Brain Tumour Detection with CNN

Import Libraries

In [1]:
```
!pip install imutils
```

```
Collecting imutils
  Downloading imutils-0.5.4.tar.gz (17 kB)
Building wheels for collected packages: imutils
  Building wheel for imutils (setup.py) ... done
  Created wheel for imutils: filename=imutils-0.5.4-py3-none-any.whl size=25858 sha256=75057b10404a3cb85b976eb40c96f06e
01ed0d37cec22d74d783cef4b327c8e5
  Stored in directory: /root/.cache/pip/wheels/86/d7/0a/4923351ed1cec5d5e24c1eaf8905567b02a0343b24aa873df2
Successfully built imutils
Installing collected packages: imutils
Successfully installed imutils-0.5.4
WARNING: You are using pip version 20.1.1; however, version 23.3.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' command.
```

In [2]:
```
import numpy as np
import pandas as pd
import os
from os import listdir
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
import imutils

from tensorflow.keras.models import Model,load_model
from tensorflow.keras.layers import Conv2D,Input,ZeroPadding2D,BatchNormalization,Flatten,Activation,Dense,MaxPooling2D
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle #shuffling the data improves the model
```

```
Using TensorFlow backend.
```

Load Images

In [3]:
```
image_dir="../input/brain-mri-images-for-brain-tumor-detection/"
```

Making directory for Augmented images

A directory is formed using os.makedirs() function for augmented images(yes/ no).

In [4]:
```
os.makedirs('../working/augmented-images')
os.makedirs('../working/augmented-images/yes')
os.makedirs('../working/augmented-images/no')
```

Augmentation of images

About the data:

The dataset contains two folders: "yes" and "no," each containing 253 brain MRI images. The "yes" folder comprises 155 brain MRI images exhibiting tumors, while the "no" folder contains 98 brain MRI images without tumors.

In [5]:
```
def augment_data(file_dir, n_generated_samples, save_to_dir):
    data_gen = ImageDataGenerator(rotation_range=10,
                                  width_shift_range=0.1,
                                  height_shift_range=0.1,
                                  shear_range=0.1,
                                  brightness_range=(0.3, 1.0),
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest'
                                 )

    for filename in listdir(file_dir):
        image = cv2.imread(file_dir + '/' + filename)
        # reshape the image
        image = image.reshape((1,)+image.shape)
        save_prefix = 'aug_' + filename[:-4]
        i=0
        for batch in data_gen.flow(x=image, batch_size=1, save_to_dir=save_to_dir,save_prefix=save_prefix, save_format=
                i += 1
                if i > n_generated_samples:
                    break
```

In [6]:
```
augmented_data_path ='../working/augmented-images/'
# augment data for the examples with label equal to 'yes' representing tumurous examples
augment_data(file_dir=image_dir+'yes',n_generated_samples=6, save_to_dir=augmented_data_path+'yes')
# augment data for the examples with label equal to 'no' representing non-tumurous examples
augment_data(file_dir=image_dir+'no', n_generated_samples=9, save_to_dir=augmented_data_path+'no')
```

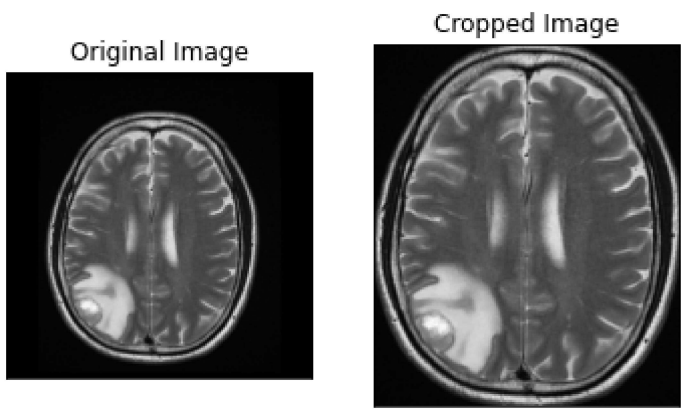
Preprocessing the data

In order to crop the specific part of the image containing tumour,cropping technique via OpenCv is used.

