Machine Learning Explainability in MRI Brain Tumour Classification

By

Jack Lennon Cannings

URN: 6497204

A dissertation submitted in partial fulfilment of the requirements for the award of:

BACHELOR OF SCIENCE IN COMPUTING AND INFORMATION TECHNOLOGY

May 2021

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Tang, H Lilian Dr

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Jack Cannings
May 2021

Abstract

Today, machine learning models are ubiquitous throughout most industries. They are used for customer retention, fraud detection, self-driving vehicles and for an endless number of other reasons. The models and algorithms have fantastic potential to change the way we live our lives, and more specifically for this project, how we can extend our lives.

Using machine learning in the healthcare industry isn't a new train of thought, the first use of an Artificial Neural Network, a type of machine learning algorithm, being used to provide clinical diagnosis dates back to the early 90s.

Before we can responsibly create models that are used for clinical diagnosis, we need to overcome the hurdle that is 'explainability'. Many machine learning models, especially those which fall under the class of artificial neural networks, have an inherent problem. Due to the deep and complex architecture of these models, it's very difficult to understand what is happening in the inner workings, and how exactly that model is coming to its conclusions. This is what's is commonly referred to as a 'black-box'.

This project aims to investigate and implement a method of model explainability, in an attempt to open the a 'black-box' model which has been trained to classify Magnetic Resonance Images of patient's brains in order diagnose various types of tumours. If we can find a reliable method of explaining a prediction of a model, one day practitioners everywhere may be able to utilise machine learning models to treat patients more reliably and efficiently.

Acknowledgements

I would like to firstly thank my wonderful partner Sophie, who has supported me through this process and has always been there to lend an ear when needed, as well as my family and friends for their constant support and encouragement.

Secondly, I would like to thank the fantastic academics at the University of Surrey, their dedication and commitment to their disciplines has been inspiriting.

Lastly, I would like to thank Dr Lilian Tang, my project supervisor. Lilian has provided so much encouragement, ideas, inspiration, and advice to me over the course of this dissertation, I would not have been able to close out this project without her guidance.

Table of Contents

Table of Figures

Table of Equations

Table of Tables

Key Terms and Acronyms Glossary

Adam	A model optimisation algorithm to update weights in an artificial neural network.
ANN	Artificial Neural Network
Backpropagation	Backward propagation of errors calculates the gradient of the loss function with respect to each weight in a network. A key method in updating model weights.
Batch Size	The number of inputs calculated by a machine learning model before weights are updated
Classification	The act of a machine learning model applying a label to a piece of data. Classifying an image as a particular class.
CNN	Convolutional Neural Network
Convergence	A model's loss approaching the global or local minimum
Convolutional Layer	A core layer in a Convolutional Neural Network which applies convolutions to an input image and outputs feature maps.
Data Augmentation	The act of taking a sample of data and applying augmentations in order to artificially produce a new data point.
Dense Layer	A layer in a neural network in which the neurons receive inputs from all neurons in its previous layer. Also known as a 'fully connected' layer.
DNN	Deep Neural Network
Epochs	The number of times a machine learning model will train over an entire dataset
Explainability	The ability to provide interpretable evidence pertaining to the results of a machine learning model
Feature Map	The image which is produced by a Convolutional Layer after the have been applied.
Filter	See 'Kernel'
Full Connected Layer	See 'Dense Layer'
GPU	Graphics Processing Unit. The workhorse which enables machine learning models to train quickly.
Hidden Layer	The layers in an artificial neural network which sit in between the input and output layers.
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
Kernel	The window in a convolutional layer which slides over an image, calculating the dot product and outputting the results to a feature map. The kernel values are updated though back propagation.
LIME	Locally Interpretable Model-Agnostic Explainability
Loss	Loss is the number which indicates how bad a machine model's prediction was for single example.
Midline Shift	Extra pressure in the brain caused by a tumour which has distorted tissue surrounding the tumour site and pushed the brain off centre.

Model Optimisation	The act of updating the weights in the network based on the backpropagation of the loss.
MRI	Magnetic Resonance Image
Neuron	A computational unit which has one or more weighted inputs. Also known as 'Perceptron' or 'Node'
Node	See 'Neuron'
Perceptron	See 'Neuron'
Perturbation	A variation of an image with random superpixels hidden.
Plane (MRI)	An angle from which an MRI scan has been captured. Axial, Sagittal and Coronal.
Pooling Layer	A core layer used to progressively reduce the spatial size of the feature maps produced by Convolutional Layers in order to reduce the parameters in the network while retaining important features.
ReLU	Rectified Non-Linear Unit. A type of Activation Function
SGD	Stochastic Gradient Descent – A model optimisation algorithm to update weights in an artificial neural network.
Sigmoid	An activation function used in layers of a neural network
SoftMax	Output generalisation function which returns an array of probabilities adding up to 1 for a multi-class problem.
Stride	The number of pixels in an image a kernel will slide over after each convolution is applied.
Superpixel	A contiguous group of pixels.

1. Introduction

1.1. Problem Background

Information published on the Cancer Research UK website [1] paints a morbid picture of how Brain Tumour cases in the UK are on the rise. Between 2015 and 2017, there were over 12,000 new Brain Tumour cases reported, which is a 39 percent increase in incident rates when compared to the early 90s and is projected to rise by another 6 percent by 2035.

As these incidents increase, the health service is going to need an efficient service to supplement diagnosis. The traditional method of trained radiologists and tissue sample analysis require a lot of training and man-hours to be effective, and even then, they are subject to influences which can affect their accuracy, such as fatigue and inexperience. I believe that machine learning models can provide this supplementary service and can be implemented alongside medical professionals to increase the efficiency and accuracy of diagnosis.

Machine learning models, and more specifically Artificial Neural Networks, are sophisticated models which have been created to resemble the functions of the human brain and are used in a vast range of classification and regression problems in many industries. While it's commonly considered that one of the most pressing and important issues in machine learning is medical diagnosis, is it in no way new. As a matter of fact, first use of an artificial neural network being used to provide second opinions in clinical diagnoses dates to the early 90s. [3]

Convolutional Neural Networks are machine learning models which has proven to be extremely accurate in image recognition tasks and training them has become increasing efficient in the last decade thanks strides in GPU computing [4] and breakthrough studies such as *Krizhevsky et al* [44]. For these reasons I believe Convolutional Neural Networks are an ideal candidate for brain tumour image classification, but before we can see these models being used within the health service, there is a hurdle to overcome, and that hurdle is called *Explainability.* Due to the nature of these models and their complexity, it is extremely difficult to understand their results, which is why they are commonly referred to as 'black-box' models. I believe that once reliable methods of opening the black-box and explaining the classifications of machine learning models is developed, trust will increase and use of them will widen.

1.2. Project Description

The aim of this project is to evaluate and test explainability methods and hopefully provide a means to opening a 'black-box' machine learning model which has been trained to classify tumours in MRI brain scans. Explainability isn't a mature area of research and because of this there aren't any universally accepted tools to provide explainability. That being said, there is currently a lot of effort being channelled into it, and as a result many new explainability methods are being tested. This project will focus on recent method called *Local Interpretable Model-Agnostic Explainability*, or LIME for short, which will be applied to one or more machine learning models called *Convolutional Neural Networks*. By training these networks and applying LIME to them, hopefully I will be able to peer into the black-box and gain and understanding into what influences the models decisions, and evaluate whether the models can be trusted, or if they are inherently flawed by some learned bias in the data.

1.3. Project Aims and Objectives

The following section contains the definition of the aims and objectives in this project. This content of this section will be referred back to in the reflection and evaluation of the project to measure whether the aims and objectives have been met, and to what degree of success.

Acquire a Suitable Dataset

The foundation of any strong and robust machine learning model is a good dataset. This objective is to procure a dataset relevant to the project's definition. The dataset will ideally contain MRI images with multiple tumour classes and healthy brains present, with a balanced distribution between the classes.

Data Exploration and Pre-Processing

Upon data acquisition, I will analyse the data to decide which data pre-processing will be appropriate. Examples may include converting RGB Images to Greyscale to reduced dimensionality, resizing the dataset as well as applying data augmentation techniques to artificially expand the size of the dataset and balance it if required.

Research, Train and Evaluate Machine Learning Models

There are many different types of machine learning models out there so it's important to understand which are the most appropriate for this problem. Once I understand which models will be most appropriate for this project, I will implement them, train them, and then evaluate them to measure how successful they are classifying brain scans. Different measures should be used for the evaluation in order have a deeper understanding of the models.

Implement and Evaluate a Machine Learning Model Explainability Method

Once that the models have been trained and I'm satisfied with the results, I can implement and apply the chosen explainability method to them. As mentioned before this project will focus on the use of *Local Interpretable Model-Agnostic Explainability* to achieve this goal. Once the explainability method has been implemented, it can be applied to the trained models in order to provide some explanations for the classifications. These explanations will hopefully reveal the most important features of the brain scans for each of the classes in the dataset. I can then decide whether I think the model can be trusted or if it is inherently flawed.

1.4. Report Structure

Section 1 – Introduction

The introduction section provides context to the reader regarding the overall goal of this project. It contains some background information related to brain tumour diagnosis in the UK, a description of this project, and how it is going to attempt to address the issue, as well as the high-level objectives and structure of the report.

Section 2 – Literature Review

In this section the reader will find reviews of literature pertaining to brain tumour statistics, diagnosis, MRI scans and types of tumours with the purpose to provide context and understanding of the diagnosis problem and distinguishing features of the tumours so the machine learning models explanations can be interpreted and understood. Technical literature reviews are also included, which look for studies with similarity to that of this project in order to learn what methods, models, and technologies I can leverage and apply in order to provide explanations of my own models.

Section 3 – Dataset Analysis & Preparation

This section will cover all sources of data used in this project and the preparations steps which were used in order to ready the data for model training. An overview of the hardware and software which will be required to make this possible will be included, as well as an implementation section which will cover how I implemented the steps to prepare the data.

Section 4 – Machine Learning Models for Brain Scan Classification

A comprehensive description of the machine learning models used in this project will be provided in this section. It will detail the distinctive difference between them, what parameters have been used for their training and the technical details regarding the implementation of the models.

Section 5 – Evaluation of the Machine Learning Models.

In this part I will provide scrutiny of the models used in the project. The training loss and accuracies will be reviewed through plotted graphs, as well as testing on held-out data in both a binary (tumour vs no tumour) context as well as a multi-class context in the form of confusion matrixes and Precision, Recall & F-1 scores.

Section 6 – Implementing Machine Learning Model Explainability.

This section will demonstrate the implementation of *Local Interpretable Model-Agnostic Explainability*, used to provide interpretation of the machine learning models used in this project. It will be a step-by-step review of how it was implemented with accompanying visualisations and snippets of code for the curious reader.

Section 7 – Evaluation of the Machine Learning Explainability Method.

This will be a section evaluating the implementation of *Local Interpretable Model-Agnostic Explainability,* which will be applied to all machine learning models used in this project, using a variety of images from each class to demonstrate thoroughness and to truly attempt to understand the models and what features of these brain scans influence the classifications the most.

Section 8 – Reflecting On Aims, Project Conclusion and Future Work

In this penultimate section I will return to the high-level objectives defined in the *Project aims & objectives* section, in order to evaluate whether the aims and have been met, and to what degree of success. I will also discuss time management, as well as potential for future works and my overall conclusions of the project and whether I am satisfied with what has been achieved.

Section 9 – Statement of Ethics

This section will contain legal and ethical considerations which may apply to this project and what has been done to meet them.

2. Literature Review

2.1. Introduction

The purpose of this section is to explore the existing literature out there to find inspiration for the direction and technical implementation of this project. I will be looking at existing studies which tackle issues such as medical condition classification using deep learning, explainability of machine learning models, as well as information pertaining to cancer diagnosis methodology and statistics. These reviews will also influence which machine learning models and what parameters will be used in this project.

2.2. Brain Tumour Statistics

I believe it best to start this literature review with some background information and statistics on Brain Tumours in the United Kingdom to emphasise the need for reliable methods of early detection and diagnosis in the healthcare industry. Much of the information in this section has been sourced from the official Brain Tumour information pages found on the Cancer Research UK Website [1]. The wrangled data which is presented on the website has been provided by many government-backed statistics authorities from around the UK. For England, the data was provided by the Office of National Statistics (ONS) [10], from Scotland the Information Services Division (ISD) [11], Wales the Welsh Cancer Intelligence and Surveillance Unit (WCISU) [12], and finally for Norther Ireland, the data is from the Northern Ireland Cancer Registry [13]. The statistics below include incidents of Brain, Central Nervous System (CNS) and other Intercranial Tumours.

It was mentioned in the introduction of this report that there were over 12,000 new cases of brain tumours in the UK between 2015 and 2017. Brain tumour cases make up around three percent of all cancer cases, so are uncommon when compared to the liked of lung, breast, bowel, and prostate cancers which make up over half of all new cancer incidents in the UK [7]. However, the increase in cases of brain tumours in the UK represents a growth of 39% since the early 90s, and this figure is only expected to grow. Another point of significance is the probability of brain tumour patients developing new brain metastasis as a progression of their original cancer diagnosis. A metastasis is a spread of the cancer cells which have originated from a different organ in the body. A study by Davis et al [15] found that 6% of patients with newly diagnosed cases of cancer in the United States would expect to develop brain metastasis as a result of their original cancer diagnosis. The most common sites for these original cancers being lung / bronchus and breast cancers.

An additional morbid truth about brain cancers is that the risk factors are few. Other than old age – which is a result of damage to the DNA accumulating over time and causing a cell mutation in the brain, the only other two identified risk factors are exposure to radiation, which is responsible for 1% of cases, and obesity which is responsible for 2%. This means that only 3% of all brain, CNS, and other intercranial tumours are classified as preventable. This statistic I believe reinforces the need for reliable detection and diagnosis of brain tumours, opposed to the minimisation of risk factors, as 97% of cases may be a surprise, and not indicated by any risk factors. This is not to say that minimising risk factors would not be beneficial, only that an overwhelming majority of these incidents would still occur.

2.3. Brain Tumour Diagnosis

Brain tumour diagnosis is a sophisticated area of oncology and the machines used to take pictures of are extremely complex and the images they produce can vary a lot. If I am going to be able to interpret the outputs from an explainability method a foundational knowledge I am going to require a foundational knowledge of the different types of MRI images, tumour types and the anatomical regions in which they appear. Therefore, in this section I will detail the results of literature reviews and research into the method of tumour identification through MRI scans and the distinguishing features of the tumours which will be present in the dataset of this project.

2.3.1. MRI Scans

Magnetic Resonance Imaging (MRI) in an extremely common type of test in neurology which provides high quality images of the brain from three different planes – The Axial, Sagittal and the Coronal, all shown in *Figure 1*. It is due to the high quality of these images produced by the Magnetic Resonance which makes them favourable over Computerised Tomography (CT – A different type of brain scan which produced similar images) for brain tumour recognition. Much of the information in this section has been sourced from [33].

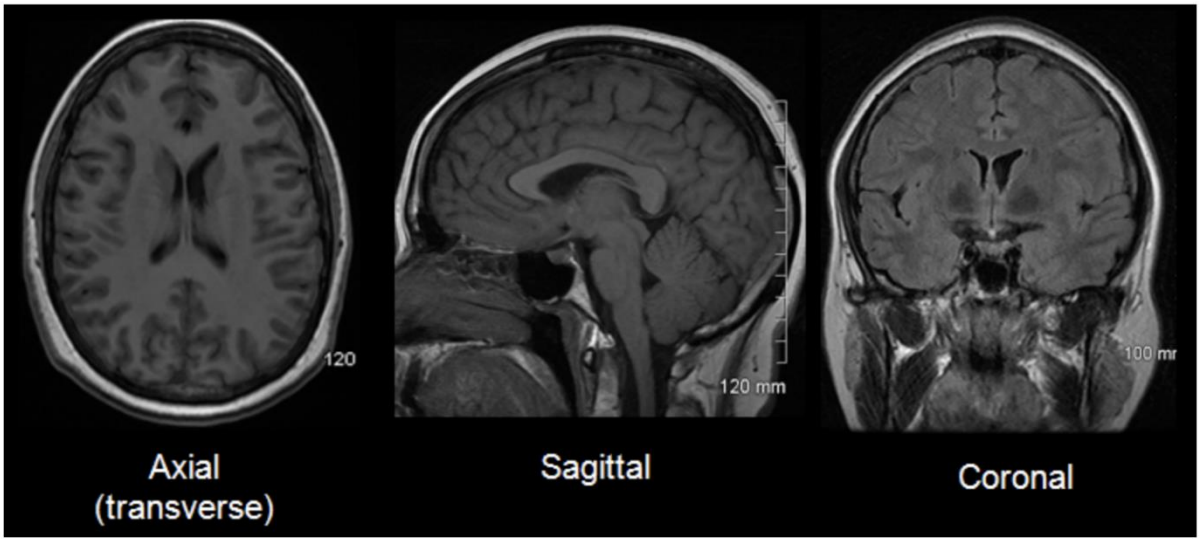

Figure 1 – MRI Images showing the brain on the three different planes (angles) – The Axial, Sagittal and Coronal.

There are three commonly used MRI Imaging Sequences which produce slightly different images each of which is determined by the choice of TR (Repetition Time) and TE (Time-to-Echo) times. TR is the amount of time between the successive pulses of Radio Frequency applied to a slice, and TE is the time between the delivery and receipt of these pulses.

The three imaging sequences are T1, T2 and FLAIR. T1-Weighted images are obtained using short TE and TR times, and conversely T2 images are achieved through longer TE and TR times. The third sequence, FLAIR, is like T2 imagery, except that the TE and TR times are very long. The differences between T1 and T2 images can be observed by looking at the Cerebrospinal Fluid – in T1 images the fluid is dark, and bright in the T2 images. Differences are shown in *Figure 2,* and some of the distinguishing features of each of sequences is detailed in *Figure 3.*

The datasets which are to be used in this project will contain images from all three angles shown in *Figure 1,* as well as images acquired through all three sequences of MRI. For information about the particulars of the dataset can be found in section 3.

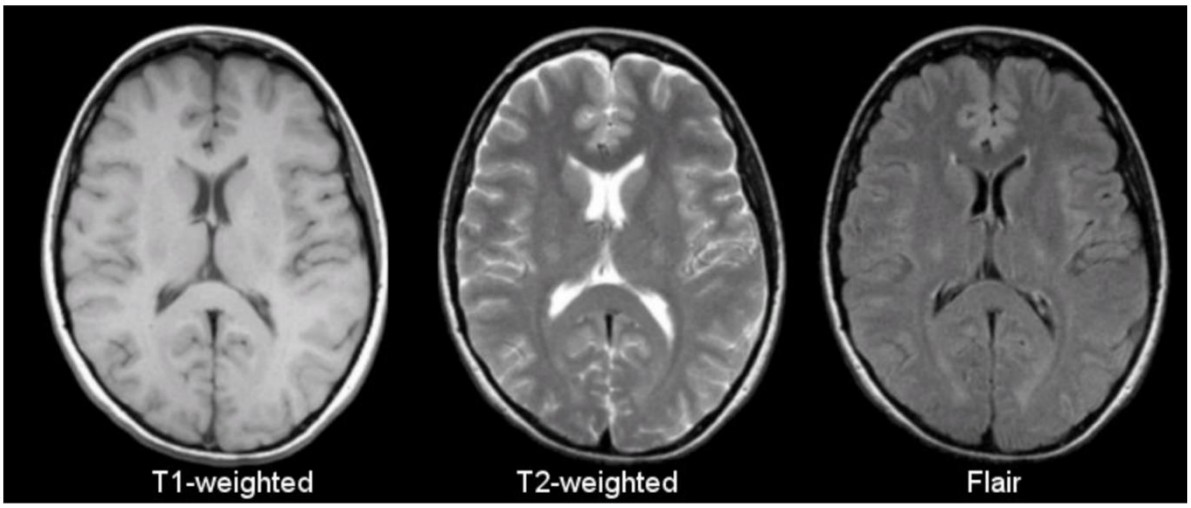

Figure 2 – Three MRI images demonstrating the different image sequences. T1-Weighted, T2-Weights and Flair.

Tissue	T1-Weighted	T2-Weighted	Flair
CSF	Dark	Bright	Dark
White Matter	Light	Dark Gray	Dark Gray
Cortex	Gray	Light Gray	Light Gray
Fat (within bone marrow)	Bright	Light	Light
Inflammation (infection, demyelination)	Dark	Bright	Bright

Figure 3 – A table containing the distinguishing details of MRI brain scans for each of the image sequences, T1, T2 and Flair.

2.3.2. Types of Brain Tumours

This project will focus three main types of tumours all of which are found in the intercranial space: The *Glioma*, *Meningioma* and *Pituitary*. It will be important to have a surface level understanding about these tumours and the distinctive features of each, so I am able to understand the model explainer used in this project. This is provided the features returned by the explainer are representative of the information researched for this section.

Of the three tumours mentioned, Meningiomas are the most common type of *primary* brain tumour which occur in the UK [18]. A *primary* tumour is a tumour that has not metastasized from another organ in the body, opposed to a secondary tumour. Meningiomas are commonly benign and form in a layer of tissue which covers the brain, and the spinal cord called the *Meninges* [18]. Meningioma can start in any of the brains regions but are most found in the cerebrum and cerebellum. An illustration of the meninges can be found in *Figure 4.* It follows that meningiomas are usually located around the

edges of a brain shown in an MRI image, where the Meninges are located. Examples of some common locations are shown in *Figure 5.*

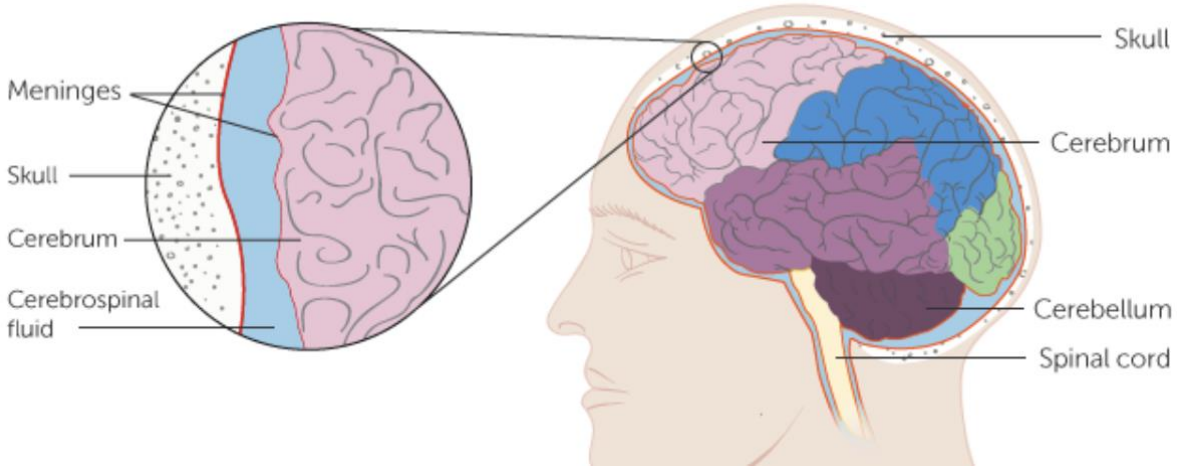

Figure 4 - Diagram of brains focusing on the meninges. Origin location for Meningioma Tumours. Source: [19]

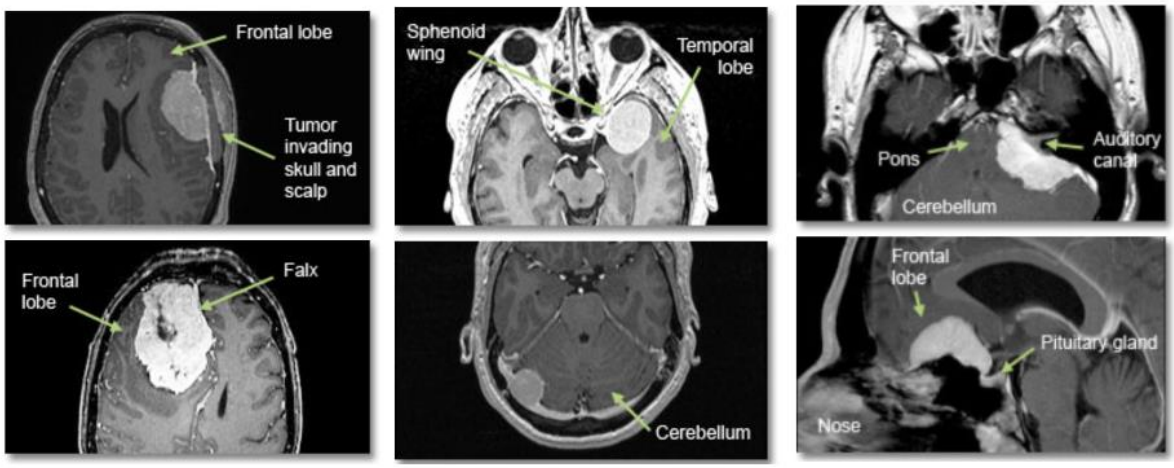

Figure 5 – A collection of common meningioma tumour locations shown on MRI brain scans. Source: Appendix 11.3.

The next type of brain tumour are the Gliomas which originate in cells called the *glial*. Of the glial, there are three different types. The astrocytes, tumours that start in these are called *astrocytoma*, or *glioblastoma*. The Oligodendrocytes, with tumours called *Oligodengrogliomas*, and finally the ependymal cells, with tumours called *ependymomas*. *Oligodengrogliomas,* and *ependymomas* are rare tumours, with the *Astrocytoma and Glioblastoma* being the most common [34]. Of these two tumours there are two classes, low grade which are slow growing and high grade which grow fast. Gliomas can also originate in the brain stem but these are very rare. All of these tumours can be benign or malignant. More details on these tumours can be found at [34].

Glioma tumours can occur in many different regions of the brain, but most commonly they occur in the frontal lobe, with a higher frequency of these being in the right hemisphere [19]. A visual representation of the distribution from [19] is shown in in *Figures 6 & 7.*

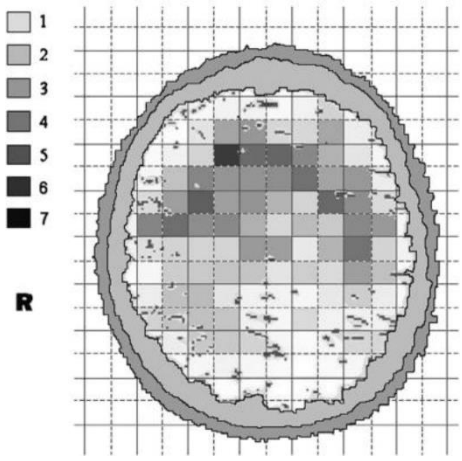

Figure 7 - Anatomic site distribution of Gliomas appearing in the axial projection of the brain. [19]

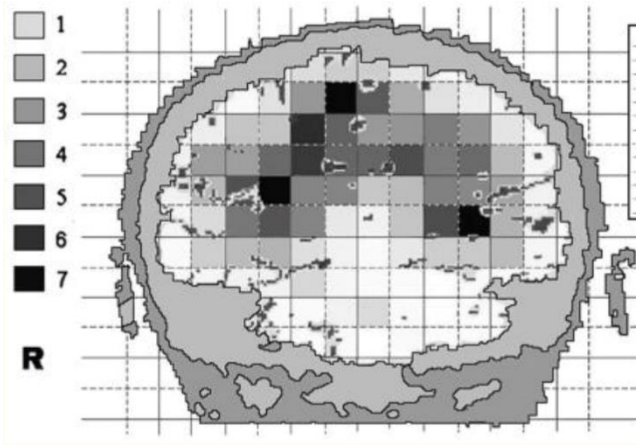

Figure 6 - Anatomic site distribution of Gliomas appearing in the coronal projection of the brain, facing the front of the head. [19]

Finally, pituitary tumours are growths that develop in the pituitary gland, which has been labelled in *Figure 8.* The occurrence of pituitary tumours is about 8/100 of all brain tumours diagnosed in the UK [35], and they are usually benign. Due to the anatomical location of the pituitary gland, these tumours can usually be easily distinguished from the Meningioma and Glioma, especially in the Sagittal and Coronal MRI planes. *Figure 9* shows an MRI scan with a pituitary tumour present sourced from a paper by Chaudhary et al discussing recent advances in imaging of the pituitary [21].

