Lebanese University
Faculty of Engineering III
Department of Electrical and Electronic Engineering

Brain Tumor Classification Using Convolutional Neural Network

A Deep learning Project

Prepared by: Hadi Al Zein (5547)

Presented to: Dr. Mohamad Aoude

Submitted for the Mini-Project course

Table of Content

1. Introduction

I. Deep learning

Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

With any typical deep learning project, we first need to identify the actual problem in order to get the right solution and it should be understood, the feasibility of the Deep Learning should also be checked (whether it should fit Deep Learning or not). Second, we need to identify the relevant data which should correspond to the actual problem and should be prepared accordingly. Third, Choose the Deep Learning Algorithm appropriately. Fourth, Algorithm should be used while training the dataset. Fifth, Final testing should be done on the dataset **(Fig.1.I.1)**.

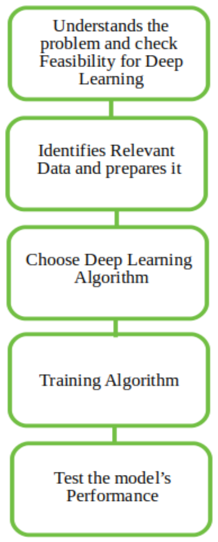

Understands the problem and check Feasibility for Deep Learning
Identifies Relevant Data and prepares it
Choose Deep Learning Algorithm
Training Algorithm
Test the model's Performance

Fig.1.I.1 *Deep Learning Project Lifecycle*

II. Problem Statement

The idea of this project is to build a neural network to apply binary classification on brain tumor MRI images. The application of such a tool is a great advancement in the biomedical engineering world and especially in the AI for medicine domain which is one of today's main concerns for technology advocates and researchers. The brain tissue is a very sensitive area of the human body, and technological approaches to such topics should be very cautious and accurate to avoid any further complications. Of course, comparing a doctor's ability to distinguish cancerous cells from non-cancerous cells is much better than a computer's ability given a single data point and with no prior knowledge. However, with the advancement in computing power and data storage, millions of data points can now be stored into the same place that come handy in computer intelligent learning process.

2. Implementation

I. Data Augmentation

The original dataset retrieved for the completion of this project is from Kaggle. The original dataset contains 2 folders: “Yes” and “No” which contain 253 Brain MRI Images. The folder “Yes” contains 155 Brain MRI Images that are tumorous and the folder “No” contains 98 Brain MRI Images that are non-tumorous.

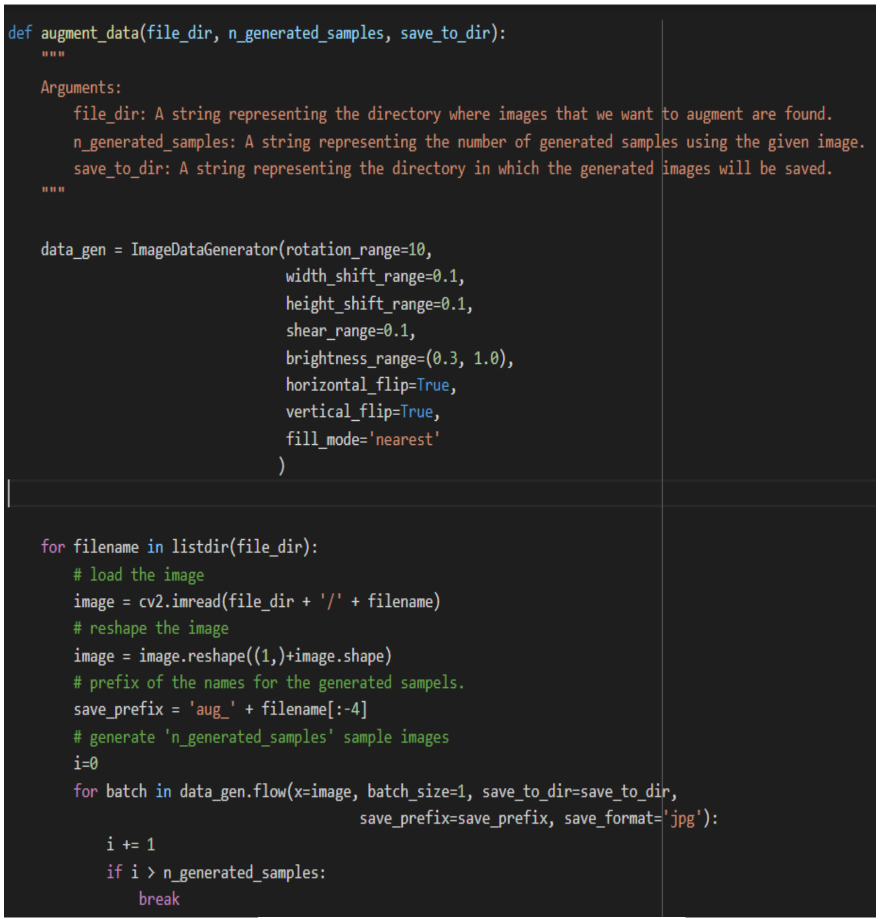
As with any other deep learning project, shortage of data continues to be the major factor to put into account and to investigate for augmentation methods to increase numbers of data points.

