

```
In [1]: import kagglehub
navoneel_brain_mri_images_for_brain_tumor_detection_path = kagglehub.dataset_download('navoneel/brain-mri-images-for-brain-tumor-detection')

print('Data source import complete.')
```

Downloading from https://www.kaggle.com/api/v1/datasets/download/navoneel/brain-mri-images-for-brain-tumor-detection?dataset_version_number=1...

100%|██████████| 15.1M/15.1M [00:00<00:00, 98.1MB/s]

Extracting files...

Data source import complete.

```
In [7]: import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
In [8]: import os
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('dark_background')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
```

```
In [15]: data = []
paths = []
result = []
encoder = OneHotEncoder()
encoder.fit([[0], [1]])

downloaded_yes_path = os.path.join(navoneel_brain_mri_images_for_brain_tumor_detection_path, 'yes')
for r, d, f in os.walk(download_path):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r, file))
for path in paths:
    img = Image.open(path)
    img = img.resize((128, 128))
    img = img.convert('RGB')
    img = np.array(img)
    if (img.shape == (128, 128, 3)):
        data.append(np.array(img))
        result.append(encoder.transform([[0]]).toarray())
```

```
In [16]: paths = []
```

```

downloaded_no_path = os.path.join(navoneel_brain_mri_images_for_brain_tumor_detecti
for r, d, f in os.walk(downloaded_no_path):
    for file in f:
        if '.jpg' in file:
            paths.append(os.path.join(r , file))
for path in paths:
    img = Image.open(path)
    img = img.resize((128 , 128))
    img = img.convert('RGB')
    img = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[1]]).toarray())

```

```

In [17]: data = np.array(data)
        result = np.vstack(result)
        data.shape

```

```

Out[17]: (171, 128, 128, 3)

```

SPLITTING THE DATA INTO TRAINING AND TESTING

```

In [18]: x_train , x_test , y_train , y_test = train_test_split(data , result , test_size =

```

```

In [19]: print(f'No of images in training data:{len(x_train)}')

```

```

No of images in training data:136

```

```

In [20]: print(f'No of images in training data:{len(x_test)}')

```

```

No of images in training data:35

```

BUILDING THE AI MODEL

```

In [21]: model = Sequential()

model.add(Conv2D(32 , kernel_size = (2,2) , input_shape=(128,128,3) , padding = 'Sa
model.add(Conv2D(32 , kernel_size = (2,2) , activation = 'relu' , padding = 'Same'))

model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size = (2,2) , activation = 'relu' , padding = 'Same'))
model.add(Conv2D(64, kernel_size = (2,2) , activation = 'relu' , padding = 'Same'))

model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size = (2,2) , strides=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

```

```

model.add(Dense(512 , activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(2 , activation = 'softmax'))

model.compile(loss = "categorical_crossentropy" , optimizer = 'Adamax')
print(model.summary())

```

/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.

```
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 32)	416
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4,128
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d (MaxPooling2D)	(None, 64, 64, 32)	0
dropout (Dropout)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	8,256
conv2d_3 (Conv2D)	(None, 64, 64, 64)	16,448
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 64)	0
dropout_1 (Dropout)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 512)	33,554,944
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1,026

Total params: 33,585,602 (128.12 MB)

Trainable params: 33,585,410 (128.12 MB)

Non-trainable params: 192 (768.00 B)

None

In [22]: len(y_train)

Out[22]: 136

MODEL EVALUATION

```
In [23]: history = model.fit(x_train , y_train , epochs = 30 , batch_size = 40 , verbose =
```

