

Brain Tumor Classification

Beatriz Vizoso
m20210666

Filipa Alves
m20210662

Helena Oliveira
r20181121

Maria Almeida
m20210611

1. Introduction

This project focuses on the classification of types of brain tumors given an MRI image of the brain, by designing and implementing a Deep Learning model. The dataset used contains 3064 contrast enhanced images from 233 patients with 3 kinds of brain tumors: meningioma, glioma and pituitary.

The goal was to develop a Deep Learning model that could input MRI images of patients with one of the previously mentioned tumors and be able to identify which of those tumors each patient was suffering from based solely on that image, making it a multiclass classification problem. This problem can have very important real-life implementations, as correct detection of brain tumors will influence the treatments given to the patient and, consequently, influence their recovery.

2. Comparison with Papers

The problem of Brain Tumor recognition has been deeply studied in the last few years. The approach used in this project was based on various other papers and works already developed, which are going to be discussed in this section.

[Kang, Ullah, & Gwak \(2021\)](#) developed a methodology using an ensemble of deep features and machine learning classifiers. They used transfer learning techniques and pre-trained deep convolutional neural networks to do a feature extraction of deep features from brain tumor images. The best deep features are then fed into machine learning classifiers to predict the final output. In this project we chose to use this approach: feature extraction and prediction using Machine Learning models was used.

The work developed by [Cheng et al \(2015\)](#) was focused on the Region of Interest (ROI) of brain tumor images. Considering that the tumor surrounding tissues can offer clues to better classify tumors, augmented tumor region was used as the ROI instead of the original region (that only included the brain tumor) used in other works.

3. Data Gathering

The dataset is originally available at [Brain Tumor Dataset](#) organized in matlab data format (.mat file). After multiple attempts to convert the matlab format into png files, and extract all relevant information such as the patient ID (cjdata.PID), a plausible implementation in Python was not found, subsequently the data in .png format was extracted from [Kaggle](#) (3 folders containing the scans of each type of tumor). The data is slightly imbalanced as showed in the table below.

Tumors	Meningioma	Glioma	Pituitary
Labels	0	1	2
Number of Images	708 (23%)	1426 (46%)	930 (30%)

Table 1: Dataset labels description

4. Data Pre-Processing

Before any type of manipulation, a random seed was settled to guarantee reproducibility of the results. The following steps of data manipulation were performed in Python.

The renaming of the files was performed to ensure every step of the pre-processing was done as intended. Additionally, 3 directories were created for train, validation and test data, and, inside each of them, 3 folders, one for each of the labels.

To split the data into train-validation-test, *random.sample()* - making sure all three sets were disjoint - was used with the rationale 80%-10%-10%. Due to the relative lack of images, the percentage of the data used for training is above the standard guidelines to ensure the model has enough images to learn the patterns. Cross Validation was avoided due to the cost associated with training k different models. Also, the proportion of labels was kept identical in the train, validation and test set, taking a step further into data representativeness.

Although more complex methods were proven to be successful (Zhang (2015)), by using the class *ImageDataGenerator* it was possible to rescale the pixel intensity values, between 0 and 255 - to a range [0,1] in all image files.

Normalization is especially important when dealing with magnetic resonance images (MRI), since the parameters and scanners might be very different for each data acquisition, which may result in large intensity variations that may harm the CNN performance.

All the images are squared, either by 256x256 pixels or 512x512 pixels, therefore the size used in the models was decreased to 150x150 pixels, since CNN benefit from small images that enable a quicker learning. As proposed in the articles in Section 2, the RGB format seems not to be relevant in this scope, therefore '*color_mode*' parameter was settled to grayscale. Thus, no unnecessary dimensions were dealt, taking a step forward to a simpler model.

The parameter 'shuffle' was set to True on the training set, which means shuffling the order in which examples are fed to the classifier, so that batches between epochs do not look alike.

