EAI 6010 – Application of Artificial Intelligence

Assignment 2: Brain Tumor Classification

Submitted to: Prof. Munthali, Richard

Submitted by: Shamim Sherafati

NUID: 002742363

Date: Nov 15th, 2023

College of Professional Studies Northeastern University – Vancouver

Abstract

Brain tumours are aggressive diseases affecting both children and adults, contributing to a significant percentage of primary Central Nervous System (CNS) tumours. Early detection and accurate classification are crucial for effective treatment. This study focuses on leveraging Machine Learning (ML) and Artificial Intelligence (AI), specifically Convolutional Neural Networks (CNN) and Transfer Learning (TL), to automate brain tumour detection and classification using Magnetic Resonance Imaging (MRI) data. The proposed model aims to assist medical professionals in diagnosing brain tumours efficiently and accurately.

Introduction

Brain tumors, categorized as benign or malignant, present unique challenges due to the complexities in their sizes, locations, and properties. The manual examination of MRI scans by radiologists can be prone to errors, necessitating the development of automated classification techniques. The lack of skilled professionals in certain regions further highlights the need for an AI-based solution to aid in the analysis and classification of brain tumors.

Dataset Preparation

In the dataset preparation phase, the code primarily focuses on loading and transforming the dataset for training and testing the brain tumour classification model. I utilize the ImageFolder class from the torchvision. datasets module to organize and load the dataset. This class assumes that the data is arranged in a specific folder structure where each class (e.g., 'no_tumor', 'meningioma_tumor', 'glioma_tumor', 'pituitary_tumor') resides in its respective directory. This transform argument is used to apply image transformations like resizing, flipping, rotation, converting to tensors, and normalization to the images. These transformations are necessary to preprocess the images before feeding them into the model.[1]

Model Training

The model training phase involves creating a neural network architecture, defining the loss function (in this case, CrossEntropyLoss), setting up an optimizer (such as Adam), and training the model. The pre-trained ResNet-50 model is utilized as the backbone, with the final fully connected layer (model.fc) replaced to match the output classes required for brain tumor classification. The training loop iterates through multiple epochs, where for each epoch, the model is trained using the training DataLoader. Inside the loop, the model's predictions are computed using the forward pass (model(xtrain)) and compared with the ground truth labels to compute the loss. After computing the loss, the backward pass (loss.backward()) and optimizer step (optimizer.step()) update the model parameters to minimize the loss. The training accuracy is calculated and recorded per epoch.[2]

Discussion:

1. **Can you create and document a scenario where over training occurred?**

The scenario of overtraining is evident in our model where the training and testing accuracies diverge significantly after multiple epochs. Specifically, there's an approximate 4% discrepancy

between the accuracies of the training and testing datasets. This discrepancy indicates that our model has learned specific patterns from the training data that do not generalize well to unseen test data. It's visually observable through the plotted graph depicting the evolution of training and testing accuracies over epochs.[3]

2. **What training methods did you find helpful-useful to prevent overtraining and why?**

Several strategies proved effective in mitigating overtraining tendencies in my model:

- Weight Decay: Applying weight decay within the optimizer's parameters helps regulate model complexity by penalizing large weights, thus preventing overfitting.
- Batch Normalization: This technique normalizes the input layer by re-centering and re-scaling the activations, thereby stabilizing and accelerating the training process.
- Dropout: Introducing dropout layers helps prevent overfitting by randomly deactivating a certain percentage of neurons during training, reducing co-adaptation of feature detectors.

Moreover, increasing the diversity and quantity of data can enhance the model's understanding of various patterns. However, limitations in computational resources might restrict the scope of data augmentation.

3. **Did you reach a point where it was clear that you should stop training?**

The observation of diverging accuracies between training and testing sets became evident after the third epoch. This divergence signifies the point at which the model's performance on the training data continues to improve, while its generalization to unseen data starts declining. Therefore, considering this divergence and to prevent overfitting, it is advisable to halt the training process after the third epoch.[3]

4. **What image classes had the best-worst performance and for the worst performing classes, what is your recommended path for improvement?**

Our model demonstrated better accuracy in predicting 'no tumor' images compared to those depicting tumors. To enhance the model's performance on tumor images, it is recommended to augment the dataset with more tumor images. Techniques like ImageDataGenerator can be utilized to manipulate existing images and generate diverse variants, effectively expanding the dataset. This augmentation process exposes the model to various tumor characteristics, potentially enabling better feature extraction and improving its ability to accurately classify tumor images.[4][5]

