NED University of Engineering and Technology

Artificial Intelligence & Expert Sytems

(CT-361)

PROJECT: *AI Powered Brain Tumor Detection System*

GROUP MEMBERS:

1. *Anum Mateen(CR-22002)*
2. *Mehak Ejaz (CR-22001)*
3. *Ayesha Majid(CR-22026)*
4. *Hafsa Usma (CR-22003)*

Contents

Brain Tumor Detection using CNN

1. Introduction

1.1. Background

Early detection of brain tumors is essential for improving patient outcomes and enabling timely intervention. Traditionally, tumor classification is performed through biopsy, which requires invasive brain surgery. While effective, this approach involves risks and delays in diagnosis.

With advancements in computational intelligence, non-invasive techniques are emerging to assist in tumor detection and classification. Convolutional Neural Networks (CNNs), a subset of deep learning, have proven highly effective in analyzing medical images such as MRI scans. These models can identify patterns and features associated with brain tumors, offering a faster and more reliable diagnostic alternative.

In this study, we explore two CNN-based models for brain tumor detection, aiming to enhance diagnostic precision and efficiency. These models have the potential to complement traditional methods, reducing reliance on invasive procedures while providing accurate results.

1.2. Objective

This study utilized MRI or CT scan images sourced from publicly available repositories such as *Kaggle* and The *Cancer Imaging Archive (TCIA)*. The dataset included labeled images categorized as "Tumor" and "No Tumor," covering various tumor types to ensure model diversity. Preprocessing was applied to standardize image resolution and format for compatibility with the CNN models. Ethical compliance was maintained by using de-identified, open-access datasets.

2. Literature Review

The detection and classification of brain tumors using advanced computational methods have been an area of significant research interest. Traditionally, methods relied on manual segmentation and feature extraction, which were time-consuming and prone to human error. However, with the advent of Convolutional Neural Networks (CNNs), automated and efficient approaches have gained prominence. Studies have demonstrated that CNNs outperform traditional machine learning methods due to their ability to learn spatial hierarchies directly from image data.

Recent works have explored the use of pre-trained models like *VGG19*, *ResNet*, and *EfficientNet*, leveraging transfer learning to improve classification accuracy on medical datasets. Despite these advancements, challenges such as imbalanced datasets, small sample sizes, and noise in medical images persist. Data augmentation and preprocessing techniques have proven essential for overcoming these limitations, enhancing model robustness and generalization. This study builds

upon these methodologies, employing CNNs for the non-invasive detection of brain tumors, aiming to bridge the gap between research advancements and clinical applications.

3. Methodology

3.1. Data Collection

This study utilized MRI or CT scan images sourced from publicly available repositories such as *Kaggle* and *The Cancer Imaging Archive (TCIA).* The dataset included labeled images categorized as "Tumor" and "No Tumor," covering various tumor types to ensure model diversity. Preprocessing was applied to standardize image resolution and format for compatibility with the CNN models. Ethical compliance was maintained by using de-identified, open-access datasets.

3.2. Data Preprocessing

3.2.1. Data Organization:

To facilitate the training and testing processes, the dataset was organized into separate folders based on class labels: Tumor and No Tumor. This structure allows for efficient preprocessing and model input. The Tumor folder contained 155 labeled images, while the No Tumor folder included 98 images. This class-wise segregation ensures a streamlined workflow for loading, augmenting, and splitting the dataset into training and testing subsets.

```
# Dataset
import zipfile

z = zipfile.ZipFile('brain_tumor_dataset.zip')

z.extractall()
```

```
folder = 'brain_tumor_dataset/yes/'
count = 1

for filename in os.listdir(folder):
    source = folder + filename
    destination = folder + "Y_" +str(count)+".jpg"
    os.rename(source, destination)
    count+=1
print("All files are renamed in the yes dir.")
```

```
All files are renamed in the yes dir.
```

Renaming the extension from jpeg,png to jpg

3.2.2. Exploratory Data Analysis:

3.2.2.1. Class Distribution

We examined the class distribution of the "Tumor" and "No Tumor" labels to check for any imbalance in the dataset. If imbalances were found, we considered techniques such as resampling or data augmentation during preprocessing to ensure the model received balanced data.

3.2.2.2. Visualization

Using visualization libraries like *matplotlib*, we explored a few MRI scans to better understand their properties. This helped us observe the image resolution, quality, and any visible patterns or inconsistencies that could impact the model's performance.

3.2.2.3. Image Characteristics

We analyzed image characteristics, such as the dimensions of the images and the distribution of pixel intensities. This step allowed us to assess how these factors might influence the model and whether noise in the images required special handling during preprocessing.

