Automation and Industrial Informatics Department

Faculty of Automatic Control and Computers
Polytechnic University of Bucharest

DISSERTATION THESIS

ML services in healthcare: development of a companion application to help medical staff in more efficient segmentation of brain tumors in MRI images

Coordinator
Prof.univ.dr.ing. Liliana Dobrică

Graduate
Cristinel-Adrian Popescu

2022

Table of Contents

1. Introduction

A central theme in machine learning is that of sequential decision-making. This is the mission to decide, from experience, the sequence of actions to be taken in an uncertain environment in order to achieve certain objectives. A series of sequential steps in decision-making tasks cover a wide range of possible applications with the potential to have an impact in areas such as robotics, healthcare, smart grids, finance or self-driving cars are just a few of them. Machine learning (abbreviated as ML) provides automated methods that can detect patterns in data and use them to perform certain tasks. In addition to learning by consolidation, two essential types of machine learning tasks can be considered: supervised learning, the task of deducing a classification, or regression from labeled training data, and the opposite: unsupervised learning being the task of drawing inferences from datasets consisting of unlabeled input data. During the last few years, computer software applications have gone a dramatic transformation from simple data processing to machine learning and in some cases artificial intelligence, thanks to the availability and accessibility of a huge volume of data collected through sensors and the internet. The idea of machine learning demonstrates and grows the fact that the computational system has the ability to improve itself with the advancement of time.

This study paper targets machine learning, healthcare services, and computer vision domains to explore and analytically evaluate an application in the healthcare computer vision domain using various deep learning and machine learning techniques like neural networks and classification. The automatic segmentation of brain tumors remains a difficult job. One of the reasons is that tumor features such as size, shape, and location are unknown until the tumor's progress through time is explored and data from earlier image scans are accessible. All of the attributes stated are unknown if simply the independent scanning is used. As a result, typical pattern recognition approaches that rely on such qualities and are commonly utilized for item feature extraction in this medical and real-world imaging.
Other knowledge, such as the anatomy of a healthy human brain or tumor manifestation, particularly magnetic resonance imaging, can be utilized. This is an advantage over picture object detection, such as people or automobile recognition, when the hue and backdrop scene alter. There is a growing interest in building such algorithms, and the automated brain tumor segmentation challenge, in particular, has nowadays attracted numerous computer vision research organizations. Because of the variety of brain tumor forms and their expression in Magnetic Resonance Imaging (MRI), most cutting-edge approaches either focus on the most prevalent tumor type, e.g., gliomas or require a specialized training database to deal with a certain tumor type. Because various brain tumor segmentation algorithms rely on different picture information, the divisions utilized in this chapter will be as follows:

threshold-based learn and adapt on the absolute difference between a brain tumor and surrounding tissues; region-based methods look for connected regions of voxels with similar properties; contour-based methods look for edges between a brain tumor and surrounding tissues; classification or clustering methods use voxel-wise intensity and texture features; and atlas-based methods use prior knowledge about the healthy brain. However, the distinction is not always evident because suggested solutions frequently share characteristics and integrate many approaches. In recent years, the majority of cutting-edge approaches have relied on atlas guiding and categorization. For this purpose, a number of classification methods have been utilized, including Support Vector Machine (SVM), Neural Networks (NNs) mostly convolutionary, derived for example U-Net model, Random Forest (RF), and K-Nearest Neighbors (KNN). With the rise in popularity of deep learning, a number of NN-based approaches began to emerge. The independent categorization of a single pixel (voxel) is a typical feature of most classification-based algorithms. Besides from tumor segmentation approaches, there are only a few ways for detecting the presence of a tumor and its approximate location. The purpose of these approaches is not to delineate the tumor boundaries precisely, but rather to make a quick conclusion about whether the tumor is there and where it is located. These strategies might be used to provide an initial estimate of the tumor contour, which is required by some segmentation algorithms.

Since the demand for digital diagnosis has increased in recent decades, artificial intelligence and its two subdivisions: deep and machine learning in medical image analysis have become a big topic. U-Net is a popular deep learning (DL) approach for magnetic resonance imaging (MRI) data, with strong segmentation results reported. Annotated ground truth data is commonly utilized in supervised training. DL techniques for segmenting brain tumors from MRIs often need annotated tumors from medical specialists, which is a time-consuming process. To address this issue, this thesis develops a unique technique that uses a multi-stream U-Net as the input of a foreground tumor region and a background tissue area given by two elliptical bounding boxes. Following that, a limited set of annotated tumor data, 369 cases divided into 250 cases for training the model, 45 cases for testing the model, and 74 cases for validation phase, were used for the whole model.

The goal of this thesis is to create and build a back-end application like a jupyter notebook python-based neural network model that performs automated segmentation of brain tumors and patient survival prediction using classifiers/algorithms such as K-Nearest Neighbors, Support Vector Machine, and Random Forest with efficient and non-exhaustive parameters adjusted via grid-search. Experiments will be carried out to evaluate the performance of various network designs. The essential building components of the U-Net structure will be revealed and deployed to accomplish this. For this research, there will be no attempts to build a fully working GUI program but in the near future, there will be attempts to make a fully functional application that can provide a complete user experience (UX). The initiative will instead concentrate on building code to train a neural network model to segment brain tumors. To focus our attention, we shall examine a U-Net structure using

convolutions. Because computational resources are restricted and time-restricted in the case of GPU or TPU-accelerator in the Kaggle Code cloud platform, the focus will be on implementing a workable solution using the computational resources that are available to us. The brain tumor datasets utilized in this study are confined to the semi-public MICCAI Brain Tumor Segmentation dataset from the 2020 competition.

2. Computer Vision in Healthcare

This chapter serves as a summary of current developments and trends. It intends to offer a brief overview of the major families of techniques, their applications, and limits.
For several years, with the advancement of technology, medical imaging has made huge steps obtaining a reliable computer-aided diagnostic approach for brain tumor segmentation has been a difficult topic since manually labeling brain MRIs to segment malformations is time consuming and wasteful. Many strategies have been presented to solve the problem of brain tumor segmentation. This chapter will cover important computer vision techniques, as well as application cases of computer vision in medical imaging, and will study various approaches by categorizing them as unsupervised and supervised methods. Methods for pre-processing and post-processing will also be explored. Image manipulation refers to a variety of processing techniques that are referred to collectively as digital imaging. Image synthesis or computer graphics entails giving a value to each of the aforementioned pixels or voxels, with the ultimate result being a 2D or 3D image having some semantic or visual information. In general, image synthesis is a parameter of image transformation. These parameters might be mathematical functions, as in scientific computing, or they can be a set of anatomical objects represented as surface or volumetric models, or as a parametric description, as in medical imaging [6]. Image analysis, on the other hand, is a picture-to-parameters transformation that begins with pictures in which each feature has a known value. This domain is often divided into image processing, image analysis, and computer vision or image understanding, and scene analysis.

In the healthcare sector, image analysis a subdomain of computer vision is commonly used. There are several reasons why digital image processing systems are used in medicine [6]:

- Newer techniques and multimodal analysis: with the prospect of investigating new imaging modalities, which may lead to significant anatomical or functional explanations, image analysis will facilitate the integrated assessment of data from several modalities;

- Morphometry: the use of computerized approaches allows for greater precision and reproducibility, resulting in increased objectivity in the measurement of morphometry parameters such as size, surface, circumference, and volume;

- Better interpretation: the new imaging modalities' sensitivity, along with the capability of modern visualization techniques, allows for more precise diagnosis than traditional exploratory procedures;

- More precise prediction: as a result, doctors can provide more finely tailored medical treatment, such as lower dosages in radiation therapy or more exact placement in brain surgery;

- Process automation: from biological specimen screening to vision-guided surgery, many medical processes can benefit from the dependability afforded by automated processing;

- Understanding volume data: detecting structures in nifty volume data is not a natural visual job that will benefit from automated processing and representation.

Medical pictures are widely utilized in oncology for tumor diagnosis, therapy planning, and monitoring. Oncologists use photos to find tumors and evaluate their various features. Depending on the job, such as the search for metastases, radiation planning, and the location of interest, the brain, several forms of medical imaging are utilized.

Imaging techniques that are routinely employed include computed tomography (CT), magnetic resonance imaging (MRI), and positron emission (PET). Positron emission tomography is a technique that involves injecting a radioactive tracer, like fludeoxyglucose, into the patient's blood in order to examine the metabolism of various tissues [29].

Because cancer cells require a large amount of glucose to divide, tumoral tissues may be recognized by their absorption of energy of the radioactive tracer. PET scans are very beneficial for tumor diagnosis and staging, identifying cancer metastases, and monitoring the effects of therapy. Unfortunately, caused by physical constraints, PET scans often have far lesser pixel density than MRI and CT scans. The absorption of X-rays by various tissues in the body is measured using computed tomography. Radiation is emitted from various angles to create a sequence of 2D radiography pictures of which a 3D image scan is generated. Even while CT scans have higher spatial resolution than MRI, they have much lower contrast between soft tissues like those seen in the brain. Furthermore, X-ray radiation may cause cancer by disrupting the DNA of bodily cells [29]. The detection of signals released by the nuclear magnetic resonance of atoms in the body is the basis for MRI acquisition. The observed signal is often created by hydrogen protons [28], which are abundant in the human body (fat and water). The atoms are placed in a high magnetic field and then disrupted by a radio wave, a process known as pulse sequence. Different contrasts are created by varying different parameters of the pulse sequence and pulsed-field gradients, which correspond to particular MRI sequences [31]. T1, T2, and FLAIR MRI sequences are routinely utilized for brain

tumor imaging (T2 with suppression of fluids). T1 is frequently acquired following the administration of a gadolinium-based contrast agent into the patient's blood, specifically to highlight the tumor or the formation of new vascular networks by the tumor [34]. Magnetic resonance imaging is very useful for visualizing brain tumors and other vital organs. They provide a strong contrast between soft tissues in the brain (as compared to other forms of imaging), and the employment of multiple MRI sequences allows for the identification of distinct tumoral sections (edema, tumor, necrosis).

2.1 Analysis Pipeline

Medical imaging equipment typically offers 2D data, but advanced sensors may directly acquire 3D data. The data can then be viewed or analyzed using 3D image synthesis methods or 2D or 3D image analysis and computer vision algorithms. The processing stages necessary to execute an analysis are done sequentially, giving rise to the concept of an analysis flow, that circulates the input data.

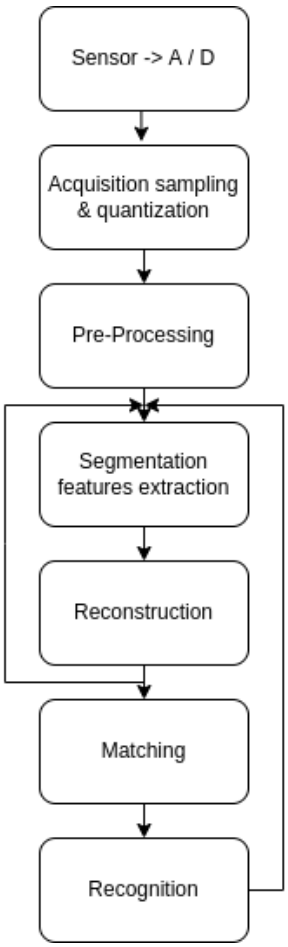


Figure 1: The general image analysis pipeline.

The following steps comprise the general image analysis and computer vision pipeline:

- Image acquisition, from sensors' analog data to digital signal building the native image: this initial stage comprises all processing steps necessary to provide intensity data, such as reconstruction from projections if necessary;

- Pre-Processing: this stage comprises data changes that are either conducted routinely on all shots for denoising or performed dynamically by an observer for improvement;

- Segmentation: the goal of this stage of processing is to divide the picture into constituent components, such as areas and edges in order to extract key features;

- Data reconstruction: a succession of m-dimensional data is paired to generate n-dimensional information, m < n, though 3D rendering is not included; the traditional way to illustrate matching data from 2D slices to generate a 3D volumetric representation;

- Matching: a sequence of n-dimensional data is recorded to combine information; this may be done to data from multiple sources or the same source at specific times;

- Recognition: the identification of items existing in the scene, which is frequently but not always the end goal of the examination.

Variations from this unidirectional flow of information are conceivable, as seen in Figure 1. Following an n-D reconstruction, the n-D picture can be segmented once again. So that [4], after recognizing some or all of the parts in the image, the entire processing step may be repeated using the additional knowledge supplied by the previously retrieved information. Although not yet widely used in medical imaging, such top-down information is rapidly being used in theory or commercial computer vision.

Some common medical imaging applications may be deconstructed using this very general approach as follows:

a. Enhancement (for example, contrast modification, pseudo-coloring): preprocessing >> 2D or 3D visualization through image synthesis;
b. Morphometry: preprocessing >> feature extraction through segmentation >> morphometric metrics;
c. Classification: preprocessing >> feature extraction >> recognition using morphological features;
d. 3D imaging and diagnosis for radiotherapy and surgery: 2D or 3D data capture >> segmentation >> parameter estimation >> assembly of 3D model >> display the results;
e. Multimodality functional imaging consists of the following steps: m-D acquisition >> segmentation >> estimation >> n-D reconstruction >> matching >> visual representation;
f. Data coding and storage: acquisition of 2D or 3D data >> preprocessing >> approximation.

Relationships with data augmentation, though that chapter focused on image analysis and computer vision, scene composition is closely related to these fields in many ways. The data visualization itself is the most obvious example, the doctor's computer will be mostly a display/monitoring device, displaying the findings of the reconstruction or simply analysis. Aside from this apparent example, numerous computer graphics ideas are frequently used in scene analysis. Of course, the opposite applies, but it is less relevant in this situation. The common playground for image analysis and modeling is the way humans see things visually, computer visualization is clearly focused on perception. Computer vision is clearly focused on perception. The mechanics of display undoubtedly has an impact on what is perceived, but what is truly important seems to be the internal representation where a particular image relates and strives to reproduce in the user. In the example of a doctor making a diagnosis based on a 3D generated image, integrating information from pattern recognition might aid in the creation of analysis and synthesis algorithms and would more powerfully express a conclusion. Models for lighting and reflection, geometric transformations, contour and surface estimation, algorithms for region segmentation, and, especially in 3D medical imaging, geometrical modeling are all significant parametric ideas in image analysis.

Fourier transforms are being used to characterize data or linear functions spectrally, to construct filters, to evaluate periodic structures, and to solve sampling and interpolation difficulties. It is also the foundation of projection algorithms for recreating slices. Correlation, which evaluates the degree of connection between a pattern and a resulted scan (MRI or CT image), is used for potentially separating previously known forms or for alignment. The matching procedure is performed in either the image or transform domain, based on the scale of the pattern. Thus, the well-known Hough transform [20] for recognizing sharp lines or circles may be classified as a pattern-matching approach. The mathematical approach to many image analysis issues is based on the morphological paradigm, which is an approach to linear techniques. The underlying notion behind mathematical morphology is similar to pattern matching: a certain pattern known as a structuring element is placed on each picture pixel. A logical test is done at each point to determine if the image is physically similar or not to this structural element. Various processes are carried out depending on the severity of the test [7]. Dilatation, degradation, releasing erosion followed by dilation, which removes tiny items, and closure are the fundamental and most widely employed processes: dilatation then erosion, which fills holes appending items). A set of morphological procedures based on those fundamental processes enables applications including noise reduction, particle separation, counting, and morphological parameter extraction. Despite its promise, mathematical morphology is utilized in biomedical image analysis rather than real medical imaging. One of the reasons for this is that the conceptual foundation was primarily designed for binary 2D images [7]. Moreover, despite the growing interest in grey-level morphology, applications of mathematical morphology to pictures with dimensions bigger than two are still in the works.

