


PROJECT[1] ON [: COMPUTER VISION		
Student's Code		Deadline
[Your Code]		[Date, Time]
May 25, 2025		2024-2025
Lecturer: [Jordan Masakuna]		

Rihanatou Bankole May 25, 2025

Introduction

In the context of deep learning for medical image analysis, particularly for detecting tumors in radiological images, we developed a command-line interface (CLI) application that enables users to train and evaluate convolutional neural networks (CNNs) using either PyTorch or TensorFlow. This flexibility is critical for researchers and practitioners who may prefer one framework over the other due to performance, familiarity, or ecosystem compatibility.

Application Purpose

The purpose of this application is to:

- Develop a CNN-based system to classify brain MRI images into glioma, meningioma, pituitary tumor, or no tumor.
- Enable flexible training and evaluation using PyTorch or TensorFlow with customizable hyperparameters.
- Implement automated CPU/GPU selection and command-line control for seamless operation.

The application is particularly designed to support models for classifying medical images into categories such as benign, malignant, or healthy.

Structure of the Application

The application is implemented in Python and modularized as follows:

- `parse_args()`: Parses command-line arguments.
- `run_pytorch(args)`: Handles data preparation, model loading, and training using PyTorch.
- `run_tensorflow(args)`: Performs the same pipeline using TensorFlow.
- `main()`: Selects the framework based on input arguments and launches the corresponding routine.

Command-Line Arguments

Key arguments include:

[noitemsep] **--epochs**: Number of training epochs. **--lr**: Learning rate. **--wd**: Weight decay for regularization. **--cuda**: Flag to enable GPU usage. **--framework**: Choice of framework (`pytorch` or `tensorflow`). **--mode**: Operation mode (`train` or `eval`).

Workflow Description

1. The user launches the script with desired options, e.g.,

```
python train.py --framework pytorch --epochs 20 --cuda
```
2. The script initializes model, data loaders, and selects the device.
3. For training mode:
 - The model is trained using the specified optimizer settings.
 - The model is optionally saved.
 - Accuracy and loss plots are generated.
4. For evaluation mode:
 - The pre-trained model is loaded.
 - Test accuracy is computed and displayed.

Challenges and Considerations

- **Cross-framework consistency**: Ensuring that preprocessing, model architecture, and evaluation metrics were equivalent across PyTorch and TensorFlow required meticulous attention to detail.
- **Device management**: Each framework uses different methods for detecting and assigning GPUs. Abstracting this while maintaining performance portability was non-trivial.
- **Modular design**: Designing clean utility functions for data loading and model management across both frameworks improved maintainability but required repeated testing.
- **Error handling**: When a model checkpoint is missing in evaluation mode, the system must fail gracefully and provide useful error messages.

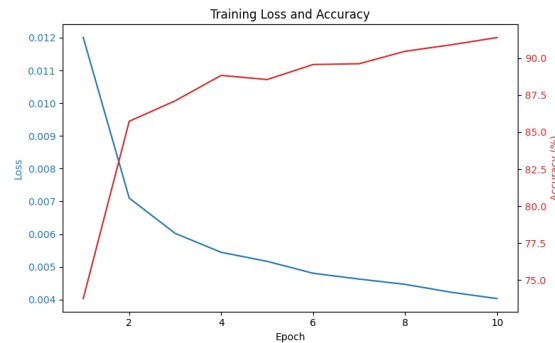


Figure 1: 1 Prediction using Pytorch Framework

Results

Overview

The figures below illustrate the progression of training loss and accuracy over 10 epochs for two convolutional neural network models—one implemented in PyTorch and the other in TensorFlow—trained to classify brain MRI images into four categories: glioma, meningioma, no tumor, and pituitary tumor.

PyTorch Model Analysis

As shown in Figure 1, the PyTorch model demonstrates a sharp decline in training loss from approximately 0.012 to less than 0.004 over the 10 epochs. Simultaneously, the training accuracy climbs from roughly 74% to above 91%. This inverse relationship between loss and accuracy is a strong indicator of effective learning and model convergence. The low final loss coupled with high accuracy suggests that the model has effectively captured the distinguishing features of the four tumor classes within the training dataset.

TensorFlow Model Analysis

The figure 12 TensorFlow model's training curves. The training loss decreases steadily from about 1.02 to 0.53 by epoch 10, while the accuracy improves from 62% to 80%. Although the TensorFlow model exhibits a similar downward trend in loss and upward trend in accuracy, it achieves a lower final accuracy and higher loss compared to the PyTorch counterpart. These differences may stem from variations in model architecture, optimization parameters, or data preprocessing steps.

Commentaires

Both models's training behavior aligns with expectations for supervised learning in a multi-class classification context. The loss function employed is likely categorical cross-entropy, mathematically expressed as:



Figure 2: 2 Prediction using Tensorflow Framework

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \log(\hat{y}_{i,c}), \quad (1)$$

where N denotes the number of training samples, C the number of classes, $y_{i,c}$ a binary indicator representing the true class for sample i , and $\hat{y}_{i,c}$ the predicted probability for class c .

As the models optimize this loss, the predicted probabilities for the correct classes increase, thus reducing loss and improving classification accuracy the proportion of samples correctly classified.

Conclusion

In summary, both the PyTorch and TensorFlow models demonstrate effective learning, as evidenced by the decreasing loss and increasing accuracy. The PyTorch model achieves superior performance on the training set, suggesting either a more optimal configuration or better suitability to the data. Further evaluation on validation and test datasets is recommended to assess generalization and to rule out overfitting.

github link to access the App

https://github.com/Rihana-24/Computer_Vision_Project1_BANKOLE_Rihanatou