brain-tumor-classification

January 24, 2024

[25]:
```
#Import libraries
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
```

0.1 One Hot Encoding the Target Classes

[26]:
```
encoder = OneHotEncoder()
encoder.fit([[0], [1]])

# 0 - Tumor
# 1 - Normal
```

[26]:
```
OneHotEncoder()
```

0.1.1 Creating 3 Important Lists

[27]:
```
data = []
paths = []
result = []

for r, d, f in os.walk(r'/kaggle/input/
↪brain-mri-images-for-brain-tumor-detection/yes'):
    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 = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[0]]).toarray())
```

[28]:
```
# This cell updates result list for images without tumor

paths = []
for r, d, f in os.walk(r"/kaggle/input/
 ↪brain-mri-images-for-brain-tumor-detection/no"):
    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 = np.array(img)
    if(img.shape == (128,128,3)):
        data.append(np.array(img))
        result.append(encoder.transform([[1]]).toarray())
```

[29]:
```
data = np.array(data)
data.shape
```

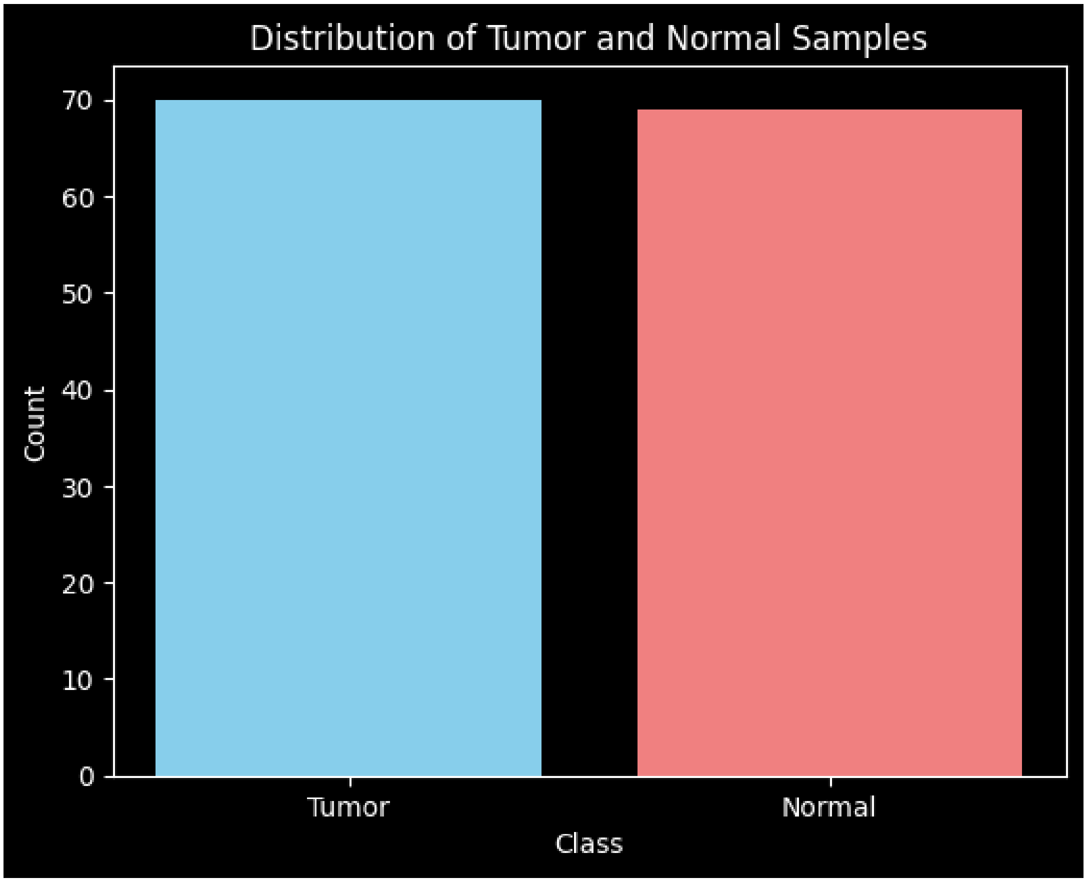
[29]: (139, 128, 128, 3)