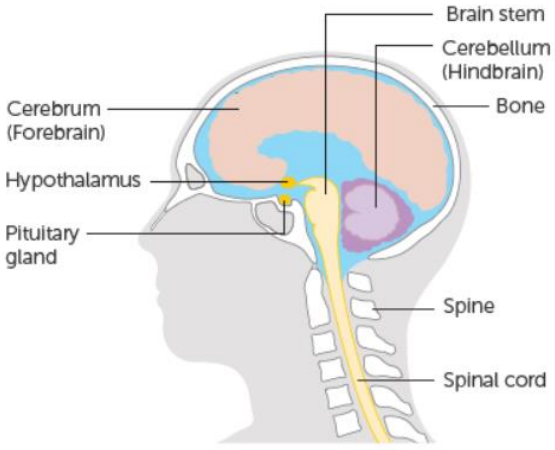

Figure 8 – An anatomical diagram of the brain labelling the Pituitary Gland

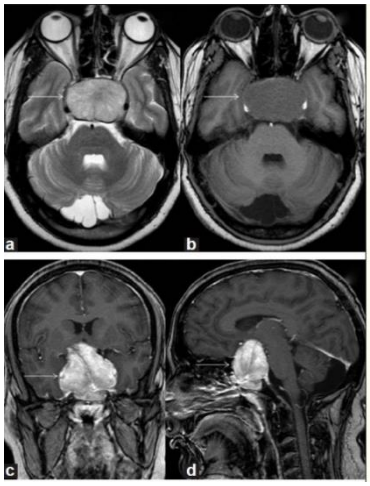

Figure 9 - MRI scans from [21] showing a pituitary tumour in the brain from all three MRI planes, Axial, Sagittal and Coronal.

The tumours discussed so far can cause abnormal affects to the brain other than the growths themselves, which can be observed in MRI scans. The tumours can create extra pressure in the brain which can distort tissue surrounding the tumour site. A clear example of this is what's called *Midline Shift.* This occurs when the pressure exerted by a mass in the brain is enough to push the brain off centre. This occurrence is considered an emergency and a sign of a more significant prognosis. It has

even been shown that patients in older age which present with this midline shift have a lower median survival period [20]. An illustration of the midline shift which can be present with a Glioma Tumour can be seen in *Figure 10*. I think this has a potential to be a very important factor in the classification of brain tumours with machine learning models.

Figure 10 - MRI scan with Glioma tumour present demonstrating the Midline Shift effect caused by pressure build up. Source: radiopaedia.org

2.4. Machine Learning Paradigms

In the machine learning subject space there are lots of different types of model which can be used, from simpler models like Logistic Regression, and K-Means Clustering, to a host of *Artificial Neural Networks* which include models such as Recurrent Neural Networks (RNN) and Long / Short Term Memory networks (LSTM). This project is going to focus on the use of a model from the *Artificial Neural Network* family called the *Convolutional Neural Network* (CNN).

The CNN model excels at image classification tasks and have been used in many problems similar to the one presented in this report [3] [17] [39] [43]. In the section, I will cover the details of the CNN at a fairly high level, explaining the role of each layer and digging slightly into the mathematics behind them, and using existing literature to justify the usage of certain parameters and functions in the models which will form the foundation of this project.

2.4.1. The Convolutional Neural Network

At a high level, the CNN is a type of deep learning model which processes data which is represented by a grid of values, such as images. It is designed to learn the features of an image, from high-level features in the earlier layers and low-level features in the lower layers. It is essentially a function which accepts an image as input, performs mathematical operations through a series of layers and then outputs probability values associated with a set of classes / labels.

When considering the structure of a CNN it's best to split it into three parts: The input layer, the hidden layers, and the output layer. The input layer is the simplest of the layers in a network and is responsible for feeding two- or three-dimensional matrix which represents and image, into the first 'hidden' layer. The hidden layer is any layer which sits in between the input and the output layer.

The input layer accepts a specific input shape which will depend on the size and shape of the dataset being used to train it. For example, in the MNIST dataset [36], the shape of images, and therefore the input layer is 28 x 28 x 1, in the format of HIGHT, WIDTH and CHANNELS.

The channels of an image simply whether it is a greyscale image (single channel), which means that image contains only one 28 x 28 layer of pixels. An image with three channels will consist of three layers of these pixels, each layer representing one of Red, Blue, or Green (RGB).

2.4.2. The Convolutional Layer & Activation Functions

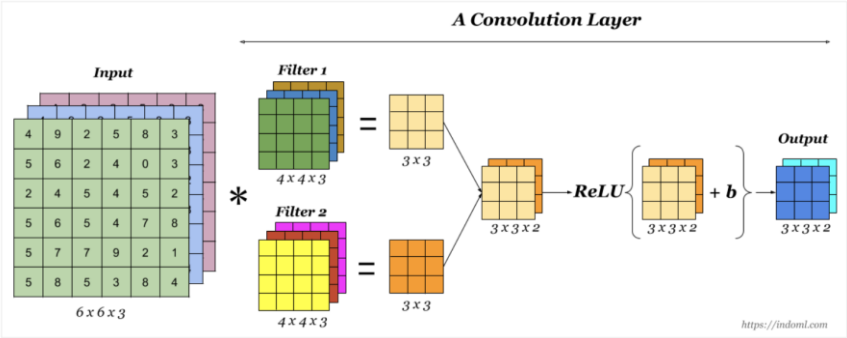

Figure 11 - A generic convolutional layer showing Input matrix, convolutional kernels, activation, and output. Source: Appendix 11.3.

The convolutional layer, a diagram of which is shown above in *Figure 11,* is responsible for extracting the features of an image. The feature maps that are generated by the convolutional layer are created via the application of convolutional kernels, also known as filters, to the input image.

These kernels are matrixes of a defined size, which slide over the input image and calculate the dot product *[Equation 1]* for each position on the image. A representation of this process is shown in *Figure 12.*

Equation 1 - Dot Product.

$$a \cdot b = \sum_{i=1}^{n} a_i \, b_i = a_1 b_1 + a_2 + b_2 \cdots a_n b_n$$

$$where,$$

$$a = Image\ Matrix\ (1st\ Vector)$$

$$b = Kernel\ Matrix\ (2nd\ Vector)$$

$$n = Dimentions\ of\ the\ kernel\ matrix\ space$$

$$a_i = Value\ of\ Image\ Matrix$$

$$b_i = Value\ in\ Kernel\ Matrix$$

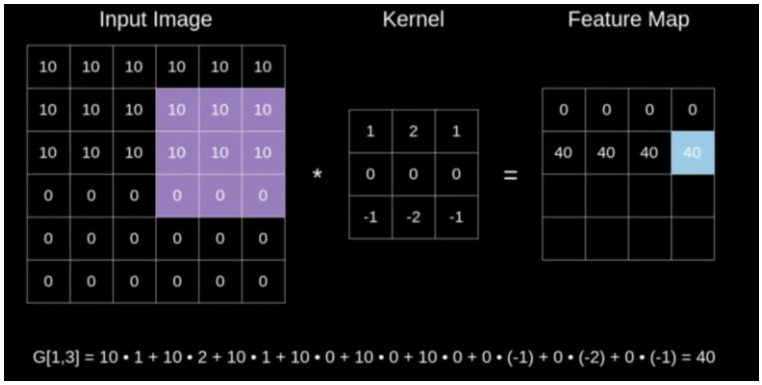

Figure 12 - Demonstration of a convolutional kernel applying the dot product to input values. Source: Appendix 11.3.

There are several kernel parameters that must be defined in a Convolutional Layer.

The first is the size of the kernel, most commonly the kernel is a 3x3 window, in which there would be 9 weights which are initialised randomly but then updated by the network through the processes of back-propagation (explained later), based on error score. As these kernels learn they will start to emphasise relevant features to the classification of the images, such as the edge detection demonstrated in *Figure 13* showing an MRI brain tumour scan with a tumour present.

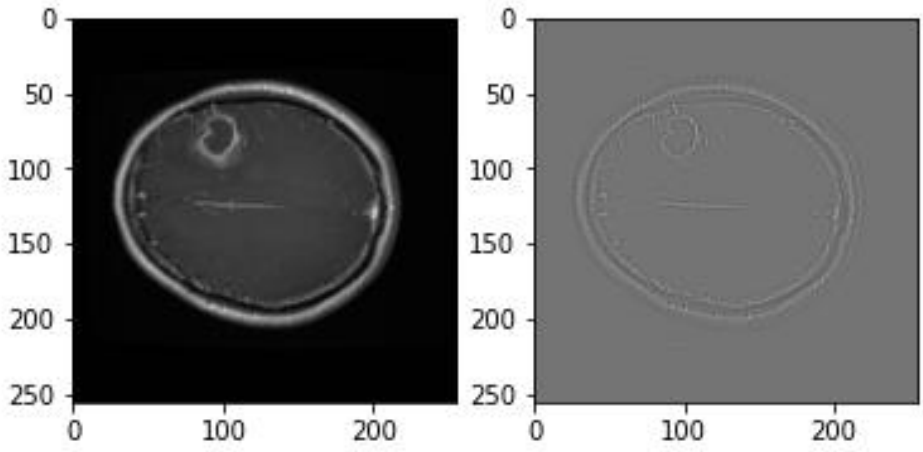

Figure 13 - Example feature map produced by a convolutional kernel with weights that produce 'edge detection' effect.

The next parameter is the *Stride.* Stride means how many pixels the window slides over an image after each convolution. The distance of this stride can dictate the size of the 'feature map', or image, that is produced. For example, in *Figure 12,* a 6x6 image with a stride of (1, 1) is translated into a 4x4 feature map.

Padding is a parameter that can be set to set a border around of 0 value pixels around an image, which means the feature map can maintain the original size of the image which is important in deeper networks, so the feature maps do not become too small. An example is shown in *Figure 14.*

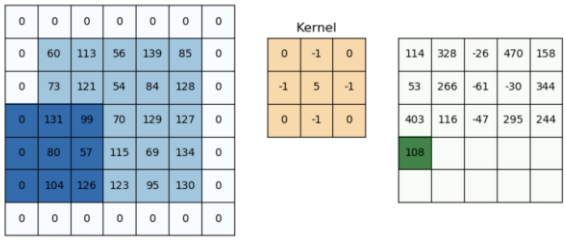

Figure 14 – Image showing a demonstration of a padding in a convolutional or pooling layer, and its associated kernel outputting a value. Source: Appendix 11.3.

The number of kernels is a use defined parameter and determines the number of features maps produced by the layer. The high-level features appearing in the beginning layers of a network can appear to be very abstract, but the deeper down a network, the clearer certain shapes and object become, these are called low-level features. A good example of this can be seen *Figure 15,* where in the first layer, only colours can be discerned, but by layer three there are certain faces present, as well as features like eyes and ears.

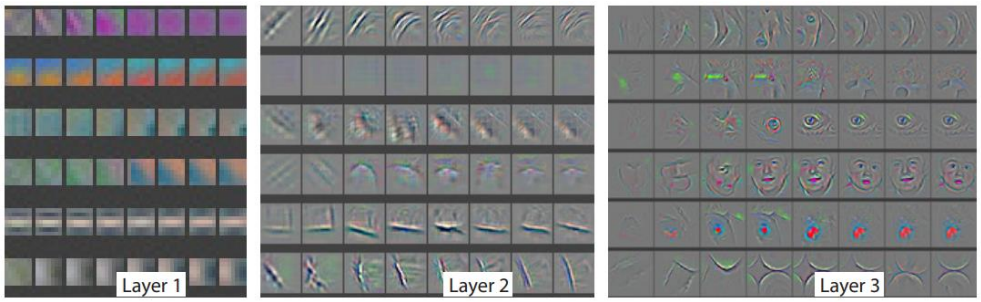

Figure 15 - Feature maps in layers of a deep convolutional network. Source: Appendix 11.3.

Finally, at the end of a convolutional layer an activation function is applied, as well as a bias. The purpose of the activation function is to introduce some non-linearity into the model, so that it may be fit onto complex non-linear problems. Without non-linear activation functions, the depth of a model would be irrelevant, and the model would never be able to learn to distinguish between classes in complex non-linear spaces.

There are many activations that have been historically used for Artificial Neural Networks, but in almost all state-of-the art CNN architectures, the function *ReLU* (Eq. 2) has been used for hidden layers, and *SoftMax* (Eq. 3) for the output layer [22].

ReLU, or Rectified Linear Units is a simple function which can be defined like so:

Equation 2 - Rectified Linea Unit (ReLU) Activation Function

$$ReLU(x) = \max(0, x)$$

The two large benefits of using ReLU is the training time, and the absence of vanishing gradients. Starting with the former; As more complex models were developed, training time became a problem as there are potentially millions of parameters that need to be learnt, and millions of activations which need to be calculated. ReLU has been shown to train networks faster (*Figure 16)* than any other activation function [23]. When thought of in the context of the size of these models, can add up to a substantially shorter training time. This was major factor in the success of AlexNet, a pioneering CNN trained for the ImageNet classification challenge by Krizhevsky et al in 2012 [44].

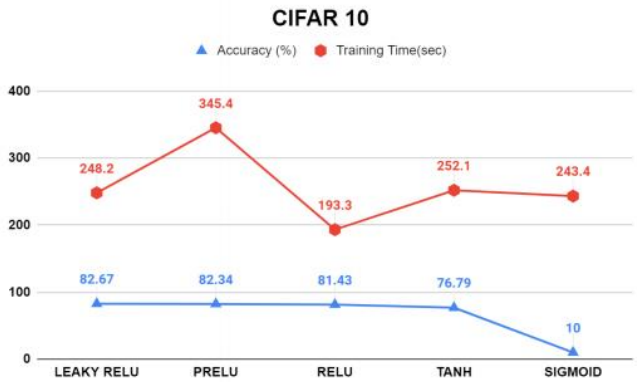

Figure 16 - Activation function training time statistics demonstrating the speed of ReLU from [24].

The second issue is vanishing gradients. As a part of the learning process, the weights in the network are updated as the total error of a classification is propagated backwards through the network. The problem comes with certain activation functions such as *Sigmoid* which squishes large inputs into a space between 1 and 0. If you observe *Figure 17,* you can see that the derivative of the sigmoid function (dashed red line) becomes very close to 0 as the inputs to the function become large. This can be a huge problem for updating weights in the early layers of a deep model, as the derivatives of each layer are multiplied down the network to from the end to the start. This is the process of back-propagation.

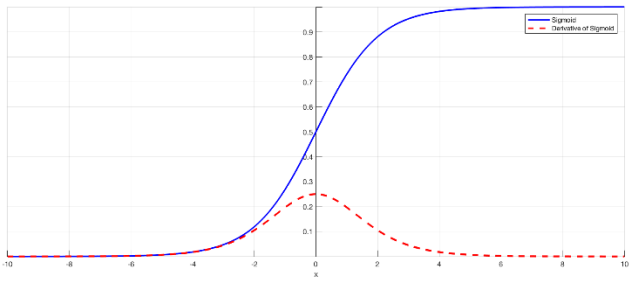

Figure 17 - Sigmoid graph showing the Sigmoid function and its derivative.

The vanishing gradient isn't a problem in shallow networks, but in large ones which use Sigmoid, the gradient decreases exponentially as we propagate back through the network multiplying these derivatives.

To combat this problem ReLU can be used. As can be observed in *Figure 18*, the first derivative of the ReLU function is simple; when the value is larger than 0, the derivative is 1, and otherwise 0. When this derivative is backpropagated through a network, there will be no degradation of shrinking of this value. It's for the reasons detailed in this section that ReLU will be the most appropriate activation function to you use in custom defined models used in this project.

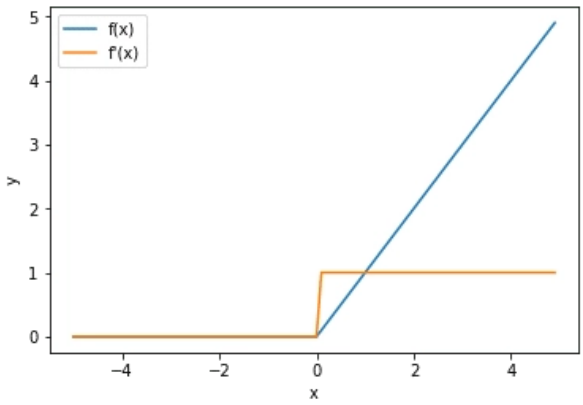

Figure 18 - ReLU activation function and its first derivative.

Moving onto SoftMax (Eq. 3) which is found on the final layer of a CNN and outputs a vector of values which range between 0 and 1, which sum up to 1, and can therefore be interpreted as a vector of probabilities. In this vector, each value is associated with a class label in the dataset. The input of the SoftMax function is a vector of values from the previous layer in the network. SoftMax is the most universally used output activation function for classification problems.

Equation 3 - SoftMax Activation Function

$$SoftMax\ (v)_i = \frac{e^{v_i}}{\sum_{j=1}^{C} e^{v_j}}$$

$$where,$$

$$V = Input\ Vector$$

$$e^{v_i} = Exponential\ Function\ for\ input\ vector$$

$$C = Number\ of\ classes\ in\ problem$$

$$e^{v_j} = Exponential\ Function\ for\ the\ output\ vector$$

2.4.3. The Pooling Layer

Another important layer in the CNN is the pooling layer, which can either be *Max Pooling,* or *Average Pooling.* Pooling is a form of image down sampling which employs the use of a kernel in the same manner as a Convolutional Layer. The kernel slides over an image, and in the case of max pooling outputs the largest value in that window, in average pooling it outputs the mean average of pixel

values (*Figure 19*). This pooling process has the benefit of extracting the sharpest and smooth features in the lower layers of a network, while also reducing the dimensions of the image resulting in faster computations, less memory footprint, and less parameters to update, which can also provide benefit by reducing the chances of overfitting, resulting in higher validation and generalisation accuracy [24].

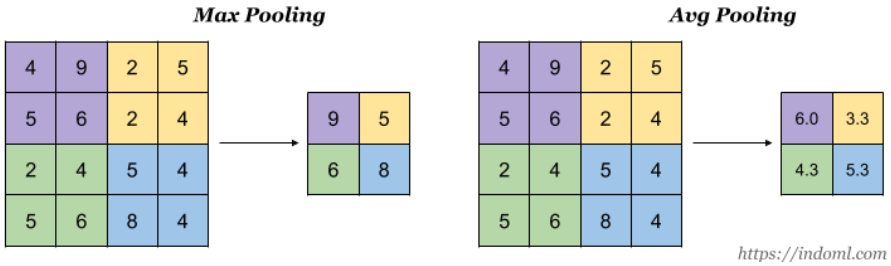

Figure 19 - Max pooling and Average pooling examples.

2.4.4. The Dense Layer

A dense layer, also known as a fully connected layer, is simply a layer in which every value in a previous layer is connected. In a CNN this usually follows a flatten layer, which as mentioned flattens the feature map matrixes from the convolutional layers into single dimension vectors.

Each neuron in the dense layer calculates a weighted sum of each of the outputs from the previous layer and adds a bias, and then the value is put into the activation function before being output to the next layer. Visualised in *Figure 20*. Dense layers are important to include in a model's architecture as they provide learning features from all regions of the feature maps in the previous layers. It's always recommended to have a dense layer before the output layer so SoftMax can consider every pixel in a feature map for its classification scores output.

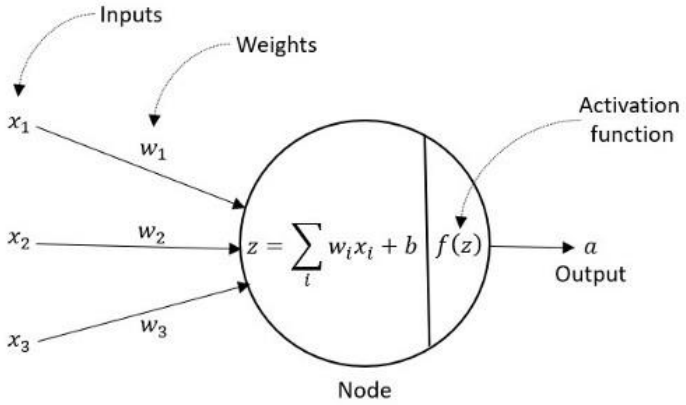

Figure 20 - A diagram of a single node (or neuron) from a neural network.

2.4.5. Loss, Optimisation & Back-Propagation

In a convolutional neural network, the weights of the connections between dense layers, and the values in the convolutional kernels are all updated through the *backpropagation of errors*, which was popularised by Rumelhard et al [25] back in the 80s. During the training process for a CNN each input image, is passed through the network and a prediction output is made. This prediction is compared to the group truth for the image, and a 'loss' is calculated, most commonly calculated by categorical cross-entropy in multi-class problems. This loss is some function of the distance between the desired output and the actual output. Backpropagation calculates the gradient of the loss function with respect to each weight in the network using the chain-rule (Eq. 4), calculating the gradient one layer at a time, from output to input.

Equation 4 - The Chain Rule

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

$$where,$$

$$\frac{dy}{dx} = Derivative\ of\ y\ with\ respect\ to\ x$$

$$\frac{dy}{du} = Derivative\ of\ y\ with\ respect\ to\ u$$

$$\frac{du}{dx} = Derivative\ of\ u\ with\ respect\ to\ x$$

The optimisation algorithm is what is responsible for changing the weights in the network based on the gradient calculated by the backpropagation algorithm. There are many popular optimisation algorithms which can be used for such problems as in this project, but *Adam* [27] is the most popular. The Adam algorithm is an alternative to stochastic gradient descent, and combines the best properties of the optimisation algorithms RMSProp, and AdaGrad and generally performs better than both of them, especially towards the end of training as gradients become sparser, and therefore is considered the best [26].

2.4.6. Transfer Learning & Data Augmentation

Transfer learning has become popularised with the rise of large, deep neural networks used to solve large image classification problems in recent times. Training accurate convolutional neural networks

can take a lot of time, resources and can be largely dependent on large datasets. The intuition behind transfer learning is that the weights from large models developed, trained, and tested by large companies for competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [37] can be used for other, similar problems, without the burden of requiring a large, labelled dataset and the resources to train from scratch. Transfer learning has proven to be effective, as concluded by Zhuang et al [38] in their comprehensive survey on the topic.

Popular models such as VGG16 [28] and ResNet [29] have been used for transfer learning in literature relating to the classification of medical conditions. The study by Magesh et al [31] used transfer learning from the VGG16 model to provide a foundation to a model which has been trained for early detection of Parkinson's disease. A further study by Prakash et al [30], focusing on the detection of Brain Tumours found that by utilising pre-trained models such as VGG16, ResNet and Inception, a classification accuracy of 100% was achievable for their dataset. Finally, a study by Badža et al [17], which uses a dataset of MRI Brain Scans containing tumours found that the VGG16 model yielded a 1.4% lower classification error than any other model present in the literature they reviewed. These promising results and therefore transfer learning models will be tested and evaluated in this project.

From the reviewed literature in above, it's clear that transfer learning benefits the training of models when datasets are small, but an additional method, called *Data Augmentation* which was used in both [30] and [17] has been shown to remedy the problem of small datasets to an extent and improves classification accuracy. Data Augmentation is the process of artificially expanding a dataset without adding any *new* images. It involves augmenting existing images in the dataset through methods such as rotating, flipping, cropping, and zooming. Using this, a dataset can be expanded and balanced, which both reinforce in the training process and makes the model more robust. An example of a brain tumour with data augmentation applied is shown in *Figure 21.*

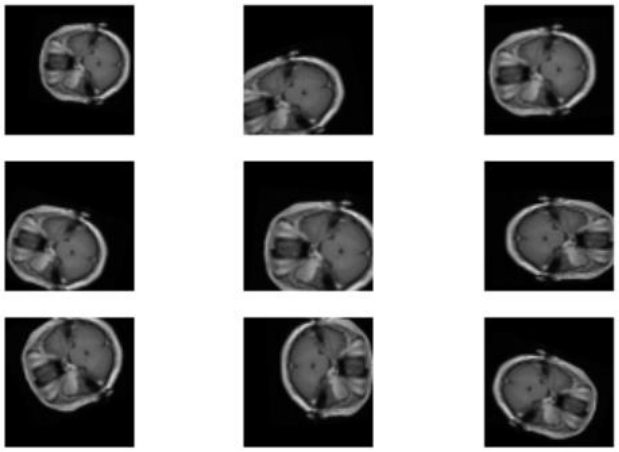

Figure 21 – Data Augmentation example.

2.5. Machine Learning in clinical Diagnosis

The capabilities of machine learning make it well-suited for classification in clinical diagnosis. As a result of this, there is a plethora of literature revolving around the use of machine learning in medical environments, and more specifically cancer prediction and detection. According to the latest PubMed statistics (*Figure 22)*, there have been >3650 scientific papers published between 1988 and 2021 with the subject of using Artificial Intelligence in Oncology related topics, with a surge of studies being published in the last four years:

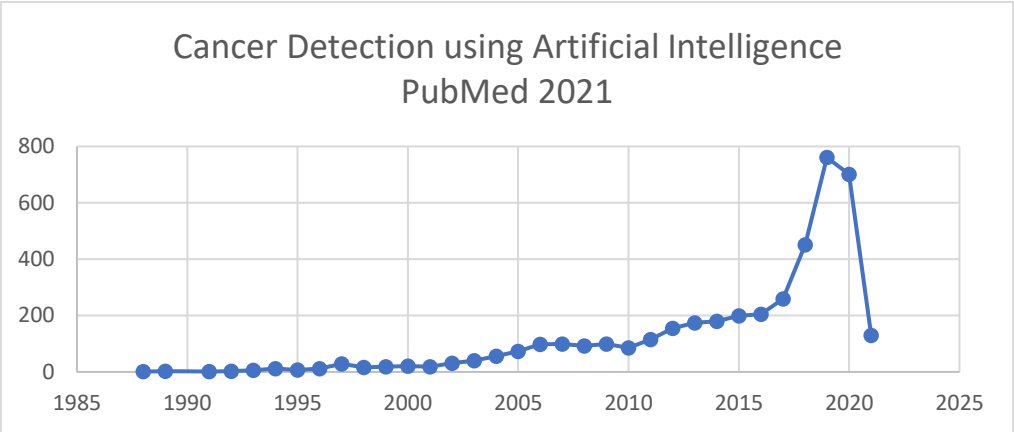

Figure 22 - Statistics showing the growth in publications with the topic of artificial intelligence for cancer detection. Source: Appendix 11.3.

As I have mentioned before in this report, one of the earliest studies dates to 1991 with the paper by *Maclin et al* [3]. Although this was a small study in comparison to the scale of those today, and despite the limitation on hardware in the early 90's, Maclin and his colleagues succeeded in training a prototype ANN on ultrasound images of 52 cases. The model was trained using a backpropagation algorithm and managed to correctly classify 47 cases which were held out of the training data.