The batch size hyperparameter was settled to 20, which means the model's parameters are updated after seeing 20 images. Therefore, Minibatches were applied, since the batch size is more than 1 and less than the size of the training set. Thus, it requires less memory, and the training process is faster.

In order to overcome overfitting, models with data augmentation in the training set were performed. Due to the lack of knowledge on how to deal with medical images, augmentation was implemented conservatively in order not to change the image completely, avoiding data bias. Only the '*rotation_range*' and '*shear_range*' parameters were defined to distort the image.

5. Performance Measure

The F1 macro metric was chosen to be evaluated by the model during training and testing, since it gives every class the same importance, and the aim is to have a model that performs well on all classes. On the contrary, F1 micro is greatly influenced by majority class. Accuracy was not an option due to the imbalanced nature of the data (Memari, 2021).

In order to tackle the imbalanced nature of the problem in hands, various techniques to mitigate this issue were searched, such as the use of focal loss to monitor, '*sample_weight*' and '*class_weight*' parameters. However, these approaches seem to be useful only on extremely imbalanced datasets, which is not the case.

6. Models

Since the problem is image classification, Convolutional Neural Networks (CNN) is the type of model for the task. The approach taken consisted of starting with a basic random model to assess how it performed on the data and then improving it along the way until a final model with the best

performance was reached. To save all the models depending on their performance, two Callbacks were implemented: *Early Stopping*, which monitored the validation loss and stopped the fitting process if it did not improve after 4 epochs, and *Model Checkpoint*, which saved the model in case there were improvements on the same metric. The optimizers Adam and Rmsprop were briefly compared and it was decided to use Adam since it is faster, although Rmsprop can give slightly better results.

It is important to note that an effort not to over tune the hyperparameters was done, to minimize information leaks – information about the validation data is leaked into the training phase.

In this Section, there is a description of the models and in Section 8, [Table 2](#) contains information regarding the models' times, total epochs, smallest validation loss and F1 scores.

6.1. Basic Random Model (results in [Table 2](#))

The first model has a simple architecture. In the convolutional base, the input shape is 150 by 150, with 1 color channel. There are four convolutional layers, the first two with 64 filters (of size 3x3, as suggested as a rule of thumb) and the remaining with 32. Each is followed by a pooling layer that takes the largest element from the rectified map within a 2x2 window (MaxPooling2D). All four layers use the Relu activation function, in order to introduce non-linearity to the model. To complete the model, the 3D output was flattened into 1D and two Dense layers were added. The first, with 50 units, uses the Relu activation function, and the second, with 3 units (since there are 3 labels), uses the softmax, which calculates the probability of an instance being from a certain class, and makes the sum of all the probabilities equal to 1. Since this is a multiclass problem, categorical crossentropy was used as the loss function, measuring the distance between the distribution outputted by the network and the true distribution of the labels.

6.2. Basic Model with Increasing Number of Filters (through CNN) (results in [Table 2](#))

This model is built exactly as the first, except now the first two convolutional layers have 32 filters (instead of 64) while the last two have 64 filters (instead of 32), and the first dense layer has 100 units. These alterations were performed having in mind that the number of filters should progressively increase in the network so that it is possible to progressively capture more complex patterns. Thus, it was possible to certify this practice since the model performed better than the previous one.

6.3. Overcoming Overfitting (results in [Table 2](#))

Looking at the results, it is possible to see that the previous models are overfitting, which means the model performs better on the train set than on unseen data (validation set). Thus, the model is not generalizing as intended, so different methods were tested to try to overcome it.

6.3.1. Reducing the Number of Layers (results in [Table 2](#))

First, a simpler model was tested, with only 3 convolutional layers (with 32, 64, 64 filters, respectively), each with a pooling layer, as the first models, and two dense layers (the first with 100 units).

6.3.2. Data Augmentation and New Callbacks (results in [Table 2](#))

In the following models, data augmentation was introduced, to increase diversity on the training set by applying transformations such as image rotation and shearing. The patience in *Early Stopping* was reduced to 3 epochs and *ReduceLROnPlateau* callback was used to reduce the learning rate if the validation loss shows no improvement after 2 epochs.