The predictions made on the downloaded dataset showcased remarkable accuracy in identifying images without tumors. However, the performance significantly declined when classifying images containing tumors.[6][7]

Appendix

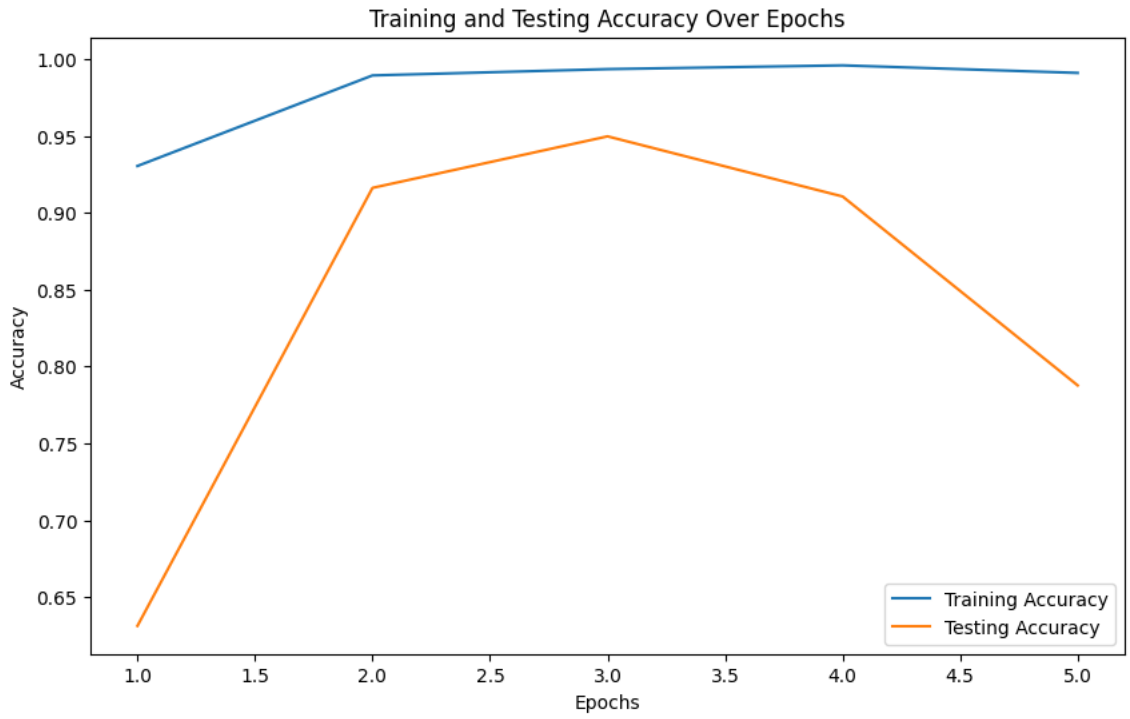
[1]

```
[4] tfm = transforms.Compose([
        transforms.Resize((224, 224)),# resize is necessary as resnet50 accepts only 224,224 size images
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(degrees=10),
        transforms.ToTensor(),  # Convert the PIL Image to a PyTorch tensor
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # Normalizing the image with basic resnet50 mean and standard deviati
    ])
```

