MNIST Data Preprocessing with PyTorch Documentation

This document describes the code for preprocessing the MNIST handwritten digit dataset using PyTorch.

Libraries

The code imports the following libraries:

- torch: The core PyTorch library for deep learning.
- torchvision: A library for computer vision tasks within PyTorch, including datasets and transformations.
- matplotlib.pyplot: Used for generating visualizations of the images.

Basic Preprocessing

1. **Transform Definition:**

    - A transforms.Compose object is created to chain multiple image transformations.
    - transforms.ToTensor(): Converts the PIL image to a PyTorch tensor with normalized pixel values (between 0 and 1).
    - transforms.Normalize((mean,), (std,)): Normalizes the pixel values based on the provided mean and standard deviation. Here, it uses the mean and standard deviation specific to the MNIST dataset (0.1307 and 0.3081 respectively).

2. **Loading Datasets:**

    - torchvision.datasets.MNIST is used to load the MNIST dataset.
    - root: Path to store the downloaded dataset (defaults to ./data).
    - train: Boolean indicating whether to load the training or test set.
    - download: Boolean indicating whether to download the dataset if not already present.

- transform: The previously defined transform object is applied to the loaded images.

3. **Data Loaders:**

    - DataLoader creates iterators over the datasets for efficient training.
    - dataset: The dataset to load (either train_dataset or test_dataset).
    - batch_size: The number of samples to group together in each batch.
    - shuffle: Whether to shuffle the data order during training (set to True for training and False for testing).

4. **Visualization Function:**

    - show_batch(images, labels): This function displays a grid of 25 images along with their corresponding labels.
    - It utilizes plt.subplots to create a 5x5 grid of subplots.
    - It iterates over the first 25 images and labels, displaying them using imshow and setting titles.
    - axis('off') hides the axis labels and ticks for a cleaner presentation.
    - plt.tight_layout and plt.show adjust the layout and display the plot.

5. **Example Usage:**

    - An iterator is created from the train_loader using iter(train_loader).
    - The first batch of data is retrieved using next(data_iter). This returns a tuple containing images and labels.
    - show_batch is called with the first 25 images and labels (images[:25] and labels[:25]) to display a sample of the preprocessed data.

Advanced Preprocessing Techniques

The code showcases additional transformations that can be applied for advanced data augmentation:

- **Random Rotation:** transforms.RandomRotation(degrees) applies a random rotation to the image within the specified degree range. (Example: transforms.RandomRotation(10) rotates images by up to 10 degrees).

- **Random Affine Transformation:** transforms.RandomAffine(degrees, translate=(tx, ty)) applies random affine transformations including rotation, scaling, shearing, and translation. Here, it performs a random translation with a maximum horizontal and vertical shift of 0.1.

- **Image Binarization:**
  - A custom function binarize_image(img) is defined to convert pixel values above 0.5 to 1 and those below to 0, essentially creating a binary image.
  - This is integrated into the transform using transforms.Lambda(lambda img: binarize_image(img)).

- **Image Resizing:**
  - transforms.Resize((height, width)) resizes the image to the specified dimensions. (Example: transforms.Resize((32, 32)) resizes images to 32x32 pixels).

- **Flatten Transform:**
  - A custom class FlattenTransform is defined to flatten the image tensor into a 1D vector. This can be useful for certain neural network architectures.
  - The **__call__** method reshapes the image using view(-1).

- **Pixel Scaling:**
  - A custom class ScalePixels is defined to scale the pixel values by a factor (e.g., 255). This can be useful for normalizing pixel values to a specific range.
  - The **__call__** method multiplies