90%
3	80	15.35%		3	21	15.90%
4	83	15.93%		4	21	15.90%
5	80	15.35%		5	20	15.15%
6	50	9.59%		6	12	9.09%

Figure 16 Possible Distribution of Classes

Convolutional Neural Network

The final part of the software is the Convolutional Neural Network. The network takes in input from the previous steps to start training the network. The input is the two NumPy arrays

containing the MRI dataset and the labels data. For this section, there are two major tasks carried out that will be explained in their own subsections.

Feature extraction

After being fed the input data from the NumPy files, the Convolutional and Pooling layers of the network extract the features from the data. The network learns the relationships between the MRIs and the attributes of the data. The output is a tensor whose values depends on the modifications performed by the various functions of the Convolutional and Pooling layers with respect to bias and weight. The Feature Extraction is carried out by 5 pairings of Convolutional and Max Pooling Layers, where their functionality is explained in the "Background" under "Convolutional Neural Networks". Every sample is convoluted by the Convolutional layer, then down sampled by the Max Pooling layer. The Convolutional and Max Pooling layers will be either 2-dimensional or 3-dimensional, depending on the model being constructed. The Classification and Regression jobs can only take place on a 1-dimensional feature map, so all the data from the Feature Extraction gets passed to the Flattening layer. In the Flattening layer, the aggregated information of the Feature Extraction process gets passed transformed from either 2-dimensional or 3-dimensional, into a 1-dimension vector by multiplying the dimensions of the previous layer's channels together. For example, a 3-dimensional sample of 16*19*64 gets converted into a vector of 19456. See Figure 17 below for a diagram of the Feature Extraction structure.

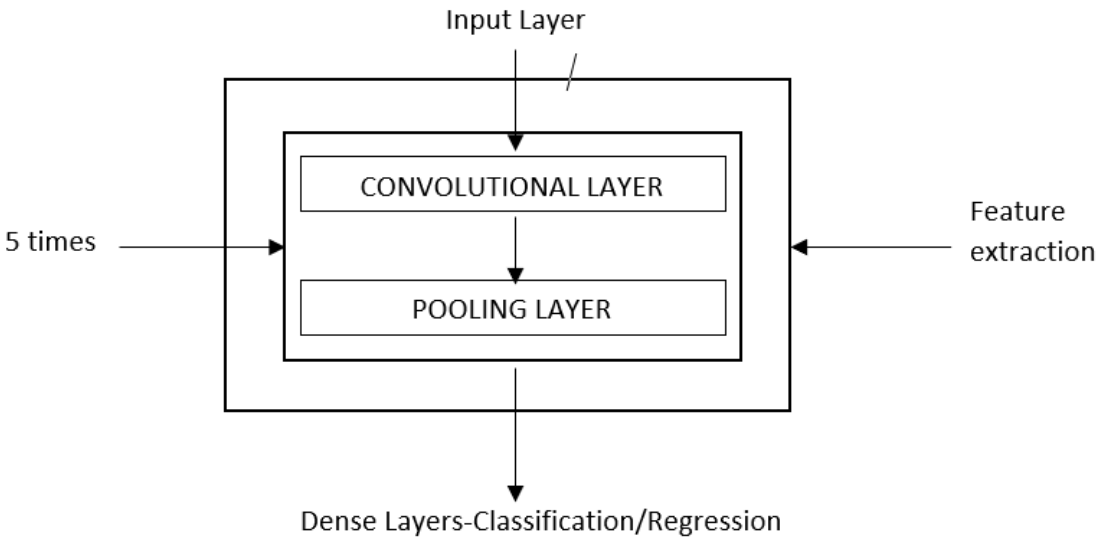


Figure 17 Convolutional Neural Network: Feature Extraction

Classification and Regression

The design of the Classification and Regression structure is generally the same across all versions. Once the features have all been extracted by the Convolutional components and flattened into a 1-dimensional vector, the information is fed into either the Regression or Classification components where the final output is produced.

For the Classification model, we use two Fully Connected Layers to classify the feature map. The Fully Connected Layers' individual neurons are connected to every neuron of the previous layer and put forward to 256 neurons in the first Fully Connected layer, which is then passed on to the 7 neurons in the final Fully Connected layer. The number of neurons we choose for the first Fully Connected layer can be any number we see fit, but for the final layer it must be the same as the number of classes, therefore the final layer has 7 neurons. The final Fully Connected layer outputs a set of predictions for the training sample, but in their original state they are a set of unnormalized set of probabilities. The final Fully Connected layer uses an activation function ('Softmax') to normalise the network's predictions so that they can be interpreted as probabilities with a range from 0 to 1.

For the Regression model, we use follow the same structure as the Classification model but change the number of neurons in the final Fully Connected layer to 1. No activation function is used for the final Fully Connected layer because Regression problems are linear and using anything other than the default 'Linear' activation function would result in a non-linear output. See Figure 18 below for a diagram of the Classification and Regression structure.

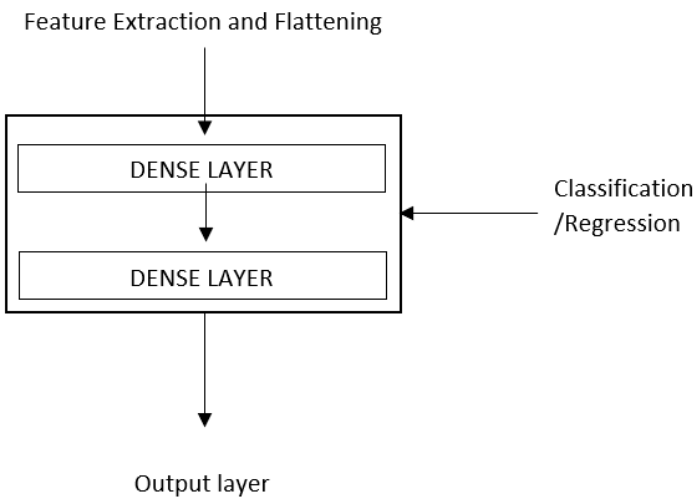


Figure 18 Convolutional Neural Network: Classification and Regression

Loss and Optimisation functions

Artificial Neural Network problems can be understood as gradient optimisation problems, as highlighted in the "Background" under "Optimisation Functions". An Artificial Neural Network learns by calculating the difference in value between the network's predictions and the true value, and then readjusting the weights and bias to minimise Loss. The types of loss and optimisation functions depend on the task (Regression or Classification).

As we have two different models, we have two different sets functions. For Regression, we are using Mean Square Error, and we use Categorical Cross-Entropy Loss for Classification, both discussed in the "Background" under "Optimal Performance". We use Mean Square Error for Regression because we want to measure the network's predictions distance to the Regression line to produce a linear result. Then, use Categorical Cross-Entropy Loss for our Classification model because the network is mapping the input to one of 7 different classes, the type of mapping that Categorical Cross-Entropy is used for.

The Optimisation functions we use for our Regression and Classification models is Adam.

Float32 from Float64

The MRIs are originally float64. With consideration to the hardware constraints, it is necessary for us to convert the MRI data from float64 to float32 format when working with 3-dimensional models. Float64 format can represent numbers much more accurately than 32-bit floats and allow larger numbers to be stored. However, they also take up twice as much memory. Given that there is only so much computational power available to us, we must make sacrifices to precision in order to develop a satisfiable working solution. Despite a loss in data, there is not a noticeable dip in performance.

2-dimensional vs 3-dimensional models

The models designed for 2-dimensional and 3-dimensional MRIs will have no differences, apart from using different dimensionalities for the Convolutional and Max Pooling layers which will be 2-dimensional or 3-dimensional depending on the model developed.

Version-Specific Approach

The development of a Convolutional Neural Network is quick to start off with, but the fine tuning of parameters is the part that will take the longest to perfect. When it comes to improving on a model, Artificial Neural Networks do not provide any suggestions for improving the performance, and a lot of it comes down to ingenuity or speculations that one can make on the results. The idea is that after evaluating the performance of a model, it is up to the engineer to try deducing where to go next. In this project, an abundance of models was tried and tested to achieve the best results, with some models only differing by something small like an adjustment to a few parameters. There were a lot of signs for encouragement with the results achieved, but there were plenty more disappointments. In this section is the design of some of the most impactful models developed through this project. Even if a model

did not provide the expected performance, every outcome was viewed as helpful feedback. The 2-dimensional models were trained on Google Colab, while the 3-dimensional models were trained on the Supercomputer Cluster. It was quick and easy to get started on Google Colab, whereas there were long wait times when training on the cluster. This was because other users would compete for the limited resources when using a GPU partition.

1-5 Hidden Layers

When factoring in what size network model to develop, we must consider the size of the dataset. With only 653 MRIs, we are unable to develop a very deep model. If we developed a model too large, we would run the risk of Overfitting the model. Therefore, we will experiment with a different number of hidden layers in the Convolutional Neural Network that to decide what design to use for the final version for use on the 3-dimensional MRIs.

For the purposes of testing and research, the idea is to build Regression models with a different number of hidden layers and neurons to see what the ideal amount is. The Classification/Regression components will be kept the same. We will develop 5 models, going from 1 hidden layer to 5, and see which gives the best performance.

1. 1 x Convolutional and Max Pooling layer-256 neurons
2. 2 x Convolutional and Max Pooling layer-128 neurons twice
3. 3 x Convolutional and Max Pooling layer-64 neurons, 128 neurons, 128 neurons
4. 4 x Convolutional and Max Pooling layer-64 neurons, 64 neurons, 128 neurons, 128 neurons
5. 5 x Convolutional and Max Pooling layer-32 neurons, 64 neurons, 128 neurons, 128 neurons, 256 neurons

It is expected that from these models, a network with depth of 4 or 5 will produce the best results.

Final Version of Models

We will be developing models for the separate high-resolution and low-resolution MRIs. In the 2-dimensional models, the MRIs are inserted into the network and converted into 2-dimensional format. We can specify which axis (x, y, z) we want to omit. The Feature Extraction is carried out by 5 Convolutional and Max Pooling layer pairings. The number of neurons in each Convolutional layer gradually builds up with the depth of the network. The 1st to 5th Convolutional Layers use 32, 64, 128, 128, and 256 filters respectfully. Each one uses the LeakyReLU activation function. Kernel sizes are set to 3*3

Each Max Pooling layer that succeeds a Convolutional Layer is kept the same, creating a consistent down sampling of the output of the Convolutional Layer that precedes. Each Max Pooling layer uses a 2*2 filter with stride 2. Eventually, the down sampled feature map is

propagated to the flattening layer, which multiplies the filters and channels of the last Max Pooling Layer's output together to create a 1-dimensional vector.

For the Classification component, it is carried out by two fully connected layers. The first layer is made up of 256 neurons nodes and uses the LeakyReLU activation function. The second and last layer is consisted by 7 neurons, representing the 7 different class labels.

For the Regression component, it is carried out by two fully connected layers. The first layer is made up of 256 neurons nodes and uses no set activation function. The second and last layer only has 1 neuron to output a numerical result.

For the 3-dimensional models, the MRIs are inserted into the network in their original 3-dimensional format. Everything is kept the same as the 2-dimensional model apart from a few things. The kernel size of the Convolutional Layer is upped from 3*3 to 3*3*3, and the Max Pooling layer filter is increased to 2*2*2.

The Classification component is once carried out by two fully connected layers. The first layer is made up of 256 neurons nodes and uses the LeakyReLU activation function. The second and last layer is once again only 1 neuron to produce a numerical result.

Implementation

The implementation, training, and evaluation was carried out across two different systems using the same programming language (Python) and an assortment of libraries. Google Colab was used for the 2-dimensional data, while the Supercomputer Cluster was used for the 3-dimensional data. An Object-Oriented Programming approach was taken because it seemed like the easiest way to manipulate the state of a model object. Apart from a few small differences, the implementation is almost the same across the two different model types. This section will be divided into two parts: - one part for general implementation common to every type of model, and another part for code that carries out specific job that is fundamental for the running of a specific model.

