Team 47 Brain tumor classification Final

January 22, 2022

1 Image classification for brain tumours

This method shows how Team 47 including Tendai Gwanzura, Angel Zelazny, Antonio Hernandez classify images of brain tumours. This first step is to unzip the file containing images in the notebook directory.

[1]:
```
import zipfile as zf
files = zf.ZipFile("brain_tumor_dataset.zip", 'r')
files.extractall('directory to extract')
files.close()
```

1.1 Import TensorFlow and other libraries to be used in creating model and analysis

[26]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

1.2 Explore the dataset and prepare it for processing

[27]:
```
pwd
```

[27]:
```
'C:\\Users\\tenda\\Downloads'
```

[28]:
```
import pathlib
```

[29]:
```
data_dir = pathlib.Path('brain_tumor_dataset')
```

[30]:
```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

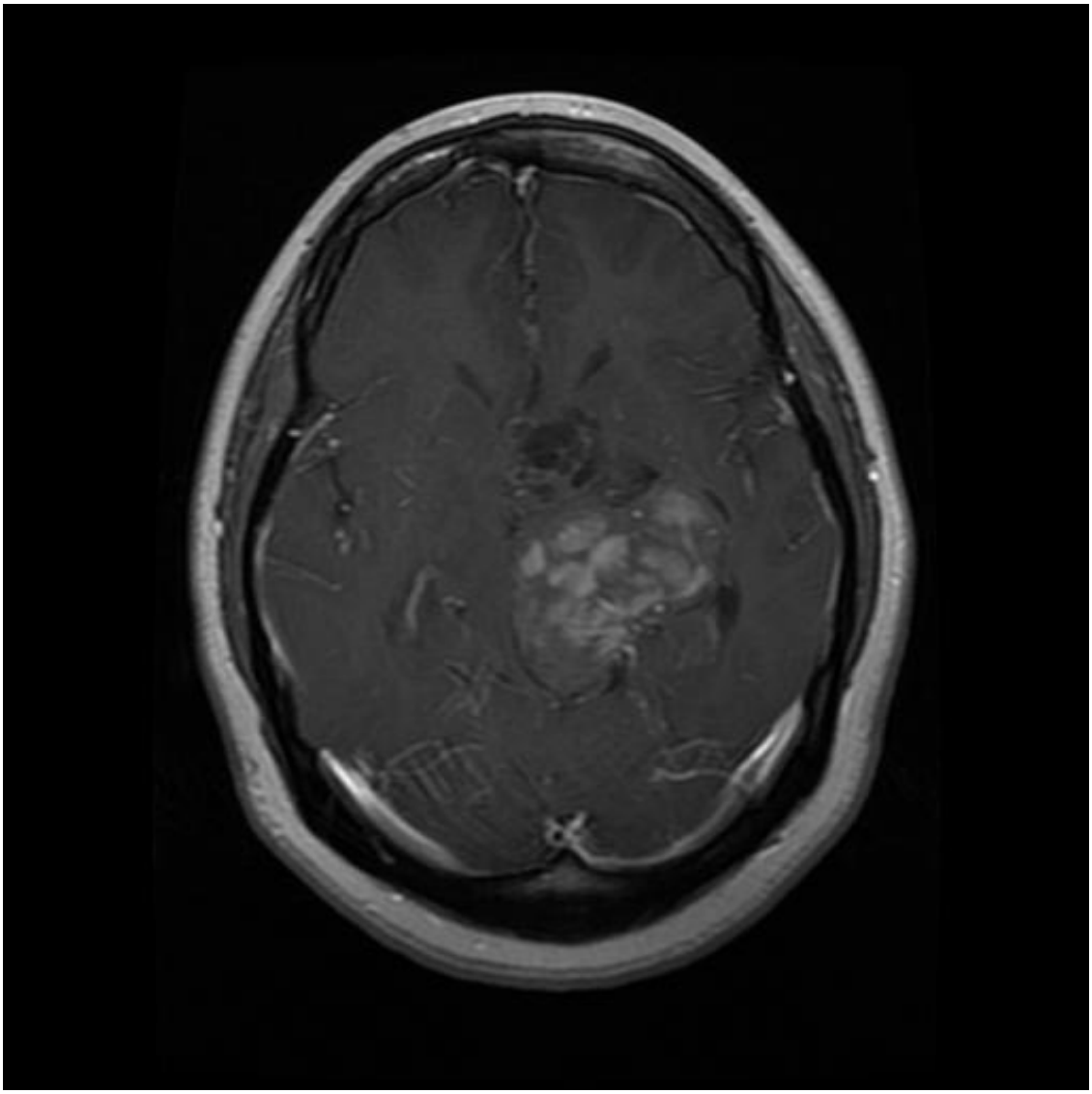
```
6115
```

