BRAIN TUMOR CLASSIFICATION USING MRI

A Deep Learning Approach

+ Mohammed Sharaf -

13/02/2024

mymsharaff@gmail.com

Introduction

DETECTING AND CLASSIFYING BRAIN TUMORS USING DEEP LEARNING

- Brain tumors, affecting both children and adults, account for a significant portion of primary Central Nervous System (CNS) tumors.
- Annual diagnoses of approximately 11,700 people highlight the urgency for accurate diagnostics and treatment planning.
- Classifications include Benign Tumor, Malignant Tumor, Pituitary Tumor, among others.
- The 5-year survival rates underscore the need for improved detection methods, with a 34% rate for men and 36% for women.

The Challenge:

MANUAL EXAMINATION OF MRI SCANS BY RADIOLOGISTS IS ERROR-PRONE DUE TO THE COMPLEXITIES OF BRAIN TUMORS AND THEIR PROPERTIES.

The Solution:

- Leveraging the power of Machine Learning (ML) and Artificial Intelligence (AI) through Deep Learning Algorithms.
- Proposal includes the application of Convolutional Neural Network (CNN), Artificial Neural Network (ANN), and Transfer Learning (TL).

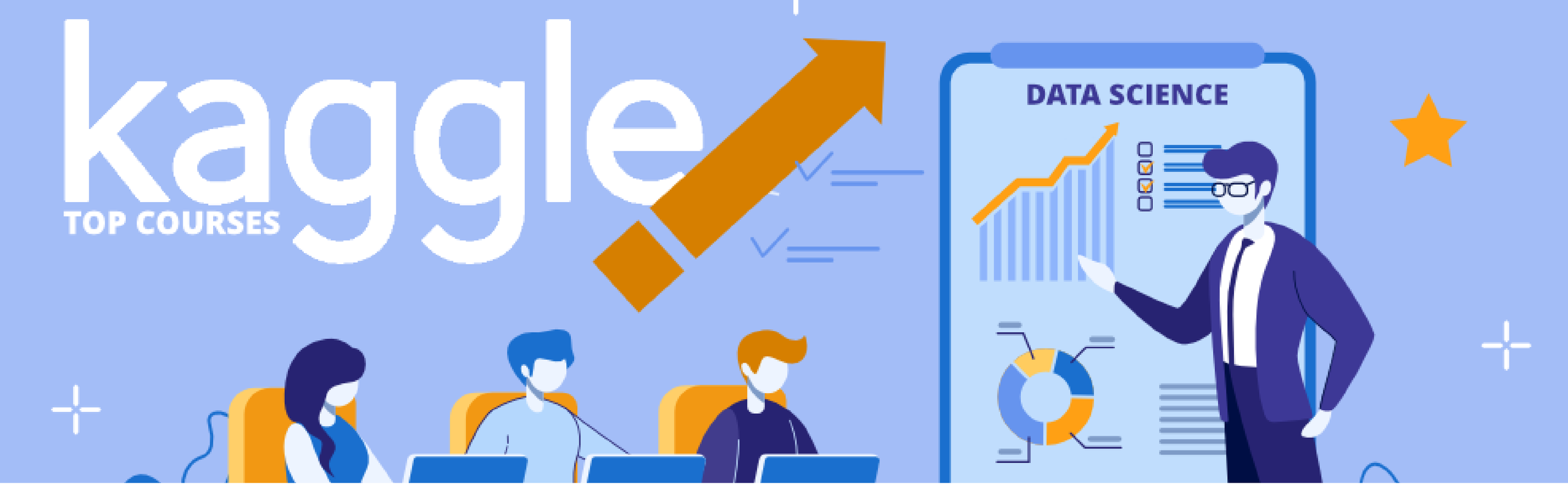
Objective

- Develop a robust deep learning model specifically designed for the precise classification of brain tumors in MRI images.

Dataset Overview

DATA COLLECTION PROCESS

- Buy data from a third-party vendor:
  - Purchasing data from reputable third-party vendors can provide a diverse and pre-annotated dataset for your project.
- Collect and annotate data on your own:
  - Manually collecting and annotating data ensures a hands-on approach, allowing for a more tailored and specific dataset based on project requirements.
- Write web scraping scripts to collect images from the internet:
  - Developing web scraping scripts enables the automated extraction of images from the internet, offering a scalable solution for data collection.