General Implementation

Dataset Pre-processing

The first step was importing the Cam-CAN dataset. Importing the images involved setting the path to the directory that harboured the directories for each unique MRI. The directory that held all the high-resolution images did not just have the MRIs; there is two unrelated files that must be ignored, which was not happening in the first attempt. In the Python script, the code will skip over anything that is not a directory, thereby ignoring the unwanted files. Every directory is labelled by their subject ID, and within them was the MRI file. The directory paths are placed into a list, but a problem that arose was how the list of MRI directories was not stored in order. This meant that the script was looking up the MRI directories in a random order, such that the list of directories to iterate through was stored like [dir123, dir54, dir2...]. There may be a time where we need to know something about Subject 1, but without a sorted array we would not know the array index containing the required information. To ensure that the list of directories was in proper order, the Sorted method was used on Listdir so that the list of directories was kept in ascending order [dir1, dir2, dir3…]. From there, we use a script to go through each directory path, extract the necessary file and use NiBabel to load the MRI image from that file. Special care had to be taken to avoid extracting the wrong MRI file. As explained in the “Background” section under “Cam-CAN dataset”, there are different types of MRI files in the dataset, so we must consistently extract the same file format. To accomplish this, Regex is used to give absolute certainty that only the file matching the Regex pattern would be loaded. The image data is either appended to a NumPy array or stored in a local directory ready for image cropping (this step will be explained in a later section). Figure 19 below shows the code that executes these tasks.

```python

mypath = '/scratch/c.sapmt8/camcan/cc700_hires/mri/pipeline/release004/data/aamod_dartel_normmni_00001'
text_file = '../CC700_mt.txt'

class Mris():

    def __init__(self):
        self.mri_arrays = []
        self.age = []
        self.df = pd.read_csv(text_file, sep='\t', engine='python')
        self.dirs = sorted(listdir(mypath))

    def extract_mris(self):
        """
        This extracts the mris individually and puts them into their own arrays
        Skips over item in directory if its a file and not a directory. For use
        with Ryan's code cut_mris.py
        """
        for folder in self.dirs:
            if not isdir(mypath + '/' + folder):
                pass
            else:
                mri_file = self.extract_correct_mri(folder)
                self.extract_mri_data(folder, mri_file)

                mri_age = self.df.loc[self.df.SubCCIDc == str(folder), 'Age'].iloc[0]
                self.multi_classication(mri_age)
                #self.regression(mri_age)
                #self.binary_classification(mri_age)

        np.save('ages.npy', self.age)

    def extract_mri_data(self, folder, mri_file):
        mri_img = nb.load(mypath + '/' + folder + '/structurals/' + mri_file)
        mri_img_data = mri_img.get_fdata()
        np.save('../data/mri_precut/mri_data_' + str(folder) + '.npy', mri_img_data)
        print(mypath + '/' + folder + '/structurals/' + mri_file)

    def extract_correct_mri(self, folder):
        """
        There are 3 different MRIs for each entry. Extracts the MRI
        that matches the regex pattern
        """
        x = listdir(mypath + '/' + str(folder) + '/structurals/')
        regex = "smwc1"
        mri_file = [mri for mri in x if re.search(regex, mri)][0]

        return mri_file
```

Figure 19 Dataset Pre-processing: Extract Cam-CAN MRIs

Figure 19 shows the extraction of the high-resolution MRI image data. Here we have the path to the directory containing all the MRIs directories, and building upon that we can get the path to the specific MRI files. From there, NiBabel can start loading the image data into our local directory ready for cropping. We use our defined Regex pattern, e.g. “smwc1”, to make sure that the script only extracts the image data for file names that match the pattern. Working with NiBabel was not the easiest task because the documentation does not have the greatest explanations or tutorials for the methods part of the library. To some degree, a portion of time was wasted trying to figure out how to use the library.

At the same time as we iteratively go through each unique MRI folder, we take the subject ID of the MRI folder and search it up in the Cam-CAN text file containing all the attributes data for the test subjects called ‘CC700_mt.txt’. The text file is stored as a data frame using Pandas, and through that we can look up the subject ID to extract the value of any chosen attribute. Figure 20 below shows this process for Multiclass classification.

```python
def multi_classication(self, mri_age):
    """
    Go through each folder, look up age in data set via id
    18-30: 0
    30-45: 1
    46-60: 2
    61+: 3
    """
    if mri_age >= 80:
        self.age.append(6)
    elif mri_age >= 71 and mri_age <=79:
        self.age.append(5)
    elif mri_age >= 61 and mri_age <= 70:
        self.age.append(4)
    elif mri_age >= 51 and mri_age <= 60:
        self.age.append(3)
    elif mri_age >= 41 and mri_age <= 50:
        self.age.append(2)
    elif mri_age >= 31 and mri_age <= 40:
        self.age.append(1)
    elif mri_age >= 18 and mri_age <= 30:
        self.age.append(0)
```

Figure 20 Data Pre-processing: Extract the labels

In figure 20, we are extracting the age of every individual for Multiclass classification, using the same dataset grouping defined in "Approach" under "Data Pre-processing". Every time the for loop iterates over a new subject ID, the ID is looked up in the text file we loaded up in Pandas and their data is located. From there, the age of the subject is passed to the *multi_classification* method. A class label is then given depending on the subject's age. In a Regression model, the same steps are followed except instead of appending a class label to the labels array, we append the actual age value.

Cropping the MRIs

This step is only carried out if we are developing a 3-dimensional model on the high-resolution images. The code has generously been shared by fellow student Ryan Codrai (Cardiff University), and uses the re, os, sys, time, NumPy, math, itertools, and skimage libraries. As it has already been explained, the computational power required to run the 3-dimensional high-resolution at in their vanilla state is too demanding on the resources available. By cropping the images, we cut down the dimensionality of each image from 181x217x181 to 96x112x96. The code goes through every high-resolution MRI and gets the aggregated value of the pixels of the MRIs at selected voxels. If the aggregate sum of the voxel values is less than a set tolerance value, the code treats the value as if it was zero. In doing this, we cut down the amount of data and eliminate any Noise coming from the empty spaces of the MRI, but also maintain the same level of resolution. First, the Python script gets the paths of all the pre-cut

MRIs, i.e. path to the MRI files that are at 181*217*181 dimensionality. All the MRIs are held in a local directory for easy access. Figure 19 below shows the code used to execute the cropping. In the figure, we set the directories of where the pre-cut MRIs are, the directory we want the results stored in, the value that an MRI voxel must be greater than to avoid being removed, and whether we resize the images. The code is shown in Figure 21 below.

```
def crop_mris(input_dir, output_dir, tolerance=0.0, resize=False):

    paths = get_paths(input_dir)
    avg = step_average(paths)
    avg[avg == 0.0] = np.nan
    masks = []
    mris_array = []

    for d1, d2 in itertools.combinations(range(avg.ndim), 2):
        masks.append(np.nanmean(avg, axis=(d1, d2)) > tolerance)

    print("Cropping and resizing")
    last_update = start = time.time()

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for i, path in enumerate(paths):
        last_update = progress(i + 1, len(paths), start, last_update)
        mri = np.load(path)
        mri = mri[:, :, masks[0]]
        mri = mri[:, masks[1], :]
        mri = mri[masks[2], :, :]
        mri[mri > 1.0] = 1.0
        mri = mri.astype(np.float32)
        mri = (mri - 0.5) / 0.5

        if resize:
            mri = transform.resize(mri, resize, anti_aliasing=True,  mode='reflect')
        np.save(os.path.join(output_dir, os.path.basename(path)), mri)
        mris_array.append(mri)
```

Figure 21 Code to crop the MRIs,

Once the process is complete, the MRIs produced have less redundant data, and a cropped MRI example is displayed below in Figure 22.

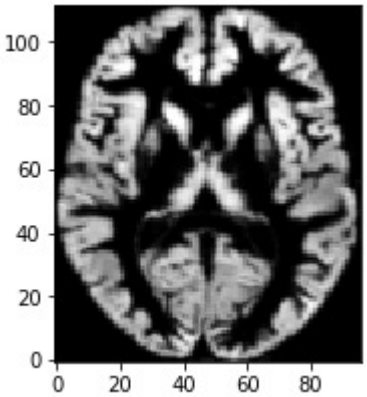


Figure 22 Cropped MRI

Training/Test Split

```
# Random state is used as a seed to the random number generator. Ensuring random numbers are generated in same order
self.train_x,self.valid_x,self.train_label,self.valid_label =
train_test_split(self.mris_array, self.mris_age, test_size=0.2, random_state=13)
```

Figure 23 Training and Test split

Figure 23 above shows how we split the dataset into training and test sets into a ratio of 80/20. The value of *test_size* determines the fraction of the dataset that goes to the test set.

One-hot Vector Encoding

The method for One-hot Vector encoding is carried out using Python, and Keras' one-hot vector encoding *to_categorical.* This method is only used when developing a Classification model because the numerical form of Ages and Memory are suitable to feed into a Neural Network. As it has already been explained in "Background" under "One-hot Vector encoding", a Convolutional Neural Network cannot take in a discrete range of data, e.g. 18-30 years old, class labels 0 to 5. The *data_preprocesses* method takes in a NumPy array containing the class labels for the MRI dataset, and invokes Keras' *to_categorical* method. Figure 24 below converts the *mris_age* array into one-hot vector format.

```
def data_preprocess(self):
    """
    Cleans up data ready to feed into neural network
    """
    # Change age labels into one-hot encoding
    self.train_y_one_hot = to_categorical(self.mris_age)
```

Figure 24 One-hot Vector Encoding of Ages array

Convert 3-dimensional MRIs to 2-dimensional slices

This implementation was carried out using Python. Figure 25 uses no additional libraries and slices the MRIs into 2-dimensional form, such as 61*73 dimensionality.

```
def reshape_2D(self, axis):
    """# Resize 2D MRIs into 61x73 images"""
    i = 0

    while i < len(self.mri_array):
        self.mris_2d.append(self.mri_array[i,:,:,axis])
        i += 1

    self.mris_2d = np.array(self.mris_2d)
```

Figure 25 3-dimensional to 2-dimensional slices for low-resolution MRIs

At the start of the object initialisation, *mris_2-dimensional* is an empty array. Through the *reshape_2-dimensional* method, we pass a valid axis value, e.g. 30 for the z axes, and the method will append every MRI as 61*73*1 size, in the array *mri_array*, to the array *mris_2-dimensional* where the z axis is a constant. This is carried out for all 653 MRIs, and then the array is saved as a NumPy array. Figure 26 below is an example of an uncropped 2-dimensional slice created from the above script.

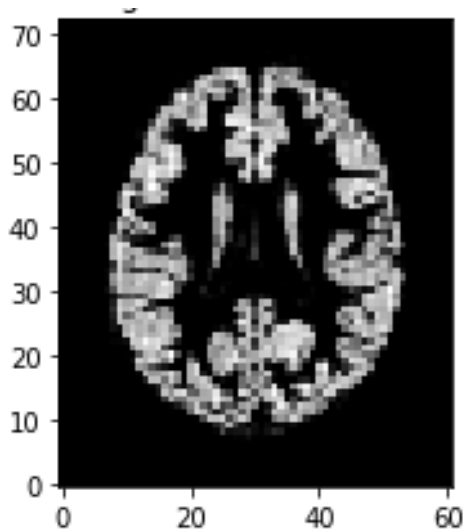


Figure 26 2-dimensional MRI slice with constant z-axis value

Convolutional Neural Network

The implementation of the Convolutional Neural Network was done using Python and various libraries. Initially, the development followed the tutorial for a 2-dimensional Convolutional Neural Network for MNIST digit recognition [31]. Later, with further experience, a model was developed for 3-dimensional MRIs. The Convolutional Neural Network was developed using Keras with a Tensorflow backend. The justification of use for these libraries is explained in "Background" under "Software Libraries used". Here, Keras acts as the front-end, whilst Tensorflow is running in the back-end. With plenty of documentation available and adequate tutorials available online, Keras was the correct choice for developing a Convolutional Neural Network. Several reputable companies use these libraries due to the ease of developing a quick prototype, CNNs support, and the ability to run on a GPU. Such companies include Uber, Google, AirBnB [32], and Facebook.