[31]:
```
result = np.array(result)
result = result.reshape(139,2)
```

0.1.2 creating a bar graph for tumor and normal

[35]:
```
import matplotlib.pyplot as plt

# Count the number of Tumor and Normal samples
tumor_count = np.sum(result[:, 0] == 1)
normal_count = np.sum(result[:, 1] == 1)

# Create labels and counts for the bar plot
labels = ['Tumor', 'Normal']
counts = [tumor_count, normal_count]

# Example colors that are complementary and visually pleasing
colors = ['skyblue', 'lightcoral']
```

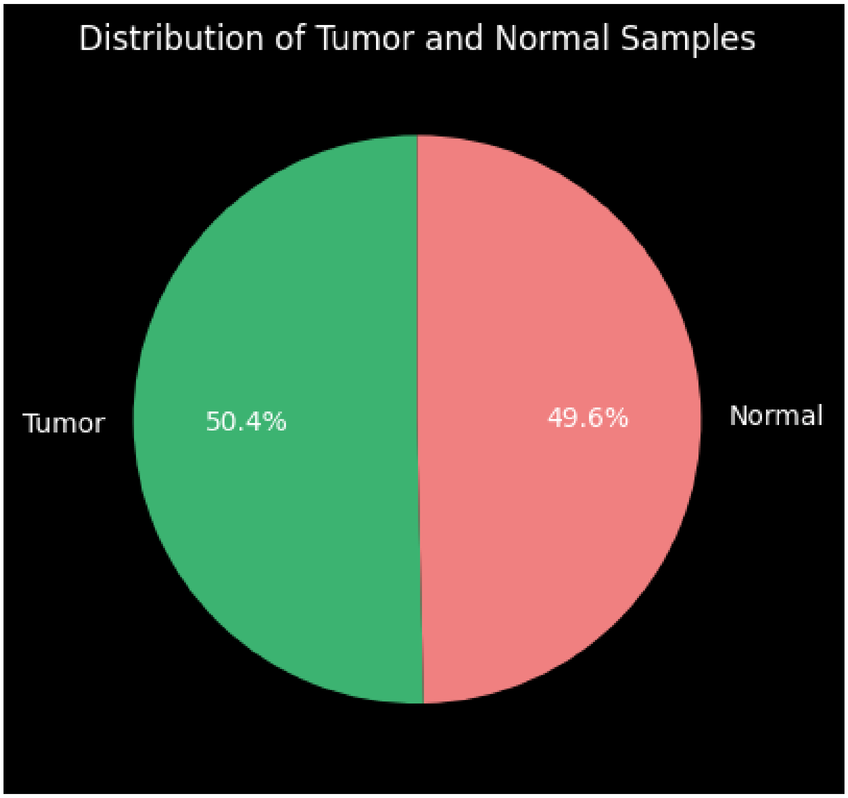
[36]:
```
# Create a bar plot
plt.bar(labels, counts, color=colors)
plt.title('Distribution of Tumor and Normal Samples')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



0.1.3 Creating a Pie Chart for Tumor and Normal

[38]:
```
# Calculate percentages
total_samples = len(result)
tumor_percentage = (tumor_count / total_samples) * 100
normal_percentage = (normal_count / total_samples) * 100

# Create labels and counts for the pie chart
labels = ['Tumor', 'Normal']
counts = [tumor_percentage, normal_percentage]
colors = ['mediumseagreen', 'lightcoral']
```

```python
# Create a pie chart
plt.pie(counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors)
plt.title('Distribution of Tumor and Normal Samples')
plt.show()
```



0.2 Splitting the Data into Training & Testing

[39]:
```python
x_train,x_test,y_train,y_test = train_test_split(data, result, test_size=0.2,␣
↪shuffle=True, random_state=0)
```

1 Model Building

[40]:
```python
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 2), input_shape=(128, 128, 3), padding =␣
↪'Same'))
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')
model.compile(loss="categorical_crossentropy", optimizer="Adamax",␣
 ↪metrics=["accuracy"])