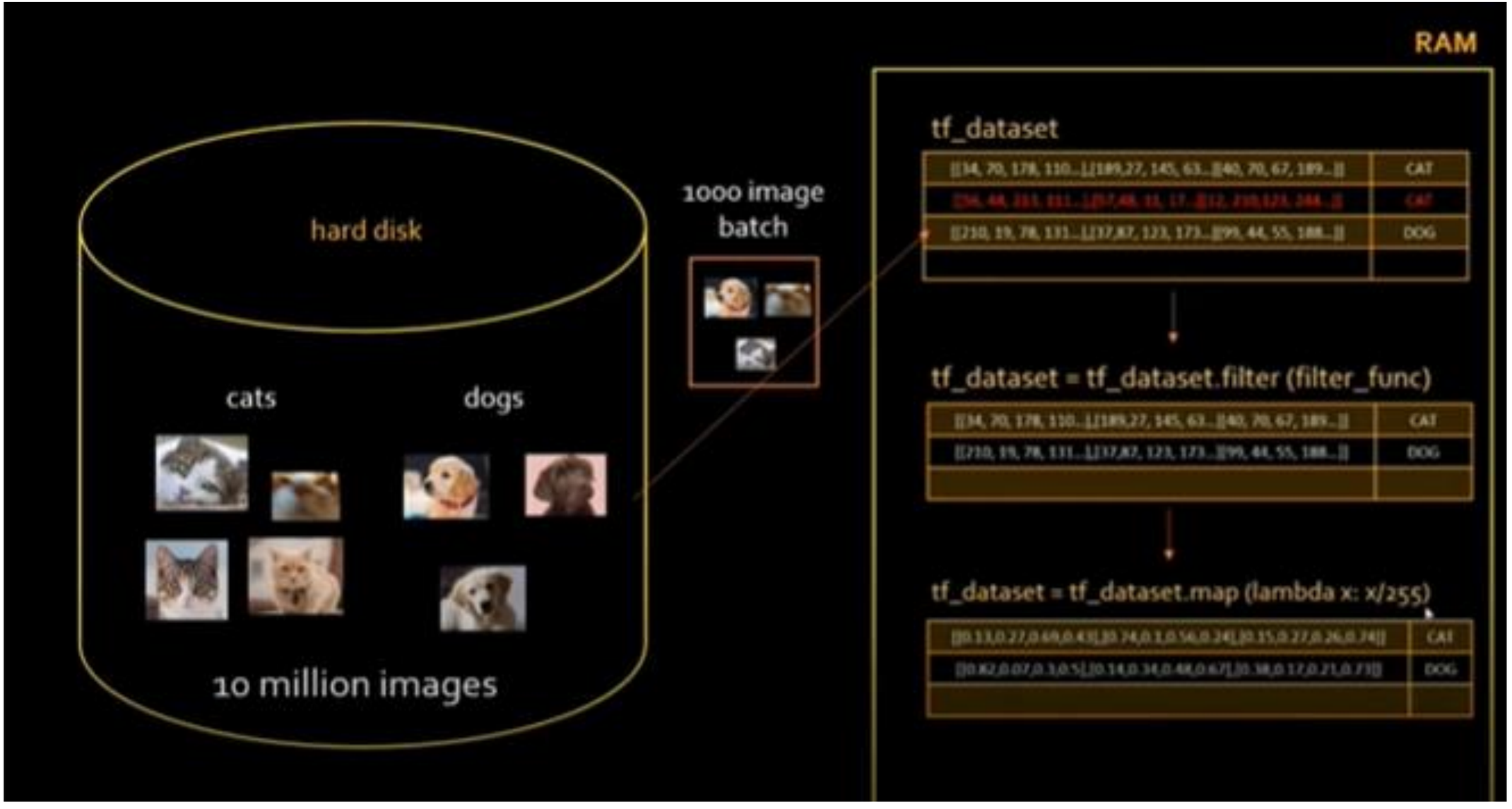


We Used Kaggle Data set it’s free and clean

Source: https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

Files :

Name	Date modified	Type
glioma_tumor	2/9/2024 10:34 PM	File folder
meningioma_tumor	2/9/2024 10:35 PM	File folder
no_tumor	2/9/2024 10:35 PM	File folder
pituitary_tumor	2/9/2024 10:36 PM	File folder


