PyTorch Porting: Explainable Deep Learning for Dementia Diagnosis

explAInation Group

German Center for Neurodegenerative Diseases (DZNE)
Rostock

April 29, 2025

Project Overview

This design specification aims to create a roadmap for porting and modernizing the functionality of an existing Keras-based deep learning pipeline for dementia-related brain MRI analysis — originally developed at https://github.com/martindyrba/Experimental — using the PyTorch ecosystem. The goal is not only to reproduce the core data loader, preprocessing, classification, and explainable AI (XAI) capabilities, but also to enhance the system's modularity, interpretability, and maintainability through machine learning (ml) engineering best practices. This is structured to adhere to clean architecture principles, support plug-and-play configuration of components, and enable robust XAI techniques such as Gradient-Weighted Class Activation Map (Grad-CAM), Layer-wise Relevance Propagation (LRP), etc. This porting process would incorporate ml engineering considerations like reproducible experimentation, logging, efficient data handling, and scalable model training — all critical to deploying reliable and interpretable models in clinical research contexts, particularly for neuro-degenerative diseases such as dementia.

1 High-Level System Architecture

This provides a modular overview of the XAI end-to-end pipeline applied to dementia diagnosis using brain magnetic resonance data. It highlights the primary building blocks, including data loading, preprocessing, model training, XAI techniques, configuration handling, and output generation. Each module is designed to be reusable, extensible, and integrated with the rest of the system.

