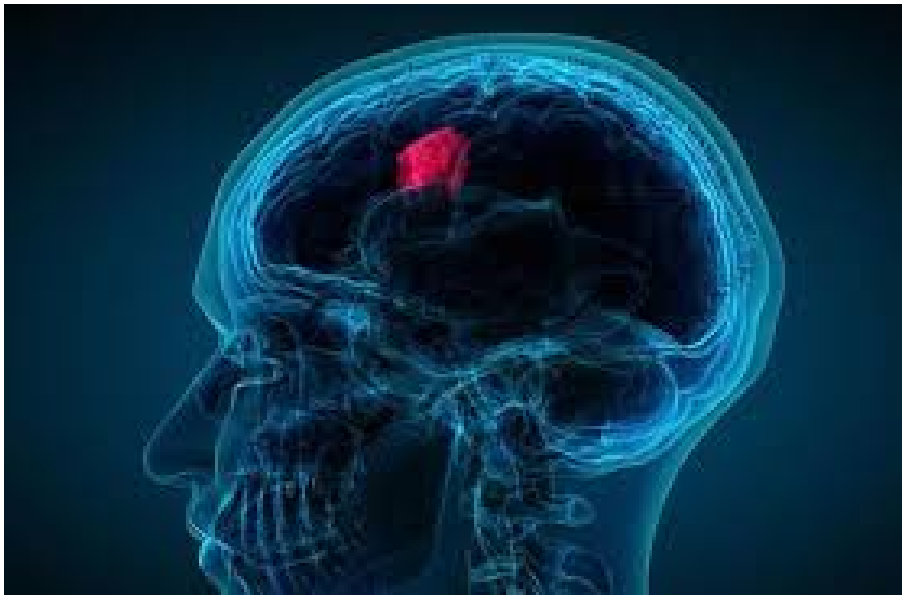
Brain tumor classification using CNN

AI Mini Project



Group Members :
GUERZIZ Ines
CHAALEL Idris
BRAHIMI Dounia
SELAMA Abdessamie

ESI-SBA 2022-2023

Introduction

Brain tumor is one of the most common and deadly diseases in the world. Detection of the brain tumor in its early stage is the key to its cure. Early detection and classification of brain tumors is an important research domain in the field of medical imaging and accordingly helps in selecting the most convenient treatment method to save patients life therefore. With the development of Artificial Intelligence (AI) and Soft Computing Techniques, Computer-Aided Diagnosis (CAD) attracts more and more attention for brain tumor diagnosis. Accuracy and efficiency are two major issues in designing CAD systems. In brain Magnetic Resonance Imaging (MRI), the tumor may appear clearly but for further treatment, the physician also needs the quantification of the tumor area. The computer and image processing techniques can provide significant help in analyzing the tumor area. In this work features based on the shape and texture of the image were tested for analysis and classification of brain tumors. After preprocessing , the ResNet50 model is used to classify MRI brain tumor images into Glioma,Meningioma,pituitary and no tumor classes .

Dataset

This dataset has been taken from kaggle. It contains 7022 images of human brain MRI images which are classified into 4 classes: glioma - meningioma - no tumor and pituitary.

- 5712 images for training
- 1311 images for testing (approximately 20% from the dataset)

This dataset is a combination of the following three datasets:

- figshare
- SARTAJ dataset
- Br35H

No tumor class images were taken from the Br35H dataset.
SARTAj dataset has a problem the the glioma class images are not categorized correctly

1.

1. A meningioma is **a primary central nervous system (CNS) tumor**. This means it begins in the brain or spinal cord. Overall, meningiomas are the most common type of primary brain tumor. However, higher grade meningiomas are very rare.

2. Glioma is **a type of tumor that occurs in the brain and spinal cord**. Gliomas begin in the gluey supportive cells (glial cells) that surround nerve cells and help them function. Three types of glial cells can produce tumors.

3. Your pituitary gland (also known as hypophysis) is **a small, pea-sized gland located at the base of your brain below your hypothalamus**. It sits in its own little chamber under your brain known as the sella turcica. It's a part of your endocrine system and is in charge of making several essential hormones.

Pre-Processing

Quality decisions must be based on quality data, So the pre-processing part has to be done to get satisfactory results, In order to to preprocess the dataset we are going to input the samples to the pipeline below :

Conversion to Grayscale:

Image obtained after scanning, usually in RGB (Red, Green, and Blue) color format. The image contains three independent planes namely Red, Green, and Blue components. In the case of RGB images, pixel intensity is represented by the combination of these three plane intensity values. In the case of grayscale image pixel values represented by the intensity values range from 0 to 256. Grey scale image ranging from black to white with different shades of gray. Light intensity at each pixel in a grayscale image lies in a single band of electromagnetic spectrum.

