UNIVERSITÀ DI PISA

Master Degree in Artificial Intelligence and Data Engineering

Computational Intelligence and Deep Learning

Brain Tumor Classifier

Lorenzo Bianchi

Github repository:
https://github.com/lorebianchi98/BrainTumorClassification

Contents

1 Introduction

A brain tumor is a collection, or mass, of abnormal cells in your brain. The skull, which encloses the brain, is very rigid. Any growth inside such a restricted space can cause problems. Brain tumors can be cancerous (malignant) or noncancerous (benign). When benign or malignant tumors grow, they can cause the pressure inside your skull to increase. This can cause brain damage, and it can be life-threatening. Early detection and classification of brain tumors is an important research domain in the field of medical imaging and accordingly helps in selecting the most convenient treatment method to save patients life therefore[1].

The best technique to detect brain tumors is Magnetic Resonance Imaging (MRI). A huge amount of image data is generated through the scans. These images are examined by the radiologist. A manual examination can be error-prone due to the level of complexities involved in brain tumors and their properties. Application of automated classification techniques using Machine Learning(ML) and Artificial Intelligence(AI) has consistently shown higher accuracy than manual classification. Hence, proposing a system performing detection and classification by using Deep Learning Algorithms using ConvolutionNeural Network (CNN), Artificial Neural Network (ANN), and TransferLearning (TL) would be helpful to doctors all around the world.

In 2021 on Kaggle it was updated a dataset containing 7022 images of brain MRI scans with both healthy and unhealthy brains[2]. This dataset could be used in order to perform task of brain tumor detection, brain tumor segmentation or brain tumor classification. Our work will be focused on this last task.

1.1 Tumor types

The classifier we aim to build will have the capability to discriminate from 3 distinct types of brain tumor: **glioma**, **meningioma** and **pituitary**.
Gliomas are tumors in the central nervous system (brain or spinal cord) and peripheral nervous system that form out of various types of glial cells (neuroglia). Glial cells are referred to as “supportive cells” because they surround, insulate, feed, repair, and protect neurons which transmit electrical signals and information throughout the nervous system. They do not directly influencing synaptic transmission and electrical signals but rather provide supportive functions for neurons and the transmission of information. [3]

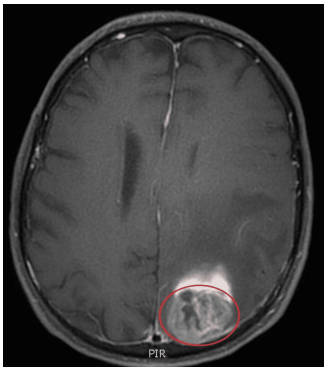


Figure 1: A **glioma** tumor

A **meningioma** develops on the brain's meninges and is a tumor of leptomeningeal origin. It is believed that they may be caused by a genetic abnormality or chromosomal defects (e.g. partially or completely missing chromosome 22), however, other studies show that meningiomas reacts to hormonal changes or trauma. [3]

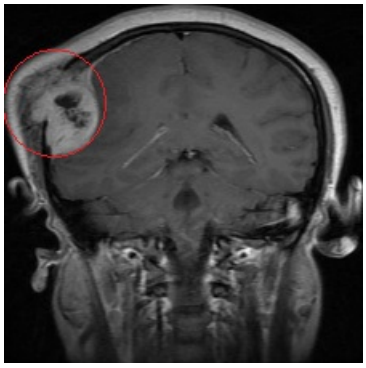


Figure 2: A **meningioma** tumor

A **pituitary** tumor occurs in the anterior body of the pituitary gland. This is a pea-sized gland that is located behind the bridge of your nose. It produces hormones that help to regulate the functions of the other endocrine glands. Pituitary tumors can affect the function of the pituitary gland increasing the production of regulatory hormones and reducing the production of regulatory hormones. [3]

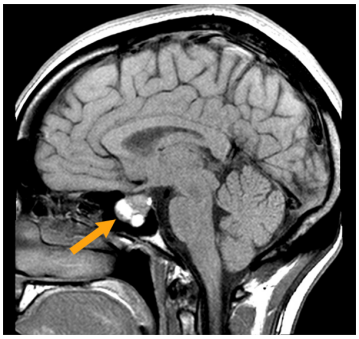


Figure 3: A **pituitary** tumor

2 Related Works

Brain tumor classification using machine learning methods has previously been studied by researchers especially over the past years. The development of artificial intelligence and deep learning-based new technologies has made a great impact in the field of medical image analysis, especially in the field of disease diagnosis. Parallel to this, many studies have been conducted on brain tumor detection and brain tumor multi-classification using CNN, even considering the grade of the tumors. For instance, Çinar and Yildirim (2020) [4] used a modified form of pre-trained ResNet-50 CNN model by replacing its last 5 layers with 8 new layers for brain tumor detection. They achieved 97.2% accuracy using MRI images with this modified CNN model. In a similar manner, Khawaldeh et al. (2017) [5] proposed a modified version of AlexNet CNN model to classify brain MRI images into healthy, low-grade glioma and high-grade glioma. An overall accuracy of 91.16% was obtained using 4069 brain MRI images.

The uploading of the dataset we will use on the Kaggle platform made many users try to perform the task of Brain Tumor Classification, and some of them reached accuracy around 97-98% combining the finetuning of CNNs with ensembling methods.

3 Dataset

This dataset contains 7022 images of human brain MRI images which are classified into 4 classes: glioma, meningioma, no tumor and pituitary. The images are an aggregation of three different datasets: figshare[6], SARTAJ dataset[7] and BR35H[8] (for healthy brain scans). The distribuition of the images per classes is shown below:

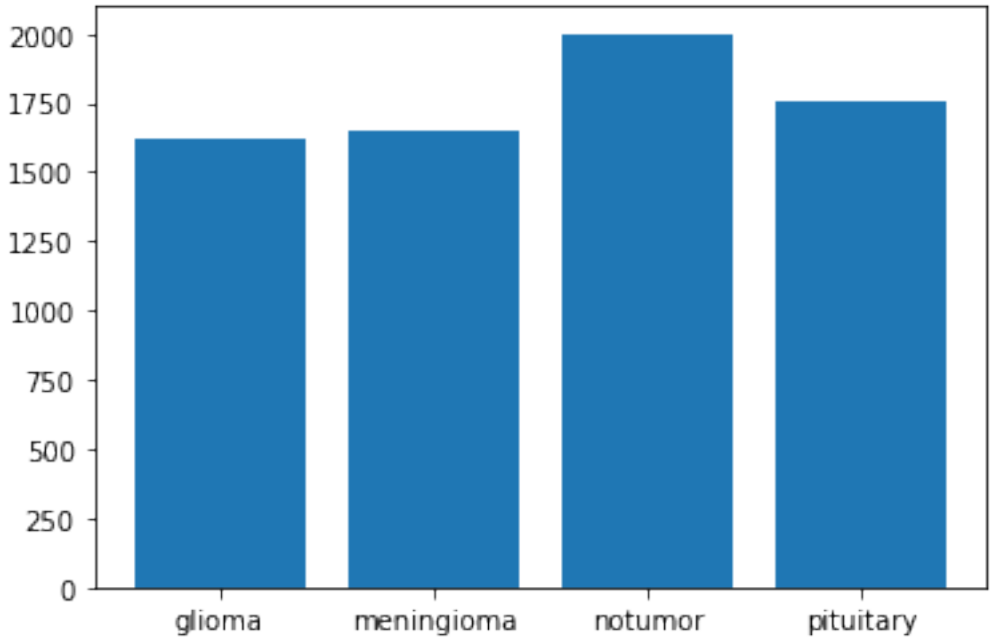


Figure 4: Data distribuition

As we can see the dataset is almost balanced, so it was not necessary to apply undersampling or to increment the number of images of the classes with less images by perform data augmentation only for this classes. The author of the dataset divided it in Training and Testing with a percentage around 81.33%-18.67%, but since this split percentage was only a suggestion and there are no challenges connected to this dataset, we modify this data split adding a validation dataset and having a train-val-test split percentage of 60%-20%-20%.

Dataset split			
	Training	Validation	Test
Glioma	972	324	325
Meningioma	987	329	329
No Tumor	1200	400	400
Pituitary	1054	351	352
Total	**4213**	**1404**	**1406**

Examples of the images in the dataset can be seen in the following figure:

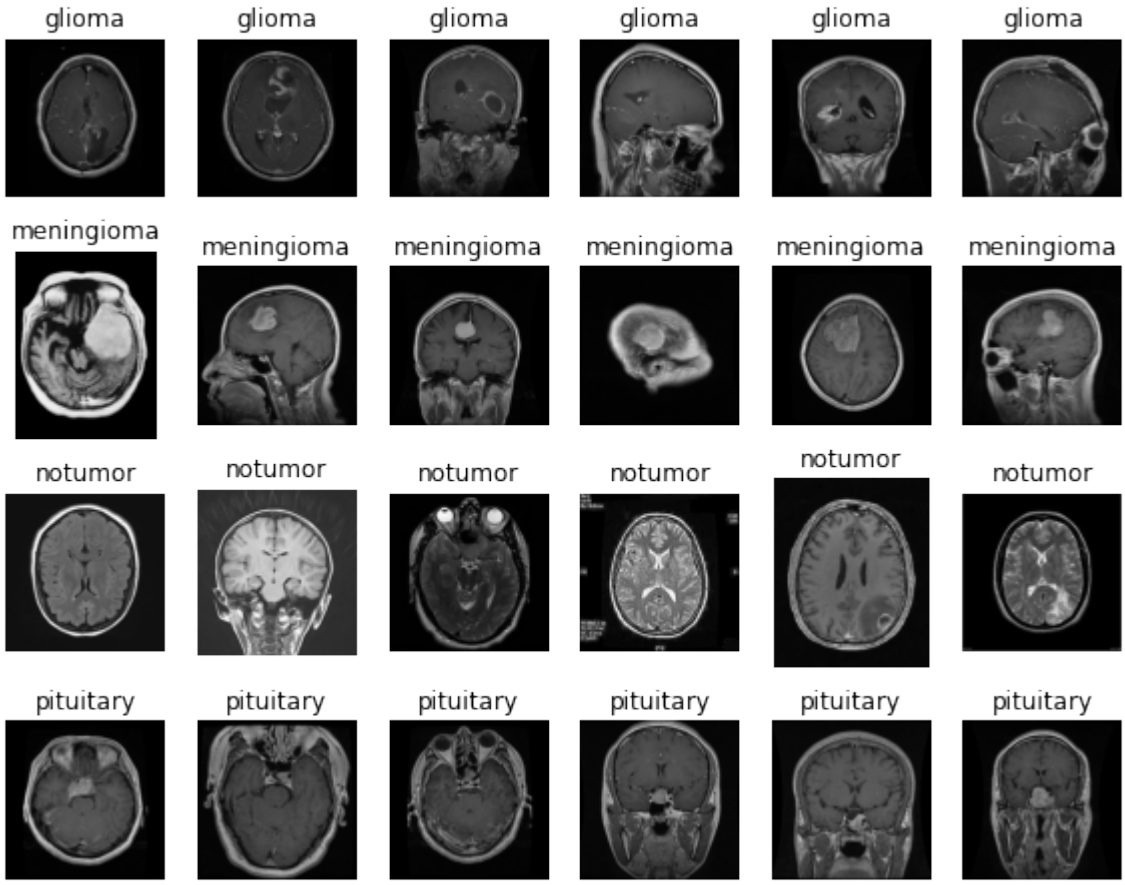


Figure 5: Some random samples taken from the dataset

The images have different sizes and are taken by different positions: we have MRI scans taken by the sides, by front or back, or from above the head. This means that the classifier we aim to build must have the capability of recognizing a brain tumor indipendently from the position where the scan was made. Also rotations and contrast of the images vary consistently from image to image.

4 Data Preprocessing

As we saw in the last chapter, the number of images for each classes is almost balanced, so we decided to not apply subsampling and data augmentation (only in this phase of preparing the dataset) in order to add or delete some images for unbalanced classes.

The MRI scans on the dataset contains images with a black background, and all the images have empty margins of black that don't contain any part of the human head. This margins could led the classification to be more dispersive, diverting the attention to the interesting part of the scans. Furthermore as we can see from the next picture, images have different dimensions.

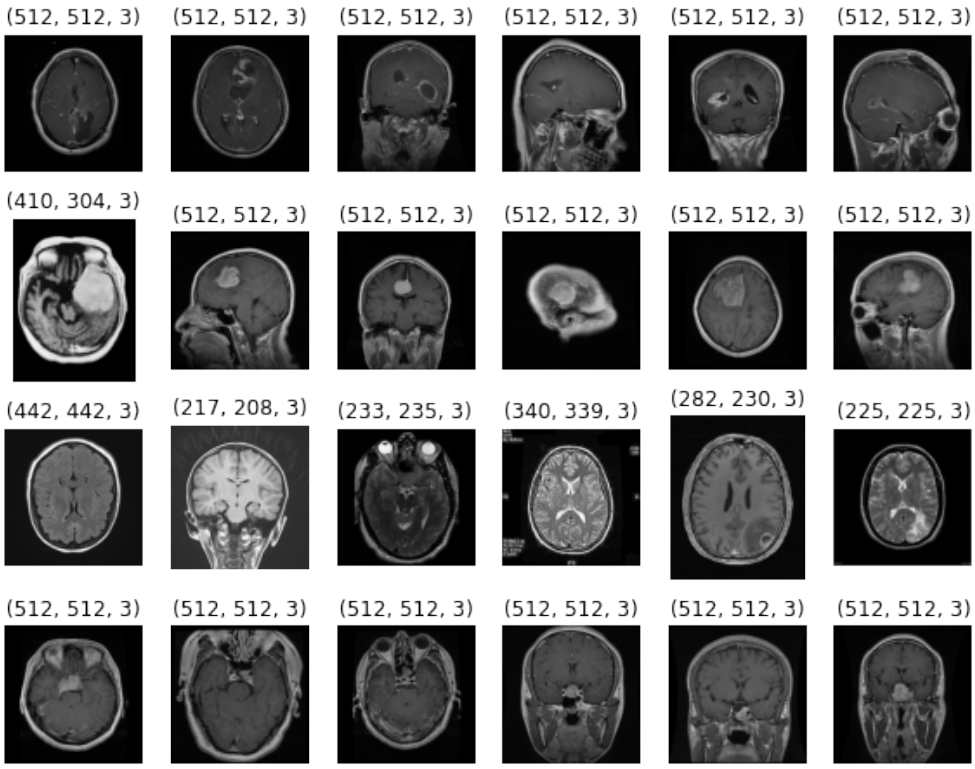


Figure 6: Shape of some images on the dataset

We can observe that despite the scans are all black and white, the channels of each image is 3.

In order to solve this two problems we cropped the black border of the scans keeping only the parts containing head parts and the resizing of the images. The new shape of the images chosen 224x224, since it is the standard dimension for all the pretrained CNN used and this will avoid to them an additional step of resizing.

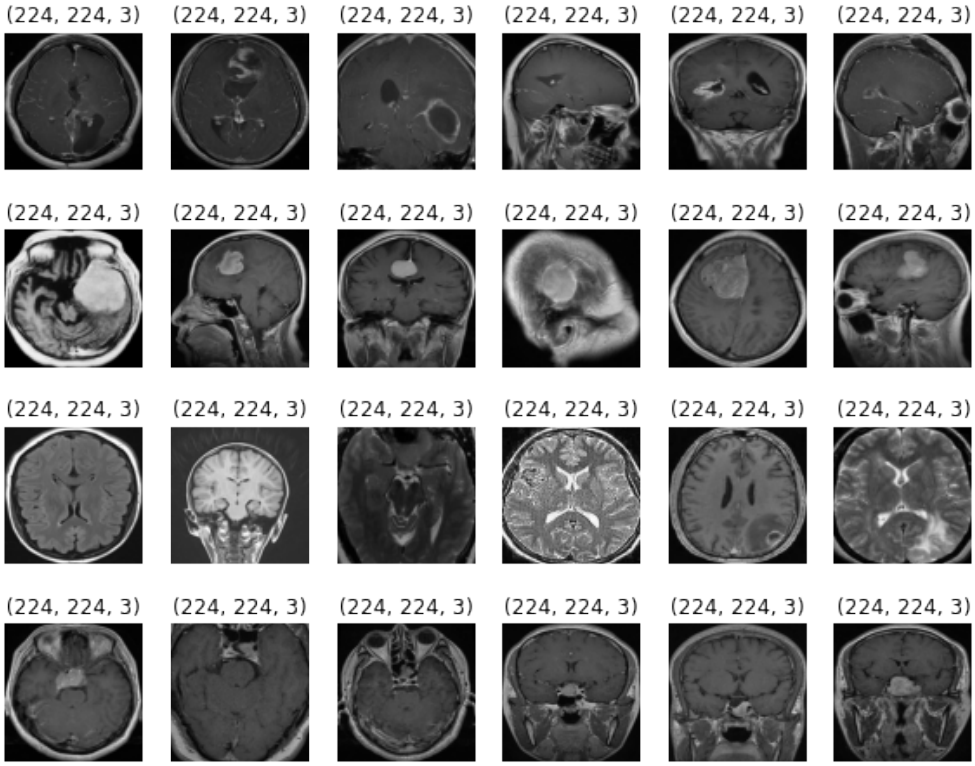


Figure 7: Preprocessed images with their shapes

5 Experiments

In order to find the model that performs better on our dataset we tried different solutions with different hyperparameters. We tried CNN built from scratch, pre-trained CNN and even Vision Transformers, a new technology that arrived in the recent years and reached the performances of the CNN. The approach used in order to find the right configurations for each type of model was the trial and error approach.

5.1 Evaluation techniques

Each model trained will be tested on our test set, and we will plot all the statistics of this evaluation aside from the Confusion Matrix. Other interesting informations to plot are accuracy and history on training and validation set after each epoch. This will let us know interesting information on the training phase.

Besides **accuracy** and **loss** on the test set, there is another particular ratio where we will focus on: the **precision on notumor** class. This index is defined as:

$$TP = Number\ of\ correctly\ classified\ as\ notumor$$

$$FP = Number\ of\ incorrectly\ classified\ as\ notumor$$

$$Precision\ on\ notumor = \frac{TP}{TP + FP}$$

The importance of this index is to be charged to the consequences to which a misclassification by our model will lead to. If the MRI scan refers to an healthy brain, but our model will classify it as any type of tumor, the patient will be supervised by doctors and additional exams will be done to him, revealing that he is in good health and he can return quickly to his normal life. A similar situation will be addressed if we misclassify the type of tumor: after other evaluation doctors will eventually found the right type of tumor and they can prescribe the right treatment for the patient. The worst situation occurs when we classify an unhealthy brain as healthy. The patient will be released without any additional exams and the brain tumor would not be treated. This could led to a growth of the tumoral mass that can cause pressure on the skull that can be life-threaning.

The precision on notumor will be fundamental in evaluate the capability of a model to not cause this type of problems, and we will focus our attention on this ratio. In particular when we will do the ensemble of our models we will try to build the best model according to this parameter.

5.2 Fighting overfitting

In order to fight the overfitting problem, we implemented two techniques that will be used in every model: **Early stopping** and **Data Augmentation**.

The **Early Stopping** technique will evaluate after each epoch of training the loss on validation set, if there is no improvement for a determinate number of epochs, the training will stop since the model is fitting too much to the training set. Then the model kept is the one obtained after the epoch with the best loss on validation set. The parameter that determine the number of epochs of no improvement before stopping the training is called patience, and in our models we will use a patience in a range from 3 to 5 depending on the model.

The Data Augmentation technique will be used in order to avoid that our model will see the same image in two different times during its training. We decided to add an input layer of Data Augmentation to each model, that will randomly flip horizontally each image and that randomly changes the contrast of the scans in a range from [-15%, +%15]. An example of application of this transformation can be seen in the picture below, where we applied Data Augmentation to the images shown in the Dataset chapter.

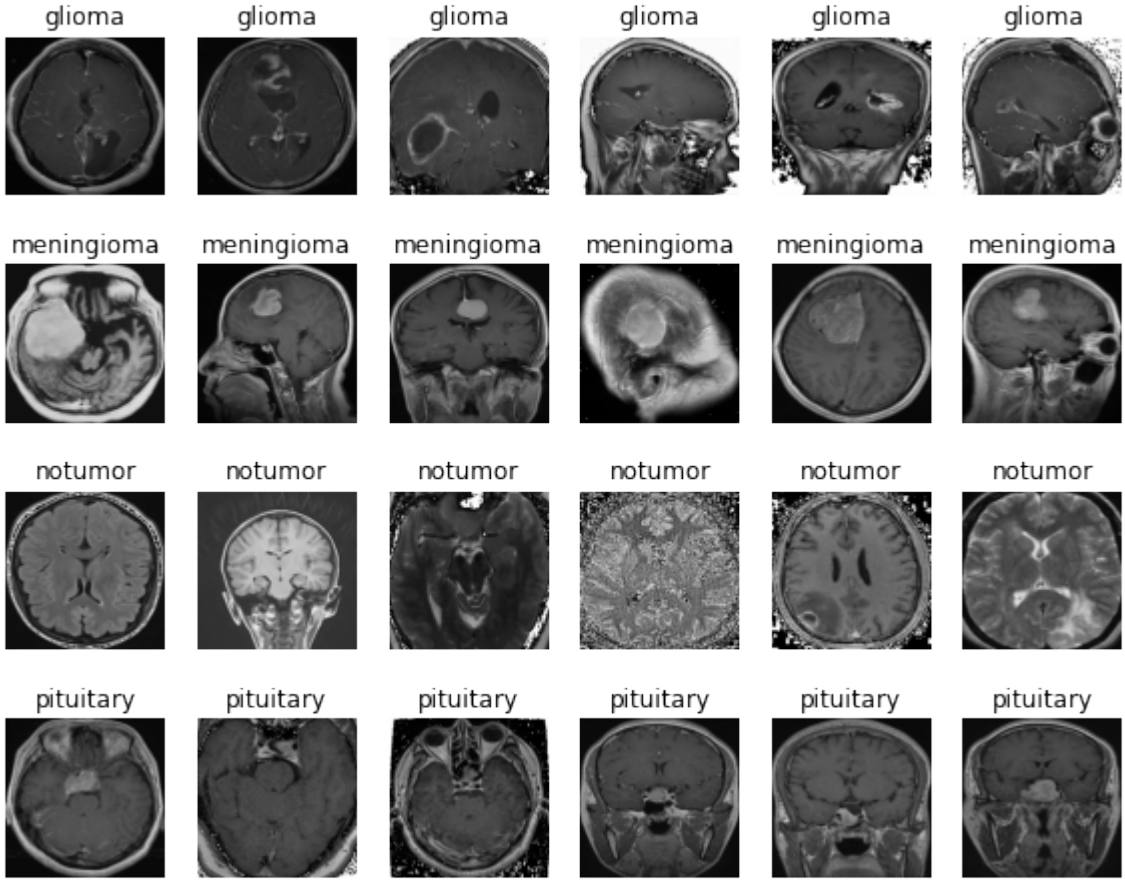


Figure 8: A random transformation to some images of the dataset

Another techniques used to fight overfitting is the **Dropout** layer. If we believe that the other two techniques are not sufficient to solve the overfitting problem, a dropout layer will be added to the model.