2.2 Segmentation, Processing, and Features

All of these three: features, preprocessing, and segmentation [6], strive to remove systematic disturbances from the picture or to improve the image for display and possibly future processing. High-frequency acquisition noises, background illumination changes, camera geometrical aberrations, and echoes are examples of typical disturbances. Preprocessing procedures are frequently used in a systematic manner on all pictures produced by a specific instrument. As a result, they are frequently gadget-dependent, and they must be quick and efficient. The goal of segmentation is to extract significant primitives and areas of interest. This is required either for making measurements that will enable classification and recognition, or even for reconstruction, which will decide on which elements would be put in correspond to create a 3D representation. A large volume of methods has been released, with three families of approaches often described: edges extraction, and regions extraction, thresholding the main concepts stated below apply to 3D or 2D recorded (tomographic) scans. The basic task of preprocessing and segmentation is to extract measurements for classification and identification issues. Regardless of the fact that there are as many distinct metrics as there are distinct issues. Certain fundamental descriptors are frequently employed. The length, average contrast on either edge of the contour, and the regional or average curve are common 2D contour-based measurements.
Simple metrics such as average brightness and color, surface, normal luminance perimeter, and compactness gradient with neighboring regions are commonly used as region-based descriptors. Scenes about the mass center are also utilized, for example, to determine the axes of an approximate ellipse, and the position of this center may aid in object location [6]. These metrics are easily generalizable to higher dimensions. Segmentation also was required for extracting primitive features, which are then approximated by some parametric form, as well as isolating sections or outlines in interactive picture modification and observation. Parameterized forms are important in image synthesis and computer vision. Geometric models are therefore the fundamental elements which that visualization pipeline must analyze in order to synthesize a picture [7]. They are also required in model-based segmentation, in which the picture is divided into portions that correspond to these models. The most often used models in medical imaging are the basic pixel as well as the 3D hexagonal planar overlay. Lookup tables for linearity correction, dynamic range modifications, histogram equalization or dynamic histogram modifications for visualizing, and local or global artifact estimates and removal are examples of preprocessing approaches [4]. They are frequently done in real-time in the graphics card memory, where they primarily serve a visual purpose. When an upgraded version of one item is sought, a specific family of methods related to low-pass filtering is applied, assuming that numerous reduced, distorted, and rotated copies of the same object are accessible. The process, known as correlation averaging, works by aligning the parts using correlation algorithms and then averaging them. The spatial resolution is typically proportional to the square root of the number of mean individual elements in terms of signal-to-noise ratio. When images are geometrically deformed, structural adjustments are necessary. This is required, for example, when registering photos from diverse sources. Finally, restoration methods are utilized inside this classification of

preprocessing families when a model of the vibration permits the development of an appropriate cleaning filter. In summary, look-up panel manipulations and chromatic aberrations masking are frequently useful for data visualization on a medical imaging workstation [20] however, caution should be exercised in this case to avoid having the data preparation modify the picture in such a way that it misleads the observer and generates incorrect interpretations. Low-pass filtering is typically required for preparing data prior to further image analysis.

Segmentation thresholding approaches allow for the determination of one or more grey-level thresholds, often by examining the image's smoothed global gray-level histogram. As a consequence, we get a binary picture made up of areas, preferably with objects separated from the backdrop. The main disadvantage is the need for adaptability: global thresholding seldom results in suitable areas. In general, the images must be post-processed, generally utilizing mathematical morphology procedures such as erosion or dilation. Edge extraction is used to delineate areas by finding their contours. These transitions, or areas of substantial variation, are first extracted by using differential operators, or customized operators that determine which sections of the picture correspond to the modeling of an ideal contour. The final contours are created by using a peak-following method to locate the gradient maximum in the resultant picture, or by using the zero values of the Laplacian. Despite the apparent ease of the task, extracting edges remains a major topic. The main disadvantage of edge extraction for outlining regions is its high susceptibility to noise; common post-processing comprises thinning techniques as gap-closing methods for comparing and discussing various edge finders. Pre-processing for noise reduction is also required and is often accomplished with a linear or a non-linear Gaussian median filter. Area extraction produces picture sections that fulfill a certain uniformity condition; for a given region, the average grey-level difference between a particular pixel and any other pixel in the region must be less than a particular value, another consideration in choosing a given pixel and just its immediate neighbors [21]. These approaches readily offer closed areas and hence closed contours for morphological measurements. The disadvantages include the difficulty of implementation and the fact that several tiny areas are often generated. Post-processing is also necessary in this case, such as form regularization and the eradication of tiny sections. To summarize, selecting a suitable segmentation method is tough. Because there is no ideal technique, interactive picture altering software is required. The idea of least commitment should be followed, which means avoiding extreme operations wherever feasible. For example, if edges are wanted, edge extraction and peaking chase more information than thresholding the picture and conducting morphological erosion.

Because third dimension segmentation provides more information, it is a smart idea to employ it as much as feasible. Although it is possible to simplify 3D edge extract to a 2D method that operates on slices, the current tendency is to operate directly on 3D pictures. Segmenting 3D pictures into, in general, n-D data into n-D regions is comparable to segmenting 2D photos into 2D regions. Thresholding, determining boundary areas in which

the data fluctuates, and extracting volumetric areas where a uniformity condition is satisfied are all approaches that may be applied. The results produced using these traditional procedures are frequently unsatisfactory, owing to scans' low data quality. Current research is focused on other approximation techniques, such as the use of 3D "inflatable balloons" [2] that adjust to critical data properties such as discontinuities surfaces. This is performed by altering the model iteratively in order to achieve a stable state among internal forces provided by the elastic characteristics of the balloon and exterior effects attraction towards the important features. 3D models were derived using volume-covering acquisition techniques.

2.3 3D Models Derived Using Volume-Covering Acquisition Techniques

The transition from 2D representation of 3D bodies via transmission techniques X-Rays to complete 3D neuroimaging (for example CT and MRI) might be regarded as a major advance in medical images. Physical measurements may now be acquired in just the same quantity of dimensions as the human body. The ability to see structural components, including anatomical and functional, had resulted in the development of novel and incredibly valuable tools for treatment planning. 3D approaches gather spatial information as tightly sampled volume elements/pixels [24], typically in a multislice process with finer sampling between the slices to reduce extra radiation or owing to time constraints.

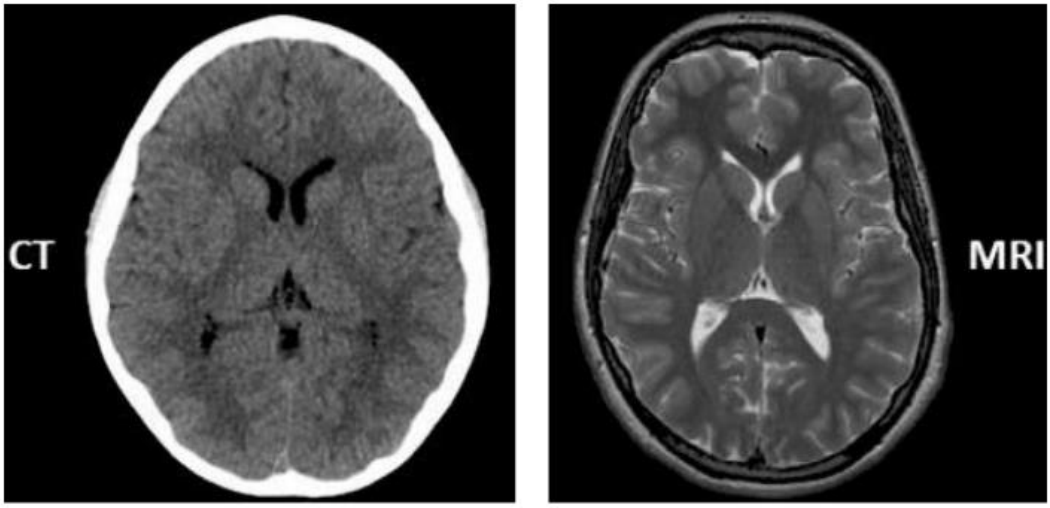


Figure 2: A brain computed tomography (CT) scan in comparison with a magnetic resonance imaging (MRI) scan [10].

The type of computer-assisted analysis is also influenced by the two alternative methods of seeing the data, either as a spatial series of 2D images or as a nii data volume. The traditional example involves matching information from 2D slices to generate a 3D detailed representation. Single slices of a series are processed using 2D picture segmentation algorithms, which frequently ignore the context information available from subsequent slices. However, seeing picture data as a continuous volume leads to the creation of actual 3D image processing techniques [8], which are frequently expansions of 2D algorithms. The pixel and surface techniques are the two primary categories of 3D data manipulation technologies.

Pixels are the fundamental parts of the metadata in the pixel approaches. Due to the high amount of data, processing such information is time-consuming. Furthermore, visualization is computationally very expensive: either the list of all pixels must be reviewed, with numerous extra operations due to hidden frames, or the area of the set of objects must be identified. Alternative pixel visualization methods are available, and their performance is becoming completely equivalent to previous systems that depict merely the object surfaces. The initial image intensities may be readily merged with edges, for example on cutting planes using pixel approaches, and surgical planning is simplified by controlling volumes rather than empty surfaces. Surface techniques frequently retrieve 3D information indirectly from 2D data slices by extracting and matching 2D shapes from these slices to generate a 3D triangulation of planar patches. The application of polygonal contour approximations from consecutive slices is commonly used for triangulation.

Various requirements, such as outward surface maximization or interior volume reduction, may be enforced. Although this approximation method is challenging, it allows for significant data reduction, extra capabilities such as volumetric and weight data, and effective display and animation systems. It can also be used as a first step before employing matching algorithms. Following the creation of 3D object representations, the results are often shown utilizing image synthesis techniques [24]. Shading and advanced lighting can be utilized for anatomical images, MRI or CT, however for practical images, the original intensities must not be changed because they include the necessary information, hence no shading or lighting algorithm should be employed. By mapping the actual intensities onto anatomically relevant surfaces, functional information is frequently integrated with morphological information. In its most basic form, n-dimensional matching entails registering a succession of n-dimensional pictures, with n, commonly being 3 or 2. The main concept behind matching is to start by identifying certain significant features from one of the n-dimensional pictures, such as utilizing segmentation-like approaches. Correlation is then achieved by looking for these specific traits in the other photos. The main distinction between matching approaches seems to be between rigid and elastic fitting, as well as between registration utilizing particular landmark pairings such as points, lines, specific patches, and completely volumetric techniques such as hierarchical correlation [3]. The essential geometrical characteristics utilized as reference pairs must be invariant through translations, rotations, and, in some cases, elastic deformations of the objects [3]. In plenty of other words, it should ideally be inherent object properties. The following functionalities are available for use:

- Pixels: since they are many, they are hard to match across slices. Their intensity, however, is typically constant when viewed from varied angles;
- Line segments: have more properties than pixels and are hence easier to match;
- Regions: less frequent and simpler to correlate, but more complex to extract and not always invariant;
- Critical points (like edges): tough to extract, but essential to the objects and hence highly robust matching characteristics.

Multimodality matching through registration techniques allows users to have quick real-time access to many types of information, examples of applications include functional or metabolic scanning, in which functionality data is mapped onto 3D anatomic structures. When digital information is compared to more general knowledge, such as that supplied by an anatomical atlas, a similar type of matching happens. In the robotics area, for example, the issue of stereo pair registration for obtaining depth data from 2D images is fairly well handled. Combining 3D geometric objects has also been presented as a solution. This is not the case with 3D medical imaging, because, in addition to the added complexity of the 3d object, the input data lacks the pleasant geometrical regularities provided by man-made things [27]. Various ways for rigid or dynamic fitting are being researched. Because the form of the items to be matched is fixed, rigid matching is significantly easier, the only degrees of freedom concern the relative displacement of the elements to be compared. When an image sequence from the same device is presented at multiple times, temporal matching establishes a correlation among characteristics from the different pictures. If the time delay between subsequent acquisitions is long enough, the search algorithm is essentially analogous to multimodality fitting: a predetermined structure is monitored through time.

Since this interval is tiny, however, different approaches can be used since it can be expected that the differences between pictures are minute. In such circumstances, the characteristics to be compared have a size of the scale of a pixel. An important stage is the recovery of the image sequences, which is the estimation of the 2D image of the 3D velocity field of the scene. Following computation, objects can be segmented based on their intrinsic motion: distinct items can have different sequences flow patterns. This segmentation may be used to both quantify the time evolution of an object's properties and extract structural information. The brightness constancy technique, which assumes that the illumination of a particular spot is constant across two frames [8], is one of the basic ways of computing optical flow. The spatial gradient constancy method asserts that the spatial luminance contour is static over time, the local correlation approach seeks correspondence across groups of nearby points, and the energetic approach employs 3D directional filters in 2D space and time with maximum response along with motion directions. It should be emphasized that the very first three approaches relate to monitoring pixels, edges, or areas between subsequent photos, respectively. Predictions of the optical flow may now be computed in practically real-time.

2.4 Object Recognition

Object recognition consists of two steps: learning and exploitation, during the learning phase, typical properties of the items to be identified, such as morphological measurements, are determined and stored. Techniques like as segmentation and feature extraction, as explained in this chapter, are used to make this judgment. Learning is frequently supervised: the user determines which features are meaningful and discriminating. During exploitation,

unknown objects are sent through the same pipeline as those used for learning, and their properties are measured as a result. These measurements are then compared to the previously learned ones, and the result shows which item is being considered. The most traditional recognition methodology is quantitative in nature: covariance matrices for discriminative measures are produced, mostly under the multivariate Gaussian hypothesis. Then, metric discriminant functions are calculated, allowing for the categorization of unknown samples. These functions result in the usage of distances in parameter space, where a point in parameter space is categorized with the nearest class center based on the distance specified. Many recognition algorithms are based on this method, which has several drawbacks, including the necessity for huge training sets, the difficulty of accounting for structural information, and the problem of establishing an adequate parameter space. Neural networks have been praised as effective classifiers due to their capacity to build nonlinear decision surfaces on the one hand, and the back-propagation learning process on the other, which eliminates the necessity for explicitly specifying the parameters space. Several applications of the neuronal model to medical imaging are emerging, including not just classification but also segmentation, which is considered as a classification across objects and background [22]. The key drawbacks of this strategy are the requirement for huge training sets or the inability to take structural information into consideration. Despite these disadvantages, the neural technique is popular due to its unsupervised nature, a training and validation sets are still required. The primary idea is to characterize objects using symbolic structures and then compare the reported shape to a model structure: these structures might vary in complexity. Comparisons utilizing graph matching or grammar are typical of symbolic techniques. In the latter scenario, an object is characterized by a series of symbols that follow a certain grammar and are then analyzed by a lexical analyzer. This paradigm allows for the consideration of structural data and does not need huge training samples. Its main disadvantages are the challenge in processing data and extracting symbols and, as a result, the difficulty in dealing with noise. The most challenging aspect of computer vision is determining how to get symbolic data from digital input.

2.5 Direction

Many of the medical image analysis and computer vision techniques discussed in this chapter are now frequently utilized in biomedical laboratories. Various trends are forming as a result of these very traditional procedures. New detectors are being created, which should lead to less intrusive and much more precise imaging techniques. Multi-modal and functional imaging are increasingly being utilized to uncover and understand functional structures. It is feasible to integrate data further to add medical information, for example, from a patient file. A great deal of work is placed into the development of integrated health information systems, particularly those linked to medical imaging. Despite the fact that it may look at first sight that the problem is primarily technological, there are numerous theoretical issues to address, including image data for processing and transfer, software development, the design of

common rules for needed storing and visualizing, and AI techniques for the creation of dedicated intelligent image databases. Feature-based indexing approaches are being developed, which will be critical for picture database retrieval. Such databases should offer access to material based on semantic content, making searches like “find all radiograph scans revealing a broken arm” viable. Knowledge acquisition is a similar problem. Although different approaches for supervised learning exist, as seen above, unsupervised extraction of important traits that will enable future recognition remains a challenge that is far from being solved. The drive toward minimally invasive therapy employing better and more accurate equipment characterizes modern surgical procedures, for example, endoscopic interventions or laser surgery. A patient will benefit from this advancement by having less risk, less discomfort, a shorter hospital stays, and a speedier recovery. These innovative procedures can be significantly aided by meticulous preoperative preparation and exact feedback data based on the image analysis [27].

Sophisticated robotic vision technologies will enable the development of medical robotics, or the construction of robots capable of performing difficult surgical treatments automatically [9]. All of these advancements necessitate the use of more powerful computer vision algorithms. Future advancements will almost probably include an increase in the utilization of top-down information if least to escape the limitations imposed by existing segmentation approaches. The 3D picture will be segmented initially, and the output will be compared to symbolic medical information from an atlas or the patient's file. This will allow for improved initial anatomical segmentation and, eventually, organ labeling via a feedback loop. Other advances in computer vision will very probably be applied in medical imaging. The computational complexity of scene analysis is a key issue; there are an infinite number of potential mappings from object classes to the digitized scene. The fundamental goal of these techniques is to reduce the size of the optimal solution, for example, by employing feature matrices, dynamic vision, or perception-based approaches such as point of focus. There is also a tendency toward less data acquisition by employing non-regular matrices, such as polar lattices, which imitate the human retina by offering high-resolution only towards the center. Connectivity and networking will be required for integration with a health information system. It will also allow for a more equitable division of financial means: a low-cost computer will be used for contact with the personnel, while the CPU intensive tasks such as analysis of an image or processing of the same image, with all computation and interactions controlled by another, more powerful computer, will be performed. It will also put contemporary workplace concepts centered on groupware settings into action.

