



ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ & ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

Διπλωματική Εργασία

«Αλγόριθμοι τεχνητής νοημοσύνης στην ψηφιακή επεξεργασία βιοϊατρικών εικόνων»



Φοιτητής: Αλεξόπουλος Νικόλαος

ΑΜ: 50106503

Επιβλέπουσα Καθηγήτρια: Μαρία Ραγκούση

ΑΘΗΝΑ-ΑΙΓΑΛΕΩ, ΦΕΒΡΟΥΑΡΙΟΣ 2023



UNIVERSITY OF WEST ATTICA

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

Diploma Thesis

“Artificial Intelligence algorithms in the digital processing of biomedical images”



Student: Alexopoulos Nikolaos

Registration Number: 50106503

Supervisor: Prof. Maria Rangoussi

ATHENS-EGALEO, February 2023

Η Διπλωματική Εργασία έγινε αποδεκτή και βαθμολογήθηκε από την εξής τριμελή επιτροπή:

Ραγκούση Μαρία, καθηγήτρια (επιβλέπουσα)	Ποτηράκης Στυλιανός, καθηγητής	Μετάφας Δημήτριος, επ. καθηγητής
(Υπογραφή)	(Υπογραφή)	(Υπογραφή)

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τους συγγραφείς.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον/την συγγραφέα του και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις θέσεις του επιβλέποντος, της επιτροπής εξέτασης ή τις επίσημες θέσεις του Τμήματος και του Ιδρύματος.

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Αλεξόπουλος Νικόλαος του Βασιλείου με αριθμό μητρώου 50106503 φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής ΜΗΧΑΝΙΚΩΝ του Τμήματος ΗΛΕΚΤΡΟΛΟΓΩΝ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ,

δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του διπλώματός μου.»

Ο Δηλών



Αλεξόπουλος Νικόλαος

I would like to thank my supervising professor Maria Rangoussi for the guidance and insight she gave me during the preparation of this thesis.

Περίληψη

Η τεχνητή νοημοσύνη έχει καινοτομήσει στον τρόπο που επεξεργαζόμαστε και αναλύουμε δεδομένα και ψηφιακό περιεχόμενο εν γένει, συμπεριλαμβανομένων και των βιοϊατρικών εικόνων. Στην παρούσα διπλωματική εργασία εξετάζεται η χρήση συνελκτικών νευρωνικών δικτύων (Convolutional Neural Networks, CNN) για την κατηγοριοποίηση μαγνητικών τομογραφιών εγκεφάλου, μιας σημαντικής διαδικασίας στο χώρο της ιατρικής διάγνωσης. Η έρευνα είναι πειραματική και εκτιμά συγκριτικά την απόδοση δύο σύγχρονων αρχιτεκτονικών CNN, του EfficientNetB0 και του ResNet18, σε ένα σύνολο δεδομένων από μαγνητικές τομογραφίες εγκεφάλου. Τα αποτελέσματα δείχνουν ότι η αρχιτεκτονική του δικτύου EfficientNetB0 υπερτερεί της αρχιτεκτονικής ResNet18 ως προς την αποτελεσματικότητα και συγκεκριμένα ως προς το ποσοστό ακρίβειας στην κατηγοριοποίηση των εικόνων. Τα αποτελέσματα αυτά αναδεικνύουν τη σημασία των τεχνικών βαθιών νευρωνικών δικτύων στην αυτοματοποιημένη κατηγοριοποίηση βιοϊατρικών εικόνων.

Τα ευρήματα αυτής της μελέτης παρέχουν χρήσιμα στοιχεία που αφορούν τους περιορισμούς και τις δυνατότητες των αλγορίθμων τεχνητής νοημοσύνης στην ψηφιακή επεξεργασία βιοϊατρικών εικόνων. Τα αποτελέσματα δείχνουν τη δυνατότητα των τεχνικών βαθιάς μάθησης να βελτιώσουν την ακρίβεια και την αποτελεσματικότητα της αυτοματοποιημένης κατηγοριοποίησης βιοϊατρικών εικόνων, η οποία έχει σημαντικό ρόλο στην έγκαιρη ανίχνευση και διάγνωση εγκεφαλικών βλαβών. Επιπλέον, η μελέτη δείχνει και τη σημασία της επιλογής κατάλληλων αρχιτεκτονικών CNN για συγκεκριμένους στόχους καθώς και την ανάγκη για μεγάλα και διαφοροποιημένα σύνολα δεδομένων για τη βελτίωση της γενίκευσης των αποτελεσμάτων.

Λέξεις – κλειδιά

Βιοϊατρικές Εικόνες, Ψηφιακή Επεξεργασία Εικόνας, Μηχανική Μάθηση, Νευρωνικά Δίκτυα, Βαθιά Νευρωνικά Δίκτυα, Συνελκτικά Νευρωνικά Δίκτυα, Κατηγοριοποίηση Εικόνων

Abstract

Artificial intelligence has revolutionized the way we process and analyze data, including biomedical images. The present thesis investigates the application of convolutional neural networks (CNNs) for the classification of MRI brain scans, a crucial task in the medical diagnosis profession from brain images. The thesis is experimental; it comparatively evaluates the performance of two state-of-the-art CNN architectures, EfficientNetB0 and ResNet18, on a dataset of MRI brain scans. The results show that the EfficientNetB0 architecture outperforms the ResNet18 architecture in terms of accuracy and efficiency, demonstrating the potential of machine learning techniques for biomedical classification of images.

The findings of this study provide useful insights into the capabilities and limitations of artificial intelligence algorithms for the digital processing of biomedical images. The results demonstrate the potential of machine learning methods to enhance the precision and efficiency of automated biomedical image classification, which has important implications for the early detection and diagnosis of brain damages. Additionally, the study highlights the importance of selecting the appropriate CNN architectures for the given task, as well as the need for large and diversified datasets to improve the generalizability of the results.

Keywords

Biomedical images, Convolutional Neural Networks, Digital Image Processing, Neural Networks, Machine Learning, Deep Learning, Image Classification

ΠΕΡΙΛΗΨΗ	6
ABSTRACT	7
LIST OF FIGURES	10
GLOSSARY	12
INTRODUCTION.....	13
DIPLOMA THESIS SUBJECT	13
PURPOSE AND GOALS	13
METHODOLOGY.....	13
INNOVATION	13
STRUCTURE.....	14
CHAPTER 1: BIOMEDICAL IMAGES	15
1.1 INTRODUCTION.....	15
1.2 IMPORTANCE OF ANALYSIS OF BIOMEDICAL IMAGES	15
1.3 BIOMEDICAL IMAGE PROCESSING	16
1.4 SYSTEM REQUIREMENTS FOR BIOMEDICAL IMAGE PROCESSING	17
1.5 CLASSIFICATION IN BIOMEDICAL IMAGES.....	17
1.6 MRI	17
1.6.1 How does MRI function	18
1.6.2 Examples of MRI use in patients with cancer.....	18
1.6.3 Pros and Cons of MRI	18
1.6.4 MRI and Brain Tumors.....	19
CHAPTER 2: MACHINE LEARNING.....	21
2.1 PROBABILITY AND INFORMATION THEORY	22
2.1.1 Why is probability important	22
2.2 DATA AND FEATURES	23
2.3 LINEAR REGRESSION	25
2.4 LOGISTIC REGRESSION	26
2.5 SUPERVISED LEARNING	26
2.5.1 Introduction.....	26
2.5.2 The House Pricing Problem.....	27
2.6 UNSUPERVISED LEARNING.....	27
2.7 SEMI-SUPERVISED LEARNING.....	28
2.8 REINFORCEMENT LEARNING	28
2.9 SUPPORT VECTOR MACHINES.....	29
2.10 THE CONCEPT OF OVERFITTING.....	30
2.11 ACTIVATION FUNCTIONS.....	31
2.11.1 Commonly used Activation Functions	31
2.11.2 How to choose the activation function.....	33
2.12 REGULARIZATION	33
2.12.1 Early Stopping	33
2.12.2 Ensemble Methods.....	33
2.13 BACKPROPAGATION.....	34
2.14 NEURAL NETWORKS	34
2.15 ARCHITECTURES OF NEURAL NETWORKS.....	35
2.15.1 The Perceptron	35
2.15.2 Feed Forward Neural Networks.....	37
2.16 CONVOLUTIONAL NEURAL NETWORKS	38

Artificial Intelligence algorithms in the digital processing of biomedical images

2.16.1 Introduction.....	38
2.16.2 An overview of how CNNs work	38
2.17 OVERALL ARCHITECTURE	39
2.18 CONVOLUTIONAL LAYER.....	41
2.19 POOLING LAYER	43
CHAPTER 3: EXPERIMENTAL PART: MULTI-CLASS CLASSIFICATION OF BRAIN MRI TUMORS USING EFFICIENTNETB0 AND RESNET18	44
3.1 INTRODUCTION	44
3.2 TUMORS REGARDING CURRENT THESIS.....	44
3.2.1 Introduction.....	44
3.2.2 Meningioma	46
3.2.3 Glioma	46
3.2.4 Pituitary Gland	47
3.3 EXPERIMENT #1: MULTI-CLASS CLASSIFICATION OF BRAIN MRIS USING EFFICIENTNETB0	47
3.3.1 Dataset.....	47
3.3.2 Dataset Preprocessing.....	49
3.3.3 Data augmentation on the Dataset	51
3.3.4 Helper Functions.....	53
3.3.5 Setting up the Datasets for Training and Validation	53
3.4 HOW DOES EFFICIENTNETB0 WORK	55
3.4.1 How to implement EfficientNetB0 Transfer Learning.....	57
3.5 BUILDING THE EFFICIENTNETB0 MODEL.....	58
3.6 TRAINING THE EFFICIENTNETB0 MODEL.....	58
3.7 INFERENCE ON THE EFFICIENTNETB0 MODEL	60
3.8 EXPERIMENT #2: MULTI-CLASS CLASSIFICATION OF BRAIN MRIS USING RESNET18	62
3.8.1 How ResNet18 Works.....	62
3.8.2 How to implement Transfer Learning with ResNet18	63
3.9 BUILDING THE RESNET18 MODEL	64
3.10 TRAINING THE RESNET18 MODEL.....	64
3.11 INFERENCE ON THE RESNET18 MODEL.....	65
3.12 DISCUSSION OVER THE RESULTS IN ACCURACY.....	66
4. CONCLUSIONS	68
BIBΛΙΟΓΡΑΦΙΑ – ΑΝΑΦΟΡΕΣ - ΔΙΑΔΙΚΤΥΑΚΕΣ ΠΗΓΕΣ	69
BIOMEDICAL IMAGING SOURCES	69
TUMORS CITATIONS	69
GRADIENT DESCENT	70
LINEAR REGRESSION	70
LOGISTIC REGRESSION.....	70
MACHINE LEARNING.....	71
NEURAL NETWORKS	71
CNNs.....	71
DEEP LEARNING.....	72
URL SOURCES.....	72
APPENDIX 1: CODING FOR EFFICIENTNETB0.....	74
APPENDIX 2: CODING PART FOR RESNET18.....	88

List of Figures

Figure 1.1: MRI scan showing a tumor in the brain

Figure 2.1: Overview and categories of machine learning algorithms. (a) Classes of various approaches are shown. (b) Overview of ML methods

Figure 2.2: Mean Squared Error equation

Figure 2.3: A linear regression problem with a single parameter to learn, w_1 , aiming to minimize the MSE on the training set using the normal equations. The plot shows the learned w_1 value and how linear regression tries to fit a line to the training points

Figure 2.4: Differences between an Underfit, Overfit and a good realistic fit

Figure 2.5: The Sigmoid Function

Figure 2.6: The Hyperbolic tangent (Tanh) function

Figure 2.7: Rectified Linear Unit (ReLU) function

Figure 2.8: Comparison between conventional ML and Deep Neural Networks

Figure 2.9: A representation of the Perceptron

Figure 2.10: An Example of a Neural Network

Figure 2.11: A visual representation of how a Convolutional Neural Network works

Figure 2.12: Illustration of the difference between a general convolution kernel and a dilated convolution kernel

Figure 2.13: : A 5-Layered CNN model

Figure 2.14: An inside look after the first convolution of a deep neural network on a dataset of digits

Figure 2.15: The convolution layer process and Feature Map

Figure 3.1: A closed MRI scanning system

Figure 3.2: A sample for each class from our Training Set

Figure 3.3: The same sample as seen in Fig 3.2, now cropped

Figure 3.4: Example of data augmentation performed in Brain MRI Scans

Figure 3.5: A portion of our code regarding the Data Augmentation transforms we used

Figure 3.3: Example2 of data augmentation in a brain MRI scan

Figure 3.4: The architecture of EfficientNetb0

Figure 3.5: Model Size vs. Accuracy

Figure 3.6: Example2 of data augmentation in a brain MRI scan

Artificial Intelligence algorithms in the digital processing of biomedical images

Figure 3.7: A portion of our code showing the shuffling procedure

Figure 3.8: The architecture of EfficientNetB0 [50]

Figure 3.9: Model Size vs. Accuracy [50]

Figure 3.10: Visual representation of Transfer Learning [51]

Figure 3.11: Message on our Python interface showing the Training and Validation Accuracy after the Training is complete

Figure 3.12: Plots showing the Training and Validation Accuracy of the EfficientNetB0

Figure 3.13: Plots showing the Training and Validation Loss of the EfficientNetB0

Figure 3.14: Confusion Matrix for the EfficientNetb0

Figure 3.15: Message on our Python interface showing the overall Accuracy of the EfficientNetB0 on unseen data

Figure 3.16 Original ResNet18 Architecture

Figure 3.17: Message on our Python interface showing the Training and Validation Accuracy after the Training is complete

Figure 3.18: Plots showing the Training and Validation Accuracy of the ResNet18

Figure 3.19: Plots showing the Training and Validation Loss of the ResNet18

Figure 3.20: Confusion Matrix for the ResNet18

Figure 3.21: Message on our Python interface showing the overall Accuracy of the ResNet18 on unseen data

Glossary

AI:	Artificial Intelligence
CNN:	Convolutional Neural Networks
ML:	Machine Learning
NN:	Neural Network
ANN:	Artificial Neural Network
CDC:	Centers for Disease Control and Prevention
CT:	Computed Tomography
MRI:	Magnetic Resonance Imaging
DNN:	Deep Neural Network
SNN:	Shallow Neural Network

Introduction

Artificial intelligence (AI) has recently gained spectacular popularity. It is changing the way we approach and solve many problems, including biomedical image analysis. AI algorithms are being developed and applied to process large datasets of biomedical images, like MRI scans, CT scans, and X-rays, to enhance the accuracy of diagnoses and treatment plans. In this thesis, we examine the use of Convolutional Neural Networks (CNNs), a popular AI architecture, to classify MRI brain scans into one of four categories: normal (healthy), meningioma, glioma, and pituitary gland tumor. We use transfer learning to leverage pre-trained models EfficientNetB0 and ResNet18, and to enhance the generalization of the model by using data augmentation techniques [42].

Diploma Thesis Subject

The application of AI in biomedical image analysis is an area that attracts strong research interest as it has the potential to transform the way we diagnose and treat diseases. Biomedical images can be complex and challenging to interpret. AI algorithms can help to identify patterns and abnormalities that may be difficult for the human eye to detect. In the case of MRI brain scans, AI models can help to identify and classify brain tumors, which can aid in the prognosis, treatment, and monitoring of patients with brain tumors.

Purpose and Goals

This thesis has the purpose of probing the application of CNNs for the categorization of MRI brain scans into one of four categories: normal (healthy), meningioma, glioma, and pituitary gland tumor. The primary goal is to develop an accurate and reliable classification model that can aid in the diagnosis of brain tumors. We aim to achieve this by leveraging the power of techniques such as transfer learning and data augmentation, to improve the performance and generalization of our model.

Methodology

We use transfer learning to leverage pre-trained models EfficientNetB0 and ResNet18, which have been trained on large-scale image datasets, to classify MRI brain scans. We fine-tune the pre-trained models on our MRI brain scan dataset and use data augmentation techniques, such as random rotation, flipping, and scaling, to improve the generalization of our neural network. We divide our dataset into two categories: training and validation sets, and train the model using stochastic gradient descent with momentum. We assess how well our model performs using a brand-new test set (i.e., a test set disjoint to the training set employed) to calculate its accuracy on unknown data (images).

Innovation

One of the innovative aspects of this thesis is the use of pre-trained models, EfficientNetB0 and ResNet18, for classifying MRI brain scans. Transfer learning allows us to leverage the

power of these pre-trained models and achieve high accuracy with a relatively small dataset. Transfer learning, in general, is a technique that has sparked a lot of discussion lately, because of its capability to improve the performance of deep learning models, while reducing the need for extensive training on large datasets.

By utilizing these pre-trained models, we can leverage the knowledge that they have gained from millions of images to better classify the medical images in our dataset. Furthermore, we are utilizing augmentation techniques to improve the generalization of our model, a procedure that can help in avoiding overfitting and therefore enhancing the model's ability to classify new and unseen data. By performing data augmentation, we can create additional training examples that are identical to our original data, but with minute variations. That way, we are going to help our model learn to recognize and classify different variations of the same image. It must be noted, that innovation doesn't necessarily mean something completely new and never done before. Rather, it could be a new way of doing something that has already been done before, or building upon existing ideas and techniques to create something new and unique.

In the case of this thesis, the innovation may lie in the particular way we approached the problem of classifying biomedical images using CNNs, the unique datasets we used, the specific CNN architectures we chose, the data augmentation techniques we applied, or any combination of these factors.

Structure

In this thesis, we are going to follow a structured approach in the pursuit of manufacturing and evaluating deep neural network models for image classification. Firstly, we will create a function to save the trained models, which ensured that we could easily access and reuse them for future experiments. Then, we will preprocess our dataset and increase its variability by applying data augmentation techniques to on the training data. Next, we will build two different models, EfficientNetB0 and ResNet18, and train them using the preprocessed dataset. Once the training is done, we will showcase the accuracy and loss for the training and validation each model to monitor their performance. Lastly, we analyze the adequacy of the models by calculating the confusion matrix and the overall accuracy on the new unseen data and how accurate it predicts each class of the brain MRI's. This structured approach allows us to effectively evaluate the performance of the models and make informed decisions for further experimentation and model improvements.

Chapter 1: Biomedical Images

1.1 Introduction

Biomedical images play a critical role in modern medicine, providing valuable information about the anatomy and function of the human body. The use of images in medicine dates back to ancient times, when physicians used simple drawings and illustrations to diagnose and treat their patients. With the advent of new technologies, for instance X-rays, CT scans, and MRI, the field of biomedical imaging has experienced tremendous growth and advancement over the past century.

Biomedical image analysis is a crucial aspect of modern medicine, providing doctors with the tools and techniques to diagnose and treat various medical conditions. The analysis of biomedical images helps to detect the presence of disease, monitor the progression of conditions, and evaluate the effectiveness of treatments. As large volumes of data are becoming more widely available, biomedical image analysis has become a rapidly growing field, with new algorithms and technologies being developed to improve the accuracy and efficiency of the analysis process.

Lately, artificial intelligence (AI) has become a potent weapon for biomedical image analysis, offering the potential to revolutionize the way we process and analyze biomedical images. The implementations of machine learning algorithms has shown promising potential in terms of enhancing the accuracy and efficiency of biomedical image analysis, making it possible to identify patterns and features in images that were previously undetectable. In this thesis, we investigate the use of AI to the digital processing of medical pictures, focusing on the classification of MRI brain scans using CNNs.

1.2 Importance of analysis of Biomedical Images

Biomedical image analysis is a crucial aspect of modern medicine, providing doctors with the tools and techniques to diagnose and treat various medical conditions. The analysis of biomedical images helps to identify the presence of disease, monitor the progression of conditions, and evaluate the effectiveness of treatments. As large amounts of data become progressively available, biomedical image analysis has become a rapidly growing field, with new algorithms and technologies being developed to improve the accuracy and efficiency of the analysis process.

As of lately, the usage of AI technology has emerged as a powerful tool for biomedical image analysis, offering the potential to revolutionize the way we process and analyze biomedical images. The use of deep learning algorithms and convolutional neural networks (CNNs) has shown great promise in improving the accuracy and efficiency of biomedical image analysis, making it possible to identify patterns and features in images that were previously undetectable. In this thesis, we look at how artificial intelligence is used in the digital processing of biomedical images, focusing on the classification of MRI brain scans using CNNs [2].

1.3 Biomedical Image Processing

Since its inception, the processing and analysis of biological images using computers has undergone revolutionary changes and experienced rapid expansion since its early 1970s exploratory phase [8]. In the past twenty years, the rapidly advancing field of computer technology has seen a quick transition from less effective and powerful computers to more efficient and compact devices [7]. It now appears possible to solve the issue of automatically analyzing and interpreting photographs for the categorization of their contents thanks to modern computing technology, including special-purpose language processors, intelligent systems, and high processing computers. Additionally, advances in sensor technology and electrical instrumentation have produced powerful and trustworthy picture capture systems. In the healthcare profession, the analysis and processing of computerized images have had the most revolutionary effects. Imaging techniques and modalities, i.e MRIs and CT scans, Hematological cell analysis on a computer, etc., that were just in the experimental research phase two decades ago, are now widely used in clinical settings [8].

Image quality, gray-level mapping, spectral analysis, area extraction, and other traditional and often straightforward image processing methods have been successfully applied to the processing and interpretation of biological images. Recent advances in algorithms for image processing, analysis, and understanding have proved to be very promising and successful in enhancing critical diagnostic information in radiology. These algorithms include image augmentation, gray-level mapping, and image reconstruction from projections, which are important techniques used in MRI and other radiological imaging modalities. Moreover, these techniques have also been successfully demonstrated in biological image processing applications such as characterizing and classifying items of interest, including the left ventricle, blood cells, chromosomes, and others. Additionally, there has been significant progress in exploring techniques for three-dimensional image analysis and color spectrum analysis for morphological and histological examinations of images of tissues or diseases, which have shown great potential. [3].

With the advancement of AI methods and intelligent systems, image analysis may now combine diagnostic and characterization knowledge. There is significant potential for the future of automated knowledge-based image analysis of biological pictures in current research. This type of analysis should be able to identify the pertinent diagnostic and defining data and convey it to the doctor more effectively in addition to enhancing diagnosis and decision-making, this ought to be able to lighten the load on attending physicians. The value of intelligent biomedical image analysis cannot be overstated, notwithstanding current doubts about how much these intelligent systems will aid in diagnosis and decision-making. Despite current uncertainties about the extent to which these intelligent systems will assist in diagnosis and decision-making, the importance of intelligent biomedical image analysis cannot be overstated [1].

Today, a wide range of applications are being considered while designing image processing systems. Additionally, they are being altered to include specialized and dedicated CPUs for particular purposes. Dedicated display environments (or peripheral devices) and high-precision computing are necessary for some applications, such as biological image analysis.

Supplementing these systems with additional processors, such as configuring parallel processing, or specialized array processors is common practice due to the need for fast computing capabilities. In this section, we will review the specifications of a traditional, yet state-of-the-art system for processing and analyzing biological images.

1.4 System Requirements for Biomedical Image processing

A general-purpose biomedical image processing and image analysis system are based on three cornerstones, namely (a) an image capture system, (b) a digital computer (analysis system), and (c) an image display environment. In more details, the biological signal or image that comes from the source must be converted in some way by the image acquisition equipment into a digital image or a large group of digital numbers that the processing computer can understand. The image capture system is a component of the equipment (the CT scanner) itself in some applications, such as computed tomography (CT). In these situations, the detection system often offers digital outputs that may be fed straight into a computer or processing unit. In certain other applications, the scanner's output takes the form of an analog image, such a chest X-ray radiograph or a film mammography. In fact, the image capture systems can include applications, such as an appropriate light source that can illuminate the radiograph (the film), as well as a digitizing camera that is ale to turn the analog image into a digital image. Laser scanners and several types of microdensitometers constitute further tools for digitizing images.