5.3 CNN from scratch

The first experiment was building a model using a CNN built from scratch. The structure of the CNN is a combination of Convolutional 2D layers with kernel size equal to 3 followed by a MaxPooling layer with pool_size equal to 2. The number of filters of the initial Convolutional layer is 32, and as the network approaches to the output the number of filters duplicate. So after the Data Augmentation layer we have a Conv2D layer with 32 filters and a kernel size of 3 followed by a MaxPooling with pool_size of 2, then there is a Conv2D with 64 filters and so on since we have a Conv2D with 256 filters followed by the last MaxPooling (we decided to change the pool size of the last layer to 5). This is our Base Network and in the next chapters we tried different experiments trying to add different networks on top of this base.

5.3.1 One Dense layer and a Dropout

The first trial was building on top of the base CNN a Dense layer with 256 neurons

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
rescaling (Rescaling)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
conv2d_3 (Conv2D)	(None, 24, 24, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 256)	0
flatten (Flatten)	(None, 4096)	0
hidden_classifier (Dense)	(None, 256)	1048832
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 4)	1028

Total params: 1,438,276
Trainable params: 1,438,276
Non-trainable params: 0

Figure 9: Summary of the model

The results of the training are shown below.

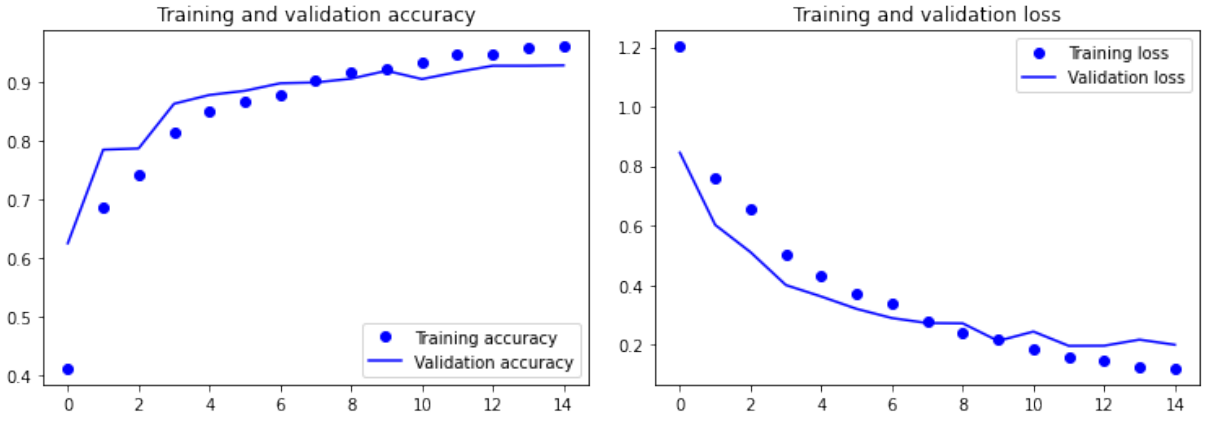


Figure 10: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.21739162504673004
Accuracy on test set: 0.9139402508735657
              precision    recall  f1-score   support

      glioma     0.9055    0.9138    0.9096       325
  meningioma     0.8136    0.8754    0.8433       329
     notumor     0.9585    0.9825    0.9704       400
   pituitary     0.9777    0.8722    0.9219       352

    accuracy                         0.9139      1406
   macro avg     0.9138    0.9110    0.9113      1406
weighted avg     0.9171    0.9139    0.9145      1406
```

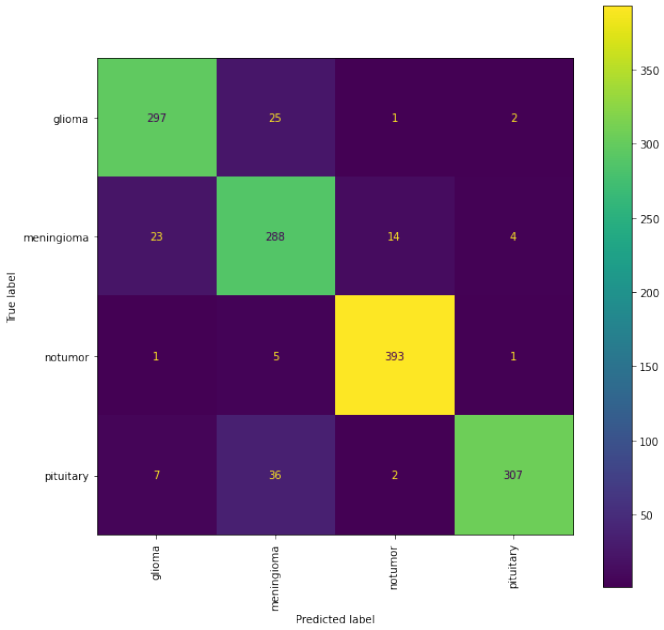


Figure 11: Evaluation of the model against the test set

We obtained a loss on the test of 0.2174 and an accuracy of 0.9139 and since this was our first trial we didn't know if this results were good or not. We tried initially to decrease the capacity of the network changing the number of neurons in the dense hidden layer to 128 neurons, and after that we tried the opposite duplicating the number of neurons to 512. The results of this two trials are shown below.

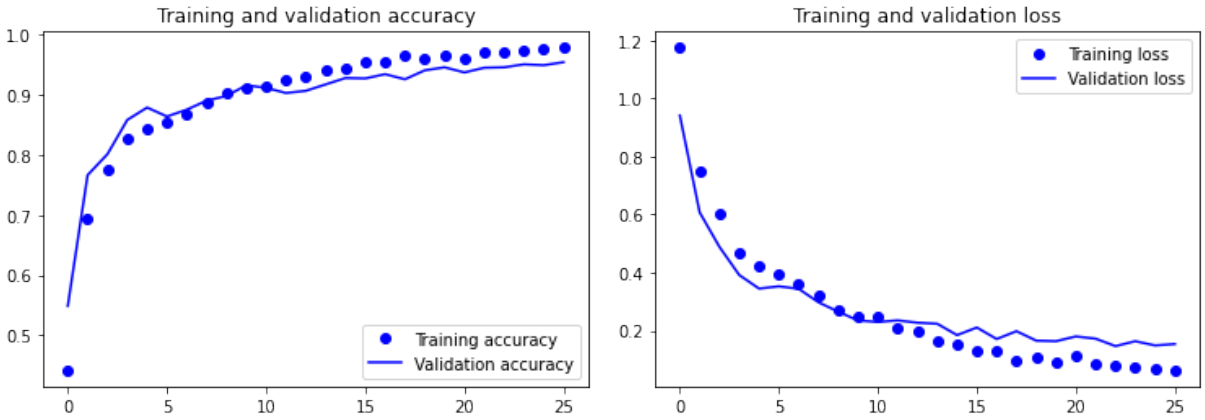


Figure 12: Accuracy and **Loss** on the training and validation set of the 128 Dense Neurons model

```
Loss on test set: 0.16589902341365814
Accuracy on test set: 0.9466571807861328
              precision    recall  f1-score   support

      glioma     0.9132    0.9385    0.9256       325
  meningioma     0.8985    0.9149    0.9066       329
     notumor     0.9825    0.9850    0.9838       400
   pituitary     0.9851    0.9403    0.9622       352

    accuracy                         0.9467      1406
   macro avg     0.9448    0.9447    0.9446      1406
weighted avg     0.9475    0.9467    0.9469      1406
```

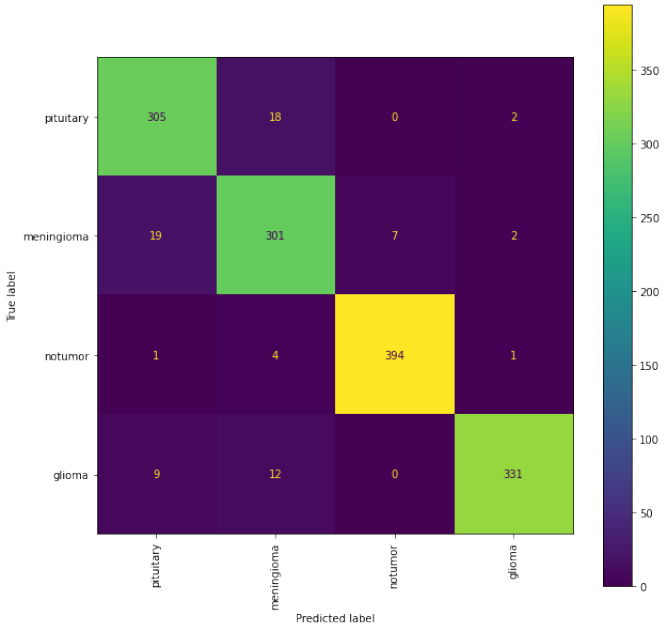


Figure 13: Evaluation and **Confusion Matrix** of the 128 Dense Neurons

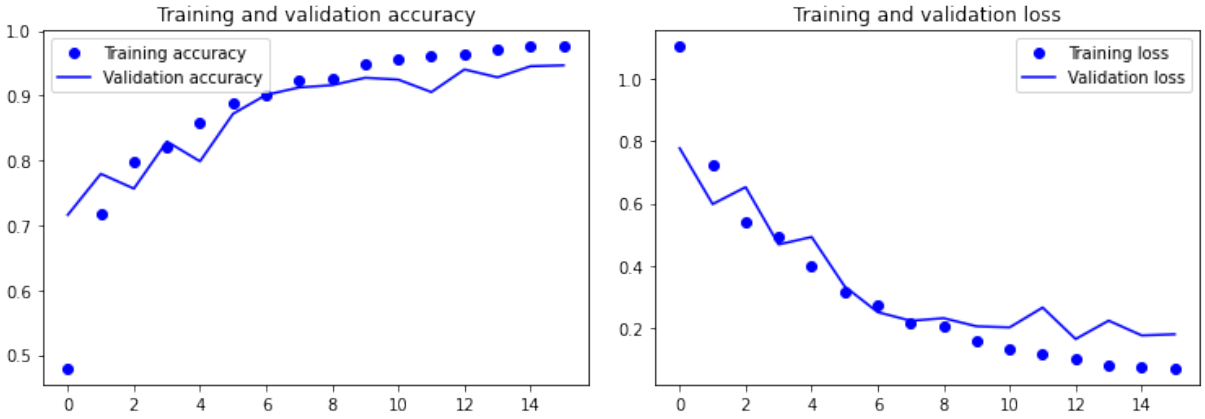


Figure 14: Accuracy and **Loss** on the training and validation set of the 512 Dense Neurons model

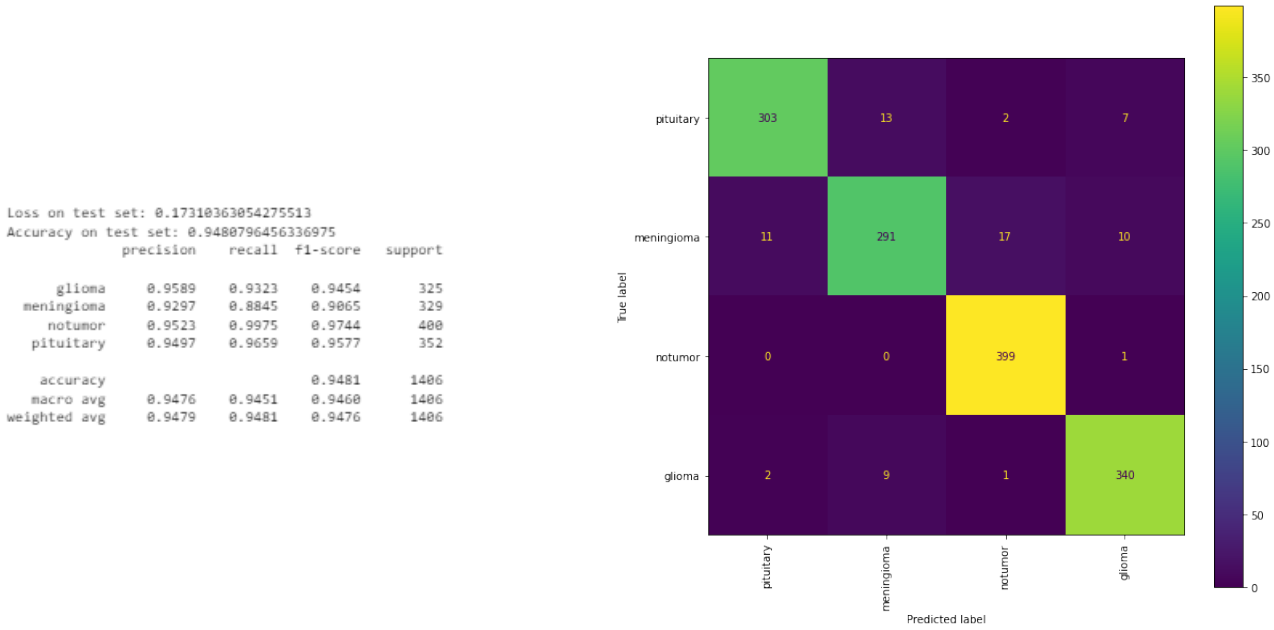


Figure 15: Evaluation of the 512 Dense Neurons model

We can see that both the models perform better. In particular the simpler model shows a precision on tumor of 0.9825 that is much more better than the others. The complex model has the better score in the accuracy value, and from the history of the training we can see a slight sign of overfitting in near the end of the training, with the curves of accuracy and loss on the validation set deviating from the training one. In the hope of solving this little problem we tried to increase the value of dropout to 0.5, and this configuration shows a reduction of this phenomenon but the performances of the model decreased.

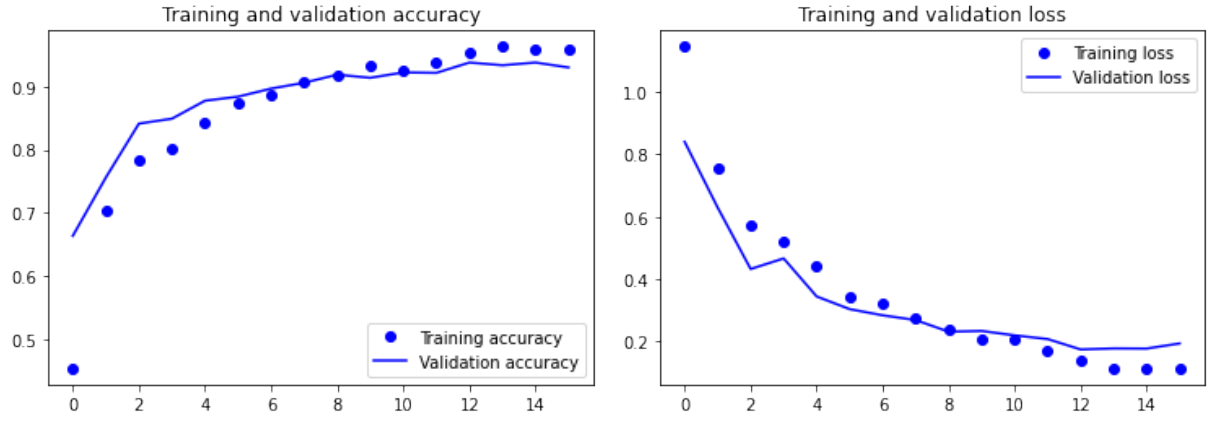


Figure 16: Accuracy and **Loss** on the training and validation set of the 512 Dense Neurons model with 0.5 Dropout

```
Loss on test set: 0.19032692909240723
Accuracy on test set: 0.933854877948761
              precision    recall  f1-score   support

      glioma     0.9631    0.8831    0.9213       325
  meningioma     0.8612    0.9240    0.8915       329
     notumor     0.9496    0.9900    0.9694       400
   pituitary     0.9645    0.9261    0.9449       352

    accuracy                         0.9339      1406
   macro avg     0.9346    0.9308    0.9318      1406
weighted avg     0.9358    0.9339    0.9339      1406
```

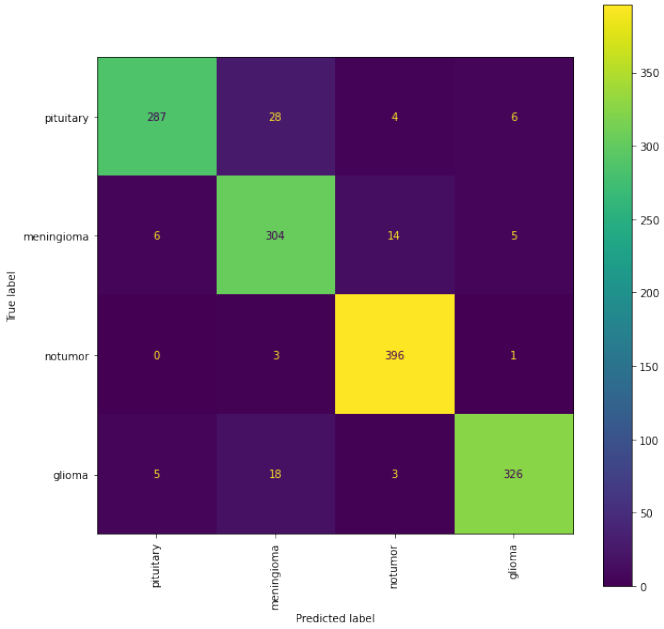


Figure 17: Evaluation and **Confusion Matrix** of the 512 Dense Neurons with 0.5 Dropout

5.3.2 One Convolutional layer

The next trial was adding a Convolution layer with 256 filters without any Max Pooling layer before the 4 classification neurons.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 224, 224, 3)]     0

sequential_6 (Sequential)    (None, 224, 224, 3)       0

rescaling_6 (Rescaling)      (None, 224, 224, 3)       0

conv2d_24 (Conv2D)           (None, 222, 222, 32)      896

max_pooling2d_24 (MaxPoolin  (None, 111, 111, 32)      0
g2D)

conv2d_25 (Conv2D)           (None, 109, 109, 64)      18496

max_pooling2d_25 (MaxPoolin  (None, 54, 54, 64)        0
g2D)

conv2d_26 (Conv2D)           (None, 52, 52, 128)       73856

max_pooling2d_26 (MaxPoolin  (None, 26, 26, 128)       0
g2D)

conv2d_27 (Conv2D)           (None, 24, 24, 256)       295168

max_pooling2d_27 (MaxPoolin  (None, 8, 8, 256)         0
g2D)

conv2d_28 (Conv2D)           (None, 6, 6, 256)         590080

flatten_6 (Flatten)          (None, 9216)              0

dropout_6 (Dropout)          (None, 9216)              0

dense_6 (Dense)              (None, 4)                 36868

=================================================================
Total params: 1,015,364
Trainable params: 1,015,364
Non-trainable params: 0
```

Figure 18: Summary of the model

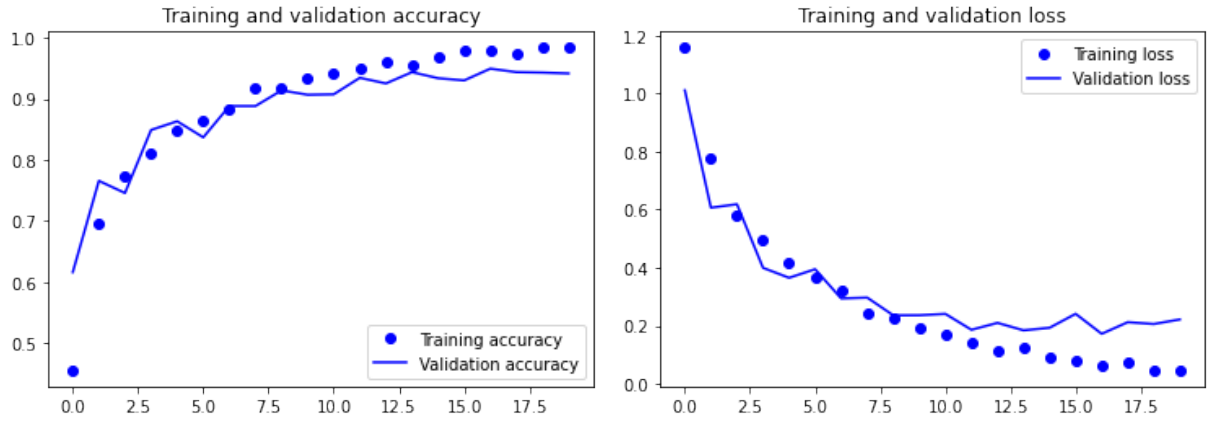
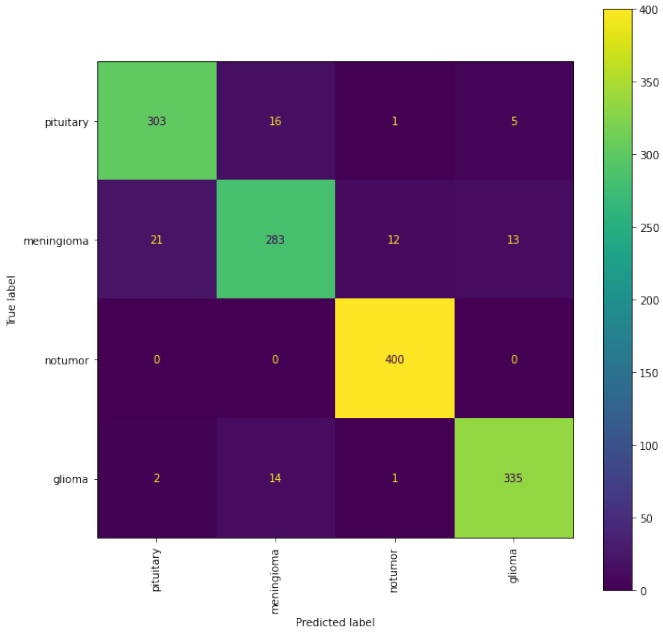