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      416

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      4128

 batch_normalization (Batch  (None, 128, 128, 32)      128
 Normalization)

 max_pooling2d (MaxPooling2  (None, 64, 64, 32)        0
 D)

 dropout_1 (Dropout)         (None, 64, 64, 32)        0

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        8256

 conv2d_3 (Conv2D)           (None, 64, 64, 64)        16448

 batch_normalization_1 (Bat  (None, 64, 64, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 32, 32, 64)        0
 g2D)
```

```
 dropout_2 (Dropout)         (None, 32, 32, 64)        0

 flatten_1 (Flatten)         (None, 65536)             0

 dense_2 (Dense)             (None, 512)               33554944

 dropout_3 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 2)                 1026

=================================================================
Total params: 33585602 (128.12 MB)
Trainable params: 33585410 (128.12 MB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
None
```

[41]:
```
y_train.shape
```

[41]: (111, 2)

[42]:
```
history = model.fit(x_train, y_train, epochs = 30, batch_size = 40, verbose =
 1,validation_data = (x_test, y_test))
```

```
Epoch 1/30
3/3 [==============================] - 4s 993ms/step - loss: 24.8259 - accuracy:
0.5586 - val_loss: 24.8785 - val_accuracy: 0.6429
Epoch 2/30
3/3 [==============================] - 3s 911ms/step - loss: 10.7795 - accuracy:
0.7658 - val_loss: 52.9129 - val_accuracy: 0.5357
Epoch 3/30
3/3 [==============================] - 3s 1s/step - loss: 5.2803 - accuracy:
0.8108 - val_loss: 20.1909 - val_accuracy: 0.6071
Epoch 4/30
3/3 [==============================] - 3s 903ms/step - loss: 4.3104 - accuracy:
0.8468 - val_loss: 5.9880 - val_accuracy: 0.7857
Epoch 5/30
3/3 [==============================] - 3s 897ms/step - loss: 2.7308 - accuracy:
0.8198 - val_loss: 8.7455 - val_accuracy: 0.7500
Epoch 6/30
3/3 [==============================] - 3s 926ms/step - loss: 2.8426 - accuracy:
0.8649 - val_loss: 6.9631 - val_accuracy: 0.7857
Epoch 7/30
3/3 [==============================] - 3s 919ms/step - loss: 3.6575 - accuracy:
0.8739 - val_loss: 3.5022 - val_accuracy: 0.7500
Epoch 8/30
3/3 [==============================] - 3s 912ms/step - loss: 0.8377 - accuracy:
0.9550 - val_loss: 2.1859 - val_accuracy: 0.8571
```