3.2.2.4. Metadata Analysis

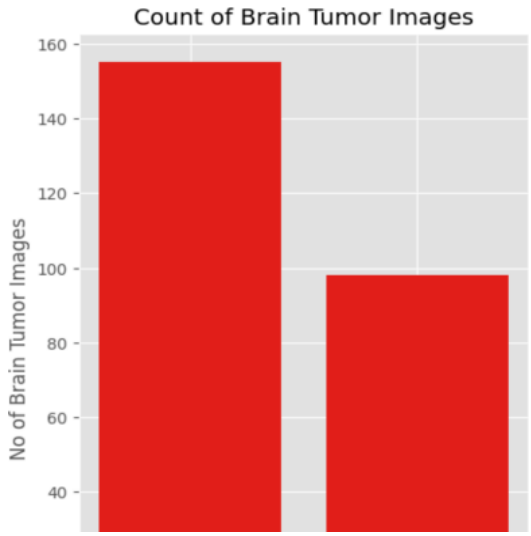
In cases where **metadata** was available, such as patient age or tumor location, we studied its correlation with the labels. This analysis helped identify potential relationships that could provide valuable insights and improve the model's predictive accuracy.

```
# EDA(Exploratory Data Analysis)
```

```
listyes = os.listdir("brain_tumor_dataset/yes/")
number_files_yes = len(listyes)
print(number_files_yes)

listno = os.listdir("brain_tumor_dataset/no/")
number_files_no = len(listno)
print(number_files_no)
```

```
155
98
```



3.2.3. Data Augmentation:

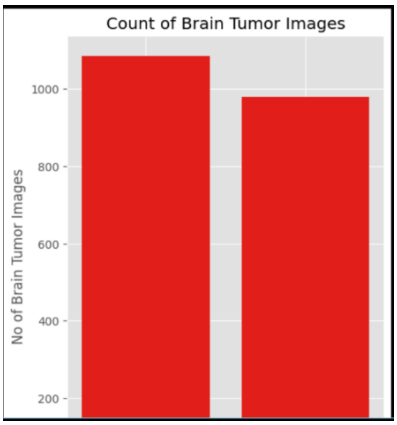
Data augmentation is a technique used to increase the diversity of a dataset by creating odified versions of existing data samples. It helps improve the model's robustness and reduces overfitting, especially when the dataset is limited.

Common Augmentation Techniques are:

- Rotation: Slight rotations simulate different orientations of images.
- Flipping: Horizontal or vertical flips introduce variability in image positioning.
- Zooming: Zoom-in or zoom-out effects emulate changes in scale.
- Brightness/Contrast Adjustment: Simulates varying lighting conditions to make the model adaptable to different environments.
- Cropping and Padding: Randomly crops or pads images, creating new perspectives within the same dataset.

These augmentations enhance the dataset, ensuring better generalization and improved model performance.

```
def augmented_data(file_dir, n_generated_samples, save_to_dir):
    data_gen = ImageDataGenerator(rotation_range=10,
                                  width_shift_range=0.1,
                                  height_shift_range=0.1,
                                  shear_range=0.1,
                                  brightness_range=(0.3, 1.0),
                                  horizontal_flip=True,
                                  vertical_flip=True,
                                  fill_mode='nearest')
    for filename in os.listdir(file_dir):
        image = cv2.imread(file_dir + '/' + filename)
        image = image.reshape((1,) + image.shape)
        save_prefix = 'aug_' + filename[:-4]
        i=0
        for batch in data_gen.flow(x = image, batch_size = 1, save_to_dir = save_to_dir, save_prefix = save_prefix, save_format = "jpg"):
            i+=1
            if i>n_generated_samples:
                break
```



Graph after data augmentation

3.2.4. Data Preparation:

Preparing the data is a crucial step to ensure compatibility with the Convolutional Neural Network (CNN) and optimize training performance. The preparing the data, the following steps were taken into consideration:

3.2.4.1. Resizing:

All images were resized to a consistent dimension, such as 128x128 or 224x224 pixels, to match the input requirements of the CNN model.

3.2.4.2. Normalization:

Pixel values were scaled to a range of 0 to 1 or -1 to 1. This step ensures faster convergence during model training by standardizing the input data.

3.2.4.3. Label Encoding:

Class labels ("Tumor" and "No Tumor") were converted into numerical form (e.g., 0 and 1) to be compatible with the model's output layer.

3.2.4.4. Splitting the Dataset:

The dataset was divided into three subsets:

- Training Set: 70% of the data for model learning.
- Validation Set: 20% of the data to tune hyper parameters and prevent overfitting.
- Test Set: 10% of the data to evaluate the model's performance on unseen data.