Figure 19: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.18747489154338837
Accuracy on test set: 0.9395447969436646
              precision    recall  f1-score   support

      glioma     0.9294    0.9323    0.9309       325
  meningioma     0.9042    0.8602    0.8816       329
     notumor     0.9662    1.0000    0.9828       400
   pituitary     0.9490    0.9517    0.9504       352

    accuracy                         0.9395      1406
   macro avg     0.9372    0.9360    0.9364      1406
weighted avg     0.9389    0.9395    0.9390      1406
```



Since a discrete degree of overfitting It has been experienced, we tried another approach to manage the dropout: instead of increasing the dropout rate which seemed not productive before, we tried to add an additional dropout layer (with 0.3 rate) before the last convolutional layer, and this approach revealed better performances:

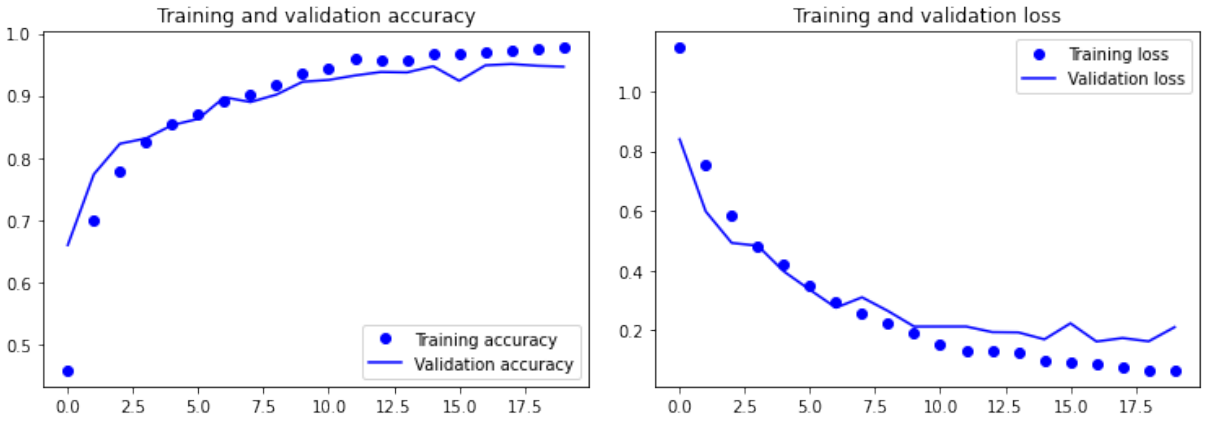
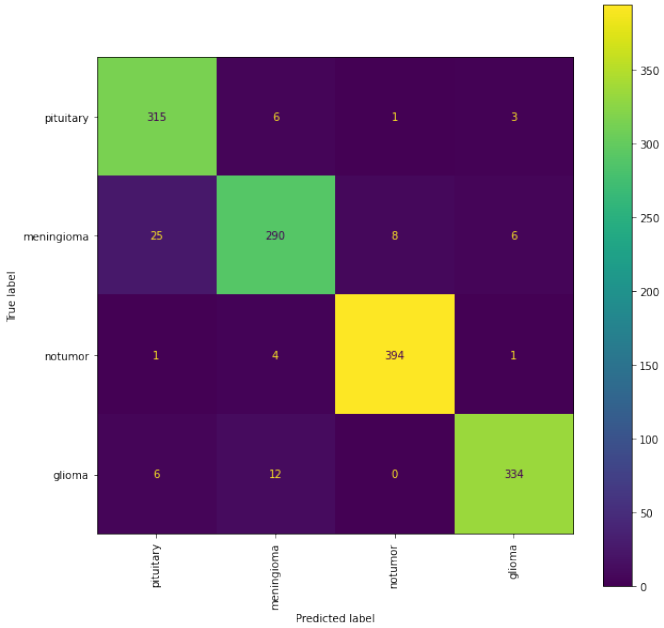


Figure 21: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.161067932844162
Accuracy on test set: 0.9480796456336975
              precision    recall  f1-score   support

      glioma     0.9078    0.9692    0.9375       325
  meningioma     0.9295    0.8815    0.9048       329
     notumor     0.9777    0.9850    0.9813       400
   pituitary     0.9709    0.9489    0.9598       352

    accuracy                         0.9481      1406
   macro avg     0.9465    0.9461    0.9459      1406
weighted avg     0.9486    0.9481    0.9479      1406
```



5.3.3 More Complex models

After this trials we tried to complicate even more the model, increasing the base model depth Conv2D layer with 512 filters and keeping the two dropout layers which seemed to perform well. We tried to add on top different structures: Convolutional layer with 512 filters, a Dense layer with 512 neurons and even one with two Dense layers, both with 512 neurons.

All this networks achieved great performances, but the best model found was the one with tjust one Dense layer with 512 Neurons.

Layer (type)	Output Shape	Param #
input_13 (InputLayer)	[(None, 224, 224, 3)]	0
sequential_6 (Sequential)	(None, 224, 224, 3)	0
rescaling_12 (Rescaling)	(None, 224, 224, 3)	0
conv2d_57 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_52 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_58 (Conv2D)	(None, 112, 112, 64)	18496
max_pooling2d_53 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_59 (Conv2D)	(None, 56, 56, 128)	73856
max_pooling2d_54 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_60 (Conv2D)	(None, 28, 28, 256)	295168
max_pooling2d_55 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_61 (Conv2D)	(None, 9, 9, 512)	1180160
max_pooling2d_56 (MaxPooling2D)	(None, 3, 3, 512)	0
flatten_11 (Flatten)	(None, 4608)	0
dropout_16 (Dropout)	(None, 4608)	0
hidden_classifier (Dense)	(None, 512)	2359808
dropout_17 (Dropout)	(None, 512)	0
dense_10 (Dense)	(None, 4)	2052

Total params: 3,930,436
Trainable params: 3,930,436
Non-trainable params: 0

Figure 23: Summary of the model

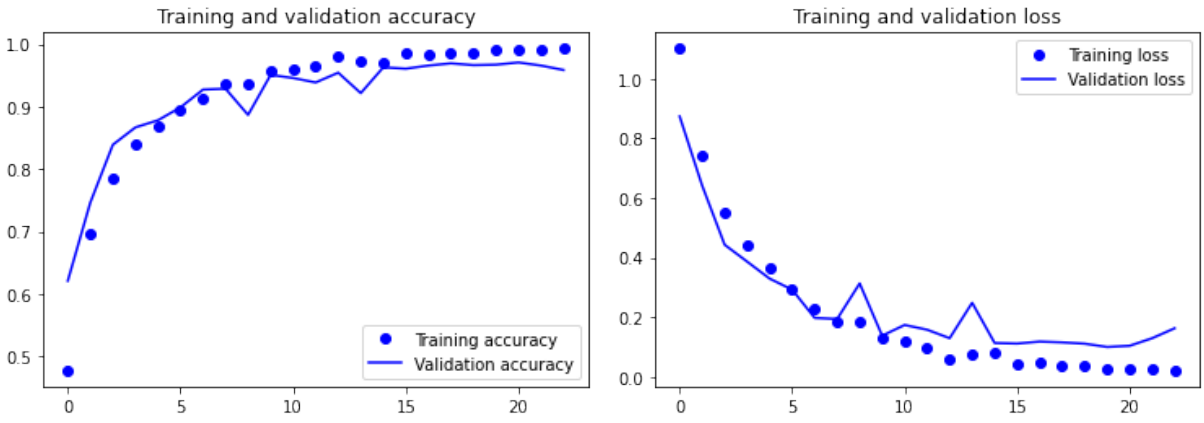
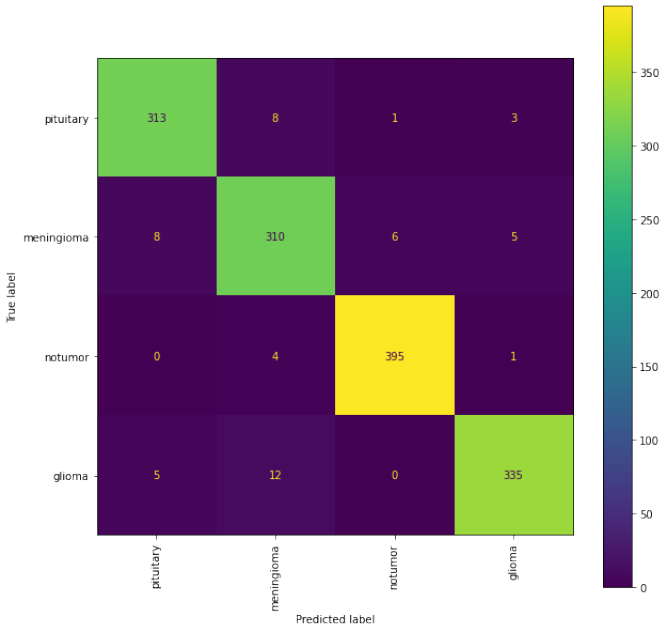


Figure 24: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.13795408606529236
Accuracy on test set: 0.962304413318634
              precision    recall  f1-score   support

      glioma     0.9601    0.9631    0.9616       325
  meningioma     0.9281    0.9422    0.9351       329
     notumor     0.9826    0.9875    0.9850       400
   pituitary     0.9738    0.9517    0.9626       352

    accuracy                         0.9623      1406
   macro avg     0.9612    0.9611    0.9611      1406
weighted avg     0.9625    0.9623    0.9623      1406
```



As we can see this model outperformed all the other models built from scratch in all the parameters we were interested in: loss, accuracy and precision on no tumor. Later on when we are gonna analyze the best models built we will use this model for reference of the CNN built from scratch.

5.4 VGG16

The first pretrained CNN used is VGG16. The structure of this CNN is shown below.

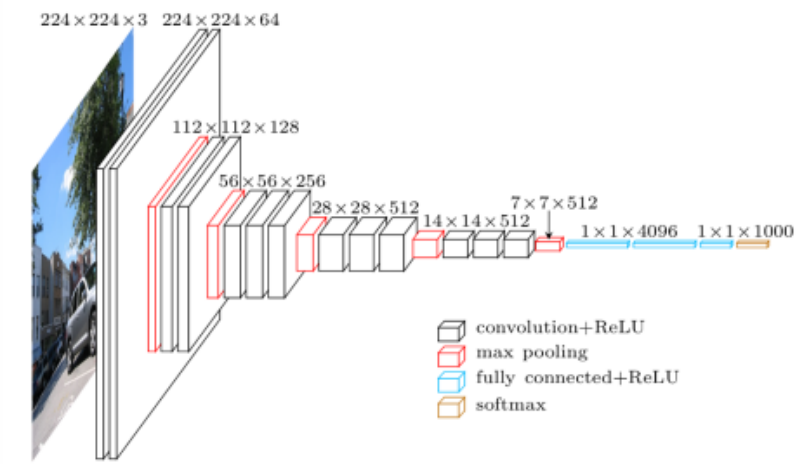


Figure 26: Structure of VGG16

The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3. The convolution stride is

fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures). All hidden layers are equipped with the rectification (ReLU) non-linearity.

5.4.1 MLP built on top training

On top of the base VGG16 model we built a MLP network and initially we performed different configurations. It is a good rule to apply a Global Average Pooling layer after every pretrained CNN, so from now on every trial with this type of networks will include this layer. The Data Augmentation layer is kept in all the following models.

The first experiment was taken in order to appreciates the importance of the Dropout layer. In fact we built on top of VGG a Dense layer with 256 neurons and we tried it initially without Dropout and later on with a Dropout with 0.5 as dropout rate.

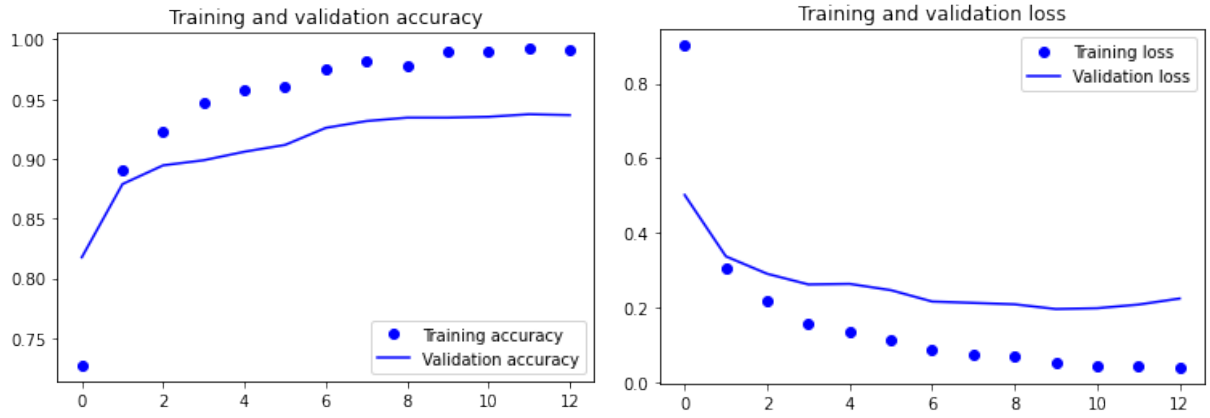


Figure 27: Accuracy and **Loss** without Dropout

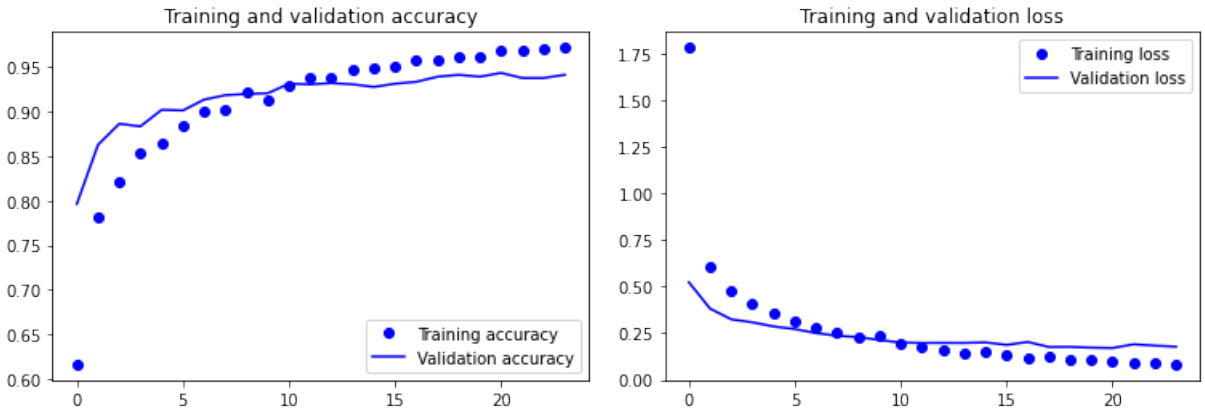


Figure 28: Accuracy and **Loss** with 0.5 Dropout

As we can see on the curves of the model without the Dropout we have an important deviation of the curves of the validation from the ones of the training, and this phenomenom is partially solved for the Dropout model.

The best performances with networks in terms of accuracy and loss on the test were achieved when we change the number of neurons in the Dense layer from 256 to 512, mantaining the same dropout rate:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 gap (GlobalAveragePooling2D  (None, 512)              0
 )

 flatten (Flatten)           (None, 512)               0

 classifier_hidden (Dense)   (None, 512)               262656

 dropout_12 (Dropout)        (None, 512)               0

 dense_12 (Dense)            (None, 4)                 2052

=================================================================
Total params: 14,979,396
Trainable params: 264,708
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 29: Summary of the model

The results of the training are shown below.

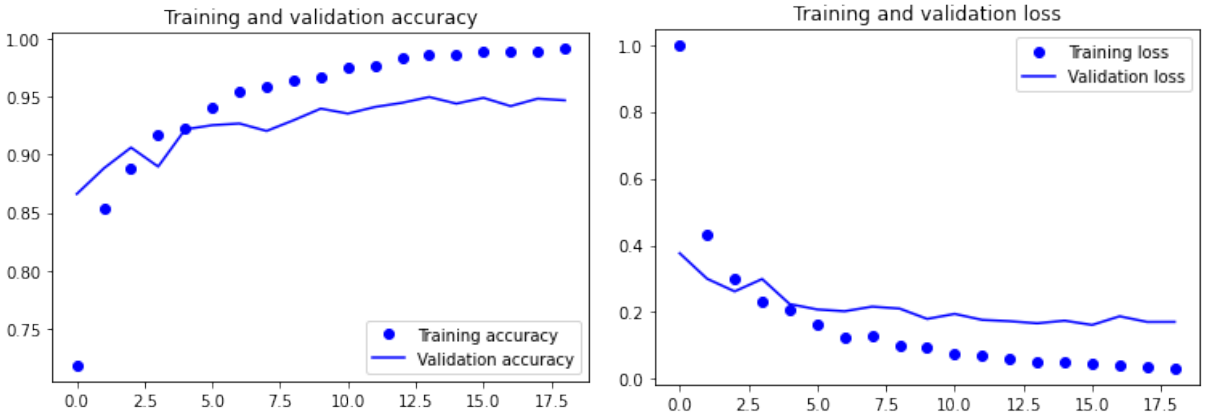
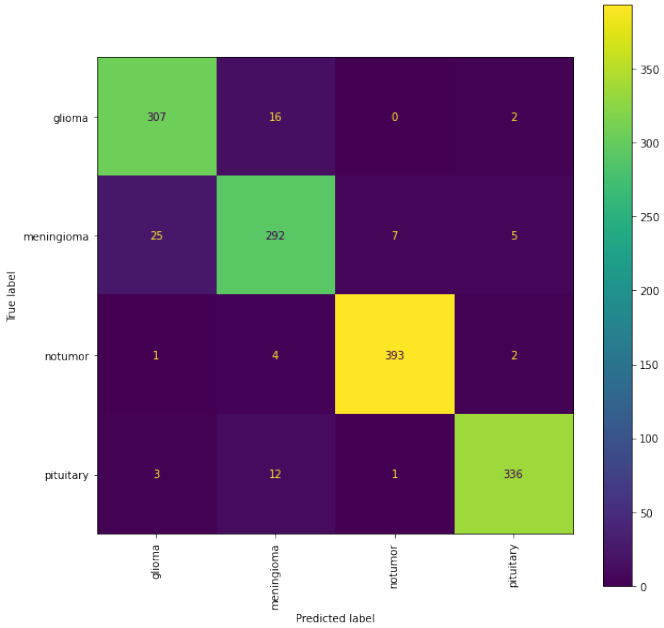


Figure 30: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.18647176027297974
Accuracy on test set: 0.9445234537124634
              precision    recall  f1-score   support

      glioma     0.9137    0.9446    0.9289       325
  meningioma     0.9012    0.8875    0.8943       329
     notumor     0.9800    0.9825    0.9813       400
   pituitary     0.9739    0.9545    0.9641       352

    accuracy                         0.9445      1406
   macro avg     0.9422    0.9423    0.9422      1406
weighted avg     0.9447    0.9445    0.9445      1406
```



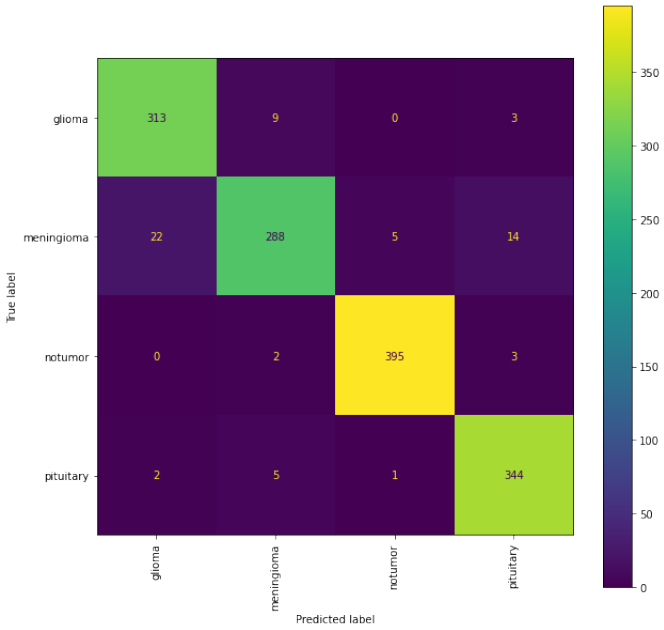
5.4.2 Model Finetuning

Now our objective was to finetune the best model obtained in the last chapter. We started to unfreeze only the last layer of VGG16 and reperforms the training adjusting the weights of this layer. The results of this training were fine and the model increased its performances:

```
Loss on test set: 0.18366014957427979
Accuracy on test set: 0.9530583024024963
              precision    recall  f1-score   support

      glioma     0.9288    0.9631    0.9456       325
  meningioma     0.9474    0.8754    0.9100       329
     notumor     0.9850    0.9875    0.9863       400
   pituitary     0.9451    0.9773    0.9609       352

    accuracy                         0.9531      1406
   macro avg     0.9516    0.9508    0.9507      1406
weighted avg     0.9532    0.9531    0.9527      1406
```



Considering this results we unfroze also the second last layer, with the goal of progressively fine tune the whole last block. Unfortunately in this phase a problem occured during the training:

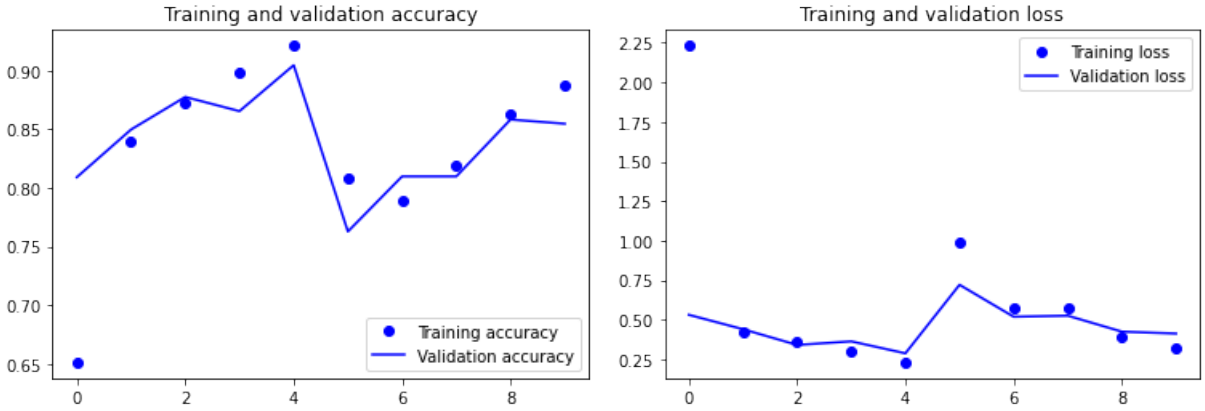


Figure 33: Accuracy and **Loss** on the training and validation set

We can observe from the training history a strange behaviour, the accuracy on both training and validation set dropped after the fifth epoch, leading the model to achieve very disappointing performances. We may suppose that our network get knocked into a bad part of parameter space corresponding to a sudden decrease in accuracy. In general, lowering the learning rate is a good approach to this kind of problem. Reperforming the training with a 0.0002 learning rate gave use in fact good results and the problem was solved.

