In [ ]:

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
```

In [ ]:

```
!kaggle datasets download -d sartajbhuvaji/brain-tumor-classification-mri
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /roo
t/.kaggle/kaggle.json'
Downloading brain-tumor-classification-mri.zip to /content
 99% 86.0M/86.8M [00:02<00:00, 30.6MB/s]
100% 86.8M/86.8M [00:02<00:00, 31.6MB/s]
```

In [ ]:

```
import zipfile
zip_ref = zipfile.ZipFile('/content/brain-tumor-classification-mri.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

In [ ]:

```
import numpy as np
import pandas as pd
import os

import keras
from keras.models import Sequential
from keras.layers import Conv2D,Flatten,Dense,MaxPooling2D,Dropout
from sklearn.metrics import accuracy_score
```

In [ ]:

```
import ipywidgets as widgets
import io
from PIL import Image
import tqdm
from sklearn.model_selection import train_test_split
import cv2
from sklearn.utils import shuffle
import tensorflow as tf
```

Folder paths

In [ ]:

```
X_train = []
Y_train = []
image_size = 150
labels = ['glioma_tumor','meningioma_tumor','no_tumor','pituitary_tumor']
for i in labels:
    folderPath = os.path.join('/content/Training',i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size,image_size))
        X_train.append(img)
        Y_train.append(i)

for i in labels:
    folderPath = os.path.join('/content/Testing',i)
    for j in os.listdir(folderPath):
        img = cv2.imread(os.path.join(folderPath,j))
        img = cv2.resize(img,(image_size,image_size))
        X_train.append(img)
        Y_train.append(i)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
```

In [ ]:

```
X_train,Y_train = shuffle(X_train,Y_train,random_state=101)
X_train.shape
```

Out[ ]:

```
(3264, 150, 150, 3)
```

In [ ]:

```
X_train,X_test,y_train,y_test = train_test_split(X_train,Y_train,test_size=0.1,random_state=101)
```

In [ ]:

```
y_train_new = []
for i in y_train:
    y_train_new.append(labels.index(i))
y_train=y_train_new
y_train = tf.keras.utils.to_categorical(y_train)

y_test_new = []
for i in y_test:
    y_test_new.append(labels.index(i))
y_test=y_test_new
y_test = tf.keras.utils.to_categorical(y_test)
```

Convolutional Neural Network

In [ ]:

```
model = Sequential()
model.add(Conv2D(32,(3,3),activation = 'relu',input_shape=(150,150,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Dropout(0.3))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(Conv2D(64,(3,3),activation='relu'))

model.add(Dropout(0.3))
model.add(MaxPooling2D(2,2))
model.add(Dropout(0.3))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Dropout(0.3))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(Conv2D(256,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))

model.add(Dropout(0.3))

model.add(Flatten())

model.add(Dense(512,activation = 'relu'))
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(4,activation='softmax'))
```

In [ ]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896       
                                                                 
 conv2d_1 (Conv2D)           (None, 146, 146, 64)      18496     
                                                                 
 max_pooling2d (MaxPooling2  (None, 73, 73, 64)        0         
 D)                                                              
                                                                 
 dropout (Dropout)           (None, 73, 73, 64)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 71, 71, 64)        36928     
                                                                 
 conv2d_3 (Conv2D)           (None, 69, 69, 64)        36928     
                                                                 
 dropout_1 (Dropout)         (None, 69, 69, 64)        0         
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 34, 34, 64)        0         
 g2D)                                                            
                                                                 
 dropout_2 (Dropout)         (None, 34, 34, 64)        0         
                                                                 
 conv2d_4 (Conv2D)           (None, 32, 32, 128)       73856     
                                                                 
 conv2d_5 (Conv2D)           (None, 30, 30, 128)       147584    
                                                                 
 conv2d_6 (Conv2D)           (None, 28, 28, 128)       147584    
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 14, 14, 128)       0         
 g2D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 14, 14, 128)       0         
                                                                 
