

```
!pip install tqdm
```

🔄 Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (4.67.1)

```
import os
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.utils import to_categorical
from PIL import Image
import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
plt.style.use('ggplot')
plt.style.use('dark_background')
import tensorflow as tf
from sklearn.preprocessing import OneHotEncoder
from tqdm import tqdm
from sklearn.model_selection import train_test_split
```

```
from zipfile import ZipFile
file_name="/content/archive (2).zip"
with ZipFile(file_name,'r') as zip:
    zip.extractall()
    print('Done')
```

🔄 Done

```
from google.colab import drive
drive.mount('/content/drive')
```

```
encoder = OneHotEncoder(sparse_output=False)
encoder.fit([[0], [1]])
```

🔄

OneHotEncoder ⓘ ?

OneHotEncoder(sparse\_output=False)

```
data=[]
paths=[]
result=[]
masks=[]
images=[]
labels=[]
for r,d,f in os.walk(r'/content/brain_tumor_dataset/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]]))
```

```
paths = []
for r, d, f in os.walk(r"/content/brain_tumor_dataset/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]]))
```

```
def load_image(file_path, target_size=(128, 128)):
    image = tf.keras.preprocessing.image.load_img(file_path, color_mode="grayscale", target_size=target_size)
    image = tf.keras.preprocessing.image.img_to_array(image)
    image = image / 255.0 # Normalize to [0, 1]
    return image
```

```
data = np.array(data)
data.shape
```

```
(3008, 128, 128, 3)
```

```
print(f'Total number of images we have: {len(data)}')
```

```
Total number of images we have: 3008
```

```
result = np.array(result)
result = result.reshape(len(data), 2)
```

```
os.chdir('/content/brain_tumor_dataset/yes')
```

```
X=[]
```

```
y=[]
```

```
for i in tqdm(os.listdir()):
```

```
    img=cv2.imread(i)
```

```
    img=cv2.resize(img,(128,128))
```

```
    X.append(img)
```

```
    y.append((i[0:1]))
```

```
    print(i[0:1])
```

```
os.chdir('/content/brain_tumor_dataset/no')
```

```
for i in tqdm(os.listdir()):
```

```
    img=cv2.imread(i)
```

```
    img=cv2.resize(img,(128,128))
```

```
    X.append(img)
```

```
    for i in range(1,99):
```

```
        y.append('N')
```

```
    print(y)
```

```
50%|██████| 77/155 [00:00<00:00, 385.54it/s]Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