This structured data preparation ensures the CNN is trained effectively and generalizes well to new data.

3.2.5. Data Splitting:

To ensure a robust model evaluation and prevent data leakage, the dataset was split using the following strategies:

- Stratified Splitting:

  The class distribution was maintained across the training, validation, and test sets. This ensured that each subset represented the overall dataset's balance between "Tumor" and "No Tumor" classes, preventing skewed learning.

- Random Shuffle:

  Before splitting, the dataset was randomly shuffled to eliminate any inherent order or bias in the data. This step ensured that the subsets were representative and independent of their original sequence.

These measures were crucial to creating reliable splits for training, validation, and testing while preserving the integrity of the dataset.

3.3. Models Building

To classify brain tumor images, a Convolutional Neural Network (CNN) model was developed with the following design considerations:

3.3.1. Architecture Design

- **Input Layer**: Processes the preprocessed MRI images, ensuring compatibility with the CNN's input dimensions.
- **Convolutional Layers**: Extract spatial features from images using learned filters. These layers capture patterns such as edges, textures, and shapes.
- **Pooling Layers**: Down sample feature maps to reduce dimensionality and computation while retaining essential features.
- **Fully Connected Layers**: Translate the extracted features into class probabilities, enabling the model to predict the correct label.
- **Output Layer**: Utilized a *softmax activation function* for multi-class classification tasks or a *sigmoid activation function* for binary classification tasks.

3.3.2. Popular Architectures

To enhance performance, pre-trained models such as *VGG19*, *ResNet50*, or *MobileNet* were considered for transfer learning. These architectures leverage learned features from large datasets, improving accuracy and reducing training time.

This approach ensures a robust and efficient model capable of accurately classifying brain tumor images.

```
base_model = VGG19(input_shape = (240,240,3), include_top=False, weights='imagenet')

for layer in base_model.layers:
    layer.trainable=False

x=base_model.output
flat = Flatten()(x)

class_1 = Dense(4608, activation = 'relu')(flat)
drop_out = Dropout(0.2)(class_1)
class_2 = Dense(1152, activation = 'relu')(drop_out)
output = Dense(2, activation = 'softmax')(class_2)

model_01 = Model(base_model.input, output)
model_01.summary()
```

```
history_01 = model_01.fit(train_generator, steps_per_epoch=10, epochs = 15, callbacks=[es,cp,lrr], validation_data=valid_generator)
                                                                                                                        Python
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `su
  self._warn_if_super_not_called()
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.5401 - loss: 0.7346
Epoch 1: val_loss improved from inf to 0.73161, saving model to model.keras
10/10 ━━━━━━━━━━━━━━━━━━━━ 57s 4s/step - accuracy: 0.5416 - loss: 0.7345 - val_accuracy: 0.4805 - val_loss: 0.7316 - learning_rate: 1.0000e-04
Epoch 2/15
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/callback_list.py:96: UserWarning: Learning rate reduction is conditioned on metric `val_acc
  callback.on_epoch_end(epoch, logs)
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 429ms/step - accuracy: 0.5287 - loss: 0.7486
Epoch 2: val_loss improved from 0.73161 to 0.69431, saving model to model.keras
10/10 ━━━━━━━━━━━━━━━━━━━━ 18s 2s/step - accuracy: 0.5284 - loss: 0.7494 - val_accuracy: 0.5357 - val_loss: 0.6943 - learning_rate: 1.0000e-04
Epoch 3/15
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step - accuracy: 0.5376 - loss: 0.7296
```

Training the model

3.4. Streamlit Application

A *Streamlit application* was developed to provide an interactive interface for the brain tumor detection model. Users can upload MRI images through the app and receive real-time predictions on whether a tumor is detected. The application features a simple, user-friendly interface and displays the uploaded image alongside the prediction for clarity and transparency. This ensures accessibility and ease of use for practical deployment.

