Classification of MRI images using CNN for tumour detection

(Part 1 - Running the models)

We will start by downloading the dataset from the Github repository and unzipping the contents of the repository into our local directory.

```
!wget https://github.com/SartajBhuvaji/Brain-Tumor-Classification-DataSet/archive/refs/hea
!unzip -q master.zip
```

```
--2022-02-15 00:48:37--  https://github.com/SartajBhuvaji/Brain-Tumor-Classification-
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/SartajBhuvaji/Brain-Tumor-Classification-DataSe
--2022-02-15 00:48:37--  https://codeload.github.com/SartajBhuvaji/Brain-Tumor-Classi
Resolving codeload.github.com (codeload.github.com)... 140.82.114.9
Connecting to codeload.github.com (codeload.github.com)|140.82.114.9|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: ‘master.zip’

master.zip              [              <=> ]  86.97M  25.9MB/s    in 3.4s

2022-02-15 00:48:41 (25.3 MB/s) - ‘master.zip’ saved [91198591]
```

Let us now set the path to the directories containing the training and the validation images.

```
import pandas as pd
TRAIN_PATH = "./Brain-Tumor-Classification-DataSet-master/Training/"
TEST__PATH = "./Brain-Tumor-Classification-DataSet-master/Testing/"
```

We will now set the batch and the image size for the models.

```
import tensorflow as tf
batch_size = 32
img_height = 256
img_width = 256
```

We will now create the input pipeline Tensorflow-compatible dataset using the images in the training directory.

```
training_dataset = tf.keras.utils.image_dataset_from_directory(
  TRAIN_PATH,
  seed=0,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

Let us similarly create a dataset using the images in the validation directory.

```
testing_dataset = tf.keras.utils.image_dataset_from_directory(
  TEST__PATH,
  seed=0,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

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

Let us view the names of the classes in the dataset(s).

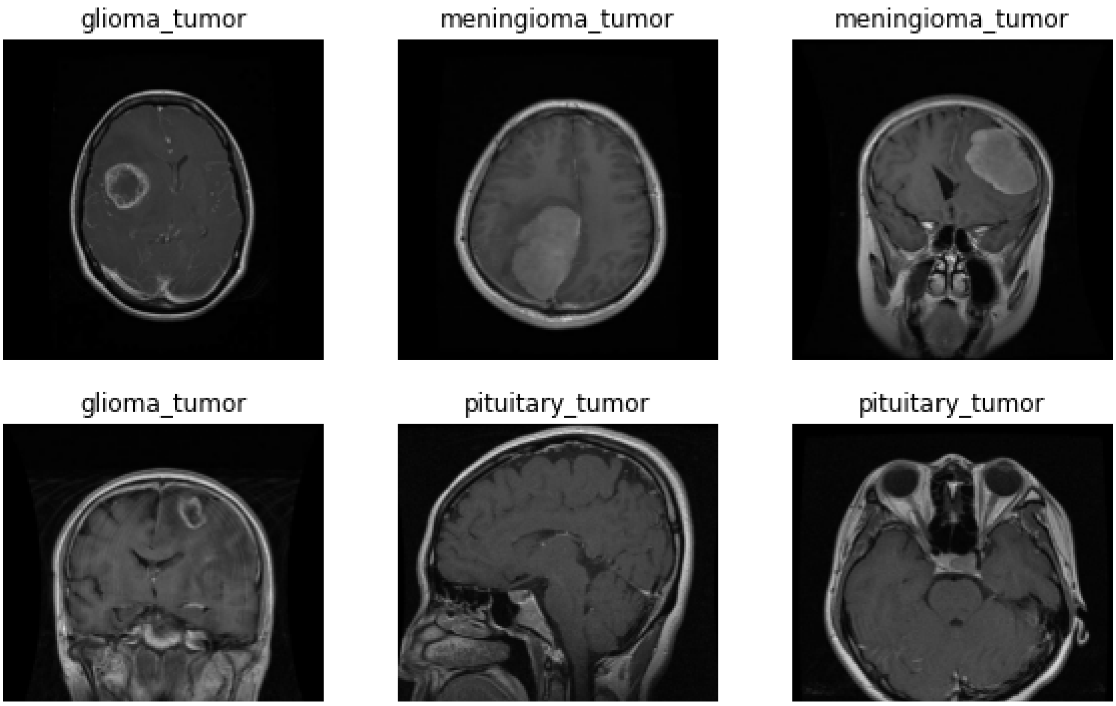
```
classes = training_dataset.class_names
print(classes)
```

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

For our own visualization, let us plot a few of the images, along with their labels, from the dataset.

```
import matplotlib.pyplot as plt

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



Let us configure Tensorflow to enable buffering for efficiency.