1.5 Classification in Biomedical Images

Following the identification of the pertinent items in the picture, image analysis seeks to give the doctors numerical measures. Features are taken from the segments in order to categorize them, in accordance with the processing scheme that mimics the technological stages. These characteristics often represent the biological use. However, the growth in the number of regularly obtained digital photos creates a brand-new area of study where the categorization of photographs as a whole is the main objective. Particularly, content-based image retrieval in clinical uses necessitates automatic classification of pixel data based on the imaging modality and technical parameters used, the patient's orientation regarding the image sensor, the body region being analyzed, and the biomedical system under investigation. Here, in order to improve subsequent modules of the image processing pipeline, the context must be automatically assessed because it is unknown in the future [2].

1.6 MRI

Magnetic Resonance Imaging (MRI) constitutes a non-invasive medical imaging method which employs a potent magnetic field and radio waves in order to produce high-resolution images of internal body structures. MRI is not based on ionizing radiation, which means that it is safer alternative option in comparison to, indicatively, to X-rays or CT scans. The technique is widely used to detect and track various medical conditions such as bone and joint disorders, brain and spinal cord injuries, tumors, and cardiovascular diseases [6].

1.6.1 How does MRI function

The protons residing within an atomic nucleus bear a positive charge of +1, and exhibit a precessional motion, akin to a spinning top, when subjected to a magnetic field. The rate of this precession is reliant on the type of atom and the magnitude of the applied magnetic field, which is measured in Tesla units. The resultant current produced by the precessing protons is able to be detected as well as measured. MRI scanner utilizes on one hand, a transmitting coil to generate the magnetic field and on the other hand, a receiving coil to sense the current. Subsequently, a computer system processes the signal information to create an image that depends on the location and strength of the signal response [6]. The ensuing image's appearance changes depending on the kind of bodily tissue, with fat appearing bright and ligaments and bones seeming darker. This is due to the fact that compounds with a high proton density and more hydrogen atoms create more signal than those with a lower proton density (for instance, water has more than bone). The water content of tissues is increased by many diseases (such as infections, inflammation, and tumors), and through MRI these tissues can be identified from nearby structures with lower water contents. Gray matter, which contains more water protons than white matter, may also be separated from it. Demyelination, stroke, and subarachnoid hemorrhage are other pathological characteristics that may be seen on an MRI scan [6].

1.6.2 Examples of MRI use in patients with cancer

In the treatment of patients with certain malignancies, MRI is crucial. It is utilized at all phases of the treatment procedure, including diagnosis, staging, selecting the best therapy, assessing the existence and severity of tumor recurrence, together with other imaging modalities including CT scanning and ultrasound. Compared to CT, MRI has a number of benefits, including better soft-tissue visualization. Target delineation can be accomplished using either approach. The possibility for using MRI alone for treatment planning exists for treatment locations with adequate tissue homogeneity, and estimations for radiation dosage are comparable to those obtained with CT [6].

1.6.3 Advantages and Disadvantages of using MRI

The below constitute certain potential *benefits* of MRI:

- As photos are so accurate, they frequently reveal just as much information as looking at the tissues firsthand. Because of this, using MRI can allow a patient to undergo fewer diagnostic procedures.
- Scans are significantly helpful since they can display soft tissue structures, i.e.: ligaments, cartilage, and organs (e.g., the brain, heart, and eyes).
- MRI does not use ionizing radiation (and thus does not expose patients to any possible dangers of ionizing radiation), and the magnetic fields it employs are not considered dangerous.

The following are some possible MRI *drawbacks*:

Artificial Intelligence algorithms in the digital processing of biomedical images

- Because MRI scanners are quite costly (costing over £1 million, for example), there aren't many of them. Therefore, there can be a delay for an MRI scan for non-urgent diseases. For instance, in May 2011, there were 105,990 patients on a waiting list for an MRI, and 2,152 of them had to wait for more than six weeks—among them, 575 had been waiting more than 13 weeks.
- Only a small percentage of biopsies are performed with MRI guidance, and in the majority of centers, ultrasound or CT guidance is still used.
- Geometric distortions in MRI pictures can occur as a result of problems such changes in the magnetic field's intensity.
- During an MRI scan, the patient lies on a motorized bed that moves into a narrow tube-shaped scanner with open ends. The confined space and the loud noises produced by the magnets may lead certain patients to face claustrophobia.
- When examining issues such as oral tumors, it is possible that the patient might start coughing or swallowing and as a result, it can make the resultant pictures less clear, since the MRI scanners are affected by sudden movement.
- MRI scanners are known to be noisy, which makes them unsuitable for researching disorders such as heart difficulties. After the examination, the results of the MRI scan are analyzed and reported at the diagnostic console; it is essential that an experienced radiologist or radiographer supervise the oncological MRI scan to guarantee appropriate picture quality and get accurate diagnostic information [4], [5].

1.6.4 MRI and Brain Tumors

MRI is the preferred test in order to evaluate primary cerebral neoplasms; when compared to CT it is known to be preferred for tumor identification, given the fact that it provides better resolution overall, which provides a high sensitivity to any change in the makeup of the patient's brain. It shall be noted that tumor kind and histological grade cannot be confidently decided on the basis of imaging such as MRI [6].

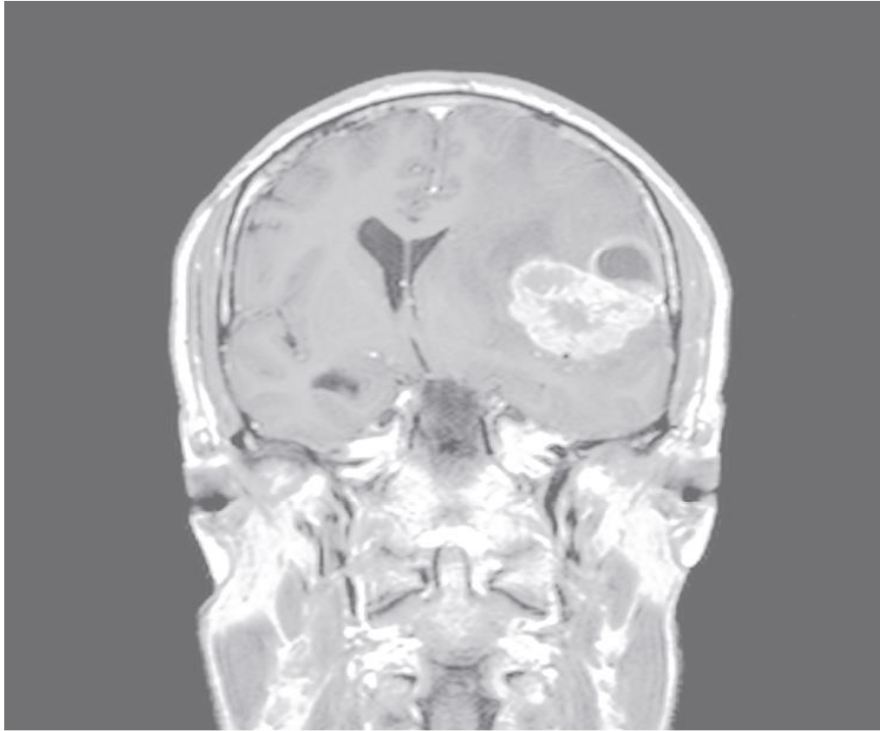


Figure 1.1: MRI Brain Scan showing a tumor in the brain [6]

Chapter 2: Machine Learning

In recent years, advances in data availability, computing power and machine learning techniques have resulted in significant breakthroughs in various scientific fields, including biology and clinical research. The application of these methods has been used to analyze molecular biology and image data, as well as in clinical practice.

Notably, though, the concept of machines learning abstract concepts has been around for more than 50 years, when the first neural networks were introduced, while approaches such as Markov chains and Bayesian statistics were used even before that. Although these techniques are known by different names in the pharmacometrics and clinical pharmacology communities, they share a common goal of using computational methods to learn and make predictions [28].

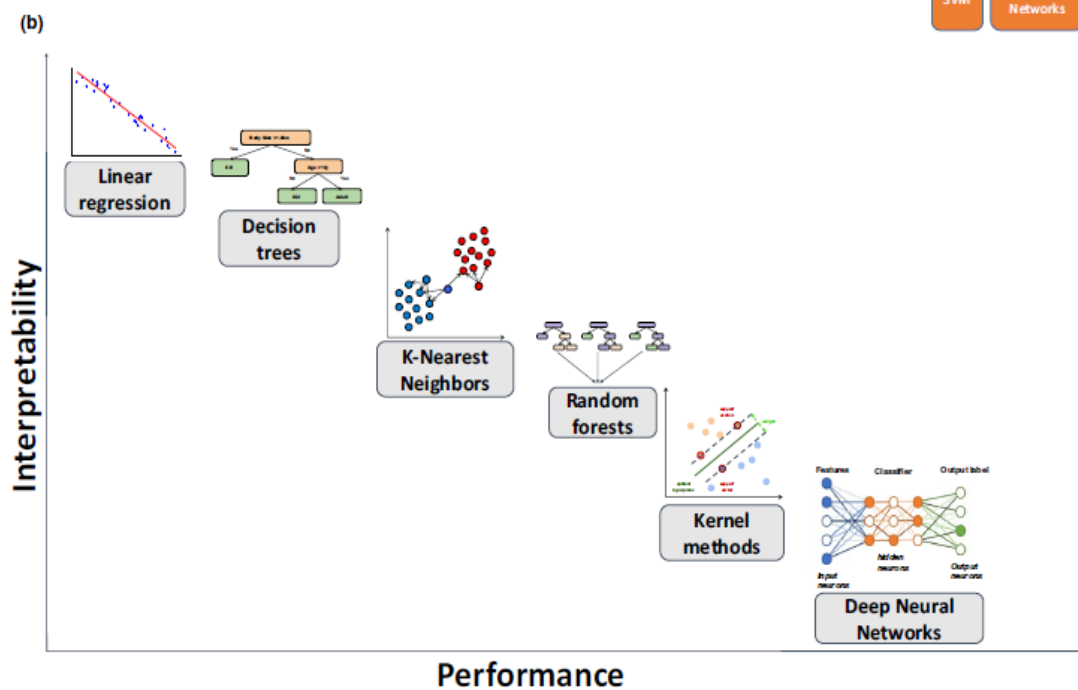
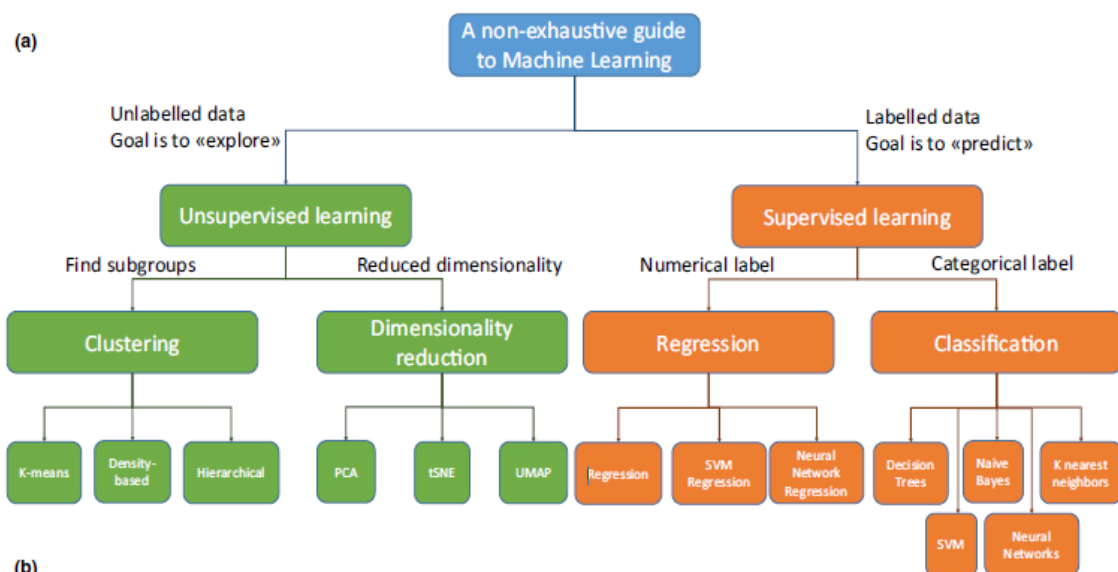


Figure 2.1: Overview and categories of machine learning algorithms. (a) Classes of various approaches are shown. (b) Overview of ML methods [28].

There are a variety of methods available for data analysis, ranging from simpler and more easily understood techniques to more complex algorithms that may offer better performance but are less transparent. The placement of these methods on a scale has to do with numerous issues, like the quantity of free parameters, the difficulty/complexity of the model, the type of data, and the specific definition of interpretability being used. Notably, positioning of methods on the scale is not an absolute measure and can be subjective.

2.1 Information and Probability Theory

The theory of probability is a concept in mathematics that helps us describe and quantify uncertain information. This theory has rules for determining new uncertain information, and it plays an essential role in artificial intelligence. There are two primary uses of probability theory in AI. First, we use probability laws to create algorithms that help AI systems reason and calculate complex expressions. Second, we can examine how the AI system behaves, using statistics and probability to ensure they are functioning optimally [43].

2.1.1 Why is probability important

Although computer science encompasses numerous areas that involve deterministic and certain entities, machine learning often needs to address aspects such as uncertainty and stochasticity. Uncertainty and stochasticity might come from a variety of sources, making it challenging to reason in their presence. Probability theory provides a framework to quantify and derive uncertain statements.

Probability theory is used in two ways in artificial intelligence applications: to design algorithms that can calculate or estimate different expressions obtained using the theory of probability, and to perform an analysis on the behavior of artificial intelligence systems using probability and statistics. Despite the deterministic nature of most computer science and software engineering environments, uncertainty is prevalent in most activities, making probability theory an essential tool for machine learning.

There are three conceivable sources that can produce uncertainty:

- First, in the system being modeled, intrinsic stochasticity, such as the probabilistic nature of subatomic particle dynamics or hypothetical card games.
- Second, inability to notice, where systems might look stochastic even though they are deterministic, when not all driving variables are observable, such as the Monty Hall problem.
- Finally, incomplete modeling, which occurs when a model discards observed information, leading to uncertainty in the model's predictions. An example of this is when a robot observes the exact location of every object but discretizes space to

predict future object locations, resulting in uncertainty about the precise position of objects.

In some cases, using a plain but unclear principle is more practical than a perplexed but specific one. Probability theory is necessary for representing and reasoning about uncertainty in AI implementations. At first, the theory of probability created to provide details in the frequencies of events, but it can also be used to represent a degree of belief in a proposition. Bayesian probability is shown to portray qualitative stages of clarity, while frequentism is used to represent the frequency with which events take place. Although these two types of probability are different, they behave in the same way when applied to common sense reasoning. Probability is an extension of logic that deals with ambiguity, and it offers a set of specific rules for estimating the probability of a proposition being accurate, based on the probability of other propositions being also accurate [43].

2.2 Data and Features

In machine learning, data is represented by datasets that contain multiple data points or samples. Each data point represents an entity that needs to be analyzed, such as a patient or a tissue sample. These data points are measured and collected using various features, which can be categorical, ordinal, or numerical. Examples of features for a patient may include their demographic information, disease history, and blood test results, as well as more complex measures such as gene expression profiles or genetic information. Every single value of the features for a particular data point are combined into a feature vector, which defines the point's location in the feature space.

The dimensionality of the feature vector and the feature space increases as the number of features increases. However, visualizing feature space dimensions of the can become difficult, so machine learning models often rely on computers to discover meaningful patterns or use minimization techniques of the dimensions to simplify the data. In pharmacometrics, longitudinal data is common, for instance, pharmacokinetic and pharmacodynamic profiles, which are based on equations derived from physiology and pharmacology. Similarly, in physical problems like weather forecasting, the temporal behavior is affected by the temperature and the air flow. Incorporating time as a continuous variable in machine learning algorithms still constitutes a challenging area of research [26].

In machine learning (ML), feature extraction, which is also known as feature engineering, is a basic step that involves selecting the appropriate features for the task at hand. It involves utilizing domain knowledge and expertise to choose features that are relevant to the problem being addressed. Once the features are chosen, data quality becomes a critical factor in ML. Drastic outliers or values can be addressed (spotted and removed) by using appropriate ML methods and visual inspection, but missing data may be more challenging to handle. Missing data is not supported by every ML methods, and in some cases, data transformation may be needed as a preprocessing step. Various approaches can be used to impute missing data, and the performance of each method has to do with the dataset and approach that was preferred [26].

Detecting biases in the data is an important aspect of machine learning, and one form of bias that needs careful consideration is selection bias. To ensure optimal performance, an unbiased and random subset of the population should be used as the samples for machine learning, but this is not always feasible in practice. Consequently, the data used for machine learning often contains biases that may bring a negative outcome on the ability of the model to proceed with generalizing beyond the training dataset, as well as the testing dataset if the biases used are similar. For instance, if the undertaken task is to make a distinction between a wolf and a husky based on their physical characteristics, but the model instead identifies snow patches in the photograph, the model's performance may suffer. To counteract these biases, machine learning engineers may choose to down-weight or exclude biased samples or features from the data. Propensity ratings are very valuable for measuring the impact of a therapeutic intervention [26].

Examining how significant the features are can yield useful insights into the extent and impact of biases, and is a recommended practice for verifying the reliability of machine learning models. In many cases, classification of clinical datasets is unbalanced, with one/more classes being underrepresented. This can present issues with various ML algorithms, such as artificial neural networks and gradient boosting methods, which may require special handling to overcome the challenges posed by the imbalance in the data. In summary, proper feature engineering, data quality assurance, and bias mitigation are essential for developing reliable and trustworthy ML models [26].

An important factor to consider in machine learning is how to define a measure of resemblance or disparity between data points in the feature space. The similarity measure is a metric that quantifies how similar or different two data points are. Choosing a similarity measure has to do with the sort of data we are working with and the specific task we want to solve [26].

For example, the Euclidean distance is the most common distance measure used for numerical feature vectors. It is determined as the square root of the sum of the squared deviations between two vectors' corresponding constituents. However, when dealing with categorical or ordinal data, other distance measures, such as the Hamming distance, are more appropriate. The Hamming distance is determined by the amount of points where the corresponding elements of two vectors differ.

Moreover, when working with text or document data, the cosine similarity is a popular resemblance metric. The cosine similarity metric computes the cosine of the angle formed by two non-zero vectors in the high-dimensional feature space. It is often used in information retrieval and natural language processing tasks, where the objective is to find similar documents or classify text data.

In some cases, the distance or similarity measure can be more complex, such as in bioinformatics, where the similarity score between two biological sequences is used to compare their structural and functional similarities. The Smith-Waterman algorithm is an example of a widely used similarity measure for biological sequences. It is a dynamic programming algorithm that computes the local alignment score between two sequences, taking into account gaps and mismatches.

In summary, the choice of a distance or similarity measure is critical in machine learning, as it affects the performance and accuracy of the models. It is essential to carefully choose the appropriate measure that best captures the characteristics of the data and the task at hand [28].

2.3 Linear Regression

Linear regression constitutes a case of supervised learning, suggesting that it's trained on a labeled dataset where the model learn estimate the outcome (also called the label) considering some input features. Regarding linear regression, the model is taught to estimate a steady output value, for example the price of a house, assuming one or more input data, such as the year it was built, the overall state on the neighborhood, the number of floors etc. [18].

Linear regression model's accuracy is measured by utilizing the mean squared error (MSE) is commonly preferred, which calculates the mean squared difference between expected and actual output values. This helps the learning algorithm into finding a set of weights that minimize the MSE on the training set, which is achieved by solving the normal equations.

$$MSE_{\text{test}} = \frac{1}{m} \sum_i (\hat{y}^{(\text{test})} - y^{(\text{test})})_i^2.$$

Figure 2.2: Mean Squared Error equation [20]

The normal equations are a set of equations that describe the optimal weights which minimize the MSE on the training set. Solving the normal equations provides a closed-form solution for the weights with a view of making predictions on new data.

Another point worth mentioning about linear regression is that it is a very simple and limited learning algorithm, and that more complex algorithms are typically used for real-world problems. However, the principles underlying the design of linear regression can be applied to more complex algorithms, and understanding linear regression is a useful starting point for learning about machine learning more broadly [41].

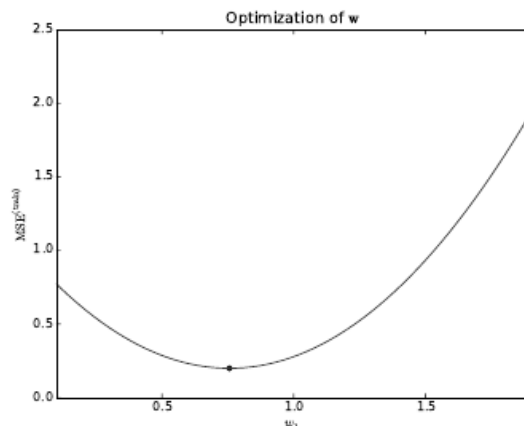
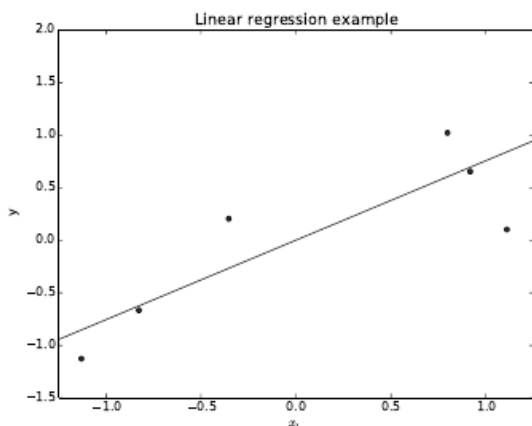


Figure 2.3: A linear regression problem with a single parameter to learn, w_1 , aiming to minimize the MSE on the training set using the normal equations. The plot shows the learned w_1 value and how linear regression tries to fit a line to the training points [41].

2.4 Logistic Regression

Logistic regression belongs to a type of model which makes predictions based on probability distributions and it is used for classification that aims to maximize the probability of the correct class label during every "loop" of training. The process of learning a model's parameters involves maximizing the likelihood of the training data through maximum likelihood estimation. This involves calculating the product of the probabilities of the observed labels for every instance, and finding the parameter values that maximize this likelihood. In logistic regression, the negative log-likelihood is used as the loss function in the output node instead of the squared error which used in the Widrow-Hoff method [28].

Logistic regression comprises of weights and a single layer to estimate the likelihood of a binary class variable being positive. The soft sigmoid function is applied to the dot product of the weights as well as the features of an instance to produce a probability estimate for the positive class. For test instances, the class with a 0.5 or greater predicted probability is chosen. The loss function for logistic regression is designed to optimize any likelihood of positive samples and negative samples separately, and this is done by maximizing $|y_i/2 - 0.5 + \hat{y}_i|$. This maximization is then consolidated into an overall expression for the likelihood L . The loss function is then set to

$$L_i = -\log(|y_i/2 - 0.5 + \hat{y}_i|)$$

for each training instance in order for the maximization to be converted to additive minimization over all training instances [28].

The gradient of the loss function in relation to the weights is then used to update the weights in the model using gradient descent. The gradient of the loss is computed as the probability of mistake on (X_i, y_i) times $(y_i X_i)$. In logistic regression, the probabilities of the errors are utilized for updates, similar to how the magnitudes of the errors are utilized for updates in the perceptron and Widrow-Hoff algorithms. Regularization is incorporated into the gradient-descent updates used in logistic regression, which are expressed as follows:

$$W \leftarrow W(1 - \alpha\lambda) + \alpha y_i X_i / (1 + \exp[y_i (W \cdot X_i)]) [28].$$

2.5 Supervised learning

2.5.1 Introduction

Supervised learning is a widely studied area in ML, with many applications in multimedia content processing. Its key characteristic is the availability of labeled training data that is used so the learning system is taught how to assign labels to new data points. This process is akin to having a "supervisor" guiding the system's learning [23]. Typically, supervised learning involves classification problems, where the goal is to assign class labels to data points. Models are induced from the labeled training data using supervised learning algorithms, and these models can be used to classify new, unlabeled data. In this chapter, our main focus is the

theory of risk minimization is the foundation of our analysis of supervised learning. A summary will be provided of the two most popular supervised learning techniques used in multimedia research: nearest neighbor classifiers and support vector machines [22].