After uploading the file there are 6115 total images in the dataset divided into yes and no (brain tumors) categories.

The yes and no categories are defined into yeses and no and some of the images are visualized in the following steps.

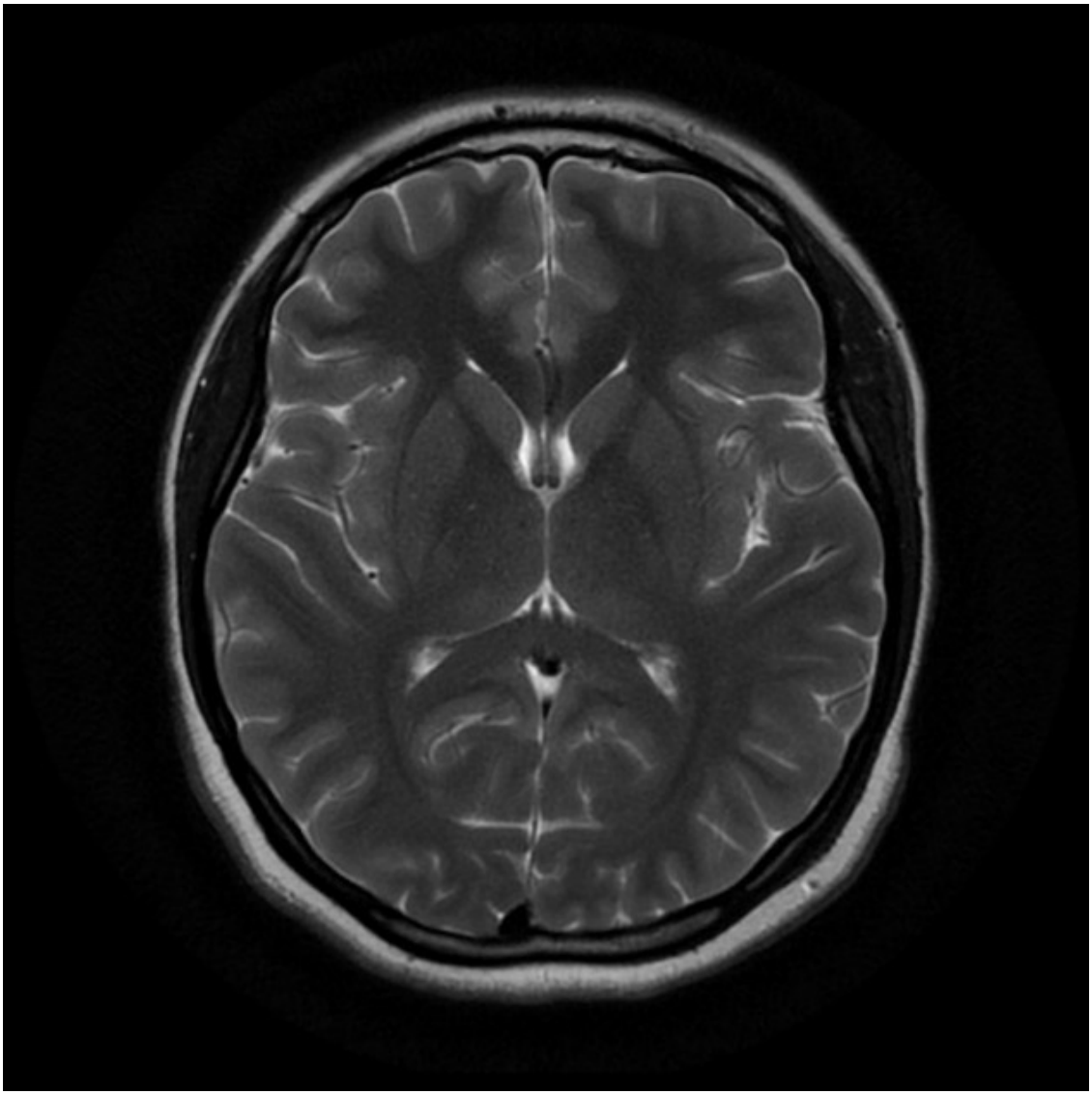
[31]:
```
yeses = list(data_dir.glob('yes/*'))
PIL.Image.open(str(yeses[0]))
```

[31]:



[32]:
```
nos = list(data_dir.glob('no/*'))
PIL.Image.open(str(nos[0]))
```

[32]:



1.3 In the following steps we will create and bifurcate the dataset into training and testing sets.

From the project scoping the following are the defined parameters for the images.

[33]:
```
batch_size = 32
img_height = 180
img_width = 180
```

For our validation split we are using 80% of the images for training, and 20% for validation.

[34]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
```

```
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 6123 files belonging to 2 classes.
Using 4899 files for training.
```