Despite the differences between a 2-dimensional and 3-dimensional model, there will not be any clear distinctions between implementation of the two models explained here because the implementation is generally the same across both models. What is more distinctive between these models is the computation carried out by the network model.

This section will be split up into three parts: The Convolutional Neural Network model (Feature extraction, Classification or Regression), the building, training, and evaluation of the model, and finally the object constructor method. In each section, the importance and role of each section of code is explained and will highlight how the codebase comes together to create a trained Convolutional Neural Network. For this implementation, a 3-dimensional, high-resolution Convolutional Neural Regression model is being used. Some of the code may seem abruptly cut off. This is to avoid inserting all the code into the section where the lines of code are similar and offer nothing substantially different.

Section 1: The Network Model

The implementation of the model is generally the same across all versions but saw the most changes. These changes include parameters, layers, 2-dimensional or 3-dimensional, and Classification or Regression. The *regression_dropout_model* method defines the architecture of a developed regression model. Despite the name, the functionality of the method is generally the same across different models. In this method we define all the layers of the model, how we fit the data, and train and evaluate network. The method is instantiated by the constructor class during the Main function (discussed later) after the construction of an object and gets passed defined values for batch size and epochs (iterations of training network). Refer to Figure 27 below for this.

```
def regression_dropout_model(self, batch_size, epochs):
```

Figure 27 Artificial Neural Network-Model definition

In the first line we define a Sequential Convolutional Neural Network structure. A Sequential model is a design made up of a linear stack of layers, like stacking LEGO bricks on top of each other. From there, we define our first Convolutional 3-dimensional layer, as every Convolutional Neural Network always starts with a Convolutional layer. This is the first Convolutional Layer in the code, therefore, Keras treats it as the Input layer. This is shown in Figure 28 below.

```
self.mris_dropout_model = Sequential()

self.mris_dropout_model.add(Conv3D(32, kernel_size=(3, 3, 3),padding='same',input_shape=(96, 112, 96,1)))
```

Figure 28 Convolutional Neural Network: Sequential model, Input Layer

As the Input layer, we pass the dimensionality of the MRIs into *input_shape.* In this version, the Input layer performs convolution using 32 filters of size 3*3*3. It uses the LeakyReLU activation

The first Max Pooling layer takes the output of the previous Convolutional Layer and down samples the result by using a filter of 2*2*2 and stride of 2. Figure 29 below shows the code for this

```
self.mris_dropout_model.add(MaxPooling3D((2, 2, 2),strides=2,padding='same'))
```

Figure 29 Convolutional Neural Network: Max Pooling

The implementation of every Convolutional and Max Pooling layer is the same, but the number of filters for certain Convolutional Layers after the first one is greater than the other. Each of the parameters for the succeeding Max Pooling layers are the same as the first one.

In the 4th and 5th layer, we implement Dropout and Batch Normalisation respectively. This is shown in Figure 30 below

```
self.mris_dropout_model.add(Conv3D(128, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))
self.mris_dropout_model.add(Dropout(0.25))

self.mris_dropout_model.add(Conv3D(256, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(BatchNormalization(momentum=0.75))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))
```

Figure 30 Convolutional Neural Network: Dropout and Batch Normalisation

In Figure 30, we set the Dropout ratio i to 0.25 so that 25% of the neurons in that layer are randomly dropped from training. Momentum for the Batch Normalisation is set to 0.75. What is momentum?

As stated, the parameters for the succeeding Convolutional and Max Pooling layers of the first pairing are the same, apart from the number of filters used for the Convolutional Layers. This is displayed in Figure 31 below.

```
self.mris_dropout_model.add(Conv3D(64, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))

self.mris_dropout_model.add(Conv3D(128, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))

self.mris_dropout_model.add(Conv3D(128, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))
self.mris_dropout_model.add(Dropout(0.25))

self.mris_dropout_model.add(Conv3D(256, (3, 3, 3), activation=LeakyReLU(alpha=0.1),padding='same'))
self.mris_dropout_model.add(BatchNormalization(momentum=0.75))
self.mris_dropout_model.add(MaxPooling3D(pool_size=(2, 2, 2),strides=2,padding='same'))
```

Figure 31 Convolutional Neural Network: Remaining Convolution and Max Pooling Layers

As Figure 31 shows, the implementation of the layers are the same as the first pair of Convolution and Max Pooling layers, with the only difference being the number of filters used in the Convolution Layers. The 2nd, 3rd, 4th, and 5th Convolutional Layers use 64, 128, 128, and 256 filters of 3*3*3 respectively.

The final Max Pooling Layer is succeeded by a flattening layer which reshapes the output from the last Max Pooling Layer of output 3*4*3 and channels 256 into a 1-dimensional vector of size (3*4*3*256) = 9216. This is shown in Figure 32 below.

```
self.mris_dropout_model.add(Flatten())
```

Figure 32 Convolutional Neural Network: Flattening layer

The resulting output from the flattening layer is then fed forward to the first Dense layer made up of 256 neurons with a LeakyReLU activation with alpha set to 0.1. The output of the first Dense layer is then passed through to the last Dense layer made of n classes, which in this case is 7 for the 7 different classes in this multiclassification model. The last Dense layer uses the Softmax activation function. This is shown in Figure 33 below.

```
self.mris_dropout_model.add(Dense(256, activation=LeakyReLU(alpha=0.1)))
self.mris_dropout_model.add(Dense(self.nClasses, activation='softmax'))
```

Figure 33 Convolutional Neural Network: Dense Layers

After the training example has been from the input layer to the output layer, each neuron of the output layer will produce a probability that the sample belongs to the respective class. Softmax is required in order to normalise the probabilities to within a range of 0 and 1.

Section 2: The Network Training, Evaluation, and Prediction

Once the model has been constructed, we can start using the model. The three tasks we put our model through are Training, Evaluation, and Prediction. Figure 34 shows where the model training takes place.

```
self.mris_dropout_model.compile(loss=keras.losses.categorical_crossentropy,
                                optimizer=keras.optimizers.Adam(),metrics=['categorical_accuracy'])

self.trained_model = self.mris_dropout_model.fit(self.train_x, self.train_label, batch_size=batch_size,
                                epochs=epochs,verbose=1,validation_data=(self.valid_x, self.valid_label))
```

Figure 34 Convolutional Neural Network: Training

Figure 34 shows the methods uses to calculate and reduce Loss in the network with the *compile()* method. The Loss function is Categorical Cross-Entropy, Adam is the Optimisation function used, and the metrics we are retrieving are for categorical accuracy. The training and test datasets are fitted to the model using the *fit()* method. Batch size and epochs are defined when the object is instantiated, but in this case, they are 16 and 20 respectively.

```
# Model evaluation on the test set
test_eval_dropout = self.mris_dropout_model.evaluate(self.valid_x, self.valid_label, verbose=1)
```

Figure 35 Convolutional Neural Network: Evaluation

In Figure 35 above, the model's performance is evaluated with the test data. What does verbose do? Verbose is set to 1. As a Classification model, this code will produce the Accuracy and Loss of the model, such that it compares the predicted class to the actual class of the training sample. Once it is finished evaluating the predictions to the true values of the sample's labels, it will return the Accuracy and Loss metrics.

```
predicted_classes = self.mris_dropout_model.predict(self.valid_x)
predicted_classes = np.argmax(np.round(predicted_classes), axis=1)

y_true = np.argmax(np.round(self.valid_label), axis=1)
y_pred = predicted_classes

uni_classes = self.classes[unique_labels(y_true, y_pred)]
np.set_printoptions(precision=2)
# Plot non-normalized confusion matrix
self.plot_confusion_matrix(y_true, y_pred, uni_classes, "Convolutional Neural Network",)
plt.savefig('mris_multiclass_accuracy_{}_layer.png'.format((len(self.mris_dropout_model.layers))))
```

Figure 36 Convolutional Neural Network: Prediction

Figure 36 above shows how the predictions are carried out on the model. The model will return a set of probabilities for which class the model thinks a test sample belongs too and store them in *predicted_classes*. The class label with the highest value in the set of probabilities is then taken as the predicted class label for the test sample's MRI. For the Classification model, the model predictions and the true class labels of the dataset are put

into a Confusion Matrix to better visualise the predictions vs the true values of the labels. The generation of a Confusion Matrix is carried out by the *plot_confusion_matrix()* function.

Section 3: The Network Main Function

The Main function starts by setting values for batch size and epochs. Then, we instantiate either a 2-dimensional or 3-dimensional model.

```
if __name__ == "__main__":
    batch_size = 16
    epochs = 20

    dropout_model_3D = MRI_Model()
    dropout_model_3D.classification_data_preprocess_3D()
    dropout_model_3D.classification_dropout_model_3D(batch_size, epochs)

#     dropout_model_2D = MRI_Model()
#     dropout_model_2D.reshape_2D(48)
#     dropout_model_2D.classification_data_preprocess_2D()
#     dropout_model_2D.classification_dropout_model_2D(batch_size, epochs)
```

Figure 37 Convolutional Neural Network: Main function

In Figure 37, we instantiate a 3-dimensional model. We set the batch size to 16 and epochs to 20. When it comes to building the network architecture, training and evaluating it, and then getting predictions, we do this from the main function by calling methods on the instantiated object, which in this case is *dropout_model_3-dimensional.*

```
class MRI_Model():
    def __init__(self):
        self.mris_array = np.load('mri_data.npy')
        self.mris_age = np.load('multiclass_ages.npy') # 185 <= 40, 152 <= 55, 154 <= 70, 162 > 70
        self.classes = np.unique(self.mris_age) # Find the unique numbers from the train labels
        self.nClasses = len(self.classes)
        self.mris_2d = []
```

Figure 38 Convolutional Neural Network: Initialising Object

In Figure 38, when any model is instantiated, the MRI data and MRI labelling is imported. For the Classification model, we extract all the unique class labels and store them in a 1-dimensional vector. *nClasses* stores the number of different unique classes found. *Mris_2-dimensional* is an array used to hold the 2-dimensional slices of the MRI data.

```
def classification_data_preprocess_3D(self):
    """
    Cleans up data ready to feed into neural network
    """
    train_y_one_hot = to_categorical(self.mris_age)
    # Random state is used as a seed to the random number generator. Ensuring random numbers are generated in same order
    self.train_x,self.valid_x,self.train_label,self.valid_label =
    train_test_split(self.mris_array, train_y_one_hot, test_size=0.2, random_state=13)
    # Make sure that the data fed to the network is the correct format
    self.train_x = self.train_x.reshape(-1,96,112,96, 1) # -1 lets numpy figure out a value that satisfies the dimensions
    self.valid_x = self.valid_x.reshape(-1,96,112,96, 1) # -1 lets numpy figure out a value that satisfies the dimensions
```

Figure 39 Convolutional Neural Network: Data Pre-processing

In Figure 39, this is where the data pre-processing happens. Between the 2-dimensional and 3-dimensional data, the only difference is the additional dimension from using 3-

dimensional data, therefore, one of either the x, y, or z value of a 3-dimensional MRI would be omitted from the reshaping. Furthermore, the train and test data split is done. *Test_size* is set to 0.2, therefore, the ratio is 80% training data and 20% test data. Finally, in the Classification model the dataset labels are one-hot vector encoded using the *to_categorical()* method. This converts every element in the *mris_age* array into a vector format.

Results and Evaluation

The aim of this project has been achieved. A Convolutional Neural Network to predict non-linear variables of a given MRI has been developed, and has shown the ability to outperform more basic, linear models such as Support Vector Machines and Naïve Bayes'. The relationship between the non-linear nature of ageing to the features extracted from the raw images has been mapped by CNNs, and results show an ability to either correctly classify MRI images or predict the continuous value of an MRI. Since this is the first time that a Convolutional Neural Network has been developed for use with the Cam-CAN dataset, it is hard to truly understand how well this model has performed compared to the many thousands of models that software engineers have developed when tackling different problems with different datasets. It is therefore worth noting that until another model is developed for this specific project, the results achieved thus far are to be taken as the best at this moment in time.