2.5.2 The House Pricing Problem

Suppose a real estate company wants to make a prediction in relation to the expected price of a house based on certain features. The first step is to gather a dataset with instances representing singular observations of houses and relevant features. These features are the recorded properties that can be helpful for the purpose of predicting prices (for instance, the total square footage and the presence of a yard). The aim is for the feature to be predicted, which in this case is the housing price. The dataset is split into training, validation, and testing datasets, and supervised learning is used to map features to the target. The model is informed by the target, and the performance of the algorithm is evaluated on the test dataset, which is new data. The fundamental steps of supervised machine learning involve acquiring a dataset and splitting it, using the training and validation datasets to inform a model, and evaluating the model via the test dataset to determine how well it predicts housing prices for unseen instances. The performance of the algorithm is compared between the training and validation datasets, and the algorithm is tuned by the validation set. However, the performance of the algorithm may be able or not proceed with generalisation depending on how different the validation set is from the test set [30]

In supervised learning, classification and regression are some of the most typical goals. The first, involves making a prediction about the classification or category that an example falls under. To illustrate, instead of predicting the exact selling price of a house, the property firm might use classification to assume the cost in which the property is probable to be sold. To accomplish this, data engineers would bin home prices into various classes to convert the numeric goal variable into a categorical variable. These classes are ordinal, which means they have a natural order related to them. However, if the task was to determine the type of siding on a house, The categories would be purely notional, meaning have no natural order and that they are independent of one another. As far as regression, it involves predicting numerical data, i.e. such as the duration of a task, room temperature. For instance, a car rental company can use regression analysis to make predictions about the duration of a car rental based on the time of the year, the location, and the type of car etc.

2.6 Unsupervised Learning

Unsupervised learning focuses on the way systems can be taught to represent input patterns in such a way that captures the statistical structure of the entire set of input patterns, without any specific target outputs or evaluations associated with each input. Unlike supervised or reinforcement learning, the unsupervised approach relies on pre-existing biases to determine which aspects of the input structure must be reflected in the output. This learning type is critical since it is believed to be more prevalent in the brain than supervised learning [24][25].

2.7 Semi-Supervised Learning

Semi-supervised learning stands for a type of machine learning which stands between supervised and unsupervised learning. It is well-suited for datasets which contain both labeled and unlabeled data, where only some features have associated targets. In some cases, labeling all the features in a dataset can be difficult or expensive, such as in medical image analysis. In these cases, a few subset of labeled images can be used to train a model, which can then be used to classify the remaining unlabeled images. The resulting labeled dataset can after that be used to further train a model that performs better than unsupervised models.

Semi-supervised learning might be considered as a method that takes advantage of both labeled and unlabeled data to improve model performance. Through the incorporation of the labeled data, the model is exposed to learning from known outcomes and, therefore, is able to produce more accurate predictions on the unlabeled data. In this way, semi-supervised learning can save time and resources while still achieving high accuracy. However, it should be noted that the quality and quantity of the labeled data we use for the purpose of training our model is heavily impacting the performance of a semi-supervised model [30].

2.8 Reinforcement Learning

Another method we use to teach our algorithm to achieve a particular task is called Reinforcement Learning. It doesn't have a single correct answer, but rather an overall outcome that is desired. This technique is similar to how humans learn through trial and error. While reinforcement learning is a powerful method, it currently has limited applications in medicine. To illustrate, suppose an algorithm is being trained to play the video game Super Mario Bros.

The objective of the game is to navigate the character from the screen's left side to its right side while avoiding enemies and hazards. In reinforcement learning, the algorithm is given the freedom to explore and interact with the game environment by trying out various controller inputs. Whenever the algorithm successfully moves the character forward without encountering any damage, it receives a reward, reinforcing that particular behavior. With repeated trials and feedback, the algorithm gradually learns which actions are beneficial, such as moving forward rather than backward and jumping over enemies instead of colliding with them. Eventually, the algorithm learns how to complete the game [30].

Likewise in the real-world settings, the number of possible states (namely, unique positions in game) can be too large to even enumerate, let alone that the best choice of move varies significantly on the knowledge of what is substantially crucial to model from a particular state. Additionally, as one starts without any knowledge of the rules, the learning system inevitably needs to gather the relevant data through its actions much as a mouse will explore a maze in order for it get familiarized with its structure. In this respect, the data that is gathered is highly biased by the actions of the user, which could be argued that it provides a particularly challenging landscape for learning [28].

The successful training of reinforcement learning methods is a critical gateway for self-learning systems, which is the holy grail of artificial intelligence. Although reinforcement

Artificial Intelligence algorithms in the digital processing of biomedical images

learning has potential in computer science and machine learning, until now it has not contributed to a significant impact in the field of clinical medicine [30].

2.9 Support Vector Machines

In the 1990s, a learning algorithm called support vector machines (SVM) was developed. This category of machine learning is based on statistical learning theory, which was previously introduced by Vapnik. Kernel functions that constitute a fundamental concept for various learning tasks, are also closely related to SVM. SVM and the kernel framework have been used in a wide range of areas including multimedia information retrieval, and pattern recognition. This discussion will focus on the use of SVM as linear discriminant functions for binary classification. A more comprehensive explanation of SVM and kernel theory can also be found in other sources [22].

2.10 The Concept of Overfitting

Neural networks, in general, are capable of approximating any function theoretically, but in the event of a data shortage available for training, this capability can be compromised when in the event that there is a shortage of data. The sparsity of training data can result in the model learning spurious patterns from the random nuances present in the data. This results in overfitting, where the model becomes highly predictive on the training data but is cannot generalize satisfactorily on unseen test data.

In the example of a linear regression model with few training points, there are infinitely many solutions that can provide zero error on the training data. These solutions, though, are not expected to generalize adequately to new points where the target is twice the first attribute and other attributes are random. When training data is scarce, models with a larger variety of parameters tend to have higher variance, making them more susceptible to overfitting. In order for the negative consequences of overfitting to be minimized, careful design methods are needed.

To improve the ability of a model to generalize, it is beneficial to increase the quantity of training instances, while increasing the complexity of the model can have the opposite effect. A general guideline suggests that the number of training data points have to be about 2 to 3 times larger than the number of parameters in the neural network, despite the fact that this may vary depending on the specific model. Models that are too simplistic may not capture the intricate relationships between features and the target, while excessively complex models may detect spurious patterns in random variations, particularly when training data is scarce. Therefore, selecting the appropriate level of model complexity is a critical decision. Models that are excessively simple may fail to capture intricate relationships between features and the target, whereas models that are excessively complex can be susceptible to detecting spurious patterns in random variations, particularly when the training data is limited. It is crucial to choose a suitable level of model complexity to strike a balance between these two extremes. This ensures that the model is capable of capturing relevant patterns in the data while avoiding overfitting, ultimately leading to better generalization performance [28].

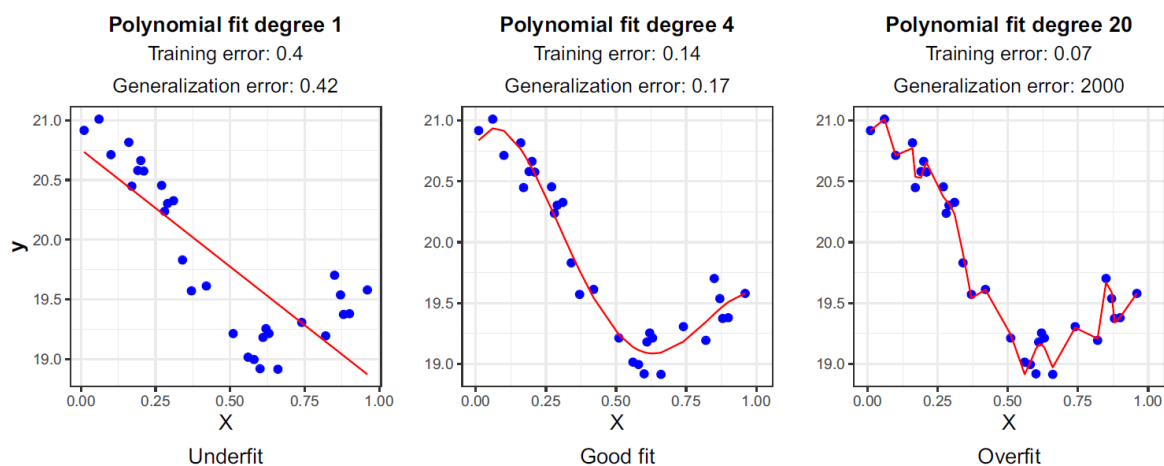


Figure 2.4: Differences between an Underfit, Overfit and a good realistic fit. [26]

2.11 Activation Functions

In neural networks we use “activation functions” to introduce nonlinearity into the output of our model. Without a nonlinear activation function, the output of the neural network would be a linear combination of the input, which severely limits the model's ability to learn complex patterns in relation to data.

The activation function takes the weighted sum of the inputs and the biases and applies a mathematical function to it. The result is the output of the neuron. Different activation functions are able to be used to change the the output's character, depending on the task.

For example, the sigmoid function outputs values between 0 and 1, which is useful for binary classification tasks where the output is interpreted as a probability. The rectified linear unit (ReLU) function is a popular choice for deep learning, since it's computationally efficient and helps mitigate the vanishing gradient problem. The tanh function is like the sigmoid function but outputs values between -1 and 1, which can be useful for tasks that require negative values [30].

2.11.1 Commonly used Activation Functions

There is a considerable amount of activation functions used in AI, including:

- The **Sigmoid function**, which is a popular choice for binary classification problems, as it maps the input to a probability between 0 and 1.

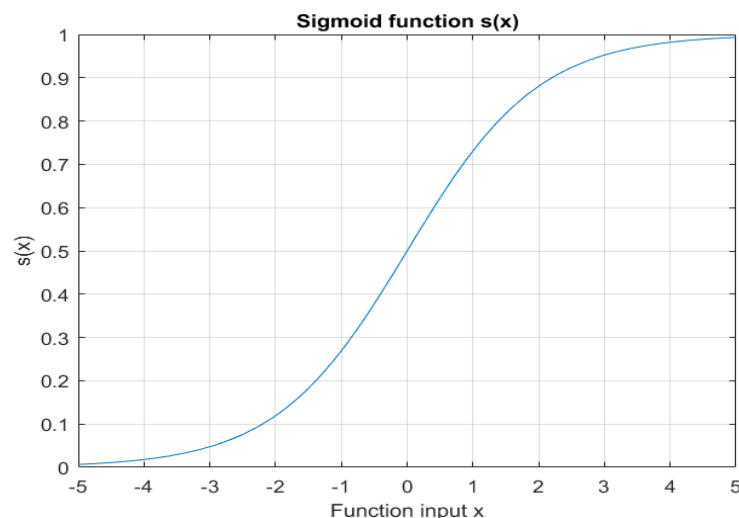


Figure 2.5: The Sigmoid Function

- The **Hyperbolic tangent (Tanh) function** is similar to the sigmoid but maps inputs to the range of -1 to 1, which can be useful for tasks that require negative values.

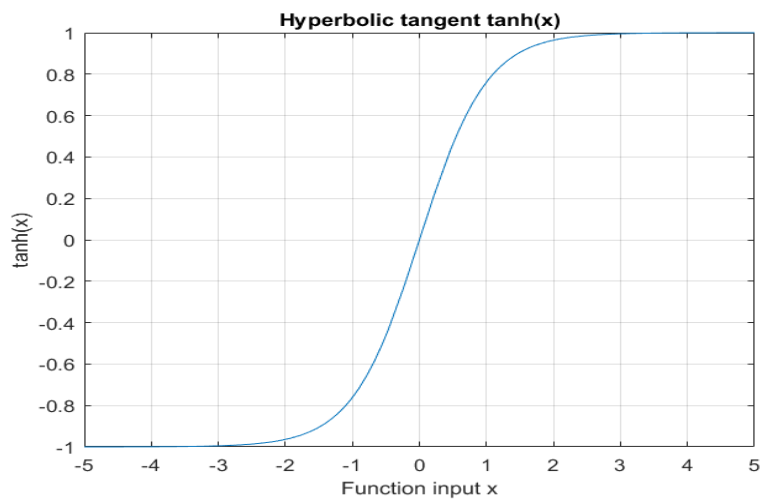


Figure 2.6: The Hyperbolic tangent (Tanh) function

- The **Rectified Linear Unit (ReLU) function**, which has become a prominent choice in recently due to its simplicity and effectiveness in training deep neural networks. The ReLU function returns zero for negative inputs and the input itself for positive inputs. It can be argued that ReLU is computationally efficient and has a fast gradient propagation, allowing for faster training times.

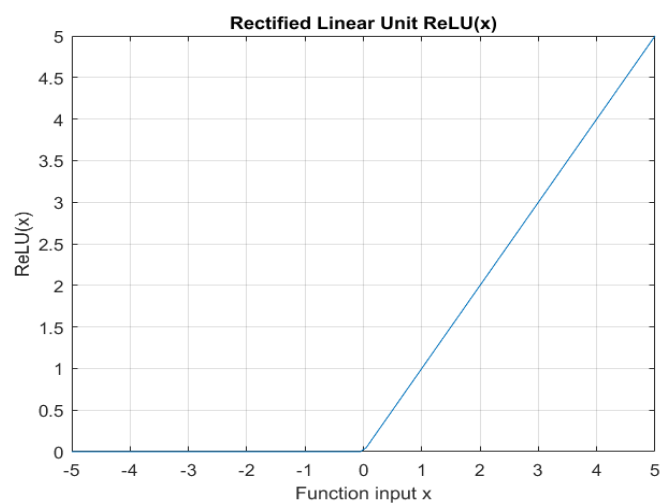


Figure 2.7: Rectified Linear Unit (ReLU) function

2.11.2 How to choose the activation function

The selection of an appropriate activation function is a crucial factor in developing effective neural networks. The selection of activation function depends on the nature of the target variable to be predicted. For instance, the sign function is commonly used for binary classification, the identity function for regression, and the sigmoid function for predicting probabilities. In multi-layered architectures, nonlinear activation functions are particularly crucial. Neurons within these networks compute both pre-activation and post-activation values, and functions such as the sigmoid, tanh, ReLU, and hard tanh are frequently employed as activation functions. These functions are monotonic and reach a saturation point at large absolute values of the argument. By utilizing nonlinear activation functions, neural networks can increase their modeling power, allowing them to perform significantly better than a single-layer linear network.

2.12 Regularization

In ML, overfitting can be a significant problem that leads to poor generalization performance. Regularization is a technique that helps to prevent overfitting by limiting the number of non-zero parameters or reducing the absolute values of the parameters in a model. This can be achieved by adding a penalty term $\lambda \|W\|_p$ to the loss function, where p is typically set to 2. The penalty term can be viewed as weight decay during updates, which gradually removes noisy patterns and less important features from the model. Regularization is particularly useful when the amount of data available for training is limited. It is interesting to note that weight decay is not commonly used in the single-layer perceptron, and other regularization techniques are employed instead. The general form of the regularized update equation is widely used in regularized machine learning models like least-squares regression. Generally, it is recommended to use more complex models with regularization rather than simpler models without regularization to achieve better performance [28].

2.12.1 Early Stopping

An other type of regularization is called early stopping. This method stops gradient stops after a couple of cycles. The stopping point is being chosen by testing the model's error on a collection of training data that has been kept back. Early stopping effectively lowers the parameter space to a narrower area inside the parameters' initial values, acting as a regularizer.

2.12.2 Ensemble Methods

Ensemble methods are a family of strategies for improving performance by combining multiple models and thus reducing variance in predictions. Bagging is a popular ensemble method that is applicable to many machine learning algorithms, including neural networks. In recent years, several neural network-specific ensemble methods have been proposed, including Dropout, Dropconnect, Stochastic Weight Averaging (SWA), and Snapshot Ensembling.

These methods work by introducing random variations to the network during training, such as dropping out or connecting certain neurons, to prevent overfitting and improve generalization. While in real-world conditions there is a 2% improvement in dropout and dropconnect, the effectiveness of these methods can depend on the style of the training and the type of data. For instance, dropout methods may be less effective if activations in hidden layers are normalized, but normalization itself can still provide benefits. Overall, ensemble methods might be particularly useful in scenarios with imbalanced datasets or when interpretability is important [30].

2.13 Backpropagation

Training a multi-layer neural network is more complex than one that has only one layer, because the weights in all layers are depended on the loss function, making it difficult to compute gradients. The backpropagation algorithm addresses this difficulty by employing dynamic programming to determine the gradient of the loss function in an efficient manner. The algorithm has two phases: forward and backward.

The input is supplied into the network and the output is generated throughout the forward phase. The loss function's derivative with regard to the output is then computed. In the backward phase, the loss function's gradient, is learned by using the chain rule of differential calculus. Such procedure is known as the backward phase because we are reversing the order in which the gradients are learned, with the output node being the first. The backpropagation algorithm is a powerful tool for training complex neural networks with multiple layers [17], [28].

2.14 Neural Networks

Artificial neural networks (ANNs) are prominent ML techniques that imitate the learning mechanism in living entities. Human nervous systems comprise of cells known as neurons, which are connected to each other through axons and dendrites, with the connecting regions called synapses. The strength of these connections changes in response to external stimuli, enabling learning in living organisms [38].

In artificial neural networks, computational units known as neurons are connected to one another through weights that simulate the strength of connections in live organisms. The input to a neuron is adjusted with a weight that impacts the output calculated at that unit. The network computes an input function by propagating computed values from the input neurons to the output neuron(s) and using weights as transitional parameters. The weights connecting the neurons are changed during training, using input-output pairs of the function to be learned, known as training data.

The training data is used to provide feedback to the correctness of the weights in the neural network. The errors made during the computation of a function serve as feedback to adjust the weights between neurons to improve future predictions. By adjusting the weights between neurons over many input-output pairs, the function computed by the neural network is refined over time to provide more accurate predictions.

Model generalization refers to the capacity to compute functions of unseen inputs accurately by training on a limited set of input-output pairs. The capacity of machine learning models to generalize their learning from observed training data to unseen samples is critical to their effectiveness [31].

Despite criticisms of the biological analogy, neural networks have proven to be a useful tool in designing machine learning models. Rather than being a direct emulation of the human brain, neural networks can be seen as a higher-level abstraction of classical ML models. Fundamentally, a neural network is made up of basic computational units, which are often inspired by traditional algorithms like regression. However, the real power of a neural network lies in the way these basic units are connected and the weights are trained jointly.

By connecting multiple units, the neural network gains the ability to learn more complex functions of the data. This requires a deep understanding of how the units should be combined and sufficient training data to train the expanded computational graph effectively. When utilized in its most fundamental way, an ANN may reduce to a classical machine learning model, but by connecting multiple units, the neural network can achieve far greater predictive power [28].

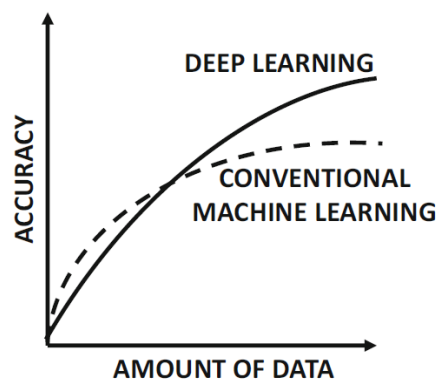


Figure 2.8: Comparison between conventional ML and Deep Neural Networks. [28]

2.15 Architectures of Neural Networks

2.15.1 The Perceptron

As an example, in the following part, we will present a neural network that classifies written numbers. In preparation for that, it is useful to explain basic terminology that allows us to name various sections of a network. Assume we have the network:

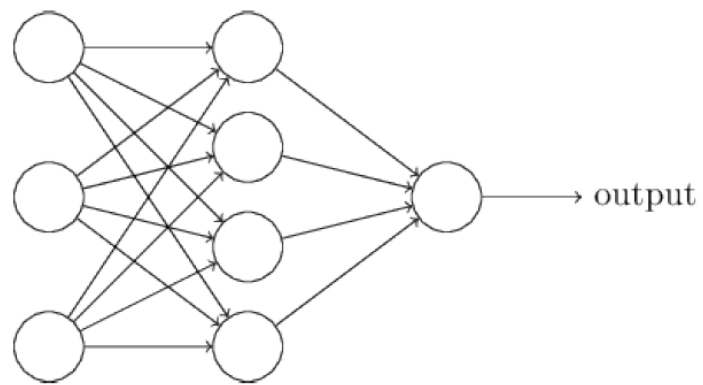


Figure 2.9: A representation of the Perceptron [28]

2.15.2 Feed Forward Neural Networks

As stated previously, the initial layer of this network is known as the input layer which is made up of input neurons. The final layer, or output layer, comprises output neurons, that in this scenario are just one. In between these two layers, there is a layer known as the “hidden layer”, which consists of a more neurons. We call those neurons "hidden" and although it might sound strange at first, it merely means that these neurons do not serve as inputs or outputs. The illustrated network has only one hidden layer, but there are other networks that have many more hidden layers [39]. The following four-layer network, for instance, has two hidden layers:

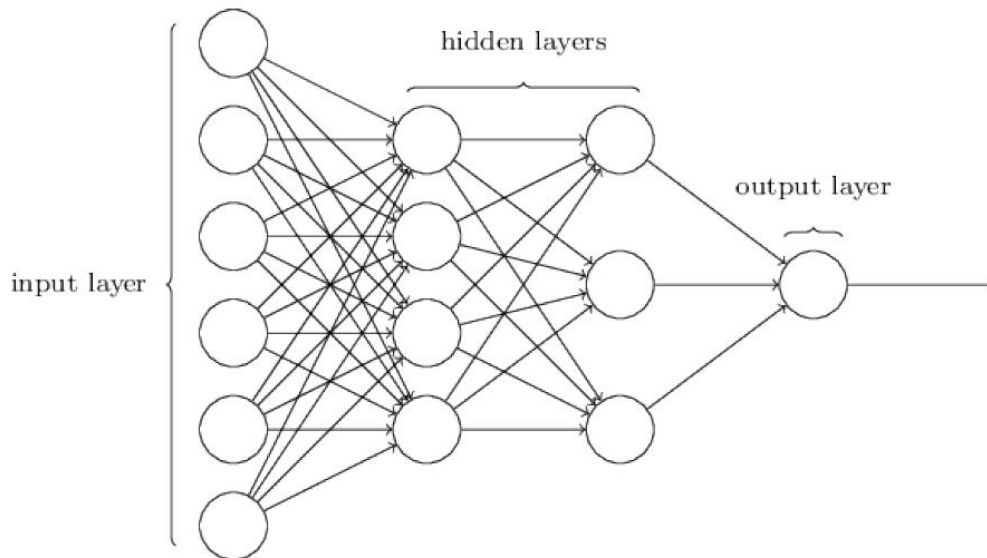


Figure 2.10: An Example of a Neural Network [29]

Neural networks can have multiple layers, and these are sometimes referred to as multilayer perceptrons (MLPs), even though they don't actually use perceptrons. The input layer contains input neurons, while the output layer contains output neurons. The design of these layers is often straightforward, with input neurons representing input data and output neurons producing the desired output.

However, the design of the hidden layers can be more challenging, and there and we have to take initiative in order to find the perfect fit. Researchers have developed heuristics to help with designing the hidden layers, taking into account factors like the available training time and the number of the overall layers we use.

Feedforward neural networks are the most preferred type of ANNs, where information flows only in one direction and there are no loops in the network. Recurrent neural networks, on the other contrary, allow for feedback loops and are more similar to the way our brains work. However, the learning algorithms for recurrent networks are not as powerful as those for feedforward networks [29].

2.16 Convolutional Neural Networks

2.16.1 Introduction

A Convolutional Neural Network (CNN) is a type of deep neural network that is mainly used for image and video analysis. It uses a special mathematical operation called convolution, that assists the network to learn visual features from the input data. CNNs are made from several layers, that include among others convolutional layers, pooling layers, and fully connected layers, which work together to learn and classify visual patterns in the input data. CNNs have become one of the most widely used and effective methods for image recognition, object CNNs are considered one of the most important neural networks in the area of deep learning and have been used in various applications such as face recognition, autonomous vehicles, and medical treatments. The development of CNNs is rooted in the Artificial Neural Networks (ANNs), which started with the MP model by McCulloch and Pitts and then evolved into the single-layer perceptron by Rosenblatt and multilayer feedforward networks by Rumelhart. CNNs were introduced by Zhang and LeCun and were further developed by Krizhevsky, who proposed deep CNNs for the ImageNet Large Scale Visual Recognition Challenge that led to the advancement of deep learning [32].