Since this early paper there have been numerous advances in computing hardware as well as machine learning methodology which have increased efficiency and accuracy of models which has in turn enabled even more sophisticated studies to be conducted. A study which supports this assertion is that of *Dabeer et al* [39]. This study focused on the use of a CNN to classify histopathological images from biopsies of breast tumours. The dataset which was used in the experiment contained 8000 images of various magnification. Each image was labelled as either benign or malignant, making this a binary classification problem. The model used in the experiment consisted of an input layer, 6 hidden layers and an output layer, a diagram of the model used is shown in *Figure 23.* below:

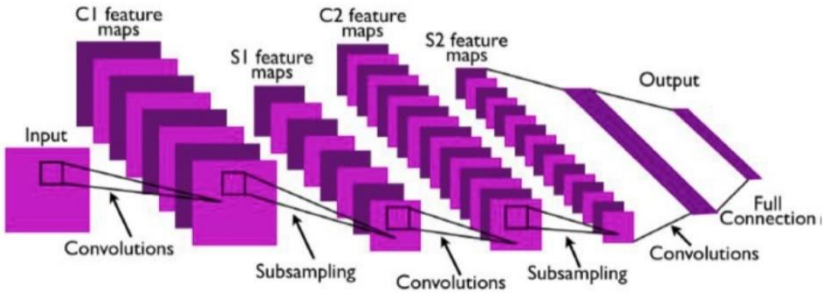

Figure 23 - Full diagram of convolutional neural network from [39].

Rectified Non-Linear Units were used for the convolutional layer activations, and SoftMax was used for the output layer. These activation functions as well as model architecture, and the roles of each layer is explained in section 2.4. Using this model, an accuracy of 93.45% was achieved on the validation set. The accuracy was then compared to those from other studies using state-of-the-art setups and it was found that they were all inferior.

A concern of mine that became apparent after researching brain tumour diagnosis and the types of brain tumours that are common is whether a machine learning model would be able to accurately distinguish between the tumours, namely the difference between Gliomas and Meningiomas. In general, the MRI is the beginning of a brain tumour diagnosis process, and once an MRI shows that there is a mass on the brain, it is common for specific diagnosis to be achieved through an intrusive biopsy. This is when a sample of the tumours tissue is taken and analysed by a pathologist under a microscope. Biopsies are the definitive way to diagnose the type of tumour. Researching this problem, I found a study by Piper et al [16] that suggests that MRI biomarkers between Gliomas and Meningiomas are significantly different, and by using MRI it may be possible to identify tumour pathology non-invasively. Additional information on the differences in these tumours can be found in section 2.3.1. Furthermore, a study by *Badža et al* [17] in which a dataset consisting of Glioma, Meningioma and Pituitary images were used to train a CNN which yielded accuracy results around 95% using the unaltered data, and 97% when applying augmentation to the images. What else is substantial about this study is that they found results comparable compared to those commonly used in transfer learning such as VGG16 [28] using a custom CNN model which was shallower. This means that these models are faster to use than these large models, with images in this study taking less than 15 MS per classification, which means the model can be run on personal computers with low-budget graphics cards. This may negate one potential constraint that is model size and complexity and hardware required to use them thus preventing their adoption into hospitals. Although the paper found results suggesting that their own model architecture was comparable for this case specific problem, the dataset used in my own project will differ largely and therefore I believe it prudent to still experiment with transfer learning from models such as VGG16 [28] and RESNET [29] to evaluate on my own data. Details about transfer learning and image augmentation can be found in section 2.4.

To summarise, it's clear from the literature that ANNs have had a prevalence in machine learning medical diagnosis from the beginning, as was found in a case study by Cruz and Wishart in 2006 [40], and that has only become more apparent with the advances in hardware technology as well as more sophisticated models becoming available with transfer learning. Despite this though, there seems to be little around model interpretability, specifically for medical diagnosis. There are few studies which I have found experimenting with explainable methods, one of which is a study by Bach et al in 2015 [41], which used a method of model explanation called Layer-Wise Relevance Propagation (More detail on this study is available in section 2.6.1.). Another being that of Jansen et al [42], which used Local Interpretable Model-Agnostic Explainability (LIME) to explain a model trained to predict the 10-year survival rate of breast cancer patients. (Additional information on LIME in section 2.6.3. and 7).

Although Neural Networks are very accurate, as supported by the literature, I have mentioned previously that they are difficult to interpret which is directly hindering their adoption into medical environments, and there is clearly a lack of literature experimenting with the use of Explainability algorithms applied to models trained for medical diagnosis, and more specifically for brain tumour classification. It is for these reasons that my project will focus on this topic area.

2.6. Deep Learning Explainability

Throughout my research I have come across various methods that can be used to explain the generalisations made by a deep learning model. I will break a few in this section, highlighting the key aspects of each method and how they are applied to deep learning models.

2.6.1. Layer-wise Relevance Propagation

Layer wise-Relevance Propagation is a method that provides pixel-wise explanations for a Non-Linear Classifier, initially proposed by Bach et al in 2015 [41]. the underlying idea behind the LRP algorithm is to apply a relevance score to each individual node in a deep neural network. This algorithm traces back through a network, recording the activation strength, i.e. the contribution of each node in the network. This can be mapped back to the input image to create heatmap representations of the importance of each group of pixels to the model's classification of an image.

In the research study conducted by Bohle et al, [43], they suggest that LRP can be used to visualise a CNN's decision in Alzheimer's Disease based classification, trained on MRI images, very similar to those in this project. Their study showed that by using LRP they provide individual Alzheimer's Disease relevance to each patients MRI. Showing the specific reasons as to why the CNN model had decided that the patient did indeed have AD. Bohle et al concluded that LRP could be used in case-by-case evaluation for Alzheimer's patients to provide valuable insight to their illness and the regions of the brain affected.

The authors of this paper suggested that their framework could be useful for other disease classification and in understanding a neural networks decision in these scenarios. I believe that there is a gap in the literature here for a study that investigates the potential for this method to be applied to the field of Oncology, specifically for brain tumour classification interpretability but due to the complexity of this approach and the scope of my project, I will not be implementing this method of Explainability, and instead will be using LIME which discussed in section 2.6.3.

2.6.2. Local Interpretable Model-Agnostic Explainability (LIME)

Local Interpretable Model-Agnostic Explainability, also known as the acronym LIME, is an explanation technique that if applied correctly, can explain predictions of *any* classifier. This method was originally proposed in the paper "Why should I trust you? Explaining the Predictions of Any Classifier" by Marco Ribeiro et al in 2016 [6]. This study has heavily influenced this project and is the reason for LIME being the chosen explainer the experiment.

Since the publication of this paper, LIME has attracted a lot of attention and has been tested on various models trained for medical diagnosis or prediction. Examples include the early detection of Parkinson's Disease using LIME applied to a CNN trained on DaTSCAN imagery [31], and the explanations of a model trained to predict the 10-year survival rate for breast cancer patients based on demographic, clinical and pathological data [42].

The intuition behind LIME is to apply a linear model around a chosen data sample to explain, which is contained in a more complex non-linear space. The linear model remains *locally* faithful, but not *globally* faithful. This is best visualised by in *Figure 24* . This is an example of a non-interpretable model

which is represented here by the blue and pink space, which would be impossible to approximate with a single linear model for all data points. The bold red cross is a chosen instance to be explained, and the other points are LIME perturbations of that same instance. The cost function used by LIME weighs them by proximity to the original instance. The dashed line is the learnt linear model which is locally faithful but not globally.

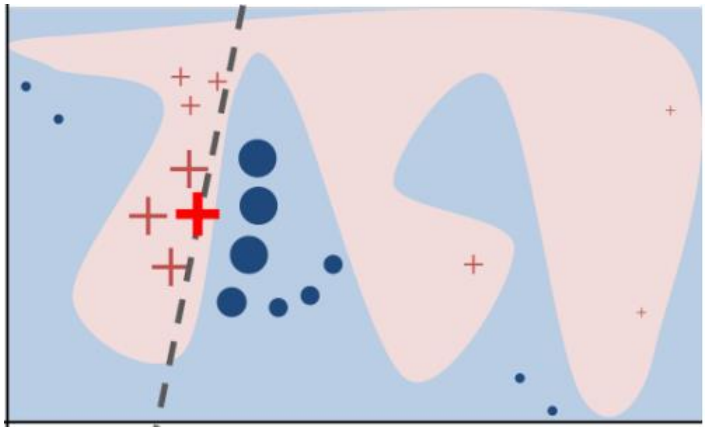

Figure 24 - Diagram showing the local faithfulness of LIME in a complex function space. Source: [6]

For image classification explanations, we can take an arbitrary image and create perturbations of that image with super pixels (a contiguous patch of similar pixels) enabled or disabled. We then pass each of these image perturbations through the model and record their weights towards specific classes which the model has been trained to classify.

Some of the logic behind LIME, *as defined in* [6]:

- Let an explanation of a model be fined as $g \in G$ where G is a class of potentially interpretable models.
- Every $g \in G$ may not be simple enough to be interpretable by a human, so let $\Omega(g)$ be a measure of complexity of an explanation $g \in G$.
- The model being explained will be denoted as f, where the probability that x belongs to a certain class is denoted $f(x)$.
- $\pi_x(z)$ is a proximity measure between an instance z (a perturbation of x) to x, as to define locality around x
- Lastly, let $\mathcal{L}(f, g, \pi_x)$ be a measure of how unfaithful $g \in G$ is approximating the model being explained f, in the locality defined by the linear model πx.

To ensure that a result is both interpretable by a human and locally faithful to the model at large, $\mathcal{L}(f, g, \pi_x)$ must be minimized while having $\Omega(g)$ be low enough to be understood by humans. The LIME explanation is defined by the following Eq. 5:

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \ \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

Equation 5 - Logical representation of the explanations produced by LIME.

For image classification, as mentioned before, perturbations of an image are used to provide the interpretable representation for classifications. These perturbations are created by a binary vector

indicating the presence or absence of a super pixel. Formally, X'={0,1}ᵖ' where ᵖ' is the number of super pixels an image has been segmented into, which are obtained using a segmentation algorithm. A function is then used to turn these pixels on or off based on the contents of the binary vector. *Figure 25* shows an image that has been segmented into Interpretable components:

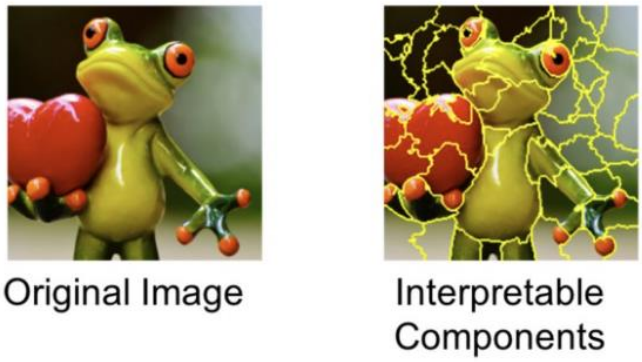

Figure 25 - Image segmentation example. Source: Appendix 11.3.

In the study, the Google pre-trained inception network is used to classify an image of a Labrador wearing a hat playing a guitar. They then use LIME to create perturbations of the image and show which of the perturbations have the lowest distance (similar probability scores) to the top three predicted classes made by inception model on the original image. Results of this experiment are shown below in *Figure 26*.

Figure 26 - LIME explanations of a number of classes from a single image, showing the most important features for each classification [6].

The ability to explain the decision of a model on a selected data sample can reveal insight into the model's trustworthiness and to understand undesirable correlations that classifier can learn during training. For example, in this study they conducted an experiment on a logistic regression classifier which has been trained to distinguish between Wolves and Huskies. The dataset was handpicked and the set of 20 images of Wolves all contained snow in the background, whereas the pictures of huskies did not.

When the model was used to classify new, unseen images of Wolves and Huskies, the model always predicted Husky if snow was present in the background (*Figure 27*), otherwise Husky. The model was trained to be a bad classifier for the purpose of seeing whether human subjects would be able to detect it.

Figure 27 – A demonstration of LIME explanation exposing the learned bias of machine learning model [6].

They then presented a set of 10 predictions without explanations containing some incorrectly classified images to a group of graduate students with some experience in machine learning. The students were asked if they trusted the model, why they did or did not, and how do they think the algorithms is distinguishing between the breeds of dogs. They were asked this set of questions twice, one before seeing the LIME explanation and once afterwards. The results are shown in *Figure 28*.

	Before	After
Trusted the bad model	10 out of 27	3 out of 27
Snow as a potential feature	12 out of 27	25 out of 27

Figure 28 - Stats showing trust in a machine learning model before and after the subjects have been shown the LIME explanations shown in Figure 26. [6]

Using LIME, the participants of the study demonstrated insight into the models learnt correlations which can help reinforce whether they can trust a model and why. This valuable insight I believe will be key in having machine learning models trusted and adopted widely into medical diagnosis and is a reason why I think LIME has great potential, and why it will be used for this project.

2.7. Literature Review Summary

To briefly summarise this section; I have explored a range of literature in order to prepare myself for the implementation of the machine learning models and the explainer. I'm confident that I now have a strong enough understanding of the best technologies and design choices to use when implementing my own models which will hopefully perform as well as those seen in some of the studies referenced. The LIME explainer seems like a promising method of interpreting the classifications of a model and I'm optimistic that I'll work in the context of this problem, and with my new foundational knowledge of MRI brain scan diagnosis I should be able to understand the results that it produces.

3. Data Analysis & Preparation

3.1. Dataset Source and Analysis

Datasets is an integral part of any successful machine learning project. It's important to have data that is fit for purpose, balanced and relevant. The dataset used in this project is a combination of two datasets containing a mix of T1, T2 and Flair Brain Scan MRI Slices which contain either a Healthy Brain, or a Glioma, Meningioma or Pituitary tumour. More information regarding MRI scans and the difference between T1, T2 and Flair can be found in section *2.3.1*. Statistics regarding the class balancing for the dataset can be found below in *Table 1.*

Table 1 – A table showing the class distribution of the datasets used in the project.

Class #	Tumour Type	Dataset 1	Dataset 2	Total
1	Glioma	1426	926	2352
2	Meningioma	708	937	1645
3	No Tumour	0	490	490
4	Pituitary	903	901	1804
Dataset Total:				6291

The first of the datasets has been sourced from a Jun Cheng [51], from the School of Biomedical Engineering, Southern Medical University, Guangzhou, China. Jun Cheng has used the dataset in two studies [46] [47].

The dataset contains 3064 Single Colour Channel (Greyscale) T1-Weighted Contract-Images from 233 patients which have been fully annotated, although this annotation information has not been used for this experiment. The dataset has been cleaned very well and each image has been stored in a mat v 7.3 file structure which contains the following information:

- Label: 1 for meningioma, 2 for glioma, 3 for pituitary tumour
- Patient ID
- Image Data
- Tumour Border: Annotated border showing regions of image with tumour indicated.
- Tumour Mask: a binary image with 1s indicating tumour region and 0s in all other locations.

The second dataset has been sourced from Kaggle [48]. This dataset consists of a mix of 3264 Three Channel Colour (RGB Formatted, although appear in greyscale) T1, T2 and Flair Weighted MRI Brain Scans. These scans either show a Glioma, Meningioma or Pituitary Tumour like the first dataset, but also contains images of healthy brains. A set of images containing 10 of each type have been put aside from this dataset which will serve as test images for the LIME explainer.

According to the dataset curator, the images has been cleaned by manual examination, and then a doctor was approached who examined all the MRIs and provided a certificate of validation. This information can be found in the discussion post with the author of the Kaggle dataset [48]. As this project is not being used for any real-life application, I believe the use of this dataset is justified. If this dataset was to be used in any other manner than purely for experimentation, then I would advise that a professional is asked to provide secondary validation.

This combined dataset in its unaltered state is very unbalanced – Glioma is overrepresented, and No Tumour is underrepresented. To remedy this issue data augmentation will be applied to the data, artificially increasing the size of the set as well as balancing the classes. More details on this can be found below in section 3.2.4.

By appropriately preparing this data I believe it will provide a solid foundation to meet the goals of this project. The majority of the dataset will use the train the machine learning models and evaluate them, and then a small subset of the data can be withheld and used to act as unseen new data, which the explainability method will be applied to.

To achieve these goals, my person computer will be used as the platform for the dataset preparation, model training and evaluation, as well as implementation for the explainability method. The hardware in the PC includes a NVIDIA 3060ti GPU with 8GB of memory which makes it suitable for loading and training using large datasets. The 3060ti also contains a large number of CUDA cores which provide efficient parallel computing capabilities and make it compatible with Nvidia CUDA-toolkit and TensorFlow-GPU packages. Having this capability will mean that model training will be much faster and more efficient than on most standard PC graphics cards.

My personal computer is running the windows 10 operating system, and Visual Studio Code will be used to write Python code implementing the function for this project, which will then be imported into Jupyter notebooks so the code can be executed cell-by-cell and data, models and the explainability method can be visualised and implemented intuitively.

I have listed some of the core python libraries which I expect to use in this project, with a description of what they will be used for:

- **Keras / TensorFlow**
  - Implementation of the Machine Learning Models.
- **NumPy**
  - Data processing, and training arrays.
- **OpenCV**
  - Image Formatting, converting images from RGB to Grayscale.
- **Pillow (PIL)**
  - Loading images from jpeg files into Python.
- **Matplotlib**
  - The visualisations of training graphs, evaluation plots and displaying images.
- **Scikit-Learn**
  - Image processing and machine learning model evaluations.

3.2. Dataset pre-processing

3.2.1. Loading images into Python

Due to the differences in format of the datasets loading them into Python requires a different approach for each dataset. Below I will detail each of the methods used:

For Dataset 1, since the contents have been stored in mat v7.3 files, a Python library called 'H5py' was used to access the structure and retrieve the relevant data.

The only data that I have extracted from the structure is each image and its correlated label which was executed using the Python function defined in *Figure 29*. This function also utilises the OS library which is built into Python and allows files from the local directory to be accessed.

The function returns an array containing two NumPy arrays, one which contains the labels and the other the images. The indexes of each image correspond to the index of the images associated label.

```
def loadDatasetOne():
    images = []
    labels = []
    for i in os.listdir('FYP/BrainTumourMatFiles'):
        h5_file = h5py.File(("FYP/BrainTumourMatFiles/"+str(i)), 'r')
        cjdata = h5_file['cjdata']
        label = cjdata.get('label')[0,0]
        image = np.array(cjdata.get('image')).astype(np.float32)
        images.append(image)
        labels.append(label)
    for i in range(0,len(labels)):
        if labels[i] == 1.0:
            labels[i] = 0
        if labels[i] == 2.0:
            labels[i] = 1
        if labels[i] == 3.0:
            labels[i] = 2
    return([images,labels])
```

Figure 29 - The Python Function defined to load in the images from dataset one into a python array.

Dataset 2 required a different function due to the storage of the images. The images are formatted as '.jpg', and each image is stored in its respected class folder. *Figure 30* shows the entire folder structure hierarchy. Due to the structure, nested loops were required to access each file and load it into Python.

Once again, the OS Library was used to access the image directories and the library known simply as *PIL*, or the *Python Image Library* was used to load the .jpg images into a NumPy format which was then appended to an array. The Python function defined to load this dataset is shown in *Figure 31.*

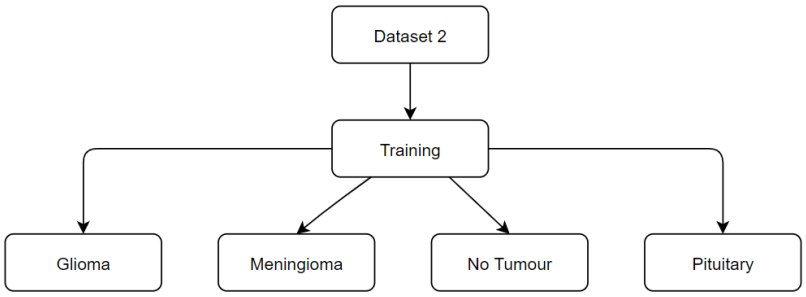

Figure 30 – Dataset 2 folder structure

```
def loadDatasetTwo():

    rootDirectory = 'FYP/newDataset/'
    training_data = []
    training_labels = []

    for root in os.listdir(rootDirectory):
        if root == 'Training':
            for t_type in os.listdir(rootDirectory + "/" + root):
                for i in os.listdir(rootDirectory + "/" + root + "/" + t_type):
                    training_data.append(np.array(Image.open(rootDirectory
                                        + "/" + root + "/" + t_type + "/" + i)))
                    training_labels.append(str(t_type))

    return([training_data, training_labels])
```

Figure 31 – Python Function defined to load and store the images from dataset 2 into an array.

3.2.2. Dimensionality Reduction

In data analysis and pre-processing there is a phrase called 'The Curse of Dimensionality'. This refers to several issues that can occur when your data has many dimensions. Large dimensionality causes sparsity in the Euclidean space, this can cause models generalise poorly on unseen data. High dimensional data also increases the overhead on storage and processing time, and very rarely does the increase in value from the increased number of dimensions outweigh the increased overheads.

Dimensionality reduction is a process of reducing the high dimensionality of a piece of data while maintaining some of the original and meaningful features. For image processing this usually means changing the size of an image and sometimes reducing the amount of colour channels, so turning an RGB image into Greyscale.

When first experimenting with CNNs (convolutional neural networks) for this project I used Dataset 1 without any dimensionality reduction. The images in that dataset, although they are single channel, the image sizes are 512x512. I immediately ran into storage problems with system RAM and GPU Memory when attempting to load the dataset and train a model. Observing these issues, the decision was made to reduce the image sizes. The images in the datasets used do not suffer much at all from being reduced in size, the key features (I.e., the tumours) which show on the MRI Scans are still distinguishable with the images reduced in size by half. *Figure 32* shows the visualisation of an image

which has had its dimensions halves, and *Figure 33* contains the Python code which was used to achieve this result.

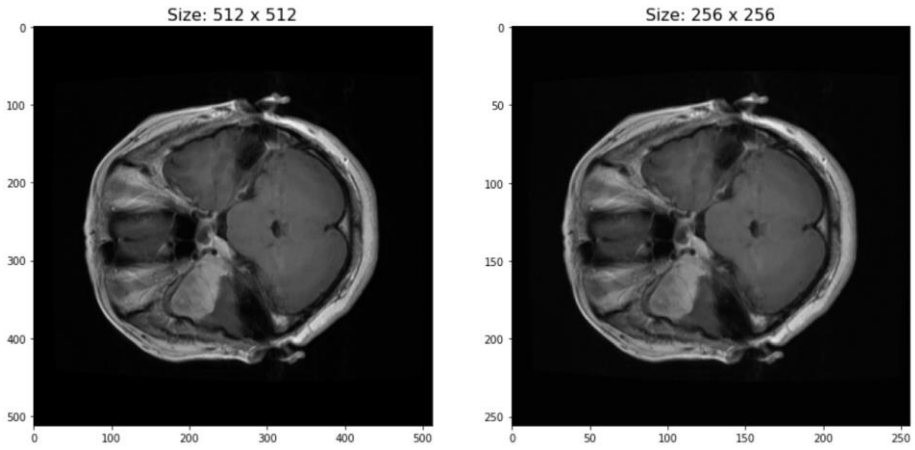

Figure 32 – Two images of the same MRI Brain Scan. The image of the left is of size 512 x 512 and the right is 256 x 256 to demonstrate the reduction of dimensionalities menial effect on the quality of the image.

```
def resizeImages(images, dataset):
    #Empty array to store resized images in
    resizeImages = []

    #Loop through all images in dataset
    for i in images:
        #Assign each image to a new variable, use CV2 to resize the images and convert to float32 bit.
        new_img = cv2.resize(i, dsize=(256, 256),interpolation=cv2.INTER_CUBIC).astype(np.float32)
        #Convert the image to numpy array
        new_img = np.asarray(new_img)

        #Check if dataset is one, otherwise skip
        if dataset == 1:
            #set new dimentions to single channel
            new_img = new_img.reshape(256,256,1)
        #Append new image to array
        resizeImages.append(new_img)
    #Return the new array as a numpy array
    return np.asarray(resizeImages)
```

Figure 33 – The python function which was defined to reduce the dimensionality of the images in the datasets. An example of the before and after of this function is shown in Figure 32.

Dataset 2 required more dimensionality reduction than dataset 1. Unlike dataset 1, the images in the second dataset were not uniform in size, therefore all images had to be adjusted to a common size to be input into the machine learning models. The images in this dataset were also in RGB format despite appearing as greyscale when observed, therefore they were all converted to single channel (greyscale) which substantially reduced the dimensionality.

To convert the RGB images into a single channel format the OpenCV Library was used and to reshape the images and then the NumPy library was used to reshape the image dimensions. *Figure 34* contains the Python function written for this process.

```
def rgb2Gray(images):
    #Define a new array to store new images in
    arr = []
    #loop through all images
    for i in images:
        #Using CV2 convert the image into grayscale
        img = cv2.cvtColor(i, cv2.COLOR_BGR2GRAY)
        #Reshape the image
        img = img.reshape(256,256,1)
        #Store the image in the new array
        arr.append(img)
    #Return the array of converted images as numpy array
    return np.asarray(arr)
```

Figure 34 - Python Function to convert images from RGB into Grayscale.

After dimensionality reduction and dataset combinations, the dataset had the following shape before any data augmentation was applied:

- Size: 6251
- Width: 256
- Height: 256
- Channels: 1

3.2.3. Image Rescaling and One-Hot-Encoding

One-hot encoding

In machine learning, categorical data is a type of variable that takes on a value from a limited selection. For example, the sex of a person would be categorical data. For computers to be able to process these values, they need to be converted into a format which can be taken as input into a machine learning model. One-hot encoding is the process of taking categorical labels in a dataset and converting them into a binary representation. The Keras library contains a useful function named *'to_categorical'* which can be used for one-hot encoding.

Applying one-hot encoding creates an additional binary feature for each of the unique labels in a dataset. For example, the dataset used in this project had the labels: *Glioma, Meningioma, No Tumour and Pituitary*. After one hot encoding, each data point is assigned a one-hot vector with four values, with each index representing a specific class, e.g. a *Glioma* image would be associated with a label [1, 0, 0, 0]. A full breakdown of the classes post one-hot encoding is shown in *Table 2.* A '1' represents presence, whereas a 0 indicates the opposite.

Table 2 - One-Hot Encoding Demonstration

	One-Hot Categories			
Text Labels	Glioma	Meningioma	No Tumour	Pituitary
Glioma	1	0	0	0
Meningioma	0	1	0	0
No Tumour	0	0	1	0
Pituitary	0	0	0	1

Image rescaling

Each pixel in an 8-bit image has a value range between 0 and 255 which indicated the intensity of said pixel. In greyscale, a 0 means a completely white tile and 255 means black. For neural networks, these pixel values can be considered as being rather large numbers which can cause adverse effects such as slow convergence and exploding gradients. This can be very expensive on resources and time. To negate this issue, we '*normalise'* each pixel in an image for every image in a dataset.

For this project, although the images are in a 32bit float format the normalisation still consisted of dividing every pixel value by 255. An example of this calculation and result can be seen in *Figure 35.* As a result of this normalisation process, computational power and model training time is reduced.

```
Image Array BEFORE Rescaling

[[150 250 150]
 [100  50  20]
 [250 250  10]]


Image Array AFTER Rescaling by /255

[[0.58823529 0.98039216 0.58823529]
 [0.39215686 0.19607843 0.07843137]
 [0.98039216 0.98039216 0.03921569]]
```

Figure 35 – The before and after of an array of 8-bit values before and after normalisation by dividing by 255 has been applied.

3.2.4. Data Augmentation

Due to the relatively small nature of this dataset, and because the classes are not balanced, data augmentation has been applied to remedy these issues.

The first augmentation was applied to dataset one, and this was a simple rotation augmentation of 90 degrees. The was necessary because the images in dataset one have been horizontally aligned, whereas dataset twos images are all vertically aligned. If this wasn't changed, it's possible that the models could develop some bias towards images in on of the datasets, which would affect the model's abilities to classify new unseen images.

The second step of augmentation was to apply the following transformation to each image in the set randomly:

- 50% of the images are flipped horizontally
- Images are rotated randomly between -20 degrees and 20 degrees.
- Images are sheared between -3 and 3 randomly.