```
AUTOTUNE = tf.data.AUTOTUNE
training_dataset = training_dataset.cache().prefetch(buffer_size=AUTOTUNE)
testing_dataset = testing_dataset.cache().prefetch(buffer_size=AUTOTUNE)
```

Let us define a callback for early stopping so that the training can be automatically stopped in case there is no decrease in validation loss.

We will add a tolerance of five (5) eopchs so that the training does not terminate in case the model is stuck in a plateau.

```
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=
```

Let us create a variable to indicate the maximum number of epochs allowed for training the models.

```
MAX_EPOCHS = 100
```

Model 1 - Basic Model

Let us define and build the basic model

```
basic_model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
```

```
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(4)
])

basic_model.build(input_shape=(None,256,256,3))
basic_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 32)       0
 2D)

 flatten (Flatten)           (None, 28800)             0

 dense (Dense)               (None, 128)               3686528

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 3,706,436
Trainable params: 3,706,436
Non-trainable params: 0
_________________________________________________________________
```

We will now compile and train the model.

```
basic_model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

basic_history = basic_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
```

```
    callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 21s 117ms/step - loss: 0.8842 - accuracy: 0
Epoch 2/100
90/90 [==============================] - 8s 84ms/step - loss: 0.4886 - accuracy: 0.79
Epoch 3/100
90/90 [==============================] - 8s 85ms/step - loss: 0.2971 - accuracy: 0.87
Epoch 4/100
90/90 [==============================] - 8s 84ms/step - loss: 0.1683 - accuracy: 0.93
Epoch 5/100
90/90 [==============================] - 8s 86ms/step - loss: 0.1074 - accuracy: 0.96
Epoch 6/100
90/90 [==============================] - 8s 84ms/step - loss: 0.0642 - accuracy: 0.97
Epoch 7/100
90/90 [==============================] - 8s 84ms/step - loss: 0.0407 - accuracy: 0.98
Epoch 8/100
90/90 [==============================] - 8s 85ms/step - loss: 0.0235 - accuracy: 0.99
Epoch 9/100
90/90 [==============================] - 8s 85ms/step - loss: 0.0089 - accuracy: 0.99
Epoch 10/100
90/90 [==============================] - 8s 85ms/step - loss: 0.0073 - accuracy: 0.99
Epoch 11/100
90/90 [==============================] - 8s 85ms/step - loss: 0.0096 - accuracy: 0.99
Epoch 12/100
90/90 [==============================] - 8s 84ms/step - loss: 0.0078 - accuracy: 0.99
Epoch 00012: early stopping
```

Save the model and training history

```
basic_model.save("/content/drive/MyDrive/mri-cnn/basic")
pd.DataFrame(basic_history.history).to_csv("/content/drive/MyDrive/mri-cnn/basic.csv")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/basic/assets
```

Model 2 - ResNet50

Let us import the ResNet50 model.

```
resnet50_model = tf.keras.applications.ResNet50(include_top=False, weights="imagenet", inp
resnet50_model.summary()
```

```
 conv5_block1_out (Activation)  (None, 8, 8, 2048)   0           ['conv5_block1_ad

 conv5_block2_1_conv (Conv2D)   (None, 8, 8, 512)    1049088     ['conv5_block1_ou

 conv5_block2_1_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block2_1_
 ization)

 conv5_block2_1_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block2_1_
 n)
```

```
 conv5_block2_2_conv (Conv2D)   (None, 8, 8, 512)    2359808     ['conv5_block2_1_

 conv5_block2_2_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block2_2_
 ization)

 conv5_block2_2_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block2_2_
 n)

 conv5_block2_3_conv (Conv2D)   (None, 8, 8, 2048)   1050624     ['conv5_block2_2_

 conv5_block2_3_bn (BatchNormal  (None, 8, 8, 2048)  8192        ['conv5_block2_3_
 ization)

 conv5_block2_add (Add)         (None, 8, 8, 2048)   0           ['conv5_block1_ou
                                                                  'conv5_block2_3_

 conv5_block2_out (Activation)  (None, 8, 8, 2048)   0           ['conv5_block2_ad

