Deep Learning for Brain Tumor Detection using Semantic Segmentation of MRI Images

Anastasios Bachtses

Department of Electrical & Computer Engineering
University of Thessaly

Supervisor:
Prof. Aspasia Daskalopulu
Volos, January 14, 2021

Deep Learning for Brain Tumor Detection using Semantic Segmentation of MRI Images

Anastasios Bachtses

Department of Electrical & Computer Engineering
University of Thessaly

Supervisor:
Prof. Aspasia Daskalopulu
Volos, January 14, 2021

Βαθιά Μάθηση για Ανίχνευση Όγκου στον Εγκέφαλο με Σημασιολογική Κατάτμηση Μαγνητικών Τομογραφιών

Μπαχτσές Αναστάσιος

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ
Πανεπιστήμιο Θεσσαλίας

Επιβλέπουσα:
Ασπασία Δασκαλοπούλου
Βόλος, 14 Ιανουαρίου 2021

Acknowledgements

I would like to express my appreciation to Prof. Aspasia Daskalopulu, my supervisor, for her valuable suggestions and guidance through each stage of the process of this research work.

I would also like to thank Prof. Michael Vassilakopoulos and Prof. Yota Tsompanopoulou for placing their trust and confidence in my abilities.

Finally, I must express my profound gratitude to my family and my friends for providing me with wise counsel and continuous support throughout this journey of studies. This accomplishment would not have been possible without them. Thank you.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The 11 points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism.

The Declarant

Bachtses Anastasios
Volos, 14 January 2021

Abstract

The last decades Brain Tumors are one of the major causes of mortality of children and adults. Among them, Gliomas are the most common and also aggressive, leading to a very short life expectancy. Thus, early diagnosis plays a determining role in improving possibilities for treatment and increasing the recovery rate of the patients. Manual identification and detection of tumor from a large amount of brain Magnetic Resonance Images (MRI) generated in clinical routine, is difficult, requires high accuracy and often depends on the experience of the doctor. Following astonishing successes in recent years, Deep Learning techniques are attracting substantial interest, proving that they can address this problem more efficiently than many traditional methods. In this thesis, an attempt has been made to develop a Deep Learning Neural Network for autonomous segmentation of Brain Tumor on MRI images. For this purpose, someone must be familiar with Machine Learning and Deep Learning terms, as well as Brain Tumor substance and Neuroimaging, which will be provided through the introduction. The appropriate collection of data, then, has to be obtained, and pre-processing techniques must be applied to the MRI images of the dataset, in order to transform them into useful input for the implemented architecture. Furthermore, this research will describe the concept of Neural Networks, focusing on Convolutional Neural Networks and Semantic Segmentation techniques. Finally, several different implementations of the model will be provided, evaluating the performance and analyzing the results.

Key words: Artificial Intelligence, Semantic Segmentation, Neural Networks, Deep Learning, Brain Tumor, Brain MRI Segmentation, Machine Learning, LGG, Prediction, Detection

Περίληψη

Τις τελευταίες δεκαετίες, οι όγκοι του εγκεφάλου είναι μια από τις κύριες αιτίες θνησιμότητας παιδιών και ενηλίκων. Μεταξύ αυτών, τα γλοιώματα είναι τα πιο κοινά και επιθετικά, συρρικνώνοντας κατά πολύ τις πιθανότητες ανάρωσης. Καθίσταται σαφές, ότι η έγκαιρη διάγνωση κατέχει καταλυτικό ρόλο στη διαδικασία της θεραπείας και αυξάνει τις πιθανότητες επιβίωσης του ασθενούς. Η χειροκίνητη ανίχνευση και κατάτμιση όγκου από μια μεγάλη ποσότητα δεδομένων από Μαγνητικές Τομογραφίες (MRI) που παράγονται σε συνεχείς ρυθμούς καθημερινότητας, είναι δύσκολη, απαιτεί υψηλή ακρίβεια και πολύ συχνά εξαρτάται από την εμπειρία του γιατρού. Σημειώνοντας εκπληκτικές επιτυχίες τα τελευταία χρόνια, οι τεχνικές Deep Learning (Βαθιάς Μάθησης) προσελκύουν όλο και περισσότερο ενδιαφέρον, αποδεικνύοντας ότι μπορούν να προσεγγίσουν αυτό το πρόβλημα πιο αποτελεσματικά από πολλές παραδοσιακές μεθόδους. Στα πλαίσια αυτής της διπλωματικής εργασίας, παρουσιάζεται μια προσπάθεια να αναπτυχθεί ένα Νευρωνικό Δίκτυο Βαθιάς Μάθησης για αυτόνομη τμηματοποίηση και ανίχνευση όγκου στον εγκέφαλο απο εικόνες MRI. Προκειμένου να επιτευχθεί αυτό, κάποιος πρέπει να είναι εξοικειωμένος με τους όρους Μηχανικής Μάθησης, Βαθιάς Μάθησης, καθώς και να υπάρχει υπόβαθρο στην Ογκολογία και τη Νευροαπεικόνιση, όροι που παρέχονται στην εισαγωγή αυτής της εργασίας. Στη συνέχεια, θα πρέπει να συλλεχθούν τα κατάλληλα δεδομένα, συγκεκριμένα αρχεία Μαγνητικών Τομογραφιών και να εφαρμοστούν τεχνικές προεπεξεργασίας στα δεδομένα αυτά, προκειμένου να τα μετατρέψουν σε χρήσιμη είσοδο για την αρχιτεκτονική του δυκτύου που αναπτύχθηκε. Επιπρόσθετα, η εργασία παρέχει μία αναλυτική περιγραφή των Νευρωνικών Δικτύων, εστιάζοντας στον τομέα των Συνελικτικών Νευρωνικών Δικτύων και της Σημασιολογικής Κατάτμησης. Τέλος, θα παρουσιαστεί η υλοποίηση του μοντέλου, αξιολογώντας την απόδοσή του και αναλύοντας τα αποτελέσματα.

Λέξεις κλειδιά: Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Βαθιά Μάθηση, Κατάτμηση Εγκεφαλικών Όγκων, Σημασιολογική Κατάτμηση, Νευρωνικά Δίκτυα

Contents

List of Figures

List of Tables

List of Acronyms

ML Machine Learning

ANN Artificial Neural Networks

MRI Magnetic Resonance Image

NN Neural Networks

CNN Convolutional Neural Networks

DNN Deep Neural Networks

RNN Recurrent Neural Networks

LSTM Long Short-Term Memory

BP Backpropagation

LGG Low-Grade Gliomas

ReLU Rectified Linear Unit

Adam Adaptive Moment Estimation

GD Gradient Descent

SGD Stochastic Gradient Descent

FC Fully Connected

CHAPTER 1

Introduction

1.1 Can machines think?

As we are aware of, the human brain is a magnificent computational device, capable of handling vast amounts of data. The brain is able, without much effort, to process information and identify patterns in our surroundings, all of which are the basis of our decisions, and then to learn from the experiences that these decisions cause. This is of course something that composes our daily life and has helped the human species in survival and evolution. As technology and computers become increasingly powerful, the idea of imitating some of the brain's functions and implement them in machines, is becoming a reality. A reality that is called **Artificial Intelligence** (AI).

In 1950, and after breaking the Nazi encryption machine, Enigma, and helping Allied Forces win World War II, mathematician Alan Turing changed history for the second time asking a straightforward question: "Can machines think?" [1]. Through his paper "Computing Machinery and Intelligence" [1], where the famous Turing Test was introduced, he established the fundamental idea of Artificial Intelligence. A few years later, in 1956, John McCarthy coined the term AI for the first time by the following definition. "Artificial Intelligence is the science and engineering of making intelligent machines and making a machine behave in ways that would be called intelligent if a human were so behaving" [2]. AI is the field of Computer Science that studies the theories and methods which produce intelligence or intelligent behavior when applied through a computer.

While AI focuses on creating computers that can learn, recognize, understand, predict, and interact, there is an associate subset of it that permits computers to follow a self-learning approach without being programmed explicitly and autonomously learn new things from data. This subset is called **Machine Learning** (ML). According to Stanford University [3] "Machine Learning is the field of AI studying how computer agents can improve their perception, knowledge, thinking, or actions based on experience or data".

Frank Rosenblatt, in 1958, designed the first algorithm attempting to mimic the thinking process of the human brain, a binary classifier named Perceptron [4]. It was a function which could decide whether or not an input, represented by a vector of numbers, belongs to some specific class. In 1960, an early single-layer Artificial Neural Network introduced by Professor Bernard Windrow and his graduate student Ted Hoff at Stanford University, which called ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element) [5]. It was implemented on a physical device, consisting of weights, biases and a summation function forming a single artificial neuron. We will discuss about these terms at the next chapter. Several years later, in 1967, the nearest neighbour algorithm was conceived, aiming to identify basic patterns from the environment, like mapping a route in a city. It was the beginning of basic pattern recognition. In 1979, a robot named Stanford Cart developed by the students of Stanford University, being able to navigate itself around a room, while two years later, Gerald Dejong introduced the idea of Explanation Based Learning (EBL), in which a computer could analyze and discard irrelevant data from a provided dataset.

Since then, and specifically during the last decade, the world has witnessed tremendous achievements of AI including winning professional players in complex games, such as Chess, Jeopardy, Go and Dota, object detection from images, speech-to-text transformation, products suggestions based on user's interests, face recognition, virtual home assistants and many more.

In effect, Machine Learning facilitates the computers to program themselves by automating the process of automation. While in traditional programming, data and programs are provided as input to a computer in order to produce a certain output, in Machine Learning, models accept both data and output in order to create the program itself. They use algorithms to parse over data, learn from data and then make smarter decisions based on what they have learnt.

A very important part of a broader family of ML, where algorithms are used in structured layers in order to implement an Artificial Neural Network that can learn and make decisions on its own, is called **Deep Learning**.

Deep Learning models are formulated by much more complex structures and has been proven that these architectures are very efficient with high demanding data and currently they are the most successful Machine Learning approach [6]. Therefore, they are applied to many fields of research and industry. The word “deep” is related to the idea of using multiple layers that contribute sequentially in the process of learning. The number of layers is the depth of the model. Today, Deep Learning models often consist of tens or even hundreds of layers that have the ability to learn automatically from getting exposed to data. The way these architectures achieve that will be discussed at the next chapter.

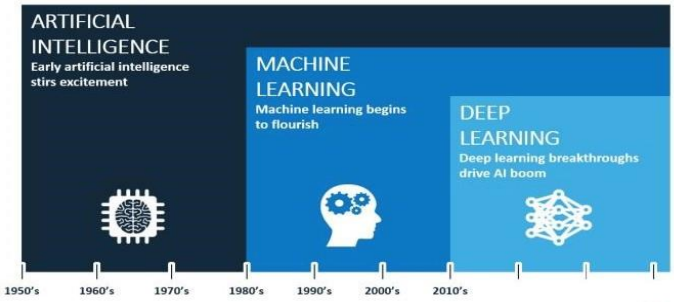

Figure 1. 1: AI, Machine Learning, Deep Learning [7]

In the Figure 1.1, it is illustrated how Deep Learning is a subset of Machine Learning, which is a subset of the broader Artificial Intelligence field.

1.2 Brain Tumors

The brain is an amazing organ that regulates the entire body functions, interprets the outside world's information, and controls almost each and every vital activity of the human body. Intelligence, senses, emotions, movement, memory, thought, speech, physical activity, and creativity are only a few of the many functions that are governed by the brain. Therefore, any kind of harm on this vital organ, will disturb the proper functioning of the entire human body, causing irreversible damages. To assist with the natural processes of our body, the brain cells grow, divide and multiply, repairing damages and replacing the old cells. However, during cell growth and division, it is possible for mistakes to be made and the process to not finished properly, producing abnormal cells. Fortunately, these abnormal cells are destroyed, most of the times, by the natural defensive mechanisms of the body, but occasionally the abnormal cells expand, multiply and form a lump of cells. When this happens in the brain, a **primary brain tumor** is formed.

Among the various problems to the brain, primary brain tumors are the most common and also life-threatening today, including tumors that originate from the tissues of the brain or its immediate surroundings. They are classified as **benign**

(noncancerous) or **malignant** (cancerous). The difference between the two types is that, benign tumors generally, do not expand to other tissues and organs, and are considered to be curative under complete surgical excursion, whereas malignant tumors are able to grow rapidly, invade and destroy nearby normal tissues, getting spread throughout the body and require chemotherapy, radiotherapy, or a combination [8].

The most common primary brain tumors are **Gliomas, Meningiomas** and **Pituitary** tumors. Gliomas are a group of tumors, that arise from brain tissues, other than nerve cells and blood vessels, and they are most of the times malignant. On the other hand, meningiomas are created from the membranes, that cover externally the brain and surround the central nervous system and they are typically benign, whereas pituitary tumors remain in the pituitary gland and even if benign, can cause other medical damage [9].