[35]:
```
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 6123 files belonging to 2 classes.
Using 1224 files for validation.
```

The direcetory is divided into yes and no which are defined as `class_names`.

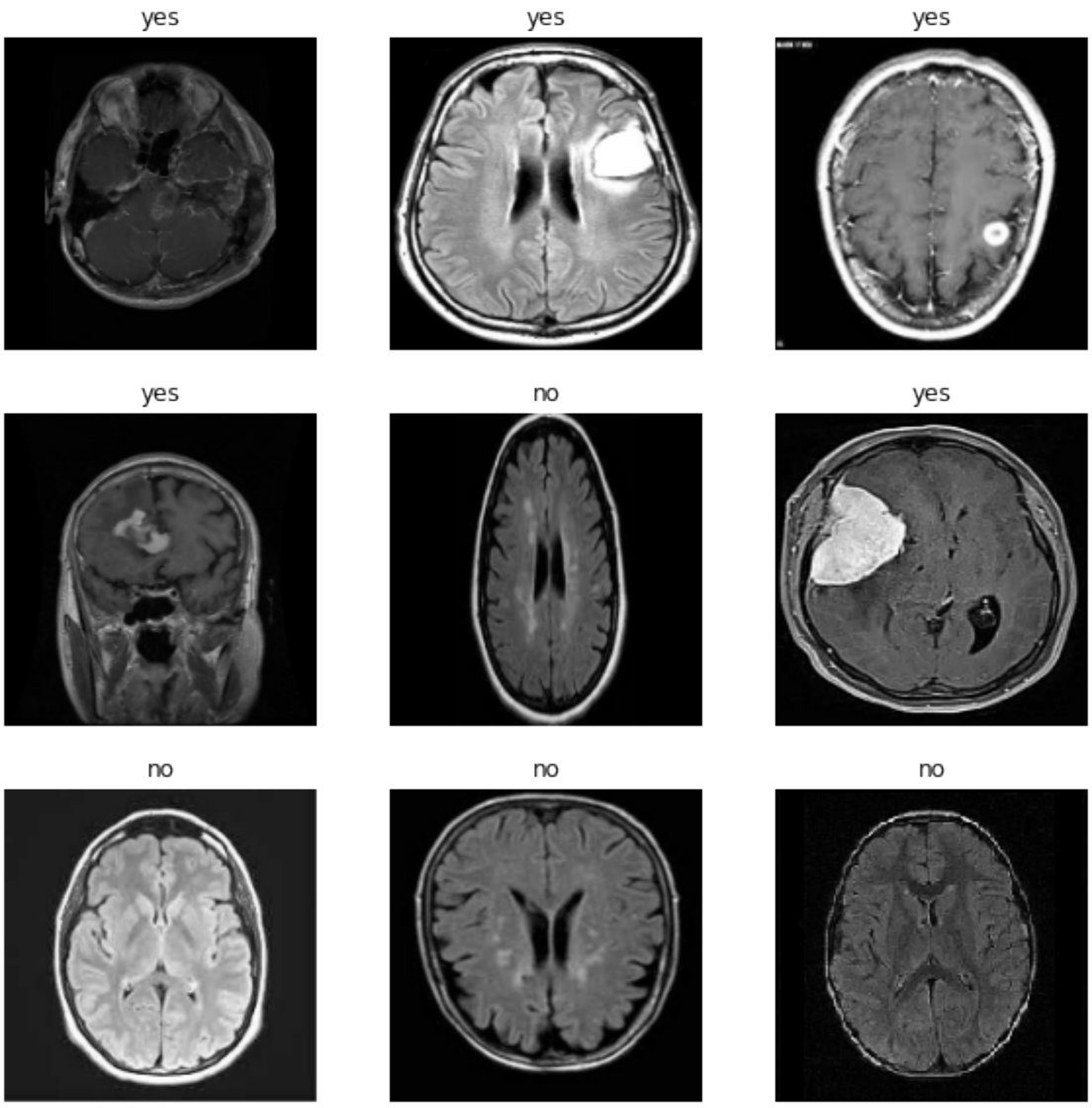
[36]:
```
class_names = train_ds.class_names
print(class_names)
```

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

1.4 The following method will visualize some of the data

The following are a sample of 9 brain images with and without tumors.