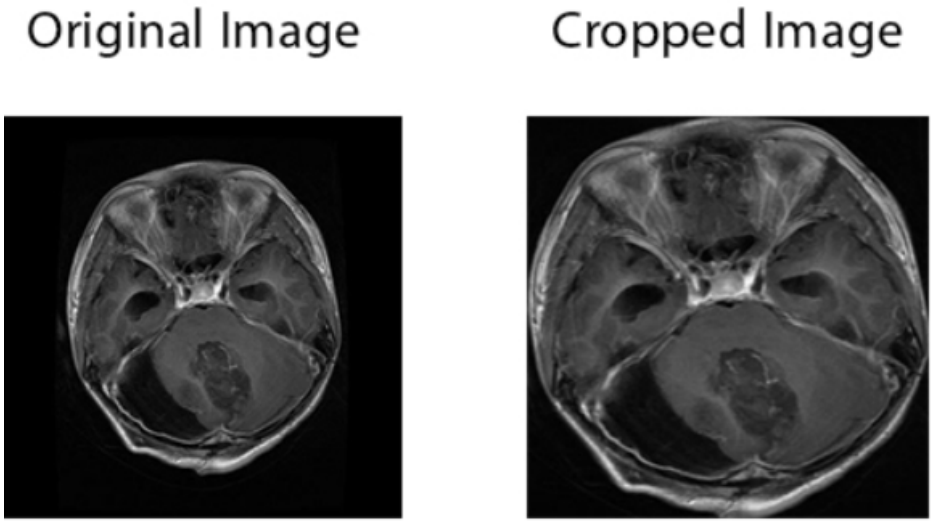
Cropping the image:

Image Cropping is a common photo manipulation process, which improves the overall composition by removing unwanted regions.

Starting by Threshold the image, then perform a series of erosions then dilations to remove any small regions of noise, after selecting the largest thresholded in the image we resize it relying on the extreme points.

```python
def crop_img(img):

    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    gray = cv2.GaussianBlur(gray, (3, 3), 0)


    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]
    thresh = cv2.erode(thresh, None, iterations=2)
    thresh = cv2.dilate(thresh, None, iterations=2)

    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)

    extLeft = tuple(c[c[:, :, 0].argmin()][0])
    extRight = tuple(c[c[:, :, 0].argmax()][0])
    extTop = tuple(c[c[:, :, 1].argmin()][0])
    extBot = tuple(c[c[:, :, 1].argmax()][0])
    ADD_PIXELS = 0
    new_img = img[extTop[1]-ADD_PIXELS:extBot[1]+ADD_PIXELS, extLeft[0]-ADD_PIXELS:extRight[0]+ADD_PIXELS].copy()

    return new_img
```

Finally finishing the first part by resizing the images and saving the data to a new cleaned data folder.



After loading the data, we performed “bilateralFilter” to remove images noise and “applyColorMap” to produce a pseudo colored image then split it into training and testing parts after the normalization of the samples (without targets).

```
for label in labels:
    trainPath = os.path.join('cleaned\Training',label)
    for file in tqdm(os.listdir(trainPath)):
        image = cv2.imread(os.path.join(trainPath, file),0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
        image = cv2.resize(image, (image_size, image_size))
        x_train.append(image)
        y_train.append(labels.index(label))

    testPath = os.path.join('cleaned/Testing',label)
    for file in tqdm(os.listdir(testPath)):
        image = cv2.imread(os.path.join(testPath, file),0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
        image = cv2.resize(image, (image_size, image_size))
        x_test.append(image)
        y_test.append(labels.index(label))
```

Normalization:

In computer vision, the pixel normalization technique is often used to speed up model learning. The normalization of an image consists in dividing each of its pixel values by the maximum value that a pixel can take (255 for an 8-bit image, 4095 for a 12-bit image, 65 535 for a 16-bit image).

```
x_train = np.array(x_train) / 255.0
x_test = np.array(x_test) / 255.0
```

Training

Introduction:

Deep convolutional neural network models may take days or even weeks to train on very large datasets.
A way to short-cut this process is to re-use the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the **ImageNet** image recognition tasks. Top performing models can be downloaded and used directly, or integrated into a new model for your own computer vision problems.
In this project, I'll be using the **ResNet50** model which will use the weights from the **ImageNet** dataset.

ResNet50 Model:

ResNet stands for Residual Network and is a specific type of convolutional neural network (CNN) introduced in the 2015 paper “Deep Residual Learning for Image Recognition” by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. CNNs are commonly used to power computer vision applications.

ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer). Residual neural networks are a type of artificial neural network (ANN) that forms networks by stacking residual blocks.

