Muhammad Nur Maajid_1103228145

This module concentrates on creating and utilizing custom datasets in PyTorch for computer vision modeling. Discussions encompass steps for importing and preparing data, data transformation, custom dataset creation, and model development using these custom datasets.

1. **Acquiring Data:**
    - The initial step involves sourcing data.
    - The Food101 dataset, comprising 101,000 images of 101 food types, is prepared.
    - For training purposes, a subset of this dataset is used: 10% of images from three categories (pizza, steak, sushi).
    - Data is organized in a standard image classification format, structured within individual class directories.
    - Data is downloaded from GitHub and set up using Python.
2. **Understanding Data (Data Preparation):**
    - Involves critical steps prior to model building, including understanding the data's structure and characteristics.
    - The data consists of pizza, steak, and sushi images, each category stored in separate directories.
    - A function is created to explore the number of directories and images in each directory.
3. **Data Transformation:**
    - This step is essential to prepare images for use in PyTorch.
    - torchvision.transforms is used to convert images into tensors and apply various transformations like resizing and horizontal flipping.
    - A function is created to visualize transformations applied to several images.
4. **Loading Image Data Using ImageFolder:**
    - torchvision.datasets.ImageFolder is used to convert image data into a PyTorch Dataset.
    - Transformations are applied during Dataset creation.
    - Training and testing Datasets are initialized using the ImageFolder function and their information is printed.
    - Classes and lengths of each Dataset are noted for future reference.
    - Examples of images and labels from the Dataset are displayed.
5. **Option 2 - Loading Image Data with Custom Dataset:**
    - If a pre-built dataset creator like torchvision.datasets.ImageFolder() is not available or doesn't fit the specific requirements, a custom Dataset can be created. This has pros and cons:
    - Pros of creating a custom Dataset:
        - Can create a Dataset from almost anything.
        - Not limited to pre-built Dataset functions in PyTorch.
    - Cons of creating a custom Dataset:
        - Although almost any data can be used, it doesn't guarantee success.
        - Custom Datasets often require more coding, which can be prone to errors or performance issues.

    - To see this in action, a custom Dataset, ImageFolderCustom, is created by subclassing torch.utils.data.Dataset. This involves importing necessary modules, creating helper functions for class names, making the custom Dataset class to replicate ImageFolder, creating data transformations for image preparation, and displaying random images from the Dataset.

6. **Alternative Forms of Transformation (Data Augmentation):**
    - Data transformation can be used to alter images in various ways, including data augmentation to enhance the diversity of the training set. An example of data augmentation described is transforms.TrivialAugmentWide, leveraging randomness to improve model performance.
    - Finally, data transformations for training and testing are created, including data augmentation for the training set and image-to-tensor conversion. These transformations are used to create DataLoaders from our custom Dataset.
7. **Model 0: TinyVGG without Data Augmentation:**
    - Simple transformations are created, such as resizing images to (64, 64) and converting them to tensors.
    - Data is loaded, converting training and testing folders into Datasets using torchvision.datasets.ImageFolder, then into DataLoaders using torch.utils.data.DataLoader.
    - A TinyVGG model class is constructed with convolutional and pooling layers, used for image classification tasks.
    - A forward pass is performed on a single image to test the model.
    - torchinfo is used to get detailed information about the model, including the number of parameters and estimated total size.
    - Training and testing loop functions are created for the model.
8. **Evaluating Model 0 and Strategies for Improvement:**
    - Model 0 yields unsatisfactory results.
    - Strategies to enhance model performance involve addressing overfitting and underfitting.
    - Strategies to counter overfitting include obtaining more data, simplifying the model, using data augmentation, transfer learning, dropout layers, learning rate decay, and early stopping.
    - Strategies to overcome underfitting involve adding more layers or units to the model, adjusting the learning rate, using transfer learning, extending training duration, and reducing regularization.
9. **Model 1 - TinyVGG with Data Augmentation:**
    - Training transformations are created with TrivialAugmentWide, resizing images, and converting them to tensors.
    - Testing transformations are made without data augmentation.
    - ImageFolder is used to convert image folders into Datasets, then DataLoaders are created with specified batch sizes and number of workers.
    - TinyVGG (model_1) is built and trained using the previous training functions with appropriate parameters.
    - Model 1's loss curve is plotted to visualize its performance.
10. **Comparing Model Results:**
    - Model results are converted into DataFrames for comparison.

    - Graphs are created to compare model_0 and model_1 in terms of loss and accuracy on training and testing sets.

11. **Making Predictions on Custom Images:**
    - This step covers how to make predictions on custom images after training the model.
    - Custom images are downloaded using Python's requests module and read into tensors compatible with the trained model.
    - Transformations are applied to the custom image to match the data used to train the model.
    - The custom image is then predicted using the trained model, considering the format, device, and size.
    - A function is created to combine these steps for easier use, accepting the model, target image path, class names, transformations, and device as inputs.