The Python function written to achieve this augmentation is shown in *Figure 37* and the results of the augmentation on a selected image in *Figure 36.*

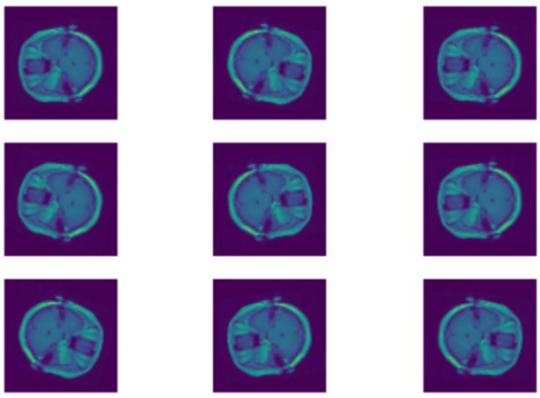

Figure 36 - Data Augmentation Example

```
def augmentImages(image_array, label_array):

    seq = iaa.Sequential([
        iaa.Fliplr(0.5), # horizontally flip 50% of the images
        iaa.Affine(
            rotate=(-20, 20),
            shear=(-3, 3))
    ])

    augmented_images = seq(images=image_array)

    return [augmented_images, label_array]
```

Figure 37 - Python Function created to produce augmented images, example of these images are shown in Figure 36.

Each augmented image was then appended to the original dataset. Through this process the combined dataset was expanded to a total of 13074 images and the class distribution is much more balanced. A class distribution breakdown is shown in *Table 3.*

Table 3 - Dataset class distribution after applying Data Augmentation.

Class #	Tumour Type	# Images
1	Glioma	3232
2	Meningioma	3270
3	No Tumour	3632
4	Pituitary	2940
Dataset Total: 13074		

4. Machine Learning Models for Brain Scan Classification

4.1. Introduction

The purpose of this section is to explore the machine learning models which are going to be applied to this project for MRI brain scan classification. This exploration will include looking at and providing explanations of the model architectures and parameters using visualisations, as well as demonstrating the implementation of these models using Python.

4.2. Model Architectures & Implementation

There are three Convolutional Neural Networks which have been used for this project. Two of which are transfer learning models have been used due to the success of them when applied to similar problems which was learnt from the Literature Review. These two models are the VGG16 [28] and ResNet50 [29]. The final model is a Custom Model which I have created myself. The purpose for creating my own model was to attempt to acquire comparable results to the transfer learning models, while being shallower and less complex. This inspiration for this has come from the success of the custom model used in [17]. The three models they have all been experimented with heavily, using a multitude of different parameters and functions. The descriptions below are of the most successful iterations of each of the models which presented the greatest results. The evaluations of these models can be found in Section 5.

4.2.1. Custom Model

Custom Model Description

The first of the models used in this project is one of my own design, with no transferred weights, trained from scratch. The model can be broken down into 5 distinct blocks. A full model diagram is shown in *Figure 38.*

The first block (shown in detail in *Figure 39*) contains the input layer which accepts images of dimensions (256, 256, 1) which feeds directly into the first two connected Convolutional 2D layers, each with 32 filters, a kernel of size *3 x 3* and padding set to '*same*' and the stride set to its default *1 x 1.* This combination guarantees that the feature maps maintain the same size as the input, preventing the images from shrinking too much too early. The layers are activated using '*ReLU*' to create non-linearity while preventing vanishing gradients. The second convolutional layer is followed by a max pooling layer, with a pool size of *2 x 2* and no padding. The output of this max pooling layer is feature maps of size *127 x 127.* The final component of this first block is a dropout layer which randomly turns off 50% of the neurons.

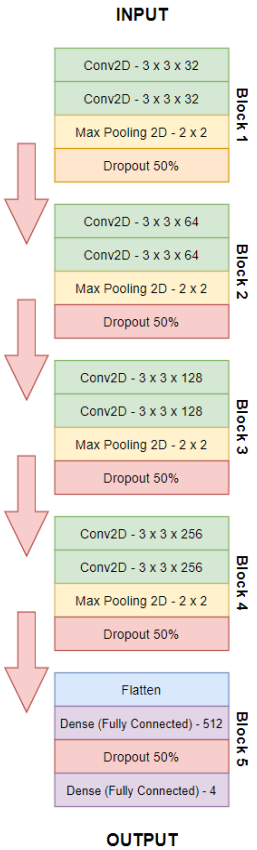

Figure 38 - Full architecture of the Custom Model described in section 4.2.1

The three middle blocks are identical in their structure with the only difference being the number of filters in the convolutional layer. Each layer consists of two convolutional layers, with *'valid'* padding (meaning no padding has been applied, therefore the image will shrink), kernel sizes of *3 x 3* and again activated by '*ReLU*'. The convolutional layers are followed by a max pooling layer with a pool size of *2 x 2* and '*valid*' padding, and finally a dropout layer set to 50%.

The number of filters output by each of the convolutional layer's doubles in each block, starting with 64 filters in the first of the three blocks, 128 in the second, and finally 256. By the end of the third block the feature maps, or images, have become quite small at *12 x 12* pixels. The three blocks described here are shown in *Figure 40.*

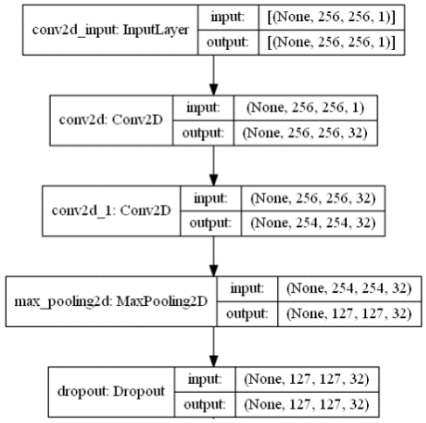

Figure 39 - The first block of the Custom Model - This diagram shows input and output shapes into and out of each of the layers in the block.

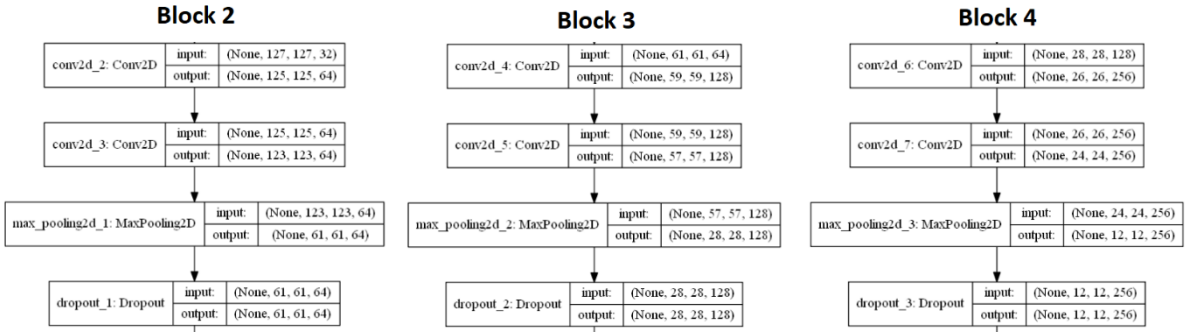

Figure 40 - Custom Model block 2–4 - These diagram shows input and output shapes of the layers in the blocks.

Finally, the last block in the model consists of a flatten layer, which takes all 2D arrays containing the feature map information from the previous block and flattens each into a 1D array. This enables the following fully connected, or *'Dense'* layer to connect to each value in the feature map. The Dense layer contains 512 fully connected neurons and is activated once again by '*ReLU*'. A dropout layer sits in-between this dense layer and the final output layer, which is a dense layer consisting of 4 neurons, one pertaining to each of the possible output classes. This layer uses the '*SoftMax*' activation function which outputs a vector of probabilities adding up to 1. The final block is visualised in *Figure 41.*

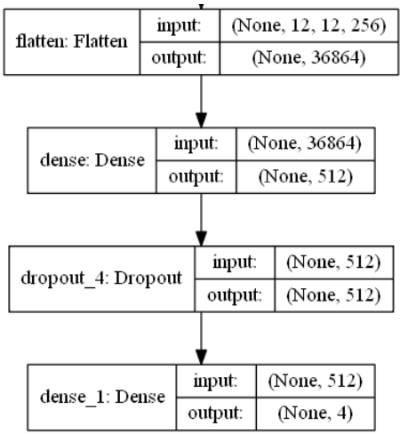

Figure 41 - Custom Model output block - This diagram shows input and output shapes of each of the layers in the block.

Custom Model Implementation

The Custom Model was implemented in Python using the Keras API. The individual layers and the sequential model can be imported as needed through the Keras library. The layers required for this model have been imported using the code shown in *Figure 42.*

```
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
```

Figure 42 - Python code showing the importation of the Keras API model and layers.

Once the model layers have been imported, the model is pieced together inside a function which returns the custom model to a chosen variable. Keras makes creating the model very simple, using the *model.add()* function. The function defined to create the custom model is shown in *Figure 43.*

```
def createCustomModel():
    #Block 1
    model = Sequential()
    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(256,256,1)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))
    #Block 2
    model.add(Conv2D(64, (3, 3), padding='valid', activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))
    #Block 3
    model.add(Conv2D(128, (3, 3), padding='valid', activation='relu'))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))
    #Block 4
    model.add(Conv2D(256, (3, 3), padding='valid', activation='relu'))
    model.add(Conv2D(256, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.5))
    #Block 5
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))

    return model
```

Figure 43 - Python Function defined to create and return the Custom Model CNN – It shows the definition of each of the block described in the work above, as well as the parameters (Padding, Activations, Number of Filters) for each layer.

The model is then compiled using the model.*compile()* function in Keras, which takes the arguments:

- **Loss**: The loss function to be used to calculate the model's loss after each batch.
- **Optimiser**: The optimiser used to minimise the loss function.
- **Metrics**: The type of training metrics to be returned by the training process. Usually Accuracy.

For this custom model, the loss function was '*Categorical Cross entropy',* the optimiser was '*Adam'.* The metrics argument was '*Accuracy'* which allows the observation of model loss and accuracy for each epoch, which can be graphed. The Python for code is shown in *Figure 44.*

```
model.compile(loss='categorical_crossentropy',
                        optimizer='adam',
                        metrics=['accuracy'])
```

Figure 44 – Python code demonstrating the compiling of the Custom Model with loss and optimisation parameters.

Finally, the model can be fit using model.*fit(),* in which many arguments can be passed, but in this project the following:

- **X**: Training Images Array
- **Y**: Training Labels Array
- **Batch Size**: Number of images passed through model before weights are updated.
- **Epochs**: Number of times the entire dataset is passed into the model.
- **Verbose**: Allows the progress and metrics to be viewed per batch / epoch during training.
- **Validation Split**: Portion of the dataset to act as unseen data to test accuracy.
- **Shuffle**: Shuffle the training data before each epoch.

The code showing the arguments passed into the fit function is shown in *Figure 45.*

```
history = model.fit(training_x, training_y, batch_size=32, epochs=50, verbose=1,
                    validation_split=0.2, shuffle=True)
```

Figure 45 - Custom Model fit function with learning parameters

The fit model is then assigned to a variable '*history*' which will contain all accuracy and loss metrics which can be saved and graphed in the future. This concludes the implementation of the Custom Model.

4.2.2. VGG16

The next model used is called the VGG16 Model [28], which was developed at the University of Oxford for the ILSVRC-2014 [37], which is a classification accuracy challenge on the ImageNet dataset. The base VGG16 model used is larger in depth than the custom model proposed in the previous section (4.2.1). The 16 in the name VGG16 stands for the number of Convolutional and Dense layers in the model. Pooling layers are not counted as they do not have learnable weights.

VGG16 Model Description

Only the first 5 blocks from the full VGG16 model architecture shown in *Figure 46* have been used for this project. The dense layers in the final block have been omitted for this project and will be replaced with custom output layers. This is a common occurrence when using VGG16 transfer learning for datasets other than ImageNet, although the weights from ImageNet on the first 5 blocks are retained.

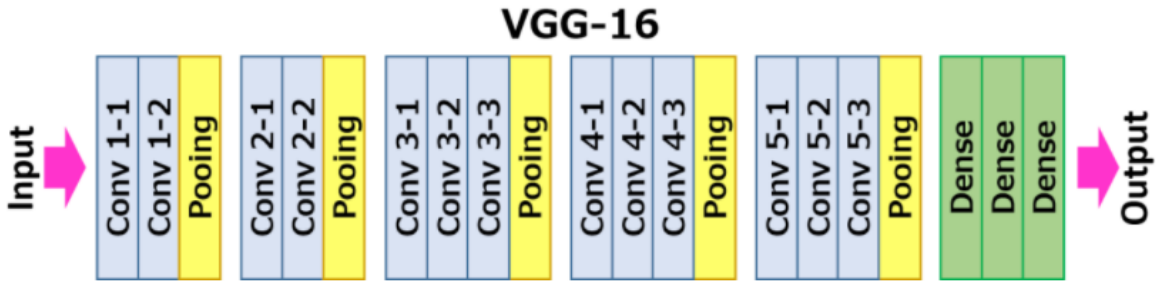

Figure 46 – The full architecture of the VGG16 model. The first 5 blocks of convolutional and pooling layers are used in this project and the dense block is replaced with a custom block.

The first block of the VGG 16 model consists of two convolutional layers activated by *'ReLU'*, each with 64, *3 x 3* convolutional kernels with 0 padding and a stride of *1 x 1.* This is followed by a max pooling layer with a kernel size of *2 x 2* and a stride of *2 x 2*. This block also contains the input layer, which accepts images with the dimensions *256 x 256 x 3*.

The second block is identical with the exception that the convolutional layers now each have 128 filters each. The third block has an additional convolutional layer, and each convolutional layer now has 256 filters each. Finally, the fourth and fifth block are the same as the third, but rather than 256 filters, each convolutional layer has 512 each. A full breakdown which includes the input and output shapes of each of the layers for each of the blocks described is shown in fig *Figure 47.*

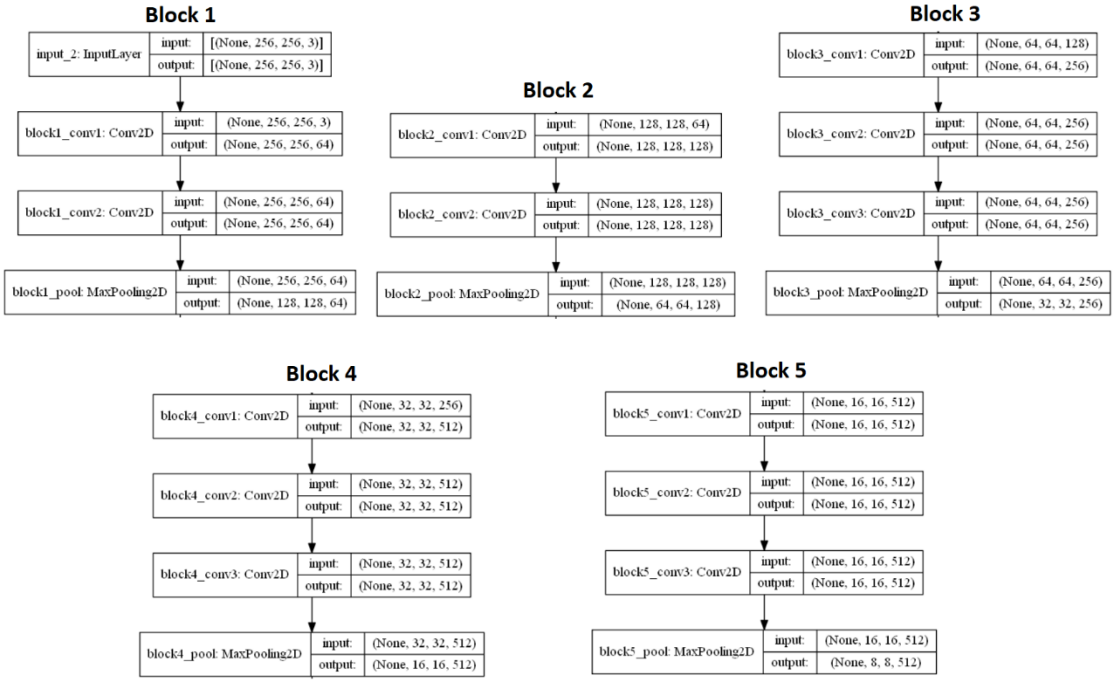

Figure 47 - The first 5 blocks of the full VGG16 Model architecture showing the input and output shapes from each of the individual layers.

Because the model has been downloaded, and the top layer which contains the dense layers and output shown in *Figure 47* above have been omitted, an additional block which contains a probability output needs to be appended before the model can be used.

A very simple block of layers has been appended for the purpose of this project. The first layer in the final block is a *flatten*, to convert all the 2D feature maps from the final convolutional layer in the VGG blocks to a 1D array. This is followed by a *dropout* layer set to 50% to reduce the chances of model overfitting, and finally there is a *Dense* layer containing 4 neurons, which is the output layer activated by *'SoftMax'.*

One final addition to the model was necessary to facilitate the use of the grayscale dataset used for training. The VGG16 model is configured for the use of ImageNet dataset and for this reason the first convolutional layer expects RGB images. To get around this, at the start of the model a single convolutional layer which accepts inputs *256 x 256 x 1* images has been added, with three convolutional filters of size *3 x 3,* and *'same'* padding to maintain image resolutions. The output of this layer is *256 x 256 x 3,* which are the required inputs for the VGG16 model. The complete model, with VGG16 wrapped inside is shown in *Figure 49.*

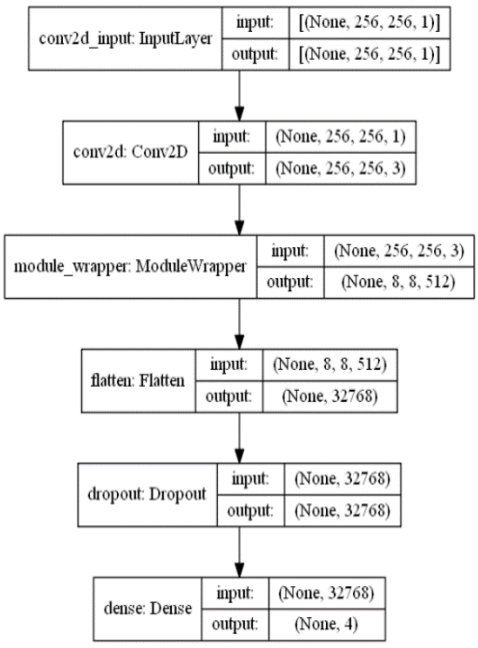

Figure 48 - Diagram showing the full architecture of VGG16 implementation for this project. The diagram also shows the input and output shapes for each of the layers in the model.

VGG16 Model Implementation

The implementation of this model was once again handled through the Keras library API, which makes the whole process very simple, and the model can be built in only a few blocks of code. Once Keras has been imported we can use Keras *Applications* to download and assign the VGG16 model to a variable, with custom arguments. The arguments passed in for the purposes of this project were to drop the final block of dense layers, use the weights from the ImageNet competition, and for the input shape to be *256 x 256 x 3.* The code implementing this is shown in *Figure 49.*

```
#import Keras and load the VGG16 Model and Weights
import keras
vgg_model = Keras.applications.VGG16(include_top=False,
                                     weights="imagenet",
                                     input_shape=(256, 256, 3))
```

Figure 49 – Python code showing the VGG16 transfer learning model import and variable assignment

Because the model is using the ImageNet weights, I don't want to train every layer when passing through my own dataset. This would defeat the point of transfer learning and will increase the total training time quite drastically. Of the 19 layers in the model, the first 14 which constitute the first four blocks of VGG16 have been frozen. This is achieved through looping over the layers and setting *'trainable'* to false. The method used to achieve this in Python is shown below in *Figure 50.*

```
for layer in vgg_model.layers[:-5]:
    layer.trainable = False
```

Figure 50 - Python code showing the freezing of VGG16 layers 1 – 14.

With those layers frozen, the model can be wrapped in the custom input and output layers to create the full network. The method to create the model and add the layers is identical to that of the custom model explained before, except this time the entire VGG16 model is added as a layer in between. *Figure 51* shows the Python code implementing the model described in this section using the Keras API.

```
def createVGGModel():

    model = Sequential()
    model.add(Conv2D(3,(3,3),padding='same',input_shape=(256,256,1)))
    model.add(vgg_model)
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))

    return model
```

Figure 51 - Python function defined to create the VGG16 model with the custom input and output layers.

Now that the model has been created the total parameters in the model come to approximately 14.8 million, with 7.6 million of those being non-trainable, and 7.2 million being trainable.

Compiling this model is the same process as that described in section 4.2.1. above. The arguments accepted are the same except for the optimisation function. For the training of the VGG16 model, *Stochastic Gradient Descent* was used, as opposed to *Adam*. For training the model, the following parameters have been used:

- **Batch Size**: 32
- **Epochs**: 30
- **Verbose**: 1
- **Validation Split**: 0.2
- **Shuffle**: True

Although using a validation split during wasn't necessary as the model was then tested on a separated set of unseen data, the results of which will be explored in section 5, it was beneficial as validation loss and accuracy could be monitored and graphed for analysis over the course of training.

4.2.3. ResNet50

The final model, ResNet50, is a variant of ResNet, a collection of models with varying architectures. ResNet was developed and entered into the ILSVRC [37] by a research team out of Microsoft. It's an extremely deep model with 50 layers, 49 of which are Convolutional, and a single Dense, with a total of over 23 million trainable parameters.

ResNet50 Model Description

The model's architecture is a lot more complicated than the more traditional CNNs and other two models in this project and because of this depth and complexity representing it in a diagram isn't an easy task.

Figure 52 shows a table containing various ResNet architectures, with ResNet50 being labelled in the middle. From the table, we can see that ResNet50 is divided into 5 blocks of convolutional layers. Blocks 2-5 each contain a set of three convolutional layers which are repeated a number of times,

shown by the multiplication of each set in the square brackets. The first block contains a convolutional layer as well as a max pooling, and the output layer consists of a dense with 1000 neurons activated by SoftMax. The reason there are 1000 is because ImageNet contains 1000 classes.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 52 - Compilation of ResNet model with ResNet50 showing the in the centre.

One of the novel additions to ResNet is the skip connection, which provides an alternate path for the gradient during backpropagation. The skip connection, as the name suggests, allows the output from some layer to skip the layers immediately after and input to later layers in the model. This approach mitigates the problem of vanishing gradients in deep neural networks. An example of a skip connection is shown in *Figure 53*.

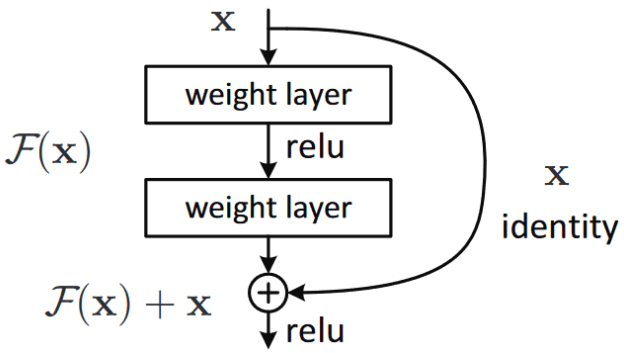

Figure 53 - ResNet50 skip connection

Due to the extremely large nature of ResNet50 and the branched nature caused by the skip connections, a block by block breakdown isn't feasible for this report. To summarise the model, all the activations for the convolutional layers are handled by '*ReLU*', and each convolutional layer is usually followed by a batch normalisation layer.

Much like the transfer learning of the VGG16 model, the inputs for ResNet50 are three channels, so the same 3 filter convolutional layer which handles the input to the VGG16 model has been added to

the beginning of the ResNet50 module enabling the input of the single channel images from my dataset. Additionally, the same output block (the flatten, followed by dropout of 50% and the 4-neuron dense layer), which was used in the VGG16 module has been added to the end of ResNet50 to handle class probability.

ResNet50 Model Implementation

The implementation of ResNet50 is much of the same as that shown in the VGG16 section, 4.2.1. *Figure 54* shows Python code of the assignment of the ResNet50 model to a variable, with the arguments of including output block to False, using ImageNet weights and setting the input shape to the dimensions of the dataset, with the exception of the channels being three, as single channel isn't natively compatible.

```
#Import the ResNet50 model and assign it to variable with desired configuration
resnet_model = Keras.applications.ResNet50(include_top=False,
                                           weights="imagenet",
                                           input_shape=(256, 256, 3))
```

Figure 54 - Python code demonstrating the assignment of the ResNet50 model to a variable with custom parameters.

The code that implements to freezing of the first four blocks of convolutional layers requires some further explanation. As shown in the Python code in *Figure 55*, The first 143 layers in the model are frozen, which is more than the stated "50 layers" which constitutes ResNet50. In ResNet's model architecture, all the merges from skip connections, batch normalisations and activations are treated as separate layers. Counting all these as individual layers means there is in fact a total of 175 layers, of which the first 143 must be frozen. This leaves the remaining 9 convolutional layers trainable. Officially, only the 49 convolutional layers, the single dense layer is counted, which explains the "50 layer" architecture.

```
#Freeze the first 143 layers of the model (first four blocks)
for layer in resnet_model.layers[:143]:
    layer.trainable = False
```

Figure 55 - Python code showing the freezing of the first four blocks of ResNet50 (143 layers).

Having frozen the first four blocks of the ResNet50 model, and after adding the input convolutional 2D layer and output dense layer (Python code showing implementation in *Figure 56*), the total parameters for the model sits at approximately 23.6 million, with 15 million of those being trainable and 8.6 million not trainable.

```
def createResNetModel():

    model = Sequential()
    model.add(Conv2D(3,(3,3),padding='same',input_shape=(256,256,1)))
    model.add(resnet_model)
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))

    return model
```

Figure 56 - Python function defined to create the ResNet50 model with the custom input and output layers.

The model was compiled in the same manner as the previous two, with categorical cross entropy being used to calculate loss, and '*Adam*' being used to minimise it. The fit function to train the model used the following parameters:

- **Batch Size**: 32
- **Epochs**: 50
- **Verbose**: 1
- **Validation Split**: 0.2
- **Shuffle**: True

4.2.4. Machine Learning Model Training & Validation Splitting

When training a model, it's important to get a feel of how it is performing by using new, unseen data and measuring the accuracy of its predictions. A very common method of validation us by using a holdout set, or a validation set.

A holdout set purely consists of a portion of the dataset which has been held out of training. A very common split, and the one which is used in this project is an 80/20 split.

The whole dataset is shuffled, and then the first 80% of the dataset is assigned to a new variable. The remaining 20% of a dataset is assigned to a separate variable and is held out of training. After the model has been trained on the 80% split, the model is used to predict the classes of the remaining 20% of the data. Using the model to predict this data, instead of using it to fit the model, means that weights in the model aren't updated. By using this held out data we can calculate various performance metrics, such as *Precision, Recall* and *F1* scores. Confusion matrixes can also be visualised to measure the number of True Positives, True Negatives, False Positives and False Negatives of the model. These measurements have been discussed in section 5 below.

All of the models defined in this section have been trained and evaluated using this method, although I must note that during the training of these models, I did include a validation split in the *fit* arguments. This may seem redundant as I am holding out 20% of the data for validation afterwards, but by including this validation split during training I am able to observe and store the models training progress and evaluate how quickly they improve through graphed visualisations.

5. Evaluation of the Machine Learning Models

5.1. Introduction

When evaluating a machine learning model, it's important to include a wide range of measurements which will determine whether the model is performing as desired. In this section I will explore the three models described in section 4, and evaluate each model by the following metrics:

- Accuracy and loss during training
- Confusion Matrixes
    - Binary (Tumour vs No Tumour)
    - Multi Class
- Accuracy, Precision, Recall and F1-Score

5.2. Model Training Loss & Accuracy

The loss of a model is calculated by the loss function at the end of each batch, which is then minimised by the optimisation function. All three models in this project share the same loss function, Categorical Cross-Entropy, the mathematical definition of this function has been defined in Eq. 6.

