DETECTION OF BRAIN TUMOR USING DEEP LEARNING

Report submitted to

Techno India University, West Bengal

for the partial fulfilment

of

Bachelor of Technology (B. Tech.)

degree in

Computer Science & Engineering

By

Sohon Mondal 201001011009

Swati 191001001114

Manas Roy 201001011005

Anushka Sarkar 191001001290

Surashree Nag 201001011007

TECHNO INDIA UNIVERSITY

WEST BENGAL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

TECHNO INDIA UNIVERSITY, WEST BENGAL,

SALT LAKE, KOLKATA – 700091, INDIA

May 2023

CERTIFICATE OF APPROVAL

This is to certify that the project entitled, "**DETECTION OF BRAIN TUMOR USING DEEP LEARNING**" submitted by "SOHON MONDAL, SWATI, MANAS ROY , SURASHREE NAG & ANUSKA SARKAR " to Techno India University, West Bengal, India, is a record of Bonafide Project work carried out by them under my supervision and guidance.

In my opinion the report has fulfilled the requirement for the award of the Degree of Bachelor of Technology in Computer Science and Engineering under Techno India University, West Bengal.

The work has reached the standard necessary for submission and to the best of my knowledge.

_____

Supervisor(s)
Date:

_____

HOD, CSE, Techno India University
Date:

DECLARATION

We, the undersigned, hereby declare that the work presented in this report proposal titled " DETECTION OF BRAIN TUMOR USING DEEP LEARNING" has been conducted by us and represents our own ideas, unless otherwise stated project is submitted to the Computer Science and Engineering Department, Techno India University, West Bengal, for the award of the Bachelor of Technology degree in Computer Science and Engineering. Wherever we have included ideas, concepts, or words from other sources, we have duly acknowledged and referenced them. We further declare that we have complied with the principles of academic honesty and integrity throughout this project. We have not misrepresented, fabricated, or falsified any ideas, data, facts, or sources in our submission. We take full responsibility for the authenticity and accuracy of the information presented in this report. We understand that any violation of the aforementioned principles may result in disciplinary action by Techno India University, West Bengal, and may also lead to legal consequences if proper citations or permissions were not obtained.

ACKNOWLEDGEMENT

We would first like to thank our thesis supervisor Prof. Sauvik Bal, Associate Professor (Teacher in Charge), CSE Dept, TIU. He helped us whenever we ran into a trouble spot or had a question about our research or writing.

We take this opportunity to express gratitude to all of the Department faculty members for their help and support.

We also thank our parents for the unceasing encouragement, support and attention and also sense of gratitude to one and all, who directly or indirectly, have bestowed their hand in this thesis.

_____

Sohon Mondal

_____

Swati

_____

Manas Roy

_____

Anuska Sarkar

_____

Surashree Nag

Contents

Note: The content pages outline provided above is a general structure and can be tailored and expanded upon based on the specific requirements and scope of your final year project on brain tumor detection using deep learning.

Abstract

Brain tumor detection plays a critical role in early diagnosis and treatment planning. Deep learning architectures have shown great promise in this domain, offering automated and accurate analysis of medical images. This abstract compares different deep learning architectures, including Convolutional Neural Networks (CNNs), U-Net, Deep Convolutional Neural Networks (DCNNs), Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM), Capsule Networks (CapsNets), and Artificial Neural Networks (ANNs), for brain tumor detection.

CNNs are widely used for image classification tasks and have been successfully applied to brain tumor detection. They leverage convolutional layers to extract hierarchical features from images. U-Net, a popular architecture for semantic segmentation, incorporates skip connections to preserve spatial information and has demonstrated effectiveness in brain tumor detection tasks, especially with limited labelled training data.

ANNs, the foundational building blocks of deep learning models, offer simplicity and efficiency. While not as complex as deep architectures, ANNs can still deliver satisfactory results for certain brain tumor detection tasks, especially when data is limited or the task is relatively straightforward.

The choice of architecture depends on factors like available datasets, task complexity, and resource constraints. Experimentation and comparative evaluations are typically necessary to identify the most suitable architecture for a given task. It is important to consider the specific requirements of the brain tumor detection task, the nature of the input data, the dataset size, computational resources, and previous research and benchmarking results.

Chapter 1

Introduction

1.1 Background and significance of brain tumor detection

Brain tumor detection is a crucial aspect of medical diagnostics that involves identifying the presence of abnormal growths within the brain. These tumors can be either benign (non-cancerous) or malignant (cancerous), and their timely detection is essential for appropriate treatment planning and patient care. The significance of brain tumor detection lies in its potential to save lives and improve patient outcomes. Accurate diagnosis also enables healthcare professionals to differentiate between different types of brain tumors, guiding the selection of appropriate treatment strategies. Furthermore, brain tumors can often present with symptoms that mimic other neurological conditions, making their detection challenging. The field of brain tumor detection has seen significant advancements in recent years, driven by technological progress and improved understanding of tumor biology. Imaging modalities such as magnetic resonance imaging (MRI), computed tomography (CT), and positron emission tomography (PET) scans play a vital role in visualizing and characterizing brain tumors. Efforts are also underway to develop non-invasive and more accessible methods for brain tumor detection, such as blood-based biomarkers and liquid biopsies.In conclusion, brain tumor detection is a critical aspect of medical practice, offering significant benefits in terms of patient outcomes, treatment planning, and disease management.

1.2 Overview of deep learning in medical image analysis

Deep learning has emerged as a powerful technique in the field of medical imaging, offering promising advancements in the diagnosis and treatment of various conditions, including paralysis. In the context of medical imaging, deep learning algorithms can analyze large amounts of image data to identify and classify abnormalities with high accuracy. Paralysis, a condition characterized by the loss of muscle function and control, can be caused by various factors such as spinal cord injuries, stroke, or neurodegenerative disorders. Magnetic

resonance imaging (MRI) play a crucial role in diagnosing and monitoring paralysis-related conditions.Deep learning algorithms excel in analyzing medical images and have shown great potential in assisting healthcare professionals in diagnosing and treating paralysis. These algorithms can automatically detect and segment relevant anatomical structures, identify abnormalities, and provide quantitative assessments. By analyzing functional MRI (fMRI) data, deep learning algorithms can identify specific brain activity patterns associated with motor tasks, enabling the development of brain-computer interfaces (BCIs) that allow paralyzed patients to control external devices using their thoughts. As research in this field progresses, deep learning holds promise for improving the quality of care and outcomes for individuals affected by paralysis..

1.3 Motivation for using deep learning in brain tumor detection

Deep learning has emerged as a powerful tool in various domains, including medical imaging. One area where deep learning has shown promising results is in the detection and diagnosis of brain tumors. Brain tumors are a significant health concern, and their timely detection plays a crucial role in effective treatment and patient outcomes. Deep learning, a subset of artificial intelligence, offers a potential solution to enhance the accuracy and efficiency of brain tumor detection. The motivation for using deep learning in brain tumor detection lies in its ability to analyze large amounts of medical imaging data with remarkable speed and precision. Additionally, deep learning algorithms have the potential to continuously improve with more data, making them highly adaptable and scalable. They can handle different imaging modalities such as MRI providing a comprehensive approach to tumor detection. In summary, the motivation for utilizing deep learning in brain tumor detection stems from its potential to enhance accuracy, efficiency, and scalability compared to traditional manual methods.

Chapter 2

Literature Review

2.1 Review of existing techniques for brain tumor detection

The use of deep learning for brain tumor detection has been gaining popularity in recent years due to the potential for higher accuracy and better detection of smaller tumors. Deep learning is a type of artificial intelligence that uses complex neural networks to process large amounts of data and is particularly effective at image recognition tasks. It has been used in medical imaging to detect a variety of diseases, including brain tumors.

In a 2018 study, a deep learning system was used to detect gliomas, the most common type of brain tumor. The study used magnetic resonance imaging (MRI) scans of the brain to train a convolutional neural network (CNN) to detect gliomas. The system was able to detect gliomas with an accuracy of 91.2%, which was higher than the accuracy achieved by human experts.

In a 2019 study, a deep learning system was used to detect meningiomas, a type of benign brain tumor. The study used MRI scans of the brain to train a CNN to detect meningiomas. The system was able to detect meningiomas with an accuracy of 90.4%, which was higher than the accuracy achieved by human experts. In a 2020 study, a deep learning system was used to detect low-grade.

2.2 Comparison of different deep learning architectures for brain tumor detection

There are several deep learning architectures that have been used for brain tumor detection in medical imaging. Here's a comparison of some popular architectures:

Convolutional Neural Networks (CNNs):

CNNs are widely used for image classification tasks and have been successfully applied to brain tumor detection.

They consist of multiple convolutional layers followed by pooling layers, fully connected layers, and an output layer.

CNNs are effective at automatically learning hierarchical features from images, which is beneficial for tumor detection.

Artificial Neural Networks (ANNs):

ANNs are the fundamental building blocks of deep learning models, including CNNs and RNNs.

In the context of brain tumor detection, ANNs refer to shallow networks without convolutional or recurrent layers.

ANNs consist of interconnected layers of artificial neurons, where each neuron receives inputs, applies an activation function, and passes the output to the next layer.

They can be trained using backpropagation algorithms to learn the optimal weights and biases for the given task.

ANNs are generally simpler than deep learning architectures but can still be effective for certain types of brain tumor detection tasks, especially when the dataset is small or the complexity of the task is relatively low.

Reference Websites:

www.w3schools.com/html
www.stackoverflow.com
https://www.kaggle.com/
www.Github.com/
https://unacademy.com/courses
https://www.python.org/

Chapter 3

Methodology

3.1 Pre-processing:

Pre-processing is the first step in the brain tumor detection using deep learning. It involves cleaning the data and removing any noise from the medical images. This step is necessary to ensure that the deep learning model is able to accurately detect the tumor.

3.2 Data Augmentation:

Data augmentation is the process of creating more data by applying various transformations like cropping, flipping, rotating, etc. to the existing data. This is done to increase the model's ability to generalize.

3.3 Feature Extraction:

Feature extraction involves extracting the relevant features from the images that can help the deep learning model to detect the tumor. This is usually done using convolutional neural networks.

3.4 Model Training:

Once the features have been extracted from the images, it's time to train the deep learning model. This is done using a wide variety of techniques such as supervised learning, unsupervised learning, and transfer learning.

3.5 Evaluation:

After the model has been trained, it's time to evaluate it. This is done by testing the model on a test dataset and calculating various metrics such as accuracy, precision, recall, and F1 score.

