HEAD TUMOR DIAGNOSTICS USING MRI, PATIENTS PATHOLOGY DATA AND MACHINE LEARNING ALGORITHMS

A Dissertation
Presented to
The Academic Faculty

by

Papadomanolakis Theodoros

In Partial Fulfillment
of the Requirements for the Degree
Electrical and Computer Engineer in the
Technical University of Crete

HEAD TUMOR DIAGNOSTICS USING MRI, PATIENTS PATHOLOGY DATA AND MACHINE LEARNING ALGORITHMS

Thesis Committee:

Professor Dr. Michael Zervakis, Supervisor
School of Electrical and Computer Engineering
Technical University of Crete

Professor Dr. Georgios Stavrakakis, Member
School of Electrical and Computer Engineering
Technical University of Crete

Dr. Eleutheria Sergaki, Co-Supervisor and Member
School of Electrical and Computer Engineering
Technical University of Crete

ACKNOWLEDGEMENTS

I would like to express my gratitude to my co-advisor Dr. Eleftheria Sergaki, who guided me through this project. Her willingness to give her time so generously has been very much appreciated. Moreover, I would like to thank my advisor Professor Dr. Michael Zervakis who supported my Thesis Thema, it is an honor for me to be my Thesis advisor. Last but not least, I would like to thank Professor Dr. Georgios Stavrakakis, as member of the commission and as my professor during my studies, he was particularly helpful to me throughout this process.

I wish to show my appreciation to neurosurgeon Dr. Antonios Krasoudakis and radiographer Mr. Makris Tsalikis Georgios for their valuable help and constructive suggestions to my thesis.

Also thanks to all my university professors, they all helped in a unique way.

Finally yet importantly, I would like to thank my family for their support throughout this process.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

LIST OF TABLES **v**

LIST OF FIGURES **vii**

Abstract **ix**

CHAPTER 1. Introduction **1**
1.1 Motivation and objectives of the study **1**
1.2 Overview of Contents **1**

CHAPTER 2. basic knowledge **3**
2.1 Clinical Background **3**
2.1.1 Anatomy of the Brain 3
2.1.2 Basic Brain tissues 4
2.1.3 Brain Tumour 5
2.1.4 Magnetic Resonance Imaging 6
2.2 Introduction to Machine Learning **8**
2.2.1 Machine Learning Tasks 9
2.2.2 Basic Algorithms Overview 12
2.3 Introduction to Deep Learning **16**
2.3.1 Types of Neural Networks 16
2.3.2 Optimization Algorithms 24
2.3.3 Loss Functions 25
2.4 Data Pre-processing **26**
2.4.1 Skull Stripping 26
2.4.2 Standardize Data 26
2.4.3 Augmentation 27
2.4.4 Discrete Wavelet Transform 27
2.4.5 Principal Component Analysis 29
2.5 Entropy **30**
2.5.1 Wavelet Entropy 30
2.6 Pixel values in Greyscale **31**
2.7 Evaluate and Tuning Machine and Deep Learning Algorithms **31**
2.7.1 Algorithms Evaluation Metrics 32

CHAPTER 3. Tools used in thesis **37**
3.1 Python **37**
3.2 SciPy **37**
3.3 Scikit-learn **38**
3.4 Google Colaboratory **38**

3.5 TensorFlow **39**
3.6 Keras **39**

CHAPTER 4. dataset exploring **41**
4.1 Description of Dataset **41**
4.2 MRI Data Acquisition **41**
4.2.1 Healthy MRI Data 41
4.2.2 Unhealthy MRI Data 42
4.3 Methods used to prepare the Dataset **42**

CHAPTER 5. algorithms used in thesis **45**
5.1 Machine Learning Algorithms **45**
5.1.1 Scenario 1: Wavelet Entropy of Image and unbalanced dataset 45
5.1.2 Scenario 2: Wavelet Entropy of Image and balanced dataset 48
5.1.3 Scenario 3: Pixel values of images and balanced dataset 51
5.2 Deep Learning Algorithms **53**
5.2.1 Scenario 4: Pixel Values of Images 53
5.2.2 Scenario 5: DWT coefficients 58

CHAPTER 6. Algorithms evaluation and future work **67**
6.1 Algorithms evaluation and results **67**
6.2 Future Work **70**

APPENDIX A. Accuracy and loss **72**

REFERENCES **78**

LIST OF TABLES

Table 1:	Evaluation metrics on Validation Dataset for algorithm 1.1	46
Table 2:	Confusion Matrix of Validation Dataset for algorithm 1.1	46
Table 3:	Evaluation metrics on Validation Dataset for algorithm 1.2	48
Table 4:	Confusion Matrix of Validation Dataset for algorithm 1.2	48
Table 5:	Evaluation metrics on Validation Dataset for algorithm 2.1	49
Table 6:	Confusion Matrix of Validation Dataset for algorithm 2.1	50
Table 7:	Evaluation metrics on Validation Dataset for algorithm 2.2	51
Table 8:	Confusion Matrix of Validation Dataset for algorithm 2.2	51
Table 9:	Evaluation metrics on Validation Dataset for algorithm 3.1	52
Table 10:	Confusion Matrix of Validation Dataset for algorithm 3.1	53
Table 11:	Evaluation metrics on Validation Dataset for algorithm 4.1	55
Table 12:	Confusion Matrix of Validation Dataset for algorithm 4.1	56
Table 13:	Evaluation metrics on Validation Dataset for algorithm 4.2	57
Table 14:	Confusion Matrix of Validation Dataset for algorithm 4.2	58
Table 15:	Evaluation metrics on Validation Dataset for algorithm 5.1	60
Table 16:	Confusion Matrix of Validation Dataset for algorithm 5.1	60

Table 17:	Evaluation metrics on Validation Dataset for algorithm 5.2	63
Table 18:	Confusion Matrix of Validation Dataset for algorithm 5.2	63
Table 19:	Evaluation metrics on Validation Dataset for algorithm 5.3	66
Table 20:	Confusion Matrix of Validation Dataset for algorithm 5.3	66
Table 21:	Accuracy, Recall and Specificity of algorithms	67
Table 22:	FPR, FNR and Precision of algorithms	68

LIST OF FIGURES

Figure 2.1: Basic Types of MRI sequences: T1, T2, Flair 8

Figure 2.2: Model of a Simple Neuron 17

Figure 2.3: Model of a Simple Network 19

Figure 2.4: Typical Convolutional Neural Network Architecture 23

Figure 2.5: 3_Level discrete wavelet decompositions 29

Figure 2.6: Confusion Matrix 35

Figure 4.1: Example of the application of skull stripping algorithm. (a) Initial MRI image, (b) MRI image after the application of skull stripping algorithm 43

Figure 5.1: Example of division in quarters of image 47

Figure 5.2: CNN model for algorithm 4.1 54

Figure 5.3: CNN model for algorithm 4.2 56

Figure 5.4: CNN model for algorithm 5.1 59

Figure 5.5: Variances against No. of principal components 61

Figure 5.6: CNN model for algorithm 5.2 62

Figure 5.7: CNN model for algorithm 5.3 65

ABSTRACT

Background and study aims: Magnetic Resonance Imaging (MRI) of the brain along with patients' pathology data can greatly assist radiologists and doctors in providing a more precise diagnosis and therapy. Because of their unpredictable appearance and shape, segmenting brain tumors from multi-modal imaging data is one of the most challenging tasks in medical image analysis. Manual detection and classification of brain tumor by an expert is still considered the most acceptable method, but it is too time-consuming, especially because of the large amount of data that have to be analyzed manually. The purpose of the present study is to train and validate AI algorithms, i.e. Machine Learning (ML) such as Support Vector Machine (SVM) and deep learning algorithms such as CNN algorithms, to classify MRI images of brains between non tumorous and tumorous.

Materials and methods: The image dataset selected contains total 291 male and female adult persons, from which 210 tumorous and 81 non-tumorous cases that a neurosurgeon partner, has segmented all visually. The healthy MRI scans are provided by "St. George" general hospital of Chania, Greece and the unhealthy MRI scans are provided from the Multimodal Image Segmentation Challenge (BRATS). All the MRI images are T2 weighted, from the axial plane. The above dataset divided into subsets. The training sub dataset amounts to 191 tumorous/66 non tumorous cases, and the validation sub dataset which amounts to 19 tumorous/15 non-tumorous cases (56%/44%). Many different scenarios of different methodologies, each one including combination of AI algorithms, using different kind of features as input, and different size of data sets, i.e. balanced or unbalanced data set (74% tumorous/26% non-tumorous), and different training techniques, were implemented in this thesis. Performance metrics such as accuracy, sensitivity and specificity are computed to evaluate the effectiveness of each implemented methodology. Standardization of the training set and 10-fold split for grid search using cross-validation was applied. The balanced dataset amounts to 66 tumorous and 66 non tumorous cases. The unbalanced set amounts to 191 tumorous /non tumorous 66 cases. Training implemented using the gray scale pixel values of the raw whole images as data features values, also using three-level discrete wavelet transform coefficients of the raw whole images and alternatively using the measure of wavelet entropy calculated from three-level discrete wavelet transform coefficients of each whole row image (and of the quarters in which the image is divided). In all cases training implemented with or without applying Principal Component Analysis (in order to reduce the dimensionality of coefficients to 15). The augmentation technique was applied in the case of balanced dataset, in order to generate dataset of 400 tumorous images and 400 non tumorous images, for training the CNN algorithm.

Results: The implemented algorithm based on CNN, trained by balanced dataset, using the discrete wavelet transform coefficients of the whole row images, provided the highest scores: 100% Sensitivity, 97% Accuracy, 93% Specificity, 95% Precision, 0% FNR and 6% FPR and the algorithm implemented based on SVM, trained by balanced dataset, using the pixel values of row whole images as features provided the second highest scores: 100% Sensitivity, 91% Accuracy, 80% Specificity, 86% Precision, 0% FNR and 20% FPR. In both algorithms, no PCA technic is applied. Moreover, it is observed that for the scenario where the training implemented using unbalanced dataset, the features extracted of the images divided in quarters provided better results than that extracted of the whole image.

ΠΕΡΙΛΗΨΗ

Υπόβαθρο και στόχοι της μελέτης: Η μαγνητική τομογραφία (MRI) του εγκεφάλου μαζί με τα δεδομένα παθολογίας των ασθενών μπορούν να βοηθήσουν σε μεγάλο βαθμό τους ακτινολόγους και τους γιατρούς στην παροχή ακριβέστερης διάγνωσης και θεραπείας. Λόγω της απρόβλεπτης εμφάνισης και του σχήματος τους, η κατάτμηση των όγκων του εγκεφάλου από δεδομένα πολλαπλών τρόπων απεικόνισης είναι ένα από τα πιο δύσκολα καθήκοντα στην ανάλυση ιατρικής εικόνας. Η χειροκίνητη ανίχνευση και ταξινόμηση του όγκου του εγκεφάλου από έναν ειδικό θεωρείται ακόμη η πιο αποδεκτή μέθοδος, αλλά είναι πολύ χρονοβόρα, ειδικά λόγω του μεγάλου όγκου δεδομένων που πρέπει να αναλυθούν χειροκίνητα. Ο σκοπός της παρούσας μελέτης είναι να εκπαιδεύσει και να αξιολογήσει αλγόριθμους τεχνητής νοημοσύνης, δηλαδή μηχανική μάθηση (Machine Learning) όπως Support Vector Machine (SVM) και αλγόριθμους νευρωνικών δικτύων όπως αλγόριθμοι CNN, για να ταξινομήσει τις εικόνες μαγνητικής τομογραφίας εγκεφάλων μεταξύ υγιών και μη υγιών.

Μέθοδοι: Το σύνολο των εικόνων που χρησιμοποιείται αναφέρονται σε ενήλικους, άντρες και γυναίκες, περιλαμβάνοντας 291 περιπτώσεις από τις οποίες 81 υγιείς και 210 που έχει παρατηρηθεί εγκεφαλικός όγκος. Όλες οι εικόνες έχουν εποπτευθεί οπτικά από τον νευροχειρούργο συνεργάτη μας Δρ. Κρασουδάκη. Οι υγιείς μαγνητικές τομογραφίες παρέχονται από το Γενικό Νοσοκομείο Χανίων «Άγιος Γεώργιος» και οι μη υγιείς παρέχονται από το Multimodal Image Segmentation Challenge (BRATS). Όλες οι εικόνες μαγνητικής τομογραφίας είναι τύπου T2, του αξονικού επιπέδου (T2 MRI). Το παραπάνω σύνολο δεδομένων χωρίζεται σε υποσύνολα. Το υποσύνολο που χρησιμοποιείται για την εκπαίδευση των αλγορίθμων αποτελείται από 191 μη υγιείς/66 υγιείς περιπτώσεις, και το υποσύνολο για την αξιολόγηση των αλγορίθμων αποτελείται από 19 μη υγιείς/15 υγιείς περιπτώσεις (56%/44%). Πολλά διαφορετικά σενάρια με διάφορες μεθοδολογίες αναπτύχθηκαν σε αυτην την εργασία. Το κάθε ένα περιέχει συνδυασμούς αλγορίθμων τεχνητής νοημοσύνης, χρησιμοποιώντας διάφορα είδη χαρακτηριστικών και διάφορα μεγέθη συνόλων δεδομένων σαν είσοδο, π.χ ισορροπημένα ή μη ισορροπημένα σύνολα (74% μη υγιής/26% υγιής περιπτώσεις). Επίσης αναπτύχθηκαν διάφορες τεχνικές εκπαίδευσης των αλγορίθμων. Μετρήσεις απόδοσης όπως η ακρίβεια και η ευαισθησία υπολογίζονται για την αξιολόγηση της αποτελεσματικότητας των μεθόδων που αναπτύξαμε. Στο υποσύνολο εκπαίδευσης των αλγορίθμων εφαρμόστηκε Standardization και 10-fold split για grid search χρησιμοποιώντας cross-validation. Το ισορροπημένο σύνολο δεδομένων περιέχει 66 υγιείς και 66 μη υγιής περιπτώσεις ενώ το μη ισορροπημένο σύνολο δεδομένων περιέχει 191 μη υγιείς και 66 υγιείς περιπτώσεις. Η εκπαίδευση των αλγορίθμων πραγματοποιήθηκε χρησιμοποιώντας τις gray scale τιμές των pixel ολόκληρων των εικόνωνς ως είσοδο, χρησιμοποιώντας επίσης τους συντελεστές του discrete wavelet transform τριών επιπέδων ολόκληρων των εικόνων και εναλακτικά χρησιμοποιώντας την τιμή wavelet entropy που υπολογίστηκε από τους συντελεστές του discrete wavelet transform τριών επιπέδων, ολόκληρης της εικόνας (και των τεταρτημορίων που διαιρείται η εικόνα). Σε όλες τις περιπτώσεις η εκπαίδευση των αλγορίθμων πραγματοποιήθηκε με ή χωρίς την εφαρμογή principal components analysis (προκειμένου να μειωθεί η διαστασημότητα των συντελεστών σε 15). Στην περίπτωση του ισορροπημένου συνόλου εφαρμόστηκε η τεχνική augmentation, προκειμένου να δημιουργηθεί ένα σύνολο δεδομένων με 400 υγιείς και 400 μη υγιείς περιπτώσεις για την εκπαίδευση του αλγορίθμου CNN. **Αποτελέσματα**: Ο αλγόριθμος που υλοποιήθηκε με βάση το CNN, εκπαιδεύτηκε από το ισορροπημένο σύνολο δεδομένων, χρησιμοποιώντας τους συντελεστές discrete wavelet transform ολόκληρων των εικόνων, είχε τα καλύτερα αποτελέσματα με τιμές: 100% Sensitivity, 97% Accuracy, 93% Specificity, 95% Precision, 0% FNR and 6% FPR. Ο αλγόριθμος που υλοποιήθηκε με βάση το SVM, εκπαιδεύτηκε από το ισορροπημένο σύνολο δεδομένων, χρησιμοποιώντας τις τιμές των pixel ολόκληρων των εικόνων ως είσοδο είχε τα επόμενα καλύτερα αποτελέσματα με τιμές: 100% Sensitivity, 91% Accuracy, 80% Specificity, 86% Precision, 0% FNR and 20% FPR. Και στους δύο αλγόριθμους, δεν εφαρμόζεται η τεχνική PCA. Επιπλέον, παρατηρείται ότι για το σενάριο όπου η εκπαίδευση υλοποιήθηκε χρησιμοποιώντας το μη ισορροπημένο σύνολο δεδομένων, τα χαρακτηριστικά που εξήχθησαν από τις εικόνες διαιρεμένες σε τέταρτα παρείχαν καλύτερα αποτελέσματα από αυτά που εξήχθησαν ολόκληρης της εικόνας.

