**2  READING THE DATASET:**[**¶**](http://localhost:8888/notebooks/brain_tumour_detection_using_deep_learning.ipynb#READING-THE-DATASET:)

**2.0.1  Setting the file paths for a training dataset and a testing dataset. It then creates two empty lists, "train\_paths" and "train\_labels", and iterates through the folders within the training dataset directory to append the file paths and their corresponding labels to the appropriate lists. Then it shuffles the order of the file paths and labels.**

### 2.0.2. Creating a pie chart that represents the distribution of labels in a dataset called "train\_labels." The chart shows the percentage of each label, "pituitary," "notumor," "meningioma," and "glioma," in the dataset. The colors of the chart are defined as specific RGB values, and the chart is labeled accordingly. The chart also has an "explode" feature which separates each slice of the pie chart by a small margin.

# 3  AS WE CAN SEE THAT THE DATASET IS REASONABLY BALANCED.[¶](http://localhost:8888/notebooks/brain_tumour_detection_using_deep_learning.ipynb#AS-WE-CAN-SEE-THAT-THE-DATASET-IS-REASONABLY-BALANCED.)

### 3.0.1  Creating two empty lists, "test\_paths" and "test\_labels", and iterates through the folders within the testing dataset directory to append the file paths and their corresponding labels to the appropriate lists. It also shuffles the order of the file paths and labels.

### 3.0.1  Creating a pie chart that represents the distribution of train and test datasets. The chart shows the percentage of each label, "train" and "test" in the dataset. The colors of the chart are defined as specific RGB values, and the chart is labeled accordingly. The chart also has an "explode" feature which separates each slice of the pie chart by a small margin.

# DATA AUGMENTATION:[¶](http://localhost:8888/notebooks/brain_tumour_detection_using_deep_learning.ipynb#DATA-AUGMENTATION:)

### 4.0.1  Defining a function called "augment\_image" that takes in an image as an input. The function uses the Python Imaging Library (PIL) to convert the image to a PIL image object. Then it applies random brightness and contrast enhancements to the image using the ImageEnhance module, which adjusts the brightness and contrast of the image by a random value between 0.8 and 1.2. Finally, the function normalizes the image by dividing it by 255 and returns the augmented image. -----------------------------------------------------------

Defining a function called "open\_images" that takes in a list of paths to images and returns the images as arrays after augmenting them. The function first uses the keras function "load\_img" to load the images and resize them to the specified size (128x128 pixels). Then it applies the augment\_image function to each image. The function then returns the augmented images as an array.

The code then calls the open\_images function on a subset of the train\_paths (train\_paths[50:59]) and assigns the result to the variable "images". It also assigns the corresponding subset of train\_labels to the variable "labels".

Then it creates a figure of size (12,6) and plots 8 images with their corresponding label, with a title of their label. it also turns off the axis and show the images on the figure. The function also updates the font size of the title of the images and shows the figure.

### --------------------------------------------------

# 5  DATA GENERATOR HELPS AUGMENT IMAGES, NORMALIZES THEM.

The datagen function generates data for training and testing. It takes three arguments: paths, labels and batch\_size (default 12) and epochs (default 1).

It uses the open\_images function to open and augment the images and the encode\_label function to convert the labels to numerical values. The function then yields the batch of images and labels, allowing the model to train on them. The for \_ in range(epochs) loop allows you to specify the number of times the entire dataset should be passed through the model. And for x in range(0, len(paths), batch\_size) is used for iterating over the dataset in batches of the specified batch\_size.

# 6  MODEL:

## 6.1  WE ARE USING VGG16 FOR TRANSFER LEARNING.

The model is built on top of VGG16, which is a pre-trained convolutional neural network (CNN) for image classification.