Equation 6 - Categorical Cross-Entropy loss function

$$CE = -\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i$$

$$where,$$

$$CE = Categorical\ Cross - Entropy$$

$$y_i = Ground\ Truth\ Class$$

$$\hat{y}_i = Predicted\ Class$$

$$N = Number\ of\ Classes$$

When training a model, the Keras library produces information which can be stored in a data structure and then graphed to monitor progress. In this sub-section the loss and accuracy graphs from each of the models will be compared. The validation scores shown below are from the validation split defined in the *fit* function for monitoring during training – more thorough testing was done of each model on unseen data which had been held out of the training process. The results of this testing will be in section 5.3.

Accuracy and Loss each have two measurements, one pertaining to the training data and the one to the validation split. First let's look at ResNet50, which was by far the least consistent in training and yielded the worst results of the three.

The ResNet50 model was trained over 50 epochs, at which point the accuracy stopped improving. By looking at the graphs in *Figure 57* below, you can see that training wasn't smooth at all, the training loss fluctuated between 0.5 and 1.5 consistently throughout training, and validation loss had a few moments where it grew very large. Fluctuation during training isn't uncommon, especially with optimisation functions like Stochastic Gradient Descent (SGB) and others which use some version of SGD, but with this model, compared to the other two, it appears to be much larger and less predictable.

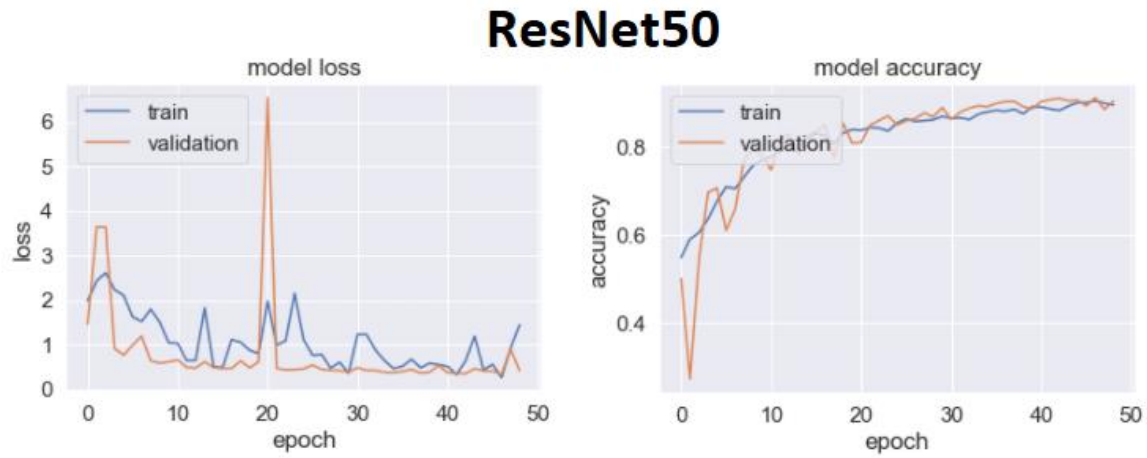

Figure 57 – The accuracy and loss of the ResNet50 model measured during training. Loss (Left) and Accuracy (Right).

The models accuracy eventually starts to settle down between epoch 40 and 50 and ends with the following scores: ***Training Accuracy****:* 89.55%, ***Validation Accuracy****:* 90.44%, ***Training Loss****:* 1.44, ***Validation Loss****:* 0.4.

Moving on now to the Custom Model, which ended up being the middle performer of the three and used the same optimisation function as the ResNet50 model, '*Adam*'. Below, shown in *Figure 58*, are the training statistics. Immediately, just by looking at the Y-Axis on model loss and accuracy you can see the range of fluctuations is much smaller than those present during ResNet50 training. The Custom Model was also trained over the course of 50 epochs and had a much smoother convergence, with consistent improvements being made to the models training accuracy and training loss. There were still fluctuations in validations accuracy and loss present, but the magnitude of these was small in comparison.

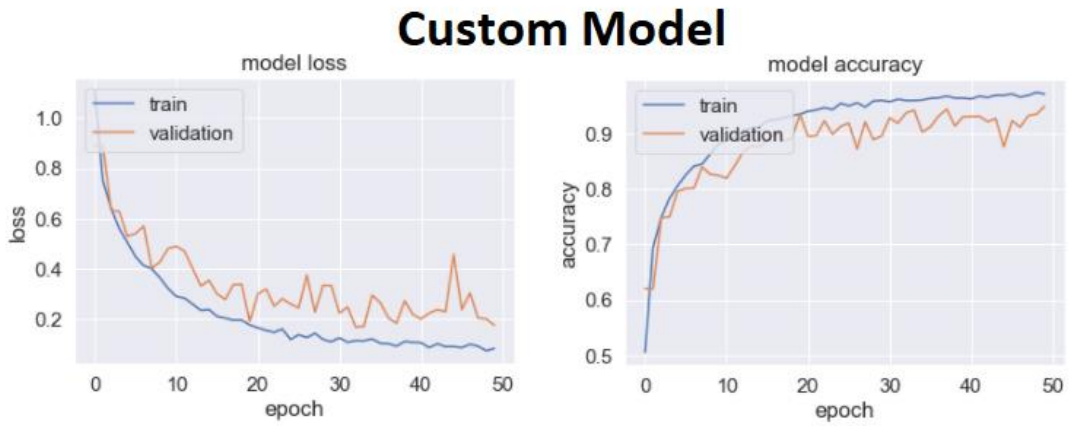

Figure 58 - The accuracy and loss of the Custom Model measured during training. Loss (Left) and Accuracy (Right).

By the end of the training process, the Custom model achieved the following scores: *Training **Accuracy**:* 97.1%, ***Validation Accuracy**:* 94.86%, ***Training Loss**:* 0.084, ***Validation Loss**:* 0.17.

Finally, of the three models the best performer was the VGG16 model. As mentioned in section 4.2.2. the VGG16 model was trained using the SGD optimiser instead of '*Adam*', and the number of epochs the model was trained over was only 30, almost half the amount of the other two models. The results of the training are shown below in *Figure 59,* and from observing these graphs it's clear that the training process was more consistent and efficient than the other two (efficient due to fewer required epochs).

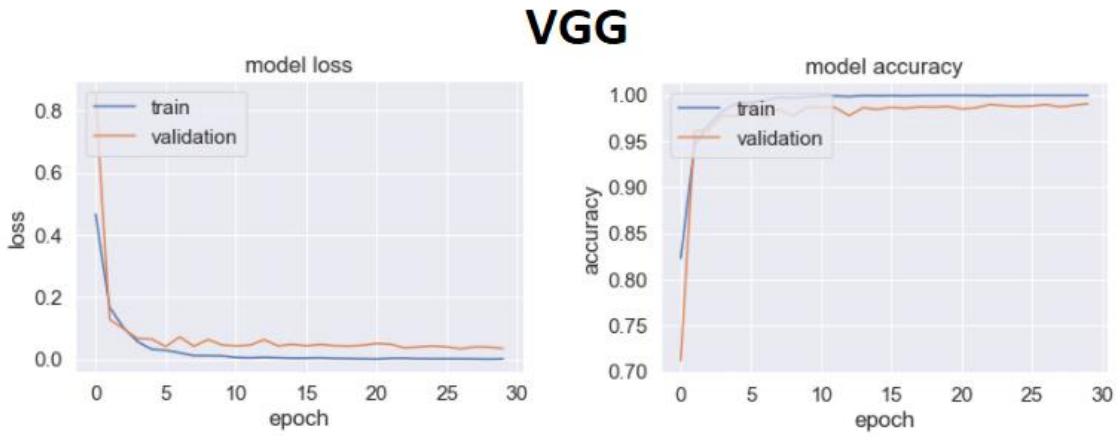

Figure 59 - The accuracy and loss of the VGG16 model measured during training. Loss (Left) and Accuracy (Right).

From the first epoch the model had a high accuracy and low loss and improved rapidly to 90% + accuracy. There was very little fluctuation during training and the both the validation and training metrics improved at a similar rate and ended up being very close to each other by the final epoch. At the end of training, the VGG model achieved the following impressive scores: ***Training Accuracy**:* 99.97%, ***Validation Accuracy**:* 99.1%, ***Training Loss**:* 0.0015, ***Validation Loss**:* 0.035.

Table 4 is a summary of training scores for all the models is shown below for side-by-side comparison:

Table 4 - Summary of all models training loss and accuracies collected during training.

	ResNet50	Custom Model	VGG16
Training Accuracy	89.55%	97.10%	99.97%
Validation Accuracy	90.44%	94.86%	99.10%
Training Loss	1.44	0.084	0.0015
Validation Loss	0.4	0.17	0.035

5.3. Further Model Evaluation on Holdout Data.

In this section I will show the results of each of the models tested on unseen data. The models will be evaluated from a multi-class and binary (tumour vs no tumour) perspective. The test data which has been used for these evaluations is a set consisting of 1307 images which have been held out of the training set. The distribution of classes in this set is as follows:

- Glioma: 340
- Meningioma: 312
- No Tumour: 273
- Pituitary: 382

The first model we to evaluate is the ResNet50 model, but before that the metrics being used to evaluate the model should be explained. The metrics returned by the function from the scikit-learn library in Python which has been used to generate the results include *Precision, Recall and F1-Score.* To understand each of these metrics, an understanding of TP, TN, FP, and FN is required. For simplicity, we will consider these in the context of a binary classification problem.

- **TP (True Positive)**
  - *Tumour present and presence is identified.*
- **TN (True Negative)**
  - *No Tumour present and no tumour presence is identified.*
- **FP (False Positive)**
  - *No tumour is present, but a tumour presence is identified.*
- **FN (False Negative)**
  - *Tumour present but no tumour presence is identified.*

Precision, defined in *Eq. 7*, is a measure of what proportion of positive identifications made by the model are actually correct – so in the binary context of this problem, how many of each of the predictions made for each class of tumour, or no tumour were correct.

Equation 7 - The Precision Metric

$$Precision = \frac{TP}{TP + FP}$$

For multi-class problems, "*Precision* is calculated as the sum of true positives across all classes divided by the sum of true positives and false positives across all classes." [50]

Recall, defined in *Eq. 8*, is a measure of what proportion of actual positives was identified correctly by the model. So again, in the binary context of the problem, what proportion of all say, *Gliomas*, were correctly identified as *Gliomas*.

Equation 8 - The Recall Metric

$$Recall = \frac{TP}{TP + FN}$$

And again, for multi-class problems, "*Recall* is calculated as the sum of true positives across all classes divided by the sum of true positives and false negatives across all classes." [50]

Lastly, *F1-Score* is a weighted average of precision and recall, with the best *F1-Score* being 1, and the worst being 0. *F1-Score* is defined below in *Eq. 9.*

Equation 9 - F1 Score Metric

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

ResNet50 Classification Metrics

Now that the metrics have been explained let's move onto the evaluations of the models on the unseen data, starting with ResNet50. *Figure 60* below shows the classification report and the associated confusion matrix for multi-class classification. In the confusion matrix, the Y-Axis represents the ground truths of the classes and the X-Axis represents the model's classifications.

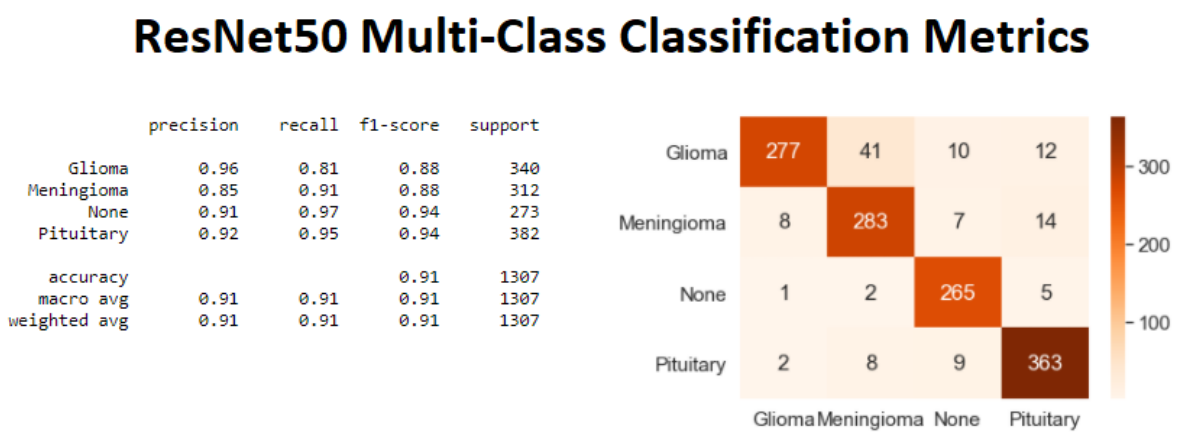

Figure 60 - ResNet50 Multi-Class Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

From looking at these stats we can see that the model performs best on classifying Pituitary Tumours and '*None*' but struggled more between the classifications of Gliomas and Meningiomas. From research summarised in the literature review regarding the differences in tumours and their common anatomical regions, it makes sense that the model would be better at diagnosing Pituitary tumours as they are more distinct in their location. From research it seems that Meningiomas are restricted to developing in the *meninges* which are a lining of the brain and spinal cord, which generally means they develop around what would appear to be the edges of a brain when viewed on a scan, Gliomas however aren't limited in the same way and can develop almost anywhere, so it could be possible that this is the reason the model struggled at distinguishing between the two more-so than other classes.

I think the most important stats to take away from this are the scores for *No Tumour*. The model has quite a few instances of predictions stating that there is no tumour when there is in fact a tumour, a false negative. Below, shown in *Figure 61* is the binary version of the classification on the test set between the class of No Tumour and Tumour which makes this error easier to see.

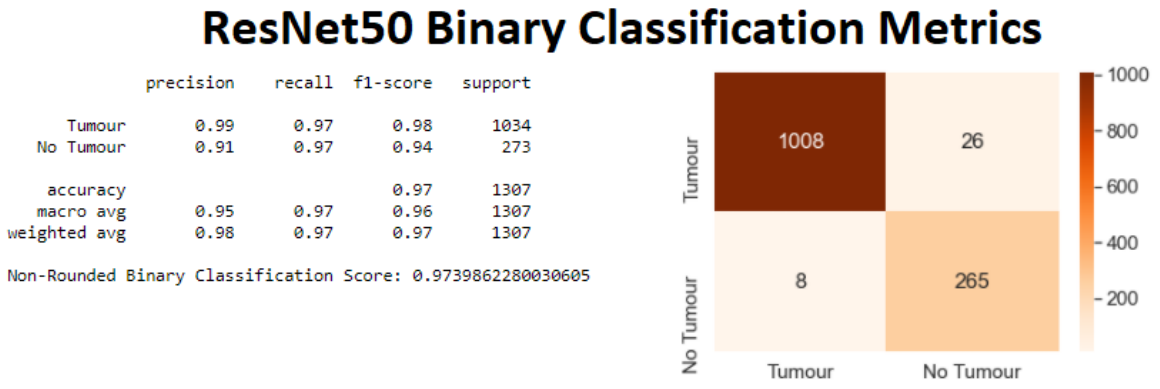

Figure 61 - ResNet50 Binary Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

Although turning this problem into a binary one makes the model seem like it is performing much better at 97% overall accuracy, the precision of No Tumour is only 91% and this should arguably be the easiest distinction for the model to make. I've highlighted this issue because in the context of medical diagnosis, a false negative is far more damning than a false positive. This model would

potentially be diagnosing a patient as healthy when in fact, they could have a life-threatening disease. The delay in correctly diagnosing a patient due to this error could mean the different between terminal and curable illness. An almost 1 in 10 probability of this happening is completely unacceptable.

Custom Model Classification Metrics

Moving on to the Custom Model, we see a fair improvement in the model's ability to distinguish between Gliomas and Meningiomas, which is expressed in the classification scores shown in *Figure 62*. The Custom Model made about half as many errors of the errors made by the ResNet50 model. The most significant improvements are the recall for Gliomas, improving 10%, and precision for Meningiomas around 6%, and we see that F1- Scores are improved across the board.

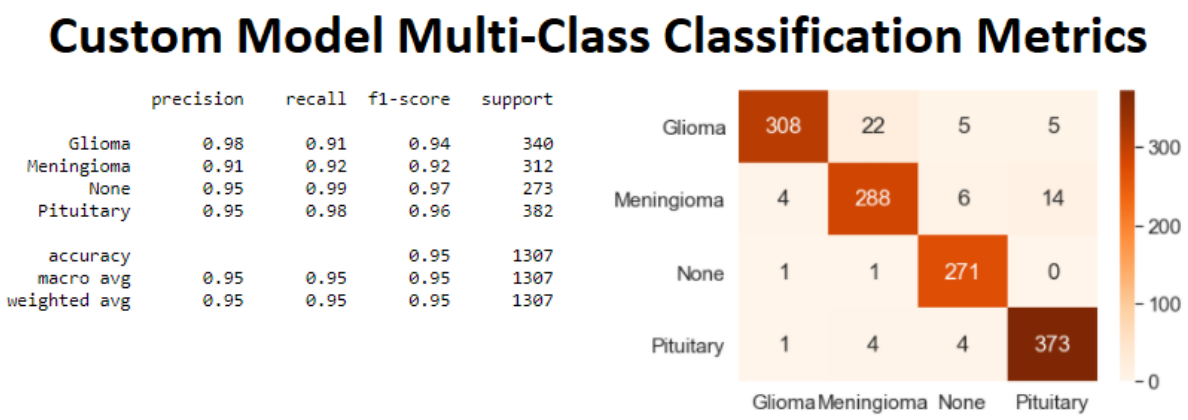

Figure 62 - Custom Model Multi-Class Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

Looking at the binary statistics, shown in *Figure 63* we can observe that there has been an improvement in the precision for classifying no tumours, but unfortunately the model still has a false negative rate of 5.5%, which isn't adequate.

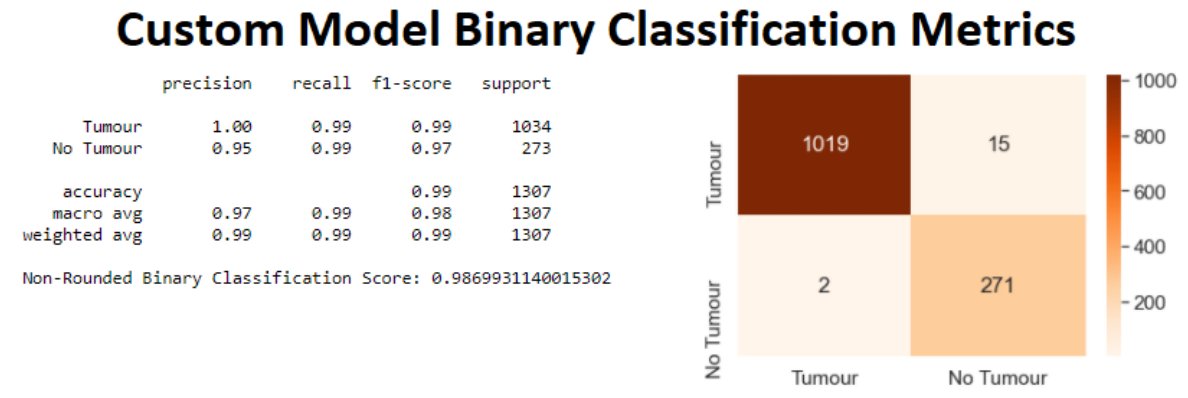

Figure 63 - Custom Model Binary Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

VGG16 Classification Metrics

Finally, the VGG16 model was tested and showed the most promising results of the three models, in both multi-class classification as well as binary. Below in *Figure 64,* we see the multi-class statistics. Immediately we can see that there are very few misclassified tumours, and the distinction between Gliomas and Meningiomas, although errors are still present, is much lower than both other models. The recall for Gliomas has improved 6% over the Custom Model and sits at 97% which is the joint-lowest score for all the metrics measures on this model.

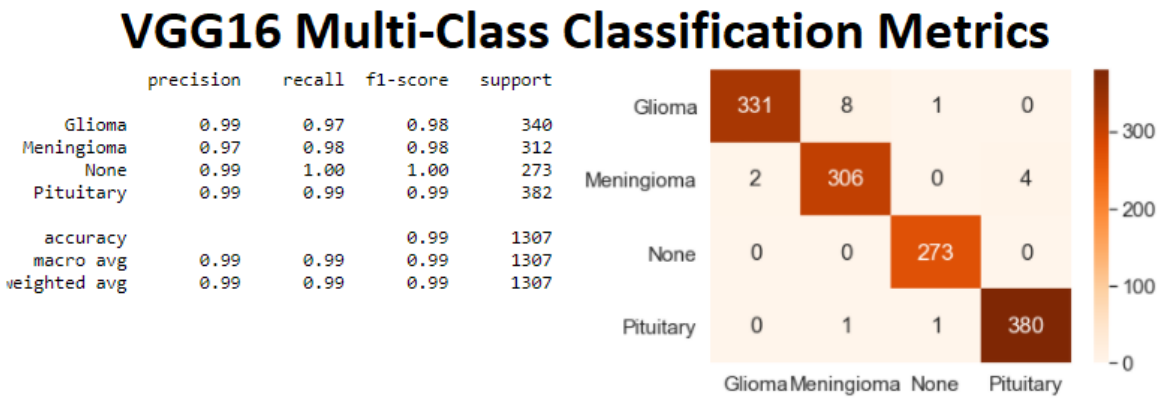

Figure 64 - VGG16 Multi-Class Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

The most significant improvement that this model has made over the other two, and the most important in the context of medical diagnosis as explained before, is the reduction in false negatives made by the model. Looking at the measurements in *Figure 65,* the results show that VGG16 has 100% recall and precision on tumour classification, 100% recall of healthy brain scans and a 99% precision accuracy of healthy brains, the overall binary classification score for this model sits at 99.85%, with only two classification errors being made in the binary context.

Although there are still two false negatives present and it's possible for an improvement to be made, there is less than a 1% chance for this module to produce a false negative result. This is the most promising result of any model tested in this project.

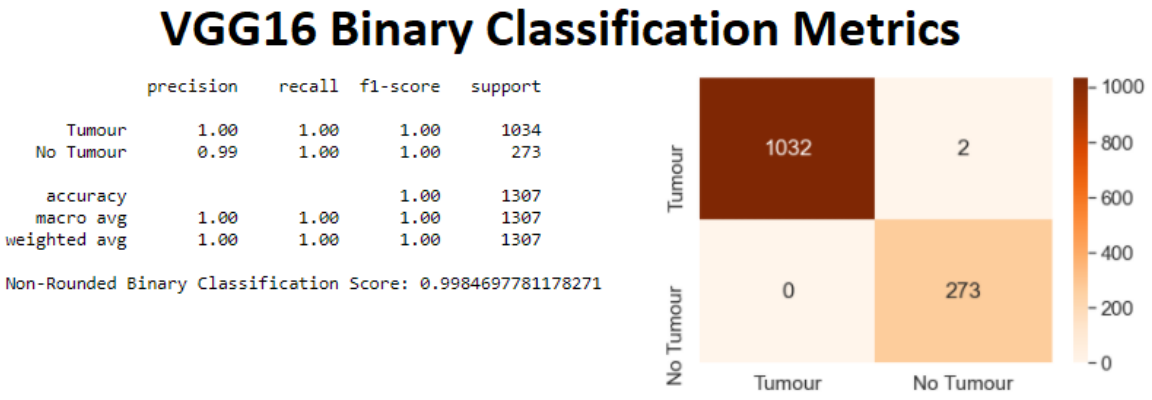

Figure 65 - VGG16 Binary Metrics – Showing Precision, Recall, f1-score, and the associated confusion matrix.

This concludes the evaluations of the machine learning models. The next section will explore the implementation of the model explainability method which will be applied to these models in order to explain their classifications.

6. Implementing Machine Learning Model Explainability

6.1. Introduction

Local Interpretable Model-Agnostic Explainability, or LIME, is the explainer of choice used for this project. As covered in the Literature Review (Section 2.), the method of explaining model was introduced by Ribeiro et al in 2016 [6]. In this section the implementation of LIME used in this project will be demonstrated using snippets of code and visual examples of segmented images and other important components. Some code used to implement LIME has been derived from the work of Cristian Arteaga [32]. The results of LIME being used to explain classifications by the models in this project will be covered in Section 7.

6.2. Implementation

All the functions written for LIME in this project are in the Python programming language and are contained in a script named 'LIME.py', which has then been imported into a Python notebook through Jupyter so the experiments can be run interactively cell-by-cell. This makes it easy to visualise and demonstrate the outputs from the LIME functions and the explainer's results.

6.2.1. Loading and Segmenting Images for Classification

Just like data pre-processing for model training, the images need to be loaded into Python variables so they can be formatted appropriately to be compatible with the machine learning models. There are three different function in the LIME script which have image loading functionality, all of which use *Pillow* (PIL) to open the image and NumPy to format them.

The reason for there being three distinct functions all of which load in an image is that the scikit-image python library which contain the segmentation functions require the image to be in an RGB format for the segmentation masks to be created, and there are two of these functions which produce slightly different segmentations, which will be explained below. The third function is to load in an image and convert it into greyscale format so it can be input into the machine learning models for classification.

The first function for loading images is named *generateGrayImage,* and takes the system path of the image, as well as the desired image dimensions as arguments. This function opens the image and then uses the OpenCV library to convert the image from its default RGB into Grayscale. This returns an image array which is ready to be passed into a model for classification, as well as being the base image for a segmentation mask to be projected onto, more on this later. The Python code implementing this function is shown below in *Figure 67.*

```
def generateGrayImage(image, dims):

  #Open the image, resize, normalise and convert to Grayscale.
  img = np.array(Image.open(image))
  new_img = cv2.resize(img, dsize=dims,interpolation=cv2.INTER_CUBIC).astype(np.float32)
  new_img = np.asarray(new_img)
  new_img /= 255
  new_img = cv2.cvtColor(new_img, cv2.COLOR_BGR2GRAY)

  #Create NumpyArray and modify shape for model input.
  to_pred_Gray = []
  to_pred_Gray = np.asarray(to_pred_Gray)
  to_pred_Gray = np.append(to_pred_Gray, new_img)
  to_pred_Gray = to_pred_Gray.reshape(1,256,256,1)

  return to_pred_Gray
```

Figure 66- generateGrayImage Python function defined to open an image and convert it to grayscale.

The next two functions are each responsible for generating a superpixel masks for the images which are to be classified and explained. Each one uses a different segmentation algorithm from the scikit-image library. The difference in the outputs of these functions is described later.

The first function, called *generateSuperpixels* (Python code shown in *Figure 68*) loads in an image to a NumPy array, and then creates a segmentation mask using the *QuickShift* algorithm [45].

```
def generateSuperpixels(image, dims, kSize, mDist, ratio):
  #array to hold image
  predictArr = []

  #Open image and resize it
  img = np.array(Image.open(image))
  img = cv2.resize(img, dsize=dims,interpolation=cv2.INTER_CUBIC).astype(np.float32)
  img = np.asarray(img)
  img /= 255

  #Mode image into Numpy array with appropriate dimentions
  predictArr = np.asarray(predictArr)
  predictArr = np.append(predictArr, img)
  predictArr = predictArr.reshape(1,256,256,3)

  #Generate the superpixels over the image
  superpixels = skimage.segmentation.quickshift(predictArr[0], kernel_size=kSize, max_dist=
mDist, ratio=ratio)

  #Assign number of superpixels generated to variable
  num_superpixels_brain = np.unique(superpixels).shape[0]

  #Show image with superpixel mask
  plt.imshow(skimage.segmentation.mark_boundaries(predictArr[0], superpixels))
  print(f'Number of superpixels generated: {num_superpixels_brain} \nMask:')

  return num_superpixels_brain, superpixels
```

Figure 67 - The Python implementation of the generateSuperpixels function defined to create the segmented tumour images using the Quickshift Algorithm.