CHAPTER 1. INTRODUCTION

1.1 Motivation and objectives of the study

Magnetic Resonance Imaging (MRI) provides detailed images of the brain and is one of the most common ways used to diagnose brain tumors. The classification and detection of brain tumors from different medical images demands high accuracy since it is crucial for human life. Patient diagnosis relies on doctor's manual evaluation of a patient and his or her test results. With no automated tools to help with a doctor's diagnosis and limited number of available doctors, not only is there a higher risk of misdiagnosis but also an increase in wait time for patients to be seen. Doctors must take the time to manually review test results and images rather than spending time with the patient. In order to improve patient care, enhanced medical technology in the form of automated tools is necessary to increase doctor efficiency and decrease patient time in hospitals and time toward recovery.

The main objective of this thesis is to examine different classification techniques for the problem of recognizing a healthy person from a patient that contains a brain tumor, based on a magnetic resonance imaging (MRI) of T2 weighted modality from the axial plane. For the completion of this task, specific methods are examined, optimized and finally combined in five scenarios and several algorithms that are presented in this thesis.

1.2 Overview of Contents

The remainder of this dissertation is organized as follows

- Chapter 2 presents the basic knowledge that we need for this thesis
- Chapter 3 presents the tools that we used to develop our algorithms
- Chapter 4 describes the data that we used in this thesis
- Chapter 5 presents the scenarios and algorithms we developed for this task
- Chapter 6 presents the evaluation of the algorithms we developed and we extract the conclusion about this thesis

CHAPTER 2. BASIC KNOWLEDGE

2.1 Clinical Background

2.1.1 Anatomy of the Brain

The brain is an amazing three-pound organ that controls all functions of the body, interprets information from the outside world, and embodies the essence of the mind and soul. Intelligence, creativity, emotion, and memory are a few of the many things governed by the brain. Protected within the skull, the brain is composed of the cerebrum, cerebellum, and brainstem.

- **Cerebrum**: is the largest part of the brain and is composed of right and left hemispheres. It performs higher functions like interpreting touch, vision and hearing, as well as speech, reasoning, emotions, learning, and fine control of movement.
- **Cerebellum**: is located under the cerebrum. Its function is to coordinate muscle movements, maintain posture, and balance.
- **Brainstem**: acts as a relay center connecting the cerebrum and cerebellum to the spinal cord. It performs many automatic functions such as breathing, heart rate, body temperature, wake and sleep cycles, digestion, sneezing, coughing, vomiting, and swallowing. (1)

2.1.2 Basic Brain tissues

The basic tissues that human brain contains are the grey matter (GM) and the white matter (WM).

Grey matter is a major component of the central nervous system, consisting of neuronal cell bodies, neuropil (dendrites and unmyelinated axons), glial cells (astrocytes and oligodendrocytes), synapses, and capillaries. Grey matter is distinguished from white matter in that it contains numerous cell bodies and relatively few myelinated axons, while white matter contains relatively few cell bodies and is composed chiefly of long-range myelinated axons. The colour difference arises mainly from the whiteness of myelin. In living tissue, grey matter actually has a very light grey colour with yellowish or pinkish hues, which come from capillary blood vessels and neuronal cell bodies. Grey matter refers to unmyelinated neurons and other cells of the central nervous system. It is present in the brain, brainstem and cerebellum and present throughout the spinal cord. Grey matter contains most of the brain's neuronal cell bodies. The grey matter includes regions of the brain involved in muscle control, and sensory perception such as seeing and hearing, memory, emotions, speech, decision making, and self-control.

White matter refers to areas of the central nervous system (CNS) that are mainly made up of myelinated axons, also called tracts. Long thought to be passive tissue, white matter affects learning and brain functions, modulating the distribution of action potentials, acting as a relay and coordinating communication between different brain regions. White matter is named for its relatively light appearance resulting from the lipid content of myelin. However, the tissue of the freshly cut brain appears pinkish-white to the naked eye because myelin is composed largely of lipid tissue veined with capillaries. Its white colour in prepared specimens is due to its usual preservation in formaldehyde. White matter is the tissue through which messages pass between different areas of grey matter within the central nervous system. The white matter is white because of the fatty substance (myelin) that surrounds the nerve fibers (axons). This myelin is found in almost all long nerve fibers and acts as an electrical insulation. This is important because it allows the messages to pass quickly from place to place. Unlike grey matter, which peaks in development in a person's twenties, the white matter continues to develop, and peaks in middle age. (2)

2.1.3 Brain Tumour

A brain tumor, known as an intracranial tumor, is an abnormal mass of tissue in which cells grow and multiply uncontrollably, seemingly unchecked by the mechanisms that control normal cells. More than 150 different brain tumors have been documented, but the two main groups of brain tumors are termed primary and metastatic. (3) Doctors refer to a tumor based on where the tumor cells originated, and whether they are cancerous (malignant) or not (benign).

- **Benign:** The least aggressive type of brain tumor is often called a benign brain tumor. They originate from cells within or surrounding the brain, do not contain cancer cells, grow slowly, and typically have clear borders that do not spread into other tissue.

- **Malignant:** Malignant brain tumors contain cancer cells and often do not have clear borders. They are considered to be life threatening because they grow rapidly and invade surrounding brain tissue.

- **Primary:** Tumors that start in cells of the brain are called primary brain tumors. Primary brain tumors may spread to other parts of the brain or the spine, but rarely to other organs.

- **Metastatic:** Metastatic or secondary brain tumors begin in another part of the body and then spread to the brain. These tumors are more common than primary brain tumors and are named by the location in which they begin.

There are over 120 types of brain and central nervous system tumors. Brain and spinal cord tumors are different for everyone. They form in different areas, develop from different cell types, and may have different treatment options. (4)

2.1.4 Magnetic Resonance Imaging

2.1.4.1 <u>Introduction</u>

Magnetic Resonance Imaging (MRI) is a non-invasive imaging technology that produces three-dimensional detailed anatomical images. It is based on sophisticated technology that excites and detects the change in the direction of the rotational axis of protons found in the water that makes up living tissues. MRIs employ powerful magnets, which produce a strong magnetic field that forces protons in the body to align with that field. When a radiofrequency current is then pulsed through the patient, the protons are stimulated, and spin out of equilibrium, straining against the pull of the magnetic field.

When the radiofrequency field is turned off, the MRI sensors are able to detect the energy released as the protons realign with the magnetic field. The time it takes for the protons to realign with the magnetic field, as well as the amount of energy released, changes depending on the environment and the chemical nature of the molecules. Physicians are able to tell the difference between various types of tissues based on these magnetic properties.

In the brain, MRI can differentiate between white matter and grey matter and can also be used to diagnose aneurysms and tumors. Because MRI does not use x-rays or other radiation, it is the imaging modality of choice when frequent imaging is required for diagnosis or therapy, especially in the brain. However, MRI is more expensive than x-ray imaging or CT scanning. (5)

2.1.4.2 MRI imaging sequences

The most common MRI sequences are T1-weighted and T2-weighted scans. T1-weighted images are produced by using short TE and TR times (Repetition Time (TR) is the amount of time between successive pulse sequences applied to the same slice. Time to Echo (TE) is the time between the delivery of the RF pulse and the receipt of the echo signal). The contrast and brightness of the image are predominately determined by T1 properties of tissue. Conversely, T2-weighted images are produced by using longer TE and TR times. In these images, the contrast and brightness are predominately determined by the T2 properties of tissue. In general, T1 and T2-weighted images can be easily differentiated by looking at the CSF. CSF is dark on T1-weighted imaging and bright on T2-weighted imaging. A third commonly used sequence is the Fluid Attenuated Inversion

Recovery (Flair). The Flair sequence is similar to a T2-weighted image except that the TE and TR times are very long. By doing so, abnormalities remain bright but normal CSF fluid is attenuated and made dark. This sequence is very sensitive to pathology and makes the differentiation between CSF and an abnormality much easier. (6)

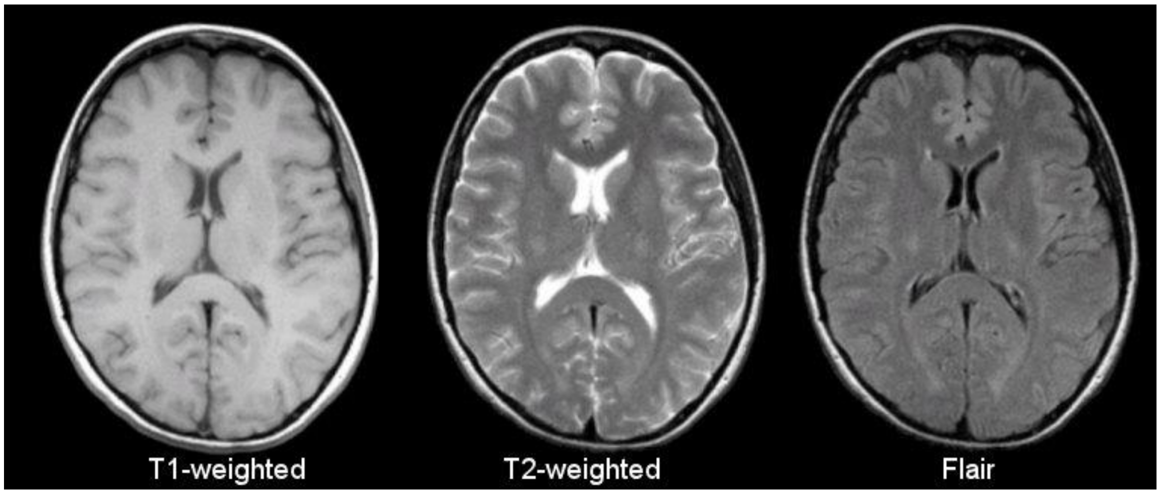


Figure 2.1: Basic types of MRI sequences: T1, T2. Flair

2.2 Introduction to Machine Learning

Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around those functions. Most machine learning algorithms can be divided into the categories of supervised learning and unsupervised learning. (7)

In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were determining whether an image contained a certain object, the training data for a supervised learning algorithm would include images with and without that object (the input), and each image would have a label (the output) designating whether it contained the object. In special cases, the input may be only partially available, or restricted to special feedback. (2)

In unsupervised learning, the algorithm builds a mathematical model from a set of data that contains only inputs and no desired output labels. Unsupervised learning algorithms are used to find structure in the data, like grouping or clustering of data points. Unsupervised learning can discover patterns in the data and can group the inputs into categories. (2)

2.2.1 Machine Learning Tasks

Machine learning enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings. Machine learning tasks are usually described in terms of how the machine learning system should process an example. An example is a collection of features that have been quantitatively measured from some object or event that we want the machine learning system to process. For example, the features of an image are usually the values of the pixels in the image. Some of the most common machine learning tasks include the following:

- **Classification**: In this type of task, the computer program is asked to specify which of k categories some input belongs to. An example of a classification task is object recognition, where the input is an image (usually described as a set of pixel brightness values), and the output is a numeric code identifying the object in the image. (7)

- **Regression**: In this type of task, the computer program is asked to predict a numerical value given some input. This type of task is similar to classification, except that the format of the output is different. An example of a regression task is the prediction of the expected claim amount that an insured person will make (used to set insurance premiums) or the prediction of future prices of securities. These kinds of predictions are also used for algorithmic trading. (7)
- **Transcription**; In this type of task, the machine learning system is asked to observe a relatively unstructured representation of some kind of data and transcribe the information into a discrete textual form. For example, in optical character recognition, the computer program is shown a photograph containing an image of text and is asked to return this text in the form of a sequence of characters (e.g., in ASCII or Unicode format). (7)
- **Machine Translation:** In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language. This is commonly applied to natural languages, such as translating from English to French. (7)
- **Structured output:** Structured output tasks involve any task where the output is a vector (or other data structure containing multiple values) with important relationships between the different elements. This is a broad category and subsumes the transcription and translation tasks described above, as well as many other tasks. One example is parsing—mapping a natural language sentence into a

tree that describes its grammatical structure by tagging nodes of the trees as being verbs, nouns, adverbs, and so on. (7)

- **Anomaly detection:** In this type of task, the computer program sifts through a set of events or objects and flags some of them as being unusual or atypical. An example of an anomaly detection task is credit card fraud detection. By modelling your purchasing habits, a credit card company can detect misuse of your cards. If a thief steals your credit card or credit card information, the thief's purchases will often come from a different probability distribution over purchase types than your own. The credit card company can prevent fraud by placing a hold on an account as soon as that card has been used for an uncharacteristic purchase. (7)
- **Synthesis and sampling:** In this type of task, the machine learning algorithm is asked to generate new examples that are similar to those in the training data. Synthesis and sampling via machine learning can be useful for media applications when generating large volumes of content by hand would be expensive, boring, or require too much time. For example, video games can automatically generate textures for large objects or landscapes, rather than requiring an artist to manually label each pixel. (7)
- **Imputation of missing values:** In this type of task, the machine learning algorithm is given a new example $x \in R^n$, but with some entries x_i of x missing. The algorithm must provide a prediction of the values of the missing entries. (7)

Of course, many other tasks and types of tasks are possible.

2.2.2 Basic Algorithms Overview

2.2.2.1 <u>Logistic Regression</u>

Logistic Regression assumes a Gaussian distribution for the numeric input variables and can model binary classification problems. (8)

2.2.2.2 <u>Linear Discriminant Analysis</u>

Linear Discriminant Analysis or LDA is a statistical technique for binary and multiclass classification. It too assumes a Gaussian distribution for the numerical values. (8)

2.2.2.3 <u>k-Nearest Neighbors</u>

The k-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction. (8)

2.2.2.4 <u>Naive Bayes</u>

Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for input variables using the Gaussian Probability Density Function. (8)

2.2.2.5 Classification and Regression Trees

Classification and Regression Trees (CART or just trees) construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function. (8)

2.2.2.6 Support Vector Machines

One of the most influential approaches to supervised learning is the support vector machine. This model is similar to logistic regression in that it is driven by a linear function $\boldsymbol{w}^{\top}\boldsymbol{x} + \boldsymbol{b}$. Unlike logistic regression, the support vector machine does not provide probabilities, but only outputs a class identity. The SVM predicts that the positive class is present when $\boldsymbol{w}^{\top}\boldsymbol{x} + \boldsymbol{b}$ is positive. Likewise, it predicts that the negative class is present when $\boldsymbol{w}^{\top}\boldsymbol{x} + \boldsymbol{b}$ is negative.

One key innovation associated with support vector machines is the kernel trick. The kernel trick consists of observing that many machine learning algorithms can be written exclusively in terms of dot products between examples. For example, it can be shown that the linear function used by the support vector machine can be re-written as $\mathrm{w}^{\top}\mathrm{x} + \mathrm{b} = \sum_{i=0}^{m} \alpha_{\mathrm{i}} \mathrm{x}^{\top} \mathrm{x}^{(\mathrm{i})}$, where $x^{(i)}$ is a training example, and $\boldsymbol{\alpha}$ is a vector of coefficients. Rewriting the learning algorithm this way enables us to replace $\mathbf{x}$ with the output of a given feature function $\varphi(\mathbf{x})$ and the dot product with a function $k\left(x, x^{(i)}\right) = \varphi(x) \cdot \varphi(x^{(i)})$ called a kernel. The operator represents an inner product analogous to $\varphi(\chi)^{\top}\varphi(\chi^{(i)})$. For some feature spaces, we may not use literally the vector inner product.