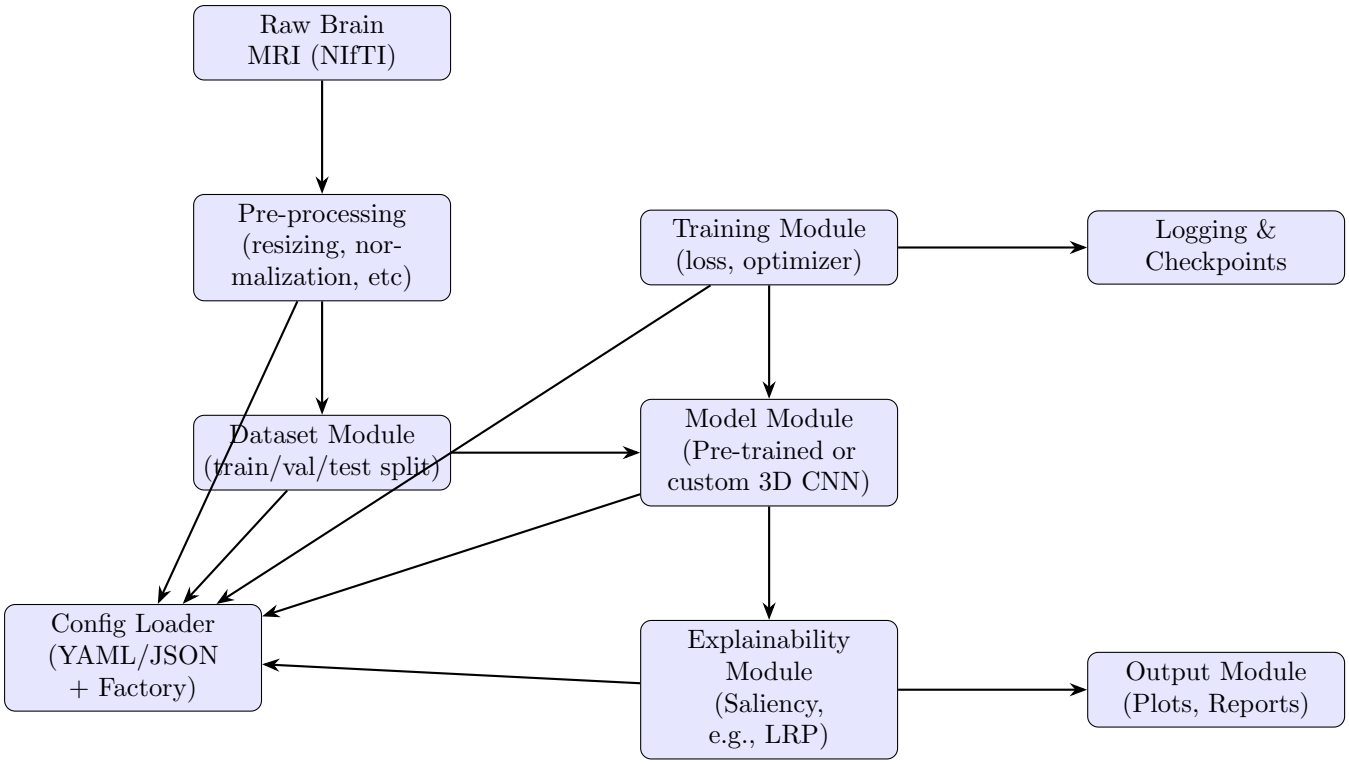

Figure 1: A high-level functional system overview of the different modules and their interactions.

2 Data Flow Diagram (DFD)

This diagram illustrates the flow of data through the system, starting from the raw MRI inputs to the final outputs such as model classification scores, relevance maps, and performance scores. It emphasizes the sequential processing stages — from data loading & transformation to model training, inference, and interpretation — offering a clear conceptual information of how information propagates through the pipeline.

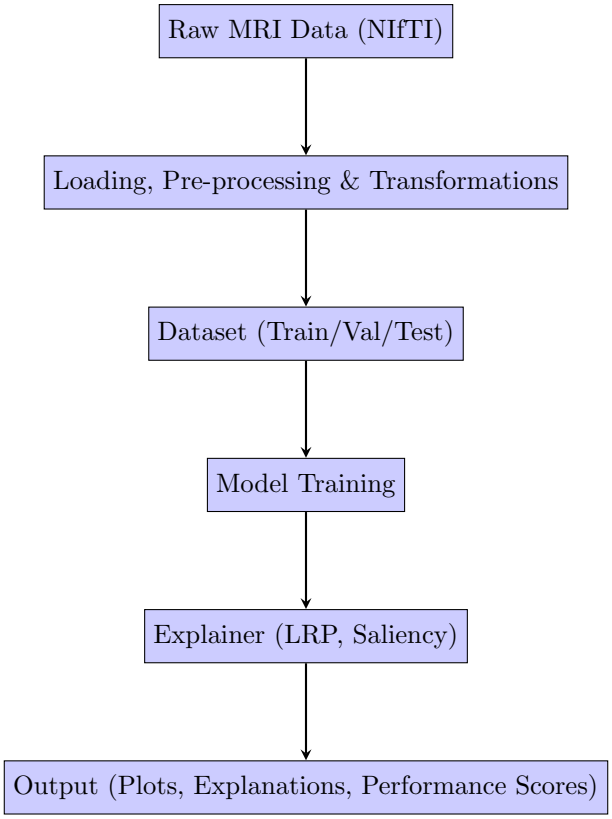

Figure 2: Data Flow Diagram

3 Project Directory Structure

The project directory is structured to promote clarity, modularity, and maintainability. The depiction below visualizes the folder hierarchy and includes inline descriptions for each file and module, explaining their specific roles in the system. By adhering to this organized structure, contributors can quickly navigate the codebase, modify components independently, and scale things efficiently.

```
explainable_mri_dementia/ // Main project root
├── config/ // YAML/JSON configs for experiments
│   └── base_config.yaml // Default hyperparameters & paths
├── data/ // Data loading & preprocessing
│   ├── transforms.py // Image transformations & augmentations
│   ├── nifti_loader.py // Load NIfTI brain MRI files
│   └── dataset_factory.py // Create datasets from configs
├── models/ // Neural network architectures
│   ├── base_model.py // Abstract base model class
│   ├── cnn_backbones.py // 3D CNN variants ResNet
│   └── model_factory.py // Model builder using config
├── explainers/ // XAI methods for interpretability
│   ├── lrp.py // LRP relevance map(s) generation
│   ├── vanilla.py // Vanilla saliency map(s) from gradients
│   ├── ...
│   └── explainer_factory.py // Factory to instantiate explainers
├── training/ // Training and evaluation logic
│   ├── trainer.py // Main training loop
│   ├── losses.py // Custom loss functions
│   └── metrics.py // Evaluation metrics
├── utils/ // Miscellaneous utilities
│   ├── logger.py // Logging utilities
│   ├── visualization.py // Plotting outputs & results
│   └── nifti_utils.py // Helpers for NIfTI file handling
├── scripts/ // Entry points and CLI scripts
│   └── train.py // Train a model via command line
├── results/ // Experimental logs and outputs
│   ├── exp_01/ // Configs, plots, logs, metrics' scores from experiment 01
│   ├── exp_02/ // Experiment 02 outputs
│   ├── ...
│   └── exp_n/ // Experiment n outputs
├── tests/ // Unit tests for modules [Optional]
├── requirements.txt // Project dependencies
└── README.md // Overview and usage instructions
```