Evaluation Metrics and Methods used

There is a total of 5 different evaluation metrics used to assess the performance of the Convolutional Neural Network. The 5 chosen metrics are enough to assess the strength of the models for Classification and Regression. For the classification model, Accuracy and Confusion Matrix metrics were used, and MAE and MSE were used for the regression model. Each of the 5 models uses the Loss metric.

Accuracy is a measure of the number of correctly predicted labels of an MRI out of all the MRI samples. The validity of accuracy in a performance evaluation comes into question when there is a data imbalance in the dataset; when a classification problem has a disproportionate number of samples across n-classes in a n-class classification problem, other metrics should be chosen instead of, or at least alongside, Accuracy. For example, in a two-class problem with 100 samples, where 90 classes are labelled 1 and 10 samples are labelled 2, it is possible for a model to achieve 90% Accuracy due to the disproportionate number of samples belonging to the class label 1.

Accuracy alone can misrepresent the strength of a model, hence why a Confusion Matrix is used alongside it. A confusion matrix is an n x n matrix, where n is the total number of classes, for summarising the performance of a classification model. It shows the predicted class for a sample against its actual class. The Confusion Matrix shows specifically where the classifier is mis-classifying samples of the data, allowing us to see where the errors are being made. Below is an example of a Confusion Matrix for a classification model that classifies animals as either a Cat, Dog, or Rabbit. Figure 40 below is an example of a Confusion Matrix to classify animals into one of three different types.

		Actual Class		
		Cat	Dog	Rabbit
Predicted Class	Cat	5	2	0
	Dog	3	3	2
	Rabbit	0	1	11

Figure 40 Confusion Matrix example

This Confusion Matrix is 3x3 due to there being 3 different classes. The columns show the actual class of an instance, whereas the rows show the predicted class of the instance. For example, 2 instances of a Dog were incorrectly predicted to be a Cat. Note that diagonally going from top left to bottom right shows the instances that were correctly classified. If a model produces a Confusion Matrix where all cells but the diagonal ones from top left to bottom right are 0, then that model is 100% accurate.

When it comes to the Cam-CAN dataset, we expect the Confusion Matrix to make some wrong predictions about the samples, but these should be within reason. If an MRI belongs to the class 18-30 years old but is predicted to belong to the class 31-40 years old, this is understandable because we expect similarities between the MRIs of these 2 groups. However, if an MRI belonging to the 18-30 years old class is predicted as being in the 61-70 years old class this raises a cause for concern. This is the power of a Confusion Matrix; we can discover insights that would not be possible when only using the Accuracy metric. It is also due to this acknowledgement that we introduce another metric alongside Accuracy, Accuracy ±1, where Accuracy ±1 is the number of correctly predicted classes and the class predictions that are 1 higher or lower than the actual value.

Mean Absolute Error (MAE) measures the average magnitude of the errors (actual value – predicted value) in a set of predictions. It is the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. The aim is to get MAE as close to 0 as possible.

Mean Square Error measures the average of the squares of the error [29]. Its value tells you how close the set of points of a regression model are to the regression line. This is done by taking the distance of the points from the regression line and squaring them. By squaring these values, it keeps the MSE positive and gives greater. Values closer to 0 are always better. An MSE of zero, where an estimator predicts input with perfect accuracy, is desirable but incredibly hard to achieve. For better understanding of the MSE and MAE metrics, look through the “Background” section under “Loss Functions”.

The final metric used is Loss. The Loss function was explained during the “Background” section under “Optimal Performance”. As we are using a type of Gradient Descent Optimisation algorithm (Adam), we expect the model’s loss to quickly decrease, but then steadily stagnate into the later iterations of training, being unable to decrease loss by a great deal.

Early Development Results

The results in this section reflect the assumptions made early in the project and justify certain design choices made. The Cam-CAN dataset was used for each evaluation, on a 2-dimensional Convolutional model for the low-resolution (61*73) images. These results are not an indicator of the project's ultimate results but aim to reinforce the justification for using Convolutional Neural Networks for the aims and objective of the project, hence why we only evaluate the performance on the 2-dimensional low-resolution MRI images.

Effect of number of Layers

It was discussed in the "Approach" under "Version Specific Approach" that we wanted to see the effect of a different number of hidden layers for a network architecture. For this, 5 Regression models are developed, with the number of hidden layers for the Feature Extraction increasing per model, up to a maximum of 5 hidden layers. The Regression component of the models are kept the same. The experiment was carried out on the high-resolution MRIs, using a 2-dimensional Regressive model. When evaluating the performance of the models, we are comparing the Mean Squared Error, Mean Absolute Error, and Loss.

$Convolutional\ Neural\ Network - Cam - CAN\ effect\ of\ Layers$			
	MSE	MAE	$Loss$
$1\ layer$	63.68	7.85	63.89
$2\ layers$	1.54	1	1.54
$3\ layers$	0.74	0.69	0.74
$4\ layers$	0.74	0.67	0.74
$5\ layers$	0.69	0.67	0.69

Table 3 Convolutional Neural Network: Effect of Layer Count

The results in Table 3 above show that, as expected, a network architecture with several layers in proportion to the size of the dataset will perform than one with too few layers. This could be down to the smaller architectures not having enough time to learn the features from the images before being fed to the output layer, where the backpropagation begins again.

Effects of Dropout

As discussed in "Optimal Performance" in the "Background", we implement Dropout to reduce Overfitting when fitting data in the network. The testing was carried out on the Cam-CAN dataset, fitted to a Multi-class Classification model. The MRIs had class labels ranging from 0 to 3, therefore, it was a 4-class model. The network architecture was three Convolutional and Max Pooling layer pairings, succeeded by two Dense layers for the classification. The Dropout followed one of the Convolutional layers at a factor $p = 0.2$, where p is the fraction of neurons randomly dropped in a layer. The batch size was 128, and there were 20 epochs (iterations of training). The Dropout is set low because of the small dataset size, relative to the network architecture.

$Convolutional\ Neural\ Network - Effect\ of\ Dropout$			
	$Dropout\ factor$	$Test\ Accuracy$	$Test\ Loss$
$Dropout$	0.2	0.50%	1.14
$No\ Dropout$	n/a	0.47%	1.15

Table 4 Network Performance with/out Dropout

As expected by Table 4 above, the network performs better when Dropout is in place. Figure 41 below shows the training and validation Accuracy and Loss for both models, where the x axis is epochs, and y axis is either Accuracy or Loss.

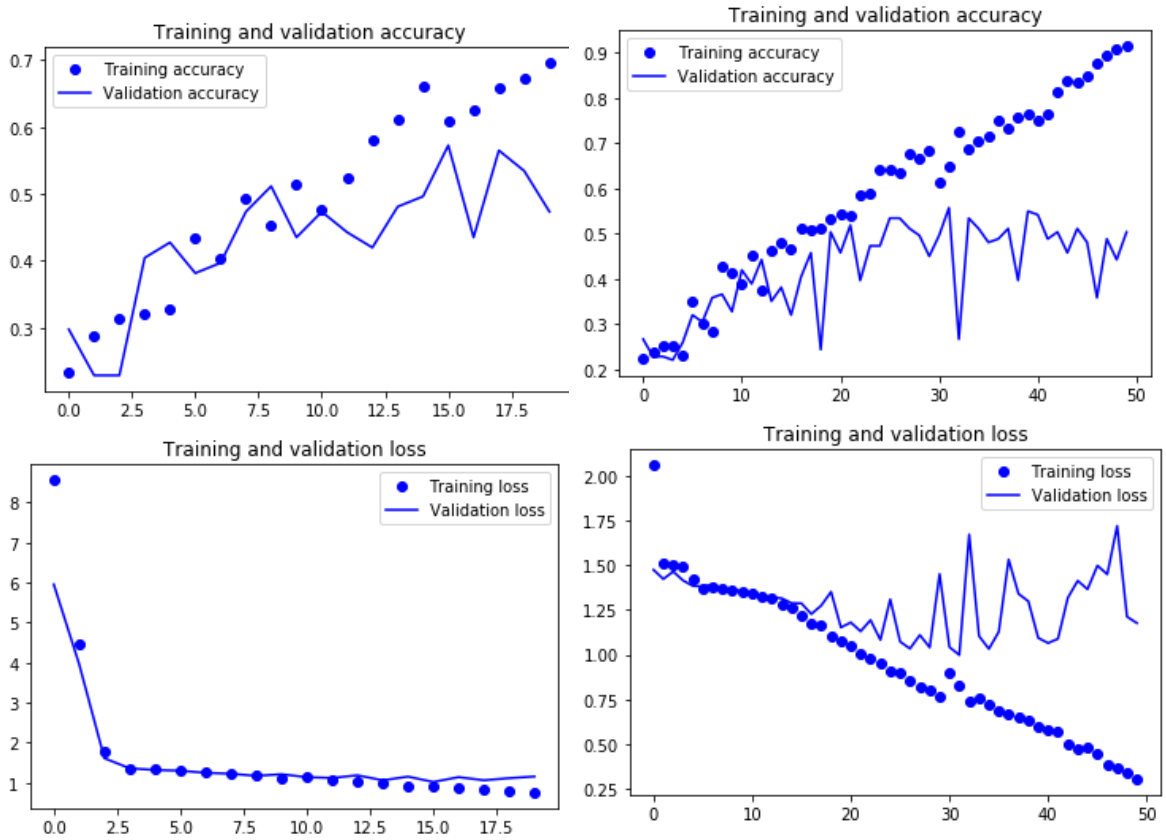


Figure 41 (Left) No Dropout, (Right) Dropout

The graphs show for both models that both models eventually stops learning, but most importantly, that the Dropout model had better performance. This can be attributed to how the model had less to train on than a network without Dropout, therefore, the weaker model must have started to pick up on Noise and fluctuations in the dataset, detriments that the model with Dropout cut down on. However, the performance suggests that there is no great benefit in using Dropout in the model. This could be down to the size of the dataset and the network, as has already been highlighted. Nonetheless, it could be beneficial to have one Dropout layer where the ratio of dropped neurons is set, thus giving enough features for the network to learn from.

Comparing Convolutional Neural Networks to Basic Models

Results show the Convolutional Neural Network models performed better than the basic models on the Classification models. The Convolutional Neural Network and basic models used the same training and test dataset to ensure no ambiguity regarding their performance on the dataset.

The comparison of results between the different models is given by their accuracy. The dataset is randomly split into training and test datasets, but each model uses the same data.

$Approach$	$Accuracy$
$Convolutional\ Neural\ Network$	0.51
$Support\ Vector\ Machines$	0.45
$Random\ Forest\ Classifcation$	0.47
$Naive\ Bayes$	0.46

Table 5 Convolutional Neural Network vs Basic Models

As we would have expected, the results in Table 5 show a Convolutional Neural Network performed better than the more basic models.

Key Results

The results in this section are indications of how well the Convolutional Neural Network has performed to meet the aims and objectives of the project, as set out from the beginning in the "Introduction" section. In this section, eight different models are compared, with the Regression and Classification models having four different models each for the different image dimensionalities and resolution. The Classification model is a 7-class model, with MRIs labelled a number between 0 and 6, with 0 being the youngest age group and 6 being the oldest age group. The group labelling is explained earlier in "Approach" under "Data flows of the System".

These different models are summed up in Figure 42 below.