This thesis is mainly focused on **Gliomas**, which are the most common brain tumors. More than 250.000 new cases of primary malignant brain tumors are diagnosed annually worldwide, 77% of which are Gliomas [10]. Arising from the supporting cells of the brain, glia, including astrocytes, ependymal cells and oligodendroglial cells, glial tumors subdivided to Astrocytomas, Ependymomas, Glioblastoma Multiforme (GBM), Medulloblastomas and Oligodendrogliomas. According to the World Health Organization (WHO) and American Brain Tumor Association [11], Gliomas, in order to get distinguished by their malignancy, are classified into scale from grade I to grade IV based on their histological properties.

Grade I Gliomas (mainly pilocytic astrocytomas) are benign tumors, since they are well delineated and non-infiltrative. After completing surgical resection, they have an excellent prognosis. Diffuse grade II Gliomas are slow-growing infiltrative tumors with continuous growth [12]. They inevitably progress into a malignant tumor. Grade III Gliomas, or anaplastic Gliomas, are rapidly developing malignant tumors while grade IV Gliomas (glioblastomas multiforme) are the most aggressive tumors with a low treatment success rate.

The grades I and II are being referred to as **Low-Grade Gliomas (LGG)** and possess a slow growth, while grade III and IV are also called **High-Grade Gliomas (HGG)** and possess a rapid growth of tumors. If the Low-Grade brain tumor is left untreated, it is probable to evolve into a High-Grade, which is a malignant brain tumor. LGG represent approximately 27% of all primary brain tumors, with an average age at the time of diagnosis ranging between 43 and 48, depending on histological subtype [13]. The current thesis is an attempt to autonomously detect LGG tumor areas inside the brain.

1.3 Neuroimaging

Diagnosing cancer at its earliest stages often provides the best chance for a cure. Considering the tumors vary widely in terms of their stature, shape and presence, it is actually very difficult to extract accurate measurements in order to properly diagnose the tumors. Manual diagnosis of tumors requires the radiologists to detect abnormalities through physical exams, laboratory tests, blood analysis and biopsy. Apart from being a time-consuming task, manual detection results are also dependent on the experience and the judgment of the radiologist and additional tests like an angiogram, spinal tap and more samples for biopsy are often needed. Therefore, **neuroimaging** tools that visualize the different structures and tissues of the brain are required most of the times, as shown in Figure 1.2 (a). The main purpose of any neuroimaging application is to scan an exact body area of the patient and produce images, in order the essential attributes to be extracted and the diagnosis to be even more accurate. Today, imaging data account for about 90% of all healthcare data [14] and hence is one of the most important sources.

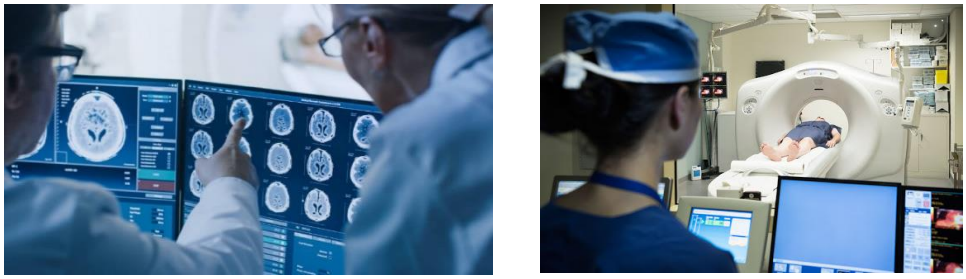

Figure 1. 2: (a) Neuroimaging results (b) MRI scanner [15]

There exists a variety of neuroimaging methods. Among them, the most popular are **Magnetic Resonance Imaging (MRI)** and **Computed Tomography (CT)**. MRI scanners, as shown at the Figure 1.2 (b), produce magnetic fields, magnetic field gradients and radio waves to capture and generate images of the organs inside the body. CT scanning involves ionizing radiation or X-ray technology that is based on the absorption of X-rays beams as they pass through the different tissues of a patient's body. MRI is based on the principle of Nuclear Magnetic Resonance (NMR) that is commonly used in spectroscopy to study the physical and chemical properties of molecules, while other methods such as Positron Emission Tomography (PET), which also requires the usage of X-rays, can also recognize brain tumors. Considering all the hazards of exposing the human body to ionizing radiation, MRI appears to be the most convenience method, other than CT and PET scan.

However, the final decision is yet susceptible to human subjectivity and observation, while early brain tumor detection mostly depends on the experience of the radiologist. Additionally, Glioma is a tumor described by varying intensity profiles and highly heterogeneity. Composed of a necrotic core, a margin of tumor activity, edema tissue, different degrees of aggressiveness and heterogeneous shape and appearance, its imaging procedure requires multiple MRI sequences. Therefore, Gliomas detection and particularly, autonomous segmentation is one of the most challenging tasks in science today.

1.4 Related Work

Being, undoubtedly, a scientific field with increasing interest, within the last few years, many significant researches have been proposed Machine and Deep Learning techniques to develop computer-aided systems, capable of diagnosing brain tumors autonomously. In this section an overview of some of the recent and prominent studies is presented.

During Brain Tumor Segmentation Challenge (BRATS) in 2014, Urban et al. [16] proposed a 3D Convolutional Neural Network (CNN) architecture for the multi-modal MRI Glioma tumor segmentation task. In their study, they segmented the whole tumor area in three regions named whole tumor, core tumor and active tumor. Their approach involved two different CNN networks using deeply supervised layers. Reported average results of the two networks shows that their proposed method achieved the dice scores of 87% for the whole tumor region, 77% for the core tumor region and 73% for the active tumor region.

In contrast to the high dimensional method of Urban et al., Zikic et al. [17] developed an interpretation method to transform 4D input data, so that standard 2D-CNN architectures can be used to solve the brain tumor segmentation task. Containing two convolutional layers with 64 filters of size ($5 \, x \, 5 \, x \, 4$) and ($3 \, x \, 3 \, x \, 4$) respectively, separated by a max-pooling layer, followed by one FC layer and a soft-max layer, this approach could remove the burden of high dimensional CNN designs, while increasing computational efficiency. Reported results after training on limited data for this study indicate BRATS dice scores of 83.7% for the whole tumor region, 73.6% for core tumor region and 69% for active tumor region.

In 2016, Havaei et al. [18]. presented a novel CNN architecture which was a fully automatic brain tumor segmentation method, based on Deep Neural Networks,

focused on glioblastomas (both Low and High grade) pictured in MRI images. Using Convolutional Neural Networks that processed local details of the brain scan along with the larger context of brain tissue, they described a cascade architecture in which the output of a basic CNN is treated as an additional source of information for a subsequent CNN. 2D multi-modality global input patches with size ($65 \, x \, 65 \, x \, 4$) were first processed by a CNN to output patches with size ($33 \, x \, 33 \, x \, 5$). Those output patches were then concatenated with the local patches of size ($33 \, x \, 33 \, x \, 4$) and delivered as an input to a two-pathway CNN with convolutional layers containing ($7 \, x \, 7$) sized filters in one path and ($13 \, x \, 13$) sized filters in the other one. The reported result reveal that this architecture achieved 88.0% for the whole tumor region, 79.0% for core tumor region and 73% for active tumor region.

One of the recent approaches, evaluating deeper CNN architectures, was the paper of Pereira et al. [19]. This approach implemented small ($3 \, x \, 3$) sized filters in convolutional layers. In this way, more layers can be added to the architecture without reducing the effective receptive field of the traditional bigger filters. The proposed CNN that had 11 layers of depth (6 convolutional layers followed by 3 FC layers with 2 max-pooling layers) obtained BRATS dice scores of 88%, 83% and 77% for whole tumor, core tumor and active tumor regions respectively.

CHAPTER 2

Theory of Neural Networks

2.1 Definition of Neural Networks

Artificial Neural Networks (ANN) are computing systems designed to simulate the way the human brain analyzes and processes information. They are the core and the fundamental mechanism behind the idea of Artificial Intelligence, nowadays solving problems that would prove impossible. ANNs have self-learning capabilities that enable them to produce better results as more data becomes available. The idea of Artificial Neural Networks was originally inspired by biological neural networks. Living organisms, from the simplest to the most complex, have a nervous system that is responsible for a number of processes, such as contact with the outside world, learning, memory and much more.

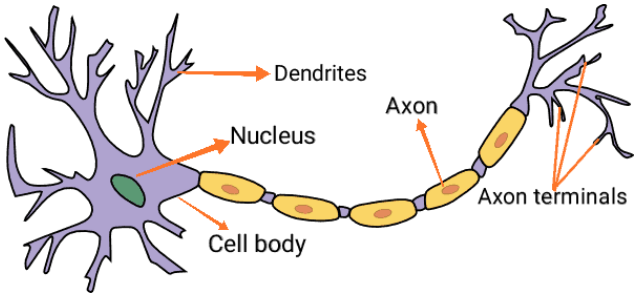

Figure 2. 1: Biological neuron [20]

The central unit of the nervous system is, of course, the brain, which is also made up of biological neural networks. Each neural network consists of billions of units,

called **neurons**. The neuron is the smallest independent unit in the network, such as e.g., the atom is the smallest unit of matter. In Figure 2.1 is represented the structure of the biological neuron. Over 80 billion neurons can be found in the human nervous system, while they are connected with approximately 10^{14} synapses [21]. Each neuron receives input signals from its dendrites and produces output signals along its single axon. Via the synapses, the axon connects the output of one neuron to the dendrites of other neurons. Neurons continuously process information, receiving and sending electrical signals. Each neuron has two states: one in which it sends a pulse of several Hz, and one that does not send a signal. In Figure 2.2 someone can observe the correspondence between biological and artificial neuron.

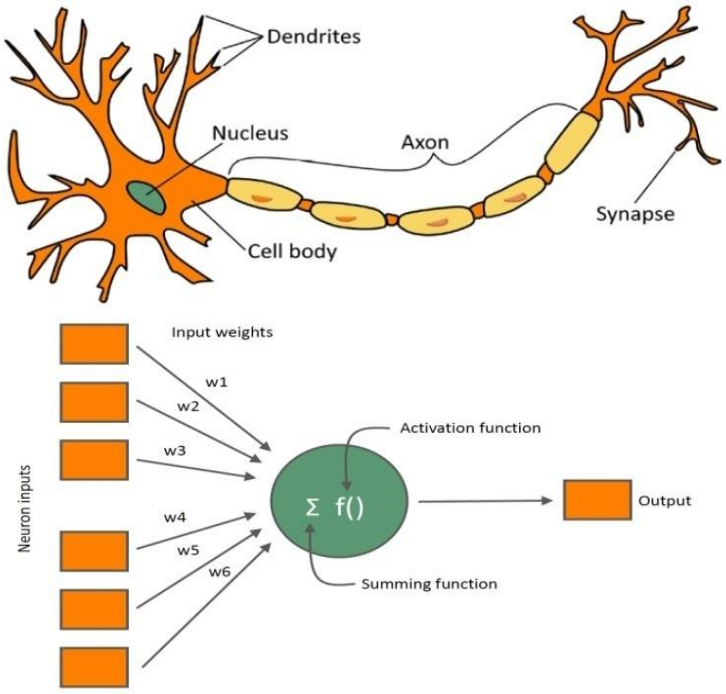

Figure 2. 2: Biological neuron side by side with Artificial neuron [22]

In the computational model of an Artificial Neural Network a neuron is represented by a node. Each signal (x_i) transmitted from one neuron to another, is multiplied by a corresponding weight value (w_i) the role of which corresponds to the role of the synapse in the biological neuron. Now, the central processing unit, the nucleus of the neuron, consists of two parts: the adder ($\sum$) and the activation function (f). The adder adds all the input signals along with their weights ($x_i w_i$) by generating their sum ($\sum_i x_i w_i$). Then an additional external parameter is added, the bias (b). In the basic model, every neuron performs the equation: ($\sum_i x_i w_i + b$). Thus, to decide whether outside connections should consider this neuron as activated or not, the whole signal is passed through the activation function $f(\sum_i x_i w_i + b)$, which is

a type of filter that forms the final value of the output. Applying a threshold, it transforms the total signal to On (1) or Off (0). If the function's input is above threshold, the neuron becomes activated, sending a spike to the next neuron and so on.

2.2 How does a Neural Network work?

A neural network contains layers of interconnected nodes with the typical structure: the input layer, the hidden layers and then, the output layer. The **input layer** is the first layer of the network and it collects input data in the form of vectors, which may represent features, single values, images, time-series or other types of data. The number of its neurons is equal to the data variables or the number of the given features. Input neurons don't perform any type of computation, but only pass the input vector to subsequent neurons. The layer or layers between the input and the output layer are called **hidden layers**. The main computation of a neural network takes place in the hidden layers. Each hidden layer receives all the inputs from the previous layer and performs the necessary calculations to generate a result, as we can observe in Figure 2.3. This result is then forwarded to the next layer, and so on. The term "hidden" implies that they are not visible to external systems, forming the inner core of the network. The number of hidden layers contained in a NN and the number of neurons in each hidden layer is determined by the designer and depend on the architecture that will be chosen. The larger the number of hidden layers, the longer it takes for the network to produce outputs and the more complex problems the neural network can solve.