 conv5_block3_1_conv (Conv2D)   (None, 8, 8, 512)    1049088     ['conv5_block2_ou

 conv5_block3_1_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block3_1_
 ization)

 conv5_block3_1_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block3_1_
 n)

 conv5_block3_2_conv (Conv2D)   (None, 8, 8, 512)    2359808     ['conv5_block3_1_

 conv5_block3_2_bn (BatchNormal  (None, 8, 8, 512)   2048        ['conv5_block3_2_
 ization)

 conv5_block3_2_relu (Activatio  (None, 8, 8, 512)   0           ['conv5_block3_2_
 n)

 conv5_block3_3_conv (Conv2D)   (None, 8, 8, 2048)   1050624     ['conv5_block3_2_

 conv5_block3_3_bn (BatchNormal  (None, 8, 8, 2048)  8192        ['conv5_block3_3_
 ization)

 conv5_block3_add (Add)         (None, 8, 8, 2048)   0           ['conv5_block2_ou
                                                                  'conv5_block3_3_

 conv5_block3_out (Activation)  (None, 8, 8, 2048)   0           ['conv5_block3_ad

==================================================================================
Total params: 23,587,712
```

We will flatten the final layer and then add our output layer with four (4) nodes.

```
res_model = tf.keras.models.Sequential()
res_model.add(resnet50_model)
res_model.add(tf.keras.layers.Flatten())
res_model.add(tf.keras.layers.Dense(4))
res_model.summary()
```

```
Model: "sequential_1"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten_1 (Flatten)         (None, 131072)            0

 dense_2 (Dense)             (None, 4)                 524292

=================================================================
Total params: 24,112,004
Trainable params: 24,058,884
Non-trainable params: 53,120
_________________________________________________________________
```

We can finally compile and train the ResNet50 model.

```
res_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
res_history = res_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 88s 871ms/step - loss: 4.7750 - accuracy: 0
Epoch 2/100
90/90 [==============================] - 74s 824ms/step - loss: 2.5800 - accuracy: 0
Epoch 3/100
90/90 [==============================] - 74s 821ms/step - loss: 1.3020 - accuracy: 0
Epoch 4/100
90/90 [==============================] - 74s 823ms/step - loss: 0.8957 - accuracy: 0
Epoch 5/100
90/90 [==============================] - 75s 828ms/step - loss: 0.5563 - accuracy: 0
Epoch 6/100
90/90 [==============================] - 75s 829ms/step - loss: 0.3237 - accuracy: 0
Epoch 7/100
90/90 [==============================] - 75s 831ms/step - loss: 0.3140 - accuracy: 0
Epoch 8/100
90/90 [==============================] - 75s 831ms/step - loss: 0.7986 - accuracy: 0
Epoch 9/100
90/90 [==============================] - 75s 833ms/step - loss: 0.9672 - accuracy: 0
Epoch 10/100
90/90 [==============================] - 75s 830ms/step - loss: 0.7650 - accuracy: 0
Epoch 11/100
90/90 [==============================] - 75s 830ms/step - loss: 0.2536 - accuracy: 0
Epoch 12/100
90/90 [==============================] - 75s 830ms/step - loss: 0.3117 - accuracy: 0
Epoch 13/100
90/90 [==============================] - 75s 830ms/step - loss: 0.1172 - accuracy: 0
Epoch 00013: early stopping
```

Save the model and training history

```
res_model.save("/content/drive/MyDrive/mri-cnn/res")
pd.DataFrame(res_history.history).to_csv("/content/drive/MyDrive/mri-cnn/res.csv")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/res/assets
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWar
  layer_config = serialize_layer_fn(layer)
/usr/local/lib/python3.7/dist-packages/keras/saving/saved_model/layer_serialization.p
  return generic_utils.serialize_keras_object(obj)
```

Model 3 - VGG16

Let us import the VGG16 model.

