MLP Classification of Alzheimer's Disease MRI Images

1. Problem

The goal of this project is to build a classifier that distinguishes MRI brain scans of subjects with Alzheimer's Disease (AD) from those of cognitively normal (CN) subjects. The input consists of 3D structural MRI volumes in NIfTI format. The task is a supervised binary classification problem, and the model must be a multi layer perceptron (MLP) implemented using PyTorch.

The main difficulty is the extremely small dataset size. There are only 9 volumes for AD and 9 volumes for CN, and only 6 from each class are used for training. This makes overfitting very likely and limits how complex the model can be.

2. Data

The dataset is a small subset of ADNI provided for this project. It contains:

- 9 NIfTI volumes for Alzheimer's Disease (AD)
- 9 NIfTI volumes for Cognitive Normal (CN)

The files are organized in two folders:

- ADNI/AD for Alzheimer's Disease subjects
- ADNI/CN for Cognitive Normal subjects

Train and test split:

- Training set: 6 AD volumes and 6 CN volumes, 12 volumes total
- Test set: 3 AD volumes and 3 CN volumes, 6 volumes total

Preprocessing

Each MRI is a 3D volume. To convert each volume into a 2D input for the MLP I use the following steps:

1. Load the NIfTI volume as a floating point array.
2. Take the middle slice along the last axis of the volume to obtain a 2D image. This gives a central brain slice and keeps the pipeline simple.
3. Apply intensity normalization:
   - First perform z score normalization using the mean and standard deviation of the slice.
   - Then shift and rescale the slice so that values lie in the range [0, 1].
4. Resize the 2D slice to 128 by 128 pixels using bilinear interpolation.
5. Treat the result as a single channel grayscale image.

The model therefore sees one 128 by 128 single channel slice for each subject instead of the full 3D volume. This reduces input size and keeps the MLP manageable at the cost of discarding some 3D information.

3. Method

3.1 Model architecture

The classifier is a feedforward multi layer perceptron that operates on a flattened image.

Input:

- A 128 by 128 grayscale slice
- Flattened to a vector of length $128 \times 128 = 16384$

- I chose two hidden layers with 256 and 64 neurons. This gives the network enough capacity to fit the data while keeping the number of parameters much smaller than the 16,384-dimensional input. With only 12 training examples, a larger or deeper MLP would overfit even more quickly. Dropout and weight decay are also used to regularize this relatively small network.

Network structure:

- Input layer: 16384 input units
- Hidden layer 1: 256 units, linear transformation followed by ReLU activation and Dropout with probability 0.3
- Hidden layer 2: 64 units, linear transformation followed by ReLU activation and Dropout with probability 0.3
- Output layer: 2 units (one for CN and one for AD), linear transformation

The network outputs raw logits for the two classes. These logits are passed to the PyTorch CrossEntropyLoss function, which internally applies softmax and computes the cross entropy loss.

3.2 Justification of architecture and parameters

The architecture is chosen with the data size in mind.

- The input is high dimensional (16384 features) but there are only 12 training examples. A deep or very wide network would overfit immediately.
- Two hidden layers provide some nonlinearity while keeping the number of parameters under control. The widths 256 and 64 give a gradual reduction in dimensionality from the input to the final representation.
- Dropout with probability 0.3 is used on both hidden layers as a regularization method. It randomly drops some activations during training, which can reduce overfitting.
- L2 weight decay is applied through the optimizer to penalize large weights and further regularize the model.

- A convolutional neural network would usually be more suitable for image data, but the project requirements ask specifically for an MLP, so the model is intentionally kept in this form.

3.3 Training procedure

Implementation details:

- Framework: PyTorch
- Loss function: CrossEntropyLoss
- Optimizer: Adam
- Learning rate: 1 × 10^-3
- Weight decay: 1 × 10^-4
- Batch size: 4
- Number of epochs: 50
- Random seed: 42
- Device: CPU

At each epoch:

1. The model is trained on the 12 training examples using mini batches of size 4.
2. After the training step, the model is evaluated on the 6 test examples.
3. Training loss, training accuracy, test loss, and test accuracy are recorded.

After the final epoch, a classification report is computed on the test set using scikit learn. The report includes precision, recall, and F1 score for each class.

4. Results

The training and testing logs show the expected pattern for such a small dataset.

Training performance:

- Training accuracy rises rapidly and reaches 100 percent.
- Training loss drops close to zero.

This indicates that the MLP can completely memorize the 12 training examples.