In some infinite dimensional spaces, we need to use other kinds of inner products, for example, inner products based on integration rather than summation.

After replacing dot products with kernel evaluations, we can make predictions using the function $f(x) = b + \sum_i a_i k(x, x^{(i)})$. This function is nonlinear with respect to **x**, but the relationship between φ(**x**) and f(**x**) is linear. Also, the relationship between **α** and f(**x**) is linear. The kernel-based function is exactly equivalent to preprocessing the data by applying φ(**x**) to all inputs, then learning a linear model in the new transformed space.

The kernel trick is powerful for two reasons. First, it enables us to learn models that are nonlinear as a function of **x** using convex optimization techniques that are guaranteed to converge efficiently. This is possible because we consider φ fixed and optimize only **α**, that is, the optimization algorithm can view the decision function as being linear in a different space. Second, the kernel function k often admits an implementation that is significantly more computationally efficient than naively constructing two φ(**x**) vectors and explicitly taking their dot product.

In some cases, φ(**x**) can even be infinite dimensional, which would result in an infinite computational cost for the naive, explicit approach. In many cases, $k(x, x'^{)})$ is a nonlinear, tractable function of **x** even when ϕ (**x**) is intractable. As an example of an infinite-dimensional feature space with a tractable kernel, we construct a feature mapping ϕ (x) over the nonnegative integers x. Suppose that this mapping returns a vector containing x ones followed by infinitely many zeros. We can write a kernel function

$k\left(x, x^{(i)}\right) = \min(x, x^{(i)})$ that is exactly equivalent to the corresponding infinite-dimensional dot product.

The most commonly used kernel is the **Gaussian kernel,** $k(u, v) = N(u - v; 0, \sigma^2 I)$, where $N(x; \mu, \Sigma)$ is the standard normal density. This kernel is also known as the radial basis function (RBF) kernel, because its value decreases along lines in **v** space radiating outward from **u**. The Gaussian kernel corresponds to a dot product in an infinite-dimensional space, but the derivation of this space is less straightforward than in our example of the min kernel over the integers.

We can think of the Gaussian kernel as performing a kind of template matching. A training example **x** associated with training label y becomes a template for class y. When a test point $\boldsymbol{x}'$ is near **x** according to Euclidean distance, the Gaussian kernel has a large response, indicating that $\boldsymbol{x}'$ is very similar to the **x** template. The model then puts a large weight on the associated training label y. Overall, the prediction will combine many such training labels weighted by the similarity of the corresponding training examples.

Support vector machines are not the only algorithm that can be enhanced using the kernel trick. Many other linear models can be enhanced in this way. The category of algorithms that employ the kernel trick is known as kernel machines, or kernel methods.

A major drawback to kernel machines is that the cost of evaluating the decision function is linear in the number of training examples, because the i-th example contributes a term $a_i k(\boldsymbol{x}, \boldsymbol{x}^{(i)})$ to the decision function. Support vector machines are able to mitigate this by learning an $\boldsymbol{\alpha}$ vector that contains mostly zeros. Classifying a new example then requires evaluating the kernel function only for the training examples that have nonzero a_i. These training examples are known as support vectors. (7)

2.3 Introduction to Deep Learning

Deep learning has a long history and many aspirations. Several proposed approaches have yet to entirely bear fruit. Several ambitious goals have yet to be realized. (7)

Modern deep learning provides a powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples. Other tasks, that cannot be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now. (7)

2.3.1 Types of Neural Networks

2.3.1.1 Multilayer Perceptrons

A Perceptron is a single neuron model that was a precursor to larger neural networks. It is a field of study that investigates how simple models of biological brains can be used to solve difficult computational tasks like the predictive modeling tasks we see in machine learning. The goal is not to create realistic models of the brain, but instead to develop robust algorithms and data structures that we can use to model difficult problems.

The power of neural networks come from their ability to learn the representation in your training data and how to best relate it to the output variable that you want to predict. In this sense neural networks learn a mapping. Mathematically, they are capable of learning any mapping function and have been proven to be a universal approximation algorithm. The predictive capability of neural networks comes from the hierarchical or multilayered structure of the networks. The data structure can pick out (learn to represent) features at different scales or resolutions and combine them into higher-order features. For example from lines, to collections of lines to shapes.

The building block for neural networks are artificial neurons. These are simple computational units that have weighted input signals and produce an output signal using an activation function.

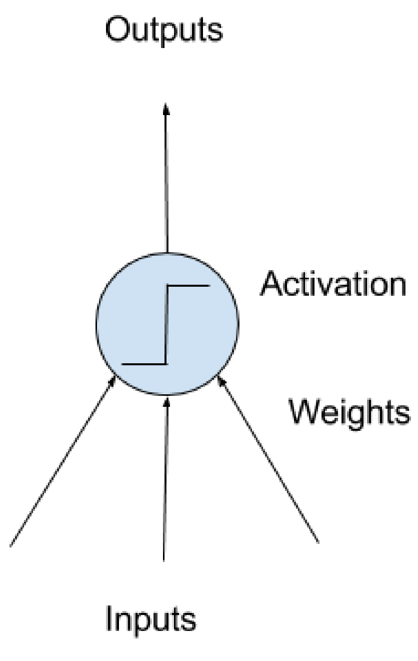


Figure 2.2: Model of a Simple Neuron

Like linear regression, each neuron also has a bias which can be thought of as an input that always has the value 1.0 and it too must be weighted. For example, a neuron may have two inputs in which case it requires three weights. One for each input and one for the bias.

Weights are often initialized to small random values, such as values in the range 0 to 0.3, although more complex initialization schemes can be used. Like linear regression, larger weights indicate increased complexity and fragility of the model. It is desirable to keep weights in the network small and regularization techniques can be used.

The weighted inputs are summed and passed through an activation function, sometimes called a transfer function. An activation function is a simple mapping of summed weighted input to the output of the neuron. It is called an activation function because it governs the threshold at which the neuron is activated and the strength of the output signal. Historically simple step activation functions were used where if the summed input was above a threshold, for example 0.5, then the neuron would output a value of 1.0, otherwise it would output a 0.0.

Traditionally nonlinear activation functions are used. This allows the network to combine the inputs in more complex ways and in turn provide a richer capability in the functions they can model. Nonlinear functions like the logistic function also called the sigmoid function were used that output a value between 0 and 1 with an s-shaped distribution, and the hyperbolic tangent function also called Tanh that outputs the same distribution over the range -1 to +1. More recently the rectifier activation function has been shown to provide better results. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less. The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation.

Neurons are arranged into networks of neurons. A row of neurons is called a layer and one network can have multiple layers. The architecture of the neurons in the network is often called the network topology.

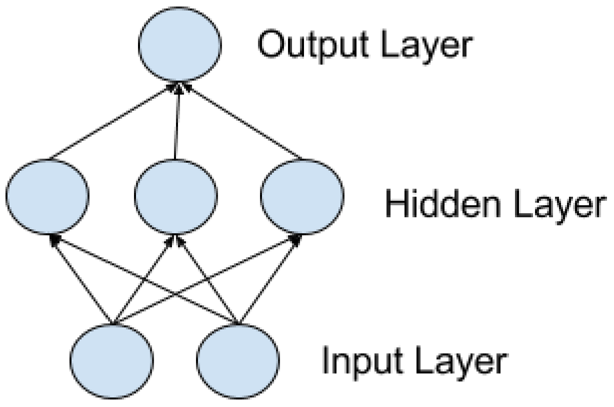


Figure 2.3: Model of a Simple Network

The bottom layer that takes input from your dataset is called the visible layer, because it is the exposed part of the network. Often a neural network is drawn with a visible layer with one neuron per input value or column in your dataset. These are not neurons as described above, but simply pass the input value through to the next layer.

Layers after the input layer are called hidden layers because they are not directly exposed to the input. The simplest network structure is to have a single neuron in the hidden layer that directly outputs the value. Given increases in computing power and efficient libraries, very deep neural networks can be constructed. Deep learning can refer to having many hidden layers in your neural network. They are deep because they would have been unimaginably slow to train historically, but may take seconds or minutes to train using modern techniques and hardware.

The final hidden layer is called the output layer and it is responsible for outputting a value or vector of values that correspond to the format required for the problem. The choice of activation function in the output layer is strongly constrained by the type of problem that you are modeling. For example:

- A regression problem may have a single output neuron and the neuron may have no activation function.
- A binary classification problem may have a single output neuron and use a sigmoid activation function to output a value between 0 and 1 to represent the probability of predicting a value for the primary class. This can be turned into a crisp class value by using a threshold of 0.5 and snap values less than the threshold to 0 otherwise to 1.
- A multiclass classification problem may have multiple neurons in the output layer, one for each class (e.g. three neurons for the three classes in the famous iris flowers classification problem). In this case a softmax activation function may be used to output a probability of the network predicting each of the class values. Selecting the output with the highest probability can be used to produce a crisp class classification value. (8)

2.3.1.2 Convolutional Neural Networks

Given a dataset of grayscale images with the standardized size of 32×32 pixels each, a traditional feedforward neural network would require 1,024 input weights (plus one bias). This is fair enough, but the flattening of the image matrix of pixels to a long vector of pixel values loses all of the spatial structure in the image. Unless all of the images are perfectly resized, the neural network will have great difficulty with the problem.

Convolutional Neural Networks expect and preserve the spatial relationship between pixels by learning internal feature representations using small squares of input data. Features are learned and used across the whole image, allowing for the objects in the images to be shifted or translated in the scene and still detectable by the network. It is this reason why the network is so useful for object recognition in photographs, picking out digits, faces, objects and so on with varying orientation. In summary, below are some of the benefits of using convolutional neural networks:

- They use fewer parameters (weights) to learn than a fully connected network.
- They are designed to be invariant to object position and distortion in the scene.
- They automatically learn and generalize features from the input domain.

There are three types of layers in a Convolutional Neural Network:

1. Convolutional Layers.
2. Pooling Layers.
3. Fully-Connected Layers.

Convolutional layers are comprised of filters and feature maps. The filters are essentially the neurons of the layer. They have both weighted inputs and generate an output value like a neuron. The input size is a fixed square called a patch or a receptive field. If the convolutional layer is an input layer, then the input patch will be pixel values. If they deeper in the network architecture, then the convolutional layer will take input from a feature map from the previous layer.

The feature map is the output of one filter applied to the previous layer. A given filter is drawn across the entire previous layer and moved one pixel at a time. Each position results in an activation of the neuron and the output is collected in the feature map. You can see that if the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by (field width - 1) input values.

The distance that filter is moved across the input from the previous layer each activation is referred to as the stride. If the size of the previous layer is not cleanly divisible by the size of the filter's receptive field and the size of the stride then it is possible for the receptive field to attempt to read off the edge of the input feature map. In this case, techniques like zero padding can be used to invent mock inputs with zero values for the receptive field to read.

The pooling layers down-sample the previous layers feature map. Pooling layers follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous layer's feature map. As such, pooling may be considered a technique to compress or generalize feature representations and generally reduce the overfitting of the training data by the model.

They too have a receptive field, often much smaller than the convolutional layer. Also, the stride or number of inputs that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap. Pooling layers are often very simple, taking the average or the maximum of the input value in order to create its own feature map.

Fully connected layers are the normal flat feedforward neural network layer. These layers may have a nonlinear activation function or a softmax activation in order to output probabilities of class predictions. Fully connected layers are used at the end of the network after feature extraction and consolidation has been performed by the convolutional and pooling layers. They are used to create final nonlinear combinations of features and for making predictions by the network. (8)

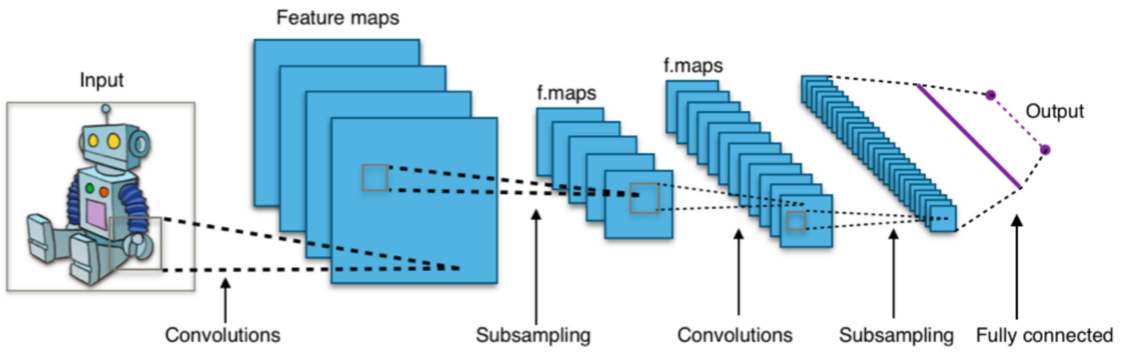


Figure 2.4: Typical Convolutional Neural Network Architecture

2.3.2 Optimization Algorithms

Optimization algorithms helps us to minimize (or maximize) an Objective function (another name for Error function) E(x) which is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values(Y) from the set of predictors(X) used in the model. Some popular optimizers are presented below:

- **Stochastic Gradient Descent**, or SGD for short, is one of the simplest optimization algorithms. It uses just one static learning rate for all parameters during the entire training phase. The static learning rate does not imply an equal update after every minibatch. As the optimizers approach an (sub)optimal value, their gradients start to decrease.
- **AdaGrad** is very similar to SGD. The key difference in design is that AdaGrad uses Adaptive gradients — it has a different learning rate for every single parameter in the neural network. Since not all parameters in a network are equally important, updating them in a different way makes perfect sense. AdaGrad updates the learning rate for each parameter based on the frequency with which it's updated. Frequently updated parameters are trained very carefully with a low learning rate. Updating these parameters too quickly could result in an irreversible distortion — wasting the parameters usefulness. Infrequently updated parameters are trained with higher learning rates in an attempt to make them more effective. This risk can be taken, because these parameters are virtually useless in the first place.

- AdaGrad has a problem where after a few batches, the learning rates become low — resulting in a long training time. **Root Mean Square Propagation** (RMSProp), attempts to solve this problem by exponentially decaying the learning rates. This makes RMSProp more volatile. By using past learning rates, both AdaGrad and RMSProp use momentum for updating. This can be compared to rolling a ball (state of the neural network) down a hill (the graph of a cost function). The longer the ball moves in a certain direction, the faster it goes, giving the ball higher momentum. The momentum of the ball is useful because it allows the network to 'roll over' local minima — not getting stuck in them. (9)

2.3.3 Loss Functions

Deep learning neural networks are trained using the stochastic gradient descent and other optimization algorithms. As part of the optimization algorithm, the error for the current state of the model must be estimated repeatedly. This requires the choice of an error function, conventionally called a loss function, that can be used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. Neural network models learn a mapping from inputs to outputs from examples and the choice of loss function must match the framing of the specific predictive modelling problem, such as classification or regression. Further, the configuration of the output layer must also be appropriate for the chosen loss function. (10) Some popular loss functions are presented below:

- **Mean Squared Error**: As the name suggests, this loss is calculated by taking the mean of squared differences between actual (target) and predicted values

- **Binary Crossentropy**: This loss function measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.
- **Categorical crossentropy**: This loss function will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

2.4 Data Pre-processing

2.4.1 Skull Stripping

The skull stripping method is an important area of study in brain image processing applications. It acts as preliminary step in numerous medical applications as it increases speed and accuracy of diagnosis in manifold. It removes non-cerebral tissues like skull, scalp, and dura from brain image.

2.4.2 Standardize Data

Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1. It is most suitable for techniques that assume a Gaussian distribution in the input variables and work better with rescaled data, such as linear regression, logistic regression and linear discriminate analysis. (8)

2.4.3 Augmentation

Data augmentation is another way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data. The field of data augmentation is not new, and in fact, various data augmentation techniques have been applied to specific problems. The main techniques fall under the category of data warping, which is an approach which seeks to directly augment the input data to the model in data space.

