ABSTRACT

Magnetic Resonance Imaging (MRI) is a critical tool in the diagnosis and management of brain tumors, including pituitary adenomas, malignant melanomas, and gliomas. These tumor's present unique challenges due to their complex nature and the critical functions of the surrounding brain tissue. Early and accurate identification is paramount for effective treatment planning and improved patient outcomes. In this context, Artificial Intelligence (AI) models have emerged as transformative tools in neuro-oncology (1). Recent advancements in AI, particularly in machine learning and deep learning, have led to the development of models that can rapidly classify brain tumor's with remarkable accuracy. For instance, a deep learning model using ResNet50 combined with Gradient-weighted Class Activation Mapping (Grad-CAM) has demonstrated a testing accuracy of 98.52%, significantly enhancing tumor detection (2).

Moreover, AI models are not only improving the speed of tumor identification but also aiding in the decision-making process by providing recommendations for treatment options. The integration of AI in medical imaging can streamline workflows, reduce the time between diagnosis and treatment, and potentially lower the cost of care (3). Importantly, these models offer a non-invasive alternative to traditional biopsy methods, which can be risky and uncomfortable for patients. The ability of AI to analyze vast amounts of imaging data quickly surpasses the capabilities of human interpretation, allowing for the detection of subtle changes that might be overlooked otherwise.

The application of AI in brain tumor classification also addresses the need for precision medicine. By tailoring treatment plans to the individual characteristics of each tumor, healthcare providers can offer more personalized and effective care. AI models can identify patterns and correlations in imaging data that are indicative of specific tumor types, grades, and even genetic markers, which are crucial for targeted therapies.

However, the implementation of AI in clinical settings must be approached with caution. The accuracy of AI models depends on the quality and diversity of the data they are trained on. It is essential to have a large and varied dataset that includes different tumor types, stages, and patient demographics to ensure the model's reliability and reduce the risk of bias. Additionally, the interpretability of AI decisions is a critical factor for trust and adoption by medical professionals. Explainable AI, which provides insights into the model's decision-making process, can bridge the gap between AI recommendations and clinical judgment.

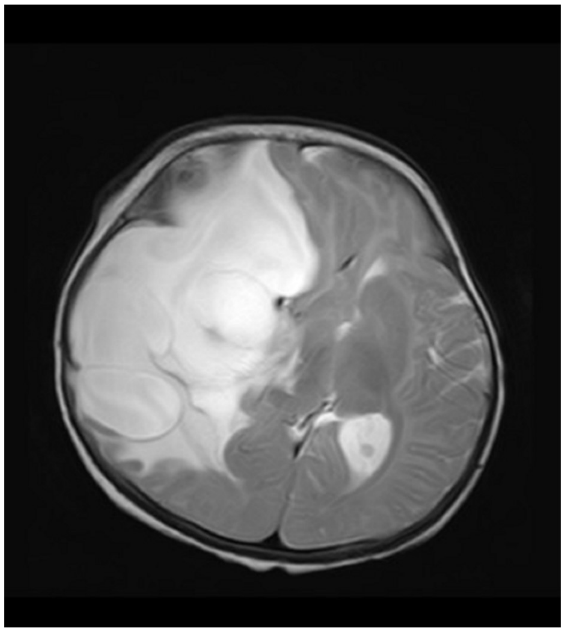
In conclusion, AI models hold great promise for revolutionizing the identification and classification of brain tumors through MRI. They offer a fast, accurate, and non-invasive means of diagnosis, which can be integral in the planning of personalized treatment strategies. As research progresses, it is anticipated that AI will become an indispensable tool in neuro-oncology, complementing the expertise of medical professionals and enhancing patient care. Continuous collaboration between AI developers, radiologists, and oncologists is crucial to refine these models and fully realize their potential in clinical practice.

TABLE OF CONTENT

1. INTRODUCTION

Brain tumors represent a diverse and complex group of neoplasms, posing significant challenges in the medical field. They are characterized by their aggressive nature and resistance to conventional therapies, leading to poor prognosis for patients. This thesis aims to delve into the intricate world of brain tumors, exploring their pathogenesis, classification, and the current state of treatment options.
Primary brain tumors, which originate in the brain, account for approximately 70% of all brain cancers. The remaining 30% are secondary tumors, which are metastatic cancers that have spread to the brain from other parts of the body. Among these, glioblastomas (GBM) are the most common and deadly, with a 5-year survival rate of less than 5%.
Despite the advancements in bioengineering, biochemistry, and bioinformatics over the past two decades, our understanding of brain tumors remains limited. Various environmental factors such as exposure to low-frequency electromagnetic fields, alcohol, tobacco use, head trauma, chemical agents, and infections are considered as risk factors for brain tumors. However, the exact mechanisms of how these factors contribute to the development of brain tumors are still not fully understood.
This thesis will present a comprehensive review of the current knowledge on brain tumors, focusing on their molecular and cellular characteristics, genetic and environmental risk factors, diagnostic methods, and treatment strategies. It will also highlight the gaps in our understanding and propose potential avenues for future research.
The goal of this thesis is to contribute to the ongoing efforts in the scientific community to unravel the mysteries of brain tumors, with the hope of improving diagnosis, treatment, and ultimately, patient outcomes.
A brain tumor refers to an abnormal growth of cells within or near the brain. These tumors can originate in the brain tissue itself (known as primary brain tumors) or spread to the brain from other parts of the body (referred to as second ary or metastatic brain tumors). Figure 1 shows the MRI scan of a brain tumor.

Fig (1) MRI scan of the frontal view of the brain with tumor.



Let's delve deeper into the intricacies of brain tumors:

1.1 Primary Brain Tumors:

Benign Brain Tumors: These noncancerous growths may arise from brain tissue. Although they lack the aggressive nature of malignant tumors, benign brain tumors can still cause significant symptoms due to their location. Over time, they may gradually increase in size, exerting pressure on surrounding brain tissue.

Malignant Brain Tumors (Brain Cancers): These tumors are more aggressive and can infiltrate healthy brain cells. They grow rapidly and have the potential to invade nearby structures. Malignant brain tumors pose a serious threat to overall health and require prompt intervention. Fig-1

1.2 Secondary Brain Tumors (Metastatic Tumors):

These tumors result from cancer cells that have spread (metastasized) from other parts of the body (such as the lungs, breast, or skin) to the brain. Metastatic brain tumors are relatively common and often occur in individuals with a history of cancer. The brain provides an ideal environment for metastasis due to its rich blood supply and protective blood-brain barrier. Cancer cells can infiltrate the brain and form secondary tumors.

1.3 Symptoms and Detection:

Brain tumors vary significantly in size. Some are discovered early because they cause noticeable symptoms, such as headaches, seizures, changes in vision, or cognitive deficits. However, tumors in less active brain regions may remain asymptomatic until they reach a substantial size. As a result, late-stage detection is possible. Imaging techniques (such as MRI or CT scans) play a crucial role in diagnosing brain tumors. Biopsy or surgical removal confirms the type and nature of the tumor.

1.4 Treatment Options:

1.4.1 Surgery: Surgical removal aims to exercise as much of the tumor as possible while preserving healthy brain tissue. The feasibility of surgery depends on the tumor's location and accessibility.

1.4.2 Radiation Therapy: High-energy beams target the tumor, damaging its DNA and inhibiting growth. Radiation therapy is often used after surgery or as the primary treatment for inoperable tumors.

1.4.3 Chemotherapy: Some brain tumors respond to chemotherapy drugs. However, the blood-brain barrier limits drug delivery to the brain.

1.4.4 Targeted Therapies: Emerging treatments specifically target tumor-related molecules or pathways.

1.4.5 Supportive Care: Managing symptoms, improving quality of life, and addressing side effects are essential components of brain tumor care.

In summary, brain tumor treatment decisions depend on factors such as tumor type, size, location, and the patient's overall health. A multidisciplinary approach involving neurosurgeons, oncologists, and other specialists ensures the best possible outcome.

1.5 World Health Organization (WHO) report on brain tumor:

Brain tumors are a significant global health concern, contributing to the burden of neurological disorders. They are characterized by their aggressive nature, resistance to conventional therapies, and poor prognoses for patients1. The fifth edition of the WHO Classification of Tumors of the Central Nervous System (CNS), published in 2021(4), is the sixth version of the international standard for the classification of brain and spinal cord tumors. Building on the 2016 updated fourth edition and the work of the Consortium to Inform Molecular and Practical Approaches to CNS Tumor Taxonomy, the 2021 fifth edition (5) introduces major changes that advance the role of molecular diagnostics in CNS tumor classification. At the same time, it remains wedded to other established approaches to tumor diagnosis such as histology and immunohistochemistry. In doing so, the fifth edition establishes some different approaches to both CNS tumor nomenclature and grading and it emphasizes the importance of integrated diagnoses and layered reports. New tumor types and subtypes are introduced, some based on novel diagnostic technologies such as DNA methylome profiling. The present review summarizes the major general changes in the 2021 fifth edition classification and the specific changes in each taxonomic category. It is hoped that this summary provides an overview to facilitate more in-depth exploration of the entire fifth edition of the WHO Classification of Tumors of the Central Nervous System.

Molecular Characteristics

Brain tumors exhibit a variety of molecular characteristics that contribute to their development and progression. These include aberrant signaling pathways, genetic mutations and polymorphisms, epigenetic alterations, and the involvement of tumor suppressor genes and oncogenes. Brain tumors can be primary (originating in the brain) or secondary (metastatic tumors that spread to the brain from other parts of the body).

About 70% of brain cancers are classified as primary tumors, while the remaining 30% are secondary tumors.

Glioblastoma

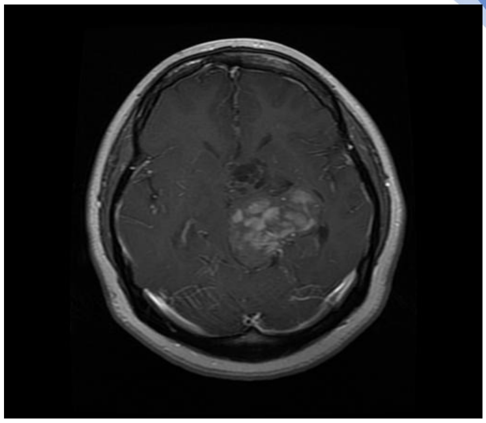
(GBM) is the most prevalent primary brain tumor and ranks among the most lethal of human cancers with conventional therapy offering only palliation. Great strides have been made in understanding brain cancer genetics and modeling these tumors with new targeted therapies being tested but these advances have not translated into substantially improved patient outcomes. Multiple chemotherapeutic agents, including temozolomide, the first-line treatment for glioblastoma, have been developed to kill cancer cells. However, the response to temozolomide in GBM is modest. Radiation is also moderately effective, but this approach is plagued by limitations due to collateral radiation damage to healthy brain tissue and development of radio resistance. Therapeutic resistance is attributed at least in part to a cell population within the tumor that possesses stem-like characteristics and tumor propagating capabilities, referred to as cancer stem cells. Within GBM, the intertumoral heterogeneity is derived from a combination of regional genetic variance and a cellular hierarchy often regulated by distinct cancer stem cell niches, most notably perivascular and hypoxic regions. With the recent emergence as a key player in tumor biology, cancer stem cells have symbiotic relationships with the tumor microenvironment, oncogenic signaling pathways, and epigenetic modifications. The origins of cancer stem cells and their contributions to brain tumor growth and therapeutic resistance are under active investigation with novel anti-cancer stem cell therapies offering potential new hope for this lethal disease. Now there are levels to the glioblastoma (GBM) which includes.