RAM
hard disk
cats
dogs
10 million images
1000 image batch
tf_dataset
[[34, 70, 178, 110...],[189,27, 145, 63...],[40, 70, 67, 189...]]
CAT
[[210, 19, 78, 131...],[37,87, 123, 173...],[99, 44, 55, 188...]]
DOG
tf_dataset = tf_dataset.filter (filter_func)
[[34, 70, 178, 110...],[189,27, 145, 63...],[40, 70, 67, 189...]]
CAT
[[210, 19, 78, 131...],[37,87, 123, 173...],[99, 44, 55, 188...]]
DOG
tf_dataset = tf_dataset.map (lambda x: x/255)
CAT
DOG

IMPORT LIBRARIES

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

[2]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
train_size = 0.8
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "MRI",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

So, here we have a total of 3160 images belonging to 4 classes.

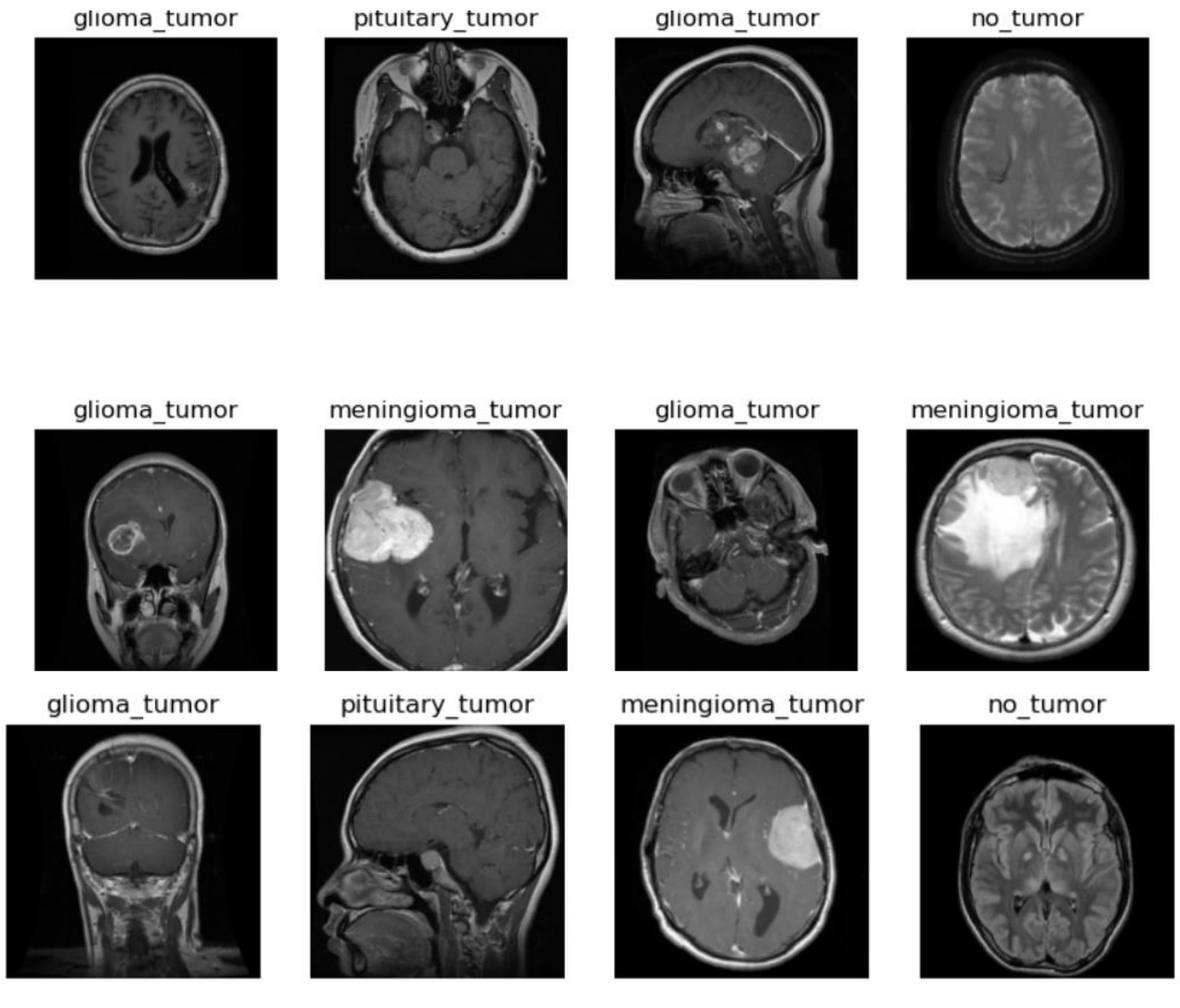
The class names are:

- Glioma Tumor,
- Meningioma Tumor,
- No Tumor,
- Pituitary Tumor."

Data Visualization

Visualize some of the images from our dataset

In [5]:
```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Function to Split Dataset

In [7]:
```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

In [8]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [9]:
```
print("total Dataset : "+str(len(dataset))+"\n train_ds : " + str(len(train_ds)) + "\n val_ds : " + str(len(val_ds)) + "\n test_d
```

```
total Dataset : 99
 train_ds : 79
 val_ds : 9
 test_ds : 11
```

Cache, Shuffle, and Prefetch the Dataset

In [10]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Building the Model

Creating a Layer for Resizing and Normalization

In [11]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

Data Augmentation

Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

In [12]:
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Applying Data Augmentation to Train Dataset

In [13]:
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

In [18]:
```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 4

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

In [19]:
```
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
conv2d_6 (Conv2D)	(32, 254, 254, 32)	896
max_pooling2d_6 (MaxPooling2D)	(32, 127, 127, 32)	0
conv2d_7 (Conv2D)	(32, 125, 125, 64)	18496
max_pooling2d_7 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_8 (Conv2D)	(32, 60, 60, 64)	36928
max_pooling2d_8 (MaxPooling2D)	(32, 30, 30, 64)	0
conv2d_9 (Conv2D)	(32, 28, 28, 64)	36928
max_pooling2d_9 (MaxPooling2D)	(32, 14, 14, 64)	0
conv2d_10 (Conv2D)	(32, 12, 12, 64)	36928
max_pooling2d_10 (MaxPooling2D)	(32, 6, 6, 64)	0
conv2d_11 (Conv2D)	(32, 4, 4, 64)	36928
max_pooling2d_11 (MaxPooling2D)	(32, 2, 2, 64)	0
flatten_1 (Flatten)	(32, 256)	0
dense_2 (Dense)	(32, 64)	16448
dense_3 (Dense)	(32, 4)	260

Total params: 183812 (718.02 KB)
Trainable params: 183812 (718.02 KB)
Non-trainable params: 0 (0.00 Byte)

Compiling the Model

In [20]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [21]:
```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

```
79/79 [==============================] - 103s 1s/step - loss: 0.2611 - accuracy: 0.8960 - val_loss: 0.2561 - val_accuracy: 0.
8889
Epoch 46/50
79/79 [==============================] - 105s 1s/step - loss: 0.2599 - accuracy: 0.8980 - val_loss: 0.2231 - val_accuracy: 0.
9306
Epoch 47/50
79/79 [==============================] - 102s 1s/step - loss: 0.2684 - accuracy: 0.9004 - val_loss: 0.2731 - val_accuracy: 0.
9028
Epoch 48/50
79/79 [==============================] - 102s 1s/step - loss: 0.2351 - accuracy: 0.9190 - val_loss: 0.2700 - val_accuracy: 0.
9028
Epoch 49/50
79/79 [==============================] - 102s 1s/step - loss: 0.2310 - accuracy: 0.9111 - val_loss: 0.2543 - val_accuracy: 0.
8993
Epoch 50/50
79/79 [==============================] - 119s 1s/step - loss: 0.2313 - accuracy: 0.9095 - val_loss: 0.2198 - val_accuracy: 0.
9097
```

In [22]:
```
scores = model.evaluate(test_ds)
```