6.3.2.1. Adding Dropout Regularization

Dropout regularization, which consists of randomly dropping out a number of output features of the layer during training, was tested with three different values: 0.5, 0.4 and 0.3.

As shown in [Table 2](#), the model with dropout of 0.4 has the smallest gap between train and validation scores, being the one who best fixes overfitting while maintaining a good f1 score. Although the model with a dropout of 0.5 got a lower validation loss and f1 scores were better, beyond the fact that is overfitting slightly more, it is also less efficient since it took more epochs to terminate.

As the dropout value increases, the model becomes simpler and thus, the overfitting decreases, as shown in [Table 2](#). Also, the higher the dropout value, the more epochs are ran by the model.

6.3.2.2. Adding L1 Regularization

An L1 regularization penalty was applied on the layer's kernel, since it is known to be more robust than L2 regularization. Three values were used for the lambda value: 0.001, 0.0001 and 0.00001. The higher the value of lambda, the simpler the model, and consequently the less prone to overfit it is, as showed in [Table 2](#).

In Figure 1, it is possible to observe this thought, since, although the L1 model with 0.00001 of lambda has better f1 score along the various epochs, it is, in fact, overfitting. It can be also verified that the number of epochs is also larger when compared with models with lower lambda values, as shown in the plot below. As shown, the regularization with the smallest parameter registered the best F1 score out of the three. However, it is overfitting. The best L1 model seems to be the one with 0.001 as the lambda value.

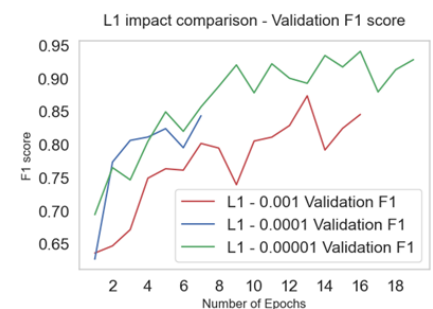


Figure 1: L1 impact comparison – Validation F1 score

7. Feature Extraction

It was decided to perform Feature Extraction to enrich the analysis done with the previous models, based on the mentioned work of Kang et al (2021).

Deep learning feature extraction is a transfer learning technique that uses a pre-trained network as a feature extractor. This means that an input image can propagate forward, stopping at a specific layer, defined by the user. It is possible to take the outputs of that layer as a feature vector to other models, or to integrate it directly into another network.

The first feature extraction approach is called standalone feature extraction. In this technique, a pre-trained model, or some portion of it, is used to pre-process images and extract relevant features of it (Brownlee, 2020). Thus, the extracted features of the images can be converted to a vector of numbers that the model will use to describe specific features in the images. These features can subsequently be used as input in the development of a new model, in this case, of machine learning models.

Another technique used in this project is integrated feature extraction, where the extracted features are directly integrated into a new convolutional network. In this kind of approach, the weights of the pre-trained model can be frozen, so that they are not updated as the new model is trained.

7.1. Standalone Feature Extraction

The model chosen to perform the feature extraction was the model that performed the best, with data augmentation and dropout of 0.4. Considering that the task to be performed after this feature extraction is the same as the task being performed in the convolutional network previously trained, brain tumor classification, it was defined that the layer where the features were going to be extracted was the last convolutional layer. In this layer, the extracted features are interpreted in the context of a classification task, which is exactly the intention of this section.

After predicting the model, the extracted features are used as train, test and validation data for machine learning models. From the literature read, Adaboost, Support Vector Machines, K-NN and Random Forest seemed to be the models with best results, so those were the models trained.

For Adaboost, the base estimator used was a Decision Tree Classifier, with a learning rate of 0.2 and number of estimators 100. In Random Forest, the number of features to consider when looking for the best split is set to the square root of the total number of features (Kang, Ullah, & Gwak (2021)) and the number of estimators to 100. In K-NN, 6 neighbors were used.