2.16.2 An overview of how CNNs work

A CNN is a type of neural network that applies convolution structures to extract features from data automatically. This eliminates the need for manual feature extraction. The design of CNN is modeled on the human visual perception system, with artificial neurons corresponding to biological neurons, and kernels functioning as receptors that detect various features. Activation functions determine which neural electric signals are transmitted to the next neuron by simulating the threshold function. Loss functions and optimizers are used to train the entire CNN system to learn and perform the expected tasks [32].

Compared with fully connected (FC) networks, CNN possesses several advantages, including local connections, weight sharing, and downsampling dimension reduction. Local connections mean that each neuron is connected to only a small number of neurons from the previous layer, which assists in reducing parameters while speeding up convergence. Weight sharing allows an array of connections to share the same weights, which reduces parameters further. Pooling layers are used to downsample an image and reduce its dimensionality. They do this by exploiting the local correlation principle in images, which retains important information while reducing the amount of data. This process is called downsampling, and it is a type of dimension reduction technique used in CNNs. [32].

Typically it can be argued that four are the main components that are required to build a CNN model: convolution, padding, stride, and pooling.

In CNNs, the process of convolution is a crucial stage in the procedure of extracting features from data. The results of convolution are referred to as feature maps. To set the size of the convolution kernel, padding is added to indirectly adjust its size, while stride is used to control the density of the convolution. This ensures that the extracted features are accurate and meaningful. Following convolution, the resulting feature maps contain a multitude of features

that may lead to overfitting, which is why pooling (downsampling) is used to obviate redundancy.

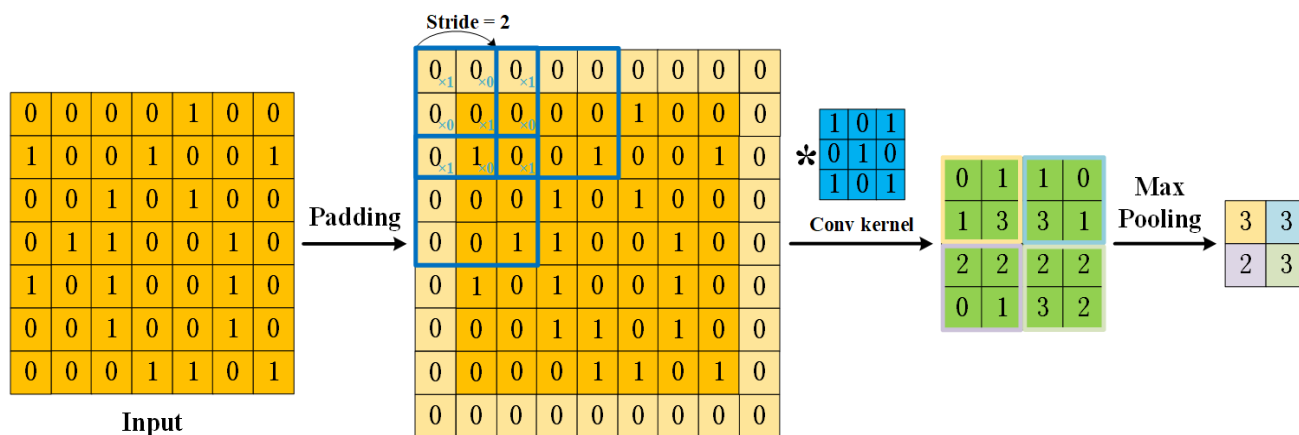


Figure 2.11: A visual representation of how a Convolutional Neural Network works [32].

Dilated convolution was proposed to enable convolution kernels to perceive larger areas. A general 3x3 convolution kernel is shown in Fig. 2.12(a), while a two-dilated 3x3 convolution kernel and a four-dilated 3x3 convolution kernel are shown in Fig. 2.12(b) and (c). The valid kernel points are still 3x3, but a two-dilated convolution has a 7x7 receptive field, and a four-dilated convolution has a 15x15 receptive field.

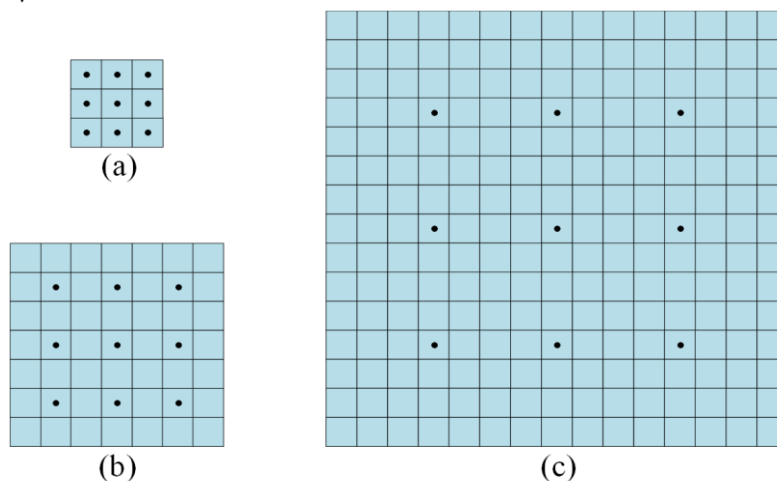


Figure 2.12: Illustration of the difference between a general convolution kernel and a dilated convolution kernel [32].

2.17 Overall Architecture

As discussed above, CNNs are specifically designed to handle image data and their architecture is optimized for this purpose. One of the key differences between CNNs and the rest types of neural networks is the three-dimensional organization of their neurons. The neurons in a CNN layer are arranged in a three-dimensional volume, which includes the spatial dimensions of the input (height and width) and a third dimension known as the depth. The depth of the activation volume does not have to do with the quantity of layers in the network, but rather to the number of feature maps in a given layer.

Artificial Intelligence algorithms in the digital processing of biomedical images

In contrast to traditional ANNs, where every neuron in a layer connects with every neuron in the preceding layer, the neurons in a CNN layer only connect to a small region of the previous layer. This is known as the local receptive field, and it allows CNNs to learn and recognize local patterns and features within an image.

For example, in a CNN designed to classify images, the input "volume" would have a dimensionality of $64 \times 64 \times 3$ (representing the height, width, and depth of the image), and the final output layer would have a dimensionality of $1 \times 1 \times n$, where "n" represents the possible number of classes. The CNN is able to condense the full input dimensionality into a smaller volume of class scores filed across the depth dimension, which allows it to classify images with high accuracy.

The case of CNN can be broken down/analysed into four main layers:

1. The **input layer**: receives the raw data, in this case, the pixel values of the image. Each pixel value is treated as a separate input node, forming the input volume of the CNN.
2. The **convolutional layer** is where the actual processing of the input image takes place. The layer comprises multiple filters, each responsible for detecting a specific feature in the input image. The filters are slid over the input volume in a sliding window fashion, computing the dot product between their weights and the region of the input they are connected to, producing a 2D activation map. We then apply the ReLu activation function the output of the activation produced by the previous layer. This ensures that the output is non-linear and can capture more complex features..
3. The **pooling layer** downsamples the output of the previous convolutional layer by a factor of 2 or more, reducing the spatial dimensions of the activation volume. This is done to reduce the computational complexity of the network and to improve its generalization ability.
4. The **fully-connected** layers will then execute the same functions as typical ANNs, trying to generate class scores from the activations for classification. One can also recommend that ReLu can be utilized between these layers in order to boost efficiency. [33].

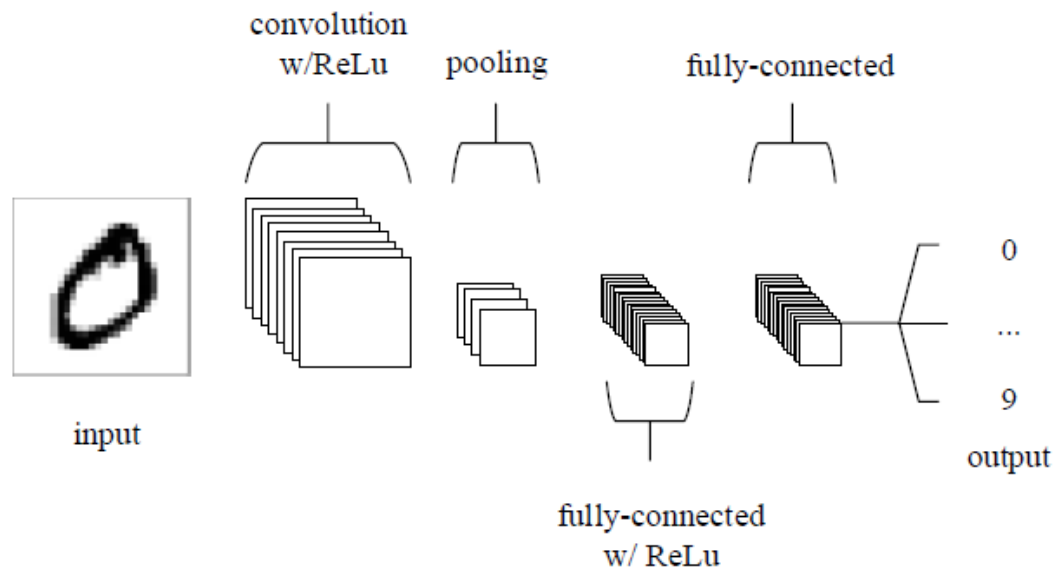


Figure 2.13: A 5-Layered CNN model [33]

CNNs can transform the input layer using convolutional and downsampling techniques in order to produce class scores to be used for classification and regression purposes. However, it is not enough to just understand the overall architecture of a CNN. Creating and optimizing these models can be time-consuming and challenging. Therefore, it is important to delve into the individual layers of the CNN, including their hyperparameters and connections, to fully comprehend how they work. [33].

2.18 Convolutional layer

The convolutional layer represents a crucial component of CNN. It uses learnable kernels to convolve across the spatial dimensionality of the input and produce a 2D activation map. Each kernel learns to fire when it recognizes a specific feature at a given spatial position of the input. The receptive field size of each neuron in a convolutional layer is small, which reduces the complexity of the model.

The depth, stride, and zero-padding are hyperparameters that can be optimized to further control the output of the convolutional layer. Parameter sharing is a technique that reduces the number of parameters being produced by the convolutional layer. Overall, convolutional layers substantially minimize the model's complexity and allow for effective training of CNNs.

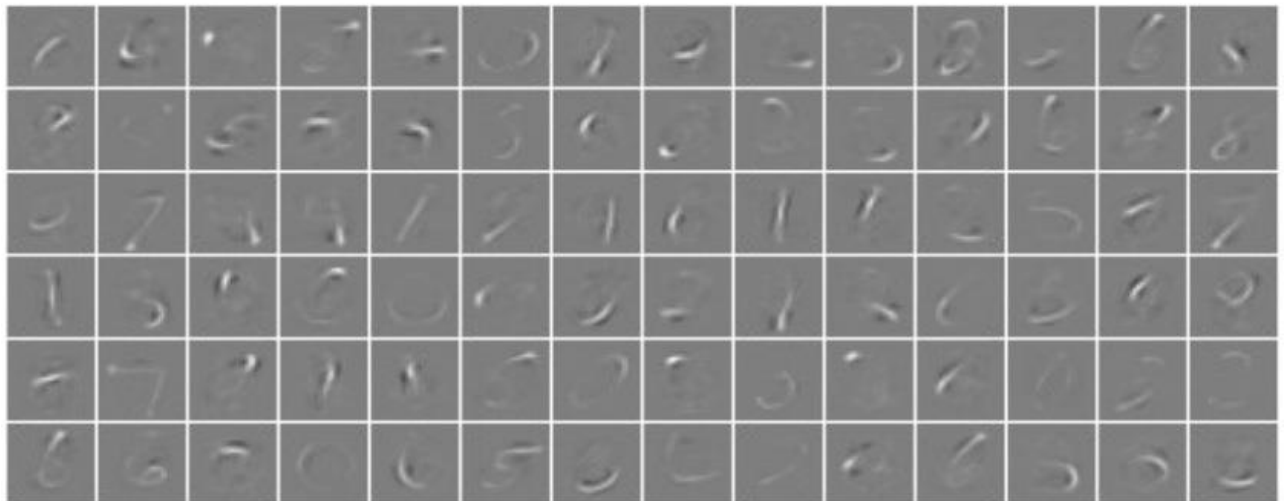


Figure 2.14: An inside look after the first convolution of a deep neural network on a dataset of digits [33].

The convolution layer produces feature maps, which enhance the distinctive characteristics of the original image. This layer operates differently from other neural network layers, as it doesn't use connection weights and weighted sums. Instead, it uses convolution filters to transform the image and generate the feature map. Figure 2.15 illustrates this process, where the * symbol represents the convolution operation and ϕ represents the activation function. The convolution layer creates one feature map for each convolution filter. For example, if the layer has four filters, it will produce four feature maps [36][40].

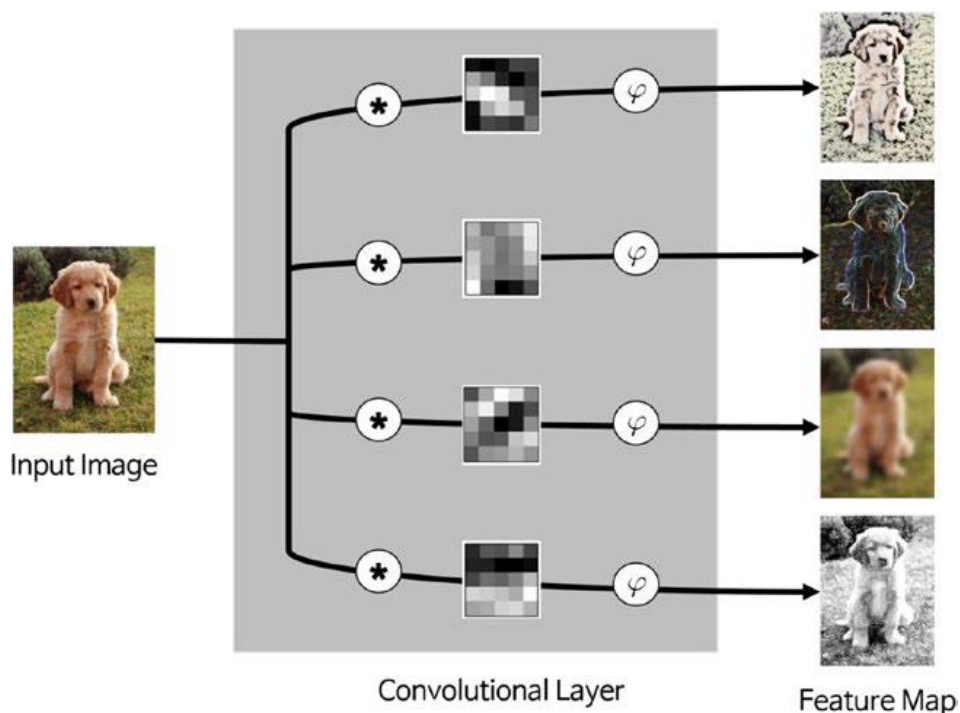


Figure 2.15: The convolution layer process and Feature Map [36]

2.19 Pooling Layer

The purpose of pooling layers in CNNs is to reduce the dimensionality of the representation and the computational complexity of the model by using the "MAX" function to scale down the activation map.

Typically, max-pooling layers with 2x2 kernels and a stride of 2 are applied to the input's spatial dimensions, resulting in a 25% reduction in size while maintaining the depth volume's standard size. Since pooling is destructive, there are only two common max-pooling methods, both with a 2x2 stride and filter size, and occasionally overlapping pooling with a 3x3 kernel size. CNN architectures may also include general-pooling layers that perform various common operations such as L1/L2-normalization and average pooling, but this tutorial will mainly analyse max-pooling.

Chapter 3: Experimental Part: Multi-Class Classification of Brain MRI Tumors Using EfficientNetB0 and ResNet18

3.1 Introduction

In this section, we will focus on developing a classification model to identify different types of brain tumors in MRI brain scans. To achieve this, we will use pre-trained convolutional neural network (CNN) architectures. CNNs are widely considered modern models for classifying images, as they can learn to detect key elements from our data themselves.

To adapt the pre-trained CNNs to our specific classification task, we will employ transfer learning. This involves taking a pre-trained model, removing the final classification layers, and training a new set of layers on our specific dataset. By doing this, we can leverage the knowledge that the pre-trained model has learned about image features, and focus on training the new set of layers on our specific task.

To increase the effectiveness of our models, a series of data augmentations is going to be implemented. This involves creating new, slightly modified versions of our existing images by applying operations such as rotations, translations, and flipping. This aids in boosting the size of our dataset and can also contribute to decreasing overfitting.

We will also carefully adjust the hyperparameters of our models. Hyperparameters are settings that control how the model learns from the data, and they can have a big influence on the model's accuracy. By carefully tuning these hyperparameters, we can ensure that our models perform as well as possible on our specific task.

To evaluate the model's efficacy, we will evaluate the juxtaposition of the accuracy of two well-known CNN architectures: EfficientNetB0 and ResNet18. We will do this by training both models on our dataset and evaluating their performance on a brand new test set. By doing this, we can get a fair comparison of how well each model performs on our specific task.

We will implement all of the above in Python, specifically by using PyTorch. PyTorch refers to a fully-featured framework regarding building deep learning models that is mainly used in applications, such as image recognition and language processing. It is written in Python, which makes it relatively easy to be learnt and used by most machine learning developers. At the end of the thesis, we will provide the code used to develop and train our models, so that others can reproduce our results and build upon our work [51].

3.2 Tumors regarding current thesis

3.2.1 Introduction

According to the Centers for Disease Control and Prevention (CDC), cancer ranks as the second the most prevalent reason of fatality worldwide. Death because of cancer can sometimes be avoided through early detection, although this is not always practicable. In comparison to cancer, a tumor might be relatively harmless, precancerous, or aggressive. In

Artificial Intelligence algorithms in the digital processing of biomedical images

contrast to malignant tumors, benign tumors may be surgically removed and often do not metastasize to further tissues and organs. [43].

Meningiomas, gliomas, and pituitary tumors are a few examples of primary brain tumors. Gliomas are tumors that develop in brain regions besides nerve cells and blood vessels. Pituitary tumors, on the contrary, are masses that exist within the cranium, and meningiomas are tumors that develop from the membranes that surround and cover the brain and central nervous system. Meningiomas often have benign characteristics, but gliomas typically have malignant characteristics; this is the most significant distinction between these three types of tumors [9].

Contrary to meningiomas, which are slow-growing tumors, pituitary tumors, even when benign, can nonetheless harm other parts of the body. Due the facts provided above, the accurate classification between these three types of tumors constitutes a very essential phase of the clinical diagnosis process and subsequently successful assessment of patients.

MRI is the most commonly employed strategy for tumor type differential diagnosis. It is subject to human subjectivity, though, and it is challenging for a person to see a lot of data. The ability of the radiologist to detect tumors early largely depends on his or her training. Before determining whether the growth is benign or aggressive, the diagnostic process for it could not be settled. A biopsy is often carried out to determine if the tissue is benign or cancerous. The biopsy of a brain tumor is often not performed prior to final brain surgery, in contrast to cancers located elsewhere in the body. It is crucial to provide an efficient diagnostics tool for tumor segmentation and classification using MRI in order to acquire exact diagnosis, prevent surgery, and subjectivity.



Figure 3.1: A closed MRI scanning system [44]

The advent of technological innovation, particularly ML and AI, has had a tremendous impact on the medical profession and has given several medical branches, including imaging, a crucial support tool. In MRI image processing, several machine-learning approaches for data classification and segmentation are provide radiologists with a second opinion.

The first objective of this thesis is to investigate how a CNN classifies 3 distinct cancer kinds from a balanced dataset. This dataset is still much smaller than datasets typically used in the field of artificial intelligence, despite being regarded as large compared to other MRI image datasets that are currently available.

3.2.2 Meningioma

Meningiomas are brain tumors that develop from the meninges, the thin tissues that coat the spinal cord and brain. They constitute the most known type of primary brain tumor, and while the majority of such are benign, some can be atypical or malignant. Meningiomas can occur anywhere along the brain and spinal cord, but they are most commonly found near the surface of the brain. Symptoms of meningiomas can vary widely depending on the location and size of the tumor, but may include headaches, seizures, changes in personality or behavior, vision problems, difficulty with speech or coordination, and weakness or numbness in the limbs [9].

Many factors influence meningioma treatment, including the tumor's magnitude and area, the person's age and general wellbeing, and the presence of any underlying medical conditions. Surgery is the primary treatment for meningiomas, in order to remove as many parts of the tumor as necessary. Radiation therapy and chemotherapy may also be used in some cases. While the long-term outlook for meningiomas is generally good, some individuals may experience a recurrence of the tumor after treatment, and others may develop additional meningiomas over time.

It is important for individuals who have been diagnosed with a meningioma to have regular follow-up care and monitoring to ensure that any recurrent or new tumors are detected and treated promptly [9], [10].

3.2.3 Glioma

A glioma is a form of brain tumor that develops from the glial cells, which are the supporting cells that surround and feed the brain's neurons. Gliomas can be either benign or malignant and are classified based on the specific type of glial cell they originate from. The most common types of gliomas include astrocytomas, oligodendrogliomas, and glioblastomas. Gliomas can occur at any age, but they are most commonly diagnosed in individuals who are in their 40s or 50s.

The exact cause of gliomas is not fully understood, but certain risk factors such as genetic predisposition, exposure to ionizing radiation, and environmental toxins may increase the likelihood of developing this type of brain tumor. Symptoms of gliomas can vary depending on the size, location, and grade of the tumor, but common symptoms include headaches, seizures, changes in personality or behavior, weakness on one side of the body, and difficulty with speech and vision.

Treatment options for gliomas depend on the type, size, and location of the tumor as well as the individual's overall health and other factors. WHO grade II and III gliomas (LGG) are less frequent and affect younger patients. Based on the molecular subtype, they have a better prognosis and exhibit some sensitivity to treatment [13]. Gliomas are often treated first with

surgery, with the intention of eliminating as much of the cancer as feasible. Radiation therapy, chemotherapy, and targeted therapy are all possible treatments. The prognosis for gliomas varies depending on the type and grade of the tumor, but in general, the outlook for malignant gliomas is poor, with a median survival of only a few years [11], [12].

3.2.4 Pituitary Gland

The pituitary gland is a pea-sized gland near the base of the brain. It is also referred to as the "master gland" as it produces and regulates a number of hormones that control various bodily functions, including growth, reproduction, metabolism, and stress response. Pituitary tumors are abnormal growths that can develop in the pituitary gland. They can be either benign or malignant, and can cause a range of symptoms which vary on their size and location.

The exact pathogenesis of pituitary tumors is unknown, but several determinants have been discovered, including certain genetic conditions and radiation exposure. Symptoms of pituitary tumors can include headaches, vision problems, fatigue, loss of libido, menstrual irregularities, and abnormal growth. In some cases, pituitary tumors can also cause excess production of certain hormones, leading to conditions such as acromegaly, Cushing's disease, or hyperprolactinemia.

Treatment options for pituitary tumors depend on the type and size of the tumor, as well as the severity of symptoms. In some cases, close monitoring of the tumor may be all that is required, while in other cases, surgery or radiation therapy may be necessary. Medications may also be used to control hormone levels or reduce the size of the tumor.

The prognosis for pituitary tumors varies depending on the type and grade of the tumor, as well as the response to treatment. Benign pituitary tumors are generally associated with a good prognosis, while malignant pituitary tumors are more aggressive and have a poorer prognosis. However, with early diagnosis and appropriate treatment, many individuals with pituitary tumors are able to manage their condition and maintain a good quality of life [14], [15].