```
Epoch 1/30
4/4  14s 3s/step - loss: 24.0021 - val_loss: 50.7963
Epoch 2/30
4/4  19s 2s/step - loss: 17.4933 - val_loss: 23.5119
Epoch 3/30
4/4  14s 3s/step - loss: 6.0681 - val_loss: 17.2948
Epoch 4/30
4/4  11s 2s/step - loss: 2.1606 - val_loss: 5.1987
Epoch 5/30
4/4  20s 3s/step - loss: 1.0822 - val_loss: 3.3590
Epoch 6/30
4/4  10s 3s/step - loss: 1.3179 - val_loss: 2.9253
Epoch 7/30
4/4  21s 2s/step - loss: 0.8441 - val_loss: 2.9233
Epoch 8/30
4/4  11s 3s/step - loss: 0.8046 - val_loss: 3.5199
Epoch 9/30
4/4  20s 3s/step - loss: 0.4246 - val_loss: 3.4923
Epoch 10/30
4/4  19s 2s/step - loss: 0.3345 - val_loss: 3.4082
Epoch 11/30
4/4  11s 2s/step - loss: 0.2364 - val_loss: 3.6424
Epoch 12/30
4/4  11s 3s/step - loss: 0.0362 - val_loss: 3.5312
Epoch 13/30
4/4  11s 3s/step - loss: 0.0490 - val_loss: 3.1208
Epoch 14/30
4/4  10s 3s/step - loss: 0.0549 - val_loss: 2.9016
Epoch 15/30
4/4  10s 2s/step - loss: 0.0279 - val_loss: 2.7635
Epoch 16/30
4/4  11s 2s/step - loss: 0.0567 - val_loss: 2.6922
Epoch 17/30
4/4  21s 3s/step - loss: 0.0307 - val_loss: 2.6511
Epoch 18/30
4/4  20s 3s/step - loss: 0.1593 - val_loss: 2.5682
Epoch 19/30
4/4  10s 2s/step - loss: 0.0688 - val_loss: 2.4471
Epoch 20/30
4/4  11s 2s/step - loss: 0.0044 - val_loss: 2.3325
Epoch 21/30
4/4  11s 3s/step - loss: 0.0253 - val_loss: 2.2957
Epoch 22/30
4/4  10s 3s/step - loss: 0.0033 - val_loss: 2.2603
Epoch 23/30
4/4  11s 3s/step - loss: 0.0047 - val_loss: 2.2155
Epoch 24/30
4/4  20s 2s/step - loss: 0.0120 - val_loss: 2.1477
Epoch 25/30
4/4  11s 2s/step - loss: 0.0068 - val_loss: 2.0782
Epoch 26/30
4/4  11s 3s/step - loss: 0.0140 - val_loss: 1.9980
Epoch 27/30
4/4  21s 3s/step - loss: 0.0737 - val_loss: 1.9235
Epoch 28/30
4/4  10s 2s/step - loss: 0.0723 - val_loss: 1.8763
```

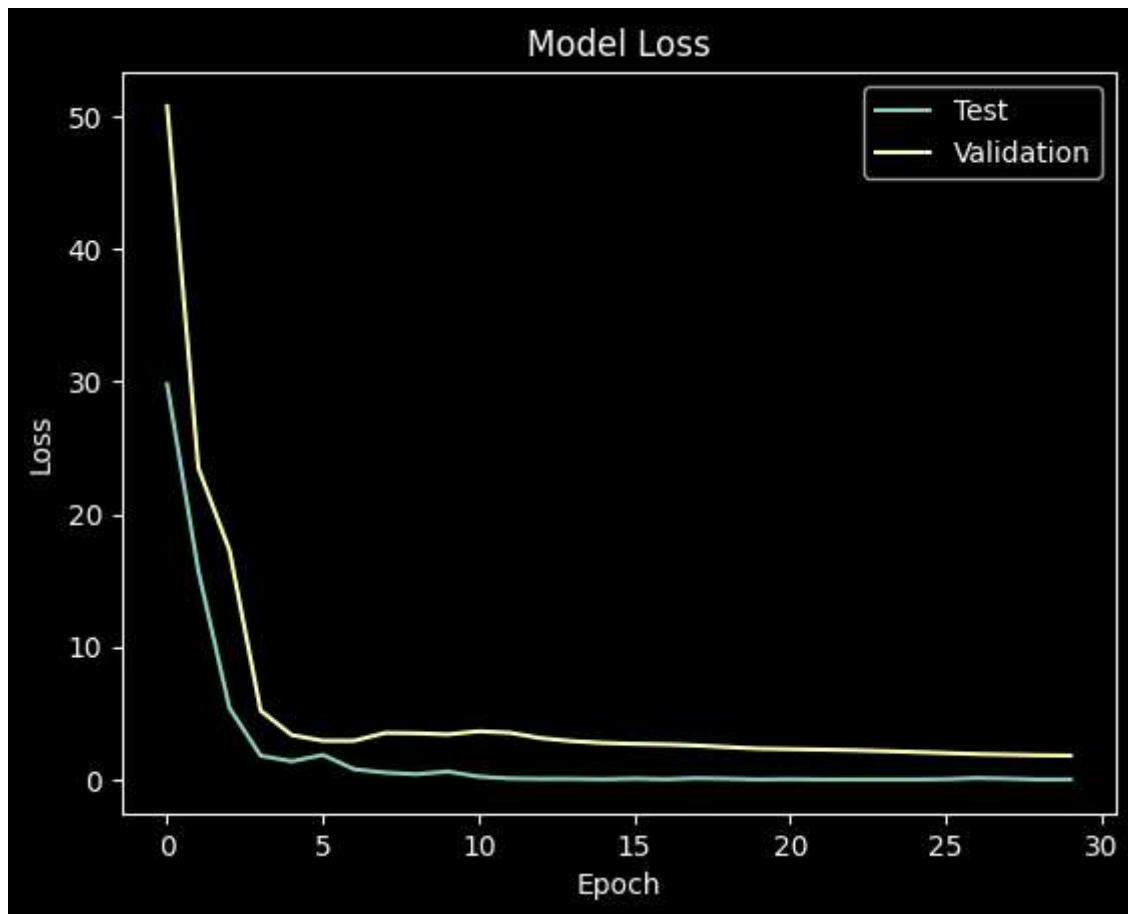
Epoch 29/30

4/4 ————— 10s 2s/step - loss: 0.0027 - val_loss: 1.8278

Epoch 30/30

4/4 ————— 11s 2s/step - loss: 0.0102 - val_loss: 1.8070