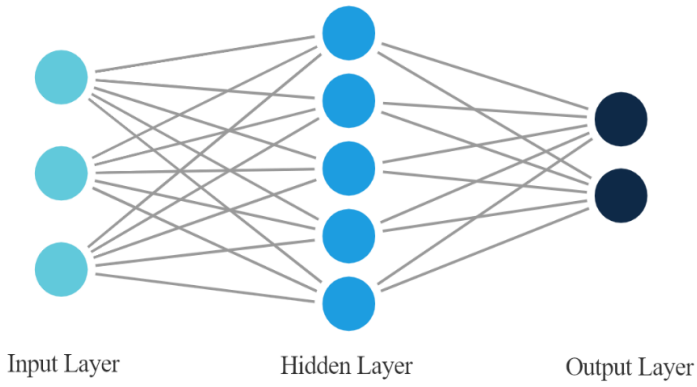

Figure 2. 3: A Neural Network with one hidden layer of five nodes, one output layer of two nodes, and three inputs. [23]

The last layer of the network is the **output layer**. Receiving input from the last hidden layer, it specifies the desired number of output values and contains output signals to which input patterns may correlate. This is where the final result is

produced and the number of its neurons is equal to the possible output variables of the results. For example, in image classification problem the output layer will consists of the number of classes into which each image is likely to be classified.

NNs are categorized according to their architecture and connectivity among its neurons in:

- **Feedforward Neural Networks.** In this model, the signal only travels in one direction, towards the output layer [24]. In Feedforward networks the information moves directly from the input layers, through the hidden layers (if any) and finally to the output layers. There are no cycles, neither loops in the network. They are widely used in pattern recognition problems.

- **Recurrent Neural Networks** (RNN) are a type of neural networks that allows previous outputs to be used as inputs while having hidden states. In a RNN the information cycles through a loop [25], allowing it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as connected handwriting recognition or speech recognition.

Apparently, the initial step on designing a neural network, is the architecture selection. The developer must decide how many input and output neurons should the network contain, how many hidden layers will be included between them and how the learning procedure will be designed. The number of input and output units usually depends on the context. For example, if the network is expected to produce a Boolean value, i.e. Yes or No, it should have only one output, while if the network is expected to produce a whole image, then the output layer should consist of a two-dimensional matrix that represents an image.

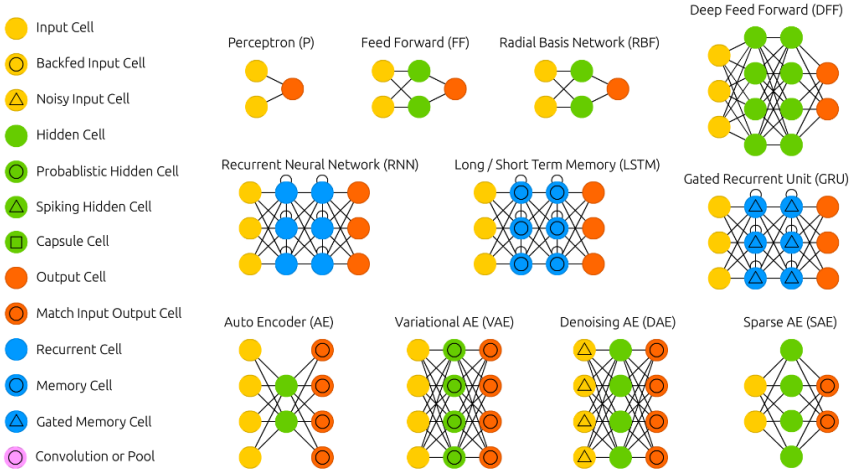

Figure 2. 4 Common NN topologies. The Neural Network Zoo [26]

In contrast, selecting the number of hidden layers is not so obvious. In most situations, there is no exact way of how to determine the most effective number of hidden units, without training several networks and estimating the generalization error for each of them. Too few hidden units will produce high training error because network is not complex enough to start recognising patterns. Too many hidden units lead to low training error but still have high generalization error due to overfitting. These situations will be discussed more at the next sections.

Once the network has been properly structured for a particular application, then it is ready to start learning in order to produce the responsive and desired outputs. This is the most genuine part of Deep Learning and also the main phase of the process and it is called **training**. Training is the ability of neural networks to learn and capture relationships between a dataset's features and a target variable. It can be defined as the gradual improvement of the network's skill to solve a problem (e.g., the gradual approach of a function). Training is an iterative process of gradually adjusting the parameters of the network (weights and biases) to approach values that constitute the solution. In order a neural network to be trained, it should learn on pairs among input data and its expected output (labels), which all together form the training data. There are two approaches of training: **supervised** and **unsupervised**. Supervised training involves a mechanism of providing the network with the desired output for every input. This is the method that was used to develop the specific models for this study. Unsupervised training is where the network has to make sense of the inputs without any outside help.

The idea is significantly based on the thinking process of humans, just as children learn by being told what they are doing right or wrong. At the real world, as humans, after an action we compare what actually happened with the outcome we wanted, figuring out the difference between the two, and using that to change our action next time. Neural networks learn things in exactly the same way, typically by a procedure of transmitting the information forward, which is called **forward propagation**, and then backward to improve their behavior, a feedback process called **backpropagation**.

At the first step of training, the forward propagation occurs when the neural network is exposed to training data, that flow through the entire network for their predictions to be calculated. When this process has been completed, having all the neurons made their calculations and the information has crossed all the layers, then the final layer will be reached with a result of prediction for those input data.

The network, then, compares its outputs (predictions), against the desired outputs (labels, targets) and estimates the difference between them, calculating the **loss** (or

error). The loss is one of the most important components of the training process, as a tool to measure how well an Artificial Neural Network is converging on the ability to predict the right answer, in relation to the correct result. The method which calculates the loss is called **Loss Function.**

Once the loss has been calculated, this information is propagated back through the system, layer by layer, until all the neurons in the network have received a loss signal that describes their relative contribution to the total loss, causing the system to modify the weights of the connections between the nodes to the correct direction. To realize what the right direction is, the network uses a technique called **Gradient Descent**. Taking advantage of the derivative of the loss function this technique changes the weights in small increments, allowing the network to decide in which direction "to descend" towards the global minimum of the loss function. Over iterations, the backpropagation procedure compels the weights to meet more appropriate values, reducing the difference between actual and intended output and, eventually, causing the network to learn.

During the training, applying this method, the same training data is processed several times. A complete pass over the whole training dataset is called an **epoch**. In every one of these iterations, the aim is to minimise the loss as close as possible to zero. The system continues to iterate over data, stopping only when the model reaches some statistically desired **accuracy** and **loss**, and under these consequences, producing accurate and appropriate predictions. If a network can't solve the problem, perhaps the designer has to reconsider the input and output data, the number of hidden layers, the size of each layer, the connections between them, the activation functions, the loss functions, and even the initial weights themselves.

2.3 Activation Functions

In this section we will describe the functions that determine each neuron's output, reminding the neuron model that described previously in Chapter 2.1. As long as, the equation $Y = \sum input_i * weight_i + bias,$ can produce values ranging from "$-inf$" to "$+inf$", we apply **activation functions** $f(Y)$ on it, in order to bound that range and deploy it as a valuable number. For this purpose, and to be able to decide whether this neuron should be considered as activated or not, this signal is usually filtered by a threshold and finally transformed into a suitable value. The most common activation functions in Machine Learning are presented below:

- **Sigmoid**

  One of the most widely used non-linear activation function is sigmoid. Sigmoid returns an output value between 0 and 1 for any given real-valued input number. The mathematical expression for sigmoid is:

$$f(x) = \frac{1}{1+ e^{-x}}$$

  In fact, large positive numbers become 1 and large negative numbers become 0. This means the output from the node will be a high signal, for a positive input, or a low one for a negative input. Mathematically, sigmoid is smooth, continuous, monotonic and bounded. The simplicity of its derivative allows the network to efficiently perform binary classification problems, because its output can be interpreted as a probability.

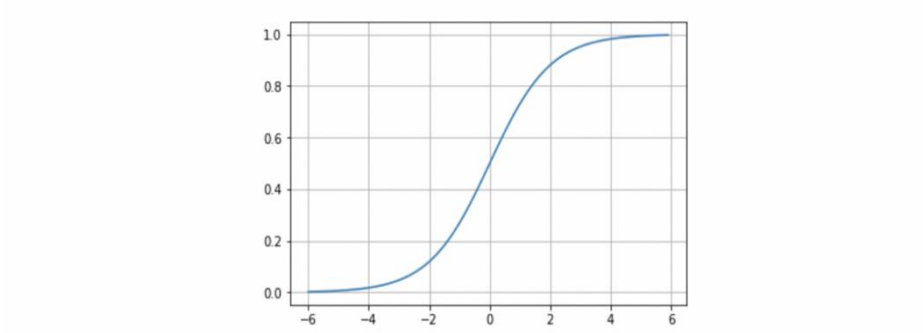

Figure 2. 5: Sigmoid function returns real numbers in the range [0,1]

- **Tanh**

  The hyperbolic tangent function, tanh, is a very similar function to sigmoid and they have many of the properties same. However, this function allows the network to map the input to any value between -1 and 1. It has a natural threshold of 0, meaning that any input value greater than 0 is considered high, or 1 in binary terms, and every input value smaller than 0 is considered as low or -1.

$$tanh(x) = \frac{2}{1+ e^{-2x}} - 1$$

  Tanh is smooth, continuous, bounded and monotonic while its derivative is not monotonic. The advantage of tanh is that negative numbers can be dealt with more easily. In effect, this allows the neural network to perhaps apply a negative penalty to the node, rather than just have the node not activated at all.

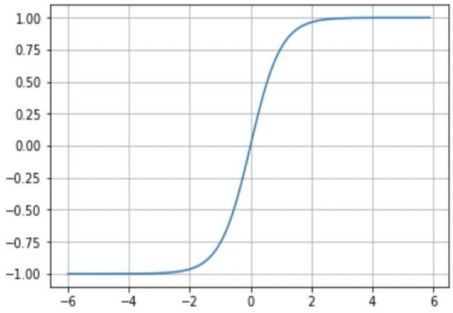

Figure 2. 6: Tanh function returns real numbers in the range [-1,1]

- **ReLU**

  The ReLU is the Rectified Linear Unit function, returning the input directly, if it is positive and 0 otherwise. It is defined as:

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

  Typically, the most usual behavior of ReLU is that, as long as the input is negative, the function will convert it to 0 and the neuron does not get activated, but when the input rises above, then the output becomes a linear relationship of the form $f(\mathrm{x}) = \mathrm{x}$. This is the main advantage of using the ReLU function over other activation functions, as it does not activate all the neurons at the same time.

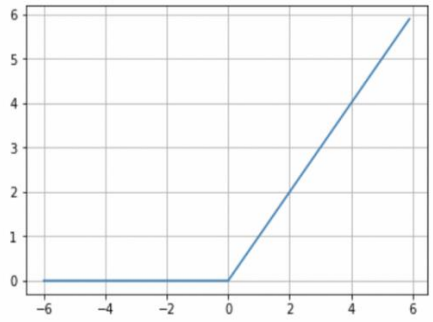

Figure 2. 7: ReLU function returns zero for x<0 and then it is linear with slope=1 when x>0

The ReLU has proven to work in many different situations and it is one of the most widely used activation functions in Deep Learning problems.

- **Softmax**
  Softmax is a function which normalizes the outputs for each class between 0 and 1, and, dividing by their sum, it returns the probability of the input value to belong in a specific class. The mathematical expression of the softmax is:

$$f(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

  Being able to handle multiple classes, this function is useful for the neurons of the output layer, for neural networks that need to classify inputs into multiple categories. It can contain numerous decision limits and return a probability distribution.

The selection of activation function is crucial when building a neural network. The designer has to configure what is the appropriate activation function for every particular problem, in order to lead to successful training process and convergence.

2.4 Backpropagation

The backpropagation algorithm was introduced during the 1970s, but its importance wasn't fully appreciated until a paper by David Rumelhart et. al. in 1986 [27]. The authors described the backpropagation algorithm and apply it to several neural networks, approaching the learning process more efficiently and proving that this algorithm makes it possible to solve problems which previously had been insoluble. Until today many important publications have been made, including Yann Le Cun [28] in 1985 (director of AI Research, Facebook).

The idea behind this algorithm is to determine the errors of the hidden layer by back-propagating the errors of the output layer. This algorithm provides the ability for neural network to calculate efficiently the gradients of the cost function. Technically, the backpropagation algorithm is a method for constantly readjusting the weights in a multilayer feed-forward neural network. It requires a network structure, consisting of one or more layers, where every layer is fully connected to the next layer. The basic steps of the learning process including the backpropagation algorithm are provided below:

1. Training pairs are prepared. Every input data x_i forms a pair with its label-target t_i. Assume that a training set consists of N pairs (x_i , t_i):

   $$Training\ set\ = \{(x_i,\ t_i)\},\quad i = 1, \dots, N,\quad x_i \in R^n,\ \ t_i \in\ R^n$$

   , where x is the input vector, containing all the x_i values from each feature, and t is the desired output vector, containing all the corresponding t_i targets.

