cs584 Final Project Report

Mouhammad BAZZI – A20522180

Department of Computer Science

Illinois Institute of Technology

BRAIN TUMORS CLASSIFICATION

This present document is the report of the project that I did. You could find all the code and other documents on this GitHub link: https://github.com/mhd-baz/CS584-TumorsBrain.

I. PROBLEM STATEMENT

A brain tumor is one of the more severe disorders affecting both children and adults. 85 to 90 percent of all primary Central Nervous System (CNS) malignancies are brain tumors. For those who have a cancerous brain or CNS tumor, the 5-year survival rate is roughly 35%. Magnetic Resonance Imaging is the most effective method for finding brain cancers (MRI). The scans provide a high-quality picture. The radiologist examines these pictures. Because of the complexity of brain tumors and their characteristics, a manual examination can be prone to inaccuracy.

The aim of this project is to try to create some **Neural Networks** that could **outperform manual categorization** in terms of accuracy. As a reference, I looked if there are some papers that tried to solve this problem with some Machine Learning model, and there is different kind of model (CNN for almost all of them) that have an accuracy between 81% and 97%.

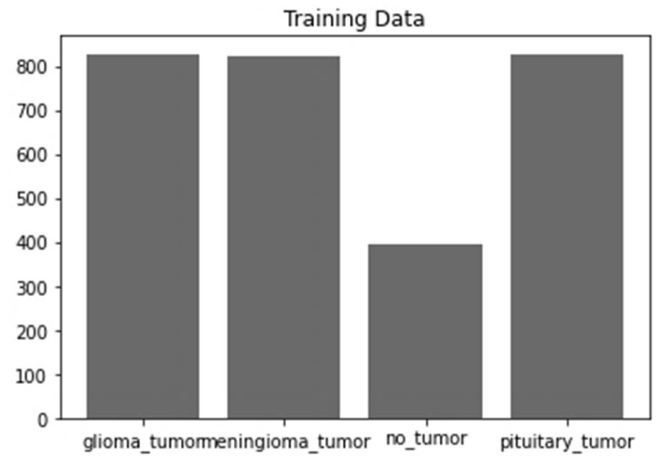
II. PROPOSED SOLUTION

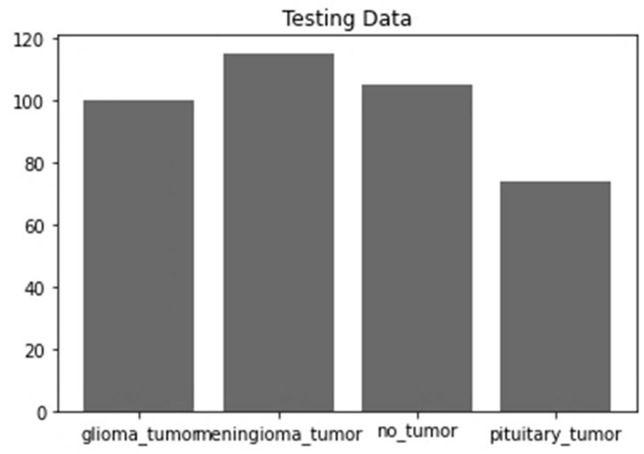
I divide this section into three to cover Data Set, Model Architectures, and Optional Pre-Processing (filters).

A. Data Sets

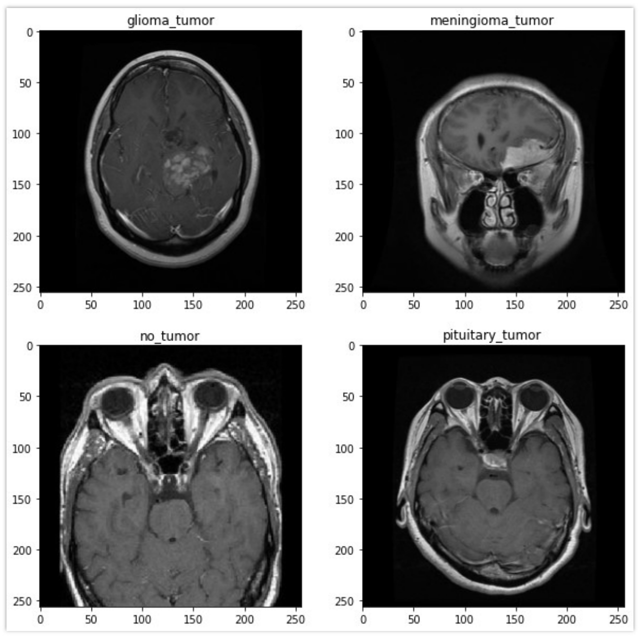
To try to solve this problem I used a similar Data set to what I found in the papers: it's the **Brain Tumors Classification (MRI) [1]**. This dataset is composed of 2 subfolders (training and testing folder). And each folder is composed respectively of 2870 and 394 images (12% of the dataset is the Test set). In each folder, there are 4 other subfolders named: ***Glioma Tumor***, ***Meningioma Tumor***, ***Pituitary Tumor,*** and ***No Tumor***. So, there are 4 classes possible and we will do a classification between 3 types of tumors and healthy brains.