For this purpose, a **data augmentation** approach was followed to increase the number of positive and negative instances. But something we had to put in mind is the **imbalanced dataset** that we’re working with. Remember that 61% of the data (155 images) are tumorous. And, 39% of the data (98 images) are non-tumorous. So in order to balance the data, we can generate 9 new images for every image that belongs to 'no' class and 6 images for every image that belongs to the 'yes' class.

As seen in **(Fig.2.I.1)**, an augment_data function was implemented for this purpose with multiple settings including a modified width shift range, a height shift range, a shear range, a brightness range, a vertical flip, a horizontal flip, and a nearest fill mode.

The function returns n_generated_samples (passed as a parameter) for every image from the original dataset.

Following this manner, we will end up having a total of 2063 examples, 52.49% of which are positive instances and 47.5% are negatives. This



```python
def augment_data(file_dir, n_generated_samples, save_to_dir):
    """
    Arguments:
        file_dir: A string representing the directory where images that we want to augment are found.
        n_generated_samples: A string representing the number of generated samples using the given image.
        save_to_dir: A string representing the directory in which the generated images will be saved.
    """

    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):
        # load the image
        image = cv2.imread(file_dir + '/' + filename)
        # reshape the image
        image = image.reshape((1,)+image.shape)
        # prefix of the names for the generated sampels.
        save_prefix = 'aug_' + filename[:-4]
        # generate 'n_generated_samples' sample images
        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='jpg'):
            i += 1
            if i > n_generated_samples:
                break
```

***Fig.2.I.1** augment_data function*

feature of our project is capable of both increasing the number of examples and facing the imbalances in the dataset.

II. Data Preparation and Preprocessing

In order to crop the part that contains only the brain of the image, a cropping technique to find the extreme top, bottom, left and right points of the brain. This technique is done using the cropping tools in OpenCV library. For better understanding of the use of this technique, we can observe **Fig.2.II.1**. This cropped version is what we'll be feeding to the neural network since it limits the number of features to be extracted from the input to only what we really care about which is the inner part of the brain tissue and not the surrounding.

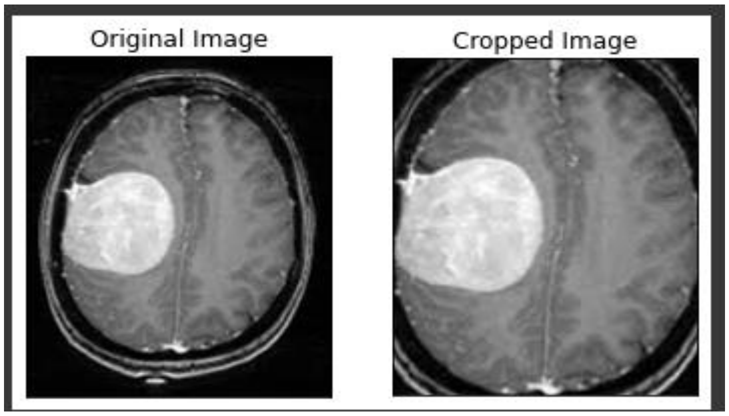


Fig.2.II.1 *Comparison between the original image and our cropped version*

This cropping technique will represent the first preprocessing tool we'll use and will be part of a set of functions to build and get the dataset ready for the network.

We introduce a load_data function that holds different preprocessing functions within it that finally returns X and y which represent the image and its label respectively. We'll feed to the function, the augmented data we've created earlier. The load_data function is formed of a several instructions including reading the image, cropping the part of the brain, resizing the image to be (240,240,3), applying normalization, then appending the image to X and the label to y.

The finalized dataset will then be split into 70% of the data for training, 15% of the data for validation, 15% of the data for testing. This is done using the train_test_split function in the sci-kit learn library. Now that the data is ready, it's time to assign a specific architecture for the neural network we're going to utilize.