* First, the VGG16 model is loaded with input\_shape=(IMAGE\_SIZE,IMAGE\_SIZE,3), include\_top=False, weights='imagenet'. The input shape is set to match the size of the images in the dataset, which is 128x128 pixels. The include\_top parameter is set to False, which means that the final fully-connected layers of VGG16 that perform the classification will not be included. The weights parameter is set to 'imagenet' which means that the model will be pre-trained with a dataset of 1.4 million images called imagenet
* Next, the for layer in base\_model.layers: loop is used to set all layers of the base\_model (VGG16) to non-trainable, so that the weights of these layers will not be updated during training.
* Then, the last three layers of the VGG16 model are set to trainable by using base\_model.layers[-2].trainable = True,base\_model.layers[-3].trainable = True and base\_model.layers[-4].trainable = True
* After that, a Sequential model is created and the VGG16 model is added to it with model.add(base\_model).
* Next, a Flatten layer is added to the model with model.add(Flatten()) which reshapes the output of the VGG16 model from a 3D tensor to a 1D tensor, so that it can be processed by the next layers of the model.
* Then, a Dropout layer is added with model.add(Dropout(0.3)) which is used to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.
* After that, a dense layer is added with 128 neurons and relu activation function is added with model.add(Dense(128, activation='relu')).
* Next, another Dropout layer is added with model.add(Dropout(0.2))
* Finally, the output dense layer is added with number of neurons equal to the number of unique labels and 'softmax' activation function is added with model.add(Dense(len(unique\_labels), activation='softmax')). The 'softmax' activation function is used to give a probability distribution over the possible classes.

### The model.summary() method in Keras prints a summary of the architecture of the model. This summary includes the layers in the model, their output shapes, the number of parameters in each layer, and the total number of parameters in the model. This can be useful for understanding the overall structure of the model and for identifying potential issues such as overfitting. In the code provided, model.summary() shows the architecture of the image classification model created using a pre-trained VGG16 model and some additional layers.

model.compile is used to configure the learning process before training the model. The optimizer is the algorithm used to update the weights of the model based on the gradients of the loss function. In this case, the Adam optimizer is used with a learning rate of 0.0001. The learning rate controls the step size at which the optimizer makes updates to the weights. A smaller learning rate will make the optimization converge slower but potentially with better results.

The loss function is used to measure how well the model is doing on the training data. The loss is a scalar value that represents the degree of error in the model's predictions. The sparse\_categorical\_crossentropy loss is used in this case, which is a measure of the dissimilarity between the predicted and actual labels.

The metrics parameter is used to specify the metrics that will be evaluated during training and testing. The sparse\_categorical\_accuracy metric is used in this case, which calculates the mean accuracy rate across all predictions for multiclass classification problems.

# 7  TRAINING THE MODEL:

The fit method takes in the following arguments:

* datagen(train\_paths, train\_labels, batch\_size=batch\_size, epochs=epochs): This argument specifies the training data generator to use. The datagen function generates batches of images and labels for training. It takes in the list of paths to the training images, the corresponding labels, the batch size and number of epochs. The datagen function will be called multiple times (once per epoch) and will yield a new batch of images and labels each time it is called.
* epochs=epochs: This argument specifies the number of times the model should go through the entire training dataset.
* steps\_per\_epoch=steps: This argument specifies the number of batches to use per epoch.

The batch\_size variable is set to 20, which means that the model will be trained on 20 images at a time. The steps variable is set to the total number of images divided by the batch size. The epochs variable is set to 4, so the model will go through the entire training dataset 4 times. The training is done by repeatedly calling the datagen function to get new batches of images and labels, and then training the model on those batches. The history variable will store the training history, which can be used to plot the training progress, or extract training statistics.

### Creating a plot of the training history of the model, including the accuracy and loss over the number of epochs. The x-axis represents the number of epochs and the y-axis shows the value of accuracy and loss. The plot is created using matplotlib library, it has two lines one for accuracy and one for loss. The plot helps in visualizing how the model is learning and how well it is performing during the training process. It is useful for identifying overfitting, underfitting, and to decide when to stop training.

# 8  EVALUATING MODEL WITH TEST SAMPLES:

Trained model to make predictions on the test set, which consists of the test\_paths and test\_labels. It uses the datagen() function to generate batches of images and labels, and for each batch it uses the model.predict() method to make predictions on the images. The predicted labels are in encoded form and using decode\_label() function they are decoded and stored in y\_pred. The actual labels are stored in y\_true. The tqdm library is used to display a progress bar for the loop.

### Generating a classification report which evaluates the performance of the model on the test dataset. The report contains various metrics such as precision, recall, f1-score and support for each class in the dataset. It also calculates a weighted average of these metrics across all classes. This report helps in understanding the overall performance of the model and identifying any specific classes where the model is performing well or poorly.

### 