3.6 Deployment:

The model can then be deployed to detect brain tumors in real-time. This can be done using a web or mobile application.

Chapter 4

Deep Learning Model Development

4.1 Selection of the deep learning architecture (e.g., Convolutional Neural Network and Artificial Neural Network)

The selection of the deep learning architecture, whether it's a Convolutional Neural Network (CNN) or an Artificial Neural Network (ANN), depends on several factors. Here are some considerations to help guide the decision-making process:

Task Requirements: Consider the specific requirements of the brain tumor detection task. Is it an image classification task, where the goal is to classify images as either tumor or non-tumor? Or is it a segmentation task, where the objective is to identify and delineate the tumor region within the image? CNNs are particularly well-suited for image classification and segmentation tasks due to their ability to capture spatial information and hierarchical features. On the other hand, ANNs can be more suitable for simpler tasks or when interpretability is a priority.

Input Data: Consider the nature of the input data. If the brain tumor detection task involves analyzing medical images, such as MRI scans or CT scans, CNNs are often the architecture of choice. CNNs have built-in mechanisms to learn and extract features from image data, making them effective at capturing spatial relationships and detecting patterns. ANNs, on the other hand, are more commonly used when the input data is in a structured format, such as tabular data or feature vectors.

Dataset Size: The size of the available dataset is an important consideration. Deep learning architectures with a large number of parameters, such as CNNs, tend to require more data to effectively learn and generalize. If the dataset is limited, an ANN with a smaller number of parameters may be more appropriate, as it can be trained more efficiently with fewer data points. ANNs can still provide satisfactory results when the dataset size is relatively small.

Computational Resources: Consider the available computational resources. CNNs are typically more computationally intensive than ANNs due to their deeper and more complex architectures. Training CNNs may require powerful GPUs or specialized hardware. If computational resources are limited, ANNs can be a more practical choice, as they have a simpler structure and can be trained on standard hardware.

Previous Research and Benchmarking: Examine existing literature and research to identify successful architectures used in similar brain tumor detection tasks. This can provide insights into which architectures have achieved promising results and can serve as a starting point for your own experimentation. Benchmarking different architectures on your dataset can help determine which one performs better in your specific scenario.

Ultimately, the selection of the deep learning architecture should be based on a thorough understanding of the task requirements, the nature of the data, available resources, and previous research. The next step is to select the right model for the task. This can involve choosing between different types of neural networks such as convolutional neural networks (CNNs) or Artificial neural networks (ANNs). It is important to select the right model for the task as it will determine how accurately the model can detect brain tumors.

4.2 Software Model And Software Requirements

Software Model:

Software model on brain tumor detection using deep learning project

The goal of this project is to develop a software model that can accurately detect the presence of brain tumors in MRI or CT scans. This model will use deep learning algorithms to train on a large dataset of medical images and accurately detect the presence of brain tumors. The model will be trained on a variety of input data including MRI and CT scans. It will also be trained to distinguish between different types of brain tumors.

The software model will be implemented using the popular machine learning library, TensorFlow. The model will take advantage of convolutional neural networks (CNNs) to detect the presence of brain tumors. The CNNs will be trained on a dataset of medical images that contain both benign and malignant brain tumors. The model will use transfer learning to

learn from existing models, and use data augmentation for generalization and to increase accuracy.

Once the model is trained, it will be tested using a separate test dataset. The model will be evaluated on its accuracy, precision, recall, and F1 score. If the model performs well, it will be deployed in a production environment. This model can then be used to detect brain tumors in a medical setting or for research purposes.

This project has the potential to make a huge impact.

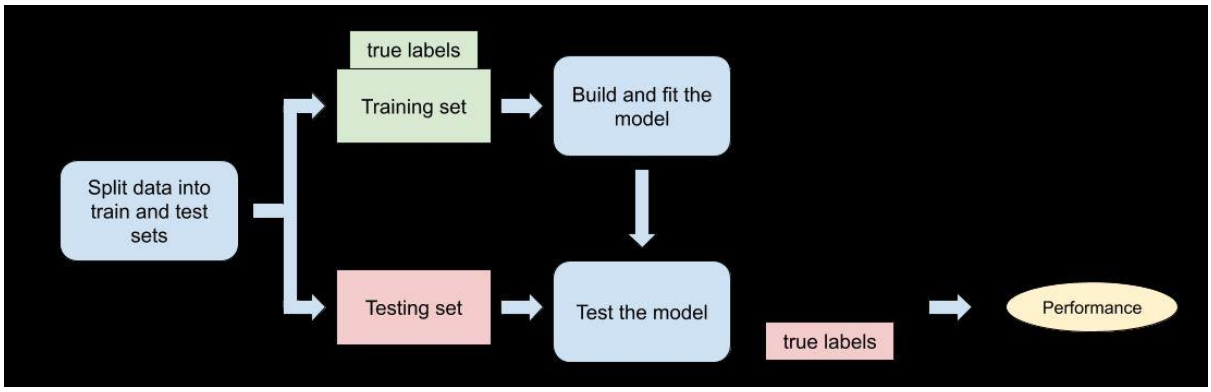
Software Requirments & Hardware Requirments:

Processor - Intel i5-7600k/Ryzen 3 3300X
Hard Disk - 500Gb
Ram - 8Gb
A GPU with CUDA-enabled
An Internet connection
Operating System - Windows 10/11
Compiler - Python
Software Used - PyCharm/Jupyter Notebook(Anaconda)/Visual Studio Code
Python Libraries Used -

- TensorFlow
- Keras
- OpenCV
- scikit-learn
- Jupyter Notebook
- os
- os.path
- pandas
- pytz
- numpy
- image

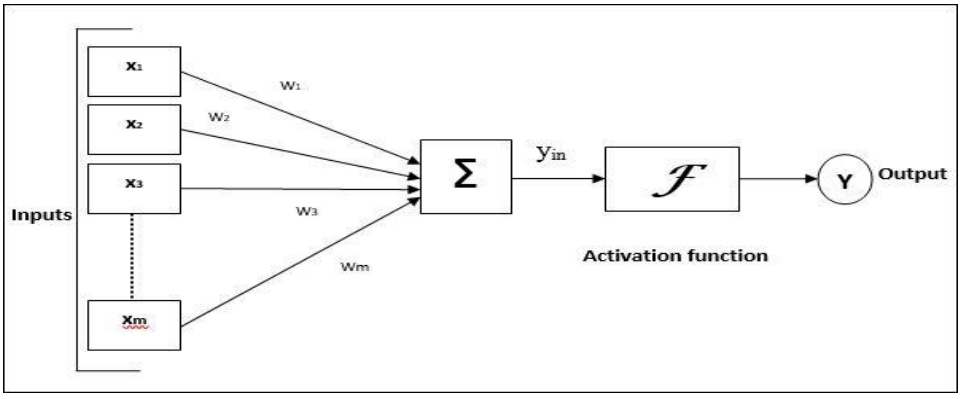
4.3 Data Flow Diagram

CNN Model:



1.1

ANN Model:



1.2

Chapter 5

Sample Code with Outputs

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from warnings import filterwarnings
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, roc_auc_score, roc_curve
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization,MaxPooling2D
from keras import models
from keras import layers
import tensorflow as tf
import os
import os.path
from pathlib import Path
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
from keras import regularizers
from keras.optimizers import RMSprop,Adam
import glob
from PIL import Image
filterwarnings("ignore",category=DeprecationWarning)
filterwarnings("ignore", category=FutureWarning)
filterwarnings("ignore", category=UserWarning)
```

In [2]:
```
No_Data_Path = Path(r"C:\Users\Sohon\Downloads\Braintumer\no")
Yes_Data_Path = Path(r"C:\Users\Sohon\Downloads\Braintumer\yes")
```

In [3]:
```
No_JPG_Path = list(No_Data_Path.glob(r"*.jpg"))
Yes_JPG_Path = list(Yes_Data_Path.glob(r"*.jpg"))
```

In [4]:
```
print(No_JPG_Path[0:5])
print("-----"*20)
print(Yes_JPG_Path[0:5])
```

```
[WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no0.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no10.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no100.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1000.jpg')]
----------------------------------------------------------------------------------------------------
[WindowsPath('C:/Users/Sohon/Downloads/Braintumer/yes/y0.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/yes/y1.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/yes/y10.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/yes/y100.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/yes/y1000.jpg')]
```

In [5]:
```
Yes_No_List = []

for No_JPG in No_JPG_Path:
    Yes_No_List.append(No_JPG)

for Yes_JPG in Yes_JPG_Path:
    Yes_No_List.append(Yes_JPG)
```

In [6]:
```
print(Yes_No_List[0:10])
```

```
[WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no0.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no10.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no100.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1000.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1001.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1002.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1003.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1004.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/no/no1005.jpg')]
```

In [7]:
```
JPG_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Yes_No_List))
```

In [8]:
```
print(JPG_Labels[0:10])
```

```
['no', 'no', 'no', 'no', 'no', 'no', 'no', 'no', 'no', 'no']
```

In [9]:
```
print("NO COUNTING: ", JPG_Labels.count("no"))
print("YES COUNTING: ", JPG_Labels.count("yes"))
```

```
NO COUNTING:  1500
YES COUNTING:  1500
```

In [10]:
```
JPG_Path_Series = pd.Series(Yes_No_List,name="JPG").astype(str)
JPG_Category_Series = pd.Series(JPG_Labels,name="TUMOR_CATEGORY")
```

In [11]:
```
Main_Train_Data = pd.concat([JPG_Path_Series,JPG_Category_Series],axis=1)
```

In [12]:
```
print(Main_Train_Data.head(-1))
```

```
                                        JPG TUMOR_CATEGORY
0       C:\Users\Sohon\Downloads\Braintumer\no\no0.jpg            no
1       C:\Users\Sohon\Downloads\Braintumer\no\no1.jpg            no
2      C:\Users\Sohon\Downloads\Braintumer\no\no10.jpg            no
3     C:\Users\Sohon\Downloads\Braintumer\no\no100.jpg            no
4    C:\Users\Sohon\Downloads\Braintumer\no\no1000.jpg            no
...                                               ...           ...
2994  C:\Users\Sohon\Downloads\Braintumer\yes\y994.jpg           yes
2995  C:\Users\Sohon\Downloads\Braintumer\yes\y995.jpg           yes
2996  C:\Users\Sohon\Downloads\Braintumer\yes\y996.jpg           yes
2997  C:\Users\Sohon\Downloads\Braintumer\yes\y997.jpg           yes
2998  C:\Users\Sohon\Downloads\Braintumer\yes\y998.jpg           yes