 conv2d_7 (Conv2D)           (None, 12, 12, 128)       147584    
                                                                 
 conv2d_8 (Conv2D)           (None, 10, 10, 256)       295168    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 5, 5, 256)         0         
 g2D)                                                            
                                                                 
 dropout_4 (Dropout)         (None, 5, 5, 256)         0         
                                                                 
 flatten (Flatten)           (None, 6400)              0         
                                                                 
 dense (Dense)               (None, 512)               3277312   
                                                                 
 dense_1 (Dense)             (None, 512)               262656    
                                                                 
 dropout_5 (Dropout)         (None, 512)               0         
                                                                 
 dense_2 (Dense)             (None, 4)                 2052      
                                                                 
=================================================================
Total params: 4447044 (16.96 MB)
Trainable params: 4447044 (16.96 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
In [ ]: model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

```
In [ ]: history = model.fit(X_train,y_train,epochs=20,validation_split=0.1)
```

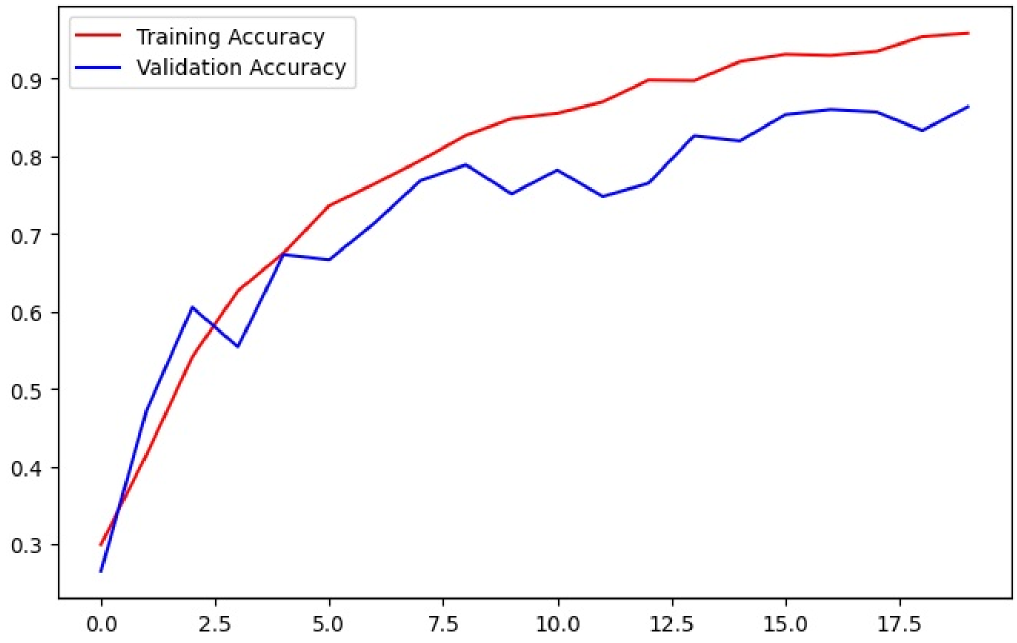
```
Epoch 1/20
83/83 [==============================] - 24s 148ms/step - loss: 1.9651 - accuracy: 0.2997 - val_loss: 1.3456 - v
al_accuracy: 0.2653
Epoch 2/20
83/83 [==============================] - 8s 93ms/step - loss: 1.2675 - accuracy: 0.4158 - val_loss: 1.1990 - val
_accuracy: 0.4728
Epoch 3/20
83/83 [==============================] - 8s 94ms/step - loss: 1.0261 - accuracy: 0.5414 - val_loss: 0.9798 - val
_accuracy: 0.6054
Epoch 4/20
83/83 [==============================] - 8s 96ms/step - loss: 0.8652 - accuracy: 0.6266 - val_loss: 0.9827 - val
_accuracy: 0.5544
Epoch 5/20
83/83 [==============================] - 8s 94ms/step - loss: 0.7675 - accuracy: 0.6754 - val_loss: 0.7400 - val
_accuracy: 0.6735
Epoch 6/20
83/83 [==============================] - 8s 94ms/step - loss: 0.6545 - accuracy: 0.7363 - val_loss: 0.7055 - val