1.5.1 Glioma:

Gliomas are primary central nervous system tumors that arise from glial progenitor cells. Gliomas have been classically classified morphologically based on their histopathological characteristics. Figure 2 shows the MRI scanned image of the glioma tumor inside the brain. However, with recent advances in cancer genomics, molecular profiles have now been integrated into the classification and diagnosis of gliomas. In this review article, we discuss the clinical features, imaging findings, and molecular profiles of adult-type diffuse gliomas based on the new 2021 World Health Organization Classifications of Tumors of the central nervous system. (fig-2). As a group, gliomas are one of the most common types of brain tumors. While the exact origin of gliomas is still unknown, they are thought to grow from glial cells or glial precursor cells. A glial cell is a type of supportive cell in the brain. The main types of supportive cells in the brain include astrocytes, oligodendrocytes, and ependymal cells. Gliomas may be considered astrocytoma, oligodendroglioma, or ependymoma. Gliomas are assigned a grade, which is an indication of how aggressive a tumor is likely to be. A higher grade is usually more aggressive and more likely to grow quickly. However, current research is helping doctors move toward using tumor genetics to better classify gliomas. This is discussed elsewhere in this guide. Unlike most tumors that start outside of the brain and CNS, most primary brain tumors like glioma are not assigned a "stage." For tumors that do not begin in the brain, a higher cancer "stage" number usually describes whether the primary tumor has spread to other parts of the body, and this information influences which treatments are selected. Primary brain tumors, like gliomas, only rarely spread outside of the brain. Thus, they do not need to be staged to help the clinical team decide on the appropriate treatments.

Currently, the types of gliomas include:

Figure 2 MRI scanned image of the brain with glioma tumor present.



Astrocytoma.

Astrocytoma is the most common type of glioma. Astrocytoma cells look like glial cells called astrocytes that are found in the cerebrum or cerebellum. Historically, there have been 4 grades of astrocytoma, which are described below:

Grade 1 or pilocytic astrocytoma is a slow-growing tumor that is most often benign and rarely spreads into nearby tissue. Benign means the tumor can grow but does not spread to other parts of the body.

Grade 2 astrocytoma is a slow-growing malignant tumor that can often spread into nearby tissue and can become a higher grade. Malignant means it is cancerous and can spread to other parts of the body.

Grade 3 or anaplastic astrocytoma is a malignant tumor that can quickly grow and spread to nearby tissues.

Grade 4 or glioblastoma is a very aggressive form of astrocytoma.

A new international classification system for primary brain tumors was unveiled by the World Health Organization (WHO) in 2021. This system divides astrocytoma's and other types of brain tumors into many subgroups depending largely on their genetic makeup and the presence or absence of certain important changes in the tumor's specific genes. The treatment team will use information from an analysis of each tumor sample to precisely classify each tumor using the new guidelines. (Learn more about biomarker testing of the tumor in the diagnosis section.) Key changes for the most common types of astrocytoma's include:
Adult diffuse astrocytoma's have now been based on whether there is a mutation in the isocitrate dehydrogenase (IDH) gene. There are 2 groups: 1) astrocytoma, IDH mutant, and 2) glioblastoma, IDH wild-type. "Wild-type" means that the gene is found in its natural, unmutated form.

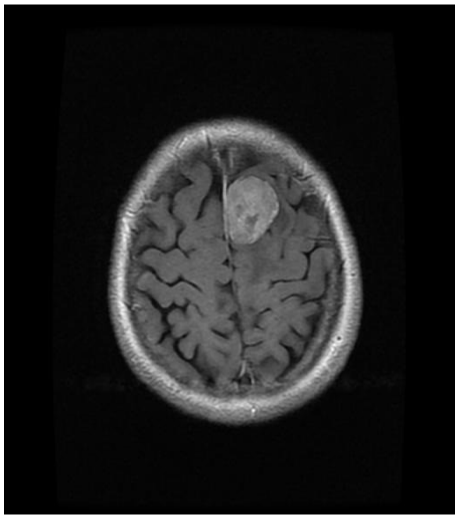
Astrocytoma.

Astrocytoma, IDH mutant, can be a grade 2, grade 3, or grade 4 tumor, based on whether there are other genetic and tumor features, including a high rate of cell division (called the mitotic index) and alterations in the CDKN2A/B genes. These were previously called IDH-mutant or secondary glioblastomas.

Glioblastoma is now only used to describe IDH wild-type tumors that also have 1 or more of the following features: loss of chromosome 10, gain of chromosome 7, TERT promoter mutation, and increased number of copies of the EGFR gene. Unlike the previous classification system, glioblastomas are do not also have to show signs of cell death and excessive growth of blood vessels.
Oligodendroglioma. Oligodendroglioma is a tumor whose cells look like glial cells called oligodendrocytes. These cells are responsible for making myelin. Myelin surrounds the nerves and is rich in protein and fatty substances called lipids. Under the 2021 WHO guidelines, these tumors must have an IDH mutation and contain chromosome 1p and 19q codeletion. They are categorized as either grade 2 oligodendroglioma, which is considered low grade, or grade 3 oligodendroglioma, which is considered a high-grade tumor with anaplastic features.

1.5.2 Non-glioma:

Figure 3 MRI scanned image of the non-glioma tumor inside the brain.



Non-glioma tumors are tumors that arise from cells in the brain that are not glial cells (fig-3 shows the MRI scanned image of the brain with non-glioma image. Types of non-glioma tumors include Meningioma is the most common primary brain tumor. It begins in the meninges and is most often noncancerous. Meningioma can cause serious symptoms if it grows and presses on the brain or spinal cord or grows into the brain tissue. Pineal gland and pituitary gland tumors. These are tumors that start in the pineal gland and pituitary gland. Primary CNS lymphoma. This is a form of lymphoma. Lymphoma is a cancer that begins in the lymphatic system. Primary CNS lymphoma starts in the brain and can spread to the spinal fluid and eyes. Medulloblastoma. Medulloblastoma is thought to start from a specific type of cell in the cerebellum. These cells are called cerebellar granule progenitor cells. It is most common in children and is usually cancerous, often spreading throughout the CNS. Learn about medulloblastoma in children. Craniopharyngioma. Craniopharyngioma is a benign tumor that begins near the pituitary gland located near the base of the brain. These tumors are uncommon. Learn about craniopharyngioma in children. Schwannoma. Schwannoma is a rare tumor that begins in the nerve sheath, or the lining of the nerves. It may often occur in the vestibular nerve, which is a nerve in the inner ear that helps control balance it is typically noncancerous

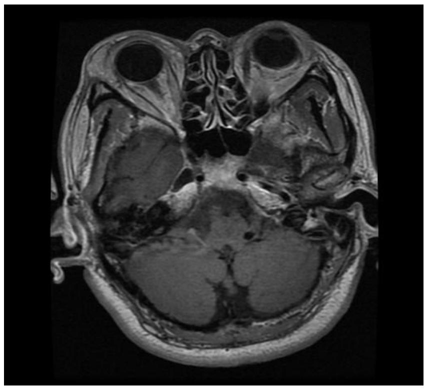
1.5.3 Pituitary Tumor:

Pituitary tumors are unusual growths that develop in the pituitary gland. This gland is an organ about the size of a pea. It's located behind the nose at the base of the brain. (fig-4 shows the MRI scanned image of the brain with a pituitary gland with a tumor) Some of these tumors cause the pituitary gland to make too much of certain hormones that control important body functions. Others can cause the pituitary gland to make too little of those hormones.

Most pituitary tumors are benign. That means they are not cancer. Another name for these noncancerous tumors is pituitary adenomas. Most adenomas stay in the pituitary gland or in the tissue around it, and they grow slowly. They typically don't spread to other parts of the body.

Pituitary tumors can be treated in several ways. The tumor may be removed with surgery. Or its growth may be controlled with medications or radiation therapy. Sometimes, hormone levels are managed with medicine. Your health care provider may suggest a combination of these treatments. In some cases, observation —

Figure-4 shows the MRI scanned image of the pituitary tumor inside the brain



Types of pituitary adenomas include:

Functioning. These adenomas make hormones. They cause different symptoms depending on the kind of hormones they make. Functioning pituitary adenomas fall into several categories, including those that make:

Adrenocorticotropic hormone. This hormone also is known as ACTH. These tumors are sometimes called corticotrope adenomas.

Growth hormone. These tumors are called somatotroph adenomas.

Luteinizing hormone and follicle-stimulating hormone. These hormones also are known as gonadotropins. Pituitary tumors that make these hormones are called gonadotroph adenomas.

Prolactin. These tumors are called prolactinomas or lactotroph adenomas.
Thyroid-stimulating hormone. These tumors are called thyrotrope adenomas.
Nonfunctioning. These adenomas don't make hormones. The symptoms they cause are related to the pressure their growth puts on the pituitary gland, nearby nerves, and the brain.
Macroadenomas. These are larger adenomas. They measure about 1 centimeter or more. That's slightly less than a half-inch. They can be functioning or nonfunctioning.
Microadenomas. These adenomas are smaller. They measure less than 1 centimeter. That's slightly less than a half-inch. They can be functioning or nonfunctioning.

1.6 Brain Tumor Imaging Modalities

Imaging plays a crucial role in the evaluation of patients with brain tumors. The two most important and commonly used imaging modalities are Computed Tomography (CT) and Magnetic Resonance Imaging (MRI)123. CT was introduced in clinical practice in 1972 and rapidly became a very important factor in radiological diagnosis1. It provides detailed images of the brain and has replaced invasive procedures such as pneumoencephalography or cerebral angiography1.MRI provides detailed morphologic information and can supply some additional insights into metabolism (MR spectroscopy) and perfusion (perfusion-weighted imaging)4. It is also capable of giving functional information on microstructural changes of tumor tissues4.In addition to CT and MRI, other imaging techniques like Positron Emission Tomography (PET) scan of the brain5 and new imaging techniques that evaluate tissue blood flow (perfusion imaging), water motion (diffusion imaging), brain metabolites (Proton magnetic resonance spectroscopy) and blood oxygen level dependent (BOLD) imaging have also been included. These imaging modalities are essential for delineating the normal and pathologic anatomy, assessing vascular supply and impairment of the blood–brain barrier, and also for assessing the location, extent of the tumor, and its relationship to the surrounding structures1. This information is very important and critical in deciding between the different forms of therapy such as surgery, radiation, and chemotherapy.

1.6.1 Magnetic Resonance Imaging (MRI).

Magnetic resonance imaging (MRI) of a brain generates several 3-dimensional image data that comprise the three anatomical views of a brain (axial, sagittal, and coronal) at different depths of a brain. Depending on the strength of the magnetic field and the sampling protocols, the image quality, slice thickness, and inter-slice gap vary. During MR imaging, a patient lay in a strong magnetic field, almost 10,000 times stronger than the earth's magnetic field, that forces the protons in the water molecule of the body to align in either a parallel (low energy) or anti-parallel (high energy) orientation with the magnetic field. Then, a radiofrequency pulse is introduced that forces the spinning protons to move out of the equilibrium state. When a radiofrequency pulse pauses, the protons return to an equilibrium state and produce a sinusoidal signal at a frequency dependent on the local magnetic field. Finally, a radio antenna within the scanner detects the sinusoidal signal and creates the image. The amount of signal produced by specific tissue types is determined by their number of mobile hydrogen protons, the speed at which they are moving, the time needed for the protons within the tissue to return to their original state of magnetization (T1), and the time required for the protons perturbed into coherent oscillation by the radiofrequency pulse to lose their coherence (T2) relaxation times. As T1 (spin-lattice, also known as longitudinal relaxation) and T2 (spin-spin, also known as traversal relaxation) times are time-dependent, the timing of the radio frequency pulse and the reading of the radiated RF energy change the appearance of the image. In addition, the repetition time (TR) describes

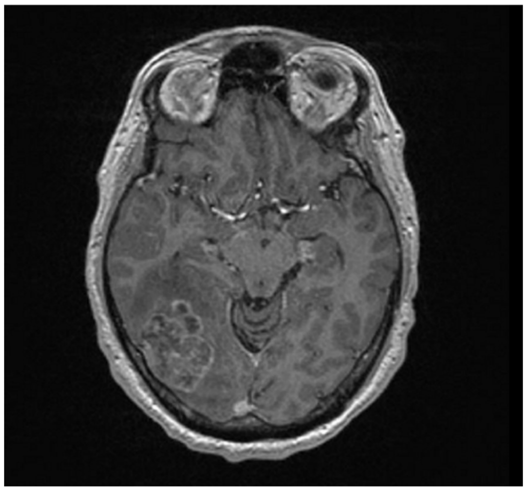


fig-5 MRI scan image of the brain

the time between successive applications of RF pulse sequences, and the echo time (TE) tells the delay before the RF energy radiated by the tissue in question is measured. The variation of T1 and T2 relaxation times between tissues gives image contrast on T1- and T2-weighted (T1-w and T2-w) images. The T1-w sequence is characterized by short TR and short TE while the T2-w sequence is characterized by long TR and short TE. Tissues with shorter T1 (for example, white matter) appear brighter when compared to tissues with a longer T1 (for example, gray matter) in magnetic resonance images. The other intermediate sequence that adopts long TR from T2-w and short TE from T1-w is a proton density-weighted (PD-w). In PD-w, the number of protons per unit volume in tissues is the main factor in determining the formation of image.

