Unified CNN Approach for Multi-Class MRI Brain Tumor Classification

Math & Physics Fun with Gus

July 15, 2023

Table of Contents

Author: Math & Physics Fun with Gus (https://mathphysicsfunwithgus.square.site)

1 | About The Data

1.1 | What is a Brain Tumor?

A brain tumor refers to an abnormal collection or mass of cells within the brain. The skull, which encloses the brain, has limited space, and any growth within this confined area can lead to complications. Brain tumors can be either cancerous (malignant) or noncancerous (benign). As benign or malignant tumors grow, they can increase the pressure inside the skull. This elevated pressure can cause brain damage and pose a life-threatening risk.

1.1.1 | The Importance of Brain Tumor Classification

The early detection and classification of brain tumors are crucial areas of research in medical imaging. Accurate classification aids in selecting the most suitable treatment method, potentially saving patients' lives.

1.1.2 | Methods

The application of deep learning approaches in healthcare has yielded significant advancements in health diagnosis. According to the World Health Organization (WHO), effective brain tumor diagnosis involves detecting the tumor, identifying its location within the brain, and classifying it based on malignancy, grade, and type. This experimental work focuses on diagnosing brain tumors using Magnetic Resonance Imaging (MRI). The process entails tumor detection, classification by grade and type, and identification of the tumor's location. Instead of employing individual models for each classification task, this method utilizes a single model for classifying brain MRI images across different classification tasks. The classification and detection of tumors employ a Convolutional Neural Network (CNN)-based multi-task approach. Additionally, a CNN-based model is employed to segment the brain and identify the location of the tumor.

1.2 | About the Dataset

This dataset is a compilation of three primary datasets: figshare, Br35H, and a removed source due to bad data.

1.2.1 | Dataset Description

The dataset comprises a total of `7023` human **brain MRI images**, categorized into four distinct classes. The dataset focuses on brain tumors and their classification. The four classes are as follows:

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, which can be cancerous or non-cancerous.

Advancing the development of machine learning models for tumor classification is crucial for driving progress in the field of neurology and making a significant impact on the lives of individuals. These models have the potential to enhance medical research, improve diagnostic accuracy, and contribute to effective treatment strategies for various types of tumors. By leveraging machine learning techniques, we can significantly aid in the advancement of neurology and ultimately improve healthcare outcomes for people affected by tumors.

The "No Tumor" class images were obtained from the `Br35H dataset`.

Note: The images in this dataset have varying sizes. After pre-processing and removing excess margins, you can resize the images to the desired dimensions.

The data link and complete description here `Brain Tumor Data on Kaggle` (https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset)

2 | Importing & Setup

In [2]:

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

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

# Building Model
from keras.utils import plot_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
```

In [3]:

```
# Global variables
SAVE = False
SEED = 111

# 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)
```

3 | Importing Data

In [5]:

```
# Setting up file paths for training and testing
USER_PATH = r"/kaggle/input/brain-tumor-mri-dataset"
train_dir = USER_PATH + r'/Training/'
test_dir = USER_PATH + r'/Testing/'

# Getting data using above function
train_paths, train_labels = get_data_labels(train_dir)
test_paths, test_labels = get_data_labels(test_dir)

# Printing traing and testing sample sizes
print('Training')
print(f'Number of Paths: {len(train_paths)}')
print(f'Number of Labels: {len(train_labels)}')
print('\nTesting')
print(f'Number of Paths: {len(test_paths)}')
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
```

4 | Data Visualization

4.2 | Data Distributions

In [10]:

```
# Image size
image_size = (150, 150)

# Training batch size
batch_size = 32
```

In [11]:

```
# 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_siz
e,
                                                    batch_size=batch_siz
e,
                                                    class_mode="categoric
al",
                                                    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="categorica
l",
                                                  shuffle=False,
                                                  seed=SEED)
```