```
vgg16_model = tf.keras.applications.VGG16(include_top=False, weights="imagenet", input_ten
vgg16_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vg
58892288/58889256 [==============================] - 1s 0us/step
58900480/58889256 [==============================] - 1s 0us/step
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0
```

```
 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808   
                                                                 
 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0         
                                                                 
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Like before, let us add our output layers.

```
vgg_model = tf.keras.models.Sequential()
vgg_model.add(vgg16_model)
vgg_model.add(tf.keras.layers.Flatten())
vgg_model.add(tf.keras.layers.Dense(4))
vgg_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 vgg16 (Functional)          (None, 8, 8, 512)         14714688  
                                                                 
 flatten_2 (Flatten)         (None, 32768)             0         
                                                                 
 dense_3 (Dense)             (None, 4)                 131076    
                                                                 
=================================================================
Total params: 14,845,764
Trainable params: 14,845,764
Non-trainable params: 0
_________________________________________________________________
```

We are now ready to compile and train the network.

```
vgg_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
vgg_history = vgg_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 127s 1s/step - loss: 2.7705 - accuracy: 0.3(
```

```
Epoch 2/100
90/90 [==============================] - 91s 1s/step - loss: 1.0757 - accuracy: 0.528
Epoch 3/100
90/90 [==============================] - 91s 1s/step - loss: 0.9219 - accuracy: 0.616
Epoch 4/100
90/90 [==============================] - 91s 1s/step - loss: 0.8214 - accuracy: 0.663
Epoch 5/100
90/90 [==============================] - 90s 1s/step - loss: 0.6688 - accuracy: 0.743
Epoch 6/100
90/90 [==============================] - 90s 1s/step - loss: 0.5753 - accuracy: 0.777
Epoch 7/100
90/90 [==============================] - 90s 1s/step - loss: 0.5617 - accuracy: 0.776
Epoch 8/100
90/90 [==============================] - 90s 1s/step - loss: 0.5124 - accuracy: 0.802
Epoch 9/100
90/90 [==============================] - 90s 1s/step - loss: 0.4417 - accuracy: 0.832
Epoch 10/100
90/90 [==============================] - 90s 1s/step - loss: 0.3813 - accuracy: 0.853
Epoch 11/100
90/90 [==============================] - 90s 999ms/step - loss: 0.3649 - accuracy: 0
Epoch 12/100
90/90 [==============================] - 90s 999ms/step - loss: 0.3138 - accuracy: 0
Epoch 00012: early stopping
```

Save the model and training history

```
vgg_model.save("/content/drive/MyDrive/mri-cnn/vgg")
pd.DataFrame(vgg_history.history).to_csv("/content/drive/MyDrive/mri-cnn/vgg.csv")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/vgg/assets
```

Model 4 - MobileNet

We will now import the MobileNet architecture.

```
mobilenet_model = tf.keras.applications.MobileNet(include_top=False, weights="imagenet", i
mobilenet_model.summary()
```

```
 conv_pw_10 (Conv2D)         (None, 16, 16, 512)       262144

 conv_pw_10_bn (BatchNormali  (None, 16, 16, 512)      2048
 zation)

 conv_pw_10_relu (ReLU)      (None, 16, 16, 512)       0

 conv_dw_11 (DepthwiseConv2D  (None, 16, 16, 512)      4608
 )

 conv_dw_11_bn (BatchNormali  (None, 16, 16, 512)      2048
 zation)

 conv_dw_11_relu (ReLU)      (None, 16, 16, 512)       0

 conv_pw_11 (Conv2D)         (None, 16, 16, 512)       262144
```

```
 conv_pw_11_bn (BatchNormali  (None, 16, 16, 512)      2048
 zation)

 conv_pw_11_relu (ReLU)      (None, 16, 16, 512)      0

 conv_pad_12 (ZeroPadding2D)  (None, 17, 17, 512)     0

 conv_dw_12 (DepthwiseConv2D  (None, 8, 8, 512)       4608
 )

 conv_dw_12_bn (BatchNormali  (None, 8, 8, 512)       2048
 zation)

 conv_dw_12_relu (ReLU)      (None, 8, 8, 512)        0

 conv_pw_12 (Conv2D)         (None, 8, 8, 1024)       524288

 conv_pw_12_bn (BatchNormali  (None, 8, 8, 1024)      4096
 zation)

 conv_pw_12_relu (ReLU)      (None, 8, 8, 1024)       0

 conv_dw_13 (DepthwiseConv2D  (None, 8, 8, 1024)      9216
 )

 conv_dw_13_bn (BatchNormali  (None, 8, 8, 1024)      4096
 zation)

 conv_dw_13_relu (ReLU)      (None, 8, 8, 1024)       0

 conv_pw_13 (Conv2D)         (None, 8, 8, 1024)       1048576

 conv_pw_13_bn (BatchNormali  (None, 8, 8, 1024)      4096
 zation)

 conv_pw_13_relu (ReLU)      (None, 8, 8, 1024)       0

=================================================================
Total params: 3,228,864
Trainable params: 3,206,976
```

