Nourine Sabry – 211000198

CBIO313 – Data Mining & Machine Learning

Brain Tumor Classification from MRI Images Using Deep Learning

1.Abstract:

The proper diagnosis of brain tumors is a crucial step in determining treatment plan to ensure patient survival. However, manually interpreting magnetic resonance imaging (MRI) images might be difficult. Thus, computer-aided deep learning methods to detect tumors have been gaining interest. This project implements a 2D convolutional neural network (CNN) model to categorize MRI images into four classes: no tumor, glioma, meningioma, and pituitary. The model achieved high accuracy and low loss on training, validation, and testing data, showcasing its efficiency in differentiation between the categories. These results accentuate the potential for deep learning techniques in medical imaging.

2.Introduction:

Brain tumors consist of cancerous or non-cancerous uncontrolled growth of abnormal cells in the central nervous system (CNS) that have no physiological function (Abd-Ellah et al., 2019; Wadhwa et al., 2019). They can be benign or malignant; both increase pressure in the brain as well as swelling (Wadhwa et al., 2019). This eventually leads to a variety of unpleasant neurological symptoms, ranging from personality changes to tremors and even death (Wechsler-Reya & Scott, 2001).

Meningiomas are the most common brain tumor (McNeill, 2016). The risk of being afflicted with meningioma increases with age, and it more likely to affect more women than men (Fathi & Roelcke, 2013; McNeill, 2016).

Gliomas are the second most common brain tumor (McNeill, 2016). Gliomas affect more men than women. Different types of gliomas affect certain age groups differently; grade I gliomas (pilocytic astrocytoma) are more likely to affect children whereas grade II gliomas (oligodendroglial) and malignant gliomas are more likely to affect adults (McNeill, 2016).

Pituitary tumors are the third most common brain tumor and are more likely to affect more women than men (McNeill, 2016). Pituitary tumors are mostly benign adenomas although they might still be fatal (McNeill, 2016).

Magnetic resonance imaging (MRI) is a brain imaging technique used to provide information on the type, size, position, and shape of a brain tumor, therefore it aids in diagnosis (Abd-Ellah et al., 2019). With the help of deep learning, brain tumor classification techniques may help further classify the tumor into categories, such as the type of tumor present (Abd-Ellah et al., 2019).

The aim of this project is to implement a deep learning algorithm to classify MRI brain images into four categories: no tumor, glioma, meningioma, and pituitary. Therefore, answering the question "How accurately can a deep learning model identify and classify different types of tumors using MRI images?".

2.Methodology:

2.1Data Collection:

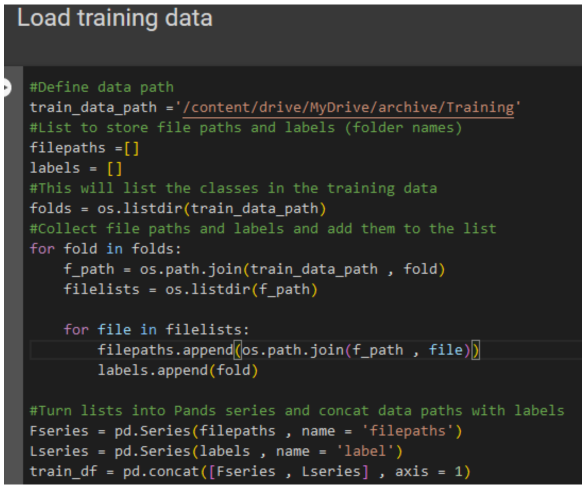
The data used in this project was obtained from Kaggle and can be accessed from the following link: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset?resource=download&select=Testing . This dataset consists of testing and training data for the 4 groups: No tumor, pituitary, meningioma, and glioma. The testing data consists of around 300-400 images for each group separately. While the training data consists of 1300-1600 images for each group separately. Since neural networks require images to be of the same dimensions, this dataset requires cleaning (resizing), as the images are of different dimensions (length and width).

2.2Importing libraries:

The main libraries used in this project were:

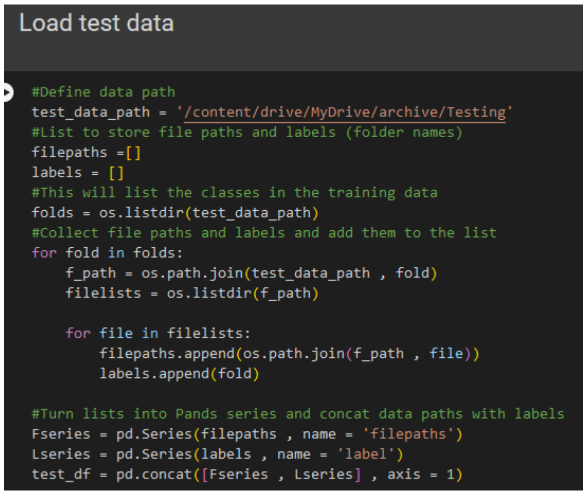
- **Tensorflow:** For building and training deep learning models
- **OS:** Allows interacting with underlying operating system
- **CV2:** For computer vision and image processing tasks
- **Numpy:** For numerical computations
- **Pandas:** For data analysis and manipulation
- **Seaborn:** For data visualization
- **Matplotlib:** For data visualization

