Tumor Detection and type identification

Shivam Amrutkar

---> LinkedIn

---> Github

---> **Github link for project**

Importing required modules

In [ ]:

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

Declaring Constants

In [ ]:

```
BATCH_SIZE = 16
IMAGE_SIZE = 512
CHANNELS = 3
EPOCHS = 20
```

Loading datasets

Training

In [ ]:

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

```
Found 2870 files belonging to 4 classes.
```

Testing

In [ ]:

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

```
Found 394 files belonging to 4 classes.
```

Class names

In [ ]:

```
class_names = train_ds.class_names
class_names
```

Out[ ]:

```
['glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor']
```

Shape of image

In [ ]:

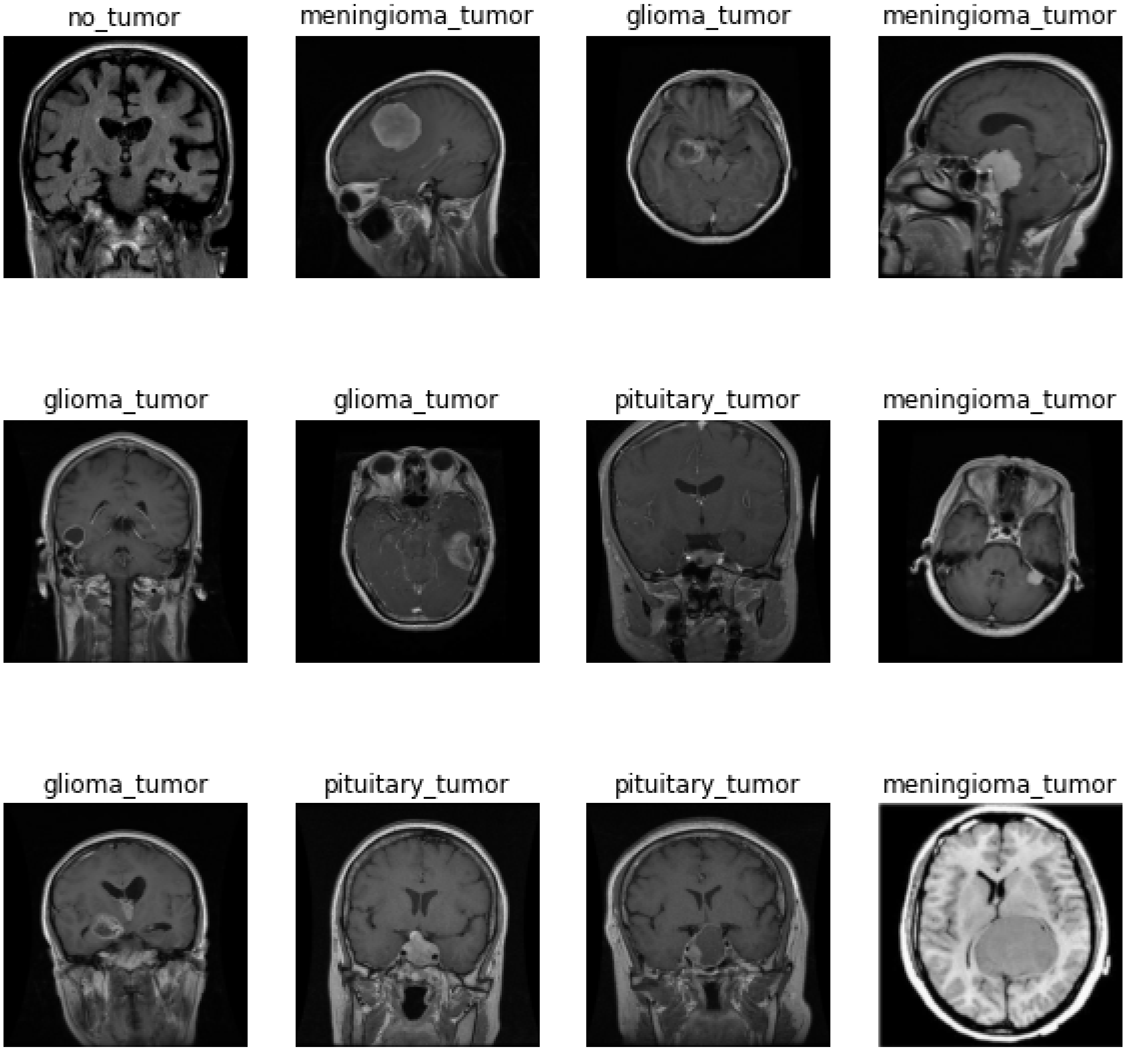
```
for image_batch, labels_batch in train_ds.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(16, 512, 512, 3)
[2 3 0 1 3 1 0 3 1 1 0 3 3 1 3 1]
```