```
Y
```

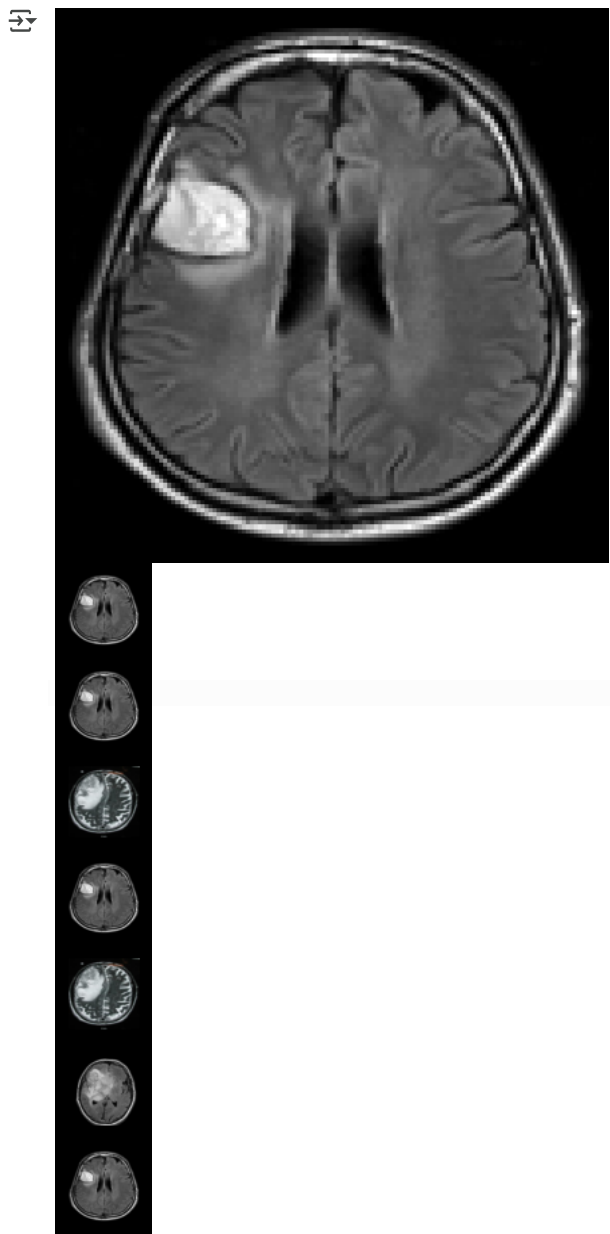
```
Y
```

```
Y
```

Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y  
Y

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.figure(figsize=(48,48))
for i in range(8):
    plt.subplot(1,8,i+1)

    plt.imshow(data[i],cmap="gray")
    plt.axis("off")
    plt.show()
```



```
x_train,x_test,y_train,y_test = train_test_split(data, result, test_size=0.2, shuffle=True, random_state=0)
```

```
print(f'Number of images in training data: {len(x_train)}')
```

```
↪ Number of images in training data: 2406
```

```
print(f'Number of images in testing data: {len(x_test)}')
```

```
↪ Number of images in testing data: 602
```

```
fig=plt.figure(figsize=(20,12))
```

```
fig=plt.figure(figsize=(20,12))
```