3.3 Experiment #1: Multi-Class Classification of Brain MRI's using EfficientNetB0

3.3.1 Dataset

For this thesis we used a set of images that were collected from various sources and include 5,525 MRI images of the brain, which are divided into different categories according to each class and set. The images are further grouped into four categories - meningioma, glioma, pituitary gland tumor, and healthy brain MRIs. Each class consists of approximately 1300 images to ensure a balanced dataset, and a validation set consisting of about 400 images for each class is used to assess the model's accuracy and prevent overfitting. The images are preprocessed to ensure a uniform size and a consistent orientation.

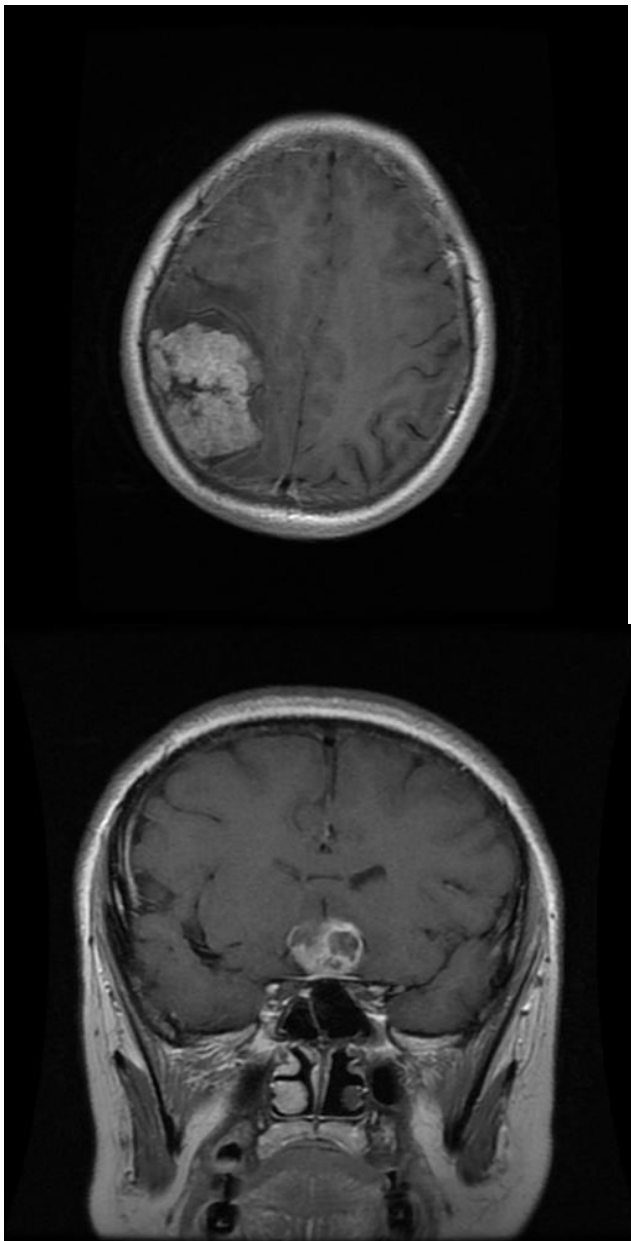
During training, the pre-trained EfficientNetB0 and ResNet18 CNN architectures will be fine-tuned and adapted to the specific task of classifying different types of brain tumors. The Adam optimizer and a cross-entropy loss function will be used to develop the model. To improve the

Artificial Intelligence algorithms in the digital processing of biomedical images

model's generalization and robustness, various data augmentation techniques will be employed, such as random rotations, flips, and brightness adjustments. The hyperparameters of the model will be tuned using a grid search approach, and The model with the highest validation set performance will be chosen.

Once the model is trained, its performance will be examined on a distinct testing dataset consisting of 100 new images for each class. The model's accuracy will be measured by comparing its predictions to the ground truth labels. This study has the possibility to enhance the accuracy and effectiveness of brain tumor detection, which can eventually result in improved patient outcomes.

Below are some examples of photos from the training set, one corresponding with each of the 4 classes.



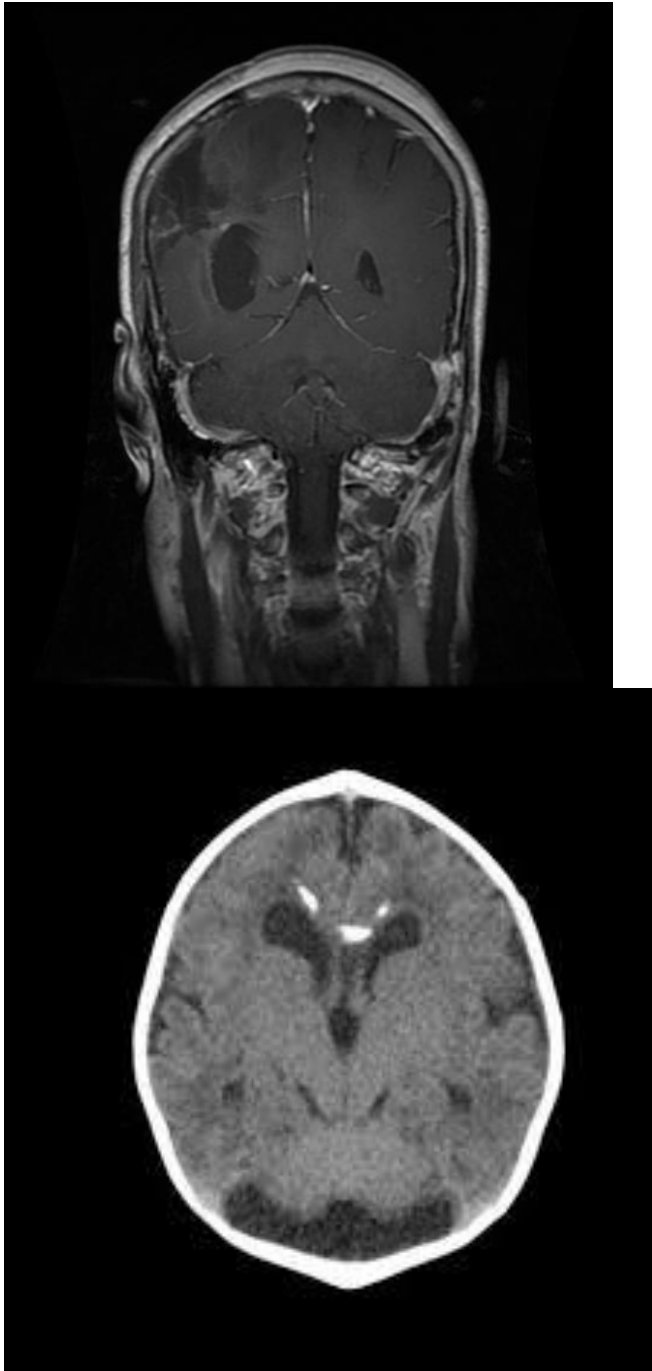


Figure 3.2: A sample for each class from our Training Set. Top Left: Meningioma; Top Right: Pituitary; Bottom Left: Glioma; Bottom Right: No Tumor (Healthy Brain)

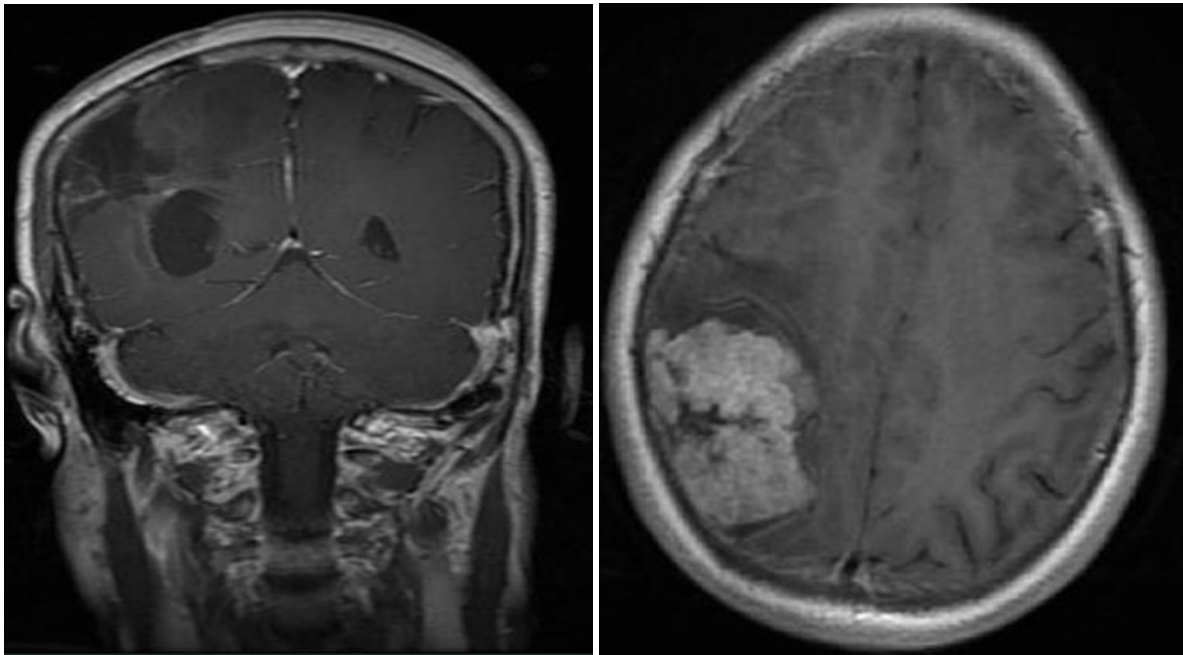
3.3.2 Dataset Preprocessing

Preprocessing input data is a step of paramount importance in the development of ML models, as it is likely to significantly affect the accuracy and reliability of the results. This is especially true when working with image datasets, where preprocessing can involve a range of techniques such as resizing, normalization, and cropping. In the case of brain MRI images, it is important to remove any extraneous black borders or margins around the actual image before feeding it to the model. These margins are often present in medical images to provide context and help distinguish different parts of the scan, but they can interfere with the model's ability to accurately detect the relevant features within the image.

Artificial Intelligence algorithms in the digital processing of biomedical images

To address this issue, a Python script was made to automatically crop out the black margins from the MRI images. The script used various image processing techniques, like edge detection and morphological operations, to identify and remove the black regions around the image. The result was a set of cropped images with the actual brain scan occupying the majority of the image space. By removing the black margins, the model was better able to put emphasis on the important information within the image and improve its accuracy in detecting abnormalities or other features of interest within the brain MRI scans.

Preprocessing of this nature can be time-consuming and complex, but it is a necessary stage in the creation of efficient machine learning models. By taking the time to preprocess the data in this way, the resulting model was better able to generalize to new, unseen images and achieve higher accuracy on the task at hand. Additionally, the ability to automatically crop the images using a Python script allowed for a more efficient and consistent preprocessing pipeline, saving time and effort in the model development process. Overall, preprocessing of image data is a crucial consideration in the development of machine learning models, and techniques such as cropping can be especially important when dealing with medical images and other types of datasets with extraneous information.



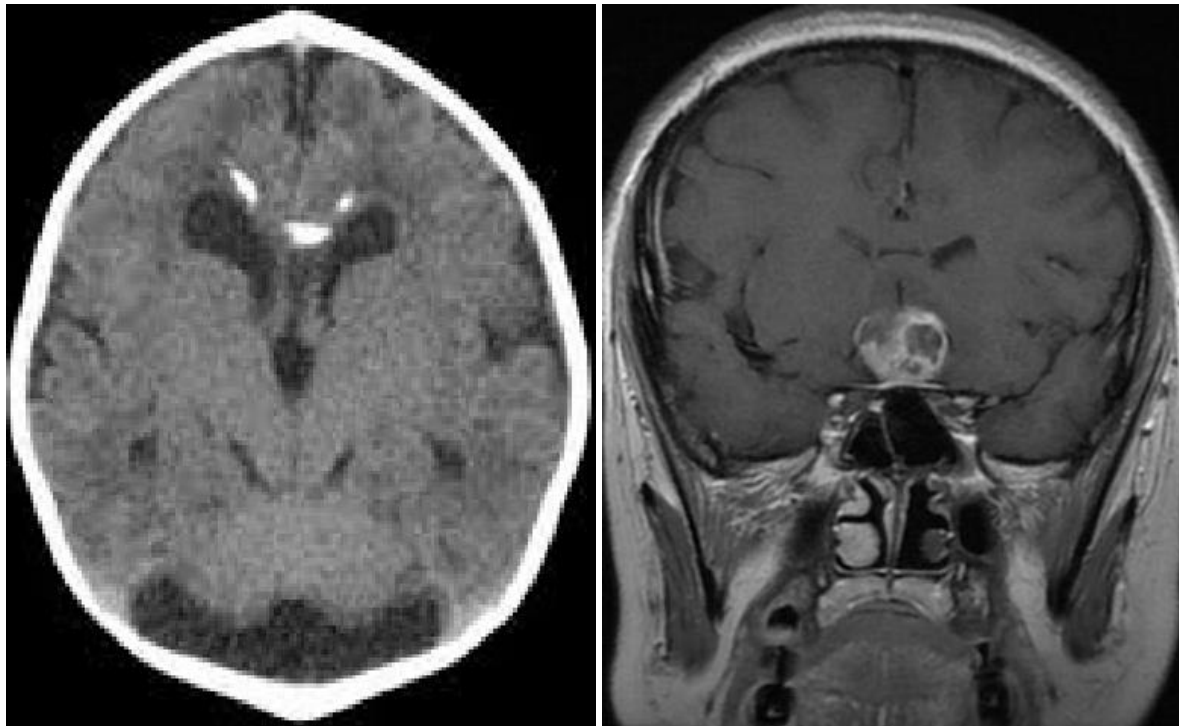


Figure 3.3: The same sample as seen in Fig 3.2, now cropped

3.3.3 Data augmentation on the Dataset

In addition to cropping the black margin of each image, we will also perform other preprocessing steps to further enhance the quality of the dataset. These steps include normalization and standardization, which are commonly used techniques in image preprocessing. Normalization ensures that the pixel values in the image are adjusted to a common range, typically between 0 and 1, to strengthen the dataset's consistency. Standardization involves leveling the pixel values to a zero mean and unit variance, which can help the model to converge more quickly during training.

Furthermore, we will apply data augmentation techniques to raise the dataset's quantity and prevent overfitting. Data augmentation refers to a common technique in the field of deep learning that involves creating new training cases by transforming existing images in various ways, such as flipping, shifting and rotating. By applying data augmentation, we can increase the variety of the images in the training set and reduce the risk of the model memorizing the training set.

We will also carefully tune the hyperparameters of the models to optimize their performance. Variables that are not learned throughout training are referred to as hyperparameters, and they are set by the developer before the training procedure begins. For instance, hyperparameters may include the learning rate, batch size, and number of epochs. These hyperparameters may have a significant effect on the model's efficiency, and therefore different settings will be tried to identify the optimal combination for our specific classification task.

Overall, the preprocessing and data augmentation steps, along with the careful hyperparameter tuning, are crucial for developing accurate and efficient deep learning models for biomedical image analysis. By providing transparent and reproducible methods in this ΠΑΔΑ, Τμήμα Η&ΗΜ, Διπλωματική Εργασία, Αλεξόπουλος Νικόλαος

study, we hope to contribute to the advancement of the field and ultimately improve the diagnosis and treatment of brain tumors.

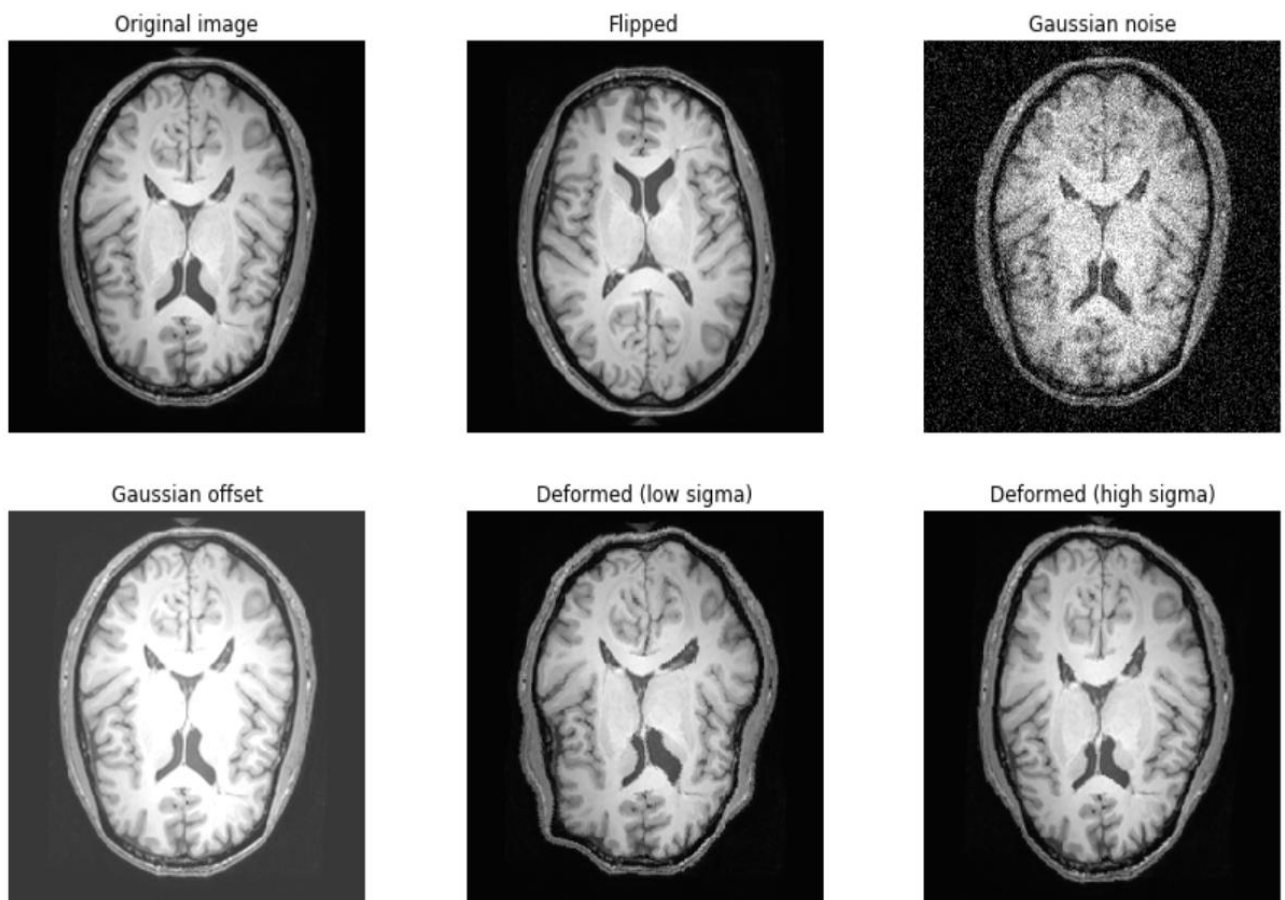


Figure 3.4: Example1 of data augmentation performed in Brain MRI Scans [46].

In addition to the differences in tumor type and location, there are also variations in the appearance of brain tumors in MRI images. For example, meningiomas often appear as well-circumscribed, spherical or lobulated masses, with clear boundaries that separate them from the surrounding healthy tissue. They may also have areas of calcification or cystic changes. In contrast, gliomas may have less distinct borders, with infiltrative growth patterns that make it difficult to separate the tumor from the surrounding healthy tissue. They may also exhibit varying degrees of contrast enhancement, which can indicate increased vascularity or changes in the blood-brain barrier.

Pituitary tumors may appear as focal masses within the pituitary gland, and can cause compression of adjacent structures or hormonal imbalances that lead to a range of symptoms.

Understanding these differences in appearance is essential for accurate brain tumor categorization. By taking advantage of deep learning models, we can train the models to identify these unique features and distinguish between the different types of tumors and healthy brain tissue. This can ultimately improve the efficiency and accuracy of brain tumor diagnosis, leading to better patient outcomes

3.3.4 Helper Functions

Along with the benefits of saving time and resources, using a type of helper functions can also help ensure reproducibility of our results. By saving the model and related data, we can easily share our work with others, who can then reproduce the results by loading the saved model and data and continuing with the experimentation. This is particularly important in medical imaging field, where reproducibility and transparency are essential for building trust and advancing the field.

Furthermore, by saving the loss and accuracy graphs, we can visually analyze the accuracy of our models and juxtapose them to each other. This can help us identify trends, such as which models are overfitting or underfitting the data, and make informed decisions on which models to further explore or discard. This can also help us adjust the hyperparameters of the models, such as the learning rate, batch size, and optimizer, to improve their performance.

In summary, by implementing these helpful functions, we can save time and resources, ensure reproducibility of our results, and analyze the performance of our models to make informed decisions and improve our workflow. This can ultimately lead to more accurate and efficient classification of brain tumors from MRI images, with potential implications for patient diagnosis and treatment.

3.3.5 Setting up the Datasets for Training and Validation

Apart from to the data augmentation techniques mentioned, we can use other methods to increase the diversity of our training set. These can include adjusting the brightness, contrast, and saturation of the images, as well as applying random cropping or padding. By applying these techniques, we can create variations of our original images that our model may not have seen before, making it more adaptable to new and different data.

Additionally, we can also use transfer learning to improve the performance of our model. Transfer learning includes using a pre-trained model, such as a VGG16 or ResNet model, and fine-tuning it to our specific problem domain. This approach can save time and computational resources, as the pre-trained model has already learned features from a large dataset and can be used as a starting point for our own model.

Once we have preprocessed our data, we can build and train our model. This typically involves defining the way the model is built, by taking into account things like how many layers we have and selecting an appropriate optimizer and loss function. We can also use techniques such as early stopping and learning rate scheduling to improve the training process and prevent overfitting. By carefully tuning these parameters, we can achieve higher accuracy and better generalization of our model to new data.

Finally, by using the functions for saving our model and loss and accuracy graphs, we can easily and straightforwardly compare and examine the results of our experiments and make informed decisions for future iterations of our model.

Artificial Intelligence algorithms in the digital processing of biomedical images

```
# Training transforms for data augmentation
def get_train_transform(IMAGE_SIZE):
    train_transform = transforms.Compose([
        transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(10),
        transforms.RandomResizedCrop(size=IMAGE_SIZE, scale=(0.08, 1.0), ratio=(0.75, 1.33), interpolation=2),
        transforms.RandomAdjustSharpness(sharpness_factor=2, p=0.5),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=0.1),
        transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
    return train_transform
```

Figure 3.5: A portion of our code regarding the Data Augmentation transforms we used

Below is a list of all the augmentation techniques we are going to use for our training transforms:

- RandomVerticalFlip: The image is being flipped vertically in an arbitrary way.
- RandomHorizontalFlip: The image is being flipped horizontally in an arbitrary way.
- RandomAffine: The image is being randomly transformed in an affine manner while maintaining its center position.
- RandomAdjustSharpness: The image is being sharpened in an arbitrary way.
- RandomResizedCrop: Cropping a random portion of image and resize it to a given size.
- RandomAdjustSharpness: Adjusts the sharpness of the image with a given probability.
- GaussianBlur: Blurs the image.

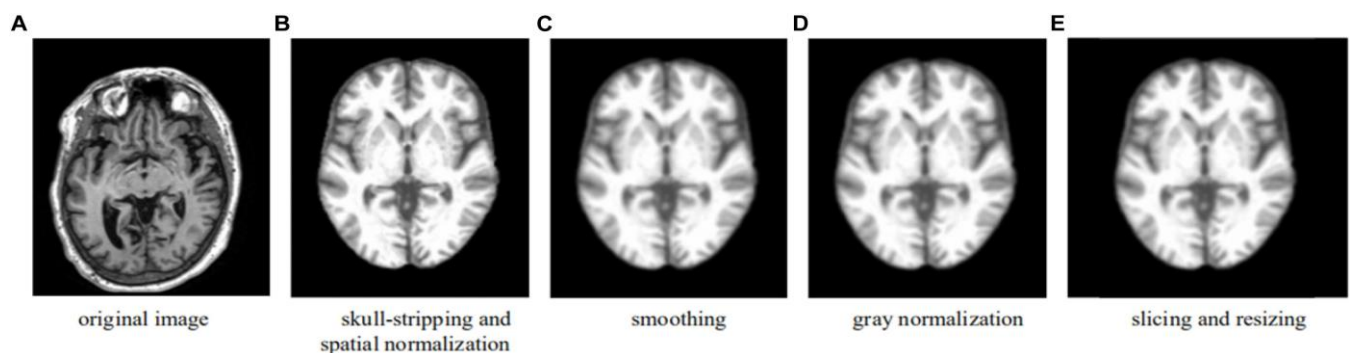


Figure 3.6: Example #2 of data augmentation in a brain MRI scan [47].