A very generic and accepted current practice for augmenting image data is to perform geometric and color augmentations, such as reflecting the image, cropping and translating the image, and changing the color palette of the image. All of the transformation are affine transformation of the original image that take the form : $y = Wx + b$

Furthermore, data augmentation has found applicability in areas outside simply creating more data. It has shown to be helpful in generalizing from computer models to real-world tasks. (11)

2.4.4 Discrete Wavelet Transform

Discrete Wavelet transform (DWT) is a mathematical tool for hierarchically decomposing an image. The transform is based on small waves, called wavelets, of varying frequency and limited duration. Wavelet transform provides both frequency and spatial description of an image. Unlike conventional Fourier transform, temporal information is retained in this transformation process. Wavelets are created by translations and dilations of a fixed function called mother wavelet.

DWT is the multiresolution description of an image the decoding can be processed sequentially from a low resolution to the higher resolution. The DWT splits the signal into high and low frequency parts. The high frequency part contains information about the edge components, while the low frequency part is split again into high and low frequency parts.

In two dimensional applications, for each level of decomposition, we first perform the DWT in the vertical direction, followed by the DWT in the horizontal direction. After the first level of decomposition, there are 4 sub-bands: LL1, LH1, HL1 and HH1. For each successive level of decomposition, the LL sub-band of the previous level is used as the input. To perform second level decomposition, the DWT is applied to LL1 band which decompose the LL1 band into the four sub-bands LL2, LH2, HL2 and HH2.

To perform third level decomposition, the DWT is applied to LL2 band which decompose this band into the four sub-bands: LL3, LH3, HL3, HH3. This results in 10 sub-bands per component. LH1, HL1 and HH1 contain the highest frequency bands present in the image, while LL3 contains the lowest frequency band. The three-level DWT decomposition is show in Figure 2.5. (12)

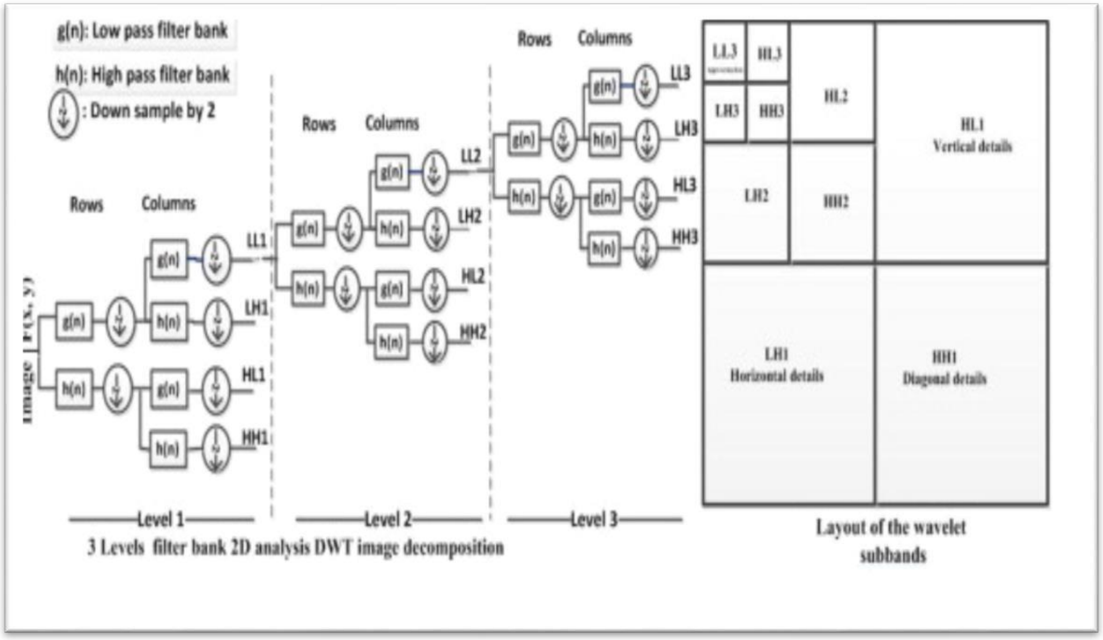


Figure 2.5: 3_Level discrete wavelet decompositions

2.4.5 Principal Component Analysis

Principal component analysis (PCA) is often used as a preliminary data reduction technique to identify a small number of factors that explain the majority of the variance observed in a much larger number of measured variables. The central idea of PCA is to reduce the dimensionality of a data set in which there are a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This reduction is achieved by transforming the data to a new set of variables, the principal components, that are minimally correlated.

Principal components may be viewed as the eigenvectors of a positive semidefinite symmetric matrix. The eigenvectors are “characteristic” vectors of a matrix. They are unique in that they remain directionally invariant under linear transformation by its parent matrix. Thus, the definition and computation of principal components are straightforward and have a wide variety of applications. The general form for the

Equation to compute scores on the first (main) component extracted (created) in a PCA: C1=b11(X1)+b12(X2)+⋯+b1p(Xp) where C1, the subject's score on principal component 1 (the first component extracted); b1p, the regression coefficient (or weight) for observed variable p, as used in creating principal component 1; and Xp, the subject's score on observed variable p. (13)

2.5 Entropy

Entropy is a tool to measure the uncertainty of the information content in given systems, and it is widely applied in signal processing, information theory, pattern recognition, and so on. Shannon entropy ,log energy entropy, Renyi Entropy and Tsallis entropy are some of the typical types of entropy. (14)

2.5.1 Wavelet Entropy

Although the coefficients by DWT can reveal the local characteristics of a signal, the number of such coefficients is usually so huge that is hard to use them as features for classification directly. Therefore, some high-level features may derive from these coefficients for better classification. (14) A measure estimated by the wavelet coefficients to provide quantitative information about the order/complexity of signals is the wavelet entropy.

In order to calculate the wavelet entropy of the signal, the wavelet coefficients $C_j(k)$ were obtained at each resolution level j. The energy at each time sample k can be calculated by equation $E(k) = \sum_{j=1}^{J} |\, C_j(k)|^2$ and total energy by $Etot = \sum_{j=1}^{J} \sum_{k=1}^{N} |\, C_j(k)|^2 = \sum_j E_j$. The relative wavelet energy, which defines energy's probability distribution in scales is given by $p_j = \frac{E_j}{Etot}$. Obviously, $\sum_j p_j = 1$ and the distribution p_j is considered as a time-scale density. The wavelet entropy is, in turn, defined as $H_{WT}(p) = -\sum_{j=1}^{J} p_j . log_2[p_j]$. The value of wavelet entropy can provide estimation of the order of the decomposed signal and subsequently of the order of the system it represents. (15)

2.6 Pixel values in Greyscale

Each of the pixels that represents an image stored inside a computer has a pixel value which describes how bright that pixel is, and what colour it should be. For a greyscale image, the pixel value is a single number that represents the brightness of the pixel. The most common pixel format is the byte image, where this number is sorted as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black and 255 is taken to be white. Values in between make up different shades of grey. (16)

2.7 Evaluate and Tuning Machine and Deep Learning Algorithms

A model evaluation is an estimate that we can use to talk about how well we think the method may actually do in practice. It is not a guarantee of performance. Once we estimate the performance of our algorithm, we can then re-train the final algorithm on the entire training dataset and get it for operational use. Some of the techniques that we can use to split up out training dataset and create useful estimated of performance for our algorithms are below :

- **Train and Test sets**: The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing sets. We can take our original dataset and split in two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results.
- ***k*-fold Cross-Validation**: Cross-validation is an approach that we can use to estimate the performance of an algorithm with less variance than a single train-set test split. It works by splitting the dataset into *k*-parts (e.g *k*=5 or *k*=10). Each split of the data is called a fold. The algorithm is trained on *k*-1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross-validation we end up with different performance scores that we can summarize using a mean and a standard deviation.
- **Grid Search Parameter Tuning**: Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. (8)

2.7.1 Algorithms Evaluation Metrics

2.7.1.1 <u>Model Accuracy</u>

Model accuracy in terms of classification models can be defined as the ratio of correctly classified samples to the total number of samples:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

For binary classification model as in my thesis, the accuracy can be defined as: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

- True Positive (TP): A true positive is an outcome where the model correctly predicts the positive class.
- True Negative (TN): A true negative is an outcome where the model correctly predicts the negative class.
- False Positive (FP): A false positive is an outcome where the model incorrectly predicts the positive class.
- False Negative (FN): A false negative is an outcome where the model incorrectly predicts the negative class.

2.7.1.2 Precision and Recall

In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labelled as belonging to the positive class) divided by the total number of elements labelled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labelled as belonging to the class). $Precision = \frac{TP}{(TP+FP)}$. High precision means that an algorithm returned substantially more relevant results than irrelevant ones.

Recall is defined as the number of true positives divided by the total number of elements that actually belong to the positive class (i.e. the sum of true positives and false negatives, which are items which were not labelled as belonging to the positive class but should have been). $Recall = \frac{TP}{(TP+FN)}$. High recall means that an algorithm returned most of the relevant results. (17)

2.7.1.3 <u>Specificity</u>

Specificity is defined as the proportion of actual negatives, which got predicted as the negative (or true negative). This implies that there will be another proportion of actual negative, which got predicted as positive and could be termed as false positives. Mathematically, specificity can be calculated as the following:

$$Specificity = \frac{TN}{(TN + FP)}$$

2.7.1.4 <u>False Negative Rate (FNR) and False Positive Rate (FPR)</u>

FNR (False Negative Rate) or (Type II Error) is the error of proportion to recognize as healthy (negatives) the images that are actually tumorous (positives). We must say that it is very important in our case to be very low because people that have tumour may not get the proper healthcare because of the results.

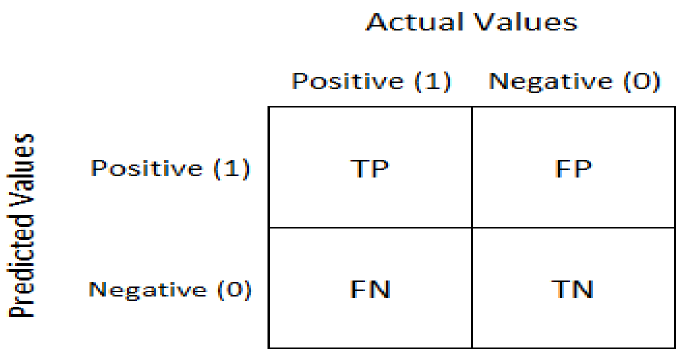
$$FNR = \frac{FN}{TP + FN}$$

FPR (False Positive Rate) or (Type I Error) is the error of proportion to recognize as tumorous (positives) the images that are actually healthy (negatives). We need that to have very low values.

$$FPR = \frac{FP}{TN + FP}$$

2.7.1.5 Confusion Matrix

The confusion matrix is a handy presentation of the accuracy of a model with two or more classes. The table presents predictions on the x-axis and true outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm. For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for prediction = 0 and actual = 0, whereas predictions for 0 that were actually 1 appear in the cell for prediction = 0 and actual = 1. (8)



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figure 2.6: Confusion Matrix

CHAPTER 3. TOOLS USED IN THESIS

3.1 Python

Python is a general purpose interpreted programming language. It is easy to learn and use primarily because the language focuses on readability. It's a dynamic language and very suited to interactive development and quick prototyping with the power to support the development of large applications. It is also widely used for machine learning and data science because of the excellent library support and because it is a general purpose programming language.

3.2 SciPy

SciPy is an ecosystem of Python libraries for mathematics, science and engineering. It is an add-on to Python that is we will need for machine learning. The SciPy ecosystem is comprised of the following core modules relevant to machine learning:

- **NumPy**: A foundation for SciPy that allows us to efficiently work with data in arrays
- **Matplotlib**: Allows us to create 2D charts and plots from data
- **Pandas**: Tools and data structured to organize and analyse data

3.3 Scikit-learn

The scikit-learn library is how we can develop and practice machine learning in Python. It is built upon and requires the SciPy ecosystem. The name scikit suggests that it is a SciPy plug-in or toolkit. The focus of the library is machine learning algorithms for classification, regression, clustering and more. It is also provides tools for related tasks such as evaluating models, tuning parameters and pre-processing data. (8)

3.4 Google Colaboratory

Google Colaboratory or Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly it does not require a setup. Colab supports many popular machine learning libraries which can be easily loaded in our notebook. We can perform the following using Google Colab.

- Write and execute code in Python
- Document our code that supports mathematical equations
- Create upload and share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets
- Integrate TensorFlow, Keras and other libraries
- Free Cloud service with free CPU and GPU

3.5 TensorFlow

TensorFlow is a Python library for fast numerical computing created and released by Google. It is a foundation library that can be used to create deep learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow. Unlike other numerical libraries indented for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines. (8)

3.6 Keras

Keras is a minimalist Python library for deep learning that can run on top of TensorFlow. It was developed to make developing deep learning models as fast and easy as possible for research and development. It run on Python 2.7 or 3.6 and can seamlessly execute on GPUs and CPUs given the underlying frameworks. It is released under the permissive MIT license. Keras was developed and maintained by Francois Chollet, a Google engineer using four guiding principles:

- **Modularity**: A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
- **Minimalism**: The library provides just enough to achieve an outcome, no frills and maximizing readability.

- **Extensibility**: New components are intentionally easy to add and use within the framework, intended for developers to trial and explore new ideas.
- **Python**: No separate model files with custom formats. Everything is native Python. (8)

CHAPTER 4. DATASET EXPLORING

4.1 Description of Dataset

Our image dataset consists of MRI brain images of male and female adult persons. Images are referred to T2 MRI of the axial plane. There are 81 non tumorous cases and 210 cases that were diagnosed with brain tumour.

4.2 MRI Data Acquisition

4.2.1 Healthy MRI Data

The healthy MRI data used in the current study were generated at the General Hospital of Chania “Agios Georgios”, Crete, using a T2 FRFSE-XL (Fast Recovery Fast Spin Echo with -90◦ refocus pulse) sequence under a General Electric Signal HDxT at 1.5 Tesla, with the following hardware : 16 channels HNS Coil made by GE Healthcare and scanning parameters set as :

- Slick thickness = 5mm
- Spacing = 10%
- TR = 2200ms
- TE = 99.3 ms
- Matrix = 320X320 interpolated to 512X512
- Bandwidth = 31.2 kHz
- Nex/Averages = 2

ASSET acceleration factor was set 1.75 (Parallel imaging Technique working on image domain provided from GE Healthcare).

4.2.2 Unhealthy MRI Data

Unhealthy MRI data acquired from BraTS, a multimodal segmentation challenge. All Brats multimodal scans are available as NIfTI files (.nii.gz) and describe T2-weigthed volumes, and were acquired with different clinical protocols and various scanners from multiple (n=19) institutions. The data are distributed after their pre-processing, i.e. interpolated to the same resolution and skull-stripped.

4.3 Methods used to prepare the Dataset

Because of the different sources of the collected MRI images it was necessary to be a certain preparation of them, so that our dataset to be consistent. Images from BraTS were in NIfTI file format as we said before, while images from General hospital of Chania were in jpeg format. Also, healthy images were in various sizes while every unhealthy image consisted of 240*240 pixels. So all our data modified using a python script, in two dimensional images in jpeg file format. The new images consist of 240*240 pixels in grayscale type with 8-bit grey level rate. After that, we applied a skull-stripping algorithm ,using python, to healthy images and the skull was removed. Now every single MRI image of our dataset is in the same format, same size and skull-stripped. Last but not least, we created a subset to validate our algorithms by taking 19 unhealthy and 15 healthy cases from our MRI dataset. This dataset is called validation dataset.