Let us put our application-specific output layer into the imported architecture.

```
mobile_model = tf.keras.models.Sequential()
mobile_model.add(mobilenet_model)
mobile_model.add(tf.keras.layers.Flatten())
mobile_model.add(tf.keras.layers.Dense(4))
mobile_model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenet_1.00_224 (Functio  (None, 8, 8, 1024)      3228864
 nal)
```

```
 flatten_3 (Flatten)         (None, 65536)             0

 dense_4 (Dense)             (None, 4)                 262148

=================================================================
Total params: 3,491,012
Trainable params: 3,469,124
Non-trainable params: 21,888
_________________________________________________________________
```

We can now compile and train our network.

```
mobile_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
mobile_history = mobile_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 42s 427ms/step - loss: 3.4591 - accuracy: 0
Epoch 2/100
90/90 [==============================] - 37s 409ms/step - loss: 0.3927 - accuracy: 0
Epoch 3/100
90/90 [==============================] - 37s 408ms/step - loss: 0.1201 - accuracy: 0
Epoch 4/100
90/90 [==============================] - 37s 409ms/step - loss: 0.0469 - accuracy: 0
Epoch 5/100
90/90 [==============================] - 37s 407ms/step - loss: 0.0358 - accuracy: 0
Epoch 6/100
90/90 [==============================] - 37s 409ms/step - loss: 0.2890 - accuracy: 0
Epoch 7/100
90/90 [==============================] - 37s 410ms/step - loss: 0.4403 - accuracy: 0
Epoch 8/100
90/90 [==============================] - 37s 407ms/step - loss: 0.4581 - accuracy: 0
Epoch 9/100
90/90 [==============================] - 37s 409ms/step - loss: 0.0878 - accuracy: 0
Epoch 10/100
90/90 [==============================] - 37s 409ms/step - loss: 0.0662 - accuracy: 0
Epoch 11/100
90/90 [==============================] - 37s 409ms/step - loss: 0.0534 - accuracy: 0
Epoch 12/100
90/90 [==============================] - 37s 409ms/step - loss: 0.0249 - accuracy: 0
Epoch 13/100
90/90 [==============================] - 37s 408ms/step - loss: 0.0068 - accuracy: 0
Epoch 14/100
90/90 [==============================] - 37s 408ms/step - loss: 0.0104 - accuracy: 0
Epoch 15/100
90/90 [==============================] - 37s 412ms/step - loss: 0.0499 - accuracy: 0
Epoch 16/100
90/90 [==============================] - 37s 409ms/step - loss: 0.2978 - accuracy: 0
Epoch 17/100
90/90 [==============================] - 37s 409ms/step - loss: 0.2518 - accuracy: 0
Epoch 18/100
```

```
90/90 [==============================] - 37s 408ms/step - loss: 0.2289 - accuracy: 0
Epoch 19/100
90/90 [==============================] - 37s 408ms/step - loss: 0.0580 - accuracy: 0
Epoch 20/100
90/90 [==============================] - 37s 407ms/step - loss: 0.0428 - accuracy: 0
Epoch 21/100
90/90 [==============================] - 37s 407ms/step - loss: 0.0124 - accuracy: 0
Epoch 22/100
90/90 [==============================] - 37s 408ms/step - loss: 0.0136 - accuracy: 0
Epoch 00022: early stopping
```

Save the model and training history

```
mobile_model.save("/content/drive/MyDrive/mri-cnn/mobile")
pd.DataFrame(mobile_history.history).to_csv("/content/drive/MyDrive/mri-cnn/mobile.csv")
```

```
WARNING:absl:Function `_wrapped_model` contains input name(s) mobilenet_1.00_224_inpu
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/mobile/assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/mobile/assets
```