When working with medical images, it's critical to be cautious when applying any form of image augmentation. This is because even minor changes to the appearance of the image can significantly impact the ability of the model to learn and proceed with generalisation of

unseen data. Thus, in the case of MRI images, we must be particularly careful not to introduce any modifications that could distort the underlying information present in the images.

In this regard, color and contrast augmentation are generally avoided for medical images since they can cause significant changes in the appearance of the images. However, techniques like random horizontal and vertical flipping, and random rotation, can be applied since they preserve the underlying information while adding diversity to the training set. These techniques create new versions of the original images, which help to ensure that the model is trained on a diverse set of images which covers a broad range of possible cases.

In addition to augmentation, normalization is a critical step in the preprocessing of the data. The ImageNet normalization values are typically used when working with pre-trained models like EfficientNetB0 and ResNet18, as these models were trained on the ImageNet dataset, and normalizing our images to match the same values helps the model to learn from our data more effectively. By standardizing our data in this way, we ensure that the model can focus on the underlying patterns and structures of the images rather than any differences in intensity or color between them.

Finally, shuffling the images during the data loading process is another essential step in ensuring the model's training is optimized. Randomizing the order of the images in each batch prevents the model from overemphasizing the first or last images in the dataset. By doing so, we ensure that each batch receives a representative selection of all types of images present in the dataset, resulting in a more diverse and robust training set.

```
train_loader = DataLoader(  
    dataset_train, batch_size=BATCH_SIZE,  
    shuffle=True, num_workers=NUM_WORKERS  
)  
valid_loader = DataLoader(  
    dataset_valid, batch_size=BATCH_SIZE,  
    shuffle=True, num_workers=NUM_WORKERS  
)  
return train_loader, valid_loader
```

Figure 3.7: A portion of our code showing the shuffling procedure

3.4 How does EfficientNetB0 work

EfficientNetB0 is a deep convolutional neural network architecture designed to achieve state-of-the-art performance on various computer vision tasks. The architecture is based on a compound scaling method that optimizes the size of the network for a given task, by scaling the network depth, width, and resolution in a balanced way. This approach allows EfficientNetB0 to achieve high accuracy on tasks like image classification, object detection, and segmentation, while requiring fewer computational resources than other leading architectures.

The EfficientNetB0 architecture consists of a series of convolutional layers, which extract features from the input image, followed by a set of fully connected layers, which classify the

image based on the extracted features. The compound scaling method used by EfficientNetB0 involves scaling the number of layers, the width of the layers, and the image resolution in a coordinated way. Specifically, the network depth is increased by adding more layers, the layer width is increased by increasing the number of channels in each layer, and the input image resolution is increased to capture more fine-grained details in the image.

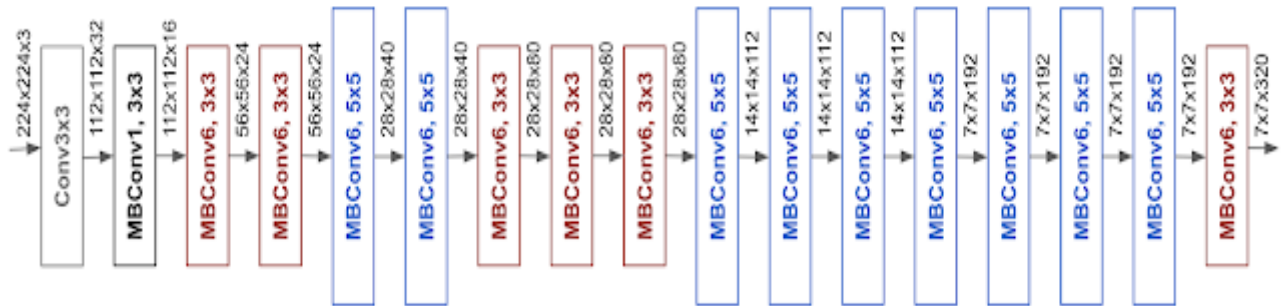


Figure 3.8: The architecture of EfficientNetb0 [48]

EfficientNetB0 also includes a set of advanced techniques to further optimize its performance, including the use of swish activation functions, which have been shown to outperform traditional ReLU activations, and the use of a dynamic scaling method, which allows the network to adjust the layer width based on the available computational resources. These techniques, combined with the compound scaling method, enable EfficientNetB0 to achieve state-of-the-art results on various computer vision tasks, with a relatively small number of parameters and computational resources.

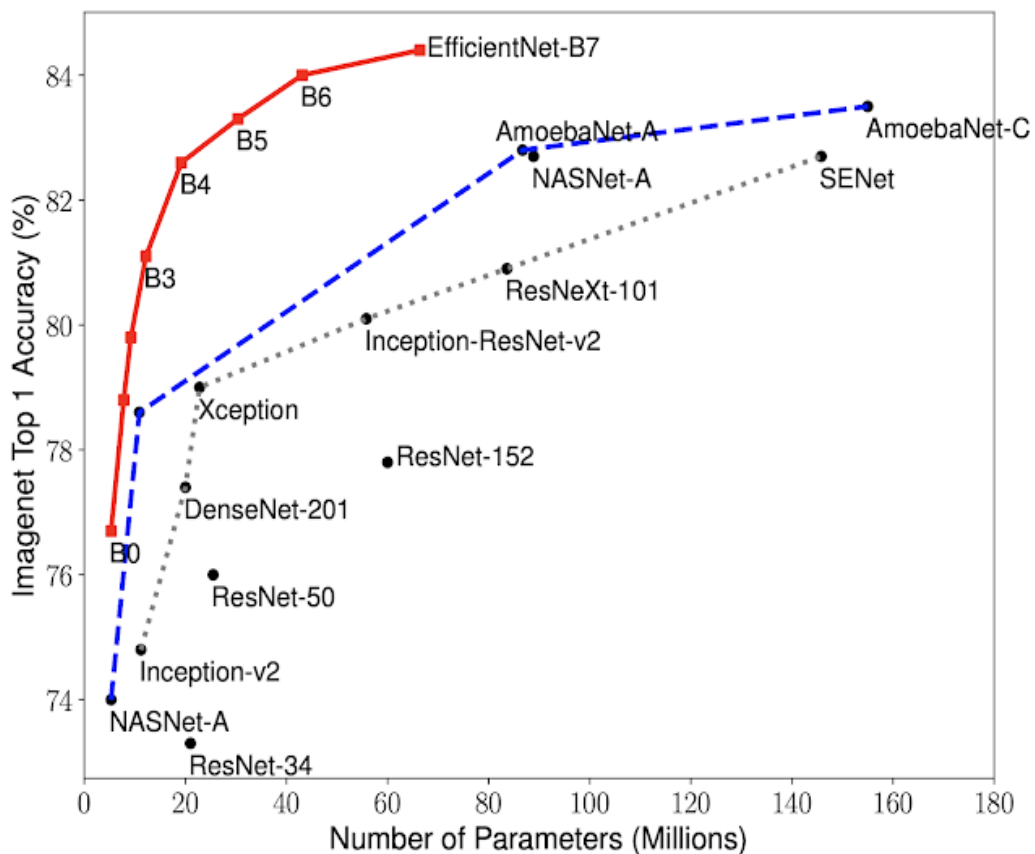


Figure 3.9: Model Size versus Accuracy [48]

Overall, EfficientNetB0 is a highly optimized deep neural network architecture that achieves state-of-the-art performance on a variety of computer vision tasks. Its compound scaling method, advanced techniques, and efficient use of computational resources make it an attractive choice for researchers and practitioners working in the field of computer vision.

3.4.1 How to implement EfficientNetB0 Transfer Learning

By the term transfer learning one refers to a technique used in machine learning that allows knowledge obtained from training one model to be transferred to a new model for a similar task. EfficientNetB0 is a deep neural network architecture that has achieved state-of-the-art results on numerous computer vision tasks, such as, among others, image classification, object detection, and segmentation. It is designed to be highly scalable, with a compound scaling method that optimizes the size of the network for a given task.

Transfer Learning: Base Model and the New Model

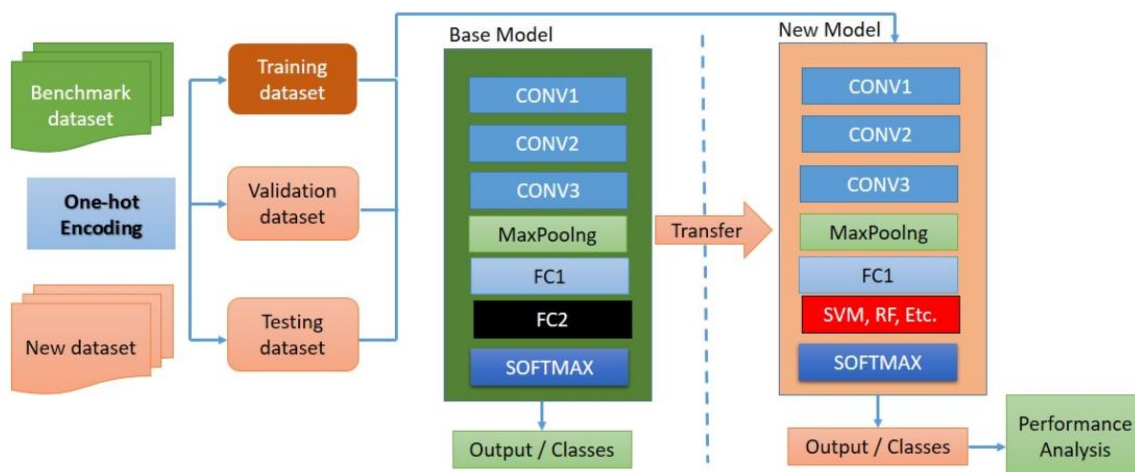


Figure 3.10: Visual representation of Transfer Learning [49]

To apply transfer learning with EfficientNetB0, one typically starts with a pre-trained model which has been trained on a large dataset, such as the popular ImageNet dataset, and fine-tunes it for a specific task. Fine-tuning involves taking the pre-trained model and modifying the last few layers to suit the new task, while retaining the weights learned from the pre-training stage. The number of layers to be modified and the quantity of new layers to be added will depend on the specific problem being tackled.

In the case of brain MRI classification, transfer learning with EfficientNetB0 could be used by taking a pre-trained EfficientNetB0 model and fine-tuning it for the specific classification task. The model could be modified by changing the last few layers of the network with new layers that are tailored to the specific classification problem. The new layers could include fully connected layers or convolutional layers that depend on the nature of each problem. Once the model has been fine-tuned, it can be trained on the brain MRI dataset, with the weights from the pre-trained model providing a good start for the optimization process.

Using transfer learning with EfficientNetB0 in this way has several advantages, including reduced training time, improved accuracy, and the ability to work with smaller datasets. The

pre-trained model has already learned to detect features and patterns in images, so the fine-tuning process can be faster and more efficient than training a model from scratch. Additionally, the transfer learning approach can help to mitigate the risk of overfitting, since the pre-trained model has already been exposed to a large amount of data. Overall, transfer learning with EfficientNetB0 is a powerful technique for enhancing the accuracy and efficiency of machine learning models, particularly in image classification tasks like brain MRI classification.

3.5 Building the EfficientNetB0 Model

The first step in building a deep learning model for image classification is to import the necessary libraries and define the constants. We have decided to resize the images to a common size of 224x224 pixels and set a batch size of 32. We then set the paths for the training and validation datasets. Since our goal is to increase the size of our training dataset, we use data augmentation techniques like random horizontal flipping, random vertical flipping, and random rotation.

These techniques enable us to create new training samples that are similar to our original data, but with variations. To ensure that our data is preprocessed in the same way, we will then create the functions for both our training and validation transformations that will be applied to both the training and validation datasets. Standardizing the data in this manner helps to prevent any discrepancies between the training and validation datasets, leading to more accurate and efficient model training.

The next step is to proceed with the creation of a function to build a deep learning model for image classification. We use a pre-trained EfficientNet-B0 model as the base model for the classification task. The function can fine-tune all the layers of the model or freeze the hidden layers based on the user's choice. The number of classes in the classification problem is passed as an argument to the function. The final layer of the model is modified so as to possess the same number of output classes as the number of classes in the problem, and a dropout layer is added to avoid overfitting. The modified model is returned by the function.

3.6 Training the EfficientNetB0 Model

We also define two functions, train and validate, that perform training and validation procedure respectively. The "train" function takes in the CNN, it loads our data, and used a loss function and an optimizer. It runs a loop that processes each batch of data from the data loader. For each batch, the neural network is fed forward to produce a predicted output, and the loss is calculated using the chosen loss function. The accuracy of the predictions is also computed. After this, a backward pass is performed to update the neural network weights. The "validate" function is similar to this process, but it does not update the weights with a backward pass.

After loading the Brain MRI training and validation datasets, we initialize the neural network using the "build_model" function, set the learning rate, maximum number of epochs, and patience for early stopping, and initialize the optimizer and loss function. We loop over each epoch and perform training and validation using the train and validate functions, respectively.

Artificial Intelligence algorithms in the digital processing of biomedical images

After processing each epoch, we keep track of the loss and accuracy values for both the training and validation datasets. We save the model with the lowest validation loss achieved during training. Additionally, we implement early stopping by halting the training process if the validation loss does not improve after a specified number of epochs. Finally, we generate plots of the loss and accuracy values and save them. At this point, the training process is considered complete.

```
[INFO]: Epoch 50 of 50
Training
 0%|          | 0/166 [00:00<?, ?it/s]
Validation
 0%|          | 0/41 [00:00<?, ?it/s]
Training loss: 0.092, training acc: 96.647
Validation loss: 0.035, validation acc: 98.856
-----
TRAINING COMPLETE
```

Figure 3.11: Message on our Python interface where it can be seen that the Training and Validation Accuracy after the Training is complete

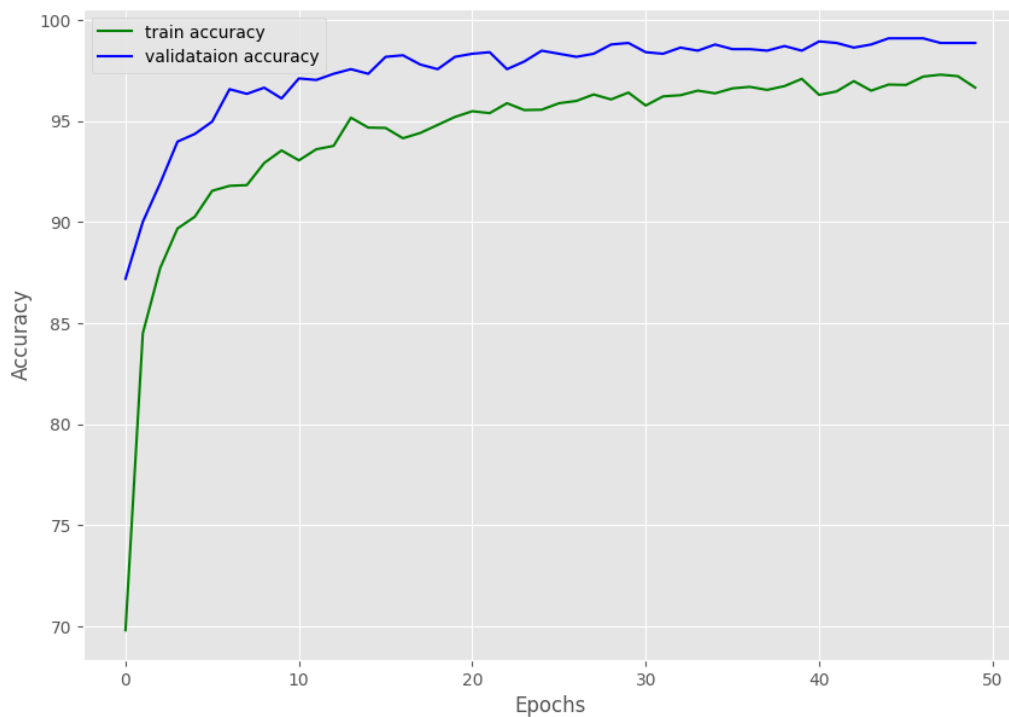


Figure 3.12: Plots showing the accuracy for the Training and Validation trajectory of the EfficientNetB0

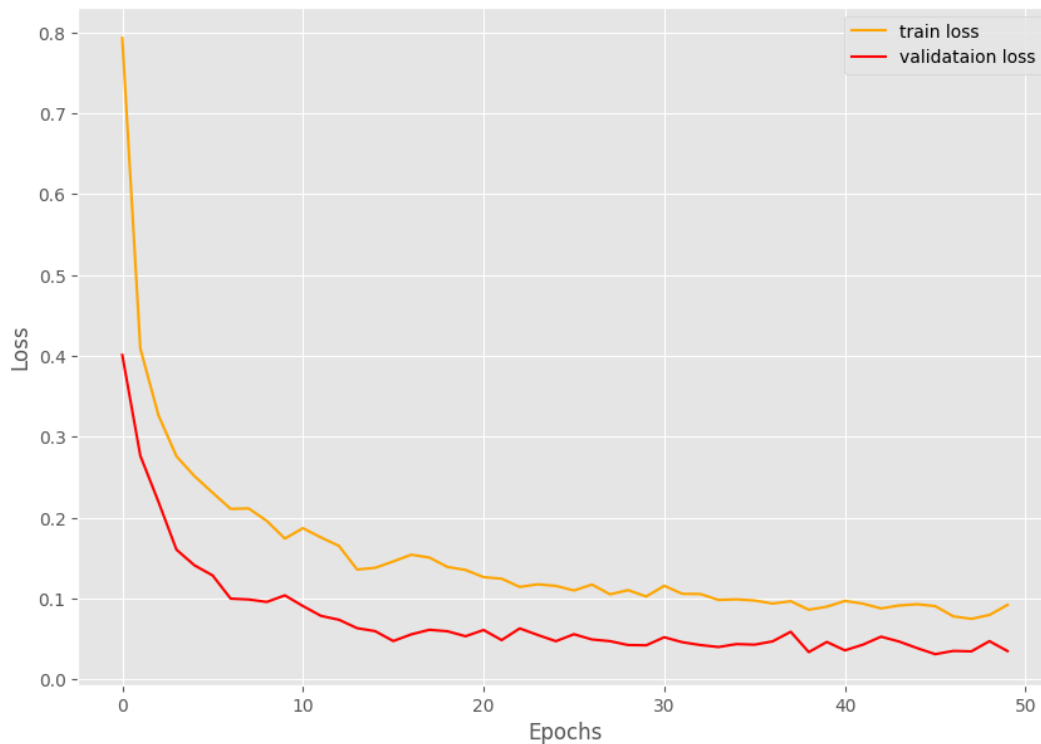


Figure 3.13: Plots showing the Training and Validation Loss of the EfficientNetB0

3.7 Inference on the EfficientNetB0 Model

In the last step of our deep learning image classification task, we perform inference on a set of test images and evaluate the accuracy of the model. This process involves setting up the necessary parameters, loading the pre-trained model, performing inference on the test data, and displaying a confusion matrix to analyze the results.

First, we import the required libraries for the task, including PyTorch, numpy, pandas, and scikit-learn. We also set up the device to be used for the task, which will be the GPU if available, or the CPU otherwise.

Next, we set up the parameters for the test images, including the directory where the images are stored, the size of the images, and the batch size for processing the images. We then create a PyTorch DataLoader to load the test images and their corresponding labels into memory for processing.

The next section of the code loads the pre-trained model and its weights from a saved checkpoint file. It then defines a function to perform the inference using the loaded model and the test data from the DataLoader. The function iterates over the new unseen data, and they go through the model to obtain predictions for each image, and compares the predicted labels with the actual labels. The function calculates how accurate the model on the test data is, and also stores the actual and predicted classes for each image.

Finally, the last section of the code uses a scikit-learn function to create a confusion matrix from the actual and predicted classes. This shows the number of accurate predictions for positives, accurate predictions for negatives, inaccurate predictions positives, and inaccurate predictions negatives for each class in the classification problem. We can use this matrix to

evaluate the performance of our model and identify any patterns in misclassifications. The confusion matrix is displayed using seaborn and matplotlib functions. Finally, the code calculates and prints the overall accuracy of the model. A confusion matrix is a tool used in machine learning and statistics to evaluate the performance of a classification model. It is a matrix that summarizes the performance of a classification model by comparing its predictions with the actual labels of a dataset.

A confusion matrix is typically a square matrix with rows and columns representing the actual and predicted classes, respectively. Each entry in the matrix represents the number of samples that were classified as belonging to a particular class. The main diagonal of the matrix shows the number of correctly classified samples for each class, while the off-diagonal elements show the misclassified samples.

The confusion matrix is a powerful tool for evaluating the performance of a classification model, as it provides information about the model's accuracy, precision, etc. These metrics are commonly used to evaluate the performance of binary and multi-class classification models, and can help identify which classes are being misclassified and why.

By analyzing the confusion matrix, we can gain insights into the strengths and weaknesses of a classification model, and use this information to improve its performance. For example, if the confusion matrix shows that a particular class is consistently misclassified, we can investigate why this is happening and adjust the model or the training data accordingly.

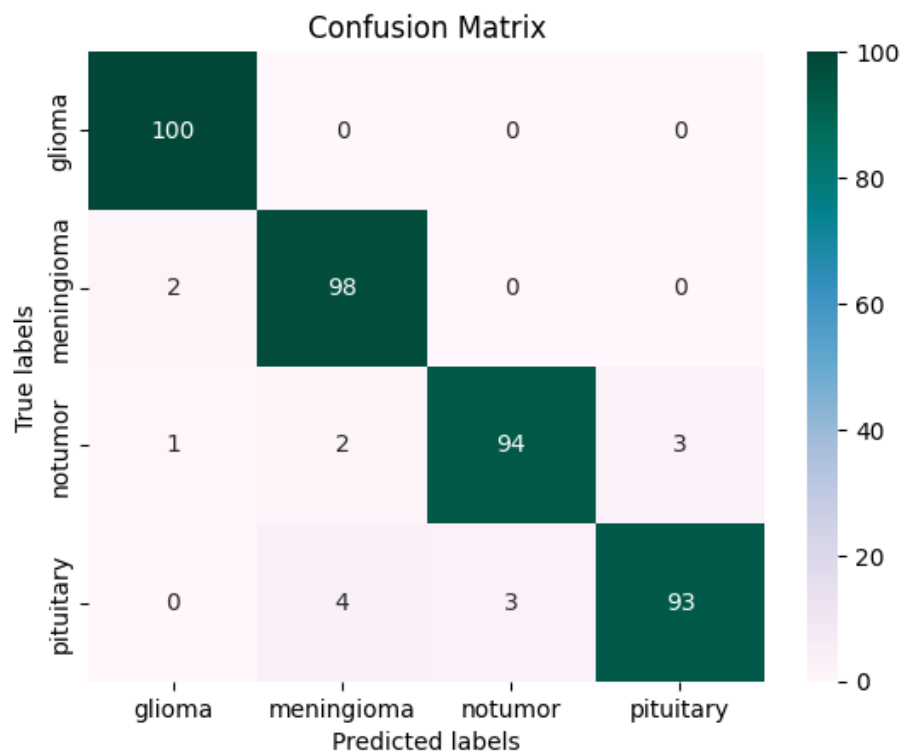
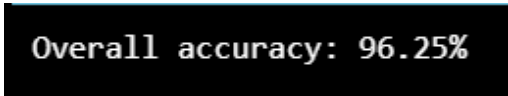


Figure 3.14: Confusion Matrix for the EfficientNetb0

To calculate the overall accuracy from a confusion matrix, you need to sum up the number of correct predictions and divide it by the total number of predictions. The formula for calculating accuracy is:

$$\text{Accuracy} = (\text{True Positive} + \text{True Negative}) / \text{Total}$$

In a confusion matrix, True Positive denotes the number of positive instances that the model has correctly classified, while True Negative represents the number of negative instances that the model has accurately predicted. When it comes to binary classification, the True Positive and True Negative values are placed on the main diagonal of the confusion matrix. For example, if we have two classes, one labeled as "Positive" and the other as "Negative," True Positive will represent the instances where the model correctly classified the "Positive" class, and True Negative will represent the instances where the model correctly classified the "Negative" class. To obtain the accuracy, we add the True Positive and True Negative values and divide by the total number of instances. Our EfficientNetB0 model performed with an accuracy of 96.25%



Overall accuracy: 96.25%

Figure 3.15: Message on our Python interface showing the overall Accuracy of the EfficientNetB0 on unseen data

3.8 Experiment #2: Multi-Class Classification of Brain MRIs Using ResNet18

3.8.1 How ResNet18 Works

ResNet18 is a convolutional neural network architecture that was introduced in 2015 to address the degradation problem in deep neural networks. The degradation problem refers to the fact that as neural networks become deeper, they may become harder to optimize and their accuracy may degrade. ResNet18 tackles this issue by using residual blocks, which are designed to allow the input of a layer to be passed directly to a later layer through a skip connection. This helps to mitigate the vanishing gradient problem, which can occur when the gradients of the loss function become very small as they propagate through the layers of the network.