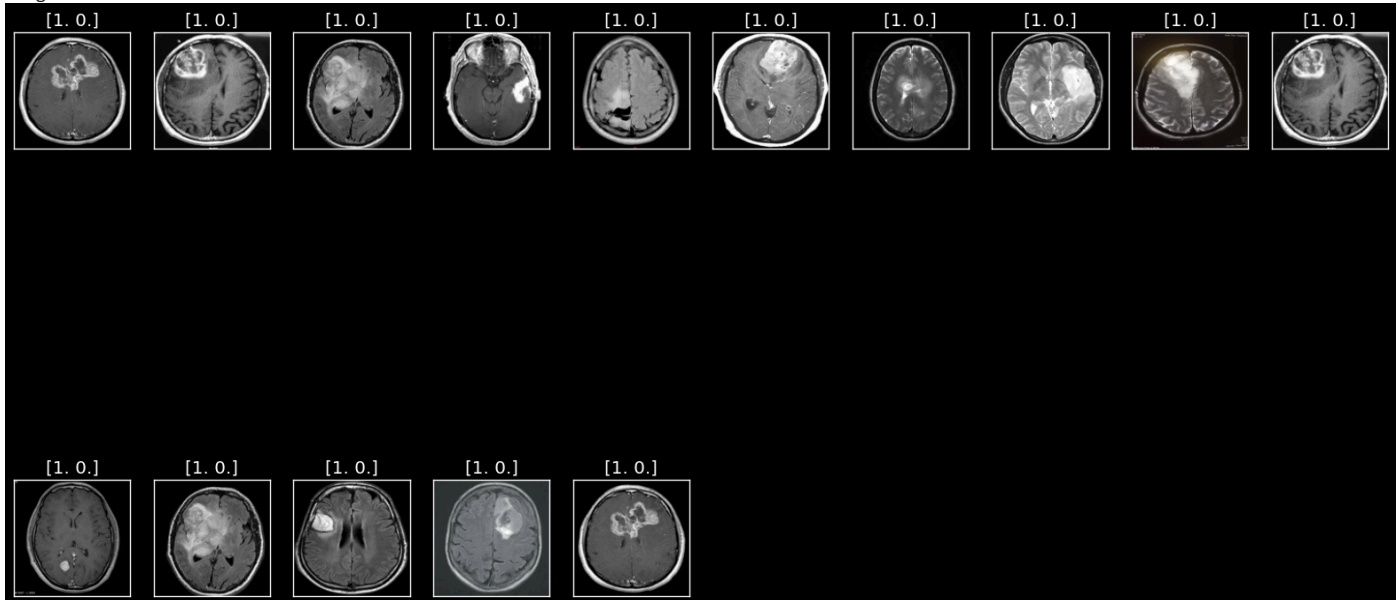
```
for i in range (15):
```

```
    ax=fig.add_subplot(2,10,i+1,xticks=[],yticks=[])
```

```
    ax.imshow(np.squeeze(x_train[i]),cmap='gray')
```

```
    ax.set_title(y_train[i])
```

```
↪ <Figure size 2000x1200 with 0 Axes>
```



```
img_shape=x_train.shape[1:]
```

```
print(img_shape)
```

```
↪ (128, 128, 3)
```

```
yes_dir = '/content/brain_tumor_dataset/yes'
```

```
no_dir = '/content/brain_tumor_dataset/no'
```

```
def load_data(yes_dir, no_dir, target_size=(128, 128)):
```

```
    pass
```

```
def unet(input_size=(128, 128, 1)):
```

```
    inputs = tf.keras.layers.Input(input_size)
```

```
    # Contracting path (Encoder)
```

```
    c1 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
```

```
    c1 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c1)
```

```
    p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)
```

```

c2 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
c2 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)

c3 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(p2)
c3 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same')(p3)
c4 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D((2, 2))(c4)

# Bottleneck
c5 = tf.keras.layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(p4)
c5 = tf.keras.layers.Conv2D(1024, (3, 3), activation='relu', padding='same')(c5)

# Expansive path (Decoder)
u6 = tf.keras.layers.Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same')(c5)
u6 = tf.keras.layers.concatenate([u6, c4])
c6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same')(u6)
c6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu', padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7, c3])
c7 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(u7)
c7 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu', padding='same')(c7)

u8 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8, c2])
c8 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(u8)
c8 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(c8)

u9 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = tf.keras.layers.concatenate([u9, c1])
c9 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(u9)
c9 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(c9)

outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
model = tf.keras.Model(inputs=[inputs], outputs=[outputs])

return model