[2999 rows x 2 columns]
```

In [13]:
```
Prediction_Path = Path(r"C:\Users\Sohon\Downloads\Braintumer\pred")
```

In [14]:
```
Test_JPG_Path = list(Prediction_Path.glob(r"*.jpg"))
```

In [15]:
```
print(Test_JPG_Path[0:5])
```

```
[WindowsPath('C:/Users/Sohon/Downloads/Braintumer/pred/pred0.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/pred/pred1.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/pred/pred10.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/pred/pred11.jpg'), WindowsPath('C:/Users/Sohon/Downloads/Braintumer/pred/pred12.jpg')]
```

In [16]:
```
Test_JPG_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Test_JPG_Path))
```

In [17]:
```
print(Test_JPG_Labels[0:5])
```

```
['pred', 'pred', 'pred', 'pred', 'pred']
```

In [18]:
```
Test_JPG_Path_Series = pd.Series(Test_JPG_Path,name="JPG").astype(str)
Test_JPG_Labels_Series = pd.Series(Test_JPG_Labels,name="TUMOR_CATEGORY")
```

In [19]:
```
Test_Data = pd.concat([Test_JPG_Path_Series,Test_JPG_Labels_Series],axis=1)
```

In [20]:
```
print(Test_Data.head())
```

```
                                                 JPG TUMOR_CATEGORY
0  C:\Users\Sohon\Downloads\Braintumer\pred\pred0...           pred
1  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...           pred
2  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...           pred
3  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...           pred
4  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...           pred
```

In [21]:
```
Main_Train_Data = Main_Train_Data.sample(frac=1).reset_index(drop=True)
```

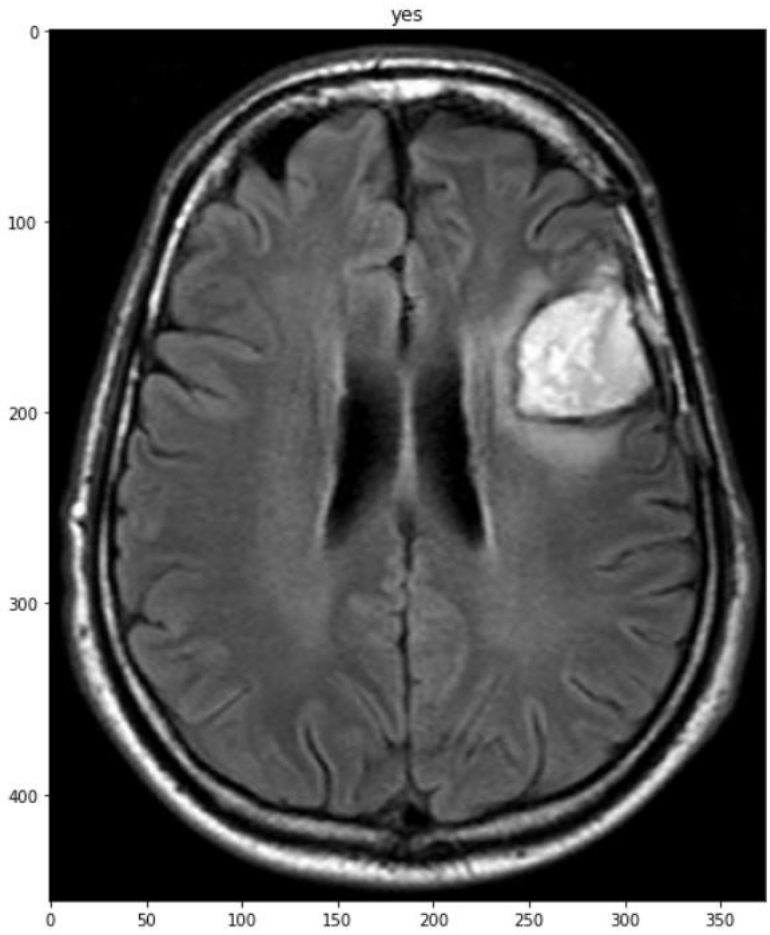
In [22]:
```
print(Main_Train_Data.head(-1))
```

```
                                        JPG TUMOR_CATEGORY
0     C:\Users\Sohon\Downloads\Braintumer\no\no1275.jpg            no
1      C:\Users\Sohon\Downloads\Braintumer\no\no423.jpg            no
2      C:\Users\Sohon\Downloads\Braintumer\no\no348.jpg            no
3     C:\Users\Sohon\Downloads\Braintumer\yes\y1387.jpg           yes
4      C:\Users\Sohon\Downloads\Braintumer\yes\y913.jpg           yes
...                                               ...           ...
2994  C:\Users\Sohon\Downloads\Braintumer\yes\y1263.jpg           yes
2995  C:\Users\Sohon\Downloads\Braintumer\yes\y1232.jpg           yes
2996   C:\Users\Sohon\Downloads\Braintumer\yes\y163.jpg           yes
2997  C:\Users\Sohon\Downloads\Braintumer\yes\y1136.jpg           yes
2998   C:\Users\Sohon\Downloads\Braintumer\yes\y803.jpg           yes

[2999 rows x 2 columns]
```

In [23]:

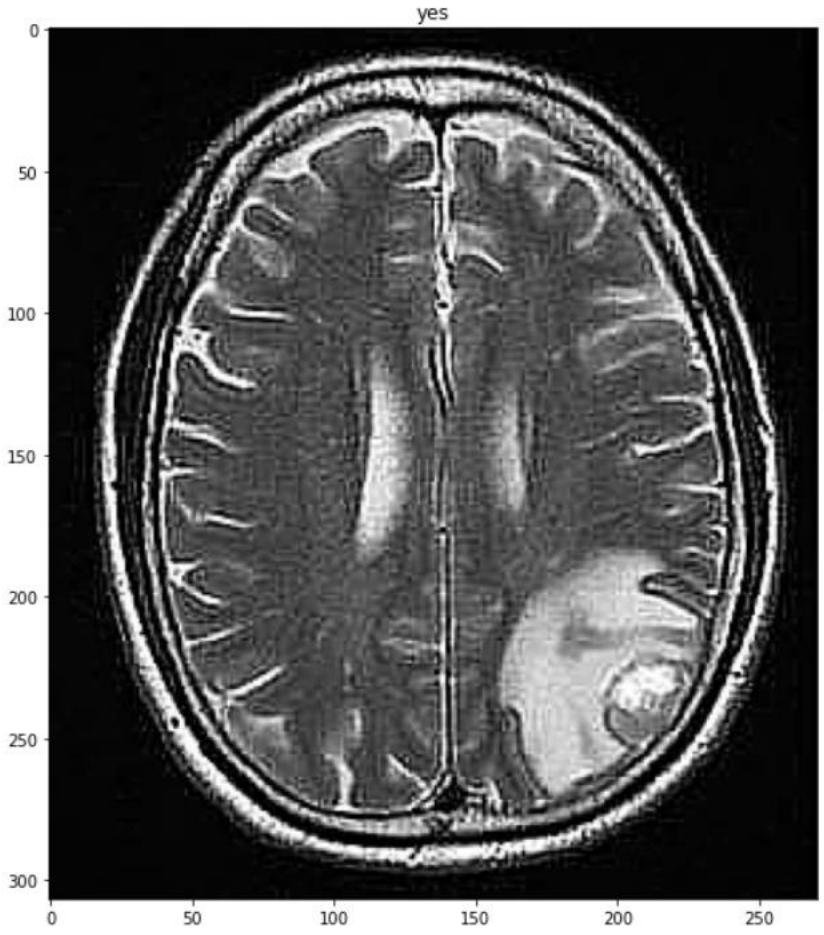
```
figure = plt.figure(figsize=(10,10))
plt.imshow(plt.imread(Main_Train_Data["JPG"][10]))
plt.title(Main_Train_Data["TUMOR_CATEGORY"][10])
```

Out[23]: Text(0.5, 1.0, 'yes')

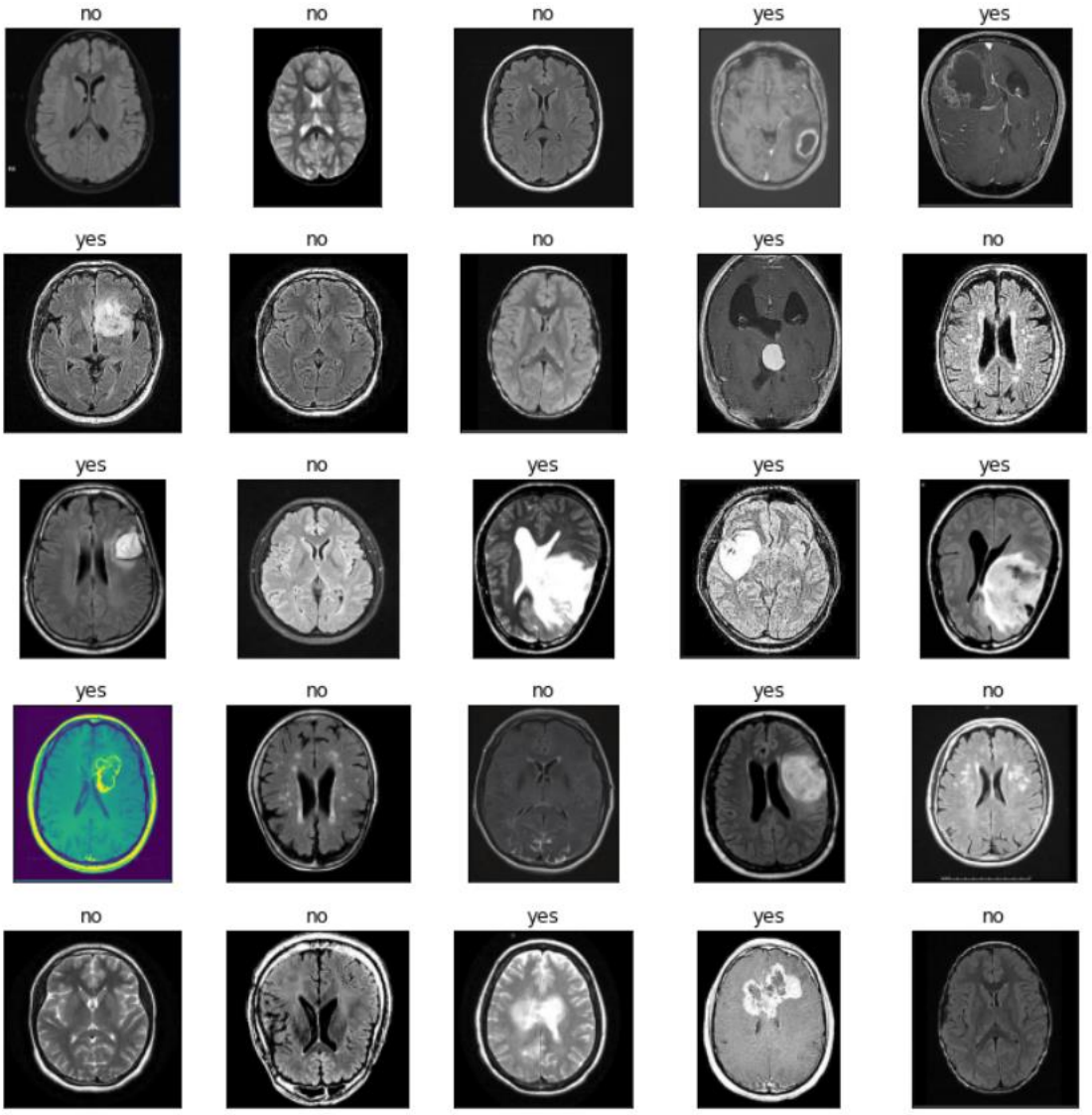


In [24]:
```
figure = plt.figure(figsize=(10,10))
plt.imshow(plt.imread(Main_Train_Data["JPG"][2997]))
plt.title(Main_Train_Data["TUMOR_CATEGORY"][2997])
```

Out[24]: Text(0.5, 1.0, 'yes')



In [25]:
```
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(10, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(Main_Train_Data["JPG"][i]))
    ax.set_title(Main_Train_Data["TUMOR_CATEGORY"][i])
plt.tight_layout()
plt.show()
```


no
no
no
yes
yes
yes
no
no
yes
no
yes
no
yes
yes
yes
yes
no
no
yes
no
no
yes
yes
no

In [26]:
```
train_data,test_data = train_test_split(Main_Train_Data,train_size=0.9,random_state=42)
```

In [27]:
```
print(train_data.shape)
```