The *generateSuperpixels* function requires arguments for the image location on the local drive, and the dimensions for its formatting, and it also accepts the following three parameters which are for Quickshift algorithm configuration:

- **Kernel Size**
  - *"Width of Gaussian kernel used in smoothing the sample density. Higher means fewer clusters." [49]*
- **Max Distance**
  - *"Cut-off point for data distances. Higher means fewer clusters." [49]*
- **Ratio**
  - *"Balance's colour-space proximity and image-space proximity. Higher values give more weight to colour-space." [49]*

The second function, called *generateSuperpixelsSLIC,* is essentially the same function as *generateSuperpixels* but instead of using *Quickshift* it uses a scikit-image algorithm called *SLIC* which also generates a superpixel mask on an image, but segments an image in a slightly different way. SLIC accepts many parameters but the two used for this project were:

- **Number Segments**
  - *"The (approximate) number of labels in the segmented output image." [49]*
- **Sigma**
  - *"Width of Gaussian smoothing kernel for pre-processing for each dimension of the image. The same sigma is applied to each dimension in case of a scalar value. Zero means no smoothing." [49]*

Both functions described above output two variables. The first is an integer which contains the number of superpixels generated for the image, and the second is an NumPy array indicating segment labels. *Figure 68* below shows each of these segmentation functions applied to the same image to demonstrate the differences in the super pixel shapes. SLIC produces a mask in which the super pixels are more regular in size, and oriented in more of a grid fashion. Quickshift divides the superpixels into patches of similar colours and therefore the super pixels appear slightly more random.

The reason why I wanted to include two different segmentation methods in this project is because the nature of Quickshift creating irregular boundaries of various sizes within the brain regions of an image.

These regions which are being separated have potential importance in the context of each other, and a CNN model may use the boundary between them to discern a specific feature based on the difference in pixel intensities. Having these regions separated from surrounding pixels may deny the model important context for its classification. Conversely, SLIC is slightly more inform in terms of superpixels sizes and shape, and less often separates tumours from surrounding tissue. This can be seen in *Figure 68*. I thought it would be interesting to observe the difference in the quality of explanations using these two different segmentation methods.

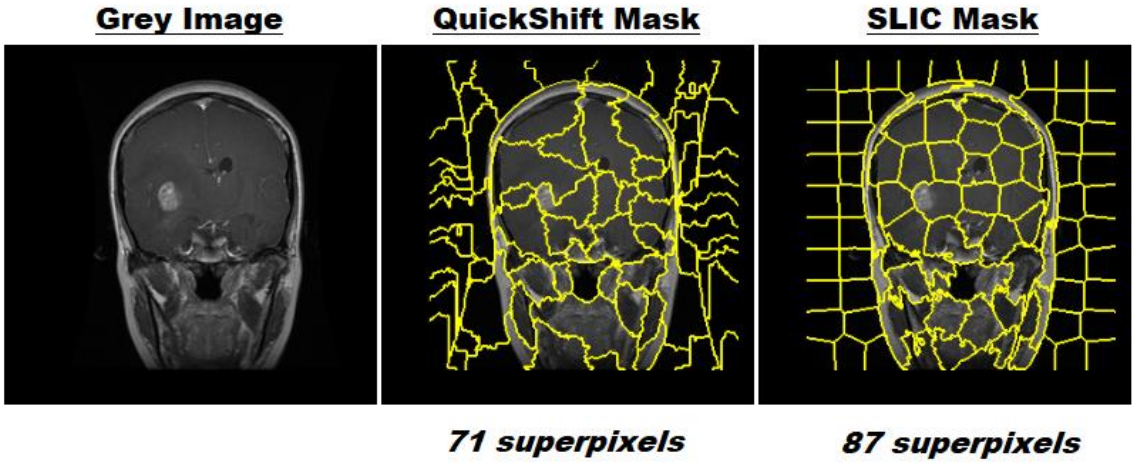

Figure 68 - Segmentation Algorithm Examples

6.2.2. Generating Image Perturbations & Perturbation Distances

Now that we have the segmentation masks for the images, we can use them to create perturbations of the greyscale image to them be passed into the model for classification. In this project, this process is handled through the Python functions *perturbate* and *generatePerturbations.* Together, these functions produce the images shown in *Figure 69.*

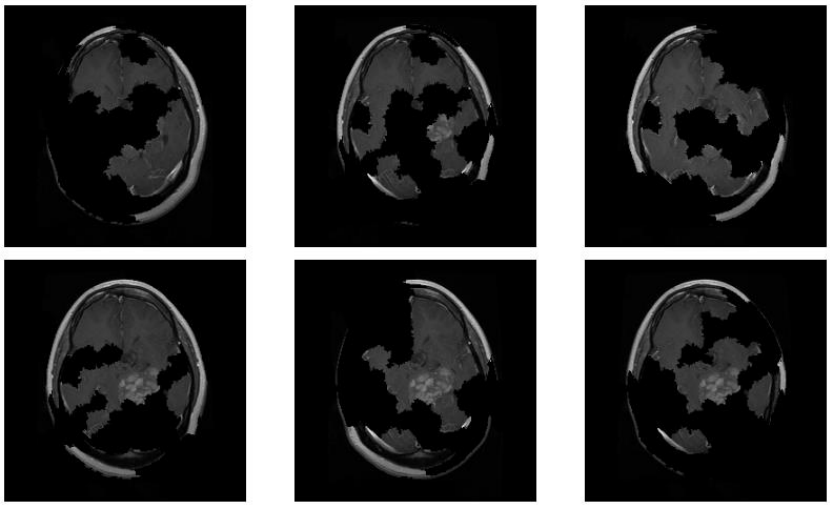

Figure 69 - Image Perturbation Examples

The *perturbate* function (the Python code of which is shown in *Figure 70*) is responsible for turning superpixels on and off for a specific perturbation. To explain this function, let ***N*** be the number of superpixels generated for an image, and ***SuperPixels*** be the superpixel labels array which was returned by the segmentation functions in the previous section. This ***SuperPixels*** array has the same shape as the base image, 256 x 256.

```
def perturbate(image, perturbation, superpixels):
      #Obtain an array of indexes which correspond to each one in the binary perturbation array.
      active_pixels = np.where(perturbation == 1)[0]
      #Create a mask of the same size of the original image, with all values set to 0.
      mask = np.zeros(superpixels.shape)
      #For each of the active_pixels:
      for active in active_pixels:
        #Change each value in mask to 1 if corresponding superpixel is present in the active_pixels array.
        mask[superpixels == active] = 1
      #Create a copy of the original greyscale image
      perturb_image = copy.deepcopy(image)
      #Multiply the mask by the new copied image. Any pixel multiplied by a 1 in mask will retain it's value, otherwi
se
      #it is turned off.
      perturb_image = perturb_image*mask[:, :, np.newaxis]
      #return the peturbed image
      return perturb_image
```

Figure 70 - The Python function defined to create the perturbated images which are showed in Figure [70].

For an individual perturbation, a binary array of length ***N*** is passed into the function as an argument, and another array variable is created which contains the indexes corresponding the '1' values in the binary index, this array will be referred to as the ***active_pixels*** array. For clarification an example of

this is shown in *Figure 71* below. These 1s in the binary array indicate which superpixels will remain active and which will be turned off.

```
Binary Array:
[1 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0]

Active Pixels in Binary Array:
[ 0  2  3  4  6 11 13 14]
```

Figure 70 - Active Pixels Example

A **mask** is then created which is a 2D NumPy array with all values set to 0 and with the same dimensions as the image to be perturbed. The **active_pixels** array is then looped though, and for each value ***V*** in **active_pixels**, an image-wide comparison is made for every value in the **mask** between the ***SuperPixels*** array and ***V.*** Each pixel index for which this comparison is *true* will be set to a 1 in the **mask** variable, otherwise it will remain as 0.

A copy of the original greyscale image Is made, and then every value in the image copy is made against the corresponding value in the **mask** variable. Any value in the copied image which is multiplied again a 0 in the **mask** is 0, and any value multiplied by a 1 in the **mask** retains its original value. What we are left with and is returned by the function is a perturbated image, where half of the superpixels will be turned off and half will remain on.

The second function, *generatePerturbations* is responsible for creating the desired amount of perturbated images, by defining a binary array for each of the desired perturbations, in which 50% of the values are set to 0. For each binary array created the *perturbate* function is invoked a perturbed image is produced. The function then uses a machine learning model to predict the classification of the perturbed image, which is stored in an array and returned by the function along with each perturbated image, as well as the binary arrays. The Python code for this function is shown in *Figure 72*.

```
def generatePertubations(num_pertubations, image, superpixels, num_superpixels, model):

  #Array to store all pertubation perdictions
  pertubationPredictions = []
  #Array to store perturbed images
  perturbed_Images = []
  #Create the binary arrays which correspond to each of the perturbations
  perturbations = np.random.binomial(1, 0.5, size=(num_pertubations, num_superpixels))
  #Loop through each of the binary arrays created
  for perturbation in perturbations:
    #Create a perturbed image
    perturb_image = perturbate(image, perturbation, superpixels)
    #Append new image to an array
    perturbed_Images.append(perturb_image)
    #Collect classification result from model of the new image
    prediction = model.predict(perturb_image[np.newaxis, :, :, :])
    #Store the result of the models classification on the perturbated image
    pertubationPredictions.append(prediction)
  #Return the predictions, perturbated images and the binary arrays.

  return np.asarray(pertubationPredictions), np.asarray(perturbed_Images), perturbations
```

Figure 71 - Python implementation of the generatePerturbation function which is defined to create a series of perturbated images.

Finally, now the perturbations have been created we can compute the distances between the original image and each of the perturbed images. The Python function defined to handle these computations is defined in *Figure 73*.

```
def generateDistances(numSuperpixels, pertubations):

  #Create an 'original image' with the length of superpixels, all turned on.
  original_image = np.ones(numSuperpixels)[np.newaxis, :]
  #Use the sklearn library to generate the Euclidean distance between original image and each perturbation.
  distances = sklearn.metrics.pairwise_distances(pertubations, original_image, metric='cosine').ravel()
  #Return the distances
  return distances
```

Figure 72 - The generateDistances function defined in Python to calculate the distances between an original image and its perturbations.

The *generateDistances* function first defines a NumPy which as the length equal to the number of super pixels, and each value in the array is a 1 to simulate that pixel being enabled. The distances for

each perturbation to this original image array are then calculated using the sklearn function *pairwise_distances*, which has been set to calculate the cosine distances (*Eq. 10*).

Equation 10 - Cosine Similarity used to calculate distances

$$\cos(A,B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

$$Where,$$

$$A_i \ \& \ B_i \ are \ components \ of \ vectors \ 'A' \ and \ 'B' \ respectively$$

$$\|A\|\|B\| = cross \ product \ of \ the \ two \ vectors \ 'A' \ and \ 'B'$$

$$n = Number \ of \ components \ in \ vector$$

The *generateDistances* function returns an array of cosine distances of the length equal to the number of perturbations, an example of these distances is shown in *Figure 74*. These distances are then used to calculate weights for the linear model, this is detailed in the next section.

```
Perturbation Distances:

[0.31824584 0.24008896 0.26841865 0.24941337 0.24941337 0.2978905
 0.28793101 0.30799332 0.2978905  0.2978905 ]
```

Figure 73 - Example of perturbation distances

6.2.3. Fitting Linear Model and Showing Explanations

At this stage everything is ready to start generating weights and fitting a linear model which is used to provide the *locally faithful* model interpretability in LIME. The first step is to calculate the sample weights which are to be used in the fitting of the linear model. The weights are calculated using the distances generated from the previous function. The formula used to calculate the weights is shown in *Eq. 11,* and the curve plot of the formula showing different kernel widths is shown in *Figure 76*. This formula is used to map a distance to a value between 0 and 1, and the kernel width is a variable which can be changed depending on how you want values to be mapped. As the cosine distance was used to calculate distances in this project, a low kernel width (0.25) can we used, as the distances are already between 0 and 1.

Equation 11 - The formula used to Calculate Weights

$$y = \sqrt{\frac{-x^2}{e^{(w^2)}}}$$

$$Where,$$

$$y = calculated\ weight$$

$$w = kernel\ width$$

$$x = cosine\ distance\ value$$

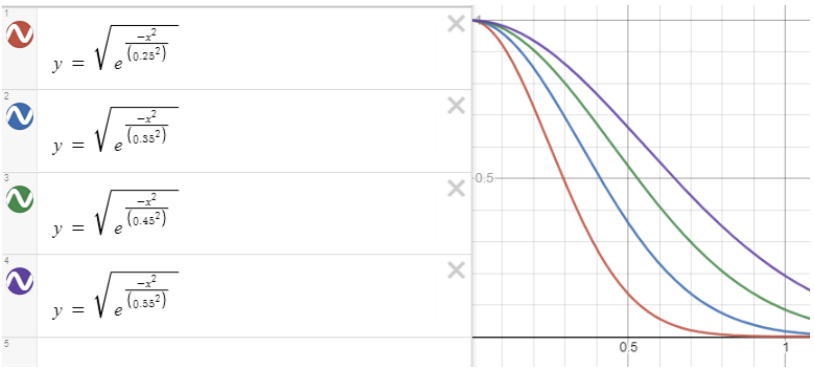

Figure 74 - Plotted weights distribution with different kernel weights

With the weights calculated the linear regression model can be fit. This is done so using the sklearn library once again, and using the *fit* function associated with it. Three parameters are passed into the model, the **X** values, which are the binary perturbation arrays which were generated by the *generatePerturbations* function, the **Y** values which are the prediction values output by the chosen model for each of the perturbed images, but it's important that we only include the probability scores for the class we want to explain. As there are four classes in this problem, each perturbation prediction is an array of length four, and only the value which corresponds to the desired class is required. The last parameter passed into the *fit* function are the calculated weights.

With the model fit, the coefficients are produced and stored in an array. Each of these coefficients corresponds to one of the superpixels in the image. The Python function which fits the linear model and returns these coefficients is called *fitLinModel* and is shown in *Figure 76.*

```
def fitLinModel(k_width, distances, class_to_explain,
                pertubation_predictions, perturbations):

  #Define the kernel width
  kernel_width = k_width
  #Calculate the weights for the model
  weights = np.sqrt(np.exp(-(distances**2)/kernel_width**2))
  #The index of the class to be explained
  class_to_explain = class_to_explain
  #Create a linear regression model
  simpler_model = LinearRegression()
  #Fit the model using the perturbations, the predictions associated with those
  # predictions and the calculated weights.
  simpler_model.fit(X=perturbations, y=pertubation_predictions
                    [:,:,class_to_explain], sample_weight=weights)
  #Record the coefficients from the model
  #Each coefficient in the model represents a superpixel
  coeff = simpler_model.coef_[0]
  #Return the coefficients

  return coeff
```

Figure 76 - Python implementation of fitLinModel, which is a function defined to create and fit a linear model in order to obtain the coefficients associated with each superpixel.

To show an explained image we define the top features ***N,*** which is an integer of how many features we want to see. In the *showExplanation* function (Python code shown in *Figure 77*), the coefficients are sorted into importance order and the top ***N*** coefficient indexes are stored in a new variable. A new mask is created with the super pixels corresponding to these top ***N*** features are set to 1, and all other a 0.

```
def showExplanation(num_top_features, coeff, num_superpixels, gray_image, superpixels):

  #Sort the coefficients and keep the top number of features defined
  top_features = np.argsort(coeff)[-num_top_features:]
  #Create a new mask
  mask = np.zeros(num_superpixels)
  #Turn on the super pixels associated with the top coefficients
  mask[top_features] = True
  #Show the explained image
  plt.imshow(perturbate(gray_image, mask, superpixels))
  #return the explained image
  return perturbate(gray_image, mask, superpixels)
```

Figure 77 - showExplanation function Python implementation which calculated the top N features in an image and displays an image showing those features.

Finally, we invoke the *perturbate* function once again, this time passing the original grayscale image in, with the new mask and the superpixels. The result produced is an interpretable image of the most

important super pixels in an image for a specified class. An example of an explanation for a Glioma showing the 10 most important features is demonstrated in *Figure 78* below. This concludes the implementation of *LIME*. In the following section, the explainer will be applied to a handful of images and the results will be observed in an attempt to understand the classifications of the machine learning models.

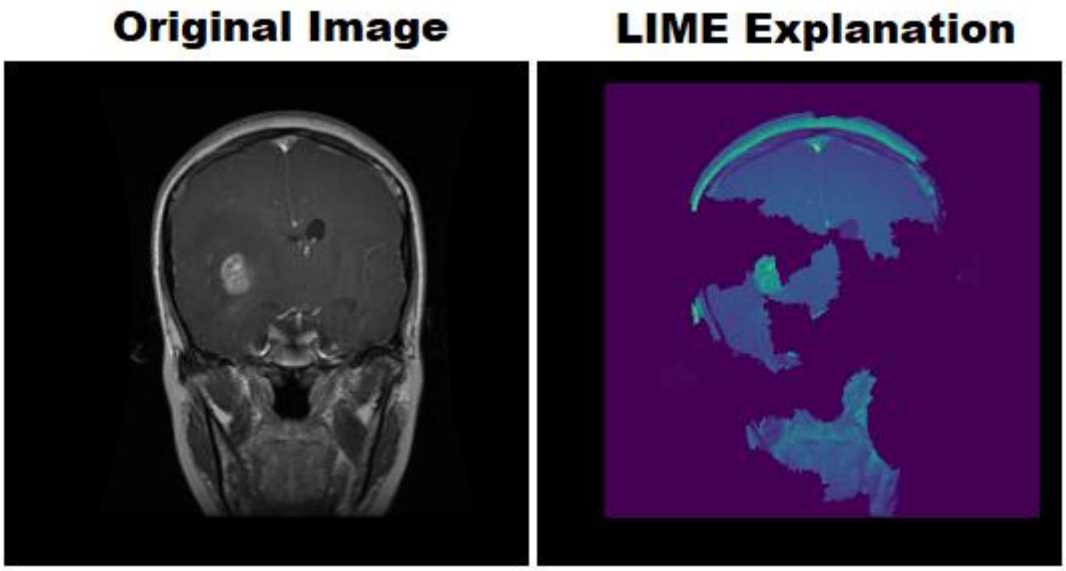

Figure 75 - LIME explanation example.

7. Evaluation of the Machine Learning Explainability Method

7.1. Introduction

To test the implementation of *Local Interpretable Model Agnostic Explanations*, a total of three images per class have been reserved from testing and validation, one image for each of the *Axial*, *Sagittal* and *Coronal* planes (discussed in section 2.3.1). The purpose of examining these explanations isn't purely to see if the models land on the tumour in the chosen number of top features every time, to then write off as a success, but is to understand how the model views the brain, what regions it gives importance to, and to see if we can discern any model bias which would make the model untrustworthy.

In this section, a comparison of each image, classified by each model, with both segmentation algorithms will be made. The testing has been separated into a section per class to try and interpret the top features output by the explainer and understand why those features are so important to each model's classification. It's worth noting before delving into the interpretable images that even if a model has erroneously classified the image, the explainer is still configured to shown top features pertaining to the true class, however low the confidence of the model may be.

7.2. Model Explanations of Glioma Image Classification

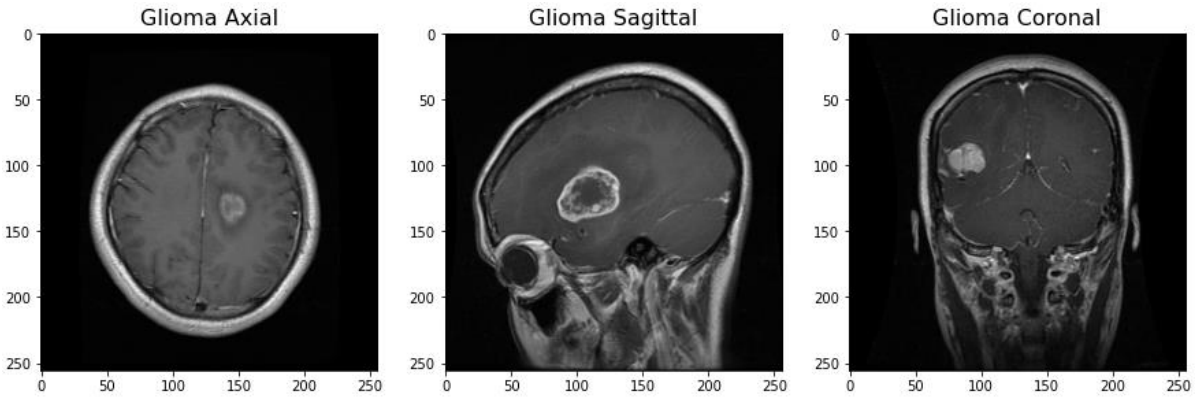

Figure 76 - Glioma Testing Images

The three brain scans with Glioma Tumours present which are going to be used for these LIME explanations are showed in *Figure 79* above. For each of the LIME explanations, for each image, I have chosen to display the top 15 features, which I have deemed to be the sweet spot between too many segments to distinguish which are the important ones, and too few to truly understand what the model is looking at.

Glioma Explanations of the Axial Plane

Starting with the Axial plane, in *Figure 80* we see the results from each of the models and the different segmentations algorithms. The only model which correctly classified this image was the VGG16 model with approx. 98% accuracy. Both other models classified the image as a Meningioma. I have mentioned before in this report that Glioma and Meningioma distinction may be a problem, and this is evident here. The explained VGG16 images show the tumour in both versions of segmentation. It also appears that a lot of the top features in these images are regions of the brain around the edges of the skull. I think it's possible that the model is checking for meningiomas which are usually present in these regions, and upon not finding one, it is looking towards the inner regions and identifies a tumour as a Glioma. This is pure speculation, but from my understanding of common tumour regions, it does not seem too farfetched.

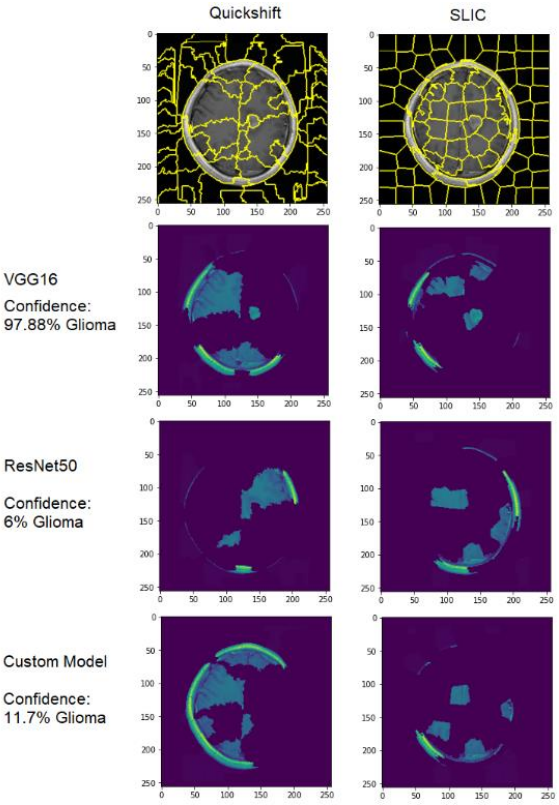

Figure 77 - Glioma Axial Explanations

Another possibility which could explain the importance of these regions of the brain is that the model is identifying other abnormalities caused by brain tumours, such as midline shift, or areas which appear to have been pushed out of position due to increased pressure in the skull. Both the ResNet50 model and Custom model completely fail to identify the tumour in the top 15 features of the image, but also seem to emphasise importance of these edge regions of the brain.

Glioma Explanations of the Sagittal Plane

For the sagittal plane, for which the results are shown in *Figure 81,* all three models correctly classify the glioma, with the VGG16 model being 100% confident and showing the tumour in the top 15 features of the scan for both segmentation types. The custom model is also successful in identifying the tumour, but only in the *SLIC* segmentation version of the scan. The ResNet50 model fails to show the tumour in either. Again, it seems that outer regions of the brain are emphasised. In the ResNet50 and Custom Model interpretations, the eye socket seems to appear in the top features. I wonder if this provides context to the brain about what plane of scan is being observed, or it is a form of bias.

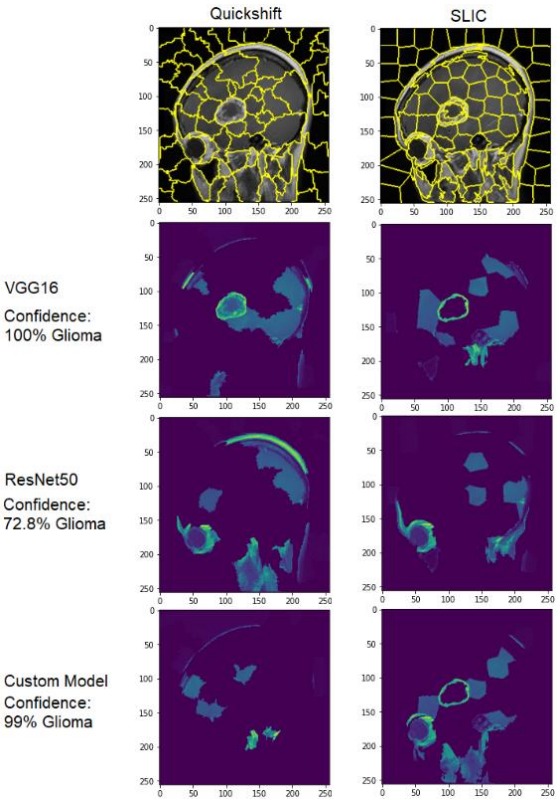

Figure 78 - Glioma Sagittal Explanation

Glioma Explanations of the Coronal Plane

Finally, for the results of the Coronal plane of a Glioma scan, which are shown in *Figure 82,* all three models correctly classify the image, and all with very high confidences, but in none of the scans for any of the models is the tumour present in the top 15 features. Increasing these top features does eventually reveal the tumour in the scans, but what's important is that the tumour does rank in the top 15 and understanding why this is the case.

It is possible that there is some bias in the model, such as the bias shown in *Figure 27* - the *Huskies vs Wolfs* example in the LIME literature review. I prefer to think that what I hypothesised in the review of the glioma axial results is the case, and that the model may be searching common tumour hotspots or looking at other abnormalities caused by tumours is the reason for these being the top-ranking features. One thing I do find curious, is that for the Custom Models classification of Quickshift, no regions of the brain matter appear in the top features, only the neck region, where a brain tumour would seldom, if ever appear. This could indicate some learned bias in the model.