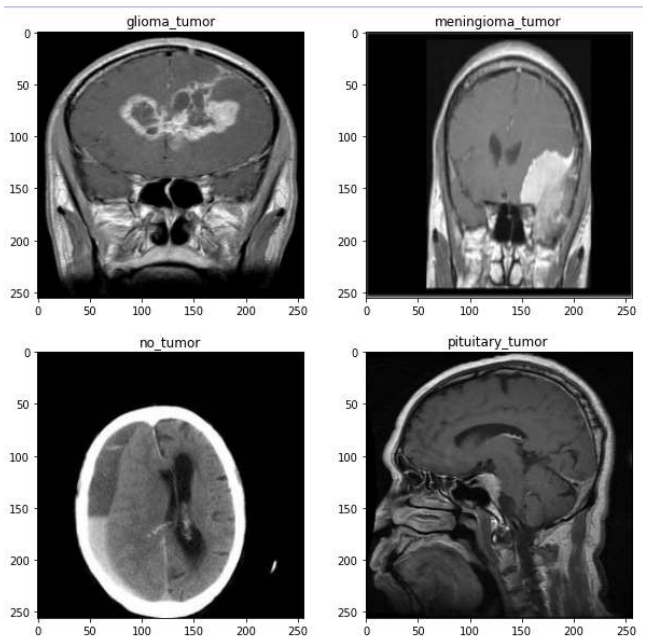
The data is composed of MRI images in grayscale with different resolutions (image sizes). The following plots show the repartition of the different classes in the Training and Testing folder:





By the way the MRI images could be taken from different points of view as we can see here in this few images samples:





After downloading locally (or uploading the data on a cloud service) I had to some steps in order to **load and pre-process all the images**:

1. I **loaded** the images using the OpenCV library and while they are loading I **reshaped** each image as 256x256. (256 is a number chosen by the user).
2. Then I **shuffled** the data and **split** the training set into training and validation sets (using a validation ratio selected by the user).
3. And finally, we **normalize** the data (dividing by 255) and we **one-hot encode** all the labels.

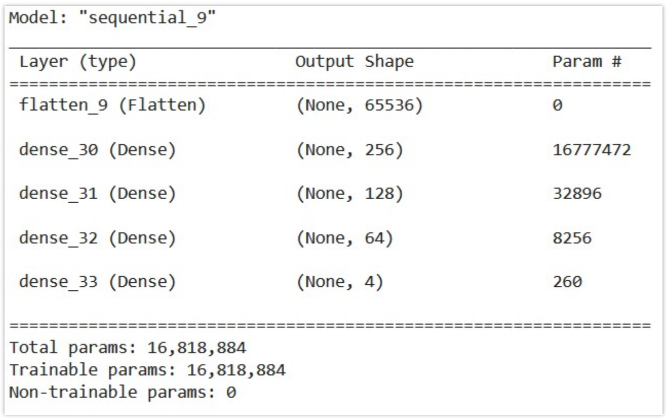
Remark: I did these steps like this in order to prepare my data for my first model (the ANN), but for my second model (the CNN) I basically did the same steps but using the *ImageDataGenerator* object from the *Keras* library. And with *ImageDataGenerator*, if the user set True to the global variable: *WITH_DATA_AUGMENTATION*, I will also perform Data Augmentation on the images at each step of the training.

B. Model Architectures

I designed two models, one simple Artificial Neural Network (fully connected layers only) and another one that is a simple CNN network. All the implementations were done with Keras.

The Artificial Neural Network (ANN):

This model is composed of one flattened layer, 3 hidden layers, and one output layer of 4 units.



```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_9 (Flatten)         (None, 65536)             0

 dense_30 (Dense)            (None, 256)               16777472

 dense_31 (Dense)            (None, 128)               32896

 dense_32 (Dense)            (None, 64)                8256

 dense_33 (Dense)            (None, 4)                 260

=================================================================
Total params: 16,818,884
Trainable params: 16,818,884
Non-trainable params: 0
```

Fig. 1: Summary of the simple ANN model

I use the SoftMax activation for the output layer and the ReLU layer for all the hidden units. We got a model with almost 17 million parameters. This model is compiled with the Adam optimizer and a learning rate of 0.0005 and the categorical cross-entropy as a loss function. The batch size while training is equal to 32.

We also can notice that there are no Dropout and Batch normalization layers and also there is no regularization used in any hidden layer.

This model is used only in order to be a reference as the worst model (in terms of accuracy) because we expect that a simple ANN will not perform well in comparison to a CNN model.

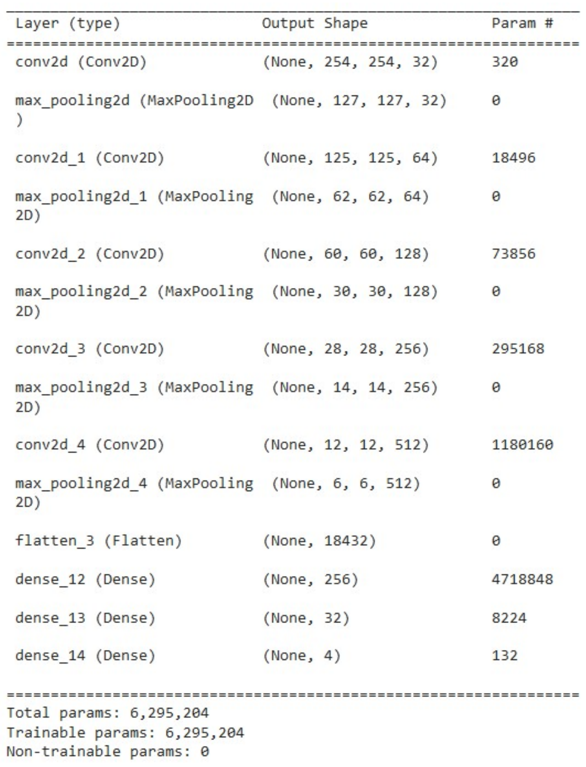
The Convolutional Neural Network (CNN):

I designed this model in order to have better performance (than the ANN) and to try to reach the 81% accuracy got in other papers.

