IDENTIFICATION AND CLASSIFICATION OF BRAIN TUMOR FROM MRI IMAGES

PURPOSE OF PROJECT: The project aims to perform morphological operations on the MRI images and identify the presence of tumor using Image Processing and classification of the tumor using LeNet-5 and CNN

AUTHORS: Kritika Singh (2017132), Suprabh Sharma (2017263)

PROJECT DETAILS:

Preprocessing Step
After the segmentation of the image, the main aim was to select a few seed points and attach them with their neighbouring pixels having identical properties. The similarity was the measure of the difference between pixel's intensity and the region's mean,d. The pixel with the smallest difference was allocated to the respective region.

Morphological Operations
Simple operations of Dilation and Erosion were performed on the image to further process the image.

Feature Extraction
1. **Feature Extraction Using DWT**: 2-D wavelet transform was applied to the image from which 4 subbands namely LL, LH, HL and HH were created. We used the low level image and the process was repeated until we received the desired level of resolution.

2. **Feature Extraction Using GLCM**: GLCM was initiated and the textual features like contrast, correlation, energy, homogeneity, entropy and variance were obtained from LL and HL bands of the first four levels of the wavelet decomposition.

Classification of Brain tumor MRI Images using Lenet-5 and CNN

Part 1
Keras is used for preprocessing of the image.
We are importing sequential and flatten because, in Keras, this is a typical process for building a CNN architecture:
Flatten operation is performed as part of a model built using the Sequential() function which lets you sequentially add on layers to create your neural network model.

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Initializing the network using the Sequential Class

```
Classifier = Sequential()
```

Adding convolutional and pooling layers:- **Modified LeNet Network**

- **INPUT Layer**

The first is the data INPUT layer. The size of the input image is uniformly normalized to 64 * 64.

- **C1 layer-convolutional layer**

Input picture: 64 * 64
Convolution kernel size: 5 * 5
Convolution kernel types: 6
Output featuremap size: 60 * 60 (64-5 + 1) = 60
Number of neurons: 60 * 60 * 6
Trainable parameters: (5 * 5 + 1)*6 , (5 * 5 = 25 unit parameters and one bias parameter per filter, a total of 6 filters)
Number of connections: (5 * 5 + 1)* 6 * 60 * 60 = 5,61,600

- **S2 layer-pooling layer (downsampling layer)**

Input: 60 * 60
Sampling area: 3 * 3
Sampling method: 4 inputs are added, multiplied by a trainable parameter, plus a trainable offset. Results via relu
Sampling type: 6
Output feature Map size: 30 * 30 (60/2)
Number of neurons: 30 * 30 *6
Trainable parameters: 3 * 6 (the weight of the sum + the offset)
Number of connections: (3 * 3 + 1) * 6 * 30 * 30

- **C3 layer-convolutional layer**

Input: all 6 or several feature map combinations in S2
Convolution kernel size: 3 * 3
Convolution kernel type: 16
Output featureMap size: 28 * 28 (30 - 3 + 1) = 28

- **S4 layer-pooling layer (downsampling layer)**

Input: 28 * 28
Sampling area: 2 * 2
Sampling type: 16
Output featureMap size: 14 * 15 (28/2)

● **C5 layer - Fully connected convolutional layer**
The fifth layer (C5) is a fully connected convolutional layer with 120 feature maps each of size 1×1. Each of the 120 units in C5 is connected to all the 400 nodes (5x5x16) in the fourth layer S4.