_accuracy: 0.6667
Epoch 7/20
83/83 [==============================] - 8s 98ms/step - loss: 0.5708 - accuracy: 0.7647 - val_loss: 0.5861 - val
_accuracy: 0.7143
Epoch 8/20
83/83 [==============================] - 8s 98ms/step - loss: 0.4987 - accuracy: 0.7946 - val_loss: 0.5485 - val
_accuracy: 0.7687
Epoch 9/20
83/83 [==============================] - 8s 98ms/step - loss: 0.4357 - accuracy: 0.8271 - val_loss: 0.4981 - val
_accuracy: 0.7891
Epoch 10/20
83/83 [==============================] - 8s 98ms/step - loss: 0.3765 - accuracy: 0.8487 - val_loss: 0.5720 - val
_accuracy: 0.7517
Epoch 11/20
83/83 [==============================] - 8s 98ms/step - loss: 0.3729 - accuracy: 0.8555 - val_loss: 0.5347 - val
_accuracy: 0.7823
Epoch 12/20
83/83 [==============================] - 8s 98ms/step - loss: 0.3376 - accuracy: 0.8702 - val_loss: 0.5951 - val
_accuracy: 0.7483
Epoch 13/20
83/83 [==============================] - 8s 100ms/step - loss: 0.2626 - accuracy: 0.8982 - val_loss: 0.5673 - va
l_accuracy: 0.7653
Epoch 14/20
83/83 [==============================] - 8s 99ms/step - loss: 0.2679 - accuracy: 0.8975 - val_loss: 0.4189 - val
_accuracy: 0.8265
Epoch 15/20
83/83 [==============================] - 8s 96ms/step - loss: 0.2160 - accuracy: 0.9221 - val_loss: 0.4570 - val
_accuracy: 0.8197
Epoch 16/20
83/83 [==============================] - 8s 96ms/step - loss: 0.1770 - accuracy: 0.9315 - val_loss: 0.5020 - val
_accuracy: 0.8537
Epoch 17/20
83/83 [==============================] - 8s 95ms/step - loss: 0.1968 - accuracy: 0.9300 - val_loss: 0.4403 - val
_accuracy: 0.8605
Epoch 18/20
83/83 [==============================] - 8s 96ms/step - loss: 0.1830 - accuracy: 0.9353 - val_loss: 0.4525 - val
_accuracy: 0.8571
Epoch 19/20
83/83 [==============================] - 8s 97ms/step - loss: 0.1370 - accuracy: 0.9542 - val_loss: 0.4640 - val
_accuracy: 0.8333
Epoch 20/20
83/83 [==============================] - 8s 96ms/step - loss: 0.1202 - accuracy: 0.9588 - val_loss: 0.4128 - val
_accuracy: 0.8639
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

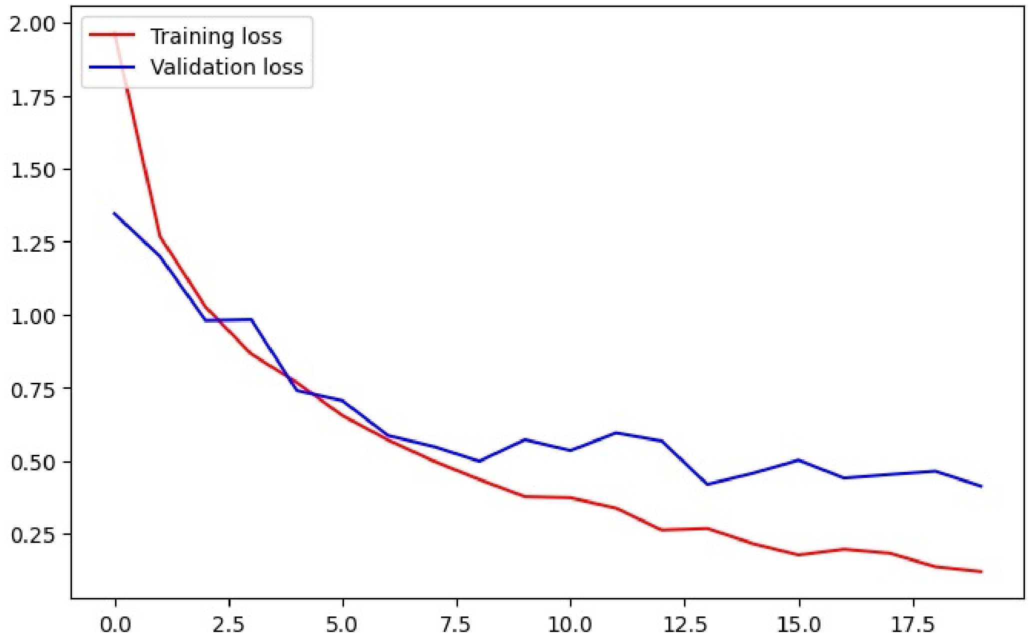
In [ ]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(len(acc))
fig = plt.figure(figsize=(8,5))
plt.plot(epochs,acc,'r',label="Training Accuracy")
plt.plot(epochs,val_acc,'b',label="Validation Accuracy")
plt.legend(loc='upper left')
plt.show()
```