The last section of this chapter presents a major European community research program, a project that was a source of inspiration for my idea to make an application for automatic segmentation and delimitation of brain tumors. That contains several of the key premises identified as essential challenges for future medical image analysis research, including multidisciplinary collaboration, clinical testing, and validation. Furthermore, software tool standards must provide interchangeability and transportability. The program is

titled “COVIRA” also known as Computer Vision in Radiology [23]. The goal of this project is to develop a comprehensive software system that will provide effective computer aid in the neuroradiological diagnosis, conformal and open neurosurgery planning, and radiation treatment planning. Is the result of a collaboration between three significant industry partners like IBM, Philips, and Siemens, eight university research groups, and six clinical institutes from Spain, Italy, Netherlands, Germany, Belgium, Crete, United Kingdom, and Switzerland. The system based mostly on software will be developed to strict specifications and will use object-oriented programming. All methods will be transferable and enter a shared pool of computer algorithms for medical diagnostic imaging 2D and 3D image analysis, in accordance with all image processing ISO standards [23]. The system's primary functional capabilities will include: 2D and 3D image visualization plus automated segmentation techniques combined with interactive imagery advanced features; reconstruction of the cerebral vasculature using data from “3D MR Angiography” and “2D Digital Subtraction Angiography”; multi-modal image validation; computerized and annotated brain anatomical map [23]. The project's overarching goal is to create and clinically evaluate prototype image analysis application systems for diagnostics and treatments management, built on usual computers and accelerator hardware and standard software tools. The partnership of academic, industrial, and medical partners presents a valuable mix of skills and expertise in image analysis and research, software engineering, medical equipment production, diagnosis, and therapy. The collaboration hopes that the research will ultimately demonstrate the clinical utility and cost-efficacy of computer aid in a variety of application domains.

3. Data

MICCAI brain tumor segmentation 2020 datasets are multisequence MR preoperative images of individuals with malignant brain tumors available as NIfTI (.nii) files and were acquired from several institutions using varied clinical methods and scanners. Four MRI sequences were collected for each patient, shown in Figure 5 and Figure 7: T1 contrast-enhanced (gadolinium), T1, T2, FLAIR (fluid-attenuated inversion recovery), and manual tumor segmentation performed by experimented doctors. For the survival prediction, we have one CSV document containing pieces of information about each patient label, a preview is available in Figure 6. The scans were obtained from 19 distinct imaging locations using various MRI technologies and clinical procedures. All of the imaging datasets were manually segmented by one to four evaluators using the same annotation technique, and their labels were approved by competent neuroradiologists. The first test dataset in our research has 45 randomly selected patients from the training dataset, and the U-Net convolutional networks are trained on the remaining 324 patients, the data distribution can be seen in Figure 3. After evaluating our strategy on the validation dataset of 74 instances, the model will be trained on

the training dataset of 250 cases. The first phase is to train the model, and the second is to evaluate it on a validation data set, which is a subset of data used to enhance overall effectiveness in testing and could also be used for hyper-parameter tweaking. After training and hyper-parameter tuning, a subset of the data is utilized to test/evaluate the model.

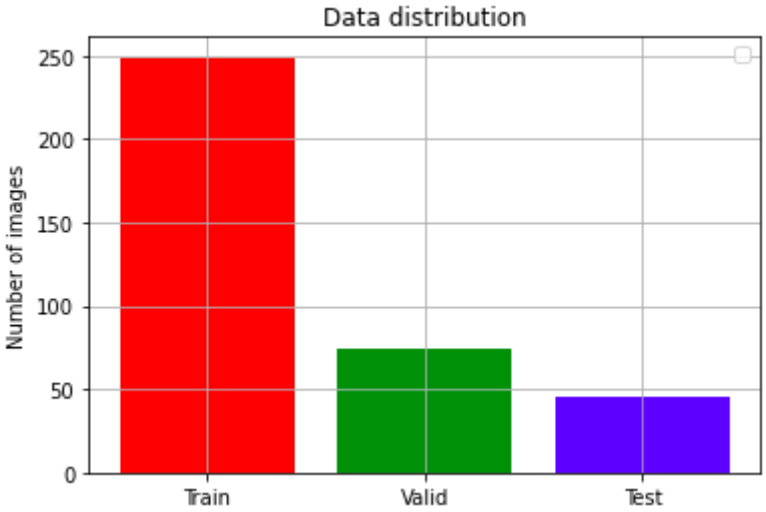


Figure 3: Data distribution.

The ground truth relates to voxel-wise annotations classified into four categories: non-tumor (class 0), necrotic core/non-enhancing tumor (class 1), tumor edema invasion (class 2), and enhancing tumor (class 3). The performance of semantic segmentation applications is determined by the Dice loss, the general formula is written in equation (1), between the performance of the algorithm's segmentation ($\tilde{Y}$) and the ground truth segmentation (Y):

$$DSC\ (\tilde{Y}, Y) = \frac{2|\tilde{Y} \cap Y|}{|\tilde{Y}| + |Y|} \qquad (1)$$

I use the Dice coefficient (2) to evaluate the segmentation results. The Dice coefficient, commonly known as the overlap indicator, is a common validation statistic in semantic image segmentation. The segmentation pair-wise overlap is determined as follows: where TP denotes true positive findings or correctly segmented tumor pixels, FP denotes false-positive results, and FN denotes false-negative segmentation results. When a non-tumorous pixel is categorized as tumorous, false-positive findings are achieved. FN also refers to the number of tumorous pixels that have been mislabeled as non-tumorous.

$$\text{Dice coefficient} = \frac{2\,x\,TP}{2\,x\,TP + FP + FN} \qquad (2)$$

For scoring, I also used categorical cross-entropy from TensorFlow Keras, computer precision, computer sensitivity, and computer specificity. They are often used together. A loss function used during multi-class classification applications is categorical cross-entropy. These are tasks in which an example may only belong to one of many potential categories, and the model must determine which one. Its formal purpose is to measure the distinction between different probability distributions. The categorical cross-entropy (Figure 4) loss function calculates the loss of an example by computing this math function (3):

$$\text{Loss} = -\sum_{i=1}^{output\ size} y_i \ x \ \log y_i' \qquad (3)$$

Where:

y'_i is the ii-th scalar value in the model output,

y_i is the appropriate target value, and the number of scalars in the model outcome represents the output size.

This loss is an excellent indicator of how distinct two discrete probability distributions are from one another. In this case, y_i represents the chance that event $_{ii}$ happens, and the total of all y_i equals 1, implying that only one event is possible. The negative sign assures that the loss decreases as the distributions approach each other.

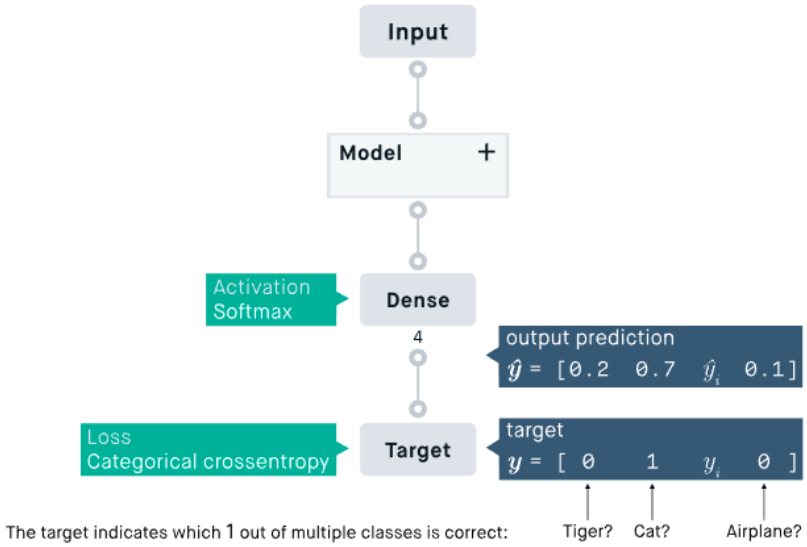


Figure 4: Categorical cross-entropy design schema [12].

Precision is a relevance-based performance statistic that applies to data obtained from a collection. It is also known as positive predictive value and is the proportion of correctly classified instances. For example, consider a computer software that recognizes a tumor from X organ (the relevant element) in a digital image is an example of precision. The computer finds eight scans that include a tumor after processing an image that contains 10 scans with

no tumor and twelve scans with a tumor. Only five of the eight components indicated as cancers are tumors (true positives), whereas the other three are not tumors (false positives). Seven tumor-containing scans were overlooked (false negatives), whereas seven tumor-free scans were appropriately eliminated (true negatives). The accuracy of the software is thus 5/8 (true positives/chosen items) as in equation (4).

$$\text{Precision} = \frac{TP}{TP + FP} \qquad (4)$$

The mathematical terms sensitivity and specificity indicate the accuracy of a test that indicates the presence or absence of a condition. Individuals who meet the requirement are labeled "positive", while those who do not are considered "negative". The proportion of true positives to total ground truth positives (TP + FN) is measured by sensitivity as in the equation (5). The proportion of true negatives (TN) to total ground truth negatives (TN + FP) is measured by specificity as in the equation (6). In other words, as the sensitivity grows, the chance of a negatively labeled pixel being a genuine negative increases, while the chance of a positively labeled pixel being a true positive increases.

$$\text{Sensitivity} = \frac{TP}{TP + FN} \qquad (5)$$

$$\text{Specificity} = \frac{TN}{TN + FP} \qquad (6)$$

For the patient survival prediction step, I also prepare for evaluation a confusion matrix, for a complete set of evaluation metrics. Confusion matrix is a means to illustrate a model's predictions on a data set in a way that demonstrates the model's performance in terms of false positives and false negatives. This method of presenting the predictions is beneficial for analyzing the model's performance. Because of changes in picture capture, the limits of image intensities vary greatly across scans. As a result, we execute a basic intensity normalization: we calculate the median value of non-zero pixels for each patient and MRI sequence individually, divide the series by this median, and multiply it by a preset constant. In reality, the median is more likely to stay steady than that of the mean, which is quickly influenced by the tumor region.

Because the datasets utilized in machine learning are so large, the network's input fluctuates greatly. Instead of normalizing the entire dataset as a preprocessing approach, batch normalization normalizes the data in tiny batches and makes the normalization network connections. For example, we won't be able to put all of those photos into memory before training an image classifier. Even though it is doable with a small dataset, it is not possible

with a huge dataset. We must be able to make full use of the supplied data. So, data generators were developed to address this issue, we can read photos on the fly when they are utilized for training. We save memory since we read the data as we go, even a computer with less (8GB or lower) RAM memory can handle a 50GB dataset (or bigger). Because the functionality of the standard Keras generator is restricted, we need to build a custom one for matching our requirements. For example, if the network has several output nodes, the conventional data generator will not operate. Above all, implementing a data generator for Keras is not difficult, and it is highly powerful and adaptable. Whole data processing steps may apply additional functions to data before it is fed into the model.

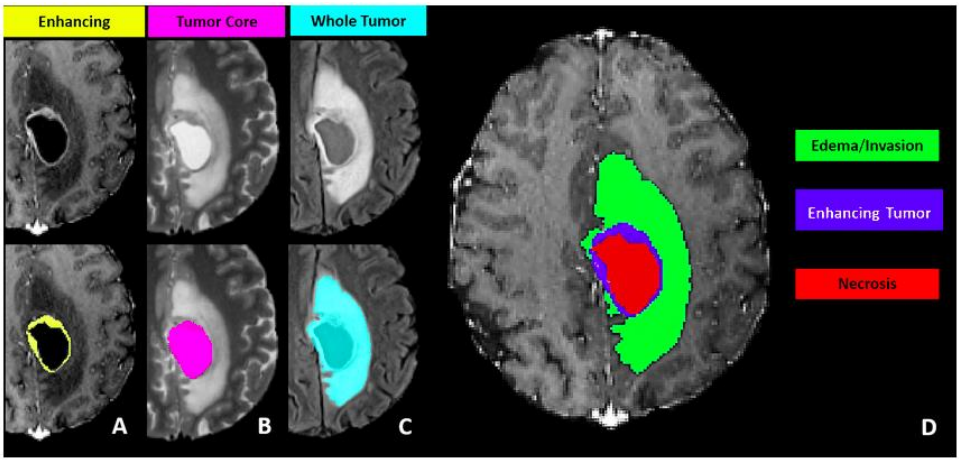


Figure 5: Semantic segmentation of brain tumor [11].

	Standard	Standard	Standard	Standard
1	Brats20ID	Age	Survival_days	Extent_of_Resection
2	BraTS20_Training_001	60.463	289	GTR
3	BraTS20_Training_002	52.263	616	GTR
4	BraTS20_Training_003	54.301	464	GTR
5	BraTS20_Training_004	39.068	788	GTR
6	BraTS20_Training_005	68.493	465	GTR
7	BraTS20_Training_006	67.126	269	GTR
8	BraTS20_Training_007	69.912	503	GTR
9	BraTS20_Training_008	68.285	1278	NA
10	BraTS20_Training_009	56.419	1155	GTR

Figure 6: Patient survival prediction data.

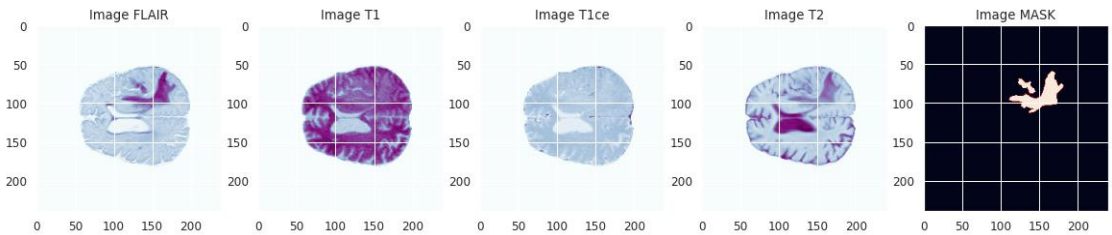


Figure 7: Slices preview from a random chosen volume.

3.1 Data Pre-Processing

Pre-processing is an essential stage in many types of data processing, not only image/signal processing. Pre-processing a dataset need to make it more understandable to an algorithm. Although several pre-processing methods are available in the literature, only a handful of them, including normalization, histogram matching, and histogram equalization, may be used on brain MRIs. Normalization is a common pre-processing step that seeks to scale data between a specified maximum and a set minimum, which can be linear or non-linear. The histograms of the photos can be compared to the histogram of a target object to normalize them, this technique may be troublesome in the brain tumor semantic segmentation problem because not all patients have a tumor. As a result of this, if the brain picture lacks a tumor area, the pixel intensities which are meant to be distinctive to the tumor location may appear after histogram matching. The goal of this strategy is to provide a more uniform histogram because then the image's contrast grows (Figure 8), which has an impact on the data. Background data, for example, become less discernible. To avoid such artifacts in the brain tumor segmentation challenge, it should be used in conjunction with a foreground mask. Using a mapping function to get the most probable linear-gradient-shaped distribution is a typical way of producing an equalized histogram. An important task focuses on adjusting certain post-processing procedures in order to enhance performance. Because our challenge is multi-class, applying deterministic techniques such as opening degradation and dilation sequentially is impractical. Conditional random fields may filter the output based on the pixel values and geographical positions of the input. I am aware that, due to similar properties, U-Net, like any other algorithm, may fail to discriminate between other abnormalities and tumor tissues. Data normalization is a fundamental procedure for preparing data for image processing training. MRI scans with various scanner types or scanning parameters have variable intensity information in brain tumor segmentation, it has no bearing on a doctor's diagnosis. Images without a normalized intensity scale, on the other hand, may fail to produce good results when it comes to automated segmentation. The pixel values in each slice were normalized to zero mean, and unit variance in this scenario. Slices for all datasets (training, validation, and evaluation/testing) are adjusted to a constant size of 128x128.