```
Epoch 9/30
3/3 [==============================] - 3s 896ms/step - loss: 0.1041 - accuracy:
0.9730 - val_loss: 2.9074 - val_accuracy: 0.8571
Epoch 10/30
3/3 [==============================] - 3s 913ms/step - loss: 0.6969 - accuracy:
0.9730 - val_loss: 4.9856 - val_accuracy: 0.7500
Epoch 11/30
3/3 [==============================] - 3s 893ms/step - loss: 0.6225 - accuracy:
0.9279 - val_loss: 9.5634 - val_accuracy: 0.6786
Epoch 12/30
3/3 [==============================] - 3s 923ms/step - loss: 0.0299 - accuracy:
0.9820 - val_loss: 13.0166 - val_accuracy: 0.6429
Epoch 13/30
3/3 [==============================] - 3s 919ms/step - loss: 0.9034 - accuracy:
0.9459 - val_loss: 10.9552 - val_accuracy: 0.6786
Epoch 14/30
3/3 [==============================] - 3s 969ms/step - loss: 0.0088 - accuracy:
1.0000 - val_loss: 5.7940 - val_accuracy: 0.6786
Epoch 15/30
3/3 [==============================] - 3s 909ms/step - loss: 0.1493 - accuracy:
0.9910 - val_loss: 2.7623 - val_accuracy: 0.8929
Epoch 16/30
3/3 [==============================] - 3s 918ms/step - loss: 0.0491 - accuracy:
0.9820 - val_loss: 1.5802 - val_accuracy: 0.9286
Epoch 17/30
3/3 [==============================] - 3s 918ms/step - loss: 0.1676 - accuracy:
0.9820 - val_loss: 1.2122 - val_accuracy: 0.9286
Epoch 18/30
3/3 [==============================] - 3s 926ms/step - loss: 0.0321 - accuracy:
0.9820 - val_loss: 1.3319 - val_accuracy: 0.9286
Epoch 19/30
3/3 [==============================] - 3s 923ms/step - loss: 0.0052 - accuracy:
1.0000 - val_loss: 1.7577 - val_accuracy: 0.8929
Epoch 20/30
3/3 [==============================] - 3s 890ms/step - loss: 5.0721e-06 -