2. Network's weights w_i are initialized randomly and combined with the bias b they create the matrix w.

3. Feed-forward computation. The input vector x is getting passed through the network and sequentially the output $\hat{y}$ is getting produced. Considering the randomness of the weights at the first iteration, the output $\hat{y}$ will obviously differ from the desired target t.

   $$z = \sum_{i=0}^{N} x_i w_i = W^T x$$

   $$\hat{y} = f(z)$$

   The squared error is then calculated.

   $$E = (\hat{y} - t)^2$$

4. Backpropagation. The aim for the neural network now is to identify how does the loss function behave and find the global minimum in order to minimize the error. In other words, it has to discover the combination of the weight values that decreases the error of the network. The main idea is creating a tool that detects in which direction is the most likely to find the minimum of the function. This process in Machine Learning is called optimization and typically the most common way to achieve that is by the calculation of **gradient descent**.

   An analogy for understanding this idea is a hypothetical scenario where a climber is stuck in the mountains and is trying to find the way to get down through heavy fog (in mathematical terms trying to find the global minimum). Considering the path down the mountain is not visible, he/she must use local information of his/her surroundings to find the direction. The method of gradient descent, involves looking at the steepness of the hill at his/her current position and then proceeding in the direction with the steepest descent (i.e., downhill). Otherwise, he/she might follow the wrong path and get lost moving towards the top of the mountain. However, assuming that the calculation of the steepness of the hill requires some time to get measured, the climber has to

decide the frequency at which he/she should stop to perform a calculation, in order to get down the mountain before the night. In this analogy, the steepness of the hill represents the slope of the error surface at any particular point. Below, in Figure 2.8 we can see the visualization of the loss function.

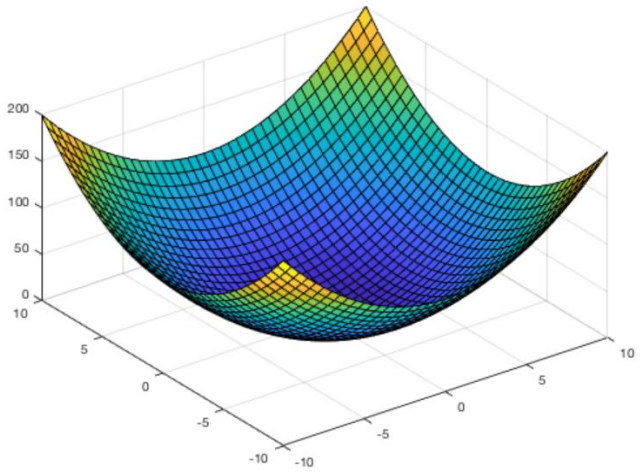

Figure 2. 8: Loss Function visualization [29]

The slope of a surface, in mathematics, can be calculated using the first derivative, gradient, of the squared error function at that point. This method actually computes the gradient of the loss function with respect to each weight and allows neural network to take small gradual and repetitive steps until finding the global minimum of the function, and as a result the appropriate values for each weight [30]. The size of these steps is the **learning rate** a of the algorithm. Learning rate is a number between 0 and 1 that determines how fast the neural network adjusts itself to the patterns from the training data. This parameter must be chosen carefully, because too small step will make the learning process slow, while too large step might lead to divergence.

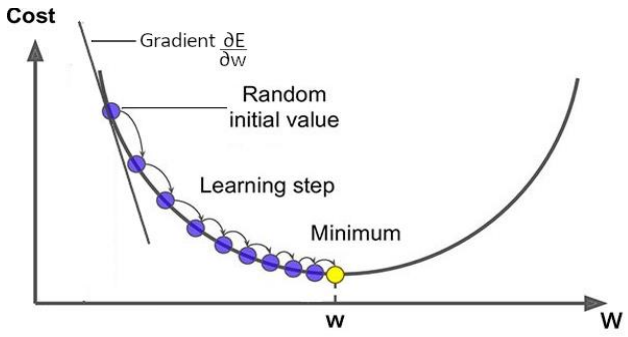

Figure 2. 9: Schematic of Gradient Descent [31]

5. Weighs update. Finally, once all the neurons have the value of the gradient that corresponds to them, the parameters are getting updated in the opposite direction to that indicated by the gradient.

$$w_i^{t+1} = {w_i}^t - a\frac{\partial E}{\partial {w_i}^t}$$

$$b^{t+1} = b^t - a\frac{\partial E}{\partial b^t}$$

In fact, the gradient, always points in the direction in which the value of the loss function increases. Therefore, if the network uses the negative of the gradient, it can get the direction in which the loss function is going to be reduced. As a result, it updates the weights, increasing or decreasing them accordingly. The update applied backwards in every layer sequentially until it reaches the input layer. At this time, one cycle of the procedure (one epoch), has been completed. These steps are repeated, for a given number of epochs, until the network approaches the correct weights values that produce relatively accurate outputs. Then the training process is finished.

2.5 Optimization Algorithms

Apart from the gradient descent fundamental method, which was described, several algorithms and techniques are used in the update procedure of a neural network, in order to reduce the loss. Some of the most efficient optimizers are described below.

- **SGD**
  As a variation of the gradient descent algorithm, Stochastic Gradient Descent (SGD), similarly updates the model in through an iterative process in order to minimize the error of the loss function. While gradient descent has to run over all the samples in the training set to operate a single update for a parameter in a particular iteration, SGD on the other hand, uses only one sample from the training set to perform the update. This method, instead of updating the weights based on the sum of the accumulated errors over all samples, uses the following rule:

$$w_i^{t+1} = {w_i}^t - a\frac{\partial E}{\partial {w_i}^t}$$

$$b^{t+1} = b^t - a\frac{\partial E}{\partial b^t}$$

Here, the error function represents the calculated error from only one training sample. As an advantage, the more frequent updates, provide immediately an insight into the performance of the model and accomplish even faster learning on some problems. However, it is computationally expensive, while taking significantly longer to train models on large datasets.

- **Adagrad**

  The Adaptive Gradient Algorithm (Adagrad) is an optimizer method that, contrary to gradient descent, where the learning rate stays fixed, allows it to adapt based on the parameters, modifying it in every weight update [32]. Adagrad algorithm changes the update step in every iteration, in a way that it will decrease if a weight is being updated too much in a short amount of time and it will increase if a weight is not being updated much. In Adagrad the neural network uses the first derivative of loss function and also uses a different learning rate α for every parameter w_i (or b) for every time step t. So, the update formula of the weights is the following:

$$g_{t+1} = g_t + \left(\frac{\partial E}{\partial {w_i}^t}\right)^2$$

$$w_i^{t+1} = {w_i}^t + \frac{a}{\sqrt{g_{t+1}} + \varepsilon}\ \frac{\partial E}{\partial {w_i}^t}$$

$$b^{t+1} = b^t + \frac{a}{\sqrt{g_{t+1}} + \varepsilon}\ \frac{\partial E}{\partial b^t}$$

  , where g_t is the gradient of the loss function with respect to the parameter w_i at a time-step t and ε in the denominator is a very small value to ensure division by zero does not occur. Using this formula, the network constantly changes the learning rate α, keeping the gradient g_t from the previous iteration to influence the learning rate which is calculating and updating the weights, as well, at the current moment.

  The main benefit of Adagrad is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 or 0.001 and leave it constantly there. However, the main weakness is its accumulation of the squared gradients in the denominator. Considering

square cannot be negative, the gradient g_{t+1} will always increase by some amount during training, regardless of a weight's past gradients g_t. This will cause the learning rate to shrink and eventually become infinitesimally small, at a point, where the algorithm is no longer able to acquire additional knowledge. The following algorithms aim to resolve this problem.

- **Adadelta**

  Adadelta is an extension of Adagrad which tends to reduce the monotonically decreasing learning rate problem of it [33]. Instead of accumulating all past squared gradients, in Adadelta the sum of gradients is recursively defined as a decaying average of all the past squared gradients. Following this method, the network does not need to set a default learning rate, while the average gradient $E[g_t{}^2]$ at any current iteration is calculated by using the average of the gradients from the previous time-steps.

$$E[g_t{}^2] = \gamma E[g_{t-1}{}^2] + (1-\gamma) g_t{}^2$$

$$w_i^{t+1} = w_i{}^t - \frac{a}{\sqrt{E[g_t{}^2] + \varepsilon}}\, g_t{}^2$$

$$b^{t+1} = b^t - \frac{a}{\sqrt{E[g_t{}^2] + \varepsilon}}\, g_t{}^2$$

  The constant term γ is set around 0.9. This formula forces the training process to continue normally, considering the learning rate does not decay. The disadvantage is that computationally this method is expensive.

- **Adam**

  Adaptive Moment Estimation (Adam) is an optimizer that computes the adaptive learning rate of each parameter, by estimating the first and second moments of the gradients [34]. Adam reduces the radically diminishing learning rates of Adagrad, implementing the exponential moving average of the gradients, in order to adjust the learning rate, instead of a simple average. In addition to storing an exponentially decaying average of past squared gradients, like Adadelta, Adam also keeps an exponentially decaying average of the past gradients. The algorithm first updates the exponential moving averages of the gradient (m_t) and the squared gradient (v_t) which is the estimates of the first and second moment.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Parameters β_1, $\beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages.

Considering moving averages are initialized as 0, the moment estimates are biased around 0 especially during the initial time-steps. This initialization bias can be easily counteracted resulting in bias-corrected estimates:

$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

The algorithm then updates the weights with the following formula:

$$w_i^{t+1} = w_i{}^t - \frac{a}{\sqrt{\widehat{v_t}} + \varepsilon} \widehat{m_t}$$

$$b^{t+1} = b^t - \frac{a}{\sqrt{\widehat{v_t}} + \varepsilon} \widehat{m_t}$$

Performing with computational efficiency and requiring very small amounts of memory, Adam optimizer is among the most popular optimization algorithms in Machine Learning today. In Stanford's Deep Learning course titled "CS231n: Convolutional Neural Networks for Visual Recognition" Adam algorithm has been suggested the as the default optimization method for Deep Learning applications.

2.6 Convolutional Neural Networks

Working with Deep Learning on images, often requires the usage of a specific category of neural networks, called **Convolutional Neural Networks (CNN)**. In recent years, Convolutional networks have been tremendously successful in solving

difficult image-driven pattern recognition tasks and processing data that has a grid-like topology. Examples include time-series data, which can be thought of as a 1D matrix, containing samples at regular time intervals, and image data, which can be thought as a 2D matrix, containing the values of each pixel. The main assumption of convolutional filters is that the inputs are images. Taking advantage of that assumption, they constrain the architecture in a more sensible way.

Similarly to other typical NN architectures, they are consisted of neurons, that connect each other with weights and biases. Each neuron receives an amount of inputs, performs a dot product and is followed by an activation function [35]. The main difference is that the layers of a CNN architecture have neurons arranged in the three-dimensional space: width, height and depth. It should be noted here, that the word "depth" refers to the third dimension of an matrix, and it is irrelevant to the total number of layers in a network. In each layer, the neurons are only connected to a small region of the layer before it. This property generates the ability of CNN to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. Moreover, in every layer they perform an action, called **convolution**, and eventually, by the end of a CNN architecture the full image will be reduced into a single vector of arranged class scores.

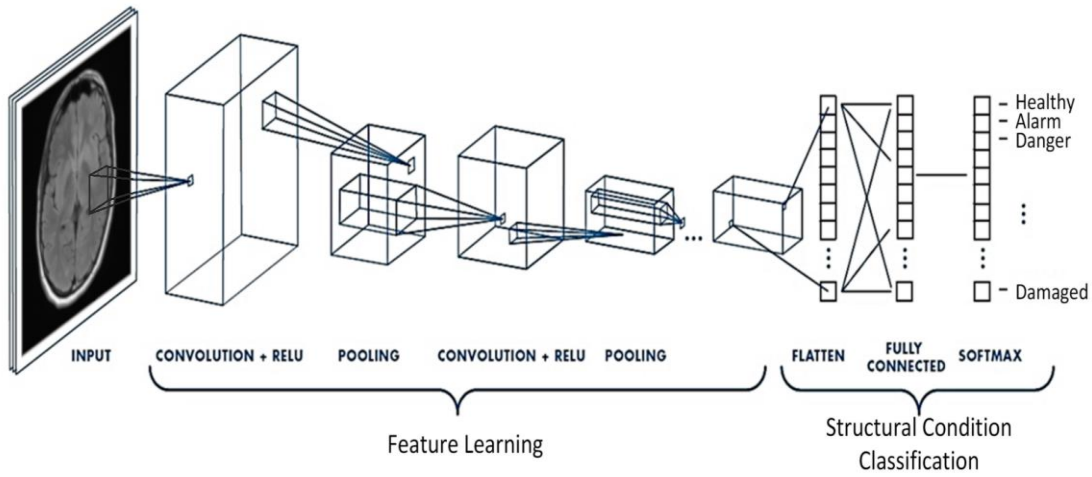

Figure 2. 10: CNN structure [36]

Typically, as it can be observed in Figure 2.10, a simple CNN architecture sequentially includes the following types of layers: **Convolution Layer, Pooling Layer,** and **Fully Connected (FC) Layer**.