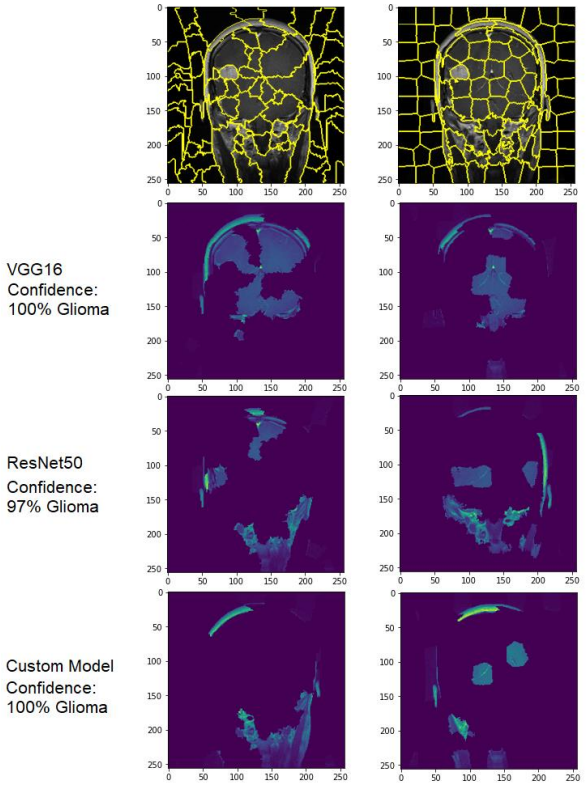

Figure 79 - Glioma Coronal Explanation

7.3. Model Explanations of Meningioma Image Classification

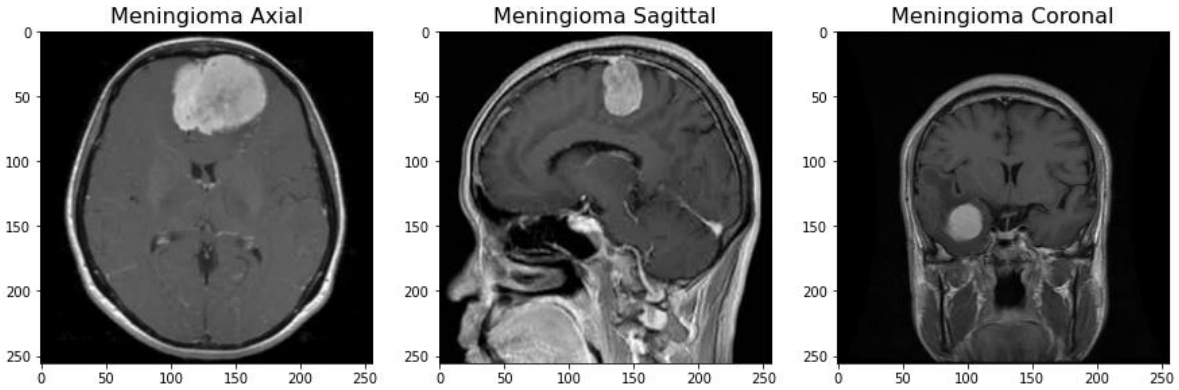

Figure 80 - Meningioma Test Images

Moving on now to the meningioma scans, and just is the case for the glioma explanations, three images of the brain will be experimented with, on for each plane where a meningioma tumour is present. *Figure 83* shows the images which are to be used for the LIME explanations, where again the top 15 features will be shown.

Meningioma Explanations of the Axial Plane

The explanations for the axial plain which are shown in *Figure 84* appear very successful in identifying the tumour region in every image, especially so in the Quickshift versions for VGG16 and ResNet50s classifications.

In the section where I reviewed gliomas in the axial plane, I hypothesised that the importance of the features showing the edge regions of the brain for glioma classification could be that the model checks the common regions in which a meningioma would appear, and the absence increases the model's confidence. It seems that the opposite may be happening here, for meningioma classification, the model seems to emphasise some inner regions, which can be seen heavily in the Custom Models explanations. In the data which the model has learned from, gliomas may be common in these regions, which could be an explanation as to why these regions are important to meningioma confidence.

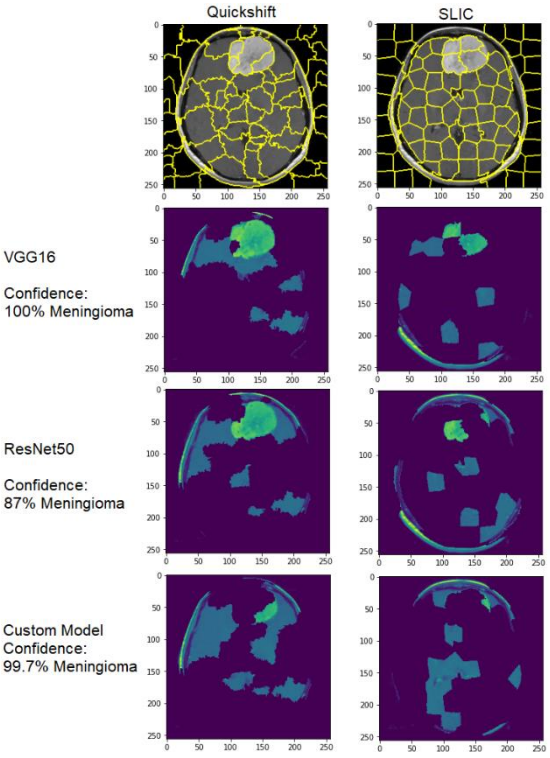

Figure 81 - Meningioma Axial Explanations

Meningioma Explanations of the Sagittal Plane

Figure 85 shows the explainers results for the sagittal plane for the meningioma class, and from looking at the VGG16 results, it seems clear why the model has classified the scans as meningioma, as the top features show the tumour appearing in those top regions of the brain. The other two models tell a different story. As a matter of fact, ResNet50 classified this image as a pituitary tumour, despite identifying the meningioma in the top features of SLIC. The reason for this being isn't clear to me. ResNet50 was the least accurate model of the three, and by looking at the explanations it seems that the model has identified illuminations of the pituitary gland, in both Quickshift and SLIC which could explain this result. The Custom Model which was less confident than VGG16 identifies the tumour in SLIC, but still seems to favour other more inner regions of the brain in its top features.

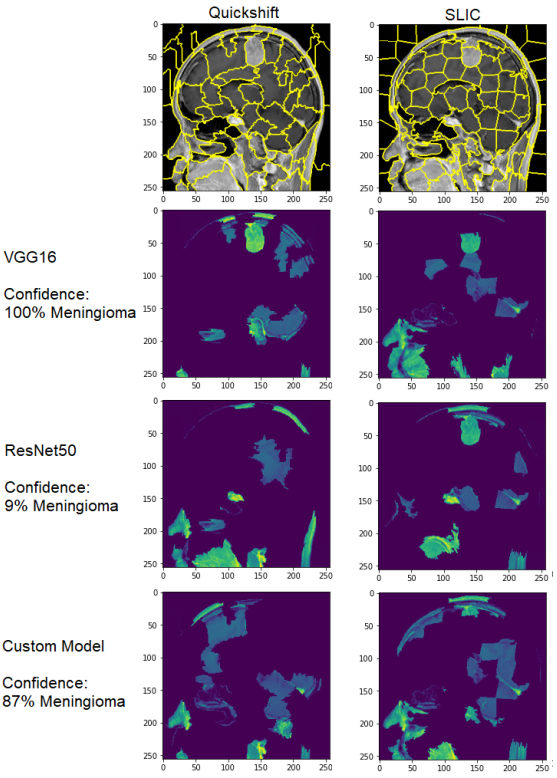

Figure 82 - Meningioma Sagittal Explanations

Meningioma Explanations of the Coronal Plane

The explanations for the coronal plane meningioma which are shown in *Figure 86,* seem very convincing and leaves very little to talk about. All models correctly classify the image, and all models identify the tumour in the top features, except for the Quickshift explanation for the Custom Model.

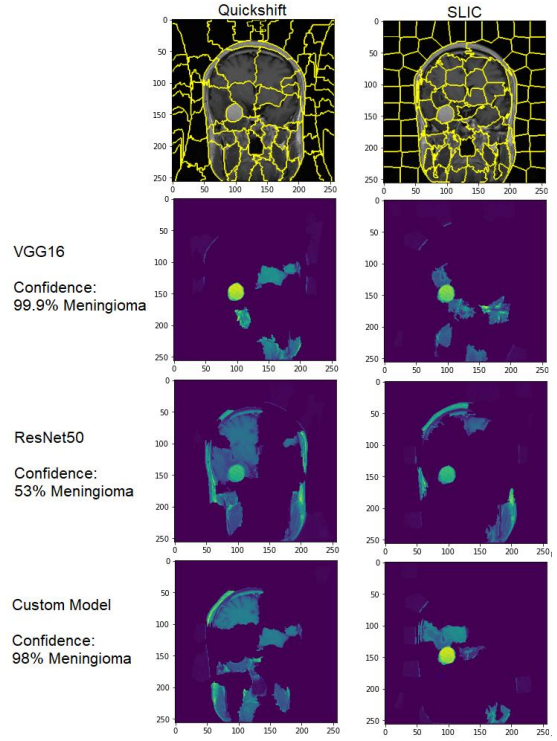

Figure 83 - Meningioma Coronal Explanations

7.4. Model Explanations of Pituitary Image Classification

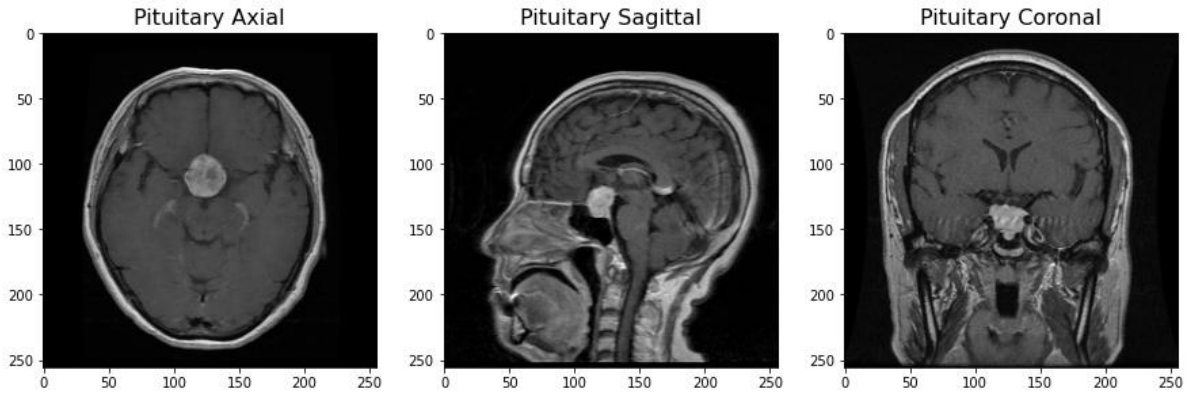

Figure 84 - Pituitary Test Images

Now for the final of the classes with tumours present, we will explore the explanations from LIME pertaining to the pituitary tumour. Again, three images, one from each plain will be experimented with, and the top 15 features will be displayed. The images to be tested are shown above in *Figure 87.*

Pituitary Explanations of the Axial Plane

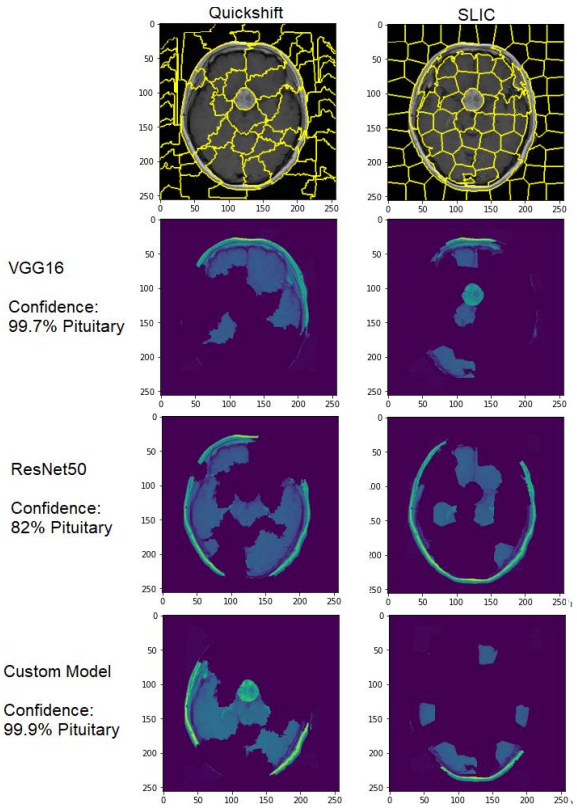

Figure 85 - Pituitary Axial Explanations

Back to the axial plane but this time for a pituitary tumour (*See Figure 88*) and the results seem to be consistent with what we have seen from the axial results of the other two classes. All models correctly classify the tumour with high confidences, but the tumour itself is only present in VGG16 SLIC and Custom Model Quickshift. Just as I have hypothesised with the axial plane for the other tumours, I believe the models classifications rely heavily on the regions in which the tumour is present and that could explain why the top features always include outer / inner regions of the brain where there is no tumour, in a kind of elimination process. In the context of pituitary tumours, I think the evidence here supports this argument even more so. In both other planes (*Sagittal and Coronal*), pituitaries are fairly obvious in their location, but for axial and the nature of MRI slices, seeing the pituitary tumour isn't guaranteed, as the specific slice may not be showing a deep enough layer of the brain for the pituitary gland to even be visible, due to the gland's anatomical location in the skull. For this reason, a tumour being in the centre of the brain on the scan may not be enough evidence to classify it as pituitary immediately. I must stress that this is speculation, and just one of many reasons that could explain the explainer's behaviour. It would be very interesting to have a discussion with a radiologist to either validate or invalidate my hypothesis and understanding of the brain's anatomy.

Pituitary Explanations of the Sagittal Plane

Looking now at results of a pituitary tumour classification from the sagittal plane, which are shown in *Figure 89*, we see a similar situation to the axial results. The VGG16 and Custom Model seem to put importance onto the outer and inner brain space before showing the tumour, which is not present in the top 15 features. Strangely, the ResNet50 model identifies the tumour in the top features, but the image is classified as a meningioma, with only 39% confidence of it being a pituitary. I've compared this image to others of the sagittal plane used in training and it does not seem to be unusual, so I think the incorrect classification must just be due to the lower overall performance of the model. The lower region of the back of the head, above the neck also seems to be an important feature. Maybe this area provides context to the model about which plane of the brain the image is showing.

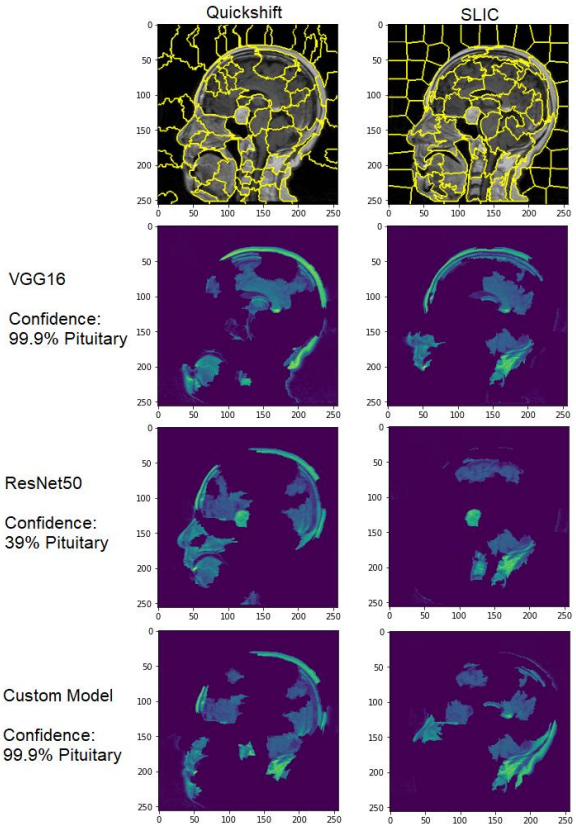

Figure 86 - Pituitary Sagittal Explanations

Pituitary Explanations of the Coronal Plane

Finally, the last of the pituitary experiments are we are observing the classifications of the coronal plane, results of which are shown in *Figure 90.*

I fail to identify anything that is new and significant which can be taken from these results which hasn't already been reflected upon in the previous scan sections. All models correctly classify this image, and the tumour does appear in one of the VGG16 explanations and both of the ResNet50 classifications. Top features again include outer and inner regions of the brain.

Now that all three planes of pituitary have been scrutinised, it appears that only roughly half of all interpretable images show the tumour In the top features, and if we discount the incorrect classification by ResNet50 in the sagittal plane, it's less than half. As I've mentioned before, I think this tells us that the tumour itself isn't the most important factor in classifying *which type* of tumour it is.

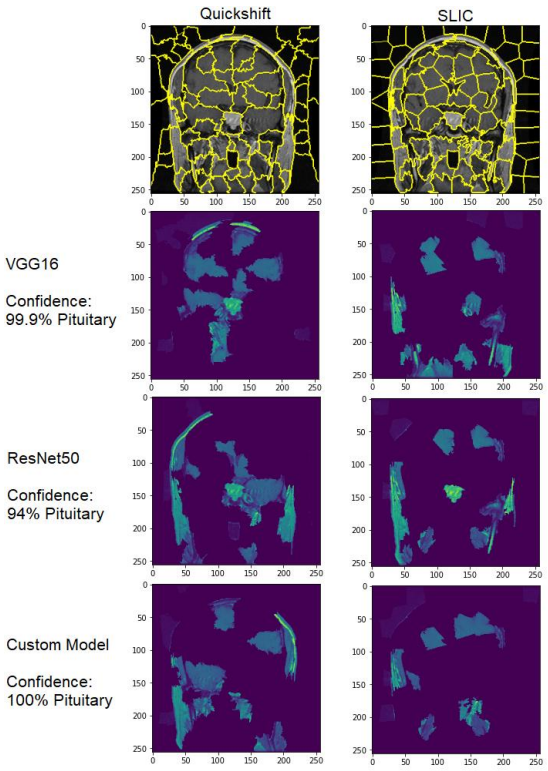

Figure 87 - Pituitary Coronal Explanations

7.5. Model Explanations of 'No Tumour' Image Classification

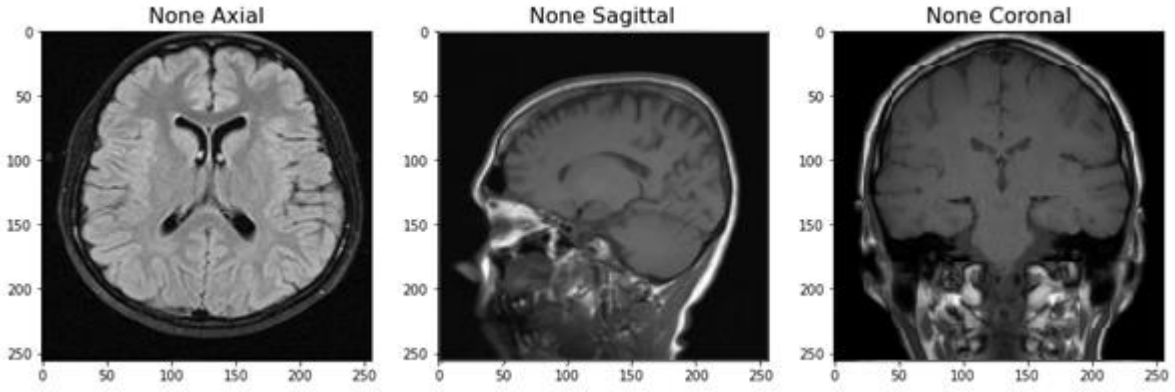

Figure 88 - No Tumour Test Images

Finally, the last set of experiments will be for the 'no tumour' class. I believe that the explanations for this class are going to be a little bit harder to interpret and understand, purely because there isn't any specific tumour we are looking for. What will be interesting is to see what areas of the brain are prioritized by each model for them to classify the image as no tumour. The images from each of the MRI planes are shown above in *Figure 91*, and again the top 15 features of each image will be displayed in the results.

No Tumour Explanations of the Axial Plane

As there are no tumours to find in this section of interpretable images, the main thing I will be looking for is if the top features are consistent with those shown in the scans with brain tumours. If there is a consistent pattern of what the model is looking at, it will reinforce the hypothesis that the model searches common hotspots of the brain to find tumours.

Starting with the axial plane, results for which are shown in *Figure 92.* It seems that the explanations are showing that many of the top features in all the models are those same outer and inner regions of the brain that were present in the explanations of the scans which contain tumours. At least this shows that the models are somewhat consistent between classes. What does inspire confidence in the models is that for these results, little of the background image appear in the top features, which I thought may happen for some of the images with no tumours if there was bias in the model.

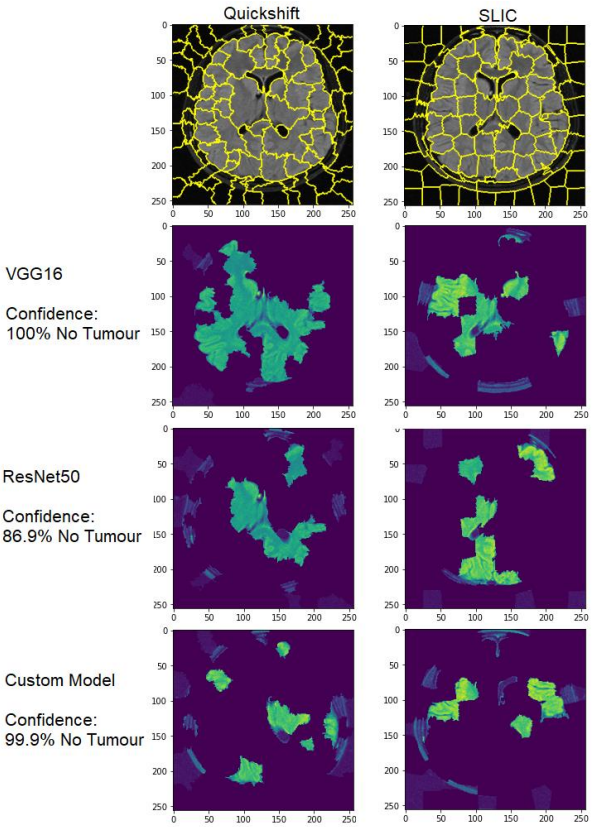

Figure 92 - No Tumour Axial Explanations

No Tumour Explanations of the Sagittal Plane

The results of the sagittal plane for no tumour are shown in *Figure 93,* and it seems like much of the same story told with the sagittal planes of the other classes. One thing to note is that the Custom Model was only 46% confident that there was no tumour in this scan and classified the image as a meningioma with just over 50% confidence. One explanation for this error could be that there is a region in the centre of the brain which does have some contour around it, but the region itself is not highlighted by the contrast die like a tumour usually would be. It could be mistaking this area and the contour around the region as a tumour site. Even though the Custom Model incorrectly classified this model, it's great to be able to see the top features and hypothesise why the model was incorrect, which wouldn't be possible without this explainer.

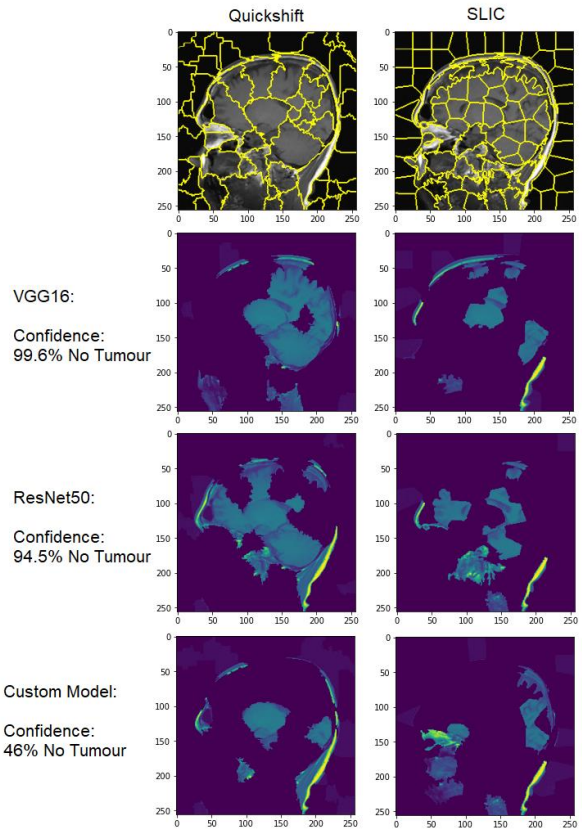

Figure 89 - No Tumour Sagittal Explanations

No Tumour Explanations of the Coronal Plane

The results for the coronal plane of no tumour classification are shown in *Figure 94* and are the last set of explanations in this series of experiments. It appears that the top features include somewhat similar regions to the explanations of other classes. In these images the Quickshift version of segmentation does appear to show a lot more of the brain, but I think this is due to the amount of superpixels generated by the QuickShift algorithm. In this image SLIC divided the brain into more superpixels than Quickshift did, therefore the top 15 features of SLIC are going to show less of the brain in total. Even with this being the case, there appears to be consistency between the two.

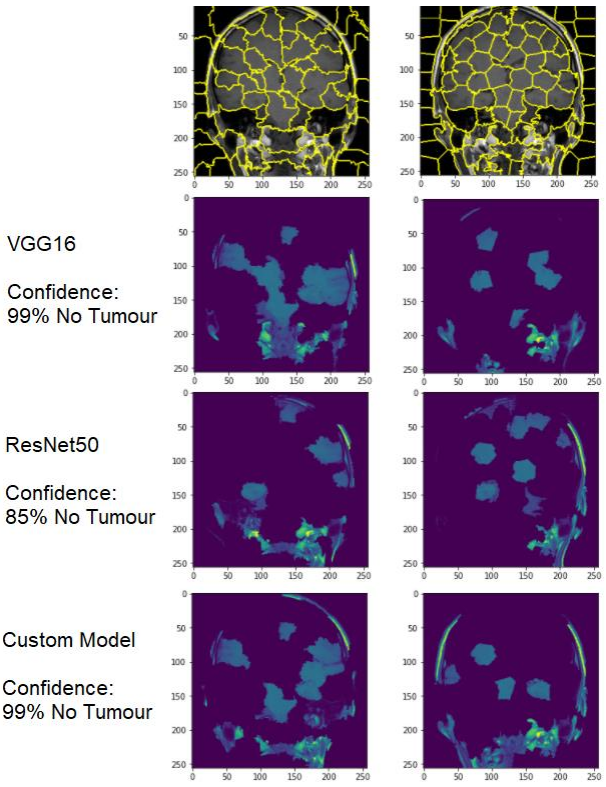

Figure 90 - No Tumour Coronal Explanations

7.6. Testing Summary

I have been quite heavy in my testing of the explainer as I wanted to compare each model, and each segmentation algorithm for each plane of MRI. By doing so I gather as much evidence as possible to observe if there was consistency in the interpretable images, and to discern whether the model was biased for any of the planes.

To conclude the experiments, I think the interpretable images have granted some great insight into the working of the models and how they make their classifications. There does appear to be consistency in the regions of the brain which appear in the top features, and the reason for this could be explained by my hypothesis that the models are checking regions in which meningiomas or gliomas would commonly appear. I don't think the LIME explanations uncovered any obvious bias in any of the models, unless my assertions have been completely false, in which case I do not have an explanation for some of the behaviours of the models, other than that it may be influenced by learned bias from the dataset. I would enjoy the opportunity to spend time with an experienced radiologist and machine learning engineer to hear their thoughts and discuss what they make of these interpretations.

8. Reflecting On Aims, Project Conclusion and Future Work.

8.1. Introduction

Now that I've reached the end of the project, I will return to the initial aims & objectives to measure if my own defined deliverables have been met, and whether the project has been a success. Considerations for future work will also be discussed, including what could be done to improve the outcome of the project, and a conclusion where I will voice my thoughts and level of satisfaction with the outcome.

8.2. Reflection on *Project Aims & Objectives*

The subheadings in this section will mirror those of section 1.3. Under each I will reflect on the objective and discuss whether I think I have met that deliverable, and what could be improved.

Acquire a Suitable Dataset

The final form of the dataset used in this project consisted of two, very different datasets which had been merged. Each of the datasets I would say were of considerably different qualities. The first of the two was the dataset from Jun Cheng, from the studies [46][47], and consisted of just Gliomas, Meningiomas and Pituitary T1-Weighted scans. This was the more reliable dataset in terms of quality and source, but it wasn't particularly well balanced with an overrepresentation of Gliomas. The second dataset source was slightly less convincing, but it was necessary if I wanted to make this a 4-class problem with the healthy brain class included.