In the current neuroimaging techniques different MRI brain scan procedures can be performed, these include, the conventional structural MRI, functional MRI, diffusion-weighted imaging (DWI), and diffusion tensor imaging (DTI) [10]. In structural MRI procedure which mainly differentiates healthy and abnormal brain tissues based on their water molecule content is the most employed standard imaging technique. This procedure helps to visualize healthy brain tissues and to map gross brain anatomy, tumoral vascularity, calcification, and radiation-induced micro haemorrhage. The structural sequences include T1-w, T2-w, FLAIR, and contrast-enhanced T1-w [10]. The functional MRI (fMRI) on the other hand is used to capture the neural activity inside a brain through the ratio of oxygenated to the deoxygenated level of blood in the neighbouring vasculature while performing a cognitive or motor task. The fMRI is used to localize the eloquent cortex and differentiate between tumor grades. The DWI captures the random motion of water molecules in a brain and it is used to characterize a tumor through identification of its cellularity and hypoxia, peritumoral edema, the integrity of WM tracts, and to differentiate between posterior fossa tumors. Whereas, diffusion tensor imaging (DTI) is used to analyse the 3D diffusion direction, also known as diffusion tensor, of the water molecule. The DTI helps to determine local effects of the tumor on white matter tract integrity including tract displacement, the existence of vasogenic edema, tumor infiltration and tract destruction .(fig-5 shows the image of a MRI scan image of the brain).

1.6.2 Computerized tomography (CT)

A computerized tomography (CT) scan of the head is an imaging test that is sometimes used to confirm a brain tumor diagnosis. This non-invasive procedure involves taking a series of X-rays from many different angles. During a CT scan, an X-ray beam moves in circles around the body, capturing many different views of the brain. As such, a CT scan can provide more detailed information about brain tissues and structures than a standard X-ray.

Sometimes, a contrast dye is given to the patient intravenously during a CT scan of the head. The dye will collect around any cancerous cells to provide heightened clarity in the resulting images. A computer then combines the X-rays into a detailed, three-dimensional image that may reveal a tumor or another abnormality , such as bleeding or swelling in the brain.

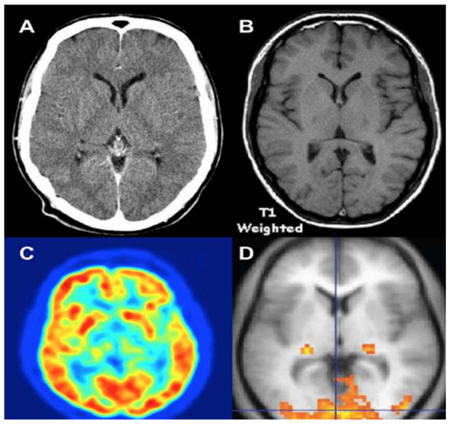


Fig-6 show the CT scan image of the brain.

2. APPROACH USED

In this project we have used CNN for our classification as Convolutional Neural Networks (CNNs) are a specialized kind of neural network used for image classification because they are particularly good at picking up on patterns in visual data. The architecture of a CNN is designed to mimic the way a human eye works, making them efficient for image recognition tasks. They use a mathematical operation called convolution, which involves a filter or kernel that passes over the image to detect features such as edges, textures, and shapes. This process creates a feature map that highlights important details and reduces the dimensionality of the image data, making it more manageable for the network to process. CNNs are composed of multiple layers that each perform different tasks. The first layer is the convolutional layer, which applies various filters to the input image to create feature maps. These feature maps then pass through a pooling layer, which reduces their size and complexity, allowing the network to focus on the most important features. After several convolutional and pooling layers, the network uses fully connected layers to classify the images based on the features detected. One of the main advantages of CNNs is their ability to learn feature hierarchies. Lower layers might learn to recognize simple patterns, like lines and edges, while deeper layers can recognize more complex features, like shapes or specific objects. This hierarchical feature learning is powerful because it allows the network to build up an understanding of images in a way that's similar to how humans perceive visual information. Another benefit of CNNs is parameter sharing, which means that the same filter is used across the entire image, significantly reducing the number of parameters in the network. This makes CNNs less prone to overfitting and reduces the computational cost compared to fully connected networks, where each input pixel is connected to a neuron. Furthermore, CNNs are translation invariant, meaning they can recognize an object no matter where it appears in the image. This is because the feature maps generated by the convolutional layers retain the spatial hierarchy of the features, allowing the network to maintain spatial awareness. In summary, CNNs are used for image classification because they are efficient at processing visual data, can learn complex feature hierarchies, have reduced computational costs due to parameter sharing, and are capable of recognizing objects regardless of their position in the image. These characteristics make CNNs a powerful tool for tasks that involve image recognition, such as facial recognition, medical image analysis, and autonomous vehicle navigation. By leveraging CNNs, we can build systems that understand and interpret visual data with a high degree of accuracy and reliability.

2.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks (6). It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The architecture of CNNs is inspired by the visual processing in the human brain, and they are well-suited for capturing hierarchical patterns and spatial dependencies within images.

2.1.1 Key components of a Convolutional Neural Network include:

Convolutional Layers: These layers apply convolutional operations to input images, using filters (also known as kernels) to detect features such as edges, textures, and more complex patterns. Convolutional operations help preserve the spatial relationships between pixels.

Pooling Layers: Pooling layers down sample the spatial dimensions of the input, reducing the computational complexity and the number of parameters in the network. Max pooling is a common pooling operation, selecting the maximum value from a group of neighbouring pixels.

Activation Functions: Non-linear activation functions, such as Rectified Linear Unit (ReLU), introduce non-linearity to the model, allowing it to learn more complex relationships in the data.

Fully Connected Layers: These layers are responsible for making predictions based on the high-level features learned by the previous layers. They connect every neuron in one layer to every neuron in the next layer.

CNNs are trained using a large dataset of labelled images, where the network learns to recognize patterns and features that are associated with specific objects or classes. Proven to be highly effective in image-related tasks, achieving state-of-the-art performance in various computer vision applications. Their ability to automatically learn hierarchical representations of features makes them well-suited for tasks where the spatial relationships and patterns in the data are crucial for accurate predictions. CNNs are widely used in areas such as image classification, object detection, facial recognition, and medical image analysis.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes.

The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

2.1.2 Convolutional Neural Network Design

The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order.

It is the sequential design that give permission to CNN to learn hierarchical attributes.

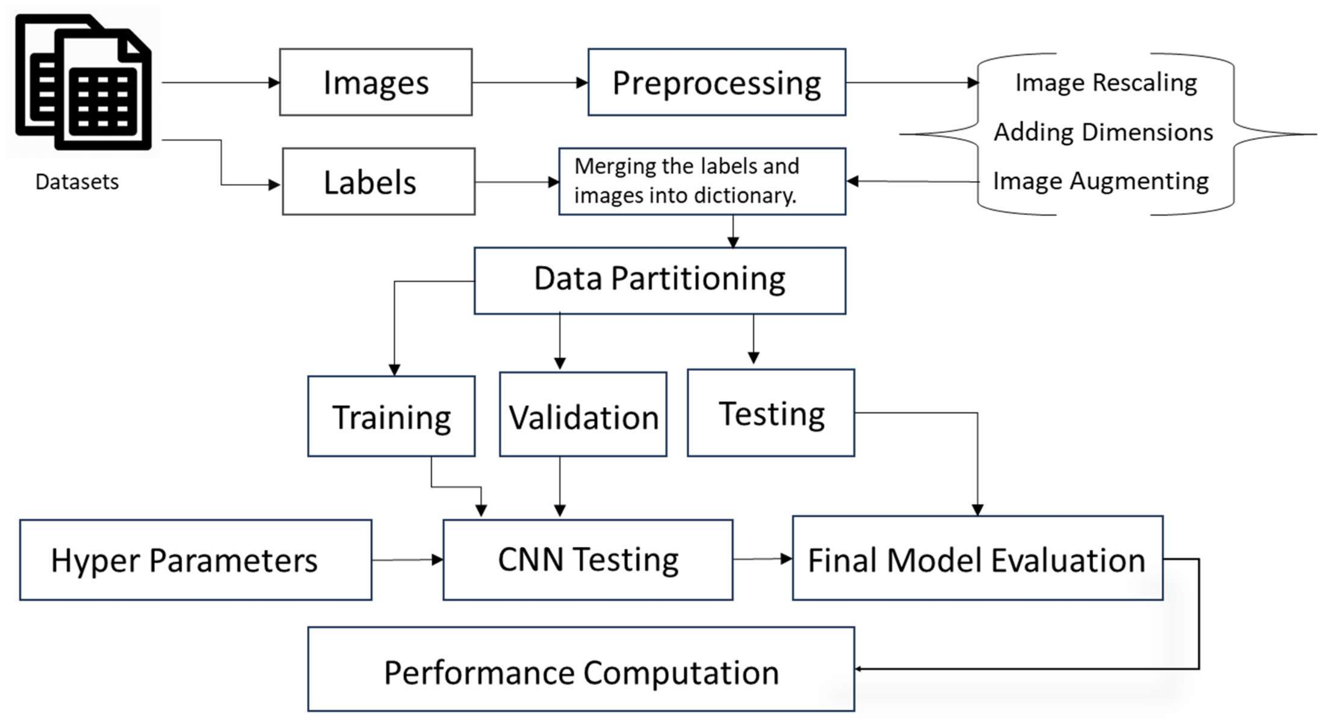


Fig (7) the working flow chart of CNN

In CNN, some of them followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers. The pre-processing needed in a ConvNet is kindred to that of the related pattern of neurons in the human brain and was motivated by the organization of the Visual Cortex.

2.1.3 Convolutional Neural Network Training

CNNs are trained using a supervised learning approach. This means that the CNN is given a set of labelled training images. The CNN then learns to map the input images to their correct labels.

The training process for a CNN involves the following steps:

Data Preparation: The training images are pre-processed to ensure that they are all in the same format and size. Also, the resolution of the image is kept higher for the model to detect better.

Loss Function: A loss function is used to measure how well the CNN is performing on the training data. The loss function is typically calculated by taking the difference between the predicted labels and the actual labels of the training images.

Loss Function Equation: This loss function is calibrated by Kullback-Leibler Divergence which then, measures difference between 2 distributions based on mutual information. When x and y independent they converge, as x and y are dependent, they diverge.

$$\mathrm{D}(\mathrm{p}||\mathrm{q}) = \mathrm{Integral}(\mathrm{p}(\mathrm{x})\log \mathrm{p}(\mathrm{x})/\mathrm{q}(\mathrm{x}))$$

Optimizer: An optimizer is used to update the weights of the CNN in order to minimize the loss function.

$$\mathrm{\theta}=\mathrm{\theta}-\mathrm{\alpha}\cdot\nabla \mathrm{J}(\mathrm{\theta})\theta=\theta-\alpha\cdot\nabla \mathrm{J}(\theta)$$

Backpropagation: Backpropagation is a technique used to calculate the gradients of the loss function with respect to the weights of the CNN. The gradients are then used to update the weights of the CNN using the optimizer.

2.1.4 CNN Evaluation

After training, CNN can be evaluated on a held-out test set. A collection of pictures that the CNN has not seen during training makes up the test set. How well the CNN performs on the test set is a good predictor of how well it will function on actual data.

The efficiency of a CNN on picture categorization tasks can be evaluated using a variety of criteria. Among the most popular metrics are:

Accuracy: Accuracy is the percentage of test images that the CNN correctly classifies.

$$\mathrm{Accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Precision: Precision is the percentage of test images that the CNN predicts as a particular class and that are actually of that class.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

Recall: Recall is the percentage of test images that are of a particular class and that the CNN predicts as that class.

The following image gives a top-level view of what CNN looks like:

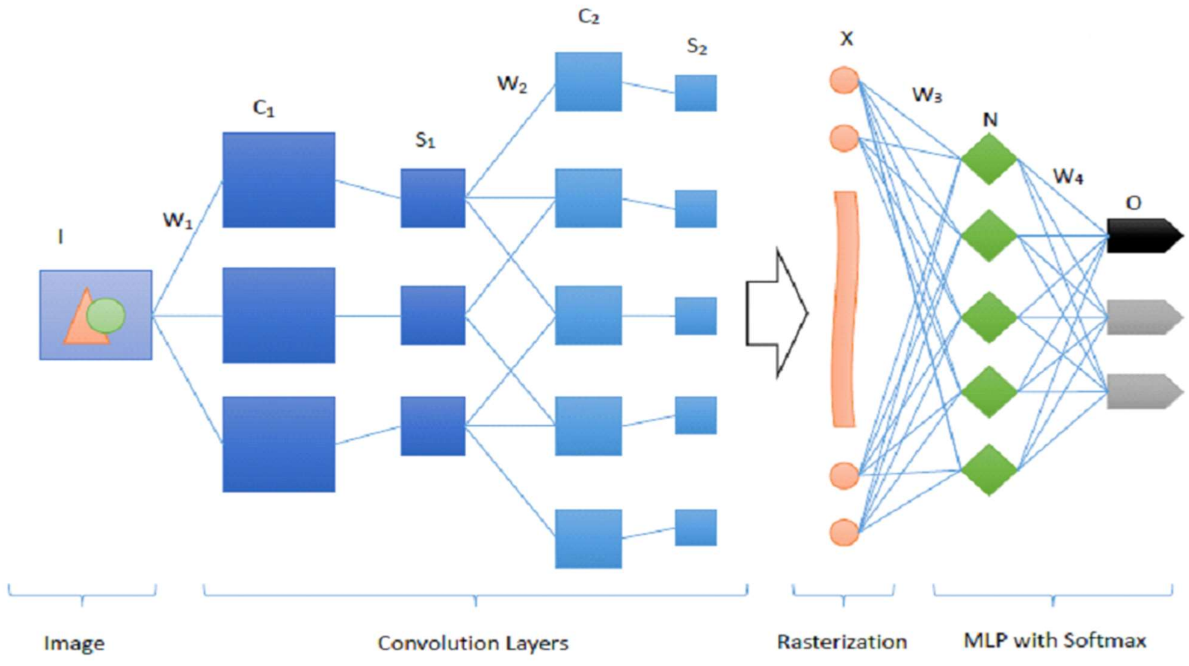


Fig (8) Image of the CNN architecture

2.2 Conventional Machine Learning Based Approach

Machine learning is a paradigm where a machine is given a task where its performance improves with experience. Machine learning techniques are commonly grouped into three major types: supervised, unsupervised, and reinforcement learning. Supervised learning is based on training a data sample from the data source with correct classification already assigned by domain experts, whereas, in unsupervised learning, the algorithm finds hidden patterns from the unlabelled data. On the other hand, reinforcement learning is carried out by making a sequence of decisions using reward signals. Therefore, the algorithm learns through receiving either rewards or penalties for the actions it performs. Machine learning has been used in the classification of brain tumors from MRI images, and promising classification performance has been reported. The traditional machine learning-based brain tumor classification techniques often consist of preprocessing, segmentation, feature extraction, and classification stages.

2.3 REST API

Utilizing REST APIs for AI model implementation in webpages is a common practice due to several compelling reasons. REST APIs provide a standardized way to enable communication between the client and server, making it easier to integrate AI models into existing systems. They operate over HTTP, which is widely supported and understood by developers, ensuring compatibility and ease of use. Moreover, REST APIs are stateless; they treat each request as an independent transaction that is unrelated to any previous request, which simplifies the architecture and improves scalability. This statelessness also allows for the responses to be cached, improving performance for frequently requested resources. The deployment of AI models via REST APIs facilitates the separation of concerns, allowing the AI model to be hosted on specialized infrastructure that is optimized for computation, while the web application can be hosted on infrastructure that is optimized for serving web content. This separation also means that updates to the AI model can be made independently of the web application, reducing downtime, and improving maintainability. Additionally, REST APIs enable the AI model to be consumed by a variety of clients, not just web applications, but also mobile apps, desktop applications, and other services, making the AI model more versatile and increasing its potential impact. Furthermore, REST APIs support a range of HTTP methods, such as GET, POST, PUT, and DELETE, which correspond to create, read, update, and delete (CRUD) operations. This aligns well with the actions that might be performed on AI models, such as training, querying, updating, and deleting. For instance, a POST request can be used to send data to the AI model for inference, and the model's response can be returned in the API response. This makes the interaction with the AI model intuitive for developers, as it follows the same patterns as other web development tasks. In terms of security, REST APIs can be secured using standard methods such as HTTPS, OAuth, and JWT tokens, ensuring that the data exchanged between the client and the AI model is protected. This is particularly important when dealing with sensitive data that may be used in AI model processing. The use of REST APIs also allows for detailed logging and monitoring of interactions with the AI model, which is crucial for debugging, performance tuning, and compliance with regulatory requirements. Lastly, the use of REST APIs for AI model implementation aligns with modern development practices, such as microservices architecture, where applications are composed of small, independent services that communicate over well-defined APIs. This approach promotes agility, as teams can develop, deploy, and scale services independently, and it allows for the reuse of services across different applications. In summary, REST APIs offer a flexible, secure, and efficient way to integrate AI models into webpages and other applications, aligning with contemporary software development methodologies and enabling the broader use of AI capabilities. For more detailed insights on deploying machine learning models as APIs, resources like GeeksforGeeks provide comprehensive guides. Additionally, for a deeper understanding of REST API principles and benefits, websites like ai-jobs.net and testfully.io offer valuable explanations.

2.4 Detailed analysis of the approach

The used approach towards the tumor detection from the MRI scan datasets required for classification of the dataset, along with the customization of the image to the desired colour grading required for the model to test and train itself which is grayscale for the model and RGB for the user to understand . Also, the image needs to be resized to the proper dimension for testing and training. A lot of variables have been used for representing different aspects for the training and testing of the model for datasets.

2.4.1 Importing TensorFlow and Keras libraries:

This code imports the tensorflow and keras Python libraries. The tensorflow library contains tools for building and training machine learning models. The keras library contains tools for building neural networks specifically, which are a type of machine learning model.
Importing these two libraries allows us to use the machine learning tools they provide in the rest of the code. The tools imported include things like model architectures, layers, optimizers, losses, and metrics.

2.4.2 Importing Matplotlib library:

This imports the matplotlib Python library which provides functionality to plot graphs and visualize data. This will allow us to visualize things like accuracy/loss curves when training models or visualizing the predictions of a model on test data. Having visualization capabilities is important for understanding how machine learning models are behaving.
Overall, these three import statements bring in external libraries that will provide the key building blocks needed to build, train, evaluate, and visualize machine learning models in the rest of the code. They don't execute any training or processing themselves, just import the tools needed to do so later.

2.5 Splitting the dataset into the Training set Test set and Validation Set

This code defines a function called get_dataset_partitions_tf that splits a dataset into training, validation and test subsets. It takes a dataset ds as input, along with the desired percentages for train, validation and test splits (defaulting to 0.8, 0.1 and 0.1). There is also a shuffle parameter to randomly shuffle the dataset first if needed, and a shuffle_size parameter to control the shuffling buffer size. The function first checks that the split percentages add up to 1.0. It then gets the size of the dataset and calculates the number of samples that should go in each split based on the percentages. If shuffling is enabled, it will shuffle the dataset with the specified buffer size. It then uses take and skip to slice up the dataset into the train, validation and test subsets accordingly. The train set takes the first train_size samples, the validation set takes the next val_size after skipping the train set, and the test set takes whatever remains after skipping the train and validation. Finally, it returns the three subset datasets train_ds, val_ds and test_ds. So, in summary, it takes a dataset, optional split percentages and shuffling parameters, splits the dataset into training/validation/test sets based on those percentages, and returns the three subset datasets. This is a common pattern for preparing data for training, validation and testing of a machine learning model. The selected code snippet get_dataset_partitions_tf splits a dataset into training, validation, and test subsets. It takes as input a dataset object that contains the full dataset. The dataset is assumed to contain examples that can be used for training, validating, and testing a machine learning model. The code first calls a function called get_dataset_partitions_tf, passing the dataset object as a parameter. This function handles splitting the dataset into three separate subsets - train_ds, val_ds, and test_ds. The get_dataset_partitions_tf function likely uses some logic to split the dataset randomly, putting a certain percentage of examples into each of the three subsets. Common splits are 60% train, 20% validation, and 20% test. The three subset datasets - train_ds, val_ds, and test_ds - are returned from get_dataset_partitions_tf and assigned to those same variable names. Finally, the code prints out the length of each subset dataset to confirm they have been created properly. The lengths should add up to the total number of examples in the original full dataset. In summary, this code takes a full dataset, splits it into train/validation/test sets, assigns those to variables, and prints their lengths. This is a common data preparation step when working with machine learning datasets in order to create proper subsets for training and evaluating models. This code is preprocessing three TensorFlow datasets - train_ds, val_ds, and test_ds - to improve training performance. It takes as input the three TensorFlow dataset variables that have already been defined and loaded with data elsewhere in the code. The output is the same three dataset variables, but now modified to cache, shuffle, and prefetch the data they contain. The cache () method keeps a copy of the dataset in memory to avoid reloading it from disk on each iteration. The shuffle (1000) randomly reshuffles the data to help prevent bias and overfitting during training. The number 1000 controls the size of the shuffle buffer. The prefetch () method overlaps data preprocessing and model execution by preloading batches in the background while the model is training on the current batch. So, in summary, these three chained methods call on each dataset are optimizing data loading, randomizing data order, and pipelining data processing to make the overall training process more efficient and effective. By applying these optimizations to the training, validation, and test datasets, the model will be trained on high-quality, randomized data batches fed efficiently to maximize performance.