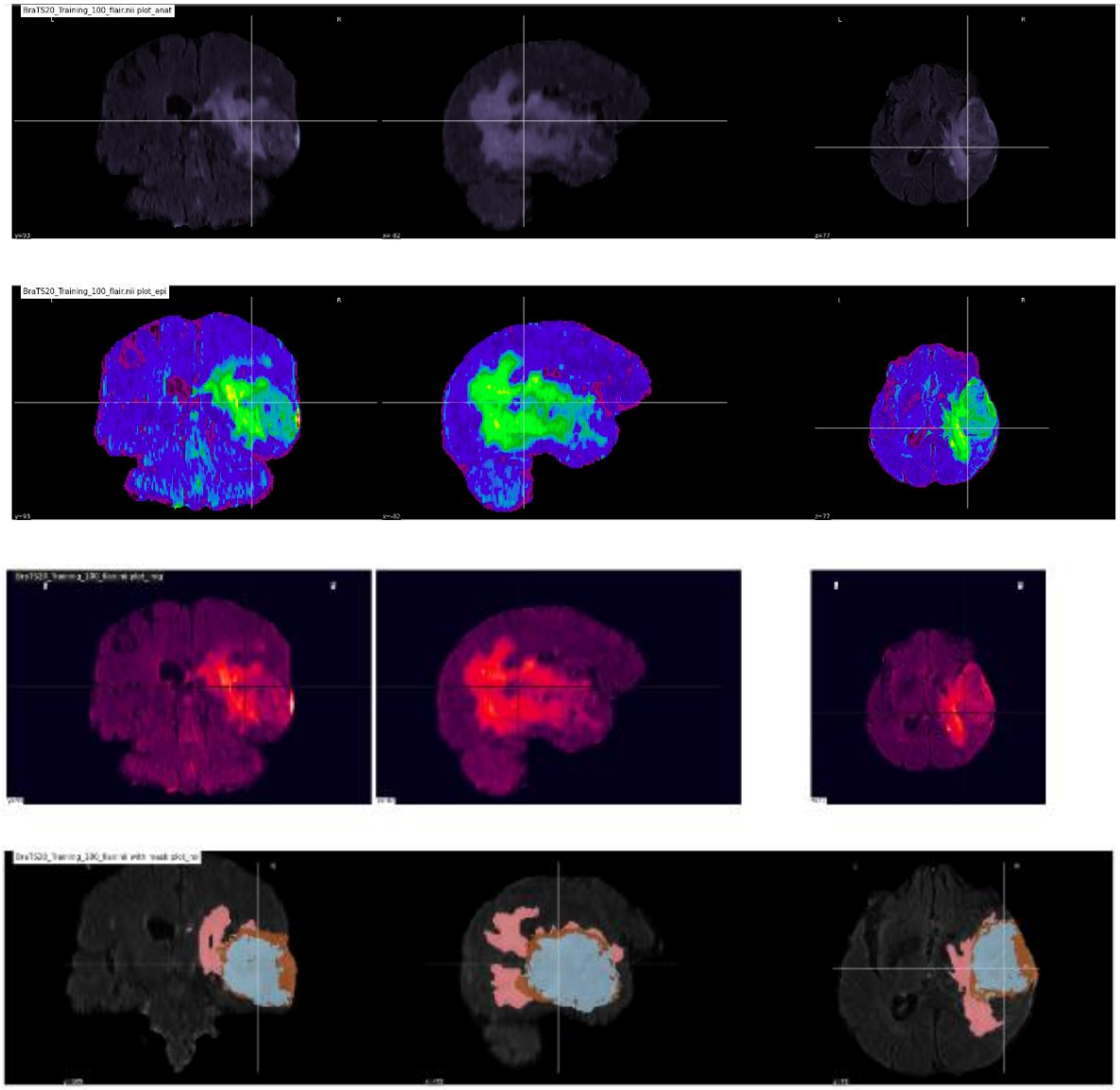


Figure 8: Showing tumor segments on sample 100 from the training dataset (YXZ axis) using several effects.

Registration is the process of superimposing two or more scans are obtained at various times, from different perspectives, and with separate sensors. The scan duration for diagnosing brain illnesses is generally between 30 and 60 minutes [1]. It is normal for the patient to move throughout the test, which causes alterations in the obtained MRI pictures. This shift can be rectified for with manual or automatic registration procedures during the data pre-processing stage. Although automatic registration algorithms often perform better than human registration, the results should be verified before to execution. Image registration may be classified into four categories, the most popular of which are multitemporal analysis as well as multimodal analysis in medical imaging [1]. Images with the same scene are taken at different moments in multitemporal analysis. The goal of registration in this situation is to align pictures in order to analyze changes in the scene that arise between image acquisitions; this may be used to examine changes in a disease. The multimodal analysis

is the use of many sensors to capture pictures of the same situation. The goal of this registration step is to coordinate information from many sources in order to obtain a more complete picture of the scene. Different imaging modalities are frequently employed on the same subject in the case of MRI, and the alignment of various imaging modalities yields a more thorough image of the subject. Due to picture variability and deterioration, there is no one universal registration approach that can be used for all registration needs. The bulk of registration techniques follows four phases in general. The first phase is feature detection, where the algorithm looks for features in the pictures to be overlaid. The second phase is feature matching, which involves matching the previously discovered features. The third phase is transformed model estimation, which is the process through which the registration algorithm discovers a mapping function that may be used to align two images. The next stage is picture resampling and transformation, which involves applying the mapping function to one of the images in order to match it with the other image. For the automation of picture registration, two techniques have emerged: area-based but also feature-based methods. The feature-based technique extracts picture structures including lines or points. Structures should be identifiable and scattered across the entire image for this strategy to function properly [1]. The method does not operate directly with picture intensity fluctuations; in such instances, an area-based method is preferable. Because there is occasionally a paucity of distinguishable items in medical photos, an area-based technique would be the preferred registration method.

Spatial resolution in the analysis of medical images in machine learning requires the same image sizes as the images used in neural networks for training, validation, and testing. In the case of MR data, different machines but also clinical protocols produce images of varying sizes, necessitating the need to resample the images to the same size. Image resizing is a scaling procedure that falls underneath image acquisition and is performed using interpolation methods. There are several interpolation methods, but they all try to add information by monitoring surrounding numbers and making an educated prediction. When reviewing the gray-value deviations of the 84 filters studied in a paper [5] on medical picture interpolation, the linear interpolation performs best for 2 grid points and the cubic convolution filter for 4 grid points. Grey-value mistakes are reduced by 28% to 75% when using cubic convolution.

Normalizing pixel-values in the situation of the BraTS dataset its very clear, all pixels with value 0 are excluded since they do not represent brain tissue, and all other pixel values are normalized using Z-score (input values that are close to zero and have a unit variation).

Volume estimation of the brain tumor at two separate times in time allows for a conclusion about tumor progression and the establishment of a treatment strategy. A voxel is a pixel with a volume of $1mm^3$ in one of the pictures from the dataset. All of the images have been interpolated to this resolution by experts, therefore the volume may be computed simply adding all of the segmented pixels together.

3.2 Software Libraries and Workspace

The models in this thesis were created using Keras, an Application Programming Interface (API) that is connected with the deep learning (DL) - machine learning (ML) - artificial intelligence (AI) platform TensorFlow. Keras distinguishes itself with its simplicity, flexibility, and power. It includes sequential models and also individual blocks for various levels. All software libraries used in this application are open source.

TensorFlow, which was originally made public by the Google Brain in 2015, is an open-source software framework designed for expressing and executing machine learning and artificial intelligence algorithms. It was designed to scale, allowing computations to be done on numerous CPUs, GPUs, and TPUs (for Google Cloud-based platforms like Google Colaboratory and Kaggle Code) for quicker calculations. TensorFlow is fast at doing computational work since it is written in C++. It can be accessed and controlled using other languages such as Python, JavaScript, or Java. Because of its accessibility, ease of use, and speed, it has become one of the most widely used machine learning libraries. There are additional libraries that can lie on top of TensorFlow, including one also used in this project, named Keras, which will be discussed next.

Keras is a high-level neural network API developed in Python that can operate on top of TensorFlow. It enables quick and easy prototyping because of its user-friendliness, modularity, and extensibility. Keras is built on top of TensorFlow and can operate on multiple CPUs, GPUs, and TPU accelerators for scalability and performance.

Scikit-learn is a Python library that incorporates a diverse set of cutting-edge machine learning methods for supervised and unsupervised problems [30]. It includes support-vector machines, random forests, gradient boosting, K-Means, and DBSCAN as classification, regression, and clustering algorithms, and is meant to work with Python numerical and scientific libraries such as NumPy, Pandas, PyTorch, and SciPy.

I use NiLearn and NiBabel for neural imaging. NiLearn is a Python module for multivariate statistics that uses the Scikit-Learn Python toolbox for applications such as predictive modeling, classification, decoding, and connection analysis. NiLearn offers simple and flexible brain volume analyses. It offers a variety of statistical and machine-learning tools, as well as instructional material and an open community.

NiBabel provides read and write access to the following neuroimaging file formats: ANALYZE (plain, SPM99, SPM2, and later), GIFTI, NIfTI, MINC1, MINC2, AFNI BRIK/HEAD, MGH, as well as Philips PAR/REC [25]. We can read and write geometry, annotation, and morphometry files from FreeSurfer. The different image format classes provide complete or

partial access to header (meta) information, as well as access to picture data through NumPy arrays.

For other things like images pre- and post-processing, file path manipulations, visualizations, and numerical computations the following packages from below are a must-have. Pillow is a Python Imaging Library (PIL) that includes image opening, manipulation, and storing capability. The current version recognizes and reads a wide range of formats.
Write capability is purposefully limited to the most widely used interchange and display formats.

Scikit-image is a Python-based open-source image processing package.
It features segmentation, geometric transformations, color space control, analysis, filtering, morphology, and feature recognition algorithms, and it is meant to work with the Python numerical and analytical libraries NumPy and SciPy.

Matplotlib is a Python charting toolkit that allows you to create static, animated, and interactive representations like line plot, scatter plot, histogram, image plot, contour, and 3D plot, these are just a few of them.

Seaborn is a matplotlib-based data visualization software package. It offers a high interface for creating visually appealing and useful statistical visualizations to illustrate random distributions.

Pandas is a data manipulation and analysis software tool developed for the Python ecosystem. It provides data structures and functions for manipulating tabular data and time-series data.

NumPy is a Python library that adds support for huge, multi-dimensional arrays and matrices, as well as a vast number of high-level mathematical functions to work on these arrays.

The Python Shutil module offers numerous high-level actions on files and groups of files. It's one of Python's standard utility modules. This module aids in the automation of file and directory copying and removal.

Glob module, short for global, returns all file paths that match a given pattern.
We may use glob to search for a certain file pattern, or we can use wildcard characters to look for files whose filename fits a given pattern. Used together with the OS module contains methods for interfacing with the operating system. Python's basic utility modules include OS. This module allows you to use operating system-specific functions on the go.

OpenCV (Open-Source Computer Vision Collection or CV2) is a library of programming functions primarily intended for real-time computer vision. on. It is developed in C++ and uses C++ as its primary interface. Python, Java, and MATLAB/OCTAVE bindings are available, an API is available for these interfaces. OpenCV applications include: 2D and 3D feature toolkits, ego-motion estimation, facial recognition, gesture recognition, the interaction between a human and a computer (HCI), mobile robotics, motion understanding, object detection, segmentation, and recognition of objects or multiple objects in a scene structure from motion (SFM), motion tracking, augmented reality, and so on. For hardware acceleration, CV2 has an OpenCL-based GPU interface and a CUDA-based GPU interface. Also, OpenCV is compatible with both desktop and mobile operating systems. To support the previously mentioned areas, OpenCV includes a statistical machine learning library that includes algorithms like deep neural networks (DNN), support vector machines (SVM), random forests, artificial neural networks (ANN), Naive-Bayes classifier, k-nearest neighbors, expectation-maximization, gradient boosting, decision trees, boosting, etc.

Kaggle, a subdivision of Google company, is a data scientist and machine learning online community. Users may use Kaggle to search and post data sets, study and construct models in an internet data-science environment, collaborate with many other data scientists and machine learning experts, and compete to solve data science challenges. Kaggle began in 2010 with machine learning tournaments and has now expanded to include a public data platform, a cloud-based workspace for data science, and artificial intelligence with all its subdivisions (deep learning and machine learning). Kaggle offers strong cloud resources and allows you to use up to 30 hours of GPU-accelerator and 20 hours of TPU-accelerator every week. The dataset, with a size of 43GB, already belongs to Kaggle due to the fact that the competition took place here and only needs to be imported. An advantage of Kaggle Code is that the environment for developing a new jupyter notebook comes pre-installed with most required software libraries, such as NumPy, Pandas, Matplotlib, Seaborn, and others. Also, the desired packages can be installed using the command to install libraries from PyPi, pip (from Python Package Index), or apt (for software package management for Debian, a Linux kernel-based operating system). Using the magic command of jupyter notebooks, namely: "!pip install name_of_package" or "!sudo apt update" followed by "!sudo apt install name_of_package", “!” symbol in front of any command suggests that we want to use specific terminal/shell commands. When the runtime session is started, all installed software packages and output files generated in the notebook will be stored in 73.1GB of Disk memory, also 13GB of RAM, a dual-core unknown CPU (Central Processing Unit), and an Nvidia Tesla P100 GPU (Graphics Processor Unit) accelerator with 16GB of video memory is available for all computing power. If a TPU (Tensor Processing Unit) is needed, the resources provided will be larger and instructions will be required for each piece of code that will be processed by a TPU otherwise it will be processed by the CPU and the working time will be exponentially bigger.

GPUs, in general, are a lot faster than CPUs for numerical computations, for the simple fact that the images are represented in the form of matrices, the algorithm employed was

intended to meet the needs of the application and to take advantage of the GPU architecture's tremendous data parallelism [32]. Meanwhile, relevant improvements are suggested to adjust for some of the selected algorithms' intrinsic shortcomings. Workloads must be partitioned and calculated in parallel on flow processing to get excellent performance on GPUs. Nvidia's CUDA is a parallel computing platform and programming style designed for generic computing on its own GPUs. CUDA allows developers to accelerate compute-intensive applications by utilizing the capabilities of GPUs for the parallelizable portion of the calculation.

4. ML Model

U-Net architecture is based on convolutional neural network whose primary goal is to semantic segmentation in medical image scans. Convolutional neural networks (CNNs) will be thoroughly described in this chapter. First, neural networks will be discussed, followed by some particular approaches employed in CNNs. I'll go over how CNNs are utilized to solve classification and segmentation issues, as well as the architecture used for pixel-wise classification. Convolutional neural network architecture is intended to make use of the benefits of a 2D grid input, which is useful for image processing. Figure 9 depicts the CNN architecture employed in this work and how an area is detected into an image. It is made up of two convolutional layers and two mean-pooling layers that are alternated; 24 convolutional filters with kernel size 5x5 are utilized in both convolutional layers. The network's input is a 4xdxd image patch (four MR sequences are included in multisequence volumes), and its output is an array of length N denoting membership in one of the N classes inside the label patched dictionary.

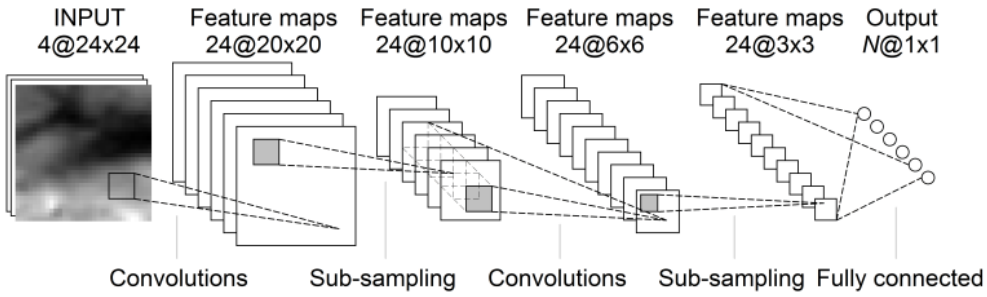


Figure 9: Convolutional neural network architecture for d = 24 [19].

4.1 Artificial Neural Networks

CPUs have long outperformed the human brain in complicated calculations per second, like floating-point computations. A single CPU core may do over five billion floating-point calculations. Such intricate procedures may be hard to compute for a person. The human brain is still thought to be far more powerful than any computer. Although the human brain cannot do sophisticated mathematical calculations, it can perform many basic operations considerably more efficiently than a CPU and is nevertheless deemed better successful in tasks such as picture categorization. Since the 1940s, scientists have attempted to mimic the human brain using computers, and a concept known as Artificial Neural Networks (ANNs) has developed. Artificial neurons are biological neurons' clones. An electrical signal is generated by a biological neuron. If its dendrites are stimulated by a sufficient number of synapses from other neurons, it generates an electrical signal that is delivered by its axon and stimulates dendrites of other neurons via its synapse. Artificial neurons, like real neurons, are linked together and stimulate one another via their connections. Perceptrons in their simplest form, can only calculate a very simple function known as neuron activation. A perceptron is just a single-layer neural network; activation functions are primarily employed for each layer of a neural network to add non-linearities; without activation functions, we will be computing a linear function regardless of how deep the neural network is. According to Figure 10: blue nodes represent the input nodes. Each of these nodes denotes a characteristic of the input problem. There are $_m$ input features in the diagram below. The i-th input feature is indicated by x_i, this layer is referred to as the input layer, and real values can only be used as inputs. Next, we have the red arrows, link the input nodes to the orange node. The synapses are the connections between neurons. Each of these synapses has a weight associated with it, which is represented by the w_i. The weight w_i is assigned to the i-th synapse, which is referred to as the weights layer. The orange node y it's the node with the output. It creates a score using the inputs and weights, then utilizes an activation function to provide a prediction of 0 or 1 based on that score. We'll need to utilize an activation function to produce a forecast based on this calculated score. Activation functions are primarily employed for each layer of a neural network to add non-linearities, without them, we would be calculating a linear function no matter how profound the neural network is. For classification tasks, we utilize an activation function in the output layer to link the scores to classes [13].