This model is composed of 5 convolution layers (the more the convolutional layers are in-depth the more we add channels). And each convolutional layer is followed by a Max-Pooling layer. Then we flatten everything and we feed a fully connected network of 2 hidden layers (with ReLU activation) and one output layer with SoftMax activation. I used this time a learning rate of 0.001 (and same metrics, loss and optimizer as for the ANN model).

This model reduces the number of parameters by almost 3 in comparison with the ANN model. So, there is a good improvement and we also expect to have better results.

By the way, this model will be trained on the data without data augmentation (this will produce a first model) and will also be trained with data augmentation (this will produce a second model). In order to compare their performance.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      320

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 256)      0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 512)       1180160

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 512)        0
 2D)

 flatten_3 (Flatten)         (None, 18432)             0

 dense_12 (Dense)            (None, 256)               4718848

 dense_13 (Dense)            (None, 32)                8224

 dense_14 (Dense)            (None, 4)                 132

=================================================================
Total params: 6,295,204
Trainable params: 6,295,204
Non-trainable params: 0
```

Fig. 2: Summary of the simple CNN model

Also, I used CNN in order to have the ability to understand how well is the model by looking at and visualizing what is learned by the model. With the ANN model, it’s hard to understand the learning process of the layers.

To do so we will implement 3 types of visualization:

1. **Visualization of the output** of the activations layers for a given input
2. **Visualization of the filters learned** by finding the image that will maximize the output of the activation using the gradient ascent algorithm.
3. And finally **visualizing attention**, i.e. the heat map on a given image of the part that contributes the most to the prediction.

IMPORTANT - Hyperparameters:

All the hyperparameters were tuned using the validation set, and the reason why there are no dropout layers or no regularization or I use the Adam optimizer and not the RMSProp, etc. is because that’s what I found the best for having the best accuracy.

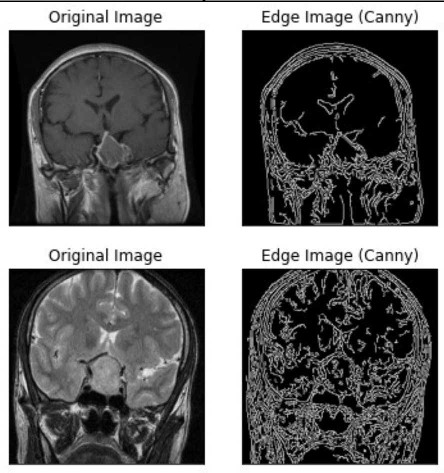
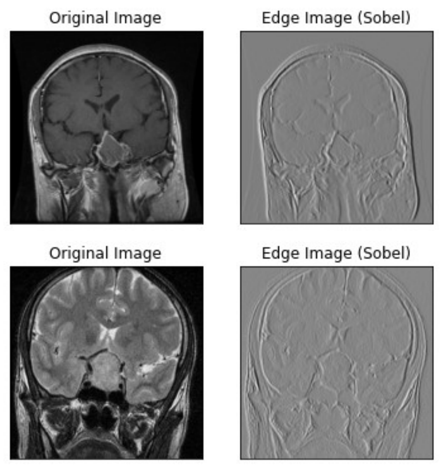
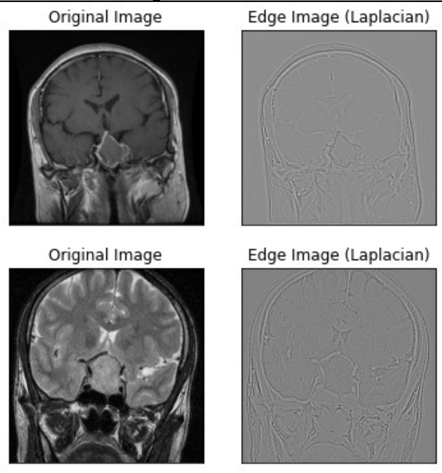
The only hyperparameters that will change in the following will be the number of epochs and all the rest of the hyperparameters will still be the same because they are the optimum that I found.

By the way the validation set is used only for the tuning, each time I evaluate on the testing set, I train from scratch the model using all the training data (training + validation) using the best hyperparameters found before (including the best number of epochs).