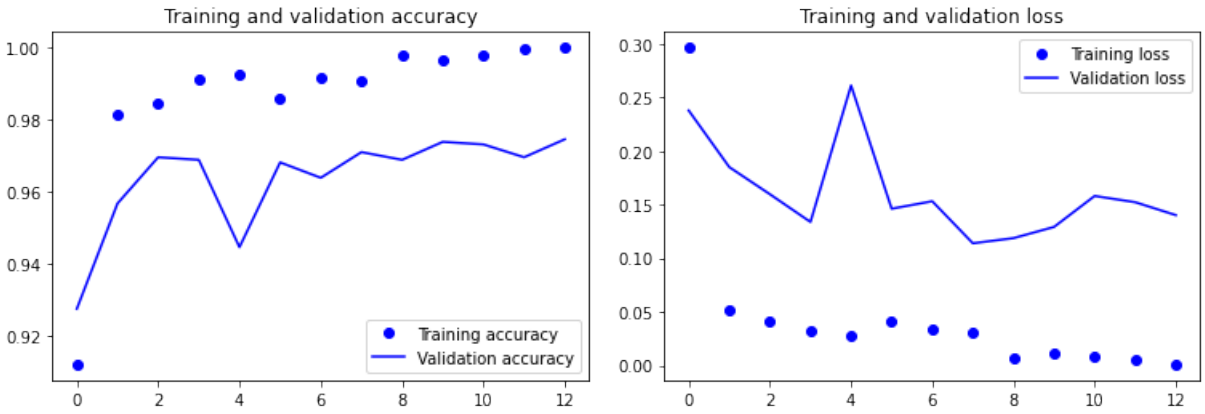
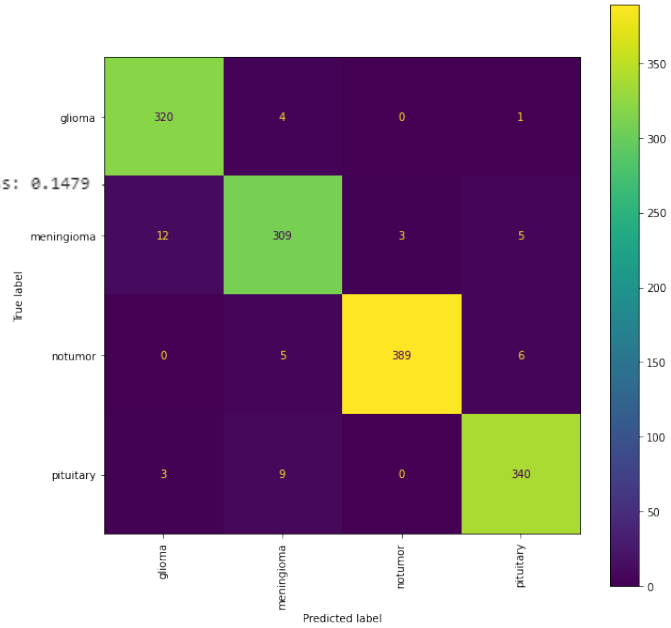


Figure 34: Accuracy and **Loss** on the training and validation set

```
22/22 [==============================] - 5s 228ms/step - loss: 0.1479
Loss on test set: 0.14785729348659515
Accuracy on test set: 0.9658606052398682
              precision    recall  f1-score   support

      glioma     0.9552    0.9846    0.9697       325
  meningioma     0.9450    0.9392    0.9421       329
     notumor     0.9923    0.9725    0.9823       400
   pituitary     0.9659    0.9659    0.9659       352

    accuracy                         0.9659      1406
   macro avg     0.9646    0.9656    0.9650      1406
weighted avg     0.9661    0.9659    0.9659      1406
```



At the end we unfroze the whole last block of the VGG16 and performs the training.

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.__operators__.getitem_3 (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add_3 (TFOpLambda)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
gap (GlobalAveragePooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
classifier_hidden (Dense)	(None, 512)	262656
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 4)	2052

Total params: 14,979,396
Trainable params: 7,344,132
Non-trainable params: 7,635,264

Figure 36: Summary of the model. The trainable parameters incremented from 264,708 to **7,344,132**

We also reload the model trained with all VGG16 and unfroze the last block in order to compare the performances from the Finetuned progressively model and the non progressively one. The results of this last two training are compared below:

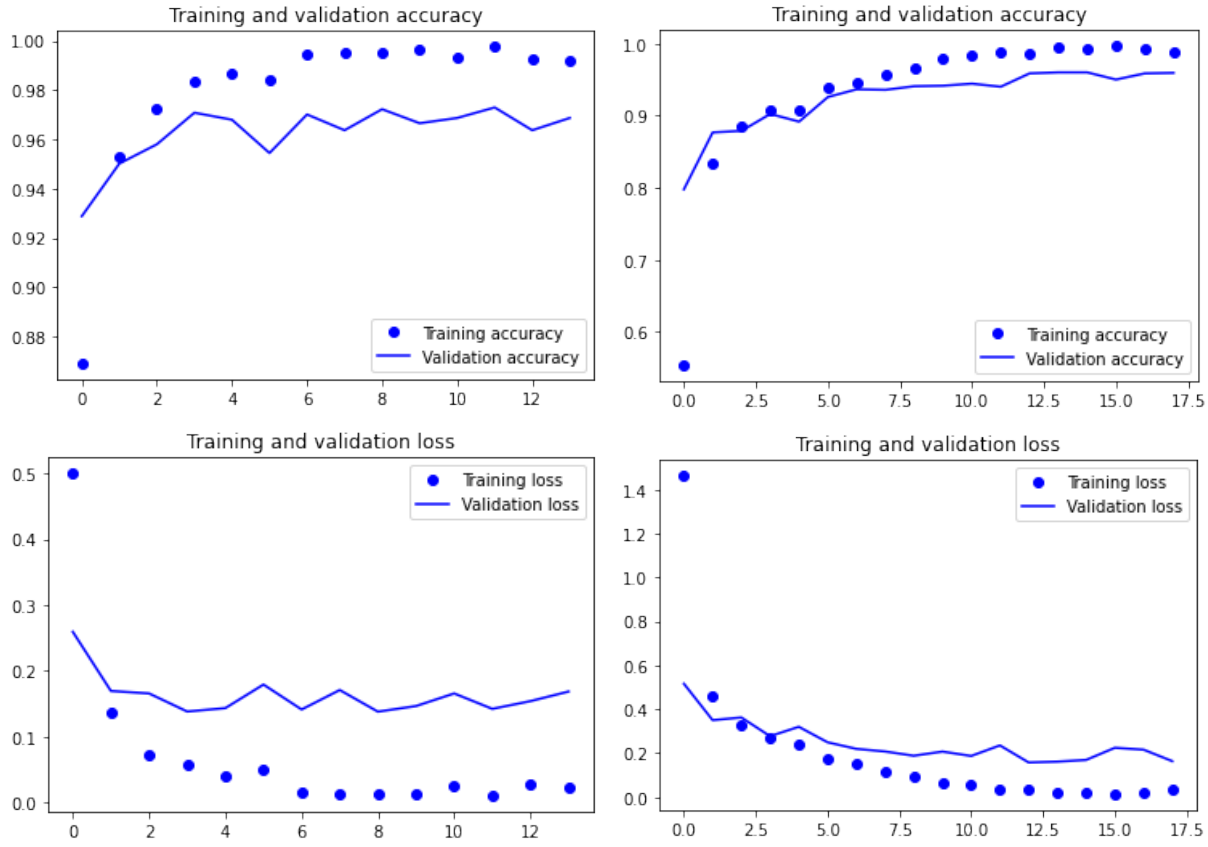


Figure 37: Accuracy and **Loss** on the training and validation set (on left the Finetuned progressively model)

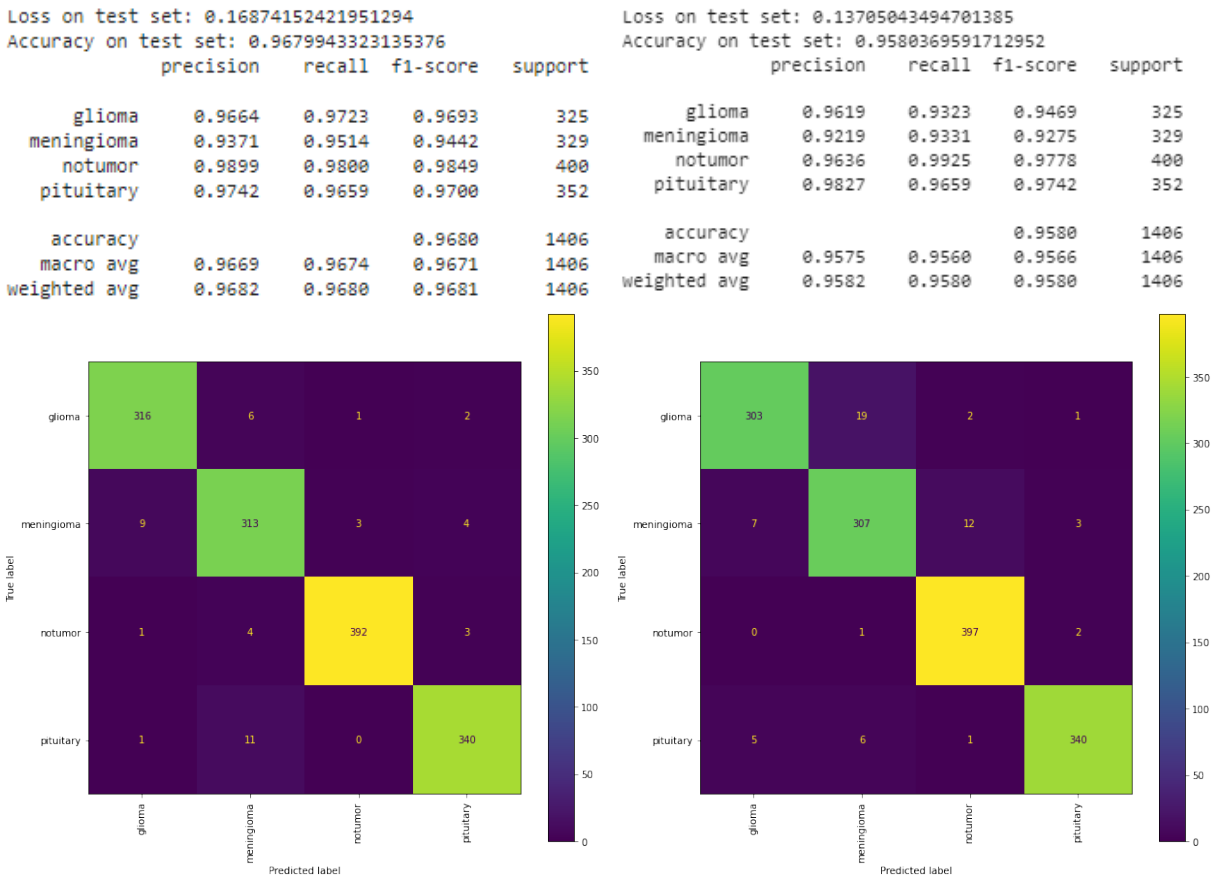


Figure 38: Evaluation against the test set (on left the Finetuned progressively model)

We have some interesting results from this two different trainings. As we could expect the training of the progressive model took more epochs than the second model, since a lot of weights of the last were trained before and they needed less adjustments. From the history of the training we can observe how the second model overfitted less than the first one and on the evaluation phase reached a better results on accuracy. On the other hand The first model have the best accuracy found until now and has a precision on no tumor of 0.99.

The interesting thing is how the two models have completely different Confusion Matrix, demonstrating how different are their prediction. These differences could be useful if we want to use these two models for ensembling, apporting to the ensembled model two very different criteria of vote.

5.5 Resnet-50

ResNet-50 is a convolutional neural network that is 50 layers deep. ResNet, short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with more than 150 layers.

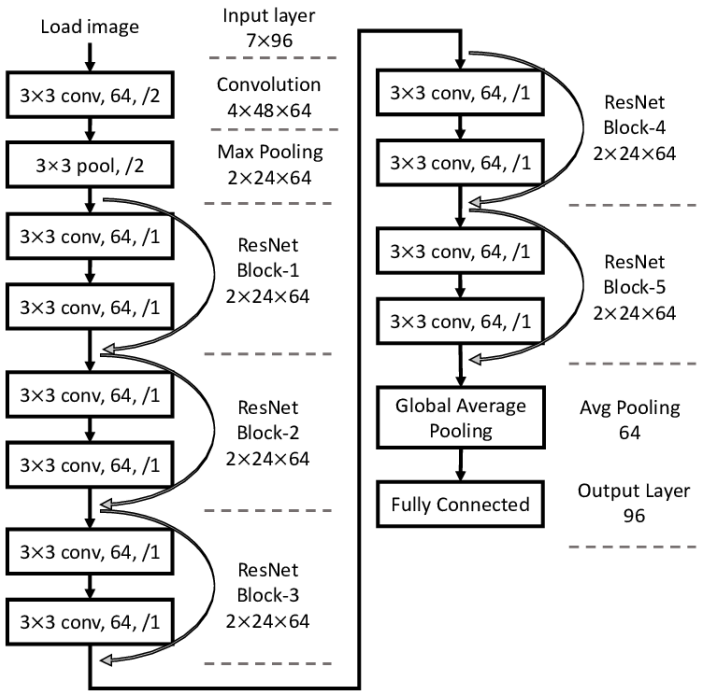


Figure 39: Structure of ResNet

ResNet leverage on Skip Connections (or Shortcut Connections), as the name suggests skips some of the layers in the neural network and feeds the output of one layer as the input to the next layers. Skip Connections help to solve the Vanishing Gradient Problem, and with them ResNet become one of the most used pretrained CNNs.

5.5.1 MLP built on top training

On top of Resnet-50 we decided to build a MLP network with a Dense 256 Neurons and a Dropout layer with 0.5 as dropout rate.

Layer (type)	Output Shape	Param #
input_12 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.__operators__.getitem_5 (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add_5 (TFOpLambda)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
gap (GlobalAveragePooling2D)	(None, 2048)	0
flatten (Flatten)	(None, 2048)	0
classifier_hidden (Dense)	(None, 256)	524544
dropout_5 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 4)	1028

Total params: 24,113,284
Trainable params: 525,572

Figure 40: Summary of the model

The results of the training are shown below.

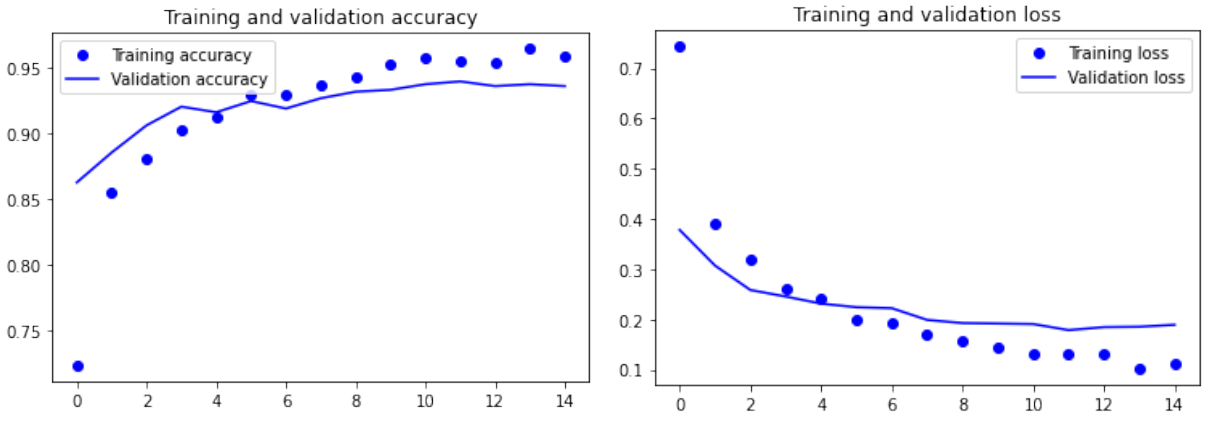


Figure 41: Accuracy and **Loss** on the training and validation set

```
Loss on test set: 0.20069462060928345
Accuracy on test set: 0.9281650185585022
              precision    recall  f1-score   support

      glioma     0.9299    0.8985    0.9139       325
  meningioma     0.8347    0.9058    0.8688       329
     notumor     0.9776    0.9800    0.9788       400
   pituitary     0.9671    0.9176    0.9417       352

    accuracy                         0.9282      1406
   macro avg     0.9273    0.9255    0.9258      1406
weighted avg     0.9305    0.9282    0.9288      1406
```

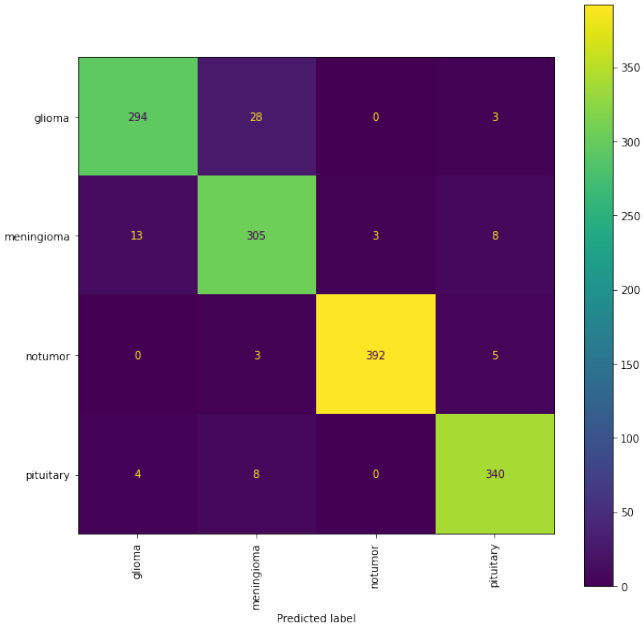


Figure 42: Evaluation of the model against the test set

5.5.2 Model Finetuning

The previous model was then finetuned with the last block unfreezed, and the results obtained are the following:

```
Model: "ResNet50"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 224, 224, 3)]     0

 sequential (Sequential)     (None, 224, 224, 3)       0

 tf.__operators__.getitem_5   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_5 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 gap (GlobalAveragePooling2D  (None, 2048)             0
 )

 flatten (Flatten)           (None, 2048)              0

 classifier_hidden (Dense)   (None, 256)               524544

 dropout_5 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 4)                 1028

=================================================================
Total params: 24,113,284
Trainable params: 4,985,092
Non-trainable params: 19,128,192
_________________________________________________________________
```

Figure 43: Summary of the model, the trainable parameters increased from 525,572 to **4,985,092**

The results of the training are shown below.

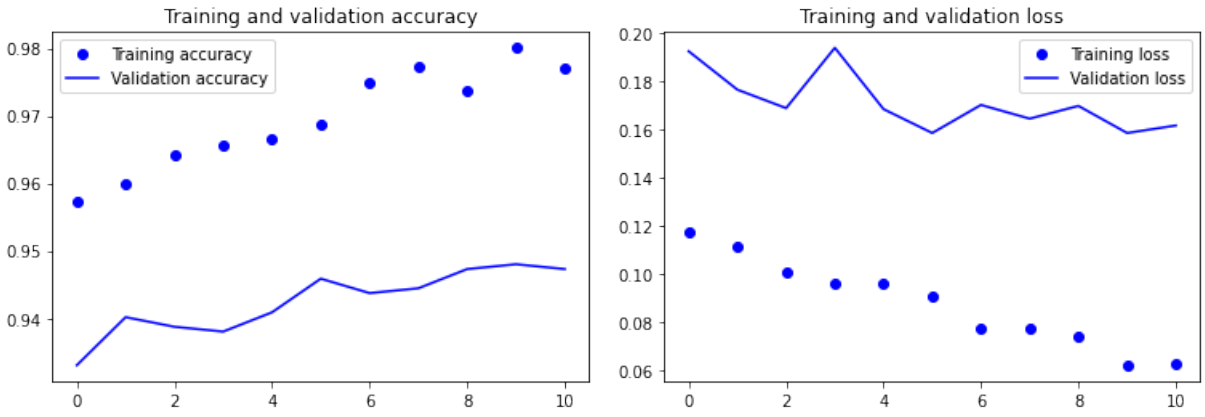


Figure 44: Accuracy and **Loss** on the training and validation set

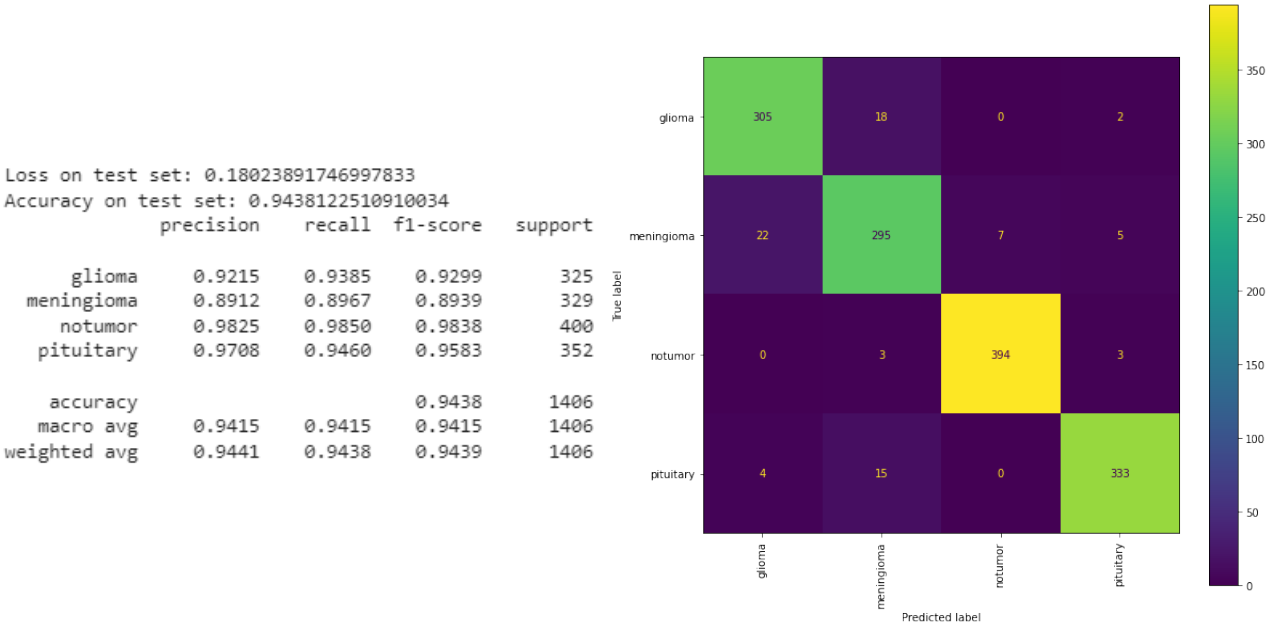


Figure 45: Evaluation of the model against the test set

5.5.3 Chopping the last block of Resnet-50

All the pretrained CNN network utilized are trained on ImageNet, which is a dataset very different from ours composed by brain MRI scans. We may hypothesize that the Convolution filters of the last layers of these CNN are prepared to recognise very specific characteristics of the images, while the first filters are used to extract low level features, that are less specific and can relates better to our dataset.