$Convolutional\ Neural\ Network\ models$							
$Regression$				$Classification$			
$Low - Resolution$		$High - resolution$		$Low - Resolution$		$High - resolution$	
$2D$	$3D$	$2D$	$3D$	$2D$	$3D$	$2D$	$3D$

Figure 42 Breakdown of models developed

The model architecture for the performance evaluation in this section is the one described as the final version in the "Approach" section, i.e. a Convolutional Neural Network with 5 pairs of Convolutional and Max Pooling layers, and 2 Dense layers. The results for each of the models are given below in Figure 43. The batch size was set to 16, and the epochs was set to 20

Convolutional Neural Network Models Performance						
Model Type	*Dimensionality*	*MRI Resolution*	*Accuracy*	*MSE*	*MAE*	*Loss*
Classification	2D	*High*	44%	~	~	1.97
Classification	2D	*Low*	32%	~	~	2.81
Classification	3D	*High*	44%	~	~	1.97
Classification	3D	*Low*	34%	~	~	2.09
Regression	2D	*High*	~	0.69	0.67	0.69
Regression	2D	*Low*	~	0.96	0.81	0.96
Regression	3D	*High*	~	0.73	0.67	0.73
Regression	3D	*Low*	~	0.74	0.7	0.74

Figure 43 Compiled Results of CNN models

Note on Classification Results

There are 7 different class labels in the Classification model (0-6). The probability of correctly predicting the class label for a random MRI is $\frac{1}{7} = 0.14$. Any model with an accuracy greater than 0.14 is performing better than a baseline.

2D Classification High-Resolution

This model evaluated the test data with **44% accuracy and 1.97 Loss**. The accuracy is much better than the expected accuracy of 0.14% The predictions are presented in the following Confusion Matrix in Figure 44.

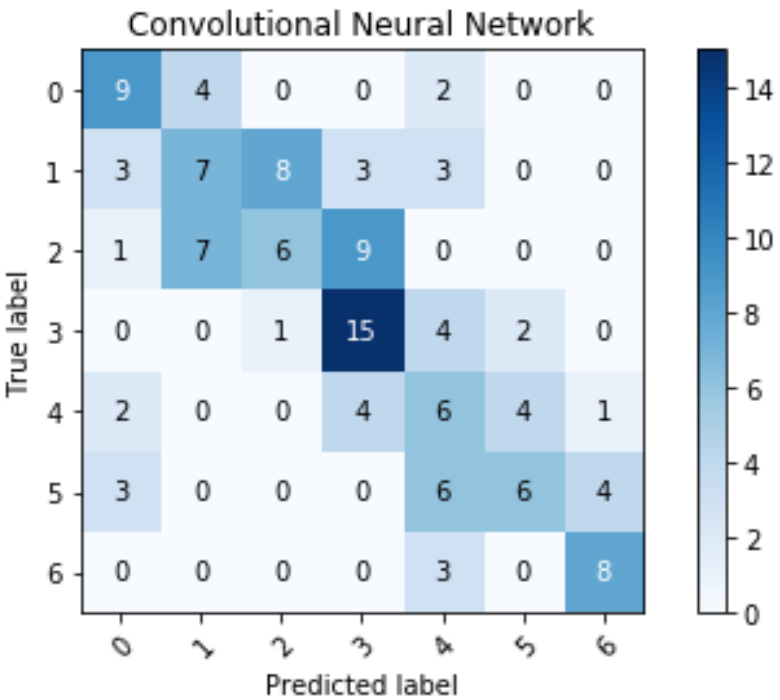


Figure 44 Result: Confusion Matrix Classification 2D High-Resolution

The Confusion Matrix shows that most predictions were correct or were classes ±1 from the actual label. 20 samples were completely incorrect (were not ±1 from the actual class).

$True\ Label\ Value$	$Accuracy$	$Accuracy\ \pm 1$
0	60%	86.6%
1	29%	75%
2	26%	95%
3	68.1%	90.9%
4	37.5%	82.35%
5	31.5%	84.21%
6	72.72%	72%
$Average$	46.4%	83.72%

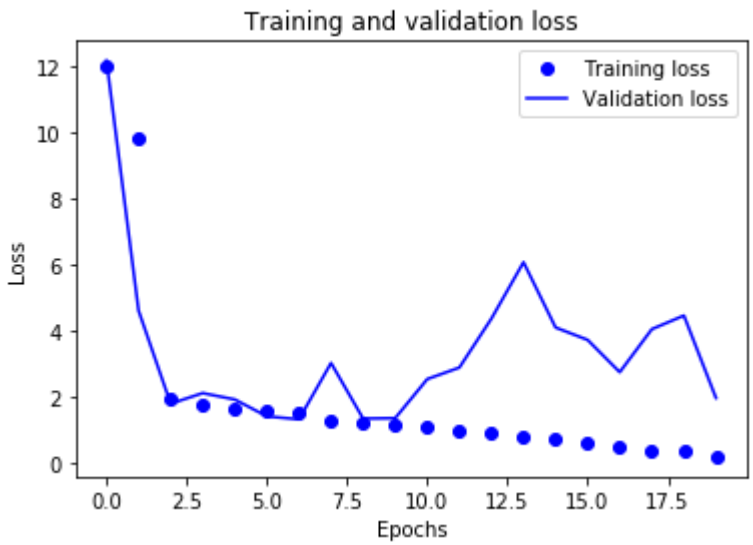


Figure 45 Result: 2D Classification High-res Loss graph

Analysis

The accuracy of the model is satisfiable. The model performed better than the baseline of 14%, achieving 44% accuracy. However, the results could be better. There appears to be signs that the model can capture the relationship between the features and the variables. On the other hand, the Loss graph in Figure 45 is a concern, as it starts off well, but halfway through the gradient sharply goes up, and slowly comes down. This indicates to me that the model is still learning or is starting to Overfit the data. Overall, the results of this model are okay, but we would have to look at ways of improving the model.

2D Classification Low-resolution

This model evaluated the test data with **32% accuracy and 2.81 Loss**. The predictions are presented in the following Confusion Matrix in Figure 46.

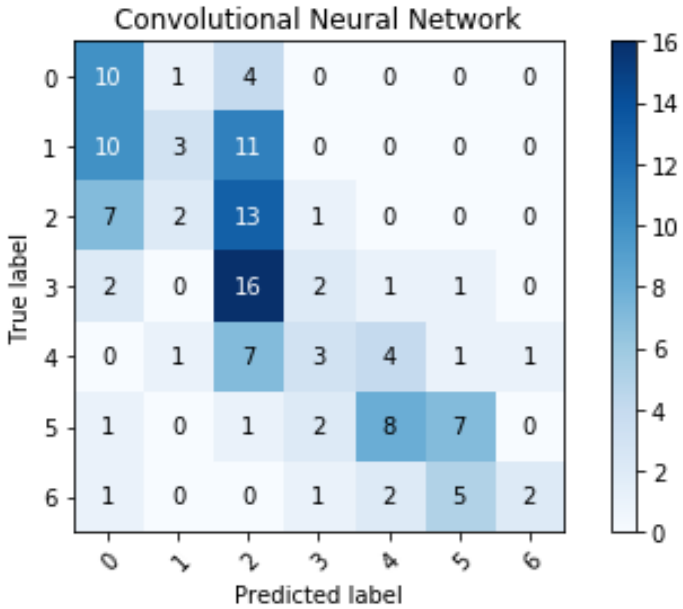


Figure 46 Result: 2D Classification Low-res Confusion Matrix

Figure 44 shows that there were a lot of unsuccessful predictions, along with less successful ±1 predictions. There were 31 completely incorrect predictions

$True\ Label\ Value$	$Accuracy$	$Accuracy\ \pm 1$
0	60%	73%
1	12.5%	100%
2	57%	69%
3	9%	86.3%
4	23%	47%
5	36%	89.4%
6	18%	63%
$Average$	30.8%	75.4%

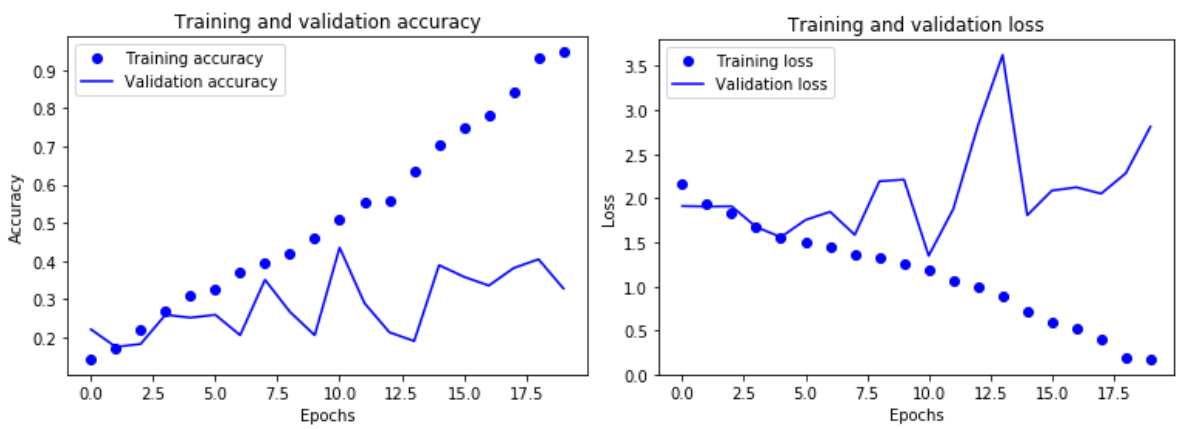


Figure 47 Results: 2D Classification Low-res Accuracy and Loss

Analysis

The results of this model are not as good as the ones from the previous model that was fed high-resolution images. This shows that a model fed the high-resolution images performs better than one without them. This comes as no surprise, given that the high-resolution images offer more parameters for the model to train from. However, it would not be right in saying that either of the models have a good standard of performance, after observing the Loss graph. There appears to be no sign that the graph is slowly reducing Loss to 0. Arguably, the model stops learn around the 10th epoch.

3D Classification High-resolution

This model evaluated the test data with **44% accuracy and 1.97 Loss.** Miraculously, this is the same set of results as the 2-dimensional High-resolution Classification model. Figure 46 below is the Confusion Matrix for this model.

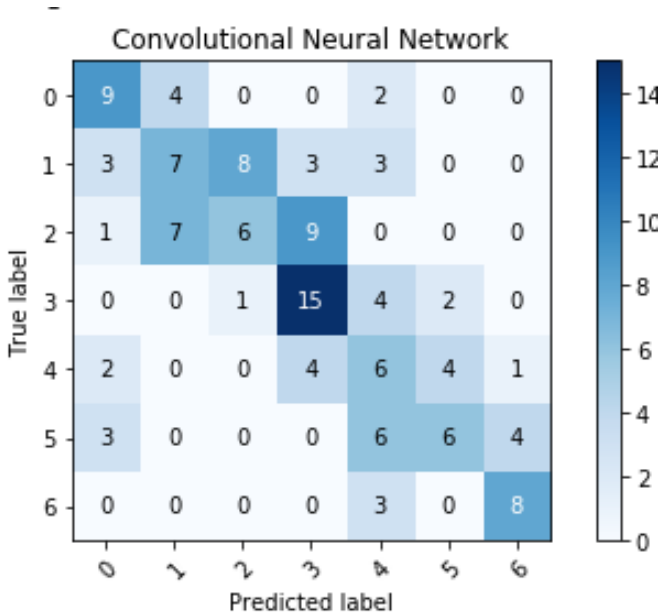


Figure 48 Results: 3D Classification High-res Confusion Matrix

$True\ Label\ Value$	$Accuracy$	$Accuracy\ \pm 1$
0	60%	86%
1	29%	75%
2	26%	95%
3	68%	90%
4	35.2%	82.3%
5	31.5%	84.2%
6	72%	72%
$Average$	45.96%	83.5%

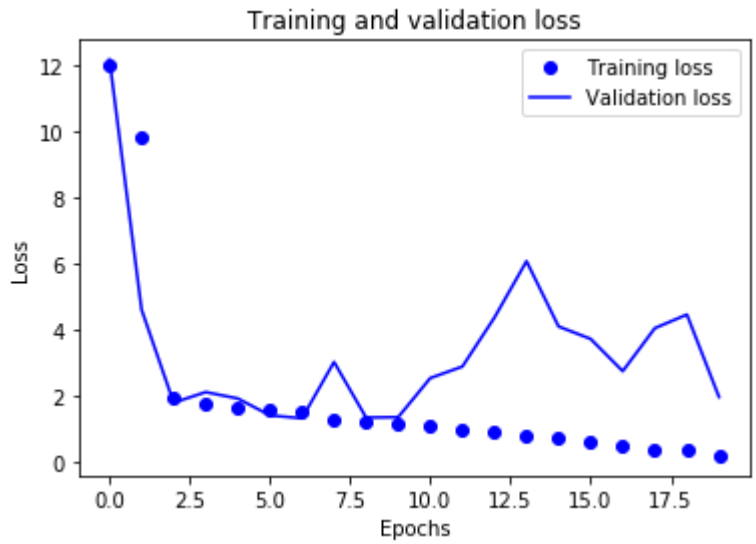


Figure 49 Results: 3D Classification High-res Loss

Analysis

This model is arguably the best performing Classification model out of the 4 developed. It possibly shows an ability to map the MRIs to the correct class labels. This is because, despite the Loss and gradient spiking up near 12 epochs, the last few epochs show a direction towards a Loss of 0. However, more iterations would be needed in order to see where the slope is actually going.