model = unet(input_size=(128, 128, 1))
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4), loss='binary_crossentropy', metrics=['accuracy', tf.keras.metrics.Mean
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 128, 128, 1)	0	-
conv2d (Conv2D)	(None, 128, 128, 64)	640	input_layer[0][0]
conv2d_1 (Conv2D)	(None, 128, 128, 64)	36,928	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 64, 64, 128)	73,856	max_pooling2d[0][0]
conv2d_3 (Conv2D)	(None, 64, 64, 128)	147,584	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 128)	0	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 32, 32, 256)	295,168	max_pooling2d_1[0][0]
conv2d_5 (Conv2D)	(None, 32, 32, 256)	590,080	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 256)	0	conv2d_5[0][0]
conv2d_6 (Conv2D)	(None, 16, 16, 512)	1,180,160	max_pooling2d_2[0][0]
conv2d_7 (Conv2D)	(None, 16, 16, 512)	2,359,808	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 512)	0	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 8, 8, 1024)	4,719,616	max_pooling2d_3[0][0]
conv2d_9 (Conv2D)	(None, 8, 8, 1024)	9,438,208	conv2d_8[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 16, 16, 512)	2,097,664	conv2d_9[0][0]
concatenate (Concatenate)	(None, 16, 16, 1024)	0	conv2d_transpose[0][0]... conv2d_7[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 512)	4,719,104	concatenate[0][0]
conv2d_11 (Conv2D)	(None, 16, 16, 512)	2,359,808	conv2d_10[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 32, 32, 256)	524,544	conv2d_11[0][0]
concatenate_1 (Concatenate)	(None, 32, 32, 512)	0	conv2d_transpose_1[0]... conv2d_5[0][0]
conv2d_12 (Conv2D)	(None, 32, 32, 256)	1,179,904	concatenate_1[0][0]
conv2d_13 (Conv2D)	(None, 32, 32, 256)	590,080	conv2d_12[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 64, 64, 128)	131,200	conv2d_13[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 256)	0	conv2d_transpose_2[0]... conv2d_3[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 128)	295,040	concatenate_2[0][0]
conv2d_15 (Conv2D)	(None, 64, 64, 128)	147,584	conv2d_14[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 128, 128, 64)	32,832	conv2d_15[0][0]
concatenate_3 (Concatenate)	(None, 128, 128, 128)	0	conv2d_transpose_3[0]... conv2d_1[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 64)	73,792	concatenate_3[0][0]
conv2d_17 (Conv2D)	(None, 128, 128, 64)	36,928	conv2d_16[0][0]
conv2d_18 (Conv2D)	(None, 128, 128, 1)	65	conv2d_17[0][0]