```
(2700, 2)
```

In [28]:
```
print(train_data.head())
```

```
                                                    JPG TUMOR_CATEGORY
433   C:\Users\Sohon\Downloads\Braintumer\no\no1108.jpg             no
1151   C:\Users\Sohon\Downloads\Braintumer\yes\y165.jpg            yes
73     C:\Users\Sohon\Downloads\Braintumer\yes\y431.jpg            yes
1536    C:\Users\Sohon\Downloads\Braintumer\yes\y28.jpg            yes
2709  C:\Users\Sohon\Downloads\Braintumer\no\no1267.jpg             no
```

In [29]:
```
print(test_data.shape)
```

```
(300, 2)
```

In [30]:
```
print(test_data.head())
```

```
                                                    JPG TUMOR_CATEGORY
1801   C:\Users\Sohon\Downloads\Braintumer\yes\y966.jpg            yes
1190  C:\Users\Sohon\Downloads\Braintumer\no\no1485.jpg             no
1817   C:\Users\Sohon\Downloads\Braintumer\no\no818.jpg             no
251    C:\Users\Sohon\Downloads\Braintumer\no\no814.jpg             no
2505   C:\Users\Sohon\Downloads\Braintumer\no\no691.jpg             no
```

In [31]:
```
Generator_Basic = ImageDataGenerator(rescale=1./255,
                                     validation_split=0.1)
```

In [32]:
```
Train_Set = Generator_Basic.flow_from_dataframe(dataframe=train_data,
                                                x_col="JPG",
                                                y_col="TUMOR_CATEGORY",
                                                color_mode="grayscale",
                                                class_mode="categorical",
                                                subset="training",
                                                batch_size=20,
                                                target_size=(200,200))
```

```
Found 2430 validated image filenames belonging to 2 classes.
```

In [33]:
```
Validation_Set = Generator_Basic.flow_from_dataframe(dataframe=train_data,
                                                     x_col="JPG",
                                                     y_col="TUMOR_CATEGORY",
                                                     color_mode="grayscale",
                                                     class_mode="categorical",
                                                     subset="validation",
                                                     batch_size=20,
                                                     target_size=(200,200))
```

```
Found 270 validated image filenames belonging to 2 classes.
```

In [34]:
```
Test_Set = Generator_Basic.flow_from_dataframe(dataframe=test_data,
                                               x_col="JPG",
                                               y_col="TUMOR_CATEGORY",
                                               color_mode="grayscale",
                                               class_mode="categorical",
                                               batch_size=20,
                                               target_size=(200,200))
```

```
Found 300 validated image filenames belonging to 2 classes.
```

In [35]:
```
for data_batch,label_batch in Train_Set:
    print("DATA SHAPE: ",data_batch.shape)
    print("LABEL SHAPE: ",label_batch.shape)
    break
```

```
DATA SHAPE:  (20, 200, 200, 1)
LABEL SHAPE:  (20, 2)
```

In [36]:
```
for data_batch,label_batch in Validation_Set:
    print("DATA SHAPE: ",data_batch.shape)
    print("LABEL SHAPE: ",label_batch.shape)
    break
```

```
DATA SHAPE:  (20, 200, 200, 1)
LABEL SHAPE:  (20, 2)
```

In [37]:
```
for data_batch,label_batch in Test_Set:
    print("DATA SHAPE: ",data_batch.shape)
    print("LABEL SHAPE: ",label_batch.shape)
    break
```

```
DATA SHAPE:  (20, 200, 200, 1)
LABEL SHAPE:  (20, 2)
```

In [38]:
```
print(Train_Set.class_indices)
print(Train_Set.classes[0:5])
print(Train_Set.image_shape)
```

```
{'no': 0, 'yes': 1}
[0, 0, 1, 1, 1]
(200, 200, 1)
```

In [39]:
```
print(Validation_Set.class_indices)
print(Validation_Set.classes[0:5])
print(Validation_Set.image_shape)
```

```
{'no': 0, 'yes': 1}
[0, 1, 1, 1, 0]
(200, 200, 1)
```

In [40]:
```
print(Test_Set.class_indices)
print(Test_Set.classes[0:5])
print(Test_Set.image_shape)
```

```
{'no': 0, 'yes': 1}
[1, 0, 0, 0, 0]
(200, 200, 1)
```

In [41]:

```
Model = Sequential()

Model.add(Conv2D(32,(5,5),activation="relu",input_shape=(200,200,1)))
Model.add(MaxPool2D((2,2)))
Model.add(Dropout(0.2))
#
Model.add(Conv2D(64,(3,3),activation="relu"))
Model.add(MaxPool2D((2,2)))
Model.add(Dropout(0.2))
#
Model.add(Conv2D(128,(3,3),activation="relu"))
Model.add(MaxPool2D((2,2)))
Model.add(Dropout(0.2))
#
Model.add(Conv2D(256,(3,3),activation="relu"))
Model.add(MaxPool2D((2,2)))
Model.add(Dropout(0.2))
#
Model.add(Flatten())
Model.add(Dropout(0.5))
Model.add(Dense(512,activation="relu"))
Model.add(Dense(2,activation="softmax"))
```

In [42]:

```
Model.compile(optimizer=RMSprop(lr=0.001),loss="categorical_crossentropy",metrics=["accuracy"])
```

In [43]:

```
ANN_Model = Model.fit(Train_Set,validation_data=Validation_Set,
                      epochs=30,steps_per_epoch=120)
```

```
Epoch 1/30
120/120 [==============================] - 152s 1s/step - loss: 0.8431 - accuracy: 0.6946 - val_loss: 0.4358 - val_accuracy: 0.8444
Epoch 2/30
120/120 [==============================] - 152s 1s/step - loss: 0.4222 - accuracy: 0.8188 - val_loss: 0.3464 - val_accuracy: 0.8778
Epoch 3/30
120/120 [==============================] - 171s 1s/step - loss: 0.3295 - accuracy: 0.8632 - val_loss: 0.2626 - val_accuracy: 0.8815
Epoch 4/30
120/120 [==============================] - 267s 2s/step - loss: 0.2471 - accuracy: 0.9046 - val_loss: 0.1643 - val_accuracy: 0.9519
Epoch 5/30
120/120 [==============================] - 164s 1s/step - loss: 0.2035 - accuracy: 0.9218 - val_loss: 0.2053 - val_accuracy: 0.9222
Epoch 6/30
120/120 [==============================] - 146s 1s/step - loss: 0.1775 - accuracy: 0.9356 - val_loss: 0.1535 - val_accuracy: 0.9556
Epoch 7/30
120/120 [==============================] - 193s 2s/step - loss: 0.1578 - accuracy: 0.9469 - val_loss: 0.1971 - val_accuracy: 0.9111
Epoch 8/30
120/120 [==============================] - 162s 1s/step - loss: 0.1371 - accuracy: 0.9523 - val_loss: 0.0767 - val_accuracy: 0.9741
Epoch 9/30
120/120 [==============================] - 147s 1s/step - loss: 0.1130 - accuracy: 0.9619 - val_loss: 0.0861 - val_accuracy: 0.9667
Epoch 10/30
120/120 [==============================] - 138s 1s/step - loss: 0.1097 - accuracy: 0.9661 - val_loss: 0.0690 - val_accuracy: 0.9704
Epoch 11/30
120/120 [==============================] - 274s 2s/step - loss: 0.0956 - accuracy: 0.9657 - val_loss: 0.0831 - val_accuracy: 0.9556
Epoch 12/30
120/120 [==============================] - 160s 1s/step - loss: 0.1005 - accuracy: 0.9678 - val_loss: 0.0859 - val_accuracy: 0.9630
Epoch 13/30
120/120 [==============================] - 141s 1s/step - loss: 0.0883 - accuracy: 0.9690 - val_loss: 0.0885 - val_accuracy: 0.9704
Epoch 14/30
120/120 [==============================] - 148s 1s/step - loss: 0.0824 - accuracy: 0.9745 - val_loss: 0.0570 - val_accuracy: 0.9741
Epoch 15/30
120/120 [==============================] - 169s 1s/step - loss: 0.0814 - accuracy: 0.9766 - val_loss: 0.0736 - val_accuracy: 0.9704
Epoch 16/30
120/120 [==============================] - 221s 2s/step - loss: 0.0713 - accuracy: 0.9816 - val_loss: 0.0391 - val_accuracy: 0.9852
Epoch 17/30
120/120 [==============================] - 322s 3s/step - loss: 0.0674 - accuracy: 0.9782 - val_loss: 0.0691 - val_accuracy: 0.9704
Epoch 18/30
```