● **F6 layer-fully connected layer**
Input: C3 120-dimensional vector
Calculation method: Calculating the dot product between the input vector and the weight vector, plus an offset, and the result is output through the sigmoid function.
Trainable parameters: 84 * (120 + 1) = 10164

```
classifier.add(Conv2D(6, kernel_size=(5,5), activation='relu', input_shape=(64,64,
3)))
classifier.add( MaxPooling2D( pool_size=(3,3)))
classifier.add(Conv2D(16, kernel_size=(3,3), activation='relu'))
classifier.add( MaxPooling2D( pool_size=(2,2)))
classifier.add(Conv2D(16, kernel_size=(5,5), activation='relu'))
classifier.add( MaxPooling2D( pool_size=(2,2)))
```

Flattening and adding two fully connected layers:

```
classifier.add(Flatten())
classifier.add(Dense(120, activation='relu'))
classifier.add(Dense(84, activation='relu'))
```

Compiling the model

```
classifier.compile(loss=keras.metrics.categorical_crossentropy,
optimizer=keras.optimizers.Adam(), metrics=['accuracy'])
```

A Convolutional Neural Network (CNN) architecture has three main parts:

- A **convolutional layer** that extracts features from a source image. Convolution helps with blurring, sharpening, edge detection, noise reduction, or other operations that can help the machine to learn specific characteristics of an image.
- A **pooling layer** that reduces the image dimensionality without losing important features or patterns.
- A **fully connected layer** also known as the dense layer, in which the results of the convolutional layers are fed through one or more neural layers to generate a prediction.

Imported confusion matrix to get a better understanding of how the model predicted the data.

Part 2 - Fitting the CNN to the images

ImageDataGenerator class
Generate batches of tensor image data with real-time data augmentation.
The data will be looped over (in batches) in both training and test data.

- **horizontal_flip**: Boolean. Randomly flip inputs horizontally.
- **shear_range**: Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)
- **zoom_range**: Float or [lower, upper]. The range for random zoom.

```
test_data = ImageDataGenerator(rescale = 1./255,
                               shear_range = 0.2,
                               zoom_range = 0.2,
                               horizontal_flip = True)
```

Flow_from_directory method
Takes the path to a directory & generates batches of augmented data.

- **directory**: string, a path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM, or TIF images inside each of the subdirectories directory trees will be included in the generator..
- **target_size**: Tuple of integers (height, width), defaults to (256, 256). The dimensions to which all images found will be resized.
- **class_mode**: Determines the type of label arrays that are returned: - "categorical" will be 2D one-hot encoded labels, - "binary" will be 1D binary labels, "sparse" will be 1D integer labels, - "input" will be images identical to input images (mainly used to work with autoencoders). - If None, no labels are returned (the generator will only yield batches of image data, which is useful to use with model.predict_generator()).
- **batch_size**: Size of the batches of data (default: 32).

```
test_set = test_data.flow_from_directory(r"C:\Users\kriti\OneDrive\Desktop\6th
sem\IP CS313a\IP project\Matlab Code and dataset\Test",
                                         target_size = (64, 64),
                                         batch_size = 32,
                                         class_mode = 'categorical')
```

Evaluating the model

```
classifier.fit_generator(training_set,
                         steps_per_epoch = 10,
                         epochs = 30,
                         validation_data = test_set,
                         validation_steps = 5)
```

Part 3 - Making new predictions
Importing the image which needs to get tested.

```
test_image = image.load_img(r"C:\Users\kriti\OneDrive\Desktop\brain
mri\ex6.jpg", target_size = (64,64))
```

And predict the model by setting the indices of probability of the test image.
For example, if the image matches the 'no' class it will be shown 'no' with the array of [1,0] similarly with 'yes' class an array of [0,1].

```
result = classifier.predict(test_image) training_set.class_indices
print(result)
if result[0][0] == 1:
   prediction = 'NO'
   print(prediction)
else:
   prediction = 'YES'
   print(prediction)
```

Part 4 - plot the confusion matrix
To plot the confusion matrix between train dataset and the cross-validation data set as to how it worked in predicting.
For this, we used sklearn to import confusion matrix and fastai for implementation of learn and ClassificationInterpretation.

```
from sklearn.metrics import confusion_matrix
from fastai import *
from fastai.vision import *
```

So once again we trained the model using learn and plotted it using intrep.

```
learn.lr_find()
learn.fit_one_cycle(6,1e-2)
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix(figsize=(8,8), dpi=50)
```