Code Report - Brain Tumor Classification

Project Supervisor

Mr. SamyanQayyumWahla

Proposed By

Ayesha Nadeem 2021-CS-9

Department of Computer Science
University of Engineering and Technology,
Lahore Pakistan

Table of Contents

Project Description:

The ”Brain Tumor Classification Using AI” project aims to create a system based on AI that can correctly categorize brain tumors into four groups glioma, meningioma, pituitary, and no tumor. The goal of this research is to examine medical images, such as brain MRI scans, and quickly and accurately classify tumors using modern machine learning and image processing techniques.

This study carefully examines medical images, especially Magnetic Resonance Imaging (MRI) scans of the brain, by combining cutting-edge machine learning and sophisticated image processing techniques. This project is significant because it has the potential to make early identification, treatment planning, and brain tumor monitoring easier, leading to better patient outcomes and lower healthcare costs.

The foundation of project is a powerful AI model that was painstakingly built using a sizable dataset of brain MRI scans. Convolutional neural networks (CNNs) and deep learning techniques are used by the AI system to recognize complex patterns and features in the photos. As a result, the system can distinguish between the four types of brain tumors: glioma, meningioma, pituitary, and no tumor.

For clinicians and radiologists, it is crucial to be able to appropriately classify brain tumors. For instance, severe treatment methods are necessary for gliomas, whereas hormone therapy may be necessary for pituitary tumors. Tumor classification that is swift and accurate informs treatment choices and guarantees that patients receive individualized therapy as soon as possible.

Furthermore, the project emphasis on efficiency and accuracy aims to reduce the anxiety and uncertainty that often accompanies the diagnostic process for patients. Timely identification and classification of brain tumors can alleviate the emotional burden on patients and their families, enabling them to make informed decisions regarding treatment and care plans.

By enhancing early diagnosis, maximising resources, decreasing diagnostic errors, and encouraging global collaboration in the field of medical AI, the initiative has the potential to have a big impact on Pakistan’s healthcare system. It is consistent with the overarching objective of using technology to improve healthcare accessibility and quality in the nation.

Code Overview:

The chosen code is a Python script using the TensorFlow and Keras libraries to build, train, and save a convolutional neural network (CNN) for the classification of brain tumor images. The dataset consists of four classes: no tumor, glioma tumor, meningioma tumor, and pituitary tumor. The script includes data loading, preprocessing, augmentation, model architecture definition, compilation, training, and model saving.

Key Components:

1. **Data Loading and Preprocessing:**
    - The dataset is loaded from the 'dataset/' directory, which contains subdirectories for each class of tumor.
    - Images are read, resized to 64x64 pixels, and stored in the dataset array along with corresponding labels.

```
import cv2
import os
from PIL import Image
import numpy as np

data_path = 'dataset/'
dataset = []
label = []
image_size = 64

# Load images for each class
for tumor_type in os.listdir(data_path + 'Training/'):
    for image_name in os.listdir(data_path + f'Training/{tumor_type
        }/'):
        if image_name.split('.')[1] == 'jpg':
            image = cv2.imread(data_path + f'Training/{tumor_type}/'
                + image_name)
            image = Image.fromarray(image, 'RGB')
            image = image.resize((image_size, image_size))
            dataset.append(np.array(image))
            label.append(int(tumor_type.split('_')[0]))

dataset = np.array(dataset)
label = np.array(label)
```

2. **Data Splitting and Normalization:**

   - The dataset is split into training and testing sets using train_test_split.
   - Data normalization is applied using the normalize function.

```
from sklearn.model_selection import train_test_split
from keras.utils import normalize

x_train, x_test, y_train, y_test = train_test_split(dataset, label
    , test_size=0.2, random_state=0)

x_train = normalize(x_train, axis=1)
x_test = normalize(x_test, axis=1)
```

3. **Data Augmentation:**

   - ImageDataGenerator is used to apply various augmentations like rotation, shifting, shearing, zooming, and flipping to increase the diversity of the training set.

```
from keras.preprocessing.image import ImageDataGenerator

# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

datagen.fit(x_train)
```

4. **Model Architecture:**

   - The model consists of several Convolutional, BatchNormalization, Activation, MaxPooling, Flatten, Dense, and Dropout layers.
   - The output layer has 4 units with a softmax activation function, assuming four classes.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout,
    Flatten, Dense, BatchNormalization
```

```
# Model
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(image_size, image_size, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add more Convolutional, BatchNormalization, Activation, and
    MaxPooling layers as needed

model.add(Flatten())
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(4)) # Assuming you have 4 classes
model.add(Activation('softmax'))
```

5. **Model Compilation:**
    - The model is compiled using the sparse categorical crossentropy loss function and the Adam optimizer with a learning rate of 0.001.

```
from keras.optimizers import Adam

model.compile(loss='sparse_categorical_crossentropy', optimizer=
    Adam(lr=0.001), metrics=['accuracy'])
```

6. **Learning Rate Scheduler:**
    - A learning rate scheduler is implemented using a custom function (lr_scheduler) and applied during training.

```
from keras.callbacks import LearningRateScheduler

# Learning Rate Scheduler
def lr_scheduler(epoch):
    return 0.001 * np.exp(-epoch / 10)

scheduler = LearningRateScheduler(lr_scheduler)
```

7. **Model Training:**
   - The model is trained using the fit method, and the training data is augmented using the data generator.