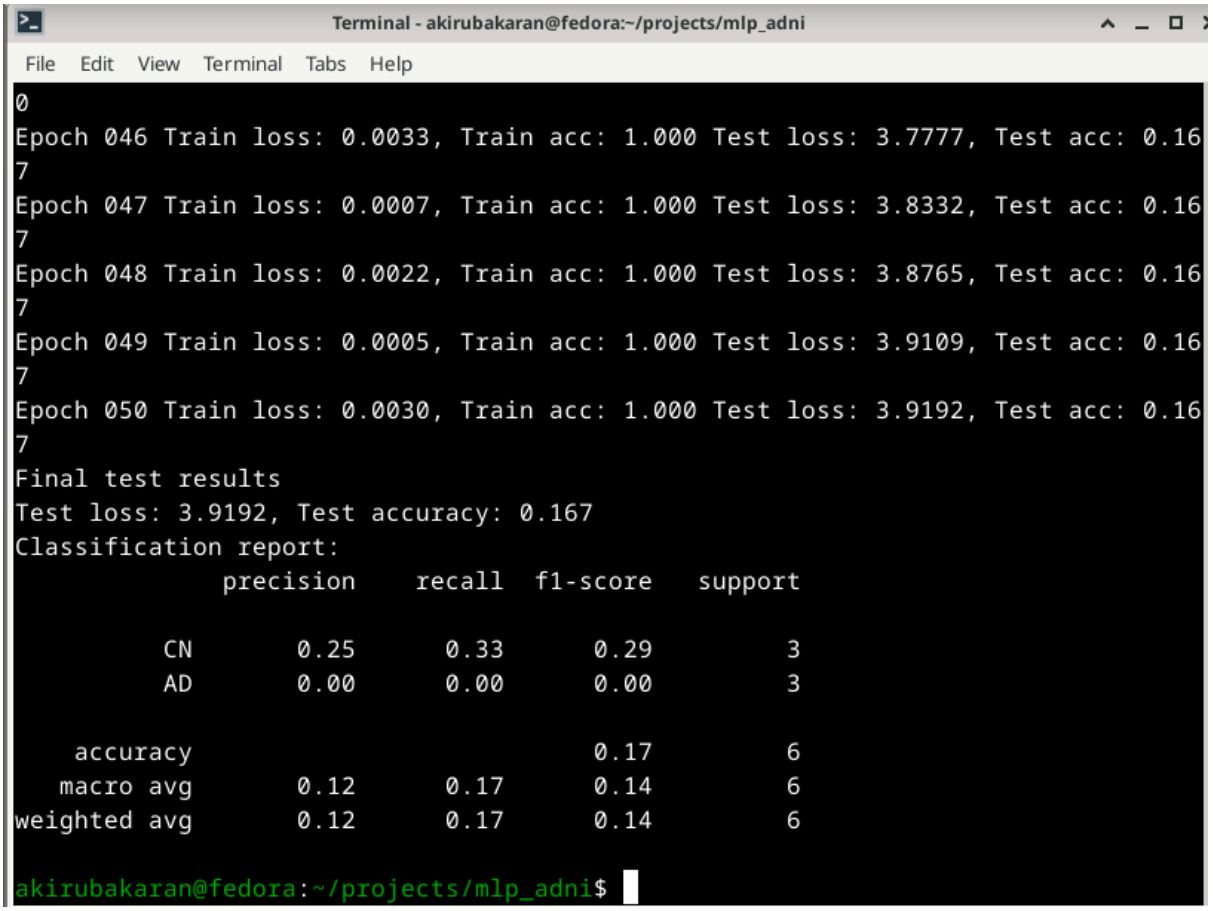
Test performance:

- Test accuracy is low and unstable across epochs. In the reported run, the final test accuracy is about 16.7 percent, which corresponds to 1 correct prediction out of 6 test samples.
- Test loss increases as training progresses, which is a classic sign of overfitting.

The final classification report on the 6 test examples is:

- CN: precision about 0.25, recall about 0.33, F1 score about 0.29

- AD: precision 0.00, recall 0.00, F1 score 0.00
- Overall accuracy: about 0.17



In some epochs the model predicts mainly one class for all test samples. In those cases precision and recall for the other class are effectively zero. With such a small test set, small changes in the output cause large swings in the metrics.

Interpretation:

- The MLP easily memorizes the training data due to the small dataset.
- Generalization to unseen data is poor. The small number of samples and the high dimensional input lead to strong overfitting.
- Using only a single 2D slice from each 3D volume loses significant information that could help distinguish AD from CN.