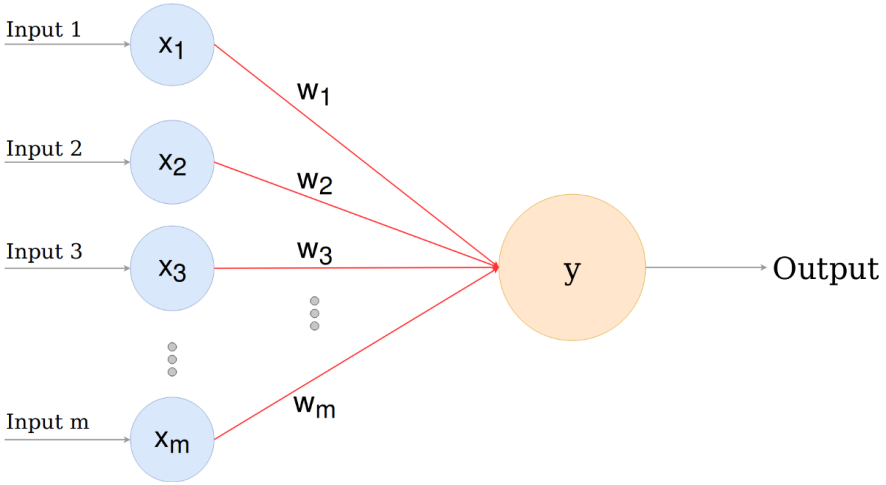


Figure 10: A single perceptron [13].

A convolution neural network has a three-dimensional layout of neurons, instead of two. A convolutional layer is the first layer, each convolutional layer neuron, only analyzes data from a limited area of the visual field. Input features are stored in batches, like a filter. The network decodes images in chunks and can perform these actions numerous times to complete the entire image processing. The image is converted from colors (RGB, CMYK or HSI) to greyscale (binary) during processing. Further pixel value fluctuations will help in the recognition of edges, allowing pictures to be classified into various categories. The output of the convolution layer goes to a fully connected neural network for classification, as shown in the above diagram. As indicated in the picture above, propagation is unidirectional when a convolutional neural network has one or more convolutional layers backed by pooling, and bidirectional when the output of the convolution layer has been sent to a fully connected neural network for classification. To extract certain sections of an image, filters are utilized. In Multi-layer perceptron (MLP), inputs are weighted and supplied into the activation function. Rectified linear unit (RELU) is used in convolution, while MLP employs a nonlinear activation function accompanied with softmax. In picture and video identification, semantic parsing, and paraphrase detection, convolutional neural networks produce excellent results. And they have advantages and disadvantages, the first advantage in using this type of network would be the fact that it requires few parameters in Deep Learning applications. The second advantage is that fewer parameters are required to learn compared to the fully connected layer. And the downside is that CNNs are pretty slow, depending on the number of hidden layers used. The second disadvantage is that they are pretty complex to design and maintain.

The softmax function reduces an array of K real values to an array of K real values that add up to 1. The softmax converts positive, negative, zero, or greater-than-one input values into values between 0 and 1, allowing them to be understood as probabilities. If one of the inputs is low or negative, the softmax converts it to a small probability, if one of the inputs is high, it becomes a large probability, it will always be between 0 and 1. The softmax function is also known as the softargmax function or the multi-class logistic regression function. This is

due to the fact that the softmax is just a generalization of logistic regression it can be used for multi class classification and is very similar to the sigmoid function used in logistic regression. Once the classes are classes are mutually exclusive. can the softmax function be used for a classifier. Many multi-layer neural networks terminate with a penultimate layer that produces real-valued scores which are not easily scaled and can be difficult to work with. In this case, the softmax comes in handy since it generates the scores into a normalized probability distribution that can be presented to the user or used as input to other systems. As a consequence, it is common practice to include a softmax function as the neural network's final layer [14].

Non-linear activation functions such as sigmoid functions (or logistic) and hyperbolic tangent are commonly used in neural networks to generate activation values for each neuron. In traditional neural network paradigms, the ReLu function has been used to calculate activation values instead. The reason for replacing sigmoid and hyperbolic tangents with ReLu is to save computation. Because the derivative of ReLu is 1 for positive input, it can accelerate deep neural network training speed when compared to traditional activation functions. Deep neural networks need not spend extra time during the training phase because of a constant [15].

The perceptron is extremely useful for classifying data collections that can be separated linearly. They run into serious problems with data collections that do not follow this pattern, as demonstrated by the XOR problem. The XOR problem demonstrates that there exists a set of four points that are not linearly separable for any classification of four points. The multilayer perceptron known as MLPs overcomes this limitation by classifying datasets that are not linearly separable. They accomplish this by learning regression and classification models for difficult data using a more robust and complex architecture [16]. The Perceptron is made up of two fully connected layers: input and output. MLPs have the same input and output layers, but also have multiple hidden layers in between, as shown in Figure 11. The MLP's algorithm is as follows: The MLP, like the perceptron, pushes inputs forward by taking the dot product of the input and the weights that exist between the input layer and the hidden layer (W---H). At the hidden layer, this dot product produces a value. MLPs employ activation functions at each calculated layer. There are numerous activation functions to consider, including ReLU, the sigmoid function, and tanh. Some of these activation functions can be used to pass the determined output at the current layer. Once calculated output at the hidden layer has been forced through the activation function, take the dot product with the corresponding weights and push it to the next layer in the MLP. The calculations will be used at the output layer for either a backpropagation algorithm that corresponds to the activation function that was chosen for the MLP, for training or a decision will be made based on the output, for testing. MLPs serve as the foundation for all neural networks and have significantly increased computer power when utilized for regression and classification problems. Because of the multilayer perceptron, computers are no longer restricted by XOR case scenarios, and they can learn rich and complex models [16].

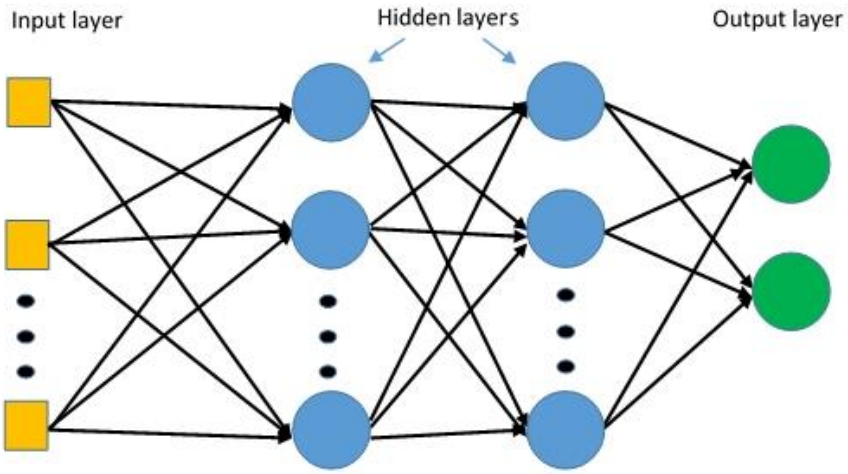


Figure 11: Multilayer perceptron [16].

4.2 U-Net

U-Net architecture is based on convolutional neural network whose primary goal is to segment medical images. It uses n convolution layers, n pooling layers, and n deconvolution layers to reconstruct a region-based output with the same dimensions as the input. It progressively downsamples the input in order to discover its deep features. Then it deconvolutionally up samples those features and concatenates the upsampling outcomes channel-wise with features that have the same dimensions. U-Net architecture is a more advanced, in-depth version of FCN-8s [26] which is the same as U-Net a fully convolutional neural network. An example of U-Net architecture can be seen in Figure 12. Although the profound features have a global meaning, they lack the local information required for voxel-level classification. The local information is provided by reintroducing the outputs of subsurface depth via concatenation. They are also similar to skip connections, with the exception that skip connections are connected as expansions rather than concatenations. Consider Figure 12, where the features at the bottom of the U-shape are challenging and represent global data. Complex features are combined with local information by trying to introduce the simpler features obtained inside the encoder to the decoder. In other words, the encoder is used first to find complex global features, and then the decoder information is merged with all these features for localization. At the upper side of the decoder, 120 features with the same spatial dimensions as the input are obtained, implying that each pixel has 120 features of its own. The number of layers in a U-Net design might vary depending on the input dimension and task complexity. There are several changes that can have an impact on network performance. New connections and connecting routes, filter size, input size, and training methods are among them. Although CNN-based semantic segmentation algorithms have been shown to outperform traditional approaches, but they still require improvements.

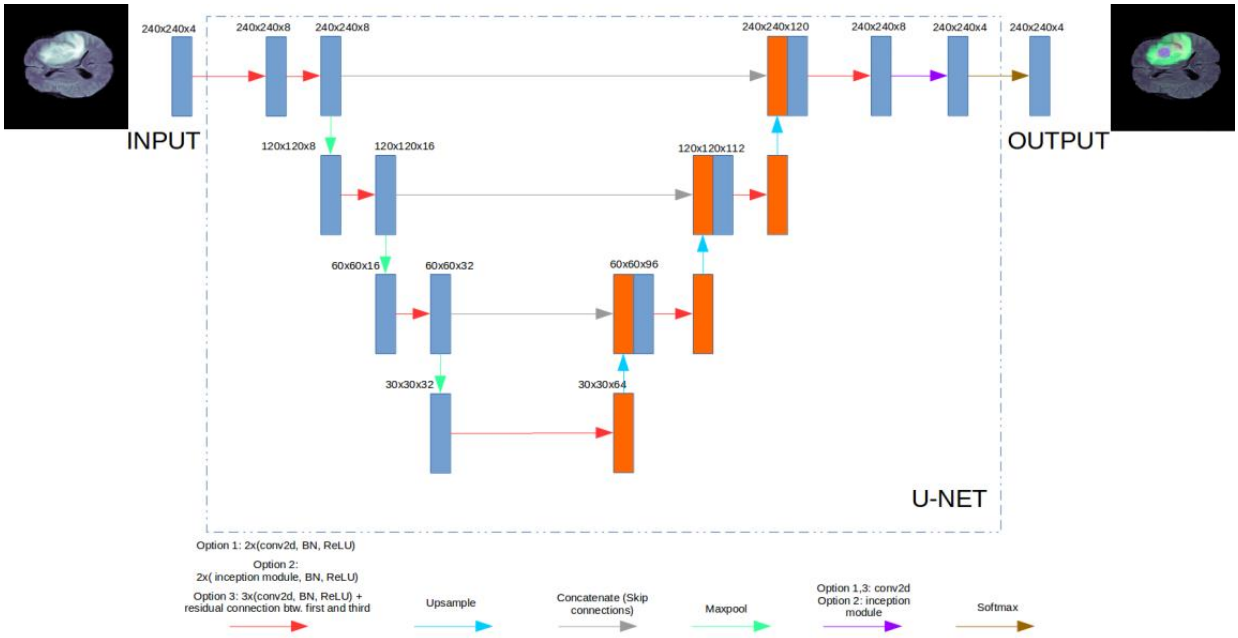


Figure 12: U-Net architecture chart used in semantic segmentation problems, including segmentation of brain tumor areas.

U-Net can take custom structures, such as classical Single-Stream or Multi-Stream. The Single-Stream U-Net has been implemented as shown in Figures 12 and 13. The network takes all modalities as inputs in four channels and outputs four channels, each providing the probability of a pixel belonging to a class. In Figure 11 we have also represented the number of filters and transposed convolutions utilized in each convolutional block.

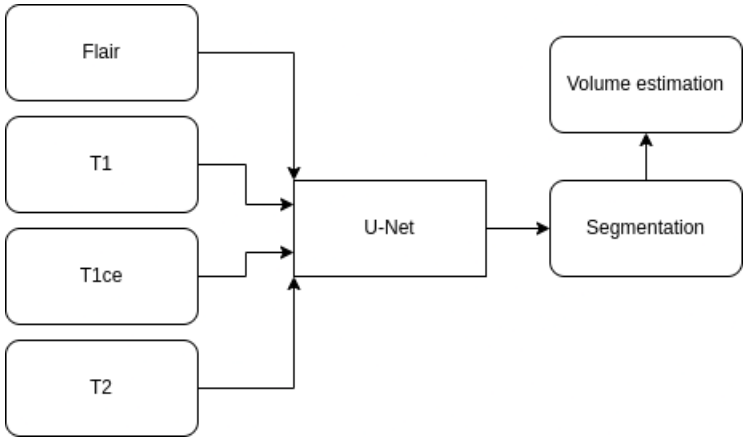


Figure 13: Single-Stream U-Net.

A Multi-Stream U-Net is the unique architecture introduced in this thesis. Each of the four modalities is input into a smaller U-Net before being concatenated and supplied into a

final convolutional block [33]. That processes each modality individually before combining the findings. This is accomplished with the U-Net design and the amount of filters and transposed convolutions shown in Figure 11. The last convolution and the softmax layer are deleted, and also the four outputs of the U-Nets are concatenated and then passed through a final convolutional block, which varies from the Single-Stream U-Net design, an example of this can be seen in Figure 14. The notion is that much of its independent information processing as possible is performed prior to layer merging. All other hyperparameters remain unchanged from the Single-Stream U-Net.

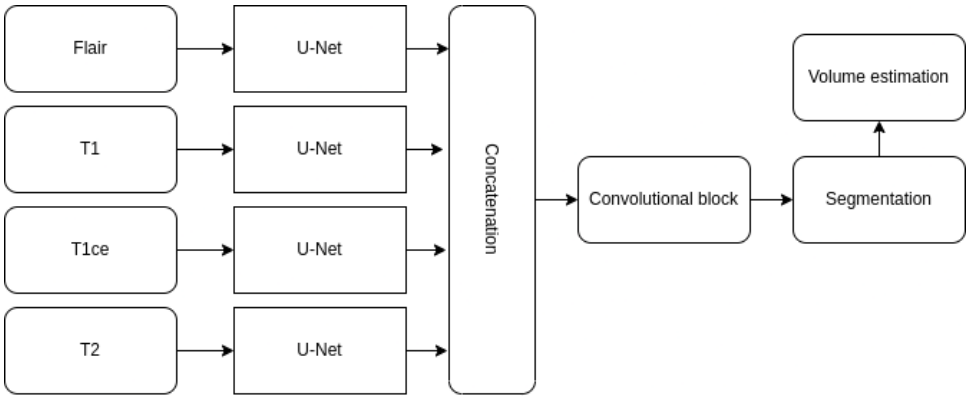


Figure 14: Multi-Stream U-Net.

4.3 Model Training and Validation

Overfitting is a major issue when training machine learning models. On the other hand, underfitting is defined as a model that cannot both model the training data and generalize to new data. Overfitting occurs when a model begins remembering the training data rather than generalizing, an example of overfitting can be seen in Figure 15. When drawing inference on out-of-sample data, memorizing the training data causes the model to perform poorly. The data was obtained by adding noise to a second-degree polynomial and then fit with three distinct polynomials. The first and last polynomials underfit and overfit the data. This creates a dilemma when evaluating models since the correct model will have a higher score in terms of mean root squared error but will most likely perform worse on the out-of-sample data point. Regularization is a broad word for strategies that try to reduce model overfitting.

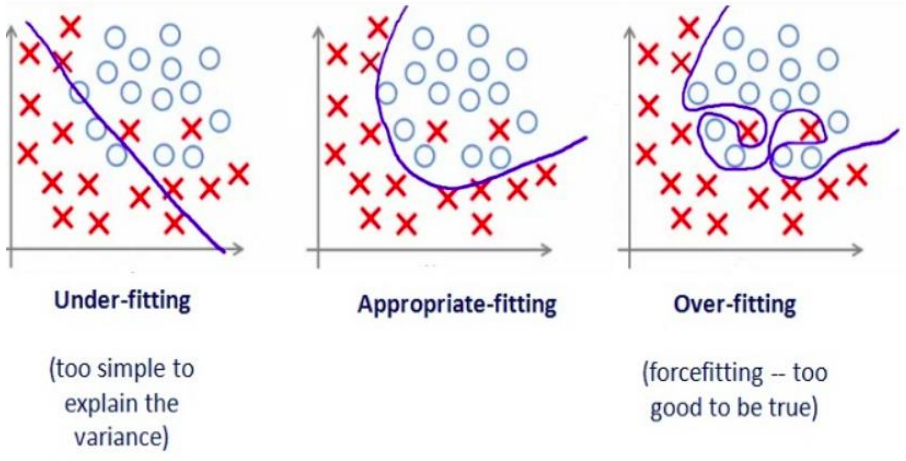


Figure 15: Robustness of fit [17].

Early stopping is a strategy in which training is halted due to a rise in the loss function on the validation data set across a preset number of epochs, this method can be found in the software library Keras, this fixed number of epochs is referred to as patience. This approach works since an increase inside the loss function when tested on validation data indicates that the model has ceased generalizing and has begun remembering the training data. So, when validation loss begins to rise, it typically indicates that the model has begun to overfit. It is advised to cease training the model at this point.

Data augmentation is a regularization technique in which the network's input data is altered in various ways. This is done to increase the amount of training examples while decreasing overfitting. A skull stripped MR picture of the brain that has been rendered, flipped, and rotated in various ways. Data augmentation examples include zooming, adding noise to data, shearing transformations, rotational transformations, and vertical/horizontal flips. The key principle is that one creates additional data in a synthetic fashion. As a result, the effective size of the dataset grows, lowering the danger of overfitting.