III. CNN Architecture

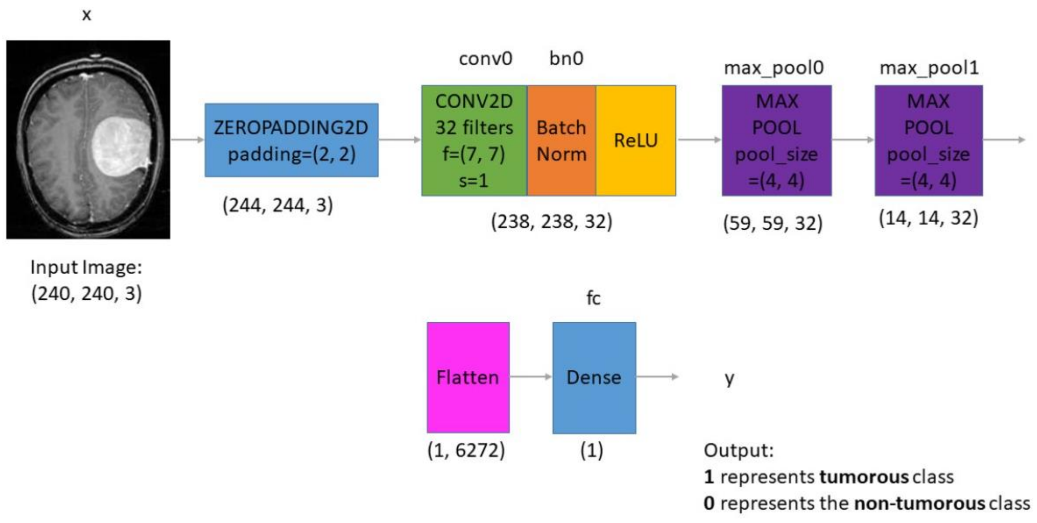
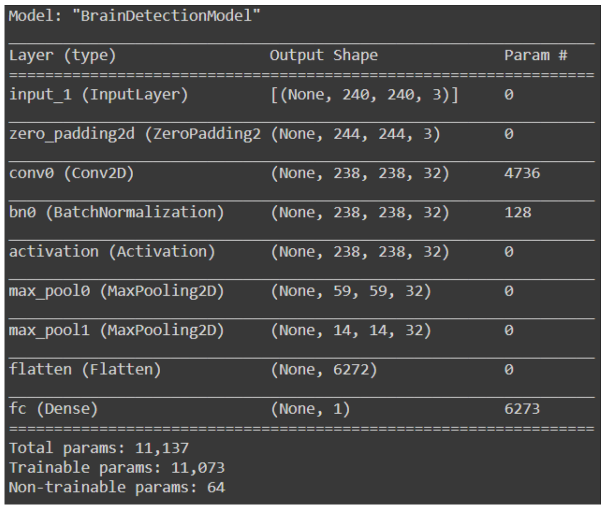


Fig.2.III.1 *CNN Architecture*

As part of the preprocessing process, the image was resized to (240,240,3), a good standard size in the world of deep learning. For this purpose, we'll first start by defining the **input placeholder** as a tensor with this input shape. Then **2D zero padding** is applied on the image to pad its borders with zeros putting us at a stage with (244,244,3) size. The next block constitutes of a **2D convolution layer** having 32 filters and strides of shape (7,7) accompanied **with batch normalization** which stabilizes the learning process and reduces the number of epochs required to train a neural network. The block is ended with the **ReLU activation function** to help the network learn complex patterns in the data. And after that, **2 max pooling layers** are added with a pool size of (4,4) each. And Last but not least, the input is **flattened** into a 1-dimensional array of size 6272. Finally, this long feature vector is connected to the final classification model, which is called a **fully-connected layer**. At this last level, it's where the actual classification is done and the image is said to represent the tumorous class or the non-tumorous class.



```
Model: "BrainDetectionModel"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 240, 240, 3)]     0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 244, 244, 3)       0
_________________________________________________________________
conv0 (Conv2D)               (None, 238, 238, 32)      4736
_________________________________________________________________
bn0 (BatchNormalization)     (None, 238, 238, 32)      128
_________________________________________________________________
activation (Activation)      (None, 238, 238, 32)      0
_________________________________________________________________
max_pool0 (MaxPooling2D)     (None, 59, 59, 32)        0
_________________________________________________________________
max_pool1 (MaxPooling2D)     (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
fc (Dense)                   (None, 1)                 6273
=================================================================
Total params: 11,137
Trainable params: 11,073
Non-trainable params: 64
```

Fig.2.III.2 *Model Summary*

To the right **(Fig.2.III.2)**, a model summary showing the output shape at each stage with regard to the stacked layers as well as the total number of parameters.