[37]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



[38]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

1.5 To make our input values small we will standardize the data to allow neural network to function smoothly.

In this step we will homogenize the image values to be in the `[0, 1]` range by using `tf.keras.layers.Rescaling`:

[39]:
```
normalization_layer = layers.Rescaling(1./255)
```

[40]:
```
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

2 In the following we are create the model to classify tumors

There are three convolution blocks in this tensor flow method which consist of (`tf.keras.layers.Conv2D`) with a max pooling layer (`tf.keras.layers.MaxPooling2D`) in each of them. The `tf.keras.layers.Dense` fuctioning layer with 128 units on top of it is operated by a ReLU activation function (`'relu'`).

[41]:
```
num_classes = len(class_names)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

2.1 In the first step we will compile the model

We are using the `tf.keras.optimizers.Adam` optimizer and `tf.keras.losses.SparseCategoricalCrossentropy` loss function. For each training epoch we will view training and validation accuracy and pass the `metrics` argument to `Model.compile`.

[42]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

2.2 We use 'Model.summary' to see all the layers of the network to give the model summary.

[43]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_3 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 flatten_1 (Flatten)         (None, 30976)             0

 dense_2 (Dense)             (None, 128)               3965056

 dense_3 (Dense)             (None, 2)                 258

=================================================================
Total params: 3,988,898
Trainable params: 3,988,898
Non-trainable params: 0
_________________________________________________________________
```

2.3 In this step we are now training the model.

[44]:
```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