This second dataset contained the same three classes from the first dataset with the addition of the healthy brains, although this class was heavily also unrepresented. One additional detail about this dataset is that it wasn't just T1-Weighted MRI scans like the first, it was a mix of T1, T2 and Flair. I suppose this could have introduced the potential for model bias, but as the two datasets were quite balanced in their size, I didn't see this as being a problem and if anything, would make the model better at generalising. The source of this second dataset was Kaggle [48], and on the discussion page for the dataset there is an image containing validation of the image classes from a doctor, but this is the only guarantee I had that the labels were accurate.

Overall, as I expressed in section 3, if this application was being developed with deployment and use on human subjects, then I do not think the datasets would be suitable, and instead a bespoke dataset should be produced. But for the use of this undergraduate project, I think the use is justified and that they served their purpose well and the models seem robust training from this data.

Data Explorations and Pre-processing

The data exploration process proved to be successful in the sense that by conducting it, I was able to understand what alterations needed to be applied to the dataset in order for it to be used in the training process. There was a fair amount of pre-processing which was required to the second dataset, as these images we not already uniform in size and processed like those in the first dataset. The only formatting which was required for the first dataset was a 90-degree rotation so all images from both datasets were vertically aligned. This was a small step which was overlooked at first, but upon applying this transformation training scores improved quite dramatically.

Data augmentation did require a lot of fine-tuning and experimentation. As I have said, these datasets were not at all balanced individually or as a merged dataset. Using data augmentation, I was able to artificially expand the overall size of the dataset, giving the models more images to learn from, but to also balance the classes out in the process.

In my first few attempts, I wasn't actively trying to balance the classes perfectly, only to expand the data and introduce some different augmentations of the existing images to make the models more robust, and this omission was reflected negatively in the testing scores. Particularly on the no tumour class, where the scores were rather poor with a lot of false-positives and false-negatives being present. After making these observations I applied additional augmentation and balanced the classes as well as I could, and the testing score were improved greatly, especially so in the VGG16 model which achieved 100% recall on no tumour and almost perfect precision.

Overall, I would say this phase of the project was a success, and it provided a lot of insight and experience of what methods can be employed to artificially increase the quality of a dataset which is reflected in training scores.

Research, Train and Evaluate Machine Learning Models

The reviewed literature which had relation to my own project uncovered a plethora of machine learning model architectures and variations which showed varying levels of success, but one constant was that transfer learning models always seemed to be the most successful, and this was also the case in my project, specifically with the VGG16 model. Unfortunately, this wasn't something that I had attempted in the first phases of this project, and a lot of custom model trial and error preceded the use of VGG16, but that's part and parcel of this type of project. It's hard not to be satisfied with the results of VGG16 in this project, especially when a lot of the custom models (not including the final one evaluated in this project) were achieving results that were far inferior.

The results, I thought, were quite good. The distinction between Gliomas and Meningiomas which I thought was going to problem with little solution didn't seem to be as prevalent as I thought, with VGG16 having between 97 and 99% recall and precision for these classes. Hopefully this isn't too good to be true, but I must always consider that there could potentially be some learned bias in the model, but as was discussed in the LIME testing, I couldn't identify any between these classes.

The results of ResNet50 were quite underwhelming, especially as I expect this model to outperform VGG16. ResNet50 is one of the best performing and most modern models currently available, and has outperformed VGG16 in the ImageNet challenge, so I don't think the inferior score achieved in this project was due to the models architecture at all, and is more likely my tuning of the model. These

models require a lot of trial and error to get right, and with the depth and training time for ResNet50, finding them was far slower than the other two models used in this project. Improving these results is definitely a topic for future work.

Finally, the results of the Custom Model I found quite pleasing, but it must be said that I didn't stay true to my reasoning for including a model of my own creation. After reading about the custom models with almost comparable performance to transfer learning in the paper by *Badža et al* [17], I wanted to see if I could also create a model which was shallower and less performance intensive which delivered results close to those of VGG and ResNet50. In the end though, my custom model ended up being almost as deep and complex as VGG16, but still performed worse, although only slightly. I still think having this model in the project provides value, especially when you consider there were no pre-trained weights for the model, and it trained faster than VGG16 and ResNet50.

To conclude, I'm pleased with the results of the models, especially VGG16 with its 99%+ scores on the unseen data, and I think that the variety of models provided some interesting interpretation results to compare from LIME.

Implement and Evaluate a Machine Learning Model Explainability Method

The final stretch of the project was to implement a version of LIME and to test it on an array of images which cover all the variety in the dataset.

Coming into this part of the project I expected to be able to use the LIME library developed by the original authors of the paper presenting LIME. Unfortunately, due to incompatibility problems with the dataset which I was using for my project I looked to implement my own LIME functions which would achieve the same goal.

Upon reflection, knowing what I now know, there were certainly ways in which I could have circumvent the issues I was facing with the LIME library. That being said, experiencing these issues forced me into implementing my own version of LIME which in part helped me develop a deeper understand the explainer, whereas just using the library would not have. I had expected the implementation of LIME to be long and arduous, but in fact it was rather benign, which was with no doubt cushioned by the helpful information which can be found online forums and from sources implementing their own version of LIME. Of which I have referenced in the introductory paragraph of the explainer implementation section. I'm pleased with the implementation my of LIME, and the way that I have the functions structured made it easy to test images and view the results.

With the testing of LIME, I wanted to be very thorough, I was aware of all the variety in the dataset which I was using, with each class having three different planes in which tumour are pictured, and also that I wanted to compare segmentation algorithms. This, with all three models considered meant that for each class 18 images were produced, which was a lot of explanations to interpret. I won't delve deep into the interpretations in this section, for that the reader can navigate to section 7, I just want to touch on my conclusion of the testing and what I thought it revealed in the models.

Overall, I think LIME did provide some valuable context for the classification of the 'black-box' machine learning models. With my foundational knowledge of brain tumour diagnosis, the explainer's interpretable images showing the top features seemed logical and supported what I believed to be a sound hypothesis regarding how a model could distinguish between tumours like Gliomas and Meningiomas. There also wasn't any obvious bias that I could notice from the images, which if present,

should have been exposed by the explainer. But I must also consider that there's a chance that there was bias present, and I wasn't able to identify it.

To conclude, I think the application of LIME was a success, and that the classification explanations provided by LIME were far more interpretable than a simple class label with a confidence score, and that after viewing the explanations, my trust in the models had increased. Finally, as a call back to the title of the paper which introduced LIME; *"Why should I trust you?"* - because the explanations presented by the explainer seemed mostly sensible and logical, and similar to how I would attempt to distinguish between classes when observing MRI images.

8.3. Time Management

Going into this project I had not defined a Gantt chart or any other detailed project timeline. I had an initial plan which was to have certain high-level deliverables ready by specific dates. The philosophy behind this choice was that I wasn't sure what the minutia of the project was going to entail, so drawing up some complex plan which wasn't going to be accurate didn't seem like it would be useful to me. The timeline which was defined for this project is shown in *Appendix 12.2.* At the time of this projects inception, key parts of this project such as the choice of explainer and machine learning models had not been decided, and I was still under the impression that I was going to be developing a single model, not three.

Due to the poor understanding of what this project was going to require and deferrals of project work because of other study commitments, the initial plans timescale wasn't accurate at all. The first mistake was to think that this project would flow in a waterfall style, when in reality there was a lot of agile cycling back and forward, making changes to the project and then testing them, until eventually I ended up with the final constituent parts of this whole project. A good example of this is the amount of work on dataset pre-processing and data augmentation which ended up being required to train an accurate model. I had initially stated that data finalisation, pre-processing, model experimentation and evaluations was going to be finished by Christmas 2020, when in fact these parts of the project were active up until April.

Another area in which time got the best of me is for the project draft delivery date in March. Prior to starting this core of this project, I expected my draft to contain an almost-finished version of this project, when in fact I only had a primitive model with poor accuracy defined, and an unfinished literature review.

These time-management failures are absolutely something which I have learnt from and will consider going into any future problems. Now that I have the experience and knowledge of conducting a project of this scale by myself, I will be much better prepared and should be able to set more realistic goals.

With this being said, I didn't feel an extensive amount of pressure building up towards the deadline of this project. I felt in control of what was happening and didn't feel like I was going to be tight on the deadline at any point, other than a slight bit of anxiety when I was approaching this end of this report and hadn't quite realised just how much there was to demonstrate and explain.

8.4. Future Work

This project has undergone a lot of modifications and widening of scope since its inception, and I'm pleased that it's achieved the goal it was set out for, but I still think that there is a lot of opportunity for this work to be built upon and extended. The first thing I would like to see is what the explainer's explanations would show if the dataset in this project was much larger, naturally balanced and contained more classes. I was slightly worried that the amount of data augmentation may skew the explanations, and although this doesn't seem to be the case, the explanations may be a higher quality with a richer dataset.

On the topic of explanations, something that I had wanted to explore was the possibility of showing the inverse of the top features, so the features which had influence on the model away from the true class. It's possible that this could have uncovered some additional patters in distinguishing between tumours, and perhaps it may have uncovered some model bias. This is something which would be interesting to explore.

Another topic of interest to be is the usage of a segmentation algorithm which isn't influenced by the contents of an image and would reliably and predictably segments images in a uniform manner. I touched upon my reason for wanting to both Quickshift and SLIC because of Quickshifts tendency of cutting out the tumour completely and how this may be denying the model context of surrounding tissue in its classifications. Although SLIC did alleviate this to a degree, and the segments were more of a uniform grid, it still suffered the same problem on occasion. I would be curious to see a segmentation algorithm which simply cuts up the image randomly and never, unless just by chance, segregates the tumour. This whole hypothesis may not even be an issue, but even so it would be valuable to see what the model returns with different segmentation algorithms.

Finally, this problem and dataset could be applied to a host of other machine learning models to see what works best. During this project, I had wanted to find a model which was pretrained on a similar problem, such as train x-rays or CT scans to see if that would provide better learning, but I ended up settling with ImageNet weights for the transfer models. This is a potential future work which could improve the quality of explanations.

8.5. Overall Conclusion

To summarise this project in as concise a manner as possible; I'm satisfied that the defined goals of the project have been met. I believe at least one of the models trained has was robust and provided accurate generalisations. I'm very pleased that the implementation of LIME provided a valuable insight into the model, what parts of a brain scan it considers with the most importance for its classification and revealed whether the model in question could be trusted.

Moving on, I would like to think that this area of work, even if not my own, could provide a strong foundation or link in the chain which eventually leads to machine learning being used ubiquitously throughout healthcare, for the prosperity of all people, and that their application will combat the horrendous growth in the rates of tumour incidents in the UK and the rest of the world.

9. Statement of Ethics

During the course of this project, Legal, Ethical, Social and Personal issues have been considered. The following section will summarise these considerations in the context of the UK legislation, which is applicable to the work completed, as well as the ethical and moral principles which must be adhered to.

The Data Protection Act 1998 is a law in the United Kingdom which dictates how data which pertains to living people should be processed and held by organisations, government, and businesses. The data used in this project consists of a series of images of human brains captured through the means of MRI machines. These images, although featuring human anatomy belonging to individuals, contain absolutely no information which could lead to the identification of the person being featured in the image. Parts of the dataset included patient numbers to accompany the scans which featured in the project, although these patient numbers have not been stored or using in any way during this project, the numbers have no other associated information which could lead to them being used to identify the hospital or the patient, and are simply used as an index for each image. To reiterate, the images and their associated class labels have been the only information stored and processed. This data is held on a personal computer which password encrypted, and the data is not part of any application or directory which can be accessed through the remotely, and it will be disposed of upon conclusion of this project.

The data procured for the use in this project was acquired in accordance with the Computer Misuse Act 1990, which is another UK legislation which makes it an illegal to access or obtain computer material in any unauthorised or clandestine manner. Both datasets which form the core data used in this project have been downloaded from open-source, free-to-access websites on the internet. The first being *Kaggle* [48], a popular dataset sharing platform which is a subsidiary of Google LLC, and the second being *Figshare*[51], which is another open access platform where scientists can share their data with reference to their original works.

The work in this project was undertaken with hope that a benefit to society and human wellbeing could be achieved, even if only a step towards it. The purpose of the models and the explainer implemented in this project was to research methods which can supplement medical intervention and ultimately preserve the life and health of individuals everywhere.

The benefit to society and human wellbeing being that tumours could be diagnosed and identified faster, more accurately and more economically, while alleviating possible points of failure which are present with human diagnosis such as fatigue, subjectivity, and inexperience. This would hopefully result in faster remedial action, fewer deaths, and fewer damaged families.

An ethical issue which arose during the research of this project is that of reproduction and misuse. I must express that it has been in no way my intention for this work to be recreated and used in any way which could negatively impact the outcome of a person's medical intervention, and I urge any readers which may want to reproduce the work in this report to be responsible, and consider the ramifications of misuse. The threshold which must be reached through scrutiny, testing and evaluation for real-life application of methods such as mine will be much higher than that which has been met in this project, and therefore it is not ready to be used for real medical diagnosis under any circumstances. This work has been purely educational and should only be considered in that context.

10. References

[1] *Cancer Research UK. 2021. Brain, other CNS, and intracranial tumours incidence statistics. [online] Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/brain-other-cns-and-intracranial-tumours/incidence*

[2] *Shimizu H, Nakayama KI. Artificial intelligence in oncology. Cancer Sci. 2020;111(5):1452-1460. doi:10.1111/cas.14377*

[3] *Maclin, P.S., Dempsey, J., Brooks, J. et al. Using neural networks to diagnose cancer. J Med Syst* ***15,*** *11–19 (1991)*

[4] *Rajat Raina, Anand Madhavan, and Andrew Y. Ng. 2009. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09). Association for Computing Machinery, New York, NY, USA, 873–880.*

[5] *Nibib.nih.gov. 2021. Magnetic Resonance Imaging (MRI). [online] Available: https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri*

[6] *Ribeiro, M., Singh, S. and Guestrin, C., 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier*

[7] *Cancer Research UK. 2021. Cancer incidence for common cancers. [online] Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/incidence/common-cancers-compared*

[8] *Cancer Research UK. 2021. Brain, other CNS and intracranial tumours risk factors. [online] Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/brain-other-cns-and-intracranial-tumours/risk-factors#heading-Six*

[9] *Cancer Research UK. 2021. Brain, other CNS and intracranial tumours incidence statistics. [online] Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/statistics-by-cancer-type/brain-other-cns-and-intracranial-tumours/incidence#heading-Two*

[10] *Ons.gov.uk. 2021. Cancer registration statistics, England Statistical bulletins - Office for National Statistics. [online] Available: https://www.ons.gov.uk/peoplepopulationandcommunity/healthandsocialcare/conditionsanddiseases/bulletins/cancerregistrationstatisticsengland/previousReleases*

[11] *Isdscotland.org. 2021. Cancer | Publications | Health Topics | ISD Scotland. [online] Available: https://www.isdscotland.org/Health-Topics/Cancer/Publications*

[12] *Phw.nhs.wales. 2021. Welsh Cancer Intelligence and Surveillance unit (WCISU) - Public Health Wales. [online] Available: https://phw.nhs.wales/services-and-teams/welsh-cancer-intelligence-and-surveillance-unit-wcisu/*

[13] *Qub.ac.uk. 2021. Northern Ireland Cancer Registry | N. Ireland Cancer Registry. [online] Available: https://www.qub.ac.uk/research-centres/nicr/*

[14] *Davis, F., Dolecek, T., McCarthy, B. and Villano, J., 2012. Toward determining the lifetime occurrence of metastatic brain tumors estimated from 2007 United States cancer incidence data. Neuro-Oncology, 14(9), pp.1171-1177.*

[15] *Cancer Treatment Centers of America. 2021. What is Metastatic Cancer? Diagnosing and Treatment. [online] Available: https://www.cancercenter.com/metastasis*

[16] *Piper, R., Mikhael, S., Wardlaw, J., Laidlaw, D., Whittle, I. and Bastin, M., 2021. Imaging signatures of meningioma and low-grade glioma: a diffusion tensor, magnetization transfer and quantitative longitudinal relaxation time MRI study.*

[17] *Badža, M. and Barjaktarović, M., 2020. Classification of Brain Tumors from MRI Images Using a Convolutional Neural Network. Applied Sciences, 10(6), p.1999.*

[18] *Cancer Research UK. 2021. Meningioma | Brain and spinal cord tumours | Cancer Research UK. [online] Available: https://www.cancerresearchuk.org/about-cancer/brain-tumours/types/meningioma*

[19] *Larjavaara, S., Mäntylä, R., Salminen, T., Haapasalo, H., Raitanen, J., Jääskeläinen, J. and Auvinen, A., 2007. Incidence of gliomas by anatomic location. Neuro-Oncology, 9(3), pp.319-325.*

[20] *Gamburg, E., Regine, W., Patchell, R., Strottmann, J., Mohiuddin, M. and Young, A., 2000. The prognostic significance of midline shift at presentation on survival in patients with glioblastoma multiforme. International Journal of Radiation Oncology*Biology*Physics, 48(5), pp.1359-1362.*

[21] *Chaudhary, V. and Bano, S., 2011. Imaging of the pituitary: Recent advances. Indian Journal of Endocrinology and Metabolism, 15(7), p.216.*

[22] *Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. ArXiv, abs/1811.03378.*

[23] *Onwujekwegn, Gerald and Yoon, Victoria Y, "Analyzing the Impacts of Activation Functions on the Performance of Convolutional Neural Network Models" (2020). AMCIS 2020 Proceedings. 4.*

[24] *En.wikipedia.org. 2021. Convolutional neural network - Wikipedia. [online] Available: https://en.wikipedia.org/wiki/Convolutional_neural_network*

[25] *Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature **323,** 533–536 (1986). https://doi.org/10.1038/323533a0*

[26] *Ruder, Sebastian. (2016). An overview of gradient descent optimization algorithms.*

[27] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[28] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[29] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[30] R. M. Prakash and R. S. S. Kumari, "Classification of MR Brain Images for Detection of Tumor with Transfer Learning from Pre-trained CNN Models," *2019 International Conference on Wireless Communications Signal Processing and Networking (WiSPNET)*, 2019, pp. 508-511, doi: 10.1109/WiSPNET45539.2019.9032811.

[31] *Magesh, Pavan & Myloth, Richard & Tom, Rijo. (2020). An Explainable Machine Learning Model for Early Detection of Parkinson's Disease using LIME on DaTscan Imagery.*

[32] Arteagac.github.io. 2021. *Cristian Arteaga*. [online] Available: https://arteagac.github.io/blog.html

[33] *Case.edu. 2021. David C. Preston. MRI Basics. [online] Available: https://case.edu/med/neurology/NR/MRI%20Basics.htm*

[34] *Cancerresearchuk.org. 2021. Glioma | Brain tumours (primary) | Cancer Research UK. [online] Available: https://www.cancerresearchuk.org/about-cancer/brain-tumours/types/glioma-adults*

[35] *Cancerresearchuk.org. 2021. Pituitary tumours | Brain tumours (primary) | Cancer Research UK. [online] Available: https://www.cancerresearchuk.org/about-cancer/brain-tumours/types/pituitary-tumours*

[36] *Yann.lecun.com. 2021. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [online] Available: http://yann.lecun.com/exdb/mnist/*

[37] *2021. ImageNet Large Scale Visual Recognition Challenge (ILSVRC). [online] Available: https://www.image-net.org/challenges/LSVRC*

[38] *Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., ... & He, Q. (2020). A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), 43-76.*

[39] *Dabeer, S., Khan, M. and Islam, S., 2019. Cancer diagnosis in histopathological image: CNN based approach. Informatics in Medicine Unlocked, 16, p.100231.*

[40] *Cruz, J. A., & Wishart, D. S. (2006). Applications of machine learning in cancer prediction and prognosis. Cancer informatics, 2, 117693510600200030.*

[41] *Bach S, Binder A, Montavon G, Klauschen F, Müller KR, et al. (2015) On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. PLOS ONE 10(7): e0130140. https://doi.org/10.1371/journal.pone.0130140*

[42] *Jansen T, Geleijnse G, Van Maaren M, Hendriks MP, Ten Teije A, Moncada-Torres A. Machine Learning Explainability in Breast Cancer Survival. Stud Health Technol Inform. 2020 Jun 16;270:307-311. doi: 10.3233/SHTI200172. PMID: 32570396.*

[43] *Böhle, M., Eitel, F., Weygandt, M. and Ritter, K., 2019. Layer-Wise Relevance Propagation for Explaining Deep Neural Network Decisions in MRI-Based Alzheimer's Disease Classification. Frontiers in Aging Neuroscience, 11.*

[44] *Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. 2012.*

[45] *Quick shift and kernel methods for mode seeking, Vedaldi, A. and Soatto, S. European Conference on Computer Vision, 2008*

[46] *Cheng J, Huang W, Cao S, Yang R, Yang W, et al. (2015) Correction: Enhanced Performance of Brain Tumor Classification via Tumor Region Augmentation and Partition. PLOS ONE 10(12): e0144479.*

[47] *Cheng J, Yang W, Huang M, Huang W, Jiang J, et al. (2016) Retrieval of Brain Tumors by Adaptive Spatial Pooling and Fisher Vector Representation. PLOS ONE 11(6): e0157112*

[48] *Kaggle.com. 2021. Brain Tumor Classification (MRI) Dataset. [online] Available: https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri/discussion/154260*

[49] *Scikit-image.org. 2021. Module: segmentation — skimage v0.19.0.dev0 docs. [online] Available: https://scikit-image.org/docs/dev/api/skimage.segmentation.html*

[50] *Brownlee, J., 2021. How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification. [online] Machine Learning Mastery. Available: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification*

[51] *figshare. 2021. brain tumor dataset. [online] Available: https://figshare.com/articles/dataset/brain_tumor_dataset/1512427*

11. Appendices

11.1. SAGE HDR Self-Certification Form

SAGE-HDR

Response ID	Completion date
640816-640807-78298606	17 May 2021, 13:04 (BST)

1	**Applicant Name**	Jack Cannings
1.a	**University of Surrey email address**	jc01425@surrey.ac.uk
1.b	**Level of research**	Undergraduate
1.b.i	**Please enter your University of Surrey supervisor's name. If you have more than one supervisor, enter the details of the individual who will check this submission.**	Hongying (H Lilian) Tang
1.b.ii	**Please enter your supervisor's University of Surrey email address. If you have more than one supervisor, enter the details of the supervisor who will check this submission.**	h.tang@surrey.ac.uk
1.c	**School or Department**	Computer Science
1.d	**Faculty**	FEPS - Faculty of Engineering and Physical Sciences Sciences

2	**Project title**	Machine Learning Explainability in MRI Brain Tumour Classification

3	**Please enter a brief summary of your project and its methodology in 250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview of the aims and objectives of your research.**	Training convolutional neural networks on datasets of MRI brains scans which do not include patient information, only tumour class, to then be explained by a model interpreter called Linear Agnostic Model Explanations (LIME). The overall goal of this project is to interpret black-box machine learning models to assess whether they are trustworthy. The project will be conducted on my home pc, data will not be stored in any external databases.

4	**Are you planning to join on to an existing Standard Study Protocol (SSP)? SSPs are overarching pre-approved protocols that can be used by multiple researchers investigating a similar topic area using identical methodologies. Please note, SSPs are only being used by one module currently.**	NO

5	**Are you making an amendment to a project with a current University of Surrey favourable ethical opinion in place?**	NO

6	**Does your research involve any animals, animal data or animal derived tissue, including cell lines?**	NO

8	**Does your project involve any of the following: human participants (including human data and/or any human tissue*); or is your project linked to engineering and/or the physical sciences?**	YES

14	**Does your project involve any type of human tissue research? This includes Human Tissue Authority (HTA) relevant, or irrelevant tissue (e.g. non-cellular such as plasma or serum), any genetic material, samples that have been previously collected, samples being collected directly from the donor or obtained from another researcher, organisation or commercial source.**	NO

15	**Does your research involve exposure of participants to any hazardous materials e.g. chemicals, pathogens, biological agents or does it involve any activities or locations that may pose a risk of harm to the researcher or participant?**	NO

16	**Will any activities in your research take place in the Surrey Clinical Research Building (CRB)?**	NO

17	**Will you be accessing any organisations, facilities or areas that may require prior permission? This includes organisations such as schools (Headteacher authorisation), care homes (manager permission), military facilities etc. If you are unsure, please contact RIGO.**	NO

18	**Will you be working with any collaborators or third parties to deliver any aspect of the research project?**	NO

19	**Is your project a service evaluation or an audit?**	NO

20	**Does your funder, collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?**	NO

21	**Are you undertaking security-sensitive research, as defined in the text above?**	NO

22	**Does your project process personal data1? Processing covers any activity performed with personal data, whether digitally or using other formats, and includes contacting, collecting, recording, organising, viewing, structuring, storing, adapting, transferring, altering, retrieving, consulting, marketing, using, disclosing, transmitting, communicating, disseminating, making available, aligning, analysing, combining, restricting, erasing, archiving, destroying.**	NO

23	**Does your project require the processing of special category2 data?**	YES

23.a	**Please ensure that you adhere to the data protection guidance**	I am an UG or PGT student and I understand that I have to abide by the 'Data protection and security for undergraduate and postgraduate taught student projects' policy found at https://research.surrey.ac.uk/ethics

24	**Are you using a platform, system or server[3] to collect, process and/or store any personal and/or special category data?**	NO

25	**Does your research involve any of the above statements? If yes, your study may require external ethical review or regulatory approval**	NO

26	**Does your research involve any of the above? If yes, your study may require external ethical review or regulatory approval**	NO

27	**Does your project require ethics review from another institution? (For example: collaborative research with the NHS REC, the Ministry of Defence, the Ministry of Justice and/or other universities in the UK or abroad)**	NO

31	**Declarations**	• I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these. • I confirm that I have provided accurate and complete information regarding my research project • I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies • I understand that if my answers to this form have indicated that I must submit an ethics and governance application, that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings

		liable for reporting to external bodies. • I understand that if I have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application (EGA) for review before conducting any research. If I have NOT selected any of the higher, medium or lower risk criteria, I understand I can proceed with my research without review and acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance

32	**If I am conducting research as a student:**	• I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct. • I confirm that if I am handling any information that can identify people, such as names, email addresses or audio/video recordings and images, I will adhere to the security requirements set out in the relevant Data protection Policy

11.2. Initial Project Timeline

Step	Task	Description	Est. Finish Date
1	Dataset Finalisation	Finalise my decision on which dataset(s) will be used for the project	Nov 2020
2	Dataset Pre-processing	Organise the data, Resize, Balance the Set	Nov 2020
3	Model Experimentation / Primitive Visualisations	Implement a protype CNN, Train and Test. Visualise some of the results of the model using heatmaps and TensorFlow.	Dec 2020
4	Model Evaluation	Evaluate the accuracy of the model, adjust parameters / hyperparameters to prevent over/underfitting.	Dec 2020
5	Start Report / Literature Review	Start developing the report structure / literature review section	Dec 2020
6	Experiment with more sophisticated visualisation techniques	Use techniques and algorithms researched from literature to achieve a better understanding of my model	Feb 2021
7	Evaluate the techniques and viability	Explore and understand the results from using more sophisticated visualisation techniques alongside the CNN model.	Feb/March 2021
8	Draft Report	Start writing up the results and critique the methods used and the viability of them to be used to support medical diagnosis.	March 2021
9	Final Report	Conclude final report including all results, critique and evaluate the quality of the project itself, and reflect.	May 2021

11.3. Figure Web URL Sources

Figure 5	https://www.uclahealth.org/neurosurgery/meningioma-brain-tumor
Figure 11	https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/
Figure 12	https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9
Figure 14	https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/
Figure 15	https://www.matthewzeiler.com/mattzeiler/deconvolutionalnetworks.pdf
Figure 22	https://pubmed.ncbi.nlm.nih.gov/
Figure 25	https://www.oreilly.com/content/introduction-to-local-interpretable-model-agnostic-explanations-lime/