2.6 Preparing the dataset using data augmentation.

The resize_and_rescale Sequential model performs image preprocessing on input images. It takes as input images of any size. The first layer Resizing resizes the images to a fixed size of IMAGE_SIZE x IMAGE_SIZE pixels. This standardizes the input image dimensions. The second layer Rescaling scales the pixel values to be between 0 and 1 by dividing each pixel value by 255. This puts the values in a common range for easier learning. By chaining together these two preprocessing layers into a Sequential model, resize_and_rescale applies these transformations back-to-back to any input image. The overall purpose is to prepare images for consistent input shape and value range before feeding into other models like neural networks. The output is images resized to a standard square size and with values normalized to 0-1. No other major logic or data transformations occur. By standardizing image sizes and pixel values, the resize_and_rescale model aims to simplify the image data for more effective training and inference. The simple preprocessing helps set up the input data for the best performance of later models. The code snippet data augmentation defines a tf.keras.Sequential model for data augmentation in image classification. It takes as input images from the training dataset. The purpose is to artificially expand the size and diversity of the training dataset through random transformations of the input images. This helps prevent overfitting and improves the generalization of the model. It contains two layers that perform random image transformations. The first layer, RandomFlip, randomly flips the images horizontally and vertically. The second layer, RandomRotation, randomly rotates the images up to 20% of the total range. The output is a transformed version of the input image. Multiple transformed versions of each original image can be generated through the random augmentations. The layers are wrapped in a tf.keras.Sequential model, which chains them together into a data processing pipeline. Each input image passes through the two augmentation layers in sequence. In summary, the data_augmentation code defines a simple data augmentation model that artificially grows the size and diversity of a training dataset through random flipping and rotation of input images. This is a common technique in image classification to improve model training. The selected code is applying a data augmentation transformation to the training data in the train_ds dataset. It takes the train_ds dataset as input, which contains the training data of image, label pairs (x, y). It uses the .map() method to apply a lambda function to each (x, y) pair in train_ds. The lambda function calls the data_augmentation() function, passing the image x and setting training=True. This augments the image data in some way (like random cropping, flipping etc) to generate more varied training data. The data_augmentation() function returns the augmented image, and the original label y is passed through unchanged. So, for each (original image, label) pair, it outputs a (new augmented image, original label) pair. This has the effect of expanding and diversifying the training data through data augmentation transformations, which can help the model generalize better. The .prefetch() call at the end sets up a buffer to optimize loading of batches during training. Overall, the code takes the original training data, transforms each image while preserving labels, and returns an augmented dataset to use for training the model. The augmentation helps expose the model to more variations in the training data for better generalization.

2.6.1 Batch size

A variable used the control how many images are processed together in each training batch. Setting a larger batch size can speed up training but requires more memory.

2.6.2 Image_size

Another variable IMAGE_SIZE variable specifies the height and width dimensions that all images will be resized to before being fed into the model. This standardizes the input data.

2.6.3 Channels

Variable CHANNELS used for specifying how many colours the input images have – typical values are 1 for grayscale or 3 for colour RGB images.

2.6.4 Epochs

The EPOCHS variable sets the number of full passes through the training dataset the model will take during training. More epochs allow the model to train for longer and possibly achieve better accuracy but can increase training time.dataset = tf.keras.preprocessing.image_dataset_from_directory() loads images from a directory to create an image dataset. It takes as input the path to a directory containing subfolders of images ("Major_Project/Training"). The images in each subfolder will be labelled with the subfolder name. It also takes additional configuration parameters:

Seed: sets random seed for shuffling the data shuffle: whether to randomly shuffle the images image_size: resize images to this size batch_size: number of images per batch The output is a TensorFlow Dataset object (dataset) that can be used for training and evaluation.

Internally, it scans the directory structure, reads images from disk, decodes them into tensors, resizes them, and shuffles/batches them. This allows easy loading of a directory of images into a consistent format for a machine learning model to consume. By configuring parameters like image size and batch size, you can customize how the images are prepared as training data. The shuffle and seed parameters control randomization to help your model generalize better. Overall, this provides a simple way to go from a directory of images to a standardized dataset for image classification. Together these variables configure some key hyperparameters or settings that will shape the image classification model's training process and performance. They don't directly process any inputs or outputs but set stage for the model training code that will follow later using these values.C**lass name** is a variable that is assigned the class_names attribute from the dataset object. This code snippet is retrieving the list of class names from a dataset that has been used to train a classification model. The class_names attribute contains the names of the different classes that the model can predict. For example, if this is an image classification model, class_names may contain labels like ["cat", "dog", "bird"]. By assigning dataset.class_names to class_names, we are extracting just the list of class names from the dataset object and storing it in a separate variable. This allows us to access the class names directly later, without having to go through the dataset object each time. The class_names variable will contain a list of strings, with each string being the name of a class from the dataset. This will likely be used later when making predictions with the model, to map the model's numeric predictions to the actual class names for interpretation. For example, if the model predicts "2", we could use class_names to see that the predicted class is "dog". In summary, this code extracts the class names from a classification dataset into a separate variable for later use in interpreting predictions. It takes the dataset object as input, and produces a list of class name strings as output. By storing the class names, it avoids repetitive calls to dataset.class_names and makes the class names directly accessible. This code snippet is iterating through the dataset and printing information about the first batch. The for loop is used to iterate through the dataset, with each iteration providing an image batch and corresponding labels batch. The dataset.take limits it to just taking the first batch, rather than iterating through the full dataset. On each iteration, it prints the shape of the image batch, which shows the dimensions of the batch of images. It then prints the actual label values for the batch by calling .numpy() on the labels_batch tensor to convert it to a NumPy array. So, in summary, it is simply extracting the first batch of images and labels from the dataset, then printing some information to inspect what the batch contains - the image shapes and the label values. This is useful for sanity checking that the data pipeline is working as expected by peeking at the first batch of data being fed into the model during training.

The key steps are:

Iterate through dataset in batches using a for loop Take just the first batch using dataset.take(1) Print the shape of the image batch Print the actual label values by converting the tensor to numpy, So it provides a simple sanity check on the dataset by printing some info about the first batch of images and labels.
This code is splitting a dataset into training, validation, and test sets for use in machine learning model development. First, it calculates the length of the full dataset to get the total number of samples. It then defines the training set size as 80% of the full dataset length and takes that number of samples from the start to create the training set. Next, it calculates the validation set size as 10% of the full dataset length and takes that number of samples from the start of the remaining data to create the validation set. The test set is created by skipping the validation samples from the remaining data after the training set was taken. So, in summary, it takes the full dataset, splits off 80% to training, then 10% to validation from the remaining 20%, and leaves the final 10% as the test set. This follows a common machine learning practice of creating disjoint training, validation, and test sets from a full dataset in order to properly develop, tune and evaluate a model. The training set is used to train the model, validation set is used to tune hyperparameters and monitor performance during training, and the test set is held back completely until final model evaluation.

2.7 CREATING THE MODEL

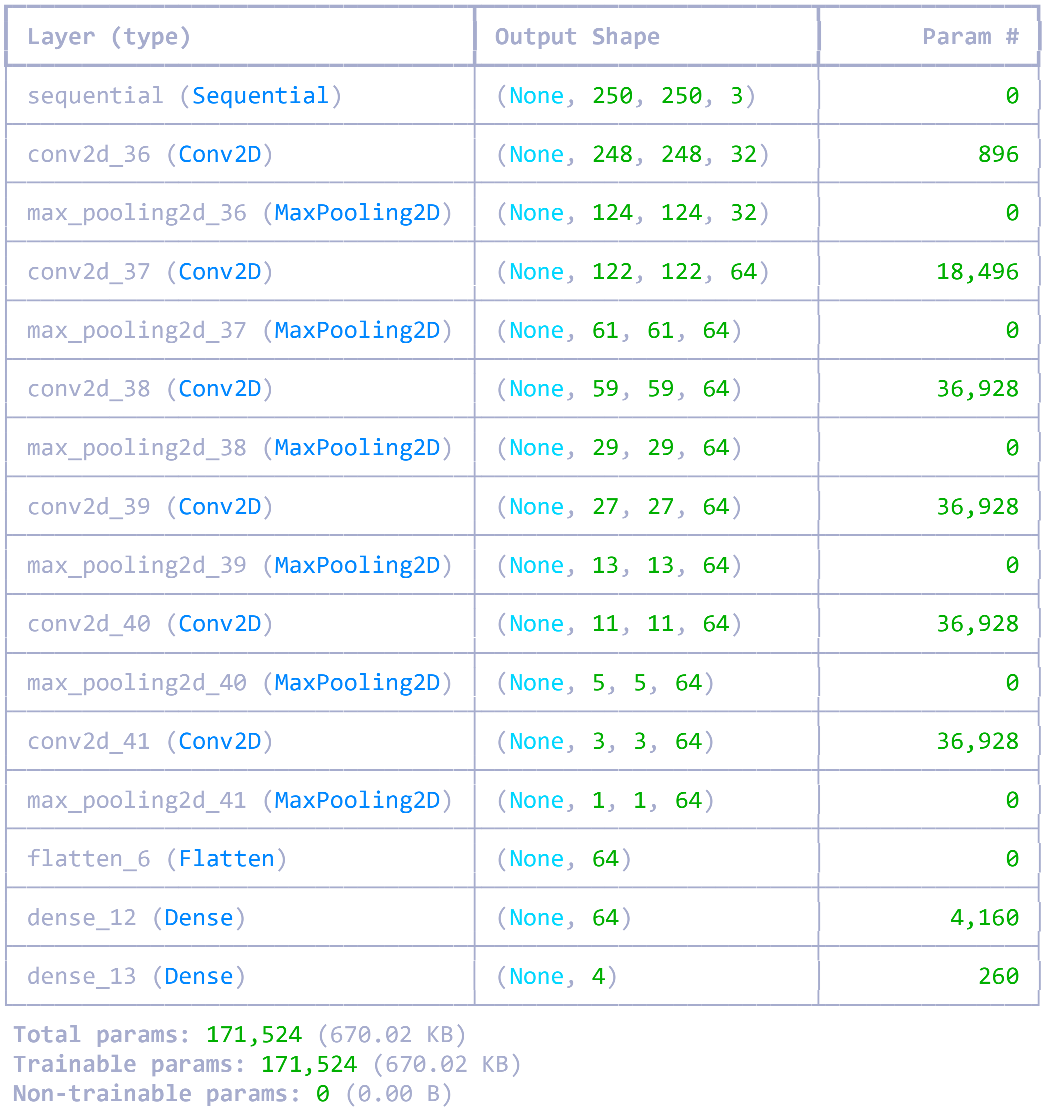
The model that does the main work of tumor detection within the scanned magnetic resonance image of the brain. There is a sequential model in Keras for an image classification task. The variable model.Sequential() constructor creates a Sequential model, which stacks layers linearly. This allows us to define the model layer by layer.
The first layer is a resize_and_rescale layer which presumably resizes and rescales the input images as preprocessing.
The next layers are Convolutional 2D layers (Conv2D) which apply convolutions over the image to extract features. The kernel size of 3x3 is specified to define the size of the convolution window. ReLU activation is used for non-linearity.
After each Conv2D layer, a MaxPooling2D layer is added to reduce the spatial dimensions and perform down sampling. The pool size of 2x2 is specified for max pooling.
As we **stack more Conv2D and MaxPooling2D layers,** the model is able to **extract higher-level features** from the images.
After the convolutional base, the Flatten layer flattens the feature maps into a 1D vector. This is fed into a fully-connected Dense layer with ReLU activation to perform classification.
Finally, **the output layer has 4 nodes** (defined by n_classes) with SoftMax activation to output classification probabilities for the 4 classes.
In summary, this model takes input images, applies convolutions and pooling to extract features, flattens the features, and applies fully-connected layers to classify the images into 4 classes based on the extracted features. The overall architecture is designed for image feature extraction and classification.
ReLU, which stands for Rectified Linear Unit, is an activation function commonly used in neural networks, particularly in deep learning models. It is defined mathematically as:

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

Function Definition:
The ReLU function is defined as \(f(x) = \max(0, x) \).
If the input \(x\) is positive, the output is equal to \(x\).
If the input \(x\) is negative, the output is zero.
Non-Linearity:
ReLU introduces non-linearity into neural networks.
This non-linearity allows neural networks to learn complex patterns and relationships in data.
Gradient Propagation:
ReLU doesn't saturate (flatten) for positive inputs, which helps gradients flow during backpropagation.
This property is crucial for training deep networks effectively.
Vanishing Gradient Problem:
Traditional activation functions like sigmoid and hyperbolic tangent can suffer from vanishing gradients.
ReLU mitigates this issue by maintaining gradient flow for positive inputs.
Common Usage:
ReLU is widely used in hidden layers of deep neural networks.
It's computationally efficient and encourages sparse activations.