3D Classification Low-resolution

This model achieved **34% accuracy** and **2.09 loss**. The Confusion Matrix for this model is displayed below in Figure

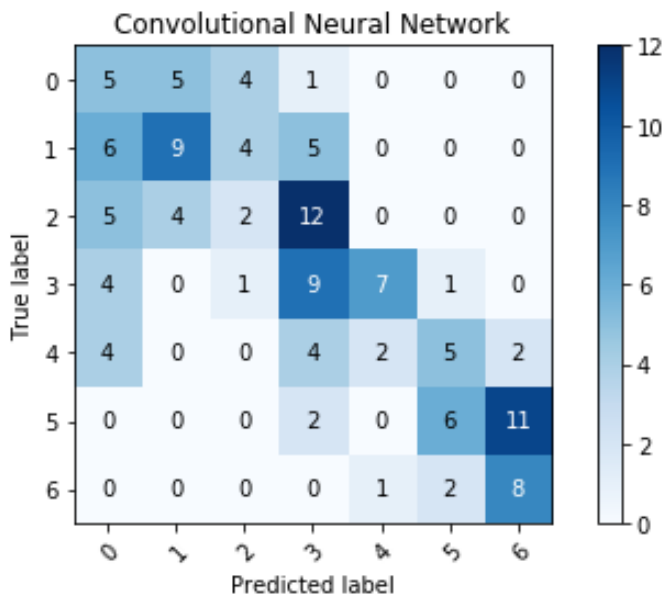


Figure 50 Result: 3D Classification Low-res Confusion Matrix

$True\ Label\ Value$	$Accuracy$	$Accuracy\ \pm 1$
0	33%	66%
1	37.5%	79.1%
2	8.7%	78.2%
3	40.9%	77.2%
4	11.8%	65%
5	31.5%	89.4%
6	72%	90%
$Average$	33.6%	77.8%

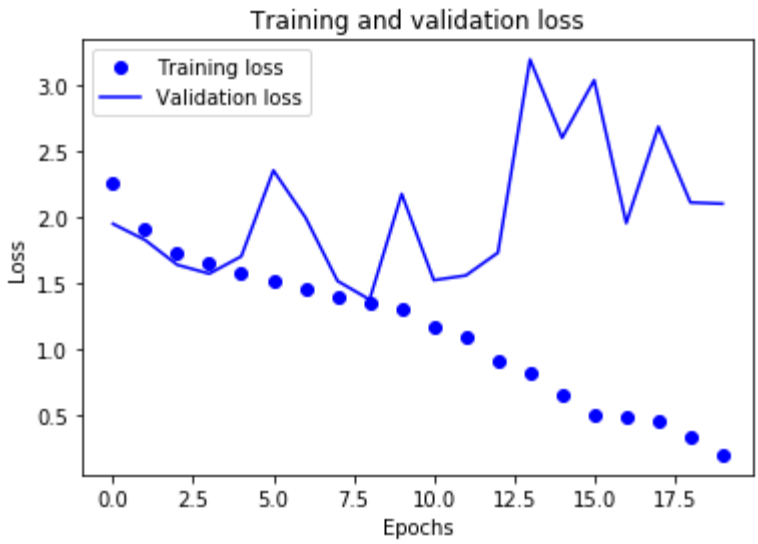


Figure 51 Results: 3D Classification Low-res Loss

Analysis

As to be expected, the low-resolution model performed roughly the same as the 2-dimensional model, highlighting the factor that the number of trainable parameters play in the strength of a model. In this model, this is no reason to believe that this model has learned to map MRIs to variables because of the trajectory and gradient of the Loss graph not decreasing towards 0.

2D Regression High-resolution

This model achieved an **MSE of 0.69** and **MAE of 0.67**.

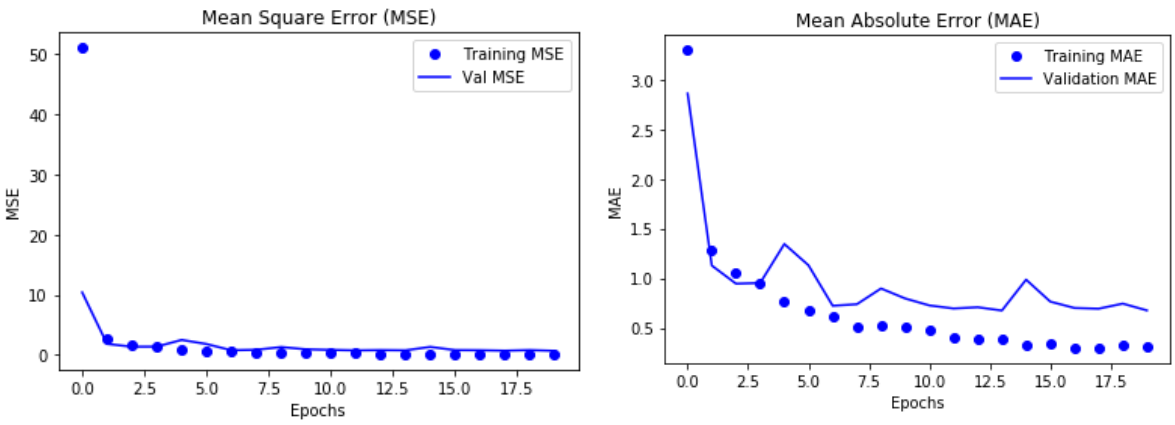


Figure 52 Result: 2D Regression High-res MSE & MAE

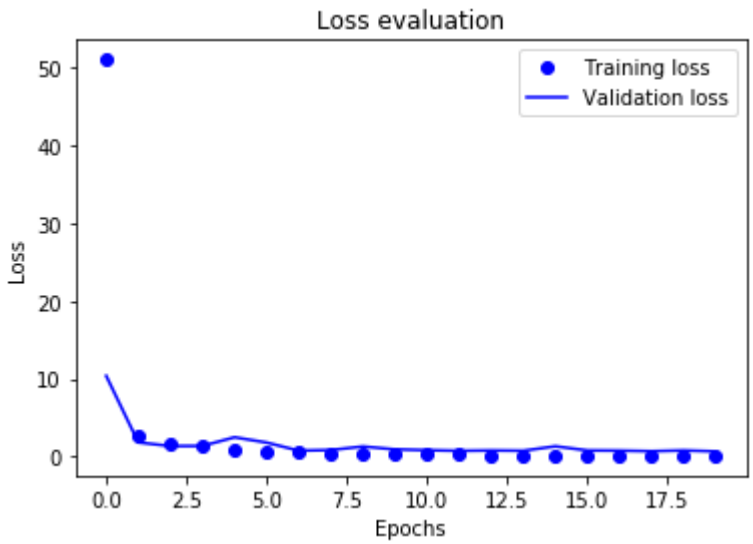


Figure 53 Result: 2D Regression High-res Loss

MSE	MAE	$Loss$
0.69	0.67	0.69

Analysis

This was a great result. Each graph shows how the metrics are making huge steps towards reaching 0. On the other hand, the gradient for Validation MAE is less stable than the other metrics and shows signs of instability. This is possibly because of the interaction between MAE and the Gradient Descent algorithm, where MAE is a more Linear gradient, and produces a smaller range of values compared to MSE. The results of this model lead me to believe that it can map the relationships between MRIs and non-linear variables.

2D Regression Low-resolution

This model achieved an **MSE of 0.96** and **MAE of 0.81,** and a **Loss of 0.96**. The performance is not as good when the model uses the low-resolution images. But this is expected because there are more parameters for the network to train with the high-resolution images.

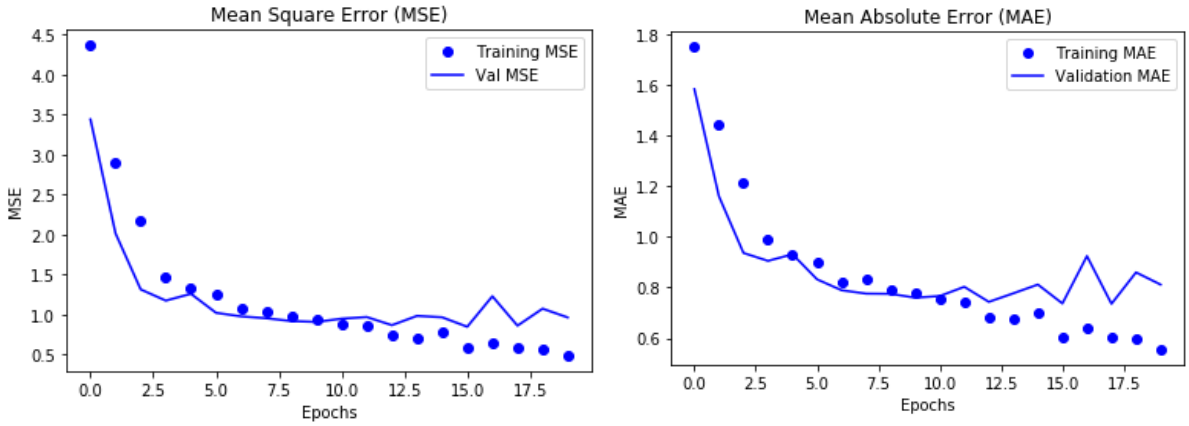


Figure 54 Result: 2D Regression Low-res MSE & MAE

MSE	MAE	$Loss$
0.96	0.81	0.96

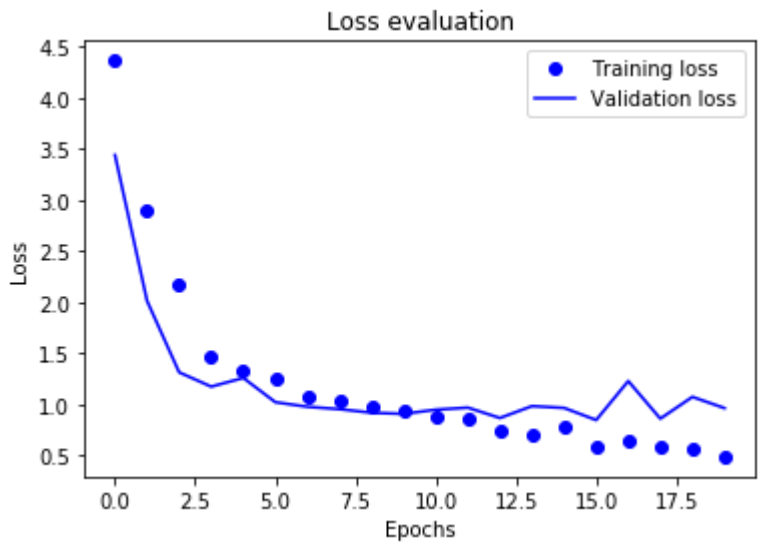


Figure 55 Result: 2D Regression Low-res Loss

Analysis

The results are reasonable. The metrics returned are good, but the graph projections raise concerns. Towards the later epochs the model seems less stable, and it is possible that the model was at its best around the 10th epoch. This could be because the model started to Overfit, or because the weights and bias that the model randomly initialized with put the model into this scenario. On the other hand, we could bring into question whether the Adam optimisation function has performed well enough, given its nature to adjust the training samples weights and bias, and the adapt the learning rate per sample. The results of this model lead me to believe that it can map the relationships between MRIs and non-linear variables, but not as well as models with the high-resolution images.

3D Regression High-resolution

This model achieved an **MSE of 0.73** and **MAE of 0.67** and a **Loss of 0.73**.

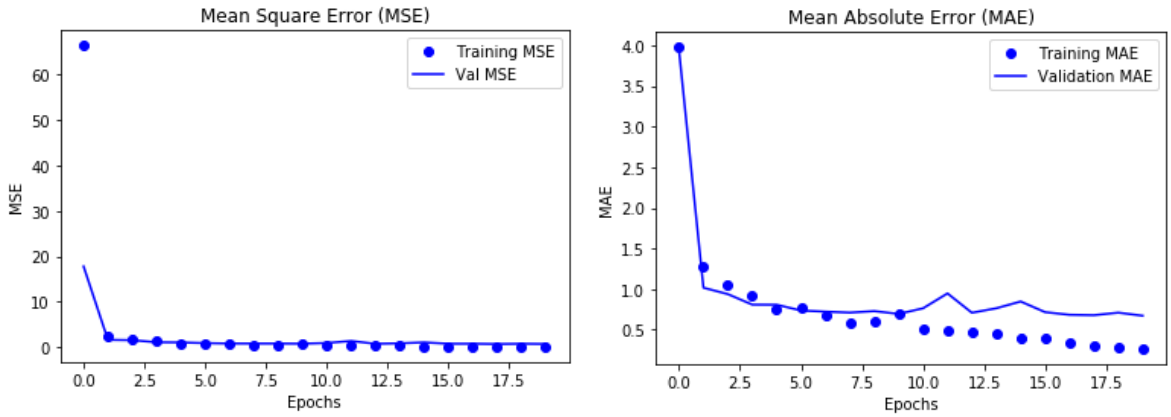


Figure 56 Results: 3D Regression High-res MSE and MAE

MSE	MAE	$Loss$
0.73	0.67	0.73

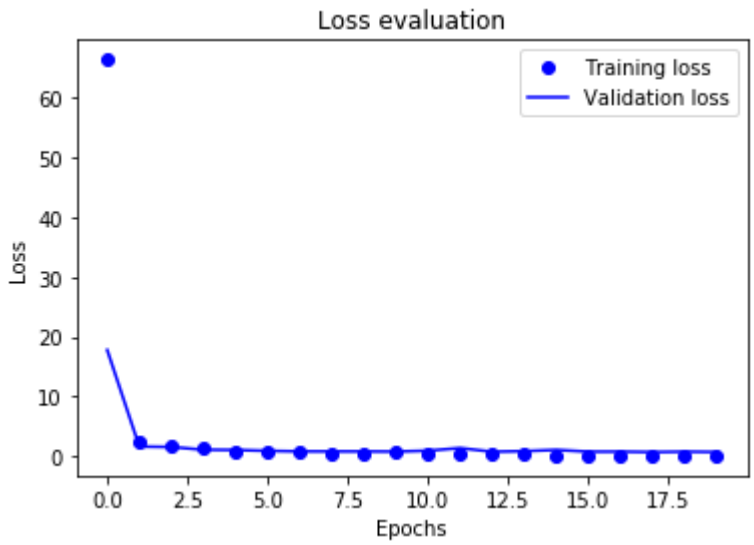


Figure 57 Results: 3D Regression High-res Loss

Analysis

The results show that the Regression model can map MRIs to the outputs with a good degree of certainty. Each graph show progression towards reaching 0.

3D Regression Low-resolution

This model achieved an MSE of 0.74 and MAE of 0.70 and a Loss of 0.74.

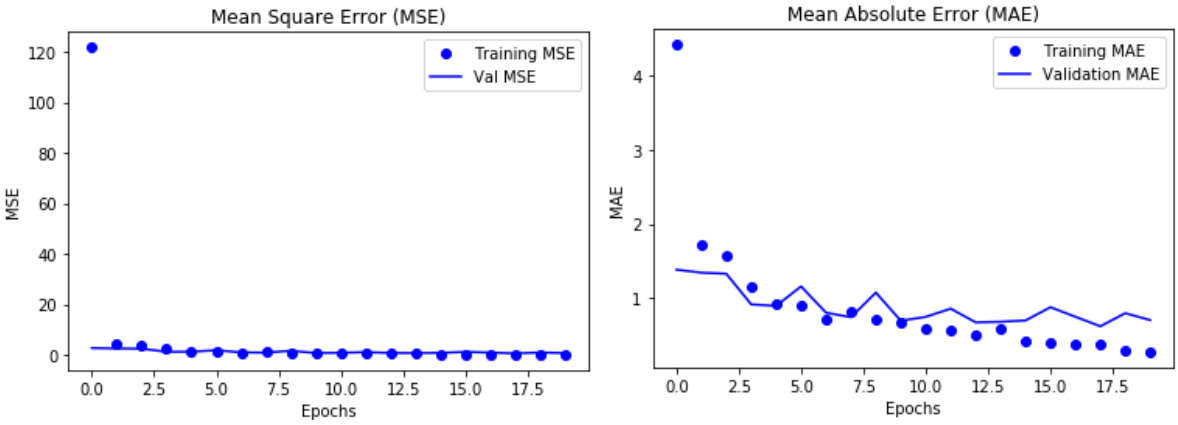


Figure 58 Result: 3D Regression Low-resolution MSE and MAE

MSE	MAE	$Loss$
0.74	0.70	0.74

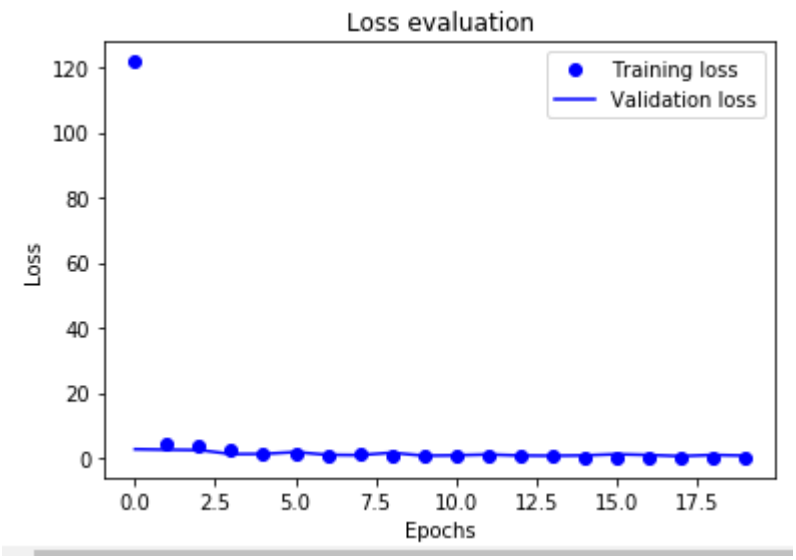


Figure 59 Result: 3D Regression Low-res Loss

Analysis

The results are strong. The model appears to map the relationship between an output and the MRIs extremely well. The metrics from this model are some of the best achieved, and is proven by the illustrations of the MSE, MAE, and Loss graphs, as the values of the 3 metrics steadily go closer towards 0. Once again, and not for the first time, MAE shows signs of uncertainty. The model must be strong because the MSE and Loss are edge further towards 0. The results of this model lead me to believe that it can map the relationships between MRIs and non-linear variables

Conclusion of Results

The analysis of the results achieved show that the Regression models performed better than the Classification models at 3-dimension and 2-dimension. This is due to the illustrations shown by the graphs between the models, where in Regression the model shows signs of learning the relationships between MRIs and variables, whereas the Classification models do not. The Regression models have consistently shown strong performances, whereas the Classification models have been just above average. However, the results achieved for the Classification models would be better had there been a better distribution of classes amongst the training and test set.

There appears to be a pattern that networks fed the same resolution MRIs will result in getting roughly the same evaluation metrics results. This was most notable when training the Classification models on the high-resolution and low-resolution MRIs.

We have shown that the depth of an Artificial Neural Network does matter, but that there is more emphasis on their needing to be a minimum number of layers to correctly train the model. As always, the number of layers chosen should be in proportion to the size of the dataset, hence why a layer depth of 4 or 5 would have been adequate to complete this task.

Future Work

The aims of the project have been met, but there are more avenues to be explored that could improve upon the results achieved in this project.

Architecture Optimisation

If there was more time available, some effort into finding optimal parameter values for things like kernel size, learning rate, and padding could have happened. The key decisions such as the model architecture, the optimisers and loss functions used etc. were choices made based on experimentation to find the best results. But the values for certain parameters were chosen only because they were the widely accepted default value. An attempt at training the model with something like different kernel sizes could have made smaller steps to a better performing model.

Future work should also look at using a different type of Gradient Optimisation function. In this project, we use the Adam optimiser, a variant of Stochastic Gradient Descent. The next approach should experiment with a different one, such as Mini-batch gradient descent.

Better Classification Training/Test Split

Unfortunately, there was not enough time to properly split the Multiclassification labels into a fair distribution. This means that the training and test split distribution shown in "Approach" under "Data flows of the system" was not the one used. Due to this, the representation of classes in the training and test datasets are random, resulting in unsavoury results because of a lack of fair representation. In the future, a function, or process, will have to be done to ensure a fair distribution amongst classes in the training and test split to get the distribution shown in Figure 56 below.

Training Set				Test Set		
Class Labels	Samples	Representation		Class Labels	Samples	Representation
0	64	12.28%		0	16	12.12%
1	84	12.28%		1	21	15.90%
2	80	15.35%		2	21	15.90%
3	80	15.35%		3	21	15.90%
4	83	15.93%		4	21	15.90%
5	80	15.35%		5	20	15.15%
6	50	9.59%		6	12	9.09%

Figure 60 Future Training/Test split distribution

Capsule Networks

Despite the success of Convolutional Neural Networks, it is believed that one of their biggest pitfalls is an inability to map the spatial relationship of objects, such that rotating an image in

an alignment that the model has never seen before could throw off the model's predictions, e.g. rotating a face in a dataset 47 degrees clockwise. This is because Convolutional Neural Networks focus on the features present on an image, and do not focus on the spatial relationship between features, like the relationship between the different parts of a face. Capsule Networks [33] are a modern neural network architecture based on Convolutional Neural Networks that work similarly, but make sure to map the spatial relationship between features of an image, such that the model performs better by recognising a face based not only on the features of a nose and, but their position on the face of an image as well. On the MNIST dataset, the model achieved a 0.25% classification error rate, and a 5.2% classification error rate on the MultiMNIST dataset.

Different Architecture Model

The network architecture designed in this project is inspired by classic architectures that stack layers on top of each other sequentially. But in recent times, modern architectures have given way to new and innovative ways to construct Convolutional Layers. Allowing for more efficient learning and better performance overall. One such modern architecture is the Inception model [34]. The Inception model is a modern architecture first introduced in 2014 for the ImageNet competition, where it took first place. The general model is displayed bellowed in Figure 27.

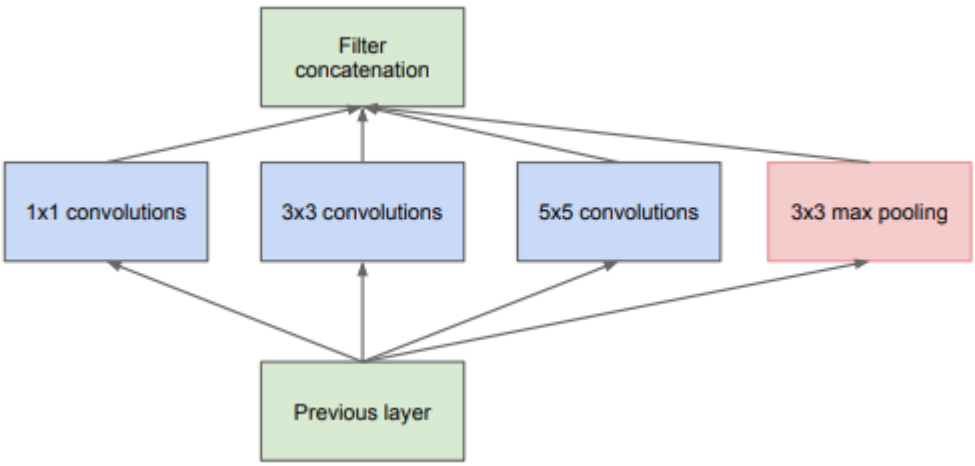


Figure 61 Inception Model [34]

This model type is based around a basic unit called an "Inception cell", where we perform a series of Convolutions of different scales (3*3, 5*5 etc.), and aggregate the results together. There was a follow up paper that improved upon the first version of Inception. In the follow up [35], it cuts down on computation cost by factorising larger convolutions into a set of smaller ones, where a filter such as 5*5 can be more computationally efficiently represented by two 3*3 filters. This is because a 2-dimensional filter of 5*5*c, where c is the number of channels, costs $25c$, whereas two 3*3*c filters would cost $18c$

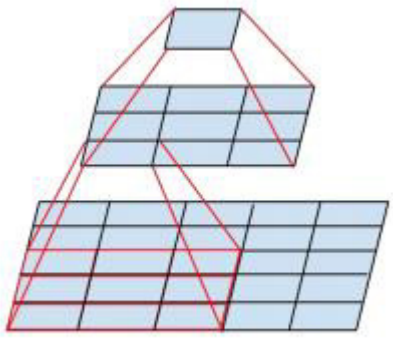


*Figure 62 Mini-network replacing 5*5 Convolution [35]*

Mapping the Relationship between MRIs and other variables

There was no time in this project to map the MRIs to any other variables. At the start, the aim was to map MRIs to age and cognitive variables such as memory. It is worth noting that a variable such as memory can be harder to map due to the factors that influence it, such as lifestyle choices, age, and IQ.

Conclusion

As this project reaches its conclusion, it is worth critically evaluating whether the aims of the project were yet. The aim of the project was to develop a Convolutional Neural Network that could predict the age and cognitive variables of an MRI. It was never the intention to aim for a specific result or baseline, but rather a proof of concept that CNNs could be used for this type of problem. This aim has been achieved, as a model has been created for 2-dimensional and 3-dimensional MRIs for both a regression and classifier model. The results of the project suggest that, in time, future Convolutional Neural Networks will be developed to a greater degree of success than the ones developed in this project, as would be expected with the introduction of newfound concepts, ideas, and software engineering tools. Should the future work outlined in this project be followed, there is no reason to believe that a greater level of success with CNNs and MRI images cannot be achieved. For the benefit of the scientific community, this work should be a starting point for someone tackling the same, or a similar, problem to the one described in this project.

The results show that a Convolutional Neural Network can be developed to predict variables, after adequate training, based off an MRI. The comparison of results also show that the performance of a Convolutional Neural Network is greater than that of a more basic model. This is evident when we compare the results of a Convolutional Neural Network to Decision Trees, Naïve Bayes, and Support Vector Machines.

The drawback with Convolutional Neural Networks, as with many Machine Learning methods, is being unable to see where a model struggles to learn from an image. The evaluation metrics used do well to assess the performance of a network but offer limited help in explicitly stating what went wrong and how to improve performance.

Not having the desired computational power to develop an optimal model will be the biggest 'what if' of this project. Despite the tremendous power of the supercomputer cluster, a stronger model could have been developed if more resources were available. The high-res 3-dimensional images and 3-dimensional Convolutional Neural Network demanded a great deal of power to run. Running the 3-dimensional model on the supercomputer cluster required cropping the high-res images to a smaller size and reducing the complexity of the 3-dimensional model. It would have been interesting to see what results would have been achieved with stronger resources.

My results have shown that the network model has the ability map an MRI to a predicted value, but I believe this is not only down to the capabilities of CNNs and the design of the model. Given the nature of neuroimage data with regards to their spatial invariance, is it possible to develop a baseline for CNNs when used with MRIS? If this is the case, then where

do we go next with CNNs for neuroimages? Even though Convolutional Neural Network architectures are not cookie cutter (one size fits all), further investigation could be spent into seeing how to optimise neural networks for use with neuroimaging data. However, this is going on a tangent away from the aim of this project.

Arguably the biggest challenge with working with neuroimages is that the images are so like each it is harder for a Convolutional Neural Network to distinguish between images in the dataset and map the relationship. Common applications of Convolutional Neural Networks are to identify images in a dataset that are slightly different in some way, like identifying a dog in different sections of an image. But MRIs tend to be spatially invariant, in that a set of MRIs are displayed in the same way at a 2-dimensional or 3-dimensional level. The most distinctive things between a pair of MRIs could be the smallest details/pixels not easy to detect by the human eye. Due to this, it is possible that a Convolutional Neural Network is more likely to pick up on noise when training.

Reflection

The final year project has proven to be my most challenging academic challenge to date. The trials, discipline, and dedication to complete this project have been challenging, and it was the toughest test of my soft and technical skills to date.

As someone who has worked with an Agile methodology in the previous workplace, it made sense for me to try and adopt similar ideas to this project. Many people may prefer to look through large amounts of documentation before experimenting and tinkering, but I have found that I need to start developing as soon as possible and learn as I go along. This allows me to start developing earlier but obtain a better grasp of the content I am following.

I took interest in this project because I have an interest in Machine Learning, and the applicability of Machine Learning to medical data made it even more enticing. I had some experience in creating basic Machine Learning models in Python prior to this project, but nothing remotely like the tools and software used to complete this task. This was the first-time hearing about Artificial Neural Networks but I feel like I am starting to understand this field of AI to a satisfactory standard. I have constructed many different models during this project, and I feel like I have only touched the surface of Artificial Neural Networks. There is plenty more to be discovered, and this could serve well if I decide to pursue a Data Science career.

Furthermore, the biggest challenge of this project has been my use of time management and task prioritisation. I am responsible for the outcome of this project, therefore the management and overseeing of the work carried out falls on me alone. One of the biggest problems when training my 3-dimensional models on the supercomputer was how my jobs could spend many hours in the queue before executing, and sometimes they took over a day! The use of my time has been the hardest part of this project because the time goes very quickly, especially when things do not always go to plan, like waiting for a Supercomputer job to finish. It is in circumstances such as these that I need to be able to take a step back, analyse the situation, re-evaluate my priorities, and create contingency plans in the event of a massive problem. Despite the long waiting times, it forced me to become patient and look for other work to get on with. Having said that, I failed to implement a function to properly manage the training and test dataset split, thus producing results for the multiclass Classifier that do not fairly reflect the capabilities and performance of the model developed. In the future, better care needs to be taken to make sure that certain tasks are placed as top priority, as they could be more important to the goal than other tasks.

Finally, resource management has been challenging. As it has been explained in this project, the most complex of Neural Network models require the most computational power. For a while, I was able to manage on Google Colab for developing my 2-dimensional models. Eventually, I had to move from Google Colab to the supercomputer cluster to run the 3-

dimensional models. I had absolutely no experience with using the cluster, so understanding how to use its resources was like learning a new challenge but was made more difficult when trying to manage my time effectively because it became a bottleneck. It is hurdles like this that forced me to re-evaluate my priorities and time so that I could give myself a realistic expectation of what work could be accomplished.

References

[36] [Online]. Available: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.

[18] A. Budhiraja, “Medium,” 15 December 2016. [Online]. Available: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[30] Abadi et al, “TensorFlow: A System for Large-Scale,” p. 1, 2016.

[24] B. Stecanella, “Monkey Learn,” 25 May 2017. [Online]. Available: https://monkeylearn.com/blog/practical-explanation-naive-bayes-classifier/.

[2] Beck et al., “Agile Manifesto,” 2001. [Online]. Available: http://agilemanifesto.org/.

[27] Brett et al, “NiBabel,” 1 April 2019. [Online]. Available: https://nipy.org/nibabel/.

[22] D. S. Gupta, “Analytics Vidhya,” 29 June 2017. [Online]. Available: https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/.

[4] Deep convolutional neural networks for brain image analysis on magnetic resonance imaging: a review, 2018.

[37] E. Allibhai, “Towards Data Science,” 17 October 2018. [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5.

[8] F. Chollet, Deep Learning with Python, 2017.

[17] G. H. A. K. I. S. R. S. Nitish Srivastava, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* 2014.

[28] Google, “Google,” n/a n/a n/a. [Online]. Available: https://research.google.com/colaboratory/faq.html.

[1] I. S. G. E. H. Alex Krizhevsky, “ImageNet Classification with Deep Convolutional Neural Networks,” 2012.

[16] J. L. B. Diederik P. Kingma, Adam: A Method for Stochastic Optimisation, 2015.

[20] J. Laufer, “Jenslaufer.com,” 20 March 2019. [Online]. Available: https://jenslaufer.com/machine/learning/image-aesthetics-quantification-with-a-convolutional-neural-network.html.

[6] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” Switzerland, 2014.

[10] K. Goltsman, “Introduction to Artificial Neural Networks,” 5 September 2017. [Online]. Available: https://datascience.foundation/sciencewhitepaper/introduction-to-artificial-neural-networks-(anns).

[7] N. Donges, “Towards Data Science,” 26 February 2018. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5.

[25] N. Liberman, “Towards Data Science,” 27 January 2017. [Online]. Available: https://towardsdatascience.com/decision-trees-and-random-forests-df0c3123f991.

[21] n/a, “Computer Science Wiki,” 27 February 2018. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.

[23] n/a, “Data School,” 25 March 2014. [Online]. Available: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/.

[26] Pedregosa et al, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* 2011.

[32] Q. Z. P. S. Shijing Yao, “Medium,” 2 May 2018. [Online]. Available: https://medium.com/airbnb-engineering/categorizing-listing-photos-at-airbnb-f9483f3ab7e3.

[15] S. Bhattarai, “A Tech Blog,” 22 June 2018. [Online]. Available: https://saugatbhattarai.com.np/what-is-gradient-descent-in-machine-learning/.

[29] S. Glen, Supercomputing Wales, 2 November 2013. [Online]. Available: https://www.statisticshowto.datasciencecentral.com/mean-squared-error/.

[31] S. Mahapatra, “Towards Data Science,” 2018 May 2019. [Online]. Available: https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a.

[19] S. Mahapatra, “Towards Data Science,” 21 May 2018. [Online]. Available: https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a.

[14] S. P. Marvin Minsky, https://becominghuman.ai/what-is-an-artificial-neuron-8b2e421ce42e, 1969.

[33] S. S. N. F. Geoffrey E. Hinton, Dynamic Routing Between Capsules, 2017.

[13] S. Saxena, “Artificial Neuron Networks(Basics) | Introduction to Neural Networks,” Becoming Human: Artificial Intellgence Magazine , 2017.

[12] S. Sharma, “Towards Data Science,” 6 September 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[34] Szegedy et al., Going Deeper with Convolutions, 2015.

[38] T. e. al., The Cambridge Centre for Ageing and Neuroscience (Cam-CAN) datarepository: Structural and functional MRI, MEG, and cognitive data from a cross-sectional adult lifespan sample, Elsevier Inc., 2018.

[11] T. Edu, 19 December 2019. [Online]. Available: https://becominghuman.ai/what-is-an-artificial-neuron-8b2e421ce42e.

[5] T. Mitchell, 1 March 1997. [Online]. Available: http://profsite.um.ac.ir/~monsefi/machine-learning/pdf/Machine-Learning-Tom-Mitchell.pdf.

[9] T. Srivastava, “Analytics Vidgya,” 20 October 2014. [Online]. Available: https://www.analyticsvidhya.com/blog/2014/10/ann-work-simplified/.

[3] Taylor et al., “The Cambridge Centre for Ageing and Neuroscience (Cam-CAN) data repository: Structural and functional MRI, MEG, and cognitive data from a cross-secitonal adult lifespan sample,” *Elsevier,* 12 September 2015.

[35] V. V. S. I. J. S. Christian Szegedy, Rethinking the Inception Architecture for Computer Vision, 2015.