To apply the Support Vector Classifier, radial basis function (RBF) was used as kernel. Because this kind of classifier is only used with binary classification problems, One-vs-Rest was applied, which means that binary classifiers that distinguish between one of the labels and the rest are built.

Even though all the models performed relatively well, there was always some overfitting in the results. Taking that into account, the model that performed the better was Adaboost, with F1-score of 0.833 and 0.817 in the train and validation sets, respectively. Even so, the model selected in Section 6.3.2.1 gave a better result.

7.2. Integrated feature extraction with pre-trained models

Two different pre-trained models were used: DenseNet121 and VGG16. The images used were in RGB since the schema of these models demands 3 channels and it was not possible to surpass this.

7.2.1. DenseNet121 (results in [Table 2](#))

Dense Convolutional Network (DenseNet) connects each layer to every other layer in a feed-forward way (Huang (2016)).

The DenseNet with 121 layers model was loaded from Keras using the pre-trained *imagenet's* weights, with '*include_top*' set to False, meaning the fully-connected output layer of the model used to make predictions is not loaded, which permits the addition and training of a new output layer. The model's attribute "trainable" was set to False, freezing its layers so they will not be updated during training. Since we were testing the model for the first time, the additions to the model were taken from a [notebook](#) on Kaggle used on an Alzheimer's dataset. To the model output it was added a dropout of 0.5, then the convolutional output was flattened, followed by a Batch Normalization, which can accelerate the training process and improve the performance of the model (the batch size was 20 in both train and validation). Three Dense layers were added, all with 1024 neurons, a *He Uniform* kernel initializer and relu activation. Batch Normalization was performed every time, such as the dropout of 0.5. In the end, the output layer was added with 3 neurons and softmax activation function.

7.2.2. VGG16 (results in [Table 2](#))

For VGG16 model (Simonyan, 2014), the code used was based on an article found on Medium (Taş, 2021). The model was loaded from keras exactly as the DenseNet121 model. The layers were also frozen. To the output of the model, a Flatten layer was built, followed by a fully connected dense layer with 1024 neurons and relu activation. In the end, an output layer like the one added in DenseNet121 was added. The results have shown overfitting, that could be reduced, however the results of DenseNet121 were already satisfactory, and it was considered that the computational expense was not worth it.

8. Results Conclusion

The results of the models presented in Sections 6 and 7 are shown in Table 2.

Model Description	F1 Train	F1 Validation	Validation Loss	Total Epochs	Time (seconds)
Basic Random Model	0,9669	0,89	0,2637	15	1560,86
Basic Model with Increasing Number of Filters	0,9669	0,9051	0,2227	15	870,04
Basic Model with Reduced the Number of Layers	0,9638	0,9156	0,2467	11	842,11
Augmented Model with Dropout of 0.5	0,9789	0,9656	0,1221	22	1825,44
Augmented Model with Dropout of 0.4	0,9644	0,9594	0,1461	17	1386,45
Augmented Model with Dropout of 0.3	0,9413	0,9332	0,1748	12	973,49
Augmented Model with L1 Regularization of 0.001	0,8548	0,8735	0,7524	16	1329,74
Augmented Model with L1 Regularization of 0.0001	0,805	0,8113	0,5739	7	547,14
Augmented Model with L1 Regularization of 0.00001	0,9786	0,9406	0,2675	19	1351,46
DenseNet121 Pre-trained Model	0,9264	0,9253	0,1671	10	2412,62
VGG16 Pre-trained Model	0,974	0,9317	0,1763	11	5783,10

Table 2: Models Results

The Augmented Model with Dropout of 0.4 has proven to be the best of all regarding both scores and overfitting (Table 2 and Figures 2 and 3).

