Multiclass Classification of MRI Images Report

Introduction

This report discusses the implementation of a multiclass classification model for MRI images. The model is designed to classify images into four classes using Convolutional Neural Networks (CNN). The code provided uses the PyTorch framework, a popular deep learning library, for constructing and training the neural network.

Background

MRI Image Classification

Magnetic Resonance Imaging (MRI) is a widely used medical imaging technique that produces detailed images of the internal structures of the body. Multiclass classification of MRI images is crucial in medical diagnostics, as it enables the automated identification of different anatomical structures or pathological conditions.

PyTorch

PyTorch is an open-source deep learning library that provides a flexible and dynamic computational graph, making it particularly well-suited for research and experimentation. It allows developers to define and train complex neural network architectures with ease.

Code Overview

Libraries Used

torch: Core PyTorch library for building and training neural networks.

matplotlib: Used for data visualization, particularly for plotting training loss curves.

pandas: Used for data manipulation and analysis, although not extensively used in the provided code.

NumPy: Used for numerical operations, especially for handling image data arrays.

torchvision: Part of PyTorch, provides utilities for working with image data.

transforms: PyTorch module for defining image transformations during data loading.

ImageFolder: PyTorch class for loading images from a directory and applying transformations.

Data Loading and Preprocessing

The code uses the `ImageFolder` class from torchvision to load training and testing data. Image data augmentation techniques such as color jittering, random horizontal flipping, and resizing are applied to the training data using the `transforms.Compose` module.

Neural Network Architecture

The neural network architecture is defined in a class named `CNN`. It is a convolutional neural network with multiple convolutional layers followed by max-pooling layers. The architecture is designed to capture hierarchical features in the input images. The final fully connected layers output the predicted class probabilities.

Training

The training process is defined in the `train` function. It uses the cross-entropy loss and the Adam optimizer. The model is trained for a specified number of epochs, and the training loss is printed for each epoch. The training loss is also stored for later analysis.

Evaluation

The `accuracy` function is defined for evaluating the model on the test set. It calculates the accuracy by comparing the predicted labels with the true labels. The accuracy is then printed, representing the model's performance on unseen data.

Results

The code concludes by printing the test set accuracy achieved by the trained CNN model.

Conclusion: The provided code demonstrates the process of building, training, and evaluating a CNN model for multiclass classification of MRI images using PyTorch. This type of model can be a valuable tool in medical image analysis, aiding in the automated diagnosis of various medical conditions. The use of PyTorch provides a flexible and efficient framework for developing deep learning models. Further research and fine-tuning may be necessary to optimize the model for specific medical imaging datasets.

1. Importing Libraries

```
import torch
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import torchvision
from torchvision import transforms, datasets
from torchvision.datasets import ImageFolder
```

2. Data Loading and Preprocessing

```
test_data_dir = '/content/Testing'
data_dir = "/content/Training"

train_transforms = transforms.Compose([
    transforms.ColorJitter(),
    transforms.RandomHorizontalFlip(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])

train = ImageFolder(data_dir, transform=train_transforms)
test = ImageFolder(test_data_dir, transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((224, 224)),
    transforms.Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
]))

train_loader = torch.utils.data.DataLoader(train, batch_size=19, shuffle=True)
test_loader = torch.utils.data.DataLoader(test, batch_size=48, shuffle=True)
```

Explanation:

- **test_data_dir, data_dir:** Directories containing test and training image data.
- **train_transforms:** Sequence of image transformations applied to the training data.
- **ImageFolder:** Loading image data from directories, applying transformations.
- **train_loader, test_loader:** PyTorch data loaders for efficiently loading batches of data during training and testing.

3. Neural Network Architecture

```
class CNN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = torch.nn.Sequential(
            # Convolutional and pooling layers
            # ...
            # Fully connected layers
            # ...
        )

    def forward(self, x):
        return self.model(x)
```

Explanation:

- **CNN class:** Inherits from `torch.nn.Module` and defines the architecture of the convolutional neural network.
- **torch.nn.Sequential:** Sequential container for organizing the layers of the neural network.
- Convolutional and pooling layers are defined to capture hierarchical features.
- Fully connected layers for final classification.

4. Training Function

```
def train(model, optimizer, criterion, train_loader, num_epochs, device, display_step=None):
    # ...
    # Training loop
    # ...
```

Explanation:

- **train function:** Takes the model, optimizer, criterion, data loader, number of epochs, and device for training.
- Training loop iterates over epochs and batches, calculates loss, and performs backpropagation.
- Loss is printed for each epoch, and a list of training losses is maintained.

5. Evaluation Function

```
def accuracy(model, test_loader, device):
    # ...
    # Evaluation loop
    # ...
    return test_acc
```

Explanation:

- **accuracy function:** Evaluates the model on the test set and calculates accuracy.
- Iterates over batches, calculates predicted labels, and compares them with true labels.
- Returns the total accuracy.

6. Model Training

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
cnn = CNN().to(device)

num_epochs = 60
learning_rate = 0.001
weight_decay = 0.01
criterion = torch.nn.CrossEntropyLoss()
optimizer1 = torch.optim.Adam(cnn.parameters(), lr=learning_rate, weight_decay=weight_decay)

train(cnn, optimizer1, criterion, train_loader, num_epochs, device, display_step=None)

acc1 = accuracy(cnn, test_loader, device)
print(f"Test set accuracy using cnn model = {100 * acc1 / len(test)} %")
```

Explanation:

- **Device check:** Determines whether to use GPU or CPU based on availability.
- **Model instantiation:** Creates an instance of the CNN model and moves it to the specified device.
- **Hyperparameters:** Learning rate, weight decay, and criterion for model training.
- **Adam optimizer:** Optimizer for updating model parameters during training.

- **Model training:** Calls the `train` function to train the model for a specified number of epochs.
- **Model evaluation:** Calls the `accuracy` function to calculate the accuracy on the test set.

This code represents a comprehensive pipeline for loading, preprocessing, building, training, and evaluating a CNN model for multiclass classification of MRI images using PyTorch.