Images with labels [Reference]

In [ ]:

```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in train_ds.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")
```



no_tumor meningioma_tumor glioma_tumor meningioma_tumor

glioma_tumor glioma_tumor pituitary_tumor meningioma_tumor

glioma_tumor pituitary_tumor pituitary_tumor meningioma_tumor

Extra layers

Resize & Rescale layers

In [ ]:

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

Rotation & Orientation layers

In [ ]:

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

Layers

In [ ]:

```
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)
```

Summary of model

In [ ]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (16, 512, 512, 3)         0         
                                                                 
 conv2d (Conv2D)             (16, 510, 510, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (16, 255, 255, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (16, 253, 253, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (16, 126, 126, 64)       0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (16, 124, 124, 64)        36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (16, 62, 62, 64)         0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (16, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPooling  (16, 30, 30, 64)         0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (16, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPooling  (16, 14, 14, 64)         0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (16, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_5 (MaxPooling  (16, 6, 6, 64)           0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (16, 2304)                0         
                                                                 
 dense (Dense)               (16, 64)                  147520    
                                                                 
 dense_1 (Dense)             (16, 4)                   260       
                                                                 
=================================================================
Total params: 314,884
Trainable params: 314,884
Non-trainable params: 0
_________________________________________________________________
```

Compilation of model

In [ ]:

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

Running epochs

In [ ]:

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 1/20
180/180 [==============================] - 32s 150ms/step - loss: 1.0664 - accuracy:
0.5279
Epoch 2/20
180/180 [==============================] - 26s 146ms/step - loss: 0.6280 - accuracy:
0.7369
Epoch 3/20
180/180 [==============================] - 26s 144ms/step - loss: 0.4163 - accuracy:
0.8272
Epoch 4/20
180/180 [==============================] - 27s 146ms/step - loss: 0.3101 - accuracy:
0.8812
Epoch 5/20
180/180 [==============================] - 26s 145ms/step - loss: 0.2440 - accuracy:
0.9098
Epoch 6/20
180/180 [==============================] - 26s 143ms/step - loss: 0.1835 - accuracy:
0.9279
Epoch 7/20
180/180 [==============================] - 27s 146ms/step - loss: 0.1506 - accuracy:
0.9449
Epoch 8/20
180/180 [==============================] - 26s 144ms/step - loss: 0.1037 - accuracy:
0.9645
Epoch 9/20
180/180 [==============================] - 26s 143ms/step - loss: 0.1158 - accuracy:
0.9575
Epoch 10/20
180/180 [==============================] - 26s 144ms/step - loss: 0.0707 - accuracy:
0.9749
Epoch 11/20
180/180 [==============================] - 26s 144ms/step - loss: 0.0929 - accuracy:
0.9707
Epoch 12/20
180/180 [==============================] - 26s 144ms/step - loss: 0.0607 - accuracy:
0.9791
Epoch 13/20
180/180 [==============================] - 26s 143ms/step - loss: 0.0563 - accuracy:
0.9826
Epoch 14/20
180/180 [==============================] - 27s 146ms/step - loss: 0.0268 - accuracy:
0.9899
Epoch 15/20
180/180 [==============================] - 26s 144ms/step - loss: 0.0733 - accuracy:
0.9732
Epoch 16/20
180/180 [==============================] - 26s 143ms/step - loss: 0.0340 - accuracy:
0.9916
Epoch 17/20
180/180 [==============================] - 26s 145ms/step - loss: 0.0100 - accuracy:
0.9972
Epoch 18/20
180/180 [==============================] - 26s 144ms/step - loss: 0.0684 - accuracy:
0.9784
Epoch 19/20
180/180 [==============================] - 26s 142ms/step - loss: 0.0241 - accuracy:
0.9930
Epoch 20/20
180/180 [==============================] - 26s 145ms/step - loss: 0.0140 - accuracy:
0.9937
```