accuracy: 1.0000 - val_loss: 2.2900 - val_accuracy: 0.8929
Epoch 21/30
3/3 [==============================] - 3s 901ms/step - loss: 2.4730e-04 -
accuracy: 1.0000 - val_loss: 2.6473 - val_accuracy: 0.8929
Epoch 22/30
3/3 [==============================] - 3s 910ms/step - loss: 0.0033 - accuracy:
1.0000 - val_loss: 2.9575 - val_accuracy: 0.8571
Epoch 23/30
3/3 [==============================] - 3s 894ms/step - loss: 8.8237e-04 -
accuracy: 1.0000 - val_loss: 3.2664 - val_accuracy: 0.8571
Epoch 24/30
3/3 [==============================] - 3s 901ms/step - loss: 1.6099e-05 -
accuracy: 1.0000 - val_loss: 3.4935 - val_accuracy: 0.8214
```

```
Epoch 25/30
3/3 [==============================] - 3s 912ms/step - loss: 2.3977e-04 -
accuracy: 1.0000 - val_loss: 3.6547 - val_accuracy: 0.7857
Epoch 26/30
3/3 [==============================] - 3s 975ms/step - loss: 0.0020 - accuracy:
1.0000 - val_loss: 3.8493 - val_accuracy: 0.7857
Epoch 27/30
3/3 [==============================] - 3s 890ms/step - loss: 0.2364 - accuracy:
0.9550 - val_loss: 3.1200 - val_accuracy: 0.8214
Epoch 28/30
3/3 [==============================] - 3s 917ms/step - loss: 1.9975e-07 -
accuracy: 1.0000 - val_loss: 2.3774 - val_accuracy: 0.8571
Epoch 29/30
3/3 [==============================] - 3s 895ms/step - loss: 0.1725 - accuracy:
0.9910 - val_loss: 2.1835 - val_accuracy: 0.8929
Epoch 30/30
3/3 [==============================] - 3s 909ms/step - loss: 4.9021e-05 -
accuracy: 1.0000 - val_loss: 2.0379 - val_accuracy: 0.8929
```

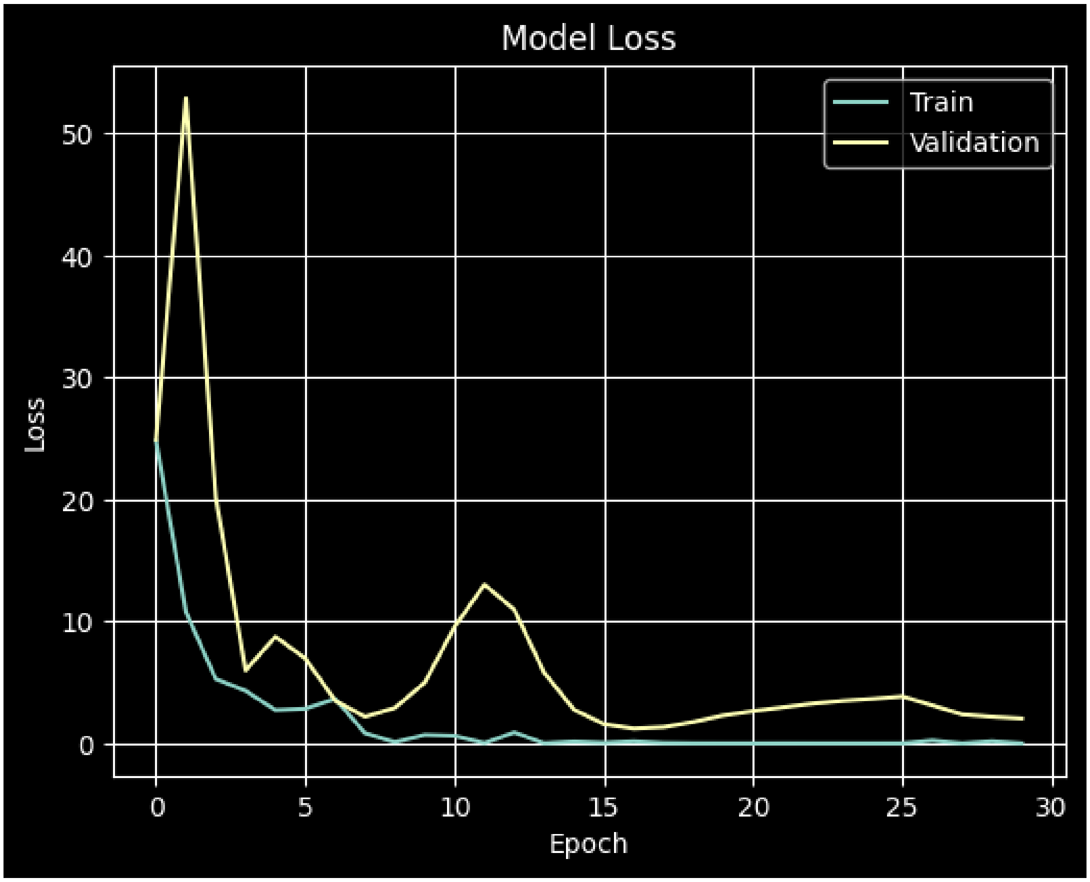
1.0.1 Accuracy of The Model

[43]:
```python
#trained model, 'x_test', 'y_test' are test data
evaluation_results = model.evaluate(x_test, y_test)