```
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(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/python3.11/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_shape\_tuple` argument to the constructor of Conv2D, Conv3D, Conv2DTranspose, or Conv3DTranspose. Use the `input\_shape` argument of the `compile` method instead.

super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 128, 128, 32)	416
conv2d_20 (Conv2D)	(None, 128, 128, 32)	4,128
batch_normalization (BatchNormalization)	(None, 128, 128, 32)	128
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 32)	0
dropout (Dropout)	(None, 64, 64, 32)	0
conv2d_21 (Conv2D)	(None, 64, 64, 64)	8,256
conv2d_22 (Conv2D)	(None, 64, 64, 64)	16,448
batch_normalization_1 (BatchNormalization)	(None, 64, 64, 64)	256
max_pooling2d_5 (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)

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

Start coding or [generate](#) with AI.

```

history = model.fit(x_train, y_train, epochs=30, batch_size=40, verbose=1, validation_data=(x_test, y_test))

```



Epoch 1/30

61/61 ————— 222s 4s/step - loss: 7.5346 - val\_loss: 2.9438

Epoch 2/30

61/61 ————— 253s 3s/step - loss: 0.2501 - val\_loss: 0.4382

Epoch 3/30

61/61 ————— 271s 4s/step - loss: 0.0888 - val\_loss: 0.2260

Epoch 4/30

61/61 ————— 210s 3s/step - loss: 0.0198 - val\_loss: 0.1982

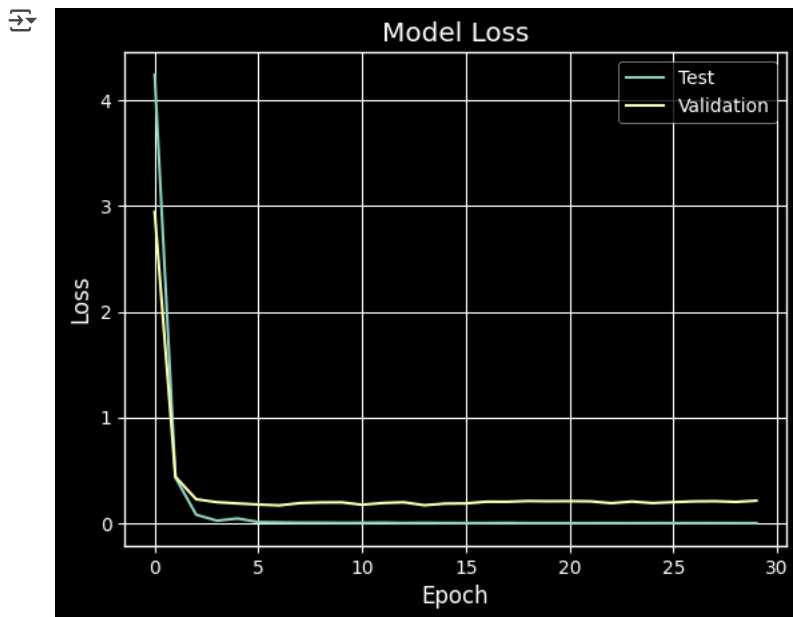
Epoch 5/30

61/61 ————— 263s 3s/step - loss: 0.0677 - val\_loss: 0.1861

Epoch 6/30  
**61/61** ————— 271s 4s/step - loss: 0.0088 - val\_loss: 0.1759  
Epoch 7/30  
**61/61** ————— 252s 3s/step - loss: 0.0080 - val\_loss: 0.1680  
Epoch 8/30  
**61/61** ————— 219s 4s/step - loss: 0.0045 - val\_loss: 0.1895  
Epoch 9/30  
**61/61** ————— 253s 3s/step - loss: 0.0039 - val\_loss: 0.1956  
Epoch 10/30  
**61/61** ————— 211s 3s/step - loss: 0.0030 - val\_loss: 0.1966  
Epoch 11/30  
**61/61** ————— 272s 4s/step - loss: 0.0024 - val\_loss: 0.1740  
Epoch 12/30  
**61/61** ————— 249s 3s/step - loss: 0.0039 - val\_loss: 0.1905  
Epoch 13/30  
**61/61** ————— 226s 4s/step - loss: 0.0015 - val\_loss: 0.1972  
Epoch 14/30  
**61/61** ————— 246s 3s/step - loss: 0.0018 - val\_loss: 0.1694  
Epoch 15/30  
**61/61** ————— 265s 3s/step - loss: 0.0029 - val\_loss: 0.1849  
Epoch 16/30  
**61/61** ————— 267s 4s/step - loss: 6.0890e-04 - val\_loss: 0.1863  
Epoch 17/30  
**61/61** ————— 265s 4s/step - loss: 0.0010 - val\_loss: 0.2030  
Epoch 18/30  
**61/61** ————— 266s 4s/step - loss: 0.0027 - val\_loss: 0.2022  
Epoch 19/30  
**61/61** ————— 249s 3s/step - loss: 2.4133e-04 - val\_loss: 0.2093  
Epoch 20/30  
**61/61** ————— 274s 4s/step - loss: 4.8587e-04 - val\_loss: 0.2072  
Epoch 21/30  
**61/61** ————— 249s 3s/step - loss: 2.5335e-04 - val\_loss: 0.2083  
Epoch 22/30  
**61/61** ————— 270s 4s/step - loss: 4.4174e-04 - val\_loss: 0.2064  
Epoch 23/30  
**61/61** ————— 254s 3s/step - loss: 5.8473e-04 - val\_loss: 0.1886  
Epoch 24/30  
**61/61** ————— 262s 3s/step - loss: 5.2975e-04 - val\_loss: 0.2045  
Epoch 25/30  
**61/61** ————— 272s 4s/step - loss: 5.8412e-04 - val\_loss: 0.1895  
Epoch 26/30  
**61/61** ————— 211s 3s/step - loss: 4.8014e-04 - val\_loss: 0.1988  
Epoch 27/30  
**61/61** ————— 261s 3s/step - loss: 5.6645e-04 - val\_loss: 0.2060  
Epoch 28/30  
**61/61** ————— 262s 3s/step - loss: 2.0333e-04 - val\_loss: 0.2077  
Epoch 29/30  
**61/61** ————— 273s 4s/step - loss: 4.5826e-04 - val\_loss: 0.2008

```
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()
```