```
Found 5712 images belonging to 4 classes.
Found 1311 images belonging to 4 classes.
```

5.1.1 Data Augmentation Class Indices

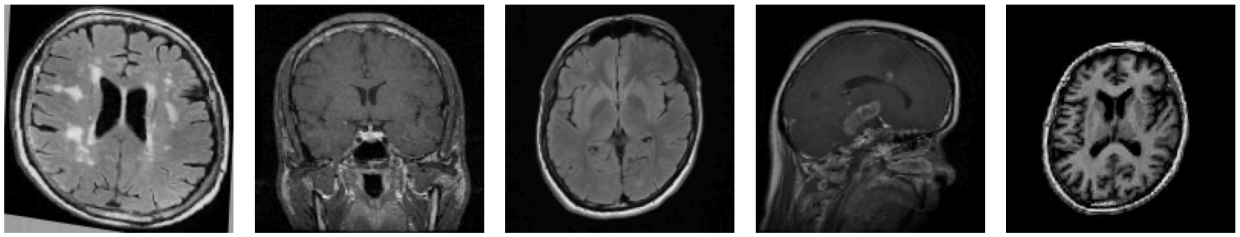
In [12]:
```
# Accessing class indices for training data generator
class_indices_train = train_generator.class_indices
class_indices_train_list = list(train_generator.class_indices.keys())


# Displaying categorical types
print("Categorical types for the training data:")
print(class_indices_train)
```

```
Categorical types for the training data:
{'glioma': 0, 'meningioma': 1, 'notumor': 2, 'pituitary': 3}
```

5.1.2 Showing Data Augmentation

In [14]:
```
show_ImageDataGenerator(train_datagen, num_samples=5, figsize=(12.5, 8),
save=SAVE)
```



5.2 Training Setup Values

In [15]:

```
# Image shape: height, width, RBG
image_shape = (image_size[0], image_size[1], 3)

# Training epochs
epochs = 40

# Steps per epoch
steps_per_epoch = train_generator.samples // batch_size

# Validation steps
validation_steps = test_generator.samples // batch_size

print(f'Image shape: {image_shape}')
print(f'Epochs: {epochs}')
print(f'Batch size: {batch_size}')
print(f'Steps Per Epoch: {steps_per_epoch}')
print(f'Validation steps: {validation_steps}')
```

```
Image shape: (150, 150, 3)
Epochs: 40
Batch size: 32
Steps Per Epoch: 178
Validation steps: 40
```

6 | Analysis Functions for CNN

6.1 | Evaluation Metrics for Multi-Class Classification Tasks

In the case of multi-class classification with four possible outputs, some of the metrics need to be adjusted. Here we explain the metrics for multi-class classification:

NOTE: In a multi-class classification problem, the concept of "True Negatives" is not applicable. True Negatives are specific to binary classification, where you have two classes (positive and negative).

A `confusion matrix` is a table that summarizes the performance of a classification model. Since we have multiple classes for us we will use it to provid a breakdown of predictions versus actual class labels for each class.

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.

	Predicted Class 1	Predicted Class 2	...	Predicted Class N
Actual Class 1	True Positive (TP)	False Positive (FP)	...	False Positive (FP)
Actual Class 2	False Positive (FP)	True Positive (TP)	...	False Positive (FP)
...	...	...	...	...
Actual Class N	False Positive (FP)	False Positive (FP)	...	True Positive (TP)

6.1.1 | Precision

Precision measures the ability of the model to correctly identify positive instances for each class among all instances predicted as positive.

For each class *c*: `Precision_c = TP_c / (TP_c + FP_c)`

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

6.1.2 Recall (Sensitivity or True Positive Rate)

Recall calculates the ability of the model to correctly identify positive instances for each class among all actual positive instances.

For each class *c*: `Recall_c = TP_c / (TP_c + FN_c)`

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

6.1.3 F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a balanced measure that combines both metrics for each class.

For each class *c*: `F1-Score_c = 2 * (Precision_c * Recall_c) / (Precision_c + Recall_c)`

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

6.1.4 Accuracy

Accuracy measures the overall correctness of the model's predictions across all classes. `Accuracy = (TP_1 + TP_2 + ... + TP_N) / (TP_1 + TP_2 + ... + TP_N + FP_1 + FP_2 + ... + FP_N + FN_1 + FN_2 + ... + FN_N)`

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

7 | Initial CNN Model Tests

```
From `model_1` I tested the following tuples and got the following test accuarcies. Since parameters `3.` rounded to two decimal places performed the best, or equally the best, we will use it as we improve our model.
```

1. filter size of `filter_size = (4, 4)` and `pool_size = (2, 2)` the accuaracy was `Test Accuracy: 0.97`
2. filter size of `filter_size = (3, 3)` and `pool_size = (3, 3)` the accuaracy was `Test Accuracy: 0.97`
3. filter size of `filter_size = (4, 4)` and `pool_size = (3, 3)` the accuaracy was `Test Accuracy: 0.98`
4. filter size of `filter_size = (3, 3)` and `pool_size = (2, 2)` the accuaracy was `Test Accuracy: 0.98`

Model `3.` was the tested by switched parts, then all to `model.add(AveragePooling2D(pool_size=(3, 3)))`.

- The results were good with a `Test Accuracy score: 0.9766`. But did not improve on upon the previous model.

Model `3.` was tested with different optimizers. The optimzer test accuracy scores were as follows.

1. Adam: `Test Accuracy: 0.982`
2. RMSprop: `Test Accuracy: 0.972`
3. Nadam: `Test Accuracy: 0.964`

In addition various trials were done to the parameters of `ImageDataGenerator()` to help ensure that overfitting would be minimized.

```
# Define the model architecture
model_1 = models.Sequential()

# Convolutional layer 1
model_1.add(Conv2D(32, (4, 4), activation="relu", input_shape=image_shape))
model_1.add(MaxPooling2D(pool_size=(3, 3)))

# Convolutional layer 2
model_1.add(Conv2D(64, (4, 4), activation="relu"))
model_1.add(MaxPooling2D(pool_size=(3, 3)))

# Convolutional layer 3
model_1.add(Conv2D(128, (4, 4), activation="relu"))
model_1.add(MaxPooling2D(pool_size=(3, 3)))

# Convolutional layer 4
model_1.add(Conv2D(128, (4, 4), activation="relu"))
model_1.add(Flatten())

# Full connect layers
model_1.add(Dense(512, activation="relu"))
model_1.add(Dropout(0.5, seed=SEED))
model_1.add(Dense(N_TYPES, activation="softmax"))

model_1.summary()
```

- Convolutional layer 1:
  - Input shape: (150, 150, 3)
  - Number of filters: 32
  - Filter size: (4, 4)
  - Activation function: ReLU
  - Output shape: (147, 147, 32) [Calculation: (150 - 4) + 1 = 147]
- Max Pooling layer 1:
  - Pool size: (3, 3)
  - Output shape: (49, 49, 32) [Calculation: 147 / 3 = 49]
- Convolutional layer 2:

    - Input shape: (49, 49, 32)
    - Number of filters: 64
    - Filter size: (4, 4)
    - Activation function: ReLU
    - Output shape: (46, 46, 64) [Calculation: (49 - 4) + 1 = 46]
- Max Pooling layer 2:
    - Pool size: (3, 3)
    - Output shape: (15, 15, 64) [Calculation: 46 / 3 = 15]
- Convolutional layer 3:
    - Input shape: (15, 15, 64)
    - Number of filters: 128
    - Filter size: (4, 4)
    - Activation function: ReLU
    - Output shape: (12, 12, 128) [Calculation: (15 - 4) + 1 = 12]
- Max Pooling layer 3:
    - Pool size: (3, 3)
    - Output shape: (4, 4, 128) [Calculation: 12 / 3 = 4]
- Convolutional layer 4:
    - Input shape: (4, 4, 128)
    - Number of filters: 128
    - Filter size: (4, 4)
    - Activation function: ReLU
    - Output shape: (1, 1, 128) [Calculation: (4 - 4) + 1 = 1]
- Flatten layer:
    - Reshapes the output to a 1D array of size 128.
- Dense layer 1:
    - Number of neurons: 512
    - Activation function: ReLU
    - Output shape: 512
- Dropout layer:

    - Dropout rate: 0.5
    - Output shape: 512
- Dense layer 2:
    - Number of neurons: N_TYPES (the number of output classes)
    - Activation function: Softmax
    - Output shape: N_TYPES

Total trainable parameters: 495,972 (1.89 MB)

8 | Final CNN Model

Here, we present the second version of our Convolutional Neural Network (CNN) architecture, which includes some important tweaks and additional modifications for improved performance. We have tested `BatchNormalization()` layers, which proved to be not so helpful. We also tested changing certain filter sizes such as the last `filter_size = (3,3)`. This change proved to have near no affect on the model's improvment on test accuracy.

Furthermore, we performed fine-tuning on certain hyperparameters, specifically β_1 and β_2, which are part of the Adam optimizer. By experimenting with different values, we explored a range of $\beta_1 \in [0.7, 0.995]$ and $\beta_2 \in [0.9, 0.9995]$ to optimize the training process. After conducting an exhaustive evaluation of multiple models, we achieved the highest validation accuracy with the following configuration:

Adam Parameters:

- `learning_rate` : The learning rate determines the step size for adjusting the model weights during training. Higher values can lead to faster convergence, but they may also risk overshooting. The default value is 0.001.
- `beta_1` : The exponential decay rate for the first moment estimates, controlling the exponential decay of the moving average of past gradients. The default value is 0.9.
- `beta_2` : The exponential decay rate for the second moment estimates, controlling the exponential decay of the moving average of past squared gradients. The default value is 0.999.
- `epsilon` : A small value added to the denominator for numerical stability. The default is 1e-7.
- `decay` : This parameter gradually decreases the learning rate over time to fine-tune the model.
- `amsgrad` : A boolean value indicating whether to use the AMSGrad variant of the Adam optimizer. The default is False.
- `clipnorm` : Caps the norm of the gradients to a specified maximum value. It provides an alternative to `clipvalue` .
- `clipvalue` : Prevents gradients from becoming too large by capping them at a specified maximum value.

Additionally, to enhance the training process and prevent overfitting, we implemented the following callbacks:

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau

EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='aut
o', baseline=None, restore_best_weights=False, start_from_epoch=0)
ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10, verbose=0, mode
='auto', min_delta=0.0001, cooldown=0, min_lr=0)
```

With these callbacks, the model's training will stop early if the loss stops decreasing (using `EarlyStopping`), and the learning rate will be reduced if the validation loss plateaus (using `ReduceLROnPlateau`).

These additional tweaks have improved the performance of our CNN model. Through experimentation and fine-tuning, we have achieved better convergence and higher validation accuracy.

In [17]:

```
# Define the model architecture
model = models.Sequential([

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

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

    # Convolutional layer 3
    Conv2D(128, (4, 4), activation="relu"),
    MaxPooling2D(pool_size=(3, 3)),

    # Convolutional layer 4
    Conv2D(128, (4, 4), activation="relu"),
    Flatten(),

    # Full connect layers
    Dense(512, activation="relu"),
    Dropout(0.5, seed=SEED),
    Dense(N_TYPES, activation="softmax")
])

model.summary()

optimizer = legacy.Adam(learning_rate=0.001, beta_1=0.869, beta_2=0.995)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metri
cs= ['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 147, 147, 32)      1568

 max_pooling2d (MaxPooling2  (None, 49, 49, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 46, 46, 64)        32832

 max_pooling2d_1 (MaxPoolin  (None, 15, 15, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 128)       131200

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 conv2d_3 (Conv2D)           (None, 1, 1, 128)         262272

 flatten (Flatten)           (None, 128)               0

 dense (Dense)               (None, 512)               66048

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 495972 (1.89 MB)
Trainable params: 495972 (1.89 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```