The **convolution layer** is the core building block of a CNN architecture. It carries the main portion of the network's computational load. Convolution is a process that extracts relationships from the input image, in a way, that maintains the valuable information, while reduces the image size. It is a mathematical operation that

performs a dot product between two matrices. The first matrix represents the initial image, while the other one represents a window of the receptive field, typically a portion of the image with predefined size. In CNN the input is usually a 3 channel RGB image. The image is represented by a ($width\ x\ height$) matrix, equal to the total amount of its pixels. Every pixel contains 3 values for each one of the three-color planes Red, Green and Blue, between 0 and 255. The second matrix is called **filter,** or **kernel**. Kernel is a matrix of integers, spatially smaller than the input image, but with the same depth.

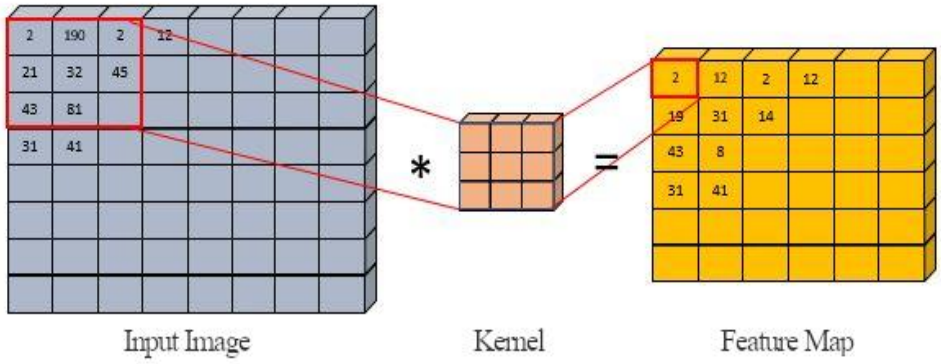

Figure 2. 11: Schematic of Convolution [37]

Sliding across the height and width of the image, each pixel is multiplied by the corresponding value in the kernel and eventually, the result is summed up creating a two-dimensional matrix, known as **feature map** (Figure 2.11). Since image consists of multiple channels, the network performs this process using multiple filters in depth. While the filter is parsing over the input image, it uses a **stride** value, which dictates by how much the filter should slide at each step. When the stride is 1, the filter moves one pixel at a time, when the stride is 2, the filter moves two pixels and so on until it covers the whole image. Considering the kernel has a defined size, sometimes during parsing it doesn't perfectly fit exactly inside the frame, and as a result, the network tends to lose information from pixels around the margins of the image. To overcome this problem, it is convenient adding extra rows or columns consisting only of zeros 0, at all the sides of the image matrix. This idea is known as **padding**.

By the time the feature map is generated properly, it is summed with the bias term and passed through a non-linear activation function. The purpose of the activation function is to introduce non-linearity into our network, with a decision-making function that determines the presence of a particular feature in the feature map.

After convolution layer is completed, in a CNN architecture is often applied an additional reduction on the size of the current image, by a process which is called **pooling**. As it can be shown in Figure 2.12, the pooling layer performs an action of extracting particular values from a region of the image, forming a new matrix. A window (i.e., $3 \, x \, 3$) passes over an image according to a set stride value. At each step, the maximum or the average value within the window is pooled into a smaller matrix. That type is called Max Pooling or Average Pooling respectively. In this way the unneeded information is getting eliminated, counting only the features that are required to know. It is common technique to periodically insert a pooling layer between two or more convolutional layers, while implementing a CNN architecture. It is used to progressively reduce the spatial size of the feature array, the number of total parameters and eventually, the computational complexity of the network.

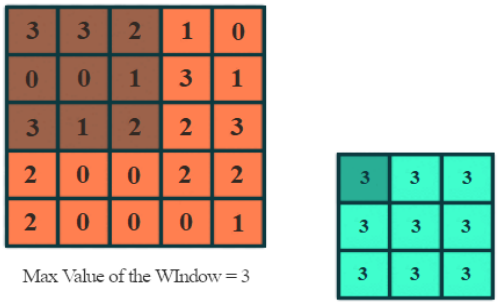

Figure 2. 12: Schematic of Max-Pooling [38]

For more precise feature extraction and better understanding of the input image, a typical CNN architecture, usually, constitutes of numerous blocks of convolutional-pooling layers sequentially.

After various convolution layers and pooling operations, the three-dimensional representation of the features, is eventually, converted into a feature vector that is delivered into a multi-layer regular neural network for classification purposes. That process is called **flattening**. The rows are combined to form a long one-dimension vector. If multiple input layers are present, its rows are also concatenated to form an even longer feature vector.

At the end of the architecture, it can be found a **fully connected** layer. Consisting of a simple structure, neurons in the FC layer have full connectivity with all neurons in the preceding and succeeding layer. It is an ordinary neural network, usually building the last layer of CNN architectures, in order to perform the intended classification or regression task. Over a series of epochs, the network combines the information of the flattened feature map and eventually creates the ability to

determines which features most correlate to a particular class. The output is generated again by an activation function, to classify the probabilities of every image as 0, 1, cat, dog, car, bicycle, tumor, no tumor etc. depending on the nature of the particular problem.

2.7 Overfitting

In Machine Leaning, model performance is evaluated by two important parameters. **Accuracy** and **generalisation**. Accuracy means how well the model generates the right prediction and generalisation means how well the model behaves on new, totally unseen data. Although, Machine Learning models are trained on given training data, as we described, their performance is evaluated on separated data, that is unseen and unknown by the perspective of the network. A model is considered good when it can generalize well from the training data to any data, that it has never seen before, and behaves nearly same way on these two, with the highest possible accuracy.

When it does not generalize well from known to unseen data, then **overfitting** may occur. Overfitting causes the model to perform perfectly, with also high accuracy on training set, while fitting poorly, with low scores on testing set [39]. This happens because a model learns too much details out of training data, including unavoidable noise, insignificant patterns or even random fluctuations. In other words, the model is very complex and strongly influenced by the training data, in the way that it negatively affects its performance on new data. A method to measure the appearance of overfitting on a model, is the usage of a small group of unseen data, named "validation set". We use this data to evaluate the model after every epoch, during the training process. The gap between the training and the validation accuracy indicates the amount of overfitting. Solutions to avoid overfitting might be expansion or augmentation of the data, including methods like image rotation, cropping or flipping and the reduction of the architecture complexity, by decreasing the number of the network's layers, parameters, size, etc.

An efficient technique for reducing overfitting, by preventing complex co-adaptations on training data, is called **dropout**. During training, some neurons are randomly ignored or "dropped out", meaning that these units are not considered during a particular forward or backward pass. This has the effect of making a layer to achieve different connectivity to the prior layer every time. In general, dropout forces a neural network to learn more robust useful features, in conjunction with

many different random subsets of the other neurons. It can be implemented between the most types of layers, such as fully connected layers, convolutional layers, and recurrent layers, at any, or all hidden layers in the network, as well as the input layer. It is not used on the output layer though.

2.8 Semantic Segmentation

In recent years, Convolutional Neural Networks have shown tremendous success specifically in the field of Computer Vision (CV). Computer Vision tasks include methods for processing digital images, understanding and extracting high-dimensional data from the real world, in order to produce numerical or symbolic information. Some of the most significant tasks in this area, sorted by their complexity of implementation, from the simplest to the most complex are: Image Classification, Object Detection and Semantic Segmentation.

Figure 2. 13: (a) Classification (b) Object Detection (c) Semantic Segmentation [40]

Image Classification is a technique, where the network assigns an input image to one specific category (Figure 2.13 (a))**.** Thus, the entire image is classified, as a whole, to one class, i.e., “Cat”. This is one of the core tasks in Computer Vision field, that, despite its simplicity, has a large variety of practical applications.

Object Detection is a technique that allows recognizing objects, such as humans, faces, vehicles, and buildings inside an image or video (Figure 2.13 (b)). Object Detection algorithms typically use extracted features and learning algorithms to recognize and classify a part of the image into a particular class. The location of an object is usually presented by bounding boxes. It is highly used in applications of picture retrieval, item tracking, security, and autonomous vehicle systems.

Instead of assigning a single class to the entire picture, or some part of it, **Semantic Segmentation** is the process of classifying every single pixel of the image to a particular label or a class (Figure 2.13 (c)). The purpose of Semantic Segmentation is to partition a given image into multiple segments, making it easier to analyze it. The various pixels that belong to the same class, are treated as single entity [41]. Some of its primary applications are in autonomous vehicles, human-computer interaction, medical applications, robotics, and photo editing tools. For example, Semantic Segmentation is very popular in self-driving cars and robotics, because the problem there, for the models, is to recognize their surroundings and understand the context of the environment in which they are operating. In this thesis, we work with Semantic Segmentation in brain MRI images. Every image consists of two specific classes, with each pixel being assigned to one of the them. Pixels in "tumor" area have the label 1, while pixels in "no tumor" area have the label 0, as shown in the Figure 2.14.

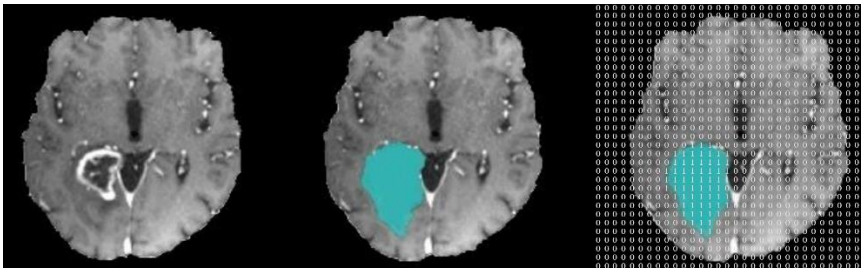

Figure 2. 14: (a) Original MRI scan (b) Segmentation of the tumor area (c) Pixel-Level labelling for Semantic Segmentation

When using CNN for Semantic Segmentation, the output requires an image of the same resolution as the input, unlike a fixed-length vector in the case of Image Classification. In order to achieve that Semantic Segmentation networks are based on an **Encoder-Decoder** structure [42]. Typically, this network involves a CNN architecture, to divide the image, followed by a transposed CNN architecture, which restores and produce the image again. Through the **encoder** part, the network repeatedly, attends to reduce the input's spatial resolution, generating smaller feature maps, where the different patterns can be distinguished more easily. This process is called **downsampling** and consists of a series of convolutional-pooling blocks, along with non-linear activation functions. The initial layers in a Convolutional Neural Network learn low-level features like lines, edges, colours etc., while the deeper layers learn high-level features like faces and objects. The encoded output is then delivered into the **decoder** part. It contains a series of transposed convolutions, where these feature representations increase their dimensionality, through the inverse process. That process is called **upsampling,** producing eventually the output image of the network.

2.9 U-Net

After convolution-pooling operations downsample the resolution of previous layers by summarizing local areas of the image into a single value, the upsample operations perform the invert action, trying to restore the resolution, by distributing a single value into more rows or columns. Since the first layers of the encoder have more information, which later will be diminished, an information loss problem will apparently occur. The **U-Net** architecture proposes the idea of providing that information to every upsampling layer in the decoder, directly from its corresponding downsampling layer in the encoder, as can be seen in the Figure 2.15 below [43]. Enhancing the concept of Convolutional Neural Networks, that described at the previous section, U-Net is one of the most dominant Deep Learning approaches to handle medical image segmentation and detection today.

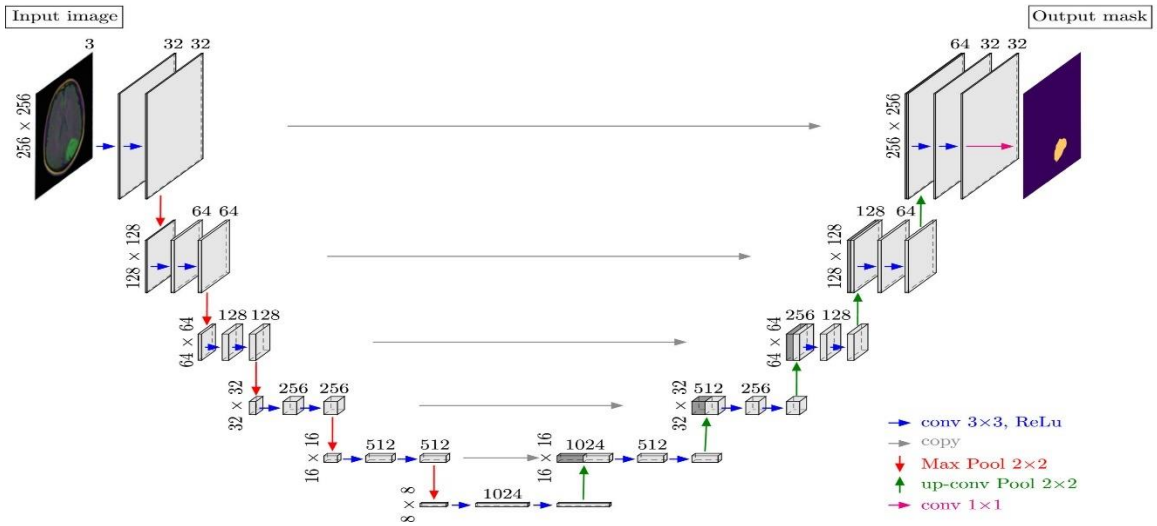

Figure 2. 15: U-Net architecture [44]

According to the above encoder-decoder U-Net architecture (Figure 2.15), this network can be divided into three basic parts:

- The downsampling path consists of 4 blocks, where each block applies two $3x3$ convolutions, followed by $2x2$ max pooling. The size of feature maps is progressively doubled at each pooling layer, after every block, as 64, 128, 256 and so on.
- The horizontal part, or the bottleneck, consists of two $3x3$ convolutions followed by a $2x2$ up-convolution.
- The upsampling path, correspondingly to the downsampling path, also consists of 4 blocks, where each block consists of two $3x3$ convolutions,

followed by $2x2$ upsampling (transpose convolution). The size of feature maps here are divided to the half after every block, 512, 256, 128 and so on.