4. Implementation

4.1. Software and Tools

For data preprocessing and model implementation, several tools and libraries were employed. *Pandas* and *NumPy* were used for managing structured data and numerical computations, while *Matplotlib* and *Seaborn* facilitated visualizations to explore patterns and data distributions. *OpenCV (cv2)* was utilized for image processing tasks such as resizing and augmentation, with additional support from *Matplotlib.image* for handling image data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os, shutil
import cv2
import matplotlib.image as mpimg
import seaborn as sns
%matplotlib inline
plt.style.use('ggplot')
```

For data augmentation, frameworks like *TensorFlow/Keras* and its *ImageDataGenerator* were employed. *PyTorch* and its torchvision.transforms module were also considered as alternatives for this purpose. The visualization style was enhanced using the *ggplot style* in *Matplotlib* to ensure consistent and clear outputs. These tools streamlined the entire process from data preprocessing to model implementation.

5. Results and Analysis

5.1. Improve Model Performance

To enhance the model's performance, transfer learning techniques were utilized:

5.1.1. Transfer Learning:

A pre-trained CNN, such as *ResNet* or *EfficientNet*, trained on a large dataset like ImageNet, was employed. This leveraged the network's pre-learned feature extraction capabilities for the brain tumor classification task.

5.1.2. Freezing Layers:

The initial layers of the pre-trained network were frozen to preserve their generic feature extraction functions, such as detecting edges and textures.

5.1.3. Fine-Tuning:

The later layers were unfrozen and retrained on the brain tumor dataset. This step allowed the model to learn domain-specific features, improving its performance on the task at hand.

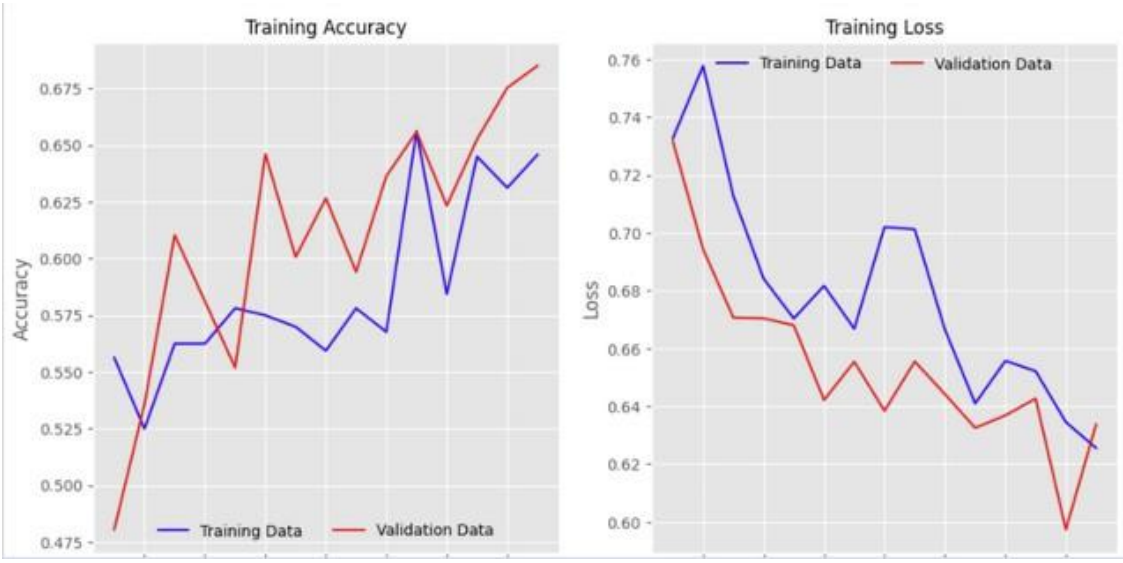
This approach ensured efficient use of computational resources while achieving high accuracy in classifying brain tumor images.

5.2. Training Accuracy and Loss

The performance of the model was evaluated on validation and testing datasets. The Validation Loss was recorded as 0.6172, with a corresponding Validation Accuracy of 65.26%. On the testing dataset, the model achieved a Testing Loss of 0.6082 and a Testing Accuracy of 66.77%. These results indicate that the model demonstrates reasonable performance and generalization on unseen data.

```
print(f'Validation Loss: {vgg_val_eval_01[0]}')
print(f'Validation Acc: {vgg_val_eval_01[1]}')
print(f'Testing Loss: {vgg_test_eval_01[0]}')
print(f'Testing Acc: {vgg_test_eval_01[1]}')
```

```
Validation Loss: 0.6172096133232117
Validation Acc: 0.6525974273681641
Testing Loss: 0.6081578731536865
Testing Acc: 0.6677419543266296
```



6. Conclusion

This project successfully employed Convolutional Neural Networks (CNNs) to classify brain tumor images, achieving a testing accuracy of 66.77%. Key steps included rigorous data preprocessing, augmentation, and transfer learning using pre-trained architectures to enhance performance. A user-friendly Streamlit application was developed, allowing real-time predictions of tumor presence based on MRI images. The application provides a practical solution for non-invasive and accessible tumor detection. Future work can focus on integrating larger datasets, optimizing model architectures, and improving classification accuracy for more reliable and robust diagnostic support.