We tried to chop the last block of Resnet-50 before connecting the base CNN to our MLP network and to train this MLP network on our dataset with the base

freezed. Than we performed the fine tuning with the last block unfreezed.

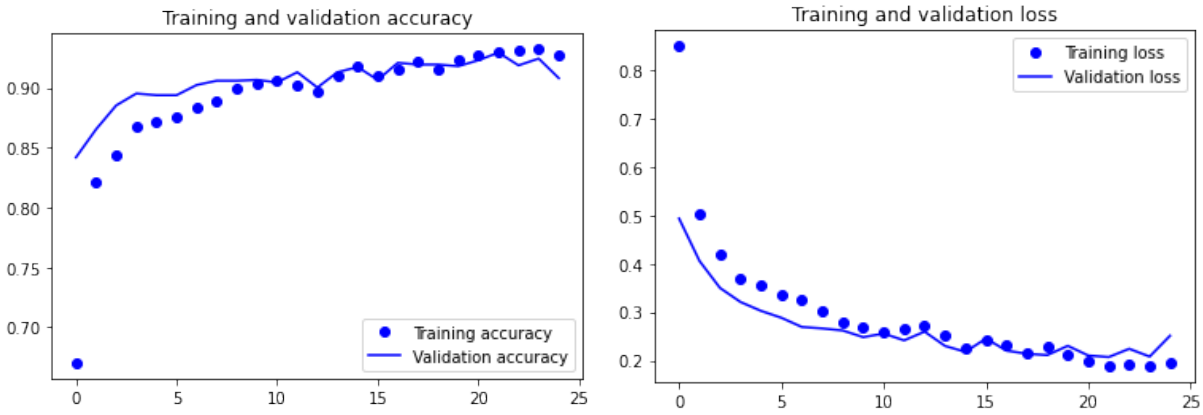


Figure 46: History of the training of the freezed chopped model

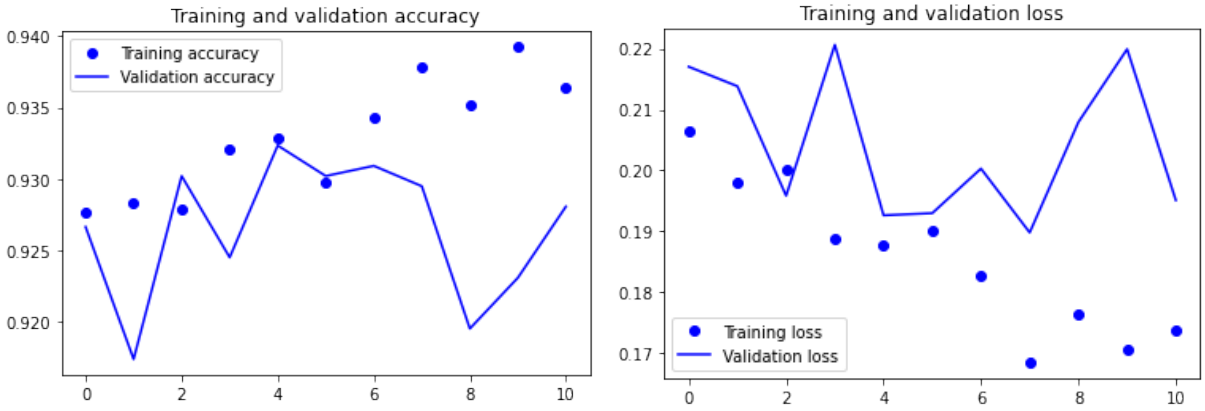


Figure 47: History of the finetuning of the model

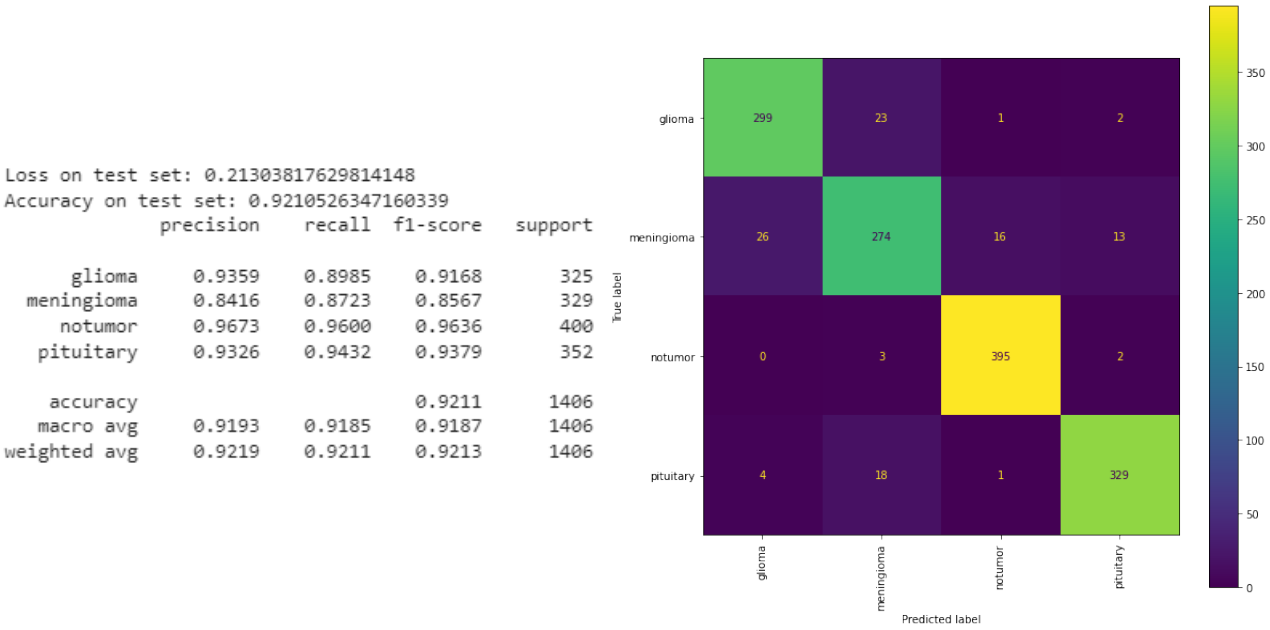


Figure 48: Evaluation of the model against the test set

As we can see from this results this models did not achieved comparable results to the complete model seen in the last chapter.

5.6 Densenet121

The Convolutional Neural Network used as base network in the following models is DenseNet. DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. We can say that each layer is receiving a “collective knowledge” from all preceding layers, unlike the traditional CNN where each layer receives informations only from the precedent layer.

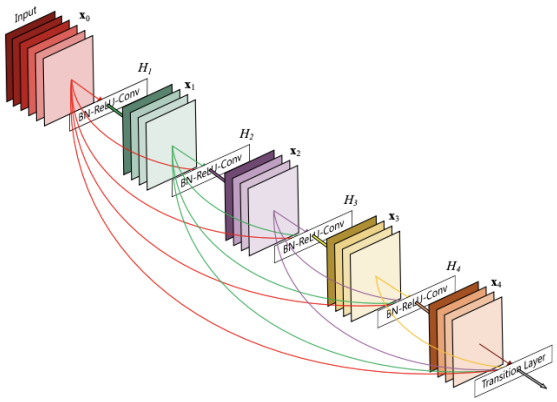


Figure 49: Structure of DenseNet

5.6.1 MLP built on top training

As we have done for Resnet, we built on top of the base CNN a Dense 256 Neurons network with a Dropout layer.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

sequential (Sequential)      (None, 224, 224, 3)       0

tf.math.truediv_2 (TFOpLamb  (None, 224, 224, 3)       0
da)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)       0
a)

tf.math.truediv_3 (TFOpLamb  (None, 224, 224, 3)       0
da)

densenet121 (Functional)     (None, 7, 7, 1024)        7037504

gap (GlobalAveragePooling2D  (None, 1024)              0
)

flatten (Flatten)            (None, 1024)              0

classifier_hidden (Dense)    (None, 256)               262400

dropout_1 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 4)                 1028

=================================================================
Total params: 7,300,932
Trainable params: 263,428
Non-trainable params: 7,037,504
_________________________________________________________________
```

Figure 50: Summary of the model

The results of the training are shown below.

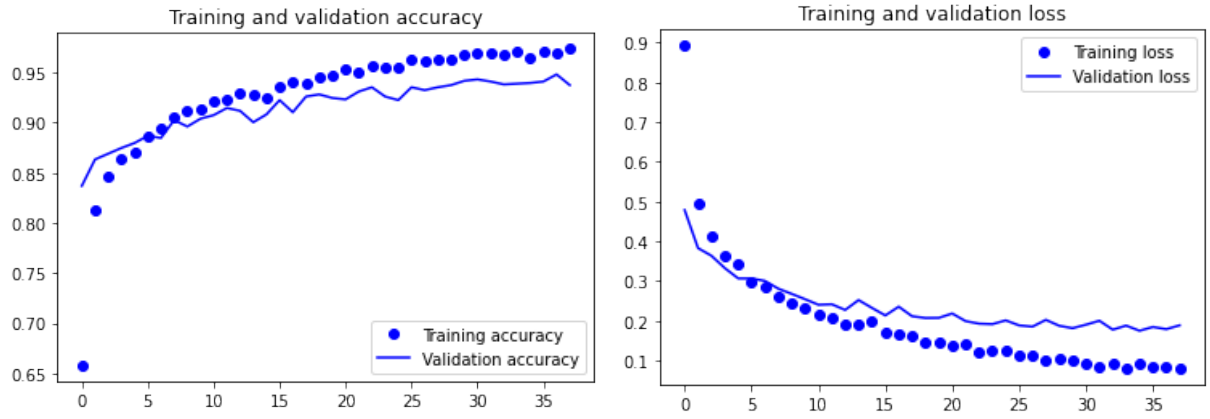
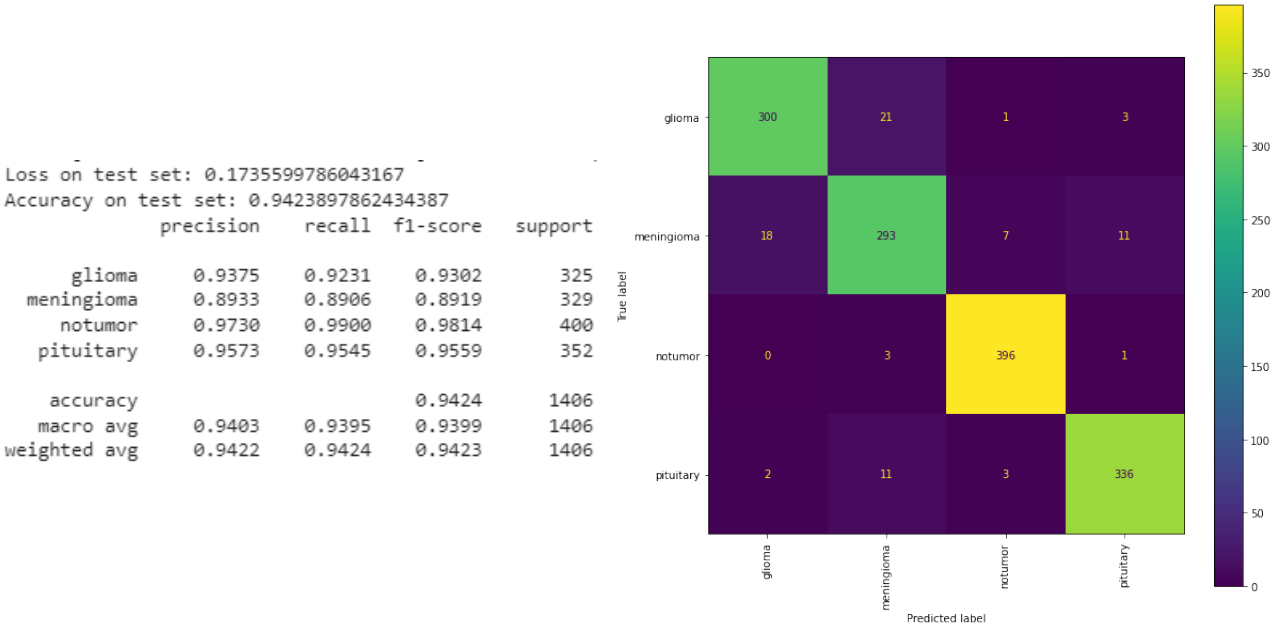


Figure 51: Accuracy and **Loss** on the training and validation set



Loss on test set: 0.1735599786043167
Accuracy on test set: 0.9423897862434387

	precision	recall	f1-score	support
glioma	0.9375	0.9231	0.9302	325
meningioma	0.8933	0.8906	0.8919	329
notumor	0.9730	0.9900	0.9814	400
pituitary	0.9573	0.9545	0.9559	352
accuracy			0.9424	1406
macro avg	0.9403	0.9395	0.9399	1406
weighted avg	0.9422	0.9424	0.9423	1406

Figure 52: Evaluation of the model against the test set

5.6.2 Model Finetuning

The previous model was then finetuned with the last block unfreezed, and the results obtained are the following:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

sequential (Sequential)      (None, 224, 224, 3)       0

tf.math.truediv_2 (TFOpLamb  (None, 224, 224, 3)       0
da)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)       0
a)

tf.math.truediv_3 (TFOpLamb  (None, 224, 224, 3)       0
da)

densenet121 (Functional)     (None, 7, 7, 1024)        7037504

gap (GlobalAveragePooling2D  (None, 1024)              0
)

flatten (Flatten)            (None, 1024)              0

classifier_hidden (Dense)    (None, 256)               262400

dropout_1 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 4)                 1028

=================================================================
Total params: 7,300,932
Trainable params: 587,012
Non-trainable params: 6,713,920
```

Figure 53: Summary of the model, the trainable parameters increased from 263,428 to **587,012**

The results of the training are shown below.

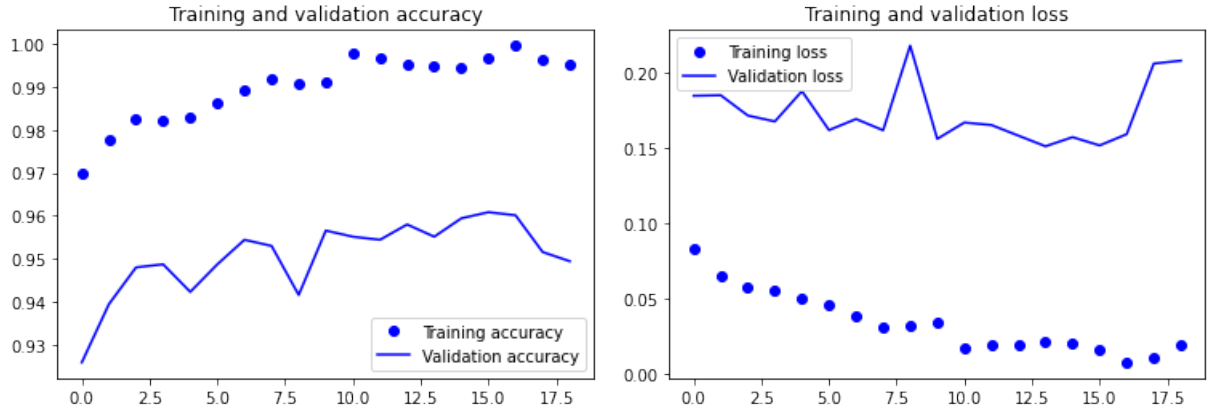


Figure 54: Accuracy and **Loss** on the training and validation set

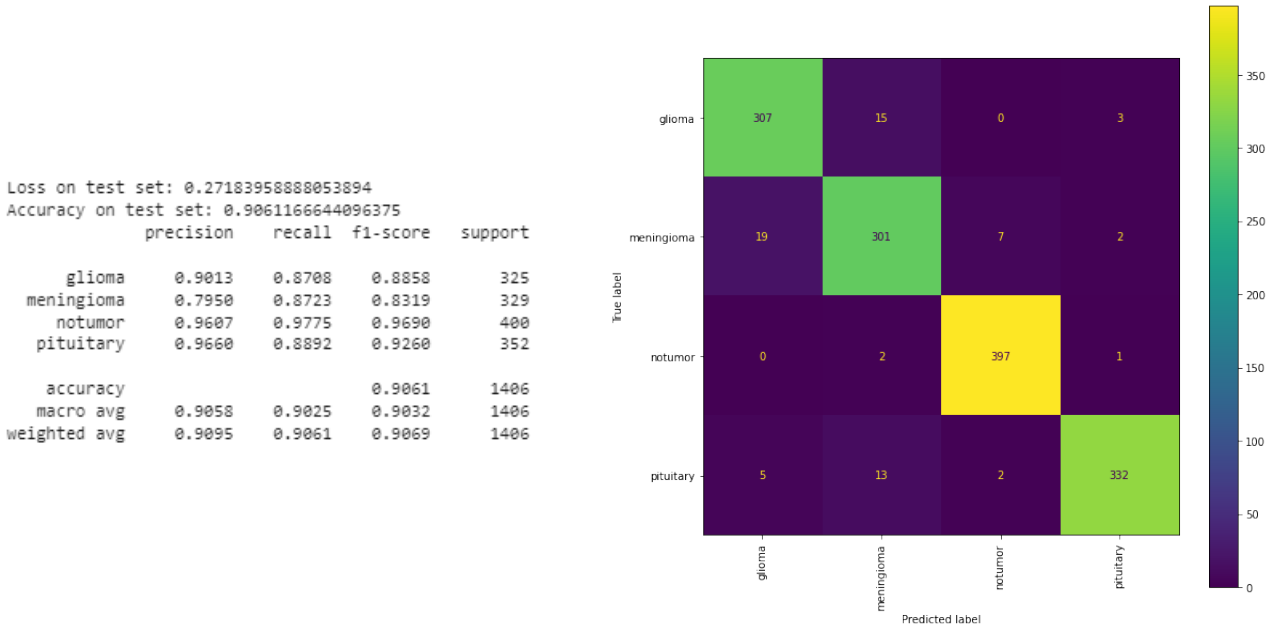


Figure 55: Evaluation of the model against the test set

6 Vision Transformer

In NLP the deep learning model which achieves the best results is the **transformer** [9]. A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. From its introduction in 2017, transformers have replaced RNN model such as LSTM in most applications.

In 2021 it was shown how the transformer architecture could be used for Image Classification [10]. The **Vision Transformer**, or **ViT**, is a model for image classification that employs a Transformer-like architecture over patches of the image, which correspond to the word token in a NLP task. An image is split into fixed-size patches, each of them are then linearly embedded, position embeddings are added, and the resulting sequence of vectors is fed to a standard Transformer encoder. For Image classification tasks an MLP is built on top of the transformer in order to classify the image.

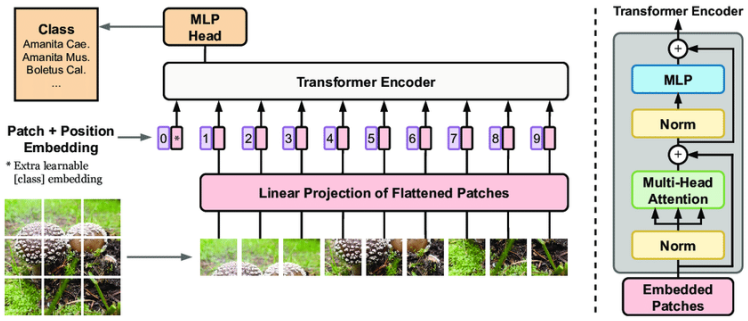


Figure 56: Structure of a Vision Transformer

After their introductions, it was shown how Vision Transformers, if pretrained on huge amount of data, reached the State of the Art performances on image classification of CNNs in term of accuracy [11]. We are now interesting on try on our dataset these types of models.

6.1 From scratch Vision Transformer

Our first experiment is to build a Vision Transformer from scratch and fitting this model to our dataset. The data to build the model was inspired by an example provided on the Keras website [12].

We defined the class for the Patch and the Patch Encoding, and we decided to have patches of 14x14 pixels, which will led to divide our images in 16x16 patches. An example of Patch extraction for an example of our dataset is shown below.

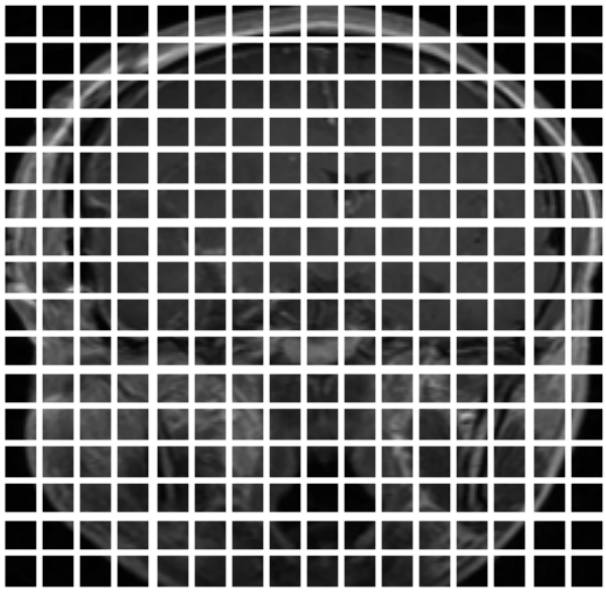


Figure 57: Example of patches on a glioma scan

The model built has 8 transformer layers and 4 heads of the multiHead Attention layer. The results of the training are shown below.