4 Technology Stack

Below is the proposed technology stack for the PyTorch porting project, with versions, URLs, and justifications for each choice:

- **Python (3.12):** `https://www.python.org/downloads/` Chosen for its stability, extensive ML ecosystem, and compatibility with PyTorch and auxiliary libraries.
- **PyTorch (2.6.0):** `https://pytorch.org/` Selected for its dynamic computation graph, native CUDA support, and strong community backing.
- **PyTorch Lightning (2.0.5, optional):** `https://www.pytorchlightning.ai/` Simplifies boilerplate around training loops, logging, and distributed setups, enhancing reproducibility.

- **MONAI (1.4.0):** `https://monai.io/` Provides domain-specific transforms, network architectures, and metrics tailored for medical imaging.
- **TorchIO (0.20.7):** `https://torchio.readthedocs.io/` Efficient 3D medical image preprocessing and augmentation utilities integrated with PyTorch datasets.
- **NiBabel (5.3.2):** `https://nipy.org/nibabel/` Robust handling of NIfTI and other neuroimagingfile formats essential for IO of MRI data.a.
- **Captum (0.8.0):** `https://captum.ai/` Official PyTorch library for integrated gradients, saliency, and other state-of-the-art XAI methods.
- **Click (8.1+):** `https://click.palletsprojects.com/`
  A composable command line interface (CLI) toolkit that simplifies user interaction, improves reproducibility, and enables intuitive execution of key workflows such as data processing, training, and evaluation.
- **PyYAML (6.0.2):** `https://pyyaml.org/` Lightweight YAML parser for reading experiment and model configuration files.
- **Git (2.39):** `https://git-scm.com/` Distributed version control to track code changes and collaborate efficiently.
- **TensorBoard (2.19.0, optional):** `https://tensorboard.dev/` Visualization of training metrics and XAI saliency maps.
- **Matplotlib (3.10.1):** `https://matplotlib.org/` Fundamental plotting library for custom visualizations in notebooks and scripts. Specialized neuroimaging visualization utilities to overlay maps on brain templates.
- **CUDA Toolkit (12.6):** `https://developer.nvidia.com/cuda-toolkit` GPU acceleration support for PyTorch models.

5 Expected Deliverables

The deliverables of this PyTorch porting project span multiple layers of functionality, usability, and extensibility. These components will be structured to support both research reproducibility and practical deployment in ML pipelines for neuroimaging.

- **Core Functionality:**
  - End-to-end data pipeline for brain MRI in NIfTI format, including loading, pre-processing, transformation(s), and dataset preparation (train/val/test splits).
  - Re-implementation of core deep learning models (e.g., 3D CNN variants) using PyTorch, modular and extensible via a simple config-based initialization.
  - Integrated support for XAI techniques such as LRP for model inference explainability.
- **Software Architecture:**
  - Clean, modular directory structure separating concerns across configuration, models, training logic, explainers, and utilities.
  - Factory patterns and dependency injection via YAML/JSON configs for reproducible and dynamic component assembly.
- **User Interfaces:**
  - **Command-Line Interface (CLI):** Script-based interaction (e.g., `train.py`) with CLI arguments for model selection, data paths, experiment name, and config overrides.
  - **Notebook Support:** Jupyter notebooks for interactive experimentation, visualization, and debugging during research and development.

  - **Python API / Scripted Use:** High-level classes or functions exposed via Python modules to allow programmatic access to data loading, model inference, training, and explanation routines.

- **Outputs and Results:**
  - Standardized logging of metrics' scores/results, losses, and artifacts (e.g., checkpoints) for each experiment.
  - Automated generation of visualizations such as relevance maps.
  - Exportable reports of model performance and explanation overlays for clinical interpretability and further analysis.