**Brain Tumor Detection using MRI Scans**

The provided code is an implementation of different deep learning models for brain tumor detection using MRI scans of patients. The code uses the Keras and TensorFlow libraries in Python for building and training the models. Here is the documentation for the code:

**Data Preparation**

The code uses two dictionaries, tumor\_images\_dict and tumor\_labels\_dict, to store the image paths and corresponding labels for tumor and healthy samples.

The cv2 library is used to read and resize the images to a fixed size of 224x224 pixels.

The resized images are stored in a list X, and their corresponding labels are stored in a list y.

Example Image for a healthy brain:



Example Image for a brain tumor:

****

**Train-Test Split**

The train\_test\_split function from the sklearn.model\_selection module is used to split the data into training and testing sets with a 75% - 25% split ratio.

The images and labels are split into X\_train, X\_test, y\_train, and y\_test for training and testing purposes.

**Artificial Neural Network (ANN)**

An ANN model with multiple dense layers is created using the keras.Sequential class.

The input shape of the model is set to (224, 224, 3) which represents the shape of the resized images.

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The fit function is called on the model with the training data to train the model for 5 epochs.

**Convolutional Neural Network (CNN) - Model 1**

A CNN model with convolutional and pooling layers followed by dense layers is created using the Sequential class.

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The fit function is called on the model with the scaled training data to train the model for 12 epochs.

**CNN with Data Augmentation and Dropout - Model 2**

Data augmentation techniques such as random horizontal flipping, rotation, and zooming are applied to the images using the layers.experimental.preprocessing module.

A CNN model with convolutional, pooling, dropout, and dense layers is created using the Sequential class.

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The fit function is called on the model with the scaled training data to train the model for 35 epochs.

**CNN with Regularization - Model 3**

A CNN model with convolutional, pooling, dropout, and dense layers is created using the Sequential class.

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The fit function is called on the model with the scaled training data to train the model for 12 epochs.

**MobileNet v2**

The MobileNet v2 model is loaded using the hub.KerasLayer class from the TensorFlow Hub.

A dense output layer with the number of classes as the number of neurons is added on top of the feature extraction layer.

The model is compiled with the Adam optimizer, sparse categorical cross-entropy loss function, and accuracy as the evaluation metric.

The fit function is called on the model with the scaled training data to train the model for 10 epochs.

**Model Evaluation**

The accuracy of each model is printed after training to evaluate its performance on the training data.

Further evaluation of the models can be done by predicting the labels for the test data using the trained models and calculating metrics such as confusion matrix and classification report.

**Classification Report**



**Conclusion**

In conclusion, the provided code implements multiple models for brain tumor detection using MRI scans of patients. The code includes an Artificial Neural Network (ANN) model, four Convolutional Neural Network (CNN) models with different configurations including dropout, data augmentation, regularization, and transfer learning using MobileNet V2. The code also includes model evaluation and prediction steps to assess the performance of the trained models. It's important to note that the accuracy and performance of the models may vary depending on the specific dataset and problem, and it's recommended to fine-tune the models and experiment with different hyperparameters and techniques to achieve the best results for a specific use case.