Adam optimization is an approach based on stochastic gradient descent consisting of the adaptive estimate of first and second-order moments. Is a popular deep learning method since it produces good results quickly. Adam differs from typical stochastic gradient descent. Stochastic gradient descent employs a single learning rate that does not change throughout training for all weight updates. Each network parameter (weight) has its own learning rate that is adjusted as learning progresses. Adam is described as integrating the benefits of two prior stochastic gradient descent improvements [18]:

- Root Mean Square Propagation (RMSProp), which additionally maintains per-parameter learning rates average of recent magnitudes of gradients for the weights, how quickly it is changing. This implies that the method performs well on both online and non-stationary issues, noisy;

– AdaGrad is an adaptive gradient algorithm that maintains a per-parameter learning rate to enhance performance in situations with sparse gradients, for natural language processing and computer vision problems.

Adam recognizes the advantages of both RMSProp and AdaGrad, and instead of changing the parameter learning rates involving the average first moment (the mean) as RMSProp does, Adam additionally uses the average of the gradients' second moments (the uncentered variance). The technique computes an exponential moving average of the gradient and the squared gradient, with the decay rates of these moving averages controlled by the constants beta1 and beta2. Moving averages with beginning values near to 1.0, beta1 and beta2 values around 1.0 (recommended) resulted in a bias of moment estimations towards zero [18]. This prejudice is eliminated by first computing biased estimates, followed by bias-corrected estimates.

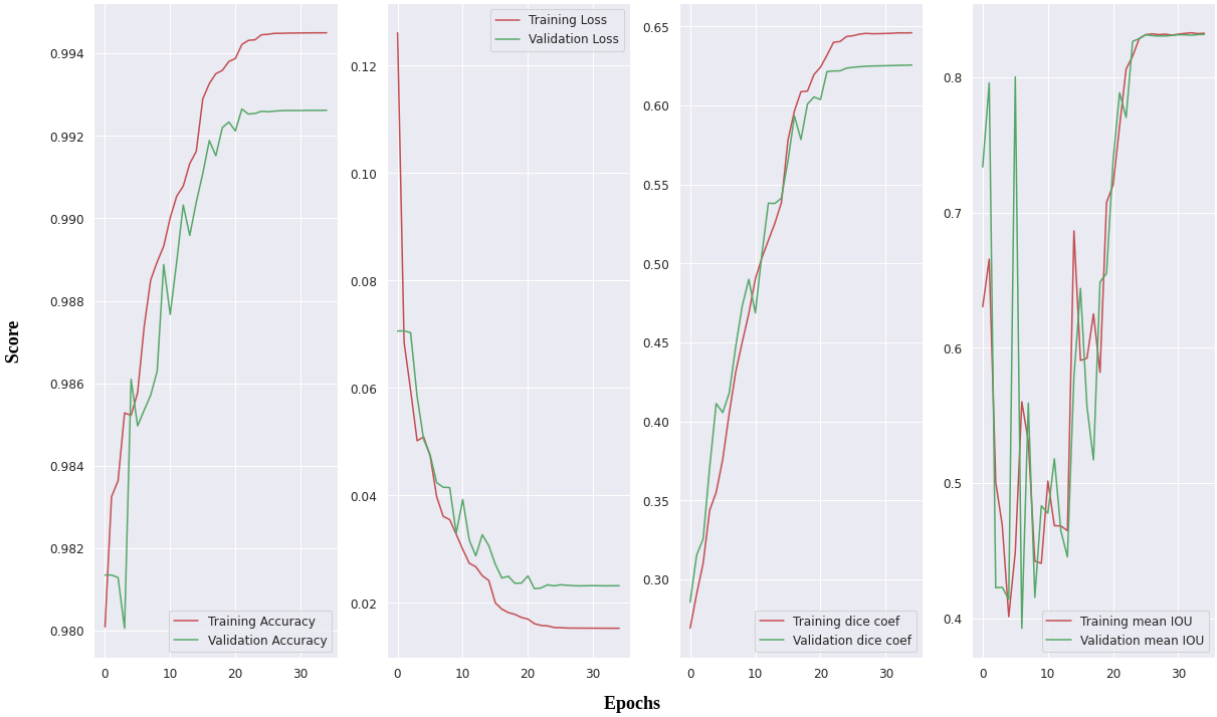
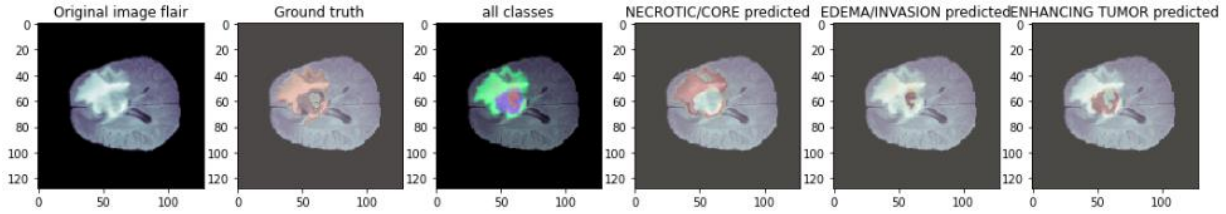
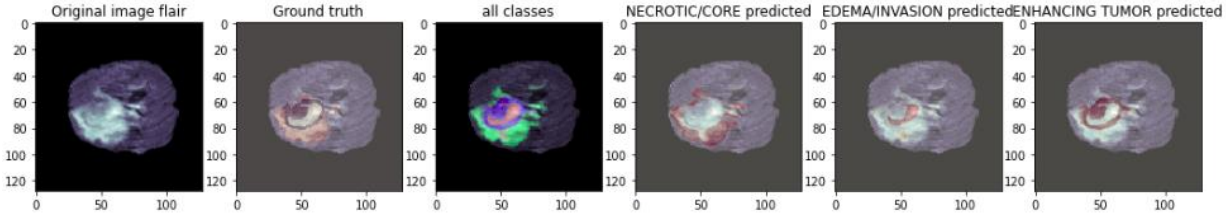


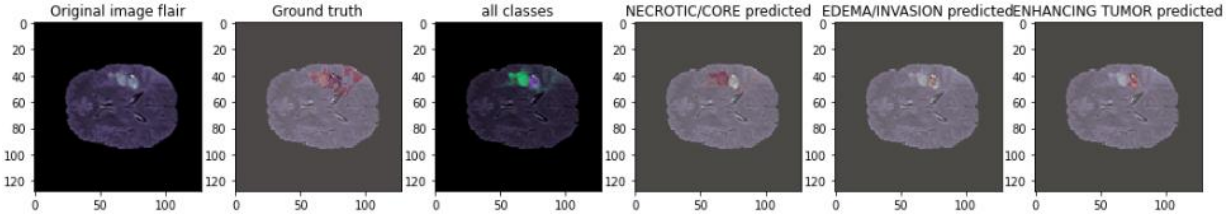
Figure 16: Performance of training process.



(a)



(b)



(c)

Figure 17: Three prediction examples (a, b, c) were detected positively, in which the segmentation of the tumor areas are displayed.

The training process outcomes are represented graphically using graphs of training and validation loss and accuracy for learning rate, dice coefficient, and IoU (Intersection over Union) approaches. IoU is simply an assessment metric for determining the accuracy of a detection algorithm on a given dataset. The graphs presented in Figure 16 for network training and validation are a function of 35 epochs and scores (accuracy and loss), respectively in the following figure some examples of the segmentation performed on all classes during the training. Analyzing Figure 16, we can see that the accuracy of training is higher than 0.994 and on validation is just below 0.993, this difference is normal given that the training set contains much more data than the validation. Overall, for accuracy, the percentage obtained is very good, with no traces of overfitting or underfitting. In machine learning, an epoch is defined as one complete run of the training dataset through the algorithm. This number of epochs is a critical hyperparameter for the algorithm. On the training set, the percentage of accuracy constantly grew as we progressed from one epoch to the next, however on the validation set, we had a few spikes owing to the smaller dataset. We can say that in the case of loss the result is similar to accuracy.

For both training and validation, the fundamental goal is to have a loss curve that decays with the number of epochs. Because the loss function is a direct result of the network's learning process, it should be valued above all other metrics during training. Furthermore, we find that our strategy consistently reduces standard deviations in all training situations and tumor subregions. The loss for the next three charts should generally be between 0 and 1. Our dice ratio is good for our multi-class problem, but a bad one in a binary bias case. For the last chart, the interpretation of IoU is quite simple. A score as close as possible to 1 indicates that the designed boundary box exactly matches the true boundary box. A score as close as possible to 0 indicates that the anticipated and true boundary boxes do not overlap. At the end of each epoch, the training loop will check if the loss is further reduced. We can see that from the epoch of 25, the process of deep learning becomes exhaustive, and our model can no longer learn effectively, and the whole process will end with the end of the epoch of 35, in order to save computational power and time. Taking into account the IoU score it ended around 0.85 at the end of the 35 epochs, the segmentations performed in the training session seen in Figure 17, are quite accurate on all tumor regions compared to the original flair image and ground truth. The necrotic tumor core, invasion, and enhancing tumor had to be clearly segmented individually, with fairly high accuracy, and then be superimposed in a single image so that the output could be compared to the overlay mask that contains all classes.

4.4 Model Evaluation

After evaluating the U-Net model on the dataset for testing, the results obtained were close to those obtained on validation, according to the comparison of the results in Figure 16 with those in Table 1 and the first part of Table 2. As mentioned in the data section, the compilation of metrics was done using categorical cross-entropy, and for the evaluation of the model: data generators were created in the data processing section with a batch size of 128. These scores are consistent with those of the quantitative results. an IoU score of 0.87 segmentation of areas of the tumor should be quite similar to the previous phase, now I'll demonstrate the evolution of each area of the tumor (the three classes: tumor core, edema or invasion, and the more intensified areas alias enhanced tumor, areas that may include the actual core) compared to the original segmentation performed by specialized medical personnel (ground truth), as can be seen in Figure 18-a and 18-b and highlighting areas that were not affected by the tumor, namely reverse segmentation, because anything else represents the pixel is not on a tumor region, simply shows us the white region where there are no tumors, the darker area represents a tumor. We randomly selected five patients from whom we took the original tumor segmentation MRI (Ground Truth) made by specialists and compared it with each prediction of tumor subregions made by the U-Net model. The subregions of the tumor were not overlapped in a single image as in the training session, in order to be able to make easier comparisons of the result obtained by the model. I noticed that in the patient with number 3, the ground truth mask that is observable with the naked eye is a very small one and according to the “Not Tumor” class prediction I discovered that

the affected area is much larger (black circle and red box). The predicted class of the tumor "Core" being surrounded by a red box being magnified so that it can be observed, and it does not correspond to the ground truth, is located below. The semantic segmentation of the "Edema" class shows the regions that will be invaded by the tumor, these being magnified can be seen clearly in the blue box. No problems were observed in the other three patients (1,2,4,5), the segmentation has a good clarity and tumor subregions can be easily observed.

Metric	Loss	Accuracy	Mean IoU	Precision	Sensitivity
Score	0.0217	0.9928 (99.2%)	0.8275	0.9933	0.9913

Table 1: Metric scores obtained after testing.

Metric	Specificity	Dice Coefficient	Dice Coef. For Necrotic (Tumor Core)	Dice Coef. For Edema (Tumor Invasion)	Dice Coef. For Enhancing Tumor
Score	0.9977	0.6232	0.6766	0.7024	0.6140

Table 2: An extension of table 1, the evaluation being made on each individual class.

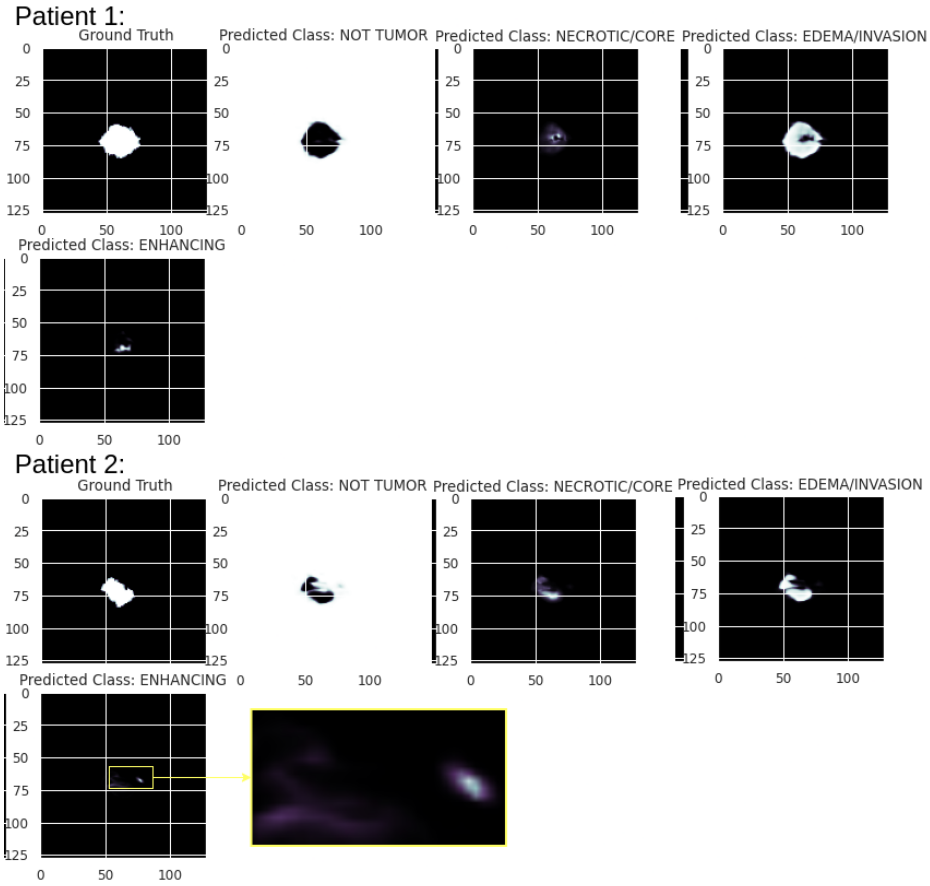


Figure 18-a: A comparison between ground truth and the four classes, in order to clearly observe the tumor subregions.

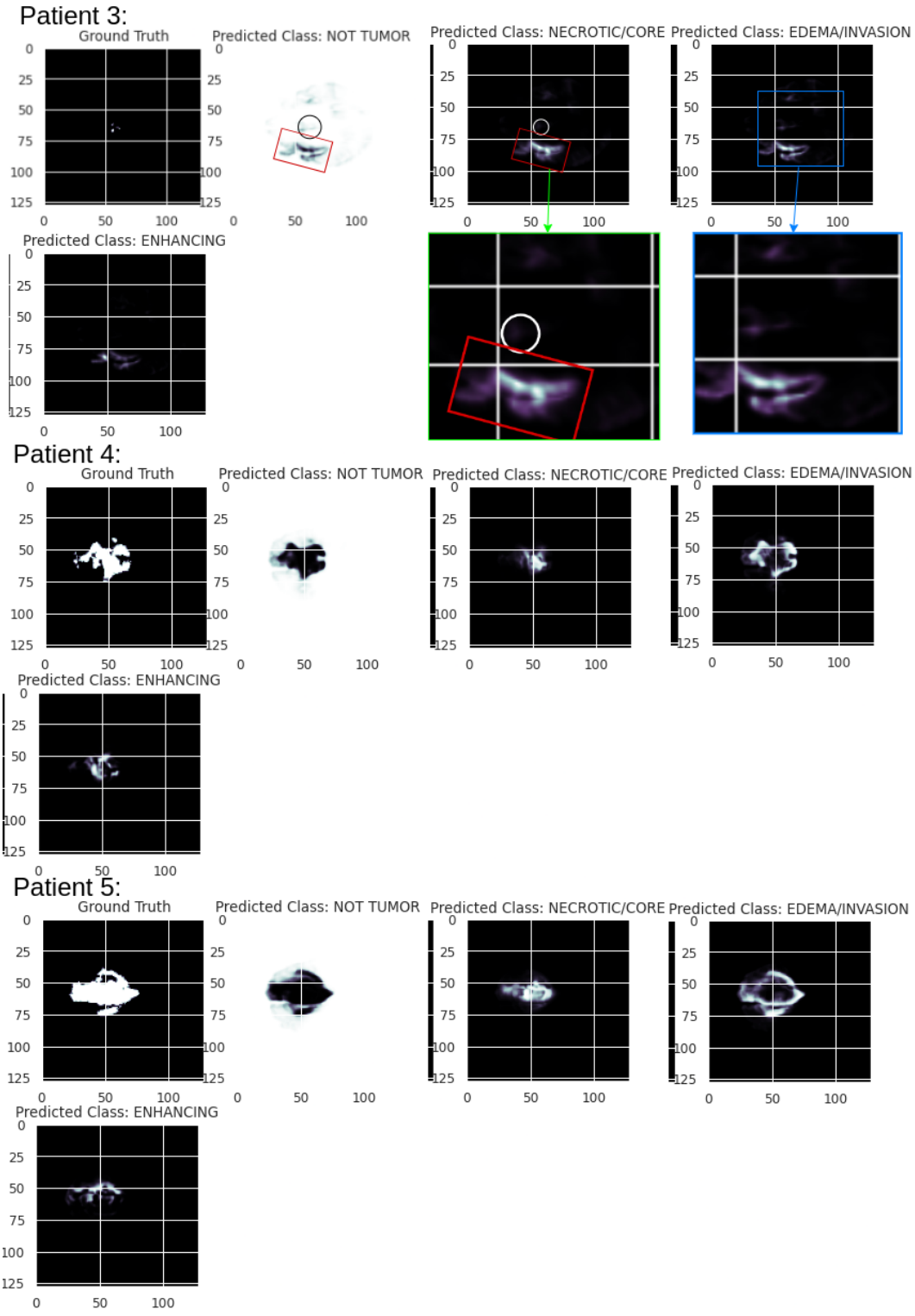


Figure 18-b: A comparison between ground truth and the four classes, in order to clearly observe the tumor subregions.