```
model.fit(datagen.flow(x_train, y_train, batch_size=16), epochs
    =20, validation_data=(x_test, y_test), callbacks=[scheduler])
```

8. **Model Saving:**
   - The trained model is saved in an HDF5 file named 'BrainTumorImproved.h5'.

```
model.save('BrainTumorImproved.h5')
```

Code Output:

The training log output displays the progression of a Convolutional Neural Network (CNN) across 20 epochs. Each epoch consists of multiple batches processed for both training and validation datasets. The key metrics provided include training loss, training accuracy, validation loss, and validation accuracy. Additionally, the learning rate for each epoch is displayed.

```
Epoch 1/20
144/144 [==============================] - 62s 419ms/step - loss: 1.4616 -
    accuracy: 0.4377 - val_loss: 2.7050 - val_accuracy: 0.2456 - lr: 0.0010
Epoch 2/20
144/144 [==============================] - 47s 325ms/step - loss: 1.1593 -
    accuracy: 0.5170 - val_loss: 2.7624 - val_accuracy: 0.1341 - lr: 9.0484e-04
Epoch 3/20
144/144 [==============================] - 8s 58ms/step - loss: 0.9793 -
    accuracy: 0.5919 - val_loss: 2.0516 - val_accuracy: 0.2561 - lr: 8.1873e-04
Epoch 4/20
144/144 [==============================] - 10s 67ms/step - loss: 0.8463 -
    accuracy: 0.6416 - val_loss: 3.2711 - val_accuracy: 0.3206 - lr: 7.4082e-04
Epoch 5/20
144/144 [==============================] - 8s 56ms/step - loss: 0.8275 -
    accuracy: 0.6594 - val_loss: 0.9498 - val_accuracy: 0.6394 - lr: 6.7032e-04
Epoch 6/20
144/144 [==============================] - 12s 83ms/step - loss: 0.7871 -
    accuracy: 0.6786 - val_loss: 1.1891 - val_accuracy: 0.6446 - lr: 6.0653e-04
Epoch 7/20
144/144 [==============================] - 10s 70ms/step - loss: 0.7332 -
    accuracy: 0.6860 - val_loss: 2.2543 - val_accuracy: 0.4617 - lr: 5.4881e-04
Epoch 8/20
```

```
144/144 [==============================] - 9s 59ms/step - loss: 0.6979 -
   accuracy: 0.7108 - val_loss: 1.0551 - val_accuracy: 0.5767 - lr: 4.9659e-04
Epoch 9/20
144/144 [==============================] - 25s 177ms/step - loss: 0.6779 -
   accuracy: 0.7199 - val_loss: 1.6830 - val_accuracy: 0.4843 - lr: 4.4933e-04
Epoch 10/20
144/144 [==============================] - 14s 99ms/step - loss: 0.6604 -
   accuracy: 0.7343 - val_loss: 1.9290 - val_accuracy: 0.5348 - lr: 4.0657e-04
Epoch 11/20
144/144 [==============================] - 9s 62ms/step - loss: 0.6410 -
   accuracy: 0.7269 - val_loss: 0.8583 - val_accuracy: 0.6551 - lr: 3.6788e-04
Epoch 12/20
144/144 [==============================] - 10s 70ms/step - loss: 0.6087 -
   accuracy: 0.7500 - val_loss: 0.9014 - val_accuracy: 0.6899 - lr: 3.3287e-04
Epoch 13/20
144/144 [==============================] - 10s 66ms/step - loss: 0.5768 -
   accuracy: 0.7644 - val_loss: 1.2896 - val_accuracy: 0.5714 - lr: 3.0119e-04
Epoch 14/20
144/144 [==============================] - 11s 78ms/step - loss: 0.5785 -
   accuracy: 0.7713 - val_loss: 0.9396 - val_accuracy: 0.6080 - lr: 2.7253e-04
Epoch 15/20
144/144 [==============================] - 39s 275ms/step - loss: 0.5553 -
   accuracy: 0.7740 - val_loss: 0.5647 - val_accuracy: 0.7735 - lr: 2.4660e-04
Epoch 16/20
144/144 [==============================] - 8s 55ms/step - loss: 0.5201 -
   accuracy: 0.8023 - val_loss: 0.7371 - val_accuracy: 0.7422 - lr: 2.2313e-04
Epoch 17/20
144/144 [==============================] - 9s 66ms/step - loss: 0.5440 -
   accuracy: 0.7853 - val_loss: 1.4777 - val_accuracy: 0.5767 - lr: 2.0190e-04
Epoch 18/20
144/144 [==============================] - 8s 53ms/step - loss: 0.5457 -
   accuracy: 0.7857 - val_loss: 0.8047 - val_accuracy: 0.6794 - lr: 1.8268e-04
Epoch 19/20
144/144 [==============================] - 9s 65ms/step - loss: 0.4947 -
   accuracy: 0.8145 - val_loss: 0.7524 - val_accuracy: 0.7178 - lr: 1.6530e-04
Epoch 20/20
144/144 [==============================] - 9s 59ms/step - loss: 0.5126 -
   accuracy: 0.7927 - val_loss: 0.6104 - val_accuracy: 0.7840 - lr: 1.4957e-04
```