C. Optional Pre-Processing (filters)

In order to take advantage of domain knowledge, I wanted to add a pre-processing step by applying some filters to the images. My idea is that maybe like this we will help the CNN model to focus on the real important things in the image by removing the non-important elements and maybe like this the model will increase its accuracy.

To do that I decided to select 3 filters as pre-process (I apply only one filter before each learning). The filters are the Sobel Filter, the Canny filter, and the Laplacian filter. Basically, their objective is to make edge detection and could probably like put more evidence the brain tumors. Here is some example of pictures before/after the application of the filters:

Canny Filter	Sobel Filter	Laplacian Filter
 Original Image Edge Image (Canny) Original Image Edge Image (Canny)	 Original Image Edge Image (Sobel) Original Image Edge Image (Sobel)	 Original Image Edge Image (Laplacian) Original Image Edge Image (Laplacian)

Each filter was implemented using the OpenCV library, and basically, they are a kind of applying gradient filter for the Canny and Sobel, and for the Laplacian, it's a kind of hessian filter.

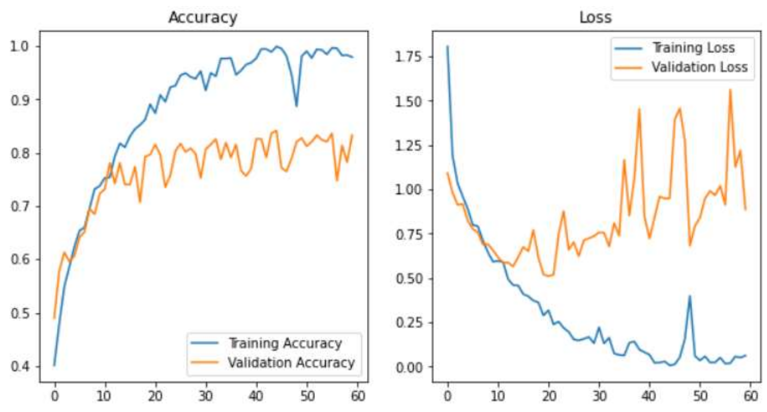
I will try this extra-pre-processing step for the ANN and CNN models. And then compare them with the previous results.

III. RESULTS AND DISCUSSION

In this section, I will present each result I got for each combination of model and pre-processing.

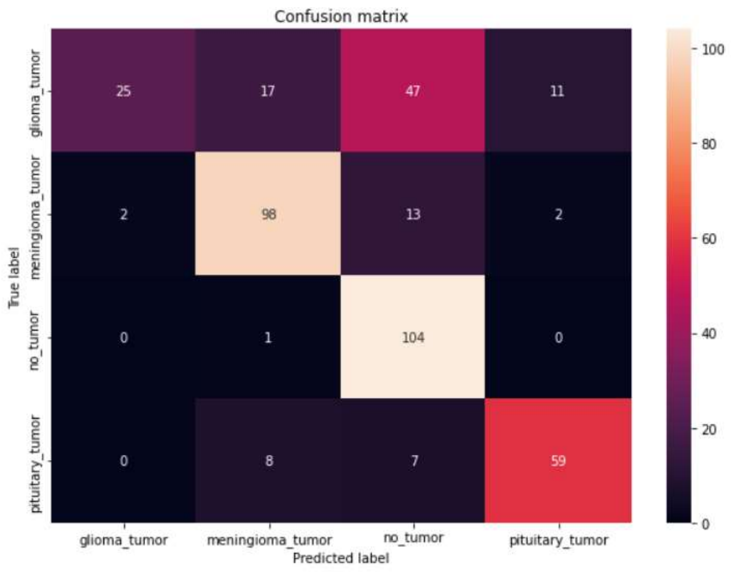
A. ANN

I first trained the ANN model described before (with the same hyperparameters as before) with 60 epochs and we got this:



We can see that we overfit after 18 epochs. So, then I evaluated my model by training it on 18 epochs with all the training set (training + validation), and we got an accuracy on the testing set of: **72.58%**.

I then outputted a confusion matrix in order to understand where our model is struggling (or performing well):

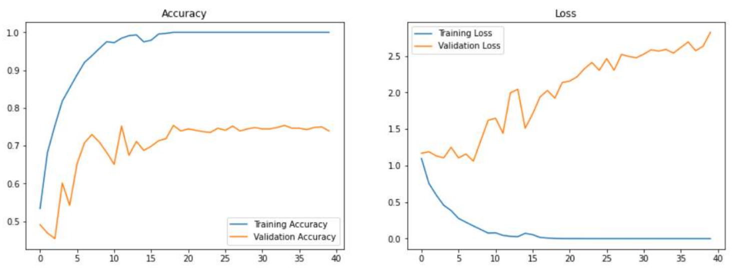


And we can see that our model is almost perfect in the prediction of the non-presence of a tumor, very good for the pituitary tumor, and extremely bad for the glioma tumor prediction. The classifier almost acts as a random classifier for the images that belong to the class of glioma tumor.

But overall for the first model, we can say that it’s not so bad and I hope that the CNN model will perform better.

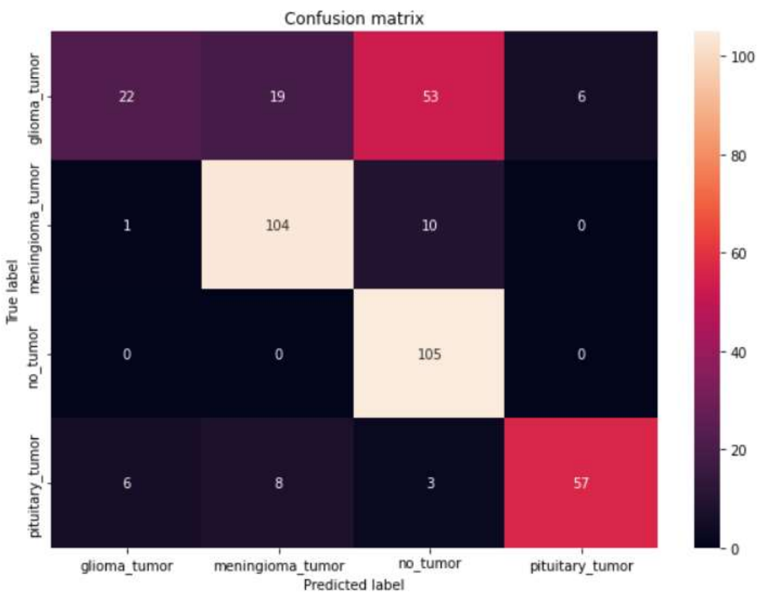
B. CNN without Data Augmentation

Then I trained the CNN model described before (with the same hyperparameters as before) with 40 epochs and we got this:



We can see that we overfit after 8 epochs. So, then I evaluated my model by training it on 8 epochs with all the training set (training + validation), and we got an accuracy on the testing set of: **73.09%**.

I then outputted a confusion matrix in order to understand where our model is struggling (or performing well):