The ResNet18 is composed of 18 layers and comprises convolutional layers, max pooling layers, batch normalization layers, and fully connected layers. The first layer is a 7x7 convolutional layer with a stride of 2, followed by a max pooling layer with the same stride. The model then employs a sequence of residual blocks, with each block containing two 3x3 convolutional layers and a skip connection that transmits the input of the block directly to its output. The number of filters in each block increases as the network goes deeper, with the first block containing 64 filters and the last block having 512 filters..

After the residual blocks, the output of the final block is passed through a global average pooling layer that reduces the spatial dimensions of the feature maps to a single value per channel. The resulting feature vector is then passed through a fully connected layer to produce the final output of the network.

ResNet18 is a relatively shallow network compared to other deep learning architectures, such as ResNet50 or EfficientNetB0, but it has been shown to achieve excellent results on various benchmark datasets, including ImageNet.

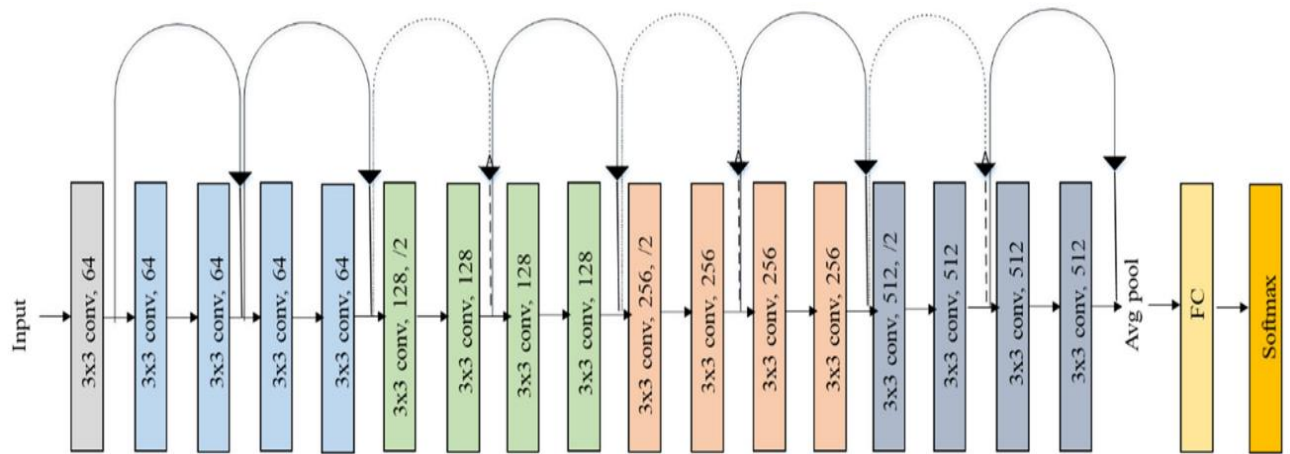


Figure 3.16 Original ResNet18 Architecture [49]

3.8.2 How to implement Transfer Learning with ResNet18

The concept of transfer learning as we saw it on EfficientNetB0 can also be applied to ResNet18. ResNet18 is a deep neural network architecture that has also been pre-trained on large datasets such as ImageNet, and its weights can be viewed as the start for a new classification task.

In transfer learning with ResNet18, the pre-trained model is usually fine-tuned by replacing the last layer or a few layers with new layers that are tailored to the specific classification problem.

The fine-tuning process includes updating the weights of the pre-trained layers and the newly added layers using a smaller dataset specific to the task at hand. As with EfficientNetB0, transfer learning with ResNet18 can reduce training time, improve accuracy, and mitigate the risk of overfitting.

3.9 Building the ResNet18 Model

To apply the ResNet18 model to our image classification task, we begin by defining the necessary constants and importing the required libraries. We then resize the images to a uniform size of 224x224 pixels and set the batch size to 32, similar to the EfficientNetB0 implementation. Augmentation techniques such as random flipping and rotation are used to increase the size of the training dataset, and standardization is applied to eliminate any inconsistencies between the training and validation datasets.

For the ResNet18 model, we use a pre-trained model as the base and fine-tune or freeze layers as per our preference. The last layer of the model is adjusted to output the same number of classes as the problem, in our case, four classes, with a dropout layer added to prevent overfitting. This entire process of modifying the model is performed on both the training and validation datasets to maintain consistency in the preprocessing.

3.10 Training the ResNet18 Model

Again, as we already did during the training of EfficientNetB0, we are going to use two functions, "train" and "validate," which are used to perform training and validation on our ResNet18 model. We will also retain the same process of initializing the neural network, setting the learning rate, maximum number of epochs, and patience for early stopping, and initializing the optimizer and loss function. Then after the training and validation processes are performed for each epoch, the model with the lowest validation loss is saved. Early stopping is also performed in the event that the validation loss does not improve after a set number of epochs. Finally, the loss and accuracy plots are printed and saved.

```
Training loss: 0.127, training acc: 95.328  
Validation loss: 0.044, validation acc: 98.398  
-----  
TRAINING COMPLETE
```

Figure 3.17: Message on our Python interface presenting the Training and Validation Accuracy after the Training is complete

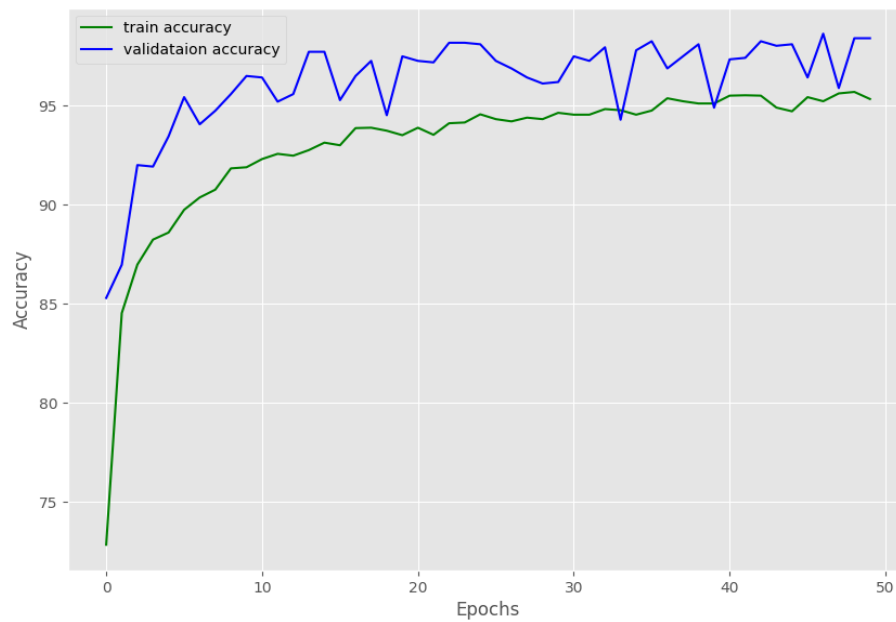


Figure 3.18: Plots that show the Training and Validation Accuracy of the ResNet18

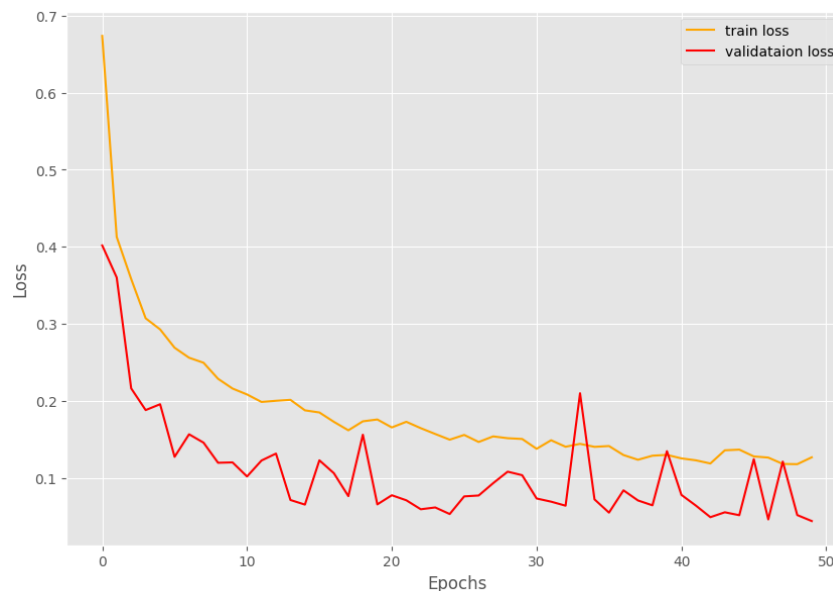


Figure 3.19: Plots that show the Training and Validation Loss of the ResNet18

3.11 Inference on the ResNet18 Model

Following the same logic from the previous experiment, the process of evaluating the accuracy of the ResNet18 model involves setting up the necessary parameters, loading the model, performing inference on the test data, and displaying a confusion matrix to analyze the results.

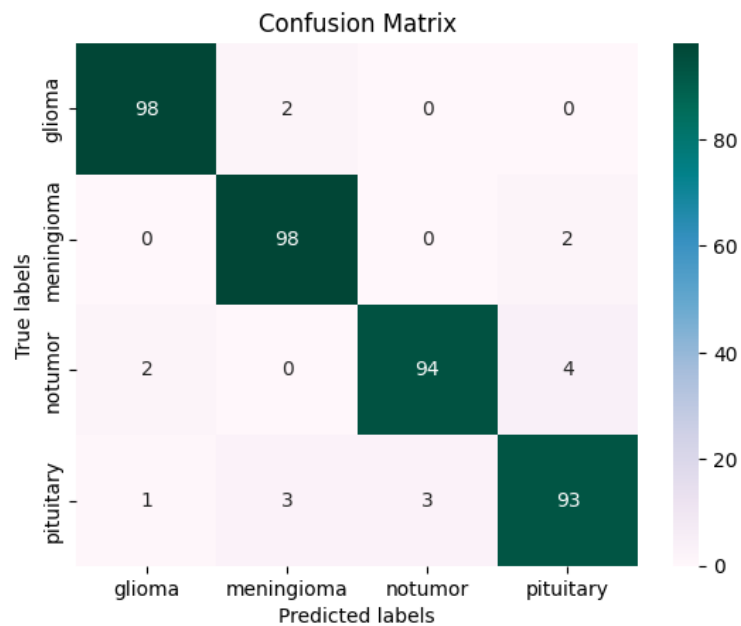


Figure 3.20: Confusion Matrix for the ResNet18

For the purpose of calculating the overall accuracy, we add up the True Positive and True Negative values and divide by the total number of instances. In this case, the ResNet18 performed at 95.75% accuracy, slightly lower from EfficientNetb0.

Overall accuracy: 95.75%

Figure 3.21: Message on our Python interface showing the overall Accuracy of the ResNet18 on unseen data

3.12 Discussion over the results in Accuracy

The results of the classification task using the two CNNs, EfficientNetB0 and ResNet18, on MRI brain scans have demonstrated high accuracy rates. The EfficientNetB0 model achieved a classification accuracy of 96.25%, whereas the ResNet18 model achieved an accuracy of 95.75%. This indicates that both models are quite satisfactory in accurately classifying biomedical images.

Comparing the two models, it is evident that EfficientNetB0 outperforms ResNet18, as it achieves a slightly higher accuracy rate. The superior performance of EfficientNetB0 may be attributed to its efficient architecture and novel compound scaling method, which enables it to achieve state-of-the-art performance with fewer parameters compared to other CNN architectures.

While the difference of 0.5% may seem small, it can have significant implications in the context of medical diagnosis, where even a small increase in accuracy can have a significant impact on patient outcomes. In particular, the early detection of brain tumors can be critical for timely treatment and better patient outcomes. In addition to high accuracy rates, the models showed high precision and recall scores, indicating a limited number of false positives and false negatives.

Artificial Intelligence algorithms in the digital processing of biomedical images

Overall, these results suggest that the use of CNNs for biomedical image classification is a promising approach with high accuracy rates, and can potentially aid in clinical decision-making processes. However, further research is necessary to fully explore the potential of CNNs in this area and to address any limitations or challenges that may arise.

4. Conclusions

Over the past years, CNNs have emerged as powerful tools for image classification, and their applications in biomedical image analysis have shown promising results. In this study, we focused on the classification of MRI brain scans using two popular CNN architectures, EfficientNetB0 and ResNet18, and achieved high accuracy rates of over 95% for both models.

It is important to note that while both models performed well, there were differences in their performance. For instance, EfficientNetB0 had a slightly higher accuracy rate and required less time for training than ResNet18. This highlights the importance of considering multiple model architectures and selecting the most appropriate one for the task at hand.

Additionally, we found that data augmentation techniques like random rotation, horizontal flip, and vertical flip were essential for improving the generalization of the models and reducing overfitting. This is consistent with previous studies that have shown the benefits of data augmentation for improving the performance of CNNs on small datasets.

Our results suggest that CNNs are a viable option for the classification of biomedical images, and further research in this area could lead to even more accurate and efficient models. However, it is important to emphasize the interpretation of the results and the clinical applicability of these models should be performed with caution, as the final diagnosis should always be made by a medical professional.

Overall, this thesis highlights the potential of using deep learning techniques in the analysis of biomedical images and provides a promising avenue for future research in the field. With the availability of larger and more diverse datasets, more sophisticated CNN architectures, and the ongoing development of explainable AI techniques, the use of CNNs in medical imaging could affect significantly the diagnostic accuracy, reducing human error, and improving patient outcomes.

Βιβλιογραφία – Αναφορές - Διαδικτυακές Πηγές

Biomedical Imaging Sources

- [1] Razzak, M.I., Naz, S., and Zaib, A. "Deep Learning for Medical Image Processing: Overview, Challenges and the Future." In Classification in BioApps, edited by N. Dey, A. Ashour, and S. Borra, Lecture Notes in Computational Vision and Biomechanics, vol. 26. Springer, Cham, 2018. https://doi.org/10.1007/978-3-319-65981-7_12
- [2] Lehmann, T.M., Meinzer, H.P., and Tolxdorff, T. "Advances in Biomedical Image Analysis-- Past, Present and Future Challenges." Methods Inf Med 43, no. 4 (2004): 308-314. <https://pubmed.ncbi.nlm.nih.gov/15472739/>
- [3] Dhawan, Atam P. "A Review on Biomedical Image Processing and Future Trends." Computer Methods and Programs in Biomedicine 31, no. 3-4 (1990): 141-183. <https://pubmed.ncbi.nlm.nih.gov/2194742/>
- [4] Rangayyan, R.M. Biomedical Image Analysis. 1st ed. CRC Press, 2004. <https://doi.org/10.1201/9780203492543>
- [5] Kurnar, M., Sinha, A., and Bansode, N.V. "Detection of Brain Tumor in MRI Images by Applying Segmentation and Area Calculation Method Using SCILAB." In 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), pp. 1-5. Pune, India, 2018. <https://doi.org/10.1109/ICCUBEA.2018.8697713>
- [6] "What is an MRI Scan and What Can It Do?" Drug Ther Bull 49, no. 12 (2011): 141-144. <https://doi.org/10.1136/dtb.2011.02.0073>
- [7] Sternberg, S.R. "Biomedical Image Processing." Computer 16, no. 1 (1983): 22-34. <https://doi.org/10.1109/MC.1983.1654163>
- [8] Preston, K. "Computer Processing of Biomedical Images." Computer 9, no. 5 (1976): 54-68. <https://doi.org/10.1109/C-M.1976.218587>

Tumors Citations

- [9] Marosi, C., Hassler, M., Roessler, K., et al. "Meningioma." Crit Rev Oncol Hematol 67, no. 2 (2008): 153-171. doi:10.1016/j.critrevonc.2008.01.010.
- [10] Maggio, I., Franceschi, E., Tosoni, A., et al. "Meningioma: Not Always a Benign Tumor. A Review of Advances in the Treatment of Meningiomas." CNS Oncol 10, no. 2 (2021): CNS72. doi:10.2217/cns-2021-0003.
- [11] Chen, R., Smith-Cohn, M., Cohen, AL., Colman, H. "Glioma Subclassifications and Their Clinical Significance." Neurotherapeutics 14, no. 2 (2017): 284-297. doi:10.1007/s13311-017-0519-x.
- [12] Ostrom, QT., Bauchet, L., Davis, FG., et al. "The Epidemiology of Glioma in Adults: A "State of the Science" Review." Neuro Oncol 16, no. 7 (2014): 896-913. doi:10.1093/neuonc/nou087.

- [13] Gusyatiner, O., Hegi, ME. "Glioma Epigenetics: From Subclassification to Novel Treatment Options." *Semin Cancer Biol* 51 (2018): 50-58. doi:10.1016/j.semcancer.2017.11.010.
- [14] Melmed, S. "Pituitary-Tumor Endocrinopathies." *N Engl J Med* 382, no. 10 (2020): 937-950. doi:10.1056/NEJMra1810772.
- [15] Trifiletti, DM., Dutta, SW., Lee, CC., Sheehan, JP. "Pituitary Tumor Radiosurgery." *Prog Neurol Surg* 34 (2019): 149-158. doi:10.1159/000493059.

Gradient Descent

- [16] Ruder, S. "An Overview of Gradient Descent Optimization Algorithms." arXiv:1609.04747.
- [17] Amari, SI. "Backpropagation and Stochastic Gradient Descent Method." *Neurocomputing* 5, no. 4-5 (1993): 185-196. doi:10.1016/0925-2312(93)90006-O.

Linear Regression

- [18] Su, X., Yan, X. and Tsai, C.-L. (2012), Linear regression. *WIREs Comp Stat*, 4: 275-294. <https://doi.org/10.1002/wics.1198>
- [19] (2007). Basics of Linear Regression. In: *Process Optimization. International Series in Operations Research & Management Science*, vol 105. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-71435-6_15

Logistic Regression

- [20] Guido, Joseph J., Paul C. Winters, and Adam B. Rains. "Logistic regression basics." MSc University of Rochester Medical Center, Rochester, NY (2006).
- [21] Lund, Bruce. "Logistic Regression, Basics and Beyond."

Supervised Learning

- [22] Cunningham, P., Cord, M., Delany, S.J. "Supervised Learning." In *Machine Learning Techniques for Multimedia*, edited by Cord, M., Cunningham, P., 2-26. Cognitive Technologies. Springer, Berlin, Heidelberg, 2008. https://doi.org/10.1007/978-3-540-75171-7_2
- [23] Hastie, T., Tibshirani, R., Friedman, J. "Overview of Supervised Learning." In *The Elements of Statistical Learning*, 2nd ed., Springer Series in Statistics. Springer, New York, NY, 2009. https://doi.org/10.1007/978-0-387-84858-7_2

Unsupervised Learning

- [24] Dayan, Peter, Maneesh Sahani, and Grégoire Deback. "Unsupervised Learning." In *The MIT Encyclopedia of the Cognitive Sciences*, edited by Robert A. Wilson and Frank C. Keil, 2nd ed., 857-859. Cambridge, MA: The MIT Press, 1999.
- [25] Berg-Kirkpatrick, Taylor, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. "Painless Unsupervised Learning with Features." In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 582-590. 2010.

Machine Learning

- [26] Badillo, S., Banfai, B., Birzele, F., Davydov, I.I., Hutchinson, L., Kam-Thong, T., Siebourg-Polster, J., Steiert, B. and Zhang, J.D. "An Introduction to Machine Learning." *Clin. Pharmacol. Ther.* 107 (2020): 871-885. <https://doi.org/10.1002/cpt.1796>
- [27] Smola, Alex, and S. V. N. Vishwanathan. "Introduction to Machine Learning." Cambridge University, UK 32, no. 34 (2008): 2008.

Neural Networks

- [28] Aggarwal, Charu C. "Neural networks and deep learning." Springer 10, no. 978 (2018): 3. <https://doi.org/10.1007/978-3-319-94463-0>
- [29] Nielsen, Michael A. *Neural networks and deep learning*. Vol. 25. San Francisco, CA, USA: Determination press, 2015.
- [30] Choi, Rene Y., Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. "Introduction to machine learning, neural networks, and deep learning." *Translational Vision Science & Technology* 9, no. 2 (2020): 14-14. <https://doi.org/10.1167/tvst.9.2.14>
- [31] Ivanyuk, V.A. and Pashchenko, F.F. "Neural networks and their application in forecasting problems," *Journal of Physics: Conference Series*, vol. 1703, no. 1, 012033, XXIII International Conference on Soft Computing and Measurement (SCM'2020), Russia, 27-29 May 2020, doi: 10.1088/1742-6596/1703/1/012033..

CNNs

- [32] Li, Z., Liu, F., Yang, W., Peng, S., and Zhou, J. "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects." *IEEE Transactions on Neural Networks and Learning Systems* 33, no. 12 (2022): 6999-7019. <https://doi.org/10.1109/TNNLS.2021.3084827>.
- [33] O'Shea, Keiron, and Ryan Nash. "An Introduction to Convolutional Neural Networks." arXiv preprint arXiv:1511.08458 (2015). <https://doi.org/10.48550/arXiv.1511.08458>
- [34] Marques, Gonçalo, Deepak Agarwal, and Izaskun de la Torre Díez. "Automated medical diagnosis of COVID-19 through EfficientNet convolutional neural network." *Applied Soft Computing* 96 (2020): 106691. <https://doi.org/10.1016/j.asoc.2020.106691>.
- [35] Xin, R., Zhang, J., & Shao, Y. (2020). Complex network classification with convolutional neural network. *Tsinghua Science and Technology*, 25(4), 447-457. doi: 10.26599/TST.2019.9010055.
- [36] Kim, P. (2017). *Convolutional Neural Network*. In *MATLAB Deep Learning*. Berkeley, CA: Apress. doi: 10.1007/978-1-4842-2845-6_6.
- [37] Korolev, S., Safiullin, A., Belyaev, M., & Dodonova, Y. (2017). Residual and plain convolutional neural networks for 3D brain MRI classification. 2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017), 835-838.

Deep Learning

- [38] Litjens G, Kooi T, Bejnordi BE, et al. A survey on deep learning in medical image analysis. *Med Image Anal.* 2017; 42:60-88. doi:10.1016/j.media.2017.07.005
- [39] Bento M, Fantini I, Park J, Rittner L, Frayne R. Deep Learning in Large and Multi-Site Structural Brain MR Imaging Datasets. *Front Neuroinform.* 2022;15:805669. Published 2022 Jan 20. doi:10.3389/fninf.2021.805669
- [40] Kandel, I. H. A. (2021). Deep Learning Techniques for Medical Image Classification. [Doctoral Thesis, NOVA Information Management School (NOVA IMS)]. <http://hdl.handle.net/10362/130159>
- [41] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [42] Harouni, A., Karargyris, A., Negahdar, M., Beymer, D., & Syeda-Mahmood, T. (2018). Universal multi-modal deep network for classification and segmentation of medical images. In 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018) (pp. 872-876). Washington, DC, USA: IEEE. doi: 10.1109/ISBI.2018.8363710.
- [43] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature* 521, no. 7553 (2015): 436-444. doi:10.1038/nature14539.