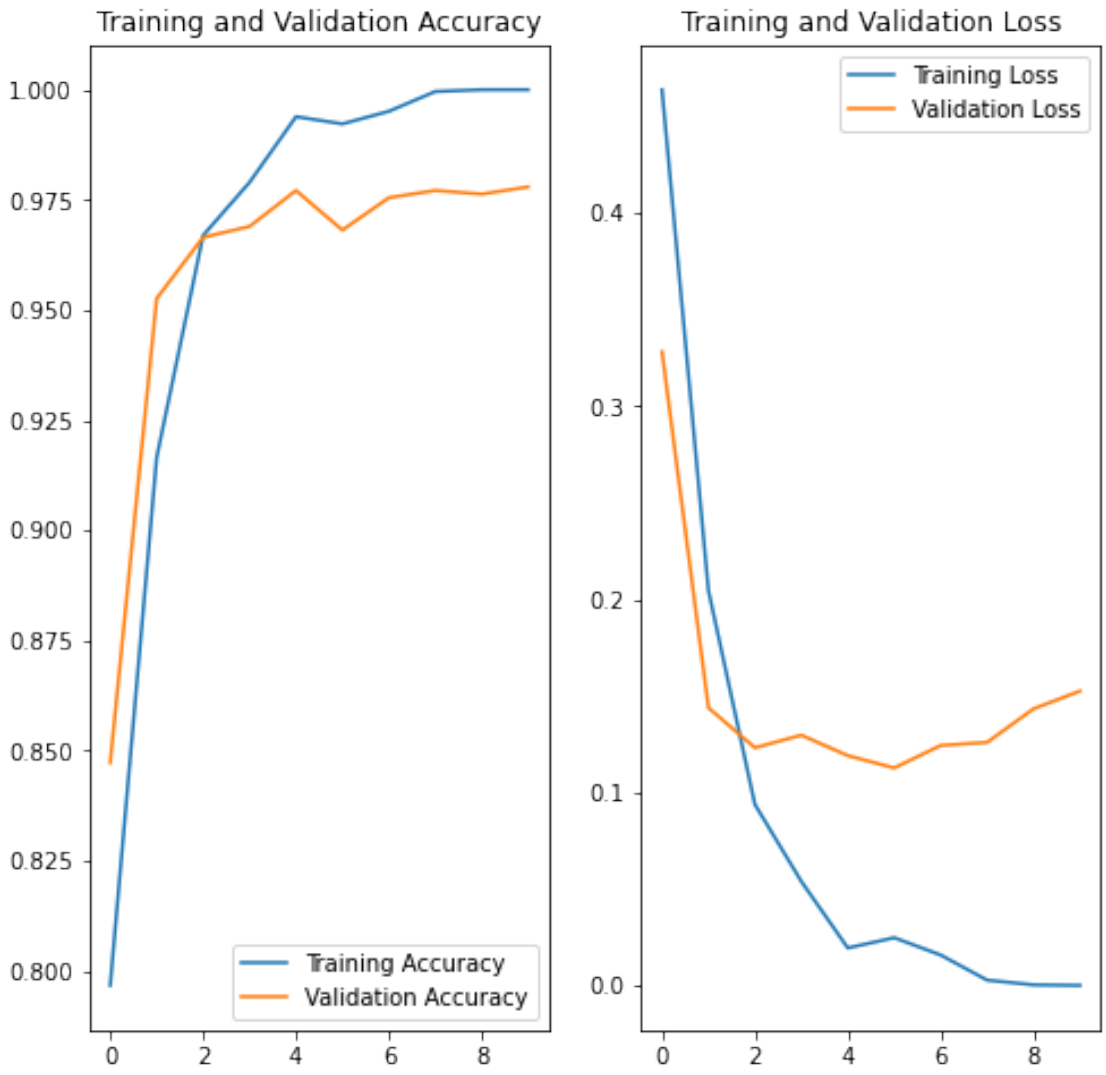
```
Epoch 1/10
154/154 [==============================] - 197s 1s/step - loss: 0.4638 -
accuracy: 0.7967 - val_loss: 0.3282 - val_accuracy: 0.8472
Epoch 2/10
154/154 [==============================] - 185s 1s/step - loss: 0.2046 -
accuracy: 0.9165 - val_loss: 0.1439 - val_accuracy: 0.9526
```

```
Epoch 3/10
154/154 [==============================] - 183s 1s/step - loss: 0.0938 -
accuracy: 0.9669 - val_loss: 0.1232 - val_accuracy: 0.9665
Epoch 4/10
154/154 [==============================] - 173s 1s/step - loss: 0.0540 -
accuracy: 0.9790 - val_loss: 0.1297 - val_accuracy: 0.9690
Epoch 5/10
154/154 [==============================] - 187s 1s/step - loss: 0.0195 -
accuracy: 0.9939 - val_loss: 0.1191 - val_accuracy: 0.9771
Epoch 6/10
154/154 [==============================] - 211s 1s/step - loss: 0.0248 -
accuracy: 0.9922 - val_loss: 0.1127 - val_accuracy: 0.9681
Epoch 7/10
154/154 [==============================] - 209s 1s/step - loss: 0.0159 -
accuracy: 0.9951 - val_loss: 0.1244 - val_accuracy: 0.9755
Epoch 8/10
154/154 [==============================] - 184s 1s/step - loss: 0.0029 -
accuracy: 0.9996 - val_loss: 0.1260 - val_accuracy: 0.9771
Epoch 9/10
154/154 [==============================] - 172s 1s/step - loss: 4.2783e-04 -
accuracy: 1.0000 - val_loss: 0.1434 - val_accuracy: 0.9763
Epoch 10/10
154/154 [==============================] - 165s 1s/step - loss: 1.6057e-04 -
accuracy: 1.0000 - val_loss: 0.1525 - val_accuracy: 0.9779
```

2.4 We are visualizing training results by creating plots of accuracy on the training and validation sets.

[45]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
```

```
plt.title('Training and Validation Loss')
plt.show()
```



The plots show that training accuracy and validation accuracy have small margins, and the model has achieved only around 98% accuracy on the validation set.

2.5 We can now use other images to see whether our model can predict on new data(Testing the model)

This image was not included in the training and validation sets obttained from the Kaggle testing dataset.It is an image of a pituitary tumour and should belong to the yes category.

[57]:
```
testing_path = pathlib.Path('TestingImage.jpg')
```

[58]:
```
img = tf.keras.utils.load_img(
    testing_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to yes with a 100.00 percent confidence.
```

[ ]:
```
Our model can accurately identify tumors and is ready for real world testing.
```