```
11/11 [==============================] - 21s 311ms/step - loss: 0.2470 - accuracy: 0.9099
```

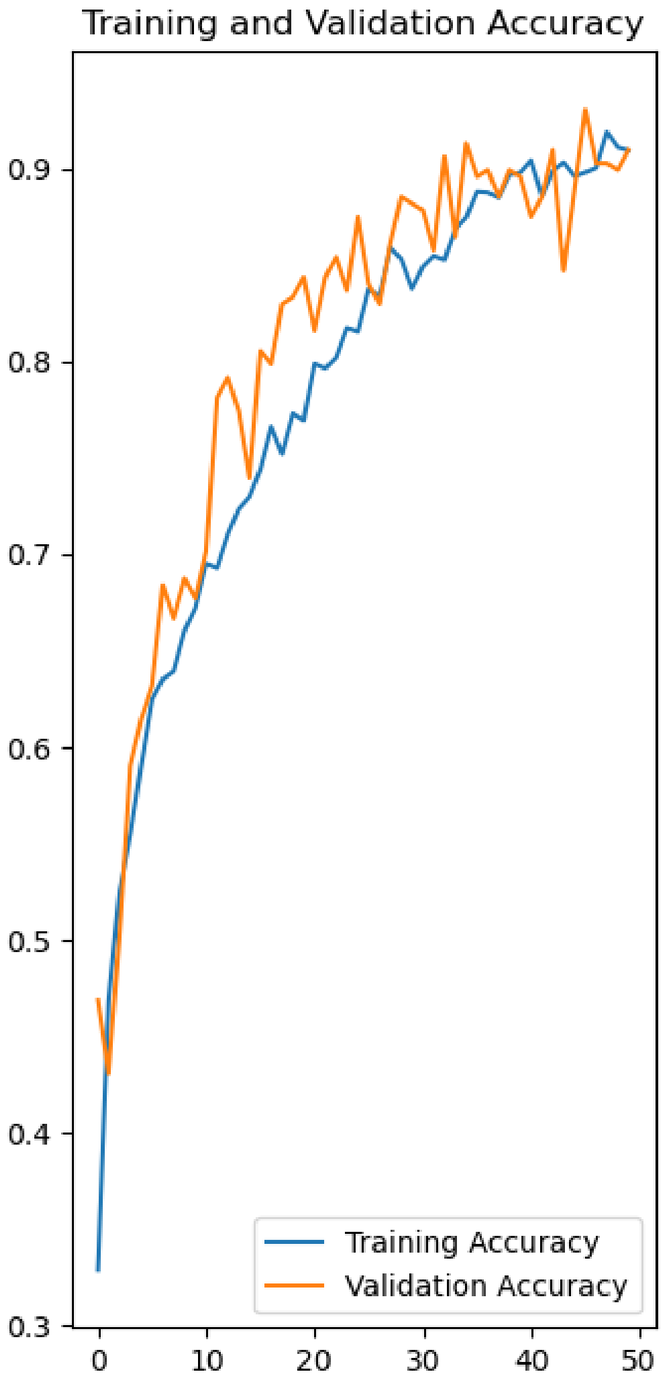
In [23]:
```
scores
```

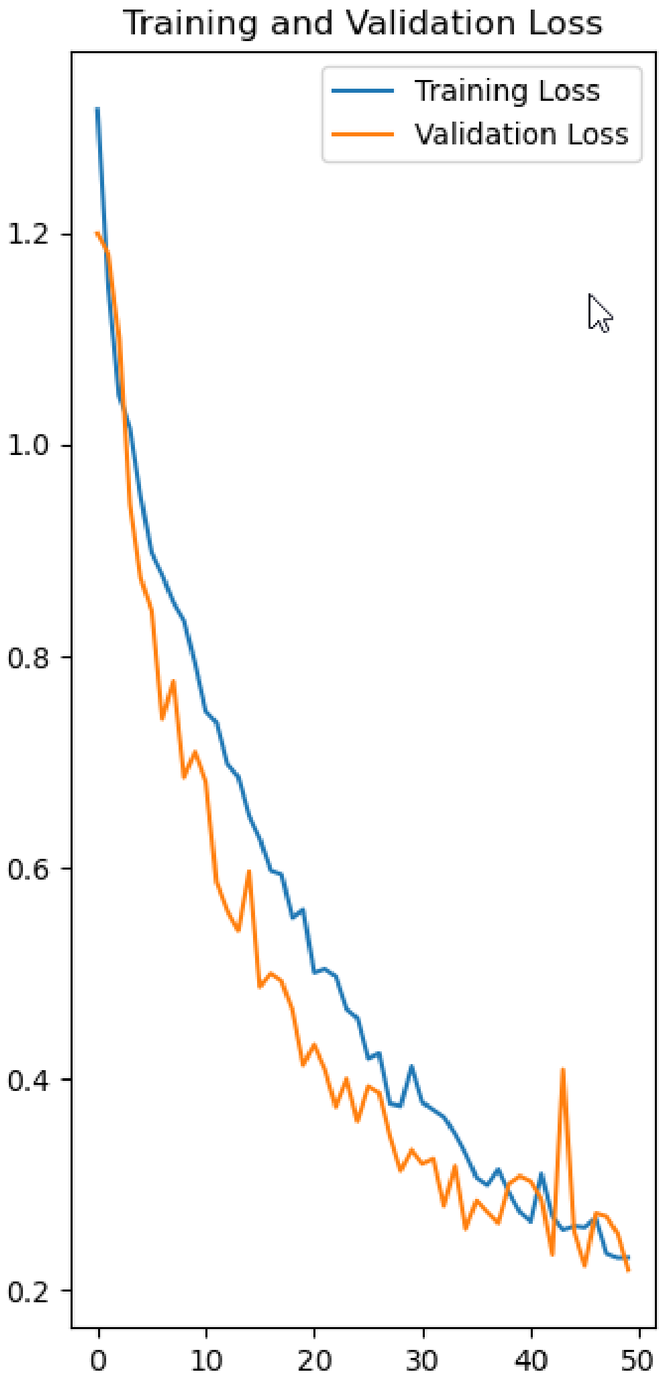
Out[23]:
```
[0.2470352202653885, 0.9098837375640869]
```

Plotting the Accuracy and Loss Curves

In [24]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [25]:
```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