In [7]:
```
def crop_brain_contour(image, plot=False):

    # Convert the image to grayscale, and blur it slightly
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 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)

    # Find contours in thresholded image, then grab the largest one
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    c = max(cnts, key=cv2.contourArea)
    # extreme points
    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])

    # crop new image out of the original image using the four extreme points (left, right, top, bottom)
    new_image = image[extTop[1]:extBot[1], extLeft[0]:extRight[0]]

    if plot:
        plt.figure()
        plt.subplot(1, 2, 1)
        plt.imshow(image)
        plt.tick_params(axis='both', which='both', top=False, bottom=False, left=False, right=False,labelbottom=False,
        plt.title('Original Image')
        plt.subplot(1, 2, 2)
        plt.imshow(new_image)
        plt.tick_params(axis='both', which='both',top=False, bottom=False, left=False, right=False,labelbottom=False, l
        plt.title('Cropped Image')
        plt.show()

    return new_image
```

After applying the cropping function

In [8]:
```
ex_img = cv2.imread(image_dir+'yes/Y107.jpg')
ex_crop_img = crop_brain_contour(ex_img, True)
```



In [9]:
```
def load_data(dir_list, image_size):

    # load all images in a directory
    X = []
    y = []
    image_width, image_height = image_size

    for directory in dir_list:
        for filename in listdir(directory):
            image = cv2.imread(directory+'/'+filename)
            image = crop_brain_contour(image, plot=False)
            image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.INTER_CUBIC)
            # normalize values
            image = image / 255.
            # convert image to numpy array and append it to X
            X.append(image)
            # append a value of 1 to the target array if the image
            # is in the folder named 'yes', otherwise append 0.
            if directory[-3:] == 'yes':
                y.append([1])
            else:
                y.append([0])

    X = np.array(X)
    y = np.array(y)

    # Shuffle the data
    X, y = shuffle(X, y)

    print(f'Number of examples is: {len(X)}')
    print(f'X shape is: {X.shape}')
    print(f'y shape is: {y.shape}')
```

```
    return X, y
```

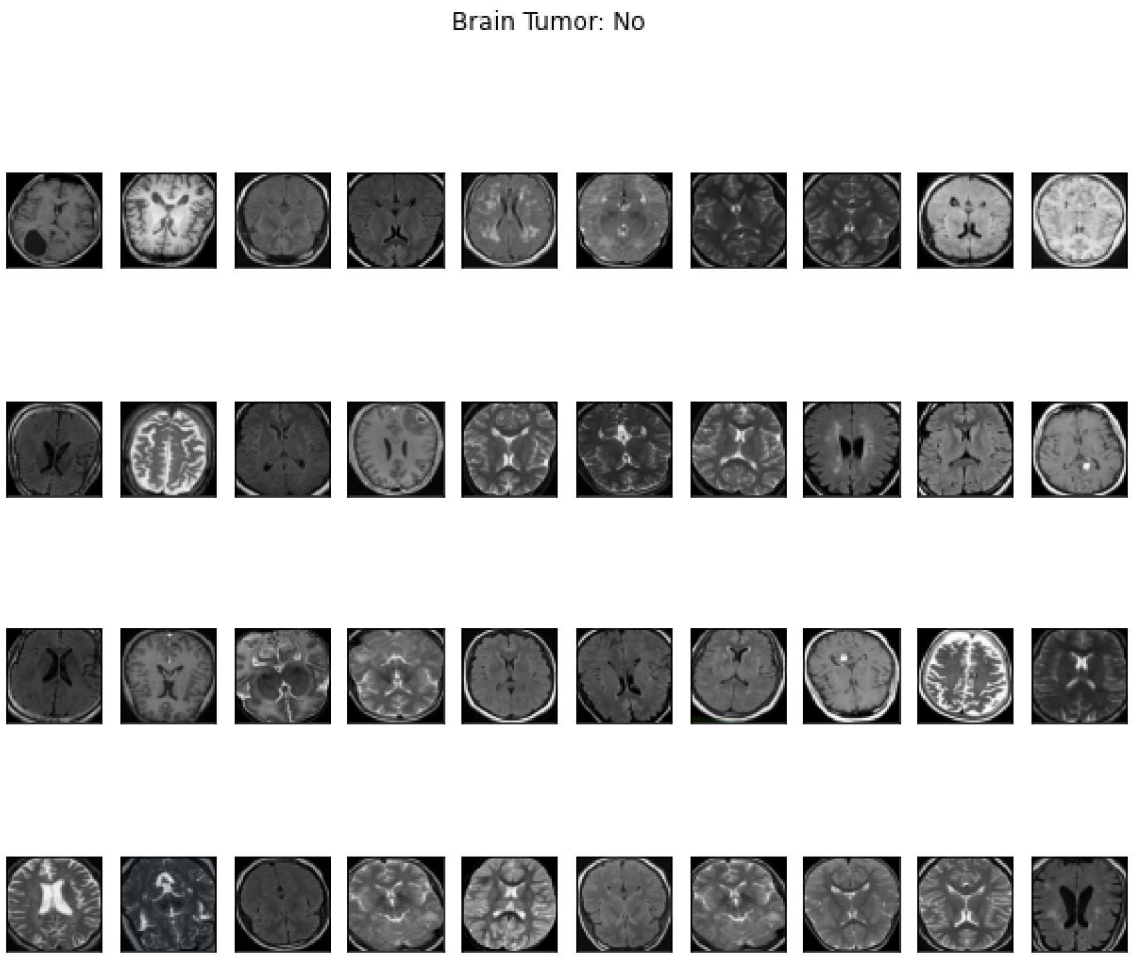
In [10]:
```
augmented_yes =augmented_data_path+'yes'
augmented_no = augmented_data_path+'no'

IMG_WIDTH, IMG_HEIGHT = (240, 240)

X, y = load_data([augmented_yes, augmented_no], (IMG_WIDTH, IMG_HEIGHT))
```

```
Number of examples is: 2064
X shape is: (2064, 240, 240, 3)
y shape is: (2064, 1)
```