Checking scores

In [ ]:

```
scores = model.evaluate(test_ds)
```

```
25/25 [==============================] - 3s 94ms/step - loss: 6.0241 - accuracy: 0.74
37
```

Visualising the scores

In [ ]:

```
history.history.keys()
```

Out[ ]:

```
dict_keys(['loss', 'accuracy'])
```

In [ ]:

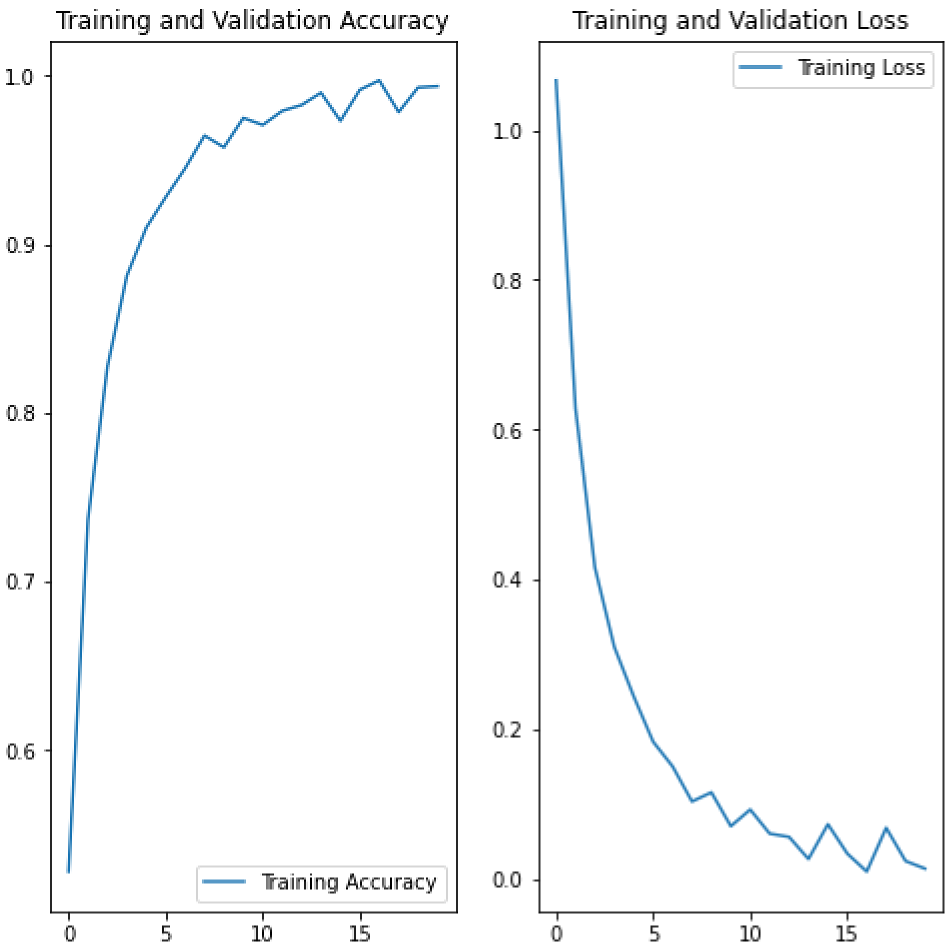
```
acc = history.history['accuracy']

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

In [ ]:

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training 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.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Testing the model

In [ ]:

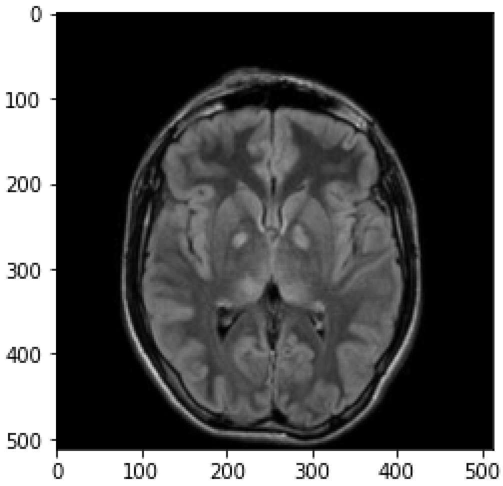
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    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: no_tumor
predicted label: no_tumor
```