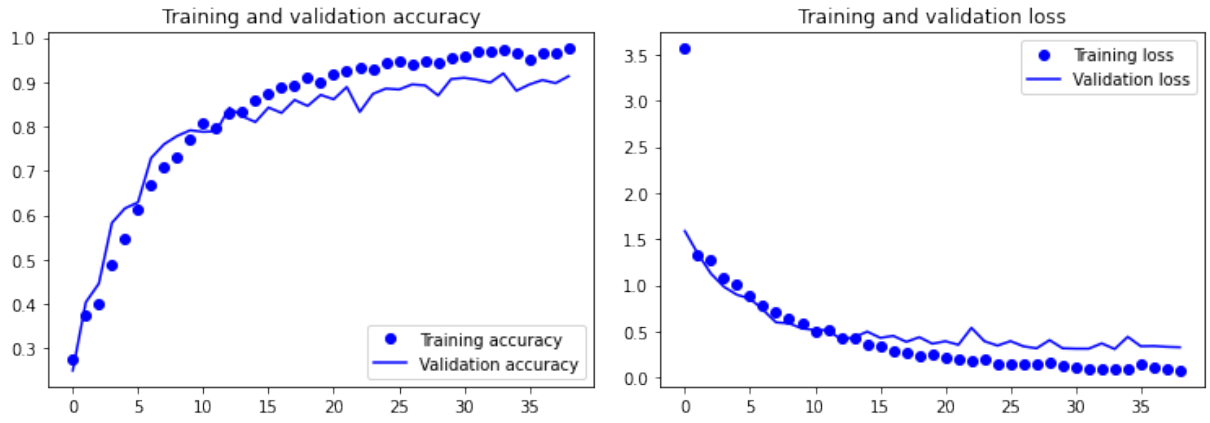


Figure 58: Accuracy and **Loss** on the training and validation set

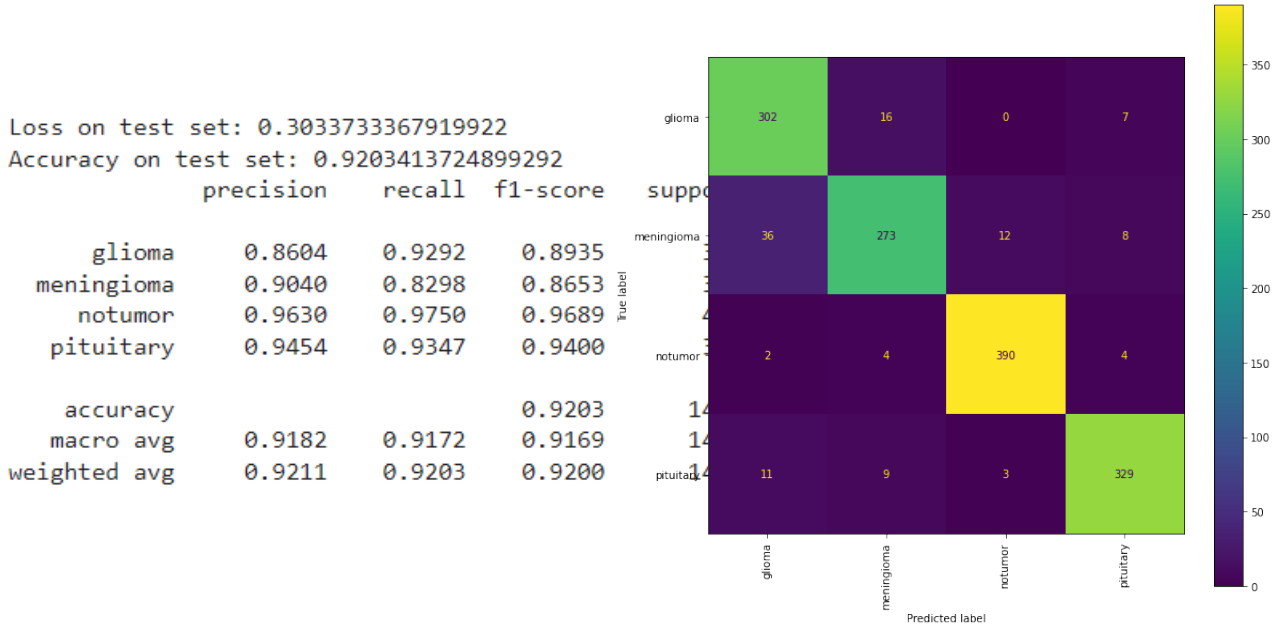


Figure 59: Evaluation of the model against the test set

We can see inferior results with respect to the CNNs, in particular the loss on the test set is particularly disappointing. But as we sayed before, the results which reached State of the art are obtained with pretrained Vision Transformers.

6.2 Pretrained Vision Transformer

We will now try to use a pretrained Vision Transformer, with an MLP built on top. The model used is Vit-B32, provided by Keras.

6.2.1 FineTuning without pretraining of the MLP network

We built on top of Vit-B32 a Dense Neural network with 256 Neurons, and we tried to perform the training once without a dropout layer and once with a layer with dropout rate equal to 0.3. The best results was obtained without dropout, and the results are the following:

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
rescaling (Rescaling)	(None, 224, 224, 3)	0
vit-b32 (Functional)	(None, 768)	87455232
flatten (Flatten)	(None, 768)	0
batch_normalization (BatchNormalization)	(None, 768)	3072
hidden_classifier (Dense)	(None, 256)	196864
dense (Dense)	(None, 4)	1028

Total params: 87,656,196
Trainable params: 87,654,660
Non-trainable params: 1,536

Figure 60: Summary of the model

The results of the training are shown below.

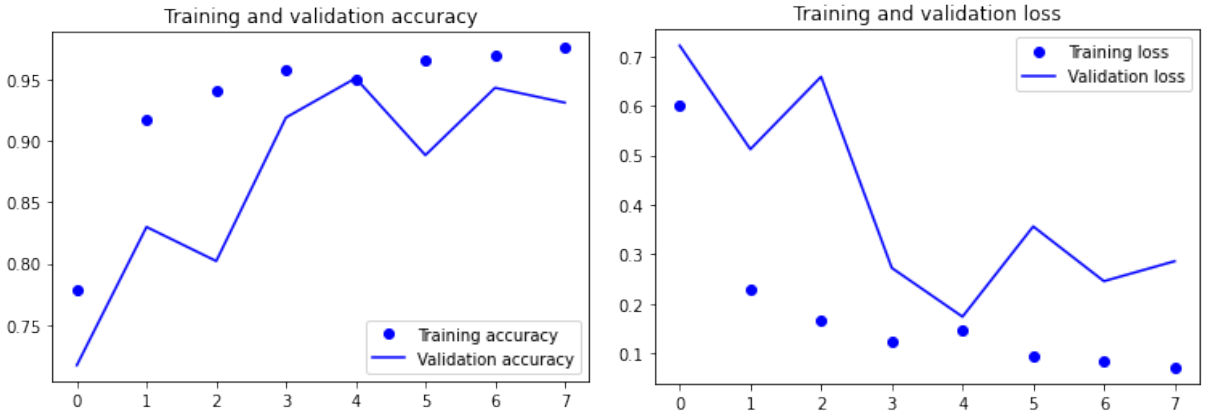


Figure 61: Accuracy and **Loss** on the training and validation set

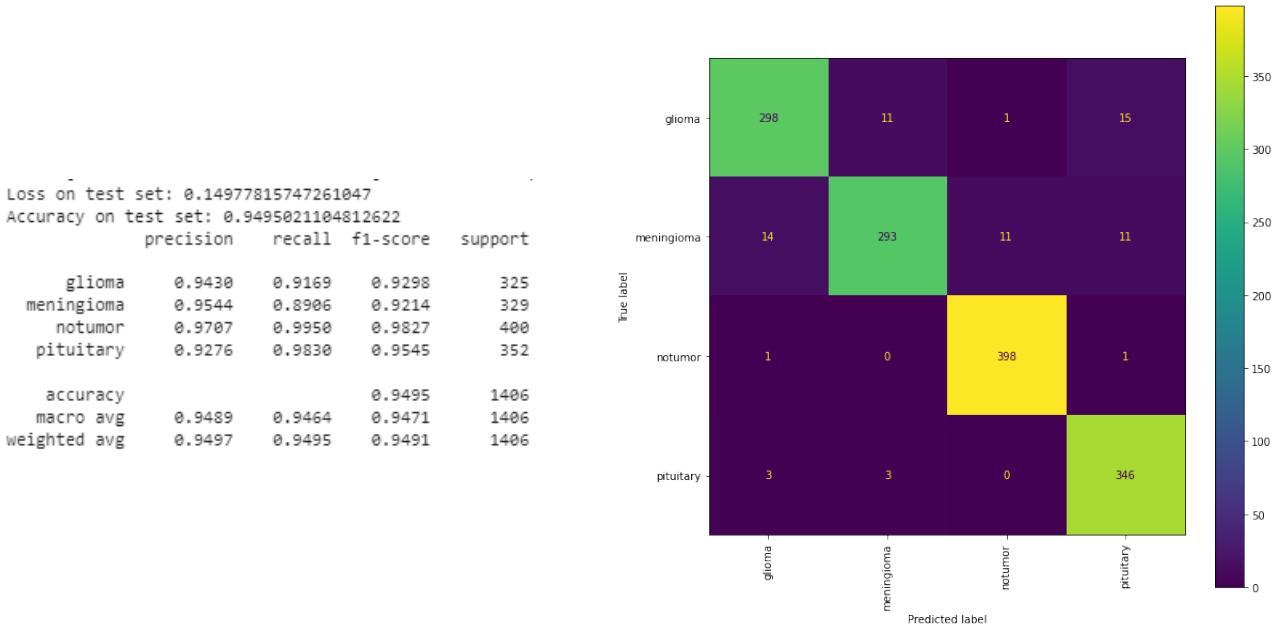


Figure 62: Evaluation of the model against the test set

As we could expect, the training history is quite unstable. This is clearly due to the lot amount of parameters that we needed to tune, since there wasn't any freezed layer in the model. Aside from this, the results are quite good and they are perfectly comparable to the ones obtained with CNNs.

6.2.2 FineTuning with pretraining of the MLP network

In the training of the CNNs we used to first pretrain the MLP network on top leaving the base CNN freezed and then we unfreeze some block of the model and perform the finetuning. Our objective is to try this approach with Vision Transformer, and to see if this improves the results obtained in the last chapter. this time when we perform finetuning we will unfreeze the all ViT model. We decided to build the same model saw in the last chapter without Dropouts, since it achieved better performances. The first phase of training gave us this results:

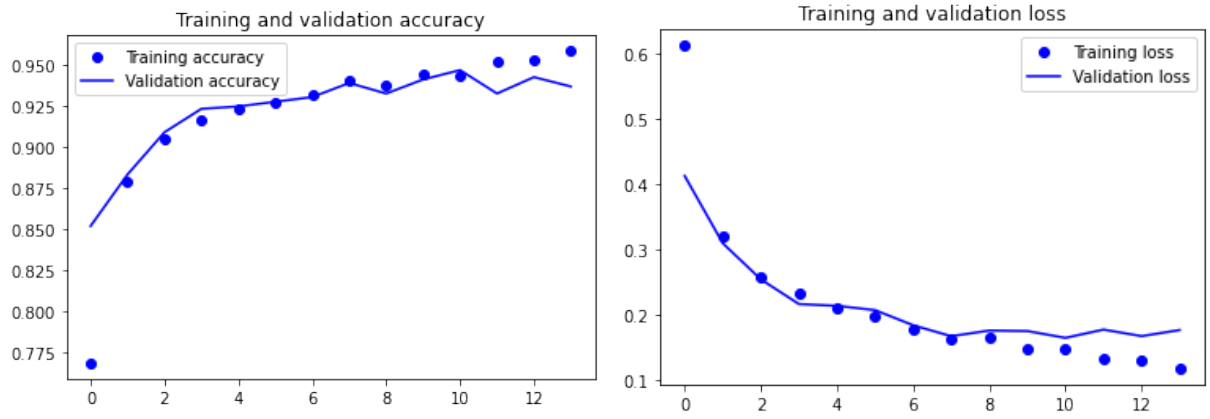
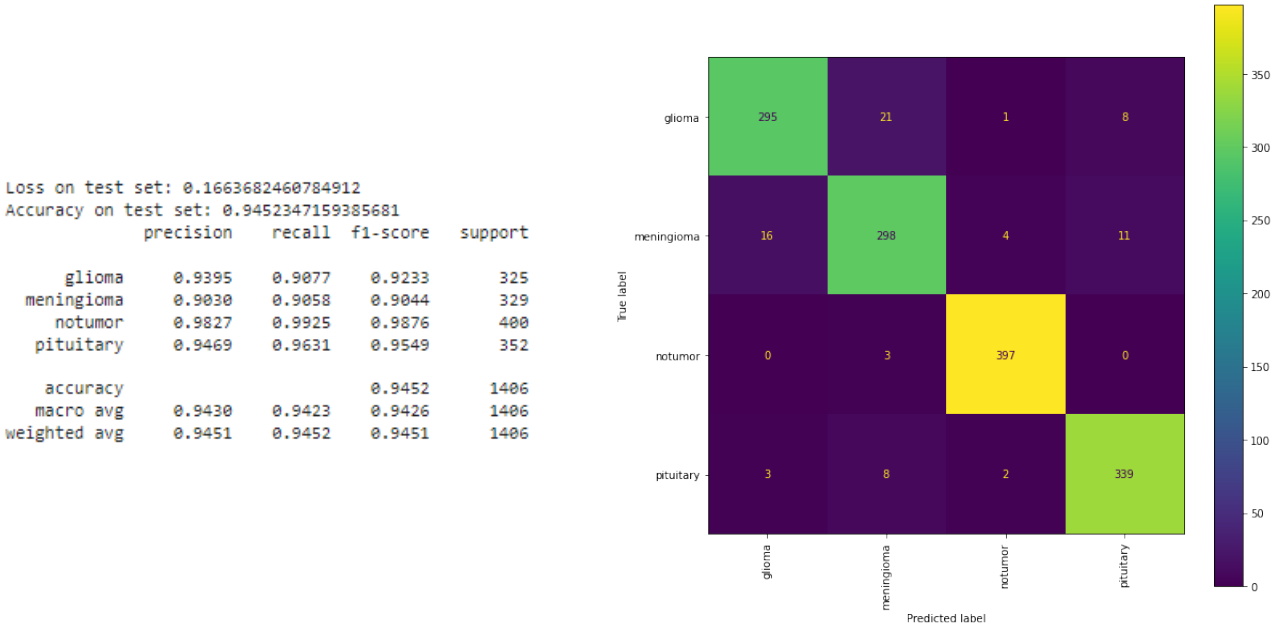


Figure 63: History of the model with ViT freezed



```
Loss on test set: 0.1663682460784912
Accuracy on test set: 0.9452347159385681
              precision    recall  f1-score   support

      glioma     0.9395    0.9077    0.9233       325
  meningioma     0.9030    0.9058    0.9044       329
     notumor     0.9827    0.9925    0.9876       400
   pituitary     0.9469    0.9631    0.9549       352

    accuracy                         0.9452      1406
   macro avg     0.9430    0.9423    0.9426      1406
weighted avg     0.9451    0.9452    0.9451      1406
```


Figure 64: Evaluation of the model with ViT freezed

The results are comparable with the training done in the last chapter, and since we only have the MLP network to train, the history of the training is more stable. Now we unfreeze ViT and reperform the training:

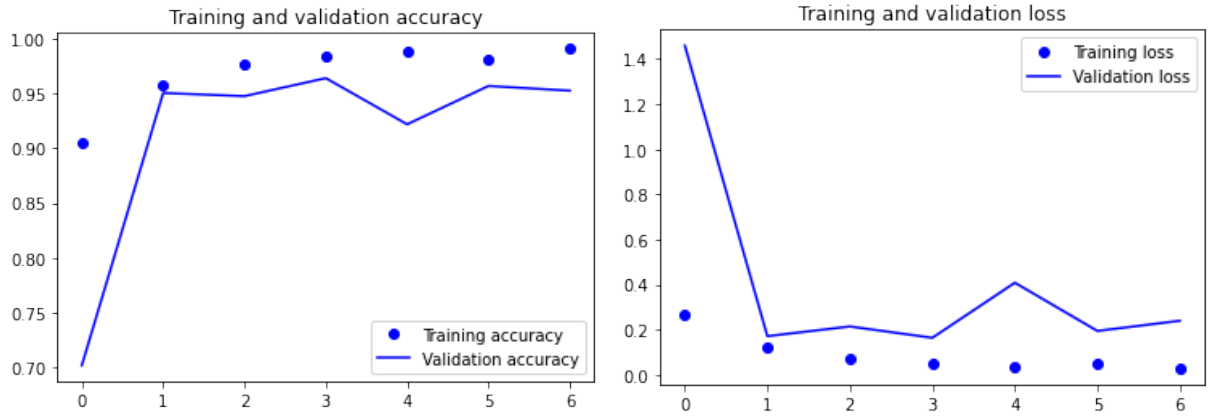
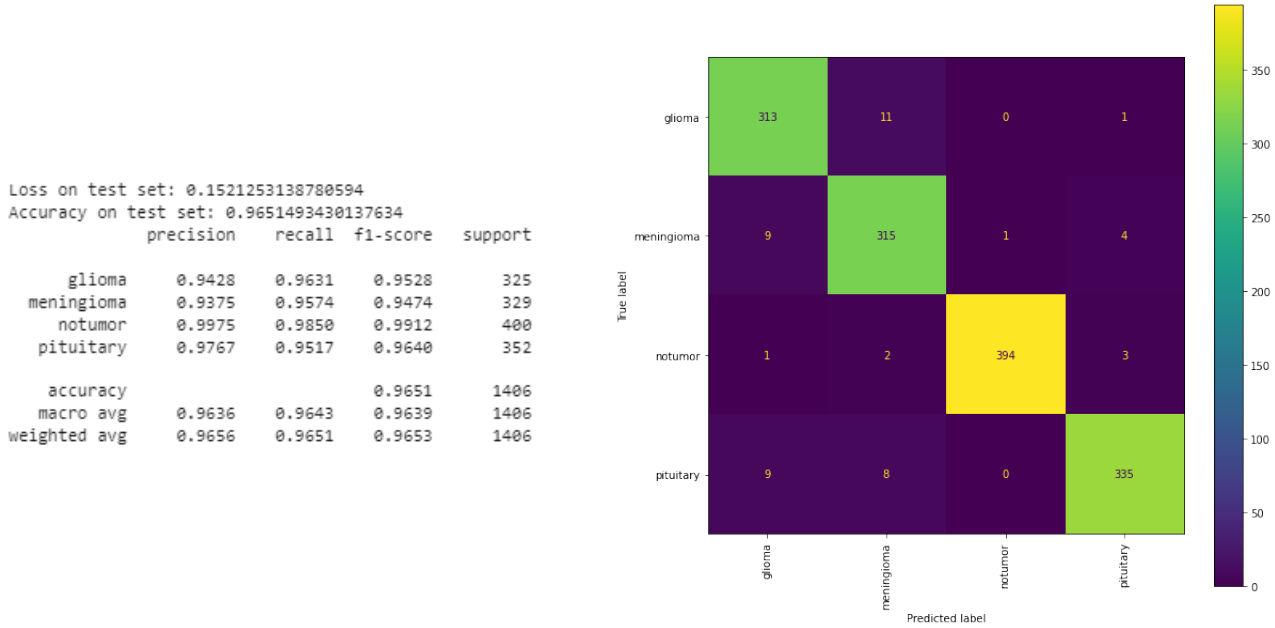


Figure 65: History of the model with ViT finetuned



```
Loss on test set: 0.1521253138780594
Accuracy on test set: 0.9651493430137634
              precision    recall  f1-score   support

      glioma     0.9428    0.9631    0.9528       325
  meningioma     0.9375    0.9574    0.9474       329
     notumor     0.9975    0.9850    0.9912       400
   pituitary     0.9767    0.9517    0.9640       352

    accuracy                         0.9651      1406
   macro avg     0.9636    0.9643    0.9639      1406
weighted avg     0.9656    0.9651    0.9653      1406
```


Figure 66: Evaluation of the model with ViT finetuned

As we can see the best results on the validation set are obtained immediately, and the results on the test set let us say that this model is one of the best model we have built. The outstanding result of this model, is the precision on tumor, which is almost perfect.

7 Explainability

The goal of this chapter is to try to understand how the model built works. In particular we will see the **intermediate activations** of the filters and the **heatmaps** of the class activations.

We will analyze only the best models found for each type on CNN analyzed (the Vision Transformer model is excluded by this part). For VGG16 we choose the model with the best accuracy (the one Finetuned progressively).

7.1 Intermediate activations

In this chapter we will display some intermediate activations of the best model based on CNN found in the last chapter having as input a glioma image, with the goal of trying to understand better how this models work.

7.1.1 CNN from scratch

In the following pictures we will display the feature maps of the first Convolutional layer and of the last one.

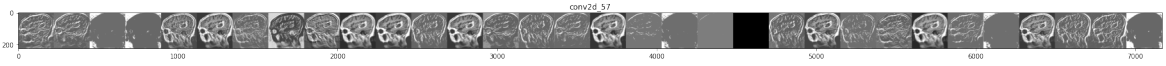


Figure 67: First feature map

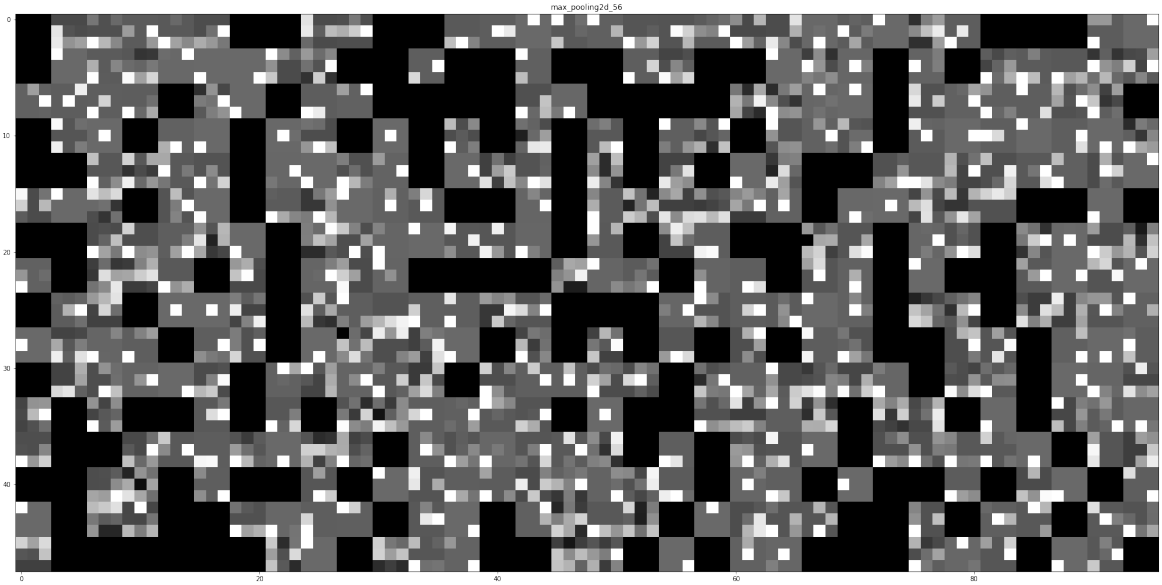


Figure 68: Last feature map

We can see how in the first layer the network is focusing on edges of the image, and in the last layer instead we only have abstract informations, not attributable for a human being to the initial images. It's interesting to notice also how the in last layer we have about 15% of the filters inactive.