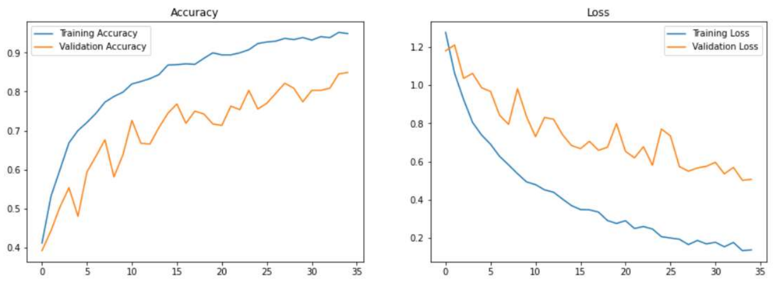
We can notice that we have a slight amelioration in the accuracy than the simple ANN model, but we can also notice that the model doesn't do the same errors as before for example for the pituitary tumor, now the model predicts only 3 times instead of 9 in the previous model a non-tumor for a brain that has a pituitary tumor.

So, that means that this model can classify better a brain that has a tumor or not but the model is not enough good to identify separately all the tumors, especially the glioma tumor.

But overall, I was expecting better results than this.

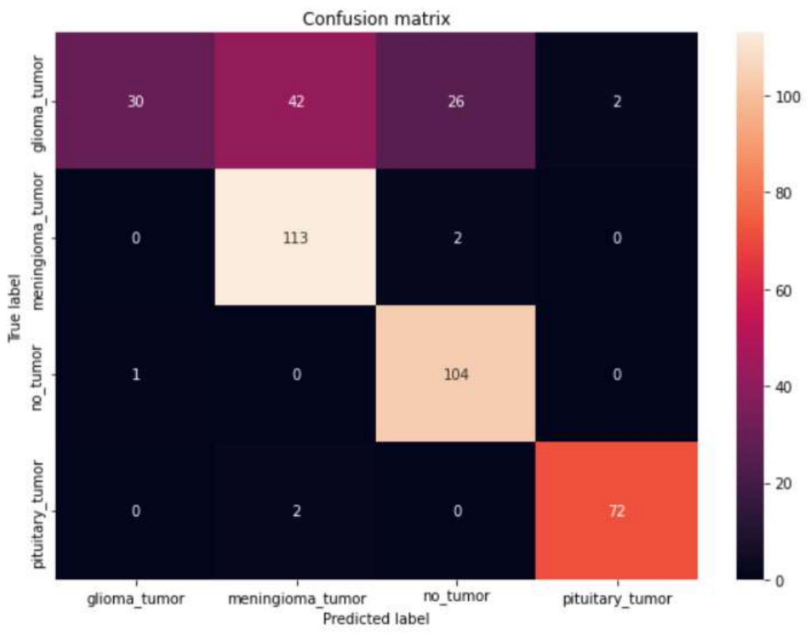
C. CNN with Data Augmentation

Now I trained the CNN model described before (with the same hyperparameters as before) on 35 epochs but this time **WITH** data augmentation (using the ***ImageDataGenerator*** object of *Keras*) and we got this:



We can see don't overfit even after 35 epochs and this is expected because our model, for each epoch, is seeing "new data", so it's very hard to overfit while there is always new data (data augmentation is regularization technique that avoids overfitting quickly). Because the training took a while, I evaluated my model by training it on 35 epochs again with all the training set (training + validation), and we got an accuracy on the testing set of: **80.96%**.

I then outputted a confusion matrix in order to understand where our model is struggling (or performing well):



We can see now that the model is perfect for classifying everything except glioma tumors. But we increased by 40% the accuracy of the detection of glioma tumors in comparison to the previous model.

So, we improved all the classes using data augmentation and we got an accuracy of almost 81%, the accuracy that we were aiming for (the same as in the paper). The data augmentation was very helpful to improve the model accuracy.

D. ANN & optional pre-processing (filters)

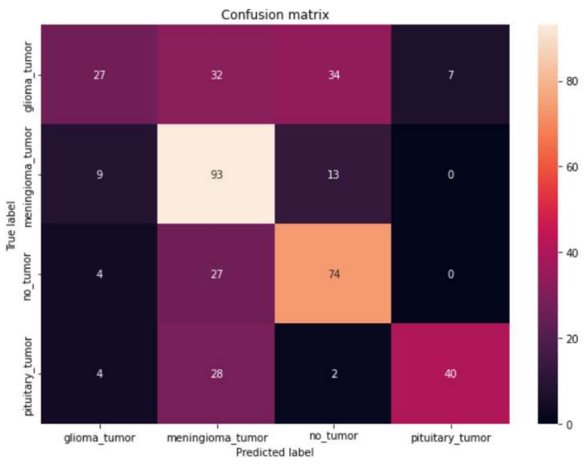
Then I wanted to see if by applying some filters before the training, and therefore by taking advantage of the domain knowledge, we could get better results. So, I decided to compare 3 different filters in the pre-processing task to if we have overall better performance and which filter is the best.

a. Sobel

I trained my ANN model by applying the Sobel filter and by using a kernel size of 3 (I found the best performance using this kernel size). And I got that I overfit after 18 epochs. I evaluate as usual the model on the testing set after 18 epochs of training and I got an accuracy of **69.03%**. So, in comparison with our simple ANN model, it's not good (we lost 3% of accuracy).

b. Canny

I trained my ANN model by applying the Canny filter and by using a minimum threshold of 80 and a maximum threshold of 180 (I found the best performance using these thresholds). And I got that I overfit after 10 epochs. I evaluate as usual the model on the testing set after 10 epochs of training and I got an accuracy of **59.03%**. It's even worst than the Sobel one. And I got the worst confusion matrix ever since the beginning of the experiments:



c. Laplacian

I trained my ANN model by applying the Laplacian filter and by using a kernel size of 3 (I found the best performance using this kernel size). And I got that I overfit after 11 epochs. I evaluate as usual the model on the testing set after 11 epochs of training and I got an accuracy of **55.83%**. So, it's even worse than the performance with the Canny Filter.

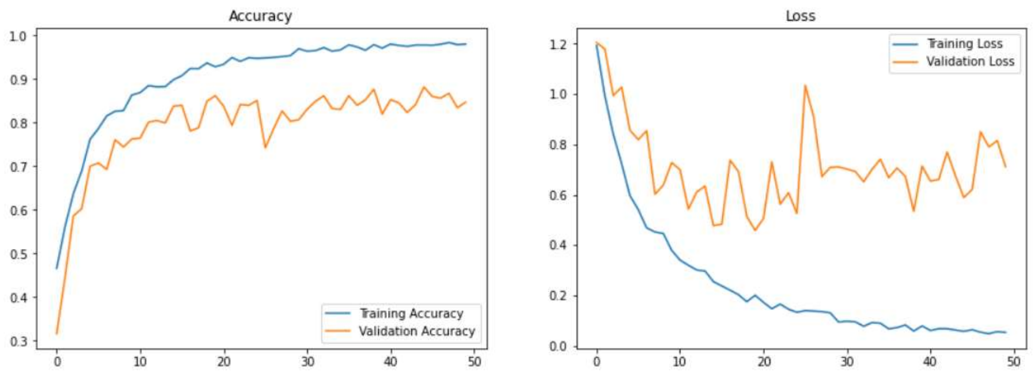
We can conclude two things from this extra pre-processing experiment:

- The first is that we overfit more quickly (except for the Sobel filter). This could be explained by the fact that we simplify the data by applying these filters. So, because the data is simpler and the model is designed for a more complex dataset we should overfit quickly.
- The second thing is that we got a way worst accuracy and this could be explained by the fact that our filters are not adapted because by applying them we lose a lot of information that could be helpful for the model.

So, this experiment is disappointing because I expected to at least improve slightly the results. But I want to try at least the best filter of the previous 3 on a CNN model because maybe this type of model (in opposition to the ANN one) could maybe be more sensitive to this pre-processing part.

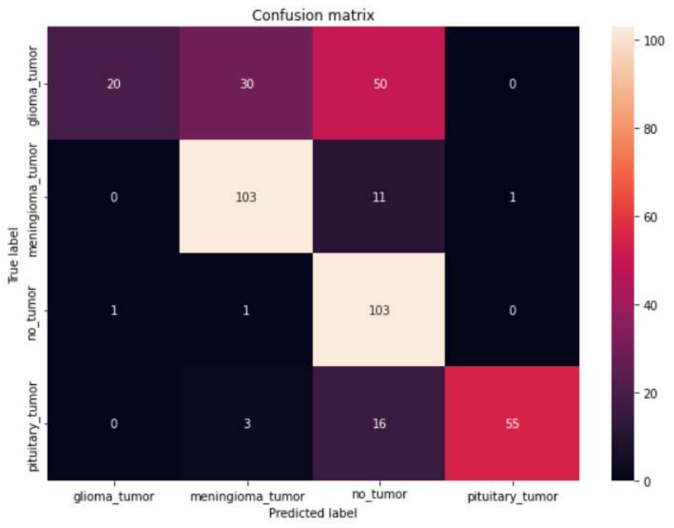
E. CNN & optional pre-processing (Sobel filter)

So as explained before, this will be my last experiment, I will take the same CNN model, apply the Sobel filter with a kernel of size 3 and use data augmentation and see what I will get.



We can see that we overfit at 14 epochs and this is even if we use data augmentation. This is expected because as said before the extra pre-processing step that we do by applying the filter simplifies the data and therefore we tend to overfit quickly, but thanks to data augmentation we overfit we don't overfit as quickly as our first CNN model.

Then we trained our model on 14 epochs using all the training data (training and validation set) and we evaluate the model on the testing set and we got an accuracy of **71.31%**. And this is the confusion matrix on the test set:



True label \ Predicted label	glioma_tumor	meningioma_tumor	no_tumor	pituitary_tumor
glioma_tumor	20	30	50	0
meningioma_tumor	0	103	11	1
no_tumor	1	1	103	0
pituitary_tumor	0	3	16	55

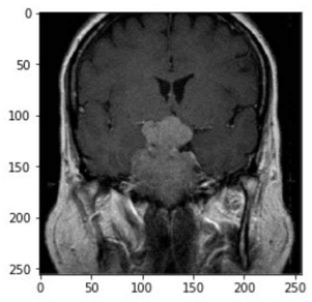
We can see that even with a CNN model we do worse performance than the 2 previous CNN models (without and with data augmentation).

So, I can conclude that this is probably for the same reason before: by applying this filter we lose important information that is needed for the model.

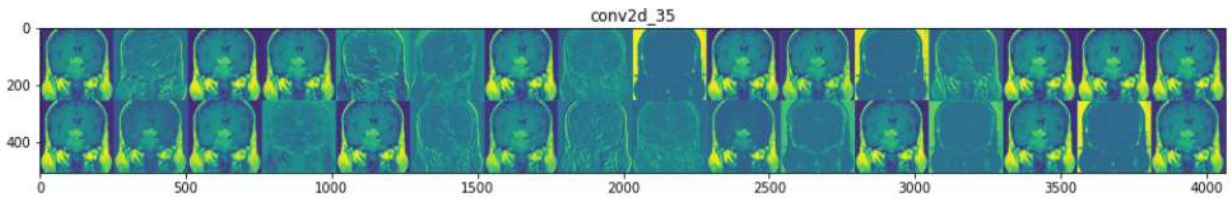
F. Visualization of our best CNN model