In [ ]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(loss))
fig = plt.figure(figsize=(8,5))
plt.plot(epochs,loss,'r',label="Training loss")
plt.plot(epochs,val_loss,'b',label="Validation loss")
plt.legend(loc='upper left')
plt.show()
```



Prediction

In [ ]:

```
# model.save('braintumor.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your mode
l as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nati
ve Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

In [ ]:

```
img = cv2.imread('/content/image(118).jpg')
img = cv2.resize(img,(150,150))
img_array = np.array(img)
img_array.shape
```

Out[ ]:

```
(150, 150, 3)
```

In [ ]:

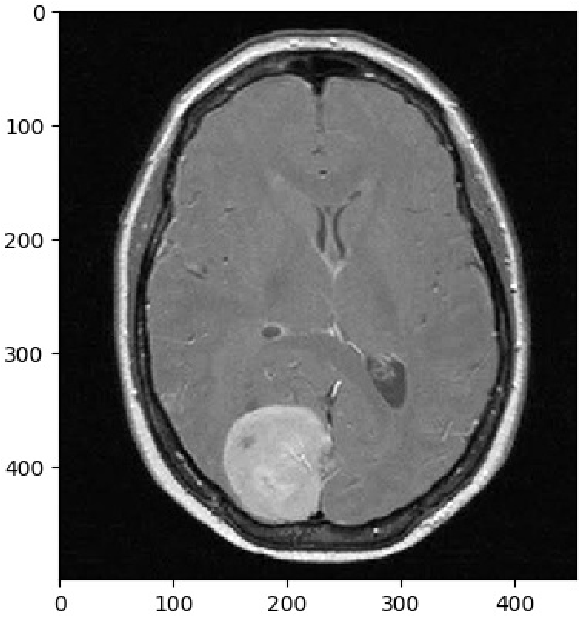
```
img_array = img_array.reshape(1,150,150,3)
img_array.shape
```

Out[ ]:

```
(1, 150, 150, 3)
```

In [ ]:

```
from tensorflow.keras.preprocessing import image
img = image.load_img('/content/image(118).jpg')
plt.imshow(img,interpolation='nearest')
plt.show()
```



In [ ]:

```
a=model.predict(img_array)
indices = a.argmax()
```

```
1/1 [==============================] - 0s 27ms/step
```

In [ ]:

```
if indices == 0:
  print("This is the Glioma Tumor")

elif indices == 1:
  print("This is the Meningioma Tumor")

elif indices == 3:
  print("This is the Pituitary Tumor")

else:
  print("There are no any Tumor.")
```

```
This is the Meningioma Tumor
```

Created by: *Abdul Mannan*

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js