7.1.2 VGG16

We show an intermediate activation on an internal layer of the model (the first Convolutional layer of the second block).

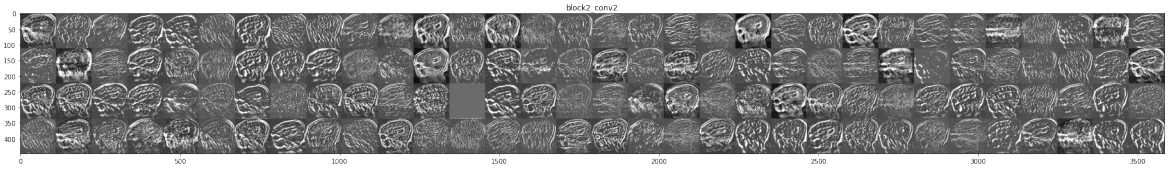


Figure 69: 4th feature map of VGG16

It's interesting to see how VGG16 have feature maps all with the same contrast, with all the edges highlighted in white.

7.1.3 Resnet-50

In Resnet-50 we have an opposite situation wrt to VGG16, with a very high variance on the constrant of the activations of the filters of the same layers.

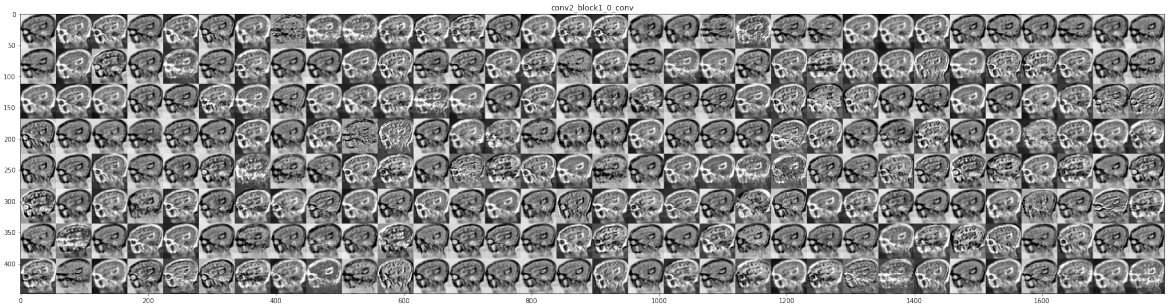


Figure 70: 4th feature map of Resnet

7.1.4 Densenet121

On Densenet we can see how the filters apply more distortion to the input image.

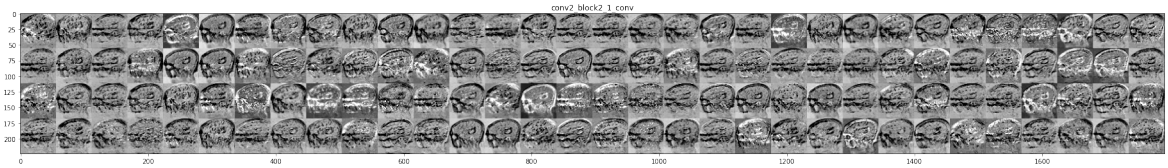


Figure 71: 4th feature map of Densenet

7.2 Heatmaps

Now we are going to display the heatmaps of all the analyzed model in this chapter for one image for class. We are interested in highlighting how the different models reason in order to classify an image, and the best situation we could see is that for a tumor image the heat part is where the tumor is located. Along with the images it is also plot the confidence with which the model decides the label of that image.

7.2.1 Glioma image

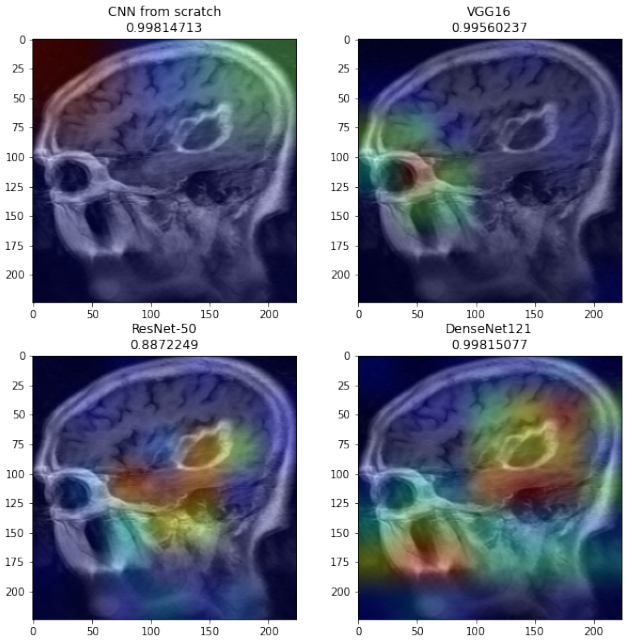


Figure 72: Heatmaps of models of a glioma image

We can see how Resnet has a perfect focus on the glioma tumor, despite its confidence is the worst of all the four classifier. Also Densenet has the tumor on its heat zone, but its focus region is a little more dispersive with respect to Resnet. VGG16 and the CNN from scratch model have a focus in a totally different part from the location of the tumor.

7.2.2 Meningioma image

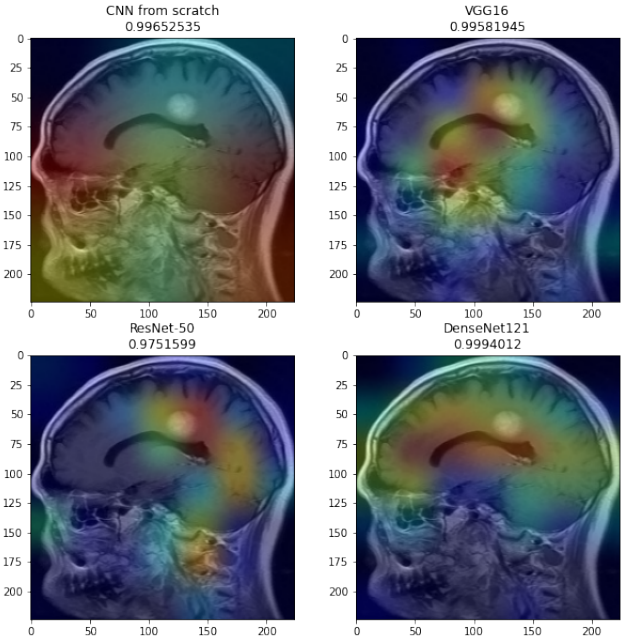


Figure 73: Heatmaps of models of a meningioma image

Again we have a situation in which the focus of the CNN from scratch is more dispersive, analyzing all the image parts, even the one not relative to the brain. We are happy to see instead how the pretrained CNNs have focus only on the part of the images that can effectively contain a brain tumor. For the meningioma image VGG and Resnet have the more correct focus on the tumor part.

7.2.3 No Tumor image

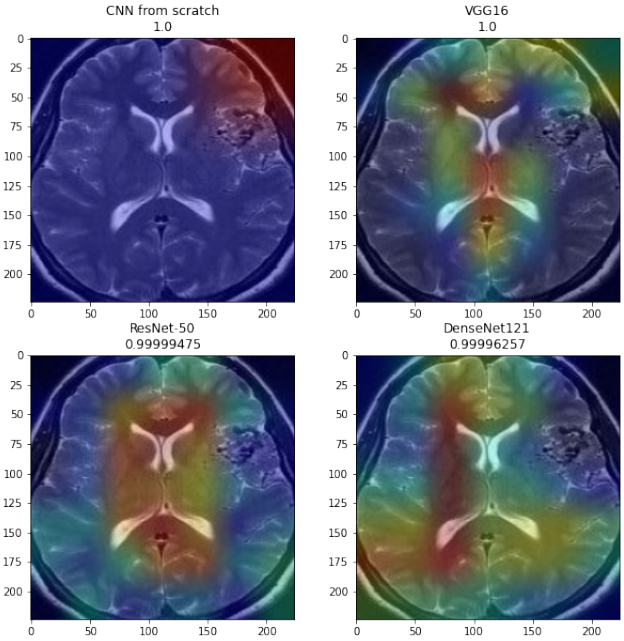


Figure 74: Heatmaps of models of a no tumor image

As could be expected, higher portions of the brain are analyzed in the no tumor images by all the CNNs, apart from the CNN from scratch that we can state has a completely different way of analyzing images.

7.2.4 Pituitary

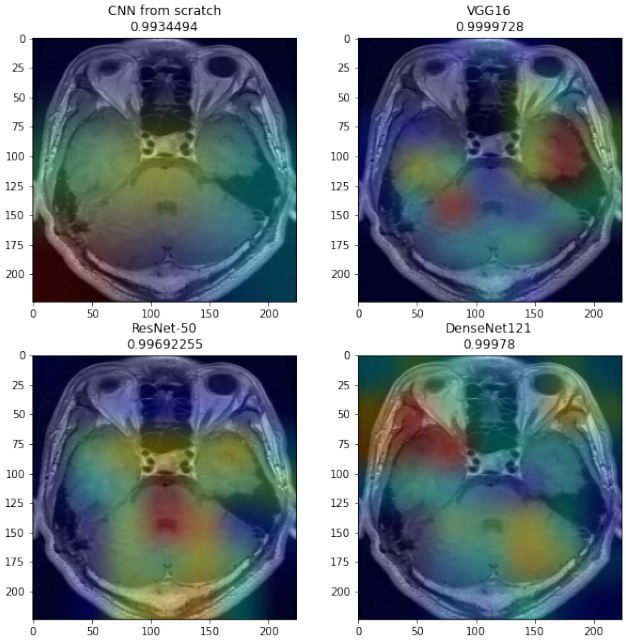


Figure 75: Heatmaps of models of a pituitary image

For the pituitary image we have again that Resnet has the most corrected focus on the tumoral gland. The Cnn from scratch in this case centrate the tumoral part of the image.

7.3 Comparison between the two VGG16 model

In the VGG16 notebook we obtained two different models, one finetuned with a progressive unfreezing of the layers of the last block and one with the last block finetuned entirely in the first step. From the Confusion Matrix we saw how the classifications of this two models were different. Now we are interested in understanding if the intermediate activations or the heatmaps of the models are different.

7.3.1 Intermediate activations

Obviously the first layers have the same feature maps since the weights of these blocks are freezed, so we are gonna compare the last feature maps in order to see the differences between the two models

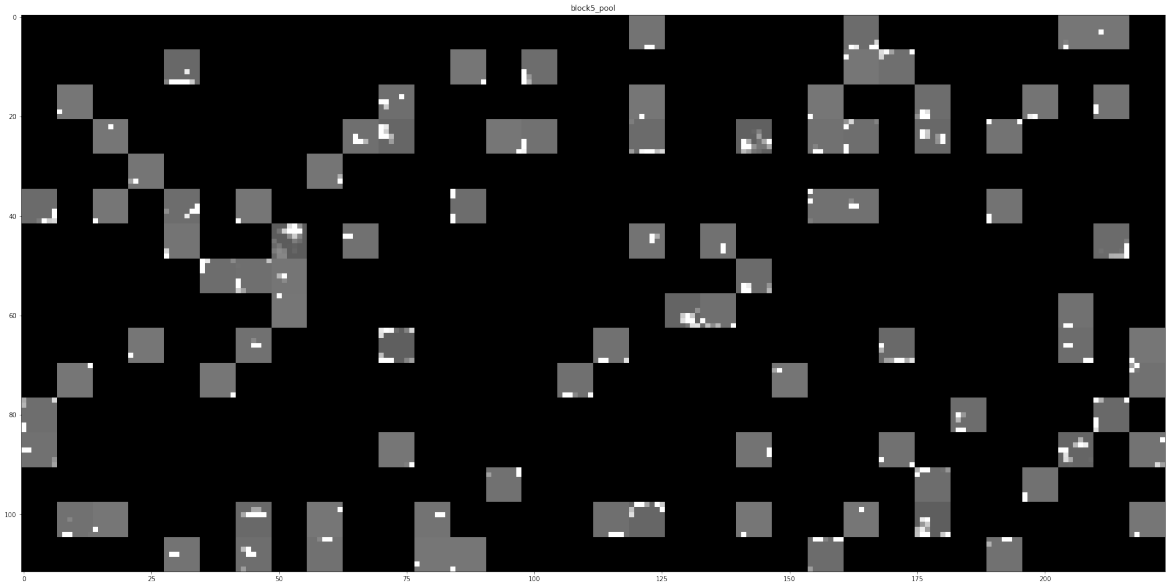


Figure 76: Last feature map of the VGG16 model finetuned progressively

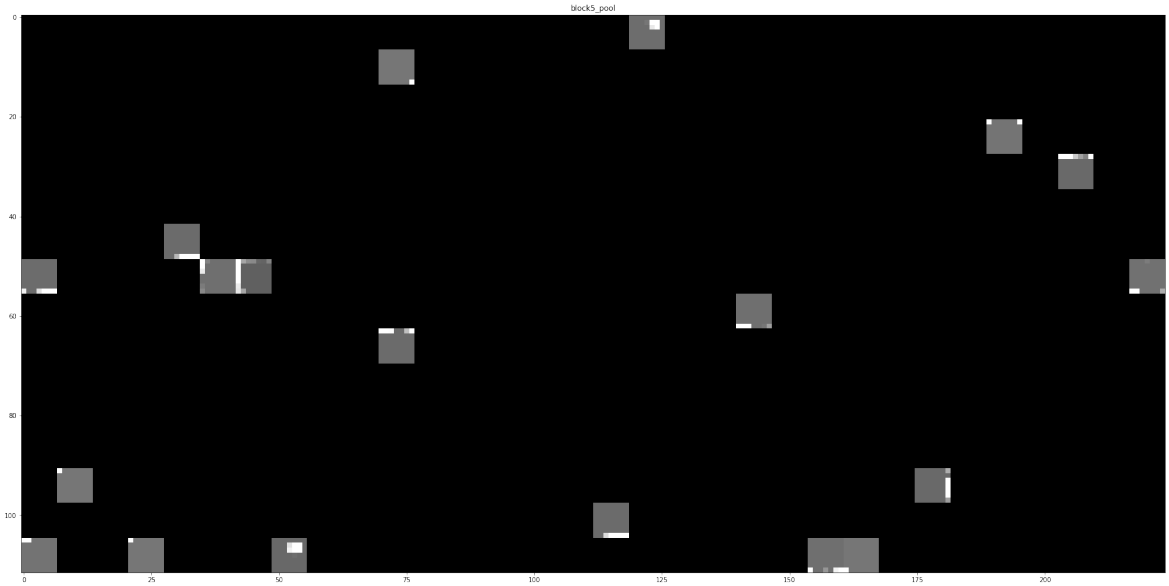


Figure 77: Last feature map of the VGG16 model finetuned in one shot

7.3.2 Heatmaps

As we can see the differences are enourmous: the first model has a huge amount of active filters with respect to the second one.

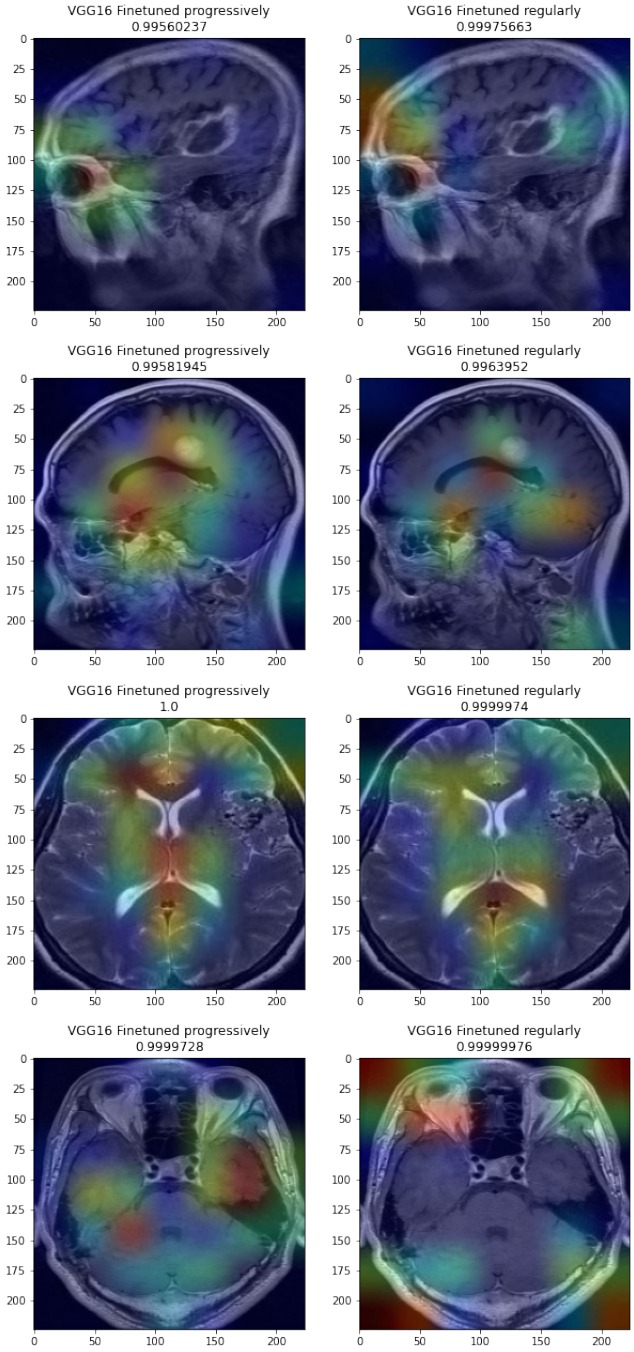


Figure 78: Heatmaps the two VGG16 models

From this heatmaps we can see how the the two models have similar focus regions on the glioma image and on the no tumor one. Also on the meningioma the situation it is not so different, but the first model has a better focus on the tumoral part of the brain. In the pituitary image we have the most important differences: in fact we can almost say the two heat regions are almost complementary and there is nothing in common from the two models.

From this heatmaps, along with the feature maps of the last layers, we can appreciate better the difference from this two model, and understand why there was so much differences in the confusion matrices of the two models.

8 Ensemble

In the Experiments chapter we built models which achieved very satisfying results, in this chapter we are gonna combine the predictions of this models with the aim of improving the performances and the robustness of the ensembled classifier with respect to the single ones. In particular we are gonna use the best model found for each type of network, with an exception for VGG16: since we saw how different the confusion matrix of the two models was, we are gonna keep all the two models with the hope of having good heterogenous predictions.

```
from_scratch_model = models.load_model(MODELS_PATH + '/Dense512Neurons_TwoDropouts0-3_AdditionalLayer/Dense512Neurons_TwoDropouts0-3_AdditionalLayer.h5')
vgg_model = models.load_model(MODELS_PATH + '/VGG16_LastBlockFineTuned/VGG16_LastBlockFineTuned.h5')
vgg2_model = models.load_model(MODELS_PATH + '/VGG16_LastBlockFineTunedEntirely/VGG16_LastBlockFineTunedEntirely.h5')
resnet_model = models.load_model(MODELS_PATH + '/ResNet50_Finetuned/ResNet50_Finetuned.h5')
densenet_model = models.load_model(MODELS_PATH + '/DenseNet121_Finetuned/DenseNet121_Finetuned.h5')
vit_model = models.load_model(MODELS_PATH + "/VisionTransformerb32_Dense256_Finetuning/"+"VisionTransformerb32_Dense256_Finetuning.h5")
```

Figure 79: Loading of the models

8.1 Average model

The easiest way to aggregate the predictions of a set of classifiers is to average their predictions and to extract the ensemble classified label from this prediction.

As we could expect, the results of this ensembled model are better of the others taken individually.

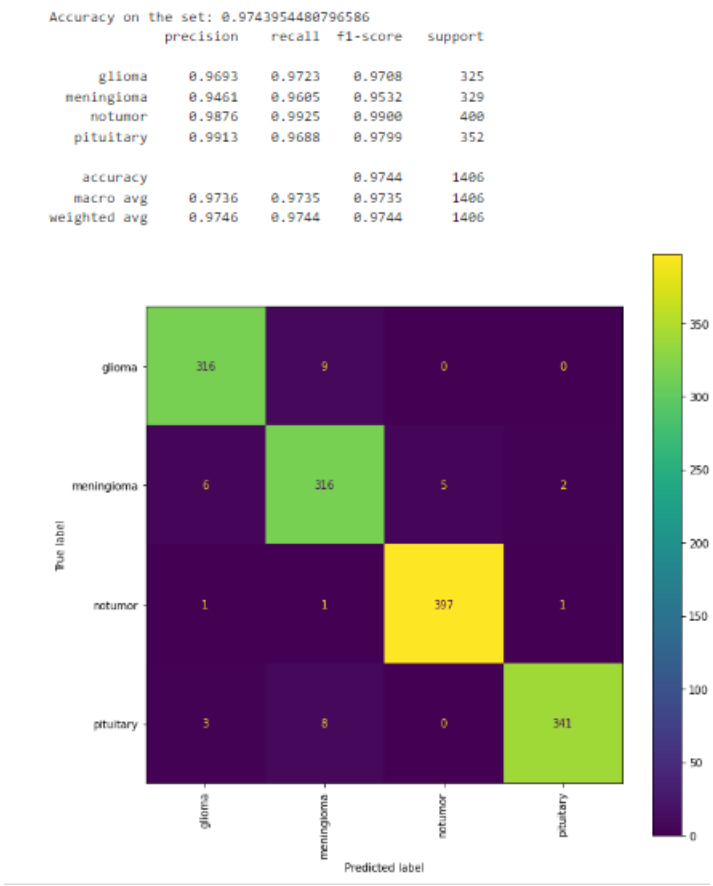


Figure 80: Average ensembled model results

8.2 Weighted Average model

Another possibility to ensembling our models is to give a weight to each model prediction, in order to improve the importance of better models in the final result. Given 6 models and an image, the prediction of the ensemble model will be:

$ensemble\_prediction = a*model1\_prediction+b*model2\_prediction+c*model3\_prediction+ d*model4\_prediction + e*model5\_prediction + f*model6\_prediction$

where a, b, c, d, e, f represent the weights associated to the models and the model predictions are vector with the element i representing the confidence with which the model believes the image belongs to class i. If the weights are all equal to $\frac{1}{6}$, we obtain the model equal to the Average Model seen in the last chapter.

We are gonna use two different techniques in order to find the best weights of the model. In all the two experiments we will build two models: one which will maximize the accuracy on the validation set and one that will maximize the precision on no tumor image.