Our best model is the CNN model with data augmentation and without applying any filter as a pre-processing step. We got an accuracy of 80.96% on the test set with this model. We know that this model is extremely accurate in classifying all the classes except the glioma tumor. But what could be interesting is to see and visualize what was learned by the model.

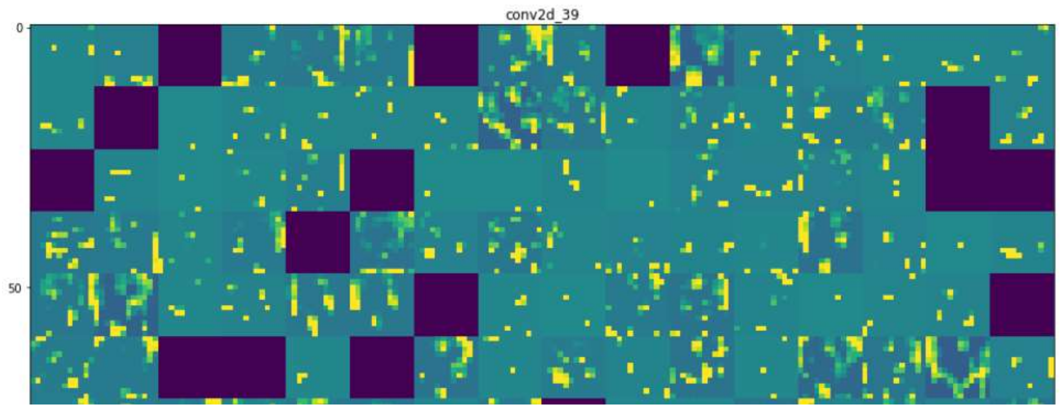
- The first thing we want to **visualize is the output of the convolutional layers** for a given input. We will use this image as input (it belongs to the class pituitary tumors):

  And That's what the activation output of the first convolution layer that we got:

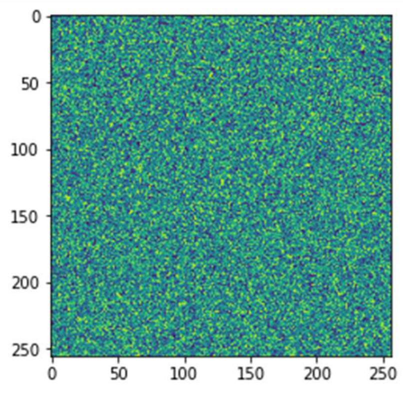
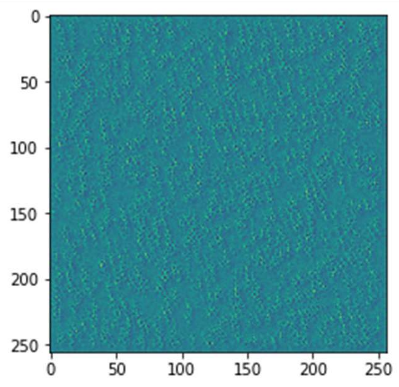
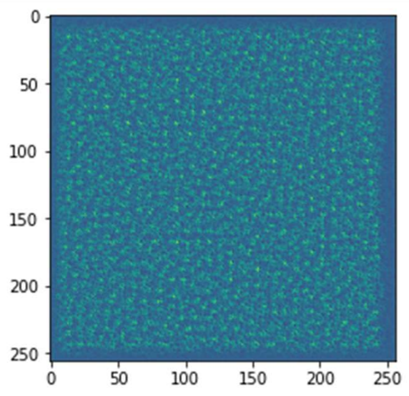
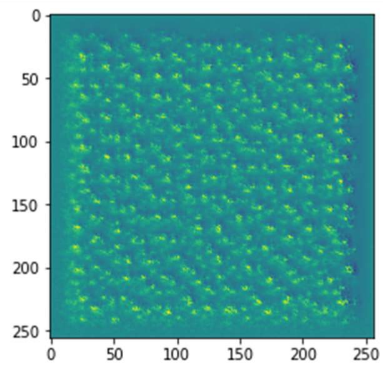
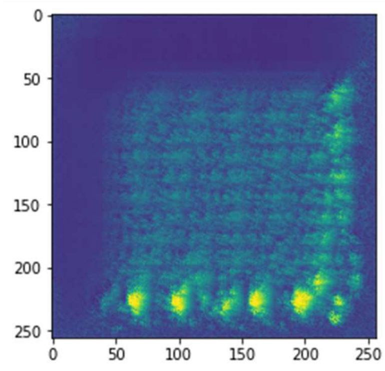
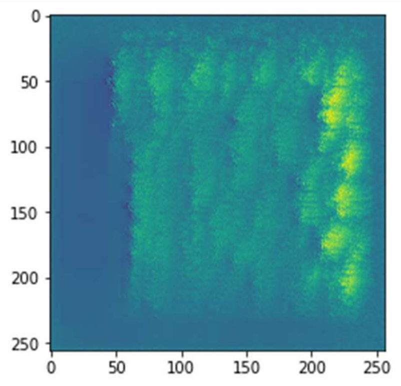
  And the following image is the same thing but for the last convolution layer (I will not show the output of all the units because there are 512 units):

  What we can notice is that the first layer basically activates some edges, or even work as the identity, so our model on the first layer detects only low-level features, and all the units output something. While the last convolutional layer seems to activate only specific regions of the image, it seems to activate high-level features. We can also notice that a lot of channel output nothing, we can interpret that by saying that these images do not have the feature that is detected by this unit.

- That's interesting to see, but it's not enough to see if the model is doing good or not. So, the next thing that I want to **visualize is the filter learned by the units of each convolutional layer**. To find this image we want to take a random image and try to update it by using the gradient ascent algorithm in order to find the image that will maximize the output of each unit of each layer.

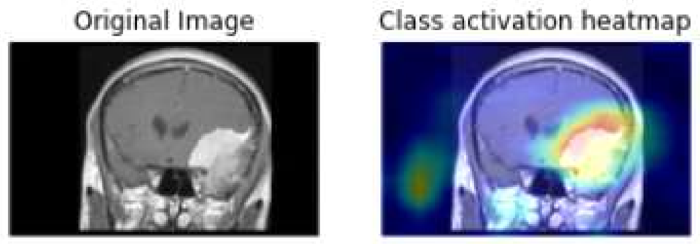
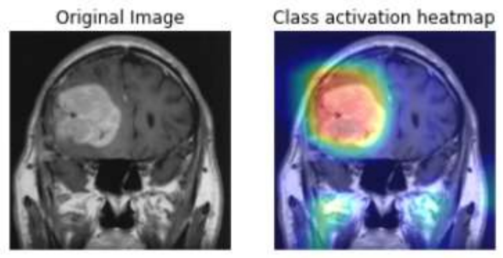
And that’s some images (filter learned by the units) that we got from different convolutional layers:

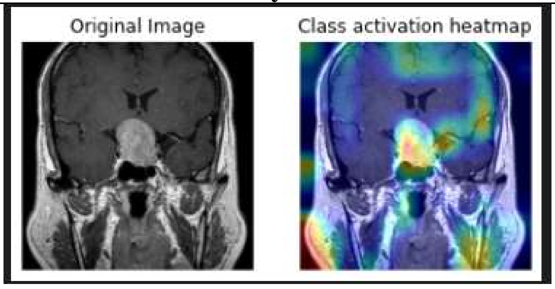
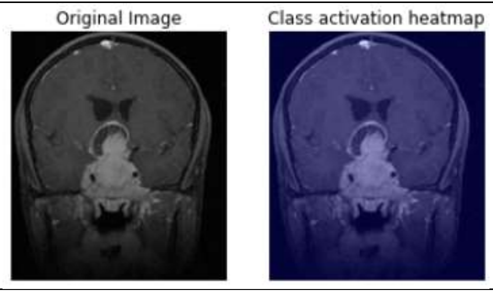
Conv Layer 1 - Filter 4	Conv Layer 2 - Filter 58	Conv Layer 3 - Filter 96
		
Conv Layer 4 - Filer 238	Conv Layer 5 - Filter 53	Conv Layer 5 - Filter 89
		

To be honest, I can’t really understand them in order to know if the model is learning the correct things or not, but the thing that I am sure about it is that the more in-depth we go in the convolutional layers, the more the filters learned by the units seems complex and some of them seems to detect some kind of shape of a tumor on key locations in the brain while the units of the first convolutional layers try to detect some edges.

- The last thing that can I try to do to know if the model is learning well or not is to **visualize the attention** of the model, to see a kind of **heatmap** on what was made the decision of the classifier.

So that’s some result we got:

Meningioma Tumor	**Meningioma Tumor**
	

Pituitary Tumor	No Tumor
	

We can see that the model is not so aberrant, the decision of the classifier is sometimes based on the real tumor location but also on some other things on the image, so it’s not a perfect classifier, but almost every time the area that explains the classification of the classifier is the area where there is the brain tumor.

It’s also interesting to see which part is activated when we look at the images that do not have any brain tumor, and we can notice that none of the parts of the brain are activated, that’s what we could expect from the model.

IV. CONCLUSIONS

To conclude we can say that the CNN model is way better than the ANN model as we could expect because we are dealing with images. Using data generation avoids overfitting as expected and also allows us to help the model to better generalize and lead us to have a model with an accuracy of **80.96%**. This is a very good accuracy, especially for a simple CNN like the one I used.

This model is also super-efficient to classify all the classes except the Glioma Tumors one. It was the hardest class to classify for each model. To confirm the fact that this model is a good classifier for the 3 others classes I tried to apply different methods to visualize different things from the classifier and to see what it really learned. And especially after visualizing the attention of the model, I can confirm that the classification is based on the coherent part of the image.

In this project, we also tried to add a pre-processing step that consist in applying a filter in the images in order to try to take advantage of the domain knowledge, but it was a failure and we had even worse results.

This project was very instructive, it helps me to apply what we saw in class and also push me to try new things (I learned a lot of things about all the filters like the Sobel, Laplacian, and Canny ones, but also, I learned how to visualize what is learned by a CNN model by using a different method).

If I could have more time to work on this project, I would try other CNN architectures (with residual blocks, or inception blocks for example).

V. REFERENCES

[1] Brain Tumors Classification (MRI). Kaggle, https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri?resource=download, 2019.

[2] Jun Cheng, Wei Huang, Shuangliang Cao, Ru Yang, Wei Yang, Zhaoqiang Yun, Zhijian Wang, Qianjin Feng, Enhanced Performance of Brain Tumor Classification via Tumor Region Augmentation and Partition. PLoS One, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4598126/, 2015.

[3] Hossam H. Sultan, Nancy M. Salem, And Walid Al-Atabany, Multi-Classification of Brain Tumor Images Using Deep Neural Network. IEEE Access,https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8723045, 2019.

[4] Wikipedia, https://en.wikipedia.org/wiki/Sobel_operator.

[5] Wikipedia, https://en.wikipedia.org/wiki/Laplace_operator.

[6] Wikipedia, https://en.wikipedia.org/wiki/Canny_edge_detector.