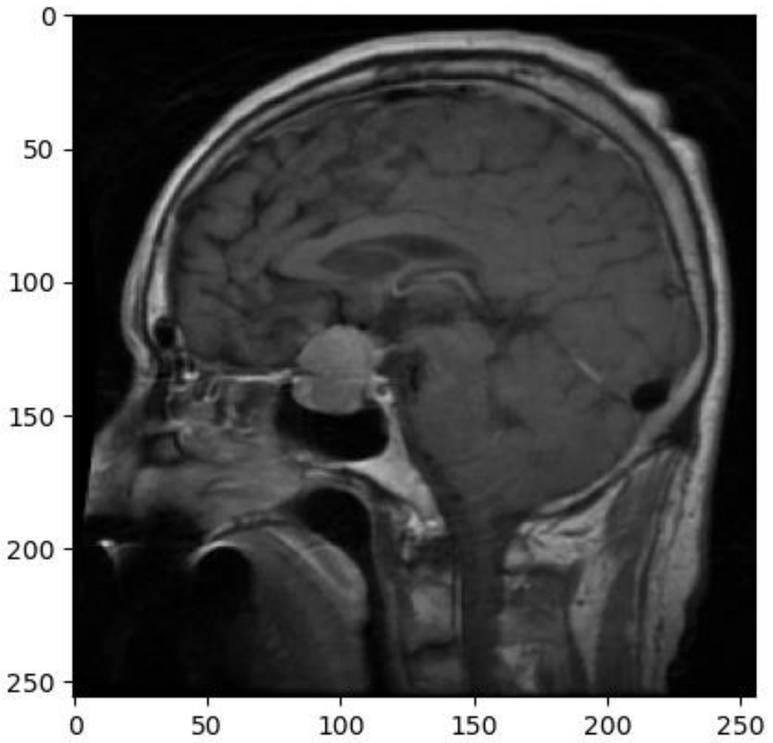

Training and Validation Accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50
Training Accuracy
Validation Accuracy


Training and Validation Loss
Training Loss
Validation Loss
1.2
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50

Run prediction on a sample image

In [26]:
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: pituitary_tumor
1/1 [==============================] - 2s 2s/step
predicted label: pituitary_tumor
```



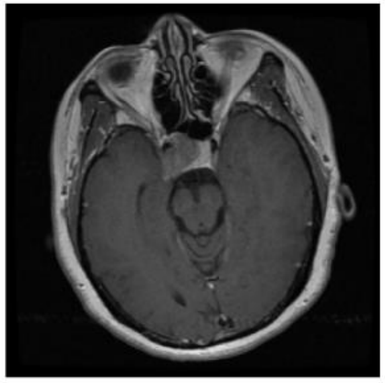
Write a function for inference

In [27]:
```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