This means that if the input (x) is positive, the output will be (x). If (x) is negative, the output will be (0). Essentially, ReLU outputs the input directly if it is positive, otherwise, it will output zero.
ReLU is favored because it introduces non-linearity into the model without affecting the ability to propagate gradients, which is essential for training deep neural networks. It helps to overcome the vanishing gradient problem, allowing models to learn faster and perform better compared to when sigmoid or hyperbolic tangent activation functions are used.

Note: - while ReLU is powerful, it has limitations too (e.g., the “dying ReLU” problem for negative inputs). Researchers have proposed variants like Leaky ReLU, Parametric ReLU, and Exponential Linear Units (ELUs) to address these limitations.



```
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
│ sequential (Sequential)         │ (None, 250, 250, 3)    │             0 │
│ conv2d_36 (Conv2D)              │ (None, 248, 248, 32)   │           896 │
│ max_pooling2d_36 (MaxPooling2D) │ (None, 124, 124, 32)   │             0 │
│ conv2d_37 (Conv2D)              │ (None, 122, 122, 64)   │        18,496 │
│ max_pooling2d_37 (MaxPooling2D) │ (None, 61, 61, 64)     │             0 │
│ conv2d_38 (Conv2D)              │ (None, 59, 59, 64)     │        36,928 │
│ max_pooling2d_38 (MaxPooling2D) │ (None, 29, 29, 64)     │             0 │
│ conv2d_39 (Conv2D)              │ (None, 27, 27, 64)     │        36,928 │
│ max_pooling2d_39 (MaxPooling2D) │ (None, 13, 13, 64)     │             0 │
│ conv2d_40 (Conv2D)              │ (None, 11, 11, 64)     │        36,928 │
│ max_pooling2d_40 (MaxPooling2D) │ (None, 5, 5, 64)       │             0 │
│ conv2d_41 (Conv2D)              │ (None, 3, 3, 64)       │        36,928 │
│ max_pooling2d_41 (MaxPooling2D) │ (None, 1, 1, 64)       │             0 │
│ flatten_6 (Flatten)             │ (None, 64)             │             0 │
│ dense_12 (Dense)                │ (None, 64)             │         4,160 │
│ dense_13 (Dense)                │ (None, 4)              │           260 │

 Total params: 171,524 (670.02 KB)
 Trainable params: 171,524 (670.02 KB)
 Non-trainable params: 0 (0.00 B)
```

This section of the content is configuring and compiling a neural network model in TensorFlow.
It starts by calling the compile method on the model variable, which is assumed to be a neural network model that has already been defined.
The compile method takes several arguments that specify how the model should be configured before training:
optimizer='adam' - This specifies that the Adam optimization algorithm should be used to update the model weights during training. Adam is a popular algorithm that adapts the learning rate as training progresses.
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False) - This specifies the loss function that will measure how well the model is doing on the training data. Sparse categorical cross entropy is a common loss for classification problems. The from_logits=False tells it that the model outputs are not raw logit values but have already been transformed into probabilities.
metrics=['accuracy'] - This specifies that the accuracy of the model's predictions should be tracked during training as an additional metric beyond the main loss function.
After compile is called, the model is fully configured and ready to be trained by calling the fit method (not shown here). The compile step handles setting up the training internals like the optimizer, loss function, and metric s.

So, in summary, this code configures and prepares a Keras neural network model for training by specifying the optimization algorithm, loss function, and metrics like accuracy that will be used and tracked during the upcoming training process. The compile call gets the model ready for efficient training.

2.8 TRAINING THE MODEL

Training is the most vital process in this approach as the model needs training again and again it is a lengthy process where a dataset arraigned into sets of all the different types of images as in the different type of brain tumor present in the MRI which then is filled into the model for training. A variable train_ds : the training dataset containing images and labels to train the model on batch_size: the number of samples per gradient update validation_data: a dataset to evaluate the model on after each epoch verbose: whether to print progress during training epochs: the number of passes through the full training set It trains the model by looping through the training data in batches for the specified number of epochs. For each batch, it makes a prediction, compares it to the true label to calculate the loss, and updates the model weights through backpropagation. After each epoch, it evaluates the model on the validation set to monitor progress.
The output is a history object containing the training loss, validation loss, and other metrics logged during training. This shows how the model improved over time. By repeatedly looping through the data to minimize the loss, the model learns to make predictions matching the true labels. The validation data helps prevent overfitting by tracking performance on data not used in training. In summary, this code fits a neural network model by training it on labelled example data over a number of epochs and tracking its progress on a validation set. The end result is a trained model that can accurately classify new examples based on patterns learned from the training data.

2.9 TESTING THE MODEL

In order to detect and classify the brain tumor the model needs to be trained firstly and the tested with n number of attempts before the tumor can be detected. Evaluation of the whole training is noted by the model which is then represented by model.evaluate(). It takes two inputs: test_ds - This is a TensorFlow dataset containing the test data that the model performance will be evaluated on. This should be data that the model has not seen during training. model - This is the trained machine learning model whose performance we want to evaluate. This should have already been trained on a separate training dataset. It returns a list of metric values reflecting how well the model performed on the test data. The exact metrics returned depend on the model, but usually include metrics like accuracy, precision, recall, F1 score, etc. Internally, model.evaluate() iterates through each sample in the test_ds, makes predictions using the model, and compares those predictions to the true labels. It aggregates the results across the entire test set to compute the overall performance metrics. So, in summary, model.evaluate() aims to quantify how well a trained machine learning model generalizes to new, unseen data. The metrics it returns give a numerical assessment of the model's real-world performance and ability to make accurate predictions. Checking performance on test data helps identify overfitting and other issues during training before the model is deployed. The test performance metrics are very important to consider before putting a model into production. (figure 8 represents the graph of training and validation accuracy and loss of the training of model)

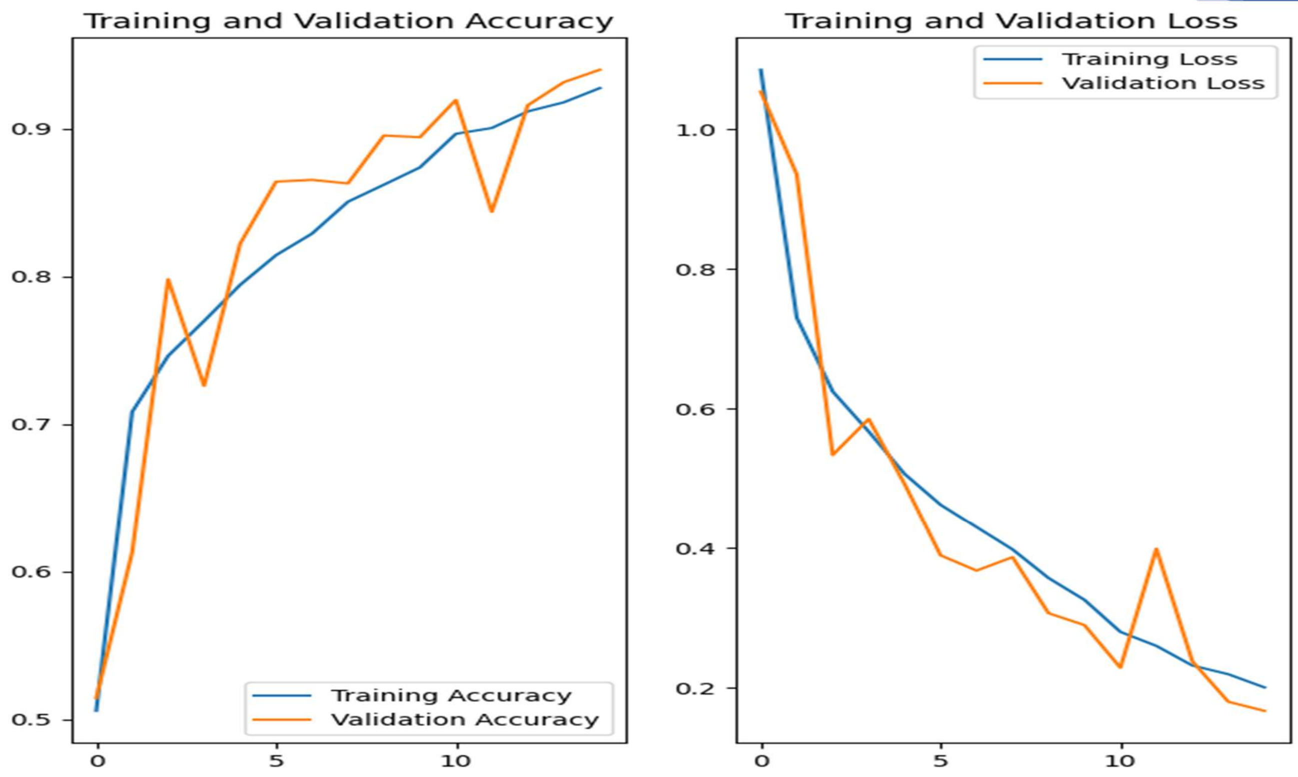


Fig (9) graph of the validation accuracy and loss of the training.

This above graph plots training and validation accuracy and loss for a neural network model over epochs.
The plt.figure() call creates a new figure (graph) with specified width and height.
plt.subplot() divides this figure into 2 subplots vertically.
The first subplot on the left plots the training and validation accuracy over epochs. The range () function generates the x values from 0 to EPOCHS, ace and val_acc contain the training and validation accuracy values for each epoch. These are plotted using plt.plot(). Labels are added with plt.legend(). A title is added with plt.title().
The second subplot on the right plots the training and validation loss over epochs, similar to the accuracy plot. loss and val_loss contain the training and validation loss values.
Overall, this code generates a figure with two subplots side by side - one showing model accuracy over epochs, and the other showing model loss over epochs. This is a common way to visualize model performance during training. The training and validation curves allow comparing how well the model fits the training data vs generalizes to new data over time.

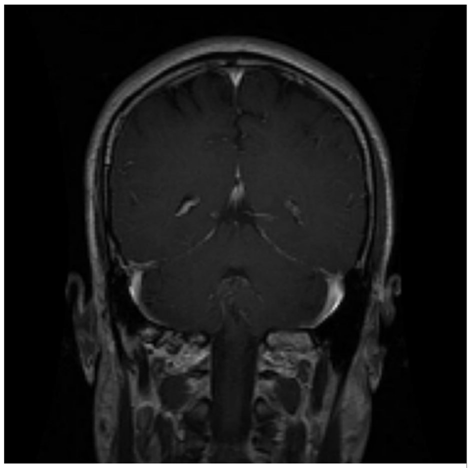


Fig (10) the first image to predict.