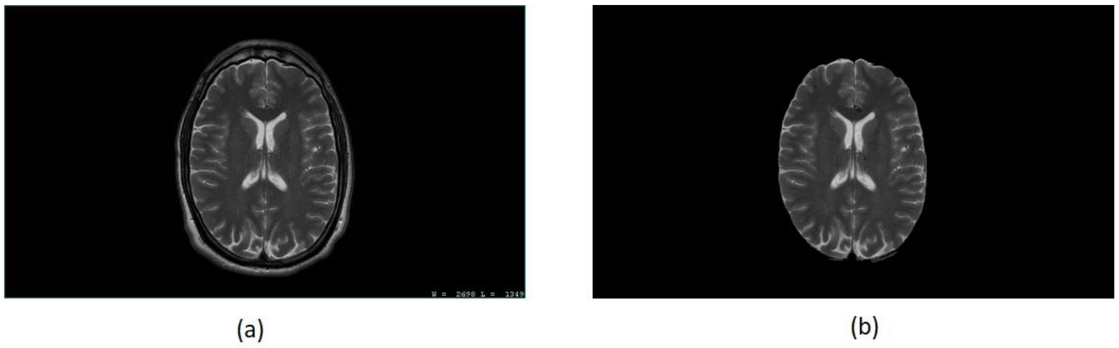


Figure 4.1 : Example of the application of skull stripping algorithm. (a) Initial MRI image, (b) MRI image after the application of skull stripping algorithm

CHAPTER 5. ALGORITHMS USED IN THESIS

In this thesis, we used machine learning and deep learning algorithms to classify healthy and unhealthy cases using different scenarios. The programming language we used was Python.

5.1 Machine Learning Algorithms

5.1.1 Scenario 1: Wavelet Entropy of Image and unbalanced dataset

In this scenario we used the entire dataset which consists of 191 unhealthy and 66 healthy cases after we took of the images for validation subset. The ratio is approximately 70% unhealthy cases and 30% healthy cases. We can observe that the dataset is quite unbalanced.

5.1.1.1 <u>Whole Image: Algorithm 1.1</u>

In this algorithm, we first applied 3-level Discrete Wavelet Transform using ‘haar’ wavelet to each image. Next we calculated the wavelet entropy of the whole image by applying the algorithm we described in chapter 2. After that, using pandas library we created our dataset in this form: [Wavelet Entropy value of image , Target]. Target is ‘1’ if the case is unhealthy and ‘0’ if the case is healthy. Next we split-out our dataset to training and test set using 80% of our dataset for training and 20% for testing. Then we standardized our training set and we did a 10-fold split, in order to do grid search to find the best values of parameters ‘C’ and ‘kernel’ for Support Vector Machine algorithm. With this method we found out that the best values were 2.0 for ‘C’ and sigmoid for ‘kernel’. After that we applied Support Vector machine with the above parameters to the training set, and we evaluated on the test set and we got accuracy 0.762557. After the tuning of Support Vector machine we applied the algorithm with optimal parameters to the entire dataset. Then we evaluated our algorithm using our validation dataset and the results are in the next tables:

Table 1: Evaluation metrics on Validation Dataset for algorithm 1.1

	precision	**recall**
0	0.38	0.2
1	0.54	0.74

Table 2: Confusion Matrix of Validation Dataset for algorithm 1.1

	Negative	**Positive**
Negative	3	12
Positive	5	14

Total accuracy on Validation Dataset was 0.53.

5.1.1.2 Image Divided in Quarters: Algorithm 1.2

In this algorithm the first thing we did was to divide each image into quarters. That means that our dataset of images for this algorithm consists of 4 equal images with 120 * 120 pixels in grayscale.

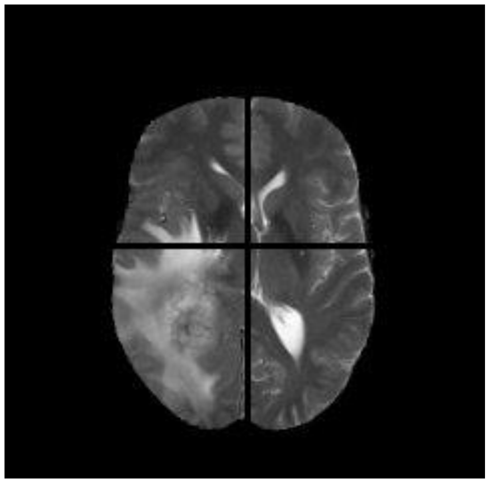


Figure 5.1: Example of division in quarters of image

After that, we named the images area 1, area 2, area 3 and area 4. Then we applied Discrete Wavelet Transform and we calculated wavelet entropy for each area. Therefore, we end up with our dataset in this form: [WE of aera1, WE of aera2, WE of aera3, WE of aera4, Target]. Next, we split-out our dataset to training and test set using 80% of our dataset for training and 20% for testing. Then we standardized our training set and we did a 10-fold split, in order to do grid search to find the best values of parameters ‘C’ and ‘kernel’ for Support Vector Machine algorithm. We found out that the best value for ‘C’ was 1.0 and for ‘kernel’ was ‘rbf’. We applied Support Vector Machine with those values to the training set and we evaluated the algorithm on the test set and we got accuracy 0,8181. Then we applied the algorithm to the entire dataset and we evaluated on the validation dataset with the results below:

Table 3: Evaluation metrics on Validation Dataset for algorithm 1.2

	precision	**recall**
0	0.69	0.6
1	0.71	0.79

Table 4: Confusion Matrix of Validation Dataset for algorithm 1.2

	Negative	**Positive**
Negative	9	6
Positive	4	15

Total Accuracy on Validation Dataset was 0.705

5.1.2 Scenario 2: Wavelet Entropy of Image and balanced dataset

This scenario is the same as scenario 1 except the dataset. In this scenario we used a balanced dataset in order to develop our algorithms. What we did was to manually undersample images from our unhealthy cases. We picked 66 unhealthy cases out of 191 and all the healthy cases which are 66. Therefore our dataset is balanced with ratio 50% healthy cases and 50% unhealthy cases.

5.1.2.1 Whole Image: Algorithm 2.1

This algorithm is identical to algorithm 1.1. So we applied Discrete Wavelet Transform to each image. Then we calculated wavelet entropy using the coefficients of the transform for each image, therefore our dataset was in this form : [Wavelet Entropy of Image, Target]. Next we split-out our dataset to training and test set using 80% of our dataset for training and 20% for testing. Then we standardized our training set and we did a 10-fold split, in order to do grid search to find the optimal values of parameters ‘C’ and ‘kernel’ for Support Vector Machine algorithm. We found out that optimal values were 2.0 for ‘C’ and ‘rbf’ for ‘kernel’. We applied Support Vector Machine with those values to the training set and we evaluated the algorithm on the test set and we got accuracy 0,8148. Then we applied the algorithm to the entire dataset and we evaluated on the validation dataset with the results below:

Table 5: Evaluation metrics on Validation Dataset for algorithm 2.1

	precision	**recall**
0	0.68	1.0
1	1.0	0.63

Table 6: Confusion Matrix of Validation Dataset for algorithm 2.1

	Negative	**Positive**
Negative	15	0
Positive	7	12

Total Accuracy on Validation Dataset was 0.795

5.1.2.2 Image Divided in Quarters: Algorithm 2.2

This algorithm is identical to algorithm 1.2. We divided images into quarters as we did in algorithm 1.2. After that we applied Discrete Wavelet Transform and we calculated wavelet entropy for each area. Our dataset was in this form: [WE of aera1, WE of aera2, WE of aera3, WE of aera4, Target]. Next we split-out our dataset to training and test set using 80% of our dataset for training and 20% for testing. Then we standardized our training set and we did a 10-fold split, in order to do grid search to find the best values of parameters ‘C’ and ‘kernel’ for Support Vector Machine algorithm. We found out that optimal values were ‘2.0’ for ‘C’ and ‘rbf’ for ‘kernel’. We applied Support Vector Machine with those values to the training dataset and we evaluated the algorithm on the test set and we got accuracy 0,962. Then we applied the algorithm to the whole dataset and we evaluated on the validation dataset with the results below:

Table 7: Evaluation metrics on Validation Dataset for algorithm 2.2

	precision	**recall**
0	0.65	1.0
1	1.0	0.58

Table 8: Confusion Matrix of Validation Dataset for algorithm 2.2

	Negative	**Positive**
Negative	15	0
Positive	8	11

Total Accuracy on Validation Dataset was 0.765

5.1.3 Scenario 3: Pixel values of images and balanced dataset

In this scenario, we used the pixel values in greyscale of our images as input to our algorithm. We also used the balanced dataset for this algorithm, the same as scenario 2. Furthermore, we used Google Colaboratory for this scenario to take advantage of GPU resources due to the big size of our input. In a normal computer it would take hours to execute this algorithm, but in Google Colaboratory it took less than one minute.

5.1.3.1 Whole Image Pixel Values: Algorithm 3.1

In this algorithm we used pixel values of each image as input. So, the first thing we did was to reshape the pixel values matrix for each image, from [240,240] to a vector of 57600 elements. After that, our dataset was in this form: [57600 pixels values in each row, Target]. Next, we split-out our dataset to training and test set using 80% of our dataset for training and 20% for testing. Then we standardized our training set and we did a 10-fold split, in order to do grid search to find the optimal values of parameters ‘C’ and ‘kernel’ for Support Vector Machine algorithm. We found out that optimal values were 0.7 for ‘C’ and ‘sigmoid’ for ‘kernel’. We applied Support Vector Machine with those values to the training dataset and we evaluated the algorithm on the test set and we got accuracy 0,95. After that, we evaluated our algorithm on validation dataset and the results are below:

Table 9: Evaluation metrics on Validation Dataset for algorithm 3.1

	precision	**recall**
0	1.00	0.8
1	0.86	1.00

Table 10: Confusion Matrix of Validation Dataset for algorithm 3.1

	Negative	**Positive**
Negative	12	3
Positive	0	19

Total Accuracy on Validation Dataset was 0.911

5.2 Deep Learning Algorithms

In all deep learning algorithms, we used Convolutional Neural Networks as our models. Furthermore, we ran our algorithms on Google Colaboratory using GPU resources, due to the big size for our inputs. Additionally, in some algorithms we used augmentation method to our balanced dataset, with the following changes:

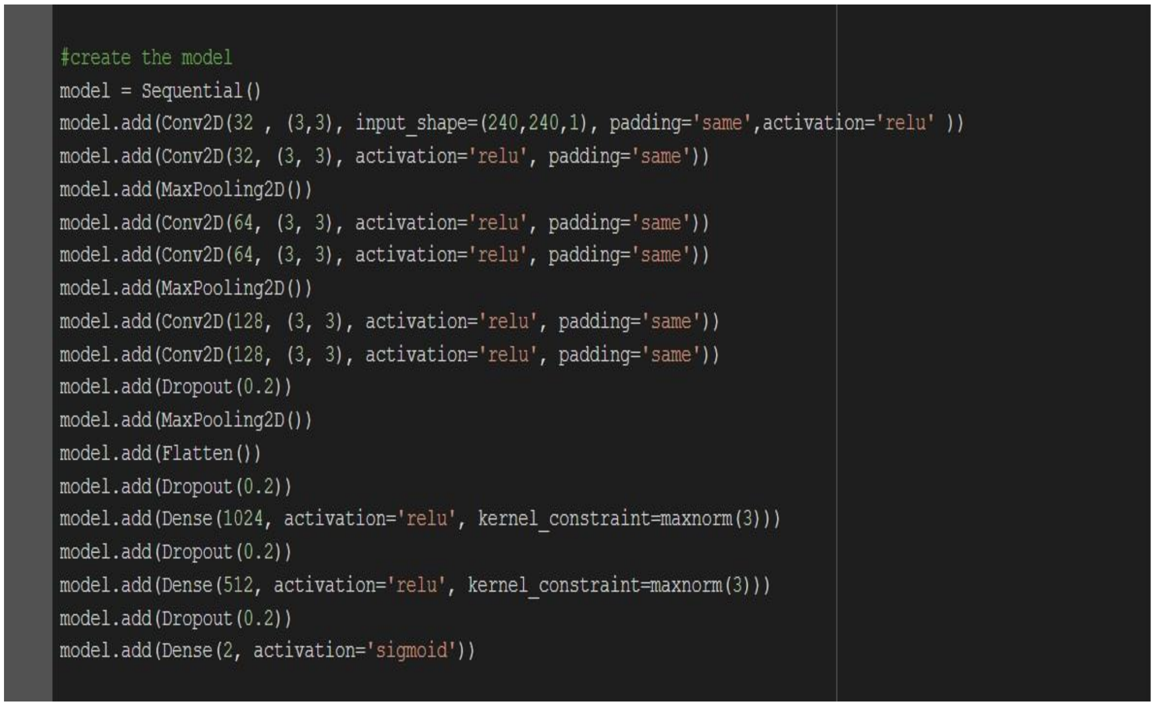
- The rotation of the images
- The left and right flip of the images
- The randomized elastic Gaussian distortion of images

5.2.1 Scenario 4: Pixel Values of Images

In this scenario, the input to our models was the pixel values of our images. In algorithm 4.1, we used our balanced dataset and in algorithm 4.2, we used augmentation method on our balanced dataset and we generated 400 healthy images and 400 unhealthy images.

5.2.1.1 Whole image pixel values and balanced dataset: Algorithm 4.1

In this algorithm, after we loaded our images from the balanced dataset, we reshaped our data in three dimensions, because CNN requires as input, images with three channels. Therefore, our data are now three-channel images (240,240,1). After that, we split out our dataset in train and test set. We used 80 percent of images for training and 20 percent for testing. Next, we normalized pixel values from 0-255 to 0-1. Then, we built our CNN model that is shown in the figure below:



```
#create the model
model = Sequential()
model.add(Conv2D(32 , (3,3), input_shape=(240,240,1), padding='same',activation='relu' ))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
```

Figure 5.2: CNN model for algorithm 4.1

First, we created a Sequential model. The Sequential model API allows us to stack all layers sequentially. As we can see we have three blocks, each one has two Conv2D layers, the ReLU activation function and a MaxPool2D of (2,2). The width and the height of the convolutional window are [3.3] and we change the size of Conv2D filters. In the first block, we have size of 32, in the second block, we have 64 and in the last one, we have 128. In the end, we have the classifier where we add Flatten Layers. Conv2D layers extract and learn spatial features, which then passed to two dense layers after they have been flattened. The last layer has output of two and sigmoid activation function, this is because we have two states, healthy and unhealthy, so we want the model to output a probability between 0 and 1. The closest to 0 if the case is healthy and to 1 if the case is unhealthy. We have also added a few Dropout layers with dropout rate set to 20%, meaning one in 5 inputs will be randomly excluded from each update cycle. The Dropout technique is a very efficient way for reducing overfitting in neural networks.