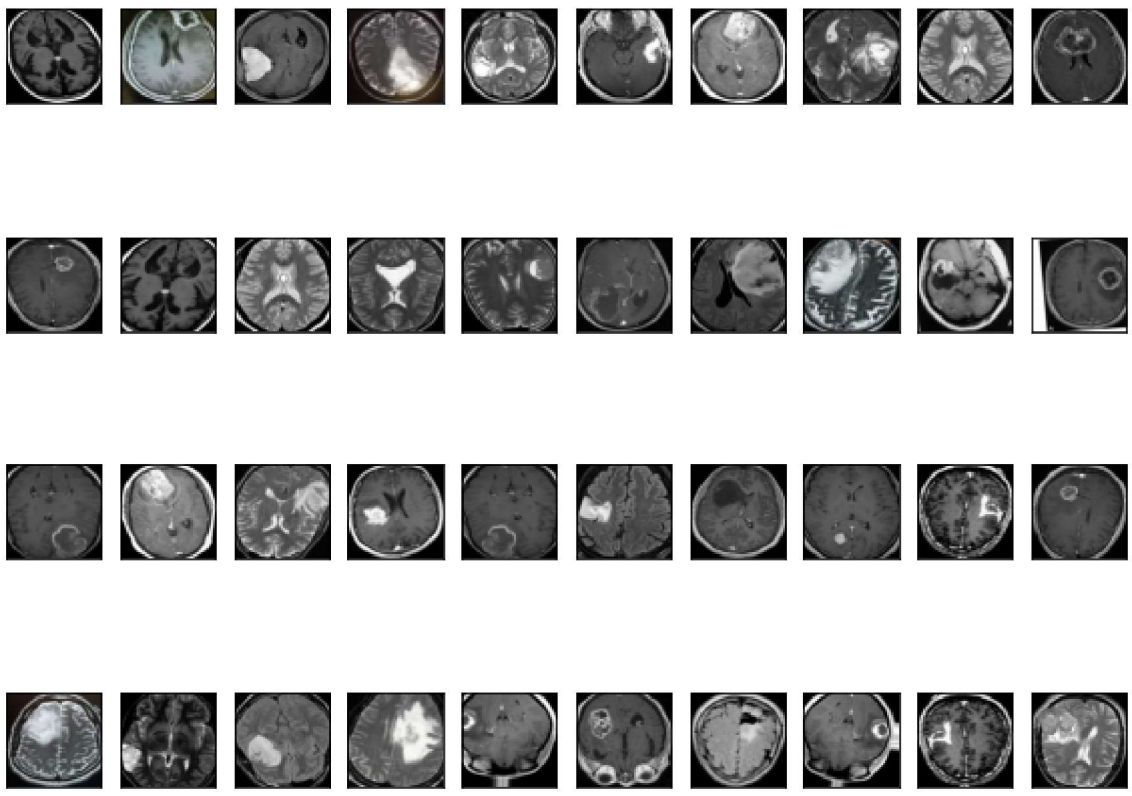
Visualization of data

In [11]:
```
def plot_sample_images(X, y, n=40):
    for label in [0,1]:
        # grab the first n images with the corresponding y values equal to label
        images = X[np.argwhere(y == label)]
        n_images = images[:n]

        columns_n = 10
        rows_n = int(n/ columns_n)

        plt.figure(figsize=(10, 8))

        i = 1 # current plot
        for image in n_images:
            plt.subplot(rows_n, columns_n, i)
            plt.imshow(image[0])

            # remove ticks
            plt.tick_params(axis='both', which='both',
                            top=False, bottom=False, left=False, right=False,
                            labelbottom=False, labeltop=False, labelleft=False, labelright=False)

            i += 1

        label_to_str = lambda label: "Yes" if label == 1 else "No"
        plt.suptitle(f"Brain Tumor: {label_to_str(label)}")
        plt.show()
```

In [12]:
```
plot_sample_images(X, y)
```



Brain Tumor: Yes



In [13]:
```
def split_data(X, y, test_size=0.2):

    X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, test_size=test_size)
    X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, test_size=0.5)

    return X_train, y_train, X_val, y_val, X_test, y_test
```

In [14]:
```
X_train, y_train, X_val, y_val, X_test, y_test = split_data(X, y, test_size=0.3)
```

In [15]:
```
print ("number of training examples = " + str(X_train.shape[0]))
print ("number of validation examples = " + str(X_val.shape[0]))
print ("number of test examples = " + str(X_test.shape[0]))
```

```
number of training examples = 1444
number of validation examples = 310
number of test examples = 310
```

In [16]:
```
def build_model(input_shape):
    X_input = Input(input_shape)
    X = ZeroPadding2D((2, 2))(X_input)

    X = Conv2D(32, (7, 7), strides = (1, 1))(X)
    X = BatchNormalization(axis = 3, name = 'bn0')(X)
    X = Activation('relu')(X)

    X = MaxPooling2D((4, 4))(X)
    X = MaxPooling2D((4, 4))(X)
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid')(X)
    model = Model(inputs = X_input, outputs = X)

    return model
```

In [17]:
```
IMG_SHAPE = (IMG_WIDTH, IMG_HEIGHT, 3)
model=build_model(IMG_SHAPE)
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 240, 240, 3)]     0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 244, 244, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 238, 238, 32)      4736
_________________________________________________________________
bn0 (BatchNormalization)     (None, 238, 238, 32)      128
_________________________________________________________________
activation (Activation)      (None, 238, 238, 32)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 59, 59, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 6273
=================================================================
Total params: 11,137
Trainable params: 11,073
Non-trainable params: 64
_________________________________________________________________
```

In [18]:
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x=X_train, y=y_train, batch_size=32, epochs=22, validation_data=(X_val, y_val))
```