5. Patient Survival Estimation

Accurate and robust predictions of overall survival for patients with tumors using automated algorithms can provide valuable suggestions for diagnosis, treatment planning, and outcome prediction, regrettably, it is hard to determine dependable and effective predictive features in an attempt to maximize the patient's life by a few months or perhaps a few years. Clinical data, such as patient age and resection status, can potentially give useful information regarding a patient's prognosis, all patients are grouped on gross total resection of gliomas (GTR). Tumor sub-regions are segmented during the first approach using a biomedical image segmentation model based on the U-Net convolutional neural network architecture promising robust performance. Despite the fact that the problem looks like a regression one (trying to predict a number), I decided to make another classification, namely, I created three categories that will define the estimation of survival time. The first constant is "SHORT" and includes a time interval between 0 and 365 days, the second constant is "MEDIUM" with an interval between 365 and 730 days, and the last one is "LONG" comprising all patients with the survival of more than 730 days. Then, features are collected from tumor sub-regions and whole tumor volume, and the three new categories are added to the working data frame, thereby replacing the survival days column. After the exploratory data analysis operations to see correlated objects, I decided to compare the performance of three classifiers in doing this prediction: Random Forest classifier, Support Vector Machine classifier, and K-Nearest Neighbors models. To fit the training data and rank the relevance of features based on variance reduction, decision trees with gradient boosting are utilized. Cross-Validation (CV) is used to determine the optimum number of top-ranking features to employ, while GridSearchCV is used to identify the best hyper-parameters from a pre-defined list. Computing segment sizes consists in finding the number of pixels for each class in volume, all images have their same original size 240x240, because for this procedure I did not rescale them to a lower resolution, no extra computational power is needed. Brain volume size is computed ignoring the background, so in the image foreground, we count the non-zero elements in the whole 3D volume. In Figure 19 we have a brain total volume size of 1,428,957 pixels for patient id 50, and for class 1 which is tumor core (necrotic) size of 844 pixels, class 2 is tumor invasion (edema) with 77 pixels, and class 3 is enhanced tumor with 917 pixels.

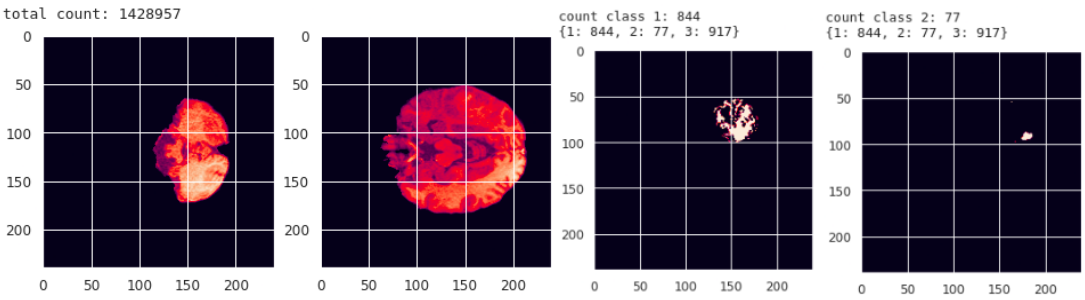


Figure 19: 3D Brain volume size with the size of predicted tumor core and invasion sub-regions.

5.1 EDA and Feature Selection

Quantitative information from MRI images can indicate brain tumor characteristics. We extract characteristics from the edema, non-enhancing core, and necrotic core based on the segmentation results and integrate them to obtain the whole tumor. The mode of feature extraction utilized is determined on the intrinsic features of the tumor subregion. Edema characteristics, for example, are derived from the FLAIR modality since it is often represented as a hyper-intense tone in FLAIR. Because the image of the necrotic (NCR) and non-enhancing (NET) tumor cores is often hypo-intense in T1ce as compared to T1, and non-enhancing solid core characteristics are retrieved from T1ce. T1ce is used to extract necrotic core tumor characteristics since they are defined by regions that demonstrate hyper-intensity in T1ce as compared to T1.

After performing the EDA in Figure 20 plotting a pairwise relationship in a data. This pair plot from Figure 20 creates a grid of axes such that each variable: age, necrotic/core, edema/invasion, and enhancing tumor will be divided on the y-axis across a single row and on the x-axis across a single column. We extracted certain characteristics that were redundant or unimportant to survival prediction. We utilized feature selection to choose from a subset of attributes with the best predictive ability to enhance performance and avoid overfitting. The most important features to find out the target (survival time: short, medium, or long) are the tumor subregions and the patient's age, and the redundant data that was removed is the patient ID, respectively "GTR" which is the same for all patients. Age, naturally, showed the greatest predictive ability of all the characteristics viewed in Figure 21.

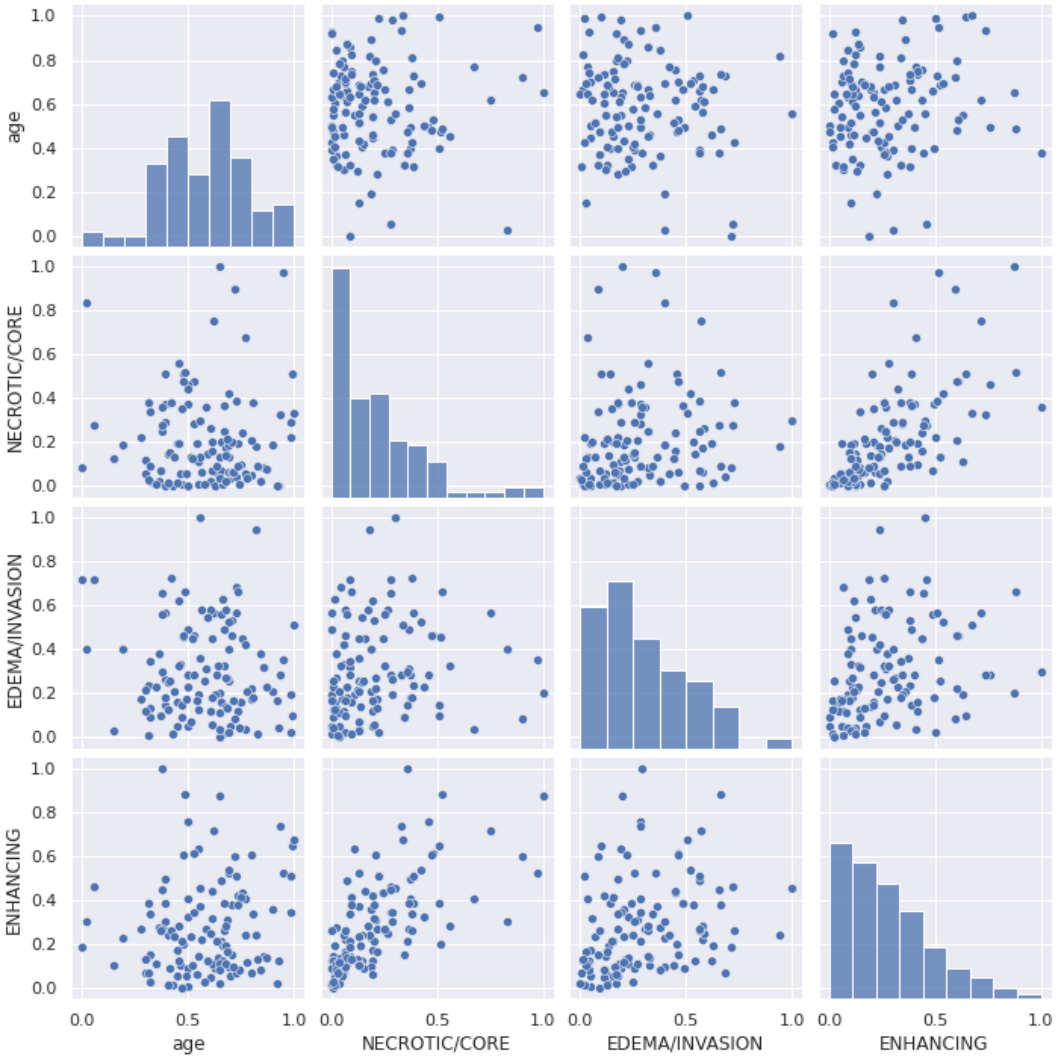


Figure 20: Seaborn pair plot of exploratory data analysis (EDA).

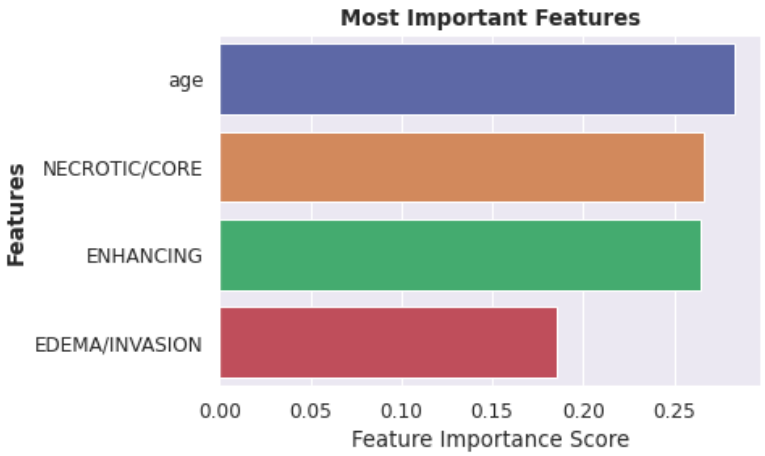


Figure 21: The most significant features.

5.2 Survival Prediction

Based on the importance of these features, we made a pandas data frame (Table 3) which I used to train and test a Random Forest Classifier (RF), a Support Vector Machine Classifier (SVM), and a K-Nearest Neighbors Classifier (KNN) using the best parameters from a grid of parameters, in the end, comparing their accuracy, in order to find the most suitable option for this problem. Analyzing the results obtained from the Table 4 and Figure 22, we can say that the SVM classifier is not a good fit for our problem, being a high complexity classifier is not a good option for our data frame which has only 119 patients, it can be seen that the test accuracy should not be higher than train since the model is optimized for the latter. Random forest and KNN are the estimators that obtained the most correct results, Random Forest is a bagging algorithm with a weak classifier is a decision tree and for this reason, the result is quite poor compared to KNN (the main advantage is that it can be used for multiclass classification) which proved to be the most suitable variant for this approach.

Index	AGE	TUMOR CORE	TUMOR EDEMA	ENHANCED TUMOR	SHORT	MEDIUM	LONG
0	54.915	0.002438	0.045368	0.005153	0	1	0
1	57.000	0.015202	0.039171	0.019636	0	0	1
2	60.000	0.004592	0.027417	0.030548	0	1	0
3	83.649	0.039530	0.048636	0.025146	1	0	0
4	60.019	0.000448	0.018200	0.007183	0	1	0

Table 3: Visualizing the first 5 entries from the patient survival data frame after removing redundant data.

Without hyper-parameters tunning	Score	Random Forest	Support Vector Machine	K-Nearest Neighbors
	Train accuracy	49.8%	47.9%	57.4%
	Test accuracy	33.3%	50.0%	50.0%
With hyper-parameters tunning (GridSearchCV)	Train accuracy	54.3%	58.4%	65.9%
	Test accuracy	45.8%	62.5%	54.2%

Table 4: Comparison of the accuracy of the three classifiers

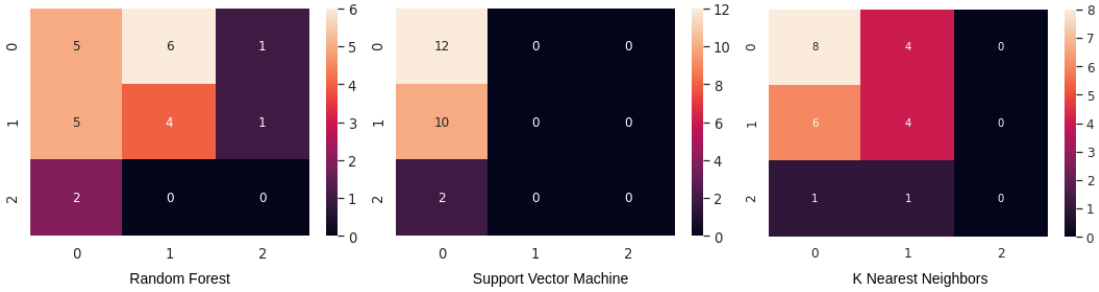


Figure 22: The confusion matrix of the three classifiers.

6. Conclusion and Future Work

The goal of this study is to provide a method that can handle brain tumor semantic segmentation in MRI images with the highest accuracy possible. To gain a better knowledge of the segmentation issue, we first analyzed brain MRI images, their many interpretations, and how they represent the presence of tumor tissues. After doing a literature analysis of methods used to segment brain tumors in MRI images and categorizing these approaches based on the algorithms used for segmentation, we used the CNN-based architecture model in our solution since the U-Net is considered to be the present state-of-the-art approach in the field of medical imaging segmentation. If brain tumor segmentation will to be employed in the future in a clinical relationship, it must be done with excellent precision. Machine learning segmentation might be utilized as a second judge in the segmentation job if high precision can be guaranteed. This technique is a helpful tool in assisting doctors in segmenting tumors and their sub-regions in the human brain and making a more accurate prediction of the patient's survival potential. Computational intelligence in performing these types of jobs is a long-term objective that seems to be to completely relieve doctors of this time-consuming process. The following findings may be drawn from the research presented in this thesis.

We developed a model based on U-Net architecture to segment MRI images using a pixel-wise classification procedure. Several experiments were run during the model's tuning phase, and the findings are being used to optimize the model's performance. The developed model demonstrated a maximum dice score coefficient of 0.62 for the entire tumor, 0.67 for the tumor core sub-region, 0.70 for the tumor invasion sub-region, and 0.61 for the enhanced tumor sub-region. We believe that these results can be improved further by using more data processing techniques, more approaches to the model architecture, and the collection of

more data, hyperparameter tuning, or more complex data manipulations, are just some of the methods that can improve the accuracy of this application. Patient survival prediction can also be improved by using more complex KNN-based architectures, also by collecting more data, and using data balancing techniques if they have large differences between targets.

The results of this study bring new suggestions for research and future development and may be used for various deeper U-Net architectures, which are typically more promising in terms of improving the accuracy of the segmented outcomes. In the presence of large datasets of MRI tumors, it is actually recommended to experiment with deeper neural network designs, i.e., the use of longer times for numerical processing. Because MRI scans are heterogeneous, the same tissue may exhibit different intensities depending on the MRI machine that captured the scan and the patient's health state; this fact has a significant impact on our algorithm's ability to detect tumors; consequently, adjusting the training set to make the same tissue look similar across different scans will greatly improve the algorithm's accuracy. Make generalizations the solution even further to perform significant clinical tasks like forecasting overall patient survival and whether the tumor is decreasing, increasing, or remaining stable in a more accurate and intuitive way. Development of a friendly user interface that is designed for clinical usage where healthcare workers may put the patient MRI scan into the network in order to get an automatic pre-processed file, adjust the parameters if needed, and obtain the prediction outcome from the system. The developed software thus becomes an assistant for the medical staff, helping them to treat more patients more efficiently in a faster time.

Finally, I would like to thank my family for all the support, as well as my scientific coordinator, professor Liliana Dobrică, for providing the necessary feedback and continuous supervision throughout the work.

7. References

1. Barbara Z. and Jan F.: Image registration methods: a survey in image and vision computing, pp. 977–1000, 2003.
2. Cohen I., Cohen L., Ayache N.: Using deformable surfaces to segment 3D images and infer differential structures. Computer vision: graphics, image processing and image understanding, 242-263, 1992.
3. Collins L., Peters T., Dai W.: Model based segmentation of individual brain structures from MRI data, Visualization in biomedical computing (VBC), 10-23, 1992.