We trained the model for 10 epochs, using ‘binary_crossentropy’ as loss function and ‘RMSprop’ with learning rate 0.0001, as optimizer. Accuracy on the test set was 1.0. Then we used our validation dataset, which we have mentioned before, to test our model on unseen data. The results are below:

Table 11: Evaluation metrics on Validation Dataset for algorithm 4.1

	precision	**recall**
0	0.94	1.00
1	1.00	0.95

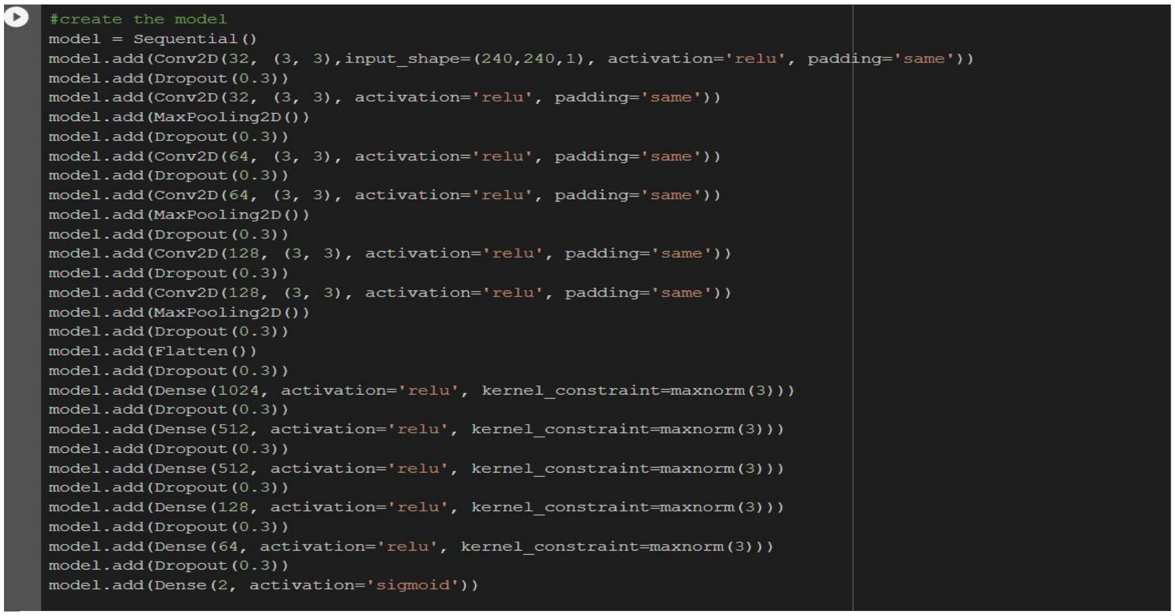
Table 12: Confusion Matrix of Validation Dataset for algorithm 4.1

	Negative	**Positive**
Negative	15	0
Positive	1	18

Total accuracy on Validation Dataset was 0.97

5.2.1.2 Whole image pixel values using Augmentation: Algorithm 4.2

In algorithm 4.2, we used augmentation method to our dataset and created 400 healthy images and 400 unhealthy images. After that, we loaded our images from the augmented dataset and reshaped them to three dimensions (240,240,1). Then we normalized pixel values from 0-255 to 0-1 and we split out our dataset. We used 80 percent of our images for training and 20 percent for testing. Then, we built our CNN model, which is shown in the figure below:



```
#create the model
model = Sequential()
model.add(Conv2D(32, (3, 3),input_shape=(240,240,1), activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Dropout(0.3))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.3))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D())
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.3))
model.add(Dense(2, activation='sigmoid'))
```

Figure 5.3: CNN model for algorithm 4.2

Again, we created a Sequential model in order to stack all layers sequentially. As we can see we have three blocks, each one has two Conv2D layers, the ReLU activation function and a MaxPool2D of (2,2). The width and the height of the convolutional window are [3.3] and we change the size of Conv2D filters. In the first block, we have size of 32, in the second block, we have 64 and in the last one, we have 128. In the end, we have the classifier where we add Flatten Layers. Conv2D layers extract and learn spatial features, which then passed to five dense layers after they have been flattened. The last layer has output of two and sigmoid activation function, this is because we have two states, healthy and unhealthy, so we want the model to output a probability between 0 and 1. We have also added a few Dropout layers with dropout rate set to 30% to prevent overfitting.

We trained the model for 30 epochs, using 'binary_crossentropy' as loss function and 'RMSprop' with learning rate 0.0001, as optimizer. Accuracy on the test set was 0.98. Then we used our validation dataset, to test our model on unseen data. The results are below:

Table 13: Evaluation metrics on Validation Dataset for algorithm 4.2

	precision	**recall**
0	0.88	1.00
1	1.00	0.89

Table 14: Confusion Matrix of Validation Dataset for algorithm 4.2

	Negative	**Positive**
Negative	15	0
Positive	2	17

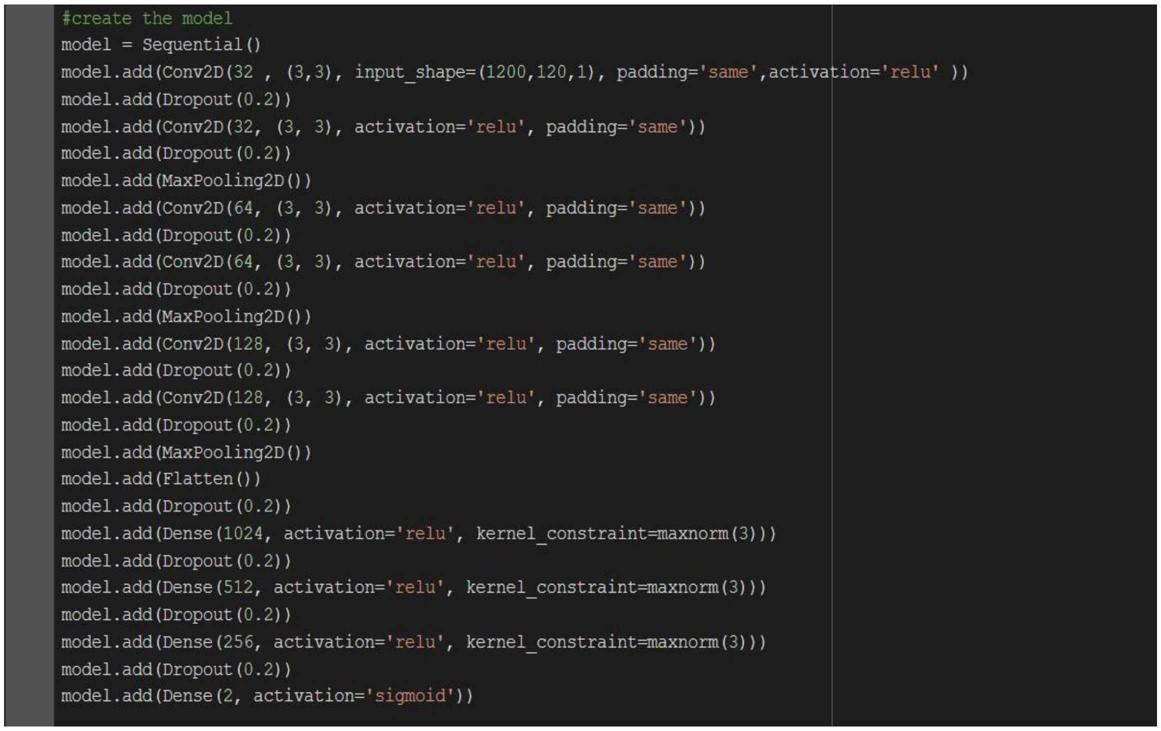
Total accuracy on Validation Dataset was 0.94

5.2.2 Scenario 5: DWT coefficients

In this scenario, we used Discrete Wavelet Transform coefficients of our images as input to our models. In algorithm 5.1 and 5.2 we used our balanced dataset and in algorithm 5.3 we used augmentation method on our balanced dataset and we generated 200 healthy and 200 unhealthy images.

5.2.2.1 DWT coefficients of image and balanced dataset: Algorithm 5.1

In this algorithm, we first applied 3-level decomposition Discrete Wavelet Transform to each image from our balanced dataset, using ‘haar’ wavelet. After that we applied zero padding to coefficients of levels 2 and 3 because we wanted equal size to all the coefficients (120,120). Next we merged all the coefficients in one matrix which size was (1200,120). Then we standardized the matrix and we reshaped it to a three dimensions matrix. Our matrix had the form: (1200,120,1). Additionally, we split out our matrix (dataset), to train and test set, using 80 percent for training and 20 percent for testing. Then we built our Convolutional Neural Network which is shown in the figure below:



```
#create the model
model = Sequential()
model.add(Conv2D(32 , (3,3), input_shape=(1200,120,1), padding='same',activation='relu' ))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(MaxPooling2D())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(MaxPooling2D())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(Dropout(0.2))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(256, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
```

Figure 5.4: CNN model for algorithm 5.1

As we can see, we began with the creation of a Sequential model in order to stack all layers sequentially. We have three blocks, each one has two Conv2D layers, the ReLU activation function and a MaxPool2D of (2,2). The width and the height of the convolutional window are [3.3] and we change the size of Conv2D filters. In the first block, we have size of 32, in the second block, we have 64 and in the last one, we have 128. In the end, we have the classifier where we add Flatten Layers. Conv2D layers extract and learn spatial features, which then passed to three dense layers after they have been flattened. The last layer has output of two and sigmoid activation function, this is because we have two states, healthy and unhealthy, so we want the model to output a probability between 0 and 1. We have also added a few Dropout layers with dropout rate set to 20% to prevent overfitting.

We trained the model for 25 epochs, using ‘binary_crossentropy’ as loss function and ‘RMSprop’ with learning rate 0.0001, as optimizer. Accuracy on the test set was 0.92. Then we used our validation dataset, to test our model on unseen data. The results are below:

Table 15: Evaluation metrics on Validation Dataset for algorithm 5.1

	precision	**recall**
0	1.00	0.93
1	0.95	1.00

Table 16: Confusion Matrix of Validation Dataset for algorithm 5.1

	Negative	**Positive**
Negative	14	1
Positive	0	19

Total accuracy on Validation Dataset was 0.97

5.2.2.2 DWT coefficients of image using PCA: Algorithm 5.2

In this algorithm, we used Principal Component Analysis in order to reduce the dimensions of Discrete Wavelet Transform coefficients. Therefore, we started by applying 3-level decomposition Discrete Wavelet Transform to each image, from our balanced dataset, using ‘haar’ wavelet. After that, we applied zero padding to coefficients of level 2 and 3 because we wanted equal size to all the coefficients (120,120). We plotted curve of the cumulative sum of variance, versus the number of principal components, which is shown in the figure below:

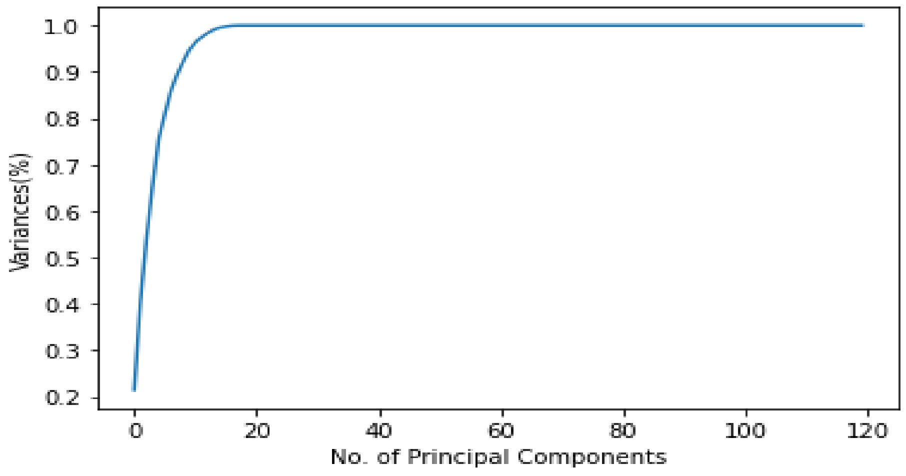
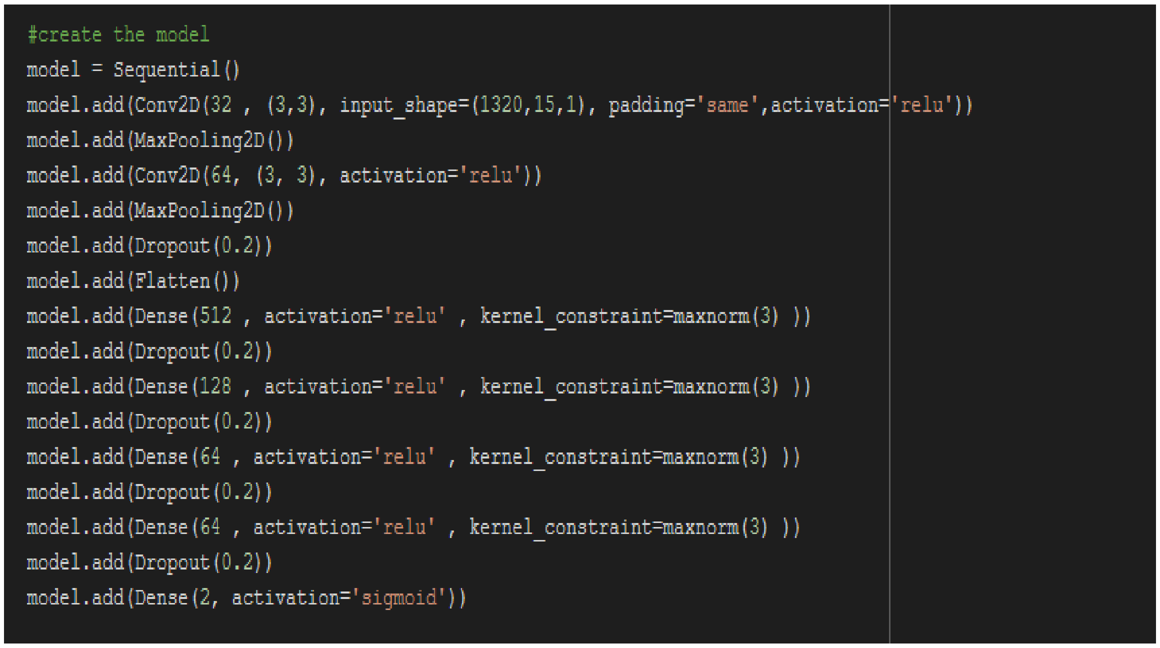


Figure 5.5: Variances against No. of principal components

We can see that the first 15 principal components can preserve more than 97% of total variance. Therefore, we reduced dimensions to 15 by applying Principal Components Analysis using PCA function, from sklearn library. Next we merged all the coefficients in one matrix which size was (1320,15). Then we standardized the matrix and we reshaped it to a three dimensions matrix. Our matrix had the form: (1320,15,1). Additionally, we split out our matrix (dataset), to train and test set, using 80 percent for training and 20 percent for testing. Then we built our Convolutional Neural Network, which is shown in the figure below:



```
#create the model
model = Sequential()
model.add(Conv2D(32 , (3,3), input_shape=(1320,15,1), padding='same',activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(128 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(64 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(64 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
```

Figure 5.6: CNN model for algorithm 5.2

We began with the creation of a Sequential model in order to stack all layers sequentially. We have two blocks, which have Conv2D layers, the ReLU activation function and a MaxPool2D of (2,2). The width and the height of the convolutional window are [3.3] and we change the size of Conv2D filters. In the first block, we have size of 32 and in the second block, we have 64. In the end, we have the classifier where we add Flatten Layers. Conv2D layers extract and learn spatial features, which then passed to four dense layers after they have been flattened. The last layer has output of two and sigmoid activation function, this is because we have two states, healthy and unhealthy, so we want the model to output a probability between 0 and 1. We have also added a few Dropout layers with dropout rate set to 20% to prevent overfitting.