The important contribution of the U-Net architecture, however, is the shortcut connections. Building skip connections, called **concatenations**, directly from the output of each convolution block to the corresponding input of the transposed-convolution block, it the same level, this method protects the network of losing information during the downsampling process, which can't be easily recovered. In this thesis, we built our model, working on U-Net architecture, as it will be described at the next chapters.

CHAPTER 3

Dataset

The dataset, in Machine Learning, is all the data used as input to the neural network. Machine Learning algorithms learn from data. They find relationships, make decisions and evaluate their performance from the data they're given. Usually, the entire dataset is divided into three subsets, depending on how it is going to be supplied to the network, during the procedure of learning. These categories are Training dataset, Validation dataset and Test dataset.

Training dataset is the data that the respective model receives as input for the purposes of its training. It consists, typically, by the training data, along with a corresponding label for each one of them. For example, in this study, the training data contains MRI images. It also contains a corresponding label, for every image, which is called, mask. The correspondence is that, each pixel of an image is related to the corresponding pixel of its mask.

A small portion of the training dataset is usually separated, forming the **validation dataset**. During training and after every epoch, the model performs predictions on both the training and the validation data. The accuracy of the model's predictions on the training data is called **training accuracy,** while the accuracy on the validation data is called **validation accuracy**. Observing these metrics, someone can inspect the progress of the training, estimate whether the correlation between training data and training labels is accurate and detect mistakes possibly made at the architecture.

After the training has been completed and the model is eventually trained, it gets evaluated, in order to test its performance on unknown data. The **test data** is therefore, a dataset used only to assess the generalization ability of a fully specified model. In particular, the testing data must be totally new and unknown to the model, since it would not make any sense to check its performance on data, that it already knows. During the testing procedure, it demonstrates predictions, that are compared to the real target values, in order to estimate the model's **test accuracy**.

Generally, the **size** of the dataset is apparently, an important issue in Machine Learning. It mainly depends on two key factors: the complexity of the problem and then, the complexity of the selected algorithm. However, these two factors do not indicate exactly, how to choose the appropriate size of the data. In case of a large amount of data, dropouts may be needed, so that the model to be developed properly and overfitting to be avoided. In case of a small dataset, the amount of data may need to be increased, or a more appropriate method may need to be applied, utilizing more efficiently the existing data.

3.1 TCGA Dataset

In this thesis, all the data obtained from The Cancer Genome Atlas (TCGA) dataset, supervised by the National Cancer Institute's Center for Cancer Genomics and the National Human Genome Research Institute funded by the US government [45]. Started in 2005, TCGA is a coordinated project to gather and catalogue genetic mutations responsible for cancer, using genome sequencing and bioinformatics. TCGA program includes genotyping, solid-tumor RNA expression, whole exome sequencing and methylation data, along with other clinical information. The initial idea was an effort to build a research community, focused on connecting cancer phenotypes to genotypes, by providing clinical images and medical information. Since then, the project, molecularly, characterized over 20,000 primary cancer and matched samples from 33 cancer types [46]. About 2.5 petabytes of data that generated through TCGA remain publicly available, for anyone in the research community to use.

A subset of this project, which is used in this thesis, is The Low-Grade Glioma (TCGA-LGG) data collection. The dataset contains brain MRI scans of 110 patients, retrieved from 5 institutions. Thomas Jefferson University (TCGA-CS, 16 patients), Henry Ford Hospital (TCGA-DU, 45 patients), UNC (TCGA-EZ, 1 patient), Case Western (TCGA-FG, 14 patients) and Case Western – St. Joseph’s (TCGA-HT, 34

patients). The entire set is organized into 110 folders, named after each case ID. Each folder contains MRI images of Lower-Grade Glioma together with manual abnormality segmentation masks. Each mask corresponds to one MRI image, as it can be shown in Figure 3.1. Dataset of registered images, together with manual segmentation masks for each case, that used in this study, is available online.

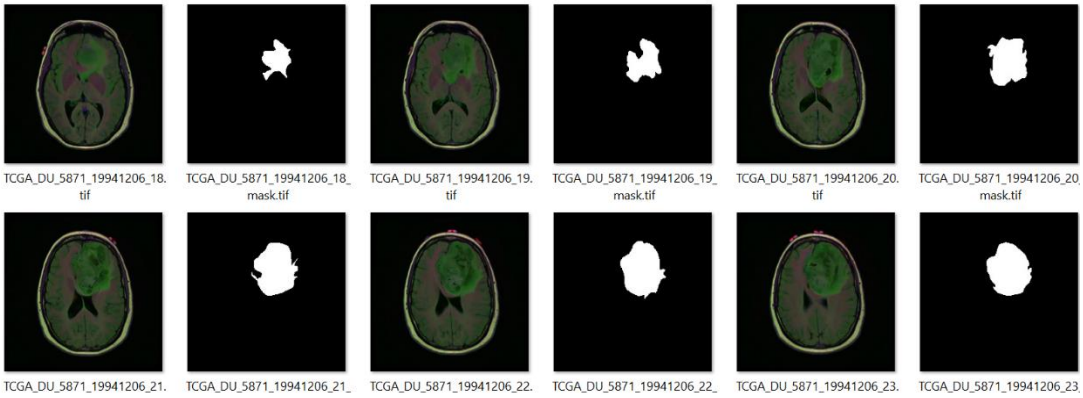

Figure 3. 1: TCGA Dataset: MRI Images and their Masks

3.2 Data Preparation

Apart from the dataset itself, among the most important phases in Machine Learning projects is pre-processing of the data. In general, the better the training data, the better the model will perform. However, the quality of the training data plays a crucial role to the success of a Machine Learning project. Even if, a vast collection of well-structured data has been stored, it might not behave, in a way, that actually works for the model. In other words, the data, in order to be useful for the training process, needs to be properly labelled, under a way, that all the information is translated into valid mathematical values, that can be managed by a neural network.

For this Thesis the MRI images from the entire TCGA dataset combined, in order to create training and testing datasets, with sufficient amount of data. All images from the 110 clinical cases were summarised into the same folder, forming a total dataset of 7.860 images, both scans and masks. Then a software was developed to eliminate the pairs, where the region of the tumor is not appearing clearly or the pairs where the masks were totally black, because that kind of features would produce noise, confusing the performance of the neural network. Creating a parser to review every mask, one by one, only the masks that provide a clear assessment of the extend of the tumor were kept into the training dataset. Eventually, after these changes the

arising dataset consisted of 2384 MRI scans and masks. We separated these files into training dataset, which consists of 1870 files, and the testing dataset consisting of 514 files. Then, the images of the training set divided once more into two folders. A folder containing the MRI scans and another one containing the corresponding masks-targets for each scan. The final overview of the dataset is presented at the graph below:

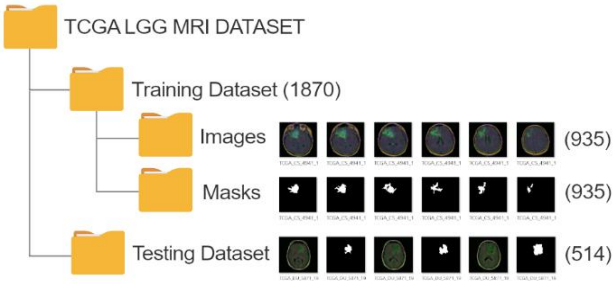

Figure 3. 2: Dataset overview

Additionally, every image's size was reduced to $128\,x\,128$ width and height respectively, because proceeding with the initial size of the images would produce substantial computational complexity during the training phase. Furthermore, all images were properly modified, so that they all have the following format. They were converted into RGB scale, meaning that every pixel contains 3 values, indicating its Red, Green and Blue intensity. Each intensity value is presented on a scale of 0 to 255. Considering a matrix for each colour, an entire picture is represented by three matrices, and when these are combined, they form a tensor. The architecture of the model which was used in this thesis requires a tensor, concluding the entire training data, with fixed input size. Thus, the input of the neural network will be a matrix of the following dimensions:

$$X = input\ matrix = (N, 128, 128, 3)$$

, where N is the entire number of MRI images found in the dataset, $128\,x\,128$ represents the width and height of each image, and 3 is the number of channels for each picture. Note that, this matrix indicates the input features and consists only of the MRI images, not the masks. Now, for each input element must be provided a desired target. Therefore, a second tensor is going to be formed with the following dimensions:

$$Y = target\ matrix = (N, 128, 128, classes)$$

, where N is the entire number of masks found in the dataset and $128x128$ represents the width and height of each mask. The variable "$classes$" refers to the

number of different groups, in which the pixels will be classified. As it was mentioned previously, someone can easily assume that there are 2 groups of pixels. White pixels (with label in RGB scale: $[255, 255, 255]$) will belong to class [1, 0] and otherwise, black pixels (with label in RGB scale: $[0, 0, 0]$) will belong to the class [0, 1]. Thus, every pixel corresponds to a particular vector, indicating its target. After shaping both the input and the target matrix, the dataset has been prepared to feed the neural network.

3.3 Tools

Before proceeding to the implementation of the model, it would be useful to describe the working environment, as well as the tools that used for this study to be accomplished.

The entire code of this project has been written in **Python**. Being a powerful programming language, with great readability, Python is used very commonly today in the field of data science [47]. Considering the very complex concepts of Linear Algebra and Calculus, its environment provides numerous libraries, to support high-level mathematical operations and complicated data calculations. Some of the most common are: Pandas, which contains functions for data manipulation, NumPy, which offers tools for math computations and array management and Matplotlib, which is used to create plots, animations and interactive visualizations.

Created by the Google Brain team, **TensorFlow** is a comprehensive open-source environment of tools, libraries and other resources, that provide fast numerical computation. Although it can be used across a range of tasks, it has a particular focus on training and building deep neural networks. It uses Python to provide a convenient front-end API for building applications, while executing those applications in high-performance C++. TensorFlow allows developers to create data structures, by describing how data flows in the structure, or a series of processing nodes.

Running on the top of TensorFlow, there is a high-level neural network library called **Keras**. Providing high-level API, used for easily building and training models, Keras acts as an interface for the TensorFlow library. Developing Deep Learning models with Keras, facilitates quick and fast prototyping, as well as smooth CPU and GPU operation.

Chapter 4

Implementation

4.1 Network Architecture

In this chapter, we will provide all the technical information about the developed neural network, that for this thesis. We will describe the architecture, the design and the required procedure, in order to, appropriately, accomplish the task of autonomous segmentation of LGG brain tumor in MRI scans. Moreover, several different approaches and implementations of the model will be presented, along with their accuracies, scores and evaluation.

The neural network constructed with the usage of U-Net architecture, consisting of a contracting path on the left side, where the spatial information is reduced while feature information is increased, and an expansive path on the right side, where the feature and spatial information are combined, through a sequence of up-convolutions and concatenations.

Following the architecture of a typical Convolutional Neural Network for downsampling, the contracting path, consists of a repeated application of two 3×3 convolutions, each followed by a rectified linear unit function (ReLU) and a 2×2 max-pooling operation with stride 2. Between them a dropout operation is applied to eliminate a number of neurons. This entire block is repeated at each downsampling step, doubling the number of feature channels every time. On the other side,

the expansive path consists of transposed-convolution layers, performing 2×2 up-convolution, that halves the number of feature channels. Each layer is followed by a concatenation, with the correspondingly cropped feature map, from the contracting path, and two 3×3 convolutions, each followed by a ReLU.

At the final layer a 1×1 convolution and a soft-max function is used, to generate the probabilities and map each feature vector to a desired class.