Model 5 - Inception v3

Let us load the InceptionNetv3 model.

```
inceptionnet_model = tf.keras.applications.InceptionV3(include_top=False, weights="imagene
inceptionnet_model.summary()
```

```
 conv2d_90 (Conv2D)             (None, 6, 6, 384)    442368      ['activation_86[0

 conv2d_91 (Conv2D)             (None, 6, 6, 384)    442368      ['activation_86[0

 conv2d_94 (Conv2D)             (None, 6, 6, 384)    442368      ['activation_90[0

 conv2d_95 (Conv2D)             (None, 6, 6, 384)    442368      ['activation_90[0

 average_pooling2d_8 (AveragePo  (None, 6, 6, 2048)  0           ['mixed9[0][0]']
 oling2D)

 conv2d_88 (Conv2D)             (None, 6, 6, 320)    655360      ['mixed9[0][0]']

 batch_normalization_87 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_90[0][0]
 ormalization)

 batch_normalization_88 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_91[0][0]
 ormalization)

 batch_normalization_91 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_94[0][0]
 ormalization)

 batch_normalization_92 (BatchN  (None, 6, 6, 384)   1152        ['conv2d_95[0][0]
 ormalization)

 conv2d_96 (Conv2D)             (None, 6, 6, 192)    393216      ['average_pooling
```

```
 conv2d_96 (Conv2D)             (None, 6, 6, 192)    553216      ['average_pooling

 batch_normalization_85 (BatchN  (None, 6, 6, 320)   960         ['conv2d_88[0][0]
 ormalization)

 activation_87 (Activation)     (None, 6, 6, 384)    0           ['batch_normaliza

 activation_88 (Activation)     (None, 6, 6, 384)    0           ['batch_normaliza

 activation_91 (Activation)     (None, 6, 6, 384)    0           ['batch_normaliza

 activation_92 (Activation)     (None, 6, 6, 384)    0           ['batch_normaliza

 batch_normalization_93 (BatchN  (None, 6, 6, 192)   576         ['conv2d_96[0][0]
 ormalization)

 activation_85 (Activation)     (None, 6, 6, 320)    0           ['batch_normaliza

 mixed9_1 (Concatenate)         (None, 6, 6, 768)    0           ['activation_87[0
                                                                  'activation_88[0

 concatenate_1 (Concatenate)    (None, 6, 6, 768)    0           ['activation_91[0
                                                                  'activation_92[0

 activation_93 (Activation)     (None, 6, 6, 192)    0           ['batch_normaliza

 mixed10 (Concatenate)          (None, 6, 6, 2048)   0           ['activation_85[0
                                                                  'mixed9_1[0][0]'
                                                                  'concatenate_1[0
                                                                  'activation_93[0

==================================================================================
```

As before, we will add our application-specific output layer with four (4) nodes.

```
inception_model = tf.keras.models.Sequential()
inception_model.add(inceptionnet_model)
inception_model.add(tf.keras.layers.Flatten())
inception_model.add(tf.keras.layers.Dense(4))
inception_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 flatten_4 (Flatten)         (None, 73728)             0

 dense_5 (Dense)             (None, 4)                 294916

=================================================================
Total params: 22,097,700
Trainable params: 22,063,268
Non-trainable params: 34,432
_________________________________________________________________
```

We are ready to compile and train the neural network.

```
inception_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
inception_history = inception_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 86s 821ms/step - loss: 2.1325 - accuracy: 0
Epoch 2/100
90/90 [==============================] - 67s 746ms/step - loss: 2.7610 - accuracy: 0
Epoch 3/100
90/90 [==============================] - 67s 745ms/step - loss: 0.8254 - accuracy: 0
Epoch 4/100
90/90 [==============================] - 67s 747ms/step - loss: 0.6639 - accuracy: 0
Epoch 5/100
90/90 [==============================] - 67s 744ms/step - loss: 0.4724 - accuracy: 0
Epoch 6/100
90/90 [==============================] - 67s 745ms/step - loss: 0.3130 - accuracy: 0
Epoch 7/100
90/90 [==============================] - 67s 747ms/step - loss: 0.1806 - accuracy: 0
Epoch 8/100
90/90 [==============================] - 67s 747ms/step - loss: 0.1599 - accuracy: 0
Epoch 9/100
90/90 [==============================] - 67s 745ms/step - loss: 0.1022 - accuracy: 0
Epoch 10/100
90/90 [==============================] - 67s 748ms/step - loss: 0.0704 - accuracy: 0
Epoch 11/100
90/90 [==============================] - 67s 746ms/step - loss: 0.0675 - accuracy: 0
Epoch 12/100
90/90 [==============================] - 67s 746ms/step - loss: 0.0752 - accuracy: 0
Epoch 13/100
90/90 [==============================] - 67s 746ms/step - loss: 0.0675 - accuracy: 0
Epoch 14/100
90/90 [==============================] - 67s 748ms/step - loss: 0.0484 - accuracy: 0
Epoch 15/100
90/90 [==============================] - 67s 748ms/step - loss: 0.0552 - accuracy: 0
Epoch 16/100
90/90 [==============================] - 67s 748ms/step - loss: 0.0590 - accuracy: 0
Epoch 17/100
90/90 [==============================] - 67s 746ms/step - loss: 0.0286 - accuracy: 0
Epoch 00017: early stopping
```

Save the model and training history

```
inception_model.save("/content/drive/MyDrive/mri-cnn/inception")
pd.DataFrame(inception_history.history).to_csv("/content/drive/MyDrive/mri-cnn/inception.c
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/inception/assets
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/inception/assets
```

Model 6 - AlexNet

The Tensorflow core applications have no prebuilt model for AlexNet.