The model is now ready to be compiled. In other words, it's now time to define the optimizer to be used which is "adam" in our case, the loss which is cross entropy, and the metrics which is going to be the accuracy.

Two important additions to our project are both Tensorboard for instant visualization of results and adding checkpoints to track the number of epochs giving the highest accuracy. The model with the saved checkpoint will be considered the best model that can be created out of the available data and resources.

Next, the model will be trained on the training data and validated with the validation data already split and categorized. The next section will discuss in details the results of this model fit.

3. Results and Discussion

The figures below represent both the training loss Vs. validation loss **(fig. 3.1)** and training accuracy Vs. validation accuracy **(fig. 3.2).** These were plotted using Matplotlib, however tensorboard can also be used to track the variation in loss and accuracy with iterations. The end results of both curves are great, and with certain optimizations, the curves can be smoothened. It's important here to approach with hyperparameter tuning. For example, we can play around with the numbers of neurons and the learning rate to alter results.

At epoch 23, we've reached a training loss of 0.0811, a training accuracy of 0.9861, a validation loss of 0.3530, and validation accuracy of 0.8226.

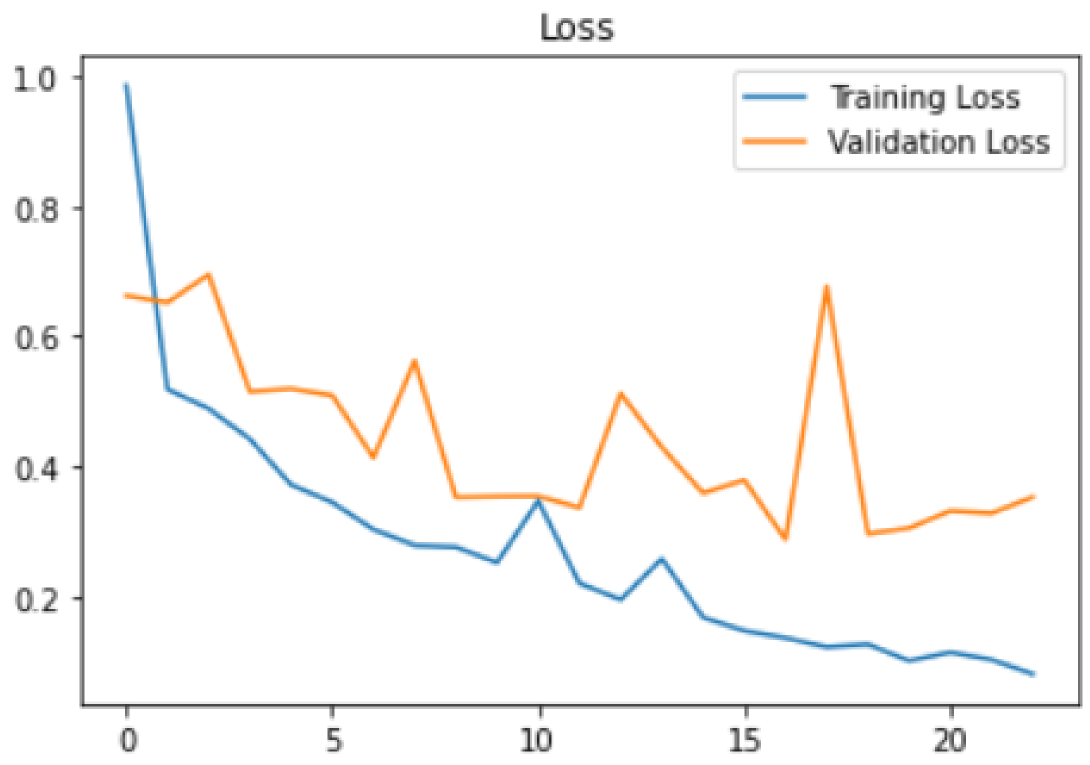


Fig.3.1 *The Variation of Loss metric for both training and validation*

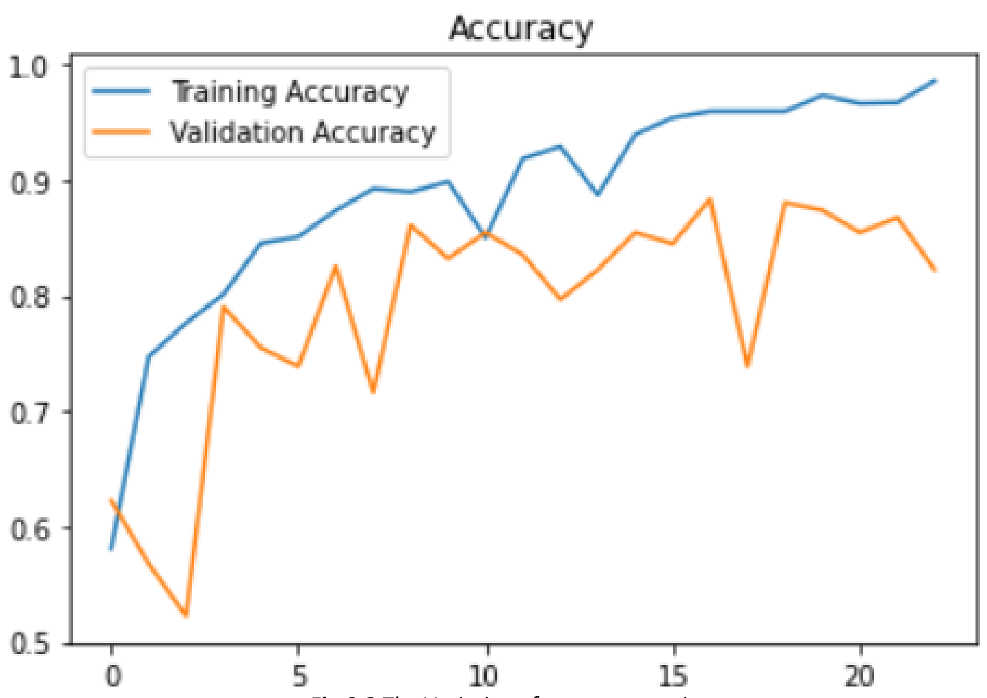


Fig.3.2 *The Variation of accuracy metric for both training and validation*