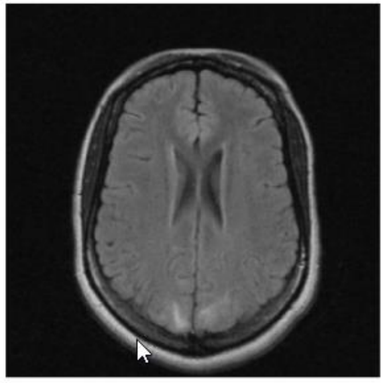
In [28]:
```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

```
1/1 [==============================] - 1s 856ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 68ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 61ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 97ms/step
```

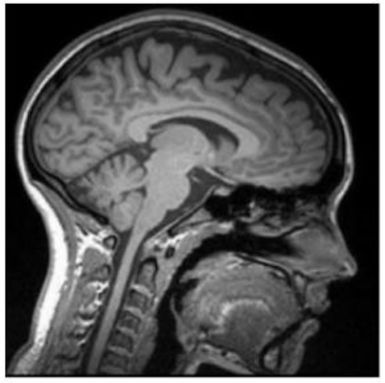
Actual: pituitary_tumor,
Predicted: pituitary_tumor.
Confidence: 100.0%



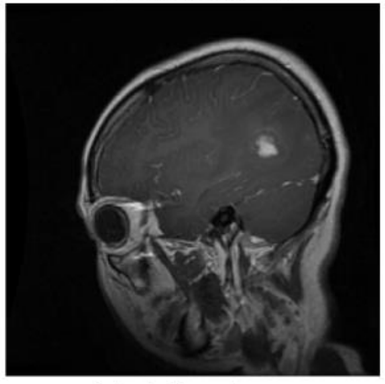
Actual: no_tumor,
Predicted: no_tumor.
Confidence: 98.24%



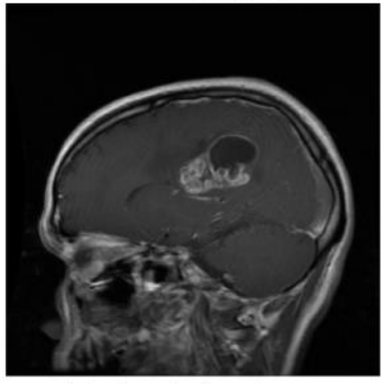
Actual: no_tumor,
Predicted: no_tumor.
Confidence: 99.35%



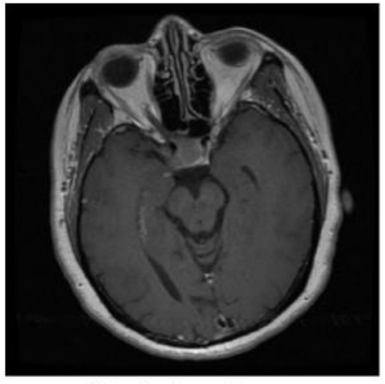
Actual: glioma_tumor,
Predicted: glioma_tumor.
Confidence: 99.11%



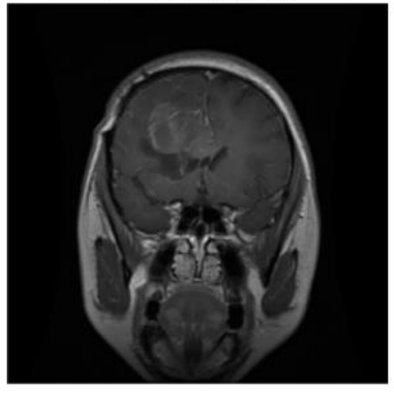
Actual: glioma_tumor,
Predicted: glioma_tumor.
Confidence: 96.44%



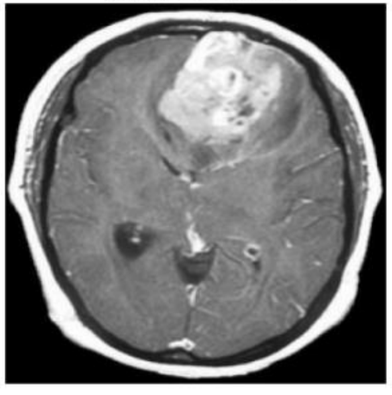
Actual: pituitary_tumor,
Predicted: pituitary_tumor.
Confidence: 99.99%



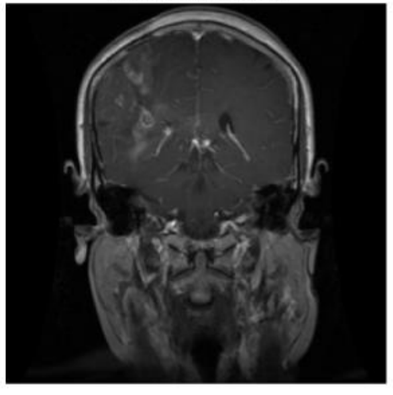
Actual: glioma_tumor,
Predicted: glioma_tumor.
Confidence: 96.26%



Actual: meningioma_tumor,
Predicted: meningioma_tumor.
Confidence: 94.16%



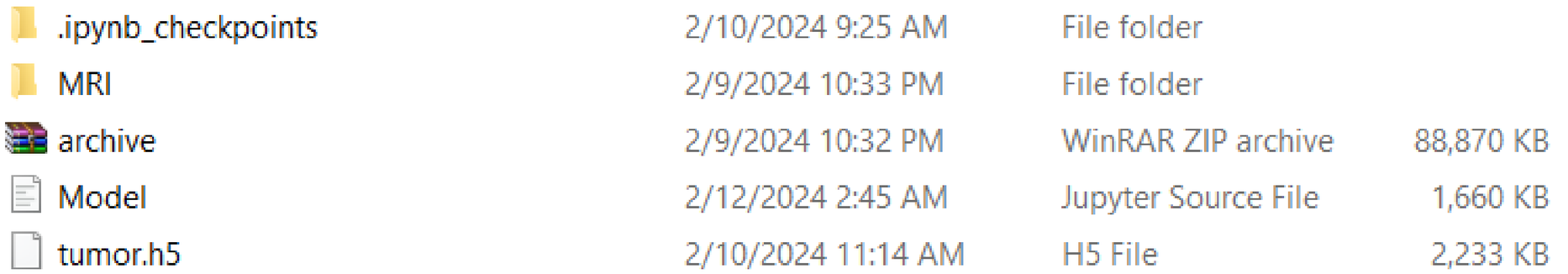
Actual: glioma_tumor,
Predicted: glioma_tumor.
Confidence: 99.9%



Saving the Model

Saving the Model

In [29]:
```
model.save("tumor.h5")
```



.ipynb_checkpoints	2/10/2024 9:25 AM	File folder	
MRI	2/9/2024 10:33 PM	File folder	
archive	2/9/2024 10:32 PM	WinRAR ZIP archive	88,870 KB
Model	2/12/2024 2:45 AM	Jupyter Source File	1,660 KB
tumor.h5	2/10/2024 11:14 AM	H5 File	2,233 KB

Future Work

Potential Improvements:

- Enhanced Model Architectures:
  - Explore more complex neural network architectures.
- Integration of Advanced Techniques:
  - Incorporate attention mechanisms or ensemble learning.
- Fine-Tuning Hyperparameters:
  - Conduct thorough hyperparameter tuning for optimal values.
- Continued Data Augmentation:
  - Experiment with additional augmentation techniques.

Limitations of Current Approach:

- Data Imbalance:
  - Address potential class imbalance issues.
- Interpretable Results:
  - Investigate methods for improving result interpretability.
- Incorporate Patient History:
  - Explore integration of patient history data.
- Real-time Deployment Challenges:
  - Overcome challenges associated with real-time deployment.

Conclusion

In conclusion, our deep learning model, trained on a dataset of 3160 MRI images, effectively classifies brain tumors into Glioma, Meningioma, No Tumor, and Pituitary Tumor. Through robust preprocessing, including resizing and data augmentation, the model demonstrates resilience and promising accuracy.

The significance of our work in medical image classification lies in its potential to streamline brain tumor diagnosis. The model's accuracy contributes to early detection, impacting patient outcomes and optimizing healthcare resources by reducing dependence on specialized radiologists.

Looking forward, our commitment extends to refining and advancing the model, addressing limitations, and embracing future implications. By contributing to the evolution of medical image classification, we aim to enhance healthcare outcomes for individuals facing brain tumor challenges.


Q&A

T	h	a
n	k	
Y	O	U