We trained the model for 80 epochs, using ‘binary_crossentropy’ as loss function and ‘RMSprop’ with learning rate 0.0001, as optimizer. Accuracy on the test set was 0.81. Then we used our validation dataset, to test our model on unseen data. The results are below:

Table 17: Evaluation metrics on Validation Dataset for algorithm 5.2

	precision	**recall**
0	0.87	0.87
1	0.89	0.89

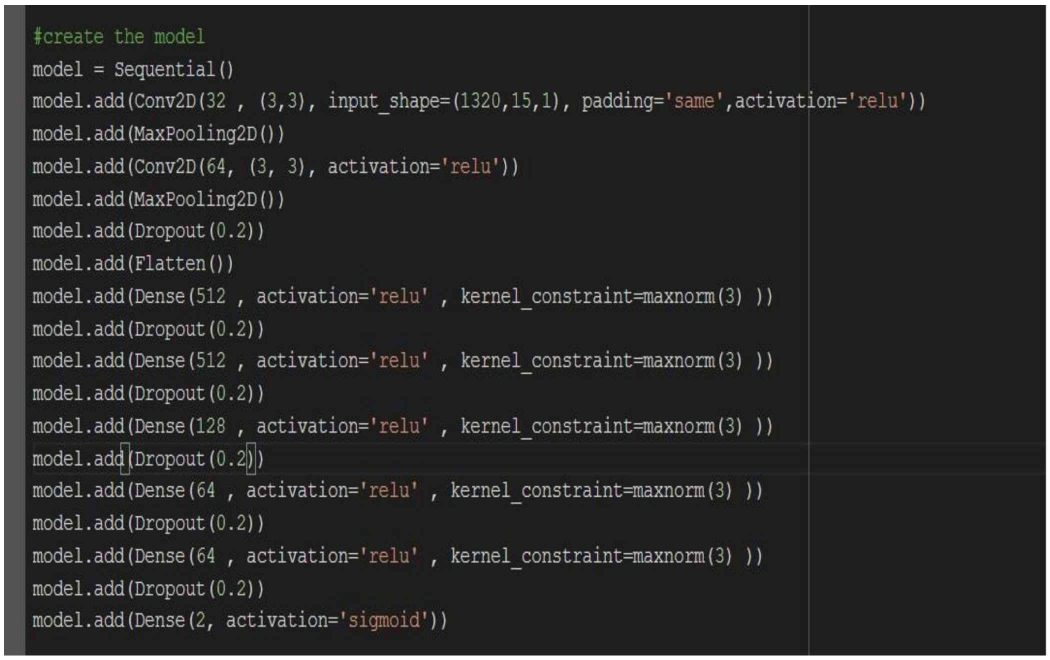
Table 18: Confusion Matrix of Validation Dataset for algorithm 5.2

	Negative	**Positive**
Negative	13	2
Positive	2	17

Total accuracy on Validation Dataset was 0.88

5.2.2.3 DWT coefficients of images using PCA and augmentation method: Algorithm 5.3

In algorithm 5.3, we used Principal Components Analysis in order to reduce the dimensions of Discrete Wavelet Transform coefficients as we did in algorithm 5.2 . The difference with algorithm 5.2 was that instead of using our balanced dataset, we used augmentation method and we created 200 healthy and 200 unhealthy images from our balanced dataset. So, we loaded the images from the augmented dataset and we applied 3-level decomposition Discrete Wavelet Transform to each image, using ‘haar’ wavelet. After that, we applied zero padding to coefficients of level 2 and 3 because we wanted equal size to all the coefficients (120,120). Then, we reduced dimensions to 15 by applying Principal Components Analysis using PCA function, from sklearn library as we did in algorithm 5.2 . Next, we merged all the coefficients in one matrix which size was (1200,15). Then we standardized the matrix and we reshaped it to a three dimensions matrix. Our matrix had the form: (1200,15,1). Additionally, we split out our matrix (dataset), to train and test set, using 80 percent for training and 20 percent for testing. Then we built our Convolutional Neural Network that is shown in the figure below:



```
#create the model
model = Sequential()
model.add(Conv2D(32 , (3,3), input_shape=(1320,15,1), padding='same',activation='relu'))
model.add(MaxPooling2D())
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(512 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(512 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(128 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(64 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(64 , activation='relu' , kernel_constraint=maxnorm(3) ))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
```

Figure 5.7: CNN model for algorithm 5.3

Again, we began with the creation of a Sequential model in order to stack all layers sequentially. We have two blocks, which have Conv2D layers, the ReLU activation function and a MaxPool2D of (2,2). The width and the height of the convolutional window are [3.3] and we change the size of Conv2D filters. In the first block, we have size of 32 and in the second block, we have 64. In the end, we have the classifier where we add Flatten Layers. Conv2D layers extract and learn spatial features, which then passed to five dense layers after they have been flattened. The last layer has output of two and sigmoid activation function, this is because we have two states, healthy and unhealthy, so we want the model to output a probability between 0 and 1. We have also added a few Dropout layers with dropout rate set to 20% to prevent overfitting.

We trained the model for 80 epochs, using ‘binary_crossentropy’ as loss function and ‘RMSprop’ with learning rate 0.0001, as optimizer. Accuracy on the test set was

0.975. Then we used our validation dataset, to test our model on unseen data. The results are below:

Table 19: Evaluation metrics on Validation Dataset for algorithm 5.3

	precision	**recall**
0	0.93	0.93
1	0.95	0.95

Table 20: Confusion Matrix of Validation Dataset for algorithm 5.3

	Negative	**Positive**
Negative	14	1
Positive	1	18

Total accuracy on Validation Dataset was 0.94

CHAPTER 6. ALGORITHMS EVALUATION AND FUTURE WORK

In this chapter, we will evaluate and compare the results of the algorithms we have developed, we will discuss the practices and techniques we used, and we will extract our conclusions. Finally, we suggest future work for our case.

6.1 Algorithms evaluation and results

This thesis aims to classify healthy from non-healthy patients based on MRI weighted T2 modality taken from the axial plane. For this purpose several scenarios and algorithms using machine and deep learning were developed. Below we present our results in detail.

In the tables below, we see and compare the results of our algorithms on the validation subset, because it contains unseen data for our algorithms.

Table 21: Accuracy, Recall and Specificity of algorithms

Algorithm	**Accuracy**	**Recall or Sensitivity**	**Specificity**
Algorithm 1.1	0.5	0.73	0.2
Algorithm 1.2	0.70	0.78	0.60
Algorithm 2.1	0.79	0.63	1.00

Algorithm 2.2	0.76	0.57	1.00
Algorithm 3.1	0.91	1.00	0.80
Algorithm 4.1	0.97	0.94	1.00
Algorithm 4.2	0.94	0.89	1.00
Algorithm 5.1	0.97	1.00	0.93
Algorithm 5.2	0.88	0.89	0.86
Algorithm 5.3	0.94	0.94	0.93

Table 22: FPR, FNR and Precision of algorithms

Algorithm	**FPR**	**FNR**	**Precision**
Algorithm 1.1	0.80	0.26	0.53
Algorithm 1.2	0.40	0.21	0.71
Algorithm 2.1	0.00	0.36	1.00
Algorithm 2.2	0.00	0.42	1.00
Algorithm 3.1	0.20	0.00	0.86
Algorithm 4.1	0.00	0.05	1.00

Algorithm 4.2	0.00	0.10	1.00
Algorithm 5.1	0.06	0.00	0.95
Algorithm 5.2	0.13	0.10	0.89
Algorithm 5.3	0.06	0.05	0.94

The first we are looking for is the Sensitivity (Recall) and Precision must be high and balanced. Then we care about FNR that must be low and then all the other metrics and of course we want them to be balanced.

We will start with machine learning algorithms and scenario 1, where we used the unbalanced dataset and wavelet entropy of images as input. As we can see, the results of algorithm 1.1 and algorithm 1.2 are not very good, but we observe a significant improvement in algorithm 1.2 where we divided our images into quarters. Then in scenario 2, which is the same with scenario 1 except we used the balanced dataset, we can see improvement in the results and the two algorithms seems to have almost the same results, but again they are not very good. In scenario 3, where we used the balanced dataset and the pixel values of images as input, we observe good results, Sensitivity and Precision are high and FNR is zero.

We continue to deep learning algorithms. We will start with scenario 4, where we used pixel values of images as input to our models. In algorithm 4.1, where we used the balanced dataset, we observe very good results. Sensitivity, Precision and all the other metrics are really high, FPR is zero and FNR is 0.05. In algorithm 4.2, where we used the augmented dataset, we observe a reduction in the metrics values in comparison with algorithm 4.1, but still is a very good model. In our last scenario 5, we have three algorithms where we used Discrete Wavelet Transform coefficients as input to our models. In algorithm 5.1, we used the balanced dataset, and we see that results are very good. Sensitivity, Precision and all the other metrics are really high and FNR is zero. In algorithm 5.2, where we applied Principal Components Analysis to reduce the dimensionality of the coefficients, and the balanced dataset, we see good results overall but they need improvement. That is why we developed algorithm 5.3, where we used augmentation to the balanced dataset and Principal Component Analysis to the coefficients. Results of algorithm 5.3 are very good. Sensitivity, Precision and the other metrics are high and FPR and FNR are very low. It was also the fastest deep learning algorithm in terms of training time.

The implemented algorithm 5.1 based on CNN, trained by balanced dataset, using the discrete wavelet transform coefficients of the whole row images, provided the highest scores: 100% Sensitivity, 97% Accuracy, 93% Specificity, 95% Precision, 0% FNR and 6% FPR and the algorithm 3.1, implemented based on SVM, trained by balanced dataset, using the pixel values of row whole images as features provided the second highest scores: 100% Sensitivity, 91% Accuracy, 80% Specificity, 86% Precision, 0% FNR and 20% FPR.

6.2 Future Work

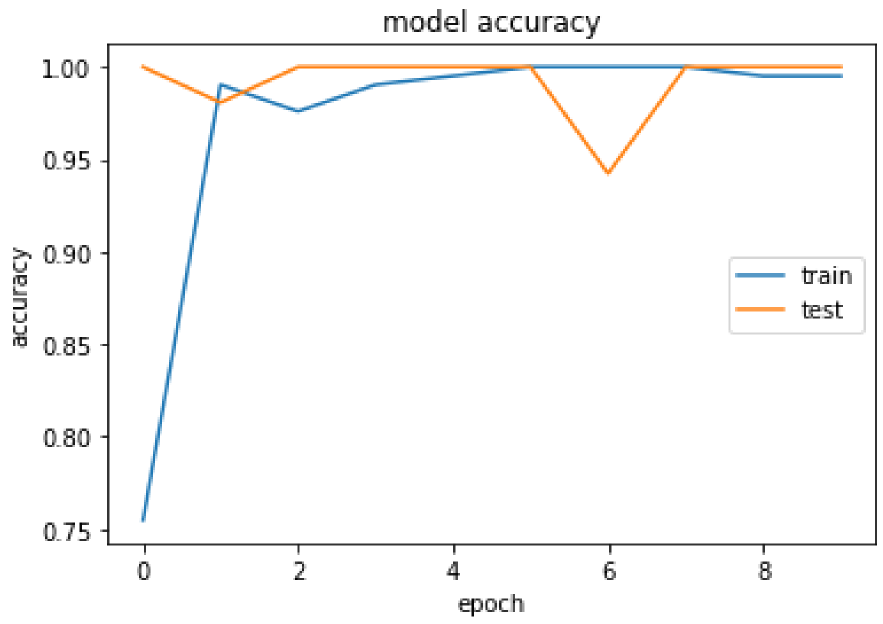
The purpose of this thesis was to classify human brain MRI images between healthy and tumorous. For this purpose, we used several machine and deep learning algorithms and techniques. Although we managed to get good results for our classification problem there is still plenty of room for improvement.

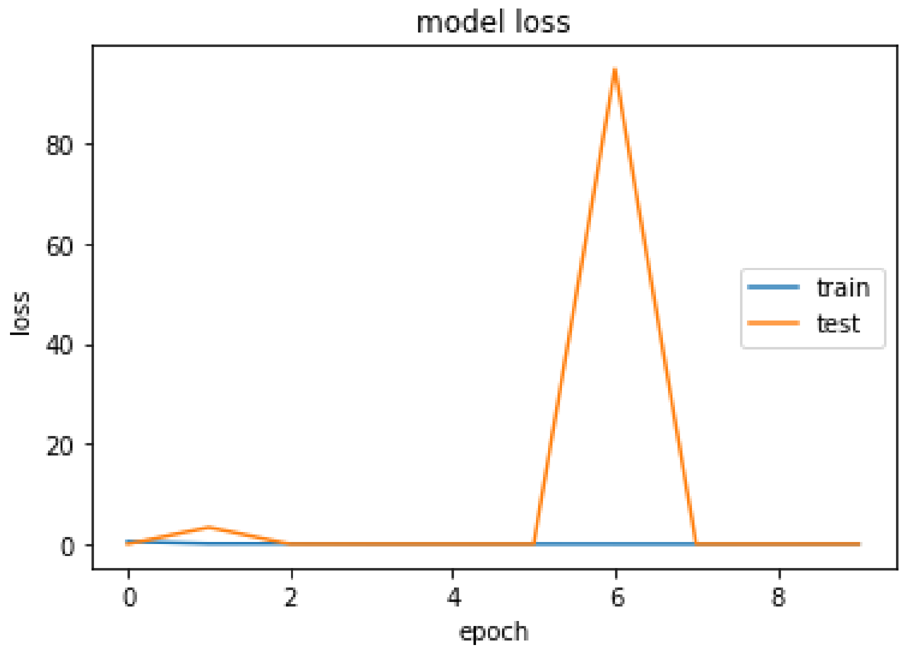
First of all, gathering more images for training would improve the results of our algorithms overall. Furthermore, we could add more features in our algorithms, such as patients' pathology data, energy of the image and so on. Additionally, the proposed algorithms could be developed to classify benign from malignant tumors or recognizing and classifying the grade of malignant tumors. Finally, the proposed algorithm could be implemented in a dataset containing other modalities also, like T1 weighted or FLAIR.

APPENDIX A. ACCURACY AND LOSS

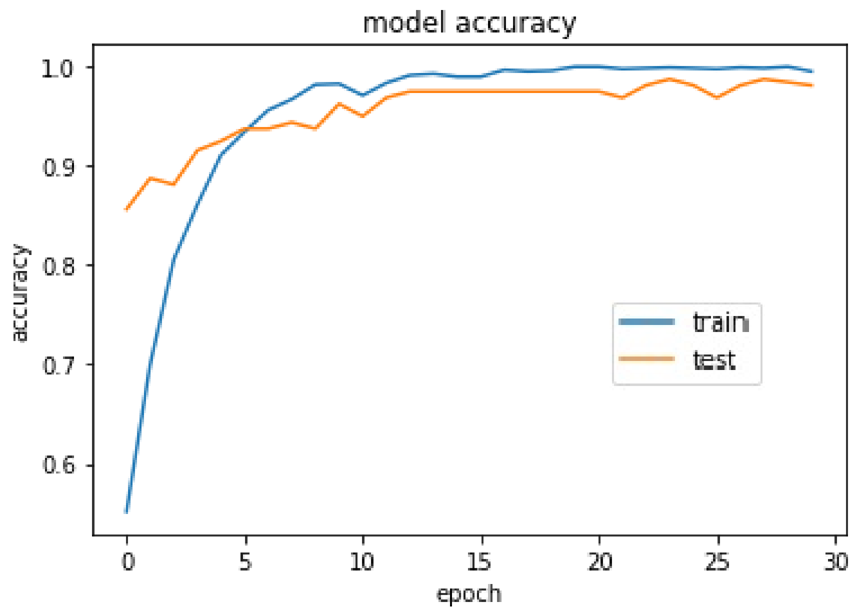
Here we can see the graphs of deep learning models during training. In graphs, we represent the Validation Accuracy and Loss from the test subset.

