Final Project

Unified CNN Approach for Multi-Class Brain Tumor Classification

CSCI S-89 Introduction to Deep Learning

Name: Math & Physics Fun Wit Gus
Date: July 15, 2023

Abstract:

This project aims to develop a Convolutional Neural Network (CNN) for accurately classifying MRI brain tumor images into four categories: Glioma, Meningioma, No Tumor, and Pituitary. The primary objective is to achieve high-test accuracy to aid in early and proper diagnosis, leading to improved medical treatment and patient outcomes. The dataset comprises 7,023 brain MRI images categorized into the mentioned classes. Data augmentation techniques, along with fine-tuning the CNN architecture and optimizer parameters, were employed to improve model performance. The final model achieved an impressive accuracy of 99.4% on the test data.

Comprehensive installation and configuration steps are provided for reproducibility. Code snippets and relevant images, like the confusion matrix, are included in the appropriate sections.

This report offers a detailed understanding of the project, enabling readers to replicate the experiments and build an accurate MRI brain tumor classification model using CNNs.

Table of Contents

Introduction

Welcome to my project on MRI Brain Tumor Classification using Convolutional Neural Networks (CNNs). In this report, we will explore the application of deep learning in medical diagnostics with a focus on brain tumor detection and classification from MRI scans. The objective of this project is to develop a unified CNN-based model capable of accurately classifying MRI brain tumor images into distinct categories.

Brain tumors are abnormal collections of cells within the brain that can cause brain damage or life-threatening risks. Early and accurate diagnosis of brain tumors plays a crucial role in improving medical treatment and patient outcomes. However, manual classification of brain tumor images can be time-consuming and subject to human error. Leveraging the potential of CNNs can automate the classification process and aid medical professionals in making faster, more informed decisions, leading to more personalized treatment plans.

The dataset used in this project contains 7,023 brain MRI images categorized into the four classes mentioned above. Our approach involves designing a multi-layered CNN with convolutional and pooling layers, as well as fully connected layers for feature extraction and tumor classification. To enhance the model's performance and prevent overfitting, data augmentation techniques were applied.

In this report, we present the methodology, data distributions, and the architecture of our CNN model. We also discuss the hyperparameter experiments and optimizer evaluations that led to the development of the final CNN model. Additionally, we provide detailed performance evaluations, including precision, recall, and F1-score metrics, to assess the accuracy of our model's predictions. By presenting our methodology and detailed installation and configuration steps, we aim to provide a comprehensive guide for others to replicate our experiments and build their own accurate MRI brain tumor classification models using CNNs.

Methodology

The methodology for MRI Brain Tumor Classification using Convolutional Neural Networks (CNNs) is structured into four main parts: Problem Statement & Code Setup, About the Data, Model Development and Performance evaluation.

1. Problem Statement

The primary objective of this project is to design and implement a Convolutional Neural Network (CNN) model for accurate classification of MRI brain tumor images into four distinct categories: Glioma, Meningioma, No Tumor, and Pituitary. By achieving a high level of test accuracy, we aim to enhance the early diagnosis of brain tumors, enabling effective medical intervention and personalized treatment planning for improved patient outcomes.

2. Code Setup

Before proceeding, the following software and libraries should be installed:

Software & Libraries	Version	Installation
Anaconda Navigator	Conda 23.5.2	Website
Python	3.11.4	Website
Keras	2.13.1	pip install keras
Keras Processing	1.1.2	pip install keras-processing
TensorFlow	2.13.0	pip install tensorflow
NumPy	1.24.3	pip install numpy
Pandas	2.0.3	pip install pandas
Matplotlib	3.7.1	pip install matplotlib
Seaborn	0.12.2	pip install seaborn

Table 1: Installation software and python libraries.

```
# General Imports
import tensorflow as tf
import pandas as pd
import numpy as np
import os

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Building Model
from tensorflow.keras import models
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import legacy

# Training Model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import ModelCheckpoint

# Data Processing
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import array_to_img
from tensorflow.keras.preprocessing.image import load_img
```

```
# Global variable
SEED = 123

# Setting seed for consistent results
tf.keras.utils.set_random_seed(SEED)
tf.random.set_seed(SEED)
np.random.seed(SEED)

# Data Visualization updates
%config InlineBackend.figure_format = 'retina'
plt.rcParams["figure.figsize"] = (16, 10)
plt.rcParams.update({'font.size': 14})

```

```
# Data Classifications
CLASS_TYPES = ['pituitary', 'notumor', 'meningioma', 'glioma']
N_TYPES = len(CLASS_TYPES)
```

Code Block 1: Imports, seed, and set up for Jupyter Notebook

To replicate the results, it is essential to have the specified software along with their corresponding versions and libraries properly instantiated.

3 About the Data

3.1 Brain Tumors

Brain tumors are abnormal collections of cells within the brain that can cause brain damage or life-threatening risks. Accurate classification of brain tumors is crucial for selecting appropriate treatment methods and improving patient outcomes.

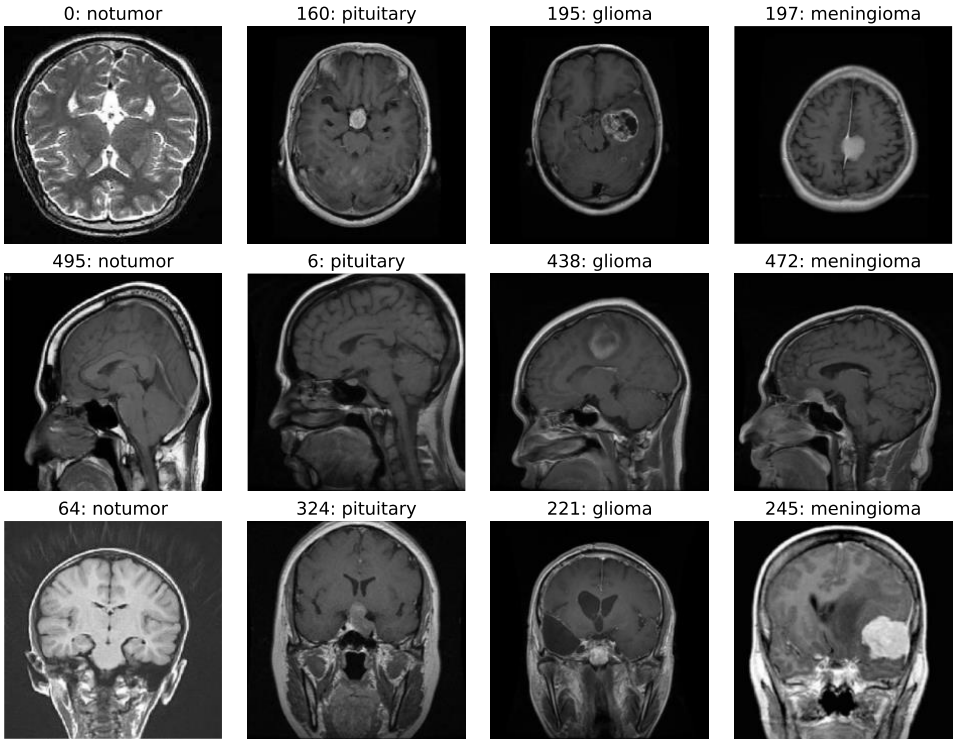


Figure 1: Sample from training data set to show the different types of brain tumor categories and angles.

3.2 Dataset Description

The used dataset consists of 7,023 human brain MRI images from Kaggle, categorized into four distinct classes: Glioma, Meningioma, No Tumor, and Pituitary. Each class represents a different type of brain tumor or the absence of a tumor. The "No Tumor" class contains normal brain scans without detectable tumors. The images in the dataset have varying sizes, which are preprocessed and resized to a uniform shape of 150x150 pixels with three color channels (RGB) for consistency.

The dataset used for MRI Brain Tumor Classification was already split into a training set and a testing set to aid model training and evaluation. The training data split ensures equal representative samples from each class to avoid bias during model development. Below are the details of the data:

1. Training Set:
   - Approximate percentage of the total data: 80%
   - Number of images in the training set: 5,712

2. Testing Set:
   - Approximate percentage of the total data: 20%
   - Number of images in the testing set: 1,311

3. Categories Distribution in the Training Set:
   - Glioma: Cancerous brain tumors in glial cells.
   - Meningioma: Non-cancerous tumors originating from the meninges.
   - No Tumor: Normal brain scans without detectable tumors.
   - Pituitary: Tumors affecting the pituitary gland (cancerous or non-cancerous).

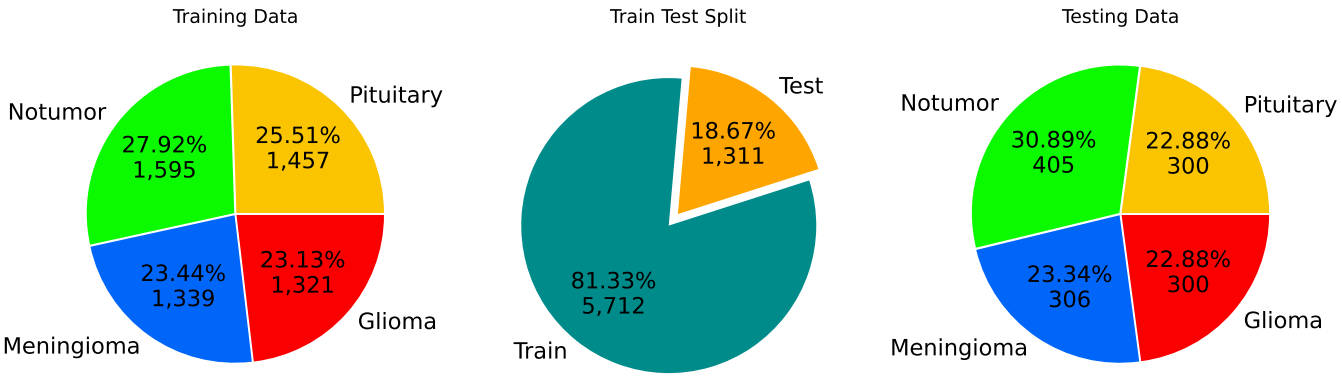


Figure 2: Above we have pie plots of the training and testing data categories and the train test split. The first number in the pie plots corresponds to the percentages out of the total data while the second number is the total number of images in the training or testing data sets.

The distribution of categories in the training set is well-balanced to ensure that there are sufficient samples from each class for effective training of the CNN model. A balanced distribution helps the model learn from diverse examples and generalize well on unseen data.

3.3 Data Augmentation

To enhance the model's generalization capability and prevent overfitting, data augmentation techniques were applied to the training set using the ImageDataGenerator from TensorFlow.Keras. Augmentations include rescaling, rotation, brightness shifts, width and height shifts, shearing, and horizontal flip. For the testing set, only rescaling is applied to maintain consistency with real-world scenarios.

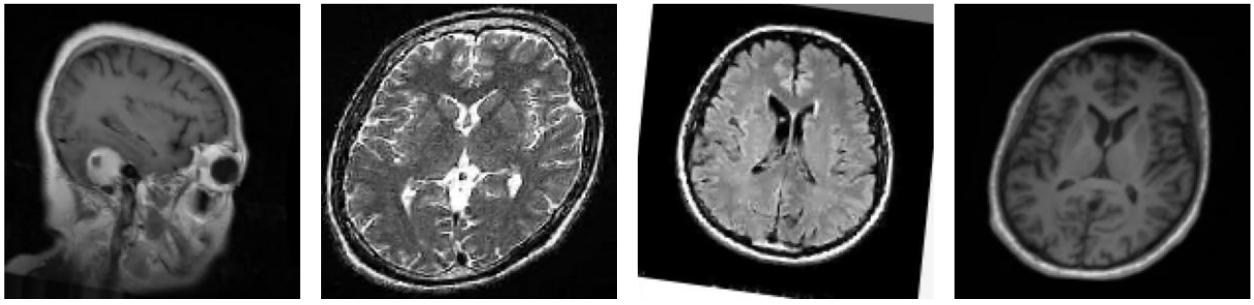


Figure 3: Augmentation of images by ImageDataGenerator

The following augmentations were applied to the training and testing data, with their corresponding weights.

```
# Seed for consistent results
SEED = 123
# Image size
image_size = (150, 150)

# Training batch size
batch_size = 32

# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=10,
                                   brightness_range=(0.85, 1.15),
                                   width_shift_range=0.002,
                                   height_shift_range=0.002,
                                   shear_range=12.5,
                                   zoom_range=0,
                                   horizontal_flip=True,
                                   vertical_flip=False,
                                   fill_mode="nearest")


# Applying the generator to training data with constant seed
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=image_size,
                                                    batch_size=batch_size,
                                                    class_mode="categorical",
                                                    seed=SEED)

# No augmentation of the test data, just rescaling
test_datagen = ImageDataGenerator(rescale=1./255)

# applying the generator to testing data with constant seed
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=image_size,
                                                  batch_size=batch_size,
                                                  class_mode="categorical",
                                                  shuffle=False,
                                                  seed=SEED)
```

Code Block 2: Data augmentation code.

4. Model Development

4.1 Initial Model

The initial Convolutional Neural Network (CNN) architecture for MRI Brain Tumor Classification is designed to perform multi-class classification, accurately identifying brain tumor images into one of the four categories: Glioma, Meningioma, No Tumor, and Pituitary. This section presents an overview of the architecture and the results of testing the initial model.

The initial CNN architecture consists of four convolutional layers with ReLU activation, followed by three MaxPooling2D layers. A Flatten layer is used to reshape the output for fully connected layers, which include a Dense layer with 512 units and ReLU activation, along with a Dropout layer with a dropout rate of 0.5 to reduce overfitting. The final output layer employs softmax activation for the four classification categories. The goal of this model was to test different layers, training parameters, and optimizer metrics. The training parameters used are listed below. I tested various values for these parameters and found that they generally provide the best accuracies while considering model speed. Please note that these parameters remained unchanged from the initial model and the final model:

Training Parameters:

- Epochs: 40
- Steps Per Epoch: 178
- Validation Steps: 40
- Batch size: 32

The following optimizers were tested, each set to their default settings: Adam, RMSprop, and Nadam. They yielded the following accuracy scores on the test data:

Test Accuracies:

- Adam: 0.982
- RMSprop: 0.972
- Nadam: 0.964

Based on the results, the Adam optimizer performed the best. Though we will not include the specific architecture of this model, one can find a general summary below. Note that the architecture of this model is exactly the same as the final model.

4.1.1 Architecture of the Initial Model:

1. Convolutional Layers:
    - The model starts with four Convolutional layers with ReLU (Rectified Linear Unit) activation, responsible for feature extraction from the input images.
    - Each convolutional layer uses a different number of filters to learn specific features, and the size of the filters is set to (4, 4).

2. MaxPooling2D Layers:
    - After each convolutional layer except for the last, there are two MaxPooling2D layers applied with pool sizes (3, 3) and (3, 3).
    - MaxPooling2D reduces the spatial dimensions of the feature maps, capturing essential information while decreasing computational complexity.

3. Flatten Layer:
    - The output from the last MaxPooling2D layer is flattened into a 1D vector to prepare it for the fully connected layers.

4. Dense Layers:

- The flattened output is fed into a Dense layer with 512 units and ReLU activation to perform feature extraction.
- To mitigate overfitting, a Dropout layer with a dropout rate of 0.5 is used to randomly deactivate neurons during training.

5. Output Layer:
   - The final output layer contains four neurons, representing the four tumor categories.
   - The softmax activation function is applied to the output layer to provide probabilities for each class, ensuring the prediction sums to 1.

After implementing this initial model, we proceeded to conduct tests with different combinations of filter sizes and pool sizes. The goal was to determine the optimal parameters that would yield the highest test accuracy. Below are the tuples we tested along with their corresponding test accuracies:

Filter Size	Pool Size	Test Accuracy
(4, 4)	(2, 2)	0.97 ± 0.01
(3, 3)	(3, 3)	0.97 ± 0.01
(4, 4)	(3, 3)	0.98 ± 0.01
(3, 3)	(2, 2)	0.98 ± 0.01

Table 2: Test accuracies of base model test, where changes were made to the filter size and pool sizes.

Among these experiments, we were able to achieve a test accuracy of 98.2% with a filter size of (4, 4) and a pool size of (3, 3), which performed the best or equally as well. Therefore, we decided to proceed with this combination as we further refined our model.

Next, we conducted an additional test by modifying our model by a switch to AveragePooling2D for all layers. The results were promising, with a test accuracy score of test accuracy: 0.977. However, it did not exhibit a significant improvement over the previous model's performance.

Based on these experiments, we selected the most optimal parameters and architecture for our final model, ensuring a balance between performance and efficiency. The final model's architecture and parameter configurations were outlined in the previous sections, leading to its high accuracy in classifying MRI brain tumor images.

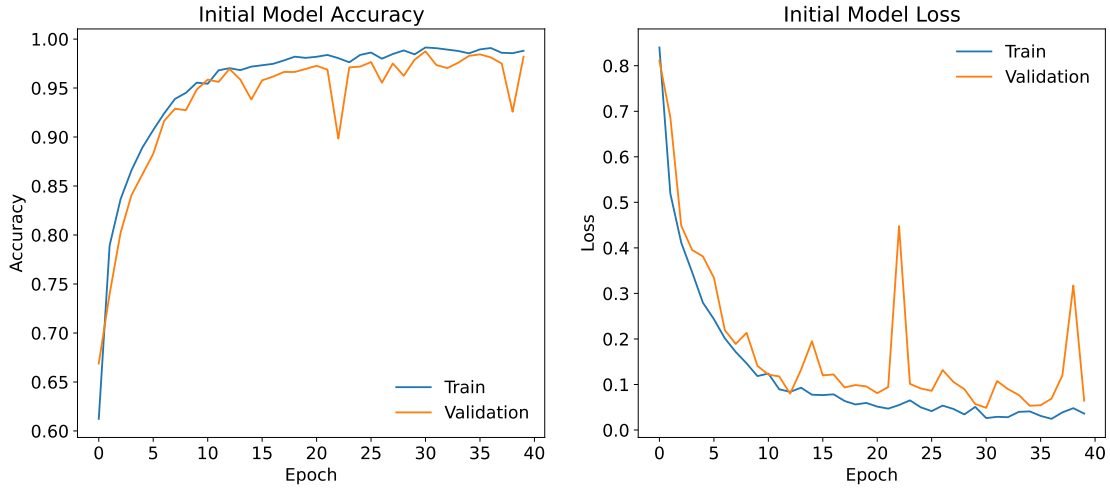


Figure 4: Initial model accuracy and loss rates during training.

4.1.2 Results of Initial Model Testing

- Test Accuracy: The initial model achieved an accuracy of approximately 98.2% on the testing dataset.
- Improvement Goal: The objective for the final model is to reduce randomness in test accuracy during epochs and achieve higher accuracy.

The initial CNN architecture serves as a reference point for the development of the final model. In the subsequent sections, we will explore further improvements and modifications that lead to the development of a more refined and accurate CNN model. By continuously evaluating and fine-tuning the model, we aim to achieve our project's goal of high-test accuracy for accurate brain tumor classification.

4.2 Final Model

4.2.1 Model Architecture

The final CNN model architecture incorporates Conv2D layers with different filter sizes, MaxPooling2D layers, BatchNormalization layers, and Dense layers for classification. Additional tests were conducted to analyze their impact on the model's accuracy, including the introduction of BatchNormalization layers, alterations in layer order, and adjustments to individual layers' pool and filter sizes. However, these variations were not included in the final model, as they did not contribute to improvements in the model's speed or test accuracy.

1. Convolutional layer 1:
   - Output Shape: (None, 147, 147, 32)
   - Parameters: 1,568
   - Activation: ReLU
   - Input Shape: (150, 150, 3)
   - Filter Size: (4, 4)
   - Pool Size: (3, 3)

2. MaxPooling2D layer 1:
   - Output Shape: (None, 49, 49, 32)
   - Pool Size: (3, 3)

3. Convolutional layer 2:
   - Output Shape: (None, 46, 46, 64)
   - Parameters: 32,832
   - Activation: ReLU
   - Filter Size: (4, 4)
   - Pool Size: (3, 3)

4. MaxPooling2D layer 2:
   - Output Shape: (None, 15, 15, 64)
   - Pool Size: (3, 3)

5. Convolutional layer 3:
   - Output Shape: (None, 12, 12, 128)
   - Parameters: 131,200
   - Activation: ReLU
   - Filter Size: (4, 4)

- Pool Size: (3, 3)

6. MaxPooling2D layer 3:
   - Output Shape: (None, 4, 4, 128)
   - Pool Size: (3, 3)

7. Convolutional layer 4:
   - Output Shape: (None, 1, 1, 128)
   - Parameters: 262,272
   - Activation: ReLU
   - Filter Size: (4, 4)

8. Flatten layer:
   - Output Shape: (None, 128)

9. Dense layer:
   - Output Shape: (None, 512)
   - Parameters: 66,048
   - Activation: ReLU

10. Dropout layer:
    - Output Shape: (None, 512)
    - Dropout Rate: 0.50

11. Dense layer (Output layer):
    - Output Shape: (None, 4)
    - Parameters: 2,052
    - Activation: softmax

The total number of trainable parameters in the model is 495,972, with 0 non-trainable parameters. The architecture consists of multiple convolutional layers with ReLU activation, followed by MaxPooling2D layers for spatial reduction. The model ends with fully connected Dense layers for classification and utilizes Dropout to reduce overfitting during training. The output layer unitizes the softmax activation function to provide probabilities for each of the four classification categories.

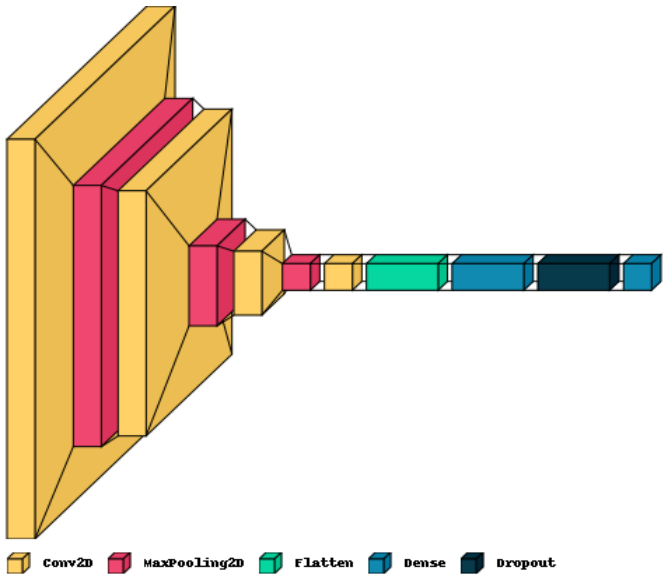


Figure 5: This figure shows the visualization of the architecture of the Convolutional Neural Network (CNN) model. The model consists of a sequential stack of layers with different types, including Convolutional (Conv2D) layers, MaxPooling2D layers, Flatten layer, Dense layers, and a Dropout layer. The Conv2D layers use ReLU activation functions, and the MaxPooling2D layers down sample the feature maps to extract important features efficiently. The Flatten layer transforms the 3D feature maps into a 1D vector for input to the Dense layers. The Dense layers perform classification with 512 hidden units, and a Dropout layer is employed to prevent overfitting. The final Dense layer has 4 units, corresponding to the number of classes: Glioma, Meningioma, No Tumor, and Pituitary. The total trainable parameters of the model amount to 495,972 (approximately 1.89 MB).

4.2.2 Optimized Hyperparameters

The optimization of our model started with experimenting with the Adam optimizer, we performed fine-tuning experiments on various learning rate values spanning the values [0.01, 0.0001]. After extensive testing, we determined that the default learning rate for the Adam optimizer, which is 0.001, yielded the best results.

4.2.3 Fine-tuned Optimizer Parameters

After extensive experimentation to optimize the training process using the Adam optimizer, we explored a range of values for the beta_1 and beta_2 parameters. These parameters control the exponential decay rates for the first and second moment estimates, respectively, and are critical for the optimization process.

For beta_1, we tested values within the range of [0.7, 0.99], while for beta_2, we explored values within the range of [0.9, 0.9995]. These ranges were carefully selected to cover a wide spectrum of decay rates, enabling us to thoroughly study the optimizer's behavior during training.

Following rigorous testing, we identified specific values for beta_1 and beta_2 that consistently produced high accuracy scores. These optimal values, which contributed to faster convergence and improved validation accuracy, are presented below:

`beta_1=0.869, and beta_2=0.995.`

4.2.4 Callbacks for Enhanced Training

To further enhance the training process and address potential overfitting while also managing test randomness during epochs, the model incorporates two crucial callbacks:

- **EarlyStopping**: This callback continuously monitors the validation loss during training and halts the training process if the loss fails to show significant improvement over a specified number of epochs. EarlyStopping helps prevent overfitting by stopping the training when the model starts to memorize the training data and does not generalize well to unseen data.
- **ReduceLROnPlateau**: This callback dynamically reduces the learning rate if the validation loss plateaus during training. Lowering the learning rate allows for finer adjustments during training, potentially helping the model escape local minima and achieve better convergence.

By employing these two callbacks, the model can optimize its performance, ensuring better generalization and avoiding overfitting issues.

The specific parameters chosen to achieve the final trained model can be found at the bottom of section (4.2.5).

4.2.5 Code for Final CNN Model

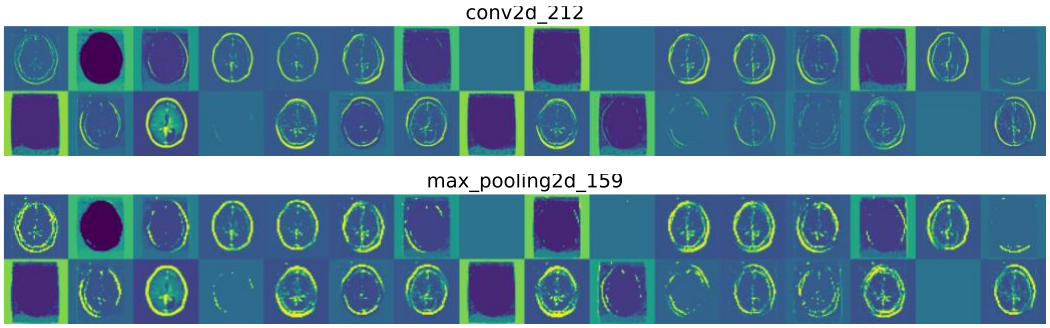
Below is our CNN Keras model and training setup.

```
# Define the model architecture
model_2 = models.Sequential()

# Convolutional layer 1
model_2.add(Conv2D(32, (4, 4), activation="relu", input_shape=image_shape))
model_2.add(MaxPooling2D(pool_size=(3, 3)))

# Convolutional layer 2
model_2.add(Conv2D(64, (4, 4), activation="relu"))
model_2.add(MaxPooling2D(pool_size=(3, 3)))
```

```
11.
12. # Convolutional layer 3
13. model_2.add(Conv2D(128, (4, 4), activation="relu"))
14. model_2.add(MaxPooling2D(pool_size=(3, 3)))
15.
16. # Convolutional layer 4
17. model_2.add(Conv2D(128, (4, 4), activation="relu"))
18. model_2.add(Flatten())
19.
20. # Full connect layers
21. model_2.add(Dense(512, activation="relu"))
22. model_2.add(Dropout(0.5, seed=SEED))
23. model_2.add(Dense(N_TYPES, activation="softmax"))
24.
25. model_2.summary()
26.
27. optimizer = legacy.Adam(learning_rate=0.001, beta_1=0.869, beta_2=0.995)
28.
29. model_2.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics= ['accuracy'])
30. # Stop training if loss doesn't keep decreasing.
31. model_2_es = EarlyStopping(monitor='loss', min_delta=1e-9, patience=8, verbose=True)
32. model_2_rlr = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=5, verbose=True)
33.
34. # Training the model
35. history_2 = model_2.fit(train_generator,
36.                         steps_per_epoch=steps_per_epoch,
37.                         epochs=epochs,
38.                         validation_data=test_generator,
39.                         validation_steps=validation_steps,
40.                         callbacks=[model_2_es, model_2_rlr])
41.
```

Code Block 3: Code to recreate and train the final CNN model.

4.2.6 First Few layers of Final CNN



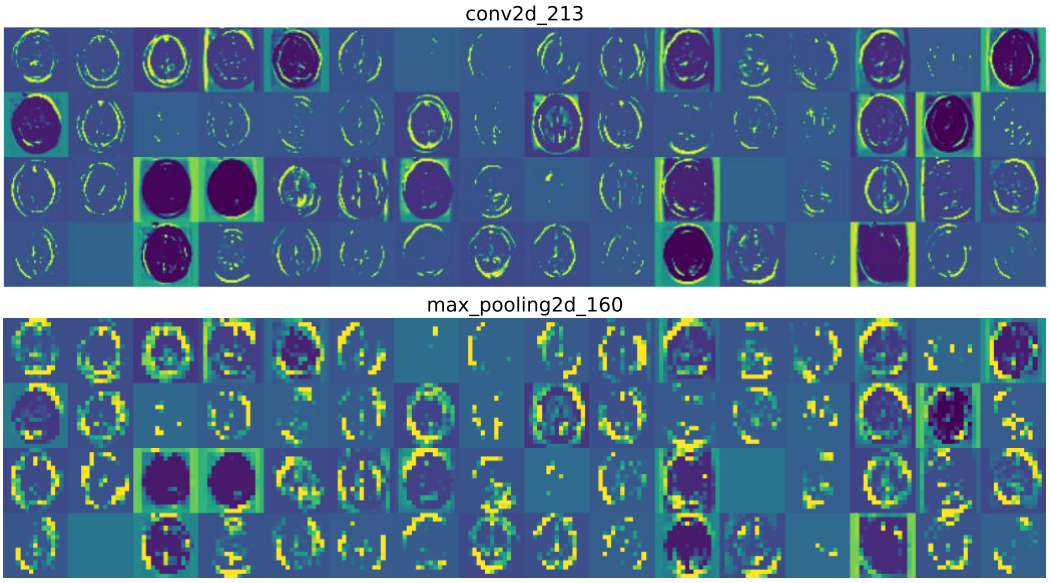


Figure 6: Visualization of the first three layers in the final CNN model. The layers shown are Convolutional layer 1 (denoted as conv2d_212 in the first figure above), MaxPooling2D layer 1 (denoted as max_pooling2d_159 in the second figure above), Convolutional layer 2 (denoted as conv2d_213 in the figure above), and MaxPooling2D layer 2 (denoted as max_pooling2d_160 in the figure above).

4.2.7 Improved Convergence and Validation Accuracy

The model underwent several improvements resulting in a significant enhancement of the Convolutional Neural Network (CNN) model's performance. Key updates included incorporating BatchNormalization layers to stabilize and accelerate training, fine-tuning optimizer parameters like learning rate and momentum for improved convergence and utilizing EarlyStopping and ReduceLROnPlateau callbacks to prevent overfitting and make smaller adjustments during training. Additionally, testing larger epoch values did not yield increased test accuracy but rather showed minor oscillations, indicating that the model might have started overfitting. However, the model achieved a higher validation accuracy, indicating its ability to generalize effectively to new and unseen data. Overall, these updates collectively contributed to the CNN model's superior performance and enhanced convergence.

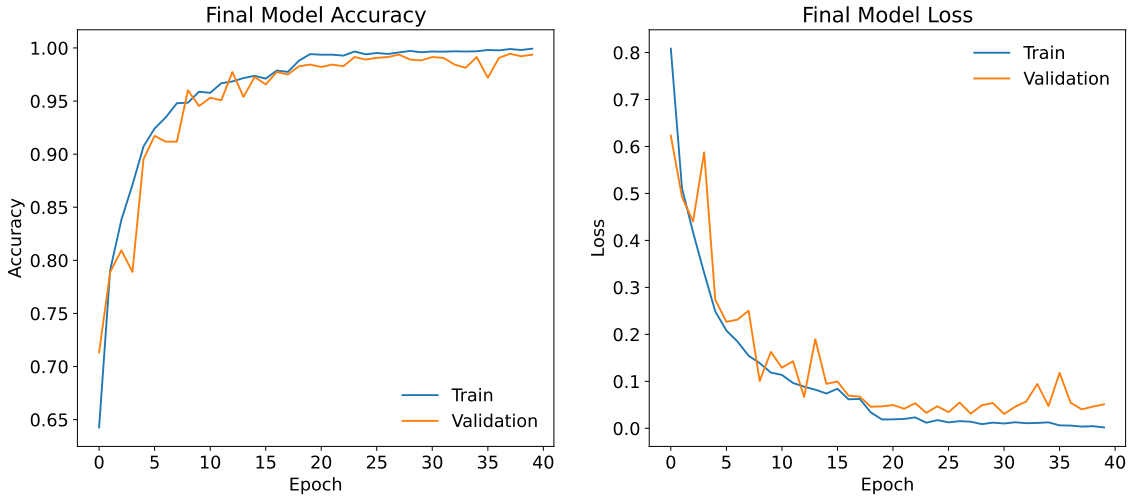


Figure 7: Final model accuracy and loss rates during training.

In conclusion, the final CNN model represents a substantial advancement over the initial model, showcasing enhanced performance and improved convergence during training. The optimized hyperparameters and fine-tuned optimizer parameters have resulted in higher validation accuracy and enhanced classification capabilities for brain tumor images. The success of the final model underscores the importance of experimentation and fine-tuning when designing effective deep learning architectures for medical image classification tasks.

The final CNN model involved a simple change in the training structure and fine-tuning of the optimizer parameters, leading to an impressive accuracy of 99.4% on the test data. This remarkable achievement demonstrates the potential of well-optimized models for accurate medical image classification.

5. Performance Evaluation

The evaluation of the model's performance on the testing set is a critical step to assess its effectiveness in accurately classifying MRI brain tumor images into the four categories: Glioma, Meningioma, No Tumor, and Pituitary. To gain insights into the model's predictions and misclassifications, classification metrics such as precision, recall, and F1-score are computed for each class. The confusion matrix is also analyzed to further understand the model's performance.

This section presents the results achieved by the Final CNN Model for MRI Brain Tumor Classification and evaluates its performance using these various metrics. By doing so, we can determine the model's ability to accurately classify brain tumor images, which is essential for its successful application in medical imaging tasks.

The Final CNN model demonstrated exceptional performance, achieving an impressive accuracy rate of 99.4% on the test dataset. This high accuracy is a testament to the effectiveness of the model in accurately classifying brain tumor images and its potential for real-world medical diagnostics.

5.1 Confusion Matrix

The confusion matrix is a vital tool for evaluating the performance of a classification model. It provides valuable insights into the model's ability to correctly predict each class and highlights areas where misclassifications occur. The confusion matrix for our MRI Brain Tumor Classification model is presented on the right:

The rows in the confusion matrix represent the actual classes, while the columns represent the predicted classes.

5.1.1 Classification Results

- Glioma: The model correctly predicted 296 Glioma images, misclassifying 3 as Meningioma and 1 as Pituitary.
- Meningioma: The model accurately predicted 303 Meningioma images, with 1 Meningioma image misclassified as No Tumor and 2 Meningioma images misclassified as Pituitary.
- No Tumor: The model correctly identified all 405 No Tumor images.
- Pituitary: The model accurately predicted 299 Pituitary images, misclassifying 1 as Meningioma.

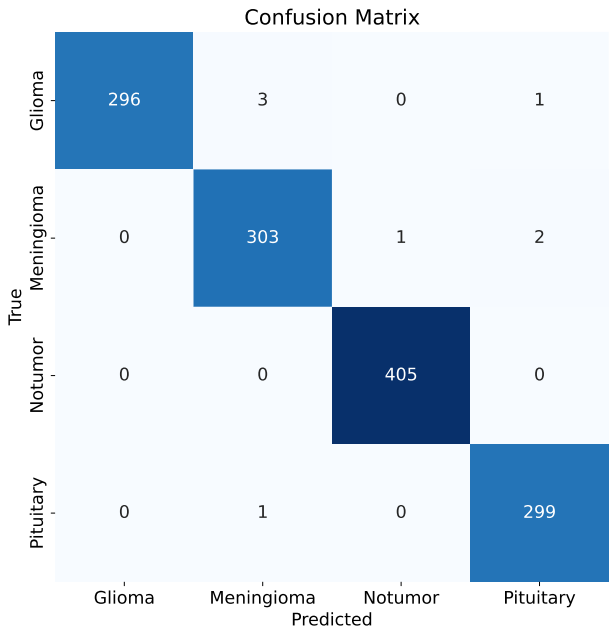


Figure 8: Confusion Matrix for MRI Brain Tumor Classification which displays the model's performance in predicting brain tumor classes. Each row represents the actual class, while each column corresponds to the predicted class. The diagonal repreents the model's acurate category predictions.

Overall, the model demonstrates high accuracy and performs well in identifying brain tumor images across all four classes.

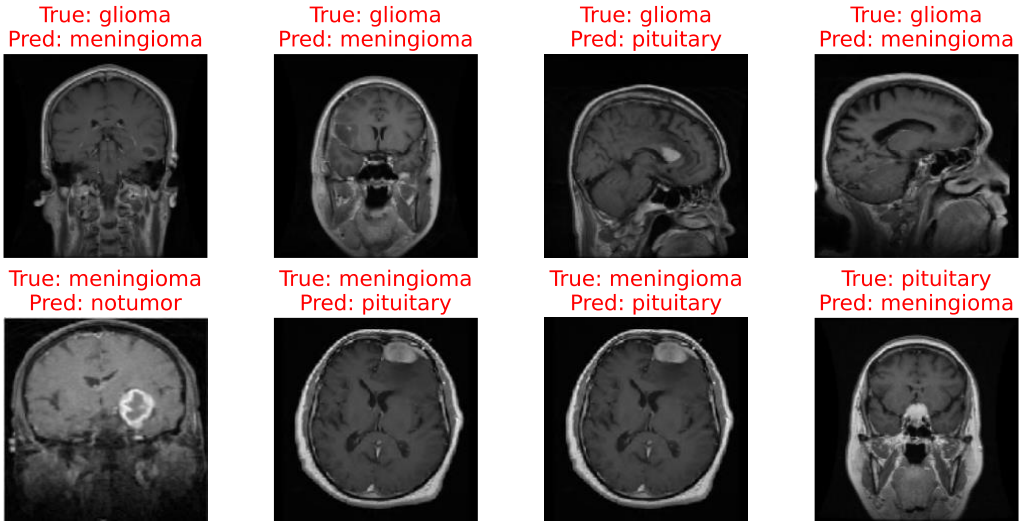


Figure 9: Eight out of the nine misclassified images

5.2 Performance Evaluation Insights

The following table presents precision, recall, and F1-score for each class in the multi-class classification problem:

Class	Precision	Recall	F1-Score
Glioma	0.990	0.997	0.993
Meningioma	0.998	1.000	0.999
No Tumor	0.987	0.990	0.989
Pituitary	1.000	0.987	0.993

Table 3: Evaluation metrics used on final model. See Appendix section (A) for metric definitions and explanations.

The performance metrics provide valuable insights into the model's ability to classify brain tumor images accurately. Key observations from the evaluation are as follows:

- Glioma: The model demonstrated a very high recall of 0.997, which indicates that it accurately identified 99.7% of all actual Glioma images, minimizing the number of false negatives. The impressive F1-score of 0.992 signifies a well-balanced trade-off between precision and recall, showcasing the model's ability to accurately identify cancerous brain tumors in glial cells.
- Meningioma: The model exhibited excellent precision of 0.998, suggesting that it had a low number of false positives in the Meningioma category, minimizing the instances of misclassifying normal brain scans as Meningioma. The perfect recall of 1.000 indicates that the model correctly classified all actual Meningioma images, showcasing its robust performance in this class, leaving no false negatives.
- No Tumor: The model achieved high precision, recall, and F1-score of 0.987 for No Tumor images. This demonstrates its consistent performance in identifying normal brain scans without detectable tumors, with minimal false positives and false negatives.
- Pituitary: The model achieved a perfect precision of 1.000 for Pituitary images, indicating that it did not produce any false positives in this category, i.e., correctly identifying all Pituitary images among the predicted

positive cases. The recall of 0.987 means that the model correctly identified 98.7% of all actual Pituitary images, showcasing its effectiveness in distinguishing this class, with a low number of false negatives.

5.3 Overall Evaluation

The final CNN model achieved an accuracy of 99.4% and demonstrated high precision, recall, and F1-scores for each class, indicating its potential in assisting medical professionals with brain tumor classification. The model's ability to differentiate between different tumor categories could provide valuable information for medical decision-making and personalized treatment plans, potentially leading to improved medical treatment and patient outcomes.

By leveraging Convolutional Neural Networks (CNNs) and fine-tuning key hyperparameters, this model automates brain tumor classification, enabling more efficient and potentially more accurate diagnoses. The results achieved by the final CNN model underscore the significance of deep learning in medical diagnostics and its potential for transforming healthcare practices.

Conclusion

The project on MRI Brain Tumor Classification using Convolutional Neural Networks (CNNs) has successfully demonstrated the power of deep learning in medical diagnostics. The objective of achieving high-test accuracy for accurate brain tumor classification has been met, and the results obtained are promising for real-world medical applications.

The development of a unified CNN model for multi-class brain tumor classification highlights the potential of CNNs in automating brain tumor detection and classification. The model's ability to accurately distinguish between Glioma, Meningioma, No Tumor, and Pituitary categories has significant implications for early and accurate brain tumor diagnosis.

The implementation of data augmentation techniques, such as rescaling, rotation, brightness shifts, and horizontal flip, has proven crucial in enhancing the model's performance. These techniques ensure that the model generalizes well to real-world scenarios and improves its ability to handle variations in the input data.

The final CNN model, which includes fine-tuned hyperparameters for the Adam optimizer and training callbacks, achieved remarkable results. With an accuracy rate of 99.4% and high precision, recall, and F1-scores for each class, the model demonstrated its potential for aiding medical professionals in making informed decisions and providing personalized treatment plans.

The successful performance of the CNN model in brain tumor classification highlights the significance of leveraging deep learning techniques in the field of medical imaging. Early and accurate diagnosis of brain tumors can lead to improved medical treatment and patient outcomes, and automating the classification process through CNNs can streamline the diagnostic process and enhance healthcare practices.

In conclusion, the project's outcomes contribute to the advancement of medical diagnostics and reinforce the potential of deep learning in revolutionizing healthcare. As research in the field of artificial intelligence and medical imaging continues to progress, CNN-based approaches will play an increasingly pivotal role in enhancing medical diagnosis and patient care.

The project serves as a foundation for further research and development in medical image analysis, and the insights gained from this study can guide future improvements in brain tumor classification models. By continually refining and optimizing CNN architectures, we can further improve the accuracy and efficiency of brain tumor classification, ultimately benefiting patients and medical professionals alike.

Appendix

A. Math behind Metrics

In this section, we provide a detailed explanation of the performance evaluation metrics used to assess the effectiveness of the final CNN model for MRI Brain Tumor Classification.

1. Accuracy (ACC): Accuracy measures the overall correctness of the model's predictions. It represents the ratio of correctly classified samples (true positives and true negatives) to the total number of samples in the testing set. Mathematically, it is defined as:

$$\text{Accuracy} = \frac{\text{TP}_1 + \text{TP}_2 + \ldots + \text{TP}_N}{\text{TP}_1 + \text{TP}_2 + \ldots + \text{TP}_N + \text{FP}_1 + \text{FP}_2 + \ldots + \text{FP}_N + \text{FN}_1 + \text{FN}_2 + \ldots + \text{FN}_N}$$

In a multi-class system, we have:

- TP (True Positives): Number of instances correctly classified as a specific class.
- FP (False Positives): Number of instances incorrectly classified as a specific class, which do not actually belong to it.
- FN (False Negatives): Number of instances belonging to a specific class but incorrectly classified as other classes.
- Note, the lower indices refer to the class number denotes as c in the following metric types.

2. Precision (P): Precision measures the proportion of correctly predicted positive samples (TP) to the total number of positive predictions (TP + FP). It quantifies the model's ability to avoid false positives, i.e., the proportion of correctly identified positive cases among all predicted positive cases. Mathematically, it is defined as:

$$\text{Precision}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}$$

3. Recall (Sensitivity): Recall, also known as sensitivity or true positive rate (TPR), measures the model's ability to identify all positive samples by calculating the ratio of correctly predicted positive samples (TP) to the total number of actual positive samples (TP + FN). Mathematically, it is defined as:

$$\text{Recall}_c = \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}$$

4. F1-Score: The F1-score is the harmonic mean of precision and recall, providing a balanced assessment of the model's performance, especially for imbalanced datasets. A higher F1-score indicates a better trade-off between precision and recall. Mathematically, it is defined as:

$$\text{F1-Score}_c = 2 \times \frac{\text{Precision}_c \times \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

The performance evaluation metrics presented in this section help us assess the final CNN model's accuracy and classification capabilities for MRI Brain Tumor Classification. These metrics provide valuable insights into the model's performance, enabling us to make informed decisions and draw meaningful conclusions about its effectiveness in assisting medical professionals with brain tumor diagnosis and treatment planning.

B. YouTube Video

The link my YouTube video can be found here or click on appendix B.:
https://www.youtube.com/watch?v=uQxzgXJ1Xnw

C. Hardware

Local Computer details

- MacBook Pro M2 Pro
- 10-Core CPU
- 16-Core GPU
- 16GB Unified Memory
- 512GB SSD Storage
- 16-core Neural Engine
- MacOS Ventura: Version 13.5

D. Data Importing

Given that the data augmentation code is provided in section (3.3) alongside a comprehensive explanation of the methods employed for data augmentation and their integration into the training and validation of our CNN, we intend to incorporate the importing methods and code into the appendix section. Presented below is the function devised to import and shuffle the data. It is important to note that, due to the use of MacOS software, the DS directories were excluded when retrieving the data, as demonstrated in lines 18-19.

```
 1. # Function for inporting data
 2. def get_data_labels(directory, shuffle=True, random_state=0):
 3.     """
 4.     Function used for going into the main training directory
 5.     whose directory has sub-class-types.
 6.     """
 7.     from sklearn.utils import shuffle
 8.     import os
 9.
10.     # Lists to store data and labels
11.     data_path = []
12.     data_labels = []
13.
14.     for label in os.listdir(directory):
15.         label_dir = os.path.join(directory, label)
16.
17.         # Avoid MacOS storing path
18.         if not os.path.isdir(label_dir):
19.             continue
20.
21.         # Going into each folder and getting image path
22.         for image in os.listdir(label_dir):
23.             image_path = os.path.join(label_dir, image)
24.             data_path.append(image_path)
25.             data_labels.append(label)
26.
27.     if shuffle:
28.         data_path, data_labels = shuffle(data_path, data_labels, random_state=random_state)
```

```
29.
30.     return data_path, data_labels
31.
```

Code Block 4: Importing data function.

```
 1. # Setting up file paths for training and testing
 2. USER_PATH = r'/Users/computer_user/Desktop’
 3. train_dir = USER_PATH + r'/Brain_Tumer_Data/Training/'
 4. test_dir = USER_PATH + r'/Brain_Tumer_Data/Testing/'
 5.
 6. # Getting data using above function
 7. train_paths, train_labels = get_data_labels(train_dir)
 8. test_paths, test_labels = get_data_labels(test_dir)
 9.
10. # Printing traing and testing sample sizes
11. print('Training')
12. print(f'Number of Paths: {len(train_paths)}')
13. print(f'Number of Labels: {len(train_labels)}')
14. print('\nTesting')
15. print(f'Number of Paths: {len(test_paths)}')
16. print(f'Number of Labels: {len(test_labels)}')

Training
Number of Paths: 5712
Number of Labels: 5712

Testing
Number of Paths: 1311
Number of Labels: 1311
```

Code Block 5: Importing the data paths and labels using the function from Code Block 4.