8.2.1 Brute Force

One way to found a good set of weights is to try all the weights within a certain step (respecting the constraint that the sum of all the weights needs to be 1) and try all the combinations keeping only the models with the best results on the validation set. Since we have 6 model, choosing a not small step size (0.05), this approach is feasible in a moderate time, but since the algorithmic complexity of this solution is $O(step^n)$, where n is the number of models, this approach becomes unfeasible if the number of model is higher or we want an higher precision decreasing the step size.

```
Model with best accuracy:
Weights: [0.2, 0.3500000000000003, 0.05, 0.4, 0.0, 0.0]
Accuracy on the set: 0.9694167852062588
              precision    recall  f1-score   support

      glioma     0.9720    0.9631    0.9675       325
  meningioma     0.9375    0.9574    0.9474       329
     notumor     0.9802    0.9900    0.9851       400
   pituitary     0.9855    0.9631    0.9741       352

    accuracy                         0.9694      1406
   macro avg     0.9688    0.9684    0.9685      1406
weighted avg     0.9696    0.9694    0.9695      1406
```

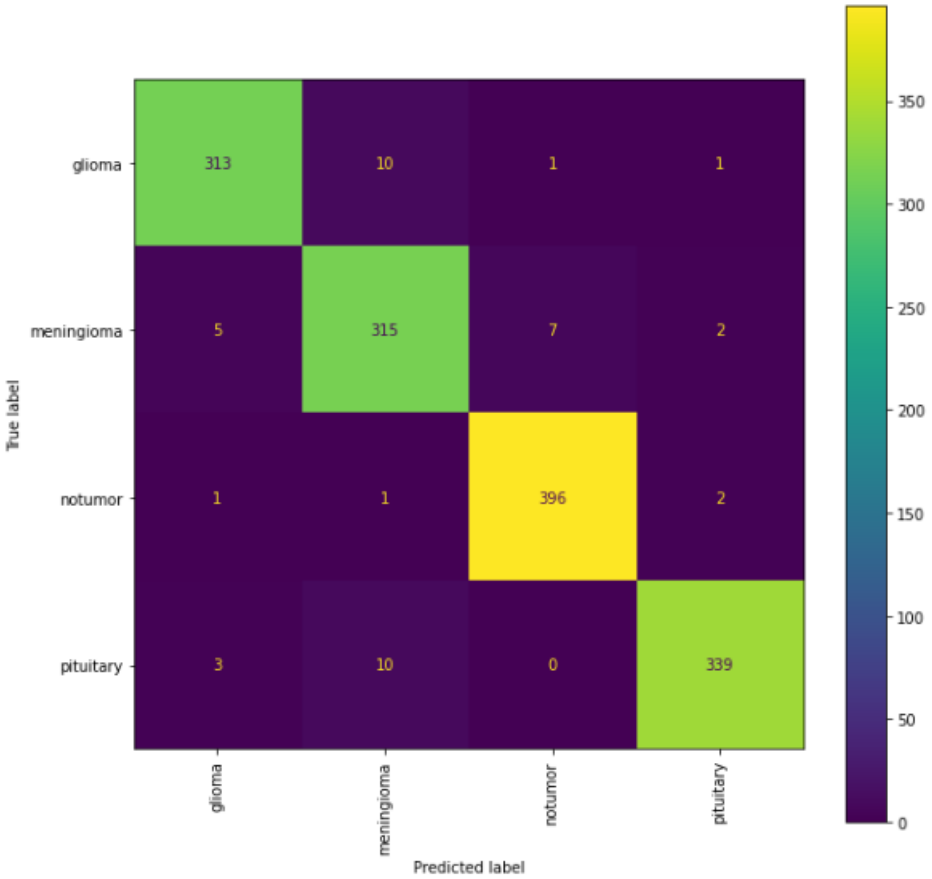


Figure 81: Best model found wrt **accuracy on validation set**

```
Model with best precision on notumor images:
Weights: [0.4, 0.15000000000000002, 0.0, 0.2, 0.0, 0.25]
Accuracy on the set: 0.9715504978662873
              precision    recall  f1-score   support

      glioma     0.9720    0.9600    0.9659       325
  meningioma     0.9349    0.9605    0.9475       329
     notumor     0.9876    0.9975    0.9925       400
   pituitary     0.9883    0.9631    0.9755       352

    accuracy                         0.9716      1406
   macro avg     0.9707    0.9703    0.9704      1406
weighted avg     0.9718    0.9716    0.9716      1406
```

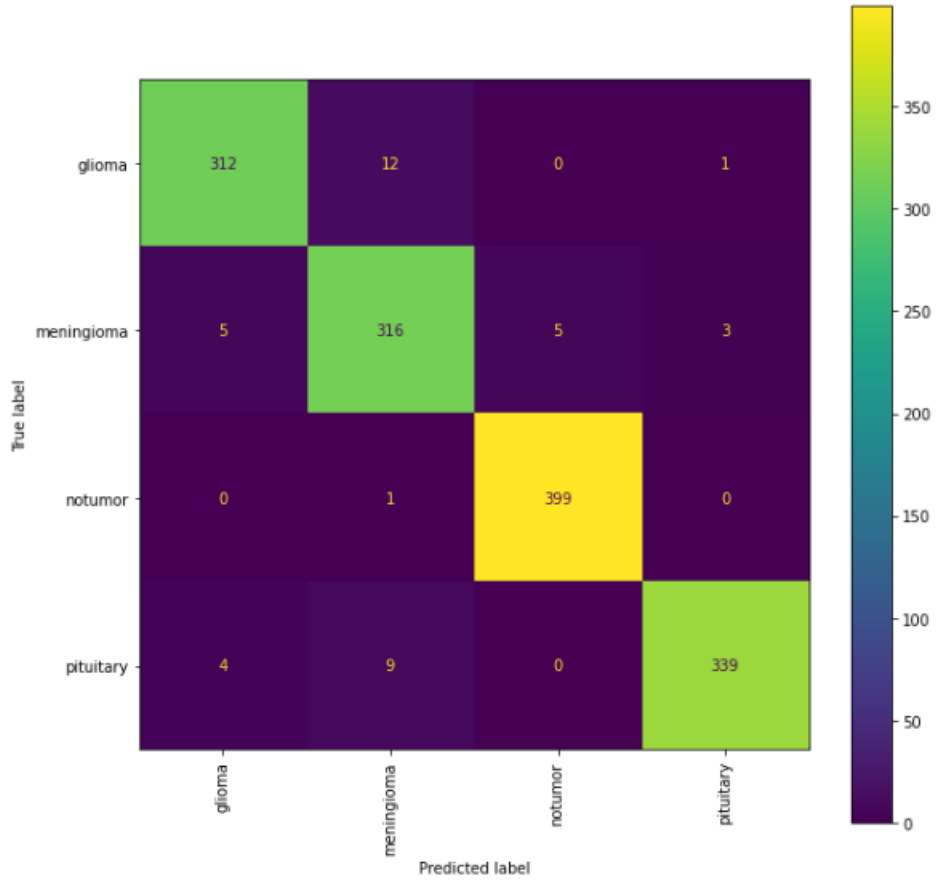


Figure 82: Best model found wrt **precision on no tumor**

8.2.2 Genetic Algorithm

The weights of the model can be found with a Genetic algorithm where chromosomes are composed by six genes (representing the weight of each model). Each transformation (crossover and mutation) will keep the sum of the genes of each chromosome equal to one.

We will run the algorithm two times with two different fitness functions: one is the accuracy on validation set and one is the precision on no tumor images. In

both the execution we will do 8 iterations, with a population size equal to 100, 50 generations per iteration, 0.5 as probability of crossover and 0.01 as probability of mutation. This parameters values were found experimentally. The two executions of the algorithm gave us the following output:

```
Iteration: 0
Generation 1, best fitness = 0.9836

Iteration: 1
Generation 1, best fitness = 0.9829

Iteration: 2
Generation 1, best fitness = 0.9829
Generation 4, best fitness = 0.9836

Iteration: 3
Generation 1, best fitness = 0.9815
Generation 3, best fitness = 0.9822
Generation 7, best fitness = 0.9829
Generation 9, best fitness = 0.9836

Iteration: 4
Generation 1, best fitness = 0.9822
Generation 3, best fitness = 0.9829
Generation 5, best fitness = 0.9836

Iteration: 5
Generation 1, best fitness = 0.9815
Generation 3, best fitness = 0.9822
Generation 4, best fitness = 0.9829
Generation 5, best fitness = 0.9836

Iteration: 6
Generation 1, best fitness = 0.9822
Generation 3, best fitness = 0.9829
Generation 6, best fitness = 0.9836

Iteration: 7
Generation 1, best fitness = 0.9822
Generation 2, best fitness = 0.9829
Generation 3, best fitness = 0.9836
```

Figure 83: Best accuracy model generation

```
Iteration: 0
Generation 1, best fitness = 0.9950

Iteration: 1
Generation 1, best fitness = 0.9950

Iteration: 2
Generation 1, best fitness = 0.9950

Iteration: 3
Generation 1, best fitness = 0.9950

Iteration: 4
Generation 1, best fitness = 0.9950

Iteration: 5
Generation 1, best fitness = 0.9950

Iteration: 6
Generation 1, best fitness = 0.9950

Iteration: 7
Generation 1, best fitness = 0.9950
```

Figure 84: Best precision on no tumor model generation

We obtained the following weights:

	Best Accuracy	Best Precision on no tumor
CNN from Scratch	0.1986029095943389	0.08920340824313877
VGG16 1	0.31829453290803167	0.3380197750624596
VGG16 2	0.008992576909267537	0.057396108034599526
ResNet-50	0.3342054907761426	0.07635628651033366
DenseNet121	0.13412602339953872	0.19225870660774622
ViT	0.005778466412680734	0.24676571554172216

The iterations on the precision on no tumor execution are not so meaningful, since at the first iteration we found as best score on the validation set 0.995 and no better score is found later, so all the new best for generation are ex aequo and the solution is chosen randomly from this solutions.

```
Accuracy on the set: 0.972972972972973
              precision    recall  f1-score   support

      glioma     0.9783    0.9692    0.9737       325
  meningioma     0.9431    0.9574    0.9502       329
     notumor     0.9826    0.9875    0.9850       400
   pituitary     0.9856    0.9744    0.9800       352

    accuracy                         0.9730      1406
   macro avg     0.9724    0.9722    0.9722      1406
weighted avg     0.9731    0.9730    0.9730      1406
```

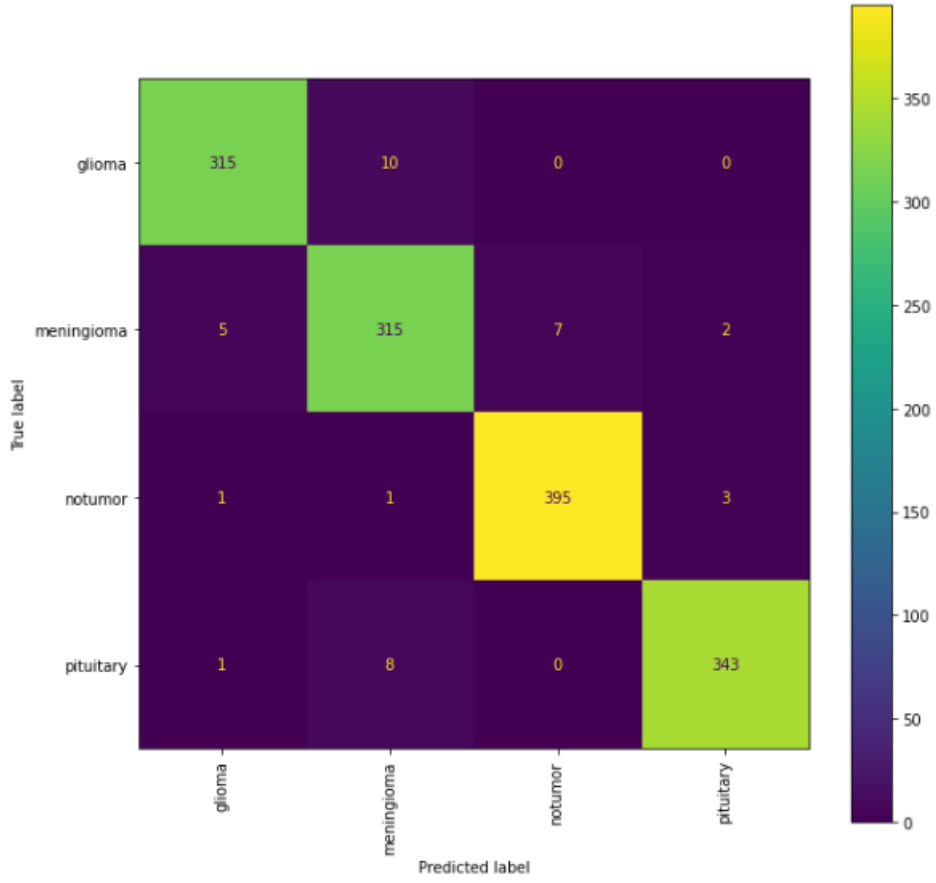


Figure 85: Best model found wrt **accuracy on validation set**

Accuracy on the set: 0.9751066856330014

	precision	recall	f1-score	support
glioma	0.9634	0.9723	0.9678	325
meningioma	0.9489	0.9605	0.9547	329
notumor	0.9950	0.9900	0.9925	400
pituitary	0.9885	0.9744	0.9814	352
accuracy			0.9751	1406
macro avg	0.9740	0.9743	0.9741	1406
weighted avg	0.9753	0.9751	0.9752	1406

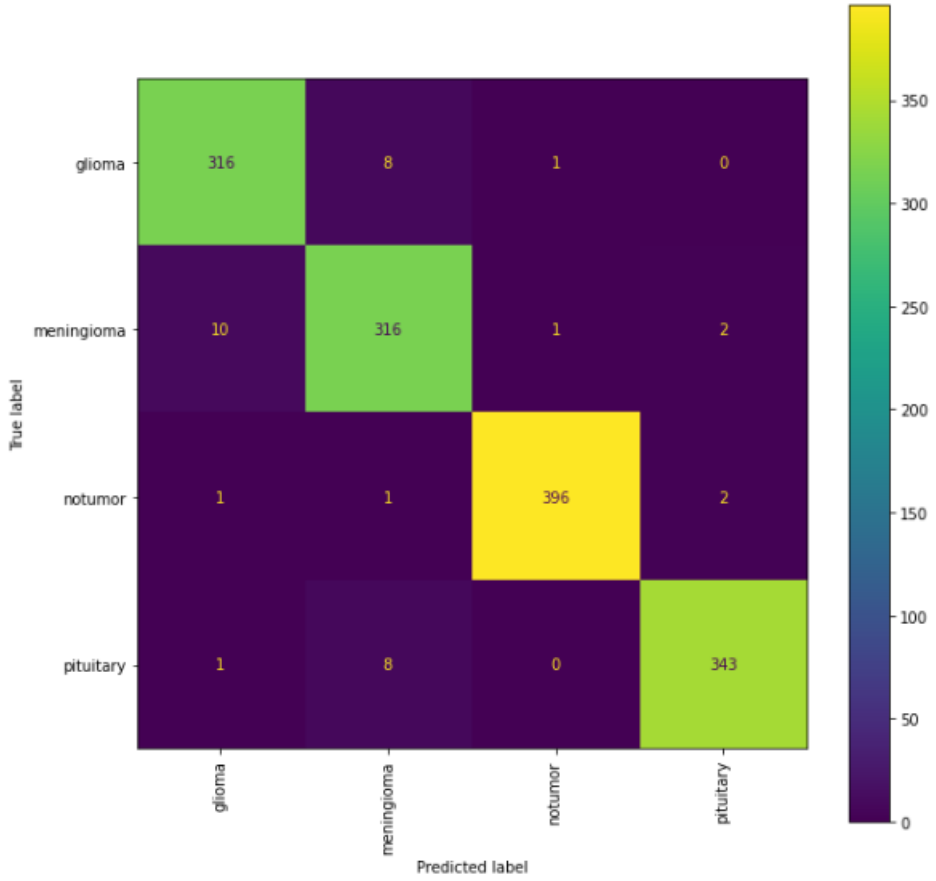


Figure 86: Best model found wrt **precision on no tumor**

Even with the considerations about the problems of the genetic algorithm for founding the model with best precision, we achieved a better results in every term than in the model found with the best accuracy on validation set. In fact, in addition to the almost perfect precision on no tumor obtained (just two misclassifications on the test set), we also obtained the best accuracy on the test set. Even if this outcome is odd considering the methodology applied, we must treat it as a probabilistic event based on particularity of our dataset splits. In fact we have that the first model misclassifies 35 images and the second 33, when we have this little difference we can attribute them to probability, and so this can not question the fact that the best method to obtain a model which maximize accuracy we have to put as fitness

function the accuracy on validation set.

It is interesting to discuss of how the weights found with genetic algorithm are similar for the best accuracy model to the ones found with the Brute Force methods, with the same decisions in which models give more importance. This similarity is not found in the best precision on no tumor model.

8.3 Error analysis

Now that with the ensembling we built our best model, the ones with best precision on no tumor images obtained with the genetic algorithm, we are now interested in visualizing the 33 misclassified images.

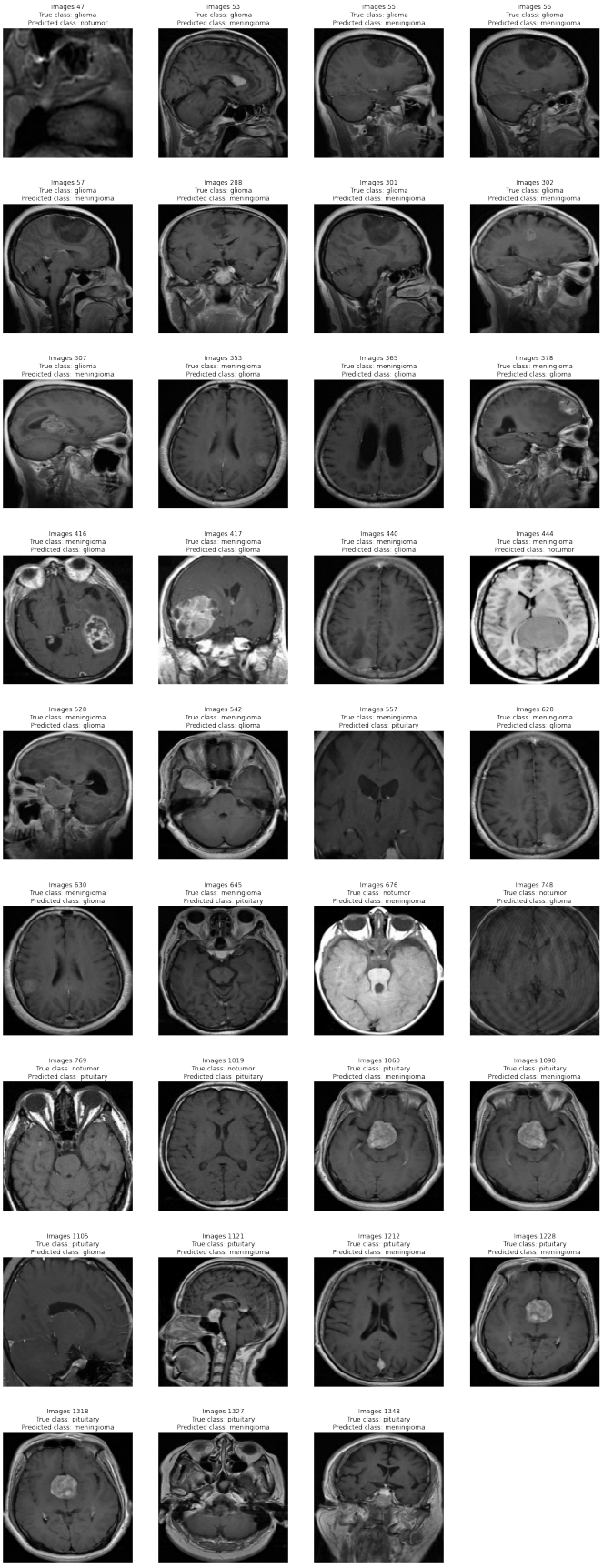


Figure 87: Error of the ensembled model

From this error there are some which are worth to mention. In image 47 there was an incorrect preprocessing of the image which made the tumor impossible to scan. In fact there was an incorrect cropping which excluded important parts of the brain, and this image is one of the two which made our model incapable of reaching th 100% of precision on notumor images.

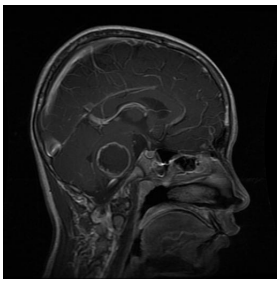


Figure 88: Uncropped original image

Images 55, 56, 57 and 301 are really similar situations: we have a glioma tumor on top of the brain, and the ensembled model recognise it has a meningioma. From this four errors we can deduce that this situation is problematic for our models used to construct the ensembled one. In general we can see how the model never made mistake for glioma images with scans taken from above. Instead all the times that our model uncorrectly predicts the image as a non tumor one, we have that the scan is made from above.

The most serious mistakes is taken on image 444, where we labelled the brain patient as healthy when there is an enourmous meningioma.

9 Conclusion

In this work we tested from scratch CNN, pre-trained CNNs and vision transformer for the classification of brain tumors. We discuss the differences of these models and then we ensembled them improving their performances. We saw how the differents techniques to find the weights are important in order to maximize the results of our model.

In terms of accuracy we obtained a model with about 97.6% of accuracy, which is totally comparable to the results discussed in the Related Work chapter. An important result we have to be proud about is that we reached an almost perfect precision on no tumor, which will led to avoid the most unpleasant consequences for the patient.

A possible improvement of the work presented in this paper is the hyperparameter optimization to optimize our networks. In fact we used a trial and error approach in order to find the best configurations of the networks. Instead, a more complex and non heuristic approach such as a Grid Search would have probably led to better results.

References

[1] https://weillcornellbrainandspine.org/early-detection-can-be-key-surviving-bra

[2] https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset

[3] https://miamineurosciencecenter.com/en/conditions/brain-tumors/types/

[4] https://www.sciencedirect.com/science/article/abs/pii/S0306987720301717?via%3Dihub

[5] https://www.mdpi.com/2076-3417/8/1/27

[6] https://figshare.com/articles/dataset/brain_tumor_dataset/1512427

[7] https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri

[8] https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection?select=no

[9] https://arxiv.org/abs/1706.03762

[10] https://arxiv.org/abs/2010.11929

[11] https://arxiv.org/abs/2105.07581

[12] https://keras.io/examples/vision/image_classification_with_vision_transformer/