The input dataset, as described previously, is prepared to contribute in the process of training. The input matrix $X = (N, 128, 128, 3)$ indicates that the first layer of the network consists of $128\ x\ 128\ x\ 3 = 49152$ neurons. The desired output for each image is the target matrix $Y = (N, 128, 128, classes)$. Consisting of numerous hidden layers, the whole network is constructed of 1,941,122 total nodes. The architecture overview of the network is presented below in Figure 4.1:

Architecture	
input (128 x 128 x 3)	deconv2D (2 x 2) - 128
conv2D (3x3) - 16	concatenate
dropout 0.1	conv2D (3 x 3) - 128
conv2D (3 x 3) - 16	dropout 0.2
maxpool (2 x 2)	conv2D (3 x 3) - 128
conv2D (3x3) - 32	deconv2D (2 x 2) - 64
dropout 0.1	concatenate
conv2D (3x3) - 32	conv2D (3 x 3) - 64
maxpool (2 x 2)	dropout 0.2
conv2D (3x3) - 64	conv2D (3 x 3) - 64
dropout 0.2	deconv2D (2 x 2) - 32
conv2D (3x3) - 64	concatenate
maxpool (2 x 2)	conv2D (3 x 3) - 32
conv2D (3x3) - 128	dropout 0.1
dropout 0.2	conv2D (3 x 3) - 32
conv2D (3x3) - 128	deconv2D (2 x 2) - 16
maxpool (2 x 2)	concatenate
conv2D (3x3) - 256	conv2D (3 x 3) - 16
dropout 0.3	dropout 0.1
conv2D (3x3) - 256	conv2D (3 x 3) - 16
	soft-max

Table 4. 1: The implemented architecture with convolution, dropout and pooling layers (a) encoder part (b) decoder part

Before starting the training, a crucial task in Deep Learning problems is setting the **hyperparameters**. Hyperparameters are basically the parameters, whose value is used to control the learning process. They determine how well the model is going to be trained, by tuning the rate of the data flow in the network, the learning rate and the optimization method that will be used. **Learning rate** defines how frequently a network updates its weights. A low learning rate slows down the learning process but converges smoothly, whereas a larger learning rate makes the learning faster but may not converge. Most optimizers have a default learning rate at 0.01 or 0.001, but it also can be manually modified. In addition, it would be inconvenient to pass the entire dataset into the CNN at once. Considering the large memory requirements of a complete iteration of forward and backward propagation, the training data is provided into smaller batches of data, in order to feed the network. Thus, the number of samples given to the network, after which the weight-update occurs, is called **batch size**. A batch size of 32, 64 or 128 is usually a fine value to start experimenting. Furthermore, we define the **validation split** at 0.1, to generate a validation dataset. The network uses the 10% of the training set to estimate the learning process, helping the best parameters to be obtained.

4.2 Implementations

Most of the times in Machine Learning problems, approaching the optimal model implies multiple implementations, setting different values of parameters each time, under the specified architecture and eventually choosing the most efficient. For this thesis experiments, all executions were exposed to the same training dataset that described at the previous chapter, over the U-Net architecture and the categorical cross-entropy. The trained model saved in *.h5* file format to store the model configuration and its weights in a single file, which after used to produce and demonstrate the results on new unseen cases of the testing dataset.

For the **1st implementation** the data provided in batches of size 32. The applied optimization algorithm was Adam optimizer, with a default learning rate, equal to 0.001. After every epoch, the model performance was evaluated on the training and the validation set, calculating loss and accuracy, in order to inspect its convergence, as well as the occurrence of overfitting. It was observed that the loss decreased significantly, during the first epochs and then it appeared to converge. The goal was to stop the training process, when the model starts to show high accuracy on the validation data, but before it largely overfits, which would be represented by a drop of validation accuracy and an increase of training accuracy. Therefore, the appropriate time to stop the training was at 50 epochs, having loss value at 0.41 and

accuracy at 98%. Then the model was exposed to the testing set. The scores of each implementation are presented at the next chapter. Table 4.2 provides a summary of the hyperparameters of the network and their values.

optimizer	Adam
learning rate	0.001
batch	32

Table 4. 2: 1st Implementation's hyperparameters

For the **2nd implementation** an attempt has been made to modify the optimization algorithm, while maintaining the other parameters the same. Particularly, the selected optimizer was the Stochastic Gradient Descent (SGD), with a learning rate at 0.01, while the batch size remained at 32. No other changes have been made. During the training process, the behavior of loss and accuracy needed more time to be stabilized, compared to the previous implementation, reaching finally at 97% accuracy and 0.41 loss after 50 epochs.

optimizer	SGD
learning rate	0.01
batch	32

Table 4. 3: 2nd Implementation's hyperparameters

The **3rd implementation**, and the last one for this thesis, used Adam optimizer again, since it appeared that this algorithm performed quite properly, during the learning procedure. This time the learning rate was 0.01, while the batch size increased to 64. During the training someone could observe a slight drop on accuracy at 96%, compared to previous experiments. The loss, however, increased significantly at 4.29.

optimizer	Adam
learning rate	0.01
batch	64

Table 4. 4: 3rd Implementation's hyperparameters

4.3 Evaluation

After the training procedure is completed and all the weights are finally adjusted, each implementation evaluated, according to its accuracy, specifically using the classification accuracy algorithm. **Classification Accuracy** is commonly the most obvious method to calculate the performance of a model. It is the division of number of correct predictions to the total number predictions made. It works well on the most cases and the most classification problems.

$$accuracy = \frac{No.of\ Correct\ Predictions}{Total\ No.of\ Predictions\ Made}$$

However, the problem arises, when the task of the classifier is to assign all the pixels in a particular class and the classes are not equally distributed into the picture. For example, at the most cases in this thesis, the "tumor" class distribution, is much smaller, compared to the distribution of the "background" inside the picture. This is highly possible and quite an ordinary scenario, as the tumor affects a small area of the whole brain most of the times. Now, making the assumption, that the area of the tumor is, let's say, only 2% of the entire image, and using a hypothetical classifier, which mistakenly always predicts "background" for every pixel, it is quite obvious that this classifier will not recognize any tumor pixel at all. For this binary problem, the classification accuracy method would estimate that, the performance of this classifier would be 98%. This sounds like an amazing score. However, considering that. in biomedical applications the cost of failing to diagnose a fatal disease is enormous, this is definitely, not the best value that describes the classification results.

In this thesis, an alternative algorithm to accuracy was preferred as scoring metric, the **Jaccard index**. The Jaccard index, also called the IoU score (Intersection over Union) is defined as the intersection of two sets defined by their union. The basic idea is to regard the image masks as sets. These sets can overlap within the picture. The mathematical expression for IoU score is provided below.

$$Jacc(y, \hat{y}) = \frac{\mathrm{y} \cap \hat{y}}{\mathrm{y} \cup \hat{y}} = \frac{TP}{TP + FP + FN}$$

, where y is the prediction, $\hat{y}$ the target of a pixel, TP the true positive, FP the false positive and FN the false negative values. If both masks are completely identical, both sets have exactly the same size and do overlap to 100%, so that intersection equals union. In this case, the IoU score is 1 and optimal. On the other hand, if the predicted mask is shifted or changed in size compared to the original mask, then the

union gets bigger than the intersection and the IoU score decreases. Regarding the previous example again, with the tumor pixels being the positive and background be the negative values, the classifier labels every pixel in the image as “background”. As a result, the sum of true positives (TP) must be 0. Therefore, the resulting Jaccard index is also 0. This score indicates that, the Jaccard index undoubtedly describes the result of this classifier more precisely than accuracy does.

Chapter 5

Results

5.1 Results

The implementations operated in this thesis, followed different methods to train a Deep Learning network, in order to successfully perform Semantic Segmentation on LGG brain tumor. At the previous chapter, three different approaches described and implemented. As we specified, the optimization algorithm varied in each implementation, as well as the batch size and the learning rate. All the trained models were exposed on the testing dataset, to evaluate their performance and, eventually, their ability for generalization. As the Table 5.1 demonstrates, the first attempt, appeared to perform quite decently, achieving a Jaccard index of 75%. This approach was the highest of this study. Despite the very high accuracy during training, the score of the second attempt, reduced at 68%, operating on the testing data. Finally, the third model achieved a Jaccard score at 73%, which was quite unexpected, considering the large loss value during training.

1st implementation	Jaccard score: 75%
2nd implementation	Jaccard score: 68%
3rd implementation	Jaccard score: 73%

Table 5. 1: Results

In addition, to reveal even practically which model appears to be the most suitable for the segmentation task, we operated predictions of the three approaches, in the

same, randomly chosen, samples of the testing data. For comparison purposes, several indicative demonstrations are going to be presented and discussed below.

The Figure 5.1 shows the predictions of the three models, alongside with the original MRI scan and the original mask. Apparently, the more a prediction resembles to the original mask-target, the more precise it is, and as a consequence, the more appropriate the model behaves.

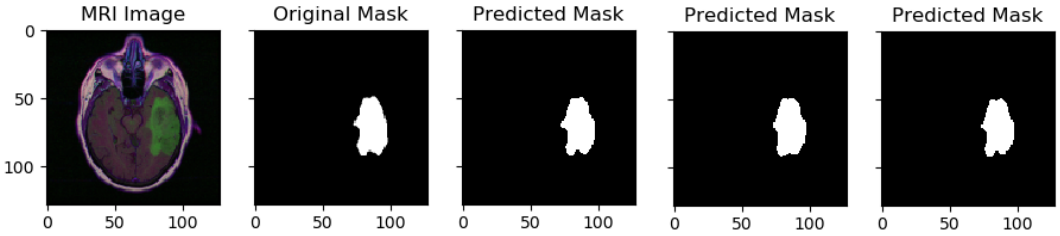

Figure 5. 1: Results demonstration (a) Original MRI image (b) Original Mask (c) Adam - 32 - 0.001 (d) SGD - 32 - 0.01 (e) Adam - 64 - 0.01

Observing the outputs, it can be assumed, that all the models detected the existence of the abnormality quite efficiently, on this particular case. Every approach achieved a great pixel-level distinguishment, without missing sharp edges or any other information.

Moving on, to a more challenging example, in Figure 5.2 the divergence among the three approaches is appearing more clearly. As it can be shown, the 1st implementation performed surprisingly well, even around the very fine details, demonstrating a prediction, very close to the original mask. The 3rd implementation, also achieved a good result. At the same time, Stochastic Gradient Descent appeared to perform with difficulties in this particular problem, being able to detect the outline of the tumor area, but, missing entirely the details inside it.

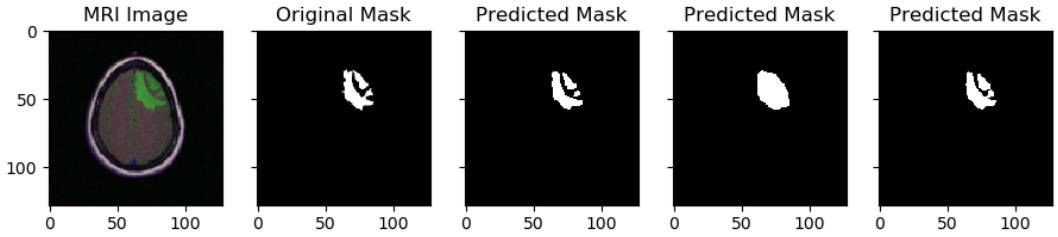

Figure 5. 2: Results demonstration (a) Original MRI image (b) Original Mask (c) Adam - 32 - 0.001 (d) SGD - 32 - 0.01 (e) Adam - 64 - 0.01

Another sample that, seemingly consisting of sharp demanding edges, requiring very sensitive approach is demonstrated in Figure 5.3. Here, the 1st implementation appeared to have an appropriate behavior, performing again the most similar to the target, segmentation of the tumor area. The other two implementations made an effort to approximate the desired segmentation, but as it can be observed, the distribution of the tumor pixels became blurry over some points, smoothing sharp edges and details.

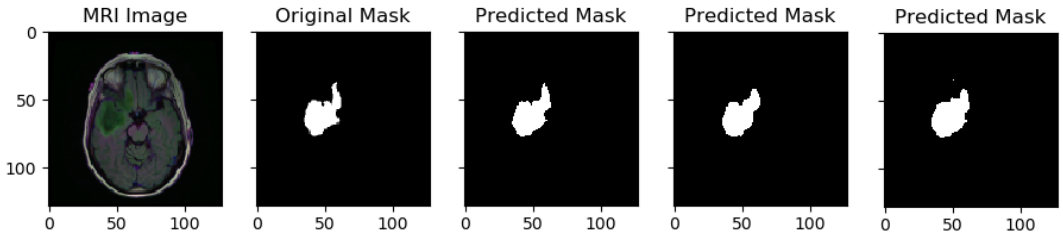

Figure 5. 3: Results demonstration (a) Original MRI image (b) Original Mask (c) Adam - 32 - 0.001 (d) SGD - 32 - 0.01 (e) Adam - 64 - 0.01

Considering the results of each method during evaluating and testing phases, someone has to admit that, the usage of Adam optimizer, along with learning rate 0.001 and batch size 32, over a U-Net architecture, appeared to be the most successful implementation of this thesis. Covering its learning behaviour, we have to also provide the plots of its training and evaluation process.

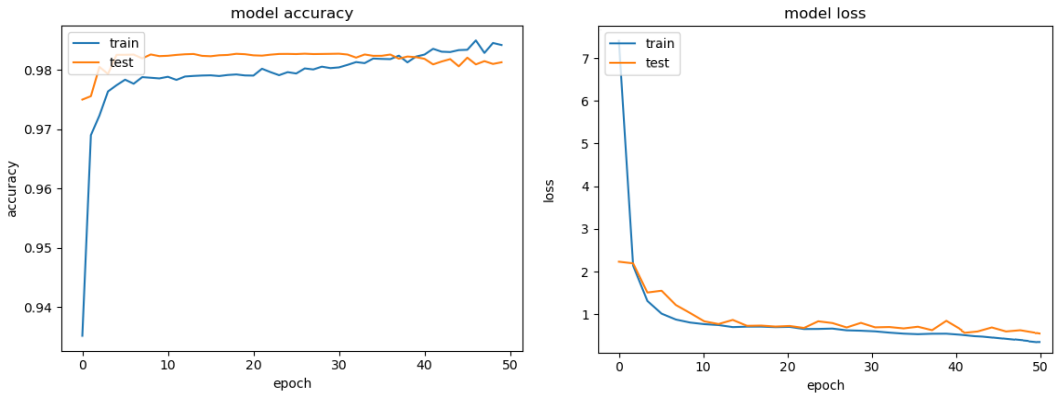

Figure 5. 4: Adam - 32 - 0.001 accuracy and loss plots

Moreover, and just for demonstration purposes, we used this particular model further, in more MRI scans, to inspect its performance on detecting and isolating

appropriate the tumor area. The images again, belong to the unknown, from the perspective of the model, data.