# 'evaluate' method returns a list of metrics, including accuracy
accuracy = evaluation_results[1]

print("Model Accuracy on Test Set: {:.2f}%".format(accuracy * 100))
```

```
1/1 [==============================] - 0s 163ms/step - loss: 2.0379 - accuracy:
0.8929
Model Accuracy on Test Set: 89.29%
```

1.0.2 Plotting Losses

[45]:
```python
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.grid(True)
plt.show()
```

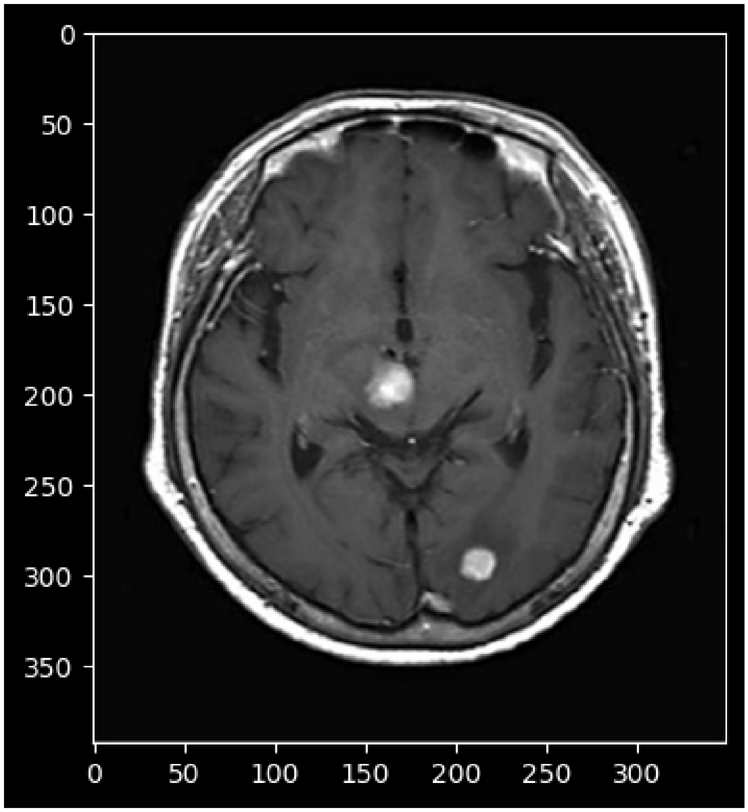


1.1 Just Checking the Model

[46]:
```
def names(number):
    if number==0:
        return 'Its a Tumor'
    else:
        return 'No, Its not a tumor'
```

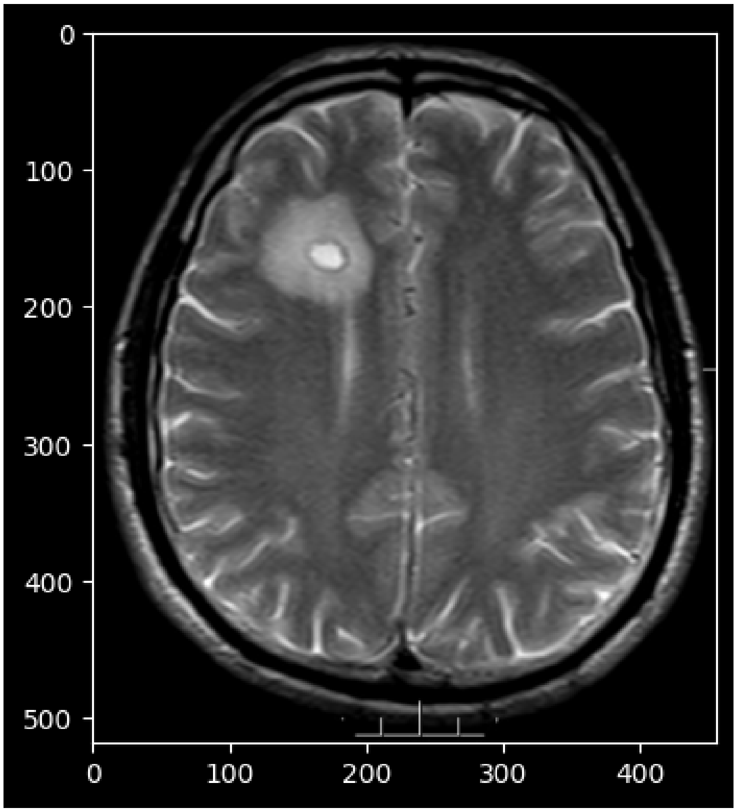
[47]:
```
from matplotlib.pyplot import imshow
img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/no/N17.
 ↪jpg")
x = np.array(img.resize((128,128)))
x = x.reshape(1,128,128,3)
res = model.predict_on_batch(x)
classification = np.where(res == np.amax(res))[1][0]
imshow(img)
print(str(res[0][classification]*100) + '% Confidence This Is ' +␣
 ↪names(classification))
```

```
100.0% Confidence This Is No, Its not a tumor
```



[51]:
```
from matplotlib.pyplot import imshow
img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/yes/Y3.
 ↪jpg")
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 A ,' +␣
 ↪names(classification))
```

```
100.0% Confidence This Is A ,Its a Tumor
```



1.2 Accuracy, Precision, recall, and F1 score

[56]:
```
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪accuracy_score

# 'x_test', 'y_test' are test data
y_pred = model.predict(x_test)

# Convert probabilities to class labels
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Calculate precision, recall, and F1 score
accuracy = accuracy_score(y_true_classes, y_pred_classes)
precision = precision_score(y_true_classes, y_pred_classes)
recall = recall_score(y_true_classes, y_pred_classes)
f1 = f1_score(y_true_classes, y_pred_classes)

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

```python
print("F1 Score: {:.4f}".format(f1))
```

```
1/1 [==============================] - 0s 161ms/step
Accuracy: 0.8929
Precision: 1.0000
Recall: 0.7857
F1 Score: 0.8800
```