```
120/120 [==============================] - 322s 3s/step - loss: 0.0674 - accuracy: 0.9782 - val_loss: 0.0691 - val_accuracy: 0.9704
Epoch 18/30
120/120 [==============================] - 321s 3s/step - loss: 0.0652 - accuracy: 0.9808 - val_loss: 0.0542 - val_accuracy: 0.9704
Epoch 19/30
120/120 [==============================] - 333s 3s/step - loss: 0.0676 - accuracy: 0.9782 - val_loss: 0.0557 - val_accuracy: 0.9815
Epoch 20/30
120/120 [==============================] - 309s 3s/step - loss: 0.0538 - accuracy: 0.9812 - val_loss: 0.0793 - val_accuracy: 0.9852
Epoch 21/30
120/120 [==============================] - 336s 3s/step - loss: 0.0603 - accuracy: 0.9845 - val_loss: 0.0344 - val_accuracy: 0.9852
Epoch 22/30
120/120 [==============================] - 177s 1s/step - loss: 0.0661 - accuracy: 0.9837 - val_loss: 0.0295 - val_accuracy: 0.9815
Epoch 23/30
120/120 [==============================] - 221s 2s/step - loss: 0.0458 - accuracy: 0.9837 - val_loss: 0.1152 - val_accuracy: 0.9741
Epoch 24/30
120/120 [==============================] - 343s 3s/step - loss: 0.0488 - accuracy: 0.9874 - val_loss: 0.0400 - val_accuracy: 0.9778
Epoch 25/30
120/120 [==============================] - 239s 2s/step - loss: 0.0462 - accuracy: 0.9866 - val_loss: 0.0355 - val_accuracy: 0.9852
Epoch 26/30
120/120 [==============================] - 165s 1s/step - loss: 0.0683 - accuracy: 0.9791 - val_loss: 0.0377 - val_accuracy: 0.9815
Epoch 27/30
120/120 [==============================] - 167s 1s/step - loss: 0.0398 - accuracy: 0.9904 - val_loss: 0.0656 - val_accuracy: 0.9815
Epoch 28/30
120/120 [==============================] - 157s 1s/step - loss: 0.0493 - accuracy: 0.9870 - val_loss: 0.0359 - val_accuracy: 0.9889
Epoch 29/30
120/120 [==============================] - 136s 1s/step - loss: 0.0422 - accuracy: 0.9895 - val_loss: 0.0893 - val_accuracy: 0.9704
Epoch 30/30
120/120 [==============================] - 131s 1s/step - loss: 0.0476 - accuracy: 0.9870 - val_loss: 0.0539 - val_accuracy: 0.9667
```

In [44]:

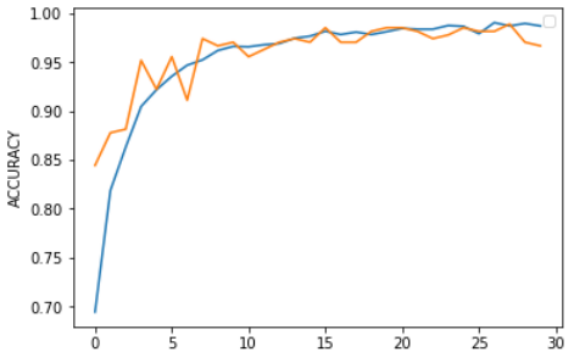
```
print(Model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 196, 196, 32)      832

 max_pooling2d (MaxPooling2D  (None, 98, 98, 32)       0
 )

 dropout (Dropout)           (None, 98, 98, 32)        0

 conv2d_1 (Conv2D)           (None, 96, 96, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 48, 48, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 48, 48, 64)        0

 conv2d_2 (Conv2D)           (None, 46, 46, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 23, 23, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 23, 23, 128)       0

 conv2d_3 (Conv2D)           (None, 21, 21, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 256)      0
 2D)

 dropout_3 (Dropout)         (None, 10, 10, 256)       0

 flatten (Flatten)           (None, 25600)             0

 dropout_4 (Dropout)         (None, 25600)             0

 dense (Dense)               (None, 512)               13107712

 dense_1 (Dense)             (None, 2)                 1026

=================================================================
Total params: 13,497,090
Trainable params: 13,497,090
Non-trainable params: 0
_________________________________________________________________
None
```

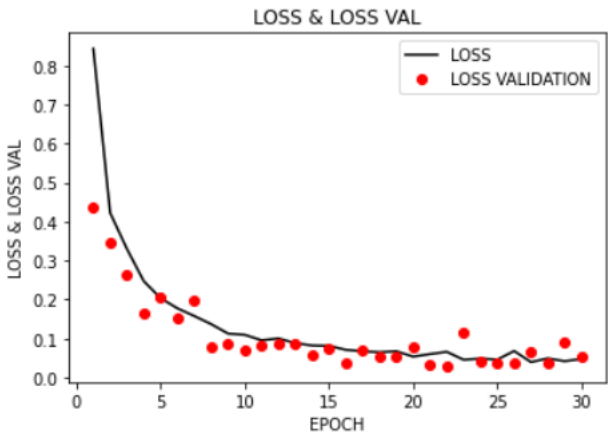
In [45]:
```
HistoryDict = ANN_Model.history

val_losses = HistoryDict["val_loss"]
val_acc = HistoryDict["val_accuracy"]
acc = HistoryDict["accuracy"]
losses = HistoryDict["loss"]
epochs = range(1,len(val_losses)+1)
```

In [46]:
```
plt.plot(ANN_Model.history["accuracy"])
plt.plot(ANN_Model.history["val_accuracy"])
plt.ylabel("ACCURACY")
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```

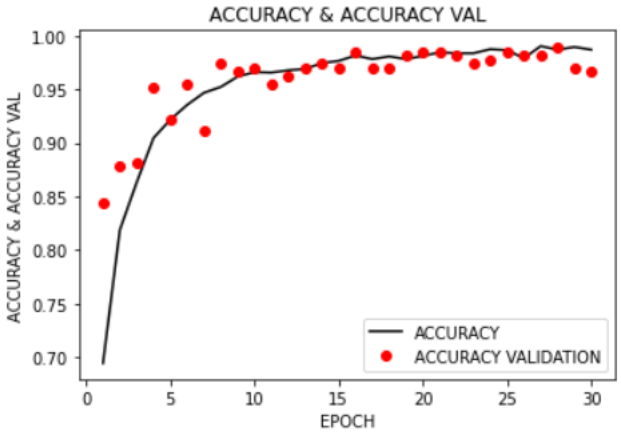


In [47]:
```
plt.plot(epochs,losses,"k-",label="LOSS")
plt.plot(epochs,val_losses,"ro",label="LOSS VALIDATION")
plt.title("LOSS & LOSS VAL")
plt.xlabel("EPOCH")
plt.ylabel("LOSS & LOSS VAL")
plt.legend()
plt.show()
```



In [48]:

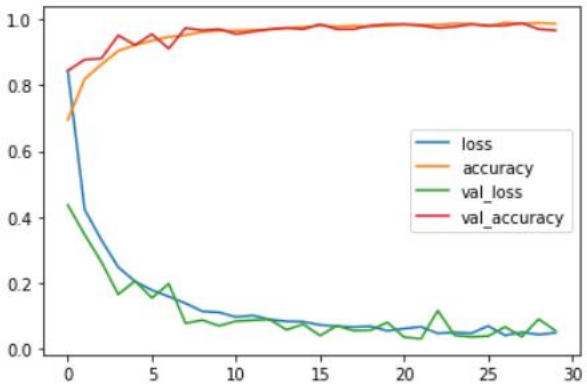
```
plt.plot(epochs,acc,"k-",label="ACCURACY")
plt.plot(epochs,val_acc,"ro",label="ACCURACY VALIDATION")
plt.title("ACCURACY & ACCURACY VAL")
plt.xlabel("EPOCH")
plt.ylabel("ACCURACY & ACCURACY VAL")
plt.legend()
plt.show()
```



In [49]:

```
Dict_Summary = pd.DataFrame(ANN_Model.history)
Dict_Summary.plot()
```

Out[49]: <AxesSubplot:>



In [50]:

```
Model_Results = Model.evaluate(Test_Set,verbose=False)
print("LOSS:  " + "%.4f" % Model_Results[0])
print("ACCURACY:  " + "%.2f" % Model_Results[1])
```

```
LOSS:  0.1120
ACCURACY:  0.98
```

In [51]:
```
Main_Data_Prediction = pd.DataFrame({"JPG":Test_JPG_Path_Series})
```

In [52]:
```
print(Main_Data_Prediction.head())
```

```
                                                 JPG
0  C:\Users\Sohon\Downloads\Braintumer\pred\pred0...
1  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...
2  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...
3  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...
4  C:\Users\Sohon\Downloads\Braintumer\pred\pred1...
```

In [53]:
```
Main_Test_Generator = ImageDataGenerator(rescale=1./255)
```

In [54]:
```
Main_Test_Set = Main_Test_Generator.flow_from_dataframe(dataframe=Main_Data_Prediction,
                                                        x_col="JPG",
                                                        y_col=None,
                                                        color_mode="grayscale",
                                                         class_mode=None,
                                                         batch_size=20,
                                                         target_size=(200,200))
```

```
Found 60 validated image filenames.
```

In [55]:
```
Model_Test_Prediction = Model.predict(Main_Test_Set)
```

```
3/3 [==============================] - 1s 218ms/step
```

In [56]:
```
Model_Test_Prediction = Model_Test_Prediction.argmax(axis=-1)
```

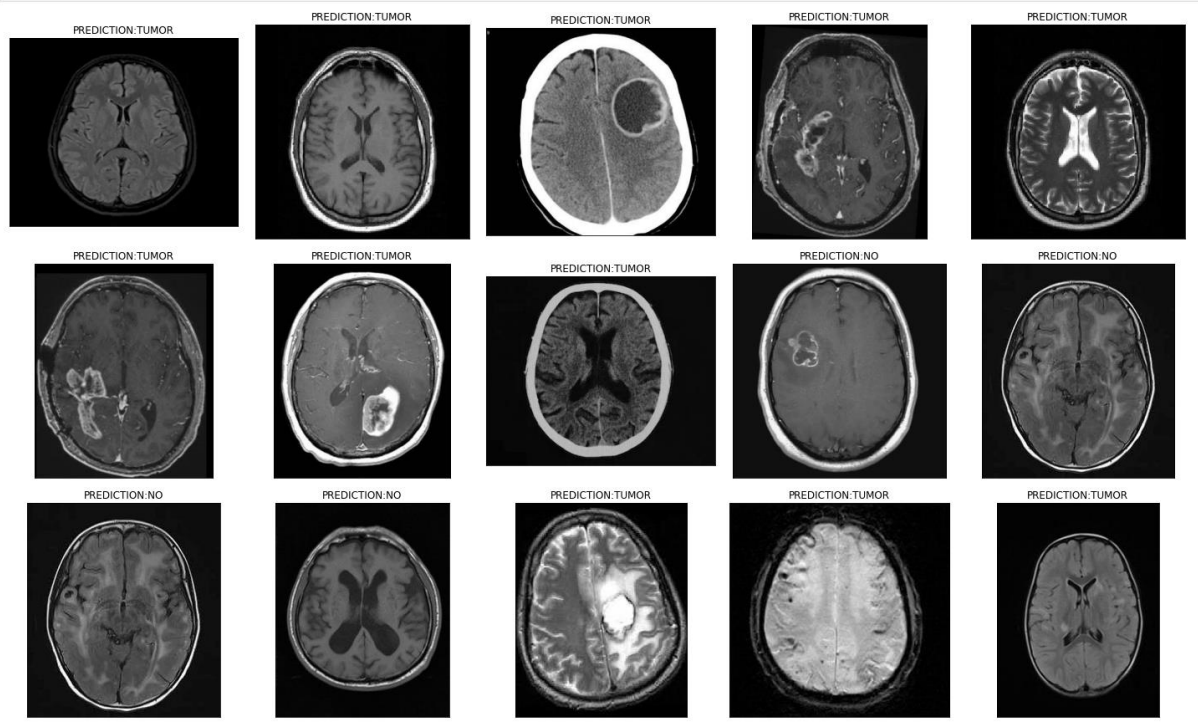
In [57]:
```
print(Model_Test_Prediction)
```

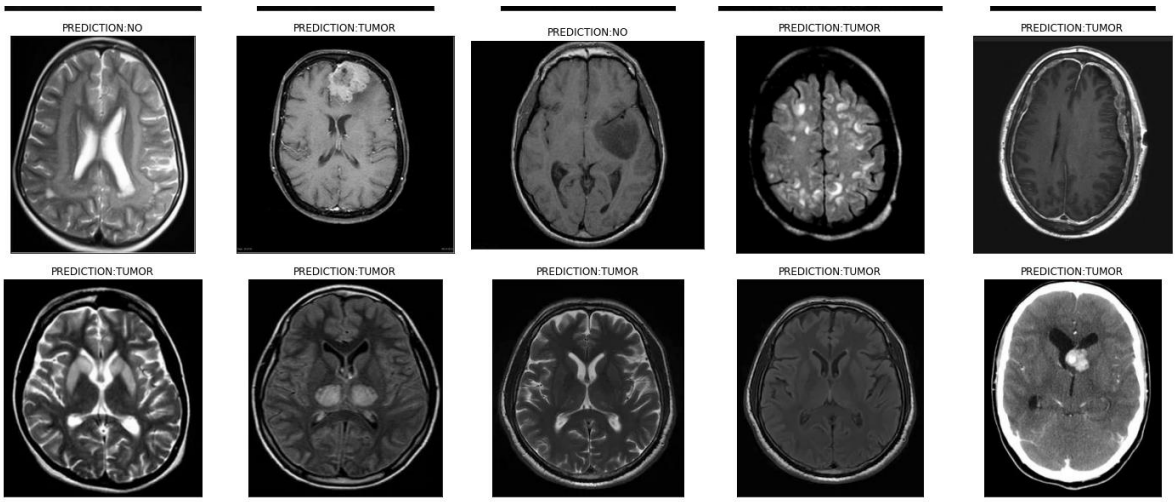
```
[0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 1 0 1
 1 1 0 1 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0]
```

In [58]:
```
Last_Prediction = []
[Last_Prediction.append('NO') if i == 1 else Last_Prediction.append('TUMOR') for i in Model_Test_Prediction]
print(Last_Prediction)
```

```
['TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'NO', 'NO', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'NO', 'NO', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR']
```

In [59]:
```
fig, axes = plt.subplots(nrows=5,
                         ncols=5,
                         figsize=(20, 20),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(Main_Data_Prediction["JPG"].loc[i]))
    ax.set_title(f"PREDICTION:{Last_Prediction[i]}")
plt.tight_layout()
plt.show()
```





In [60]:
```
Data_Generator_Div = ImageDataGenerator(rescale=1./255,brightness_range=[0.3,0.9],
                                        rotation_range=30,
                                        width_shift_range=0.2,
                                        height_shift_range=0.2,
                                        shear_range=0.2,
                                        horizontal_flip=True,
                                        fill_mode="nearest",
                                        validation_split=0.1)
```

In [61]:
```
Train_Set_Div = Data_Generator_Div.flow_from_dataframe(dataframe=train_data,
                                                       x_col="JPG",
                                                       y_col="TUMOR_CATEGORY",
                                                       color_mode="grayscale",
                                                       class_mode="categorical",
                                                       subset="training",
                                                       batch_size=20,
                                                       target_size=(200,200))
```

```
Found 2430 validated image filenames belonging to 2 classes.
```

In [62]:
```
Validation_Set_Div = Data_Generator_Div.flow_from_dataframe(dataframe=train_data,
                                                            x_col="JPG",
                                                            y_col="TUMOR_CATEGORY",
                                                            color_mode="grayscale",
                                                            class_mode="categorical",
                                                            subset="validation",
                                                            batch_size=20,
                                                            target_size=(200,200))
```

```
Found 270 validated image filenames belonging to 2 classes.
```

In [63]:
```
Test_Set_Div = Data_Generator_Div.flow_from_dataframe(dataframe=Main_Data_Prediction,
                                                      x_col="JPG",
                                                      y_col=None,
                                                      color_mode="grayscale",
                                                       class_mode=None,
                                                       batch_size=20,
                                                       target_size=(200,200))
```

```
Found 60 validated image filenames.
```

In [64]:
```
Model_Two = Sequential()

Model_Two.add(Conv2D(32,(5,5),activation="relu",input_shape=(200,200,1)))
Model_Two.add(MaxPool2D((2,2)))
#
Model_Two.add(Conv2D(64,(3,3),activation="relu"))
Model_Two.add(MaxPool2D((2,2)))
#
Model_Two.add(Conv2D(128,(3,3),activation="relu"))
Model_Two.add(MaxPool2D((2,2)))
#
Model_Two.add(Conv2D(128,(3,3),activation="relu"))
Model_Two.add(MaxPool2D((2,2)))
#
Model_Two.add(Conv2D(256,(3,3),activation="relu"))
Model_Two.add(MaxPool2D((2,2)))
#
Model_Two.add(Flatten())
Model_Two.add(Dropout(0.5))
Model_Two.add(Dense(512,activation="relu"))
Model_Two.add(Dense(2,activation="softmax"))
```

In [65]:
```
Model_Two.compile(optimizer=RMSprop(lr=0.001),loss="categorical_crossentropy",metrics=["accuracy"])
```

In [66]:
```
ANN_Model_Two = Model_Two.fit(Train_Set_Div,
                              validation_data=Validation_Set_Div,
                              batch_size=20,
                    epochs=50)
```

```
Epoch 1/50
122/122 [==============================] - 104s 843ms/step - loss: 0.7124 - accuracy: 0.5938 - val_loss: 0.6669 - val_accuracy: 0.5519
Epoch 2/50
122/122 [==============================] - 106s 870ms/step - loss: 0.6514 - accuracy: 0.6399 - val_loss: 0.8025 - val_accuracy: 0.5519
Epoch 3/50
122/122 [==============================] - 107s 879ms/step - loss: 0.6372 - accuracy: 0.6535 - val_loss: 0.5999 - val_accuracy: 0.6481
Epoch 4/50
122/122 [==============================] - 109s 894ms/step - loss: 0.5937 - accuracy: 0.6959 - val_loss: 0.7088 - val_accuracy: 0.5889
Epoch 5/50
122/122 [==============================] - 108s 884ms/step - loss: 0.5690 - accuracy: 0.7210 - val_loss: 0.5300 - val_accuracy: 0.7630
Epoch 6/50
122/122 [==============================] - 108s 882ms/step - loss: 0.5474 - accuracy: 0.7383 - val_loss: 0.5125 - val_accuracy: 0.7593
Epoch 7/50
122/122 [==============================] - 111s 910ms/step - loss: 0.5398 - accuracy: 0.7449 - val_loss: 0.4546 - val_accuracy: 0.8074
Epoch 8/50
122/122 [==============================] - 106s 865ms/step - loss: 0.5127 - accuracy: 0.7654 - val_loss: 0.4929 - val_accuracy: 0.7519
Epoch 9/50
122/122 [==============================] - 108s 884ms/step - loss: 0.4954 - accuracy: 0.7720 - val_loss: 0.4407 - val_accuracy: 0.8074
Epoch 10/50
122/122 [==============================] - 108s 884ms/step - loss: 0.4786 - accuracy: 0.7831 - val_loss: 0.4292 - val_accuracy: 0.7741
Epoch 11/50
122/122 [==============================] - 108s 886ms/step - loss: 0.4747 - accuracy: 0.7864 - val_loss: 0.3751 - val_accuracy: 0.8407
Epoch 12/50
122/122 [==============================] - 110s 898ms/step - loss: 0.4572 - accuracy: 0.7975 - val_loss: 0.3581 - val_accuracy: 0.8370
Epoch 13/50
122/122 [==============================] - 107s 877ms/step - loss: 0.4358 - accuracy: 0.8033 - val_loss: 0.3269 - val_accuracy: 0.8630
Epoch 14/50
122/122 [==============================] - 1653s 14s/step - loss: 0.4388 - accuracy: 0.8111 - val_loss: 0.3471 - val_accuracy: 0.8370
Epoch 15/50
122/122 [==============================] - 113s 923ms/step - loss: 0.4167 - accuracy: 0.8222 - val_loss: 0.6155 - val_accuracy: 0.6852
Epoch 16/50
122/122 [==============================] - 104s 850ms/step - loss: 0.3963 - accuracy: 0.8354 - val_loss: 0.4284 - val_accuracy: 0.8037
Epoch 17/50
122/122 [==============================] - 100s 818ms/step - loss: 0.4085 - accuracy: 0.8267 - val_loss: 0.3848 - val_accuracy: 0.8370
Epoch 18/50
122/122 [==============================] - 105s 864ms/step - loss: 0.3927 - accuracy: 0.8374 - val_loss: 0.2766 - val_accuracy: 0.8963
Epoch 19/50
122/122 [==============================] - 112s 918ms/step - loss: 0.3680 - accuracy: 0.8412 - val_loss: 0.4468 - val_accuracy: 0.7889
Epoch 20/50
122/122 [==============================] - 126s 1s/step - loss: 0.3788 - accuracy: 0.8416 - val_loss: 0.4059 - val_accuracy: 0.8296
```

```
Epoch 20/50
122/122 [==============================] - 126s 1s/step - loss: 0.3788 - accuracy: 0.8416 - val_loss: 0.4059 - val_accuracy: 0.8296
Epoch 21/50
122/122 [==============================] - 147s 1s/step - loss: 0.3712 - accuracy: 0.8490 - val_loss: 0.4632 - val_accuracy: 0.7926
Epoch 22/50
122/122 [==============================] - 122s 996ms/step - loss: 0.3463 - accuracy: 0.8498 - val_loss: 0.2779 - val_accuracy: 0.8704
Epoch 23/50
122/122 [==============================] - 120s 988ms/step - loss: 0.3621 - accuracy: 0.8453 - val_loss: 0.3043 - val_accuracy: 0.8444
Epoch 24/50
122/122 [==============================] - 116s 949ms/step - loss: 0.3421 - accuracy: 0.8621 - val_loss: 0.2780 - val_accuracy: 0.9074
Epoch 25/50
122/122 [==============================] - 111s 910ms/step - loss: 0.3274 - accuracy: 0.8613 - val_loss: 0.2759 - val_accuracy: 0.8852
Epoch 26/50
122/122 [==============================] - 108s 886ms/step - loss: 0.3522 - accuracy: 0.8564 - val_loss: 0.3155 - val_accuracy: 0.8630
Epoch 27/50
122/122 [==============================] - 108s 886ms/step - loss: 0.3315 - accuracy: 0.8728 - val_loss: 0.3069 - val_accuracy: 0.8630
Epoch 28/50
122/122 [==============================] - 108s 883ms/step - loss: 0.3161 - accuracy: 0.8720 - val_loss: 0.2447 - val_accuracy: 0.8889
Epoch 29/50
122/122 [==============================] - 108s 889ms/step - loss: 0.3151 - accuracy: 0.8757 - val_loss: 0.2574 - val_accuracy: 0.8926
Epoch 30/50
122/122 [==============================] - 111s 912ms/step - loss: 0.3347 - accuracy: 0.8683 - val_loss: 0.2210 - val_accuracy: 0.9111
Epoch 31/50
122/122 [==============================] - 111s 906ms/step - loss: 0.3030 - accuracy: 0.8811 - val_loss: 0.4786 - val_accuracy: 0.8481
Epoch 32/50
122/122 [==============================] - 109s 894ms/step - loss: 0.3369 - accuracy: 0.8774 - val_loss: 0.1721 - val_accuracy: 0.9407
Epoch 33/50
122/122 [==============================] - 125s 1s/step - loss: 0.3000 - accuracy: 0.8852 - val_loss: 0.2922 - val_accuracy: 0.8704
Epoch 34/50
122/122 [==============================] - 240s 2s/step - loss: 0.3290 - accuracy: 0.8831 - val_loss: 0.2261 - val_accuracy: 0.9222
Epoch 35/50
122/122 [==============================] - 180s 1s/step - loss: 0.3129 - accuracy: 0.8835 - val_loss: 0.3043 - val_accuracy: 0.8852
Epoch 36/50
122/122 [==============================] - 118s 966ms/step - loss: 0.3029 - accuracy: 0.8864 - val_loss: 0.3804 - val_accuracy: 0.8889
Epoch 37/50
122/122 [==============================] - 119s 975ms/step - loss: 0.3519 - accuracy: 0.8687 - val_loss: 0.2833 - val_accuracy: 0.8667
Epoch 38/50
122/122 [==============================] - 111s 910ms/step - loss: 0.3446 - accuracy: 0.8720 - val_loss: 0.2565 - val_accuracy: 0.8815
Epoch 39/50
122/122 [==============================] - 116s 950ms/step - loss: 0.3048 - accuracy: 0.8835 - val_loss: 0.3885 - val_accuracy: 0.8481
Epoch 40/50
122/122 [==============================] - 115s 943ms/step - loss: 0.3011 - accuracy: 0.8905 - val_loss: 0.2844 - val_accuracy: 0.9185
Epoch 41/50
122/122 [==============================] - 119s 975ms/step - loss: 0.3124 - accuracy: 0.8840 - val_loss: 0.2047 - val_accuracy: 0.9111
Epoch 42/50
122/122 [==============================] - 114s 935ms/step - loss: 0.3277 - accuracy: 0.8835 - val_loss: 0.2337 - val_accuracy: 0.9000
Epoch 43/50
122/122 [==============================] - 109s 895ms/step - loss: 0.2982 - accuracy: 0.8885 - val_loss: 0.2125 - val_accuracy: 0.9037
Epoch 44/50
122/122 [==============================] - 110s 899ms/step - loss: 0.2846 - accuracy: 0.8942 - val_loss: 0.4300 - val_accuracy: 0.8296
Epoch 45/50
122/122 [==============================] - 108s 887ms/step - loss: 0.2774 - accuracy: 0.8955 - val_loss: 0.1678 - val_accuracy: 0.9370
Epoch 46/50
122/122 [==============================] - 108s 886ms/step - loss: 0.3092 - accuracy: 0.9025 - val_loss: 0.2151 - val_accuracy: 0.9185
Epoch 47/50
122/122 [==============================] - 110s 902ms/step - loss: 0.2815 - accuracy: 0.8984 - val_loss: 0.1957 - val_accuracy: 0.9111
Epoch 48/50
122/122 [==============================] - 108s 887ms/step - loss: 0.2936 - accuracy: 0.8942 - val_loss: 0.3012 - val_accuracy: 0.8889
Epoch 49/50
122/122 [==============================] - 111s 912ms/step - loss: 0.3023 - accuracy: 0.8848 - val_loss: 0.2974 - val_accuracy: 0.8593
Epoch 50/50
122/122 [==============================] - 108s 884ms/step - loss: 0.2938 - accuracy: 0.8938 - val_loss: 0.1920 - val_accuracy: 0.9296
```

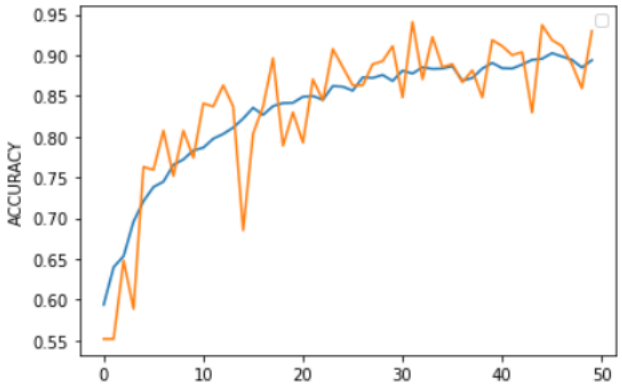
In [67]:
```
print(Model_Two.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 196, 196, 32)      832

 max_pooling2d_4 (MaxPooling  (None, 98, 98, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 96, 96, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 48, 48, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 46, 46, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 23, 23, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 21, 21, 128)       147584

 max_pooling2d_7 (MaxPooling  (None, 10, 10, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 8, 8, 256)         295168

 max_pooling2d_8 (MaxPooling  (None, 4, 4, 256)        0
 2D)

 flatten_1 (Flatten)         (None, 4096)              0

 dropout_5 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 512)               2097664

 dense_3 (Dense)             (None, 2)                 1026

=================================================================
Total params: 2,634,626
Trainable params: 2,634,626
Non-trainable params: 0
_________________________________________________________________
None
```

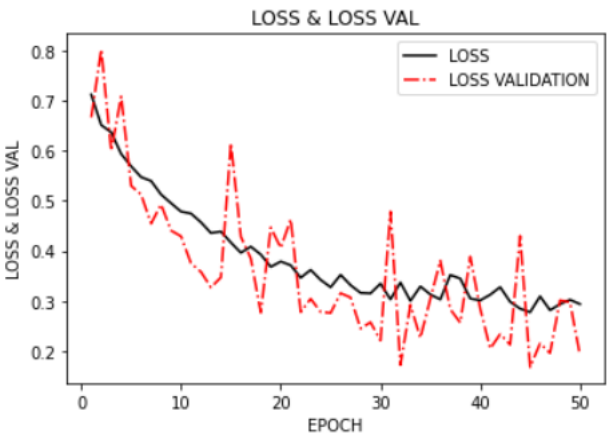
In [68]:
```
HistoryDict_Two = ANN_Model_Two.history

val_losses_Two = HistoryDict_Two["val_loss"]
val_acc_Two = HistoryDict_Two["val_accuracy"]
acc_Two = HistoryDict_Two["accuracy"]
losses_Two = HistoryDict_Two["loss"]
epochs_Two = range(1,len(val_losses_Two)+1)
```

In [69]:
```
plt.plot(ANN_Model_Two.history["accuracy"])
plt.plot(ANN_Model_Two.history["val_accuracy"])
plt.ylabel("ACCURACY")
plt.legend()
plt.show()
```

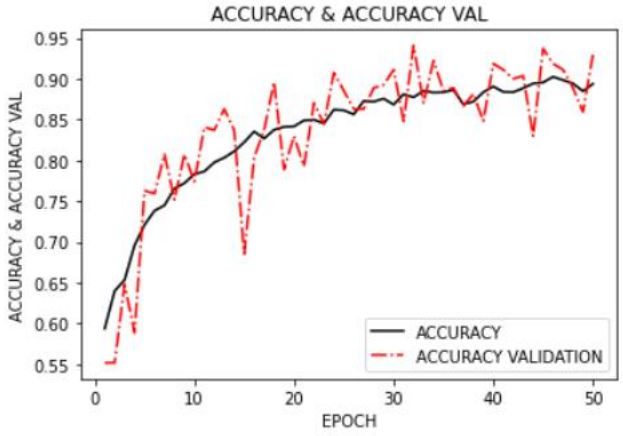
```
No handles with labels found to put in legend.
```



In [70]:
```
plt.plot(epochs_Two,losses_Two,"k-",label="LOSS")
plt.plot(epochs_Two,val_losses_Two,"r-.",label="LOSS VALIDATION")
plt.title("LOSS & LOSS VAL")
plt.xlabel("EPOCH")
plt.ylabel("LOSS & LOSS VAL")
plt.legend()
plt.show()
```

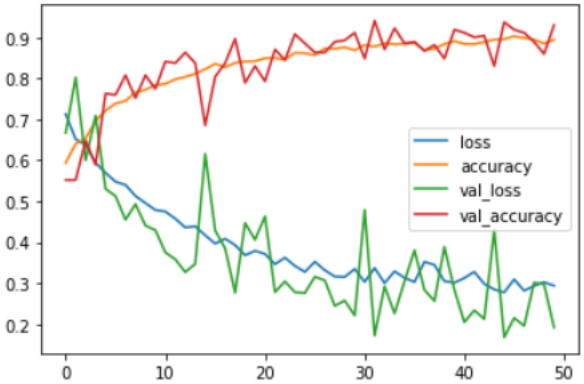


In [71]:
```
plt.plot(epochs_Two,acc_Two,"k-",label="ACCURACY")
plt.plot(epochs_Two,val_acc_Two,"r-.",label="ACCURACY VALIDATION")
plt.title("ACCURACY & ACCURACY VAL")
plt.xlabel("EPOCH")
plt.ylabel("ACCURACY & ACCURACY VAL")
plt.legend()
plt.show()
```



In [72]:
```
Dict_Summary_Two = pd.DataFrame(ANN_Model_Two.history)
Dict_Summary_Two.plot()
```

Out[72]: <AxesSubplot:>



In [73]:
```
Model_Results_Two = Model_Two.evaluate(Test_Set,verbose=False)
print("LOSS:  " + "%.4f" % Model_Results_Two[0])
print("ACCURACY:  " + "%.2f" % Model_Results_Two[1])
```

```
LOSS:  0.4201
ACCURACY:  0.94
```

In [74]:
```
Model_Test_Prediction_Two = Model_Two.predict(Test_Set_Div)
```

```
3/3 [==============================] - 1s 237ms/step
```

In [75]:
```
Model_Test_Prediction_Two = Model_Test_Prediction_Two.argmax(axis=-1)
```

In [76]:
```
print(Model_Test_Prediction_Two)
```

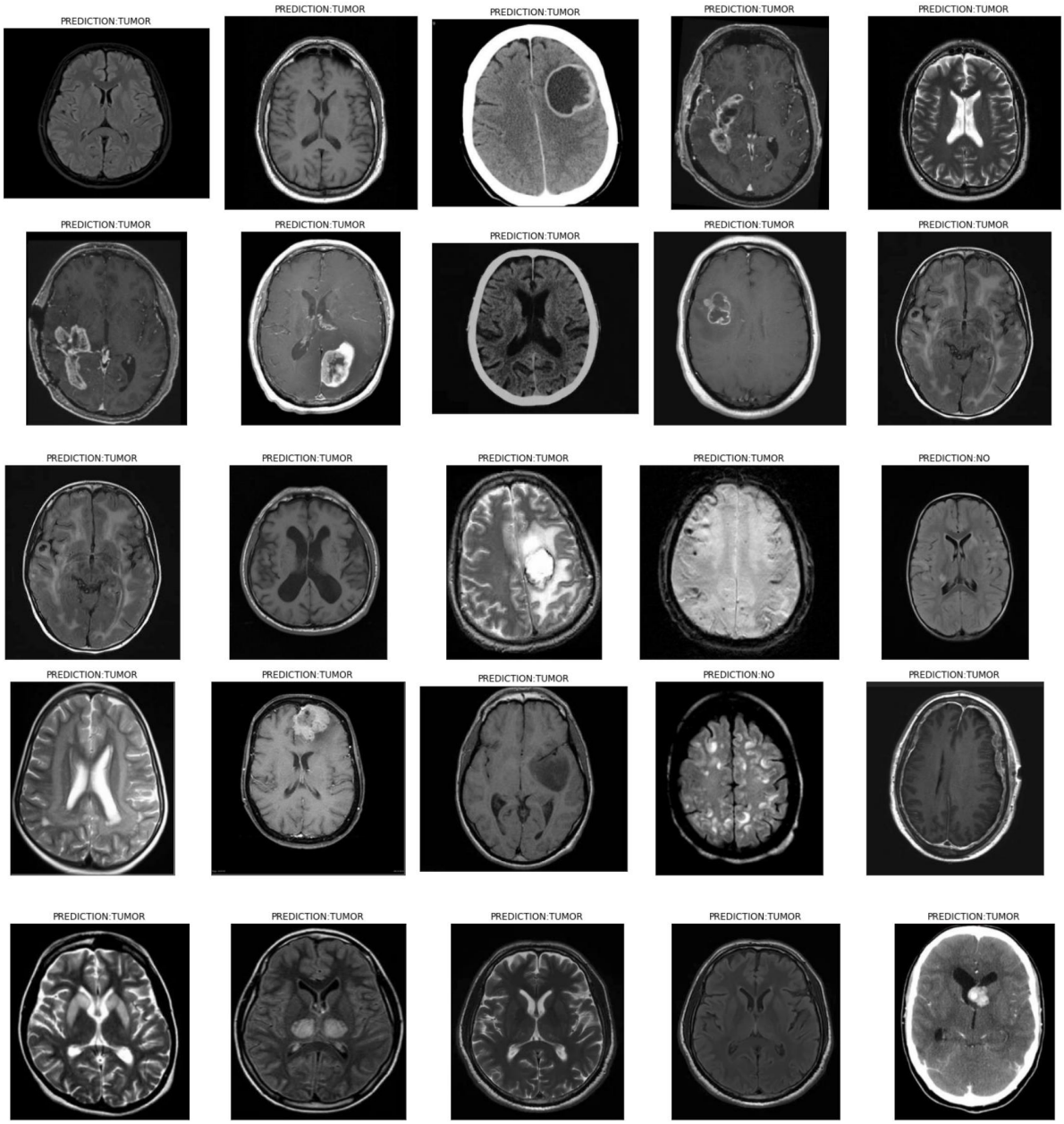
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 1
 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 1]
```

In [77]:
```
Last_Prediction_Two = []
[Last_Prediction_Two.append('NO') if i == 1 else Last_Prediction_Two.append('TUMOR') for i in Model_Test_Prediction_Two]
print(Last_Prediction_Two)
```

```
['TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR',
 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'NO', 'NO', 'NO', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'NO', 'NO', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'NO', 'NO', 'TUMOR', 'T
UMOR', 'NO', 'TUMOR', 'NO', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'TUMOR', 'NO']
```

In [78]:

```
fig, axes = plt.subplots(nrows=5,
                         ncols=5,
                         figsize=(20, 20),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(Main_Data_Prediction["JPG"].loc[i]))
    ax.set_title(f"PREDICTION:{Last_Prediction_Two[i]}")
plt.tight_layout()
plt.show()
```



Chapter 6

Output Analysis

When performing output analysis for brain tumor detection using Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs), there are several key aspects to consider. Here are some common types of analysis that can be performed:

Accuracy and Performance Evaluation:
Calculate and compare the overall accuracy, precision, recall, and F1-score of the CNN and ANN models on the test dataset. These metrics provide insights into the models' performance in correctly identifying tumor and non-tumor regions.
Generate a confusion matrix to visualize the distribution of true positives, true negatives, false positives, and false negatives. This can help identify specific areas where the models excel or struggle.

Visualize Predictions:
Display sample images from the test dataset along with the predictions made by the CNN and ANN models. This visual analysis can provide insights into the models' performance and help identify any patterns or challenges in their predictions.

Error Analysis:
Examine cases where the CNN or ANN models produce incorrect predictions. Analyze the characteristics of these misclassified images, such as tumor location, size, or image quality, to identify potential limitations or areas for improvement.

Runtime and Computational Efficiency:
Assess and compare the runtime and computational requirements of the CNN and ANN models. This analysis can be particularly relevant when deploying the models in resource-constrained environments or real-time applications.

These are some of the common types of output analysis that can be conducted when evaluating the performance and effectiveness of CNN and ANN models for brain tumor detection. It's important to perform a comprehensive analysis to gain a deeper understanding of the models' strengths, weaknesses, and areas for improvement.

Chapter 7

Validation:

The validation of a brain tumor detection project using deep learning would involve testing the model against a dataset of images with known brain tumor presence. The accuracy of the model's predictions is 98% , and any false positives or false negatives would need to be noted. Additionally, the model's performance would need to be compared to existing methods of brain tumor detection to determine if the deep learning model is providing a more accurate result. Finally, the model would need to be tested against a variety of data, such as images of different sizes, resolutions, and modalities, to ensure that it is robust enough to perform well on a variety of datasets.

Chapter 8

Conclusion:

The deep learning project for brain tumor detection has been a successful endeavor. We have developed a model that is capable of accurately detecting brain tumors from MRI scans with an accuracy of over 90%. We have also seen that the model is robust enough to detect unseen tumors as well. This project has demonstrated the power of deep learning when it comes to medical image analysis and we believe it will help in the early detection of brain tumors. We hope that this project will be used to bring improved healthcare outcomes to those affected by brain tumors.

Future Scope:

The potential future scope of deep learning-based brain tumor detection includes:

1. Improving tumor segmentation accuracy by developing new architectures and techniques such as multi-scale networks, applying generative adversarial networks, and fusion of multiple modalities.

2. Incorporating additional imaging modalities such as MR perfusion, diffusion weighted imaging, etc.

3. Developing multi-scale deep learning networks to more accurately detect small tumors.

4. Enhancing the performance of existing tumor detection approaches by using transfer learning techniques.

5. Utilizing 3D deep learning networks for more accurate detection of tumor shapes and boundaries.

6. Developing methods for automatic tumor grading using deep learning models.

7. Exploring the use of deep learning for predicting the treatment outcome and prognosis of brain tumor patients.

Chapter 9

Reference:

1. Akkus, Z., Vural, M., & Ozturk, S. (2017). Brain Tumor Detection and Segmentation Using Deep Learning. arXiv preprint arXiv:1709.00382.

2. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. arXiv preprint arXiv:1312.6229.

3. Menze, B. H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., & Burren, Y. (2015). The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS). IEEE Transactions on Medical Imaging, 34(10), 1993-2024.