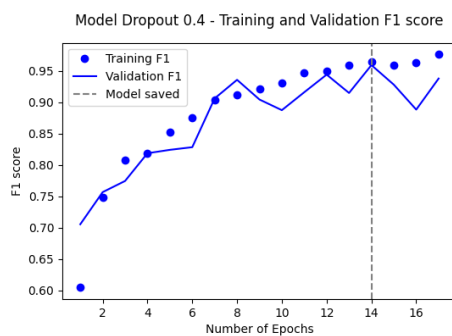


Figure 2: Model Dropout 0.4 – Training and Validation F1 score

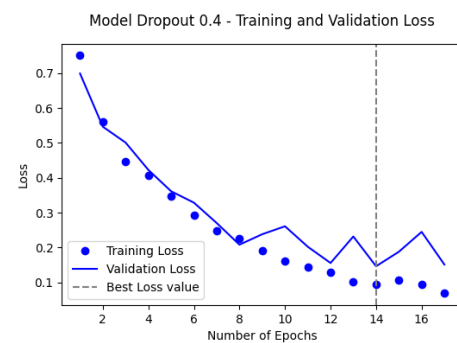


Figure 3: Model Dropout 0.4 – Training and Validation Loss Score

In order to access the generalization power of the model developed, unseen data (test set) was presented to the model. The results are presented in Tables 3 and 4.

		Predicted Labels		
		Meningioma	Glioma	Pituitary
True Labels	Meningioma	65	5	1
	Glioma	5	125	2
	Pituitary	1	0	92

Table 3: Confusion Matrix – Test Set – Model Dropout 0.4

It seems the most frequent prediction error is that the model is not able to correctly distinguish meningioma from glioma, while, for instance, pituitary is never classified as glioma and only once as meningioma. It would be important to know if these errors would lead to a significant difference on the patient treatment and if, in fact, meningioma and glioma are actually difficult to distinguish.

	Precision	Recall	F1-score
Meningioma	0.92	0.92	0.92
Glioma	0.96	0.95	0.96
Pituitary	0.97	0.99	0.98
	Average		0.95

Table 4: Classification report – Test Set – Model Dropout 0.4

Having in mind the data has multiclass labels, both recall and precision are crucial to maximize. It is desired to miss as few positive or negative instances as possible, since a false positive of meningioma tumor, for example, might lead to a different treatment. Similarly, a false negative might also be problematic if the predicted class tumor requires a very different treatment approach.

Therefore, F1-score was the chosen metric, since it considers both recall and precision. The global F1 score for the test set is about 0.95.

9. Grad-CAM

Knowing the importance of the type of data we are predicting, and since in CNN it is not clear what is happening inside the network, it was important to understand which parts of the image the model is using to classify the images. For that purpose, Grad-CAM was applied to images of the test dataset.

Gradient-Weighted Class Activation Mapping (Grad-CAM) is a technique that uses the gradients of a target (in this case, each of the classes) as weights to highlight important regions in images, which makes it possible to see what each layer of the network looks at in each input image.

In some of the images of the test dataset, Grad-CAM seemed to properly identify the tumor, while in others it would look at irrelevant parts of the image, like the eye or the shape of the head. Thus, a human analysis was done, to understand if this could be caused by a possible bias in the dataset, which would explain the good results in train and test data, even though the model would consider irrelevant parts of the image as relevant. However, a pattern that could explain this behavior could not be identified. That and possible causes for what can be deemed as wrong identification of the relevant area at first sight are discussed in Section 10.

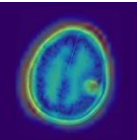
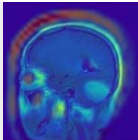
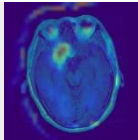
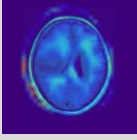
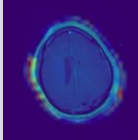
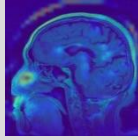
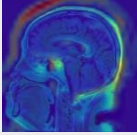
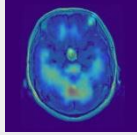
		Predicted Class		
		Meningioma	Glioma	Pituitary
Actual Class	Meningioma			
	Glioma			
	Pituitary		No pituitary image was predicted as glioma Tumor	

Table 95: Actual and predicted classes with Grad-CAM images

10. Conclusions and Limitations

While developing a Deep Learning model that can correctly classify MRI images into three different categories of brain tumors was quite challenging, the results obtained were quite satisfactory, although, as this is a medical problem that could cost lives, these should be used carefully and

demand more expertise on the matter. Nevertheless, it could potentially be used as a first step towards the final diagnosis, doing an initial filtration of the images.