ResNet50 Architecture:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Special Characteristics of ResNet50:

The 50-layer ResNet architecture includes the following elements:

- **A 7×7 kernel convolution** alongside 64 other kernels with a 2-sized stride.
- **A max pooling layer** with a 2-sized stride.
- **9 more layers**—3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times.
- **12 more layers** with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.
- **18 more layers** with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times.
- **9 more layers** with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times.

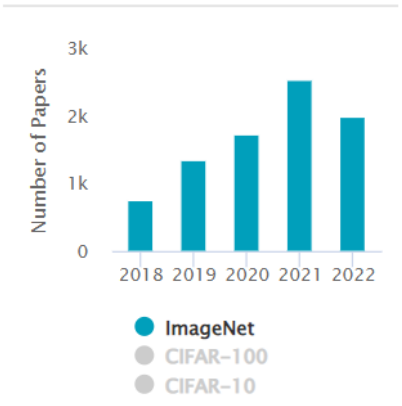
(up to this point the network has 50 layers)

- **Average pooling**, followed by a fully connected layer with 1000 nodes, using the softmax activation function.

About ImageNet:

The ImageNet dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset has been used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection.The ImageNet project does not own the copyright of the images, therefore only thumbnails and URLs of images are provided.

- Total number of non-empty WordNet synsets: 21841
- Total number of images: 14197122
- Number of images with bounding box annotations: 1,034,908
- Number of synsets with SIFT features: 1000
- Number of images with SIFT features: 1.2 million



Implementation:

First, we load the pre-trained weights and do not include the ImageNet Classifier at the top.This will move all the layer's weights from trainable to non-trainable. This is called "freezing" the layer: the state of a frozen layer won't be updated during training

```
net = ResNet50(
    weights='imagenet',
     include_top=False,
     input_shape=(image_size,image_size,3))
```

After that,we add these layers :
GlobalAveragePooling2D -> This layer acts similar to the Max Pooling layer in CNNs, the only difference being that it uses the Average values instead of the Max value while pooling. This really helps in decreasing the computational load on the machine while training.

Dropout -> This layer omits some of the neurons at each step from the layer making the neurons more independent from the neighboring neurons. It helps in avoiding overfitting. Neurons to be omitted are selected at random. The rate parameter is the likelihood of a neuron activation being set to 0, thus dropping out the neuron

Dense -> This is the output layer which classifies the image into 1 of the 4 possible classes. It uses the softmax function which is a generalization of the sigmoid function.

```
model = net.output
model = GlobalAveragePooling2D()(model)
model = Dropout(0.4)(model)
model = Dense(4, activation="softmax")(model)
model = Model(inputs= net.input, outputs= model)

#compile our model.
adam = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=adam, loss = 'categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Where we have used the Adam optimizer and a learning rate =0.0001

```
conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

global_average_pooling2d (Glob  (None, 2048)        0           ['conv5_block3_out[0][0]']
alAveragePooling2D)

dropout (Dropout)              (None, 2048)         0           ['global_average_pooling2d[0][0]'
                                                                ]

dense (Dense)                  (None, 4)            8196        ['dropout[0][0]']

==================================================================================================
Total params: 23,595,908
Trainable params: 23,542,788
Non-trainable params: 53,120
```

Finally we train our model using 15 epochs and a batch size of 64

```
file_writer_cm = tf.summary.create_file_writer(logdir)

tensorboard = TensorBoard(logdir, histogram_freq=1)


BATCH_SIZE = 64
EPOCHS = 15

Checkpoint = ModelCheckpoint(filepath = 'model-{epoch:02d}-{val_accuracy:.2f}-{val_loss:.2f}.h5',monitor = 'val_loss', verbose =

ES = EarlyStopping(monitor = 'val_loss',min_delta = 0.001,patience = 5,mode = 'min',restore_best_weights = True,verbose = 1)

RL = ReduceLROnPlateau(monitor = 'val_loss',factor = 0.3,patience = 5,verbose = 1,mode = 'min')

callbacks = [ES,RL,tensorboard,Checkpoint,LambdaCallback(on_epoch_end=log_confusion_matrix)]