OUTPUT-actual label: glioma_tumor, predicted label: glioma_tumor ,0s 216ms/step (Fig-10)

The predict function is used to make predictions using a trained machine learning model. It takes a model and an image as inputs. 39 The image is first converted into a numpy array using img_to_array. This puts the image data into a format the model can understand. The image array is then reshaped into a single sample with expand_dims. This converts the flat array into a "batch" of one image. The model makes a prediction on the image by calling model.predict. This runs the image through the model and returns the predicted classes and probabilities. The class with the highest probability is considered the model's predicted class. The index of this class is found using np.argmax. The class name itself is looked up from the class_names list using the predicted index. The confidence score is calculated by taking the maximum predicted probability and converting to a percentage. The predict function returns the predicted class name and confidence percentage as outputs. So, in summary, it takes an image, runs it through a trained model, looks up the best matching class, and returns the prediction - converting the outputs into a readable class name and confidence score along the way. The main logic flow is preparing the data, making the prediction, and interpreting the results.

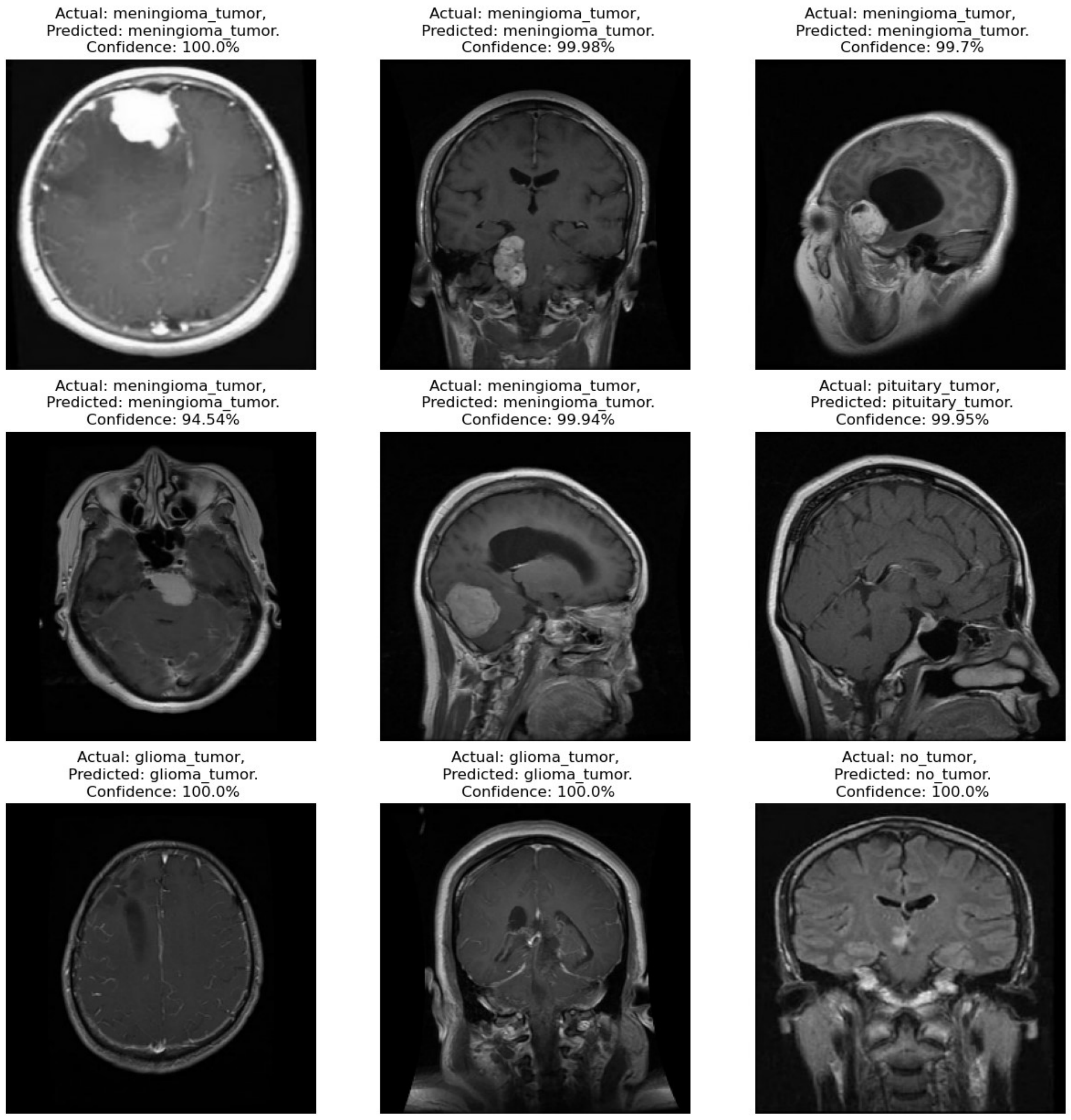


Fig (11) Shows the final result of the confidence level.

A 3x3 grid of subplot axes visualizing 9 test images, their true vs predicted classes, and the model's confidence percentage. The main logic flow is: Loop through the first batch of images and labels from the test dataset For each of the first 9 images: Create a subplot and display the test image Make a prediction on the image using the model Get the true label by looking up the integer index in class_names Print the true and predicted classes and confidence percentage as the subplot title Turn off axes lines to make a clean visualization So in summary, it takes a batch of test data, runs the images through the model to make predictions, compares to the true labels, and visualizes the results in a grid to provide an overview of how well the model performs on real examples. The main steps are the prediction logic and mapping between integer encodings and human-readable class names. This provides an intuitive visualization for assessing the model's performance.

Number of accurate predictions: 880

Number of wrong predictions: 16

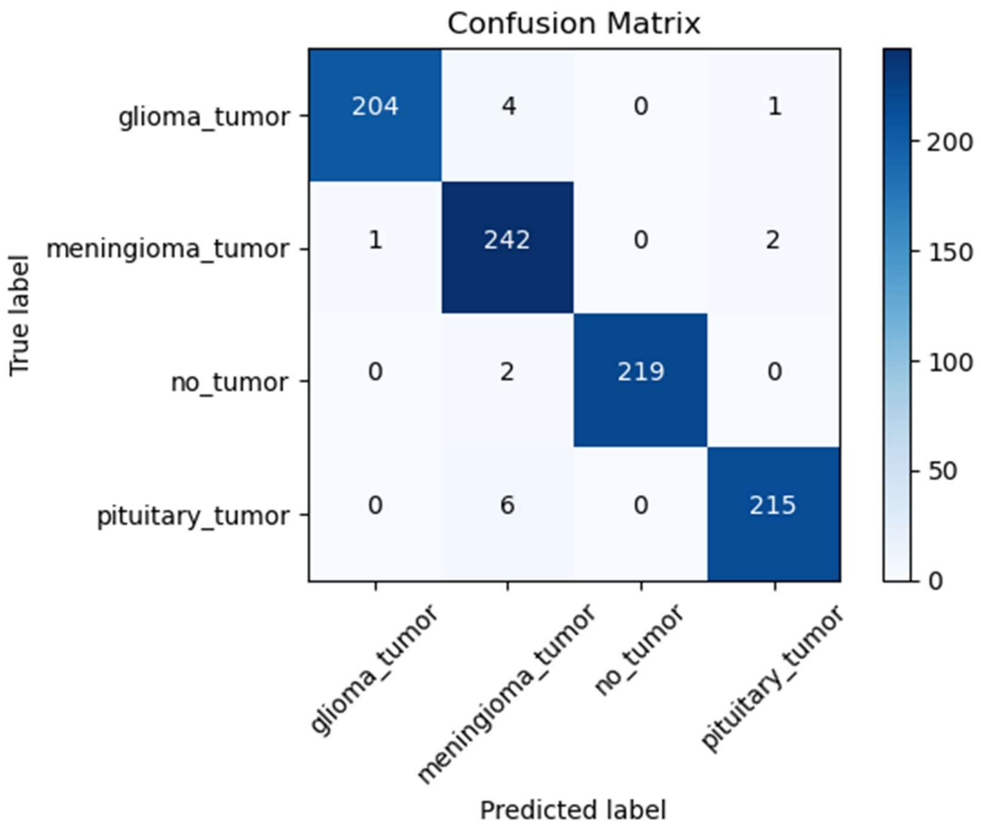


Fig (11) shows the confusion matrix of the testing stage

2.10 Frontend (HTML5, CSS, JS, BOOTSTRAP)

In this project we have used HTML, CSS, JavaScript, Bootstrap 5.1 and CanmanJS for the development of the frontend and the connection with the server and backend. The structures used for programming languages are given below:

2.10.1 HTML STRUCTURE

The HTML code represents a web page for brain tumor classification.

It consists of the following sections:

Header Section:

Displays the title ("MRI Brain Tumor Classification") and a brief description.

Includes a link ("Check Now") to navigate to the main content section.

3D Brain Model Section:

Embeds a 3D model of a plastinated human brain using an iframe.

The model is interactive and can be explored.

Main Content Section:

Divided into two columns:

Dataset Upload Section:*

Allows users to upload image files (presumably MRI scans) for tumor classification.

Features:

A file input field (<input type="file">) where users can select multiple image files.

A progress area (to display progress during file upload).

An uploaded area (to display results after processing).

A "Submit" button that triggers the submit() function (not shown in this snippet).

2.10.2 JavaScript and Interaction

The web page relies on JavaScript for functionality.

The script.js and main.js files are included.

The vanilla-tilt.min.js library provides an interactive effect for the "Upload Files" element (tilt effect).

2.10.3 Server-Side Implementation (7)

The actual server-side implementation is not provided in this snippet.

The /predict endpoint (mentioned in the FastAPI code) should handle image processing and classification.

Key steps in the server-side implementation:

Read image data from the uploaded file.

Convert the image data to a numpy array.

Call a pre-trained model for classification.

Extract the predicted class and confidence.

Convert the image data to base64 format.

Return classification results as a JSON response.

2.11 SERVER SITE PROGRAMMING

Server-side programming refers to code and processes that occur on servers, away from the user's browser (8). In contrast, client-side programming involves code (usually HTML, CSS, and JavaScript) that runs within the user's browser. The primary purpose of server-side programming is to handle dynamic content generation, data processing, and interaction with databases. When a user interacts with a website (e.g., clicks a link, submits a form, or performs a search), their browser sends an HTTP request to the web server. The request includes**:**

A URL identifying the resource.

An HTTP method (e.g., GET, POST) that defines the required action.

Additional information encoded in URL parameters, POST data, or cookies.

The web server processes the request, constructs an appropriate response, and sends it back to the browser.

The response contains an HTTP status line (e.g., "HTTP/1.1 200 OK" for success) and the requested resource (e.g., an HTML page or an image).

Server-side code dynamically generates the response based on the user's request.

Procedure:

Define an API using FastAPI.

Set up CORS middleware to allow specific origins.

Load a pre-trained TensorFlow model (named MODEL) for image classification.

Define class names (CLASS_NAMES) corresponding to different tumor types.

Implement an endpoint (/ping) to check if the API is alive.

Implement an endpoint (/predict) to process uploaded image files:

Read the image data from the uploaded file.

Convert the image data to a numpy array.

Expand the image dimensions to create a batch.

Call the pre-trained model on the input data.

Extract the predicted class and confidence from the model's output.

Convert the image data to base64 format.

Return the classification results (class, confidence, and base64 image) as a JSON response.

Implementation Details:

The server listens on http://localhost:8000.

The /ping endpoint responds with "Hello, I am alive."

The /predict endpoint processes uploaded image files and returns classification results.

Ensure that the pre-trained model (Model_Main/3) is available and properly configured.

The read_file_as_image function converts raw image data to a numpy array.

The CLASS_NAMES list contains tumor class labels.

The server-side implementation should handle actual image classification and prediction.

CORS Middleware:

The CORSMiddleware is essential for handling Cross-Origin Resource Sharing (CORS) in your API (9).

CORS allows you to control which origins (e.g., domains) are permitted to access your API.