We will build the model on our own.

```
alexnet_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='rel
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(4096, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])

alexnet_model.build(input_shape=(None,256,256,3))
alexnet_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_97 (Conv2D)          (None, 62, 62, 96)        34944

 batch_normalization_94 (Bat  (None, 62, 62, 96)       384
 chNormalization)

 max_pooling2d_7 (MaxPooling  (None, 30, 30, 96)       0
 2D)

 conv2d_98 (Conv2D)          (None, 30, 30, 256)       614656

 batch_normalization_95 (Bat  (None, 30, 30, 256)      1024
 chNormalization)

 max_pooling2d_8 (MaxPooling  (None, 14, 14, 256)      0
 2D)

 conv2d_99 (Conv2D)          (None, 14, 14, 384)       885120
```

```
 batch_normalization_96 (Bat  (None, 14, 14, 384)      1536
 chNormalization)

 conv2d_100 (Conv2D)         (None, 14, 14, 384)       1327488

 batch_normalization_97 (Bat  (None, 14, 14, 384)      1536
 chNormalization)

 conv2d_101 (Conv2D)         (None, 14, 14, 256)       884992

 batch_normalization_98 (Bat  (None, 14, 14, 256)      1024
 chNormalization)

 max_pooling2d_9 (MaxPooling  (None, 6, 6, 256)        0
 2D)

 flatten_5 (Flatten)         (None, 9216)              0

 dense_6 (Dense)             (None, 4096)              37752832

 dropout (Dropout)           (None, 4096)              0

 dense_7 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_8 (Dense)             (None, 10)                40970

=================================================================
Total params: 58,327,818
Trainable params: 58,325,066
Non-trainable params: 2,752
_________________________________________________________________
```

Let us compile and train the AlexNet model.

```
alexnet_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
alexnet_history = alexnet_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 5/100
90/90 [==============================] - 11s 126ms/step - loss: 1.1048 - accuracy:
Epoch 6/100
90/90 [==============================] - 11s 126ms/step - loss: 1.1182 - accuracy:
Epoch 7/100
90/90 [==============================] - 11s 126ms/step - loss: 1.0511 - accuracy:
Epoch 8/100
90/90 [==============================] - 12s 128ms/step - loss: 0.9755 - accuracy:
Epoch 9/100
90/90 [==============================] - 11s 127ms/step - loss: 0.9343 - accuracy:
Epoch 10/100
```

```
90/90 [==============================] - 11s 127ms/step - loss: 0.9387 - accuracy:
Epoch 11/100
90/90 [==============================] - 12s 128ms/step - loss: 0.9197 - accuracy:
Epoch 12/100
90/90 [==============================] - 11s 127ms/step - loss: 0.8904 - accuracy:
Epoch 13/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7953 - accuracy:
Epoch 14/100
90/90 [==============================] - 11s 127ms/step - loss: 0.8727 - accuracy:
Epoch 15/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7750 - accuracy:
Epoch 16/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7289 - accuracy:
Epoch 17/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7285 - accuracy:
Epoch 18/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7193 - accuracy:
Epoch 19/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7022 - accuracy:
Epoch 20/100
90/90 [==============================] - 11s 127ms/step - loss: 0.7323 - accuracy:
Epoch 21/100
90/90 [==============================] - 11s 127ms/step - loss: 0.6812 - accuracy:
Epoch 22/100
90/90 [==============================] - 11s 127ms/step - loss: 0.5751 - accuracy:
Epoch 23/100
90/90 [==============================] - 11s 127ms/step - loss: 0.5924 - accuracy:
Epoch 24/100
90/90 [==============================] - 11s 127ms/step - loss: 0.5515 - accuracy:
Epoch 25/100
90/90 [==============================] - 11s 127ms/step - loss: 0.6511 - accuracy:
Epoch 26/100
90/90 [==============================] - 12s 128ms/step - loss: 0.5781 - accuracy:
Epoch 27/100
90/90 [==============================] - 11s 127ms/step - loss: 0.5879 - accuracy:
Epoch 28/100
90/90 [==============================] - 11s 126ms/step - loss: 0.5588 - accuracy:
Epoch 29/100
90/90 [==============================] - 11s 127ms/step - loss: 0.5459 - accuracy:
Epoch 30/100
90/90 [==============================] - 12s 127ms/step - loss: 0.4622 - accuracy:
Epoch 31/100
90/90 [==============================] - 11s 126ms/step - loss: 0.5897 - accuracy:
Epoch 32/100
90/90 [==============================] - 11s 127ms/step - loss: 0.4947 - accuracy:
Epoch 00032: early stopping
```