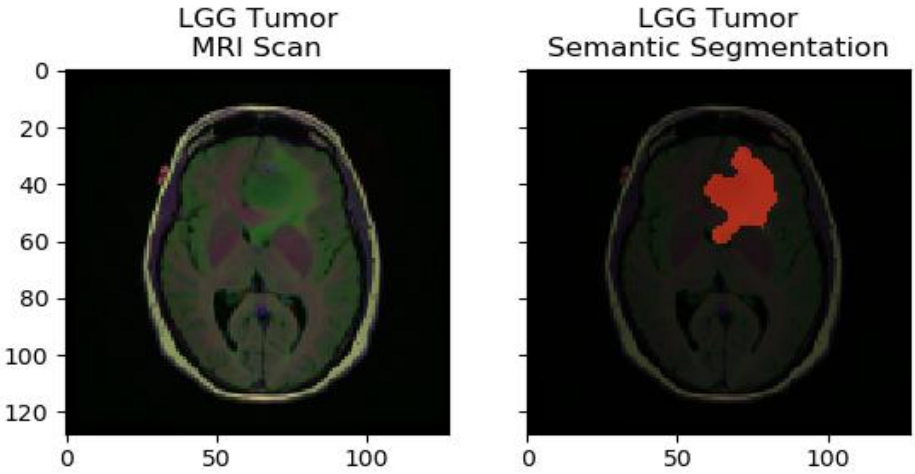

Figure 5. 5: Adam - 32 - 0.001 LGG segmentation

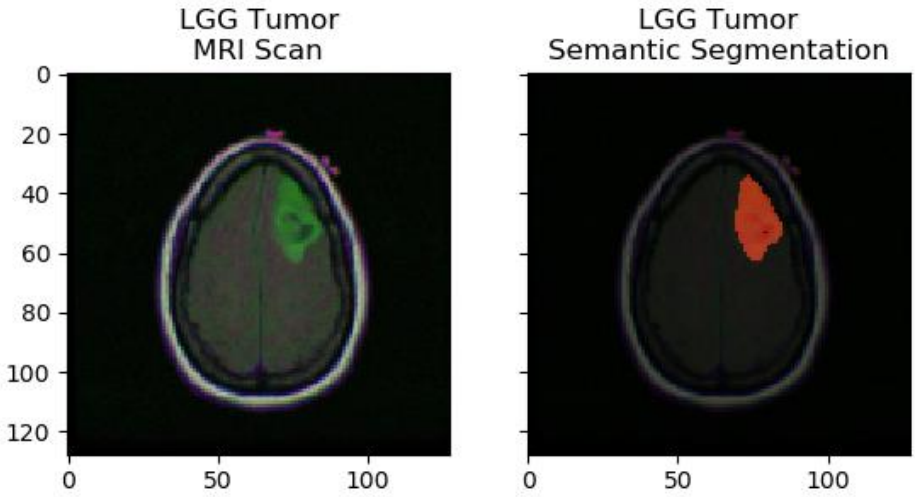

Figure 5. 6: Adam - 32 - 0.001 LGG segmentation

5.2 Conclusion

Reaching overall satisfactory Jaccard scores, the models showed their ability to perform autonomous segmentation of Low-Grade-Glioma from a given MRI scan. Through the experiments for this thesis, we tried different training approaches of Deep Learning models, investigating if these specific approaches would be able to meet the requirements of this particular task. Observing the scores and their behaviour, we can assume that, the trained models accomplished quite efficiently the

task of brain tumor segmentation. However, since there are still possibilities of error, neither doctors nor patients can undoubtedly rely on these predictions for the time being. Combining these models with medical examinations can lead to more accurate diagnosis. Concluding, the implemented methods in this thesis, shown pleasing, but not superlative results. Compared to Related Work researches, that mentioned in Chapter 1, this thesis achieved a less competitive overall performance. The purpose of this work was to show the ability of autonomous segmentation of Low-Grade Glioma using Deep Learning networks, such as Convolutional Neural Networks and U-Net architectures, which could contribute for future research and investigation.

The entire Python code of this thesis is online and available at GitHub. It can be found through the following link:
https://github.com/bachtses/Brain-Tumor-Segmentation-TCGA

5.3 Future Work

In the future, we aim to develop and restructure the implemented architecture, to increase accuracy, obtain more data for training, try different optimization algorithms, as well as loss functions, and eventually improve the performance of the model. Additionally, complementary information from other imaging modalities, such as Positron Emission Tomography (PET) and Magnetic Resonance Spectroscopy (MRS) may improve the current methods, eventually, leading to the development of clinically acceptable automatic Glioma segmentation methods for better diagnosis. The ideal scenario, however, is to focus Deep Learning techniques on early stages diagnosis, in order to develop the ability to predict the future existence of an abnormality, based on the tumor evolution of other cases. Under these circumstances, the ultimate goal is to make strong prediction models, that can safely help biomedical applications in cancer prediction, providing the best chance for a cure.

Bibliography

[1] Turing, A.M., 2009. Computing machinery and intelligence. In Parsing the Turing test (pp. 23-65). Springer, Dordrecht.

[2] Ashby, W.R., 1956. Automata Studies: Annals of Mathematics Studies. Number 34 (No. 34). Princeton University Press.

[3] Manning. C., 2020. Artificial Intelligence Definitions. Stanford University.

[4] Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), p.386.

[5] Widrow, B. and Hoff, M.E., 1960. Adaptive switching circuits (No. TR-1553-1). Stanford Univ Ca Stanford Electronics Labs.

[6] LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. nature, 521(7553), pp.436-444.

[7] Oppermann A. (2019). Artificial Intelligence vs. Machine Learning vs. Deep Learning [Online]. Available at: https://towardsdatascience.com/artificial-intelligence-vs-machine-learning-vs-deep-learning-2210ba8cc4ac (Accessed: 11 January 2021).

[8] Sinha, K. and Sinha, G.R., 2014, March. Efficient segmentation methods for tumor detection in MRI images. In 2014 IEEE Students' Conference on Electrical, Electronics and Computer Science (pp. 1-6). IEEE.

[9] American Association of Neurological Surgeons, 2019, November. Classification of Brain Tumours. Available online: https://www.aans.org/en/Media/Classifications-of-Brain-Tumors.

[10] Ostrom, Q.T., Bauchet, L., Davis, F.G., Deltour, I., Fisher, J.L., Langer, C.E., Pekmezci, M., Schwartzbaum, J.A., Turner, M.C., Walsh, K.M. and Wrensch, M.R., 2014. The epidemiology of glioma in adults: a “state of the science” review. Neuro-oncology, 16(7), pp.896-913.

[11] American Brain Tumor Association, http://www.abta.org.

[12] Mandonnet, E., Delattre, J.Y., Tanguy, M.L., Swanson, K.R., Carpentier, A.F., Duffau, H., Cornu, P., Van Effenterre, R., Alvord Jr, E.C. and Capelle, L., 2003. Continuous growth of mean tumor diameter in a subset of grade II gliomas. Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society, 53(4), pp.524-528.

[13] Ostrom, Q.T., Gittleman, H., Fulop, J., Liu, M., Blanda, R., Kromer, C., Wolinsky, Y., Kruchko, C. and Barnholtz-Sloan, J.S., 2015. CBTRUS statistical report: primary brain and central nervous system tumors diagnosed in the United States in 2008-2012. Neuro-oncology, 17(suppl_4), pp.iv1-iv62.

[14] EMC with Research & Analysis by IDC. 2014. The Digital Universe Driving Data Growth in Healthcare. https://www.emc.com/analyst-report/digital-universe-healthcare-vertical-report-ar.pdf.

[15] German Economic Institute. https://www.iwkoeln.de/en/topics/financial-and-social-policy/the-pharmaceutical-and-medical-technology-industry.html

[16] Urban, G., Bendszus, M., Hamprecht, F. and Kleesiek, J., 2014. Multi-modal Brain Tumor Segmentation using Deep Convolutional Neural Networks.

[17] Zikic, D., Ioannou, Y., Brown, M. and Criminisi, A., 2014. Segmentation of brain tumor tissues with convolutional neural networks. Proceedings MICCAI-BRATS, pp.36-39.

[18] Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.M. and Larochelle, H., 2017. Brain tumor segmentation with deep neural networks. Medical image analysis, 35, pp.18-31.

[19] Pereira, Sérgio, Adriano Pinto, Victor Alves, and Carlos A. Silva. "Brain tumor segmentation using convolutional neural networks in MRI images." IEEE transactions on medical imaging 35, no. 5 (2016): 1240-1251.

[20] Chandra A. (2018). McCulloch-Pitts Neuron - Mankind’s First Mathematical Model of a Biological Neuron [Online]. Available at: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[21] Zhang, J., 2019. Basic Neural Units of the Brain: Neurons, Synapses and Action Potential. arXiv preprint arXiv:1906.01703.

[22] Marais, F. (2019). EE Publishers Available at: https://www.ee.co.za/article/application-of-machine-learning-algorithms-in-boiler-plant-root-cause-analysis.html

[23] Chawdary D. (2020). How to Build and Train Your First Neural Network Available online: https://medium.com/towards-artificial-intelligence/how-to-build-and-train-your-first-neural-network-9a07d020c4bb

[24] Zell, A., 1994. Simulation neuronaler netze (Vol. 1, No. 5.3). Bonn: Addison-Wesley.

[25] Sherstinsky, A., 2020. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. Physica D: Nonlinear Phenomena, 404, p.132306.

[26] Leijnen, S. and Veen, F.V., 2020. The Neural Network Zoo. In *Multidisciplinary Digital Publishing Institute Proceedings* (Vol. 47, No. 1, p. 9).

[27] Rumelhart, D.E., Hinton, G.E. and Williams, R.J., 1986. Learning representations by back-propagating errors. nature, 323(6088), pp.533-536.

[28] LeCun, Y., Touresky, D., Hinton, G. and Sejnowski, T., 1988, June. A theoretical framework for back-propagation. In Proceedings of the 1988 connectionist models summer school (Vol. 1, pp. 21-28). CMU, Pittsburgh, Pa: Morgan Kaufmann.

[29] Kathuria A. (2018). Intro to optimization in deep learning: Gradient Descent https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

[30] Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

[31] McCracken C. (2018). Deep Neural Networks: Choosing a Learning Rate https://medium.com/@colemccracken/deep-neural-networks-choosing-a-learning-rate-172b97ef459

[32] Ward, R., Wu, X. and Bottou, L., 2019, May. AdaGrad stepsizes: Sharp convergence over nonconvex landscapes. In International Conference on Machine Learning (pp. 6677-6686). PMLR.

[33] Zeiler, M.D., 2012. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[34] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[35] Valueva, M.V., Nagornov, N.N., Lyakhov, P.A., Valuev, G.V. and Chervyakov, N.I., 2020. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. Mathematics and Computers in Simulation, 177, pp.232-243.

[36] Tabian J., Fu H. and Khodaei Z.S., (2019). A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures https://www.mdpi.com/1424-8220/19/22/4933

[37] DataTechNotes., (2018). https://www.datatechnotes.com/2018/09/image-convolution-example-in-r.html

[38] Shafkat I., (2018). Intuitively Understanding Convolutions for Deep Learning https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[39] Hawkins, D.M., 2004. The problem of overfitting. Journal of chemical information and computer sciences, 44(1), pp.1-12.

[40] ODSC - Open Data Science., (2020). Using the CNN Architecture in Image Processing https://medium.com/@ODSC/using-the-cnn-architecture-in-image-processing-65b9eb032bdc

[41] Long, J., Shelhamer, E. and Darrell, T., 2015. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[42] Xing, Y., Zhong, L. and Zhong, X., 2020. An Encoder-Decoder Network Based FCN Architecture for Semantic Segmentation. Wireless Communications and Mobile Computing, 2020.

[43] Ronneberger, O., Fischer, P. and Brox, T., 2015, October. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

[44] ReasearchGate., (2018). https://www.researchgate.net/figure/Convolutional-neural-network-CNN-architecture-based-on-UNET-Ronneberger-et-al_fig2_323597886

[45] The Cancer Genome Atlas. Accessed 10 August 2020. Available from: http://cancergenome.nih.gov/.

[46] Gao, G.F., Parker, J.S., Reynolds, S.M., Silva, T.C., Wang, L.B., Zhou, W., Akbani, R., Bailey, M., Balu, S., Berman, B.P. and Brooks, D., 2019. Before and after: comparison of legacy and harmonized TCGA genomic data commons' data. Cell systems, 9(1), pp.24-34.

[47] The python tutorial. Accessed: 05 September 2020. https://docs.python.org/3/tutorial/index.