By specifying allowed origins (e.g., localhost, 127.0.0.1, or localhost:3000), you ensure that requests from those origins can interact with your API. Figure 13 shows the full working of the frontend

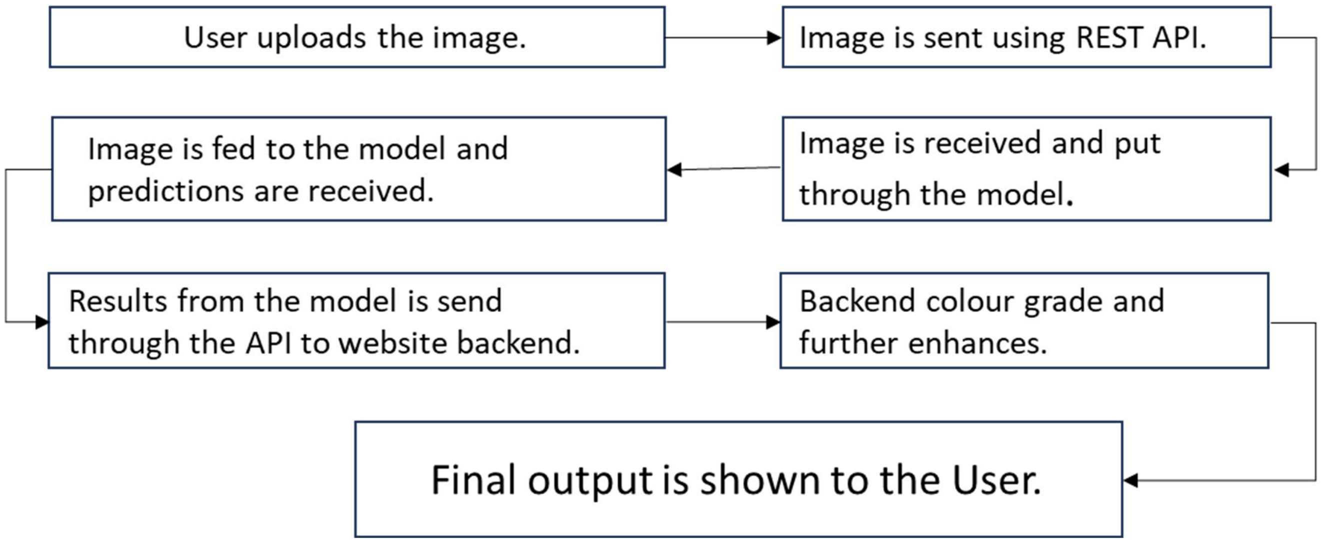

User uploads the image.
Image is sent using REST API.
Image is received and put through the model.
Image is fed to the model and predictions are received.
Results from the model is send through the API to website backend.
Backend colour grade and further enhances.
Final output is shown to the User.

Fig13 flowchart of the frontend

3. Results and Outcomes

As per the approach we have used results have gotten out after the model has predicted the type of tumor present inside the MRI picture scan of the brain. These results include the 3 kind of tumors that the model was fed upfront for training . Now , as the image fed into the model was just a normal JPEG/JPEC/PNG formats which then were converted into the required grayscale scale which is the output after this the class of the image and the confidence level of the prediction is shown to the user scanning an image.

4 Conclusion and future scope

In conclusion, the research presented in this dissertation demonstrates the significant potential of Convolutional Neural Networks (CNN) in the classification of brain tumors from MRI scans. The study's findings underscore the accuracy, efficiency, and reliability of CNNs in distinguishing between different types of brain tumors, which is a critical step in the diagnosis and treatment planning for patients. The implementation of CNN models has shown a remarkable ability to learn complex features from medical imaging data, outperforming traditional machine learning methods. This advancement could revolutionize the field of medical imaging and provide clinicians with powerful tools to enhance patient outcomes. Future work should focus on refining these models, improving their interpretability, and integrating them seamlessly into clinical workflows. The promise of AI in healthcare is immense, and this research is a testament to its transformative potential in oncology diagnostics.

In the realm of brain tumor classification, several machine learning techniques can complement Convolutional Neural Networks (CNNs) to enhance performance and accuracy. Ensemble methods, which combine multiple models to make a single prediction, can be particularly effective. Techniques such as Random Forests and Gradient Boosting Machines can provide robustness against overfitting and improve generalization by aggregating the predictions of numerous decision trees. Support Vector Machines (SVMs) are another valuable addition, known for their effectiveness in high-dimensional spaces, which is common in medical imaging data. SVMs can classify data with a clear margin of separation, making them useful for distinguishing between different tumor types. Transfer learning is another approach that can be utilized alongside CNNs. By leveraging pre-trained models on large datasets, transfer learning allows for the application of knowledge gained in one domain to be applied to another, which can be particularly beneficial when dealing with limited medical imaging data. Additionally, feature extraction techniques can be employed to reduce the dimensionality of the data and identify the most relevant features for classification, which can then be fed into machine learning models like k-Nearest Neighbours (k-NN) or Naive Bayes classifiers.

Moreover, deep learning techniques such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks can be used to analyse sequential data, which could be useful in tracking tumor development over time. Generative Adversarial Networks (GANs) can also be applied to generate synthetic medical images, which can augment the training data and improve the robustness of CNNs. Furthermore, advanced techniques like Quantum Machine Learning (QML) are emerging, which promise to exploit quantum computing principles to process information in fundamentally new ways, potentially leading to breakthroughs in complex tasks like brain tumor classification. Additionally, the integration of explainable AI (XAI) methods can help in making the decision-making process of CNNs more transparent and interpretable for clinicians. In summary, while CNNs are powerful tools for brain tumor classification, their performance can be significantly enhanced by integrating them with other machine learning techniques. These complementary methods can address the limitations of CNNs and provide a more holistic approach to the classification and analysis of brain tumors, ultimately leading to better diagnostic tools and patient outcomes.

5 REFERENCES

1. Najjar, R., 2023. Redefining radiology: a review of artificial intelligence integration in medical imaging. Diagnostics, 13(17), p.2760.

2. Rudie, Jeffrey D., Andreas M. Rauschecker, R. Nick Bryan, Christos Davatzikos, and Suyash Mohan. "Emerging applications of artificial intelligence in neuro-oncology." Radiology 290, no. 3 (2019): 607-618.

3. Pierre, K., Gupta, M., Raviprasad, A., Sadat Razavi, S.M., Patel, A., Peters, K., Hochhegger, B., Mancuso, A. and Forghani, R., 2023. Medical imaging and multimodal artificial intelligence models for streamlining and enhancing cancer care: opportunities and challenges. Expert Review of Anticancer Therapy, 23(12), pp.1265-1279.

4. Ozair, A., Bhat, V., Alisch, R.S., Khosla, A.A., Kotecha, R.R., Odia, Y., McDermott, M.W. and Ahluwalia, M.S., 2023. DNA methylation and histone modification in low-grade gliomas: current understanding and potential clinical targets. Cancers, 15(4), p.1342.

5. Louis, D.N., Perry, A., Wesseling, P., Brat, D.J., Cree, I.A., Figarella-Branger, D., Hawkins, C., Ng, H.K., Pfister, S.M., Reifenberger, G. and Soffietti, R., 2021. The 2021 WHO classification of tumors of the central nervous system: a summary. Neuro-oncology, 23(8), pp.1231-1251.

6. Modarres, Ceena, Nicolas Astorga, Enrique Lopez Droguett, and Viviana Meruane. "Convolutional neural networks for automated damage recognition and damage type identification." *Structural Control and Health Monitoring* 25, no. 10 (2018): e2230.

7. Jayade S., Ingole D.T., Ingole M.D. Review of Brain Tumor Detection Concept using MRI Images; Proceedings of the 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET); Shegoaon, India. 27–28 December 2019.

8. Yang Y., Yan L.F., Zhang X., Han Y., Nan H.Y., Hu Y.C., Hu B., Yan S.L., Zhang J., Cheng D.L., et al. Glioma Grading on Conventional MR Images: A Deep Learning Study With Transfer Learning. *Front. Neurosci.* 2018;12:804. doi: 10.3389/fnins.2018.00804.

9. Nazir M., Shakil S., Khurshid K. Role of deep learning in brain tumor detection and classification (2015 to 2020): A review. *Comput. Med. Imaging Graph.* 2021;91:101940. doi: 10.1016/j.compmedimag.2021.101940.

10. El-Kenawy E.S.M., Mirjalili S., Abdelhamid A.A., Ibrahim A., Khodadadi N., Eid M.M. Meta-Heuristic Optimization and Keystroke Dynamics for Authentication of Smartphone Users. *Mathematics.* 2022;10:2912. doi: 10.3390/math10162912.

11. El-kenawy E.S.M., Albalawi F., Ward S.A., Ghoneim S.S.M., Eid M.M., Abdelhamid A.A., Bailek N., Ibrahim A. Feature Selection and Classification of Transformer Faults Based on Novel Meta-Heuristic Algorithm. *Mathematics.* 2022;10:3144. doi: 10.3390/math10173144.

12. El-Kenawy E.S.M., Mirjalili S., Alassery F., Zhang Y.D., Eid M.M., El-Mashad S.Y., Aloyaydi B.A., Ibrahim A., Abdelhamid A.A. Novel Meta-Heuristic Algorithm for Feature Selection, Unconstrained Functions and Engineering Problems. *IEEE Access.* 2022;10:40536–40555. doi: 10.1109/ACCESS.2022.3166901.

13. Ibrahim A., Mirjalili S., El-Said M., Ghoneim S.S.M., Al-Harthi M.M., Ibrahim T.F., El-Kenawy E.S.M. Wind Speed Ensemble Forecasting Based on Deep Learning Using Adaptive Dynamic Optimization Algorithm. *IEEE Access.* 2021;9:125787–125804. doi: 10.1109/ACCESS.2021.3111408.

14. El-kenawy E.S.M., Abutarboush H.F., Mohamed A.W., Ibrahim A. Advance Artificial Intelligence Technique for Designing Double T-shaped Monopole Antenna. *Comput. Mater. Contin.* 2021;69:2983–2995. doi: 10.32604/cmc.2021.019114.

15. Samee N.A., El-Kenawy E.S.M., Atteia G., Jamjoom M.M., Ibrahim A., Abdelhamid A.A., El-Attar N.E., Gaber T., Slowik A., Shams M.Y. Metaheuristic Optimization Through Deep Learning Classification of

COVID-19 in Chest X-Ray Images. *Comput. Mater. Contin.* 2022;73:4193–4210. doi: 10.32604/cmc.2022.031147.
16. Lee G., Nho K., Kang B., Sohn K.A., Kim D. Predicting Alzheimer’s disease progression using multi-modal deep learning approach. *Sci. Rep.* 2019;9:1952. doi: 10.1038/s41598-018-37769-z.
17. Sharma K., Kaur A., Gujral S. Brain Tumor Detection based on Machine Learning Algorithms. *Int. J. Comput. Appl.* 2014;103:7–11. doi: 10.5120/18036-6883.
18. Agrawal M., Jain V. Prediction of Breast Cancer based on Various Medical Symptoms Using Machine Learning Algorithms; Proceedings of the 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI); Tirunelveli, India. 28–30 April 2022; pp. 1242–1245.
19. Rabbi M.F., Mahedy Hasan S.M., Champa A.I., AsifZaman M., Hasan M.K. Prediction of Liver Disorders using Machine Learning Algorithms: A Comparative Study; Proceedings of the 2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT); Dhaka, Bangladesh. 28–29 November 2020; pp. 111–116.
20. Swain D., Pani S.K., Swain D. A Metaphoric Investigation on Prediction of Heart Disease using Machine Learning; Proceedings of the 2018 International Conference on Advanced Computation and Telecommunication (ICACAT); Bhopal, India. 28–29 December 2018; pp. 1–6.