```
from zipfile import ZipFile


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


with ZipFile('/content/drive/MyDrive/archive (2).zip','r') as zipObj:
  zipObj.extractall('/content/drive/MyDrive/brain_tumor')


import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from PIL import Image



# Set up plotting style
plt.style.use('dark_background')


# One-hot encoding for labels
encoder = OneHotEncoder()
encoder.fit([[0], [1]])
```

```
OneHotEncoder
OneHotEncoder()
```

```
# Adjusted paths for Google Colab
data_path_yes = '/content/drive/MyDrive/brain_tumor/brain_tumor_dataset/yes'
data_path_no = '/content/drive/MyDrive/brain_tumor/brain_tumor_dataset/no'


# This cell updates result list for images with tumor
data = []
result = []


# Images with tumor
for file in os.listdir(data_path_yes):
    if file.endswith('.jpg'):
        img = Image.open(os.path.join(data_path_yes, file))
        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())



# Images without tumor
for file in os.listdir(data_path_no):
    if file.endswith('.jpg'):
        img = Image.open(os.path.join(data_path_no, file))
        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())



data = np.array(data)
result = np.array(result).reshape(-1, 2)



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

```
# CNN Model architecture
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')
```

```
# Print model summary
print(model.summary())
```

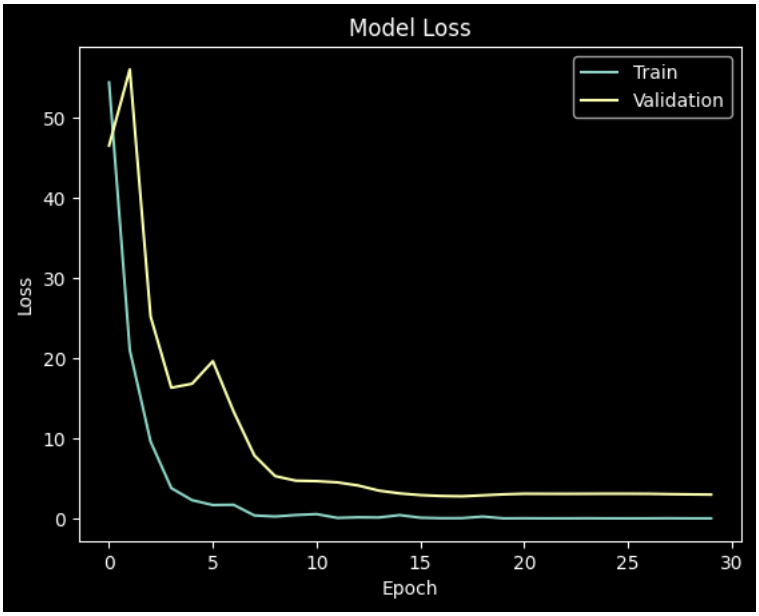
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      416

 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 (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_1 (Dropout)         (None, 32, 32, 64)        0

 flatten (Flatten)           (None, 65536)             0

 dense (Dense)               (None, 512)               33554944

 dropout_2 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 2)                 1026

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

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

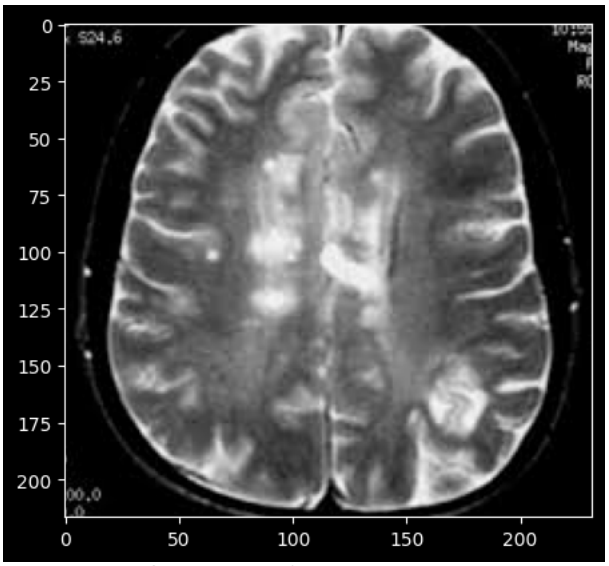
```
Epoch 2/30
3/3 [==============================] - 10s 4s/step - loss: 21.0050 - val_loss: 56.0838
Epoch 3/30
3/3 [==============================] - 9s 3s/step - loss: 9.5963 - val_loss: 25.2169
Epoch 4/30
3/3 [==============================] - 11s 4s/step - loss: 3.7772 - val_loss: 16.3145
Epoch 5/30
3/3 [==============================] - 11s 4s/step - loss: 2.2707 - val_loss: 16.8186
```