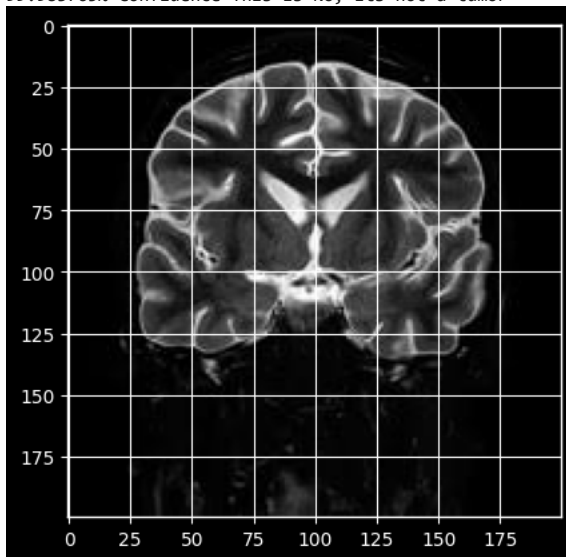




```
def names(number):
    if number==0:
        return 'Its a Tumor'
    else:
        return 'No, Its not a tumor'
```

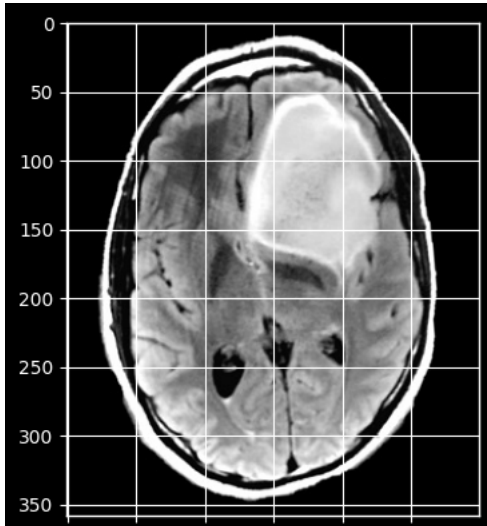
```
from matplotlib.pyplot import imshow
img = Image.open(r"/content/brain_tumor_dataset/no/19 no.jpg")
x = np.array(img.resize((128,128)))
x = x.reshape(1,128,128,3)
res = model.predict_on_batch(x)
classification = np.where(res == np.amax(res))[1][0]
imshow(img)
print(str(res[0][classification]*100) + '% Confidence This Is ' + names(classification))
```

99.983765% Confidence This Is No, Its not a tumor



```
from matplotlib.pyplot import imshow
img = Image.open(r"/content/brain_tumor_dataset/yes/Y153.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 Its a Tumor



```
def preprocess_image(file_path, target_size=(128, 128)):
    """Load and preprocess the image."""
    image = tf.keras.preprocessing.image.load_img(file_path, target_size=target_size) # Changed color_mode to default (RGB)
    image = tf.keras.preprocessing.image.img_to_array(image)
    image = image / 255.0 # Normalize to [0, 1]
    return np.expand_dims(image, axis=0) # Add batch dimension

def predict_and_visualize(image_path, model, target_size=(128, 128)):
    test_image = preprocess_image(image_path, target_size)
    predicted_mask = model.predict(test_image)

    # Load the original image for visualization
    original_image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size) # Changed color_mode to default (RGB)
    original_image = tf.keras.preprocessing.image.img_to_array(original_image)

    # Ensure predicted_mask has the correct number of dimensions
    predicted_mask = np.expand_dims(predicted_mask, axis=-1) # Add a channel dimension if missing

    # Resize predicted mask to match original image dimensions
    predicted_mask_resized = tf.image.resize(predicted_mask,
                                              (original_image.shape[0], original_image.shape[1]))
    predicted_mask_resized_np = np.squeeze(predicted_mask_resized.numpy())

    # Visualize the raw prediction mask and overlay
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(np.squeeze(original_image), cmap='gray')
    plt.title('Original Image')

    plt.subplot(1, 3, 2)
    plt.imshow(predicted_mask_resized_np, cmap='jet') # Using jet colormap to highlight prediction
    plt.colorbar()
    plt.title('Raw Prediction Mask')

    plt.subplot(1, 3, 3)
    plt.imshow(np.squeeze(original_image), cmap='gray')
    plt.imshow(predicted_mask_resized_np, cmap='jet', alpha=0.5)
    plt.title('Overlay')

    plt.show()
```