Save the model and training history

```
alexnet_model.save("/content/drive/MyDrive/mri-cnn/alexnet")
pd.DataFrame(alexnet_history.history).to_csv("/content/drive/MyDrive/mri-cnn/alexnet.csv")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/alexnet/assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/alexnet/assets
```

Model 7 - LeNet

Like AlexNet, Tensorflow does not have a prebuilt model for LeNet. Therefore, we will define the model on our own.

```
lenet_model = tf.keras.Sequential()
lenet_model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', i
lenet_model.add(tf.keras.layers.AveragePooling2D())
lenet_model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
lenet_model.add(tf.keras.layers.AveragePooling2D())
lenet_model.add(tf.keras.layers.Flatten())
lenet_model.add(tf.keras.layers.Dense(units=120, activation='relu'))
lenet_model.add(tf.keras.layers.Dense(units=84, activation='relu'))
lenet_model.add(tf.keras.layers.Dense(units=4))
```

We are now ready to compile and train LeNet

```
lenet_model.compile(optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
lenet_history = lenet_model.fit(
  training_dataset,
  validation_data=testing_dataset,
  epochs=MAX_EPOCHS,
  callbacks=[es]
)
```

```
Epoch 1/100
90/90 [==============================] - 5s 45ms/step - loss: 9.0646 - accuracy: 0.6
Epoch 2/100
90/90 [==============================] - 4s 41ms/step - loss: 0.2646 - accuracy: 0.9
Epoch 3/100
90/90 [==============================] - 4s 42ms/step - loss: 0.1433 - accuracy: 0.9
Epoch 4/100
90/90 [==============================] - 4s 42ms/step - loss: 0.0894 - accuracy: 0.9
Epoch 5/100
90/90 [==============================] - 4s 42ms/step - loss: 0.0548 - accuracy: 0.9
Epoch 6/100
90/90 [==============================] - 4s 40ms/step - loss: 0.0087 - accuracy: 0.9
Epoch 7/100
90/90 [==============================] - 4s 40ms/step - loss: 0.0040 - accuracy: 1.0
Epoch 8/100
90/90 [==============================] - 4s 40ms/step - loss: 0.0011 - accuracy: 1.0
Epoch 9/100
90/90 [==============================] - 4s 40ms/step - loss: 6.6158e-04 - accuracy:
Epoch 10/100
90/90 [==============================] - 4s 40ms/step - loss: 5.1745e-04 - accuracy:
Epoch 11/100
90/90 [==============================] - 4s 39ms/step - loss: 4.2063e-04 - accuracy:
Epoch 12/100
90/90 [==============================] - 4s 40ms/step - loss: 3.4883e-04 - accuracy:
Epoch 13/100
90/90 [==============================] - 4s 40ms/step - loss: 2.9345e-04 - accuracy:
Epoch 00013: early stopping
```

Save the model and training history

```
lenet_model.save("/content/drive/MyDrive/mri-cnn/lenet")
pd.DataFrame(lenet_history.history).to_csv("/content/drive/MyDrive/mri-cnn/lenet.csv")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/lenet/assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/mri-cnn/lenet/assets
```

Conclusion

We have used the following models for the MRI tumour classification task -

1. Basic CNN
2. ResNet50
3. VGG16
4. MobileNet
5. InceptionNet v2
6. AlexNet
7. LeNet

We splitted the dataset into training and validation sets. Once splitted, we trained the different models using the training set.

We used the validation set to determine the training performance at the end of each epoch.

We used early stopping with a high tolerance to stop the training earlier than the scheduled time in case the training was stuck in a minima without further improvement in performance.

We have saved the models trained and the training histories for further analysis.