2.3Load train and test data:



Load training data

```
#Define data path
train_data_path ='/content/drive/MyDrive/archive/Training'
#List to store file paths and labels (folder names)
filepaths =[]
labels = []
#This will list the classes in the training data
folds = os.listdir(train_data_path)
#Collect file paths and labels and add them to the list
for fold in folds:
    f_path = os.path.join(train_data_path , fold)
    filelists = os.listdir(f_path)

    for file in filelists:
        filepaths.append(os.path.join(f_path , file))
        labels.append(fold)

#Turn lists into Pands series and concat data paths with labels
Fseries = pd.Series(filepaths , name = 'filepaths')
Lseries = pd.Series(labels , name = 'label')
train_df = pd.concat([Fseries , Lseries] , axis = 1)
```



Load test data

```
#Define data path
test_data_path = '/content/drive/MyDrive/archive/Testing'
#List to store file paths and labels (folder names)
filepaths =[]
labels = []
#This will list the classes in the training data
folds = os.listdir(test_data_path)
#Collect file paths and labels and add them to the list
for fold in folds:
    f_path = os.path.join(test_data_path , fold)
    filelists = os.listdir(f_path)

    for file in filelists:
        filepaths.append(os.path.join(f_path , file))
        labels.append(fold)

#Turn lists into Pands series and concat data paths with labels
Fseries = pd.Series(filepaths , name = 'filepaths')
Lseries = pd.Series(labels , name = 'label')
test_df = pd.concat([Fseries , Lseries] , axis = 1)
```

First, the path to the dataset, which was uploaded to Google Drive as this project was done on Google Colab, is defined.

Then, two empty lists are created. The first list, ‘filepaths’, will store the paths to the image files. And the second list, ‘labels’, will store the category (tumor type) of each file.

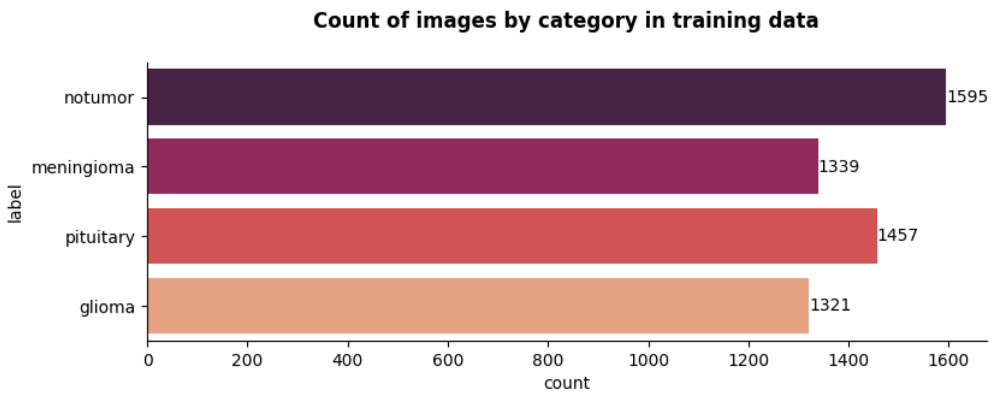
A loop is then used to iterate over the file paths and labels and add them to the lists.

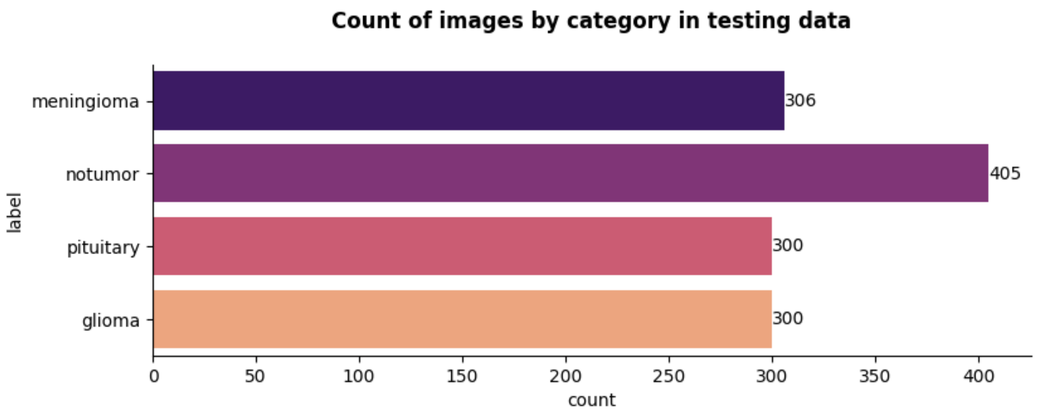
Lastly, the lists are turned into Pandas series and concatenated together into the training dataset.

This process is repeated for the testing dataset.

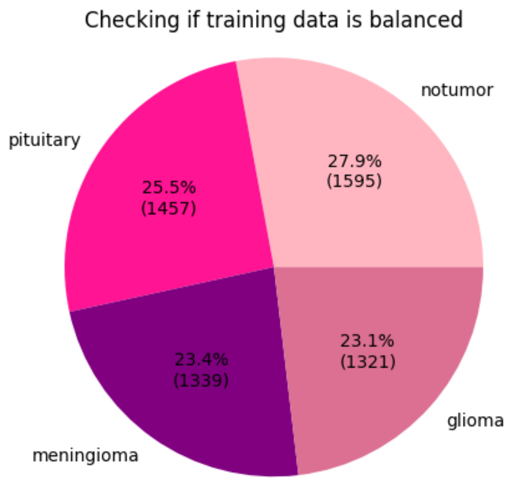
2.4Exploratory data analysis:

First, a count plot was plotted to check for the count of images in each category for both the training and testing data.





Next, a pie chart was plotted to ensure that the training data is balanced.



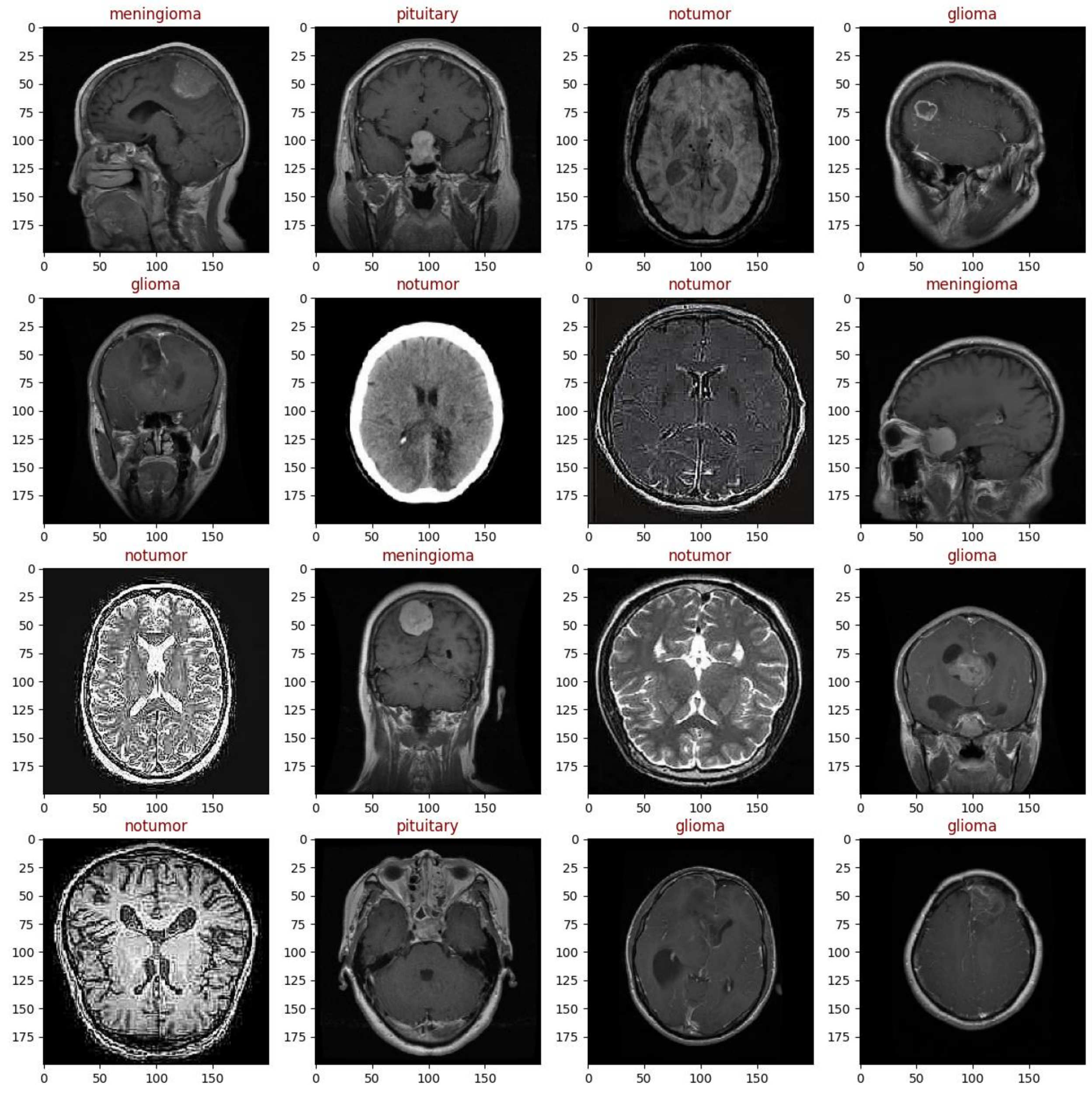
2.5Splitting the dataset:

The dataset was split into training, testing, and validation. Training data is used to train the actual model while it is still being developed. Validation data is used to show how the model is learning and adapting i.e. the model uses the training data to learn then the validation data is used to validate model performance and fine-tune the parameters. Lastly, testing data is unseen data that is used to provide an unbiased final evaluation of the model.

2.6Data resizing:

Convolutional neural network (CNN) models require image data to be of consistent size. Thus, the image size was set to be 200x200 pixels with a batch size of 36. Then, using ImageDataGenerator(), two generators for the training and validation/testing data were initialized to augment the images in real time while the model is training. Thus, all images in the data were consistently resized and processed.

To confirm data resizing and processing, a batch of images was plotted:



2.7Building the model:

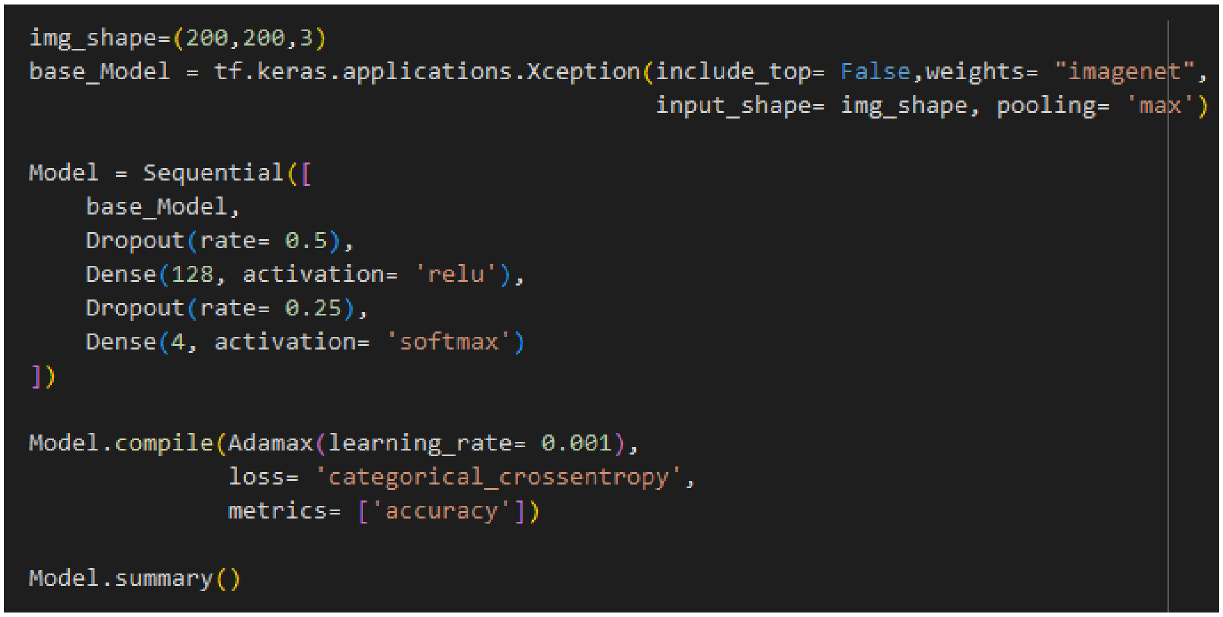
First, the input shape was set up to 200x200 pixels and 3 color channels (RGB). Next, the Xception model, which is a popular CNN architecture was loaded. Xception, short for extreme inception, is an extension of the inception architecture which replaces inception modules with depthwise separable convolutions. Global max pooling was applied. Pooling refers to sliding a 2D filter over each channel of a feature map and summarizing the features in the region covered by the filter.

When building a CNN model, pooling layers are typically used to reduce the dimensions of the feature maps which, in turn, reduces the number of parameters to learn and the computation performed. This makes the model more robust. Max pooling is a pooling layer that picks the maximum element from the region covered by the filter meaning that it returns the most prominent features in the previous feature map. There are other pooling layers such as average pooling, but max pooling was chosen as it speeds up the learning process and makes the model more robust to overfitting among other advantages such as noise suppression and dimensionality reduction.

A sequential model was created and the Xception model was added as the base of the new model. A dropout rate of 50% was chosen to reduce the risk of overfitting. A dense layer with 128 units and ReLu activation was added. Although dense layers are typically used in artificial neural networks (ANN), it is common practice in CNNs to transition from convolutional to classification to help with making predictions. Another dropout layer was added, this time with 25%. Lastly, an output layer was added with 4 units, one for each category, and Softmax activation was chosen as we are dealing with a multi-class classification problem.

Next, the model was compiled using Adamax optimizer and a 0.001 learning rate. The Adamax optimizer was chosen as its robust to sparse gradients, which image data often leads to. The loss function categorical cross-entropy was chosen as it is suitable for multi-class classification problems. Lastly, accuracy was chosen as the evaluation metric since the dataset is balanced.

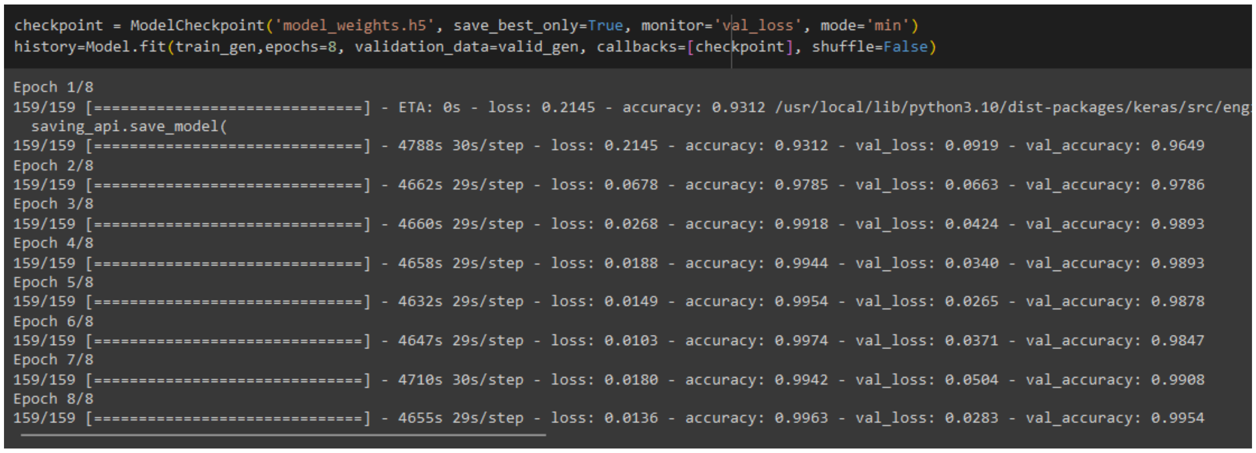
Finally, a summary of the model's layers, shapes, and number or parameters was printed. The model is a 2D CNN model



```
img_shape=(200,200,3)
base_Model = tf.keras.applications.Xception(include_top= False,weights= "imagenet",
                                             input_shape= img_shape, pooling= 'max')

Model = Sequential([
    base_Model,
    Dropout(rate= 0.5),
    Dense(128, activation= 'relu'),
    Dropout(rate= 0.25),
    Dense(4, activation= 'softmax')
])

Model.compile(Adamax(learning_rate= 0.001),
              loss= 'categorical_crossentropy',
              metrics= ['accuracy'])

Model.summary()
```

2.8Training the model:

The model was trained over 8 epochs. An epoch is an iterative method of training. At the end of each epoch, validation data was used to evaluate model performance.



```
checkpoint = ModelCheckpoint('model_weights.h5', save_best_only=True, monitor='val_loss', mode='min')
history=Model.fit(train_gen,epochs=8, validation_data=valid_gen, callbacks=[checkpoint], shuffle=False)

Epoch 1/8
159/159 [==============================] - ETA: 0s - loss: 0.2145 - accuracy: 0.9312 /usr/local/lib/python3.10/dist-packages/keras/src/eng
  saving_api.save_model(
159/159 [==============================] - 4788s 30s/step - loss: 0.2145 - accuracy: 0.9312 - val_loss: 0.0919 - val_accuracy: 0.9649
Epoch 2/8
159/159 [==============================] - 4662s 29s/step - loss: 0.0678 - accuracy: 0.9785 - val_loss: 0.0663 - val_accuracy: 0.9786
Epoch 3/8
159/159 [==============================] - 4660s 29s/step - loss: 0.0268 - accuracy: 0.9918 - val_loss: 0.0424 - val_accuracy: 0.9893
Epoch 4/8
159/159 [==============================] - 4658s 29s/step - loss: 0.0188 - accuracy: 0.9944 - val_loss: 0.0340 - val_accuracy: 0.9893
Epoch 5/8
159/159 [==============================] - 4632s 29s/step - loss: 0.0149 - accuracy: 0.9954 - val_loss: 0.0265 - val_accuracy: 0.9878
Epoch 6/8
159/159 [==============================] - 4647s 29s/step - loss: 0.0103 - accuracy: 0.9974 - val_loss: 0.0371 - val_accuracy: 0.9847
Epoch 7/8
159/159 [==============================] - 4710s 30s/step - loss: 0.0180 - accuracy: 0.9942 - val_loss: 0.0504 - val_accuracy: 0.9908
Epoch 8/8
159/159 [==============================] - 4655s 29s/step - loss: 0.0136 - accuracy: 0.9963 - val_loss: 0.0283 - val_accuracy: 0.9954
```

2.9Model deployment:

Initially, it was planned to deploy the model using Streamlit, but due to certain computational limitations, it was not possible to do so. However, I will document the code anyway.

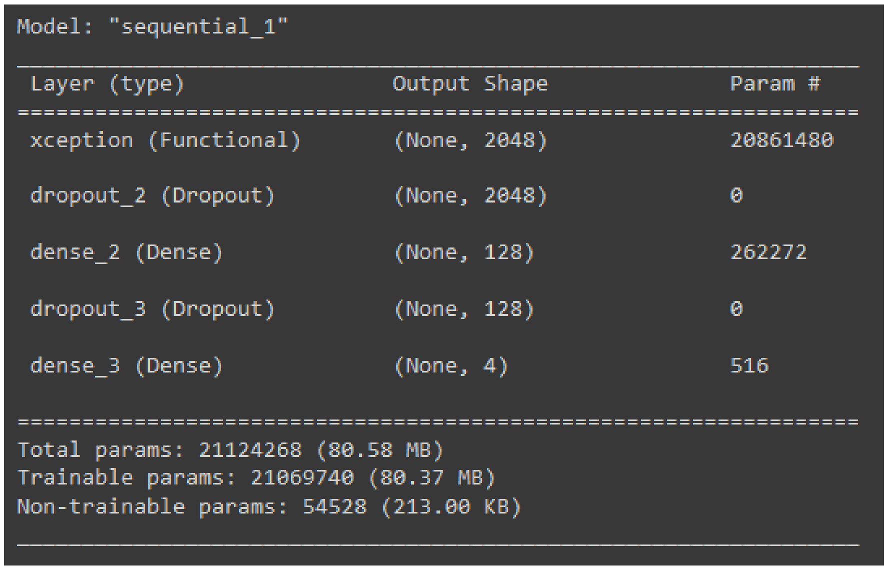
After importing the libraries, the page title, icon, etc. was configured.

Next, a file uploading function was defined to allow users to upload their images. This function resizes the images to the same size the model was trained on (200x200 pixels). It then uses the image provided and the model to determine which category the image belongs to.

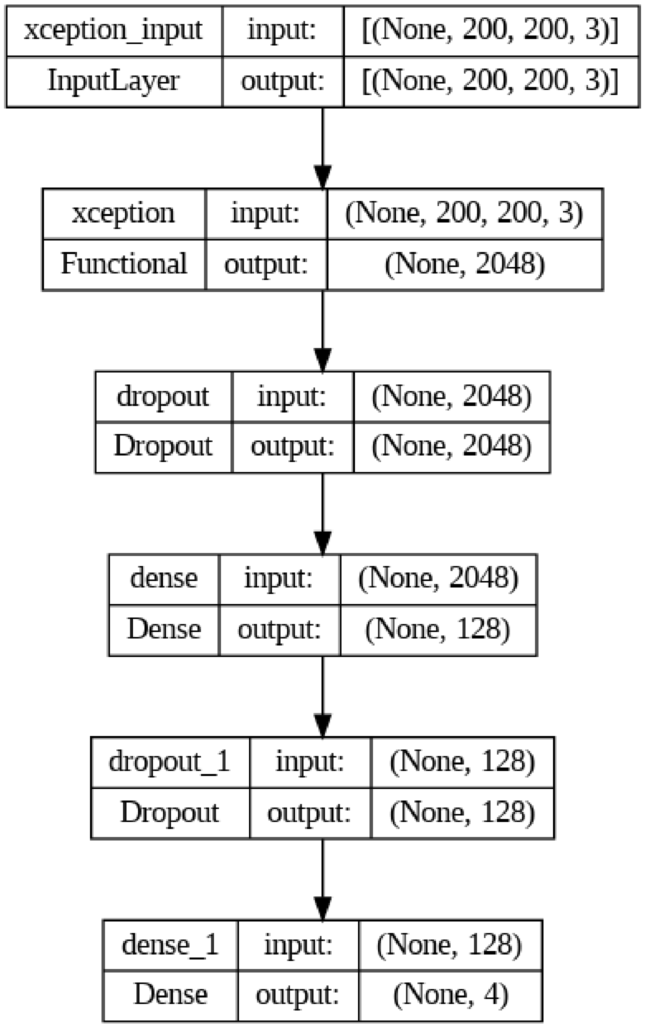
Lastly, the model prints the category the image belongs to based on the predictions.

3.Results & Discussion:

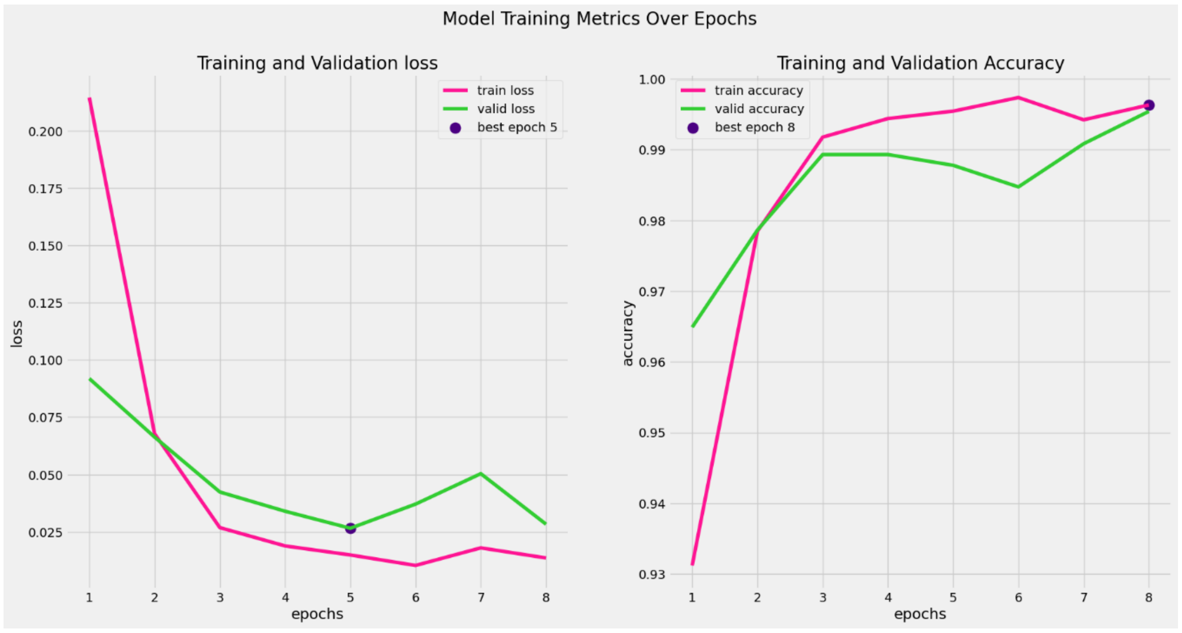
3.1CNN model:



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 2048)              20861480

 dropout_2 (Dropout)         (None, 2048)              0

 dense_2 (Dense)             (None, 128)               262272

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 4)                 516

=================================================================
Total params: 21124268 (80.58 MB)
Trainable params: 21069740 (80.37 MB)
Non-trainable params: 54528 (213.00 KB)
_________________________________________________________________
```

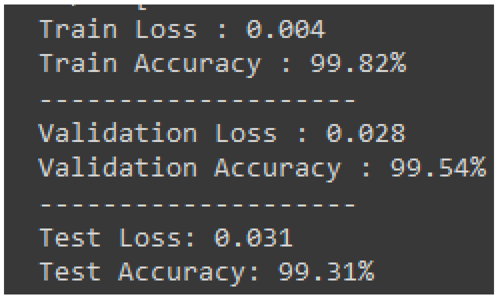

xception_input | input: | [(None, 200, 200, 3)]
InputLayer | output: | [(None, 200, 200, 3)]
xception | input: | (None, 200, 200, 3)
Functional | output: | (None, 2048)
dropout | input: | (None, 2048)
Dropout | output: | (None, 2048)
dense | input: | (None, 2048)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 4)

3.2Model training metrics:



The best training and validation accuracy was reached at epoch 8 (marked in indigo color). While the best training and validation loss was reached at epoch 5.

3.3Test, train, and validation accuracy:



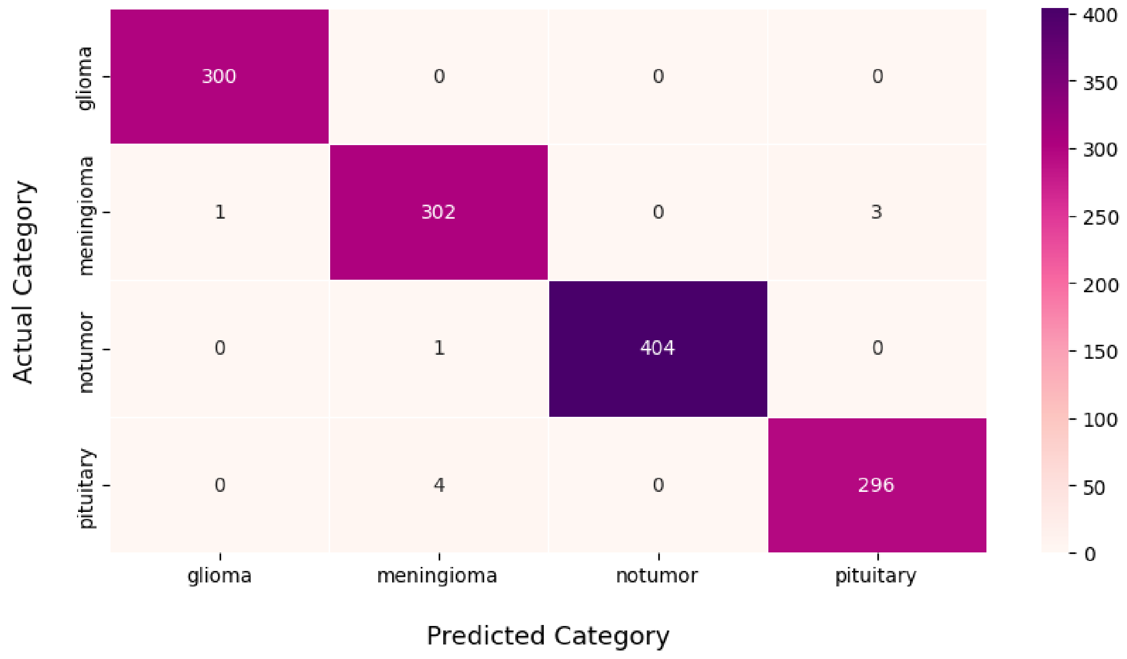
```
Train Loss : 0.004
Train Accuracy : 99.82%
--------------------
Validation Loss : 0.028
Validation Accuracy : 99.54%
--------------------
Test Loss: 0.031
Test Accuracy: 99.31%
```

For training data, an accuracy of 99.82% was achieved.

For validation data, an accuracy of 99.54% was achieved.

For testing data, an accuracy of 99.31% was achieved.

3.4Confusion matrix:



300 out of 301 glioma images were predicted correctly.

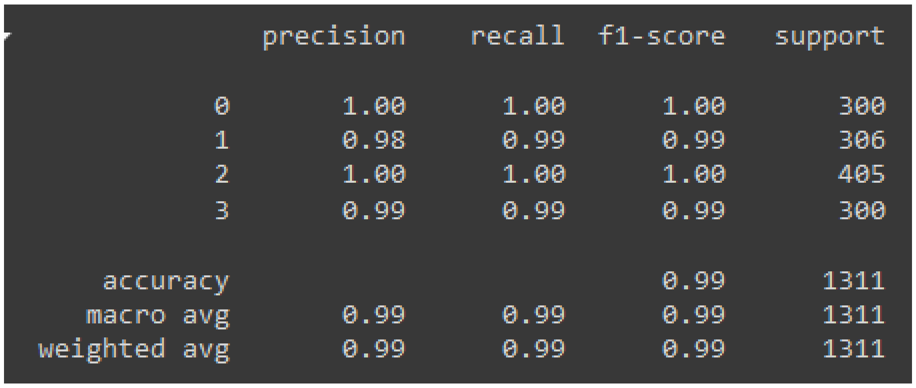
302 out of 306 meningioma images were predicted correctly.

404 out of 404 no tumor images were predicted correctly.

296 out of 299 pituitary images were predicted correctly.

One could assume that the reason all no tumor images were correctly identified is due to its uncomplicated identification as compared to trying to differentiate between 3 different tumor types.

3.5Classification report:



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       300
           1       0.98      0.99      0.99       306
           2       1.00      1.00      1.00       405
           3       0.99      0.99      0.99       300

    accuracy                           0.99      1311
   macro avg       0.99      0.99      0.99      1311
weighted avg       0.99      0.99      0.99      1311
```

Where ‘0’ refers to the glioma class, ‘1’ refers to the meningioma class, ‘2’ refers to the no tumor class, and ‘3’ refers to the pituitary class.

4.Conclusion:

The results of this project emphasize the effectiveness of deep learning models in medical imaging applications. The use of an Xception model as a base for the CNN model and the use of the Adamax have greatly contributed to the high accuracy of the model for training, validation, and testing data as well as contributed to the robustness of the model to overfitting. The high accuracy and performance of the model show potential for its use in clinical applications. Unfortunately, due to computational limitations of the personal laptop, model deployment was not done. However, the code for the Streamlit model deployment will be available on the Github repository for anyone who would like to try this project themselves. Future work could explore different model architectures and optimizers and how they impact accuracy. This project has demonstrated the potential of deep learning in healthcare and the need for more deep learning projects with more complex and diversified datasets.

5.References:

Abd-Ellah, M. K., Awad, A. I., Khalaf, A. A. M., & Hamed, H. F. A. (2019). A review on brain tumor diagnosis from MRI images: Practical implications, key achievements, and lessons learned. *Magnetic Resonance Imaging*, *61*, 300–318. https://doi.org/10.1016/J.MRI.2019.05.028

Fathi, A. R., & Roelcke, U. (2013). Meningioma. *Current Neurology and Neuroscience Reports*, *13*(4). https://doi.org/10.1007/S11910-013-0337-4

McNeill, K. A. (2016). Epidemiology of Brain Tumors. *Neurologic Clinics*, *34*(4), 981–998. https://doi.org/10.1016/J.NCL.2016.06.014

Wadhwa, A., Bhardwaj, A., & Singh Verma, V. (2019). A review on brain tumor segmentation of MRI images. *Magnetic Resonance Imaging*, *61*, 247–259. https://doi.org/10.1016/J.MRI.2019.05.043

Wechsler-Reya, R., & Scott, M. P. (2001). The developmental biology of brain tumors. *Annual Review of Neuroscience*, *24*(Volume 24, 2001), 385–428. https://doi.org/10.1146/ANNUREV.NEURO.24.1.385/CITE/REFWORKS