4. Dhawan P., Juvvadi S.: Knowledge-based analysis and understanding of medical images, Computer Methods, and Programs in Biomedicine, 21-239, 1991.
5. Erik Meijering: Quantitative Comparison of Sinc-Approximating Kernels for Medical Image Interpolation in MICCAI (Medical Image Computing and Computer-Assisted Intervention), Springer, pp. 210–217, 1999.
6. Foley D., Van Dam A., Feiner S.F., Hughes J.: Computer graphics - Principles and practice, New York, 541-551, 836-838, 1990.
7. Haralick R., Stemberg S., Zhuang X.: Image analysis using mathematical morphology. IEEE Transactions on Pattern Analysis and Machine Intelligence, 532-550, 1987.
8. Hoehne K.R., Bernstein R.: Shading 3D images from CT using grey-level gradients, IEEE Trans, Medical Imaging, 45-47, 1986.
9. Hoffmann D., Benabid L., Cinquin P., Lavallee S.: Potential use of robots in endoscopic neurosurgery, 93-97, 1992.
10. https://www.researchgate.net/Difference-between-a-MRI-scan-and-a-CT-scan_fig2_339429935, Accessed: 2022.
11. https://healthcare-in-europe.com/media/story_section_image/4598/image-01-radiology-brain-tumour-van-cauter.jpg, Accessed: 2022.
12. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy, Accessed: 2022.
13. https://towardsdatascience.com/the-perceptron-3af34c84838c, Accessed: 2022.
14. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, Accessed: 2022.
15. https://deepai.org/machine-learning-glossary-and-terms/relu, Accessed: 2022.
16. https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron, Accessed: 2022.
17. https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76, Accessed: 2022.
18. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, Accessed: 2022.
19. https://docs.ecognition.com/Resources/Images/ECogUsr/UG_CNN_scheme.png, Accessed: 2022.
20. Illingworth J., Kittler J.: A survey of the Hough transform, Computer Vision, Graphics and Image Processing 44-87, 1988.
21. Jain K., Farrokhnia F.: Unsupervised texture segmentation using gabor filters, Pattern Recognition 1167- 1186, 1991.
22. Kippenhan J., Barker W., Pascal S.: Evaluation of a neural network classifier for PET scans of normal and Alzheimer's disease subjects. The J. of Nuclear Medicine 1459-1467, 1992.
23. Kuhn M., Brendergast D.: COVIRA - Computer Vision in Radiology: Advances in medical informatics: results of the AIM exploratory action, ed. Jaap Noothoven van Goor et al. (Studies in health technology and informatics), pp. 88-101, 1992.
24. Marc Levoy: Volume Rendering, EEE Computer Graphics Applications 10, 33–40, 1990.
25. NiBabel, DOI 10.5281/zenodo.4295521.

26. Ozan and Schlemper: Attention U-Net - Learning Where to Look for the Pancreas, Computer Vision and Pattern Recognition, pp 4-10, 2018.
27. Pellegrini C., Hu Z., Pun T.: An expert system for guiding image segmentation. Computed Medical Imaging and Graph, 13-24, 1990.
28. Pellizzari A.: Accurate 3D registration of CT and MR images of the brain, Computerized Assisted Tomography, 20-26, 1989.
29. Sanjiv Gambhir: Molecular imaging of cancer with positron emission tomography, Nature Reviews Cancer, vol. 2, 683, 2002.
30. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
31. Scott Atlas: Magnetic resonance imaging of the brain and spine, 11, 2009.
32. Volodymyr Kindratenko: Numerical Computations with GPUs, ISBN 978-3-319-06548-9, Springer, pp. 39 - 405, 2014.
33. Yucheng S., Jing Z., Bin X., Weisheng Li: Medical image segmentation based on active fusion-transduction of multi stream features, Volume 220, pp 5-10, 2021.
34. Zhuxian Z., Zheng R. L.: Gadolinium-based contrast agents for magnetic resonance cancer imaging, 1-18, 2013.

8. Abbreviations

AI - Artificial Intelligence
ANN - Artificial Neural Network
API – Application Programming Interface
CNN - Convolutional Neural Networks
CPU - Central Processing Unit
CSV – Comma Separated Values
CT - Computed Tomography
DL - Deep Learning
FLAIR - Fluid Attenuated Inversion Recovery
GPU - Graphics Processing Unit
GUI - Graphical User Interface
KNN - K Nearest Neighbors
ML - Machine Learning
MRI - Magnetic Resonance Imaging
NN - Neural Network
PET - Positron Emission Tomography
RF - Random Forest
SVM - Support Vector Machine
TPU - Tensor Processing Unit
UX - User Experience

9. Annexes

The whole Jupyter Notebook code along with markdown annotations and cell outputs can be seen on GitHub in my public repository: https://github.com/cristinelpopescu/ml-services-in-healthcare. Some screenshots of implementation can be also seen below:

```python
# Dice loss as defined above for 4 classes
def dice_coef(y_true, y_pred, smooth=1.0):
    class_num = 4
    for i in range(class_num):
        y_true_f = K.flatten(y_true[:,:,:,i])
        y_pred_f = K.flatten(y_pred[:,:,:,i])
        intersection = K.sum(y_true_f * y_pred_f)
        loss = ((2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth))
        if i == 0:
            total_loss = loss
        else:
            total_loss = total_loss + loss
    total_loss = total_loss / class_num
    return total_loss



# Define per class evaluation of dice coef
def dice_coef_necrotic(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,1] * y_pred[:,:,:,1]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,1])) + K.sum(K.square(y_pred[:,:,:,1])) + epsilon)

def dice_coef_edema(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,2] * y_pred[:,:,:,2]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,2])) + K.sum(K.square(y_pred[:,:,:,2])) + epsilon)

def dice_coef_enhancing(y_true, y_pred, epsilon=1e-6):
    intersection = K.sum(K.abs(y_true[:,:,:,3] * y_pred[:,:,:,3]))
    return (2. * intersection) / (K.sum(K.square(y_true[:,:,:,3])) + K.sum(K.square(y_pred[:,:,:,3])) + epsilon)



# Computing Precision
def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision


# Computing Sensitivity
def sensitivity(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    return true_positives / (possible_positives + K.epsilon())


# Computing Specificity
def specificity(y_true, y_pred):
    true_negatives = K.sum(K.round(K.clip((1-y_true) * (1-y_pred), 0, 1)))
    possible_negatives = K.sum(K.round(K.clip(1-y_true, 0, 1)))
    return true_negatives / (possible_negatives + K.epsilon())
```

```python
def build_unet(inputs, ker_init, dropout):
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(inputs)
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv1)

    pool = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(pool)
    conv = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv)

    pool1 = MaxPooling2D(pool_size=(2, 2))(conv)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(pool1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv2)

    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(pool2)
    conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv3)


    pool4 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(pool4)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv5)
    drop5 = Dropout(dropout)(conv5)

    up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(drop5))
    merge7 = concatenate([conv3,up7], axis = 3)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(merge7)
    conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv7)

    up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv7))
    merge8 = concatenate([conv2,up8], axis = 3)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(merge8)
    conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv8)

    up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv8))
    merge9 = concatenate([conv,up9], axis = 3)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(merge9)
    conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv9)

    up = Conv2D(32, 2, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(UpSampling2D(size = (2,2))(conv9))
    merge = concatenate([conv1,up], axis = 3)
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(merge)
    conv = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer = ker_init)(conv)

    conv10 = Conv2D(4, (1,1), activation = 'softmax')(conv)

    return Model(inputs = inputs, outputs = conv10)

input_layer = Input((IMG_SIZE, IMG_SIZE, 2))

model = build_unet(input_layer, 'he_normal', 0.2)
model.compile(loss="categorical_crossentropy", optimizer=keras.optimizers.Adam(learning_rate=0.001), metrics =
['accuracy',tf.keras.metrics.MeanIoU(num_classes=4), dice_coef, precision, sensitivity, specificity, dice_coef_necrotic,
dice_coef_edema ,dice_coef_enhancing])
```

```python
# MRI scan type must one of 1.flair; 2.t1; 3.t1ce(t1 contrast-enhaced); 4.t2; or the already segmented version
# Returns volume of specified study at `path`
def imageLoader(path):
    image = nib.load(path).get_fdata()
    X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channels))
    for j in range(VOLUME_SLICES):
        X[j +VOLUME_SLICES*c,:,:,0] = cv2.resize(image[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));
        X[j +VOLUME_SLICES*c,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));

        y[j +VOLUME_SLICES*c] = seg[:,:,j+VOLUME_START_AT];
    return np.array(image)


# Load nifti file at "path" and load each slice with mask from volume
# Choose the MRI type & resize to `IMG_SIZE` 128pixels by default
def loadDataFromDir(path, list_of_files, mriType, n_images):
    scans = []
    masks = []
    for i in list_of_files[:n_images]:
        fullPath = glob.glob( i + '/*'+ mriType +'*')[0]
        currentScanVolume = imageLoader(fullPath)
        currentMaskVolume = imageLoader( glob.glob( i + '/*seg*')[0] )
        # for each slice in 3D volume, find also it's mask
        for j in range(0, currentScanVolume.shape[2]):
            scan_img = cv2.resize(currentScanVolume[:,:,j], dsize=(IMG_SIZE,IMG_SIZE),
interpolation=cv2.INTER_AREA).astype('uint8')
            mask_img = cv2.resize(currentMaskVolume[:,:,j], dsize=(IMG_SIZE,IMG_SIZE),
interpolation=cv2.INTER_AREA).astype('uint8')
            scans.append(scan_img[..., np.newaxis])
            masks.append(mask_img[..., np.newaxis])
    return np.array(scans, dtype='float32'), np.array(masks, dtype='float32')
```

```python
# Train and save the model
train_unet =  model.fit(training_generator,
                        epochs=35,
                        steps_per_epoch=len(train_ids),
                        callbacks= callbacks,
                        validation_data = valid_generator
                        )
model.save("model_x1_1.h5")
```

```python
def predictByPath(case_path,case):
    files = next(os.walk(case_path))[2]
    X = np.empty((VOLUME_SLICES, IMG_SIZE, IMG_SIZE, 2))

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_flair.nii');
    flair=nib.load(vol_path).get_fdata()

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_t1ce.nii');
    ce=nib.load(vol_path).get_fdata()

    vol_path = os.path.join(case_path, f'BraTS20_Training_{case}_seg.nii');
    seg=nib.load(vol_path).get_fdata()


    for j in range(VOLUME_SLICES):
        X[j,:,:,0] = cv2.resize(flair[:,:,j+VOLUME_START_AT], (IMG_SIZE,IMG_SIZE))
        X[j,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE,IMG_SIZE))


    return model.predict(X/np.max(X), verbose=1)


def showPredictsById(case, start_slice = 60):
    path = f"../input/brats20-dataset-training-validation/BraTS2020_TrainingData/MICCAI_BraTS2020_TrainingData/BraTS20_Training_{case}"
    gt = nib.load(os.path.join(path, f'BraTS20_Training_{case}_seg.nii')).get_fdata()
    origImage = nib.load(os.path.join(path, f'BraTS20_Training_{case}_flair.nii')).get_fdata()
    p = predictByPath(path,case)

    core = p[:,:,:,1]
    edema= p[:,:,:,2]
    enhancing = p[:,:,:,3]

    plt.figure(figsize=(18, 50))
    f, axarr = plt.subplots(1,6, figsize = (18, 50))

    for i in range(6): # For each image, add brain background
        axarr[i].imshow(cv2.resize(origImage[:,:,start_slice+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE)), cmap="bone", interpolation='none')
```

```python
# Data Generator class
class DataGenerator(keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, list_IDs, dim=(IMG_SIZE,IMG_SIZE), batch_size = 1, n_channels = 2, shuffle=True):
        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.list_IDs = list_IDs
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        Batch_ids = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(Batch_ids)

        return X, y

    def on_epoch_end(self):
        'Update indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, Batch_ids):
        'Generates data containing batch size samples' # X : (n_samples, *dim, n_channels)
        # Init
        X = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, self.n_channels))
        y = np.zeros((self.batch_size*VOLUME_SLICES, 240, 240))
        Y = np.zeros((self.batch_size*VOLUME_SLICES, *self.dim, 4))


        # Generate data
        for c, i in enumerate(Batch_ids):
            case_path = os.path.join(TRAIN_DATASET_PATH, i)

            data_path = os.path.join(case_path, f'{i}_flair.nii');
            flair = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_t1ce.nii');
            ce = nib.load(data_path).get_fdata()

            data_path = os.path.join(case_path, f'{i}_seg.nii');
            seg = nib.load(data_path).get_fdata()

            for j in range(VOLUME_SLICES):
                X[j +VOLUME_SLICES*c,:,:,0] = cv2.resize(flair[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));
                X[j +VOLUME_SLICES*c,:,:,1] = cv2.resize(ce[:,:,j+VOLUME_START_AT], (IMG_SIZE, IMG_SIZE));

                y[j +VOLUME_SLICES*c] = seg[:,:,j+VOLUME_START_AT];

        # Generate masks
        y[y==4] = 3;
        mask = tf.one_hot(y, 4);
        Y = tf.image.resize(mask, (IMG_SIZE, IMG_SIZE));
        return X/np.max(X), Y

training_generator = DataGenerator(train_ids)
valid_generator = DataGenerator(val_ids)
test_generator = DataGenerator(test_ids)

```

```python
# Count number of pixels for each segment for each slice in volume

def maskSizeForSlice(path,i_slice):
    totals = dict([(1, 0), (2, 0), (3, 0)])
    image_volume=nib.load(path).get_fdata()
    # Flatten 3D image into 1D array and convert mask 4 to 2
    arr=image_volume[:,:,i_slice].flatten()
    arr[arr == 4] = 3

    unique, counts = np.unique(arr, return_counts=True)
    unique = unique.astype(int)
    values_dict=dict(zip(unique, counts))
    for k in range(1,4):
        totals[k] += values_dict.get(k,0)
    return totals
```

```python
# ID: Age, Categories
def getListAgeDays(id_list):
    x_val = []
    y_val = []
    for i in id_list:
        if (i not in age_dict):
            continue
        masks = getMaskSizesForVolume(nib.load(TRAIN_DATASET_PATH +
f'BraTS20_Training_{i[-3:]}/BraTS20_Training_{i[-3:]}_seg.nii').get_fdata())
        brain_vol = getBrainSizeForVolume(nib.load(TRAIN_DATASET_PATH +
f'BraTS20_Training_{i[-3:]}/BraTS20_Training_{i[-3:]}_t1.nii').get_fdata())
        masks[1] = masks[1]/brain_vol
        masks[2] = masks[2]/brain_vol
        masks[3] = masks[3]/brain_vol
        merged=[age_dict[i],masks[1],masks[2],masks[3]] # Add Segments
        x_val.append(merged)
        if (days_dict[i] < 365):
            y_val.append([1,0,0])
        elif (days_dict[i] >= 365 and days_dict[i] < 730):
            y_val.append([0,1,0])
        else:
            y_val.append([0,0,1])

    return np.array(x_val), np.array(y_val)

X_all, y_all = getListAgeDays(train_and_test_ids)

print(f'X_test: {X_all.shape}')
df = pd.DataFrame(np.concatenate((X_all, y_all), axis=1) , columns = ["age",f"{SEGMENT_CLASSES[1]}",f"
{SEGMENT_CLASSES[2]}",f"{SEGMENT_CLASSES[3]}","short","medium","long"])
df.head()
```

```python
# Plot the most important features
df = pd.DataFrame(X_train, columns = ["age",f"{SEGMENT_CLASSES[1]}",f"{SEGMENT_CLASSES[2]}",f"
{SEGMENT_CLASSES[3]}"])
feature_scores = pd.Series(rfc.feature_importances_, index=df.columns).sort_values(ascending=False)

sns.barplot(x=feature_scores, y=feature_scores.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features',fontweight='bold')
plt.title("Most Important Features", fontweight='bold')
plt.show()
```

```python
# KNN
from sklearn.neighbors import KNeighborsClassifier

knn  = KNeighborsClassifier(n_neighbors=38, p=2, weights='distance')
knn.fit(X_train,y_train_multi)
accuracies = cross_val_score(knn, X_train, y_train_multi)
y_pred = knn.predict(X_test)

# KNN confusion matrix
y_pred=y_pred.astype(int)
n_values = np.max(y_pred) + 1
y_pred_hot=np.eye(n_values)[y_pred]
cm_knn = confusion_matrix(y_test.argmax(axis=1), y_pred_hot.argmax(axis=1))
sns.set(font_scale=1.1)
sns.heatmap(cm_knn, annot=True, annot_kws={"size": 10})
plt.show()

# KNN with GridSearchCV
grid = {
    'n_neighbors':np.arange(1,75),
    'p':np.arange(1,5),
    'weights':['uniform','distance']
    }

knn = KNeighborsClassifier()
knn_grid = GridSearchCV(knn,grid,cv=5)
knn_grid.fit(X_train,y_train_multi)

```