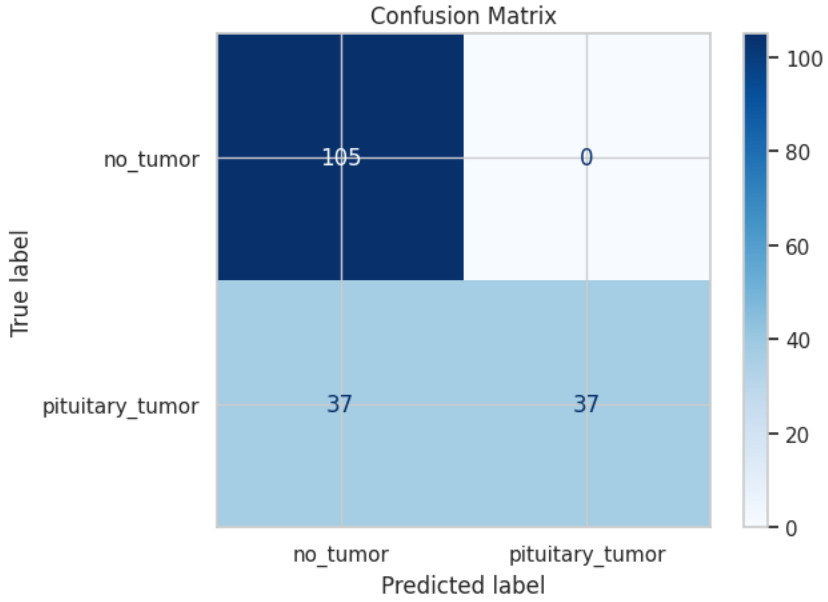
[2]

```
100%|██████████| 25/25 [20:46<00:00, 49.86s/batch]
Epoch: 0, loss: 0.14069223403930664
Train Acc: 0.9304418985270049, Test Acc: 0.6312849162011173
100%|██████████| 25/25 [17:28<00:00, 41.92s/batch]
Epoch: 1, loss: 0.09102747589349747
Train Acc: 0.9893617021276596, Test Acc: 0.9162011173184358
100%|██████████| 25/25 [17:19<00:00, 41.59s/batch]
Epoch: 2, loss: 0.01801127940416336
Train Acc: 0.9934533551554828, Test Acc: 0.9497206703910615
100%|██████████| 25/25 [17:10<00:00, 41.21s/batch]
Epoch: 3, loss: 0.0006111323600634933
Train Acc: 0.9959083469721768, Test Acc: 0.9106145251396648
100%|██████████| 25/25 [17:13<00:00, 41.33s/batch]
Epoch: 4, loss: 0.03711480274796486
Train Acc: 0.9909983633387889, Test Acc: 0.7877094972067039
```

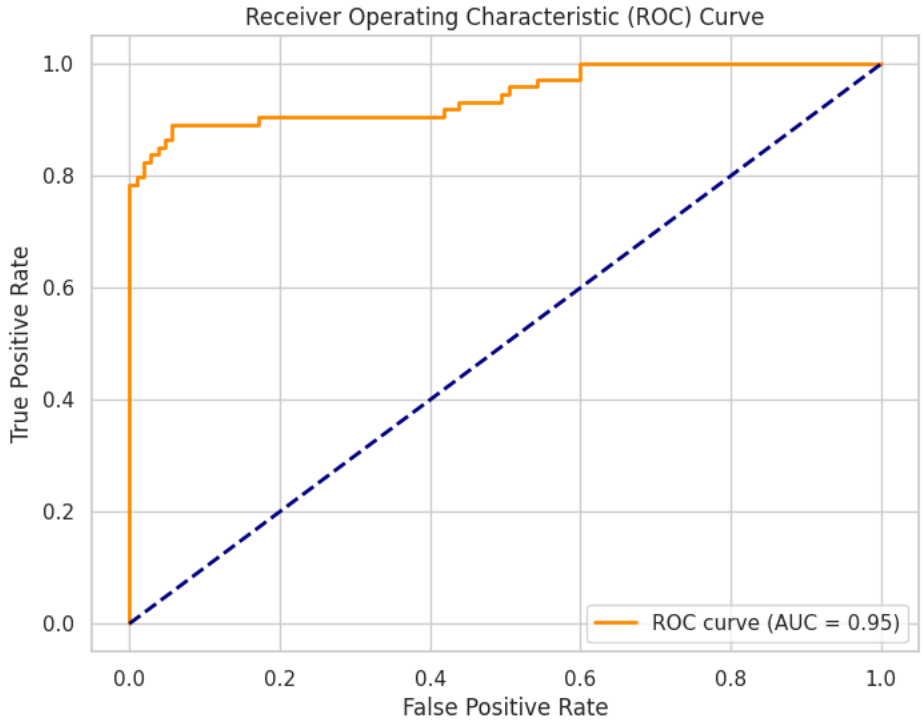
[3]