Image segmentation would have been a useful approach to complement this study, however, there was no information of the true mask (areas of the tumor). In fact, the data in the format mat indicates the tumor mask: a binary image with 1s indicating tumor region, however a feasible way to pre-process this data was not found.

The results obtained with Grad-CAM can make sense when analysed by an expert. Perhaps what this technique identifies as relevant is indeed important to understand which tumor we are facing. Having an expert analyzing the images of Grad-CAM could be an interesting approach in the future, since none of the members of the group has extensive enough knowledge on this field.

11. References

- Github Link for this project: https://github.com/helenado/DeepLearning_Group_Project
- Brain tumor dataset: https://figshare.com/articles/dataset/brain_tumor_dataset/1512427
- Kaggle Brain Tumor Image Dataset: <https://www.kaggle.com/datasets/denizkavi1/brain-tumor>
- Amidi, A., & Amidi, S. (n.d.). *A detailed example of data generators with Keras*. Stanford University. <https://stanford.edu/%7Eshervine/blog/keras-how-to-generate-data-on-the-fly>
- Brownlee, J. (2020, August 18). *Transfer Learning in Keras with Computer Vision Models*. Machine Learning Mastery. url.it/3n3at
- Cheng, J., Huang, W., Cao, S., Yang, R., Yang, W., Yun, Z., Wang, Z., & Feng, Q. (2015). Enhanced Performance of Brain Tumor Classification via Tumor Region Augmentation and Partition. *PLOS ONE*, 10(10), e0140381. <https://doi.org/10.1371/journal.pone.0140381>
- Duong, B. T. (2022, March 30). *Explainable AI: Brain Tumor Classification with EfficientNet and Gradient-Weighted Class Activation Mapping (Grad-CAM) Visualization*. Medium. url.it/3n3ar
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- Huang, G. (2016, August 25). *Densely Connected Convolutional Networks*. Cornell University. <https://arxiv.org/abs/1608.06993>
- Kang, J., Ullah, Z., & Gwak, J. (2021). MRI-Based Brain Tumor Classification Using Ensemble of Deep Features and Machine Learning Classifiers. *Sensors*, 21(6), 2222. <https://doi.org/10.3390/s21062222>
- Manav, D. (2020, December 27). *DenseNet 121 feature extraction*. Kaggle. url.it/3n3ax
- Memari, I. (2021, December 27). *Precision, Recall, Accuracy, and F1 Score for Multi-Label Classification*. Medium. url.it/3n3am
- Nayak, D. R., Padhy, N., Mallick, P. K., Zymbler, M., & Kumar, S. (2022). Brain Tumor Classification Using Dense Efficient-Net. *Axioms*, 11(1), 34. <https://doi.org/10.3390/axioms11010034>
- Reiff, D. (2022, April 18). *Understand your Algorithm with Grad-CAM - Daniel Reiff*. Medium. <https://medium.com/@daniel.reiff2/understand-your-algorithm-with-grad-cam-d3b62fce353>
- Simonyan, K. (2014, September 4). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Cornell University. <https://arxiv.org/abs/1409.1556>
- Taş, F. (2021, December 26). *CNN, Transfer Learning with VGG-16 and ResNet-50, Feature Extraction for Image Retrieval with Keras*. Medium. url.it/3n3ah
- Zhang, Y. J. (2015). Machine Learning for Image Classification. *Encyclopedia of Information Science and Technology, Third Edition*, 215–226. <https://doi.org/10.4018/978-1-4666-5888-2.ch021>