Furthermore, we can experiment with the best model (the one with the best validation accuracy). This best model represents the checkpoint giving the highest validation accuracy. We can compute the F1 score **(Fig.3.3)** for this best model on the testing data.

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Fig.3.3 *F1 score*

And the resulting F1 score for the testing data was 0.849162.

From here, we can also plot the confusion matrix with regard to the testing data **(Fig. 3.4)**.

From the confusion matrix, it's safe to say the true positives are of great majority. However the **alerting notification** comes from the both the number of false positives which is 9 and false negatives which is 45 out of 309 samples. The first causes false surgeries and the latter means ignoring the case.

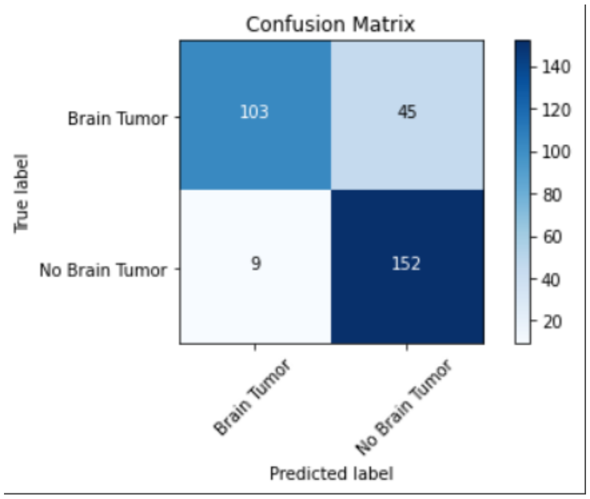


Fig.3.3 *F1 score*

4. Conclusion

The results of this model were greatly satisfying. However, it’s important to continue raising awareness on the need for “almost perfect” detectors or classifiers when it comes to health and medicine and especially when the targeted human body part which is the brain considering it one of the most critical organs. From here, we can say the very accurate models are to be implemented in the medicinal industry and that comes from the fact that models continue to evolve and get better with the increased numbers of modifications and parameter tuning as well as the size of the dataset.

5. References

1. https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

2. https://keras.io/api/layers/convolution_layers/convolution2d/

3. https://keras.io/api/layers/

4. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html

5. https://stats.stackexchange.com/questions/49226/how-to-interpret-f-measure-values

6. https://machinelearningmastery.com/how-to-accelerate-learning-of-deep-neural-networks-with-batch-normalization/