```
Epoch 7/30
3/3 [==============================] - 11s 4s/step - loss: 1.6956 - val_loss: 13.3422
Epoch 8/30
3/3 [==============================] - 10s 4s/step - loss: 0.3652 - val_loss: 7.8534
Epoch 9/30
3/3 [==============================] - 9s 3s/step - loss: 0.2341 - val_loss: 5.2808
Epoch 10/30
3/3 [==============================] - 11s 3s/step - loss: 0.4189 - val_loss: 4.7033
Epoch 11/30
3/3 [==============================] - 10s 4s/step - loss: 0.5299 - val_loss: 4.6518
Epoch 12/30
3/3 [==============================] - 9s 3s/step - loss: 0.0574 - val_loss: 4.4954
Epoch 13/30
3/3 [==============================] - 11s 3s/step - loss: 0.1313 - val_loss: 4.1051
Epoch 14/30
3/3 [==============================] - 10s 4s/step - loss: 0.1082 - val_loss: 3.4646
Epoch 15/30
3/3 [==============================] - 8s 3s/step - loss: 0.4111 - val_loss: 3.1174
Epoch 16/30
3/3 [==============================] - 11s 3s/step - loss: 0.0801 - val_loss: 2.9036
Epoch 17/30
3/3 [==============================] - 10s 4s/step - loss: 0.0229 - val_loss: 2.7960
Epoch 18/30
3/3 [==============================] - 8s 3s/step - loss: 0.0286 - val_loss: 2.7536
Epoch 19/30
3/3 [==============================] - 10s 3s/step - loss: 0.2225 - val_loss: 2.8785
Epoch 20/30
3/3 [==============================] - 11s 4s/step - loss: 0.0016 - val_loss: 2.9990
Epoch 21/30
3/3 [==============================] - 8s 3s/step - loss: 0.0130 - val_loss: 3.0752
Epoch 22/30
3/3 [==============================] - 10s 3s/step - loss: 0.0033 - val_loss: 3.0674
Epoch 23/30
3/3 [==============================] - 10s 4s/step - loss: 0.0044 - val_loss: 3.0621
Epoch 24/30
3/3 [==============================] - 8s 3s/step - loss: 0.0132 - val_loss: 3.0690
Epoch 25/30
3/3 [==============================] - 10s 3s/step - loss: 0.0015 - val_loss: 3.0761
Epoch 26/30
3/3 [==============================] - 11s 4s/step - loss: 1.8958e-04 - val_loss: 3.0747
Epoch 27/30
3/3 [==============================] - 8s 3s/step - loss: 0.0024 - val_loss: 3.0664
Epoch 28/30
3/3 [==============================] - 10s 3s/step - loss: 0.0138 - val_loss: 3.0243
Epoch 29/30
3/3 [==============================] - 10s 4s/step - loss: 0.0020 - val_loss: 2.9929
Epoch 30/30
3/3 [==============================] - 9s 3s/step - loss: 8.3261e-04 - val_loss: 2.9683
```

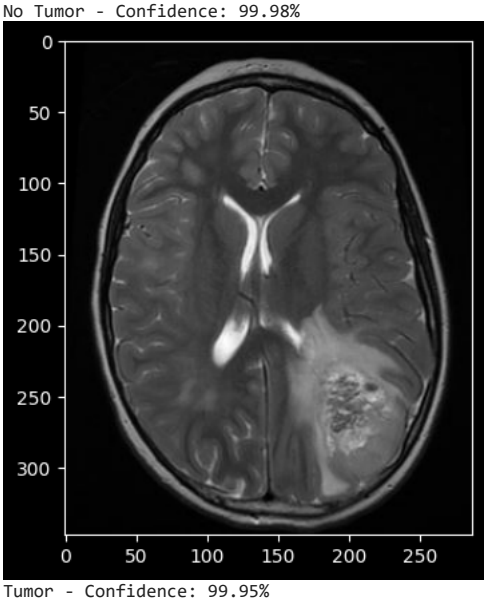
```
# Plotting the training history
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```




```
# Function to interpret prediction results
def names(number):
    if number == 0:
        return 'Tumor'
    else:
        return 'No Tumor'


# Test images
test_images = [
    '/content/drive/MyDrive/brain_tumor/brain_tumor_dataset/no/15 no.jpg',
    '/content/drive/MyDrive/brain_tumor/brain_tumor_dataset/yes/Y100.JPG'
]


# Predictions for test images
for img_path in test_images:
    img = Image.open(img_path)
    x = np.array(img.resize((128, 128)))
    x = x.reshape(1, 128, 128, 3)
    res = model.predict_on_batch(x)
    classification = np.argmax(res)
    plt.imshow(img)
    plt.show()
    print(f"{names(classification)} - Confidence: {res[0][classification]*100:.2f}%")
```