```
In [25]: plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Test', 'Validation'], loc='upper right')
plt.show()
```



```
In [24]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import numpy as np

y_pred_proba = model.predict(x_test)
y_pred = np.argmax(y_pred_proba, axis=1)

y_true = np.argmax(y_test, axis=1)

accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
```

```

print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")

print("\nClassification Report:")
print(classification_report(y_true, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_true, y_pred))

```

2/2 ————— 1s 265ms/step

Accuracy: 0.8000
Precision: 0.8007
Recall: 0.8000
F1-Score: 0.7997

Classification Report:

	precision	recall	f1-score	support
0	0.81	0.76	0.79	17
1	0.79	0.83	0.81	18
accuracy			0.80	35
macro avg	0.80	0.80	0.80	35
weighted avg	0.80	0.80	0.80	35

Confusion Matrix:

```

[[13  4]
 [ 3 15]]

```

TESTING THE MODEL

```

In [26]: def names(number):
          if number == 0:
              return "It's a Tumor"
          else:
              return "No, Its not a Tumor"

```

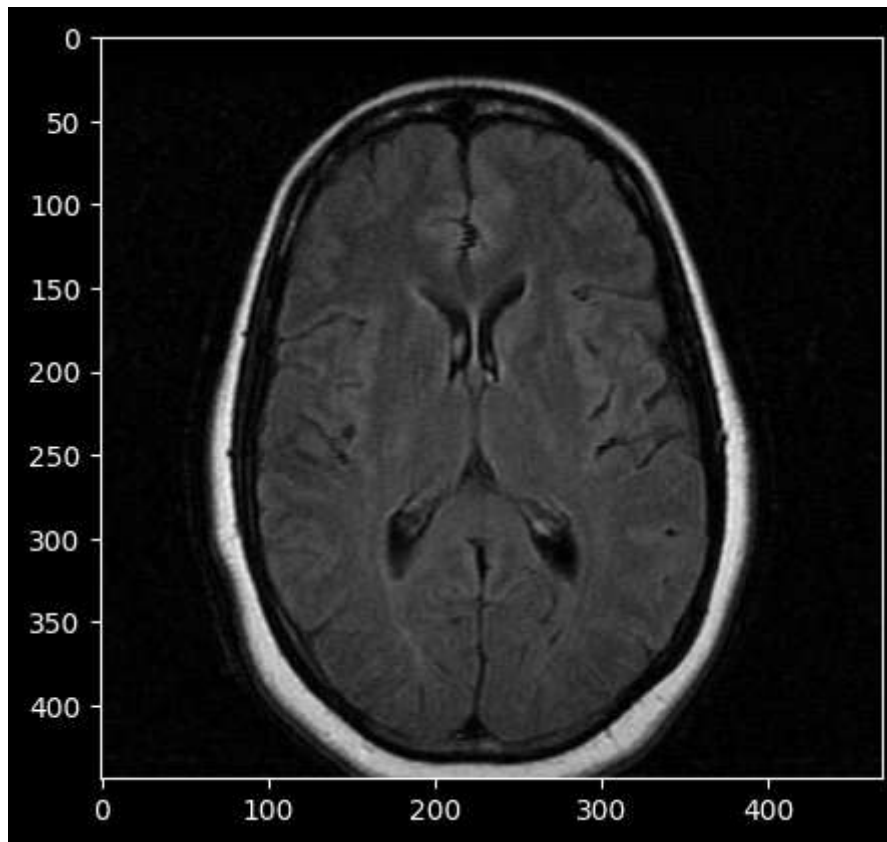
```

In [30]: from matplotlib.pyplot import imshow
          import numpy as np
          from PIL import Image

          img_path_to_test = paths[0]
          img = Image.open(img_path_to_test)
          x = np.array(img.resize((128,128)))
          x = x.reshape(1,128,128,3)
          res = model.predict_on_batch(x)
          classification = np.where(res == np.amax(res))[1][0]
          imshow(img)
          print(str(res[0][classification]*100) + '% Confidence This Is ' + names(classificat

```

99.984% Confidence This Is No, Its not a Tumor



In []: