Title
Exploring Machine Learning Algorithms for Automated Segmentation of Brain Tumors from MRI Scans

MSc Research Project
(MSCDAD_C)

Sushmitha vurutur sridhar
Student ID: 22201378

School of Computing
National College of Ireland

Supervisor: Hamilton Niculescu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sushmitha vurutur sridhar		
Student ID:	22201378		
Programme:	MSCDAD_C	**Year:**	2023-2024
Module:	MSc Research Project		
Supervisor:	Hamilton Niculescu		
Submission Due Date:	12/08/2024		
Project Title:	Exploring Machine Learning Algorithms for Automated Segmentation of Brain Tumors from MRI Scans		
Word Count:	1128	**Page Count :**	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Sushmitha vurutur sridhar

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

The purpose of this manual is to provide a comprehensive guide for setting up and configuring the environment necessary to replicate the brain tumor segmentation project. This manual will walk you through the installation of required software and libraries, the setup of the data pipeline, the configuration and training of machine learning models, and the execution of the provided Jupyter Notebook. The manual aims to ensure that users can easily reproduce the project results, understand the underlying processes, and apply the techniques to similar tasks.

2. Minimum System Requirements

2.1 Hardware Requirement

To successfully set up and run the brain tumor segmentation project, your system should meet the following minimum requirements:

- **Operating System:** Windows 10/11, macOS 10.15 or higher, or a Linux distribution such as Ubuntu 18.04 or higher.
- **Processor:** Intel Core i5 or equivalent AMD processor, with at least 4 cores.
- **Memory (RAM):** 8 GB or higher (16 GB recommended for smoother operation).
- **Graphics Processing Unit (GPU):** NVIDIA GPU with CUDA support (optional but recommended for faster training).
- **Storage:** At least 20 GB of free disk space to store the dataset and trained models.

Specifically, this project was run on below specification

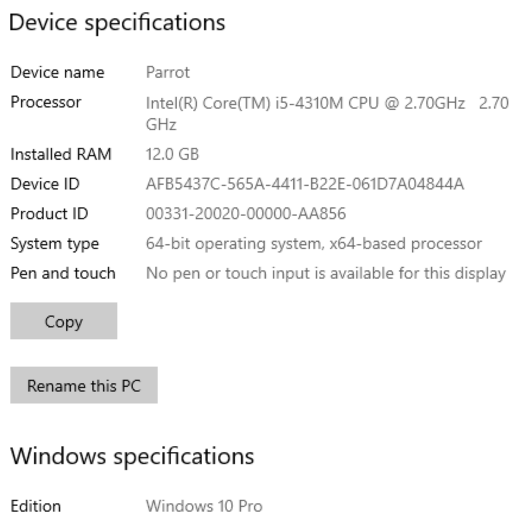


Figure 1: Host System Hardware Specifications

2.2 Software Requirement

- **Software:**
  - Python 3.8 or higher
  - Jupyter Notebook
  - Anaconda (optional, for easier environment management)
  - Required Python libraries: TensorFlow, Keras, OpenCV, Scikit-learn, Matplotlib
- **Internet Connection:** Required for downloading datasets, libraries, and for running Jupyter Notebook on Google Colab (if applicable).

3. Setting Up the Enviroment

3.1 Mounting Google Drive

The code was executed of Jupyter Notebook on Google Colab. In Google Colab, you first need to mount your Google Drive to access any datasets or files stored there. This is done by using the drive module from the google.colab package. After executing this code, you'll be prompted to authorize Google Colab to access your Google Drive. Once authorized, your Google Drive will be mounted at /content/drive/.

```
[ ] # Load dataset from google drive

    from google.colab import drive
    drive.mount('/content/drive/')

    Mounted at /content/drive/
```

3.2 Importing Required Libraries

The next step is to import all the necessary libraries for the project. These libraries include TensorFlow, Keras, OpenCV, Scikit-Image, NumPy, Matplotlib, and others, which are essential for handling data preprocessing, model building, and visualization.

- **TensorFlow & Keras:** Used for building and training the neural network models.
- **os, glob:** For handling file paths and directory structures.
- **skimage:** Used for image processing tasks.
- **cv2:** OpenCV library, used for image manipulation and preprocessing.
- **NumPy:** For numerical operations on arrays.
- **Matplotlib:** For data visualization.

```
# importing libraries
import tensorflow as tf
import keras
import os
import glob
import skimage
from skimage import io
import random
import cv2
import numpy as np
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory

from tensorflow.keras.utils import img_to_array,array_to_img, load_img
import matplotlib.pyplot as plt
from keras import backend as K
%matplotlib inline
```

Note:

Google Colab Environment: Google Colab already has most of these libraries pre-installed, but if you encounter any missing libraries, you can install them using *pip install library_name*.

4. Data Pipeline Configuration

4.1 Data Collection

- **Description of Datasets Used**

For this project, the dataset used is the "Brain MRI Images for Brain Tumor Detection," which is publicly available on Kaggle. This dataset consists of 253 brain MRI images categorized into two classes: images with brain tumors and images without brain tumors. The dataset is particularly useful for training and testing machine learning models for the binary classification of brain tumor presence. **Dataset Link:** Brain MRI Images for Brain Tumor Detection

Instructions on How to Download and Prepare the Dataset

- **Downloading the Dataset:**
  - Visit the Kaggle dataset page.
  - Click on the "Download" button to download the dataset to your local machine.
  - If using Google Colab, you can also use the Kaggle API to download the dataset directly into your Google Drive.

- **Preparing the Dataset:**

After downloading and extracting the dataset, it is essential to organize the images into appropriate directories to facilitate easy access during the training of the models. Typically, the dataset should be structured in a directory format that includes subdirectories for each class, such as "yes" for tumor images and "no" for non-tumor images. This organization ensures that the data is properly categorized, making it straightforward to use in machine learning pipelines.

- **Data Preprocessing**

Steps to Preprocess the MRI Images

Preprocessing is a crucial step in preparing the data for training machine learning models. For MRI images, preprocessing typically involves the following steps:

Resizing:

Resize all images to a uniform size to ensure that the input dimensions match the model requirements. In this case, you can resize the images to 224x224 pixels, which is a common input size for CNNs.

```python
# Image data specifications
img_width, img_height = 224, 224

data_dir = '/content/drive/MyDrive/Datasets/brain_tumor_dataset_split'
TRAIN_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/train'
TEST_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/test'
VAL_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/val'


train_samples = sum([len(files) for r, d, files in os.walk(TRAIN_DIR)])
validation_samples = sum([len(files) for r, d, files in os.walk(VAL_DIR)])
test_samples = sum([len(files) for r, d, files in os.walk(TEST_DIR)])
epochs = 25
batch_size = 20
```

4.2 Data Augmentation:

Apply data augmentation techniques to artificially increase the size of the dataset and help the model generalize better. Common augmentation techniques include rotation, flipping, scaling, and adding noise.

```python
# Enhanced Data Augmentation using ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,        # Randomly rotate images by up to 20 degrees
    width_shift_range=0.2,    # Randomly shift images horizontally by 20% of the width
    height_shift_range=0.2,   # Randomly shift images vertically by 20% of the height
    shear_range=0.2,          # Apply random shearing transformations
    zoom_range=0.2,           # Randomly zoom into images by 20%
    horizontal_flip=True,     # Randomly flip images horizontally
    vertical_flip=True,       # Randomly flip images vertically
    fill_mode='nearest'       # Fill in pixels after transformations
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    VAL_DIR,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 202 images belonging to 2 classes.
Found 24 images belonging to 2 classes.
```

5. Data Splitting

Instructions on Splitting the Dataset

To ensure the model's performance is evaluated accurately, the dataset should be split into three subsets: training, validation, and test sets.

- **Training Set (80% of data) -** Used for training the model.
- **Validation Set (10 of data) -** Used to tune the model's hyperparameters and monitor overfitting during training.
- **Test Set (10 of data) -** Used to evaluate the final performance of the model.

```
[ ] import splitfolders
    splitfolders.ratio(data_dir, output=output_folder, seed=23, ratio=(.8, .1, .1), group_prefix=None)

    Copying files: 253 files [01:36,  2.62 files/s]
```

```
[ ] # Image data specifications
    img_width, img_height = 224, 224

    data_dir = '/content/drive/MyDrive/Datasets/brain_tumor_dataset_split'
    TRAIN_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/train'
    TEST_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/test'
    VAL_DIR = '/content/drive/MyDrive/brain_tumor_dataset_split/val'


    train_samples = sum([len(files) for r, d, files in os.walk(TRAIN_DIR)])
    validation_samples = sum([len(files) for r, d, files in os.walk(VAL_DIR)])
    test_samples = sum([len(files) for r, d, files in os.walk(TEST_DIR)])
    epochs = 25
    batch_size = 20
```

6. Model Configuration

Training the Model

Step-by-Step Guide to Training the Model Using the Preprocessed Data

Training the models involves feeding the preprocessed MRI images into the selected CNN architectures and optimizing their parameters to minimize the loss function. Here is a step-by-step guide:

1. **Load the Preprocessed Data:**

Ensure that the data is loaded into the appropriate format (e.g., TensorFlow Dataset) and split into training, validation, and test sets.

```
[ ] # Configure datasets for performance
    AUTOTUNE = tf.data.AUTOTUNE
    train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
    validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
    test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Define the Model Architecture:

Use the pre-trained weights of MobileNetV2, VGG16, and ResNet50, and customize the final layers to match the number of output classes (binary classification: tumor vs. non-tumor).

```
[ ] # Modeling
    from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D
    from keras.layers import Activation, Dropout, Flatten, Dense
    from keras.applications.vgg16 import VGG16
    from keras.applications.resnet50 import ResNet50
    from keras.applications.mobilenet import MobileNet
    from keras.applications import MobileNetV2

    # Define input shape for the models
    input_shape = (224, 224, 3)

    # Number of classes
    num_classes = len(train_classes)

    # MobileNetV2 model
    mobile_model = Sequential([
        MobileNetV2(
            include_top=False,
            weights="imagenet",
            input_shape=input_shape,
            pooling='max'
        ),
        Dense(128, activation='relu'),
        Dropout(0.1),
        Dense(num_classes, activation='softmax')
    ])
    # define the shapes of all layers by passing a dummy input
    mobile_model.build(input_shape=(None, 224, 224, 3))

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
    9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Compile the Model:

Choose an optimizer (e.g., Adam), a loss function (e.g., binary cross-entropy), and evaluation metrics (e.g., accuracy, Dice coefficient).

```
# Compile and train models

steps_per_epoch = train_samples // batch_size
validation_steps = validation_samples // batch_size
test_steps = test_samples // batch_size

models = [mobile_model, vgg_model, resnet_model]
model_names = ['MobileNetV2','VGG16','ResNet50']
histories = []


for model, name in zip(models, model_names):
    model.compile(loss='categorical_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  metrics=['accuracy'])

    print(f"Training {name} model...")
    history = model.fit(
                  train_dataset,
                  epochs=25,
                  validation_data=validation_dataset
              )

    # Store the history
    histories.append(history)

    # Evaluate the model
    test_loss, test_acc = model.evaluate(test_dataset)
    print(f'{name} Test accuracy:', test_acc)
```

```
Training MobileNetV2 model...
Epoch 1/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 94s 5s/step - accuracy: 0.5309 - loss: 7.3432 - val_accuracy: 0.6250 - val_loss: 4.8390
Epoch 2/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 77s 4s/step - accuracy: 0.8381 - loss: 0.5049 - val_accuracy: 0.6250 - val_loss: 4.2896
```

Train the Model:

Set the number of epochs and batch size, and initiate the training process.

```
Epoch 13/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 207s 16s/step - accuracy: 0.8876 - loss: 0.2885 - val_accuracy: 0.7083 - val_loss: 1.1953
Epoch 14/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 176s 16s/step - accuracy: 0.8298 - loss: 0.3999 - val_accuracy: 0.7500 - val_loss: 1.0495
Epoch 15/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 173s 16s/step - accuracy: 0.9267 - loss: 0.2627 - val_accuracy: 0.7083 - val_loss: 1.1981
Epoch 16/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 199s 15s/step - accuracy: 0.8898 - loss: 0.2876 - val_accuracy: 0.8333 - val_loss: 0.6645
Epoch 17/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 170s 15s/step - accuracy: 0.9378 - loss: 0.1595 - val_accuracy: 0.8333 - val_loss: 0.5526
Epoch 18/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 210s 16s/step - accuracy: 0.9210 - loss: 0.1846 - val_accuracy: 0.7500 - val_loss: 0.7351
Epoch 19/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 171s 15s/step - accuracy: 0.8892 - loss: 0.2341 - val_accuracy: 0.6667 - val_loss: 2.4591
Epoch 20/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 173s 16s/step - accuracy: 0.9091 - loss: 0.2345 - val_accuracy: 0.6250 - val_loss: 3.2791
Epoch 21/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 201s 15s/step - accuracy: 0.8961 - loss: 0.2852 - val_accuracy: 0.6667 - val_loss: 1.4322
Epoch 22/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 203s 16s/step - accuracy: 0.9099 - loss: 0.2416 - val_accuracy: 0.6667 - val_loss: 1.5378
Epoch 23/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 173s 16s/step - accuracy: 0.8732 - loss: 0.2453 - val_accuracy: 0.8333 - val_loss: 0.4794
Epoch 24/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 202s 16s/step - accuracy: 0.9675 - loss: 0.1619 - val_accuracy: 0.7083 - val_loss: 0.7509
Epoch 25/25
11/11 ━━━━━━━━━━━━━━━━━━━━ 173s 15s/step - accuracy: 0.9379 - loss: 0.2264 - val_accuracy: 0.9583 - val_loss: 0.2377
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 1s/step - accuracy: 0.8099 - loss: 0.2621
ResNet50 Test accuracy: 0.8148148059844971
```

7. Model Performance Monitoring

Explanation of How to Monitor Model Performance

Monitoring model performance involves tracking various metrics during training, validation, and testing phases. The key metrics to monitor include:

- **Accuracy:** Measures the proportion of correctly classified images out of the total images.
- **Sensitivity (Recall):** Measures the proportion of actual positives (tumor images) correctly identified.
- **Specificity:** Measures the proportion of actual negatives (non-tumor images) correctly identified.
- **Dice Coefficient:** A metric that balances precision and recall, particularly useful for segmentation tasks.

Test the Models on Unseen Data

Once the model has been trained and validated, it should be tested on a completely unseen test set to evaluate its generalization ability. This involves using the evaluate method in Keras to calculate performance metrics on the test data.

```
[ ] # Evaluate the best model on the test data
    test_loss, test_acc = best_model.evaluate(test_dataset)
    print('Best model Test accuracy:', test_acc)
```

Visualizations of Model Performance Using Matplotlib

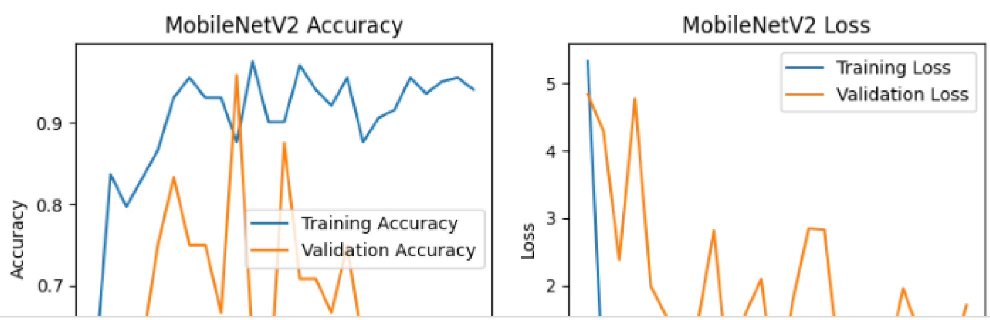
Visualizing the model's performance can help in understanding how well the model is learning and whether it is overfitting. Common plots include training and validation accuracy/loss over epoch

```python
# Plot accuracy and loss curves for each model
for name, history in zip(model_names, histories):
    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title(f'{name} Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'{name} Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()
```



Evaluate machine learning models by generating confusion matrices and classification reports. It helps in visualizing how well the model has performed on the test data

```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np

for model, name in zip(models, model_names):
    print(f'Evaluating {name} model...')

    # Predict on the test dataset
    Y_pred = model.predict(test_dataset)
    y_pred = np.argmax(Y_pred, axis=1)

    # Extract true labels from the test dataset
    y_true = []
    for _, labels in test_dataset:
        y_true.extend(np.argmax(labels.numpy(), axis=1))
    y_true = np.array(y_true)

    # Ensure y_pred has the same length as y_true
    y_pred = y_pred[:len(y_true)]

    cm = confusion_matrix(y_true, y_pred)
    clr = classification_report(y_true, y_pred, target_names=class_names)

    plt.figure(figsize=(8, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
    plt.title(f'{name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

    print(f'{name} Classification Report:\n', clr)
```

```
Evaluating MobileNetV2 model...
```