No Tumor - Confidence: 99.98%



Tumor - Confidence: 99.95%

```
def predict_tumor_status(image_path):
    img = Image.open(image_path)
    img = img.resize((128, 128))
    x = np.array(img)
    # Ensure the array has 3 dimensions (for RGB channels)
    if len(x.shape) == 2:
        x = np.expand_dims(x, axis=2)  # Add third dimension for single-channel images
        x = np.repeat(x, 3, axis=2)  # Convert to 3 channels (RGB)
    x = x.reshape(1, 128, 128, 3)
    res = model.predict_on_batch(x)
    classification = np.argmax(res)
    result_label = names(classification)
    confidence = res[0][classification] * 100
    return img, result_label, confidence



import matplotlib.pyplot as plt

# Function to interactively get prediction for multiple images
def get_predictions():
    num_images = int(input("How many images do you want to predict? "))
    image_paths = []
    for i in range(num_images):
        image_path = input(f"Enter the path of image {i + 1}: ")
        image_paths.append(image_path)

    for image_path in image_paths:
        img, result_label, confidence = predict_tumor_status(image_path)
```

```
            # Determine colors for the bar graph
            if result_label == 'Tumor':
                tumor_confidence = confidence
                no_tumor_confidence = 100 - confidence
            else:
                tumor_confidence = 100 - confidence
                no_tumor_confidence = confidence

            # Display the image
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
            fig.suptitle(f"Predicted: {result_label}\nConfidence: {confidence:.2f}%", fontsize=14, color='black')

            # Plot the image
            ax1.imshow(img)
            ax1.set_title('MRI Image')
            ax1.axis('off')

            # Plot the bar graph
            labels = ['No Tumor', 'Tumor']
            sizes = [no_tumor_confidence, tumor_confidence]
            colors = ['#6495ED', '#FF6347'] if result_label == 'Tumor' else ['#FF6347', '#6495ED']
            bars = ax2.bar(labels, sizes, color=colors)
            ax2.set_title('Prediction Confidence')
            ax2.set_ylabel('Confidence (%)')
            ax2.set_ylim(0, 100)

            # Add confidence scores on top of bars
            for bar, size in zip(bars, sizes):
                ax2.text(bar.get_x() + bar.get_width() / 2, size + 1, f'{size:.2f}%', ha='center', va='bottom', color='black')

            # Add final prediction result at the end of the image
            result_color = 'red' if result_label == 'Tumor' else 'green'
            ax1.text(0.5, -0.1, f"Final Prediction: {result_label}", ha="center", va="center", size=12, color=result_color, transform=ax1.transAx

            plt.show()

    # Example usage of the interactive prediction function
    get_predictions()
```

```
How many images do you want to predict? 3
Enter the path of image 1: /content/drive/MyDrive/brain_tumor/brain_tumor_dataset/no/no
Enter the path of image 2: /content/drive/MyDrive/brain_tumor/brain_tumor_dataset/no/no
Enter the path of image 3: /content/drive/MyDrive/brain_tumor/brain_tumor_dataset/yes/Y1
```