In [ ]:

```
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
```

Testing on test dataset

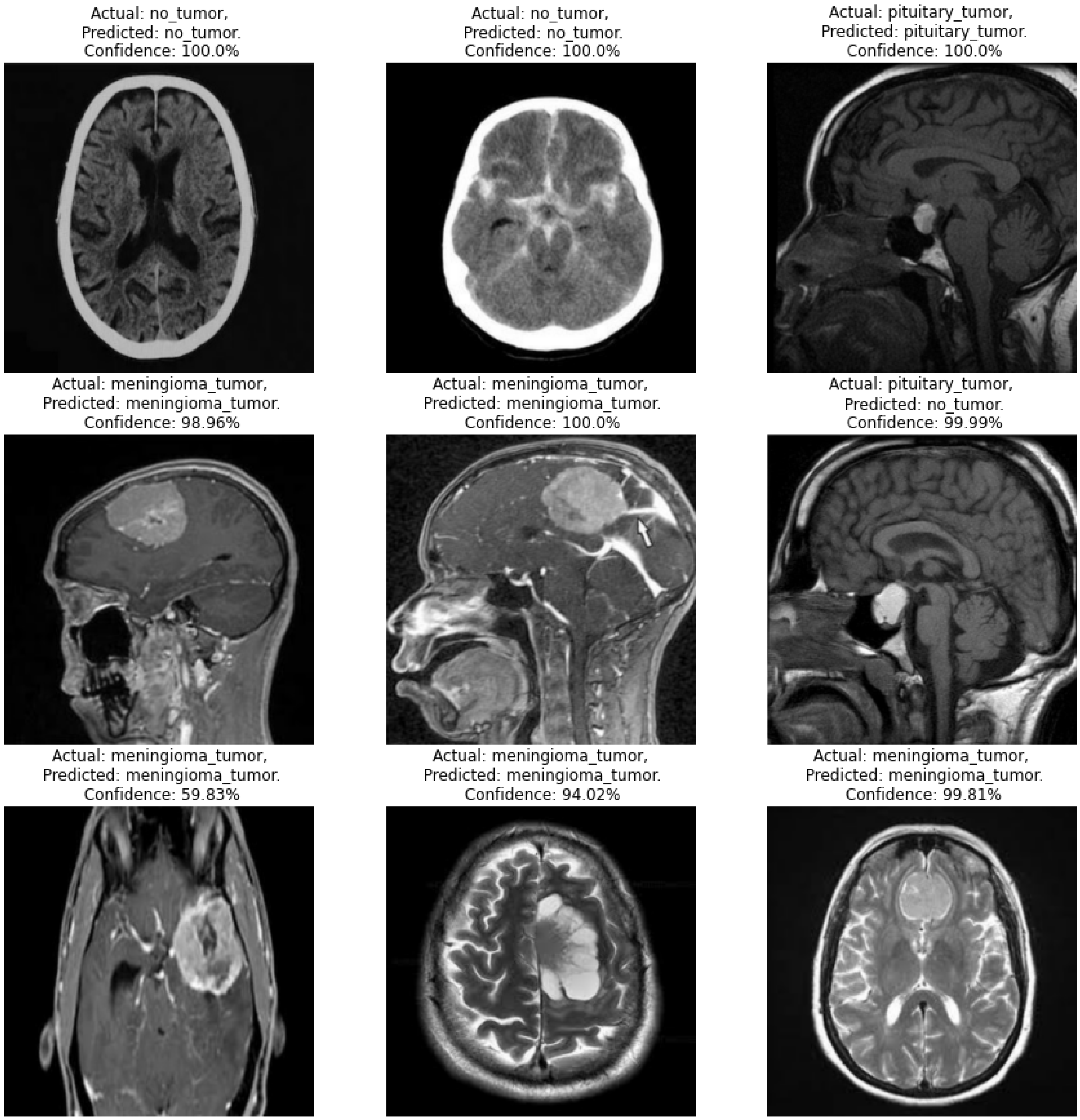
In [ ]:

```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(50):
        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 Confider

        plt.axis("off")
```



Saving the model

In [ ]:

```
model.save("drive/MyDrive/1")
```

```
INFO:tensorflow:Assets written to: drive/MyDrive/1/assets
INFO:tensorflow:Assets written to: drive/MyDrive/1/assets
```

In [ ]:

```
model.save("content/1")
```

```
INFO:tensorflow:Assets written to: content/1/assets
INFO:tensorflow:Assets written to: content/1/assets
```

Notes:

- Reference for this project was taken from this playlist and this github repo
- I used this Dataset