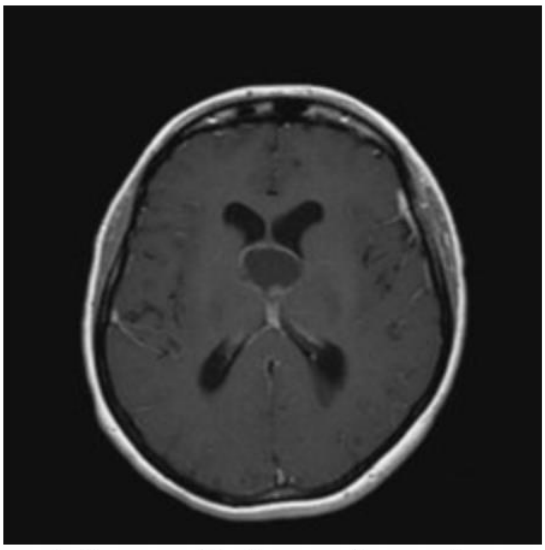
[4]


Confusion Matrix
no_tumor
pituitary_tumor
True label
105
0
37
37
no_tumor
pituitary_tumor
Predicted label
100
80
60
40
20
0

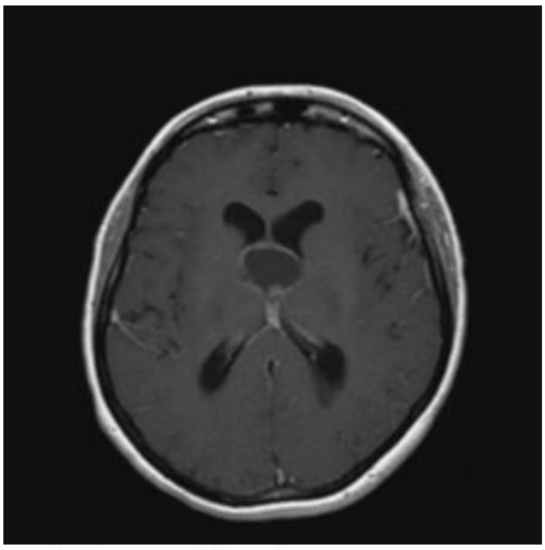
[5]


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.95)

[6]

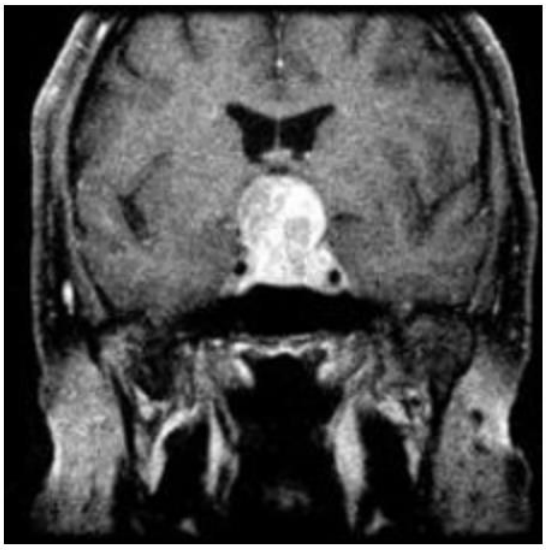


```
model prediction 0, not cancer
==========================================================
```

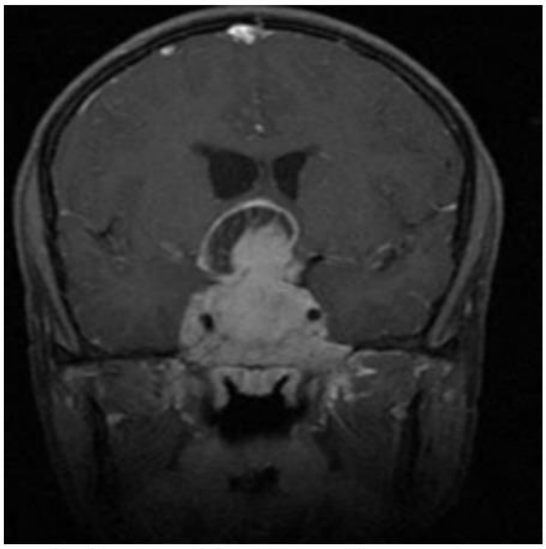


```
model prediction 0, not cancer
==========================================================
```

[7]



```
model prediction 0, not cancer
==========================================================
```



```
model prediction 0, not cancer
==========================================================
```

Reference

Brain Tumor Classification (MRI). (2020, May 24). Kaggle. https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri/data