history = model.fit(datagen.flow(x_train, y_train, batch_size=20),validation_data = (x_val,y_val),epochs = EPOCHS,callbacks = ca
```

callback -> set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

LambdaCallback -> will log the confusion matrix on every epoch.

ModelCheckpoint -> callback to save the Keras model or model weights at some frequency.

EarlyStopping -> stop training when a monitored metric has stopped improving.

ReduceLROnPlateau -> reduce learning rate when a metric has stopped improving.

Evaluation and testing

After building a machine learning or deep learning model and training it on the data set (brain tumor data set)... What's next?.We are going to evaluate the model that we've obtained, and improve the model based on what we learn evaluating it.

The train/test/validation split phase:

The most important thing we can do to properly evaluate the model is to not train the model on the entire dataset. A typical train/test split would be to use 80% of the data for training and 20% of the data for testing.
It's important to use new data when evaluating our model to prevent the likelihood of overfitting to the training set. However, sometimes it's useful to evaluate our model as we're building it to find the best parameters of a model, but we can't use the test set for this evaluation or else we'll end up selecting the parameters that perform best on the test data but maybe not the parameters that generalize best. To evaluate the model while still building and

tuning the model, we create a third subset of the data known as the validation set, , 20% of the data for validation

Metrics:

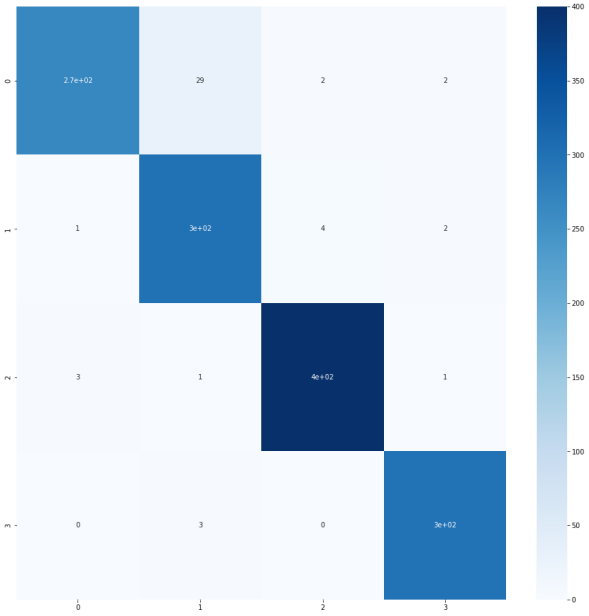
common metrics used to evaluate models

Classification metrics:

When performing classification predictions, there's four types of outcomes that could occur.

True positives: are when you predict an observation belongs to a class and it actually does belong to that class.
True negatives: are when you predict an observation does not belong to a class and it actually does not belong to that class.
False positives: occur when you predict an observation belongs to a class when in reality it does not.
False negatives: occur when you predict an observation does not belong to a class when in fact it does.

These sixteen outcomes are plotted on a confusion matrix for the brain tumor model. The following confusion matrix is an example for the case of multi-class classification predictions (four labels classification). We would generate this matrix after making predictions on the test data and then identifying each prediction as one of the sixteen possible outcomes described above.



The three main metrics used to evaluate a classification model are accuracy, precision, and recall.

Accuracy

It is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

Accuracy = Correct predictions / all prediction

Precision:

is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class.

$$\text{Precision} = TP / TP + FP$$

Recall:

is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class.

$$\text{Recall} = TP / TP + FN$$

	precision	recall	f1-score	support
0	0.99	0.89	0.94	300
1	0.90	0.98	0.94	306
2	0.99	0.99	0.99	405
3	0.98	0.99	0.99	300
accuracy			0.96	1311
macro avg	0.96	0.96	0.96	1311
weighted avg	0.97	0.96	0.96	1311

After testing the model using the test dataset we got the following accuracy and loss

```
loss,acc = model.evaluate(x_test,y_test)
```

```
41/41 [==============================] - 71s 2s/step - loss: 0.1083 - accuracy: 0.9634
```

Conclusion

Convolutional neural networks remain a growing area of research in automated brain tumor classification . In this work , we have implemented a theoretical introduction to the classification of brain tumor MRI image data set using deep learning algorithms. The experimental goal is to analyze the performance of CNN in processing brain tumor classification problems. Throughout the project, python 3.7 is used as the development language and the Keras library to build the CNN model. In order to improve the performance of the model, we studied the types and functions of activation functions in CNN, the meaning and effects of loss functions, the difference and selection of different pooling methods, the use of optimizers, and the parameter selection of convolutional layer, pooling layer and fully connected layer.

The Brain Tumor Kaggle dataset was used in the project. Before training, a data preprocessing method is used, and the steps include image standardization and image cropping of each MRI image. In training, different parameters are used on the CNN model to evaluate the impact of input parameters on function performance. After the training, we evaluate the performance of the model through some indicators, including accuracy, precision, recall and F-measure. Therefore, The accuracy of the CNN classifier on the dataset is 96%.

Finally, we have obtained satisfactory classification results by adjusting the parameters.This architecture may even be further improved by including more brain MRI images with different weights and with various contrast enhancement techniques to allow the architecture to be potentially more generalized and robust application for larger image databases.