```
Epoch 1/22
46/46 [==============================] - 2s 43ms/step - loss: 0.7538 - accuracy: 0.6039 - val_loss: 0.6335 - val_accura
cy: 0.7323
Epoch 2/22
46/46 [==============================] - 2s 37ms/step - loss: 0.4599 - accuracy: 0.7853 - val_loss: 0.6037 - val_accura
cy: 0.7290
Epoch 3/22
46/46 [==============================] - 2s 37ms/step - loss: 0.4100 - accuracy: 0.8193 - val_loss: 0.6394 - val_accura
cy: 0.5710
Epoch 4/22
46/46 [==============================] - 2s 37ms/step - loss: 0.3477 - accuracy: 0.8560 - val_loss: 0.6107 - val_accura
cy: 0.6452
Epoch 5/22
46/46 [==============================] - 2s 38ms/step - loss: 0.3197 - accuracy: 0.8608 - val_loss: 0.4693 - val_accura
cy: 0.8129
Epoch 6/22
46/46 [==============================] - 2s 38ms/step - loss: 0.3087 - accuracy: 0.8650 - val_loss: 0.4729 - val_accura
cy: 0.7903
Epoch 7/22
46/46 [==============================] - 2s 41ms/step - loss: 0.2645 - accuracy: 0.8954 - val_loss: 1.3481 - val_accura
cy: 0.4871
Epoch 8/22
46/46 [==============================] - 2s 38ms/step - loss: 0.3081 - accuracy: 0.8580 - val_loss: 0.4056 - val_accura
cy: 0.8194
Epoch 9/22
46/46 [==============================] - 2s 38ms/step - loss: 0.2478 - accuracy: 0.9079 - val_loss: 0.7633 - val_accura
cy: 0.7000
Epoch 10/22
46/46 [==============================] - 2s 37ms/step - loss: 0.2247 - accuracy: 0.9086 - val_loss: 0.4895 - val_accura
cy: 0.7968
Epoch 11/22
46/46 [==============================] - 2s 37ms/step - loss: 0.2170 - accuracy: 0.9190 - val_loss: 0.6964 - val_accura
cy: 0.7484
Epoch 12/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1744 - accuracy: 0.9363 - val_loss: 0.4427 - val_accura
cy: 0.8290
Epoch 13/22
46/46 [==============================] - 2s 38ms/step - loss: 0.2009 - accuracy: 0.9204 - val_loss: 0.7253 - val_accura
cy: 0.7097
Epoch 14/22
46/46 [==============================] - 2s 40ms/step - loss: 0.2256 - accuracy: 0.9017 - val_loss: 1.7402 - val_accura
cy: 0.5581
Epoch 15/22
46/46 [==============================] - 2s 37ms/step - loss: 0.2070 - accuracy: 0.9197 - val_loss: 0.3956 - val_accura
cy: 0.8581
Epoch 16/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1342 - accuracy: 0.9564 - val_loss: 0.3909 - val_accura
cy: 0.8548
Epoch 17/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1424 - accuracy: 0.9494 - val_loss: 0.8974 - val_accura
cy: 0.7194
Epoch 18/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1544 - accuracy: 0.9432 - val_loss: 0.4088 - val_accura
cy: 0.8484
Epoch 19/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1347 - accuracy: 0.9529 - val_loss: 0.6333 - val_accura
cy: 0.7935
Epoch 20/22
46/46 [==============================] - 2s 38ms/step - loss: 0.1122 - accuracy: 0.9661 - val_loss: 0.5252 - val_accura
cy: 0.8323
Epoch 21/22
46/46 [==============================] - 2s 37ms/step - loss: 0.1178 - accuracy: 0.9605 - val_loss: 0.4408 - val_accura
cy: 0.8387
Epoch 22/22
46/46 [==============================] - 2s 41ms/step - loss: 0.1449 - accuracy: 0.9446 - val_loss: 0.4693 - val_accura
cy: 0.8677
```

Out[18]:
```
<tensorflow.python.keras.callbacks.History at 0x7891a02c0310>
```

In [19]:
```
history = model.history.history
```

Plotting of Accuracy

In [20]:
```
def plot_metrics(history):

    train_loss = history['loss']
    val_loss = history['val_loss']
    train_acc = history['accuracy']
    val_acc = history['val_accuracy']

    # Loss
    plt.figure()
    plt.plot(train_loss, label='Training Loss')
    plt.plot(val_loss, label='Validation Loss')
    plt.title('Loss')
    plt.legend()
    plt.show()

    # Accuracy
    plt.figure()
    plt.plot(train_acc, label='Training Accuracy')
```

```
    plt.plot(val_acc, label='Validation Accuracy')
    plt.title('Accuracy')
    plt.legend()
    plt.show()
```

In [21]:
```
plot_metrics(history)
```