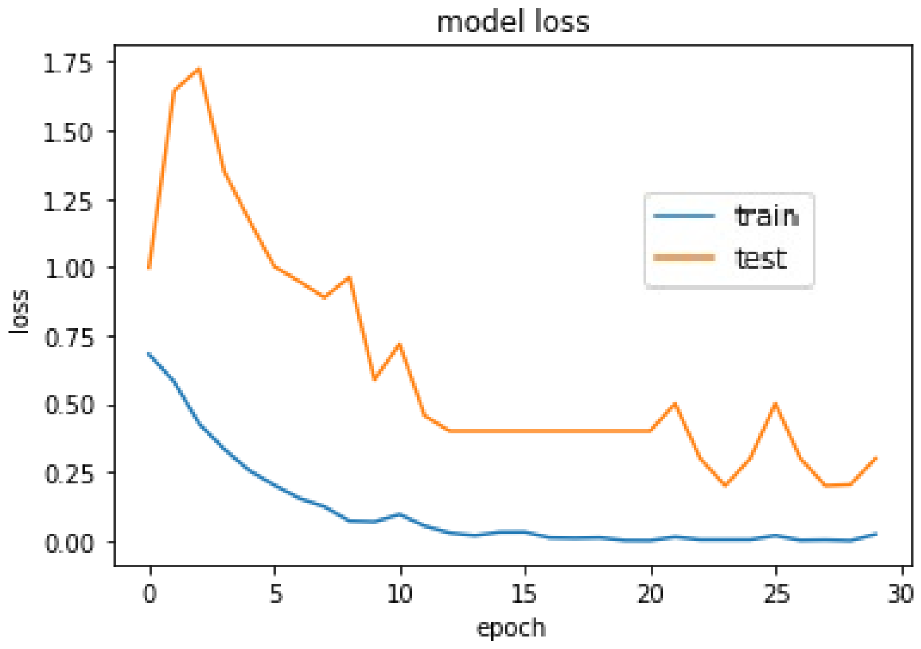
Algorithm 4.1



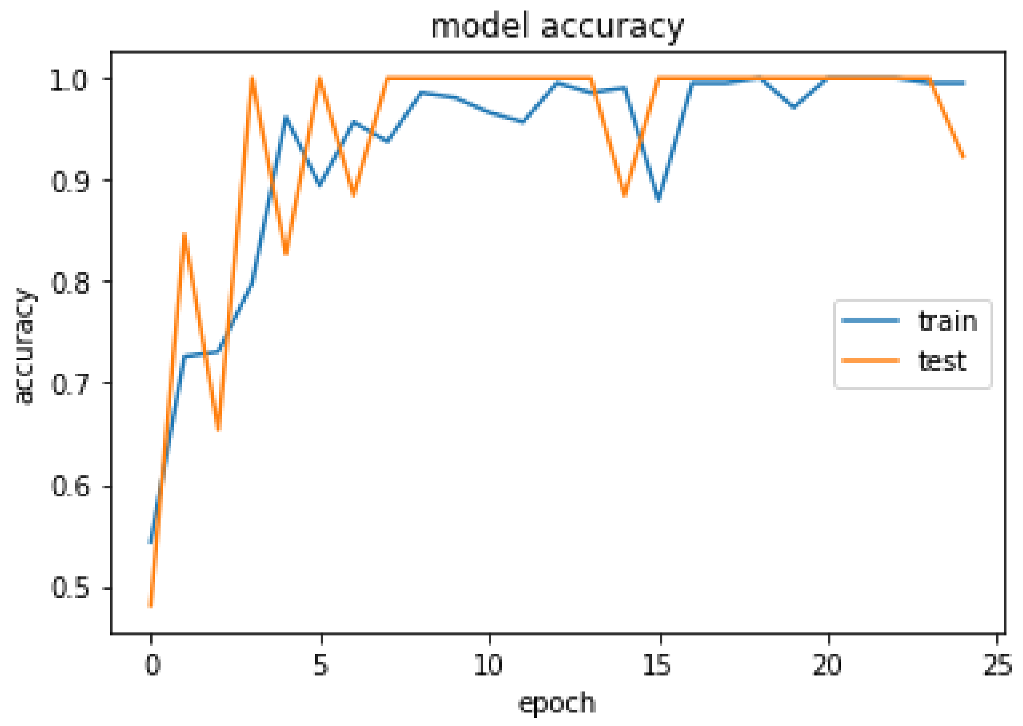


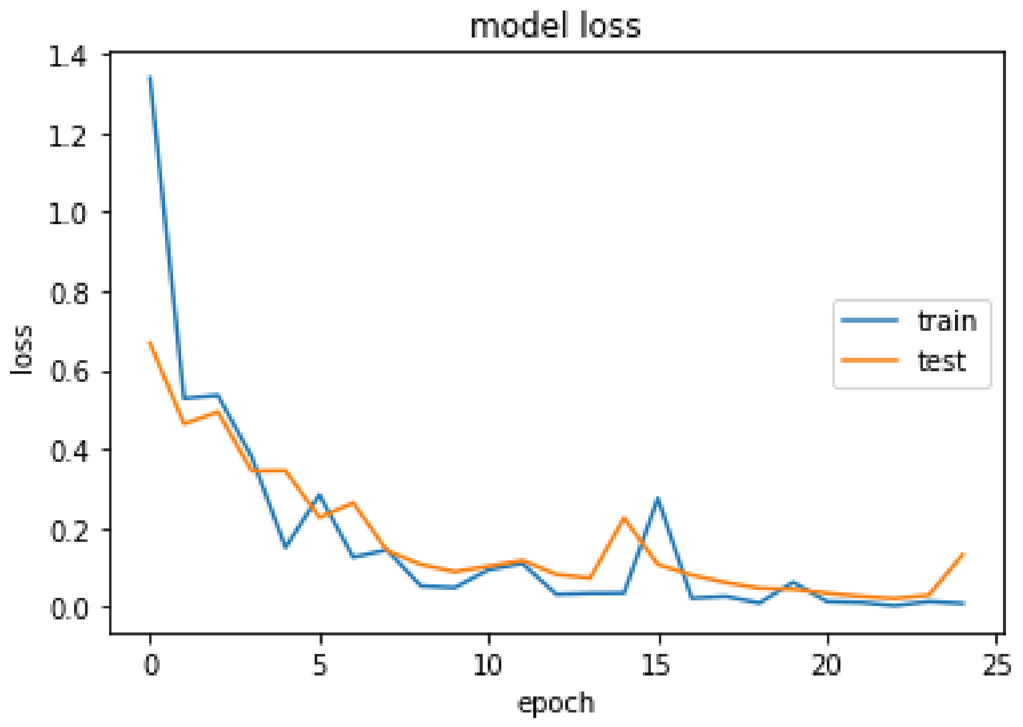
Algorithm 4.2



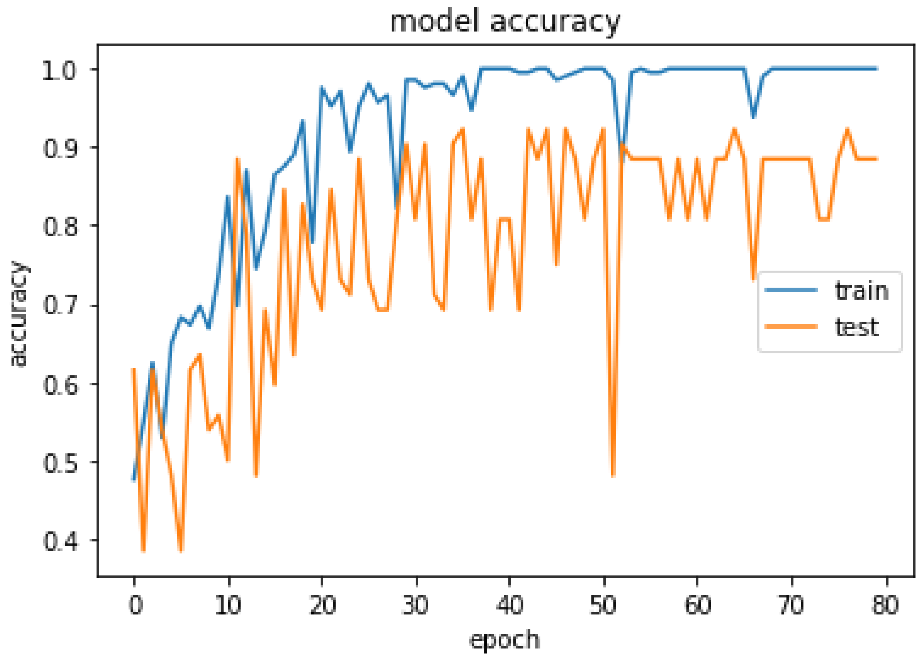


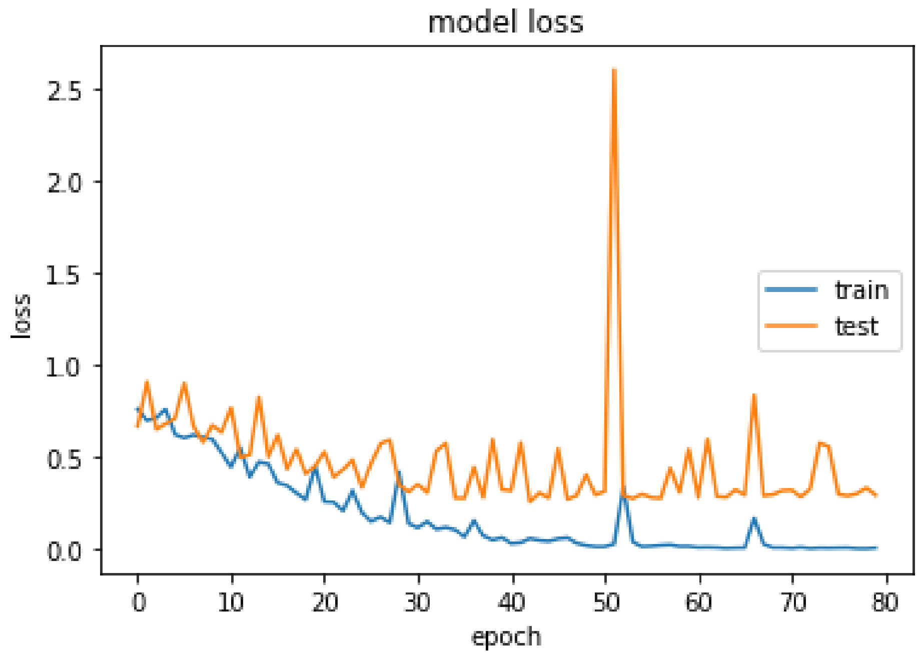
Algorithm 5.1



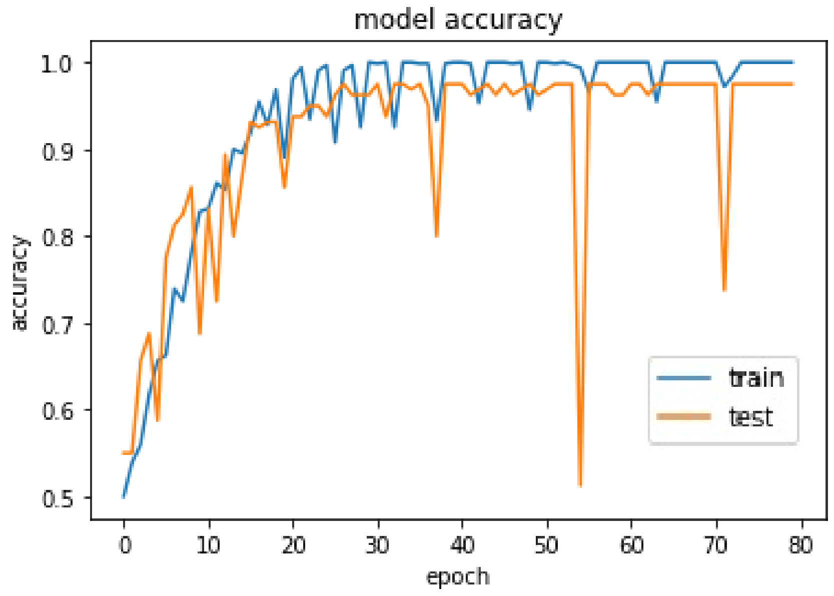


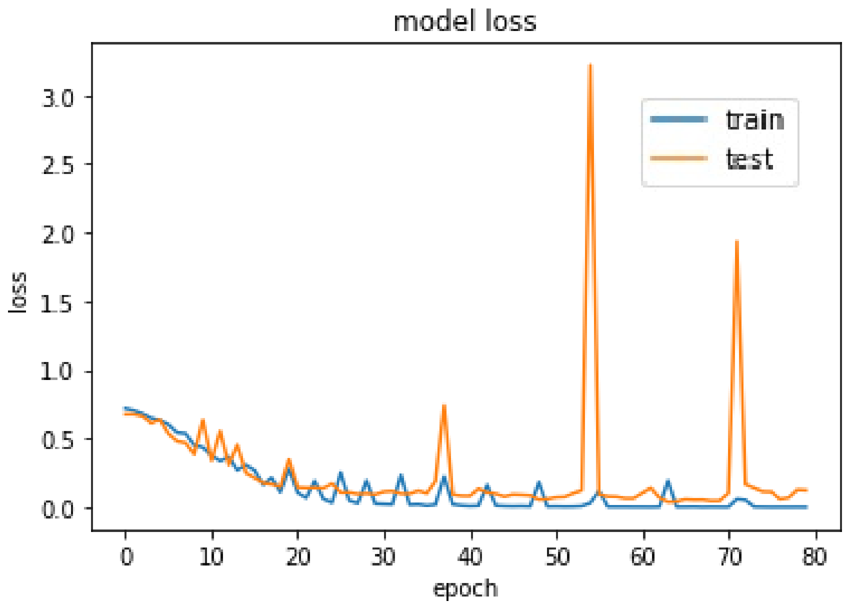
Algorithm 5.2





Algorithm 5.3





REFERENCES

1. MAYFIELD brain and Spine. *Anatomy of the Brain.* [Online] 09 2018. https://mayfieldclinic.com/pe-anatbrain.htm.

2. **contributors, Wikipedia.** Wikipedia. *Machine Learning.* [Online] https://en.wikipedia.org/wiki/Machine_learning#Overview.

3. Brain Tumors. *American Association of Neurological Surgeons.* [Online] https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Brain-Tumors.

4. National Brain Tumor Sosciety. *UNDERSTANDING BRAIN TUMORS.* [Online] https://braintumor.org/brain-tumor-information/understanding-brain-tumors/.

5. U.S Department of Health & Human Services/National Institutes of Health. *Magnetic Resonance Imaging (MRI).* [Online] https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri.

6. **David C Preston, MD.** SCHOOL OF MEDICINE CASE WESTERN RESERVE UNIVERSITY. *Magnetic Resonance Imaging (MRI) of the Brain and Spine: Basics.* [Online] 11 30, 2006. https://casemed.case.edu/clerkships/neurology/Web%20Neurorad/MRI%20Basics.htm.

7. **Courville, Ian Goodfellow and Yoshua Bengio and Aaron.** *Deep Learning.* s.l. : MIT Press, 2016. http://www.deeplearningbook.org.

8. **Brownlee, Jason.** *Machine Learning Mastery with Python.* 2019.

9. **Wierenga, Rick.** Heartbeat. *An in-depth look into optimizers used for machine learning* . [Online] 12 4, 2019. https://heartbeat.fritz.ai/an-empirical-comparison-of-optimizers-for-machine-learning-models-b86f29957050.

10. **Brownlee, Jason.** Machine Learning Mastery. *Loss and Loss Functions for Training Deep Learning Neural Networks.* [Online] 10 23, 2019. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

11. *The Effectiveness of Data Augmentation in Image Classification using DeepLearning.* **Standford, Jason Wang & Luis Perez.** 2017, Vol. I.

12. *Image Watermarking Using 3-Level Discrete Wavelet Transform (DWT).* **Nikita Kashyap Department of Electronics & Telecommunication Engineering, Shankaracharya Technical Campus, Bhilai, India & G. R. SINHA Associate Director, Faculty of Engg & Tech, Shri Shankaracharya Technical Campus, Bhilai, India.** 2012.

13. **M.S. Landis, J.P. Pancras,J.R. Graney,R.K. Stevens,K.E. Percy,S. Krupa.** Chapter 18 Receptor Modeling of Epiphytic Lichens to Elucidate the Sources and Spatial Distribution of Inorganic Air Pollution in the Athabasca Oil Sands Region. *Developments in Environmental Science.* s.l. : Elsevier.

14. **Zhou, Taiyong Li & Min.** ECG Classification Using Wavelet Packet Entropy and Random Forests. *Entropy.* 2016.

15. **Giorgos A. Giannakakis, Nikolaos N. Tsiaparas, Monika-Filitsa S. Xenikou,Charalabos Papageorgiou, Konstantina S. Nikita, Senior Member, IEEE.** Wavelet Entropy Differentiations of Event Related Potentials in Dyslexia. 8th IEEE International Conference on BioInformatics and BioEngineering, BIBE 2008. 1-6. 10.1109/BIBE.2008.4696836.

16. **R. Fisher, S. Perkins, A. Walker and E. Wolfart.** Pixel Values. [Online] 2003. https://homepages.inf.ed.ac.uk/rbf/HIPR2/value.htm.

17. **Singh, Bhajandeep.** Heartbeat. *Evaluation Metrics for Machine Learning Models.* [Online] 10 25, 2019. https://heartbeat.fritz.ai/evaluation-metrics-for-machine-learning-models-d42138496366.

18. **B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, et al. "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)", IEEE Transactions on Medical Imaging 34(10), 1993-2024 (2015) DOI: 10.1109/TMI.2014.2377694.**

19. **S. Bakas, H. Akbari, A. Sotiras, M. Bilello, M. Rozycki, J.S. Kirby, et al., "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features", Nature Scientific Data, 4:170117 (2017) DOI: 10.1038/sdata.2017.**

20. **S. Bakas, M. Reyes, A. Jakab, S. Bauer, M. Rempfler, A. Crimi, et al., "Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation,**

Progression Assessment, and Overall Survival Prediction in the BRATS Challenge", arXiv preprint arXiv:1.