URL Sources

- [44] A closed MRI scanner system (<https://www.spineuniverse.com/exams-tests/magnetic-resonance-imaging-mri>), last accessed on 23/2/2023
- [45] CDC: Leading Causes of Death (<https://www.cdc.gov/nchs/fastats/leading-causes-of-death.htm>), last accessed on 23/2/2023
- [46] Example of data augmentation in Brain MRI's (<https://blog.tensorflow.org/2018/07/an-introduction-to-biomedical-image-analysis-tensorflow-dltk.html>), last accessed on 23/2/2023
- [47] Data augmentation in MRI's (<https://www.frontiersin.org/articles/10.3389/fnins.2020.00259/full>), last accessed on 23/2/2023
- [48] The Architecture of EfficientNetB0 (<https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html>)
- [49] Ramzan F, Khan MUG, Rehmat A, et al. A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Diseases doi:10.1007/s10916-019-1475-2 (<https://amitray.com/transfer-learning-step-by-step-guide/>)
- [50] Rath, Sovit. "Brain MRI Classification Using PyTorch EfficientNetB0." *Debugger Cafe*. Last modified January 24, 2022. debuggercafe.com/brain-mri-classification-using-pytorch-efficientnetb0/.

[51] "PyTorch." NVIDIA, accessed March 6, 2023, <https://www.nvidia.com/en-us/glossary/data-science/pytorch/>.

APPENDIX 1: Coding for EfficientNetb0

Utils.py (Saves the model and plots)

```
import torch
import matplotlib
import matplotlib.pyplot as plt

matplotlib.style.use('ggplot')

def save_model(epochs, model, optimizer, criterion):
    """
    Function to save the trained model to disk.
    """
    torch.save({
        'epoch': epochs,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'loss': criterion,
    }, f"../outputs/model/model.pth")

def save_plots(train_acc, valid_acc, train_loss, valid_loss):
    """
    Function to save the loss and accuracy plots to disk.
    """
    # accuracy plots
    plt.figure(figsize=(10, 7))
    plt.plot(
        train_acc, color='green', linestyle='-',
        label='train accuracy'
    )
    plt.plot(
        valid_acc, color='blue', linestyle='-',
        label='validataion accuracy'
    )

    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.savefig(f"../outputs/accuracy.png")
```

```
# loss plots
plt.figure(figsize=(10, 7))
plt.plot(
    train_loss, color='orange', linestyle='-',
    label='train loss'
)
plt.plot(
    valid_loss, color='red', linestyle='-',
    label='validataion loss'
)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig(f"../outputs/loss.png")
```

Cropping script to remove margins around the image

```
import numpy as np
from tqdm import tqdm
import cv2
import os
import imutils

os.chdir('C:\\Users\\nick_\\a_thesis\\input\\validation')
os.chdir('C:\\Users\\nick_\\a_thesis\\input\\training')

def crop_img(img):
    """
    Finds the extreme points on the image and crops the rectangular
    """
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (3, 3), 0)

    # threshold the image, then perform a series of erosions +
    # dilations to remove any small regions of noise
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    # find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)

    # find the extreme points
    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])
    ADD_PIXELS = 0
    new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS,
        extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()

    return new_img
```

```
if __name__ == "__main__":
    training = "C:\\Users\\nick_\\a_thesis\\input\\training"
    validation = "C:\\Users\\nick_\\a_thesis\\input\\validation"
    training_dir = os.listdir(training)
    validation_dir = os.listdir(validation)
    IMG_SIZE = 256

    for dir in training_dir:
        save_path = 'cleaned/training/'+ dir
        path = os.path.join(training,dir)
        image_dir = os.listdir(path)
        for img in image_dir:
            image = cv2.imread(os.path.join(path,img))
            new_img = crop_img(image)
            new_img = cv2.resize(new_img,(IMG_SIZE,IMG_SIZE))
            if not os.path.exists(save_path):
                os.makedirs(save_path)
            cv2.imwrite(save_path+'/'+img, new_img)

    for dir in validation_dir:
        save_path = 'cleaned/validation/'+ dir
        path = os.path.join(validation,dir)
        image_dir = os.listdir(path)
        for img in image_dir:
            image = cv2.imread(os.path.join(path,img))
            new_img = crop_img(image)
            new_img = cv2.resize(new_img,(IMG_SIZE,IMG_SIZE))
            if not os.path.exists(save_path):
                os.makedirs(save_path)
            cv2.imwrite(save_path+'/'+img, new_img)
```

Loading Datasets/ Applying Data Augmentation

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Required constants.
TRAIN_DIR = '../input/training'
VALID_DIR = '../input/validation'
IMAGE_SIZE = 224 # Image size of resize when applying transforms.
BATCH_SIZE = 32
NUM_WORKERS = 4 # Number of parallel processes for data preparation.
```

```
# Training transforms for data augmentation
def get_train_transform(IMAGE_SIZE):
    train_transform = transforms.Compose([
        transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.RandomRotation(10),
        transforms.RandomResizedCrop(size=IMAGE_SIZE, scale=(0.08, 1.0),
                                      ratio=(0.75, 1.33), interpolation=2),
        transforms.RandomAdjustSharpness(sharpness_factor=2, p=0.5),
        transforms.RandomAffine(degrees=0, translate=(0.1, 0.1),
                                scale=(0.9, 1.1), shear=0.1),
        transforms.GaussianBlur(kernel_size=(5, 9), sigma=(0.1, 5)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
    return train_transform

# Validation transforms
def get_valid_transform(IMAGE_SIZE):
    valid_transform = transforms.Compose([
        transforms.Resize((IMAGE_SIZE, IMAGE_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]
        )
    ])
    return valid_transform

def get_test_transform(IMAGE_SIZE):
    return get_valid_transform(IMAGE_SIZE)
```

```
def get_datasets():
    """
    Function to prepare the Datasets.
    Returns the training and validation datasets along
    with the class names.
    """
    dataset_train = datasets.ImageFolder(
        TRAIN_DIR,
        transform=(get_train_transform(IMAGE_SIZE))
    )
    dataset_valid = datasets.ImageFolder(
        VALID_DIR,
        transform=(get_valid_transform(IMAGE_SIZE))
    )
    return dataset_train, dataset_valid, dataset_train.classes

def get_data_loaders(dataset_train, dataset_valid):
    """
    Prepares the training and validation data loaders.
    :param dataset_train: The training dataset.
    :param dataset_valid: The validation dataset.
    Returns the training, validation and test data loaders.
    """
    train_loader = DataLoader(
        dataset_train, batch_size=BATCH_SIZE,
        shuffle=True, num_workers=NUM_WORKERS
    )
    valid_loader = DataLoader(
        dataset_valid, batch_size=BATCH_SIZE,
        shuffle=True, num_workers=NUM_WORKERS
    )
    return train_loader, valid_loader
```

Building the EfficientNetB0 Model

```
import torchvision.models as models
import torch.nn as nn
def build_model(pretrained=True, fine_tune=True, num_classes=4):
    if pretrained:
        print('pre-trained model loaded')
    else:
        print('blank model loaded')
    model = models.efficientnet_b0(pretrained=pretrained)
    if fine_tune:
        print('fine-tuning the layers')
        for params in model.parameters():
            params.requires_grad = True
    elif not fine_tune:
        print('Hidden layers status: Frozen')
        for params in model.parameters():
            params.requires_grad = False

    # Replace the last layer (classifier)
    # with a new one that has num_classes outputs
    model.classifier[1] = nn.Sequential(

        nn.Dropout(0.2),
        nn.Linear(in_features=1280, out_features=num_classes)

    )

    return model
```


Training the EfficientNetB0 Model

```

import torch
import argparse
import torch.nn as nn
import torch.optim as optim
import time
import sys
import numpy as np
import os
os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
os.environ['PYTORCH_CUDA_ALLOC_CONF'] = "max_split_size_mb:2048"
torch.cuda.empty_cache()

from tqdm.auto import tqdm

from model import build_model
from datasets import get_datasets, get_data_loaders
from utils import save_model, save_plots

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

# Training function.
def train(model, trainloader, optimizer, criterion):
    model.train()
    print('Training')
    train_running_loss = 0.0
    train_running_correct = 0
    counter = 0
    for i, data in tqdm(enumerate(trainloader), total=len(trainloader)):
        counter += 1
        image, labels = data
        image = image.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        # Forward pass.
        outputs = model(image)
        # Calculate the loss.
        loss = criterion(outputs, labels)
        train_running_loss += loss.item()
        # Calculate the accuracy.
        _, preds = torch.max(outputs.data, 1)
        train_running_correct += (preds == labels).sum().item()
        # Backpropagation
        loss.backward()
        # Update the weights.
        optimizer.step()

    # Loss and accuracy for the complete epoch.
    epoch_loss = train_running_loss / counter
    epoch_acc = 100. * (train_running_correct / len(trainloader.dataset))
    return epoch_loss, epoch_acc

```

Artificial Intelligence algorithms in the digital processing of biomedical images

```
# Validation function.
def validate(model, testloader, criterion):
    model.eval()
    print('Validation')
    valid_running_loss = 0.0
    valid_running_correct = 0
    counter = 0
    with torch.no_grad():
        for i, data in tqdm(enumerate(testloader), total=len(testloader)):
            counter += 1

            image, labels = data
            image = image.to(device)
            labels = labels.to(device)
            # Forward pass.
            outputs = model(image)
            # Calculate the loss.
            loss = criterion(outputs, labels)
            valid_running_loss += loss.item()
            # Calculate the accuracy.
            _, preds = torch.max(outputs.data, 1)
            valid_running_correct += (preds == labels).sum().item()

    # Loss and accuracy for the complete epoch.
    epoch_loss = valid_running_loss / counter
    epoch_acc = 100. * (valid_running_correct / len(testloader.dataset))
```

Artificial Intelligence algorithms in the digital processing of biomedical images

```
# Load the training and validation datasets.
dataset_train, dataset_valid, dataset_classes = get_datasets()
print(f"[INFO]: Number of training images: {len(dataset_train)}")
print(f"[INFO]: Number of validation images: {len(dataset_valid)}")
print(f"[INFO]: Class names: {dataset_classes}\n")
# Load the training and validation data loaders.
train_loader, valid_loader = get_data_loaders(dataset_train, dataset_valid)
# Learning parameters.
lr = 0.0001
# Define the maximum number of epochs
max_epochs = 50
# Define the number of epochs without improvement after which training will be stopped
patience = 5
# Initialize the best validation loss to a high value
best_val_loss = float('inf')

# def one_hot_encode(labels, num_classes): #added later pgt
#     return torch.eye(num_classes)[labels]

# labels = torch.tensor([0,1])
# num_classes =2
# one_hot_labels = one_hot_encode(labels, num_classes) # addedlater ends

device = ('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Computation device: {device}")
print(f"Learning rate: {lr}")
print(f"Epochs to train for unless interrupted: {max_epochs}\n")
model = build_model(
    pretrained=True,
    fine_tune=True,
    num_classes=len(dataset_classes)
).to(device)

# Total parameters and trainable parameters.
total_params = sum(p.numel() for p in model.parameters())
print(f"{total_params:,} total parameters.")
total_trainable_params = sum(
    p.numel() for p in model.parameters() if p.requires_grad)
print(f"{total_trainable_params:,} training parameters.")

# Optimizer.
optimizer = optim.Adam(model.parameters(), lr=lr)

# Loss function.
criterion = nn.CrossEntropyLoss()

# Lists to keep track of losses and accuracies.
train_loss, valid_loss = [], []
train_acc, valid_acc = [], []
```

Artificial Intelligence algorithms in the digital processing of biomedical images

```
# Start the training.
for epoch in range(max_epochs):
    print(f"[INFO]: Epoch {epoch+1} of {max_epochs}")
    train_epoch_loss, train_epoch_acc = train(model, train_loader,
                                              optimizer, criterion)
    valid_epoch_loss, valid_epoch_acc = validate(model, valid_loader,
                                                criterion)

    train_loss.append(train_epoch_loss)
    valid_loss.append(valid_epoch_loss)
    train_acc.append(train_epoch_acc)
    valid_acc.append(valid_epoch_acc)
    val_loss = np.mean(valid_loss)
    print(f"Training loss: {train_epoch_loss:.3f}, training acc: {train_epoch_acc:.3f}")
    print(f"Validation loss: {valid_epoch_loss:.3f}, validation acc: {valid_epoch_acc:.3f}")
    print('-'*50)
    time.sleep(5)

# Check if the validation loss has improved
if val_loss < best_val_loss:
    best_val_loss = val_loss
    counter = 0
    # Save the trained model weights.
    save_model(max_epochs, model, optimizer, criterion)
else:
    counter += 1

# If the counter has reached the patience limit, stop the training
if counter >= patience:
    print(f"Early stopping at epoch {epoch}")
    break

# Save the loss and accuracy plots.
save_plots(train_acc, valid_acc, train_loss, valid_loss)
print('TRAINING COMPLETE')
```

Inference for EfficientNetb0

```
import torch
import argparse
import torch.nn as nn
import torch.optim as optim
import time
import sys
import numpy as np
import os
import pandas as pd
#os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
os.environ['PYTORCH_CUDA_ALLOC_CONF'] = "max_split_size_mb:2048"
torch.cuda.empty_cache()

from tqdm.auto import tqdm

from model import build_model
from datasets import get_test_transform
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

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

TEST_DIR = '../input/test_images'
IMAGE_SIZE = 224 # Image size of resize when applying transforms.
BATCH_SIZE = 100
NUM_WORKERS = 4 # Number of parallel processes for data preparation.

dataset_test = datasets.ImageFolder(
    TEST_DIR,
    transform=(get_test_transform(IMAGE_SIZE))
)
test_loader = DataLoader(dataset_test, batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
# Load the trained model.
model = build_model(pretrained=False, fine_tune=False, num_classes=4)
checkpoint = torch.load('../outputs/model/model.pth', map_location=device)
print('Loading trained model weights...')
model.load_state_dict(checkpoint['model_state_dict'])
model.to(device)
```

```
# Validation function.
def inference(model, testloader):

    # Keep track of the actual and predicted classes for each image.
    gt_classes = []
    pred_classes = []

    model.eval()
    print('Inference')
    test_running_correct = 0
    with torch.no_grad():
        for index, data in tqdm(enumerate(testloader), total=len(testloader)):

            image, labels = data
            image = image.to(device)
            labels = labels.to(device)
            # Forward pass.
            outputs = model(image)

            # Calculate the accuracy.
            _, preds = torch.max(outputs.data, 1)
            test_running_correct += (preds == labels).sum().item()

            # Get the ground truth classes and the predicted classes
            gt_classes += labels.tolist()
            pred_classes += preds.tolist()

    # Accuracy for the complete epoch.
    epoch_acc = 100. * (test_running_correct / len(testloader.dataset))
    return epoch_acc, gt_classes, pred_classes

test_acc, gt_classes, pred_classes = inference(model, test_loader)
```

Artificial Intelligence algorithms in the digital processing of biomedical images

```
from sklearn.metrics import confusion_matrix as conf_matrix
import seaborn as sns
import matplotlib.pyplot as plt

confusion_matrix = conf_matrix(gt_classes, pred_classes)

ax= plt.subplot()
sns.heatmap(confusion_matrix, annot=True, fmt='g', cmap='PuBuGn', ax=ax);
class_names = ['glioma', 'meningioma', 'notumor', 'pituitary']

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(class_names); ax.yaxis.set_ticklabels(class_names);

gt_classes = np.array(gt_classes)
pred_classes = np.array(pred_classes)
total_samples = gt_classes.shape[0]
correct_predictions = sum(np.diag(confusion_matrix))
overall_accuracy = correct_predictions / total_samples
overall_accuracy_percent = overall_accuracy * 100
print("Overall accuracy: {:.2f}%".format(overall_accuracy_percent),)
```

APPENDIX 2: Coding part for ResNet18

Utils, Datasets, Training and Inference scripts are basically the same. We apply slight changes to our model script, in order to change it to ResNet18

Model for ResNet18

```
import torchvision.models as models
import torch.nn as nn
def build_model(pretrained=True, fine_tune=True, num_classes=4):
    if pretrained:
        print('[INFO]: Loading pre-trained weights')
    else:
        print('[INFO]: Not loading pre-trained weights')
    model_res = models.resnet18(pretrained=pretrained)
    if fine_tune:
        print('[INFO]: Fine-tuning all layers...')
        for params in model_res.parameters():
            params.requires_grad = True
    elif not fine_tune:
        print('[INFO]: Freezing hidden layers...')
        for params in model_res.parameters():
            params.requires_grad = False

    # Change the final classification head.
    model_res.fc = nn.Sequential(
        nn.Dropout(0.5),
        nn.Linear(in_features=512, out_features=num_classes)
    )

    return model_res
```


Training the resNet18 Model

```
import torch
import argparse
import torch.nn as nn
import torch.optim as optim
import time
import sys

from tqdm.auto import tqdm

from model_res import build_model
from datasets import get_datasets, get_data_loaders
from utils_res import save_model, save_plots

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

# Training function.
def train(model_res, trainloader, optimizer, criterion):
    model_res.train()
    print('Training')
    train_running_loss = 0.0
    train_running_correct = 0
    counter = 0
    for i, data in tqdm(enumerate(trainloader), total=len(trainloader)):
        counter += 1
        image, labels = data
        image = image.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        # Forward pass.
        outputs = model_res(image)
        # Calculate the loss.
        loss = criterion(outputs, labels)
        train_running_loss += loss.item()
        # Calculate the accuracy.
        _, preds = torch.max(outputs.data, 1)
        train_running_correct += (preds == labels).sum().item()
        # Backpropagation
        loss.backward()
        # Update the weights.
        optimizer.step()

    # Loss and accuracy for the complete epoch.
    epoch_loss = train_running_loss / counter
    epoch_acc = 100. * (train_running_correct / len(trainloader.dataset))
    return epoch_loss, epoch_acc
```

```

# Validation function.
def validate(model_res, testloader, criterion):
    model_res.eval()
    print('Validation')
    valid_running_loss = 0.0
    valid_running_correct = 0
    counter = 0
    with torch.no_grad():
        for i, data in tqdm(enumerate(testloader), total=len(testloader)):
            counter += 1

            image, labels = data
            image = image.to(device)
            labels = labels.to(device)
            # Forward pass.
            outputs = model_res(image)
            # Calculate the loss.
            loss = criterion(outputs, labels)
            valid_running_loss += loss.item()
            # Calculate the accuracy.
            _, preds = torch.max(outputs.data, 1)
            valid_running_correct += (preds == labels).sum().item()

    # Loss and accuracy for the complete epoch.
    epoch_loss = valid_running_loss / counter
    epoch_acc = 100. * (valid_running_correct / len(testloader.dataset))
    return epoch_loss, epoch_acc

# Load the training and validation datasets.
dataset_train, dataset_valid, dataset_classes = get_datasets()
print(f"[INFO]: Number of training images: {len(dataset_train)}")
print(f"[INFO]: Number of validation images: {len(dataset_valid)}")
print(f"[INFO]: Class names: {dataset_classes}\n")
# Load the training and validation data loaders.
train_loader, valid_loader = get_data_loaders(dataset_train, dataset_valid)
# Learning parameters.
lr = 0.0001
epochs = 50
device = ('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Computation device: {device}")
print(f"Learning rate: {lr}")
print(f"Epochs to train for: {epochs}\n")
model_res = build_model(
    pretrained=True,
    fine_tune=True,
    num_classes=len(dataset_classes)
).to(device)

```

```
# Total parameters and trainable parameters.
total_params = sum(p.numel() for p in model_res.parameters())
print(f"{total_params:,} total parameters.")
total_trainable_params = sum(
    p.numel() for p in model_res.parameters() if p.requires_grad)
print(f"{total_trainable_params:,} training parameters.")

# Optimizer.
optimizer = optim.Adam(model_res.parameters(), lr=lr)

# Loss function.
criterion = nn.CrossEntropyLoss()

# Lists to keep track of losses and accuracies.
train_loss, valid_loss = [], []
train_acc, valid_acc = [], []

# Start the training.
for epoch in range(epochs):
    print(f"[INFO]: Epoch {epoch+1} of {epochs}")
    train_epoch_loss, train_epoch_acc = train(model_res, train_loader,
                                              optimizer, criterion)
    valid_epoch_loss, valid_epoch_acc = validate(model_res, valid_loader,
                                                criterion)

    train_loss.append(train_epoch_loss)
    valid_loss.append(valid_epoch_loss)
    train_acc.append(train_epoch_acc)
    valid_acc.append(valid_epoch_acc)
    print(f"Training loss: {train_epoch_loss:.3f}, training acc: {train_epoch_acc:.3f}")
    print(f"Validation loss: {valid_epoch_loss:.3f}, validation acc: {valid_epoch_acc:.3f}")
    print('-'*50)
    time.sleep(5)

# Save the trained model weights.
save_model(epochs, model_res, optimizer, criterion)

# Save the loss and accuracy plots.
save_plots(train_acc, valid_acc, train_loss, valid_loss)
print('TRAINING COMPLETE')
```

Predictions for Resnet18

```

import torch
import argparse
import torch.nn as nn
import torch.optim as optim
import time
import sys
import numpy as np
import os
import pandas as pd
#os.environ['CUDA_LAUNCH_BLOCKING'] = "1"
os.environ['PYTORCH_CUDA_ALLOC_CONF'] = "max_split_size_mb:2048"
torch.cuda.empty_cache()

from tqdm.auto import tqdm

from model_res import build_model
from datasets import get_test_transform
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

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

from sklearn.metrics import confusion_matrix as conf_matrix
import seaborn as sns
import matplotlib.pyplot as plt

confusion_matrix = conf_matrix(gt_classes, pred_classes)

ax= plt.subplot()
sns.heatmap(confusion_matrix, annot=True, fmt='g', cmap='PuBuGn', ax=ax);
class_names = ['glioma', 'meningioma', 'notumor', 'pituitary']

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(class_names); ax.yaxis.set_ticklabels(class_names);

gt_classes = np.array(gt_classes)
pred_classes = np.array(pred_classes)
total_samples = gt_classes.shape[0]
correct_predictions = sum(np.diag(confusion_matrix))
overall_accuracy = correct_predictions / total_samples
overall_accuracy_percent = overall_accuracy * 100
print("Overall accuracy: {:.2f}%".format(overall_accuracy_percent),)

```

```

# Load the trained model.
model_res = build_model(pretrained=False, fine_tune=False, num_classes=4)
checkpoint = torch.load('../outputs/model/model_res.pth', map_location=device)
print('Loading trained model weights...')
model_res.load_state_dict(checkpoint['model_state_dict'])
model_res.to(device)

TEST_DIR = '../input/test_images'
IMAGE_SIZE = 224 # Image size of resize when applying transforms.
BATCH_SIZE = 100
NUM_WORKERS = 4 # Number of parallel processes for data preparation.

dataset_test = datasets.ImageFolder(
    TEST_DIR,
    transform=(get_test_transform(IMAGE_SIZE))
)
test_loader = DataLoader(dataset_test, batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)

# Validation function.
def inference(model_res, testloader):

    # Keep track of the actual and predicted classes for each image.
    gt_classes = []
    pred_classes = []

    model_res.eval()
    print('Inference')
    test_running_correct = 0
    with torch.no_grad():
        for index, data in tqdm(enumerate(testloader), total=len(testloader)):

            image, labels = data
            image = image.to(device)
            labels = labels.to(device)
            # Forward pass.
            outputs = model_res(image)

            # Calculate the accuracy.
            _, preds = torch.max(outputs.data, 1)
            test_running_correct += (preds == labels).sum().item()

            # Get the ground truth classes and the predicted classes
            gt_classes += labels.tolist()
            pred_classes += preds.tolist()

    # Accuracy for the complete epoch.
    epoch_acc = 100. * (test_running_correct / len(testloader.dataset))
    return epoch_acc, gt_classes, pred_classes

test_acc, gt_classes, pred_classes = inference(model_res, test_loader)

```